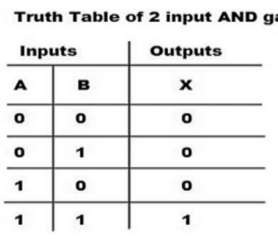
Logic Gate

- an elementary building block of a digital circuit.
- used to create simple to complex logic circuit or integrated circuits.
- Used by complex microprocessor or ICs
- Accept one or more inputs and produce one output with some logical condition between them.
- Deal with binary signals i.e. True or False, ON or OFF, 1 or 0;
- The state of the output is dependent on the input states.
- All logic gates implements some Boolean function which correlates output with input through some logical operation
- Logic gates are mainly designed with the electronic switches using diodes and transistors.
- There are three basic gates AND, OR and NOT.
- XOR, XNOR, NAND and NOR gates are derived gates.
- NAND and NOR gates are also known as universal logic gates.
- An XOR gate is inequality detector gate and can be used in comparator, adders, parity generators etc.
- Digital systems are constructed using logic gates.

AND Gate

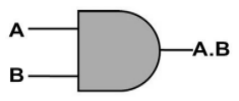
- **It will produce a high output when all the inputs are high otherwise the output is low . (X=A.B)**



Truth Table of 2 input AND gate

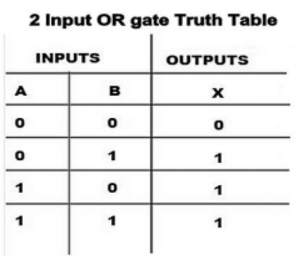
Inputs		Outputs
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

Symbol :



OR Gate

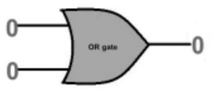
- An OR gate produces a high output when any one
- of the input is high .It produces a low output when all the
- inputs are low. (X=A+B)



2 Input OR gate Truth Table

INPUTS		OUTPUTS
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

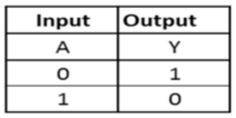
Symbol :



NOT Gate

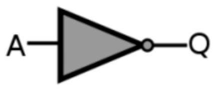
It produces high output when the input is low and vice versa. The NOT gate is also called as an inverter. (Q=A')

Truth Table



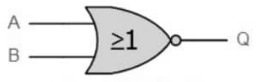
Input	Output
A	Y
0	1
1	0

Symbol :



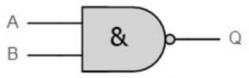
NOR Gate

NOR gate is OR gate followed by NOT gate. (X=(A+B)')

Symbol	Truth Table		
	A	B	Q
 2-input NOR Gate	0	0	1
	0	1	0
	1	0	0
	1	1	0
Boolean Expression Q = A NOR B			

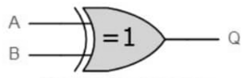
NAND Gate

NAND gate is combination of AND and NOT gates. The NAND gate provides AND functions with inverted output.

Symbol	Truth Table		
	A	B	Q
 2-input NAND Gate	0	0	1
	0	1	1
	1	0	1
	1	1	0
Boolean Expression Q = A NAND B			

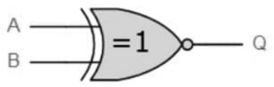
XOR Gate

- Exclusive OR gate is basically designed to exclude the condition of standard OR gate so as to generate real binary addition.
- The output of an XOR gate is at logic '1' when the inputs are dissimilar and at logic '0' when the inputs are similar.
- As XOR gate generates logic '1' only when inputs are not equal, hence this gate is also known as "inequality detector" gate.

Symbol	Truth Table		
	A	B	Q
 2-input Ex-OR Gate	0	0	0
	0	1	1
	1	0	1
	1	1	0
Boolean Expression Q = A XOR B			

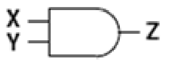
XNOR Gate

- XNOR is obtained by the combination of NOT and XOR gates.
- The output of an XNOR gate is at logic '1' when the inputs are similar and at logic '0' when the inputs are dissimilar. The logic equation for two input XOR gate is given by
- As XNOR gate generates logic '1' only when both the inputs are equal, hence this gate is also known as "equality detector" gate.

Symbol	Truth Table		
	A	B	Q
 2-input Ex-NOR Gate	0	0	1
	0	1	0
	1	0	0
	1	1	1
Boolean Expression Q = A XNOR B			

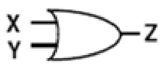
Logic gates and truth tables

◆ AND $X \bullet Y$ XY



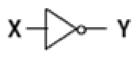
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

◆ OR $X+Y$



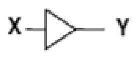
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

◆ NOT $\overline{X}$ X′



X	Y
0	1
1	0

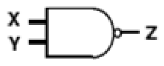
◆ Buffer X



X	Y
0	0
1	1

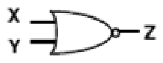
Logic gates and truth tables (con’t)

◆ NAND $\overline{X \bullet Y}$ $\overline{XY}$



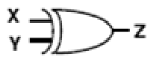
X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

◆ NOR $\overline{X+Y}$



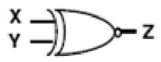
X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

◆ XOR $X \oplus Y$



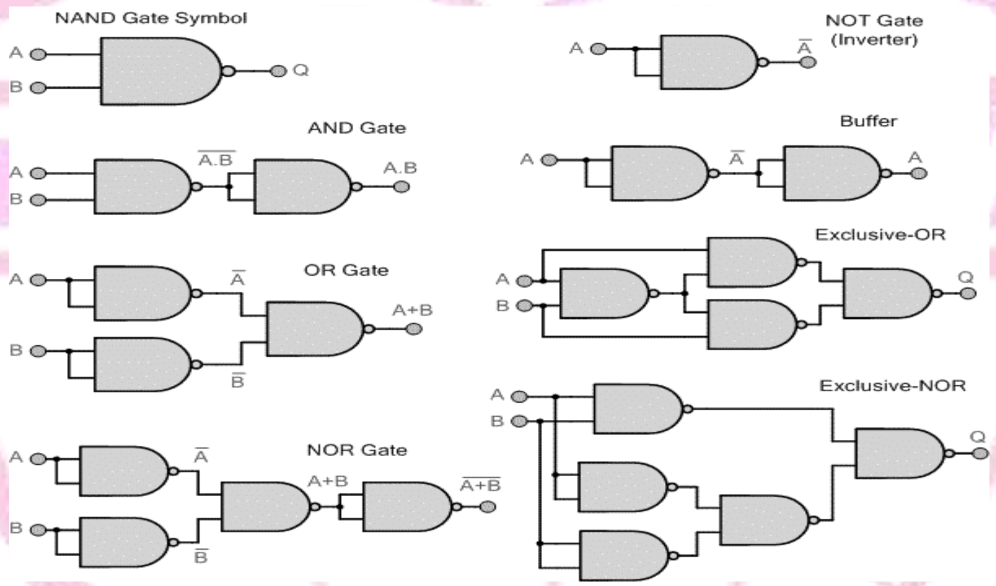
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

◆ XNOR $\overline{X \oplus Y}$

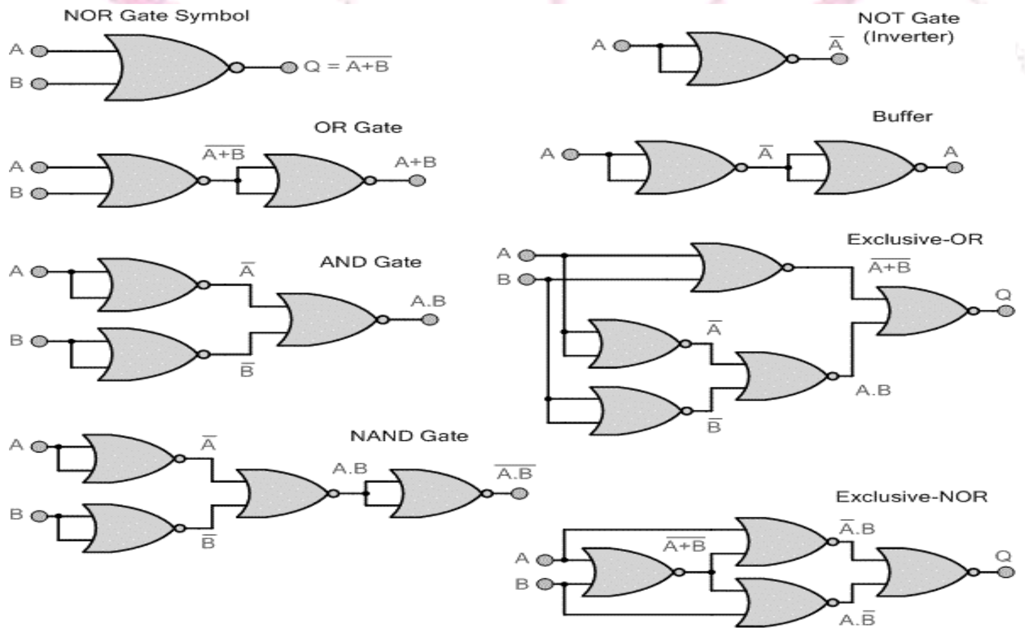


X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

Logic Gates by using NAND Gate



Logic gates using NOR Gate



Implementation of All GATES using universal GATES

Implemented gate	NAND GATE	NOR GATE
NOT	1	1
OR	3	2
AND	2	3
EXOR	4	5
EXNOR	5	4
NOR	4	---
NAND	---	4

Boolean Algebra

- Boolean algebra is a mathematical function to represent the logical conditions.
- To process the output of a complex circuit we use boolean algebra.
- All arithmetic operations performed with Boolean quantities have one of two possible outcomes: either 1 or 0.

Laws of Boolean Algebra

1. **Commutative Law:-**
   A+B = B+A
   A.B = B.A

2. **Associative Law:-**
   (A+B)+C =A+(B+C)
   (A.B).C = A.(B.C)

3. **Distributive Law:-**
   A(B+C) = AB+AC
   A+BC = (A+B)(A+C)

4. **Absorption Law:-**
   A+AB = A
   A(A+B) =B

5. **Idempotence Law:-**
   A+A = A
   A.A =A
6. **Ivolutionary Law:-**
   (A')' =A

Some basic rules of Boolean algebra:-

- **A+0 = A**
- **A+1 =1**
- **A.0 =0**
- **A.1=A**
- **A+A = A**

- **A+A’ =1**
- **A.A = A**
- **A.A’ =0**
- **(A’)’ =A**
- **A+A’B = A+B**
- **(A+B)(A+C) = A+BC**

Boolean Algebra Theorms

DeMorgan Theorms:-
(A.B)’ = A’ + B’
(A+B)’ = A’ * B’

In general form
(A1.A2.A3.......An)’ = A1’+A2’+A3’+........+An’
(A1+A2+A3+........+An)’ = A1’A2’A3’......An’

Consensus Theorem/Redundancy Theorem

A variable is associated with some variable and its compliment is associated with some other variable and the next term is formed by the left over variables, then the term becomes redundant.
This theorem is used to eliminate redundant term.

It is applicable only if a Boolean function,
1. Contains 3-variables.
2. Each variable used two times.
3. Only one variable is in complemented or uncomplemented form.

Then, the related terms to that complemented and uncomplemented variable is the answer.
Consensus theorem can be extended to any number of variables.
For Example:-
1. AB+A’B+BC =AB+A’B
2. (A+B) (B’+ C) (C + A) = (A+B) (B’+ C) = AB’ +BC

Duality Theorem

"Dual expression" is equivalent to write a negative logic of the given Boolean relation.
For this we have to:-
(i) change each OR sign by an AND sign and vice-versa
(ii) complement any '0' or '1' appearing in expression
(iii) keep literals/variables as it is.

Example:- A’BC+ABC’+AB’C’ = (A’+B+C) (A+B+C’) (A+B’+C’)

Self Dual Theorm

A boolean function is said to be Self dual if and only if its dual is equivalent to the given boolean function.
The given function is neutral i.e., (the number of minterms is equal to the number of maxterms).
The function does not contain two mutually exclusive terms.
Ex:- f(A,B,C) = AB+BC+AC
It dual (A+B).(B+C).(A+C) is equivanent to the same function.

Important Points

- Every Self-dual function is neutral but every neutral function is not Self-dual.
- Self-duality is closed under complement i.e, the complement of a Self-dual function is also Self-dual.
- Total number of combinations with n variables:- $\mathbf{2^n}$
- Total number of boolean function with n variables:- $\mathbf{2^{2^\wedge n}}$
- Total number of dual with n variables: $\mathbf{2^{2^\wedge n-1}}$

Complementary Theorem

For obtaining complement expression we have to
(i) change each OR sign by AND sign and vice-versa
(ii) complement any '0' or '1' appearing in expression.
(ii) complement the individual literals/variables.

Ex:- The compliment of the function- AB'C+A'BC'+ AB'C'= (A'+B+C') (A+B'+C) (A'+B+C)

Simplify Boolean Expression

1. The Boolean expression A(A + B) is equal to

2. The Boolean function (x + y) (x' + z) (y + z) is equal to

3. The minimized form of the logical expression (A'B'C' + A'BC' + A'BC + ABC') is

4. If X and Y are Boolean variables, X xor Y xor XY is equivalent to

Representation of Boolean Functions

Representation of Boolean Functions is called boolean expression.
A truth table of a boolean function can be represented as a boolean expression.
A function of 'n' Boolean variables denoted by (A1,A2, A3.......An) is another variable of algebra and takes one of the two possible values either 0 or 1.

Boolean Function can be represented as:-

a) **Canonical form:-** All the terms contain all the variables either in complementary or in uncomplentary form
   Example: F(A, B, C) = A'BC + ABC + A'BC'

b) **Minimal form:-** Minimum numbers of literal/variables
   Example: F(A, B, C)=A+ABC+A'BC=A

Minterms and Maxterms

- As we know n-binary variables have 2^n possible combinations.
- Minterm is a product term, it contains all the variables either complementary or uncomplimentary form for that combination the function output must be '1'.
- Maxterm is a sum term, It contains all the variables either complementary or uncomplimentary form for that combination the function output must be '0'.

Sum of Product (SOP) Form

- The SOP expression usually takes the form of two or more variables ANDed together. Each product term may be minterm or implicant.
- Y = A'BC+ AB'+ AC
- Y = AB'+BC'
- This form is also called the "disjunctive normal form".
- The SOP expression is used most often because it tends itself nicely to the development of truth tables and timing diagrams.
- SOP circuits can also be constructed easily by using a special combinational logic gates called the AND-OR-INVERTER gate.
- SOP forms are used to write logical expression for the output becoming Logic 1.

- **Notation of SOP expression:-**
- f(A, B, C) = Σ(3, 5, 6, 7)
- Y = $m_3 + m_5 + m_6 + m_7$
- Y= A’BC+ AB’C + ABC’ + ABC

Input(3 variables)			Output (Y)
A	B	C	
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

Product of Sum (POS) Form

- The POS expression usually takes the form of two or more order variables within parentheses, ANDed with two or more such terms.
- Ex:- Y = (A+B’+C) (BC’ + D)
- This form is also called the "Conjunctive normal form".
- Each individual term in standard POS form is called Maxterm.
- POS forms are used to write logical expression for output be coming Logic '0’.

From the above truth table,
we get:-
f(A B C) = π m(0, 1, 2, 4)
Y = $m_0 \times m_1 \times m_2 \times m_4$
or
Y = (A+B+C)(A+B+C’)(A+B’+C)(A’+B+C)

Input(3 variables)			Output (Y)
A	B	C	
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

We also conclude that From the above truth table, and from above equations:-
If Y = Σm(3, 5, 6, 7)
Then Y = πm(0, 1, 2, 4)

Standard Sum of Product Form

- In this form the function is the sum of number of product terms where each product term contains all the variables of the function, either in complemented or uncomplemented form.
- It is also called canonical SOP form or expanded SOP form.
- The function Y = A + BC’ can be represented as canonical form as:-
  Y = ABC+AB’C+AB’C’+ABC’+A’BC’

Standard Product of Sum Form

This form is also called canonical POS form or expanded POS form.
The function Y=(B+C')(A+B') can be represented in canonical form as:
Y = (B+C'+AA')(A+B'+CC') = (B+C'+A) (B + C' + A')(A+B'+C)(A+B'+C')

Truth Table Form

A truth table is a tabular form representation of all possible combinations of given function.
Y = A'B+B'C is represented as truth table:

Input(3 variables)			**Output (Y)**
0	**0**	**0**	**0**
0	**0**	**1**	**1**
0	**1**	**0**	**1**
0	**1**	**1**	**1**
1	**0**	**0**	**0**
1	**0**	**1**	**1**
1	**1**	**0**	**0**
1	**1**	**1**	**0**

1. Simplify the equation y(A,B) = π m(1,3).

Simplification of Boolean expressions

There are three methods to simplify the Boolean expression

1. Boolean Laws and theorems
2. K maps
3. Quine Mc-Cluskey or Tabulation Method

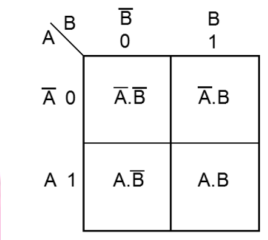
Karnaugh map

- The "Karnaugh map" is a graphical method which provides a systematic method for simplifying and manipulating the Boolean expressions or to convert a truth table to its corresponding logic circuit in a simple, orderly process.
- In this technique, the information contained in a truth table or available in SOP or POS form is represented on K-map.
- Although this technique may be used for any number of variables, it is generally used up to 6-variables beyond which it becomes very cumbersome.
- In n-variable K-map there are 2^n cells.
- "Gray code" has been used for the identification of cells:

Two-variable K-Map

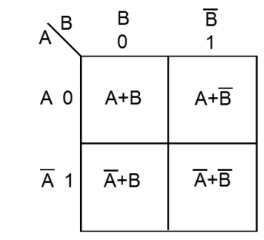
- Four cells
- Four minterms (maxterms)

A. SOP: -



A \ B	$\bar{B}$ 0	B 1
$\bar{A}$ 0	$\bar{A}.\bar{B}$	$\bar{A}.B$
A 1	$A.\bar{B}$	$A.B$

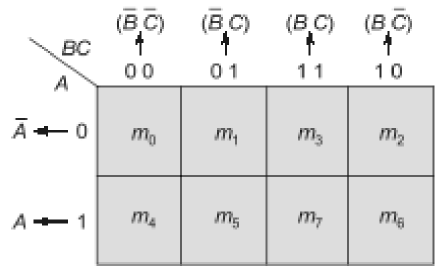
B. POS: -



A \ B	B 0	$\bar{B}$ 1
A 0	$A+B$	$A+\bar{B}$
$\bar{A}$ 1	$\bar{A}+B$	$\bar{A}+\bar{B}$

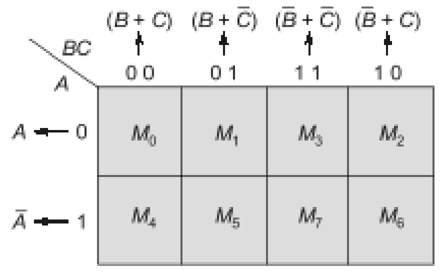
Three variable K-map

- Eight cells
- Eight minterms(maxterms)



A \ BC	$(\bar{B}\bar{C})$ 00	$(\bar{B}C)$ 01	(BC) 11	$(B\bar{C})$ 10
$\bar{A}$ ← 0	m_0	m_1	m_3	m_2
A ← 1	m_4	m_5	m_7	m_6

(For SOP)

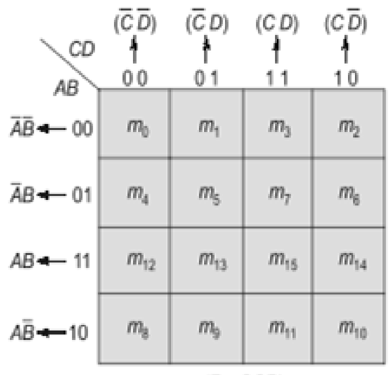


A \ BC	$(B+C)$ 00	$(B+\bar{C})$ 01	$(\bar{B}+\bar{C})$ 11	$(\bar{B}+C)$ 10
A ← 0	M_0	M_1	M_3	M_2
$\bar{A}$ ← 1	M_4	M_5	M_7	M_6

(For POS)

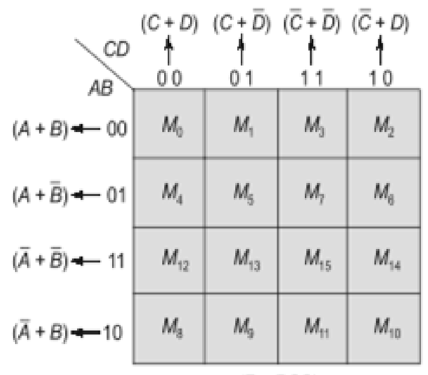
4- variable K-map

- Sixteen cells
- sixteen minterms (maxterms)



AB \ CD	$(\bar{C}\bar{D})$ 00	$(\bar{C}D)$ 01	(CD) 11	$(C\bar{D})$ 10
$\bar{A}\bar{B}$ ← 00	m_0	m_1	m_3	m_2
$\bar{A}B$ ← 01	m_4	m_5	m_7	m_6
AB ← 11	m_{12}	m_{13}	m_{15}	m_{14}
$A\bar{B}$ ← 10	m_8	m_9	m_{11}	m_{10}

(For SOP)



AB \ CD	$(C+D)$ 00	$(C+\bar{D})$ 01	$(\bar{C}+\bar{D})$ 11	$(\bar{C}+D)$ 10
$(A+B)$ ← 00	M_0	M_1	M_3	M_2
$(A+\bar{B})$ ← 01	M_4	M_5	M_7	M_6
$(\bar{A}+\bar{B})$ ← 11	M_{12}	M_{13}	M_{15}	M_{14}
$(\bar{A}+B)$ ← 10	M_8	M_9	M_{11}	M_{10}

(For POS)

Minimize the following boolean function
$F(A, B, C) = \Sigma m(0, 1, 6, 7) + \Sigma d(3, 5)$

Minimize the following boolean function
$F(A, B, C, D) = \Sigma m(0, 1, 2, 5, 7, 8, 9, 10, 13, 15)$

Simplify the following boolean function
$F(W, X, Y, Z) = \Sigma m(1, 3, 4, 6, 9, 11, 12, 14)$

Minimize the following boolean function
$F(A, B, C, D) = \Sigma m(1, 3, 4, 6, 8, 9, 11, 13, 15) + \Sigma d(0, 2, 14)$

K-MAP Simplification Rules

- Construct the K-map and place 1s in those cells corresponding to the 1's in the truth table.Place 0's in the other cells.
- Examine the map for adjacent 1's and loop those 1's which are not adjacent to any other 1's.
- These are called isolated 1's.
- Next, look for those 1's which are adjacent to only one other 1. Loop any pair containing such a 1.
- Loop any octet even it contains some 1's that have already been looped.
- Loop any quad that contains one or more 1's which have not already been looped, making sure to use the minimum number of loops.
- Loop any pairs necessary to include any 1's that have not yet been looped, making sure to use the minimum number of loops.
- Form the OR sum of all the terms generated by each loop

Don't Care Condition

- Some logic circuits can be designed so that there are certain input combinations for which there are no specified output levels, usually because these input combinations will never occur.
- So, a circuit designer is free to make the output for any "don't care" condition either a 0 or 1 in order to produce the simplest output expression.
- The two conditions can be better understood by constructing the K-map for the given two conditions.

(i) In terms of SOP and don't care conditions.

$f(A,B,C,D) = \Sigma m(1, 3, 7, 11, 15)+d(0,2,5)$

(ii) In terms of POS and don't care conditions.

$f(A, B, C, D) = \pi m(4, 5, 6, 7, 8, 12). d(1, 2, 3, 9, 11,14)$

Implicants, Prime Implicants and Essential Prime Implicants

Implicant: implicant is a product term on the given function for that combination the function output must be 1.
Prime Implicant: Prime implicant is a smallest possible product term of the given function, removing any one of the literal from which is not possible.
Essential Prime Implicant: Essential prime implicant is a prime implicant it must cover atleast one minterm, which is not covered by any other prime implicant.

For the given K-map, find implicant, Prime implicant and Essential Prime Implicant can be represented as:-

1	1		1
		1	1