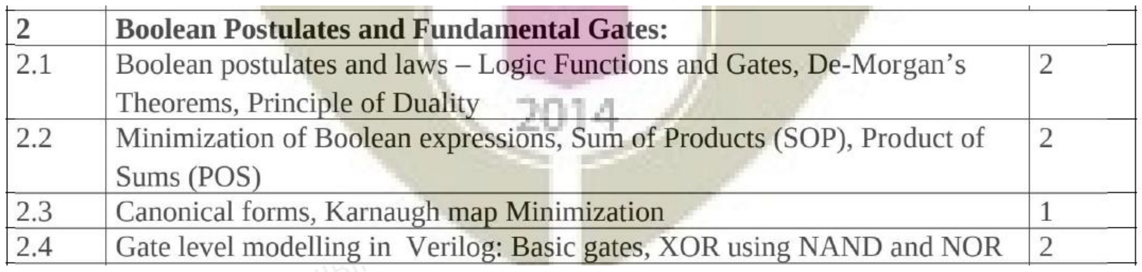
APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

LOGIC CIRCUIT DESIGN (ECT-203)

MODULE 2 NOTES

PREPARED BY : JIBIN EP, ASSISTANT PROFESSOR, EKC TC MANJERI

INDEX



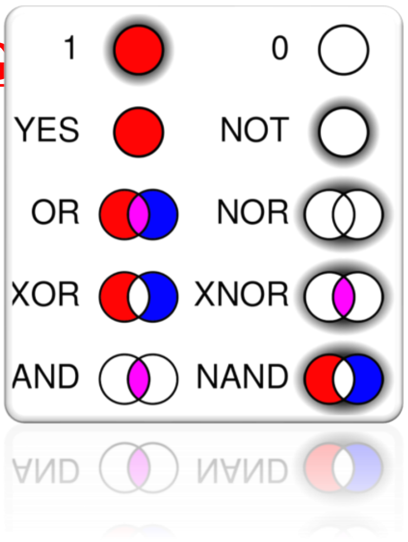
2	**Boolean Postulates and Fundamental Gates:**	
2.1	Boolean postulates and laws – Logic Functions and Gates, De-Morgan's Theorems, Principle of Duality	2
2.2	Minimization of Boolean expressions, Sum of Products (SOP), Product of Sums (POS)	2
2.3	Canonical forms, Karnaugh map Minimization	1
2.4	Gate level modelling in Verilog: Basic gates, XOR using NAND and NOR	2

MODULE 2
(TOPIC-1)

THE LOGIC GATES

https://youtu.be/MY8pLZ_MOI

DEFENITION OF LOGIC G



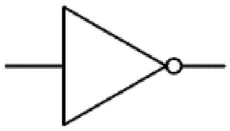
- It is a physical device which preforms logic operation on one or more logical input and produces a single output

Classification of Logic Gates

- BASIC GATES
  - NOT
  - AND
  - OR
- UNIVERSAL GATES
  - NAND
  - NOR
- ARITHMETIC GATES
  - XOR
  - XNOR

BASIC GATES

1.NOT GATE (INVERTER)



A NOT gate can only have one input and the output is the inverse of the input..

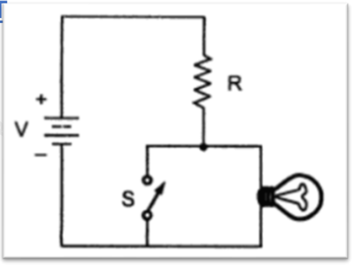
1. TRUTH TABLE

INPUT (A)	OUTPUT (Y)
0	1
1	0

2. BOOLEAN EXPRESSION

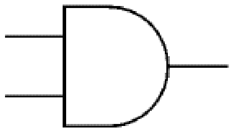
$$Y= \overline{A}$$

4. SWITCHING CIRCUIT



INPUT	OUTPUT
Switch Open	Lamp ON
Switch Close	Lamp OFF

2.AND GATE



An AND gate can have two or more inputs, its output is true if all inputs are true. The output Q is true if input A AND input B are both true

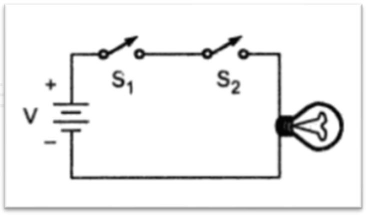
1. TRUTH TABLE

INPUT (A)	INPUT (B)	OUTPUT(Y)
0	0	0
0	1	0
1	0	0
1	1	1

2. BOOLEAN EXPRESSION

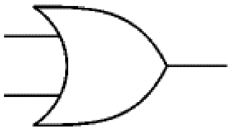
Y=AB

3. SWITCHING CIRCUIT



INPUT S1	INPUT S2	OUTPUT
Open	Open	Lamp OFF
Open	Close	Lamp OFF
Close	Open	Lamp OFF
Close	**Close**	**Lamp ON**

3.OR GATE



An OR gate can have two or more inputs, its output is true if at least one input is true.

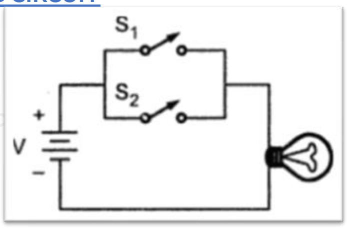
1. TRUTH TABLE

INPUT (A)	INPUT (B)	OUTPUT(Y)
0	0	0
0	1	1
1	0	1
1	1	1

2. BOOLEAN EXPRESSION

Y=A+B

3. SWITCHING CIRCUIT



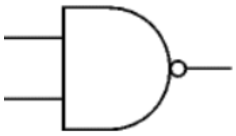
INPUT S1	INPUT S2	OUTPUT
Open	Open	Lamp OFF
Open	Close	Lamp ON
Close	Open	Lamp ON
Close	Close	Lamp ON

UNIVERSAL GATES

NAND and NOR gates are widely known to be universal logic gates, meaning that any other logic gate be made from NAND or NOR gates

3.NAND GATE

(NOT + AND)



This is an AND gate with the output inverted , A NAND gate can have two or more inputs, its output is true if NOT all inputs are true.

1. TRUTH TABLE

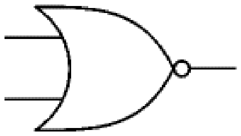
INPUT (A)	INPUT (B)	OUTPUT(Y)
0	0	1
0	1	1
1	0	1
1	1	0

2. BOOLEAN EXPRESSION

$Y = \overline{AB}$

3.NOR GATE

(NOT + OR)



This is an OR gate with the output inverted, A NOR gate can have two or more inputs, its output is true if no inputs are true

1. TRUTH TABLE

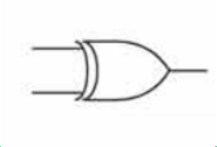
INPUT (A)	INPUT (B)	OUTPUT(Y)
0	0	1
0	1	0
1	0	0
1	1	0

2. BOOLEAN EXPRESSION

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

ARITHMETIC GATES

3.XOR GATE
(EXCLUSIVE OR GATE)



This is an arithmetic gate, which produces low output when the inputs are same

1. TRUTH TABLE

INPUT (A)	INPUT (B)	OUTPUT(Y)
0	0	0
0	1	1
1	0	1
1	1	0

2. BOOLEAN EXPRESSION

$$Y = A \oplus B$$

3.XNOR GATE
(EXCLUSIVE NOR GATE)

This is an arithmetic gate, which produces high output when the inputs are same

1. TRUTH TABLE

INPUT (A)	INPUT (B)	OUTPUT(Y)
0	0	1
0	1	0
1	0	0
1	1	1

2. BOOLEAN EXPRESSION

$$Y = A \Theta B$$

MODULE 2
(TOPIC-2)

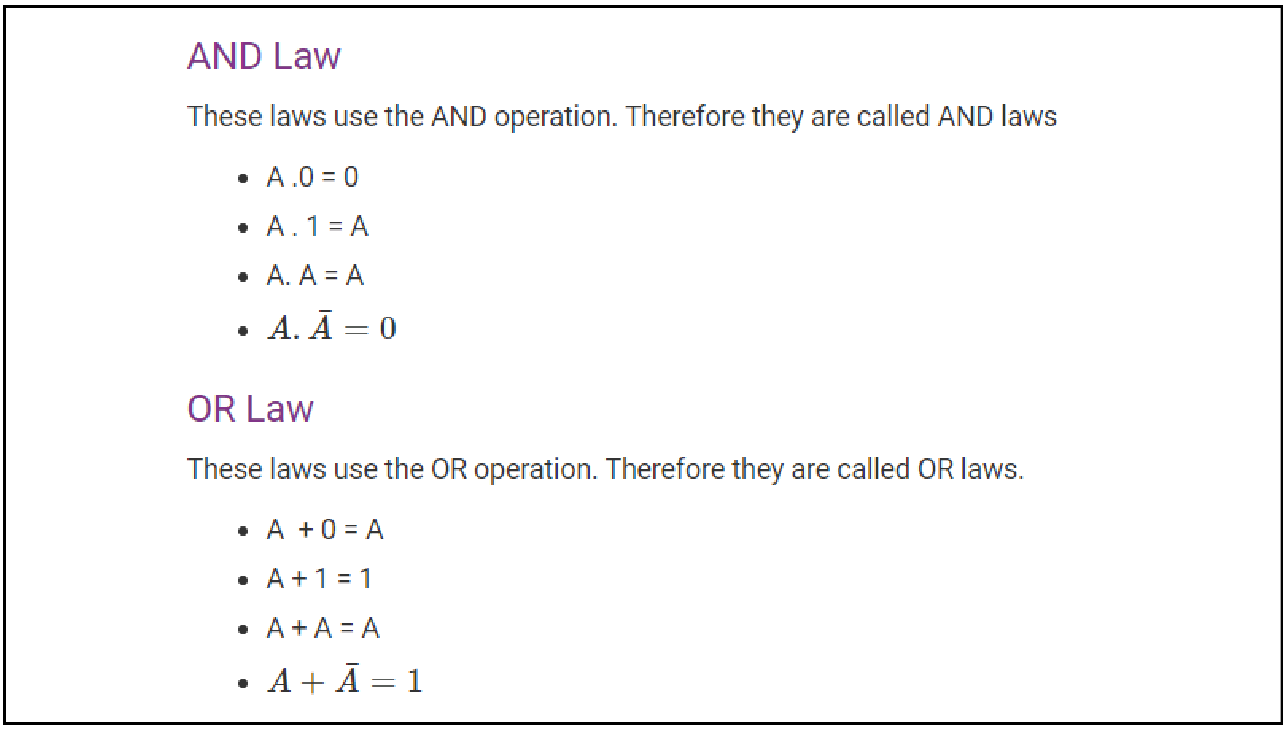
BOOLEAN ALGEBRA

https://youtu.be/djWQBrgtrxs YouTube

Inversion Law

This law uses the NOT operation. The inversion law states that double inversion of variable results in the original variable itself.

- $A + \bar{\bar{A}} = 1$



AND Law

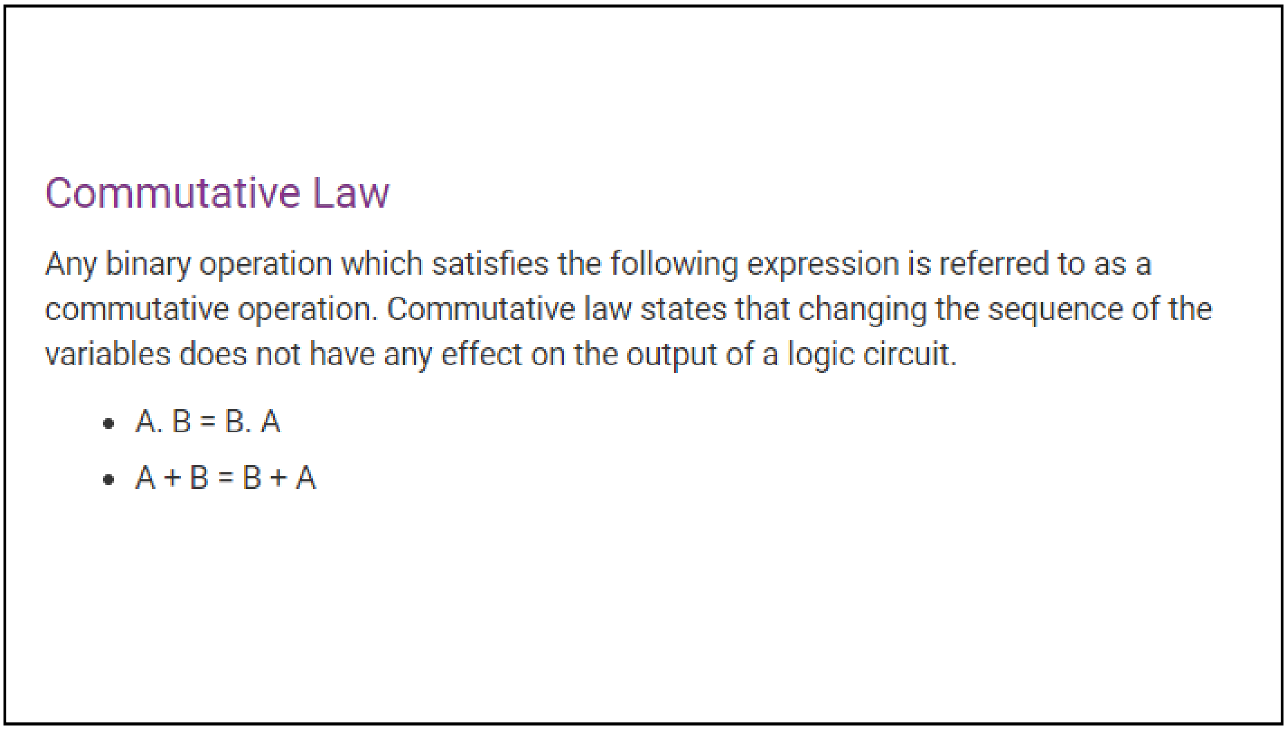
These laws use the AND operation. Therefore they are called AND laws

- A .0 = 0
- A . 1 = A
- A. A = A
- $A.\bar{A} = 0$

OR Law

These laws use the OR operation. Therefore they are called OR laws.

- A + 0 = A
- A + 1 = 1
- A + A = A
- $A + \bar{A} = 1$



Commutative Law

Any binary operation which satisfies the following expression is referred to as a commutative operation. Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit.

- A. B = B. A
- A + B = B + A

Associative Law

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

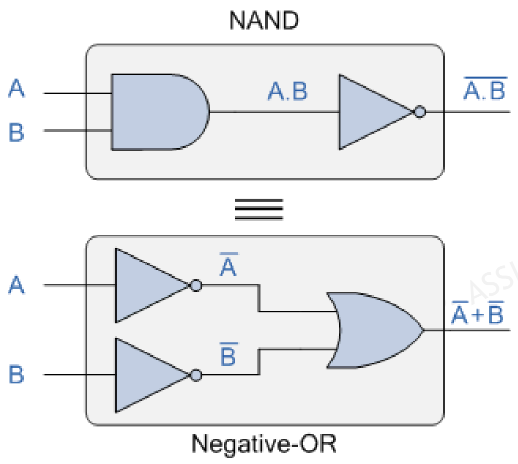
- $(A.B).C = A.(B.C)$
- $(A+B)+C = A+(B+C)$

Distributive Law

Distributive law states the following conditions:

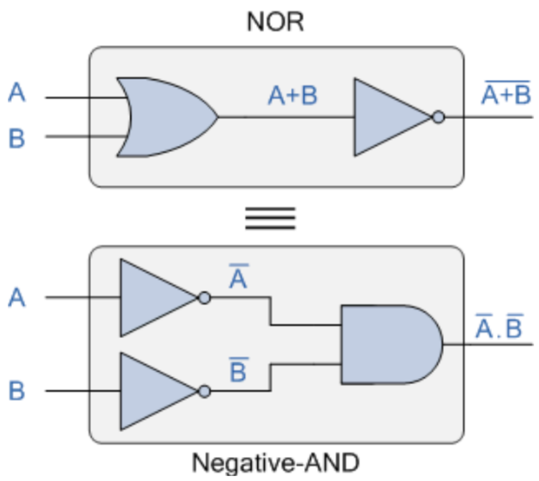
- $A.(B+C) = (A.B)+(A.C)$
- $A+(B.C) = (A+B).(A+C)$

DeMorgan’s First Law Implementation using Logic Gates



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

DeMorgan’s Second Theorem



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

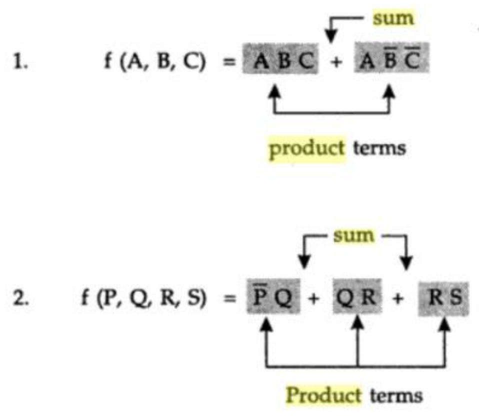
MODULE 2
(TOPIC-3,4)

SUM OF PRODUCT
PRODUCT OF SUM

https://youtu.be/FOGczanvVEk
https://youtu.be/_4RFZ0s3Qks

Sum Of Product (SOP)

- Each SOP Consist of two or more product terms(AND) that ORed together


sum
1. $f(A, B, C) = ABC + A\bar{B}\bar{C}$
product terms
sum
2. $f(P, Q, R, S) = \bar{P}Q + QR + RS$
Product terms

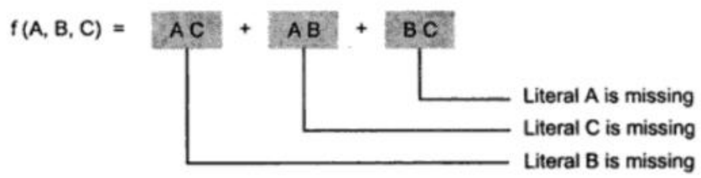
Canonical SOP

Example 2.3 : *Convert the given expression in standard SOP form.*

$$f(A, B, C) = AC + AB + BC$$

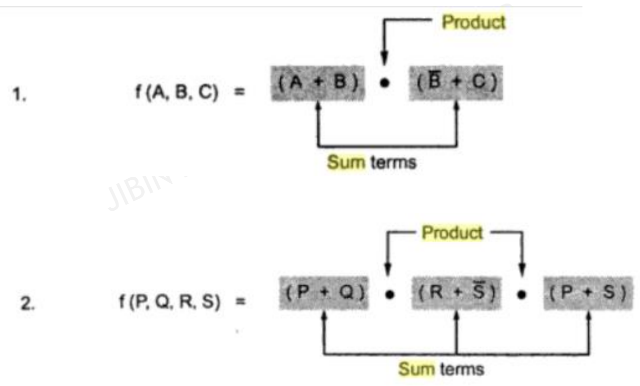
Solution :

Step 1 : Find the missing literal/s in each product term

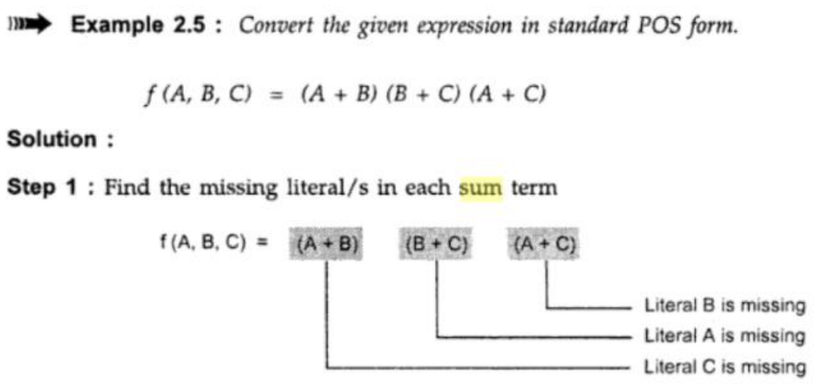


Product of Sum (POS)

- Each POS Consist of Two or more Sum Terms(OR) that AND ed Together


Product
1. f (A, B, C) = (A + B) • (B̄ + C)
Sum terms
Product
2. f (P, Q, R, S) = (P + Q) • (R + S̄) • (P + S)
Sum terms

Canonical POS



Example 2.5 : *Convert the given expression in standard POS form.*

$$f(A, B, C) = (A + B)(B + C)(A + C)$$

Solution :

Step 1 : Find the missing literal/s in each sum term

$f(A, B, C) = (A + B) \; (B + C) \; (A + C)$

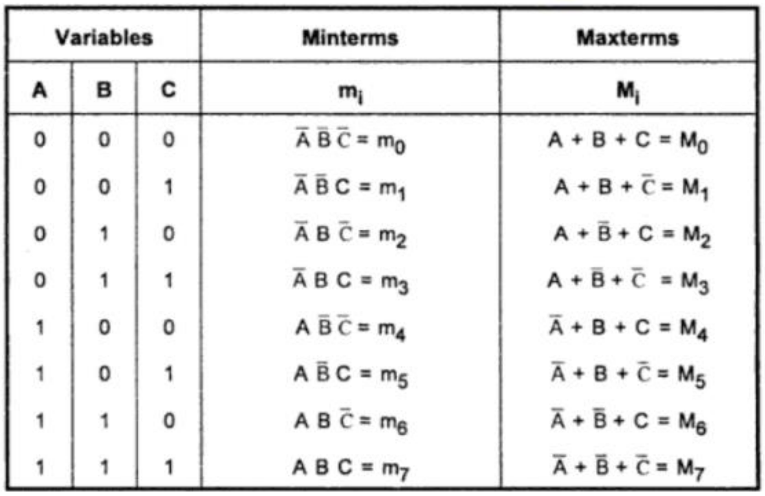
Literal B is missing

Literal A is missing

Literal C is missing

Mintem and Maxterm

- Each individual term in Standard SOP is called Minterm
- Each individual term in Standard POS is called Maxterm



Variables			Minterms	Maxterms
A	B	C	m_i	M_i
0	0	0	$\bar{A}\,\bar{B}\,\bar{C} = m_0$	$A + B + C = M_0$
0	0	1	$\bar{A}\,\bar{B}\,C = m_1$	$A + B + \bar{C} = M_1$
0	1	0	$\bar{A}\,B\,\bar{C} = m_2$	$A + \bar{B} + C = M_2$
0	1	1	$\bar{A}\,B\,C = m_3$	$A + \bar{B} + \bar{C} = M_3$
1	0	0	$A\,\bar{B}\,\bar{C} = m_4$	$\bar{A} + B + C = M_4$
1	0	1	$A\,\bar{B}\,C = m_5$	$\bar{A} + B + \bar{C} = M_5$
1	1	0	$A\,B\,\bar{C} = m_6$	$\bar{A} + \bar{B} + C = M_6$
1	1	1	$A\,B\,C = m_7$	$\bar{A} + \bar{B} + \bar{C} = M_7$

MODULE 2
(TOPIC-5)

NAND NOR IMPLEMENTATION

https://youtu.be/TNj8WSXXH-Q YouTube

CONTENTS

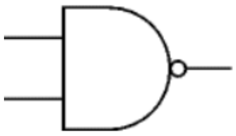
NAND
- NOT
- AND
- OR
- NOR

NOR
- NOT
- AND
- OR
- NAND

NAND GATE

3.NAND GATE

(NOT + AND)



This is an AND gate with the output inverted , A NAND gate can have two or more inputs, its output is true if NOT all inputs are true.

1. TRUTH TABLE

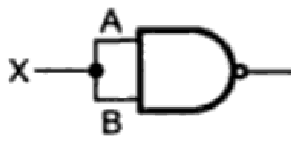
INPUT (A)	INPUT (B)	OUTPUT(Y)
0	0	1
0	1	1
1	0	1
1	1	0

2. BOOLEAN EXPRESSION

$$Y = \overline{AB}$$

NOT FUNCTIO

An inverter can be made from NAND gate by connecting all the input together and creating, in effect a single common input



PROOF

$$Y = \overline{AB}$$

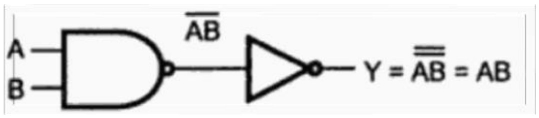
$$Y = \overline{XX}$$ DEMORGANS LAW

$$Y = \bar{X} + \bar{X}$$ REFERENCE : A+A=A

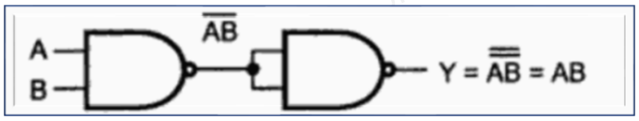
$$Y = \bar{X}$$

AND FUNCTIO

An AND Function is Generated by simply inverting the output of NAND Gate



A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

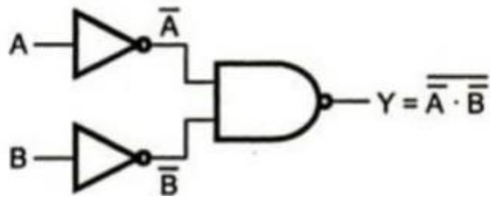


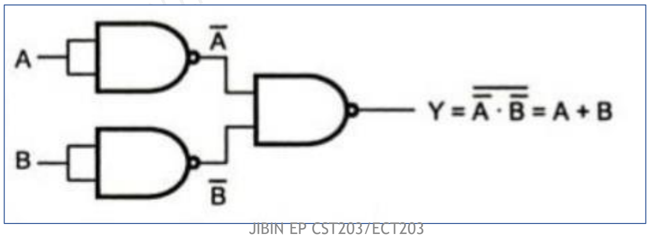
A	B	$\overline{AB}$	$\overline{\overline{AB}}$
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

OR FUNCTION

An OR Function is Generated by simply inverting both the inputs of NAND Gate

$$Y = A + B$$
$$= \overline{\overline{A}} + \overline{\overline{B}} \quad [\overline{\overline{A}} = A]$$
$$= \overline{\overline{A} \cdot \overline{B}} \quad \text{DeMorgan's Theorem 1}$$

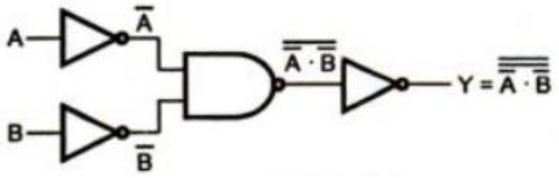




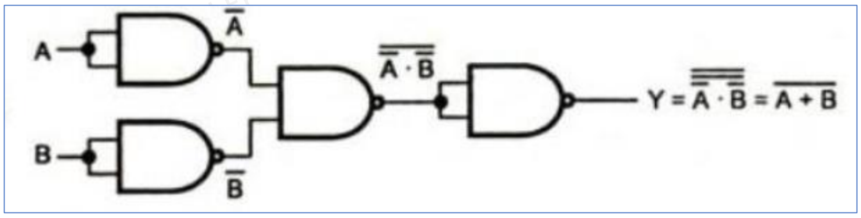
NOR FUNCTION

An NOR Function is Generated by simply inverting both the inputs of NAND Gate and invert the output

$$Y = \overline{A + B}$$
$$= \overline{A} \,.\, \overline{B} \quad \text{DeMorgan's Theorem 2}$$
$$= \overline{\overline{\overline{A} \cdot \overline{B}}} \quad [\overline{\overline{A}} = A]$$



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

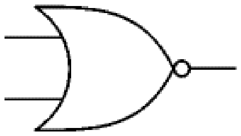


A
$\overline{A}$
B
$\overline{B}$
$\overline{\overline{A} \cdot \overline{B}}$
$Y = \overline{\overline{\overline{A} \cdot \overline{B}}} = \overline{A + B}$

NOR GATE

3.NOR GATE

(NOT + OR)



This is an OR gate with the output inverted, A NOR gate can have two or more inputs, its output is true if no inputs are true

1. TRUTH TABLE

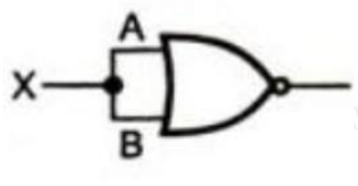
INPUT (A)	INPUT (B)	OUTPUT(Y)
0	0	1
0	1	0
1	0	0
1	1	0

2. BOOLEAN EXPRESSION

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

NOT FUNCTIO

An inverter can be made from NOR gate by connecting all the input together and creating, in effect a single common input



PROOF

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

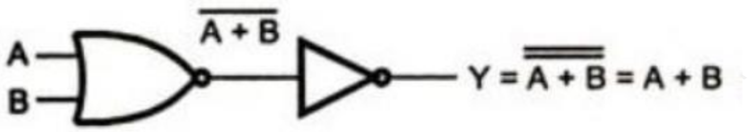
$$Y = \overline{X + X}$$

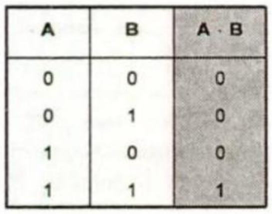
$$Y = \overline{X}$$

REFERENCE : A+A=A

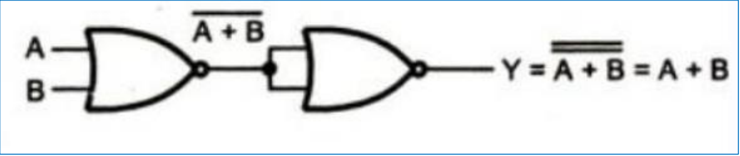
OR FUNCTION

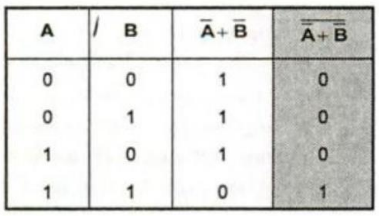
An OR Function is Generated by simply inverting the output of NOR Gate





A	B	A · B
0	0	0
0	1	0
1	0	0
1	1	1



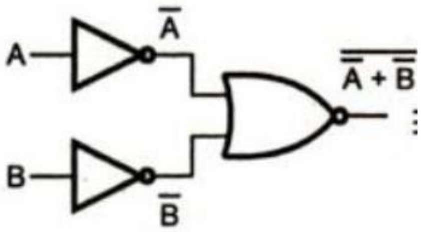


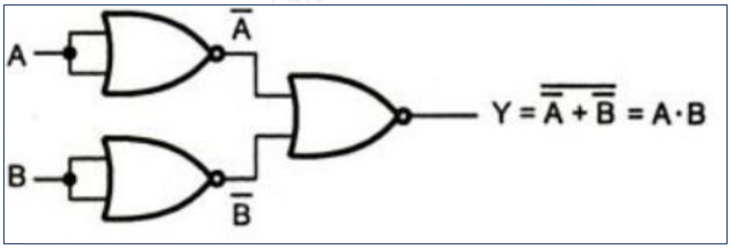
A	B	$\overline{A}+\overline{B}$	$\overline{\overline{A}+\overline{B}}$
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

AND FUNCTIO

An AND Function is Generated by simply inverting both the inputs of NOR Gate

$$Y = A \cdot B$$
$$= \overline{\overline{A}} \cdot \overline{\overline{B}} \qquad [\overline{\overline{A}} = A]$$
$$= \overline{\overline{A} + \overline{B}} \qquad \text{DeMorgan's Theorem 2}$$

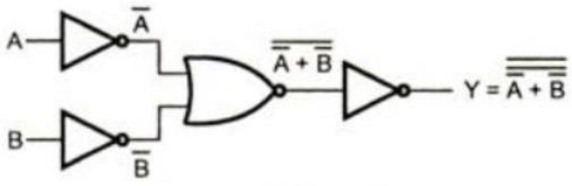




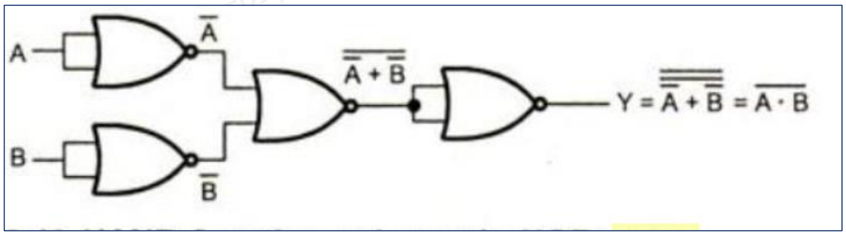
NAND FUNCTI

An NAND Function is Generated by simply inverting both the inputs of NOR Gate and invert the output

$$Y = \overline{A \cdot B}$$
$$= \overline{A} + \overline{B} \qquad \text{DeMorgan's Theorem 1}$$
$$= \overline{\overline{\overline{A} + \overline{B}}} \qquad [\overline{\overline{A}} = A]$$



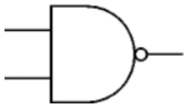
A
$\overline{A}$
B
$\overline{B}$
$\overline{\overline{A}+\overline{B}}$
$Y = \overline{\overline{\overline{A}+\overline{B}}}$



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

CONCLUSION

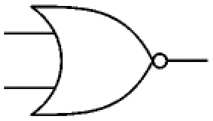
3.NAND GATE



NOT	AND	OR	NOR
• Connecting all the input together	• Inverting the output of NAND Gate	• Inverting the Inputs of NAND Gate	• Inverting the Inputs of NAND Gate and invert the output

CONCLUSION

3.NOR GATE



NOT	OR	AND	NAND
• Connecting all the input together	• Inverting the output of NOR Gate	• Inverting the Inputs of NOR Gate	• Inverting the Inputs of NOR Gate and invert the output

MODULE 2
(TOPIC-6,7,8)

KARNAUGH MAP

https://youtu.be/ezp0B2F6L0k

https://youtu.be/UoxRJkMJ-pI https://youtu.be/ZMARz-Xr0F4

YouTube

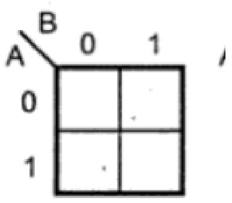
Karnaugh Map

- We have simplified the Boolean functions using Boolean postulates and theorems. It is a time consuming process and we have to re-write the simplified expressions after each step.
- To overcome this difficulty, **Karnaugh** introduced a method for simplification of Boolean functions in an easy way. This method is known as Karnaugh map method or K-map method. It is a graphical method, which consists of 2^n cells for 'n' variables. The adjacent cells are differed only in single bit position.

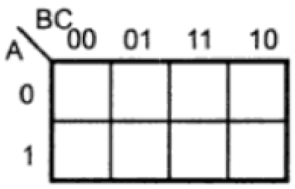
TYES OF KMAP

2 Variable	• SOP • POS
3 Variable	• SOP • POS
4 Variable	• SOP • POS
5 Variable	• SOP • POS

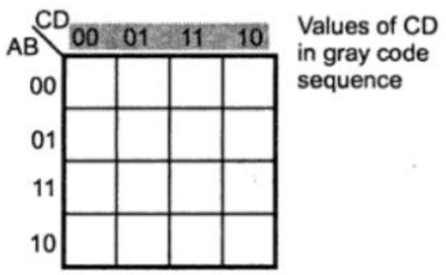
TYPES OF KMAP



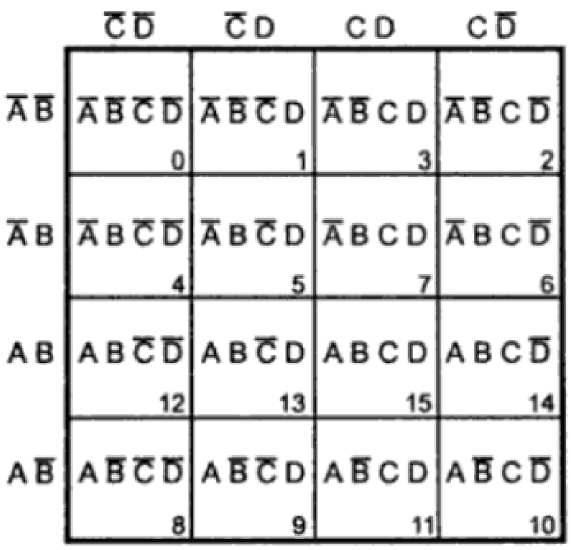
2-Variable map
(4 cells)



3-Variable map
(8 cells)



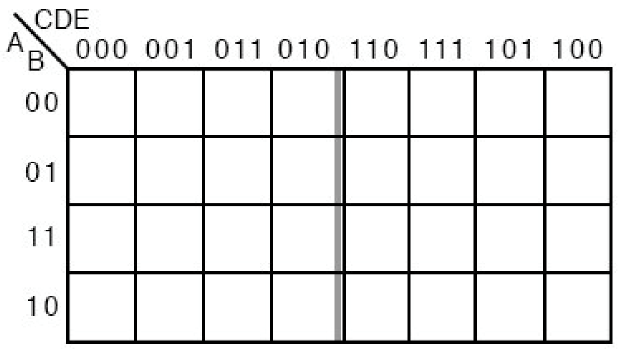
4-Variable map
(16 cells)



	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$ 0	$\overline{A}\,\overline{B}\,\overline{C}D$ 1	$\overline{A}\,\overline{B}CD$ 3	$\overline{A}\,\overline{B}C\overline{D}$ 2
$\overline{A}B$	$\overline{A}B\overline{C}\,\overline{D}$ 4	$\overline{A}B\overline{C}D$ 5	$\overline{A}BCD$ 7	$\overline{A}BC\overline{D}$ 6
AB	$AB\overline{C}\,\overline{D}$ 12	$AB\overline{C}D$ 13	$ABCD$ 15	$ABC\overline{D}$ 14
$A\overline{B}$	$A\overline{B}\,\overline{C}\,\overline{D}$ 8	$A\overline{B}\,\overline{C}D$ 9	$A\overline{B}CD$ 11	$A\overline{B}C\overline{D}$ 10

4-Variable map
(d)

TYPES OF KMAP



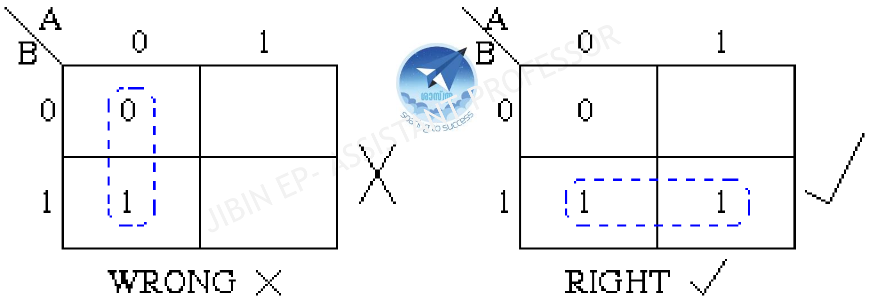
AB \ CDE	000	001	011	010	110	111	101	100
00								
01								
11								
10								

5-variable Karnaugh map (Gray code)

RULES OF KMAP

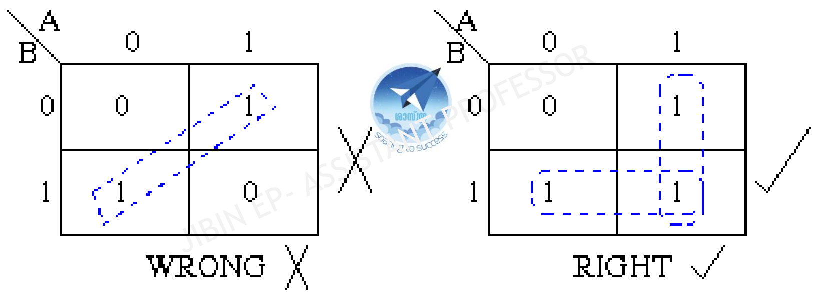
RULE 1

Groups may not include any cell containing a zero



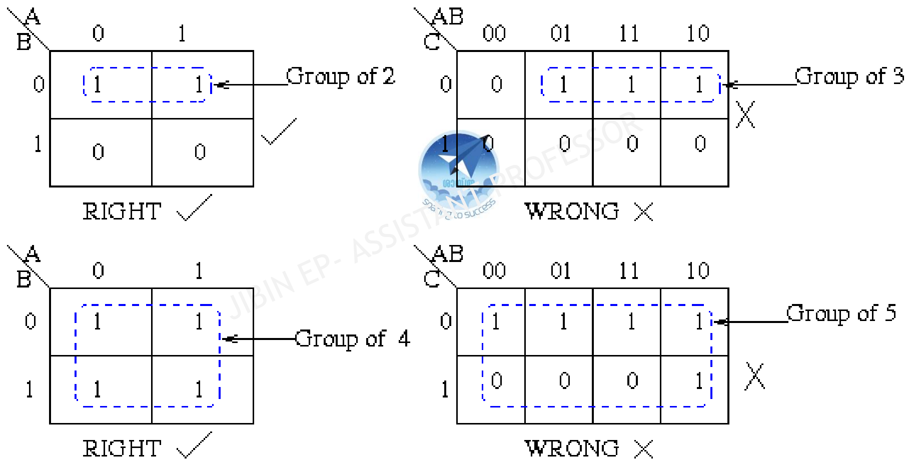
RULE 2

Groups may be horizontal or vertical, but not diagonal



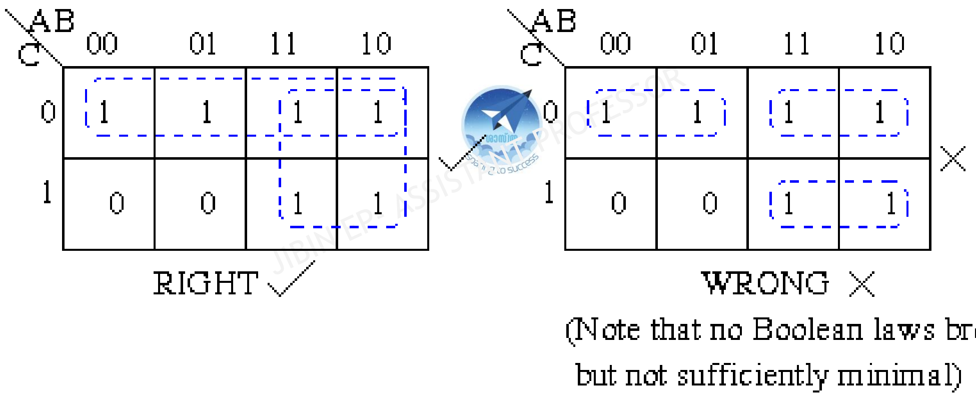
RULE 3

Groups must contain 1, 2, 4, 8,16 or in general 2^n cells.



RULE 4

Each group should be as large as possible.

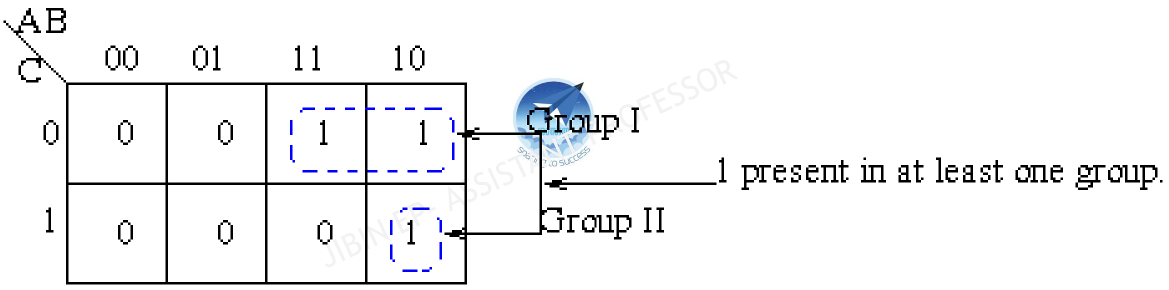


C \ AB	00	01	11	10
0	1	1	1	1
1	0	0	1	1

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

RULE 5

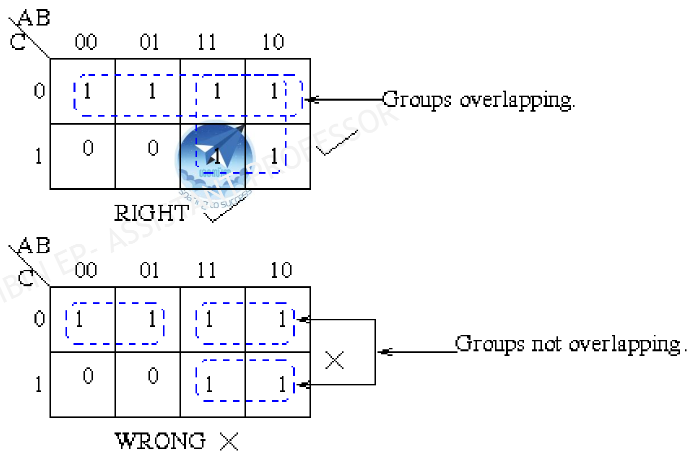
Each cell containing a one must be in at least one group.



C \ AB	00	01	11	10
0	0	0	1	1
1	0	0	0	1

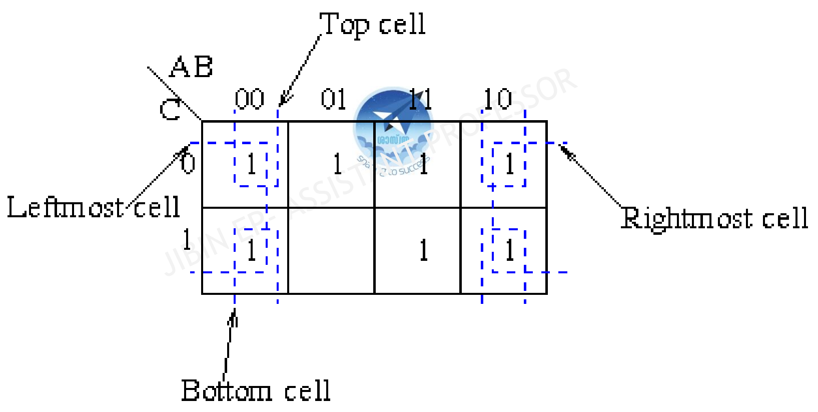
RULE 6

Groups may overlap



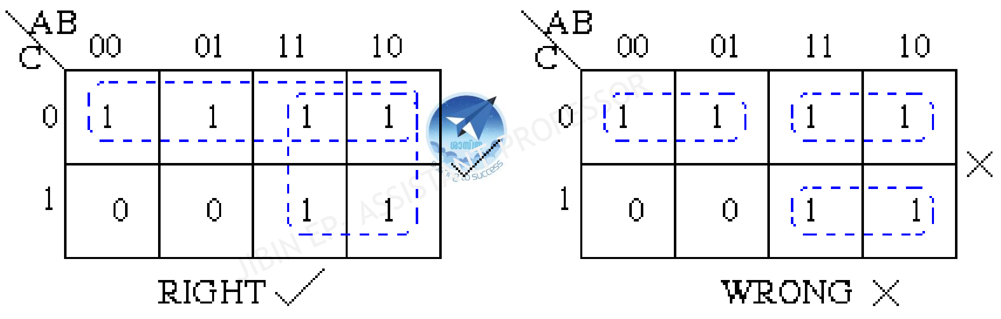
RULE 7

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.



RULE 8

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

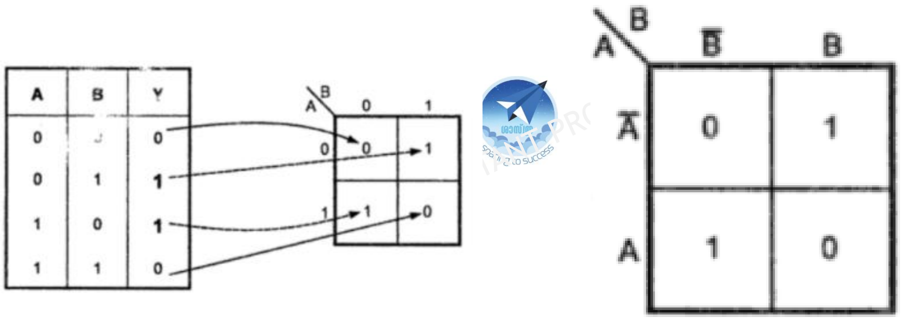


RULES OF KMAP

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

- No zeros allowed.
- No diagonals
- Only power of 2 number of cells in each group.
- Groups should be as large as possible.
- Every one must be in at least one group.
- Overlapping allowed.
- Wrap around allowed.
- Only power of 2 number of cells in each group.

Representation of Truth Table on KMAP

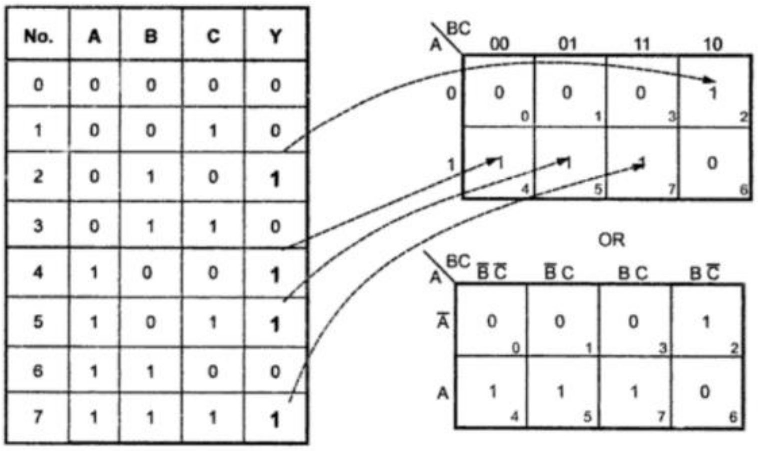


A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

A \ B	0	1
0	0	1
1	1	0

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

Representation of Truth Table on KMAP



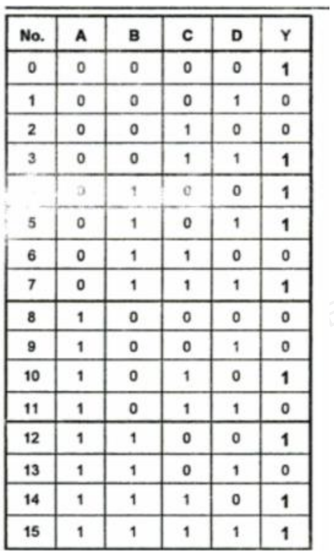
No.	A	B	C	Y
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

A \ BC	00	01	11	10
0	0 (0)	0 (1)	0 (3)	1 (2)
1	1 (4)	1 (5)	1 (7)	0 (6)

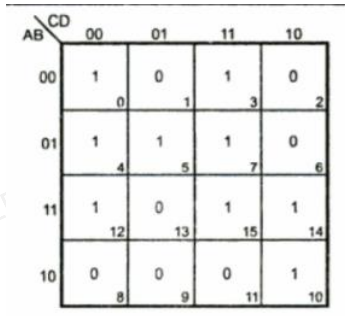
OR

A \ BC	$\overline{B}\,\overline{C}$	$\overline{B}\,C$	$B\,C$	$B\,\overline{C}$
$\overline{A}$	0 (0)	0 (1)	0 (3)	1 (2)
A	1 (4)	1 (5)	1 (7)	0 (6)

Representation of Truth Table on KMAP

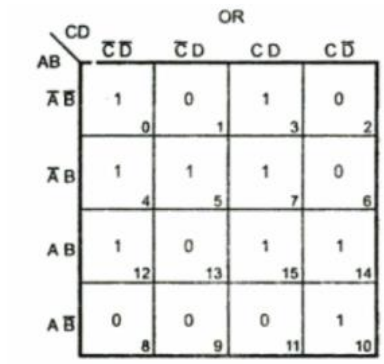


No.	A	B	C	D	Y
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	1



AB \ CD	00	01	11	10
00	1_{0}	0_{1}	1_{3}	0_{2}
01	1_{4}	1_{5}	1_{7}	0_{6}
11	1_{12}	0_{13}	1_{15}	1_{14}
10	0_{8}	0_{9}	0_{11}	1_{10}

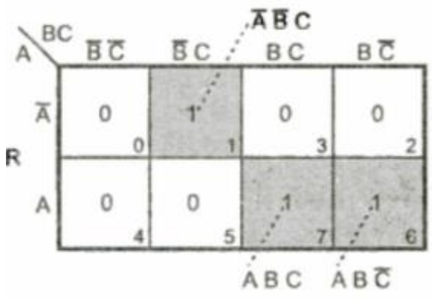
OR



AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1_{0}	0_{1}	1_{3}	0_{2}
$\bar{A}B$	1_{4}	1_{5}	1_{7}	0_{6}
AB	1_{12}	0_{13}	1_{15}	1_{14}
$A\bar{B}$	0_{8}	0_{9}	0_{11}	1_{10}

Representation of Boolean Expression on KMAP

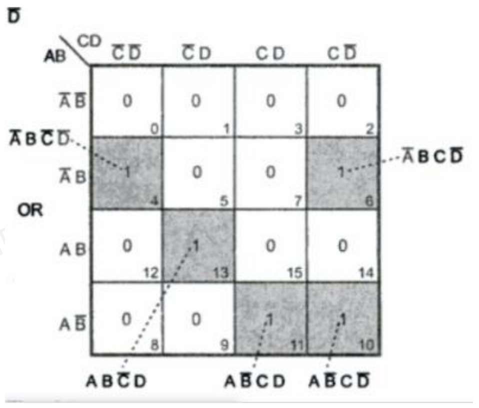
Plot Boolean expression $Y = AB\bar{C} + ABC + \bar{A}\bar{B}C$ *on the Karnaugh map.*



A \ BC	$\bar{B}\bar{C}$	$\bar{B}C$	BC	$B\bar{C}$
$\bar{A}$	0_{0}	1_{1}	0_{3}	0_{2}
A	0_{4}	0_{5}	1_{7}	1_{6}

Representation of Boolean Expression on KMAP

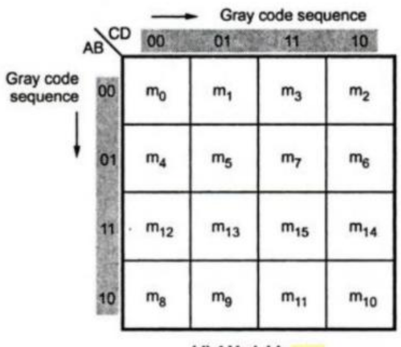
$Y = \bar{A}B\bar{C}\bar{D} + A\bar{B}C\bar{D} + \bar{A}BC\bar{D} + A\bar{B}CD + AB\bar{C}D$ on the Karnaugh map.



AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0 (0)	0 (1)	0 (3)	0 (2)
$\bar{A}B$	1 (4)	0 (5)	0 (7)	1 (6)
AB	0 (12)	1 (13)	0 (15)	0 (14)
$A\bar{B}$	0 (8)	0 (9)	1 (11)	1 (10)

Representation of Boolean Expression on KMAP

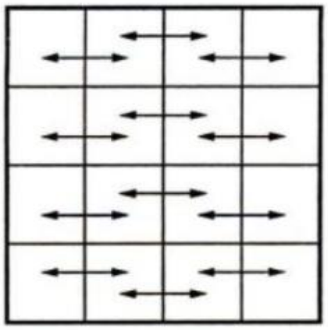
$$\sum m(1,5,7,9,15)$$

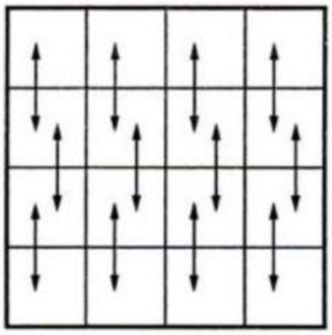


AB \ CD	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

(d) 4 Variable map

Grouping of Cells

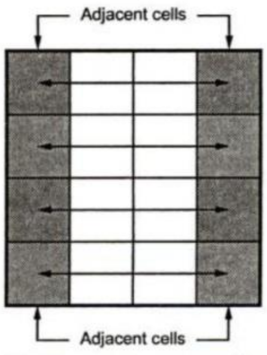

(a) Neighbouring cells in the row are adjacent


(b) Neighbouring cells in the column are adjacent

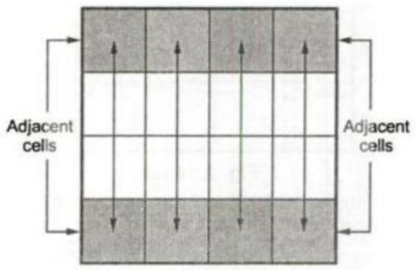
Grouping of Cells

(a) Neighbouring cells in the row are adjacent

(b) Neighbouring cells in the column are adjacent

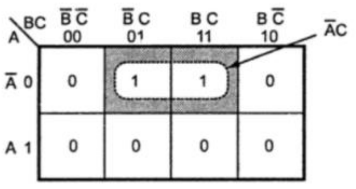


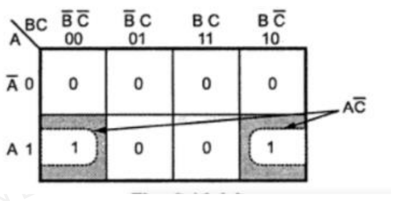
(c) Leftmost and corresponding rightmost cells are adjacent

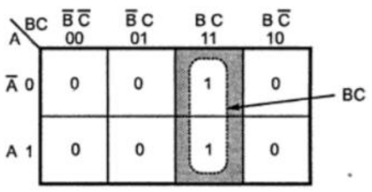


(d) Top and corresponding bottom cells are adjacent

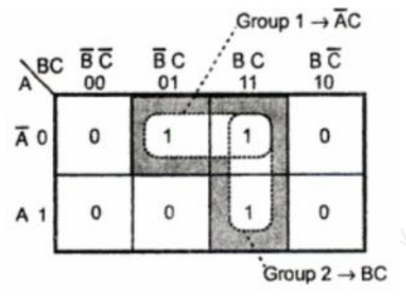
Grouping of two adjacent ones

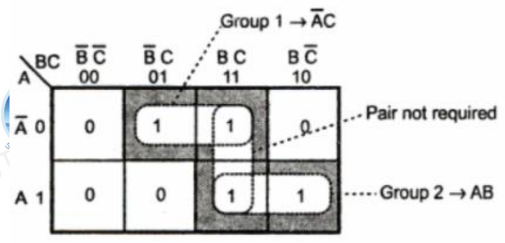




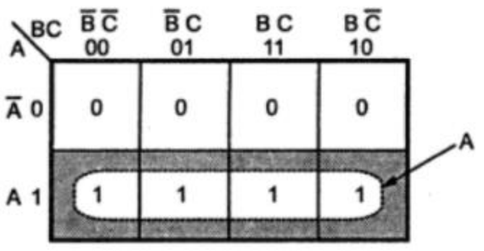


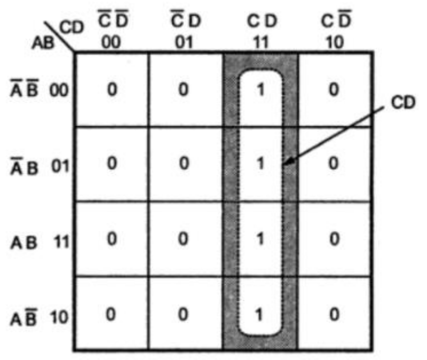
Grouping of two adjacent ones



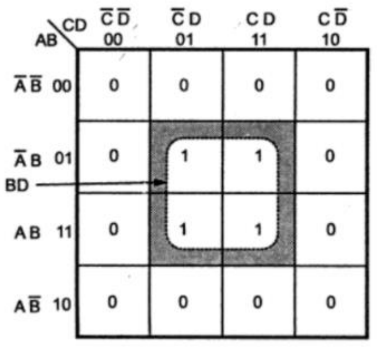


Grouping of Four adjacent ones

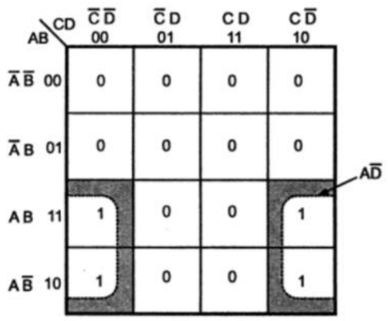




Grouping of Four adjacent ones



(c) Y = BD



(d) Y = $A\overline{D}$

Grouping of Four adjacent ones

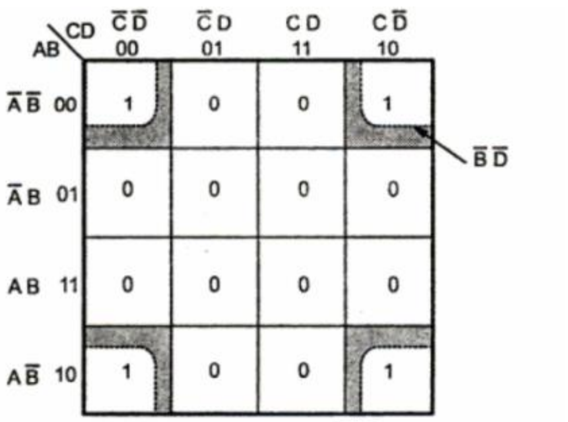
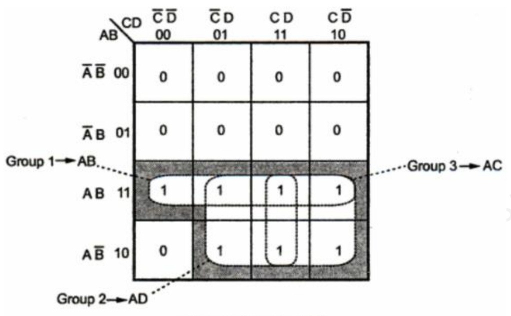


Fig. 2.15 (e) $Y = \bar{B}\bar{D}$

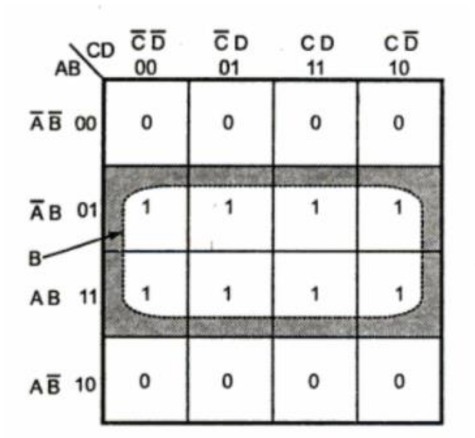
Grouping of Four adjacent ones



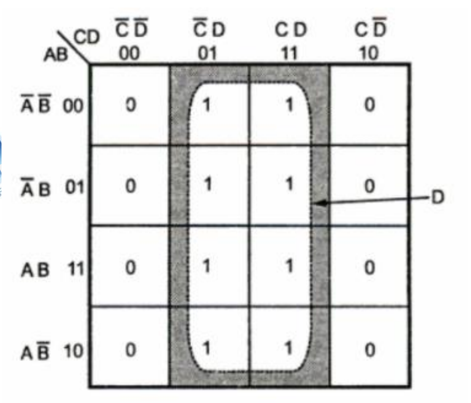
(f) $Y = AB + AD + AC$

1	1	1	1
	1	1	1

Grouping of Eight adjacent ones

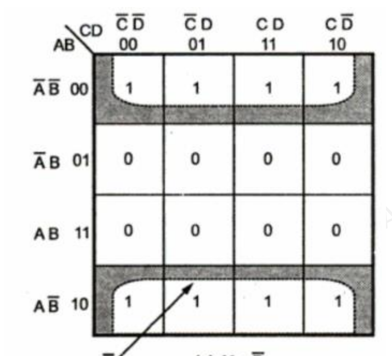


(a) Y = B

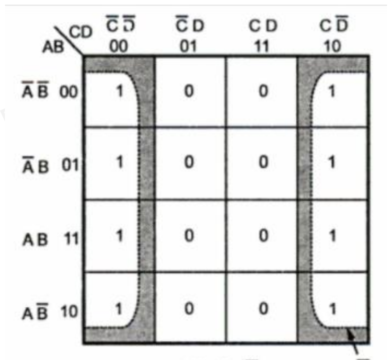


(b) Y = D

Grouping of Eight adjacent ones

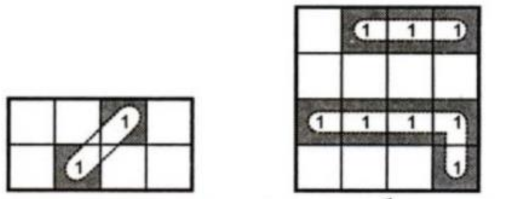


(c) Y = $\bar{B}$



(d) Y = $\bar{D}$

Illegal Grouping



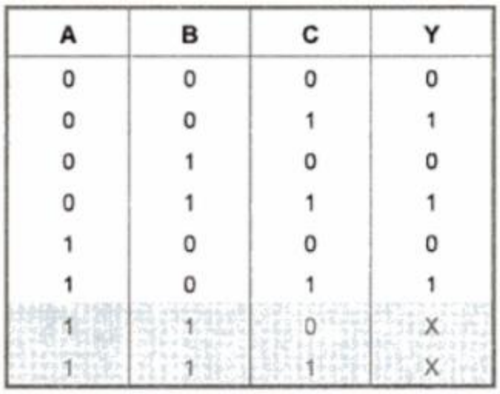
(a) Diagonal grouping is illegal

(b) Grouping of odd number of cells is illegal

DON'T CARE TERMS
(INCOMPLETELY SPECIFIED FUNCTIONS)

- In Some logic circuits, certain input conditions never occur. Therefore the corresponding output never appears
- In Such cases output level is not defined, it can be either HIGH or LOW
- These output levels are indicated by 'X' or 'd' in the truth tables and are called **don't care conditions**

DON'T CARE TERMS (INCOMPLETELY SPECIFIED FUNCTIONS)

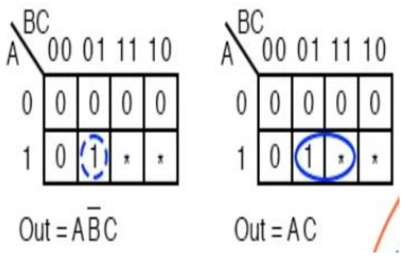


A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	X
1	1	1	X

Here the output levels are defined for input conditions from 000 to 101
For remaining two conditions of input , output is not defined. Hence these are called **Don't care conditions**

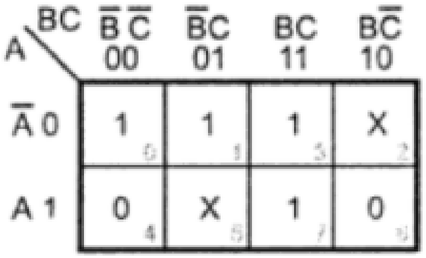
DON'T CARE TERMS

- When forming groups of cells, treat the don't care cell as either a **1** or a **0**, or ignore the don't cares.
- This is helpful if it allows us to form a larger group than would otherwise be possible without the don't cares. There is no requirement to group all or any of the don't cares.
- Only use them in a group if it simplifies the logic.

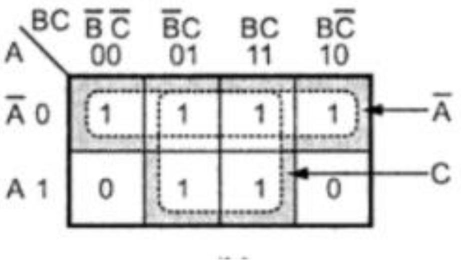


Reduce the following Boolean Expression using KMAP

$f(A,B,C) = \sum m(0,1,3,7)$+d(2,5)



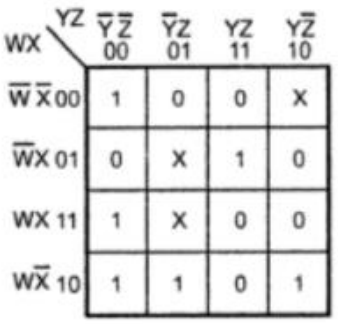
A \ BC	$\overline{B}\,\overline{C}$ 00	$\overline{B}C$ 01	BC 11	$B\overline{C}$ 10
$\overline{A}$ 0	1 (0)	1 (1)	1 (3)	X (2)
A 1	0 (4)	X (5)	1 (7)	0 (6)



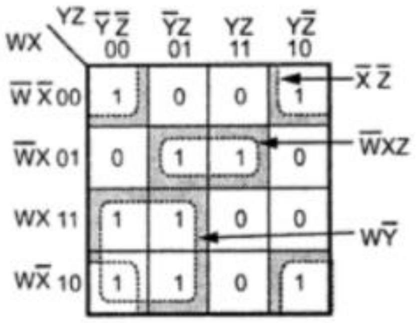
A \ BC	$\overline{B}\,\overline{C}$ 00	$\overline{B}C$ 01	BC 11	$B\overline{C}$ 10
$\overline{A}$ 0	1	1	1	1
A 1	0	1	1	0

Reduce the following Boolean Expression using KMAP

$f(A,B,C,D) = \sum m(0,7,8,9,10,12)$+d(2,5,13)



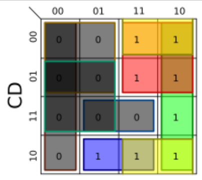
WX \ YZ	$\overline{Y}\,\overline{Z}$ 00	$\overline{Y}Z$ 01	YZ 11	$Y\overline{Z}$ 10
$\overline{W}\,\overline{X}$ 00	1	0	0	X
$\overline{W}X$ 01	0	X	1	0
WX 11	1	X	0	0
$W\overline{X}$ 10	1	1	0	1



WX \ YZ	$\overline{Y}\,\overline{Z}$ 00	$\overline{Y}Z$ 01	YZ 11	$Y\overline{Z}$ 10
$\overline{W}\,\overline{X}$ 00	1	0	0	1
$\overline{W}X$ 01	0	1	1	0
WX 11	1	1	0	0
$W\overline{X}$ 10	1	1	0	1

$$F(W, X, Y, Z) = \overline{X}\,\overline{Z} + \overline{W}\,X\,Z + W\,\overline{Y}$$

ERANAD KNOWLEDGE CITY



soaring to success

KARNAUGH MAP (PART-2)

https://youtu.be/ZMARz-Xr0F4

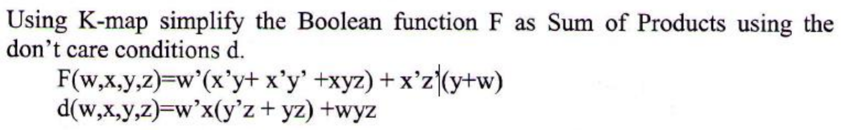
CONTENTS

PREVIOUS VIDEO

- Introduction to KMAP
- Rules of KMAP
- Representation of KMAP

Problems to workout

1

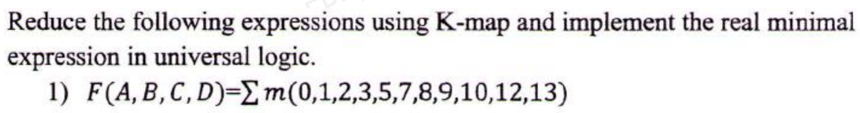


Using K-map simplify the Boolean function F as Sum of Products using the don't care conditions d.

$F(w,x,y,z)=w'(x'y+x'y'+xyz)+x'z'(y+w)$

$d(w,x,y,z)=w'x(y'z+yz)+wyz$

2



Reduce the following expressions using K-map and implement the real minimal expression in universal logic.

1) $F(A,B,C,D)=\sum m(0,1,2,3,5,7,8,9,10,12,13)$

Problems to workout

3 b) Simplify the given Boolean function $F(w, x, y, z) = \sum(2, 3, 12, 13, 14, 15)$

i) Sum of Products and ii) Product of Sums (use K Map)

4 Simplify the Boolean function $F(w, x, y, z) = \sum m(0, 5, 7, 8, 9, 10, 11, 14, 15)$

5 **Example 3.10 :** *Reduce the following function using Karnaugh map technique and implement using basic gates*

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

Problems to workout

6 **Example 3.9 :** *Simplify the logic function specified by the truth Table 3.1 using the Karnaugh map method. Y is the output variable, and A, B and C are the input variables.*

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

Problems to workout

7

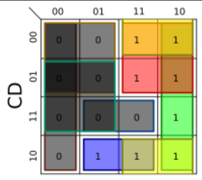
Example 3.8 : *Reduce the following function to its minimum sum of products form :*

$$Y = \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}B\overline{C}D + \overline{A}BCD + \overline{A}BC\overline{D} + AB\overline{C}\,\overline{D} + AB\overline{C}D + ABCD + A\overline{B}CD$$

8

Example 3.7 : *Reduce the following four variable function to its minimum sum of products form :*

$$Y = \overline{A}\,\overline{B}C\overline{D} + ABC\overline{D} + A\overline{B}C\overline{D} + A\overline{B}CD + A\overline{B}\,\overline{C}\,\overline{D} + AB\overline{C}\,\overline{D} + \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$$



KARNAUGH MAP (PART-3)

https://youtu.be/ezp0B2F6L0k

SIMPLIFICATION OF POS EXPRESSION (MAXTERM KMAP)X

STPES TO SOLVE

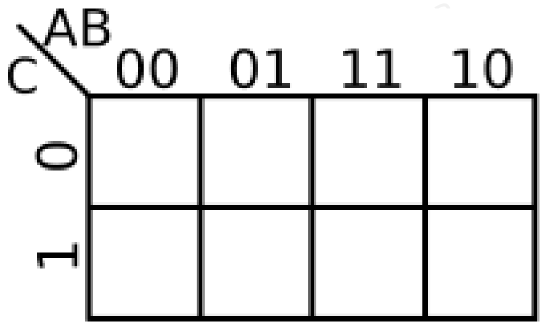
1. • Plot the K-Map and place 0’s in those cells corresponds to the 0s in the TT or Maxterms in the Product of Sum Expression
2. • Check the K-map for adjacent 0’s and encircle those 0’s
3. • Form the simplified POS Expression by taking POS Terms of all the groups

Example 2.21 : *Minimize the expression*

$$Y = (A+B+\bar{C})(A+\bar{B}+\bar{C})(\bar{A}+\bar{B}+\bar{C})(\bar{A}+B+C)(A+B+C)$$

Solution : $(A+B+\bar{C}) = M_1$, $(A+\bar{B}+\bar{C}) = M_3$, $(\bar{A}+\bar{B}+\bar{C}) = M_7$,

$(\bar{A}+B+C) = M_4$, $(A+B+C) = M_0$



MODULE 2
(TOPIC-9)

VERILOG PROGRAMMING

https://youtu.be/TQNl363ZrAI YouTube

Verilog Programming

- Verilog is a HARDWARE DESCRIPTION LANGUAGE (HDL).
- It is a language used for describing a digital system like a network switch or a microprocessor or a memory or a flip–flop.
- It means, by using a HDL we can describe any digital hardware at any level.
- Designs, which are described in HDL are independent of technology, very easy for designing and debugging, and are normally more useful than schematics, particularly for large circuits

Verilog Operators

- Arithmetic
- Logical
- Relational
- Equity
- Bit Wise
- Shift

1.Arithmetic Operations

Operator Symbol	Operations Performed
*	Multiply
/	Divide
+	Add
-	Subtract
%	Modulus
**	Power (Exponent)

2.Logic Operations

Operator Symbol	Operations Performed
!	Logical Negation
&&	Logical AND
||	Logical OR

3.Relational Operator

Operator Symbol	Operations Performed
>	Greater than
<	Less than
>=	Greater than or equal
>=	Less than or equal

4.EquityOperator

Operator Symbol	Operations Performed
==	Equality
!=	Inequality

5.Bitwise Operations

Operator Symbol	Operations Performed
~	Bitwise Negation
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~^ / ^~	Bitwise XNOR

6.Shift Operations

Operator Symbol	Operations Performed
>>	Right Shift
<<	Left Shift
>>>	Arithmetic Right Shift
<<<	Arithmetic Left Shift

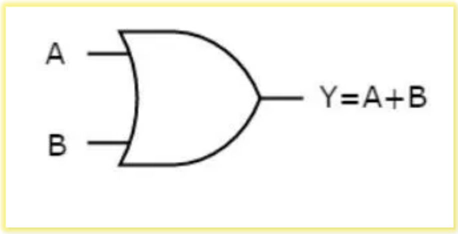
Gate Flow Modeling

- Designing a complex circuit using Basic Logic Gates is the goal of Gate Level Modeling
- We specify the gates of the circuit in our code. Verilog supports describing circuits using basic logic gates as predefined primitives.
- These primitives are instantiated like modules except that they are predefined in Verilog and do not need a module definition.

Data flow modeling

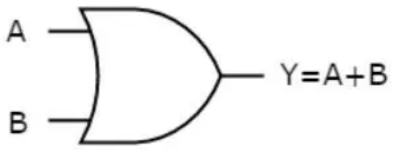
- Compared to gate-level modeling, dataflow modeling is a higher level of abstraction.
- Data Flow describes the circuits by their function rather than by their gate structure. That becomes handy because gate-level modeling becomes very complicated for large circuits because let’s face it, a digital circuit with a bunch of gates can seem quite daunting.
- Hence, dataflow modeling is a very important way of implementing the design.
- We require the boolean logic equation and **Continous Assignment Statement** to build the designs. The continuous assignments are made using the keyword assign.

OR GATE



Verilog code for OR gate using gate-level modeling

The output of the OR gate is high if at least one of the inputs is high else the output is low. Here's the logical representation of the OR gate.



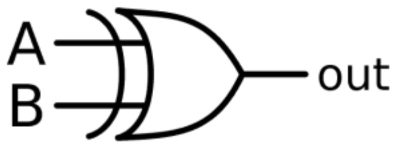
```
module OR_2_gate_level(output Y, input A,
B);
   or(Y, A, B);
End module
```

Verilog code for OR gate using Data-Flow modeling

The boolean equation of an OR gate is Y = A + B.

```
module OR_2_data_flow (output Y, input A, B);
 Assign Y = A | B;
End module
```

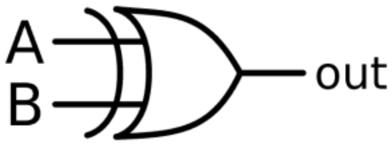
XOR GATE



Verilog code for XOR gate using Gate Level modeling

Logic Circuit of the XOR gate

Here’s the logical representation of the XOR gate. It has a graphic symbol similar to that of the OR gate, except for the additional curved line on the input side.



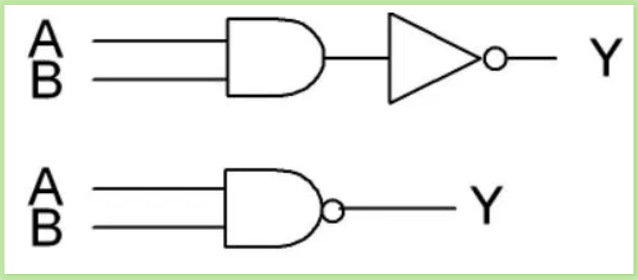
```
module XOR_2_gate_level(output Y, input A,
B);
xor (Y, A, B);
end module
```

Verilog code for OR gate using Data Flow modeling

The boolean equation of an XOR gate is
$Y = (A \oplus B)$.

```
module XOR_2_data_flow (output Y, input A,
B); assign Y = A ^ B;
end module
```

NAND GATE



Verilog code for OR gate using Data Flow modeling

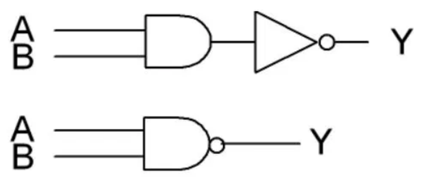
Tthe Boolean equation for a NAND gate is
Y = (A.B)’ or ~(A & B).

```
module NAND_2_data_flow (output Y, input A,
 B); assign Y = ~(A & B);
end module
```

Verilog code for NAND gate using Gate Level modeling

Logic Circuit of the NAND gate

The NAND gate is the complement of AND function. Its graphic symbol consists of an AND gate's graphic symbol followed by a small circle. Here's the logical representation of the NAND gate.



```
module NAND_2_gate_level(output Y, input
A,B); wire Yd;
and(Yd, A, B);
not(Y, Yd);
end module
```

Verilog code for NAND gate using Gate Level modeling

- wire in Verilog represents an electrical connection

```
wire Yd;
and(Yd, A, B);
not(Y, Yd);
endmodule;
```

- Here and is the operation performed on A, B, to get its output in Yd.
- Then Yd is passed through an inverter, and the output is obtained in Y.
- The compiler understands the and and the not operation the same way we do.
- endmodule terminates the module.

PREPARED BY

Jibin EP
Assistant Professor
Department of Electronics and Communication
Eranad Knowledge City Technical Campus Manje
Contact :9633908979, 70121716607
Email : epjibin@gmail.com, jibin@ekc.edu.in

YouTube

https://www.youtube.com/ShastraTechnicalInstitute

117