

MODULE 2

BOOLEAN ALGEBRA, LOGIC GATES AND COMBINATIONAL CIRCUITS

Important topics

1. Basic Theorems and Definitions.
2. SOP ,POS and canonical forms.
3. Basic gates and universal gates with figure and truth table.
4. Implement NOT,AND,OR using NOR, NAND gates.
5. Solve K map with examples
6. Design logic diagrams.

1.Axiomatic definitions of Boolean Algebra



In 1854, George Boole developed an algebraic system now called *Boolean algebra*. For the formal definition of Boolean algebra, we shall employ the postulates formulated by E. V. Huntington in 1904.

Definition

- Boolean Algebra is used to analyse and simplify the digital (logic) circuits. It uses only the binary numbers i.e. 0 and 1.
- It is an algebraic structure defined on a set of at least two elements $B = \{0, 1\}$, together with three binary operators denoted by (“+”, “.”, “-”) that satisfies the following basic identities.

1. $X + 0 = X$	OR operator	2. $X \cdot 1 = X$	AND operator
3. $X + 1 = 1$		4. $X \cdot 0 = 0$	
5. $X + X = X$		6. $X \cdot X = X$	
7. $X + \bar{X} = 1$		8. $X \cdot \bar{X} = 0$	
9. $\bar{\bar{X}} = X$	→ NOT operator		
10. $X + Y = Y + X$		11. $XY = YX$	Commutative
12. $(X + Y) + Z = X + (Y + Z)$		13. $(XY)Z = X(YZ)$	Associative
14. $X(Y + Z) = XY + XZ$		15. $X + YZ = (X + Y)(X + Z)$	Distributive
16. $\overline{X + Y} = \bar{X} \cdot \bar{Y}$		17. $\overline{X \cdot Y} = \bar{X} + \bar{Y}$	DeMorgan's

In addition to above we have

Absorption law

OR of a variable with the AND of that variable and another variable is equal to that variable itself

$$A + (A \cdot B) = A$$

AND of a variable with the OR of that variable and another variable is equal to that variable itself

$$A \cdot (A + B) = A$$

Idempotence law

AND of a variable itself is equal to that variable only

$$A \cdot A = A$$

OR of a variable itself is equal to that variable only

$$A + A = A$$

Redundant Literal Rule (RLR)

$A + A'B = A + B \rightarrow$ (OR of a variable with the AND of the complement of that variable with another variable, is equal to OR of two variables)

$A(A' + B) = AB \rightarrow$ (AND of a variable with the OR of the complement of that variable with another variable, is equal to AND of two variables)

Consensus Theorem

$$AB + A'C + BC = AB + A'C$$

$$(A + B) + (A' + C) + (B + C) = (A + B) + (A' + C)$$

2. Two-valued Boolean Algebra

A two-valued Boolean algebra is defined on a set of two elements, $B = \{0, 1\}$, with rules for binary operators $+$, \cdot , $'$ (OR, AND & NOT), with truth table as below.

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

x	x'
0	1
1	0

Postulates and Theorems of Boolean Algebra

Postulate 2	(a)	$x + 0 = x$	(b)	$x \cdot 1 = x$
Postulate 5	(a)	$x + x' = 1$	(b)	$x \cdot x' = 0$
Theorem 1	(a)	$x + x = x$	(b)	$x \cdot x = x$
Theorem 2	(a)	$x + 1 = 1$	(b)	$x \cdot 0 = 0$
Theorem 3, involution		$(x')' = x$		
Postulate 3, commutative	(a)	$x + y = y + x$	(b)	$xy = yx$
Theorem 4, associative	(a)	$x + (y + z) = (x + y) + z$	(b)	$x(yz) = (xy)z$
Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$
Theorem 6, absorption	(a)	$x + xy = x$	(b)	$x(x + y) = x$

3. Basic Theorems and Definitions

THEOREM 1(a): $x + x = x$.

Statement	Justification
$x + x = (x + x) \cdot 1$	postulate 2(b)
$= (x + x)(x + x')$	5(a)
$= x + xx'$	4(b)
$= x + 0$	5(b)
$= x$	2(a)

THEOREM 1(b): $x \cdot x = x$.

Statement	Justification
$x \cdot x = xx + 0$	postulate 2(a)
$= xx + xx'$	5(b)
$= x(x + x')$	4(a)
$= x \cdot 1$	5(a)
$= x$	2(b)

THEOREM 2(a): $x + 1 = 1$.

Statement	Justification
$x + 1 = 1 \cdot (x + 1)$	postulate 2(b)
$= (x + x')(x + 1)$	5(a)
$= x + x' \cdot 1$	4(b)
$= x + x'$	2(b)
$= 1$	5(a)

THEOREM 6(a): $x + xy = x$.

Statement	Justification
$x + xy = x \cdot 1 + xy$	postulate 2(b)
$= x(1 + y)$	4(a)
$= x(y + 1)$	3(a)
$= x \cdot 1$	2(a)
$= x$	2(b)

THEOREM 6(b): $x(x + y) = x$ by duality.

The theorems of Boolean algebra can be proven by means of truth tables. In truth tables, both sides of the relation are checked to see whether they yield identical results for all possible combinations of the variables involved. The following truth table verifies the first absorption theorem:

x	y	xy	$x + xy$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

4.Boolean Functions, Simplifications of Boolean functions using axioms and theorems.

- Every Boolean expression must be simplified to a simple form as possible in order to reduce hardware cost and complexity

Techniques for these reductions are

- Multiply all variables necessary to remove brackets
- Look for identical terms
- Look for a variable and its negation in the same term, this term can be dropped.

Boolean algebra Simplification Table

Name	AND form	OR form
Identity law	$1A = A$	$0 + A = A$
Null law	$0A = 0$	$1 + A = 1$
Idempotent law	$AA = A$	$A + A = A$
Inverse law	$A\bar{A} = 0$	$A + \bar{A} = 1$
Commutative law	$AB = BA$	$A + B = B + A$
Associative law	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributive law	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption law	$A(A + B) = A$	$A + AB = A$
De Morgan's law	$\overline{AB} = \bar{A} + \bar{B}$	$\overline{A + B} = \bar{A}\bar{B}$

Examples:

1. Expression $f(A,B)=A.(A+B)$ reduced to A

The Boolean Expression: $A.(A + B)$

Multiplying out the brackets gives us:

	$A.(A+B)$	Start
multiply:	$A.A + A.B$	Distributive Law
but:	$A.A = A$	Idempotent Law
then:	$A + A.B$	Reduction
thus:	$A.(1 + B)$	Annulment Law
equals to:	A	Absorption Law

2.Expression $f(A,B,C)=(A+B)(A+C)$ reduced to $A+B.C$

Boolean Expression: $(A + B)(A + C)$

Again, multiplying out the brackets gives us:

	$(A + B)(A + C)$	Start
multiply:	$A.A + A.C + A.B + B.C$	Distributive law
but:	$A.A = A$	Idempotent Law
then:	$A + A.C + A.B + B.C$	Reduction
however:	$A + A.C = A$	Absorption Law
thus:	$A + A.B + B.C$	Distributive Law
again:	$A + A.B = A$	Absorption Law
thus:	$A + B.C$	Result

3.Expression $f(A,B,C)=AB(B'C+AC)$ reduced to ABC

Boolean Expression: $AB(\overline{B}C + AC)$

	$AB(\overline{B}C + AC)$	Start
multiply	$A.B.\overline{B}.C + A.B.A.C$	Distributive Law
again:	$A.A = A$	Idempotent Law
then:	$A.B.\overline{B}.C + A.B.C$	Reduction
but:	$B.\overline{B} = 0$	Complement Law
so:	$A.0.C + A.B.C$	Reduction
becomes:	$0 + A.B.C$	Reduction
as:	$0 + A.B.C = A.B.C$	Identity Law
thus:	ABC	Result

5.Standard forms-POS, SOP and Canonical Form of Boolean Functions.

To implement a Boolean function with lesser number of gates we have to minimize literals(variables)and the number of terms. Boolean variables are either in complimented form or

in uncomplimented form and the terms are arranged in one of the two standard forms of Boolean functions

1. Sum of Product form (SOP)
2. Product of Sum form (POS)

Sum of Product (SOP) Form

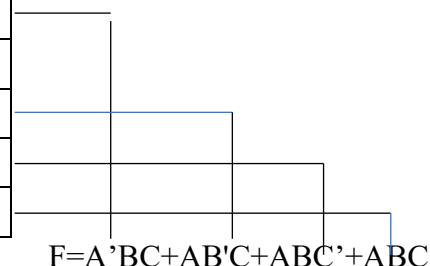
- In this SOP form of Boolean function representation, the variables are operated by AND (product) to form a product term and all these product terms are ORed (summed or added) together to get the final function.
- A sum-of-products form can be formed by adding (or summing) two or more product terms using a Boolean addition operation.
- Here the product terms are defined by using the AND operation and the sum term is defined by using OR operation

SOP form can be obtained by

- Writing an AND term for each input combination, which produces HIGH output.
- Writing the input variables if the value is 1, and write the complement of the variable if its value is 0.
- OR the AND terms to obtain the output function.
- Ex: Boolean expression $f(A,B,C) = A'BC + AB'C + ABC' + ABC$
By taking binary values expression can be written as $\rightarrow 011 + 101 + 110 + 111$
In truth table mark 1 corresponding to the terms in the expression.

Truth Table

A	B	C	Y / OUTPUT
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



Canonical SOP

If each term in SOP contains all the literals then these are known as canonical(standard) SOP expression

Eg. 1: $AB' + A'B \rightarrow$ Canonical form

Eg 2: $AB' + ABC \rightarrow$ Non Canonical since the literal C is missing in the first term.

To convert an expression from non canonical SOP to Canonical SOP

- Multiply each SOP by $(X + X')$ where X is the missing variable & expand
- Repeat above steps until all resulting product terms contains all the variable in either complimented or normal form

Above Eg.2 Expression can be covert to canonical form by multiplying $(C+C')$ to first term

$$\text{ie. } AB' \cdot (C+C') + ABC \rightarrow AB'C + AB'C' + ABC$$

Product of Sums (POS) Form

In this POS form, all the variables are ORed, i.e. written as sums to form sum terms. All these sum terms are ANDed (multiplied) together to get the product-of-sum form. This form is exactly opposite to the SOP form. So this can also be said as "Dual of SOP form". POS form can be obtained by

- Writing an OR term for each input combination, which produces LOW output.
- Writing the input variables if the value is 0, and write the complement of the variable if its value is 1.
- AND the OR terms to obtain the output function.

Ex: Boolean expression for function $F = (A + B + C) (A + B + C') (A + B' + C) (A' + B + C)$

By taking binary values expression can be written as $\rightarrow (0+0+0)(0+0+1)(0+1+0)(1+0+0)$

In truth table mark 1 corresponding to the terms in the expression.

Truth Table

A	B	C	Y / OUTPUT
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

$$F = (A + B + C) (A + B + C') (A + B' + C) (A' + B + C)$$

Converting POS to canonical POS form

ADD by $(X + X')$ where X is the missing variable

Apply rule $\rightarrow A + BC = (A + B) (A + C)$

Repeat above steps until all resulting sum terms contains all the variable in either complimented or normal form

Eg. 1: $(A+B')(A'+B) \rightarrow$ Canonical form

Eg 2: $(A+B')(A+B+C) \rightarrow$ Non Canonical since the literal C is missing in the first term.

Above Eg.2 Expression can be covert to canonical form by multiplying $(C+C')$ to first term

$$\text{ie. } (A+B') + (C.C') \cdot (A+B+C) \rightarrow (A+B'+C)(A+B'+C')(A+B+C)$$

Example 2:

$$F(A, B) = (A + B) \cdot A'$$

$$F(A, B) = \underbrace{(A + B)}_{\text{Max term}} \underbrace{A'}_{\text{Not a max term}}$$

$$= (A + B) \cdot (A' + (BB')) \quad (\because BB' = 0)$$

$$= (A + B) \cdot (A' + B) \cdot (A' + B')$$

Max term and Min Term

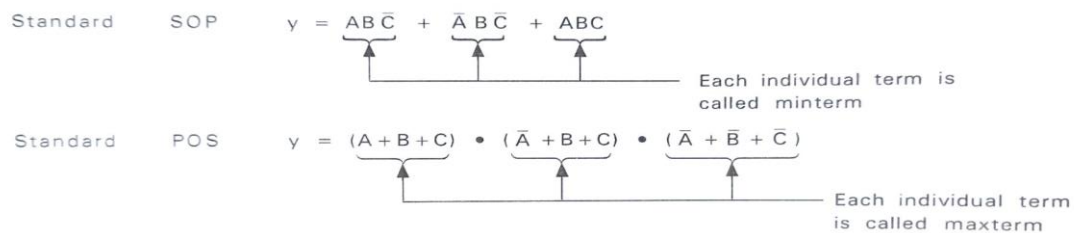


Fig. 1.1 Concept of Maxterm and Minterm

Minterm

0 → represents complement variable

1 → represents normal variable

Maxterm

0 → represents normal variable

1 → represents complement variable

Min term(m) or (Σ)	MAXTERM(M) or (Π)
<ul style="list-style-type: none"> Each individual product term in SOP is called minterm(m0,m1,m2..) 0 → A' 1 → A A binary variable may appear in its normal form and compliment form Consider 2 binary variables , A & B combined with an AND operation Since each variable may appear in either form, there are four possible combinations; A'B' , A'B , AB' & AB Each of these four AND terms is called a minterm or standard product 2^n different minterm can be obtained by writing the binary equivalent of numbers 	<ul style="list-style-type: none"> Each individual sum term in POS is called maxterm(M1,M2,M3....) 0 → A 1 → A' These are the sum terms which contains all the variables in either normal form or compliment form 2^n different maxterm can be obtained by writing the binary equivalent of numbers

Variables			Min terms	Max terms
A	B	C	m_i	M_i
0	0	0	$A' B' C' = m_0$	$A + B + C = M_0$
0	0	1	$A' B' C = m_1$	$A + B + C' = M_1$
0	1	0	$A' B C' = m_2$	$A + B' + C = M_2$
0	1	1	$A' B C = m_3$	$A + B' + C' = M_3$
1	0	0	$A B' C' = m_4$	$A' + B + C = M_4$
1	0	1	$A B' C = m_5$	$A' + B + C' = M_5$
1	1	0	$A B C' = m_6$	$A' + B' + C = M_6$
1	1	1	$A B C = m_7$	$A' + B' + C' = M_7$

KARNAUGH MAP (K MAP)

- A Karnaugh map or a K-map refers to a pictorial method that is utilized to minimize various Boolean expressions without using the Boolean algebra theorems along with the equation manipulations.
- A Karnaugh map can be a special version of the truth table.
- We can easily minimize various expressions that have 2 to 4 variables using a K-map.
- K-map can easily take two forms, namely, Sum of Product or SOP and Product of Sum or POS
- In K-map, if the number of variables is three, the number of cells is $2^3=8$, and if the number of variables is four, the number of cells is $2^4=16$.
- The K-map takes the SOP and POS forms. The K-map grid is filled using 0's(for POS) and 1's(SOP). The K-map is solved by making group.
- CELL: Smallest unit of a K map. Input variables are cell coordinates and the output variable is cell content
- PAIR : A group of two adjacent cells in K map . A pair cancel 1 variable in a K map simplification
- QUAD :A group of four adjacent cells in K map . A quad cancel 2 variables in a K map simplification
- OCTET: A group of eight adjacent cells in K map . A quad cancel 3 variable in a K map simplification

Solving an Expression Using K-Map

1. Select a K-map according to the total number of variables.
2. Identify maxterms or minterms as given in the problem.
3. For SOP, put the 1's in the blocks of the K-map with respect to the minterms (elsewhere 0's).
4. For POS, putting 0's in the blocks of the K-map with respect to the maxterms (elsewhere 1's)
5. Making rectangular groups that contain the total terms in the power of two, such as 2,4,8 .. and trying to cover as many numbers of elements as we can in a single group.
6. From the groups that have been created in step 5, find the product terms and then sum them up for the SOP form. Or find the sum terms and multiply them for POS form

2-Variable K MAP

- Number of variables n is 2 and hence $2^2=4$ cells are needed to form 2 variable K map
- Coordinates of each cell corresponds to a unique combination of two input variables A, B

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

3Variable K MAP

- Number of variables n is 3 and hence $2^3=8$ cells are needed to form 3 variable K map
- Coordinates of each cell corresponds to a unique combination of three input variables A, B

AB \ C	0	1
00	m_0	m_1
01	m_2	m_3
11	m_6	m_7
10	m_4	m_5

A \ BC	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

www.electricaltechnology.org

Simplification rules through K map

- Group 0's with 0's and 1's with 1's
- Group can overlap each other
- Group can contains 2^n number of cells
- Group can have only be horizontal or vertical
- Each group should be as large as possible
- Opposite grouping or corner grouping is allowed
- There should be a few groups as possible

$$F(A,B) = \Sigma 1,2,3 \text{ or } \Pi 0)$$

For SOP

		B	
		\overline{B} 0	B 1
A	\overline{A} 0	0	1
	A 1	1	1
		0	1
		2	3

B + A

For POS

		B	
		B 0	\overline{B} 1
A	A 0	0	1
	\overline{A} 1	1	1
		0	1
		2	3

A + B

Example 3: $F(P,Q,R,S) = \Sigma 0,2,5,7,8,10,13,15$

Ans: SOP $\rightarrow QS + Q'S'$

		RS			
		00	01	11	10
PQ	00	1	0	0	1
	01	0	1	1	0
	11	0	1	1	0
	10	1	0	0	1
		0	1	3	2
		4	5	7	6
		12	13	15	14
		8	9	11	10

as k-map is assumed to be connected so we can make group this way

as we have to take maxm. elements in a group so we've made 1 group of 4 1's not 2 groups of 2 1's

Example 4:

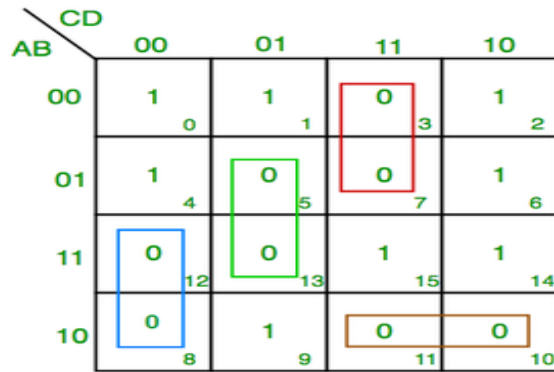
$F(A,B,C) = \pi(0,3,6,7)$

Reduced to $(A' + B')(B' + C')(A + B + C)$

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

Example 5:

$F(A,B,C,D) = \pi(3,5,7,8,10,11,12,13) \rightarrow (C+D'+B')(C'+D'+A).(A'+C+D).(A'+B+C')$



Don't Care Conditions

The “Don't Care” conditions allow us to replace the empty cell of a K map to form a grouping of the variables which is larger than that of forming groups without don't care. While forming groups of cells, we can consider a “Don't Care” cell as 1 or 0 or we can also ignore that cell. Therefore, the “Don't Care” condition can help us to form a larger group of cells. A Don't Care cell can be represented by a cross(X)

<u>SOP including don't care</u>	<u>POS including don't care</u>
<p>$f = m(1, 5, 6, 11, 12, 13, 14) + d(4)$</p> <p>Reduced to $f = BC' + BD' + A'C'D + AB'CD$</p>	<p>$F(A, B, C, D) = M(6, 7, 8, 9) + d(12, 13, 14, 15)$</p> <p>Reduced to $F = (A' + C)(B' + C')$</p>

Simplify using K-map.

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

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

Solution :

Where m indicates the minterms representation and d indicates the don't care conditions.

Enter 1 to the cells 1, 3, 7, 11, 15

Enter X to the cells 0, 2, 5

	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$$F(A, B, C, D) = \bar{A}D + \bar{A}\bar{B} + CD$$

You can group the don't care (X) with 1, but should not make grouping for only don't care grouping should always have at least one 1.

Simplify the Boolean expressing using K-map.

$$F(A, B, C, D) = \Pi M(4, 5, 6, 7, 8, 12) \cdot d(0, 1, 2, 3, 9, 11, 14)$$

Solution :

Enter 0 to the cells 4, 5, 6, 7, 8 and 12

Enter X to the cells 0, 1, 2, 3, 9, 11 and 14

CD

AB

	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$\bar{A} + C + \bar{D}$

Invalid Grouping because it does not have atleast one 0

Need not worry about this don't care case

$$F(A, B, C, D) = A(C + \bar{D})$$




Refer more examples done in classroom.

Basic and universal gates-Representation, truth table,

Logic Gates





- Logic gate is an electronic circuit which makes logic decisions
- It have only one output and two or more inputs except for the not gate, which has only one input
- Output signals appears only for certain combinations of input signals
- Gates do the manipulation of binary information
- 3 basic logic gates: are OR gate , NOT gate, AND gate
- Logic gates are the building blocks and are available in the form of various IC families
- Input and output relationship can be represented in a tabular form in a truth table

BASIC GATES

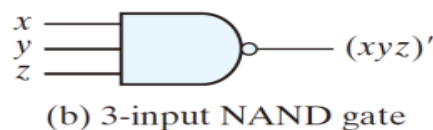
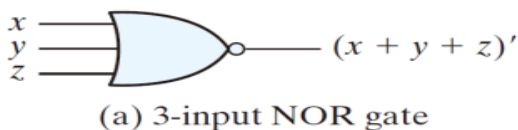
AND		$F = x \cdot y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = x + y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	1
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
Inverter NOT GATE /		$F = x'$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	F	0	1	1	0									
x	F																	
0	1																	
1	0																	

UNIVERSAL GATES,EX-OR, and EX-NOR

NAND and NOR gates are called UNIVERSAL GATES, because all other gates can be realised using the NAND and NOR gates.

NAND		$F = (xy)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = (x + y)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	0
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

3 INPUT NOR & NAND



NOT GATE

NOT gate has only one input and one output
It performs inversion or complementation

- HIGH input produce LOW output
- LOW input produce HIGH output

Hence output is the compliment of its input

Symbol	Truth Table	
<p>Inverter or NOT Gate</p>	A	Q
	0	1
	1	0
Boolean Expression $Q = \text{NOT } A, \bar{A}$		

AND GATE

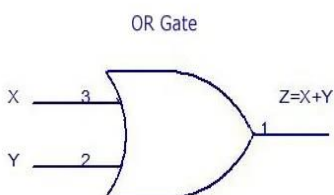
- Output of an AND gate attains state 1 if and only if all the inputs are in state 1.
- The Boolean expression of AND gate is $Y = A.B$
- It performs logical multiplication
- When any of the inputs are LOW, the output is LOW
- AND gate is composed of two or more inputs and only one output

Symbol	Truth Table		
<p>2-input AND Gate</p>	A	B	Q
	0	0	0
	0	1	0
	1	0	0
	1	1	1
Boolean Expression $Q = A.B, A \text{ AND } B$			

OR GATE

- Output of an OR gate attains state 1 if one or more inputs attain state 1.
- The Boolean expression of the OR gate is $Y = A + B$, read as Y equals A 'OR' B.
- It performs logical addition
- OR gate has two or more inputs and one output
- HIGH on the output is produced when any of the inputs are HIGH

2 Input OR Gate



TRUTH TABLE		
INPUTS		OUTPUT
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

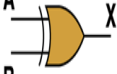
ExOR-GATE

XOR or Ex-OR gate is a special type of gate.

It can be used in the half adder, full adder and subtractor.

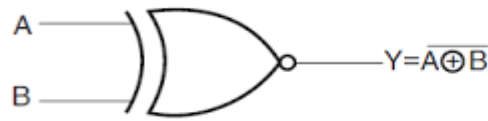
The exclusive-OR gate is abbreviated as EX-OR gate or sometime as X-OR gate.

Boolean Expression $X = A'B + AB'$

Boolean Expression	Logic Diagram Symbol	Truth Table															
$X = A \oplus B$		<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	X	0	0	0	0	1	1	1	0	1	1	1	0
A	B	X															
0	0	0															
0	1	1															
1	0	1															
1	1	0															

Ex-NOR GATE

- XNOR gate is a special type of gate. It can be used in the half adder, full adder and subtractor.
- The exclusive-NOR gate is abbreviated as EX-NOR gate or sometime as X-NOR gate



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

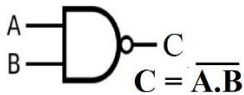
$$Y = (\overline{A \oplus B}) = (A.B + \overline{A}.\overline{B})$$

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

NAND GATE

- NAND gate is actually a combination of two logic gates: AND gate followed by NOT gate.
- So its output is complement of the output of an AND gate.
- This gate can have minimum two inputs, output is always one.
- By using only NAND gates, we can realize all logic functions: AND, OR, NOT, X-OR, X-NOR, NOR. So this gate is also called universal gate

NAND GATE



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

Truth Table

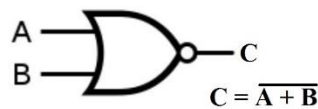
INPUT		OUTPUT
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

ProjectIoT123.com

NOR GATE

- NOR gate is actually a combination of two logic gates: OR gate followed by NOT gate.
- So its output is complement of the output of an OR gate.
- This gate can have minimum two inputs, output is always one.
- By using only NOR gates, we can realize all logic functions: AND, OR, NOT, X-OR, X-NOR, NAND. So this gate is also called universal gate.

NOR GATE



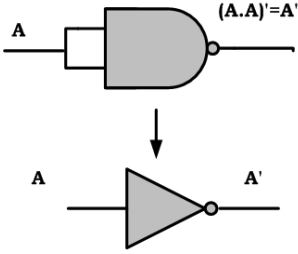
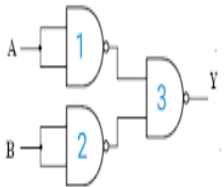
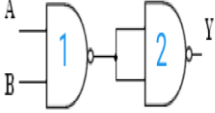
$$C = \overline{A + B}$$

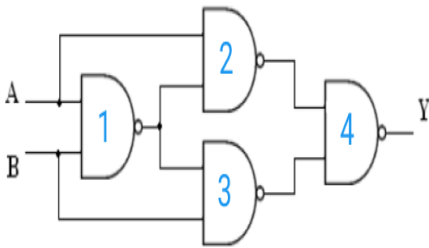
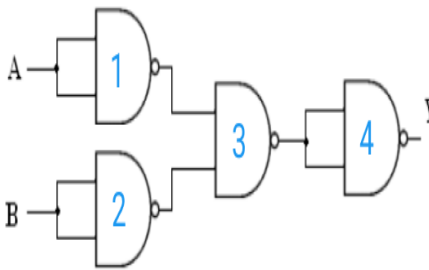
TRUTH TABLE

INPUT		OUTPUT
A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

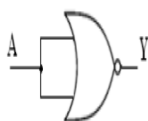
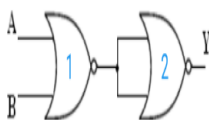
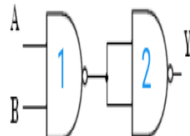
ProjectIoT123.com

Implement NOT,AND,OR ,Ex-OR, Ex-NOR NAND gates

NOT USING NAND	OR USING NAND	AND USING NAND																														
	<p>(c) OR gate: $Y = A + B$</p>  <table border="1" data-bbox="834 495 909 640"> <thead> <tr> <th>A</th><th>B</th><th>Y</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	1	<p>(b) AND gate: $Y = A \cdot B$</p>  <table border="1" data-bbox="1374 450 1457 640"> <thead> <tr> <th>A</th><th>B</th><th>Y</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	A	B	Y	0	0	0	0	1	0	1	0	0	1	1	1
A	B	Y																														
0	0	0																														
0	1	1																														
1	0	1																														
1	1	1																														
A	B	Y																														
0	0	0																														
0	1	0																														
1	0	0																														
1	1	1																														

Ex-OR USING NAND	Ex-NOR USING NAND																														
<p>(e) Ex-OR gate: $Y = A \oplus B$</p>  <table data-bbox="647 1059 754 1249"><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	0	<p>(d) NOR gate: $Y = (A + B)'$</p>  <table data-bbox="1343 1097 1450 1310"><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	Y	0	0	1	0	1	0	1	0	0	1	1	0
A	B	Y																													
0	0	0																													
0	1	1																													
1	0	1																													
1	1	0																													
A	B	Y																													
0	0	1																													
0	1	0																													
1	0	0																													
1	1	0																													

Implement NOT,AND,OR ,Ex-OR, Ex-NOR NOR gates

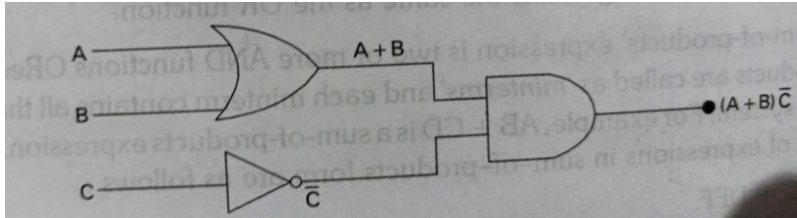
NOT USING NOR	OR USING NOR	AND USING NOR																																				
<p>(a) NOT gate: $Y = A'$</p>  <table data-bbox="450 1780 504 1904"><tr><th>A</th><th>Y</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	Y	0	1	1	0	<p>(c) OR gate: $Y = A + B$</p>  <table data-bbox="893 1780 968 1982"><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	1	<p>(b) AND gate: $Y = A \cdot B$</p>  <table data-bbox="1347 1780 1423 2004"><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Y	0	0	0	0	1	0	1	0	0	1	1	1
A	Y																																					
0	1																																					
1	0																																					
A	B	Y																																				
0	0	0																																				
0	1	1																																				
1	0	1																																				
1	1	1																																				
A	B	Y																																				
0	0	0																																				
0	1	0																																				
1	0	0																																				
1	1	1																																				

Design logic diagrams

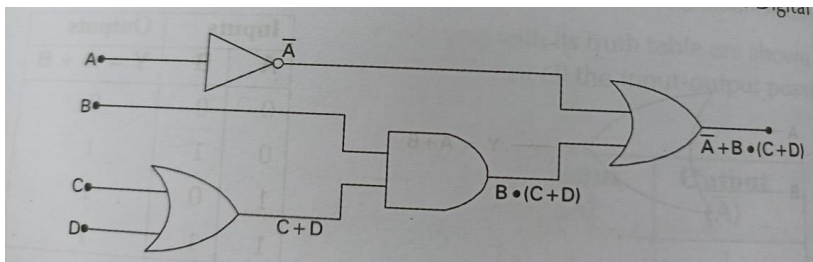
Logic gates are used for implementing any Boolean expression

Boolean expressions indicates the types of gate networks, which can be systematically progressing from input to output on the gates

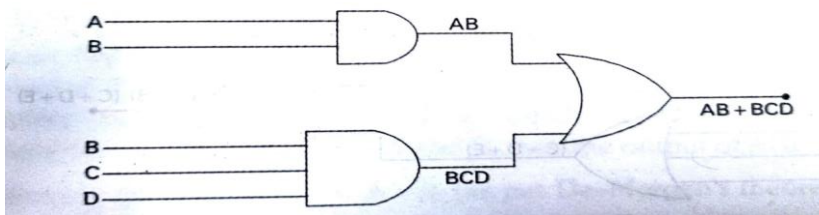
Example: Design a circuit for $(A+B).C'$



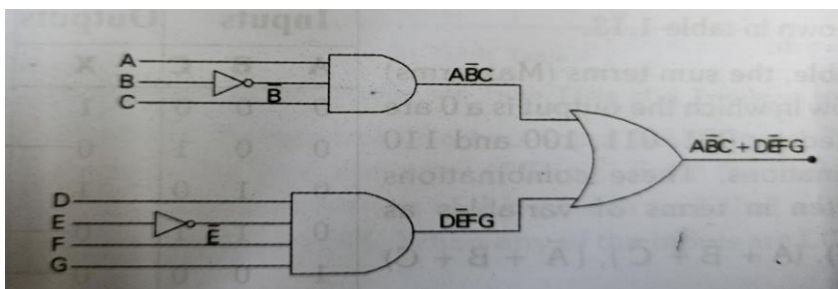
Example :2 $A'+B.(C+D)$



Example 3: $AB+BCD$



Example:4 $AB'C+DE'FG$



Questions:

Reduce Boolean expression using K map and prepare circuit.

1. $F(A, B, C, D) = \sum m(0, 1, 2, 5, 7, 8, 9, 10, 13, 15)$
2. $F(A, B, C, D) = \sum m(1, 3, 4, 6, 8, 9, 11, 13, 15) + \sum d(0, 2, 14)$
3. $F(A, B, C, D) = \prod (0, 1, 2, 4, 5, 7, 10, 15) + d(3, 8, 9)$
4. $F(A, B, C, D) = \pi(3, 5, 7, 8, 10, 11, 12, 13)$