Unit-4: Introduction to Boolean Algebra

Chapter-1 : Boolean Algebra

Learning Objective

At the end of the chapter students will:

- Learn Fundamental concepts and basic laws of Boolean algebra.
- Learn about Boolean expression and will be able to generate same.
- Understand the basic operations used in computers and other digital circuits.
- Be able to represent Boolean logic problems as - truth table .
- Understand how logic relates to computing problems.

Boolean Algebra was introduced by George Boole in 1854. He was an English mathematician, who primarily worked on differential equations and the calculus of variations. But the work for which he is best known is -application of algebraic systems to non-mathematical reasoning. He proposed that logical propositions should be expressed as algebraic equations, so now logic was reduced to algebra. In his logical deductions he replaced the operation of multiplication by AND, and addition by OR.

However, it is Claude Shannon, whose work on using Boolean Algebra to design Switching Circuits in late 1930s, became the foundation of computer Logic and Design. As Boolean logic allows things to be mapped into bits and bytes, it has application in telephone switching circuits, electronics, computers (both hardware and software), etc.

Boolean Algebra, which is algebra of two values may be (True, False) or (Yes, No) or (0, 1), is an important tool in analyzing, designing and implementing digital circuits.

Boolean Algebra is made up of

- Elements -which are variables or constants with value 1 or 0.
- Operators -which are And, Or and Not.
- Axioms and Theorems.

A boolean variable is a symbol used to represent a logical quantity. It will take value from the domain {0, 1}, and boolean constant is single digit binary value (bit) viz. 0 or 1.

There are three fundamental operators- AND, OR and NOT. AND is a binary operator, to perform logical multiplication, it is represented by '.' OR is also a binary operator, to perform logical addition. It is represented by '+'. NOT is a unary operator, to complement the operand. Not is represented as ' or $\bar{}$. Complement is the inverse of a variable/ constant. In case of boolean algebra, since the variable/constant can have value 0 or 1 so complement will be 1 or 0.

Note: A binary operator is applied on two operands and unary to one.

Just like algebra, Boolean algebra also have axioms and theorems, which describe how logical quantities behave. We know that axiom is a statement which is considered to be true, and theorems are to be proved.

First axiom is Closure Property, it states that Boolean operations (+ or .) on Boolean variables or constants will always result into a Boolean value.

Following are other axioms and theorems of Boolean algebra:

Identity: states that, sum of anything and zero, or product of anything and one, is same as the original anything. So identity with respect to + is 0, and with respect to . is 1

$A + 0 = A$ and $A.1 = A$

Commutative: property tells us, we can reverse the order of variables, that are either ORed together or ANDed together without changing the truth of the expression.

$A + B = B + A$ and $A . B = B . A$

Distributive: law states that, ORing variables and then ANDing the result with single variable is equivalent to ANDing the result with a single variable with each of the several variables and then ORing the products. Vice versa with respect to operators and terms is also true.

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

Complement: states that sum of a Boolean quantity with its complement or product of a Boolean quantity with its complement results into identity

$A + A' = 1$ and $A . A' = 0$

Indempotency: states that when we sum or product a boolaen quantity to itself, the resultant is original quantity.

$A + A = A$ and $A . A = A$

Null Element: No matter what the value of Boolean variable, the sum of variable and 1 will always be 1. Also the product of Boolean variable and 0 will always be 0.

$A + 1 = 1$ and $A . 0 = 0$

Involution: states that complementing a variable twice, or any even number of times, results in the original Boolean value.

$(A')' = A$

Associative: this property tells us that we can associate, group of sum or product variables together with parenthesis without altering the truth of the expression

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

Absorption: is also known as covering, have three forms

1. A + AB = A and A . (A + B) = A
2. A + (A' . B) = A + B and A . (A' + B) = A . B
3. (A + B) . (A' + C) . (B + C) = (A + B) . (A' + C) AND (A . B) + (A' . C) + (B . C) = A . B + A' . C

De Morgan's Theorem: states that the complement of a sum / product, equals the product / sum of the complements.

(A + B) ' = A' . B' and (A . B)' = A' + B'

Apart from these axioms and theorems, there is a principle. Duality Principle which states that starting with Boolean relation, you can device another Boolean relation by

1. Changing each OR operation to AND
2. Changing each AND operation to OR
3. Complementing the identity (0 or 1) elements.

For example dual of relation

A + 1 = 1 is A . 0 = 0.

You must have noticed that second part of axioms and theorems is actually the result of this principle only. Practical consequence of this theorem is, it is sufficient to prove half the theorem, the other half comes gratis from the principle.

If we compare Boolean Algebra with Arithmetic algebra, we will find that

- In Boolean algebra + can distribute over .
- Logical subtraction and logical division are not available in Boolean algebra, as there is no additive or multiplicative inverse available.
- Complement is available in Boolean algebra
- Two valued Boolean algebra contains 0 & 1 only.

Boolean Function -is an expression formed with boolean variable(s), boolean constant(s), boolean operator(s), parenthesis and equal sign. It will always result into a boolean value, as specified by closure property. There are three ways to represent a boolean expression/function viz.

1. Algebraic
2. Truth Table
3. Circuit Diagram

Algebraic Representation:

Axioms and Theorems written above are examples of algebraic representation, but let's have some more examples:

F = a.b

formed using two boolean variables a & b with AND operator. This function will be equal to 1, for a = 1 and b = 1, otherwise it will be 0.

F2 = xyz'

formed using three boolean variables x, y and z with AND operator. Function will be 1 if x = 1, y = 1, and z' = 1 i.e. z = 0, in all other situations F will be 0.

They can also be represented as

F (a, b) = a.b

F (x,y,z) = x.y.z'

These are examples of some simple Boolean functions. Complex boolean functions can be formed using simple expressions. Following are the examples of such functions:

f (a, b, c)= (a+b).c

f (x,y) = (x+y). (x'+y')

f (x, y, z)= x. (y+z). (x+y')

f (w, x, y, z)= y'z + wxy' + wx'z

When an expression/function contains two or more operators, the order of operations is decided by parenthesis and precedence of operators. Moving from high to low precedence, following table gives the precedence of Boolean operators.

Operator	**Precedence**
()	High
NOT	↓
AND	
OR	Low

Truth Table representation

A truth table is a chart of 1's and 0's, arranged to indicate the results of all possible input options. Number of columns in the table is decided by the number of Boolean variables in the function. Each variable will be represented in a column. For n variable expression, there will be 2 rows in the table. For filling these rows in the table, they will be the binary representation of the integers from 0 to (2 -1), in the same order.

Let's fill the table for one variable -a

a
0
1

Since the table is for one variable there will be only two rows $2^1 = 2$ rows. For two variables - a & b, the table will have 4 rows ($2^2 = 4$), listed below

a	b
0	0
0	1
1	0
1	1

Three Variables - a, b & c, 8 rows

a	b	c
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

and so on. The trick for filling the table is, starting from right most column of the table - 1st column will be combination of 0,1,0,1,…… (2^0 zeros and then same number of 1s. In 2nd column it will be 0,0,1,1,0,0,….. 2^1 zeros and then same number of ones. In 3rd column 0,0,0,0,1,1,1,1,0,0, …… (2^2 number of zeros and ones), and so on.

Now let's use one variable table, to represent a simple boolean function f (a) = a'

a	f (a)
0	1
1	0

Second column of table gives the value of Boolean function i.e. a'. This table tells that if a is true, a' will be false and vice versa, as is evident from 2nd column of table.

Truth table for the boolean function applied on two variables with all basic Boolean operators i.e. AND, OR will be

a	b	f = a.b	f = a+b
0	0	0	0

0	1	0	1
1	0	0	1
1	1	1	

Now let's represent some more functions written earlier in algebraic form using truth table:

F (x,y,z) = xyz'

x	y	z	f2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

From our earlier explanation we know that F will be one for x = 1, y = 1 and z = 0.

f (a, b, c) = (a+b).c

a	b	c	(a+b)	f3 = (a+b).c
0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	1	1
1	0	0	1	0
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1

f (x,y) = (x+y). (x'+y')

x	y	x'	y'	(x+y)	(x'+y')	f4 = (x+y).(x'+y')
0	0	1	1	0	1	0
0	1	1	0	1	1	1
1	0	0	1	1	1	1
1	1	0	0	1	0	0

Third way of representation of the Boolean function will be taken up later in the Unit.

For now let's move on to proving the various theorems of Boolean algebra:

Theorems can be proved in two ways:

i) **Algebraic Method:** In this method, we use axioms & theorems. This method is similar to what you do in mathematics.

ii) **Truth Table:** Here we use truth table to show that expression given on LHS results into same values for expression on RHS. This actually is verification of theorem, not proving.

Associative:

Theorem states that A + (B+C) = (A+B) + C and A.(B.C) = (A.B).C

Let's algebraically prove it:

Let X = [A + (B + C)] [(A + B) + C]

then

X = [(A + B) + C] A + [(A + B) + C] (B + C)
= [(A + B) A + C . A] + [(A + B) + C] (B + C)
= A + [(A + B) + C] (B + C)
= A + [(A + B) + C] B + [(A + B) + C] C
= A + (B + C)

But also

X = (A + B) [A + (B + C)] + C [A + (B + C)]
= (A + B) [A + (B + C)] + C
= A [A + (B + C)] + B [A + (B + C)] + C
= (A + B) + C

Thus A + (B + C) = (A + B) + C

We know, second form of the theorem will also be true. (Duality Principle)

Truth table method:

A	B	C	B+C	LHS A+(B+C)	(A+B)	RHS (A+B)+C
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

As LHS= RHS, hence verified

Null Element states that A + 1 = 1 and A . 0 = 0

Algebraic method:

Taking LHS

A+1	= (A+1).1	by Identity
	= (A+1). (A+A')	by Complement
	= A + (1.A')	by Distribution
	= A+ (A'.1)	by Indempotancy & Commutative
	= A+A'	by Identity
	= 1	by Complement

By applying the principle of Duality, we get A.0 = 0

Truth Table method:

A	1	LHS A+1	RHS
0	1	1	1
1	1	1	1

As LHS= RHS, hence verified

Involution states that ((A')') = A

Taking LHS

(A')'	= (A')' + 0	by Identity	
	= (A')' + (A'.A)	by Complement	
	= ((A')' + A'). ((A')' + A)	by Distribution	
	= 1. ((A')'+A)	by Complement	
	= (A')'+A	by Identity	(1)

Taking LHS again

(A')'	= (A')'. 1	by Identity	
	= (A')'. (A+A')	by Complement	
	= (A')' .A) + ((A')'.A')	by Distribution	
	= (A')' .A) + 0	by Complement	
	= (A')'. A		(2)

Substituting the value of (A')' = (A')'. A in (1), we get

(A')' + A	= (A')'.A + A	
	= A + (A')'.A	by Commuting

= A + A. (A')' by Commuting

= A by Absorption

We know, second form of the theorem will also be true. (Duality Principle)

Truth Table method:

A	(A')	LHS (A')'	RHS A
0	1	0	0
1	0	1	1

As LHS= RHS, hence verified

Idempotency states that A+A= A and A . A = A

Algebraic method:

Taking L.H.S

A+A = (A+A).1 by identity

= (A+A). (A+A') by complement

= (AA+AA') + (AA + AA') by distribution

= AA+0

= A

We know, second form of the theorem will also be true. (Duality Principle)

Truth Table method:

A	RHS A	LHS A+A
0	0	0
1	1	1

As LHS = RHS, hence verified

Absorption Law:

i) A + AB = A and A . (A + B) = A

Algebraic method:

Taking LHS

A + AB = (A.1) + (A.B) by Identity

= A. (1+B) by Distribution

= A.1 by Null Element

= A

Using Duality, we know that A.(A+B) = A is also true

Truth Table method:

A	B	A.B	LHSA+AB	RHS A
0	0	0	0	0
0	1	0	0	0
1	0	0	1	1
1	1	1	1	1

As LHS= RHS, hence verified

ii) (A + A'B) = A+ B and A . (A' + B) = A . B

Algebraic method:

Taking LHS

A + A'B = (A+ A') . (A+B) by Distribution

= 1. (A+B) by Complement

= A+B by Identity

Because of duality, A.(A'+B) = A.B is also true.

Truth Table method:

A	B	A'B	LHS A+A'B	RHS A+B
0	0	0	0	0
0	1	1	1	1
1	0	0	1	1
1	1	0	1	1

As LHS= RHS, hence verified

iii) ((A+B).(A'+C). (B+C)) = (A+B). (A'+C) AND ((A.B) + (A'.C) + (B.C)) = A.B + A'.C

Taking LHS

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

= (A+B).(A'+C). (B+C). (B+C) by Idempotency

= (A+B).(B+C).(A'+C). (B+C) by Commuting

= (B+A).(B+C).(C+A'). (C+B) by Commuting

= (B+A.C) . (C+ A'B) by Distributive

= (B+AC).C + (B+AC). A'B by Distribution

= B.C + A.C.C + B. A.'.B + A.C.A'B by Distribution

= B.C + A.(C.C) + (B.B). A'+ C. (A. A').B by Associativity and Idempotency

= B.C + A.C + B.A'+ 0 by Complement and Idempotency

= (B+A).C + B.A'+ A. A' by Complement and Distributive

= (B+A).C + (B+A).A' by Distributive

= (B+A).(C + A') by Distributive

= (A+B). (A'+C) by Commuting

Using Duality Principle we can say that A.B + A'+C + B. C = A. B. A'.C is also true

Truth Table method:

A	B	C	A+B	A'+C	B+C	LHS (A+B).(A'+C). (B+C)	RHS (A+B). (A'+C)
0	0	0	0	1	0	0	0
0	0	1	0	1	1	0	0
0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	1
1	0	0	1	0	0	0	0
1	0	1	1	1	1	1	1
1	1	0	1	0	1	0	0
1	1	1	1	1	1	1	1

As LHS= RHS, hence verified

De Morgan's Theorem:

States that (A+B)' = A'.B' and (A.B)' = A' + B'

Algebraic method:

Assuming that DeMorgan's theorem is true, then through this we can find complement of a function/expression. We also know that

X+X' = 1 and X.X'=0

So by showing that

i) (A+B) + A'.B' = 1

ii) (A+B). (A'.B') = 0

We shall establish the DeMorgan's theorem.

i) (A+B) + A'.B' =1

Taking LHS

= (A+B) + (A'. B')= ((A+B)+ A'). ((A+B)+ B') By Distribution
= (A+(B+ A')). (A+(B+ B')) By Associativity
= (A+(A'+B)). (A+(B+ B')) By Commutativity
= (A+ (A'+B)). (A+1) By Complement
= ((A+ A')+B).(A+1) By Commutativity
= (1+B). (A+1) By Complement
= 1.1 By Null element
= 1

ii) (A+B). (A'.B')= 0

Taking LHS

= (A.(A'.B') + (B. (A'.B'))
= ((A.A').B') + ((B.A'). B') By Associativity
= 0 + ((B'.A'). B') By Null Element
= 0 + ((A'.B).B') By Commutativity
= 0 + (A'+(B.B')) By Associativity
= 0 + 0 By Null Element
= 0

Truth Table method:

A	B	A+B	LHS (A+B)'	A'	B'	RHS A'.B'
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

As LHS = RHS, hence verified.

Hence De Morgan's theorem is established. Using Duality we can say that (A . B)' = A' + B'

LET'S REVISE

- **Boolean Algebra:** is an algebra that deals with binary variables and logic operations
- **Variable:** A variable is a symbol, usually an alphabet used to represent a logical quantity. It can have a 0 or 1 value
- **Boolean Function:** consists of binary variable, constants 0 & 1, logic operation symbols, parenthesis and equal to operator.
- **Complement:** A complement is the inverse of a variable and is indicated by a' or bar over the variable.
- A **binary variable** is one that can assume one of the two values 0 and 1.
- **Literal:** A Literal is a variable or the complement of a variable
- **Truth table:** it provides the basic method of describing a Boolean function.
- **List of axioms and theorems:**

Identity	A + 0 = A	A. 1 = A
Complement	A + A' = 1	A. A' = 0
Commutative	A + B = B + A	A. B = B. A
Assosiative	A + (B + C) = (A + B) + C	A. (B. C) = (A. B). C
Distributive	A. (B + C) = A. B + A. C	A + (B. C) = (A + B). (A + C)
Null Element	A + 1 = 1	A. 0 = 0
Involution	(A')' = A	
Indempotency	A + A = A	A. A = A
Absorption	A + (A. B) = A	A. (A + B) = A
	A + A'. B = A + B	A. (A' + B) = A. B
	(A+B).(A'+C).(B+C) = (A+B).(A'+C)	(A.B)+(A'.C)+(B.C) = A.B+A'.C
	(A+B) (B+C) (A'+C)=(A+B) (A'+C)	
De Morgan's	(A + B)' = A'. B'	(A. B)' = A'. B'

EXERCISE

1. The commutative law of Boolean algebra states that

   A + B = A . B

   i) True

   ii) False

2. Applying Dr. Morgan's theorem to the expression (ABC)', we get

   i) A'+B'+C'

   ii) (A+B+C)'

   iii) A+B'+C.C'

   iv) A (B+C)

3. Which Boolean Algebra theorem allow us to group operands in an expression in any order without affecting the results of the operation:

   i) Associative

   ii) Commutative

   iii) Boolean

   iv) Distributive

4. Applying Dr. Morgans theorem to the expression ((x+y)'+z)', we get _____

   i) (x+y)z

   ii) (x'+y')z

   iii) (x+y)z'

   iv) (x'+y')z'

5. Applying the distributive law to the expression A (B+C'+D), we get_____

   i) AB+AC+AD

   ii) ABCD

   iii) A+B+C+D

   iv) AB+AC'+AD

6. A variable is a symbol used to represent a logical quantity that can have a value of 1 or 0

   i) True

   ii) False

7. The OR operation is Boolean multiplication and the AND operation is Boolean addition
   i) True
   ii) False
8. In Boolean Algebra, A+1 = 1
   i) True
   ii) False
9. In the Commutative law, in ORing and ANDing of two variables. The order in which the variables are ORed or ANDed makes no difference
   i) True
   ii) False
10. Verify using truth table, DeMorgan's theorem for three variables.
11. Construct a truth table for the following functions
   i) F1(A,B,C) = A + BC'
   ii) F2(A,B,C) = AC + BC + AB'
   iii) F3(w,x,y,z) = y'z + wxy' + wxz' + wx'z
   iv) (x + y) . (y + z) . (z + x)
12. Expand the following expressions using De Morgan's theorem
   i) F1(A,B,C) = (AB)'(ABC)'(A'C)'
   ii) F2(A,B,C) = ((AB + B'C).(AC + A'C'))'
13. State the fundamental axioms of Boolean algebra.
14. Explain principle of Duality of Boolean algebra.
15. What do you mean by literal?
16. List the theorems of Boolean Algebra.
17. What is the difference between operator in mathematics and logic?
18. Write Dual of
   i) A + 1 = 1
   ii) x + x'y = x + y
   iii) x + x' = 1
   vi) x'(y + z) + x'(y' + z')

19. How is Boolean Algebra different from Arithmetic algebra?

20. Prove following

    i) C + A (C + B) + BC = C + AB

    ii) (A + C)A + AC + C = A + C

    iii) BC + A(B+C) = AB + BC + AC

    iv) B + A (B + C) + BC = B + AC

    v) x+ (x. y) = x

    vi) (a' + b') . (a' + b) . (a + b') = a' . b'

21. For the given truth table, write Boolean expression for function G1, G2, G3, G4, G5

X	Y	Z	G1	G2	G3	G4	G5
0	0	0	0	0	0	0	1
0	0	1	0	1	0	1	0
0	1	0	1	0	0	1	0
0	1	1	1	1	0	1	0
1	0	0	1	0	0	1	1
1	0	1	0	0	1	0	1
1	1	0	0	1	1	0	0
1	1	1	0	0	1	0	1

22. Express the OR operator in the terms of AND and NOT operator.

23. List the reasons used for algebraic proof of Associative theorm.

Chapter-2: Boolean Functions & Reduce Forms

Learning Objective

At the end of the chapter students will:

- Understand how logic relates to computing problems
- Handle simple Boolean function in SOP and POS form
- Apply rules to simplify /expand Boolean terms / functions
- Simplify the Boolean function using K map and algebraic method
- Establish the correspondence between Karnaugh maps, truth table and logical expressions.

Till now, we were using the available boolean function, now let's learn how to make a boolean function. In a boolean function, a binary variable can appear in any form, normal (a) or complemented (a'). Now consider a boolean expression formed on two variables a & b using AND operator. Since each variable may appear in any form, there are four possible ways of combining these two variables viz. a'b', ab', a'b, ab.

Similarly if we combine them using OR operator, again there are four possible ways viz. a+b, a'+b, a+b', a'+b'

For now we will concentrate on the terms formed using AND operator only. Each of them is called a minterm or Standard Product Term. A minterm, is obtained from AND term, with each variable being complemented, if the corresponding bit of binary number (in the truth table) is 0 and normal (i.e. non complemented) if the bit is 1. These terms are designated using m.

Following table provide minterm for 3 variables:

a	b	c	Minterm	Designation
0	0	0	**a'b'c'**	m_0
0	0	1	**a'b'c**	m_1
0	1	0	**a'bc'**	m_2
0	1	1	**a'bc**	m_3
1	0	0	**ab'c'**	m_4
1	0	1	**ab'c**	m_5
1	1	0	**abc'**	m_6
1	1	1	**abc**	

For four variables, there will be 16 minterms designated by m_0 to m_{15}.

Similarly, OR terms are called maxterms or Standard Sum Term. A maxterm is obtained from OR operator, variable being complemented if the corresponding bit of binary number (in the truth table) is 1 and normal if the bit is 0. These terms are designated by M

We already have maxterms for 2 variables, maxterms for 3 variables will be:

a	b	c	maxterm	designation
0	0	0	**a+b+c**	M_0
0	0	1	**a+b+c'**	M_1
0	1	0	**a+b'+c**	M_2
0	1	1	**a+b'+c'**	M_3
1	0	0	**a'+b+c**	M_4
1	0	1	**a'+b+c'**	M_5
1	1	0	**a'+b'+c**	M_6
1	1	1	**a'+b'+c**	M_7

Maxterm for four variables can be obtained in similar manner and they are designated as M_0 to M_{15} . If we put both maxterms and minterms together, we will have following table

a	b	c	minterm	maxterm
0	0	0	**a'b'c'**	**a+b+c**
0	0	1	**a'b'c**	**a+b+c'**
0	1	0	**a'bc'**	**a+b'+c**
0	1	1	**a'bc**	**a+b'+c'**
1	0	0	**ab'c'**	**a'+b+c**
1	0	1	**ab'c**	**a'+b+c'**
1	1	0	**abc'**	**a'+b'+c**
1	1	1	**abc**	**a'+b'+c'**

By now you might have noted that each maxterm is complement of its corresponding minterm.

For n variables, there will be 2^n , minterms (i.e. Standard Product Term) denoted as m , $0 < i < 2^n - 1$, and 2^n maxterms (i.e. Standard Sum Term), their complement, denoted as M_i , $0 < i < 2^n - 1$.

Now let's use this information for algebraic representation of boolean function from the truth table. This is done by forwarding a minterm for each combination of variables which produces a 1 in the function, and then by ORing these minterms.

For eg.

a	b	c	F1	minterm
0	0	0	**0**	**a'b'c'**
0	0	1	**1**	**a'b'c**
0	1	0	**0**	**a'bc'**
0	1	1	**0**	**a'bc**
1	0	0	**1**	**ab'c'**
1	0	1	**0**	**ab' c**
1	1	0	**1**	**abc'**
1	1	1	**1**	**abc**

Since **a'b'c, ab'c', abc'** and **abc** minterms are resulting into 1, in function (F_1) column, so algebraic representation of the function F_1 will be

F_1 (a,b,c)= a'b'c + ab'c' + abc' + abc

This is a boolean expression, expressed as sum of minterms and , will give value 1.

This form of expression is known as **Sum of Products or SOP** expression. F_1 in SOP form can also be represented as

i) $m_1 + m_4 + m_6 + m_7$

Instead of using Product terms, we can just mention the minterm designation.

ii) $\Sigma (1, 4, 6, 7)$

This one is mathematical representation of the above expression.

Let's take some more examples:

x	y	z	F_2	F_3	F_4
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	1
1	0	0	1	1	0
1	0	1	0	1	0
1	1	0	1	1	1
1	1	1	0	0	1

Algebraic representation in SOP form, of the three functions, represented in the truth table is

F_2 (x,y,z) = x'yz + xy'z' + xyz'

OR

$m_3 + m_4 + m_6$

$F_3(x,y,z) = x'y'z + x'yz + xy'z' + xy'z + xyz'$

OR

$m_1 + m_3 + m_4 + m_5 + m_6$

$F_4(x,y,z) = x'yz' + x'yz + xyz' + xyz$

OR

m2 + m3 + m6 + m7

All of them will result into value 1.

What if we have to represent the function in complement form i.e. for its value 0. Simple, we will OR the minterms, from the truth table, for which function value is 0. So we have,

$F_1' = x'y'z' + x'yz' + x'yz + xy'z$ (the remaining minterms)

On evaluating it for complement, we get

$F_1 = (x+y+z).(x+y'+z).(x+y'+z').(x'+y+z')$

What we get is the **Products of Sum form (POS) of the function F_1.**

POS form of the expression gives value 0, and we work on the complements. POS is formed by **ANDing maxterms.**

POS also can be represented in many ways. One way we have seen, others are

i) $F_1 = M_0.M_2.M_3.M_5$

Instead of using maxterms, we can use designation of it.

ii) $F_1 = \Pi(0, 2, 3, 5)$

This one is mathematical representation of the above expression.

Let's look both, SOP and POS expression for same function

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

SOP expression for F will be

$x'y'z + x'yz'$ or $\Sigma(1,2)$

POS equivalent of it will be found using complement of minterms with value zero, we know this

$(F') = (x'y'z' + x'yz + xy'z' + xy'z + xyz' + xyz)'$

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

OR

$= \Pi\ (0,3,4,5,6,7)$

Conversion from one form to another is easy. Short cut for it is:

i) Interchange the symbol (OR to AND / Σ to Π)

ii) List those terms, which are missing in original.

Expressions which were written till now, are in Canonical form. In this form,every term of the function should have all the literals. When, minterms or maxterms from truth table, are used for representation, the function is in canonical form only, as terms contains all variables (may be in any form). Since we can use either minterms or maxterms to form expression, we can have **Canonical SOP** expression or **Canonical POS** expressions.

Till now, we were given truth table and we were representing Boolean Function in algebraic form. Vice versa of it, i.e. for a given Boolean function in algebraic representation, we can represent it in Truth table. This was taken up in details in previous chapter.

What we have worked on was Canonical Boolean functions. There is another form of representation, also - both for SOP & POS. It is known as **reduced/simplified form.**

Simplified expressions are usually more efficient and effective when implemented in practice. Also simpler expressions are easier to understand, and less prone to errors. So the Canonical expressions are simplified/minimized to obtain a reduced expression (Standard form). This expression is then used for implementation of circuits in computers, *we will learn it in next chapter*.

Boolean function may be reduced in many ways, but we will be taking up the following two ways of reduction in this course.

1. Algebraic Manipulations
2. Karnaugh Map.

Algebraic reduction of Boolean function/expression: in this reduction, we apply certain algebraic axioms and theorems to reduce the number of terms or number of literals in the expression. In this way of reduction, there are no fixed rules that can be used to minimize the given expression. It is left to individual's ability to apply axioms or theorems for minimizing the expression.

Example 1:

F (a, b)	= (a+b)' + ab'	
	= a'b'+ab'	by De Morgan's theorem
	= (a'+a)b'	by Distribution
	= 1.b'	by Complement
	= b'	by Identity

Example 2:

F (a, b)	= ab+ ab'+ a'b	
	= a(b+b') + a'b	by Distribution
	= a.1 + a'b	by complement
	= a + a'b	by Identity
	= a+b	by absorption

Algebraic transformation, can always be used to produce optimal reduced form, but does not guarantee the reduced form. Also, it is not necessary to reach to same transformation, when worked on by different group of people. However, there are other methods using which, we will always reach to optimal and same solution every time.

Before moving ahead with other methods of reduction lets wait, to see how vice versa will be done, conversion of reduced from to Canonical form? Yes you guessed right, using theorems and axioms. Lets do it for 2nd example

F(a,b) = a + b	Reduced expression
= a . 1 + b . 1	by Identity
= a.(b + b') + b . 1	by Complement
= a . b + a . b' + b . 1	by Distribution
= a . b + a . b' + 1 . b	by Commuting
= a . b + a . b' + (a + a') . b	by Complement
= a . b + a . b' + a . b + a' . b	by Distribution
= a . b + a . b' + a' . b	by Indempotancy

Now let's learn other way of reduction.

Karnaugh Map (K-map):

A K-map is nothing more than a special form of truth table representation useful for reducing logic functions into minimal Boolean expressions. Karnaugh map was developed by Maurice Karnaugh, at Bell Labs in 1953. The map reduces boolean expression more quickly and easily, as compared to algebraic reduction.

Using K-map for reduction is 4 step process

1. Drawing K-map
2. Filling it, i.e. representing the boolean expression in map
3. Grouping the terms for reducing purpose
4. Reading the reduced expression from map

1. Drawing K-map

A K-map is a diagram, made up of squares, where each square represents one minterm. So in a two variable map there will be 4 squares and for three variables, map will contain 8 squares. K-map diagram for two variables -a,b, will look like:

a	b	minterm
0	0	a'b'
0	1	a'b
1	0	ab'
1	1	ab

a \ b	b'	b
a'	a'b' m_0	a'b m_1
a	ab' m_2	ab m_3

For three variables - a, b,c, we can draw it as:

a \ bc	b'c'	b'c	bc	bc'
a'	m_0	m_1	m_3	m_2
a	m_4	m_5	m_7	m_6

OR

ab \ c	c'	c
a'b'	m_0	m_1
a'b	m_2	m_3
ab	m_6	m_7
ab'	m_4	m_5

For four variables - a, b, c, d

ab \ cd	c'd'	c'd	cd	cd'
a'b'	m_0	m_1	m_3	m_2
a'b	m_4	m_5	m_7	m_6
ab	m_{12}	m_{13}	m_{15}	m_{14}
ab'	m_8	m_9	m_{11}	m_{10}

OR

cd \ ab	a'b'	a'b	ab	ab'
c'd'	m_0	m_4	m_{12}	m_8
c'd	m_1	m_5	m_{13}	m_9
cd	m_3	m_7	m_{15}	m_{11}
cd'	m_2	m_6	m_{14}	m_{10}

The cells of the map are marked by the minterms as explained in first map.

For eg.

If you look at the square assigned m_3 (minterm), it corresponds to binary no. 0100 of the truth table.

Now to understand, why the squares have been marked, as they are?

If you look at any two adjacent cell in the map, they differ only by one variable, which is normal in one cell and complemented in other or vice-versa. For eg. From m_3 to m_7 only a changes its state from complemented to normal. And we know, using Boolean axiom (a'+a=1) that we can eliminate the variable from the resultant reduced term.

2. Filling the K-map

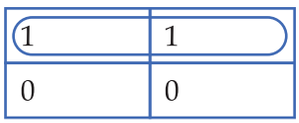
Once a K-map is drawn, next is to fill the map i.e. mark the function in the map. For representation, the function should be in canonical SOP form. If we have the function represented in truth table, then it will be in canonical form only. Otherwise, we know how to convert it. We will mark 1 in all those cells of K-map, for which there is a minterm in the Boolean function/expression.

3. Grouping the minterms

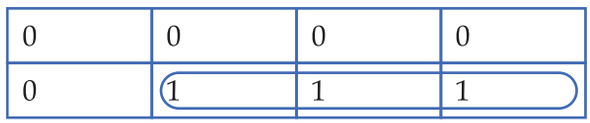
Next is grouping the 1s in the map. We group 1s of adjacent cells.

Following are the rules for grouping. Remember our goal should be to maximize the size of group (i.e. no. of 1s included in the group) and to minimize the number of groups.

- A group must contain 1, 2, 4, 8, 16, …. cells, i.e. in the power of 2.

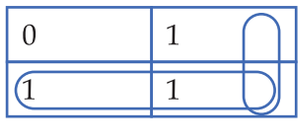


Correct

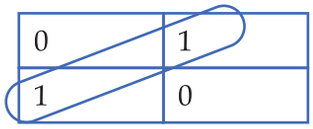


Incorrect

- Cells which are grouped, should be adjacent cells, i.e. horizontal or vertical, not diagonal.

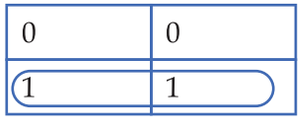


Correct

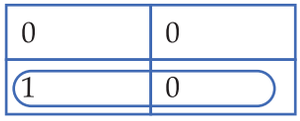


Incorrect

- Groups should not include any cell containing zero



Correct



Incorrect

- Each group should be as large as possible

1	1	1	1
0	0	1	1

Correct

1	1	1	1
		1	1

Incorrect

- Groups may overlap

1	1	1	1
0	0	1	1

Correct

1	1	1	1
		1	1

Incorrect

- Groups may wrap around the table. The left most cell(s) in a row may be grouped with rightmost cell(s), and the top cell(s) in a column may be grouped with the bottom cells(s). Cells in corners of the map are also adjacent, for that purpose.

0	1	1	0
0	0	0	0
0	0	0	0
0	1	1	0

Correct

0	0	0	1
0	0	0	0
0	0	0	0
1	0	0	0

Incorrect

- There should be as few groups as possible, as long as this does not contradict any of the previous rule. Once the grouping is done, last step is

4. Reading the reduced expression from K-map

- Each group corresponds to one product term, from K-map
- The term is determined by finding the common literals in that group, i.e. the variables which change their state (from normal to complement or vice-versa) are to be omitted in resultant product term.

  What we have to do is, find the variable(s) listed on top and / or side, which are the same for entire group and ignore variable(s) which are not same in the group.

  Using the above strategy, write the result. Lets look at some examples

  Simplify the Boolean Function F(x,y,z) = x'yz + xy'z' + xyz + xyz'

  Step 1: Drawing the K-map

	yz	*y'z'*	*y'z*	*yz*	*yz'*
x					
x'					
x					

Step 2: Marking of function in map

yz / x	y'z'	y'z	yz	yz'
x'	0	0	1	0
x	1	0	1	1

Step 3: Grouping

yz / x	y'z'	y'z	yz	yz'
x'	0	0	1 (1)	0
x	1	0	1	1 (2)

Step 4: Reading reduced expression from map

For Group 1, variable x is changing its state and for Group 2, variable y is changing its state. So x in 1st group and y in 2nd group will be omitted. The resultant expression will contain yz, because of 1st group and xz' for 2nd. As this is SOP expression so both the terms will be joined through OR operator. Resultant reduced function will be

F = yz + xz'

Example:

F (w,x,y,z) = Σ (5,7,13,15)

wx / yz	y'z'	y'z	yz	yz'
w'x'	0 (0)	0 (1)	0 (3)	0 (2)
w'x	0 (4)	1 (5)	1 (7)	0 (6)
wx	0 (12)	1 (13)	1 (15)	0 (14)
wx'	0 (8)	0 (9)	0 (11)	0 (10)

F (w,x,y,z) = xz

Now let's take an example where the function is represented using Truth table, instead of algebraic representation

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Using minterm designation we will represent the function in K map.

yz / x	y'z'	y'z	yz	yz'
x'	0	1	0	0
x	0	1	1	1

f (x,y,z) = y'z + xy

So far we were working with SOP expression/function. However, it is possible to reduce POS function also. Although we know k-map naturally reduces the expression is SOP form. In POS reduction, the approach is much the same except for, instead of minterms, we work with maxterms. We know a maxterm results into zero.

Now if we look at drawing of k-map it will be following for three variable

BC / A	B+C	B+C'	B'+C'	B'+C
A	M_0	M_1	M_3	M_2
A'	M_4	M_5	M_7	M_6

OR

C / AB	C	C'
A+B	M_0	M_1
A+B'	M_2	M_3
A'+B'	M_6	M_7
A'+B	M_4	M_5

Similarly a four variable map may be created.

Next is representing the Boolean expression/function in K- map. Representation remains same except, now we represent 0 instead of 1.

For reduction, 0's from the k-map will be grouped in same fashion as 1's were grouped, and reduced expression will be represented using maxterms.

Example:

f(w, x, y, z) = II (10, 11, 14, 15)

Using maxterm numbers, we will represent the function, i.e. 0 will be represented in map. For reduced POS expression 0's will be grouped

yz / *wx*	*y'z'*	*y'z*	*yz*	*yz'*
w'x'	1	1	1	1
w'x	1	1	1	1
wx	1	1	0	0
wx'	1	1	0	0

f(w, x, y, z) = w'+y'

f (a, b, c) = a'b'c + abc' +abc is an Sop expression, whose POS reduced form is denied.

bc / *a*	*b+c*	*b+c'*	*b'+c'*	*b'+c*
a	0	0	0	1
a'	0	0	1	1

As we are working for POS, 0's will be grouped and every group will result into sumterm.

f(a,b,c) = (a+c').b

LET'S REVISE

- A Boolean function can be expressed algebraically from a given truth table by forming a minterm and then taking the OR of all those terms.
- **Minterm:** An n variable minterm is a product term with n literals resulting into 1.
- **Maxterm:** An n variable maxterm is a sum term with n literals resulting into 0.
- A sum-of-product expression is logical OR of two or more AND terms
- A product-of-sum is logical AND of two or more OR terms
- If each term in SOP / POS form contains all the literals, then it is canonical form of expression
- To convert from one canonical form to another, interchange the symbol and list those numbers missing from the original form.
- The Karnaugh map (K-map) provides a systematic way of simplifying Boolean algebra expressions.
- For minimizing a given expression in SOP form, after filling the k map look for combination of one's. Combine theses one's in such a way that the expression is minimum.
- For minimizing expression in POS form we mark zeros, from the truth table, in the map. Combine zeros in such a way that the expression is minimum.
- **Sum Term:** is a single literal or the logical sum of two or more literals.
- **Product term:** is a single literal or the logical product of two or more literals.

EXERCISE

1. Determine the values of A, B, C and D that make the product term A'BC'D equal to 1

   i) A = 0, B = 1, C = 0, D = 1

   ii) A = 0, B = 0, C = 0, D = 1

   iii) A = 1, B = 1, C = 1, D = 1

   iv) A = 0, B = 0, C = 1, D = 0

2. The binary value of 1010 is converted to the product term A'B'C'D

   i) True

   ii) False

3. Which of the following expression is in SOP form?

   i) (A+B) (C+D)

   ii) AB (CD)

   iii) (A) B (CD)

   iv) AB+CD

4. A truth table for SOP expression ABC'+ AB'C + A'B'C has how many input combinations?

   i) 1

   ii) 2

   iii) 4

   iv) 8

5. POS equivalent of ABC+ AB'C'+AB'C+ABC'+A'B'C will be

   i) (A'+B'C') (A'+B+C') (A'+B+C)

   ii) (A'+B'+C') (A+B'+C) (A+B'+C)

   iii) (A+B+C) (A+B'+C) (A+B'+C')

   iv) (A+B+C) (A'+B+C') (A+B'+C)

6. Converting the Boolean expression LM+M (NO+PQ) to SOP form we get _____

   i) LM +MNOPQ

   ii) L+MNQ+MPQ

iii) LM+ M+NO+MPQ

iv) LM+MNO+MPQ

7. State whether AC+ABC = AC is

i) True or

ii) False

8. A student makes a mistake somewhere in the process of simplifying the following Boolean expression:

i) ab + a (b+c)

= ab+ab+c

= ab+c

Determine, where the mistake was made, and what proper sequence of steps should be used to simplify the original expression.

ii) (x'y+z)'

= (x'y)'.z'

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

= x+y'.z'

Determine what the mistake is?

9. When grouping cells within k-map, the cells must be combined in groups of _____

i) 2s

ii) 1, 2, 4, 8, etc

iii) 4s

iv) 3s

10. Mapping the SOP expression A'B'C' + A'BC' + A'BC +ABC' we get _____

i)

	C'	C
AB		
A'B'	1	1
A'B	1	
AB'		1

ii)

	A	B
AB	1	
A'B'	1	
A'B		1
AB'	1	1

iii)

	C'	C
AB		1
A'B'		
A'B	1	1
AB'	1	1

iv)

	C'	C
AB	1	1
A'B'		
A'B	1	
AB'		1

v)

	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	0	1	1	0
10	0	0	0	0

vi)

	C'D'	C'D	CD	CD'
A'B'	0	0	0	0
A'B	0	0	0	0
AB	1	1	1	1
AB'	0	0	0	0

11. If you look at the following k-map, you should notice that only two of the input variables- A, B, C, D change their state, in the marked group. The other two variables hold the same value '1'. Identify which variable change, and which stay the same:

	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	0	1	1	0
10	0	0	0	0

	C'D	CD	CD'
A'B'	0	0	0	0
A'B	0	1	1	0
A B	0	1	1	0
AB'	0	0	0	0

12. Give truth table for

    i) Z = x'+y'+z

    ii) Z = x(y+xz+x')

13. Use Boolean algebra to find the most simplified SOP expression for F=ABD+CD+ ACD+ABC+ABCD

    i) F = ABD+ABC+CD

    ii) F= CD+AD

    iii) F = BC+AB

    iv) F= AC+AD

14. From the truth table below, determine SOP and POS expression

A	B	C	Output X
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

15. Specify which axiom(s)/theorem(s) are being used in the following Boolean reductions:
    i) x'y' + x'y'z = x'y'
    ii) 1+A = 1
    iii) D+CD =D
    iv) a'.a'=a'
    v) (bc)' + bc = 1
    vi) xyz+zx = xz
    viii) ca'b' + ab = ab + c
16. Construct a truth table for the following functions and from the truth table obtain an expression for the inverse function
    i) F1(A,B,C) = A + BC'
    ii) F2(A,B,C) = AC + BC + AB'
17. Examine the given truth table and then write both SOP & POS boolean expressions describing the output.

A	B	C	Output
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

18. Write POS Boolean expressions for F (a, b) = ab'+a'b. Show through Boolean algebra reduction that the SOP & POS expressions are indeed equivalent to one another.
19. Minimize the following Boolean functions using algebraic method
    Z = f(a, b, c) = a'b'c' +a'b + abc' +ac
    Z = f (a,b,c) = a'b +bc' +bc+ ab'c'
    Z = f (a,b,c) = a'b'c' +
20. Reduce the following Boolean expression using k-map
    i) F (a, b, c) = a'b'c'+ a'bc'+ a 'bc' + a'bc + ab'c' + abc' to SOP form
    ii) F(w,x,y,z) = (w + x) (x + z') (w' + y' + z) to SOP form

iii)

A	B	C	F1
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

To SOP & POS form

ii)

A	B	C	D	F2
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

To SOP & POS Form

22. Obtain the minterm canonical form of the Boolean expression by algebraic method

i) xyz + xy + x'(yz' + y'z)

ii) ab + abc + a'b + ab'c

Chapter-3: Application of Boolean Logic

Learning Objective:

Boolean Expression through Logic gates

- At the end of the chapter the students will:
- Learn about gates that process logic signals.
- Understand basic terminology, type of logic gates
- Learn about logic circuits and Boolean expression
- Learn how to design small circuits
- Learn to determine output of the gate(s) / circuit for the given input values.
- Learn about universality of NAND & NOR gate
- Explore the application of Boolean algebra in the design of electronic circuits.

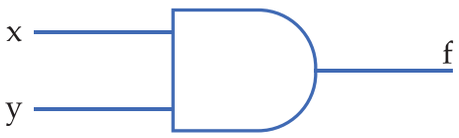
Boolean Expression through Logic gates

We know that a Boolean function is an algebraic expression formed with Boolean variables, operator OR, AND and NOT, parenthesis and equal to sign. Any Boolean function can be transformed in a straight forward manner from an algebraic expression into logic circuit diagram. This logic circuit diagram can be composed of basic gates AND, OR and NOT or advance gates, NAND and NOR, etc.

Logic circuits that perform logical operations such as AND, OR and NOT are called gates. A gate is block of hardware that produces a logic 0 or a logic 1 output, in response to the binary signal(s) applied to it as input.

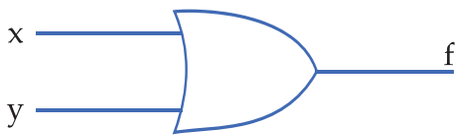
Let's look at the symbols used to represent digital logic gates

AND $f(x, y) = x.y$



For logical multiplication operation

OR $f(x, y) = x+y$

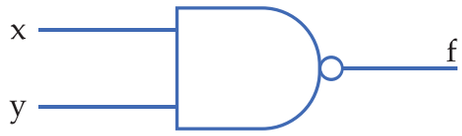


For logical addition operation

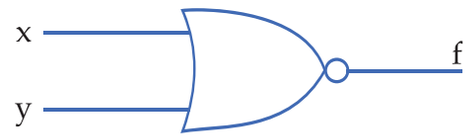
NOT $f(x) = x'$

For Complementation

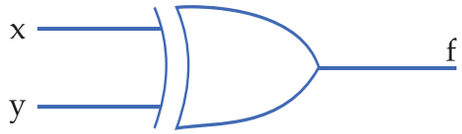
NAND $f(x,y) = (x.y)'$



NOR $f(x,y) = (x+y)'$



XOR $f(x,y) = x \oplus y = x'y + xy'$

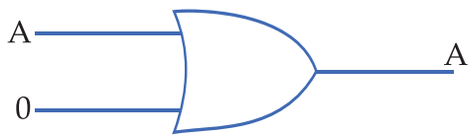


All the gates, except NOT, have two inputs represented on LHS and one output - on the RHS. Gates can have more than two inputs also. As the logic circuit is a way of representing Boolean expression, let's represent, axioms and theorems in this form.

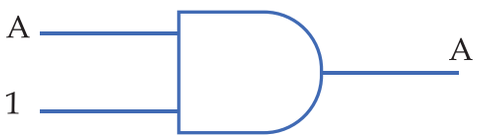
When a Boolean Function is implemented with logic gates, each literal in the function becomes an input to a gate. That's why we minimize the function - to reduce the number of inputs to gate also to reduce the total number of gates in the circuit. Also, let's assume that literal in both the forms- normal and complemented is available. This will help us in drawing neat circuit diagrams.

1. Identity

i) $A + 0 = A$



ii) $A . 1 = A$



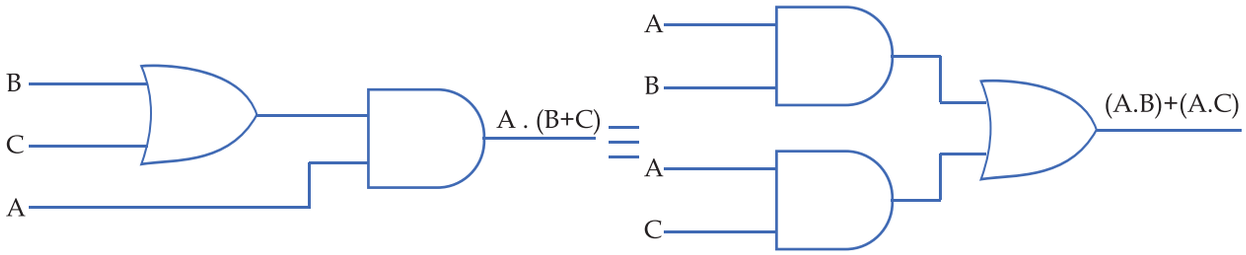
2. Commutative

i) A+B = B+A A

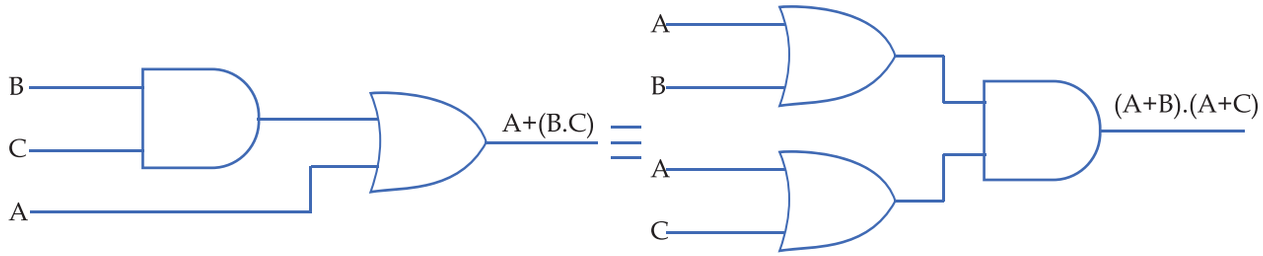
ii) A . B = B . A A

3. Distributive

i) A.(B+C) = AB+AC

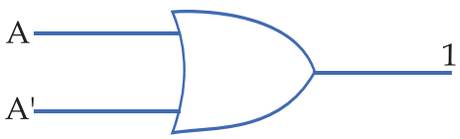


ii) A+(B.C) = (A+B).(A+C)

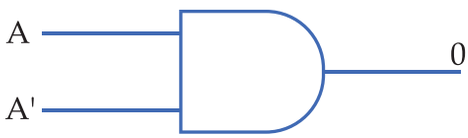


4. Complement

i) A+A' = 1

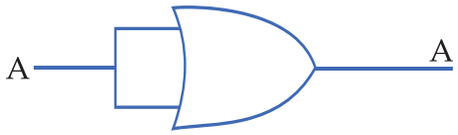


ii) A.A' = 0

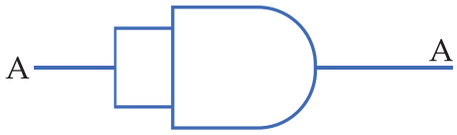


5. Indempotency

i) A+A = A

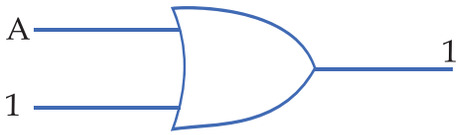


ii) A.A = A

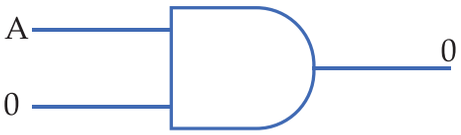


6. Null Element

i) A+1 = 1



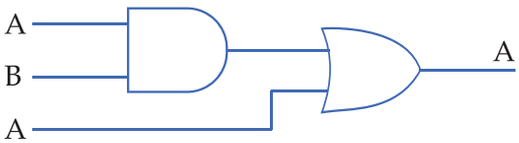
ii) A.0 = 0



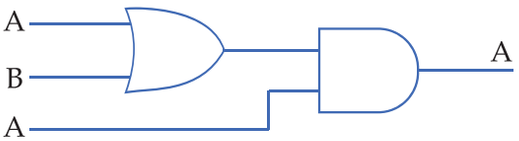
7. Involution (A')' = A

8. Absorption

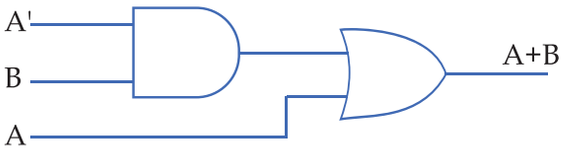
First law: i) A+ (AB) = A



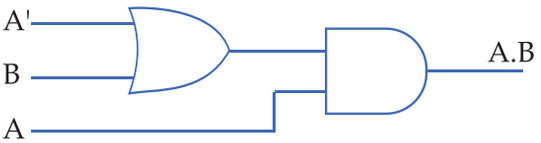
ii) A.(A+B) = A



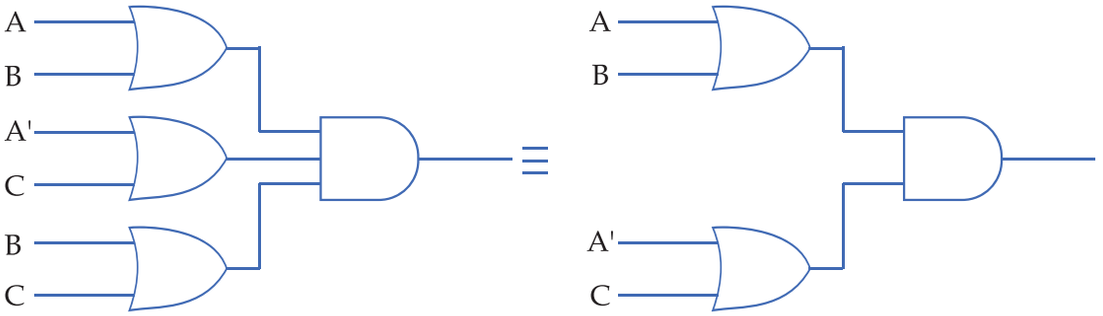
Second law: i) A+A'B = A+B



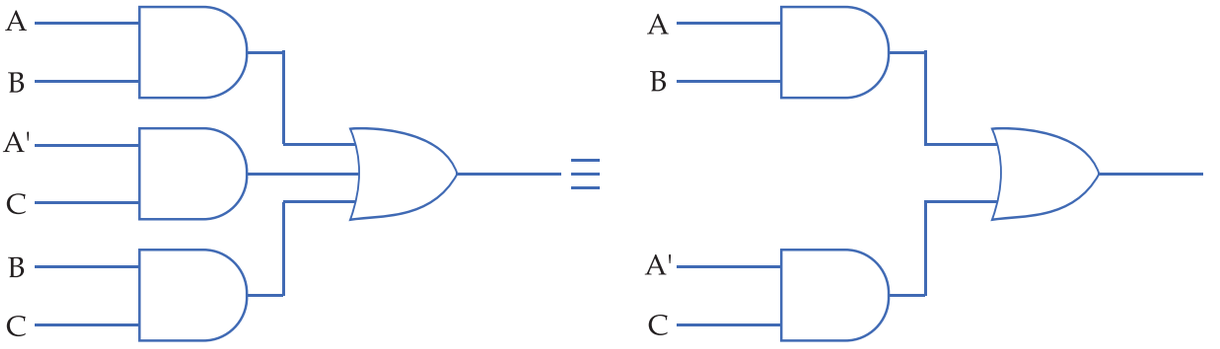
ii) A.(A'+B) = A.B



Third law: i) (A + B) (A' + C) (B + C) = (A + B) (A' + C)

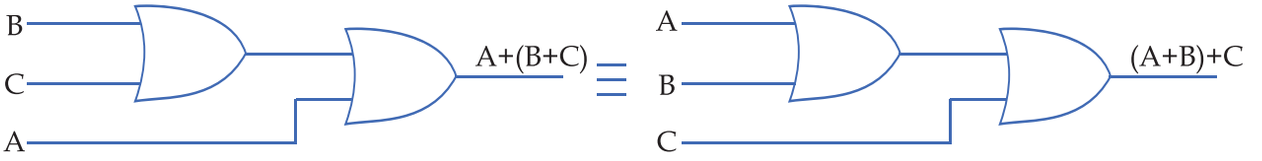


ii) $(A \cdot B) + (A' \cdot C) + (B \cdot C) = A \cdot B + A' \cdot C A$

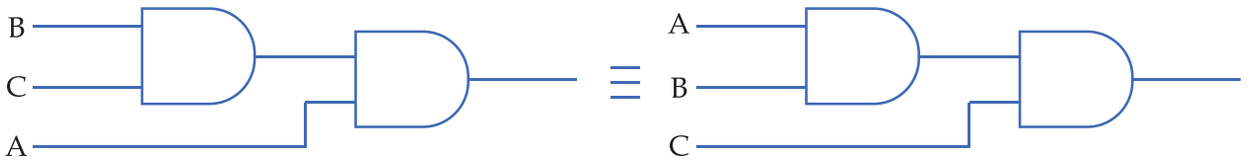


9. Associative

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



ii) $A\,(B.C) = (A.B)\,C$



10. DeMorgan's Theorem

i) $(A+B)' = A'.B'$

(NOR) **(Negative AND)**

ii) $(A.B)' = A'+B'$

(NAND) **(Negative OR)**

We have already verified these theorems using truth table and algebraic method in previous chapter.

Any Boolean function can be associated with a logic circuit, in which the inputs and outputs, represent the statement of Boolean algebra. Whatever is the complexity of function, they all can be constructed using three basic gates. For implementation of a Boolean function in logic circuit form:

The function will either be in SOP form or POS form. The SOP (Sum-of- Product) form is implemented in the following manner:

i) Each AND term is represented by AND gate (one gate for each AND term)

ii) Output of each AND gate is connected as input to single OR gate.

POS (Product-of-Sums) is implemented as

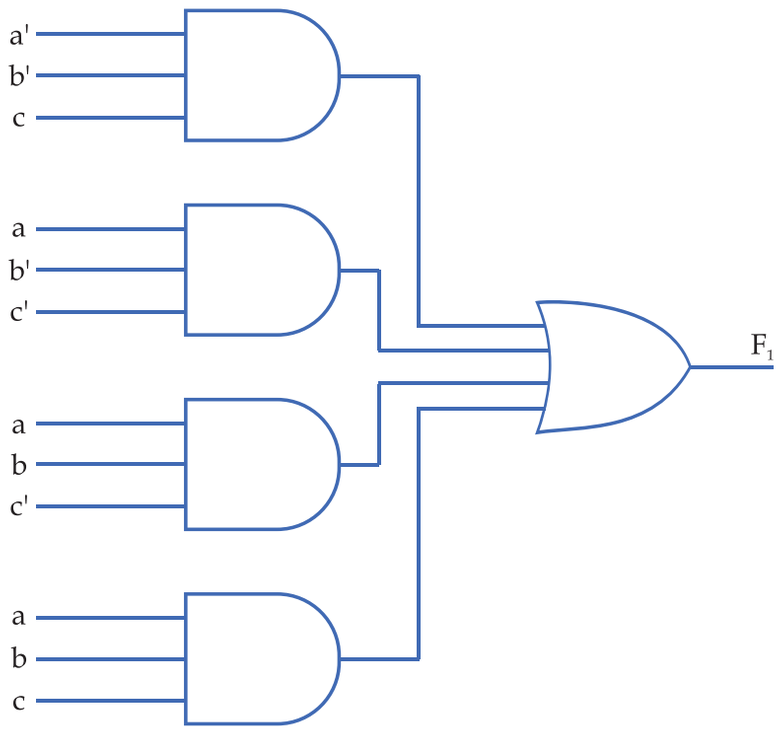
i) Each OR term is represented using an OR gate.

ii) Output of all the OR gate(s) are connected as input to single AND gate.

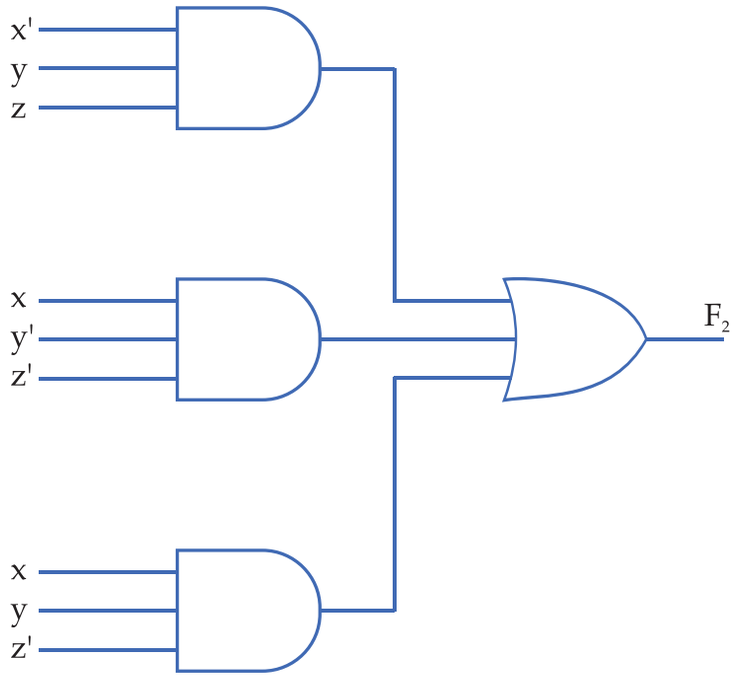
In both the representations, *it is assumed that both normal and complemented inputs, for all the variables, are available*. So inverters are not needed. Also in both the types of representation, two levels of gates are used. In SOP form AND gates are connected to OR gate and in POS form OR gates are connected to AND gates. Thus the implementation is known as two-level implementation.

Let's illustrate some Boolean functions from previous chapter in circuit form.

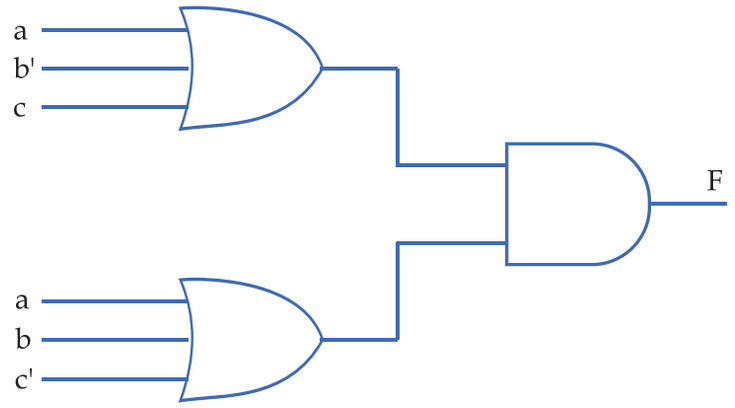
$F_1 = a'b'c + ab'c' + abc' + abc$



$F_2 = x'yz + xy'z' + xyz'$



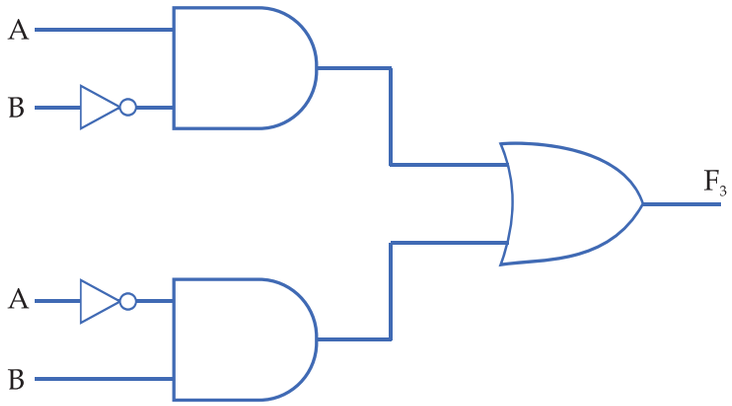
$F_3(a,b,c) = (a+b'+c) . (a+b+c')$



How to get a Boolean expression for a Logic Circuit?

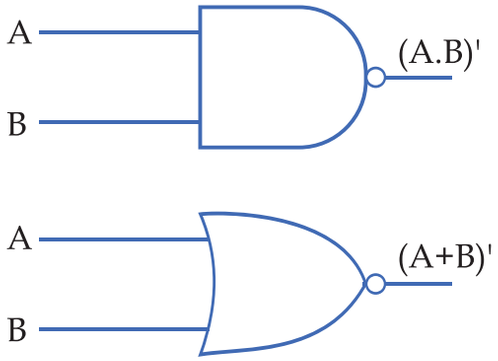
To derive the Boolean expression for a given logic circuit begin at the left most inputs and work towards the final output, by writing the expression for each gate.

For example:



- The expression for the left-most first gate will be B'
- This output along with A becomes input to AND gate so its output will be A.B'
- Similarly, working on 2nd set of gate from Left most side we will have A' and then A'.B
- Output of these two AND gates is input to OR gate, therefore the expression for the OR gate will be A.B'+ A'.B which is the final output expression for the entire circuit

The circuits made, till now have one type of gate(s) AND / OR at first level and another type of gate OR / AND at second level. As these are simple circuits, but in a complex circuit we might have a function which uses both type of gate(s) at first level. Building a really big complex digital system using many types of gates will make the job tedious. We need to have a gate, which can replace the function of all other gates, hence can be used to implement any digital circuit, such gates are also known as universal gates. NAND and NOR are such universal gates. NAND for 2 input is (A.B)' and NOR for 2 input is (A+B)'



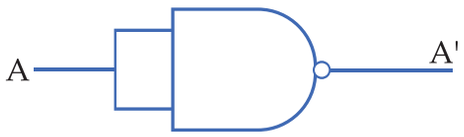
An SOP expression, represented by AND gate(s) at first level and OR gate at second level, can naturally be implemented by only NAND gate(s). Similarly a POS expression, which uses OR gates at first level and AND gate at second level can naturally be implemented through NOR gate(s) only.

Now let's see how each logic gate can be created with universal gate, so that entire circuit can be implemented through single type of gate.

Realization of all functions using NAND gate

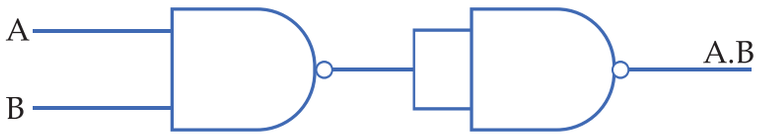
NOT A' = (A.A)'

i.e. A nand A



AND A.B = ((A.B)')'

i.e. (A nand B) nand (A nand B)



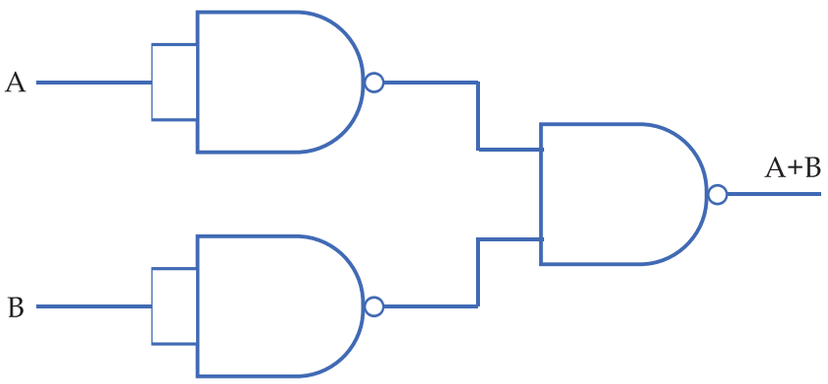
OR = A+B

= ((A+B)')'

= (A'.B')'

=((A.A)'. (B.B)')'

i.e. (A nand A) nand (B nand B)

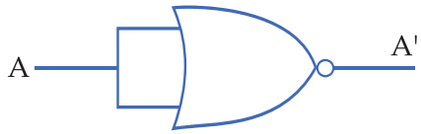


Realization of logic functions using NOR gate

NOT = (A)'

= (A+A)'

i.e. A NOR A



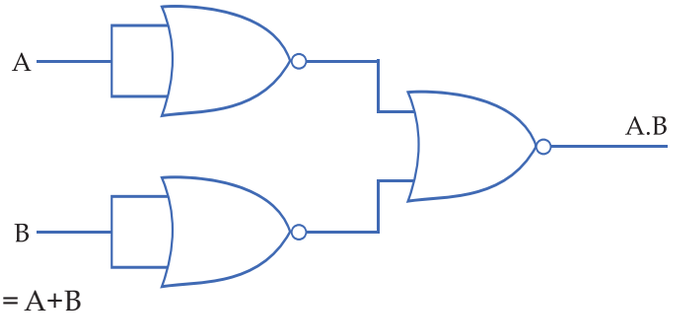
AND = A.B

= ((A.B)')'

= (A'+B')'

= ((A nor A) + (B nor B))'

i.e. (A nor A) nor (B nor B)

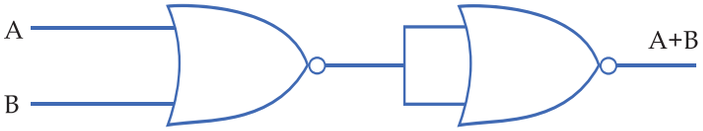


OR = A+B

= ((A+B)')'

= (A nor B)'

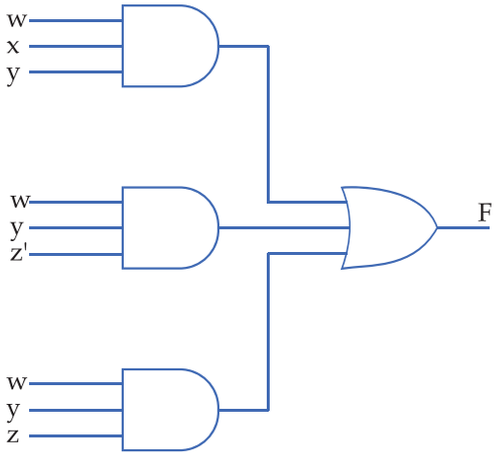
i.e. (A nor B) nor (A nor B)



Let's take some example of implementing complete circuit through universal gate

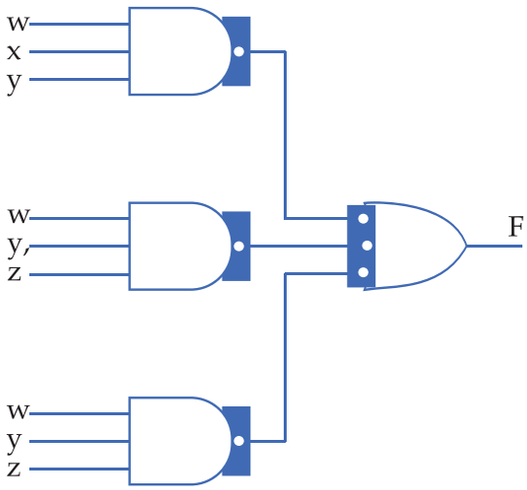
F (w,x,y,z) = wxy + wyz' + wyz

This is an SOP expression and will be represented by AND OR gate

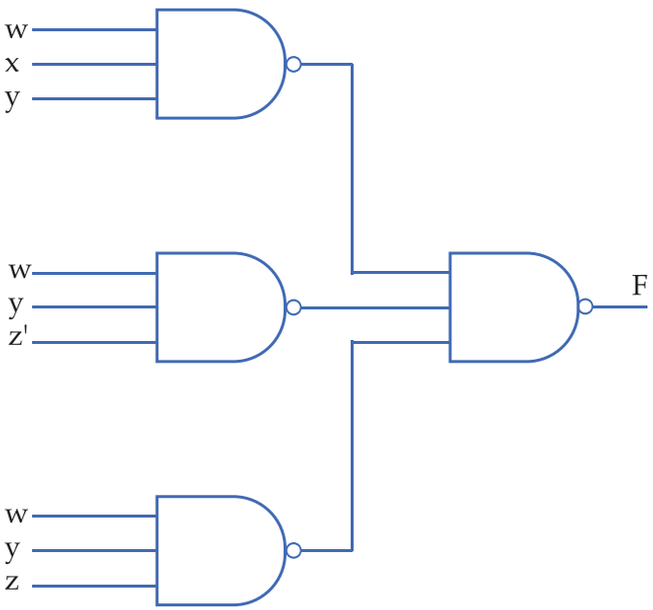


Let's apply De Morgan's theorem to implement the above circuit with minimum number of NAND gates, because if each gate is replaced by universal gate using above solution we will end up using a large number of gates and increase the size of circuit. You may try this for a Boolean function, which can then be compared by the solution given below.

If we introduce bubbles at AND gate(s) output and OR gate(s) input, the resultant will remain same as A = (A')'. Applying this, the above circuit will become



All the gates at first level are NAND gates. If you look at the second level gate it is (A'+B'+C') assuming A, B & C are output from 1st, 2nd, and 3rd gate respectively. (A'+B'+C') = (A.B.C)' using DeMorgan's Theorem. So now the complete circuit may be represented using NAND gate as:

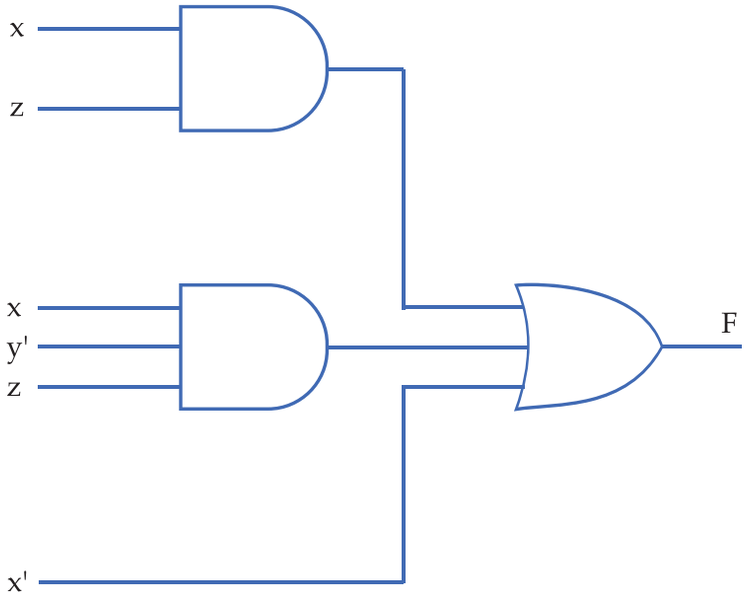


So the trick is replace AND gate at 1st level and OR gate at 2nd level by NAND gate.

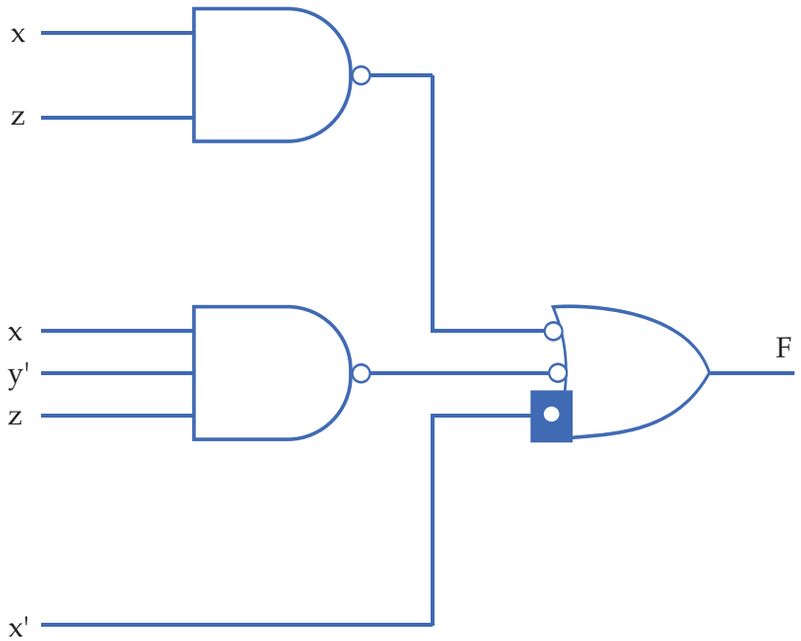
Let's take another kind of example

F (x,y,z) = xz + xy'z + x'

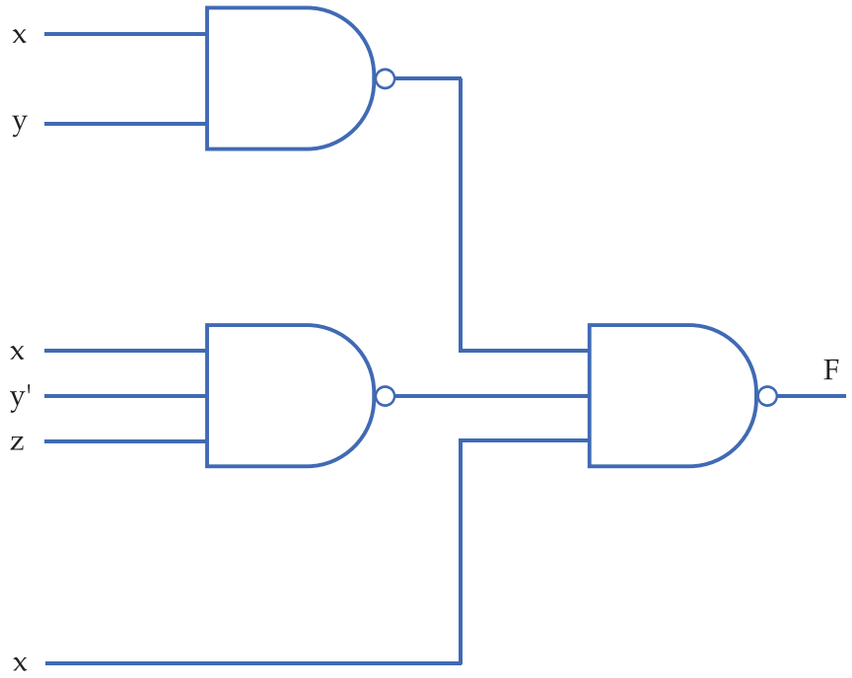
Its AND → OR diagram will be



If we replace all gates through NAND only, then the single input going to second level gate (OR) will get complemented, which will change the input being provided, hence function will change.



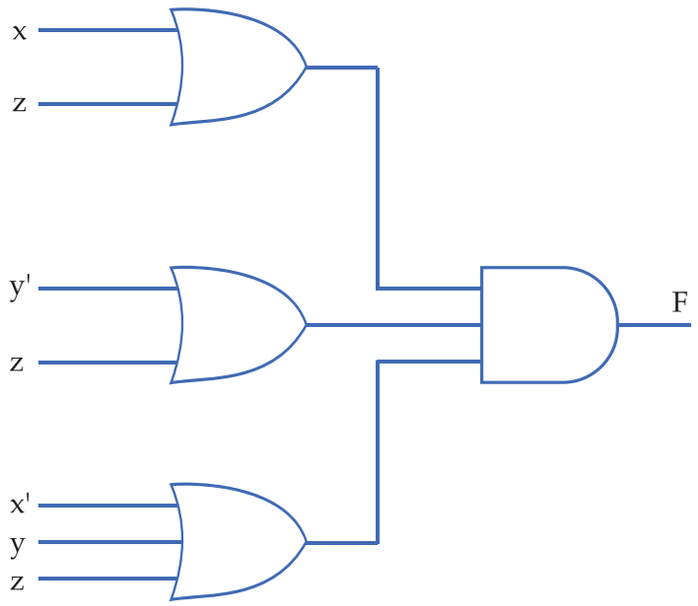
This can be handled by sending an inverted input to 2nd level gate whenever there is single input which passes directly to second level gate. So the correct solution will be



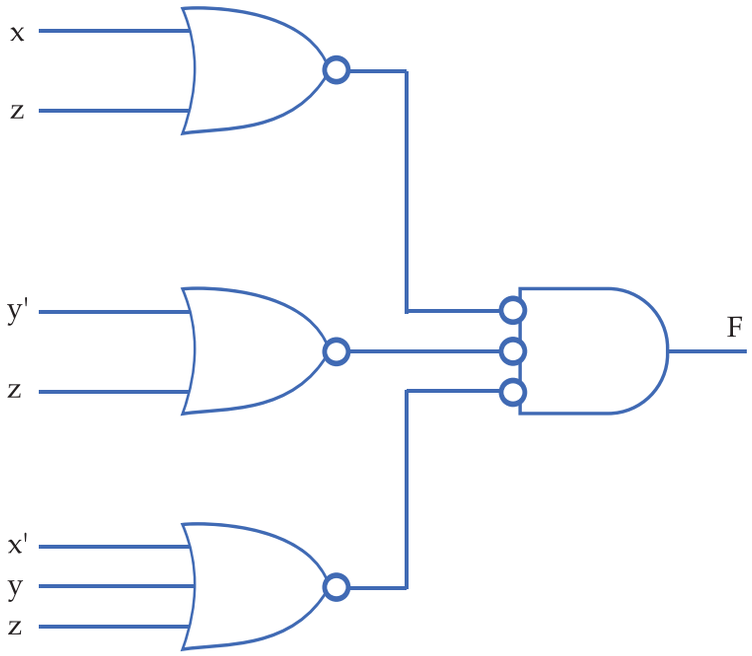
Now let's implement a complete circuit using NOR gate(s)

F (x,y,z) = (x+z) (y'+z) (x'+y+z)

Its natural representation is OR → AND



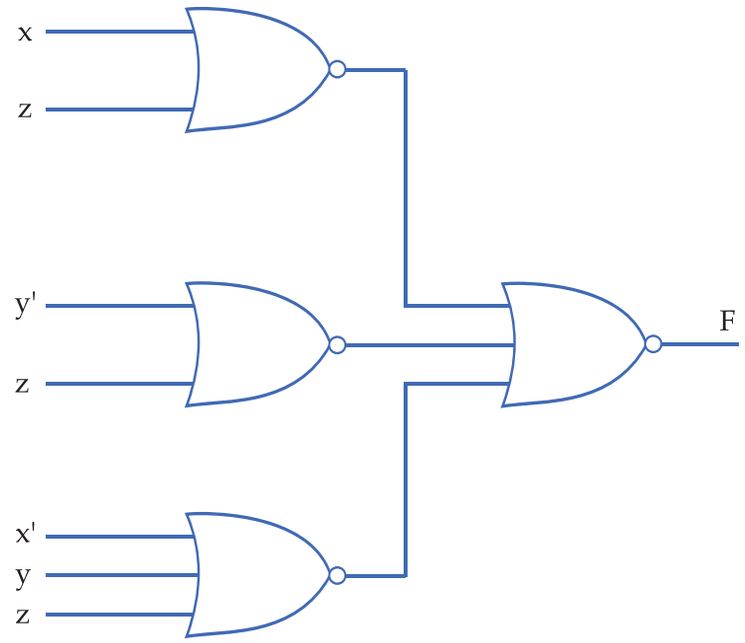
Again using the same concept, we have x z



bubbles to be highligted

Which is OK as the output of first level gate is complemented and it is again complemented before providing to next level gate.

The second level gate is equivalent to NOR gate as per DeMorgan's theorem. So the circuit implementation using NOR gate only will be



In this implementation also, if an input is directly moving to second level gate in normal OR → AND representation then, in NOR implementation the input will be complemented before moving to 2nd level gate.

Now we know SOP expression can be implemented using AND → OR circuit or universal NAND gate. POS expression can be implemented using OR → AND or universal NOR gate. Representing POS using only NAND gates will require conversion of POS to SOP form and then implement same using only NAND only gates. We have already learnt the way of converting expression from one form to another.

Let's take an example:

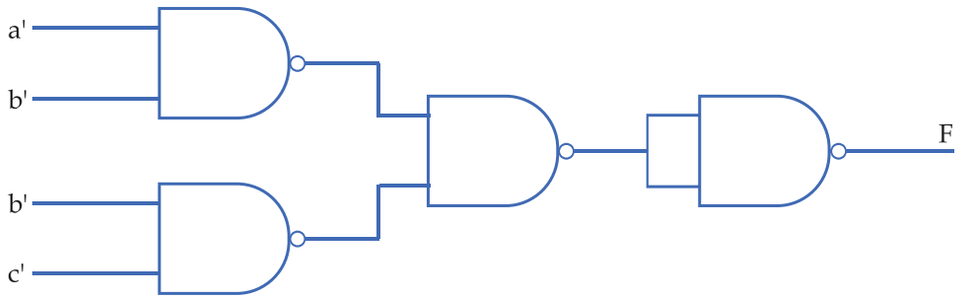
F (a,b,c) = (a+b).(b+c)

= [((a+b) . (b+c))']'

= [(a+b)' + (b+c)']'

= [(a'.b') + (b'.c')]'

So the circuit will be



For representation of SOP through only NOR gates, we can proceed in similar manner.

One of the most recent use of Boolaen logic is, Internet search engine and Database search. You have applied operator AND, OR and NOT in SQL queries and already know it's usage. Let's learn the usage of Boolean operators for searching Internet.

We know, Internet is a vast computer database and the proper use of Boolean operators - AND, OR and NOT would increase the effectiveness of web search. These operators can act as effective filters for finding just the information one need from Internet. So most of the search engine support some form of Boolean query using Boolean operators (check the help section of your favorite search engine supported by it). An engine is a search program, that allows us to search its data base.

These operators can be used in two ways:

i) You can use the words -AND, OR, NOT

ii) You can use symbols (math equivalents) - + for AND, -for NOT, OR is the default setting in many search engine.

Note: When you are using symbols, don't add space between the symbols and your query.

OR operator is used to broaden the search. It's interpreted as "at least one is required, more than one or all can be returned". So when search strings (keyword) are joined using OR, the resultant will include - any, some or all of the keywords used in the statement. The operator will ensure that you do not miss anything valuable.

AND operator is used to narrow the search. It is interpreted as "all is required ". So when search string i.e. keywords are joined using AND, the resultant will include the documents containing all the keywords.

NOT operator will again narrow down the search, this time by excluding the search string i.e. keyword.

LET'S REVISE

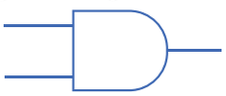
- Gate is an electronic system that performs a logical operation on a set of input signal(s). They are the building blocks of IC.
- An SOP expression when implemented as circuit - takes the output of one or more AND gates and OR's them together to create the final output.
- An POS expression when implemented as circuit - takes the output of one or more OR gates and AND's them together to create the final output.
- Universal gates are the ones which can be used for implementing any gate like AND, OR and NOT, or any combination of these basic gates; NAND and NOR gates are universal gates.
- Implementation of a SOP expression using NAND gates only
  1) All 1st level AND gates can be replaced by one NAND gate each.
  2) The output of all 1st level NAND gate is fed into another NAND gate.

  This will realize the SOP expression
  3) If there is any single literal in expression, feed its complement directly to 2nd level NAND gate.

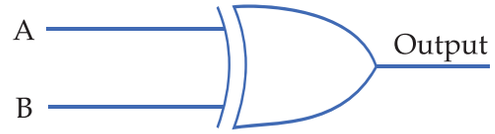
  Similarly POS using NOR gate can be implemented by replacing NAND by NOR gate.
- Implementation of POS / SOP expression using NAND / NOR gates only.
  1) All literals in the first level gate will be fed in their complemented form.
  2) Add an extra NAND / NOR gate after 2nd level gate to get the resultant output.

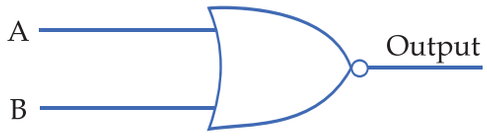
EXERCISE

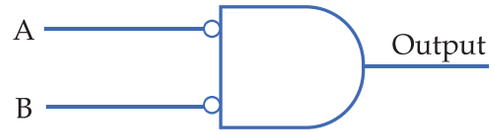
1. One of the DeMorgan's theorems states that (x=y)' = x'y'. Simply stated, this means that logically there is no difference between:
   i) A NOR and an AND gate with inverted inputs
   ii) A NAND and an OR gate with inverted inputs
   iii) An AND an a NOR gate with inverted inputs
   iv) A NOR and a NAND gate with inverted inputs
2. An AND gate with 'bubbles' on its input performs the same function as a(n) _____ gate
   i) NOT
   ii) OR
   iii) NOR
   iv) NAND
3. How many gates would be required to implement the following Boolean expression before simplification of X4 + X (X+Z) + Y (X+Z)
   i) 1
   ii) 2
   iii) 4
   iv) 5
4. The NAND and NOR gates are referred to as universal gates because either,
   i) Can be found in almost all digital circuit
   ii) Can be used to build all other types of gate.
   iii) Are used in all countries of the world
   iv) Were the first gates to be integrated
5. The boolean expression x=A'+B'+C' is logically equivalent to which single gate?
   i) NAND
   ii) NOR
   iii) AND
   iv) OR
6. The symbol shown below is for a 2-input NAND gate
   i) True
   ii) False
7. By applying DeMorgan's theorem to a NOR gate two identical truth tables can be produced
   i) True
   ii) False

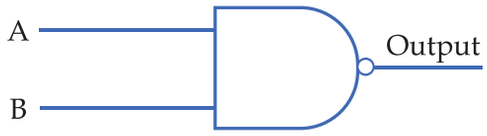


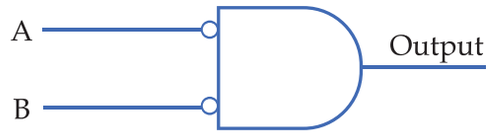
8. Give the truth table of following gate

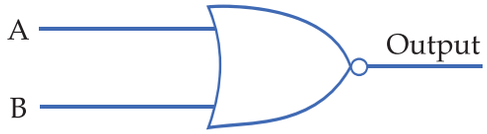
i) 

ii) 

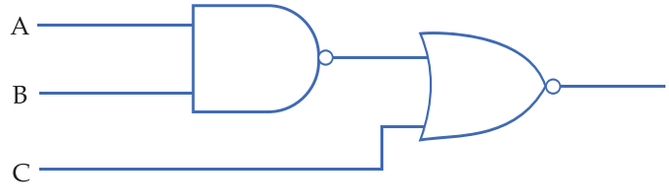
iii) 

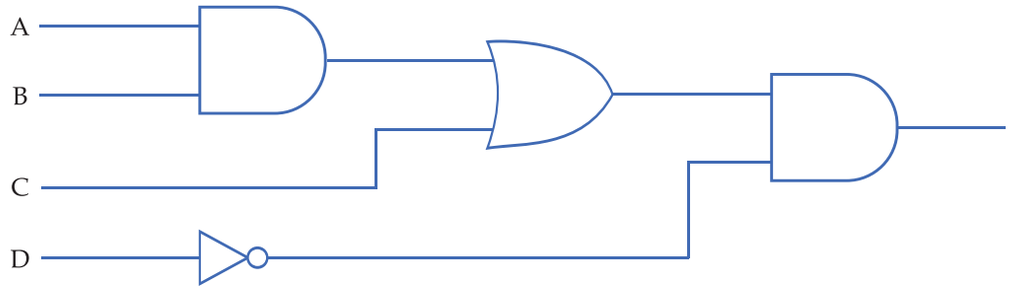
iv) 

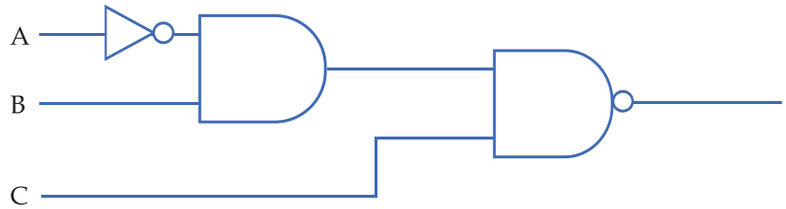
v) 

vi) 

9. Convert the following logic gate circuit into a Boolean expression, writing Boolean sub-expression next to each gate.

i) 

ii) 

iii) 

10. Draw the circuit diagram for the following using AND, OR and NOT gate(s)

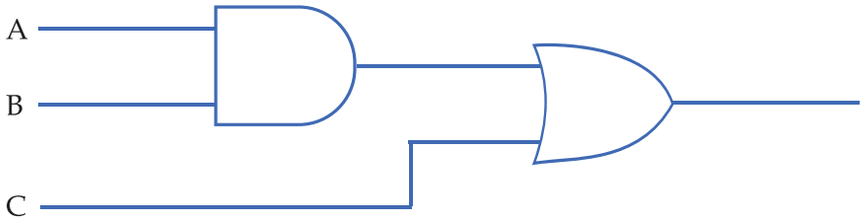
i) F(a,b,c,) = a + bc'

ii) F(x,y,z,) = xy + yz + xy'

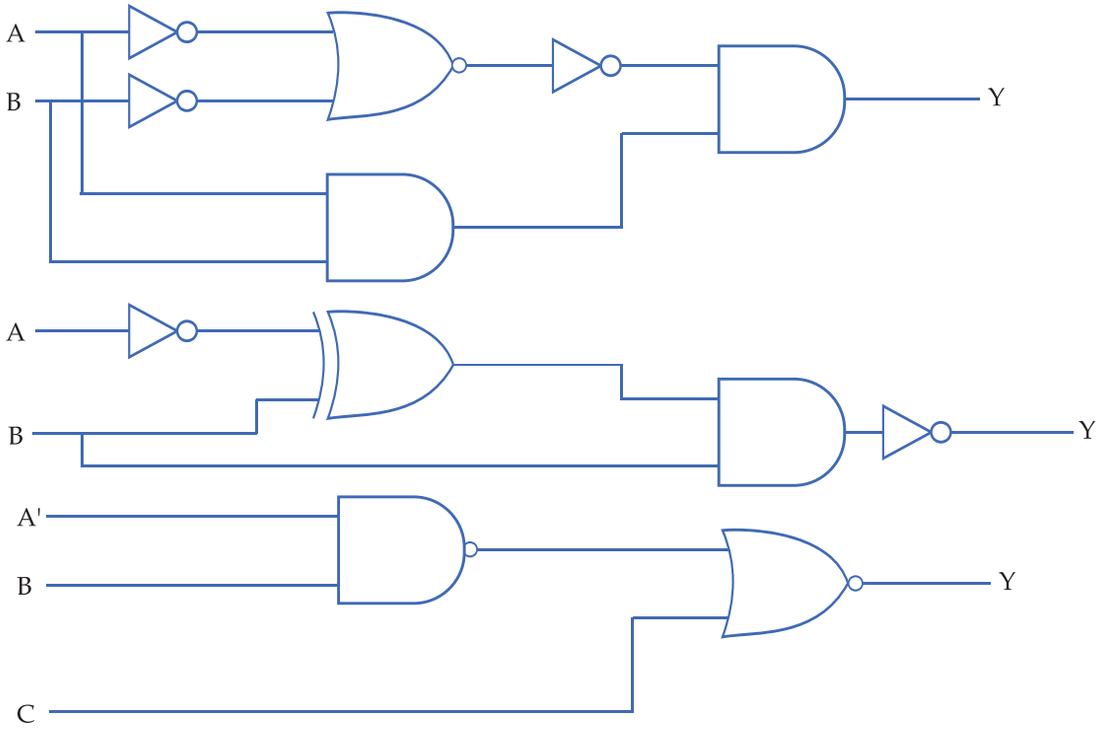
iii) f(w,x,y,z) = yz' + wxy' + wxz' + wx'z

iv) f(x,y,z) = (x + y). (y + z) . (z + x)

11. Convert the following circuit diagram using NAND gate(s) only. Do not simplify the expression.



12. Draw truth table and write Boolean function for following circuit



13. Draw the logic gate diagram for following expression using NOR gates only
    i) f(a,b,c) = ?(2,4,5,7,8)
    ii) f(x,y,z) = ?(0,1,3,4)
    iii) f(w,x,y,z) = ?(2,4,5,6,9)
14. Which gates are known as universal gate? Why?
15. Show how AND, OR Not gate functions can be implemented using only
    i) NAND gate
    ii) NOR gate