Boolean algebra

(1)

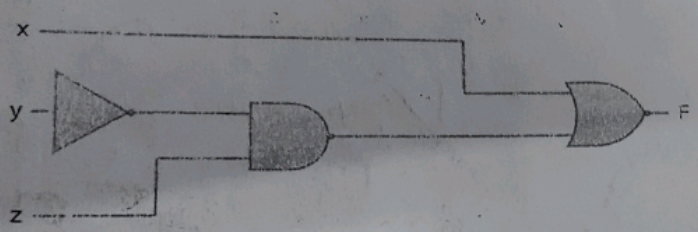
Boolean algebra can be considered as an algebra that deals with binary variables and logic operations. Boolean algebraic variables are designated by letters such as A, B, x, and y. The basic operations performed are AND, OR, and complement.

The Boolean algebraic functions are mostly expressed with binary variables, logic operation symbols, parentheses, and equal sign. For a given value of variables, the Boolean function can be either 1 or 0. For instance, consider the Boolean function:

F = x + y'z

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

F = x + y'z



- The Boolean function F = x + y'z is transformed from an algebraic expression into a logic diagram composed of AND, OR, and inverter gates.
- Inverter at input 'y' generates its complement y'.
- There is an AND gate for the term y'z, and an OR gate is used to combine the two terms (x and y'z).
- The variables of the function are taken to be the inputs of the circuit, and the variable symbol of the function is taken as the output of the circuit.

Note. A truth table can represent the relationship between a function and its binary variables. To represent a function in a truth table, we need a list of the 2^n combinations of n binary variables.

The truth table for the Boolean function F = x + y'z can be represented as:

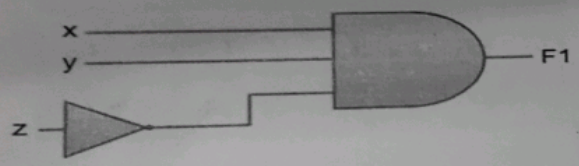
F = x + y'z

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

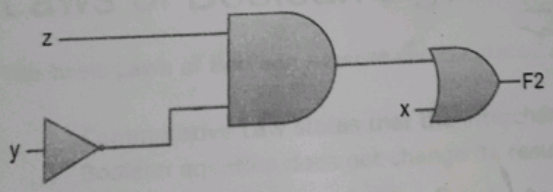
Examples of Boolean algebra simplifications using logic gates

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

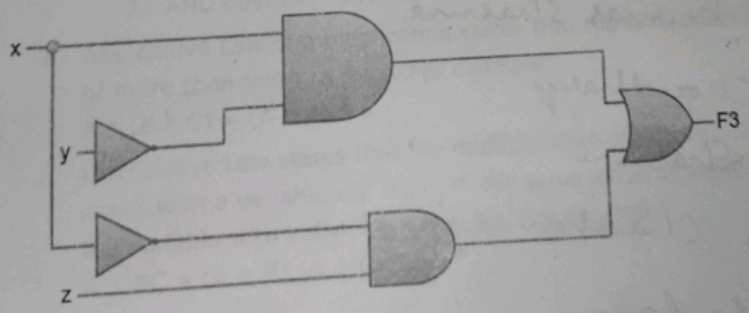
1. F1 = xyz'



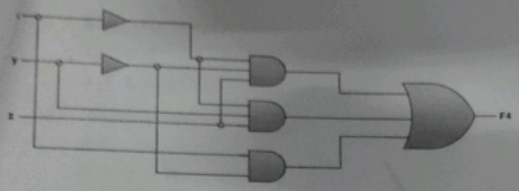
2. F2 = x + y'z



3. F3 = xy' + x'z



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



Truth tables for F1= xyz', F2= x+y'z, F3= xy'+x'z and F4= x'y'z+x'yz+xy'

x	y	z	F1	F2	F3	F4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

Laws of Boolean algebra

The basic Laws of Boolean Algebra can be stated as follows:

- Commutative Law states that the interchanging of the order of operands in a Boolean equation does not change its result. For example:
  1. OR operator → A + B = B + A
  2. AND operator → A * B = B * A
- Associative Law of multiplication states that the AND operation are done on two or more than two variables. For example:
  A * (B * C) = (A * B) * C
- Distributive Law states that the multiplication of two variables and adding the result with a variable will result in the same value as multiplication of addition of the variable with individual variables. For example:
  A + BC = (A + B) (A + C).

- Annulment law:
  A.0 = 0
  A + 1 = 1
- Identity law:
  A.1 = A
  A + 0 = A
- Idempotent law:
  A + A = A
  A.A = A
- Complement law:
  A + A' = 1
  A.A' = 0
- Double negation law:
  ((A)')' = A
- Absorption law:
  A.(A+B) = A
  A + AB = A

De Morgan's Law is also known as De Morgan's theorem, works depending on the concept of Duality. Duality states that interchanging the operators and variables in a function, such as replacing 0 with 1 and 1 with 0, AND operator with OR operator and OR operator with AND operator.

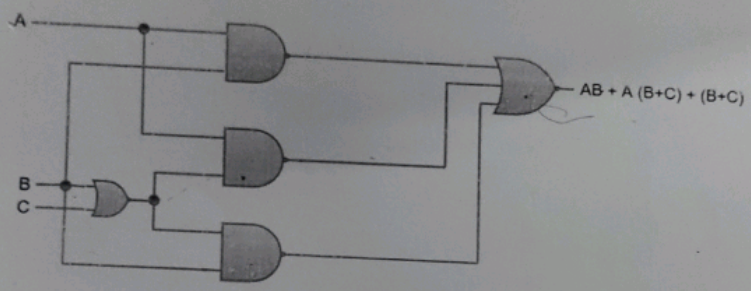
De Morgan stated 2 theorems, which will help us in solving the algebraic problems in digital electronics. The De Morgan's statements are:

1. "The negation of a conjunction is the disjunction of the negations", which means that the complement of the product of 2 variables is equal to the sum of the compliments of individual variables. For example, (A.B)' = A' + B'.
2. "The negation of disjunction is the conjunction of the negations", which means that compliment of the sum of two variables is equal to the product of the complement of each variable. For example, (A + B)' = A'B'.

Simplification using Boolean algebra

Let us consider an example of a Boolean function: **AB+A (B+C) + B (B+C)**

The logic diagram for the Boolean function AB+A (B+C) + B (B+C) can be represented as:



We will simplify this Boolean function on the basis of rules given by Boolean algebra.

AB + A (B+C) + B (B+C)

AB + AB + AC + BB + BC {Distributive law; A (B+C) = AB+AC, B (B+C) = BB+BC}

AB + AB + AC + B + BC {Idempotent law; BB = B}

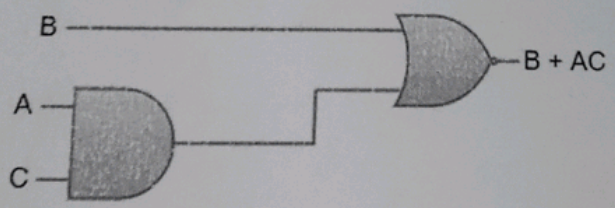
AB + AC + B + BC {Idempotent law; AB+AB = AB}

AB + AC +B {Absorption law; B+BC = B}

B + AC {Absorption law; AB+B = B}

Hence, the simplified Boolean function will be B + AC.

The logic diagram for Boolean function B + AC can be represented as:



Duality Principle

According to this principle, if we have postulates or theorems of Boolean Algebra for one type of operation then that operation can be converted into another type of operation (i.e., **AND can be converted to OR and vice-versa**) just by interchanging **'0 with 1', '1 with 0', '(+) sign with (.) sign'** and **'(.) sign with (+) sign'**. This principle ensures if a theorem is proved using postulates of Boolean algebra, then the dual of this theorem automatically holds and we need not prove it again separately.

Some Boolean expressions and their corresponding duals are given below,

Given Expression	Dual	Given Expression	Dual
0 = 1	1 = 0	A. (A+B) = A	A + A.B = A
0.1 = 0	1 + 0 = 1	AB = A + B	A+B = A.B
A.0 = 0	A + 1 = 1	(A+C) (A +B) = AB + AC	AC + AB = (A+B). (A+C)
A.B = B. A	A + B = B + A	A+B = AB + AB +AB	AB = (A+B).(A+B).(A+B)
A.A = 0	A + A = 1	AB + A + AB = 0	((A+B)).A.(A+B) = 1
A. (B.C) = (A.B). C	A+(B+C) = (A+B) + C		

Reducing Boolean Expressions

Every Boolean expression must be reduced to its simplest form before realizing it because each logic operation in the expression is carried out using hardware. Thus, realizing the simplest expression requires less circuitry hence reduces the cost of the system. Also, it is highly reliable and less complex in nature. For reducing the Boolean expression, we use the axioms and laws of Boolean algebra (see them in our previous article). Some instructions for reducing the given Boolean expression are listed below,

1. Remove all the parenthesis by multiplying all the terms if present.
2. Group all similar terms which are more than one, then remove all other terms by just keeping one.
   Example: ABC + AB +ABC + AB = ABC +ABC + AB +AB = ABC +AB

3. A variable and its negation together in the same term always results in a 0 value, it can be dropped.
   Example: A. BCC + BC = A. 0 + BC = BC
4. Look for the pair of terms that are identical except for one variable which may be missing in one of the terms. The larger term can be dropped.
   Example: ABCD + ABC = ABC(D + 1) = (ABC).1 = ABC
5. Look for the pair of terms that have the same variables, with one or more variables complimented. If a variable in one term of such a variable is complimented while in the second term it is not, then such terms can be combined into a single term with that variable dropped.
   Example
   ABCD + ABCD = ABC(D + D) = (ABC).1 = ABC
   OR
   AB(C+D) + AB(C+D) = AB [(C+D) + (C+D)] = AB. 1 =AB

Methods to simplify the boolean function

The methods used for simplifying the Boolean function are as follows –

- Karnaugh-map or K-map, and
- NAND gate method.

Karnaugh-map or K-map

The Boolean theorems and the De-Morgan's theorems are useful in manipulating the logic expression. We can realize the logical expression using gates. The number of logic gates required for the realization of a logical expression should be reduced to a minimum possible value by K-map method. This method can be done in two different ways, as discussed below.

Sum of Products (SOP) Form

It is in the form of sum of three terms AB, AC, BC with each individual term is a product of two variables. Say A.B or A.C etc. Therefore such expressions are known as expression in SOP form. The sum and products in SOP form are not the actual additions or multiplications. In fact they are the OR and AND functions. In SOP form, 0 represents a bar and 1 represents an unbar. SOP form is represented by Σ.

Given below is an example of SOP.

$Y(A,B) = A + B$

$A \cdot 1 + B \cdot 1$

$A \cdot (B + \bar{B}) + B \cdot (A + \bar{A})$

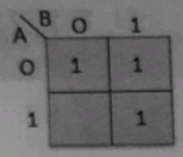
$AB + A\bar{B} + BA + B\bar{A}$

$(AB + A\bar{B} + \bar{A}B)$ ($\because AB + AB = AB$)

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

In SOP form $\bar{A}\bar{B} + \bar{A}B + AB$

00 01 11



$\sum m(0,1,3)$

Answer: $\bar{A}\bar{B} + \bar{A}B + AB = \bar{A} + B$

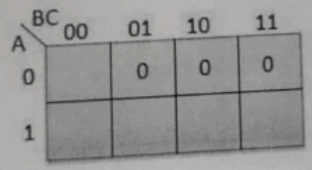
Product of Sums (POS) Form

It is in the form of product of three terms (A+B), (B+C), or (A+C) with each term is in the form of a sum of two variables. Such expressions are said to be in the product of sums (POS) form. In POS form, 0 represents an unbar and 1 represents a bar. POS form is represented by $\sqcap$.

Given below is an example of POS.

In POS form $(B + \bar{C})(\bar{A} + \bar{B})(\bar{B} + C)$

0 1 1 1 1 0



$\sqcap m(1,3,2)$

Answer : $(A + \bar{C})(A + \bar{B})$

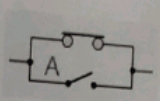
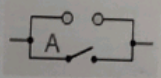
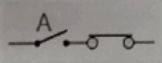
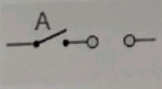
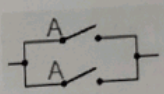
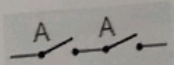
Boolean Algebra is the mathematics we use to analyse digital gates and circuits. We can use these "Laws of Boolean" to both reduce and simplify a complex Boolean expression in an attempt to reduce the number of logic gates required. *Boolean Algebra* is therefore a system of mathematics based on logic that has its own set of rules or laws which are used to define and reduce Boolean expressions.

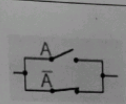
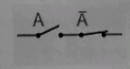
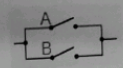
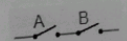
The variables used in **Boolean Algebra** only have one of two possible values, a logic "0" and a logic "1" but an expression can have an infinite number of variables all labelled individually to represent inputs to the expression, For example, variables A, B, C etc, giving us a logical expression of A + B = C, but each variable can ONLY be a 0 or a 1.

Examples of these individual laws of Boolean, rules and theorems for Boolean Algebra are given in the following table.

LOGIC GATES

Truth Tables for the Laws of Boolean

Boolean Expression	Description	Equivalent Switching Circuit	Boolean Algebra Law or Rule
A + 1 = 1	A in parallel with closed = "CLOSED"		Annulment
A + 0 = A	A in parallel with open = "A"		Identity
A . 1 = A	A in series with closed = "A"		Identity
A . 0 = 0	A in series with open = "OPEN"		Annulment
A + A = A	A in parallel with A = "A"		Idempotent
A . A = A	A in series with A = "A"		Idempotent
NOT A = A	NOT NOT A (double negative) = "A"		Double Negation

A + A = 1	A in parallel with NOT A = "CLOSED"		Complement
A . A = 0	A in series with NOT A = "OPEN"		Complement
A+B = B+A	A in parallel with B = B in parallel with A		Commutative
A.B = B.A	A in series with B = B in series with A		Commutative
A+B = A.B	invert and replace OR with AND		de Morgan's Theorem
A.B = A+B	invert and replace AND with OR		de Morgan's Theorem

The basic **Laws of Boolean Algebra** that relate to the *Commutative Law* allowing a change in position for addition and multiplication, the *Associative Law* allowing the removal of brackets for addition and multiplication, as well as the *Distributive Law* allowing the factoring of an expression, are the same as in ordinary algebra.

Each of the *Boolean Laws* above are given with just a single or two variables, but the number of variables defined by a single law is not limited to this as there can be an infinite number of variables as inputs too the expression. These Boolean laws detailed above can be used to prove any given Boolean expression as well as for simplifying complicated digital circuits.

A brief description of the various **Laws of Boolean** are given below with A representing a variable input.

Description of the Laws of Boolean Algebra

- Annulment Law – A term AND´ed with a "0" equals 0 or OR´ed with a "1" will equal 1
-

he digital world is based on the binary number system. Numerically, this involves only
and 1.) According to the need, we can use these symbols or we can equate them with
dealing with digital logic, we can specify that:

0 = False = No
1 = True = Yes

tem, either every statement or condition must be 'true' or 'false'. It
lse.

uilding block of a digital circuit. A simple logic gate has two
nt, every terminal is in one of the two binary conditions, low
ge levels. In m
imate

- A . 0 = 0 A variable AND'ed with 0 is always equal to 0
- A + 1 = 1 A variable OR'ed with 1 is always equal to 1

- Identity Law – A term OR´ed with a "0" or AND´ed with a "1" will always equal that term
  - A + 0 = A A variable OR'ed with 0 is always equal to the variable
  - A . 1 = A A variable AND'ed with 1 is always equal to the variable
- Idempotent Law – An input that is AND´ed or OR´ed with itself is equal to that input
  - A + A = A A variable OR'ed with itself is always equal to the variable
  - A . A = A A variable AND'ed with itself is always equal to the variable
- Complement Law – A term AND´ed with its complement equals "0" and a term OR´ed with its complement equals "1"
  - A . A = 0 A variable AND'ed with its complement is always equal to 0
  - A + A = 1 A variable OR'ed with its complement is always equal to 1
- Commutative Law – The order of application of two separate terms is not important
  - A . B = B . A The order in which two variables are AND'ed makes no difference
  - A + B = B + A The order in which two variables are OR'ed makes no difference
- Double Negation Law – A term that is inverted twice is equal to the original term
  - $\overline{\overline{A}} = A$ A double complement of a variable is always equal to the variable
- de Morgan´s Theorem – There are two "de Morgan's" rules or theorems,
- (1) Two separate terms NOR´ed together is the same as the two terms inverted (Complement) and AND´ed for example: $\overline{A+B} = \overline{A} \cdot \overline{B}$
- (2) Two separate terms NAND´ed together is the same as the two terms inverted (Complement) and OR´ed for example: $\overline{A \cdot B} = \overline{A} + \overline{B}$

orld is based on the binary number system. Numerically, this involves only
cording to the need, we can use these symbols or we can equate them with
th digital logic, we can specify that:

0 = False = No
1 = True = Yes

her every statement or condition must be 'true' or 'false'. It

block of a digital circuit. A simple logic gate has two
ery terminal is in one of the two binary conditions, low
vels. In most logic gates, the low state is approximately
ely five volts positive (+5 V). In other words, a logic
or more input signals to produce the desired output
of a combinati

A
B
C
INPUTS

Figure 2.

term logic g
basic electr
ed with each
logical and

Other algebraic Laws of Boolean not detailed above include:

- Distributive Law – This law permits the multiplying or factoring out of an expression.
  - A(B + C) = A.B + A.C (OR Distributive Law)
  - A + (B.C) = (A + B).(A + C) (AND Distributive Law)
- Absorptive Law – This law enables a reduction in a complicated expression to a simpler one by absorbing like terms.
  - A + (A.B) = A (OR Absorption Law)
  - A(A + B) = A (AND Absorption Law)
- Associative Law – This law allows the removal of brackets from an expression and regrouping of the variables.
  - A + (B + C) = (A + B) + C = A + B + C (OR Associate Law)
  - A(B.C) = (A.B)C = A . B . C (AND Associate Law)

Boolean Algebra Functions

Using the information above, simple 2-input AND, OR and NOT Gates can be represented by 16 possible functions as shown in the following table.

Function	Description	Expression
1	NULL	0
2	IDENTITY	1
3	Input A	A
4	Input B	B
5	NOT A	A
6	NOT B	B
7	A AND B (AND)	A . B
8	A AND NOT B	A . B
9	NOT A AND B	A . B
10	NOT AND (NAND)	A . B
11	A OR B (OR)	A + B
12	A OR NOT B	A + B
13	NOT A OR B	A + B
14	NOT OR (NOR)	A + B
15	Exclusive-OR	A . B + A . B
16	Exclusive-NOR	A . B + A . B

Laws of Boolean Algebra Example No1

Using the above laws, simplify the following expression: $(A + B)(A + C)$

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

$A.A + A.C + A.B + B.C$ – Distributive law

$A + A.C + A.B + B.C$ – Idempotent AND law $(A.A = A)$

$A(1 + C) + A.B + B.C$ – Distributive law

$A.1 + A.B + B.C$ – Identity OR law $(1 + C = 1)$

$A(1 + B) + B.C$ – Distributive law

$A.1 + B.C$ – Identity OR law $(1 + B = 1)$

$Q =$ $A + (B.C)$ – Identity AND law $(A.1 = A)$

Then the expression: $(A + B)(A + C)$ can be simplified to $A + (B.C)$ as in the Distributive law.