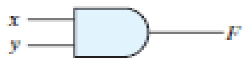
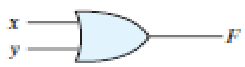
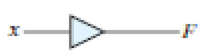
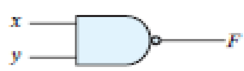
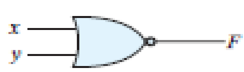
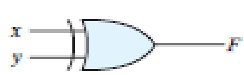
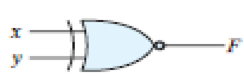
Digital Logic Gates

Boolean functions are expressed in terms of AND, OR, and NOT operations, it is easier to implement a Boolean function with these type of gates.

Name	Graphic symbol	Algebraic function	Truth table
AND		$F = x \cdot y$	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x + y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$F = x'$	x | F 0 | 1 1 | 0
Buffer		$F = x$	x | F 0 | 0 1 | 1
NAND		$F = (xy)'$	x y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = (x + y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Properties of XOR Gates

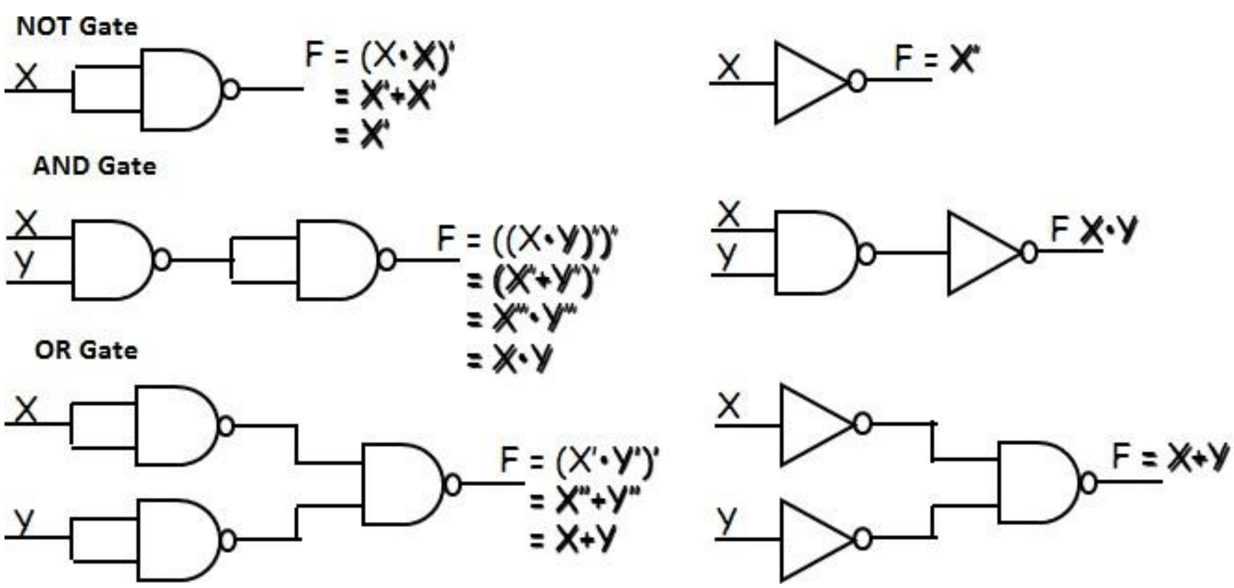
- XOR (also ⊕) : the "not-equal" function
- XOR(X,Y) = X ⊕ Y = X'Y + XY'
- Identities:
  - X ⊕ 0 = X
  - X ⊕ 1 = X'
  - X ⊕ X = 0
  - X ⊕ X' = 1
- Properties:
  - X ⊕ Y = Y ⊕ X
  - (X ⊕ Y) ⊕ W = X ⊕ (Y ⊕ W)

Universal Logic Gates

NAND and NOR gates are called Universal gates. All fundamental gates (NOT, AND, OR) can be realized by using either only NAND or only NOR gate. A universal gate provides flexibility and offers enormous advantage to logic designers.

NAND as a Universal Gate

NAND Known as a "universal" gate because ANY digital circuit can be implemented with NAND gates alone.
To prove the above, it suffices to show that AND, OR, and NOT can be implemented using NAND gates only.



Boolean Algebra: In 1854, George Boole developed an algebraic system now called Boolean algebra. In 1938, Claude E. Shannon introduced a two-valued Boolean algebra called switching algebra that represented the properties of bistable electrical switching circuits. For the formal definition of Boolean algebra, we shall employ the postulates formulated by E. V. Huntington in 1904.

Boolean algebra is a system of mathematical logic. It is an algebraic system consisting of the set of elements (0, 1), two binary operators called OR, AND, and one unary operator NOT. It is the basic mathematical tool in the analysis and synthesis of switching circuits. It is a way to express logic functions algebraically.

Boolean algebra, like any other deductive mathematical system, may be defined with aset of elements, a set of operators, and a number of unproved axioms or postulates. A *set* of elements is anycollection of objects having a common property. If **S** is a set and ***x*** and ***y*** are certain objects, then **x** Î **S**denotes that ***x*** is a member of the set **S**, and ***y*** Ï**S** denotes that ***y*** is not an element of **S**. A set with adenumerable number of elements is specified by braces: **A** = {1,2,3,4}, *i.e.* the elements of set **A** are thenumbers 1, 2, 3, and 4. A *binary operator* defined on a set S of elements is a rule that assigns to each pair ofelements from S a unique element from S._ Example: In *a***b*=*c*, we say that * is a binary operator if it specifies a rule for finding *c* from the pair (*a*,*b*)and also if *a*, *b*, *c* Î S.

Axioms and laws of Boolean algebra

Axioms or Postulates of Boolean algebra are a set of logical expressions that we accept without proof and upon which we can build a set of useful theorems.

	AND Operation	OR Operation	NOT Operation
Axiom1 :	0.0=0	0+0=0	$\overline{0}$=1
Axiom2:	0.1=0	0+1=1	$\overline{1}$=0
Axiom3:	1.0=0	1+0=1	
Axiom4:	1.1=1	1+1=1	

AND Law

Law1: A.0=0 (Null law)

Law2: A.1=A (Identity law)

Law3: A.A=A (Impotence law)

OR Law

Law1: A+0=A

Law2: A+1=1

Law3: A+A=A (Impotence law)

CLOSURE: The Boolean system is *closed* with respect to a binary operator if for every pair of Boolean values,it produces a Boolean result. For example, logical AND is closed in the Boolean system because it accepts only Boolean operands and produces only Boolean results.

_ A set *S* is closed with respect to a binary operator if, for every pair of elements of *S*, the binary operator specifies a rule for obtaining a unique element of *S*.

_ For example, the set of natural numbers N = {1, 2, 3, 4, … 9} is closed with respect to the binary operator plus (+) by the rule of arithmetic addition, since for any *a*, *b* Î N we obtain a unique *c* Î N by the operation *a* + *b* = c.

ASSOCIATIVE LAW:

A binary operator * on a set S is said to be associative whenever $(x * y) * z = x * (y * z)$ for all x, y, z Î S, forall Boolean values x, y and z.

COMMUTATIVE LAW:

A binary operator * on a set S is said to be commutative whenever $x * y = y * x$ for all $x, y, z \in S$

IDENTITY ELEMENT:

A set ***S*** is said to have an identity element with respect to a binary operation * on S if there exists an element $e \in S$ with the property $e * x = x * e = x$ for every $x \in S$

BASIC IDENTITIES OF BOOLEAN ALGEBRA

- *Postulate 1(Definition)*: A Boolean algebra is a closed algebraic system containing a set K of two or more elements and the two operators · and + which refer to logical AND and logical OR •$x + 0 = x$
- $x \cdot 0 = 0$
- $x + 1 = 1$
- $x \cdot 1 = 1$
- $x + x = x$
- $x \cdot x = x$
- $x + x' = x$
- $x \cdot x' = 0$
- $x + y = y + x$
- $xy = yx$
- $x + (y + z) = (x + y) + z$
- $x (yz) = (xy) z$
- $x (y + z) = xy + xz$
- $x + yz = (x + y)(x + z)$
- $(x + y)' = x' y'$
- $(xy)' = x' + y'$

- $(x')' = x$

DeMorgan's Theorem

(a) $(a + b)' = a'b'$

(b) $(ab)' = a' + b'$

Generalized DeMorgan's Theorem

(a) $(a + b + \ldots z)' = a'b' \ldots z'$
(b) $(a.b \ldots z)' = a' + b' + \ldots z'$

Basic Theorems and Properties of Boolean algebra Commutative law

Law1: A+B=B+A Law2: A.B=B.A

Associative law

Law1: A + (B +C) = (A +B) +C Law2: A(B.C) = (A.B)C

Distributive law

Law1: A.(B + C) = AB+ AC Law2: A + BC = (A + B).(A +C)

Absorption law

Law1: A +AB =A

Solution: A(1+B)
A

Law2: A(A +B) = A

Solution: A.A+A.B
A+A.B
A(1+B)
A

Consensus Theorem

Theorem1. AB+ A'C + BC = AB + A'C Theorem2. (A+B). (A'+C).(B+C) =(A+B).(A'+C)

The BC term is called the consensus term and is redundant. The consensus term is formed from a PAIR OF TERMS in which a variable (A) and its complement (A') are present; the consensus term is formed by multiplying the two terms and leaving out the selected variable and its complement

Consensus Theorem1 Proof:

AB+A'C+BC=AB+A'C+(A+A')BC
=AB+A'C+ABC+A'BC

=AB(1+C)+A’C(1+B)

= AB+ A’C

Principle of Duality

Each postulate consists of two expressions statement one expression is transformed into the other by interchanging the operations (+) and (·) as well as the identity elements 0 and 1.
Such expressions are known as duals of each other.
If some equivalence is proved, then its dual is also immediately true.
If we prove: (x.x)+(x’+x’)=1, then we have by duality: (x+x)·(x’.x’)=0

The Huntington postulates were listed in pairs and designated by part (a) and part (b) in below table.

Table for Postulates and Theorems of Boolean algebra

Part-A	**Part-B**
A+0=A	A.0=0
A+1=1	A.1=A
A+A=A (Impotence law)	A.A=A (Impotence law)
$A+\bar{A}=1$	$A.\bar{A}=0$
$\bar{\bar{A}}=A$ (double inversion law)	--
Commutative law: A+B=B+A	A.B=B.A
Associative law: A + (B +C) = (A +B) +C	A(B.C) = (A.B)C
Distributive law: A.(B + C) = AB+ AC	A + BC = (A + B).(A +C)
Absorption law: A +AB =A	A(A +B) = A
DeMorgan Theorem: $\overline{(A+B)} = \bar{A}.\bar{B}$	$\overline{(A.B)} = = \bar{A}+\bar{B}$
Redundant Literal Rule: $A+\bar{A}B=A+B$	$A.(\bar{A}+B)=AB$
Consensus Theorem: AB+ A’C + BC = AB + A’C	(A+B). (A’+C).(B+C) =(A+B).(A’+C)

Boolean Function

Boolean algebra is an algebra that deals with binary variables and logic operations.
A Boolean function described by an algebraic expression consists of binary variables, the constants 0 and 1, and the logic operation symbols.
For a given value of the binary variables, the function can be equal to either 1 or 0.

F(vars) = *expression*

Set of binary Variables

Operators (+, •, ‘)
Constants (0, 1)
Groupings (parenthesis)
Variables

Consider an example for the Boolean function

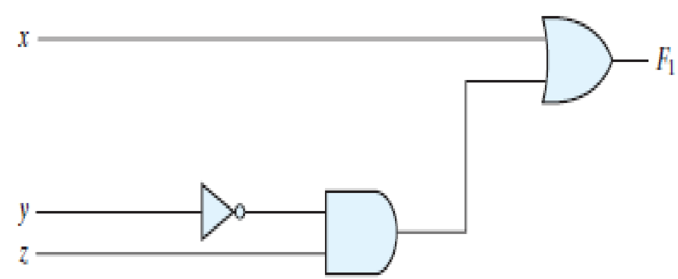
F1 = x + y’z

The function F1 is equal to 1 if x is equal to 1 or if both y' and z are equal to 1. F1 is equal to 0 otherwise. The complement operation dictates that when y' = 1, y = 0. Therefore, F1 = 1 if x = 1 or if y = 0 and z = 1.
A Boolean function expresses the logical relationship between binary variables and is evaluated by determining the binary value of the expression for all possible values of the variables.

A Boolean function can be represented in a truth table. The number of rows in the truth table is 2^n, where n is the number of variables in the function. The binary combinations for the truth table are obtained from the binary numbers by counting from 0 through $2^n - 1$.

Truth Table for F1

x	y	z	F_1
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



Gate Implementation of F1 = x + y'z

Note:
Q: Let a function F() depend on n variables. How many rows are there in the truth table of F() ?
A: 2^n rows, since there are 2^n possible binary patterns/combinations for the n variables.

Truth Tables

- Enumerates all possible combinations of variable values and the corresponding function value
- Truth tables for some arbitrary functions
  F1(x,y,z), F2(x,y,z), and F3(x,y,z) are shown to the below.

x	y	z	F_1	F_2	F_3
0	0	0	0	1	1
0	0	1	0	0	1

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

- Truth table: a unique representation of a Boolean function
- If two functions have identical truth tables, the functions are equivalent (and vice-versa).
- Truth tables can be used to prove equality theorems.
- However, the size of a truth table grows exponentially with the number of variables involved, hence unwieldy. This motivates the use of Boolean Algebra.

Boolean expressions-NOT unique

Unlike truth tables, expressions epresenting a Boolean function are NOT unique.

- Example:
  - F(x,y,z) = x'•y'•z' + x'•y•z' + x•y•z'
  - G(x,y,z) = x'•y'•z' + y•z'
- The corresponding truth tables for F() and G() are to the right. They are identical.
- Thus, F() = G()

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

Algebraic Manipulation (Minimization of Boolean function)

- Boolean algebra is a useful tool for simplifying digital circuits.
- Why do it? Simpler can mean cheaper, smaller, faster.
- Example: Simplify F = x'yz + x'yz' + xz.

  F= x'yz + x'yz' + xz

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

  = x'y•1 + xz

$= x'y + xz$

- Example: Prove
  $x'y'z' + x'yz' + xyz' = x'z' + yz'$
- **Proof:**
  $x'y'z' + x'yz' + xyz'$
  $= x'y'z' + x'yz' + x'yz' + xyz'$
  $= x'z'(y'+y) + yz'(x'+x)$
  $= x'z' \bullet 1 + yz' \bullet 1$
  $= x'z' + yz'$

Complement of a Function

- The complement of a function is derived by interchanging (• and +), and (1 and 0), and complementing each variable.
- Otherwise, interchange 1s to 0s in the truth table column showing F.
- The *complement* of a function IS NOT THE SAME as the *dual* of a function.

Example

- Find G(x,y,z), the complement of $F(x,y,z) = xy'z' + x'yz$
  Ans: $G = F' = (xy'z' + x'yz)'$
  $= (xy'z')' \bullet (x'yz)'$ *DeMorgan*
  $= (x'+y+z) \bullet (x+y'+z')$ *DeMorgan* again

Note: The complement of a function can also be derived by finding the function's *dual,* and then complementing all of the literals

Canonical and Standard Forms

We need to consider formal techniques for the simplification of Boolean functions.
Identical functions will have exactly the same canonical form.

- Minterms and Maxterms
- Sum-of-Minterms and Product-of- Maxterms
- Product and Sum terms
- Sum-of-Products (SOP) and Product-of-Sums (POS)

Definitions

Literal: A variable or its complement
Product term: literals connected by •
Sum term: literals connected by +
Minterm: a product term in which all the variables appear exactly once, either complemented or uncomplemented.

Maxterm: a sum term in which all the variables appear exactly once, either complemented or uncomplemented.

Canonical form: Boolean functions expressed as a sum of Minterms or product of Maxterms are said to be in canonical form.

Minterm

- Represents exactly one combination in the truth table.
- Denoted by m_j, where j is the decimal equivalent of the minterm's corresponding binary combination (b_j).
- A variable in m_j is complemented if its value in b_j is 0, otherwise is uncomplemented.

Example: Assume 3 variables (A, B, C), and j=3. Then, b_j = 011 and its corresponding minterm is denoted by m_j = A'BC

Maxterm

- Represents exactly one combination in the truth table.
- Denoted by M_j, where j is the decimal equivalent of the maxterm's corresponding binary combination (b_j).
- A variable in M_j is complemented if its value in b_j is 1, otherwise is uncomplemented.

Example: Assume 3 variables (A, B, C), and j=3. Then, b_j = 011 and its corresponding maxterm is denoted by M_j = A+B'+C'

Truth Table notation for Minterms and Maxterms

- Minterms and Maxterms are easy to denote using a truth table.

Example: Assume 3 variables x,y,z (order is fixed)

x	y	z	Minterm	Maxterm
0	0	0	x'y'z' = m_0	x+y+z = M_0
0	0	1	x'y'z = m_1	x+y+z' = M_1
0	1	0	x'yz' = m_2	x+y'+z = M_2
0	1	1	x'yz = m_3	x+y'+z'= M_3
1	0	0	xy'z' = m_4	x'+y+z = M_4
1	0	1	xy'z = m_5	x'+y+z' = M_5
1	1	0	xyz' = m_6	x'+y'+z = M_6
1	1	1	xyz = m_7	x'+y'+z' = M_7

Canonical Forms

- Every function F() has two canonical forms:
  - Canonical Sum-Of-Products (sum of minterms)
  - Canonical Product-Of-Sums (product of maxterms)

Canonical Sum-Of-Products:

The minterms included are those m_j such that F( ) = 1 in row j of the truth table for F( ).

Canonical Product-Of-Sums:

The maxterms included are those M_j such that F( ) = 0 in row j of the truth table for F( ).

Example

Consider a Truth table for $f_1(a,b,c)$ at right

The canonical sum-of-products form for f_1 is

$f_1(a,b,c) = m_1 + m_2 + m_4 + m_6$

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

The canonical product-of-sums form for f_1 is

$f_1(a,b,c) = M_0 \bullet M_3 \bullet M_5 \bullet M_7$

$= (a+b+c)\bullet(a+b'+c')\bullet (a'+b+c')\bullet(a'+b'+c')$.

- Observe that: $m_j = M_j'$

a	b	c	f_1
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Shorthand: ∑ and ∏

- $f_1(a,b,c) = \sum m(1,2,4,6)$, where ∑ indicates that this is a sum-of-products form, and m(1,2,4,6) indicates that the minterms to be included are m_1, m_2, m_4, and m_6.
- $f_1(a,b,c) = \prod M(0,3,5,7)$, where ∏ indicates that this is a product-of-sums form, and M(0,3,5,7) indicates that the maxterms to be included are M_0, M_3, M_5, and M_7.
- Since $m_j = M_j'$ for any *j*,
  $\sum m(1,2,4,6) = \prod M(0,3,5,7) = f_1(a,b,c)$
-

Conversion between Canonical Forms

- Replace ∑ with ∏ (or *vice versa*) and replace those *j'*s that appeared in the original form with those that do not.
    - Example:

$$\begin{aligned} f_1(a,b,c) &= a'b'c + a'bc' + ab'c' + abc' \\ &= m_1 + m_2 + m_4 + m_6 \\ &= \sum(1,2,4,6) \\ &= \prod(0,3,5,7) \\ &= (a+b+c)\bullet(a+b'+c')\bullet(a'+b+c')\bullet(a'+b'+c') \end{aligned}$$

Standard Forms

Another way to express Boolean functions is in standard form. In this configuration, the terms that form the function may contain one, two, or any number of literals.

There are two types of standard forms: the sum of products and products of sums.

The sum of products is a Boolean expression containing AND terms, called product terms, with one or more literals each. The sum denotes the ORing of these terms. An example of a function expressed as a sum of products is

$$F1 = y' + xy + x'yz'$$

The expression has three product terms, with one, two, and three literals. Their sum is, in effect, an OR operation.

A product of sums is a Boolean expression containing OR terms, called sum terms. Each term may have any number of literals. The product denotes the ANDing of these terms. An example of a function expressed as a product of sums is

$$F2 = x(y' + z)(x' + y + z')$$

This expression has three sum terms, with one, two, and three literals. The product is an AND operation.

Conversion of SOP from standard to canonical form

Example-1.

Express the Boolean function $F = A + B'C$ as a sum of minterms.

Solution: The function has three variables: A, B, and C. The first term A is missing two variables; therefore,

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

This function is still missing one variable, so

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

$= ABC + ABC' + AB'C + AB'C'$

The second term B'C is missing one variable; hence,

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

Combining all terms, we have

$F = A + B'C$

$= ABC + ABC' + AB'C + AB'C' + A'B'C$

But AB'C appears twice, and according to theorem $(x + x = x)$, it is possible to remove one of those occurrences. Rearranging the minterms in ascending order, we finally obtain

$F = A'B'C + AB'C + AB'C + ABC' + ABC$

$= m1 + m4 + m5 + m6 + m7$

When a Boolean function is in its sum-of-minterms form, it is sometimes convenient to express the function in the following brief notation:

$F(A, B, C) = \sum m\,(1, 4, 5, 6, 7)$

Example-2.

Express the Boolean function $F = xy + x'z$ as a product of maxterms.

Solution: First, convert the function into OR terms by using the distributive law:

$F = xy + x'z = (xy + x')(xy + z)$

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

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

The function has three variables: x, y, and z. Each OR term is missing one variable; therefore,

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

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

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

Combining all the terms and removing those which appear more than once, we finally obtain

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

$F = M0M2M4M5$

A convenient way to express this function is as

follows: $F(x, y, z) = \pi M(0, 2, 4, 5)$

The product symbol, π, denotes the ANDing of maxterms; the numbers are the indices of the maxterms of the function.