

UNIT -2

BOOLEAN ALGEBRA AND SWITCHING FUNCTIONS

- Fundamental postulates of Boolean algebra
- Basic theorems and properties
- Switching functions
- Canonical and Standard forms
- Algebraic simplification digital logic gates, properties of XOR gates
- Universal gates
- Multilevel NAND/NOR realizations

Boolean Algebra: Boolean algebra, like any other deductive mathematical system, may be defined with a set of elements, a set of operators, and a number of unproved axioms or postulates. A set of elements is any collection of objects having a common property. If S is a set and x and y are certain objects, then $x \in S$ denotes that x is a member of the set S , and $y \notin S$ denotes that y is not an element of S . A set with a denumerable number of elements is specified by braces: $A = \{1,2,3,4\}$, i.e. the elements of set A are the numbers 1, 2, 3, and 4. A *binary operator* defined on a set S of elements is a rule that assigns to each pair of elements 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 $a, b, c \in S$.

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 $N = \{1, 2, 3, 4, \dots, 9\}$ is closed with respect to the binary operator plus (+) by the rule of arithmetic addition, since for any $a, b \in N$ we obtain a unique $c \in N$ 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 $x, y, z \in S$, for all 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 $x \in S$

INVERSE:

A set S having the identity element e with respect to a binary operator $*$ is said to have an inverse whenever, for every $x \in S$, there exists an element $y \in S$ such that $x * y = e$

DISTRIBUTIVE LAW:

If $*$ and (\bullet) are two binary operators on a set S , $*$ is said to be distributive over (\bullet) whenever $x * (y \bullet z) = (x * y) \bullet (x * z)$

HUNTINGTON POSTULATES:

- ✓ Closure with respect to the operator $+$ and \bullet
- ✓ Identity element with respect to $+$ (0) and \bullet (1) $x+0=0+x=x$; $x \bullet 1=1 \bullet x=x$
- ✓ Commutative with respect to $+$ and \bullet ; $x+y = y+x$; $x \bullet y = y \bullet x$;
- ✓ Distributive over $+$ and \bullet ; $x \bullet (y+z) = (x \bullet y) + (x \bullet z)$; $x+(y \bullet z) = (x+y) \bullet (x+z)$... **Not valid in ordinary algebra**
- ✓ For every element of $x \in B$, there exists an element $x' \in B$ such that (a) $x + x' = 1$ and (b) $x \bullet x' = 0$.
- ✓ There exists at least two elements $x, y \in B$ such that $x \neq y$.

Huntington postulates do not include the associative law. However, this law holds for Boolean algebra. The distributive law of $+$ over (\bullet) is valid for Boolean algebra but not for ordinary algebra. Boolean algebra does not have additive or multiplicative inverses, \therefore no subtraction or division. The operator complement is not available in ordinary algebra. Ordinary algebra deals with real numbers, Boolean algebra deals with only two elements.

TWO-VALUED BOOLEAN ALGEBRA:

A two-valued Boolean algebra is defined on a set of two elements, $B = \{0,1\}$ with rules for the two binary operators $+$ and (\bullet) as shown in the following operator tables:

x	y	$x \bullet y$	$x+y$	x'
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

- Verify that the Huntington postulates hold true.

BASIC THEOREMS & PROPERTIES OF BOOLEAN ALGEBRA:

Duality Principle states that every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged.

- **Postulates a and b**

Postulate 2	$x + 0 = x$	$x \bullet 1 = x$
Postulate 3, Commutative	$x + y = y + x$	$xy = yx$
Postulate 4, Distributive	$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$
Postulate 5	$x + x' = 1$	$x \bullet x' = 0$

BASIC THEOREMS & PROPERTIES OF BOOLEAN ALGEBRA:

- Theorems *a* and *b*

Theorem	<i>a</i>	<i>b</i>
Theorem 1	$x + x = x$	$x \bullet x = x$
Theorem 2	$x + 1 = 1$	$x \bullet 0 = 0$
Theorem 3, Involution	$(x')' = x$	
Theorem 4, Associative	$x + (y + z) = (x + y) + z$	$x \bullet (y \bullet z) = (x \bullet y) \bullet z$
Theorem 5, DeMorgan	$(x + y)' = x'y'$	$(x \bullet y)' = x' + y'$
Theorem 6, Absorption	$x + xy = x$	$x(x + y) = x$

<p>Proof of Theorem 1(a)</p> $x + x = x$ $x + x = (x + x) \cdot 1 \quad \text{by postulate 2(b)}$ $= (x + x) \cdot (x + x')$ $= x + xx' \quad \text{by postulate 5(a)}$ $= x + 0 \quad \text{by postulate 4(b)}$ $= x \quad \text{by postulate 5(b)}$	<p>Proof of Theorem 1(b)</p> $x \bullet x = x$ $xx = xx + 0 \quad \text{Postulate 2(a)}$ $= xx + xx' \quad \text{Postulate 5(b)}$ $= x(x + x')$ $= x \bullet 1 \quad \text{Postulate 5(a)}$ $= x \quad \text{Postulate 2(b)}$
<p>Proof of Theorem 2(a)</p> $x + 1 = 1$ $= 1 \bullet (x + 1)$ $= (x + x')(x + 1)$ $= x + x' \bullet 1$ $= x + x'$ $= 1$	<p>Proof of Theorem 2(b)</p> <p>$x \bullet 0 = 0$ by duality</p> <p>The Duality Principle, also called De Morgan duality, asserts that Boolean algebra is unchanged when all dual pairs are interchanged.</p>
<p>Proof of Theorem 3</p> <p>$(x')' = x$</p> <p>We know that x' is the complement of x. If $x + x' = 1$ and $x \cdot x' = 0$, then $x + x' = 1 \Rightarrow x' + x = 1$ and $x \cdot x' = 0 \Rightarrow x' \cdot x = 0$ complement of $x \Rightarrow x$ is the complement $\Rightarrow (x')' = x$</p>	
<p>Proof of Theorem 4(a)</p> $x + (y + z) = (x + y) + z$ <p>Let $A = x + (y + z)$ and $B = (x + y) + z$ To Show: $A = B$</p> <p>First,</p> $xA = x[x + (y + z)]$ $= xx + x(y + z)$ $= x + x(y + z)$ $= x(1 + (y + z))$ $= x$ $xB = x[(x + y) + z] = x(x + y) + xz$ $= x + xz = x$	

Therefore $xA = xB = x$

Second,

$$\begin{aligned} x'A &= x'[x + (y + z)] = x'x + x'(y + z) \\ &= xx' + x'(y + z) = 0 + x'(y + z) \\ &= x'(y + z) \end{aligned}$$

$$\begin{aligned} x'B &= x'[(x + y) + z] \\ &= x'(x + y) + x'z = (x'x + x'y) + x'z \\ &= (xx' + x'y) + x'z = (0 + x'y) + x'z \\ &= x'y + x'z = x'(y + z) \end{aligned}$$

Therefore $x'A = x'B = x'(y + z)$

Finally,

$$A = A \cdot 1$$

$$= A(x + x')$$

$$= Ax + Ax'$$

$$= xA + x'A$$

$$= xB + x'A$$

$$= xB + x'B$$

$$= Bx + Bx'$$

$$= B(x + x')$$

$$= B \cdot 1$$

$$= B$$

Since $A = x + (y + z)$ and $B = (x + y) + z$, we have shown that $x + (y + z) = (x + y) + z$

Proof of Theorem 5(a)

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

X	Y	(x+y)	(x+y)'	x'	y'	x'y'
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

DeMorgan's Theorems:

a. $(A + B) = A * B$

b. $A * B = A + B$

Note: * = AND operation

Proof of DeMorgan's Theorem (b):

For any theorem $X=Y$, if we can show that $X Y = 0$, and that $X + Y = 1$, then

by the complement postulates, $A A = 0$ and $A + A = 1$,

$X = Y$. By the uniqueness of the complement, $X = Y$.

Thus the proof consists of showing that $(A * B) * (A + B) = 0$; and also that $(A * B) + (A + B) = 1$.

Prove: $(A * B) * (A + B) = 0$

$$\begin{aligned}(A * B) * (A + B) &= (A * B) * A + (A * B) * B \text{ by distributive postulate} \\ &= (A * A) * B + A * (B * B) \text{ by Associativity postulate} \\ &= 0 * B + A * 0 \text{ by Complement postulate} \\ &= 0 + 0 \text{ by Nullity theorem} \\ &= 0 \text{ by identity theorem}\end{aligned}$$

$$(A * B) * (A + B) = 0$$

Prove: $(A * B) + (A + B) = 1$

$$\begin{aligned}(A * B) + (A + B) &= (A + A + B) * (B + A + B) \text{ by distributivity } B * C + A = (B + A) * (C + A) \\ (A * B) + (A + B) &= (A + A + B) * (B + B + A) \text{ by associativity postulate} \\ &= (1 + B) * (1 + A) \text{ by complement postulate} \\ &= 1 * 1 \text{ by nullity theorem} \\ &= 1 \text{ by identity theorem}\end{aligned}$$

$$(A * B) + (A + B) = 1$$

Since $(A * B) * (A + B) = 0$, and $(A * B) + (A + B) = 1$,

$A * B$ is the complement of $A + B$, meaning that $A * B = (A + B)'$;

(note that ' = complement or NOT - double bars don't show in HTML)

Thus $A * B = (A + B)''$.

The involution theorem states that $A'' = A$. Thus by the involution theorem, $(A + B)'' = A + B$.

This proves DeMorgan's Theorem (b).

DeMorgan's Theorem (a) may be proven using a similar approach.

Proof of Theorem 6(a)

$$x + xy = x$$

$$x + xy = x \bullet 1 + xy = x(y+1) = x \bullet 1 = x$$

Proof of Theorem 6(b)

$$x(x+y) = x \text{ By duality}$$

Operator Precedence: **1. Parenthesis 2. NOT 3. AND 4. OR**

Prove $x + x'y = x + y$

$$x + x'y = (x+x')(x+y) = 1 \bullet (x+y) = x+y$$

Prove $xy+x'z+yz = xy+x'z$

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

$$= xy(1+z) + x'z(1+y)$$

$$= xy+x'z$$

Simplify $x'y'z + yz + xz$

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

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

$$= z (1 + y) = z(1) = z$$

Simplify $(x+y) [x' (y'+z')] + x'y'+x'z'$

$$\begin{aligned}
 &= (x+y) [x + (y'+z)'] + x'y'+x'z' \\
 &= (x+y) (x + yz) + x'y' + x'z' \\
 &= x + xyz + xy + yz + x'y' + x'z' \\
 &= x + xy + yz + x'y' + x'z' \\
 &= x + yz + x'y' + x'z' \\
 &= x + y' + yz + x'z' \\
 &= x + z' + y' + z \\
 &= 1
 \end{aligned}$$

SWITCHING FUNCTIONS

Let $T(x_1, x_2, x_3, \dots, x_n)$ be a switching expression. Each of the variables can assume any of two values 0 or 1 and hence there are 2^n combinations for determining the values of **T**. For example $T = x'z + xz' + x'y'$. Then $T(0,0,1) = 0'1 + 01' + 0'1' = 1 + 0 + 0 = 1$. Similarly **T** can be computer for every combination and a truth table may be built.

x	y	z	T
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

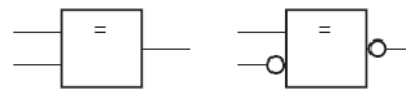
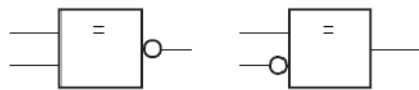
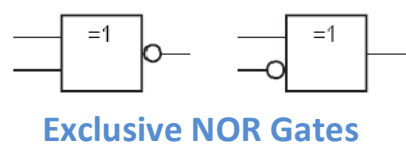
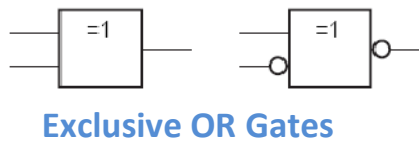
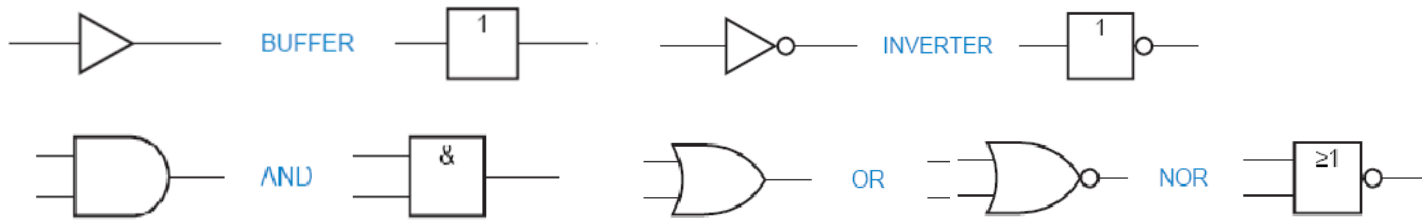
When built a truth table for $T = x'z + xz' + y'z'$, this will be identical to the above. Hence different switching functions may produce the same assignments. If a truth table is given for functions say **f** and **g**, then **(fg)**, **(f+g)**, **f'** and **g'** can easily built.

x	y	z	f	g	f'	g'	fg	f+g
0	0	0	1	0	0	1	0	1
0	0	1	0	1	1	0	0	1
0	1	0	1	0	0	1	0	1
0	1	1	1	1	0	0	1	1
1	0	0	0	1	1	0	0	1
1	0	1	0	0	1	1	0	0
1	1	0	1	1	0	0	1	1
1	1	1	1	0	0	1	0	1

Simplify

- i) $T(A,B,C,D) = A'C' + ABD + BC'D + AB'D' + ABCD'$ Ans : $A'C' + A[BD + D'(B'+C)]$
- ii) $T(A,B,C,D) = A'B + ABD + AB'CD' + BC$ Ans : $A'B + BD + ACD'$

IEEE Standard Logic Symbols



CANONICAL AND STANDARD FORMS

A Boolean (logic) function can be expressed in a variety of algebraic forms. For example $y = c \cdot a' + c \cdot b = c(a' + b) = c(c' + b + a')$

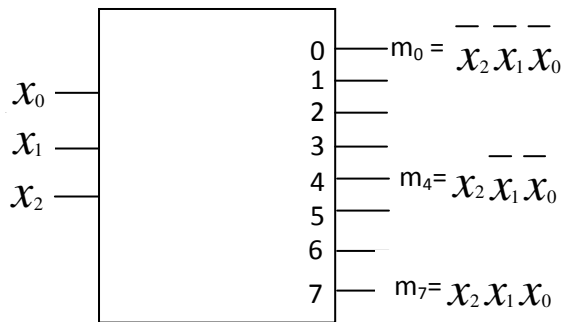
Each algebraic form entails specific gate implementation. A Boolean function can be uniquely described by its truth table, or in one of the canonical forms. Two dual canonical forms of a Boolean function are available: (a) The sum of Minterms (SoM) form (b) The product of Maxterms (PoM) form. A Minterm is a product of all variables taken either in their direct or complemented form. A Maxterm is a sum of all variables taken either in their direct or complemented form.

Minterms. n -to- 2^n Decoders

Consider, for example, all possible logic products of three variables $x = (x_2, x_1, x_0)$. There are $2^3 = 8$ different Minterms that can be written in the form $m_i = \overline{x_2} \overline{x_1} \overline{x_0}$, Where x' represents either variable x or its complement x' . All three-variable Minterms are listed in the following table:

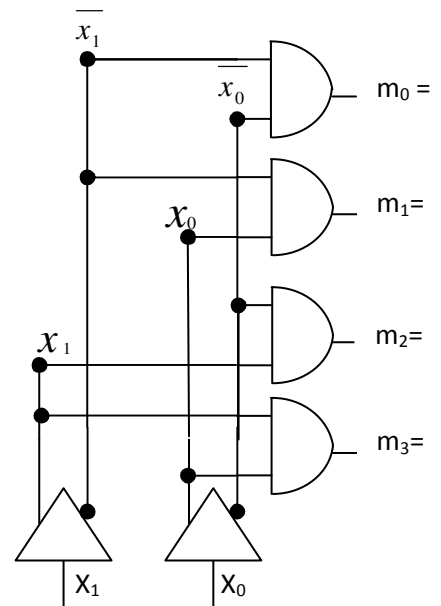
x	x ₂	x ₁	x ₀	Minterms	m ₀	m ₁	m ₂	m ₃	m ₄	m ₅	m ₆	m ₇
0	0	0	0	$m_0 = \overline{x_2} \overline{x_1} \overline{x_0}$	1	0	0	0	0	0	0	0
1	0	0	1	$m_1 = \overline{x_2} \overline{x_1} x_0$	0	1	0	0	0	0	0	0
2	0	1	0	$m_2 = \overline{x_2} x_1 \overline{x_0}$	0	0	1	0	0	0	0	0
3	0	1	1	$m_3 = \overline{x_2} x_1 x_0$	0	0	0	1	0	0	0	0
4	1	0	0	$m_4 = x_2 \overline{x_1} \overline{x_0}$	0	0	0	0	1	0	0	0
5	1	0	1	$m_5 = x_2 \overline{x_1} x_0$	0	0	0	0	0	1	0	0
6	1	1	0	$m_6 = x_2 x_1 \overline{x_0}$	0	0	0	0	0	0	1	0
7	1	1	1	$m_7 = x_2 x_1 x_0$	0	0	0	0	0	0	0	1

The logic circuit that generates all Minterms is called an n-to-2ⁿ decoder:



Example: Logic structure of a 2-to-4 decoder

x	x ₂	x ₁	x ₀	Minterms	M ₀	M ₁	M ₂	M ₃	M ₄
0	0	0	0	$m_0 = \overline{x_1} \overline{x_0}$	1	0	0	0	0
1	0	0	1	$m_1 = \overline{x_1} x_0$	0	1	0	0	0
2	0	1	0	$m_2 = x_1 \overline{x_0}$	0	0	1	0	0
3	0	1	1	$m_3 = x_1 x_0$	0	0	0	1	0



The Sum-of-Minterms (SoM) canonical form of a logic function

Any logic function y of n variables can be expressed as the logic sum of products of Minterms and the respective values of the function, that is:

$$y = f(x_{n-1}, \dots, x_0) = \sum_{i=0}^{2^n-1} y_i m_i$$

It is clearly equivalent to the sum of minterms for which the values of the function are 1, say, $y_j = 1$

$$y = f(x_{n-1}, \dots, x_0) = \sum_{\text{for all } j \text{ such that } y_j=1} m_j$$

Example of a 3-variable function

x	x ₂	x ₁	x ₀	y	m _j
0	0	0	0	0	
1	0	0	1	0	
2	0	1	0	1	m ₂
3	0	1	1	0	
4	1	0	0	1	m ₄
5	1	0	1	1	m ₅
6	1	1	0	0	
7	1	1	1	1	m ₇

$$\begin{aligned} y &= 0 \cdot m_0 + 0 \cdot m_1 + 1 \cdot m_2 + 0 \cdot m_3 + 1 \cdot m_4 + 1 \cdot m_5 + 0 \cdot m_6 + 1 \cdot m_7 \\ &= m_2 + m_4 + m_5 + m_7 \\ &= \Sigma(2,4,5,7) \text{ —a commonly used short notation} \end{aligned}$$

$$= \overline{x_2} \overline{x_1} \overline{x_0} + \overline{x_2} \overline{x_1} x_0 + \overline{x_2} x_1 \overline{x_0} + \overline{x_2} x_1 x_0$$

MAXTERMS

A logic sum (OR) of all variables taken in their direct or complemented form is called a Maxterm, M_i .

- A Maxterm is a complement of an equivalent Minterm

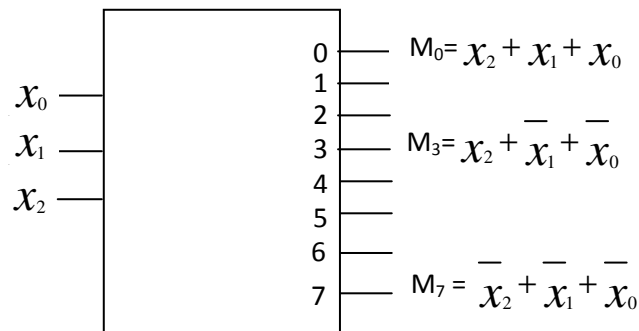
$$M_i = \overline{m_i} = \overline{\tilde{x}_{n-1} \cdot \dots \cdot \tilde{x}_0} = \tilde{x}_{n-1} + \dots + \tilde{x}_0$$

For example, all three-variable Maxterms are listed in the following table:

X	X ₂	X ₁	X ₀	MAXTERMS	M ₀	M ₁	M ₂	M ₃	M ₄	M ₅	M ₆	M ₇
0	0	0	0	M ₀ = $x_2 + x_1 + x_0$	0	1	1	1	1	1	1	1
1	0	0	1	M ₆ = $x_2 + x_1 + \overline{x_0}$	1	0	1	1	1	1	1	1
2	0	1	0	M ₅ = $x_2 + \overline{x_1} + x_0$	1	1	0	1	1	1	1	1
3	0	1	1	M ₄ = $x_2 + \overline{x_1} + \overline{x_0}$	1	1	1	0	1	1	1	1
4	1	0	0	M ₃ = $\overline{x_2} + x_1 + x_0$	1	1	1	1	0	1	1	1

5	1	0	1	$M_2 = \overline{x_2} + x_1 + \overline{x_0}$	1	1	1	1	1	0	1	1
6	1	1	0	$M_1 = \overline{x_2} + \overline{x_1} + x_0$	1	1	1	1	1	1	0	1
7	1	1	1	$M_0 = \overline{x_2} + \overline{x_1} + \overline{x_0}$	1	1	1	1	1	1	1	0

The logic circuit that generates all Maxterms is also called an n-to- 2^n decoder:



Example Express the Boolean function $F = A + BC'$ in a **sum of the (Products) Minterms**.

Firstly expand all terms with all three variables (A,B,C).

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

$$= ABC + \text{ABC}' + AB'C + AB'C' + \text{BC}'A + BC'A' =$$

Repeat Terms

$$= ABC + \text{ABC}' + AB'C + AB'C' + A'BC'$$

$$= m_7 + m_6 + m_5 + m_4 + m_2$$

This may also be expressed in short notation as $F(A,B,C) = \sum (2,4,5,7)$

Alternately, build a truth table (for $F = A + BC'$) and derive the expression

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Now the equation is formed by identifying the values of ABC for all combinations that produce '1' as output (F).

$$F = ABC + ABC' + AB'C + AB'C' + A'BC' = m_7 + m_6 + m_5 + m_4 + m_2$$

Product of Maxterms:

Example: $F = xy + x'z$... express this in terms of Maxterms

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

However this function has three variables; Let us convert each term into a three variable term

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

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

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

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

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

$$F = M4 + M5 + M0 + M2$$

$$\text{Conveniently it is written as } F(x,y,z) = \prod(0,2,4,5)$$

Conversion between canonical forms

Example;

$$F(A,B,C,D) = \sum (1,4,5,6,7) \text{ The complement of this is}$$

$$F'(A,B,C,D) = \sum (0,2,3) = m_0 + m_2 + m_3 ; \text{ Complement of this is}$$

$$F = (m_0 + m_2 + m_3)' = m_0' \cdot m_2' \cdot m_3' = M_0 \cdot M_2 \cdot M_3 = \prod(0,2,3)$$

Standard Forms: This is other way to express Boolean function. All Minterms and Maxterms must have all the variables in each term. However in standard form the expression will be in most minimal and completely simplified form. In the above example $F = A + BC'$ is in standard form or it is in **sum of the products** form, where as $F = ABC + ABC' + AB'C + AB'C' + A'BC'$ is in **sum of Minterms form**. Similarly for the case of Maxterms too. The Maxterms will have all the variables in each term where as in standard form it will be in simplified form and called **Product of Sums**. For example $F = (x' + y)(z + x)(z + y)$. This is in a **standard form** and also said to be the **Product of Sums, where as** $F = (x' + y + z) + (x' + y + z') + (x + y + z)(x + y' + z)$ is in **Product of Maxterms form**.

Examples: $F_1 = y' + xy + x'yz'$ **Sum of the products in two level implementation**

$F_2 = x(y'+z)(x'+y+z')$ **Product of Sums..... in two level implementation**

$F_3 = AB + C(D+E)$ **Non-Standard form ----- Can be two level or Three level Implementation**

Algebraic simplification digital logic gates, properties of XOR gates

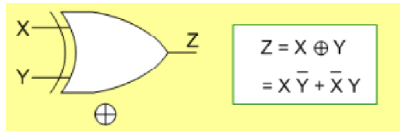
In addition to AND, OR, NOT, NAND and NOR gates, exclusive-OR (XOR) and exclusive-NOR (XNOR) gates are also used in the design of digital circuits. These have special functions and applications. These gates are particularly useful in arithmetic operations as well as error-detection and correction circuits. XOR and XNOR gates are usually found as 2-input gates.

XOR Gate:

The exclusive-OR (XOR), operator uses the symbol \oplus , and it performs the following logic operation:

$$X \oplus Y = X Y' + X' Y$$

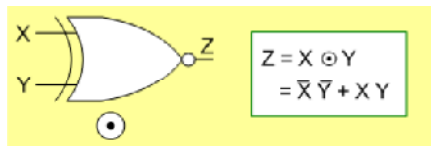
The graphic symbol and truth table of XOR gate is shown in the figure.



XNOR:

The exclusive-NOR (XNOR), operator uses the symbol \ominus , and it performs the logic operation. **X**

$\ominus Y = X Y + X' Y' = (X \oplus Y)'$. The graphic symbol and truth table of XNOR (Equivalence) gate is shown in the figure.



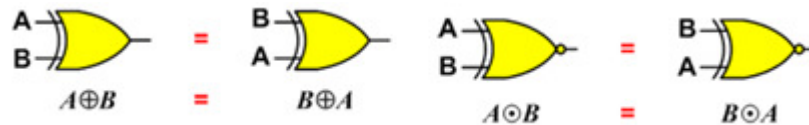
The result is 1 when either both X and Y are 0's or when both are 1's. That is why this gate is often referred to as the Equivalence gate. The truth tables clearly show that the exclusive-NOR operation is the complement of the exclusive-OR. This can also be shown by algebraic manipulation as follows:

$$\begin{aligned} (X \oplus Y)' &= (X Y' + X' Y)' \\ &= (X Y')' (X' Y)'' = (X' + Y) (X + Y') \\ &= (X Y + X' Y') \\ &= X \ominus Y \end{aligned}$$

Properties of XOR/XNOR Operations:

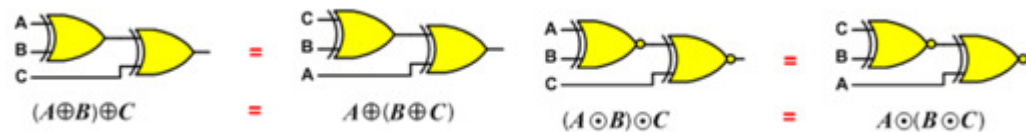
1- Commutativity

$A \oplus B = B \oplus A$, and $A \ominus B = B \ominus A$



2- Associativity

$A \oplus (B \oplus C) = (A \oplus B) \oplus C$, and $A \ominus (B \ominus C) = (A \ominus B) \ominus C$



Basic Identities of XOR Operation:

Any of the following identities can be proven using either truth tables or algebraically by replacing the \oplus operation by its equivalent Boolean expression:

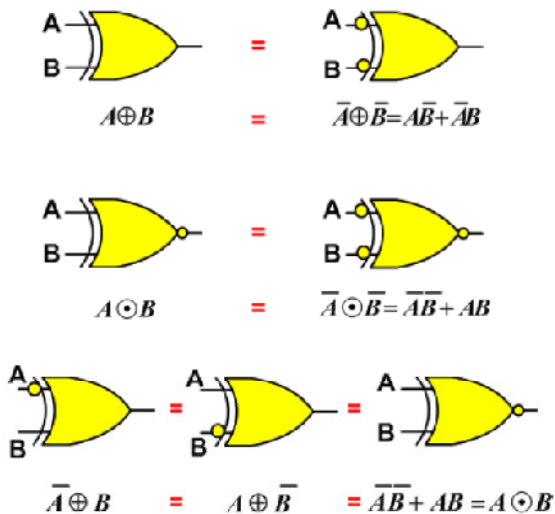
$$\begin{aligned} X \oplus 0 &= X \\ X \oplus 1 &= X' \\ X \oplus X &= 0 \\ X \oplus X' &= 1 \\ X \oplus Y' &= X' \oplus Y = (X \oplus Y)' = X \end{aligned}$$

Basic Identities of XOR Operation:

Any of the following identities can be proven using either truth tables or algebraically by replacing the \oplus operation by its equivalent Boolean expression:

$$\begin{aligned} X \oplus 0 &= X \\ X \oplus 1 &= X' \\ X \oplus X &= 0 \\ X \oplus X' &= 1 \\ X \oplus Y' &= X' \oplus Y = (X \oplus Y)' = X \odot Y \end{aligned}$$

The figure provides a graphical presentation of important XOR/XNOR rules and gate equivalence.

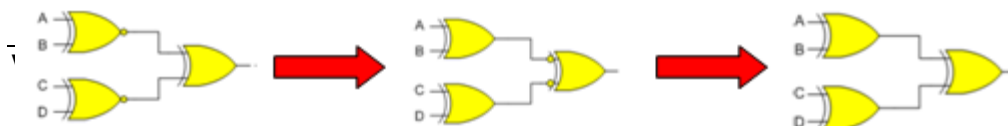


Example:

$$\text{Show that } (A \odot B) \oplus (C \odot D) = A \oplus B \oplus C \oplus D$$

Proving the above identity is easier done using graphical equivalence between gates as specified by the previous figure.

The following figure shows a step-by-step approach starting by the logic circuit corresponding to the left-hand-side of the identity and performing equivalent gate transformations till a circuit is reached that corresponds to the right-hand-side of the identity.

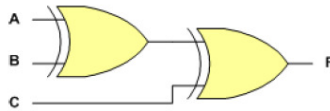


ODD Function:

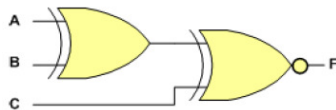
$X \oplus Y \oplus Z = 1$, IFF (if and only if) the number of 1's in the input combination is odd.

X	Y	Z	Odd
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	0
1	1	1	1

Likewise, $A \oplus B \oplus C \oplus D = 1$, IFF the number of 1's in the input combination is odd. In general, an exclusive-OR function of n-variables is an odd function which has a value of 1 IFF the number of 1's in the input combination is odd, otherwise it has a value of 0. Since XOR gates are only designed with 2 inputs, the 3-input XOR function is implemented by means of two 2-input XOR gates, as shown in figure.

**EVEN Function:**

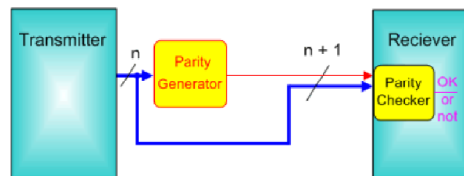
The complement of an odd function is an even function. The even function is equal to 1 when the number of 1's in the input combination is even. The complement of an odd function (an even function) is obtained by replacing the output gate with an exclusive-NOR gate, as shown in figure.

**Parity Generation and Checking:**

Exclusive-OR functions are very useful in systems using parity bits for error-detection. A parity bit is used for the purpose of detecting errors during transmission of binary information. A parity bit is an extra bit included with a binary message to make the total number of 1's in this message (including the parity bit) either odd or even.

The message, including the parity bit, is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity does not correspond with the one transmitted.

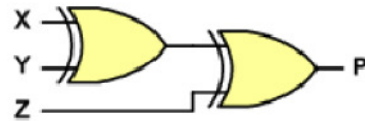
The circuit that generates the parity bit at the transmitter side is called a parity generator. The circuit that checks the parity at the receiver side is called a parity checker.



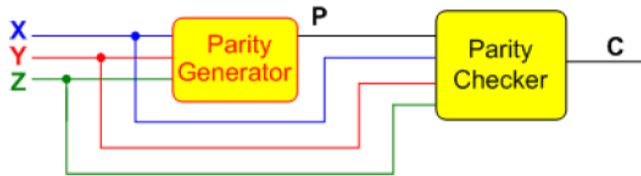
x	y	z	Parity Bit
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

As an example, consider a 3-bit message to be transmitted together with an even parity bit. The table shows the truth table for the even parity generator.

The three bits, X, Y, and Z, constitute the message and are the inputs to the even parity generator circuit whose output is the parity bit P. For even parity, whenever the message bits (X, Y & Z) have an odd number of 1's, the parity bit P must be 1. Otherwise, P must be 0. Therefore, P can be expressed as a three-variable exclusive-OR function: $P = X \oplus Y \oplus Z$. The logic diagram for the even parity generator circuit is shown in the figure.



The 4 bits (X, Y, Z & P) are transmitted to their destination, where they are applied to a parity-checker circuit to check for possible errors in the transmission.



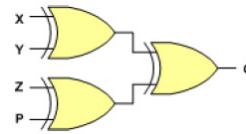
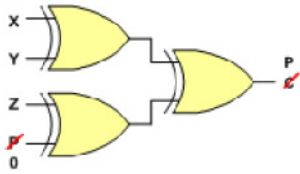
Since the information was transmitted with even parity, the received four bits must have an even number of 1's. The parity checker generates an error signal ($C = 1$), whenever the received four bits have an odd number of 1's. The table below shows the truth table for the even-parity checker.

Received Data					Parity Error Check	Received Data					Parity Error Check
x	y	z	p	C	x	y	z	p	C		
0	0	0	0	0	1	0	0	0	1		
0	0	0	1	1	1	0	0	1	0		
0	0	1	0	1	1	0	1	0	0		
0	0	1	1	0	1	0	1	1	1		
0	1	0	0	1	1	1	0	0	0		
0	1	0	1	0	1	1	0	1	1		
0	1	1	0	0	1	1	1	0	1		
0	1	1	1	1	1	1	1	1	0		

Obviously, the parity checker error output signal C is given by the following expression:

$C = X \oplus Y \oplus Z \oplus P$. The logic diagram of the even-parity checker is shown in the figure.

It is worth noting that the parity generator can also be implemented with the circuit of this figure if the input P is connected to logic-0 and the output is marked with P. This is because $Z \oplus 0 = Z$, causing the value of Z to pass through the gate unchanged.



The advantage of this is, the same circuit can be used for both parity generation and checking.

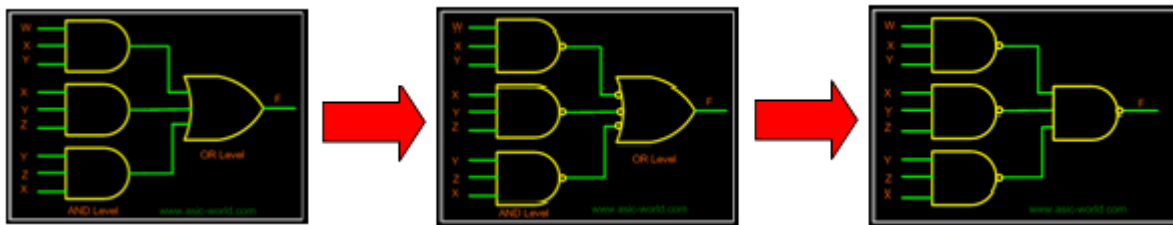
Universal Gates:

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. But there are some rules that need to be followed when implementing NAND or NOR based gates. Any logic function can be implemented using NAND gates. To achieve this, first the logic function has to be written in Sum of Product (SOP) form. Once logic function is converted to SOP, then is very easy to implement using NAND gate. In other words any logic circuit with AND gates in first level and OR gates in second level can be converted into a NAND-NAND gate circuit.

Consider the following SOP expression

$$F = W.X.Y + X.Y.Z + Y.Z.W$$

The above expression can be implemented with three AND gates in first stage and one OR gate in second stage as shown in figure



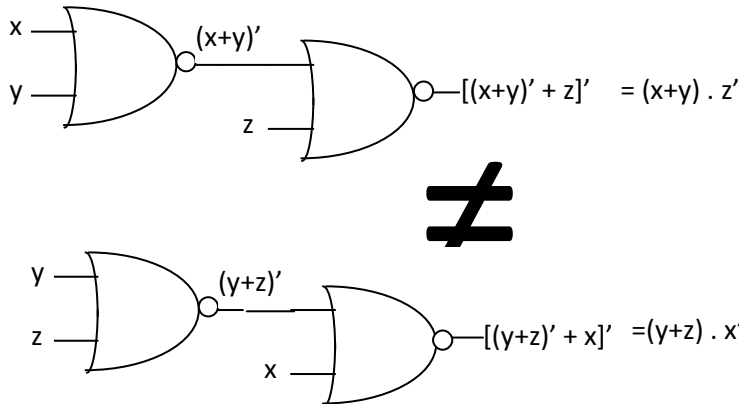
olean function	Operator Symbol	Name	Comments
$F_0 = 0$		NULL	Binary Constant 0
$F_1 = xy$	$x.y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x but not y
$F_3 = x$		Transfer	x'
$F_4 = x'y$	y/x	Inhibition	y but not x
$F_5 = y$		transfer	Y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive - OR	x or y but not both
$F_7 = x + y$	$x+y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	X equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y, then x
$F_{12} = x'$	X'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x, then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	NOT-AND
$F_{15} = 1$		Identity	Binary Constant

- Two functions that produce 0 or 1
- Four functions with unary operator: Complement and transfer
- Ten functions with binary operator that defines eight different operations: AND, OR, NAND, NOR, Ex-Or, Equivalence, inhibition and implication.

Nonassociativity of NAND and NOR Gates

$(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z) \dots$ For NOR gates

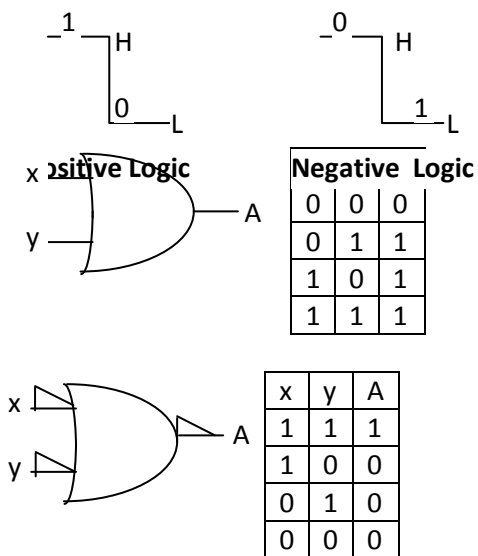
$(x \uparrow y) \uparrow z \neq x \uparrow (y \uparrow z) \dots$ For NAND gates



The same can be proved even for NAND gate.

Hence for a multiple in-put NAND or NOR shall be understood as multiple input OR gate followed by an inverter only for NOR and multiple input AND gate followed by an inverter only for NAND.

Positive Logic and Negative LOGIC



Multilevel NAND/NOR realizations

Incompletely Specified Function

A	B	C	D	OUT	A'B'C'D+
0	0	0	0	0	A'B'CD'+
0	0	0	1	1	A'BCD'+
0	0	1	0	1	AB'C'D+
0	0	1	1	0	AB'CD'+
0	1	0	1	0	ABCD'+.....
0	1	1	0	1	
1	0	0	1	1	
1	0	1	0	1	
1	1	0	1	0	
1	1	1	0	1	
1	1	1	1	0	
0	1	0	0	D	
1	0	0	0	D	
1	0	1	1	D	
1	1	0	0	D	
1	1	1	1	D	

Don't Care Terms. Can be assigned '0' or '1' for minimisation processes

Examples:

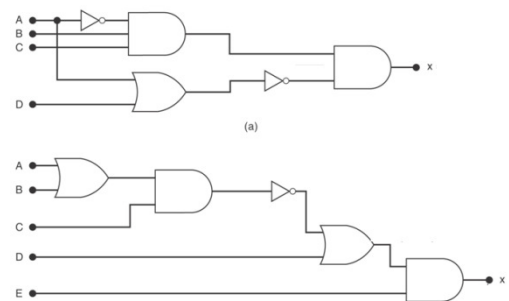
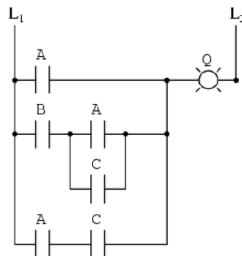
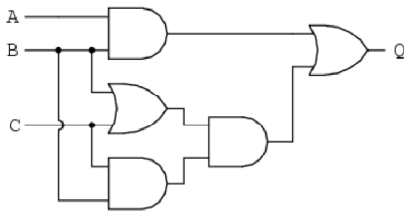
- $F = [(ab)' + (cd)' + e]'$
- $F = a'[b'+c(d+e')+f'g'] + hi'j + k$
- $F = [(a+b')c + d]e' + f$
- $F_1 = (A' + B)C + F' + DE \dots F \text{ as given in (1)}$

Assignments

Implement the following expressions using i) NOR gates only ii) NAND gates only

- $F(A,B,C,D) = \sum m(11,12,13,14,15)$
- $F(A,B,C,D) = \sum m(3,7,11,12,13,15)$
- $F(A,B,C,D) = \prod m(3,7,11,12,13,15)$

Simplify



SOME QUESTIONS

1.(a)State Duality theorem.List Boolean laws and their Duals.

(b)Simplify the following Boolean functions to minimum number of literals:

i. $F = ABC + ABC' + A'B$

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

(c)Realize XOR gate using minimum number of NAND gates

2. Simplify the following Boolean expressions using K-map and implement them using NOR gates:

(a) $F(A, B, C, D) = AB'C' + AC + A'CD'$

(b) $F(W, X, Y, Z) = W'X'Y'Z' + WXY'Z' + W'X'YZ + WXYZ$

3.(a)Simplify the following Boolean functions to minimum number of literals:

i. $(a + b)' (a' + b)'$

ii. $y(wz' + wz) + xy$

(b) Prove that AND-OR network is equivalent to NAND-NAND network.

(c) State Duality theorem. List Boolean laws and their Duals.

4.(a)What are don't-care conditions? Explain its advantage with example.

5. (a) List the Minterms and Maxterms for three binary variables.Draw the truthtable and express the Boolean function $F(A,B,C)$ whose minterms are 1,3,5,7as Canonica Sum of Minterms form.

(b) Simplify the following Boolean functions to minimum number of literals:

i. $F = X'Y' + XYZ + X'Y$

ii. $F = X + Y[Z + (X+Z)']]$

(c) For the logic expression $Y = AB' + A'B$:

i. Obtain the truth table.

ii. Name the operation performed.

iii. Realize this using AND, OR, NOT gates.

6.(a)State and prove the following Boolean laws:

i. Commutative

ii. Associative

iii. Distributive.

(b)Find the complement of the following Boolean functions and reduce them to minimum number of literals:

i. $(bc' + a'd)(ab' + cd')$

ii. $b'd + a'bc' + acd + a'bc$

(c) Which gate can be used as parity checker? Why?
