

Boolean Algebra & Logic Gates



ATTY. MANUEL O. DIAZ JR.

Basic Definitions



- 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 \mathbf{S} is a set and \mathbf{x} and \mathbf{y} are certain objects, then $\mathbf{x} \in \mathbf{S}$ denotes that \mathbf{x} is a member of the set \mathbf{S} , and $\mathbf{y} \notin \mathbf{S}$ denotes that \mathbf{y} is not an element of \mathbf{S} .

Basic Definitions



- A set with a denumerable number of elements is specified by braces: $\mathbf{A} = \{1,2,3,4\}$, *i.e.* the elements of set \mathbf{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



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

Associative Law



- A binary operator $*$ on a set S is said to be associative whenever

$$(x * y) * z = x * (y * z) \quad \text{for all } x, y, z \in S$$

Commutative Law



- A binary operator $*$ on a set S is said to be commutative whenever

$$x * y = y * x \quad \text{for all } x, y, z \in S$$

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

$$e * x = x * e = x \quad \text{for every } x \in S$$

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 \cdot are two binary operators on a set S , $*$ is said to be distributive over \cdot whenever

$$x * (y \cdot z) = (x * y) \cdot (x * z)$$

Huntington Postulates



- Closure with respect to the operator $+$ and \cdot .
- Identity element with respect to $+$ (0) and \cdot (1)
- Commutative with respect to $+$ and \cdot .
- Distributive over $+$ and \cdot .
- For every element of $x \in B$, there exists an element $x' \in B$ such that (a) $x + x' = 1$ and (b) $x \cdot x' = 0$.
- There exists at least two elements $x, y \in B$ such that $x \neq y$.

Boolean Algebra *vs* Ordinary Algebra



- Huntington postulates do not include the associative law. However, this law holds for Boolean algebra.
- The distributive law of $+$ over \cdot 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 \cdot as shown in the following operator tables:

xy	$x \cdot y$
00	0
01	0
10	0
11	1

xy	$x+y$
00	0
01	1
10	1
11	1

x	x'
0	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	$x + 0 = x$	$x \cdot 1 = x$
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	$x + x' = 1$	$x \cdot x' = 0$

Basic Theorems & Properties of Boolean Algebra



- **Theorems *a* and *b***

Theorem 1	$x + x = x$	$x \cdot x = x$
Theorem 2	$x + 1 = 1$	$x \cdot 0 = 0$
Theorem 3, Involution	$(x')' = x$	
Theorem 4, Associative	$x + (y + z) = (x + y) + z$	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$
Theorem 5, DeMorgan	$(x + y)' = x'y'$	$(x \cdot y)' = x' + y'$
Theorem 6, Absorption	$x + xy = x$	$x(x + y) = x$

Proof of Theorem 1(a)



$$x + x = x$$

$$x + x = (x + x) \cdot 1$$

by postulate 2(b)

$$= (x + x) \cdot (x + x')$$

by postulate 5(a)

$$= x + xx'$$

by postulate 4(b)

$$= x + 0$$

by postulate 5(b)

$$= x$$

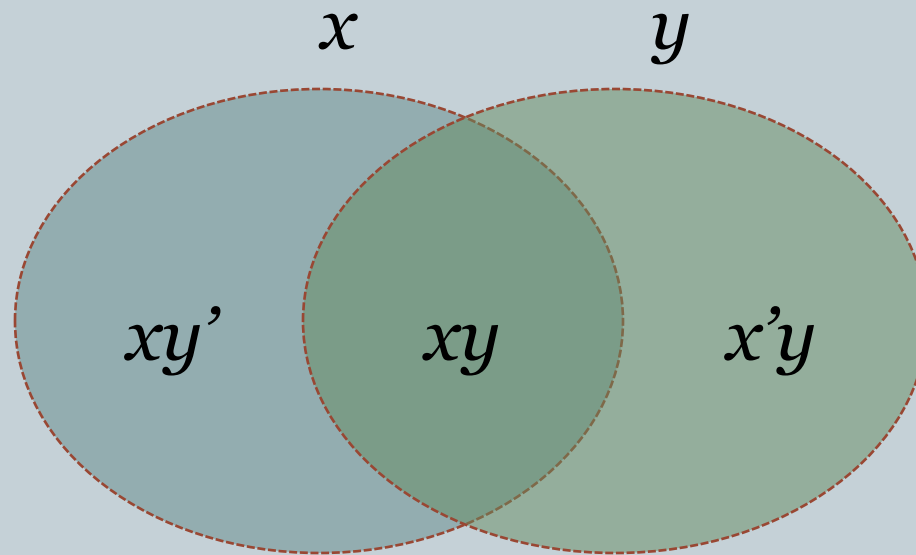
by postulate 2(a)

Operator Precedence



- Parenthesis
- NOT
- AND
- OR

Venn Diagram



$x'y'$

Boolean Function



- A Boolean function is an expression formed with binary variables, the two binary operators OR and AND, the unary operator NOT, parenthesis, and equal sign.
- A binary variable can take the value of 0 or 1, and for a given value of the variables, the function can be either 0 or 1.

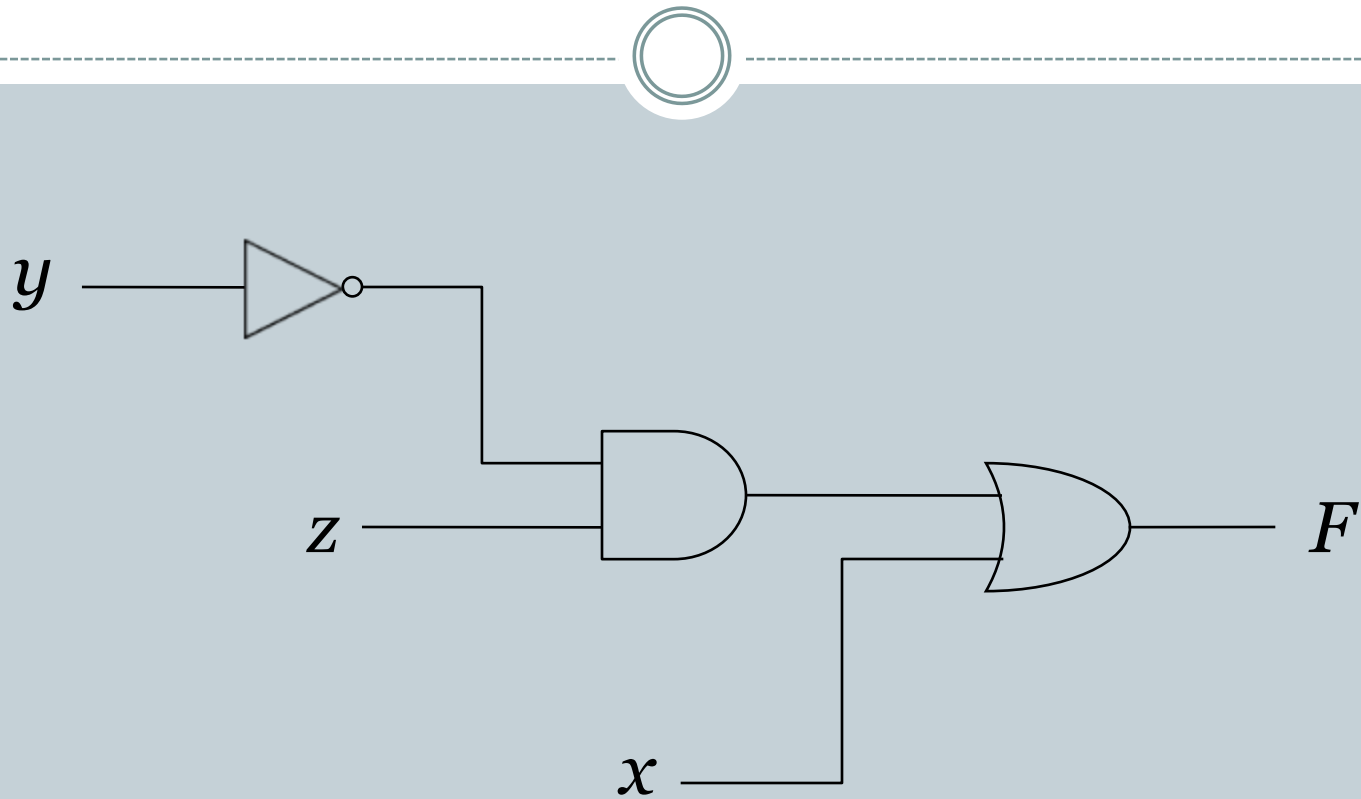
Truth Table



- Any Boolean function can be represented in a truth table, where the number of rows is 2^n , and n is the number of binary variables in the function.
- Example: $F = x + y'z$

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

Logic Diagram



Logic diagram for $F = x + y'z$

Algebraic Manipulation



- A *literal* is a primed or unprimed variable.
- When a Boolean function is implemented with logic gates, *each literal* in the function designates an *input* to a gate, and *each term* is implemented with a *gate*.
- The *minimization* of the number of literals and the number of terms results in a circuit with *less* equipment.

Algebraic Manipulation: Examples



- $x + x'y = (x + x')(x + y) = 1 \cdot (x + y) = x + y$
- $x(x' + y) = xx' + xy = 0 + xy = xy$
- $x'y'z + x'yz + xy' = x'z(y' + y) + xy' = x'z + xy'$
- $xy + x'z + yz = xy + x'z + yz(x + x')$
 $= xy + x'z + xyz + x'yz$
 $= xy(1 + z) + x'z(1 + y)$
 $= xy + x'z$
- $(x + y)(x' + z)(y + z) = (x + y)(x' + z)$ *by duality.*

Complement of a Function



- The complement of a function F is F' and is obtained from an interchange of 0's for 1's and 1's for 0's in the value of F .

- Examples

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

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

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

- A simpler procedure for deriving the complement of a function is to take the dual of the function and complement each literal.

Minterms



- A binary variable may appear either in its normal form (x) or in its complement form (x').
- Consider two binary variables x and y combined with an AND operation. Since each variable may appear in either the normal or complementary form, there are four possible combinations: xy , xy' , $x'y$, and $x'y'$.
- Each is called a *minterm* or *standard product*.
- Any Boolean function can be expressed as a sum of minterms (by *sum* is meant the ORing of terms).
- n variables can be combined to form 2^n minterms.

Maxterms



- n variables forming an OR term, with each variable being primed or unprimed, provide 2^n possible combinations, called *max terms* or *standard sums*.
- Any Boolean function can be expressed as a product of maxterms (by *product* is meant the *ANDing* of terms).
- Each maxterm is the complement of its corresponding minterm, and *vice versa*.
- Boolean functions expressed as a *sum* of minterms or *product* of maxterms are said to be in *canonical form*.

Minterms & Maxterms for 3 Binary Variables



	Minterms		Maxterms	
$x y z$	Term	Designation	Term	Designation
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 Form



- The function $F_1 = x'y'z + xy'z' + xyz$ is in canonical form. It can be written as $F_1 = m_1 + m_4 + m_7$ or in short notation, $F_1 = \Sigma(1,4,7)$.
- The function $F_2 = (x + y + z)(x + y + z')(x + y' + z)$ is likewise in canonical form. It can be written as $F_2 = M_0 + M_1 + M_2$ or in short notation, $F_2 = \pi(0,1,2)$.
- $m_j' = M_j$ which means that
 $F(x,y,z) = \pi(0,2,4,5) = \Sigma(1,3,6,7)$ and that
 $F'(x,y,z) = \Sigma(0,2,4,5) = \pi(1,3,6,7)$.

Standard Form



- The *sum of products* is a Boolean expression containing AND terms, called *product terms*, of one or more literals each. The *sum* denotes ORing of these terms.
- *Ex. $F_1 = y' + xy + x'yz'$*
- 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.
- *Ex. $F_2 = x(y' + z)(x' + y + z' + w)$*

Conversion Between Forms



- From *sum of products*, obtain *standard products* by using Postulate 2(b) and Postulate 5(a).
- From *product of sums*, obtain *standard sums* by using Postulate 2(a) and Postulate 5(b).
- From *sum of products* to *product of sums* and vice versa, use Postulate 4.

Other Logic Operations



- $F_0 = 0$ Null, binary constant 0
- $F_1 = xy = x \cdot y$ AND, x and y
- $F_2 = xy' = x / y$ Inhibition, x but not y
- $F_3 = x$ Transfer, x
- $F_4 = xy' + x'y = x \oplus y$ Exclusive-OR, x or y but not both
- $F_5 = x + y$ OR, x or y
- $F_6 = (x + y)' = x \downarrow y$ NOR, Not-OR
- $F_7 = xy + x'y' = x \odot y$ Equivalence, x equals y
- $F_8 = x'$ Complement, Not x

Other Logic Operations



- $F_9 = x' + y = x \supset y$

Implication, if x then y

- $F_{10} = (xy)' = x \uparrow y$

NAND, Not-AND

- $F_{11} = 1$

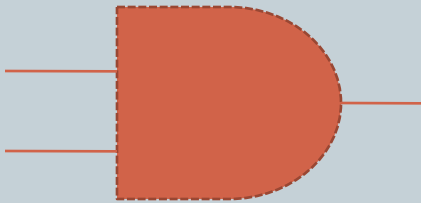
Identity, *binary constant 1*

Truth Table for Various Logic Operations



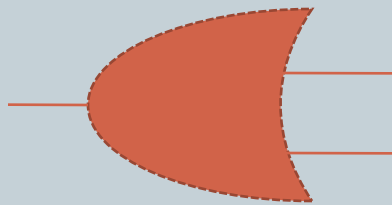
x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Symbol			\cdot	$/$		$/$		\oplus	$+$	\downarrow	\odot	$'$	\subset	$'$	\supset	\uparrow	

AND



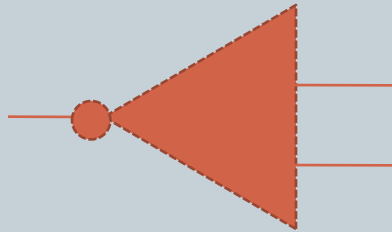
x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

OR



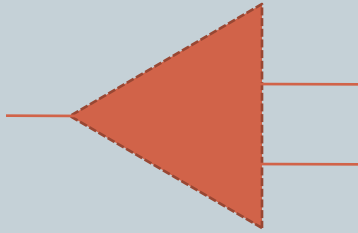
x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

INVERTER



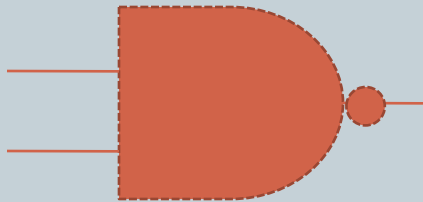
x	F
0	1
1	0

BUFFER



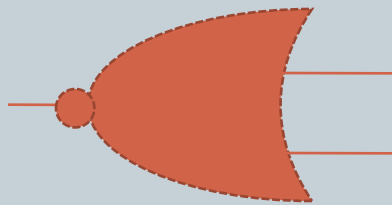
x	F
0	0
1	1

NAND



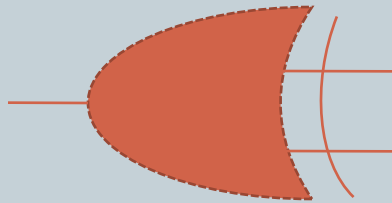
<i>x</i>	<i>y</i>	F
0	0	1
0	1	1
1	0	1
1	1	0

NOR



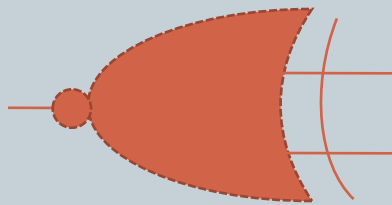
x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

XOR



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

XNOR



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

End of Chapter 2



PREPARE FOR A LONG TEST.