

CHAPTER-2

Boolean Algebra and Logic Gates

Digital Design (with an introduction to the Verilog HDL) 6th Edition,
M. Morris Mano, Michael D. Ciletti



INDIAN INSTITUTE OF INFORMATION TECHNOLOGY,
DESIGN AND MANUFACTURING,
KANCHEEPURAM

- Dr. Kalpana Settu
Assistant Professor
ECE, IIITDM Kancheepuram

Binary Logic and Gates

- ❑ Binary variables take on one of two values.
- ❑ Logical operators operate on binary values and binary variables.
- ❑ Basic logical operators are the logic functions AND, OR and NOT.
- ❑ Logic gates implement logic functions.
- ❑ Boolean Algebra: a useful mathematical system for specifying and transforming logic functions.
- ❑ We study Boolean algebra as a foundation for designing and analyzing digital systems!

Binary Logic and Gates

- ❑ Recall that the two binary values have different names:
 - True/False
 - On/Off
 - Yes/No
 - 1/0
- ❑ We use 1 and 0 to denote the two values.

Binary Logic and Gates

- ❑ The three basic logical operations are:
 - AND
 - OR
 - NOT
- ❑ AND is denoted by a dot (\cdot).
- ❑ OR is denoted by a plus ($+$).
- ❑ NOT is denoted by an overbar ($\bar{}$), a single quote mark ($'$) after, or tilde (\sim) before the variable.

Chapter-2

- ☐ **Basic Definitions**
- ☐ **Axiomatic Definition of Boolean Algebra**
- ☐ **Basic Theorems and Properties of Boolean Algebra**
- ☐ **Boolean Functions**
- ☐ **Canonical and Standard Forms**
- ☐ **Other Logic Operations**
- ☐ **Digital Logic Gates**

Basic Definitions

Boolean algebra:

- A deductive mathematical system 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, usually having a common property
Ex: $B = \{0, 1\}$
 - A set of operators
Ex: $+, *, \dots$

consider the relation $a * b = c$. We say that $*$ is a binary operator if it specifies a rule for finding c from the pair (a, b) and also if $a, b, c \in S$. However, $*$ is not a binary operator if $a, b \in S$, and if $c \notin S$.

Most Common Postulates

□ The most common postulates used to formulate various algebraic structures are as follows:

1. **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 .

Ex: natural number $N = \{1, 2, 3, \dots\}$ is **closed** with respect to “+”

$$1 + 2 = 3: 1, 2 \in N; 3 \in N$$

natural number $N = \{1, 2, 3, \dots\}$ is **not closed** with respect to “–”

$$1 - 2 = -1: 1, 2 \in N; -1 \notin N$$

2. **Associative law.** A binary operator $*$ on a set S is said to be associative whenever $(x * y) * z = x * (y * z)$ for all $x, y, z, \in S$

Most Common Postulates

3. *Commutative law.* A binary operator $*$ on a set S is said to be Commutative whenever $x * y = y * x$ for all $x, y \in S$

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

Ex: The element 0 is an identity element with respect to the binary operator $+$ on the set of integers $I = \{ \dots, -3, -2, -1, 0, 1, 2, 3, \dots \}$, since

$$x + 0 = 0 + x = x \text{ for any } x \in I$$

Most Common Postulates

5. 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

$$x * y = e$$

Ex: In the set of integers, I , and the operator $+$, with $e = 0$, the inverse of an element a is $(-a)$, since $a + (-a) = 0$.

6. Distributive law. If $*$ and \bullet are two binary operators on a set S , $*$ is said to be distributive over \bullet whenever

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

Most Common Postulates

- ❑ The field of real numbers is the basis for arithmetic and ordinary algebra. The operators and postulates have the following meanings:
 - The binary operator $+$ defines addition.
 - The additive identity is 0.
 - The additive inverse defines subtraction.
 - The binary operator \cdot defines multiplication.
 - The multiplicative identity is 1.
 - For $a \neq 0$, the multiplicative inverse of $a = 1/a$ defines division (i.e., $a \cdot 1/a = 1$).
 - The only distributive law applicable is that of \cdot over $+$: $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$

History of Boolean Algebra

- ❑ 1854: George Boole developed an algebraic system now called *Boolean algebra*.
- ❑ 1904: E. V. Huntington formulated a set of postulates that formally define the Boolean algebra
- ❑ 1938: C. E. Shannon introduced a two-valued Boolean algebra called switching algebra that represented the properties of bistable electrical switching circuits



George Boole



Edward Vermilye Huntington



Claude Elwood Shannon

Axiomatic Definition of Boolean Algebra

Boolean algebra is an algebraic structure defined by a set of elements, B , together with two binary operators, $+$ and \cdot , provided that the following (Huntington) postulates are satisfied:

1. (a) The structure is closed with respect to the operator $+$.
(b) The structure is closed with respect to the operator \cdot .
2. (a) The element 0 is an identity element with respect to $+$; that is, $x + 0 = 0 + x = x$.
(b) The element 1 is an identity element with respect to \cdot ; that is, $x \cdot 1 = 1 \cdot x = x$.
3. (a) The structure is commutative with respect to $+$; that is, $x + y = y + x$.
(b) The structure is commutative with respect to \cdot ; that is, $x \cdot y = y \cdot x$.
4. (a) The operator \cdot is distributive over $+$; that is, $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.
(b) The operator $+$ is distributive over \cdot ; that is, $x + (y \cdot z) = (x + y) \cdot (x + z)$.
5. For every element $x \in B$, there exists an element $x' \in B$ (called the complement of x) such that (a) $x + x' = 1$ and (b) $x \cdot x' = 0$.
6. There exist at least two elements $x, y \in B$ such that $x \neq y$.

Boolean algebra vs. Normal algebra

Boolean algebra

- Associate law not included (but still valid)
- Distributive law of $+$ over \cdot is valid
 $+$ over \cdot (i.e., $x + (y \cdot z) = (x + y) \cdot (x + z)$)
- No additive or multiplicative inverses
- Postulate 5 defines complement
- No. of elements is not clearly defined
 - 2 for two-valued Boolean algebra

Ordinary Algebra

- Associate law included
- Distributive law of $+$ over \cdot not valid
- Have additive and multiplicative inverses
- No complement operator
- Deal with real numbers
 - Infinite set of elements



Boolean Algebra

- In Boolean algebra, one defines the elements of the set B , and variables such as x , y , and z are merely symbols that *represent* the elements.
- In order to have a Boolean algebra, one must show that
 1. the elements of the set B ,
 2. the rules of operation for the two binary operators, and
 3. the set of elements, B , together with the two operators, satisfy the six Huntington postulates.

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 .

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

x	x'
0	1
1	0

The Huntington postulates are valid for the set $B = \{0, 1\}$ and the two binary operators $+$ and \cdot .

- That the structure is *closed* with respect to the two operators is obvious from the tables, since the result of each operation is either 1 or 0 and $1, 0 \in B$.

Two-Valued Boolean Algebra

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

x	x'
0	1
1	0

2. From the tables, we see that

(a) $0 + 0 = 0$ $0 + 1 = 1 + 0 = 1$;

(b) $1 \cdot 1 = 1$ $1 \cdot 0 = 0 \cdot 1 = 0$.

This establishes the two *identity elements*, 0 for $+$ and 1 for \cdot , as defined by postulate 2.

3. The *commutative* laws are obvious from the symmetry of the binary operator tables.

Two-Valued Boolean Algebra

4. (a) The *distributive* law $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$ can be shown to hold from the operator tables by forming a truth table of all possible values of x, y , and z . For each combination, we derive $x \cdot (y + z)$ and show that the value is the same as the value of $(x \cdot y) + (x \cdot z)$:

x	y	z	$y + z$	$x \cdot (y + z)$	$x \cdot y$	$x \cdot z$	$(x \cdot y) + (x \cdot z)$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

- (b) The *distributive* law of $+$ over \cdot can be shown to hold by means of a truth table similar to the one in part (a).



Two-Valued Boolean Algebra

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

x	x'
0	1
1	0

5. From the complement table, it is easily shown that

(a) $x + x' = 1$, since $0 + 0' = 0 + 1 = 1$ and $1 + 1' = 1 + 0 = 1$.

(b) $x \cdot x' = 0$, since $0 \cdot 0' = 0 \cdot 1 = 0$ and $1 \cdot 1' = 1 \cdot 0 = 0$.

6. Postulate 6 is satisfied because the two-valued Boolean algebra has two elements, 1 and 0, with $1 \neq 0$.

➤ Binary logic is a two-valued Boolean algebra

➤ also called “switching algebra” by engineers

Basic Theorems and Properties of Boolean Algebra

•Duality

- Every Boolean algebraic expression remains valid if the operators and identity elements are interchanged
 - Part (a) and part (b) are dual in the Huntington postulates
 - For two-valued Boolean algebra:
 - Interchange the binary operators: $\text{AND} \Leftrightarrow \text{OR}$
 - Interchange the identity elements: $1 \Leftrightarrow 0$
- Ex: $X + 1 = 1 \rightarrow X \cdot 0 = 0$

Basic Theorems and Properties of Boolean Algebra

Table 2.1

Postulates and Theorems of Boolean Algebra

Postulate 2	(a)	$x + 0 = x$	(b)	$x \cdot 1 = x$
Postulate 5	(a)	$x + x' = 1$	(b)	$x \cdot x' = 0$
Theorem 1	(a)	$x + x = x$	(b)	$x \cdot x = x$
Theorem 2	(a)	$x + 1 = 1$	(b)	$x \cdot 0 = 0$
Theorem 3, involution		$(x')' = x$		
Postulate 3, commutative	(a)	$x + y = y + x$	(b)	$xy = yx$
Theorem 4, associative	(a)	$x + (y + z) = (x + y) + z$	(b)	$x(yz) = (xy)z$
Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$
Theorem 6, absorption	(a)	$x + xy = x$	(b)	$x(x + y) = x$

Basic Theorems and Properties of Boolean Algebra

THEOREM 1(a): $x + x = x$.

Statement	Justification
$x + x = (x + x) \cdot 1$	postulate 2(b)
$= (x + x)(x + x')$	5(a)
$= x + xx'$	4(b)
$= x + 0$	5(b)
$= x$	2(a)

THEOREM 1(b): $x \cdot x = x$.

Statement	Justification
$x \cdot x = xx + 0$	postulate 2(a)
$= xx + xx'$	5(b)
$= x(x + x')$	4(a)
$= x \cdot 1$	5(a)
$= x$	2(b)



Basic Theorems and Properties of Boolean Algebra

THEOREM 2(a): $x + 1 = 1$.

Statement	Justification
$x + 1 = 1 \cdot (x + 1)$	postulate 2(b)
$= (x + x')(x + 1)$	5(a)
$= x + x' \cdot 1$	4(b)
$= x + x'$	2(b)
$= 1$	5(a)

THEOREM 2(b): $x \cdot 0 = 0$ by duality.

THEOREM 3: $(x')' = x$. From postulate 5, we have $x + x' = 1$ and $x \cdot x' = 0$, which together define the complement of x . The complement of x' is x and is also $(x')'$.

Basic Theorems and Properties of Boolean Algebra

THEOREM 6(a): $x + xy = x$.

By means of truth table

Statement	Justification
$x + xy = x \cdot 1 + xy$	postulate 2(b)
$= x(1 + y)$	4(a)
$= x(y + 1)$	3(a)
$= x \cdot 1$	2(a)
$= x$	2(b)

x	y	xy	x + xy
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

THEOREM 6(b): $x(x + y) = x$ by duality.

Basic Theorems and Properties of Boolean Algebra

- The algebraic proofs of the associative law and DeMorgan's theorem are long and will not be shown here. However, their validity is easily shown with truth tables.

□ DeMorgan's Theorems

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

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

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

Basic Theorems and Properties of Boolean Algebra

□ DeMorgan's Theorems

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

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

x	y	xy	$(xy)'$	x'	y'	$x'+y'$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

Basic Theorems and Properties of Boolean Algebra

□ Associative law

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

b. $x (y z) = (x y)z$

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

Try to check their validity with truth tables!!!!

Basic Theorems and Properties of Boolean Algebra

Operator Precedence

1) Parentheses

$$(\dots) \bullet (\dots)$$

2) NOT

$$x' + y$$

3) AND

$$x + x \bullet y$$

4) OR

$$x + x \bullet y$$

$$x [y + z \overline{(w + x)}]$$

$$(w + x)$$

$$\overline{(w + x)}$$

$$z \overline{(w + x)}$$

$$y + z \overline{(w + x)}$$

$$x [y + z \overline{(w + x)}]$$

Boolean Functions

- ❑ A Boolean function is described by an algebraic expression that consists of
 - Binary variables
 - The constant 0 and 1
 - The logic operation symbols
- ❑ For a given value of the binary variables, the function can be equal to either 1 or 0

Ex: $F_1 = x + y'z$

- $F_1 = 1$, if $x = 1$
- $F_1 = 1$, if $y = 0$ and $z = 1$
- $F_1 = 0$, otherwise

Truth Tables

- ❑ A Boolean function can be represented in a **truth table**.
 - An unique representation
- ❑ A truth table includes
 - A list of combinations of 1's and 0's assigned to the binary variables
 - A column that shows the value of the function for each binary combination
- ❑ Ex: $F_1 = x + y'z$
 - No. of binary variables = 3
 - No. of rows = $2^3 = 8$

$2^3 = 8$ rows

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

- ❑ A Boolean function can be transformed from an algebraic expression into a circuit diagram composed of logic gates connected in a particular structure.

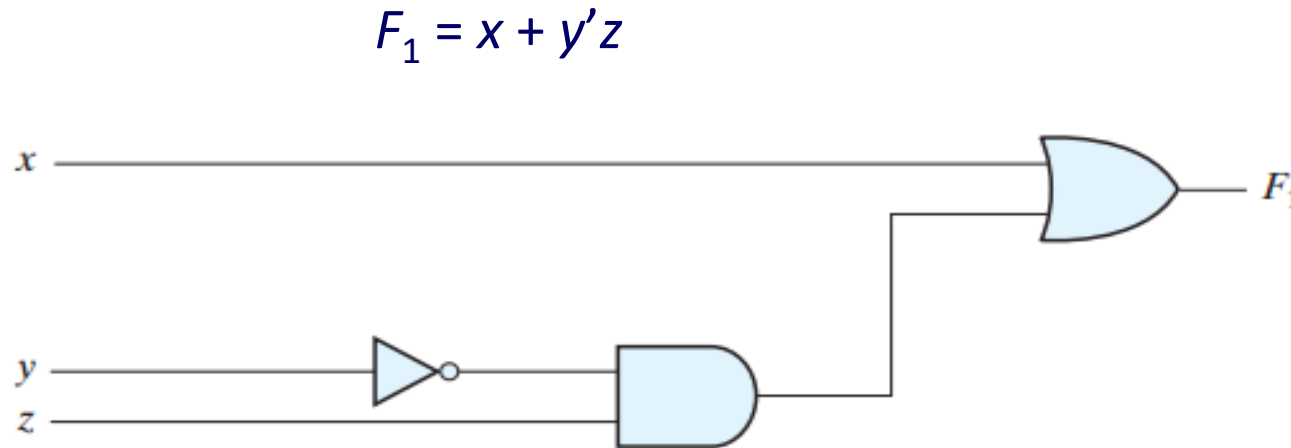


FIGURE 2.1

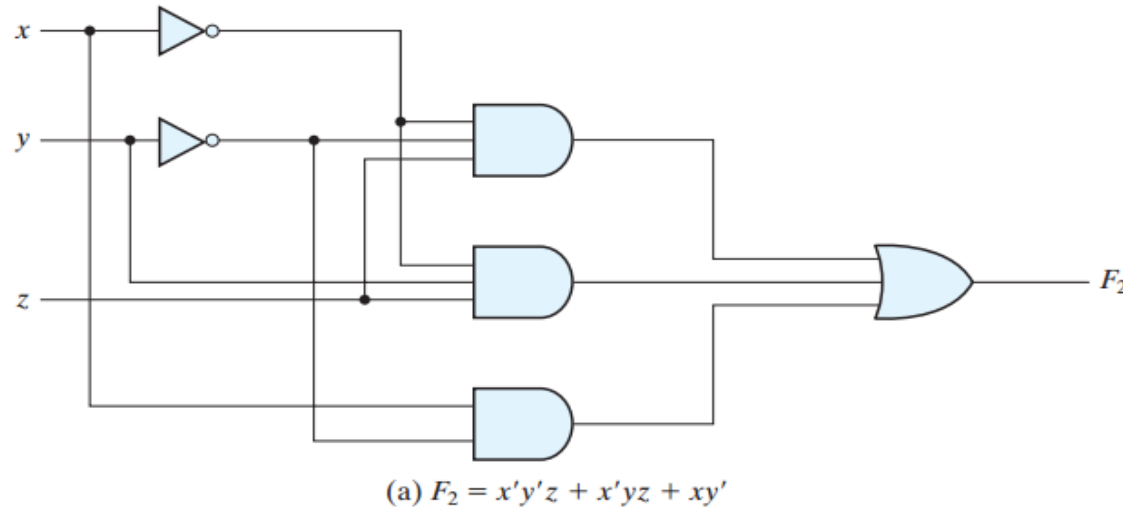
Gate implementation of $F_1 = x + y'z$

Gate Implementation

- ❑ There is only one way that a Boolean function can be represented in a truth table.
- ❑ However, when the function is in algebraic form, it can be expressed in a variety of ways, all of which have equivalent logic.
- ❑ The forms of the algebraic expressions have large impact on the logic circuit diagram
 - Not an unique representation
- ❑ Designers are motivated to reduce the complexity and number of gates (why?)

Gate Implementation

➤ **Example** $F_2 = x'y'z + x'yz + xy'$



➤ **Example: Complexity reduction**

$$F_2 = x'y'z + x'yz + xy' = x'z(y' + y) + xy' = x'z + xy'$$

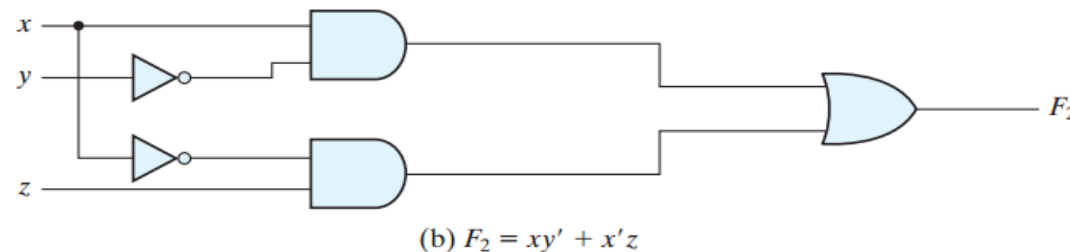


FIGURE 2.2
Implementation of Boolean function F_2 with gates

Algebraic Manipulation

- ❑ The manipulation of Boolean algebra consists mostly of reducing an expression for the purpose of obtaining a simpler circuit.
- ❑ **Literal:** a single variable within a **term** that may be complement or not

$$F_2 = x'y'z + x'yz + xy' \rightarrow 3 \text{ terms, } 8 \text{ literals}$$
$$= x'z(y' + y) + xy'$$

$$F_2 = x'z + xy' \rightarrow 2 \text{ terms, } 4 \text{ literals}$$

- ❑ Reducing the number of terms and the number of literals can often lead to a simpler circuit
- ❑ Simplify methods:
 - By map methods described in Chapter 3 (up to 5 variables)
 - By computer minimization programs
 - By a cut-and-try procedure employing the algebraic manipulation techniques (the only manual method)

Algebraic Manipulation

EXAMPLE 2.1

Simplify the following Boolean functions to a minimum number of literals.

1. $x(x' + y) = xx' + xy = 0 + xy = xy.$
 2. $x + x'y = (x + x')(x + y) = 1(x + y) = x + y.$
 3. $(x + y)(x + y') = x + xy + xy' + yy' = x(1 + y + y') = x.$
- } Dual of each other

An easier way to simplify function 3 is by means of postulate 4(b) from Table 2.1 : $(x + y)(x + y') = x + yy' = x.$

Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$
---------------------------	-----	----------------------	-----	---------------------------

Consensus theorem

$$\begin{aligned} \text{4. } \underline{xy + x'z + yz} &= xy + x'z + yz(x + x') \\ &= xy + x'z + xyz + x'yz \\ &= xy(1 + z) + x'z(1 + y) \\ &= \underline{xy + x'z}. \end{aligned}$$

$$\text{5. } (x + y)(x' + z)(y + z) = (x + y)(x' + z), \text{ by duality from function 4.}$$

Consensus Theorem

- The two important statements of the **Consensus Theorem** are shown below where we can easily see the term **BC** in the first statement and the term **(B+C)** in the second statement doesn't appear on the Right-hand side or after simplification.
- The term **BC** and the term **(B+C)** are called consensus term or redundant term as their absence doesn't make any changes in the final output.

$AB + A'C + \underline{BC} = AB + A'C$
 $(A+B)(A'+C)(\underline{B+C}) = (A+B)(A'+C)$

Consensus term from A and A' $\longrightarrow (B'C')(C'D') = B'C'D'$
 Consensus term from B and B' $\longrightarrow D'(A C') = AC'D'$

Complement of a Function

- ❑ The complement of a function F is F' and is obtained from an interchange between 0 and 1.
- ❑ The complement of a function can be derived algebraically through **DeMorgan's theorems**.

Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$
---------------------	-----	-------------------	-----	-------------------

- ❑ DeMorgan's theorems can be extended to three or more variables.

$$\begin{aligned}(A + B + C)' &= (A + x)' && \text{let } B + C = x \\ &= A'x' && \text{by theorem 5(a) (DeMorgan)} \\ &= A'(B + C)' && \text{substitute } B + C = x \\ &= A'(B'C') && \text{by theorem 5(a) (DeMorgan)} \\ &= A'B'C' && \text{by theorem 4(b) (associative)}\end{aligned}$$

Complement of a Function

- ❑ DeMorgan's theorems can be extended to three or more variables.

$$(A + B + C + D + \dots + F)' = A'B'C'D' \dots F'$$

$$(ABCD \dots F)' = A' + B' + C' + D' + \dots + F'$$

EXAMPLE 2.2

Complement by DeMorgan's theorem

Find the complement of the functions $F_1 = x'yz' + x'y'z$ and $F_2 = x(y'z' + yz)$. By applying DeMorgan's theorems as many times as necessary, the complements are obtained as follows:

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

$$\begin{aligned} F_2' &= [x(y'z' + yz)]' = x' + (y'z' + yz)' = x' + (y'z')'(yz)' \\ &= x' + (y + z)(y' + z') \\ &= x' + yz' + y'z \end{aligned}$$

Theorem 5, DeMorgan

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

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

Complement of a Function

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

EXAMPLE 2.3

Complement by taking duals and complementing each literal

Find the complement of the functions F_1 and F_2 of Example 2.2 by taking their duals and complementing each literal.

1. $F_1 = x'yz' + x'y'z$.

The dual of F_1 is $(x' + y + z')(x' + y' + z)$.

Complement each literal: $(x + y' + z)(x + y + z') = F'_1$.

2. $F_2 = x(y'z' + yz)$.

The dual of F_2 is $x + (y' + z')(y + z)$.

Complement each literal: $x' + (y + z)(y' + z') = F'_2$.

Canonical & Standard Forms

Minterms and Maxterms

- ❑ Consider two binary variables x and y combined with an AND operation. There are four possible combinations: $x'y'$, $x'y$, xy' , and xy . Each AND term is called a **minterm**, or a **standard product**.
- ❑ n variables can be combined to form 2^n minterms. A symbol for each minterm is of the form m_j .
- ❑ Similarly, two binary variables x and y combined with an OR operation: $x+y$, $x+y'$, $x'+y$, $x'+y'$, each OR term is called a **maxterm**, or **standard sums**.
- ❑ n variables can be combined to form 2^n maxterms. A symbol for each maxterm is of the form M_j .

j denotes the decimal equivalent of the binary number



Minterms and Maxterms

- Each minterm is obtained from an AND term of the n variables, with each variable being **primed if the corresponding bit is a 0** and **unprimed if it is 1**.
- Each maxterm is obtained from an OR term of the n variables, with each variable being **unprimed if the corresponding bit is a 0** and **primed if it is 1**.

Table 2.3

Minterms and Maxterms for Three 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

Make minterm to be one

Make maxterm to be zero

Minterms and Maxterms

- A Boolean function can be expressed algebraically from a given truth table by forming a **minterm** for each combination of the variables that produces a **1** in the function and then taking the **OR** of all those terms.

Table 2.4

Functions of Three Variables

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Any Boolean function can be expressed as a sum of minterms (with “sum” meaning the ORing of terms).

$$f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$$

$$f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$$

Minterms and Maxterms

- ❑ A Boolean function can be expressed algebraically from a given truth table by forming a **maxterm** for each combination of the variables that produces a **0** in the function, and then form the **AND** of all those maxterms.

Table 2.4

Functions of Three Variables

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Any Boolean function can be expressed as a product of maxterms (with “product” meaning the ANDing of terms).

$$\begin{aligned}f_1 &= (x + y + z)(x + y' + z)(x + y' + z')(x' + y + z')(x' + y' + z) \\ &= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6\end{aligned}$$

$$\begin{aligned}f_2 &= (x + y + z)(x + y + z')(x + y' + z)(x' + y + z) \\ &= M_0 M_1 M_2 M_4\end{aligned}$$

Minterms and Maxterms

- ❑ Any Boolean function can be expressed as a “sum” of minterms, “sum” meaning the OR of terms.
- ❑ Any Boolean function can be expressed as a product of maxterms, product” meaning the AND of terms.
- ❑ Boolean functions expressed as a sum of minterms or product of maxterms are said to be in **canonical form**.

Sum of Minterms

- ❑ It is sometimes convenient to express a Boolean function in its sum-of-minterms form.
- ❑ If the function is not in this form, it can be made so by first expanding the expression into a sum of AND terms. Each term is then inspected to see if it contains all the variables. If it misses one or more variables, it is ANDed with an expression such as $x + x'$, where x is one of the missing variables.

EXAMPLE 2.4

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

$$\begin{aligned} A &= AB(C + C') + AB'(C + C') \\ &= ABC + ABC' + AB'C + AB'C' \end{aligned}$$

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

$$\begin{aligned} F &= A + B'C \\ &= ABC + ABC' + AB'C + AB'C' + A'B'C \end{aligned}$$

$$\begin{aligned} F &= A'B'C + AB'C' + AB'C + ABC' + ABC \\ &= m_1 + m_4 + m_5 + m_6 + m_7 \end{aligned}$$

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

The summation symbol Σ stands for the ORing of terms; the numbers following it are the indices of the minterms of the function.

Sum of Minterms

- ❑ An alternative procedure for deriving the minterms of a Boolean function is from the truth table.

Table 2.5

Truth Table for $F = A + B'C$

A	B	C	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

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

The minterms whose sum defines the Boolean function are those which give the 1's of the function in a truth table.

Product of Maxterms

- ❑ To express a Boolean function as a product of maxterms, it must first be brought into a form of OR terms. This may be done by using the distributive law, $x + yz = (x + y)(x + z)$. Then any missing variable x in each OR term is ORed with xx' .

EXAMPLE 2.5

Express the Boolean function $F = xy + x'z$ as a product of maxterms. First, convert the function into OR terms by using the distributive law:

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

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

$$\begin{aligned} x' + y &= x' + y + zz' = (x' + y + z)(x' + y + z') \\ x + z &= x + z + yy' = (x + y + z)(x + y' + z) \\ y + z &= y + z + xx' = (x + y + z)(x' + y + z) \end{aligned}$$

Product of Maxterms

Combine all the terms together and remove repeated terms

$$\begin{aligned} F &= (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z') \\ &= M_0 M_2 M_4 M_5 \end{aligned}$$

A convenient expression:

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

The product symbol Π stands for the ANDing of terms; the numbers following it are the indices of the maxterms of the function.

Conversion between Canonical Forms

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

From the truth table, we can get F :

$$F'(A, B, C) = \underline{\Sigma(0, 2, 3)} = m_0 + m_2 + m_3$$

Take complement of F :

$$F = (m_0 + m_2 + m_3)' = m'_0 \cdot m'_2 \cdot m'_3 = M_0 M_2 M_3 = \underline{\Pi(0, 2, 3)}$$

$$m'_j = M_j$$

- ❑ The maxterm with subscript j is a complement of the minterm with the same subscript j

Table 2.5

Truth Table for $F = A + B'C$

A	B	C	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

Conversion between Canonical Forms

- ❑ To convert from one canonical form to the other, interchange the symbols Σ and Π and list those numbers **missing** from the original form.

$$F = xy + x'z$$

Truth Table for $F = xy + x'z$

x	y	z	F
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	1

Minterms

$$F(x, y, z) = \Sigma(1, 3, 6, 7)$$

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

Maxterms

Standard SOP

➤ Standard SOP expressions are important in constructing truth tables, and in the Karnaugh map simplification method.

EXAMPLE 4-20

Develop a truth table for the standard SOP expression $\overline{A}\overline{B}C + A\overline{B}\overline{C} + ABC$.

Solution

There are three variables in the domain, so there are eight possible combinations of binary values of the variables as listed in the left three columns of Table 4-6. The binary values that make the product terms in the expressions equal to 1 are

TABLE 4-6				
Inputs			Output	
A	B	C	X	Product Term
0	0	0	0	
0	0	1	1	$\overline{A}\overline{B}C$
0	1	0	0	
0	1	1	0	
1	0	0	1	$A\overline{B}\overline{C}$
1	0	1	0	
1	1	0	0	
1	1	1	1	ABC

$\overline{A}\overline{B}C$: 001; $A\overline{B}\overline{C}$: 100; and ABC : 111. For each of these binary values, place a 1 in the output column as shown in the table. For each of the remaining binary combinations, place a 0 in the output column.



Standard Forms

- ❑ Another way to express Boolean functions is in **standard form**.
- ❑ In this configuration, the terms that form the function may contain any number of literals.
- ❑ Two types of standard forms: a sum of products and a products of sums.
- ❑ **Sum of products:** 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.

$$F_1 = y' + xy + x'yz'$$

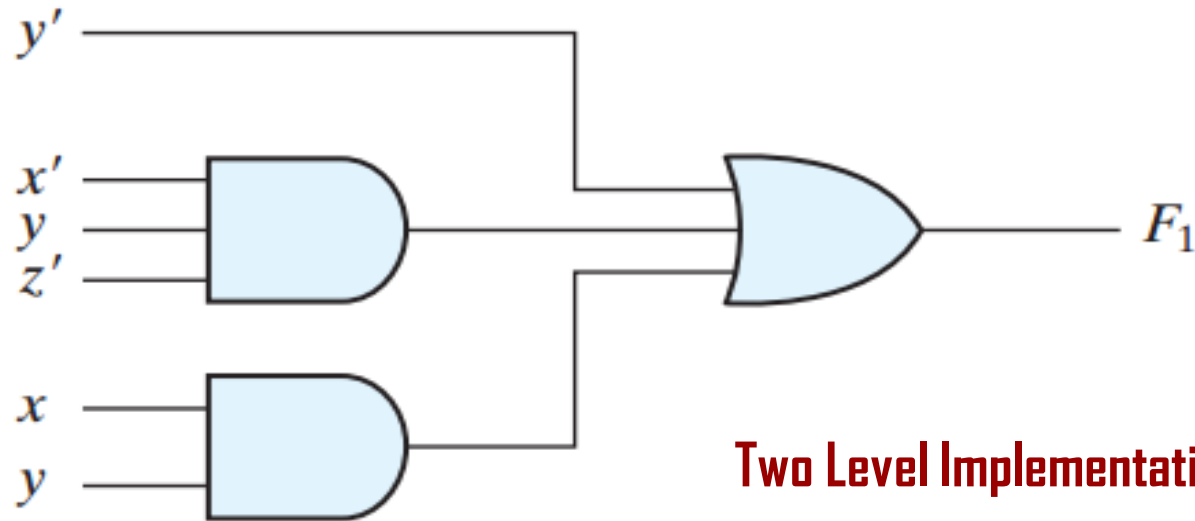
- ❑ **Product of sums:** A product of sums is a Boolean expression containing OR terms, called sum terms. Each term may have any number of literals. The product denotes the ANDing of these terms.

$$F_2 = x(y' + z)(x' + y + z')$$

Standard Forms

- ❑ A sum-of-products expression consists of a group of AND gates (for the product terms) followed by a single OR gate.

$$F_1 = y' + xy + x'yz'$$



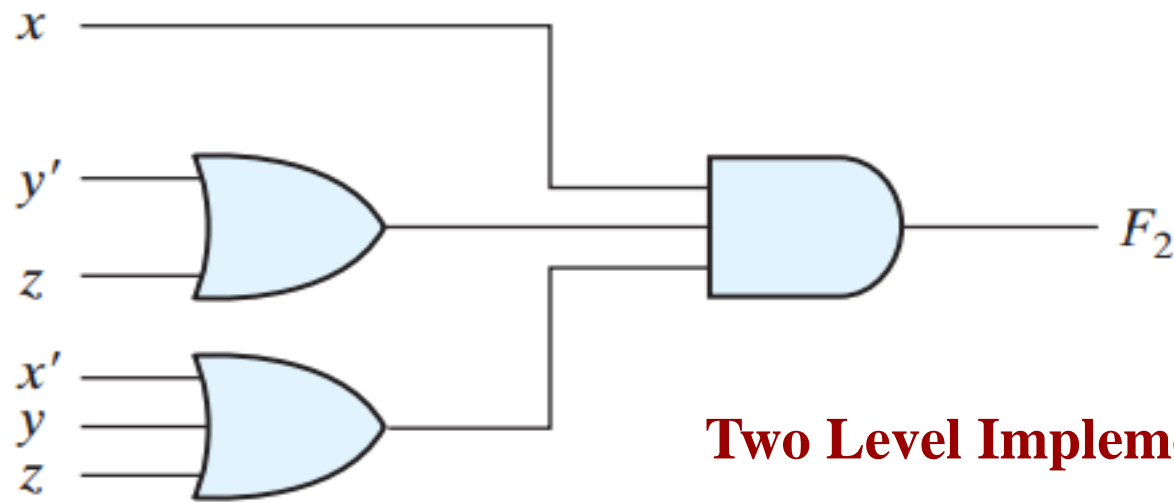
Two Level Implementation

(a) Sum of Products

Standard Forms

- ❑ A product-of-sums expression consists of a group of OR gates (for the sum terms) followed by an AND gate.

$$F_2 = x(y' + z)(x' + y + z')$$



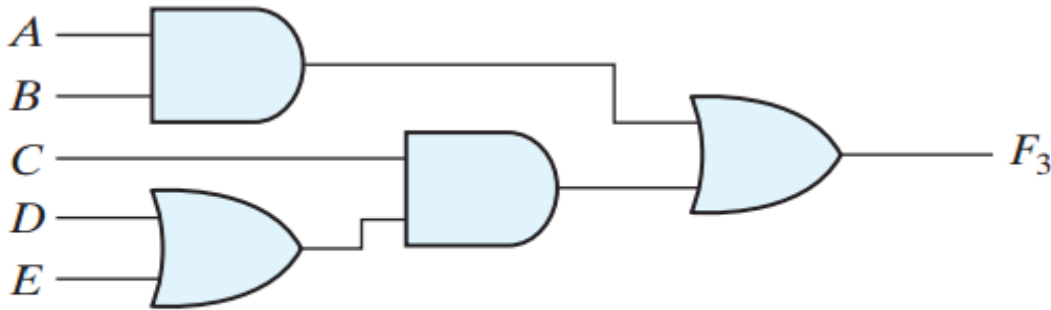
(b) Product of Sums

Standard type of expression results in a two-level structure of gates.

Standard Forms

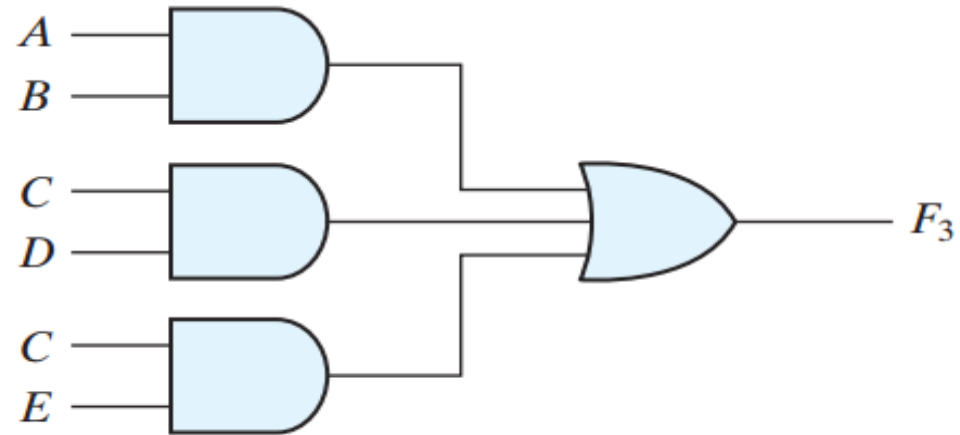
- ❑ If a Boolean function is expressed in a nonstandard form, it can be changed to a standard form.

$$F_3 = AB + C(D + E)$$



(a) $AB + C(D + E)$

$$\begin{aligned} F_3 &= AB + C(D + E) \\ &= AB + CD + CE \end{aligned}$$



(b) $AB + CD + CE$

Three Level Implementation

Two Level Implementation

In general, a two-level implementation is preferred because it produces the least amount of delay through the gates when the signal propagates from the inputs to the output.

Other Logic Operation

- ❑ There are 2^{2n} functions for n binary variables.
- ❑ 16 functions of two binary variables (2^{2n})

Table 2.7

Truth Tables for the 16 Functions of Two Binary Variables

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

- ❑ AND and OR functions are only 2 of a total of 16 possible functions formed with two binary variables

Table 2.7
Truth Tables for the 16 Functions of Two Binary Variables

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

Boolean Expressions for the 16 Functions of Two Variables

Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		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
$F_4 = x'y$	y/x	Inhibition	y , but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$ $x\hat{y}$	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 XNOR	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 1

- There are two constant functions : **Zero** and **One**. For every combination of variable values, the *Zero* function will return 0, whereas the *One* function will return to 1.
- There are four functions of one variable, which indicate **Complement** and **Transfer** operations. Specifically, the *Complement* function will produce the complement of one of the binary variables. The *Transfer* functions by contrast will reproduce one of the binary variables at the output.
- There are ten functions that define eight specific binary operations: **AND**, **Inhibition**, **XOR**, **OR**, **NOR**, **Equivalence**, **Implication**, and **NAND**.

Digital Logic Gates

Boolean Expressions for the 16 Functions of Two Variables





Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		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
$F_4 = x'y$	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 1

$F_0, F_{15} \rightarrow$ constants





Inhibition and implication \rightarrow not commutative or associative



Digital Logic Gates

Name	Graphic symbol	Algebraic function	Truth table															
AND		$F = x \cdot y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = x + y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	1
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
Inverter		$F = x'$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	F	0	1	1	0									
x	F																	
0	1																	
1	0																	
Buffer		$F = x$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	x	F	0	0	1	1									
x	F																	
0	0																	
1	1																	

Digital Logic Gates

Name	Graphic symbol	Algebraic function	Truth table															
NAND		$F = (xy)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = (x + y)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	0
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

Digital Logic Gates

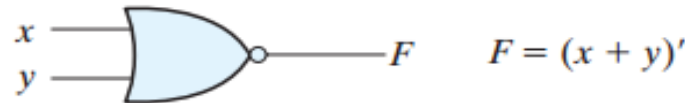
- ❑ NAND and NOR are two very important gates.

NAND



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

NOR

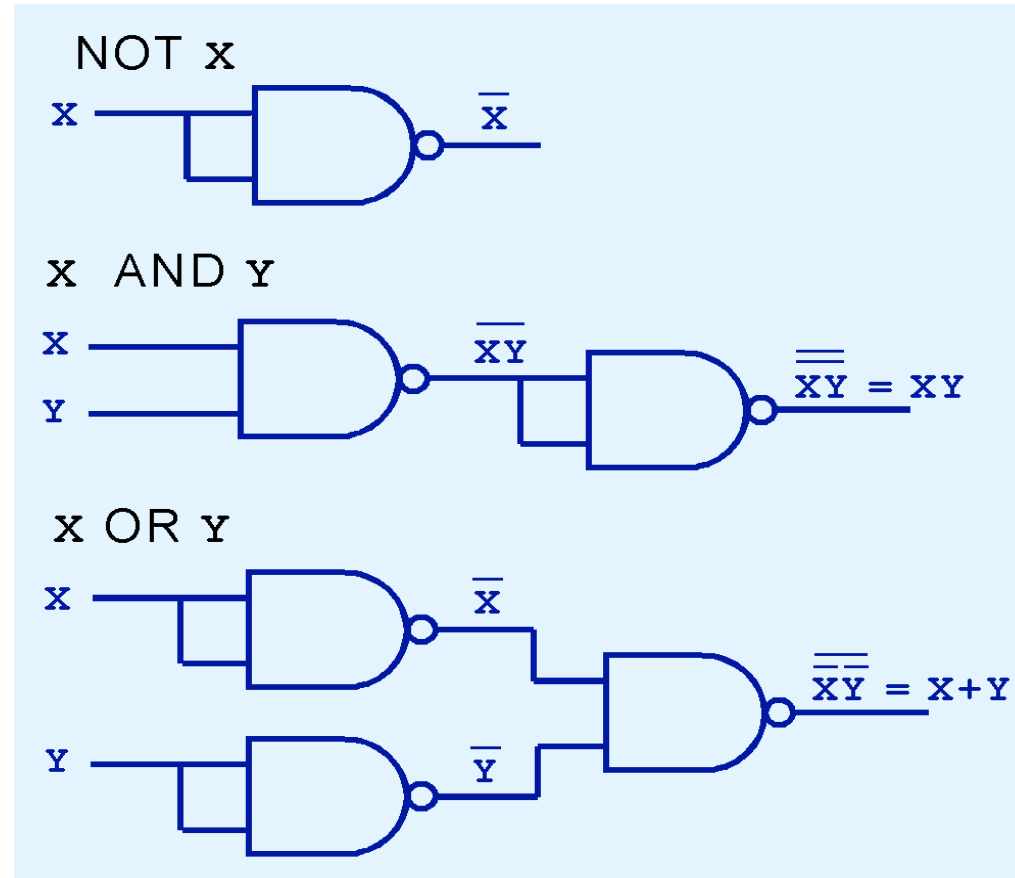


x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

- ❑ NAND and NOR gates are used extensively as standard logic gates and are in fact far more popular than the AND and OR gates. This is because NAND and NOR gates are easily constructed with transistor circuits and because digital circuits can be easily implemented with them.

Digital Logic Gates

- NAND and NOR are known as *universal gates* because they are inexpensive to manufacture and any Boolean function can be constructed using only NAND or only NOR gates.



Digital Logic Gates

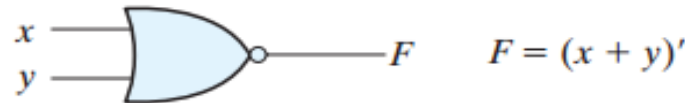
- ❑ NAND and NOR are two very important gates.

NAND



x	y	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

- ❑ NAND and NOR gates are used extensively as standard logic gates and are in fact far more popular than the AND and OR gates. This is because NAND and NOR gates are easily constructed with transistor circuits and because digital circuits can be easily implemented with them.

Extension to Multiple Inputs

- ❑ A gate (except for the inverter and buffer) can be extended to have multiple inputs if the binary operation it represents is commutative and associative.

- **AND and OR are commutative and associative**

$$\begin{aligned}x \cdot y &= y \cdot x \\ x + y &= y + x\end{aligned}\quad \text{commutative}$$

$$\begin{aligned}(x \cdot y) \cdot z &= x \cdot (y \cdot z) = x \cdot y \cdot z \\ (x + y) + z &= x + (y + z) = x + y + z\end{aligned}\quad \text{associative}$$

Which indicates that the gate inputs can be interchanged and that the AND and OR functions can be extended to three or more variables.

Extension to Multiple Inputs

NAND and NOR gates with Multiple Inputs

- **NAND and NOR are not associative**

$$(x \downarrow y) \downarrow z = [(x + y)' + z]' = (x + y)z' = xz' + yz'$$

$$x \downarrow (y \downarrow z) = [x + (y + z)']' = x'(y + z) = x'y + x'z$$

To overcome this difficulty, we define the multiple NOR (or NAND) gate as a complemented OR (or AND) gate.

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

$$x \uparrow y \uparrow z = (xyz)'$$

Extension to Multiple Inputs

NAND and NOR gates with Multiple Inputs

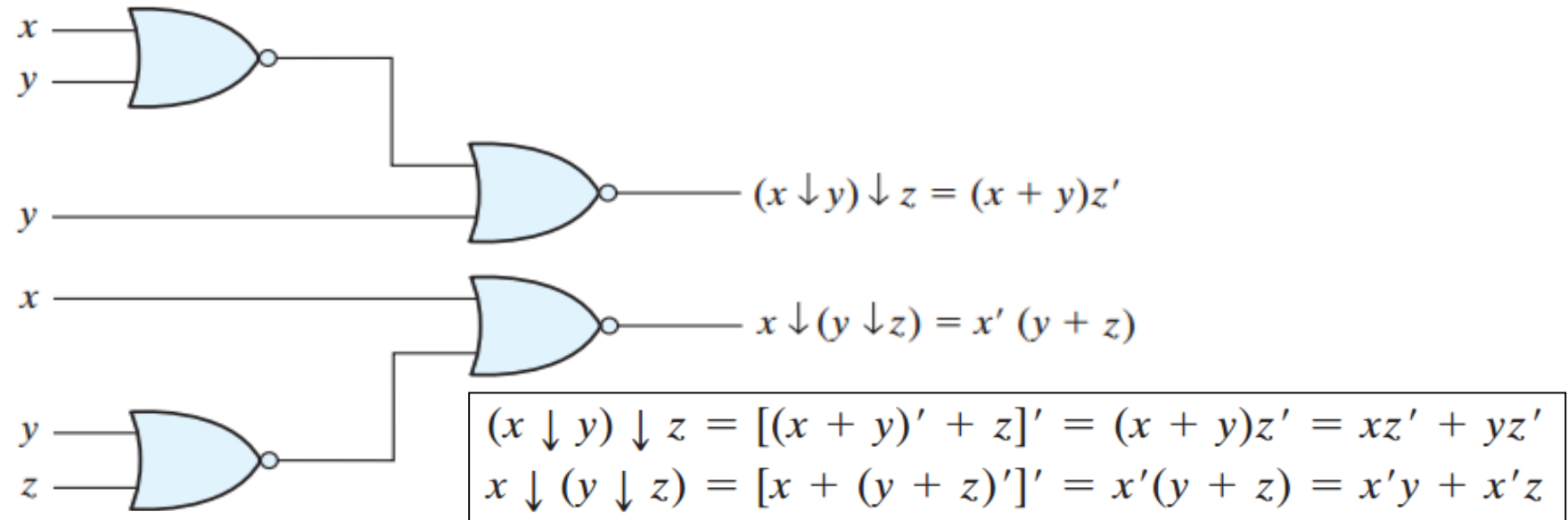
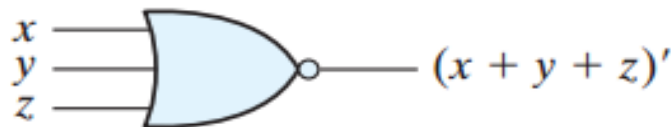
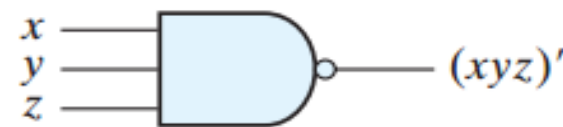


FIGURE 2.6

Demonstrating the nonassociativity of the NOR operator: $(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$



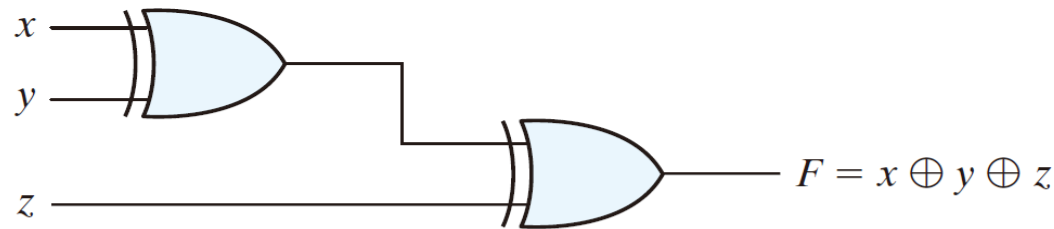
(a) 3-input NOR gate



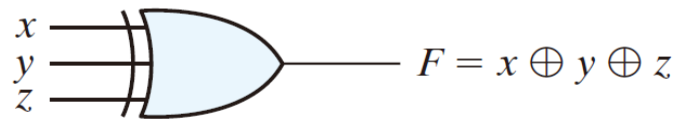
(b) 3-input NAND gate

Extension to Multiple Inputs

- The XOR and XNOR gates are **commutative** and **associative**
- Multiple-input XOR gates are uncommon?
- XOR is an odd function: **it is equal to 1** if the **inputs** variables have **an odd number of 1's**



(a) Using 2-input gates



(b) 3-input gate

x	y	z	F
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

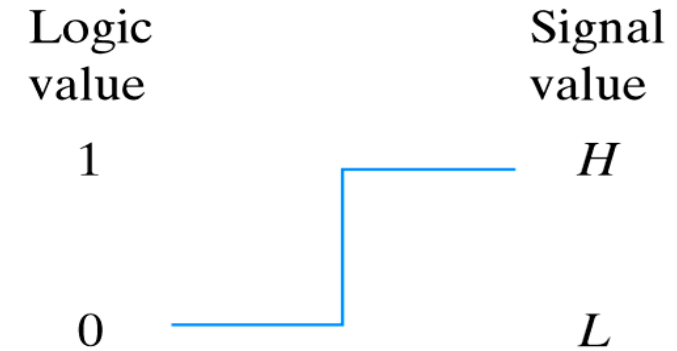
(c) Truth table

FIGURE 2.8

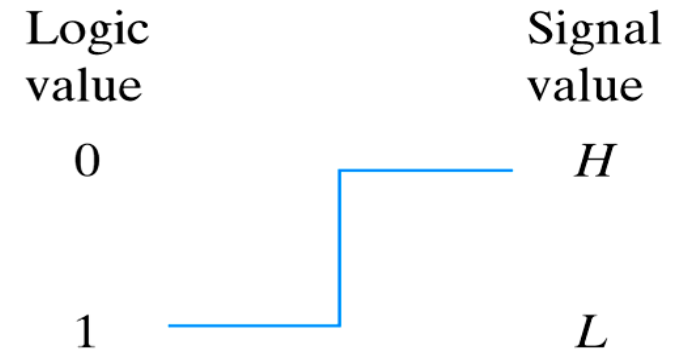
Three-input exclusive-OR gate

Positive and Negative Logic

- ❑ The binary signal at the inputs and outputs of any gate has one of two values, except during transition. One signal value represents logic 1 and the other logic 0.
- ❑ The higher signal level is designated by H and the lower signal level by L.
- ❑ Choosing the high-level H to represent logic 1 defines a **positive logic system**.
- ❑ Choosing the low-level L to represent logic 1 defines a **negative logic system**.



(a) Positive logic



(b) Negative logic

FIGURE 2.9

Signal assignment and logic polarity

Positive and Negative Logic

x	y	z
L	L	L
L	H	L
H	L	L
H	H	H

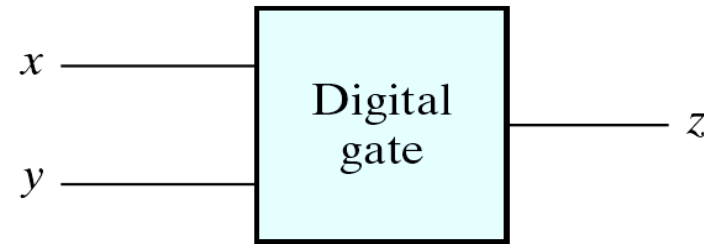
(a) Truth table with H and L

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

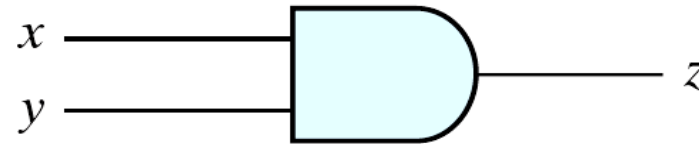
(c) Truth table for positive logic

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

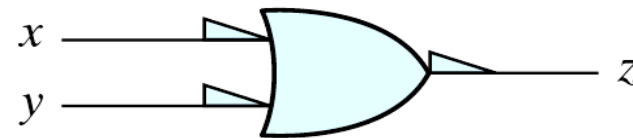
(e) Truth table for negative logic



(b) Gate block diagram



(d) Positive logic AND gate



(f) Negative logic OR gate

Programmable Logic

- A Programmable Logic Device (PLD) can be programmed to implement logic. There are various technologies available for PLDs. Many use an internal array of AND gates to form logic terms. Many PLDs can be programmed multiple times.
- Two major categories of user-programmable logic are **PLD** (programmable logic device) and **FPGA** (field-programmable gate array).
- In general, the required logic for a PLD is developed with the aid of a computer. The logic can be entered using a Hardware Description Language (HDL) such as VHDL.

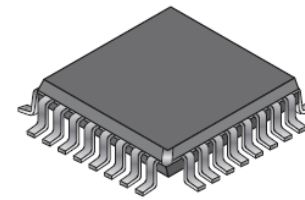
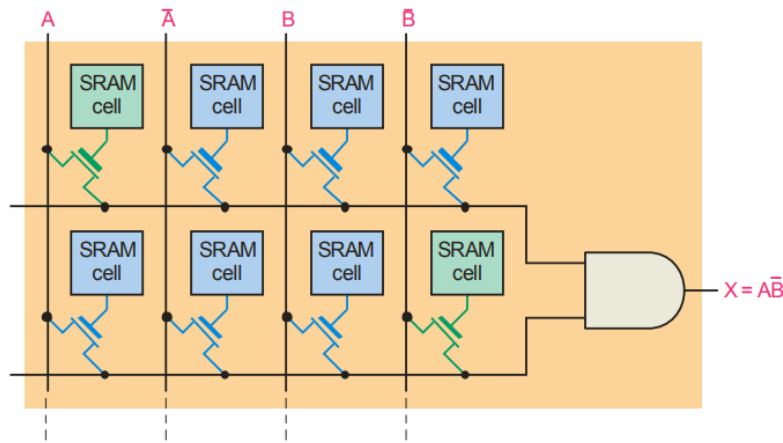


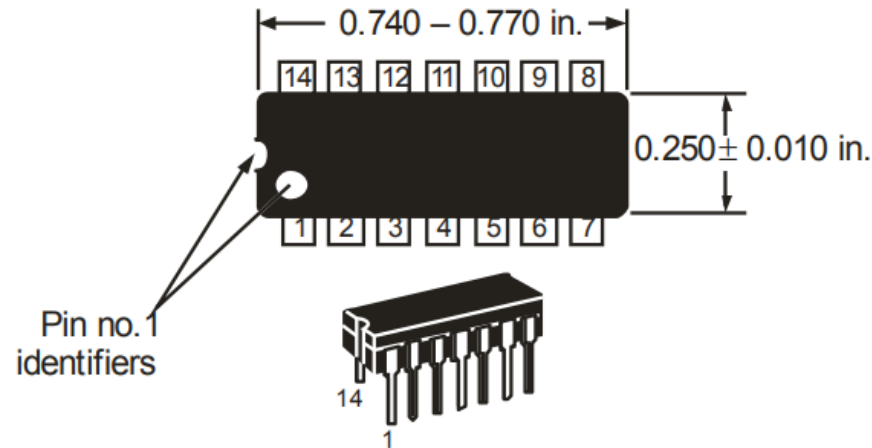
FIGURE 1-31 A typical SPLD package.



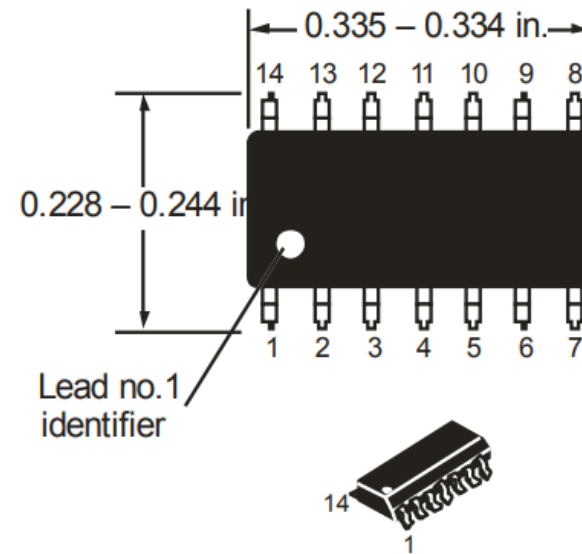
FIGURE 1-36 Basic setup for programming a PLD or FPGA. Graphic entry of a logic circuit is shown for illustration. Text entry such as VHDL can also be used. (Photo courtesy of Digilent, Inc.)

Fixed-Function Logic

- Fixed-function logic integrated circuits have been around for a long time and are available in a variety of logic functions. Unlike a PLD, a fixed-function IC comes with logic functions that cannot be programmed in and cannot be altered.



DIP package
(dual in-line package)



SOIC package
(small-outline integrated circuit)

Fixed-Function Logic

- Major fixed function logic families are **TTL** transistor–transistor logic, **ECL** emitter-coupled logic, **MOS** metal-oxide semiconductor, **CMOS** complementary metal-oxide semiconductor
- **TTL** is a logic family that has been in use for 50 years and is considered to be standard.
- **ECL** has an advantage in systems requiring high-speed operation.
- **MOS** is suitable for circuits that need high component density.
- **CMOS** is preferable in systems requiring low power consumption, such as digital cameras, personal media players, and other handheld portable devices.
- **CMOS** has become the dominant logic family, while TTL and ECL continue to decline in use.

Fixed-Function Logic

The most important parameters distinguishing logic families are listed below;

- **Fan-out** specifies the number of standard loads that the output of a typical gate can drive without impairing its normal operation. A standard load is usually defined as the amount of current needed by an input of another similar gate in the same family.
- **Fan-in** is the number of inputs available in a gate.
- **Power dissipation** is the power consumed by the gate that must be available from the power supply.
- **Propagation delay** is the average transition delay time for a signal to propagate from input to output. For example, if the input of an inverter switches from 0 to 1, the output will switch from 1 to 0, but after a time determined by the propagation delay of the device. The operating speed is inversely proportional to the propagation delay.
- **Noise margin** is the maximum external noise voltage added to an input signal that does not cause an undesirable change in the circuit output.

The End

Reference:

1. Digital Design (with an introduction to the Verilog HDL) 6th Edition, M. Morris Mano, Michael D. Ciletti

Note: The slides are supporting materials for the course “Digital Circuits” at IIITDM Kancheepuram. Distribution without permission is prohibited.

