Chapter 2: Boolean Algebra and Logic Gates

2.1 Introduction

2.2 Basic Definitions

2.3 Axiomatic Definition of Boolean Algebra

2.4 Basic Theorems and Properties of Boolean Algebra

2.5 Boolean Functions

2.6 Canonical and Standard Forms

2.7 Other Logic Operations

2.8 Digital Logic Gates

2.9 Integrated Circuits

Basic Definitions

- Mathematical methods that simplify binary logics or circuits rely primarily on Boolean algebra.
- Boolean algebra: 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. A = {1, 2, 3, 4} indicates that set A has the elements of 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*.
- 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*.

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

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 with respect to a binary operation * on S if there exists an element $e \in S$ with the property that

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

Example: The element 0 is an identity element with respect to the binary operator + on the set of integers $I = \{c, -3, -2, -1, 0, 1, 2, 3, c\}$, since $x + 0 = 0 + x = x$ for any $x \in I$

The set of natural numbers, N, has no identity element, since 0 is excluded from the set.

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 $x * y = e$

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

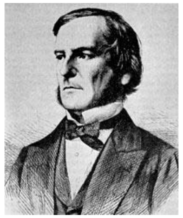
Field

- A *field* is an example of an algebraic structure.
- The field of real numbers is the basis for arithmetic and ordinary algebra.
  - The binary operator + defines addition.
  - The additive identity is 0.
  - The additive inverse defines subtraction.
  - The binary operator • defines multiplication.
  - The multiplicative identity is 1.
  - For $a \neq 0$, the multiplicative inverse of $a = 1/a$ defines division (i.e., $a \bullet 1/a = 1$).
  - The only distributive law applicable is that of • over +:

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

Axiomatic Definition 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



- Two binary operators, + and •, (Huntington) postulates:

1. (a) The structure is closed with respect to the operator +.
   (b) The structure is closed with respect to the operator •.
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 •; that is, $x \bullet 1 = 1 \bullet x = x$.
3. (a) The structure is commutative with respect to +; that is, $x + y = y + x$.
   (b) The structure is commutative with respect to • ; that is, $x \bullet y = y \bullet x$.
4. (a) The operator • is distributive over +; that is, $x \bullet (y + z) = (x \bullet y) + (x \bullet z)$.
   (b) The operator + is distributive over •; that is, $x + (y \bullet z) = (x + y) \bullet (x + z)$.

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) $x \bullet x = 0$.
6. There exist at least two elements $x, y \in B$ *such that* $x \neq y$.

- Comparing Boolean algebra with arithmetic and ordinary algebra

1. Huntington postulates do not include the associative law. However, this law holds for Boolean algebra and can be derived (for both operators) from the other postulates.
2. The distributive law of + over • (i.e., $x + (y \bullet z) = (x + y) \bullet (x + z)$) is valid for Boolean algebra, but not for ordinary algebra.
3. Boolean algebra does not have additive or multiplicative inverses; therefore, there are no subtraction or division operations.
4. Postulate 5 defines an operator called the complement that is not available in ordinary algebra.
5. Ordinary algebra deals with the real numbers, which constitute an infinite set of elements. Boolean algebra deals with the as yet undefined set of elements, B, but in the two-valued Boolean algebra defined next (and of interest in our subsequent use of that algebra), B is defined as a set with only two elements, 0 and 1.

Two-valued Boolean Algebra

- B = {0,1}
- The rules of operations

AND

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

OR

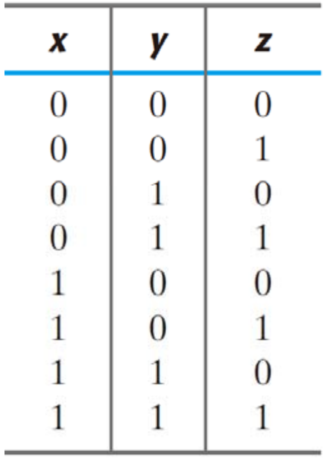
x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

NOT

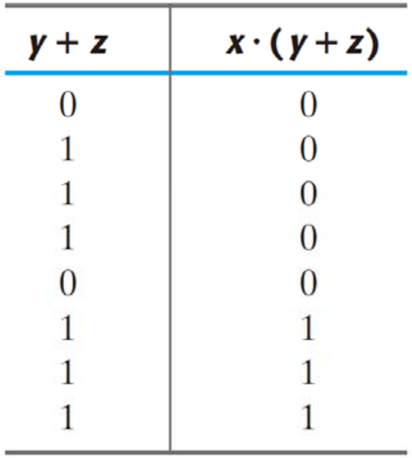
x	x'
0	1
1	0

- Closure: the result of each operation is either 1 or 0 and 1, 0 $\in$ *B*.
- Identity elements: 0 for + and 1 for $\cdot$
- The commutative laws are obvious from the symmetry of the binary operator tables.

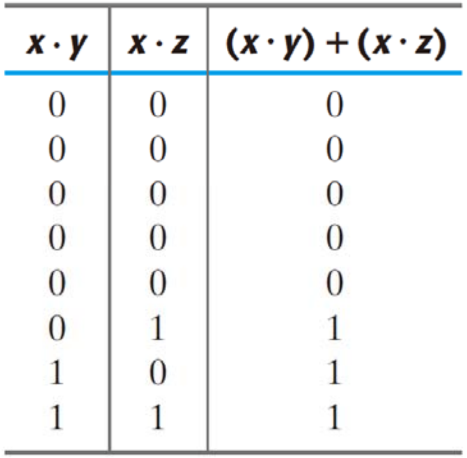
- Distributive laws:
  - $x \bullet (y + z) = (x \bullet y) + (x \bullet z)$
  - $x + (y \bullet z) = (x + y) \bullet (x + z)$



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



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



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

y • z	x+(y • z)
0	0
0	0
0	0
1	1
0	1
0	1
0	1
1	1

x+y	x+z	(x+y) • (x+z)
0	0	0
0	1	0
1	0	0
1	1	1
1	1	1
1	1	1
1	1	1
1	1	1

- Complement
  - $x+x'=1$: 0+0'=0+1=1; 1+1'=1+0=1
  - $x \cdot x'=0$: 0 • 0'=0 • 1=0; 1 • 1'=1 • 0=0
- Has two distinct elements 1 and 0, with $0 \neq 1$
- We have just established a two-valued Boolean algebra:
  - a set of two elements
  - + : OR operation; • : AND operation
  - a complement operator: NOT operation
  - Binary logic is a two-valued Boolean algebra
  - also called “switching algebra” by engineers

Basic Theorems and Properties of Boolean Algebra

- Duality
  - the binary operators are interchanged; AND ⇔ OR
  - the identity elements are interchanged; 1 ⇔ 0

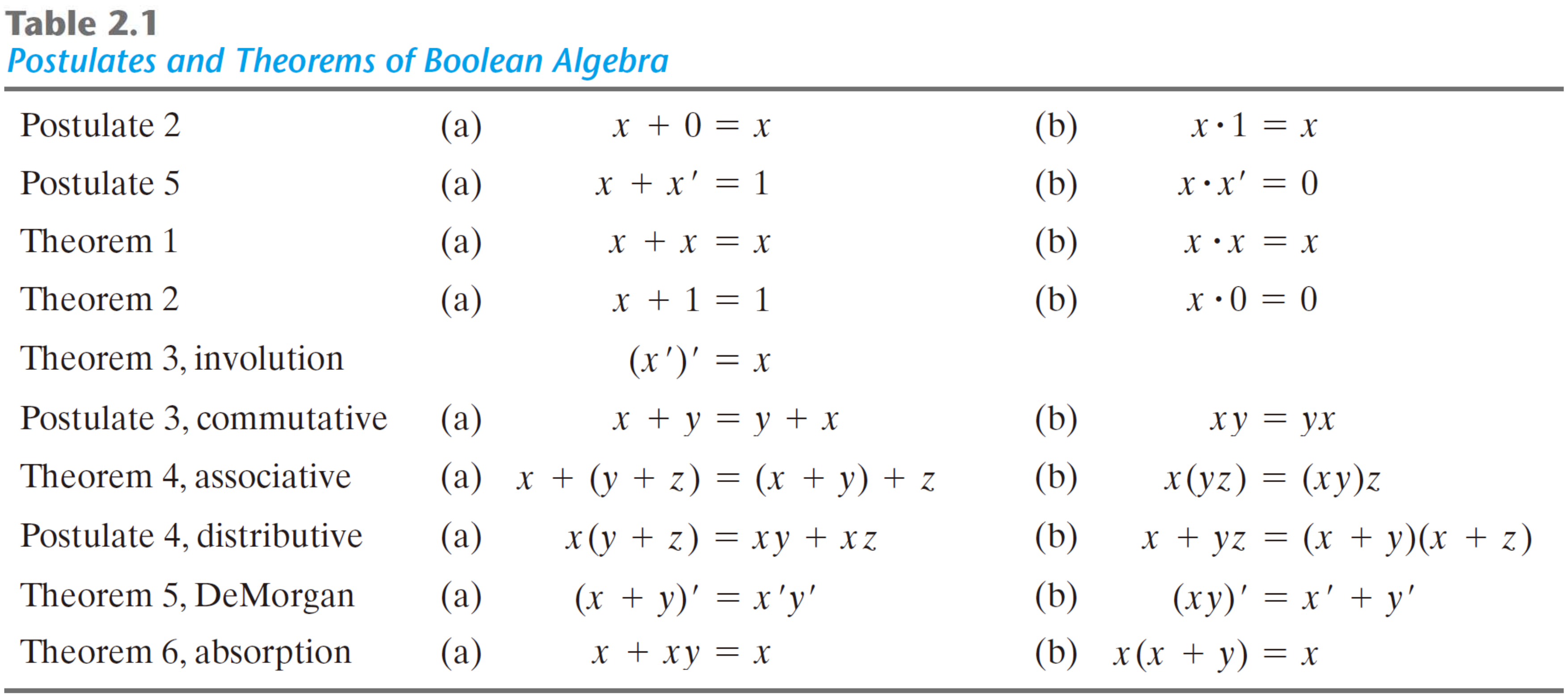


Table 2.1
Postulates and Theorems of Boolean Algebra

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)	$x(x + y) = x$

- Theorem 1(a): $x+x = x$

$$\begin{aligned} x+x &= (x+x) \bullet 1 && \text{by postulate: 2(b)} \\ &= (x+x)\,(x+x') && \text{5(a)} \\ &= x+xx' && \text{4(b)} \\ &= x+0 && \text{5(b)} \\ &= x && \text{2(a)} \end{aligned}$$

- Theorem 1(b): $x \bullet x = x$

$$\begin{aligned} x \bullet x &= x\,x + 0 && \text{by postulate: 2(a)} \\ &= xx + xx' && \text{5(b)} \\ &= x\,(x + x') && \text{4(a)} \\ &= x \bullet 1 && \text{5(a)} \\ &= x && \text{2(b)} \end{aligned}$$

- Theorem 1(b) is the dual of theorem 1(a)

- Theorem 2(a): $x + 1 = 1$

$x + 1 = 1 \bullet (x + 1)$ by postulate: 2(b)

$= (x + x')(x + 1)$ 5(a)

$= x + x' \bullet 1$ 4(b)

$= x + x'$ 2(b)

$= 1$ 5(a)

- Theorem 2(b): $x \bullet 0 = 0$ by duality
- Theorem 3: $(x')' = x$
  - Postulate 5 defines the complement of x, $x + x' = 1$ and $x \bullet x' = 0$
  - The complement of x' is x is also $(x')'$

- Theorem 6(a): $x + xy = x$

$x + xy = x \bullet 1 + xy$ by postulate: 2(b)

$= x (1 + y)$ 4(a)

$= x \bullet 1$ 2(a)

$= x$ 2(b)

- Theorem 6(b): $x (x + y) = x$ by duality
- By means of truth table

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

DeMorgan's Theorems

– $(x+y)'=x'y$‘

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

– $(x\,y)'=x'+y'$

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

Operator Precedence

- The operator precedence for evaluating Boolean expressions is

  1. parentheses
  2. NOT
  3. AND
  4. OR

- Examples
  - $x\, y' + z$
  - $(x\, y + z)'$

Boolean Functions

- A Boolean function is an algebraic expression consists of
  - binary variables
  - binary operators OR and AND
  - unary operator NOT
  - parentheses
- A Boolean function expresses the logical relationship between binary variables and is evaluated by determining the binary value of the expression for all possible values of the variables.
- Examples
  - $F_1 = x + y z'$ ➔ $F_1 = 1$ if $x = 1$ or if $y = 0$ and $z = 1$, others $F_1 = 0$.
  - $F_2 = x' y' z + x' y z + x y'$ ➔

    $F_2 = 1$ if $(x = 0, y = 0, z = 1)$ or $(x = 0, y = 1, z = 1)$ or $(x = 1, y = 0)$,

    others $F_2 = 0$.

What are the others?

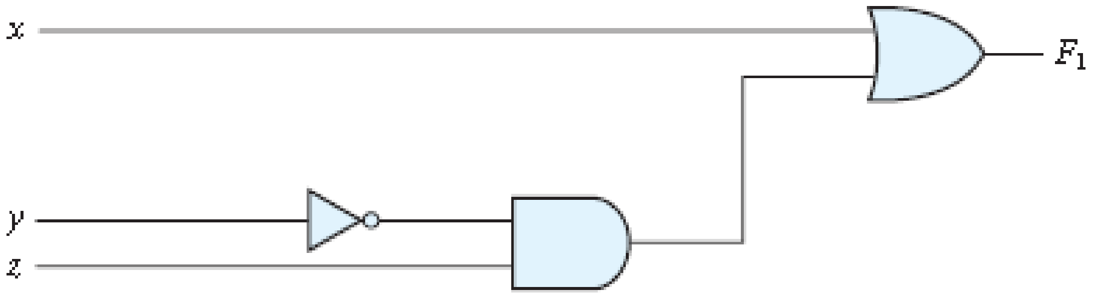
Truth Table

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

Table 2.2
Truth Tables for F_1 and F_2

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

Implementation of F_1 with logic gates

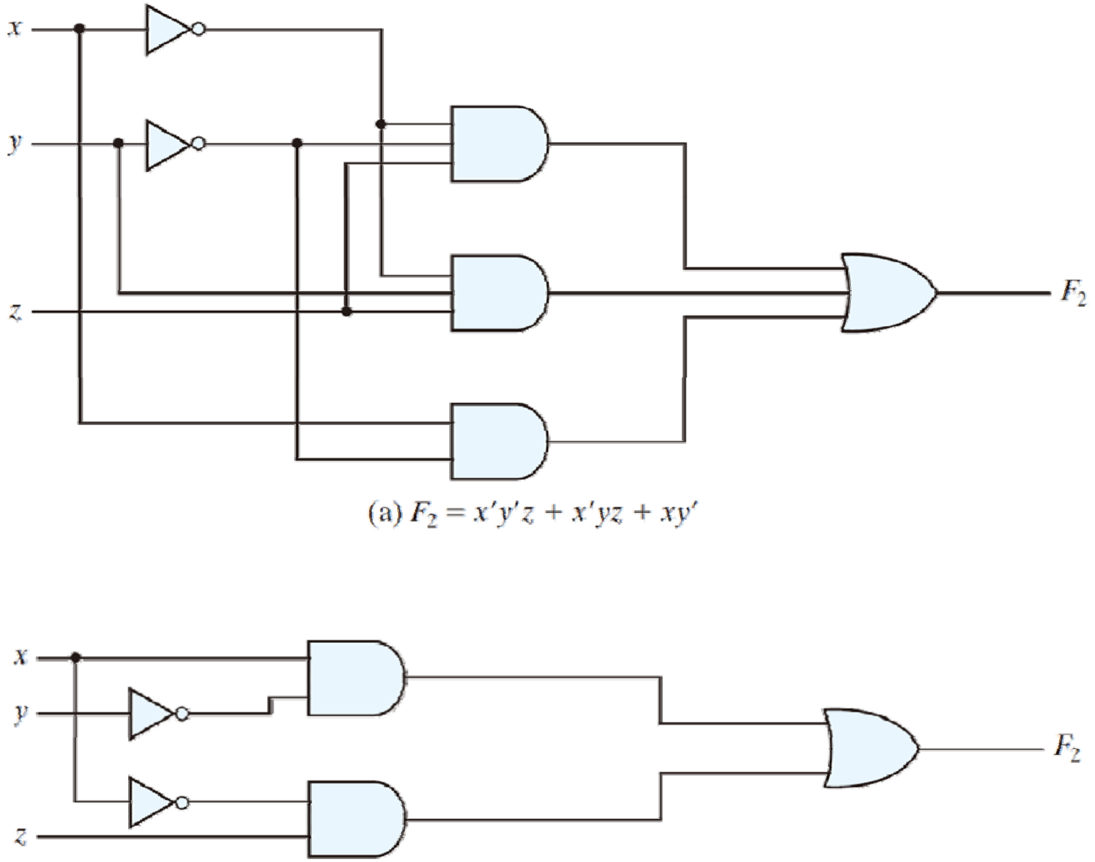


$F_1 = x + y z'$

Equivalent Logics

- Boolean function can be represented in truth table only in one way.
- In algebraic form, it can be expressed in a variety of ways, all of which have equivalent logic.
- Using Boolean algebra, it is possible to obtain a simpler expression for the same function with less number of gates and inputs to the gate.
- Designers work on reducing the complexity and number of gates to significantly reduce the circuit cost.

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



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

(b) $F_2 = xy' + x'z$

FIGURE 2.2
Implementation of Boolean function F_2 with gates

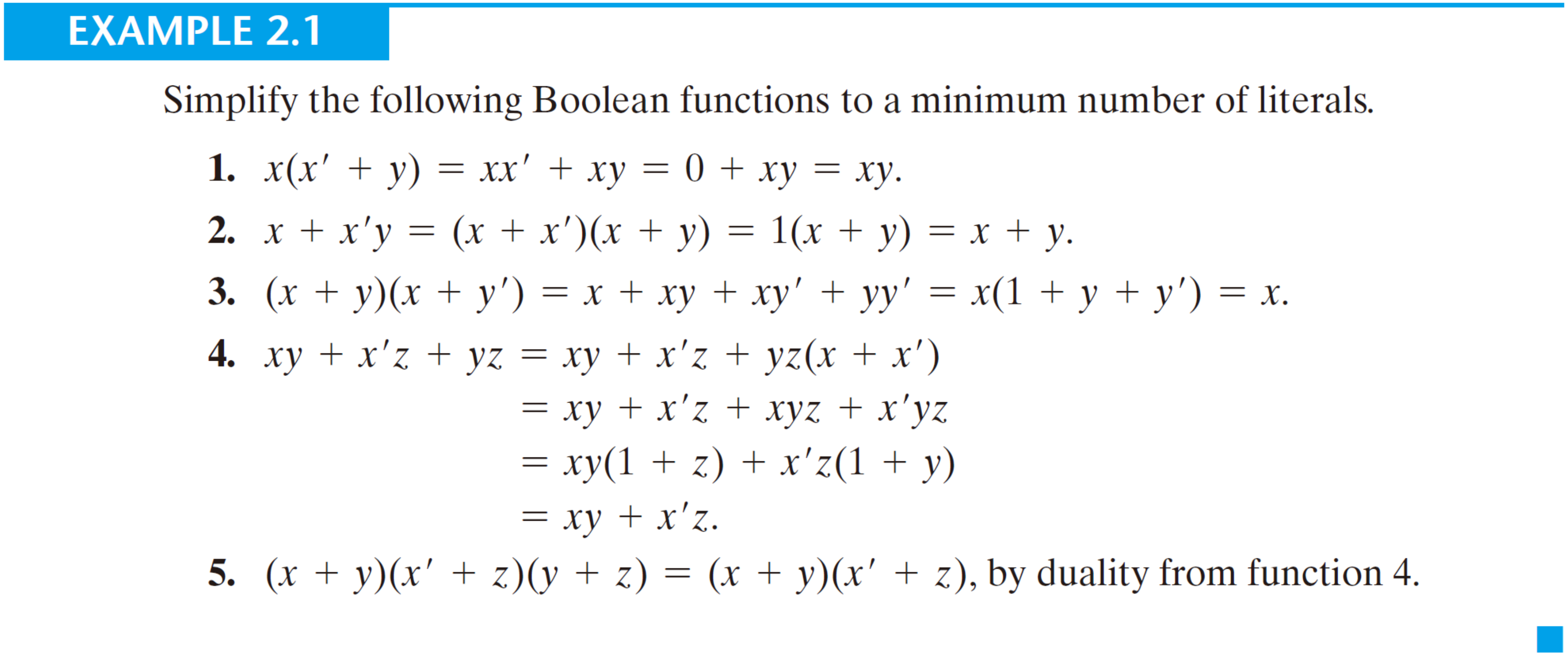
Algebraic Manipulation

- To minimize Boolean expressions
  - literal: a complemented or un-complemented variable (an input to a gate)
  - term: an implementation with a gate
  - The minimization of the number of literals and the number of terms => a circuit with less equipment

$$F_2 = x'y'z + x'yz + xy' \quad \Rightarrow \text{3 terms, 8 literals}$$
$$= x'z(y' + y) + xy'$$
$$= x'z + xy' \quad \Rightarrow \text{2 terms, 4 literals}$$

- Functions of up to five variables can be simplified by the map method described in the next chapter.
- For complex Boolean functions and many different outputs, designers of digital circuits use computer minimization programs that are capable of producing optimal circuits with millions of logic gates.

Minimization of Boolean Function



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. $(x + y)(x + y') = x + xy + xy' + yy' = x(1 + y + y') = x.$

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

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

■

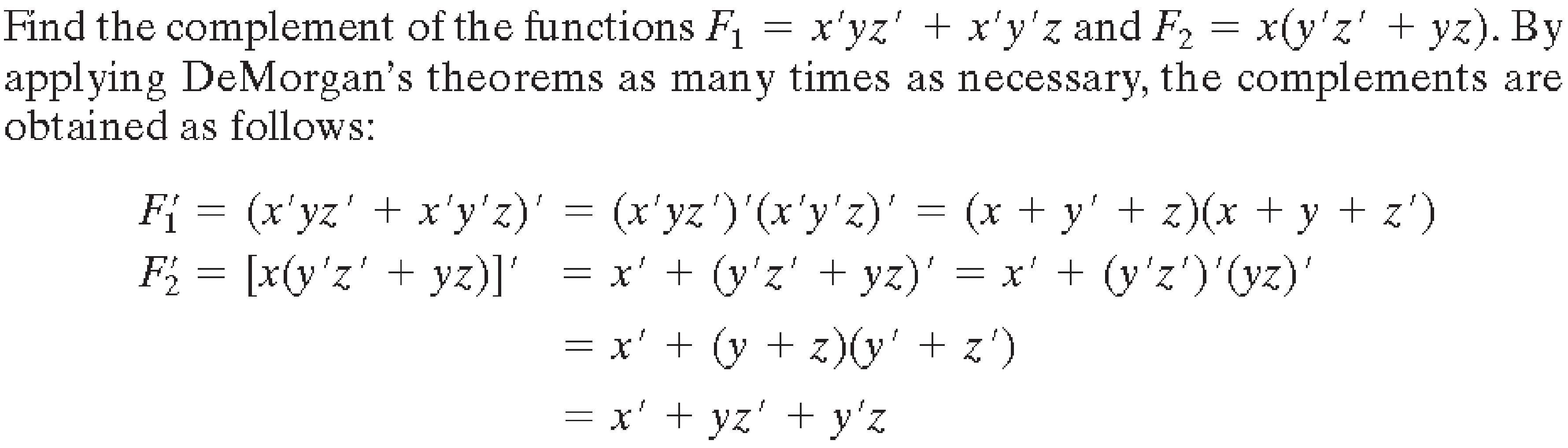
Complement of a Function

- F' is obtained from an interchange of 0's for 1's and 1's for 0's in the value of F
- The complement of a function may be derived using DeMorgan's theorem.
- Three-variable DeMorgan's theorem:

$$\begin{aligned}
(A + B + C)' &= (A + X)' && \text{let } B + C = X \\
&= A'X' && \text{by DeMorgan's} \\
&= A'(B + C)' && X = B + C \\
&= A'(B'C') && \text{by DeMorgan's} \\
&= A'B'C' && \text{associative}
\end{aligned}$$

- Generalized form
  - $(A + B + C + \ldots + F)' = A'B'C' \ldots F'$
  - $(ABC \ldots F)' = A' + B' + C' + \ldots + F'$

EXAMPLE 2.2

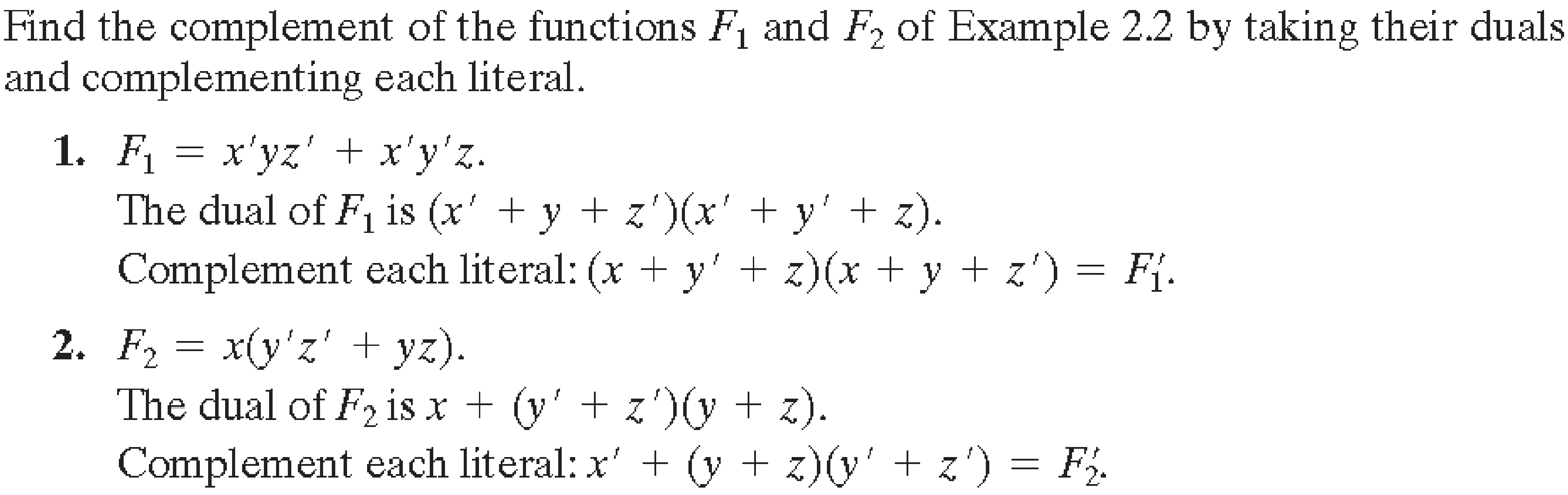


Find the complement of the functions $F_1 = x'yz' + x'y'z$ and $F_2 = x(y'z' + yz)$. By applying DeMorgan's theorems as many times as necessary, the complements are obtained as follows:

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

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

EXAMPLE 2.3



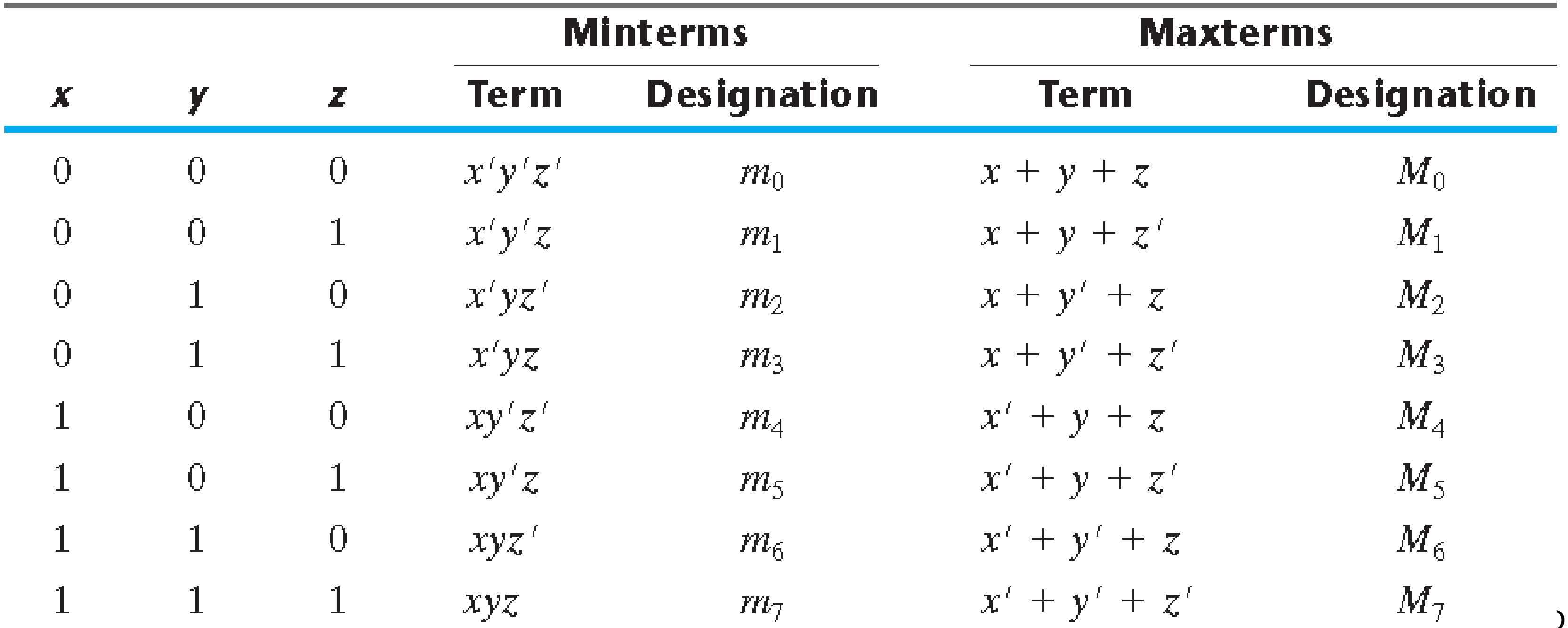
Find the complement of the functions F_1 and F_2 of Example 2.2 by taking their duals and complementing each literal.

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

Minterms and Maxterms

- A minterm (standard product): an AND term consists of all literals in their normal form or in their complement form
- For example, two binary variables *x* and *y*, has 4 minterms
  - *xy, xy', x'y, x'y‘*
- *n* variables can be combined to form 2^n minterms (m_j, $j = 0 \sim 2^n$-1)
- A maxterm (standard sum): an OR term; 2^n maxterms (M_j, $j = 0 \sim 2^n$-1)
- Each maxterm is the complement of its corresponding minterm, and vice versa.



			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: Sum of Minterms

- An Boolean function can be expressed by
  - a truth table
  - sum of minterms ➔ $f = \Sigma\, m_j$
  - product of maxterms ➔ $f = \Pi\, M_j$
  - $f_1 = x'y'z + xy'z‘ + xyz = m_1 + m_4 + m_7$
  - $f_2 = \mathrm{x'yz} + \mathrm{xy'z} + \mathrm{xyz‘} + \mathrm{xyz} = m_3 + m_5 + m_6 + m_7$

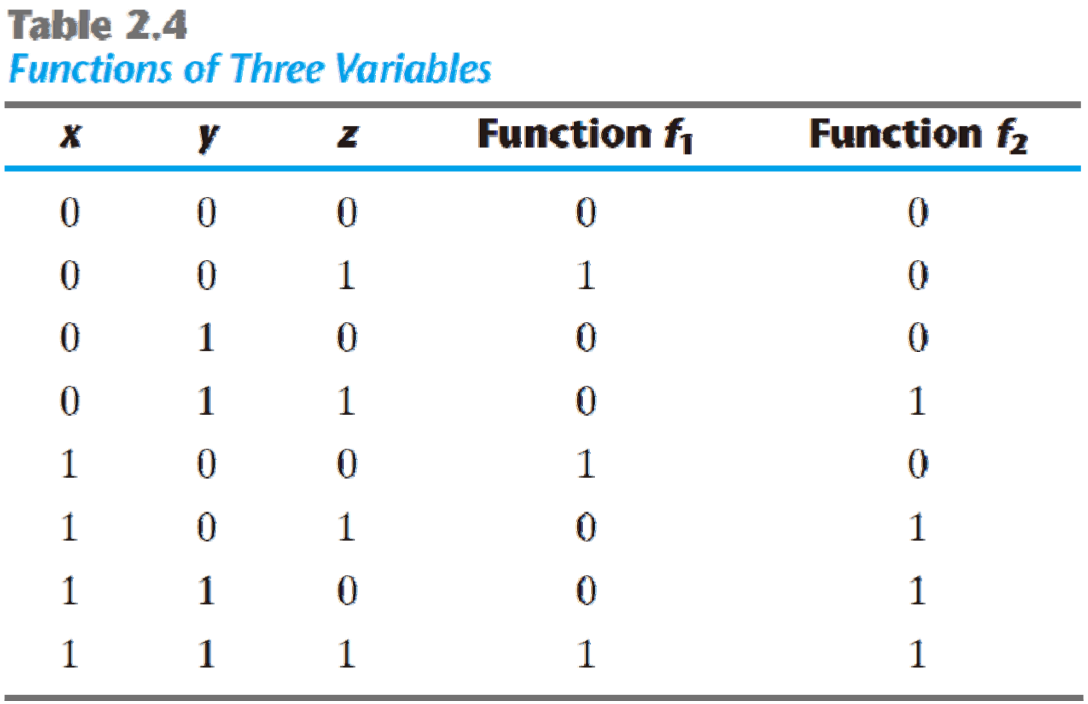


Table 2.4
Functions of Three Variables

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

Canonical Form: Product of Maxterms

- The complement of a Boolean function
  - the minterms that produce a 0
  - $f_1' = m_0 + m_2 + m_3 + m_5 + m_6 = x'y'z' + x'yz' + x'yz + xy'z + xyz'$
  - $f_1 = (f_1')' = (x + y + z)(x + y' + z)\ (x + y' + z')\ (x' + y + z')(x' + y' + z)$
  - $= M_0\, M_2\, M_3\, M_5\, M_6$
  - $f_2 = (x + y + z)(x + y + z')(x + y' + z)(x' + y + z)$
    $= M_0\, M_1\, M_2\, M_4$
- Canonical form: any Boolean function expressed as a sum of minterms or a product of maxterms

Minterm Expansion

- **EXAMPLE 2.4:** Express the Boolean function *F=A+B’C* as a sum of minterms.
  - F = A + B'C = A (B + B') + B'C = AB + AB' + B'C
  - = AB(C + C') + AB'(C + C') + (A + A')B'C
  - = ABC + ABC‘ + AB'C + AB'C‘ + A'B'C
  - = A'B'C + AB'C' + AB'C + ABC‘ + ABC
  - $= m_1 + m_4 + m_5 + m_6 + m_7$
  - F(A,B,C) = Σ (1, 4, 5, 6, 7)
  - or, built the truth table first

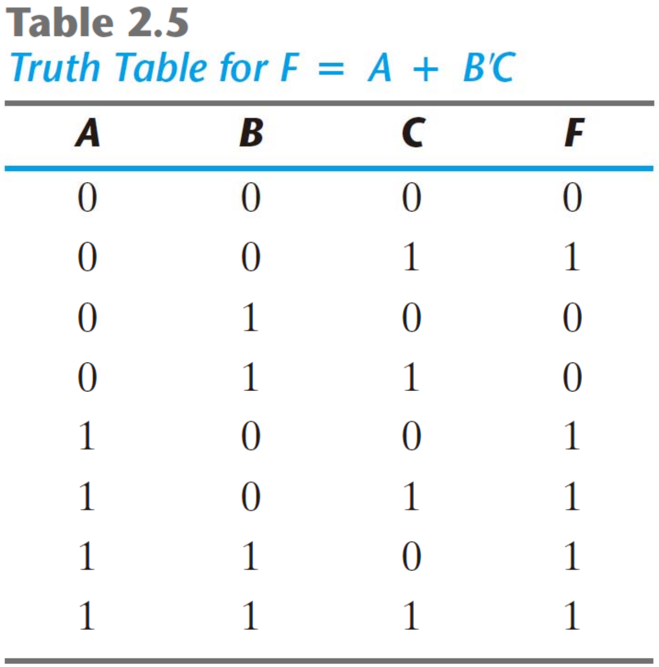


Table 2.5
Truth Table for F = A + B'C

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

Maxterm Expansion

EXAMPLE 2.5: Express the Boolean function $F = xy + x'z$ as a product of maxterms.

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

- $x' + y = x' + y + zz' = (x' + y + z)(x' + y + z')$
- $x + z = x + z + yy' = (x + y + z)(x + y' + z)$
- $y + z = y + z + xx' = (x + y + z)(x' + y + z)$

- $F = (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z') = M_0 M_2 M_4 M_5$
- $F(x,y,z) = \Pi\,(0,2,4,5)$

- check this result with truth table

Canonical Form Conversion

- Conversion between Canonical Forms
  - $F(A,B,C) = \Sigma(1,4,5,6,7)$ ➔ $F'(A,B,C) = \Sigma(0,2,3) = m_0 + m_1 + m_2$
  - By DeMorgan's theorem

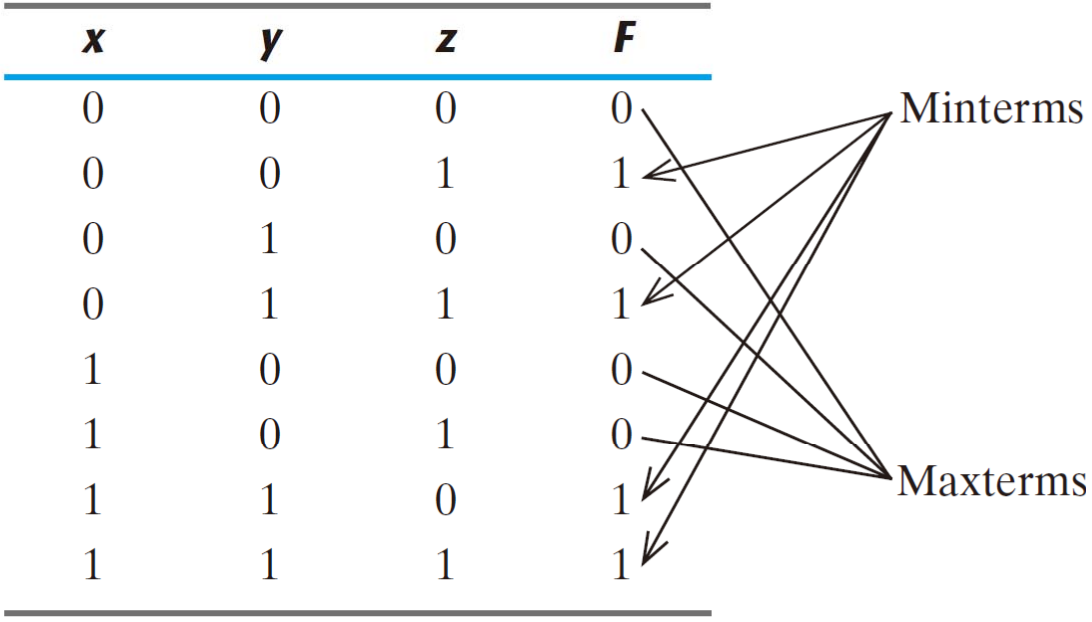
    $F = (m_0 + m_1 + m_2)' = m'_0 \bullet m'_2 \bullet m'_3$

    $= M_0 M_2 M_3 = \Pi(0, 2, 3)$
  - $m_j' = M_j$
- sum of minterms ⇔ product of maxterms
  - interchange the symbols Σ and Π and list those numbers missing from the original form
    - Σ of 1's ⇔ Π of 0's

Conversion Example

- $F = xy + x'z$
- $F(x, y, z) = \Sigma(1, 3, 6, 7)$
- $F(x, y, z) = \Pi\,(0, 2, 4, 5)$

Table 2.6
Truth Table for $F = xy + x'z$

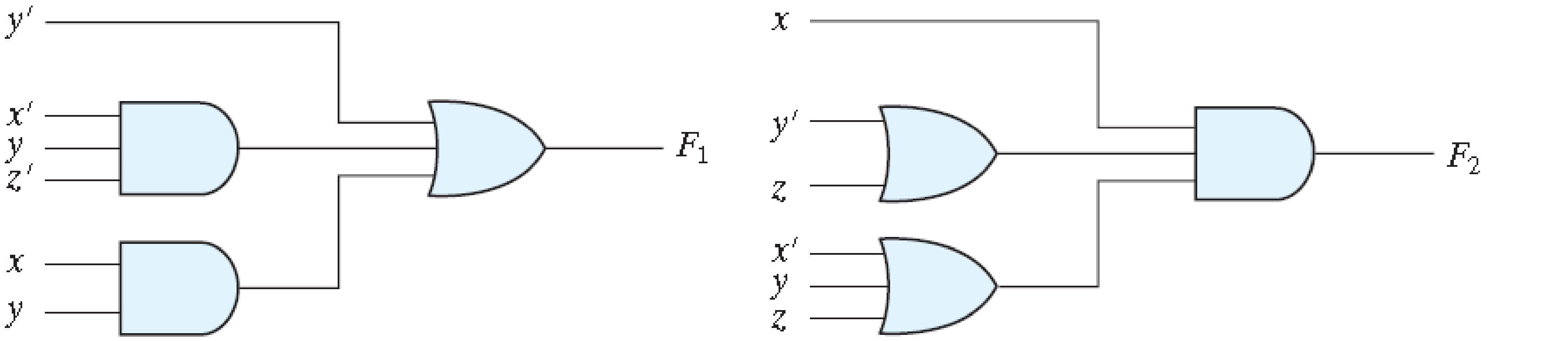


x	*y*	*z*	*F*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Minterms
Maxterms

Standard Forms

- Canonical forms are baseline expression and seldom used, they are not minimum
- Two standard forms are used usually
  - sum of products $F_1 = y' + xy + x'yz'$
  - product of sums $F_2 = x(y‘ + z)(x‘ + y + z’)$



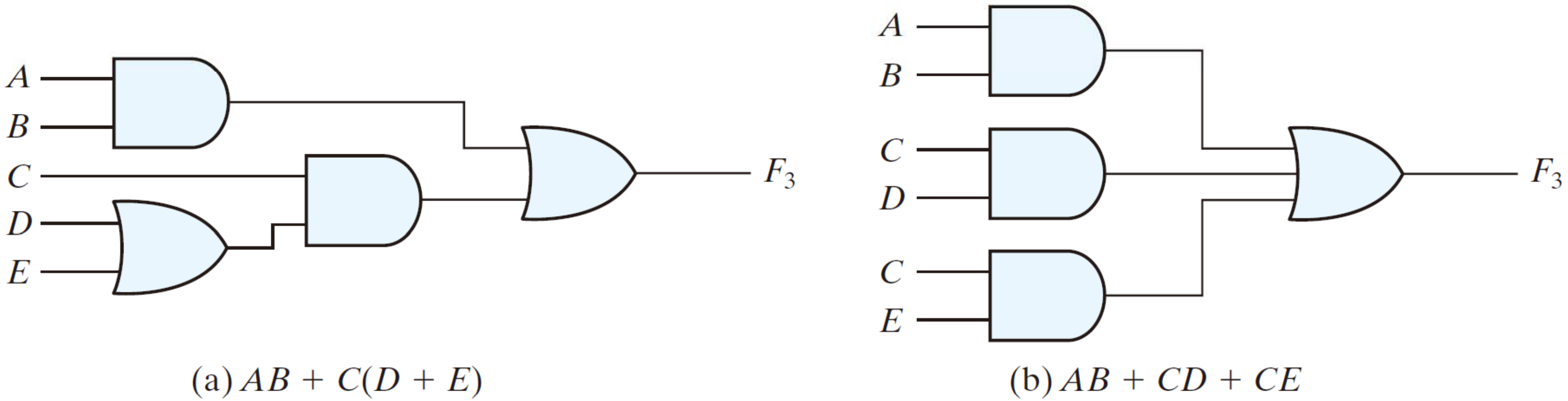
(a) Sum of Products

(b) Product of Sums

- This circuit configuration is referred to as a *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. However, the number of inputs to a given gate might not be practical.

Nonstandard Forms

- $F_3 = AB + C(D + E)$
  $= AB + C(D + E) = AB + CD + CE$



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

(b) $AB + CD + CE$

FIGURE 2.4
Three- and two-level implementation

- Which kind of gate will have the least delay (high switching speed)?
- The delay through a gate is largely dependent on the circuit design and technology, as well as manufacturing process used. (taught in VLSI design)

Other Logic Operations

- 2^n rows in the truth table of *n* binary variables
- 2^{2^n} functions for *n* binary variables (each row may either be 0 or 1)
- 16 (2^{2^2})functions of two binary variables

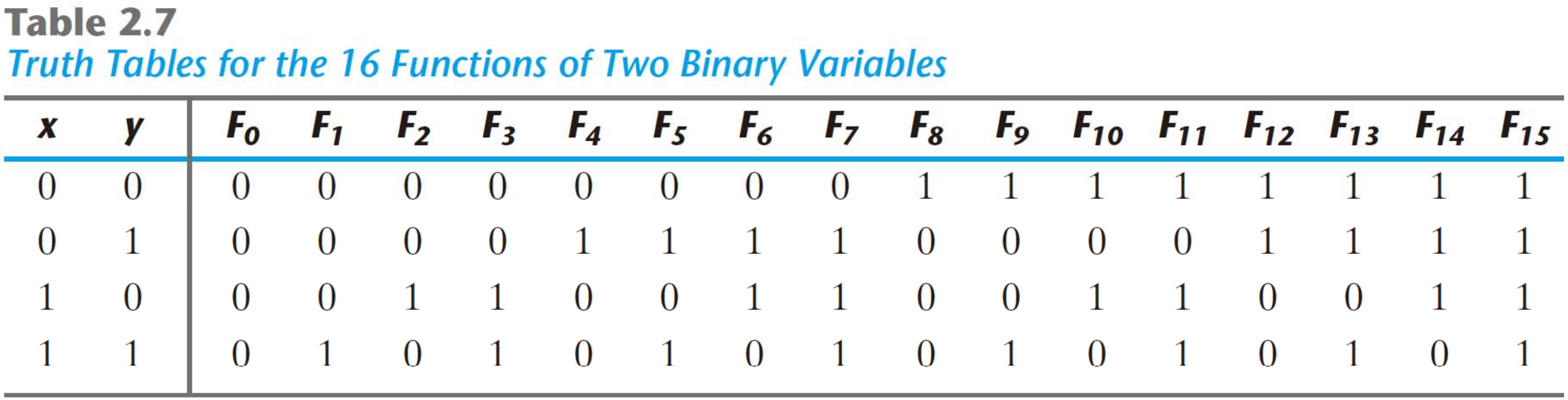


Table 2.7

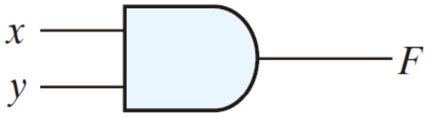
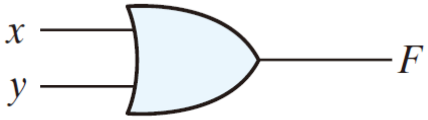
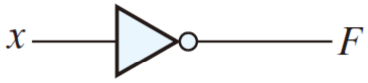
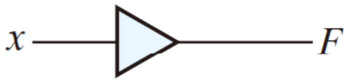
Truth Tables for the 16 Functions of Two Binary Variables

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

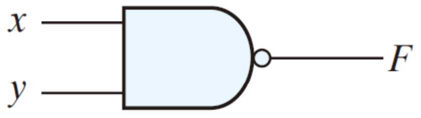
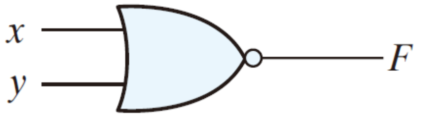
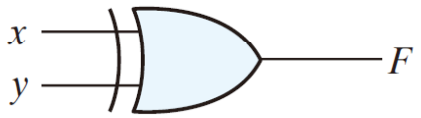
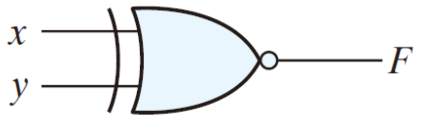
Table 2.8
Boolean Expressions for the 16 Functions of Two Variables

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

Digital Logic Gates of Two Inputs

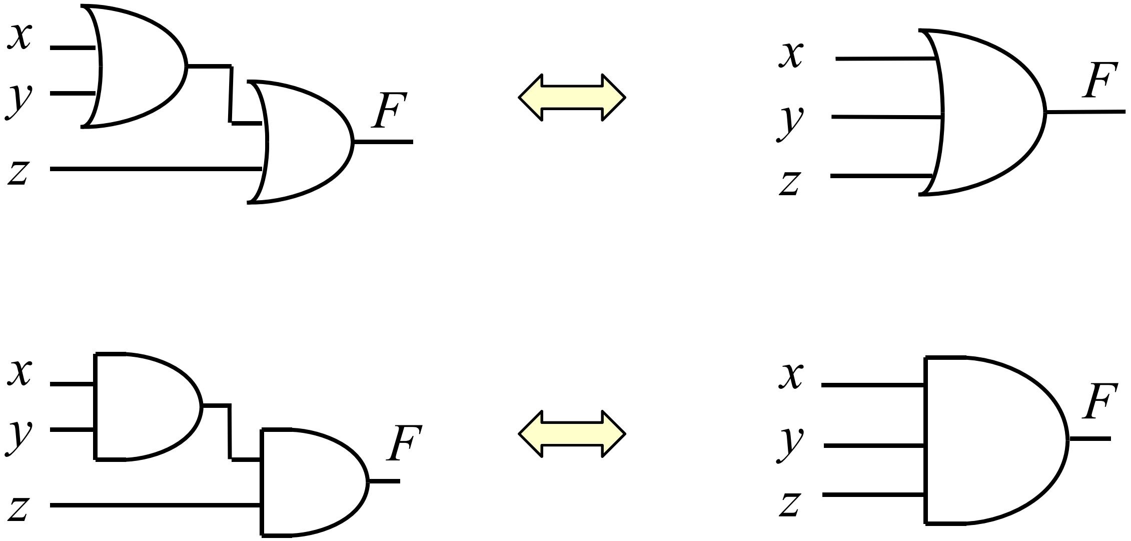
Name	Graphic symbol	Algebraic function	Truth table
AND		$F = x \cdot y$	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x + y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$F = x'$	x | F 0 | 1 1 | 0
Buffer		$F = x$	x | F 0 | 0 1 | 1

Digital Logic Gates of Two Inputs

			x	y	F
NAND		$F = (xy)'$	0	0	1
			0	1	1
			1	0	1
			1	1	0
NOR		$F = (x + y)'$	0	0	1
			0	1	0
			1	0	0
			1	1	0
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	0	0	0
			0	1	1
			1	0	1
			1	1	0
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	0	0	1
			0	1	0
			1	0	0
			1	1	1

Extension to Multiple Inputs

- A gate can be extended to multiple inputs
  - if its binary operation is commutative and associative
- AND and OR are commutative and associative
  - commutative: $x + y = y + x$, $xy = yx$
  - associative: $(x + y) + z = x + (y + z) = x + y + z$, $(x\,y)z = x(y\,z) = x\,y\,z$



Multiple-input NOR/NAND

- NAND and NOR are commutative but not associative => they are not extendable

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

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

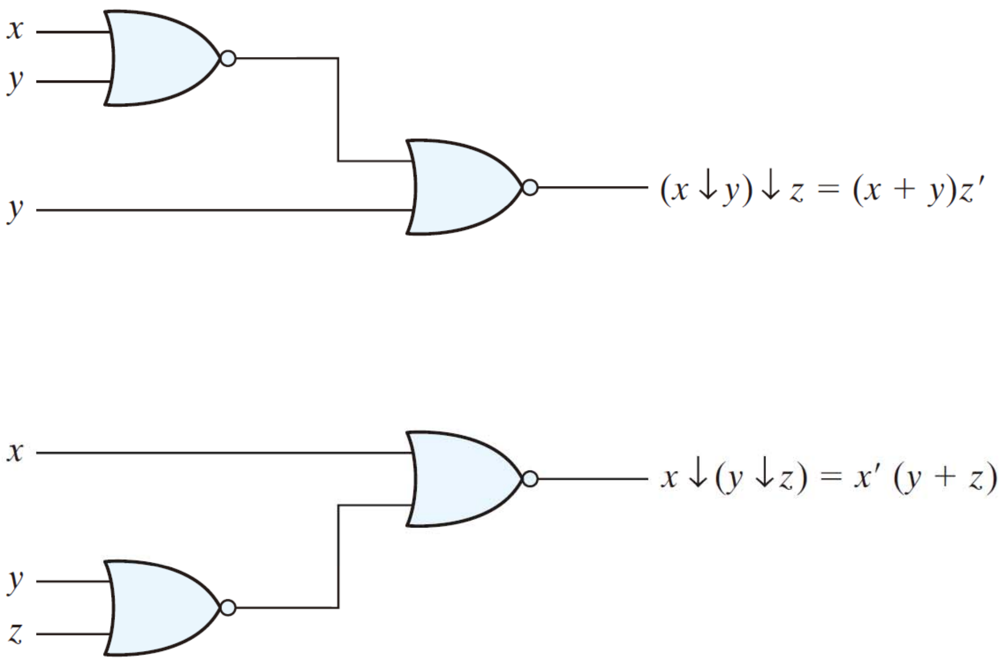
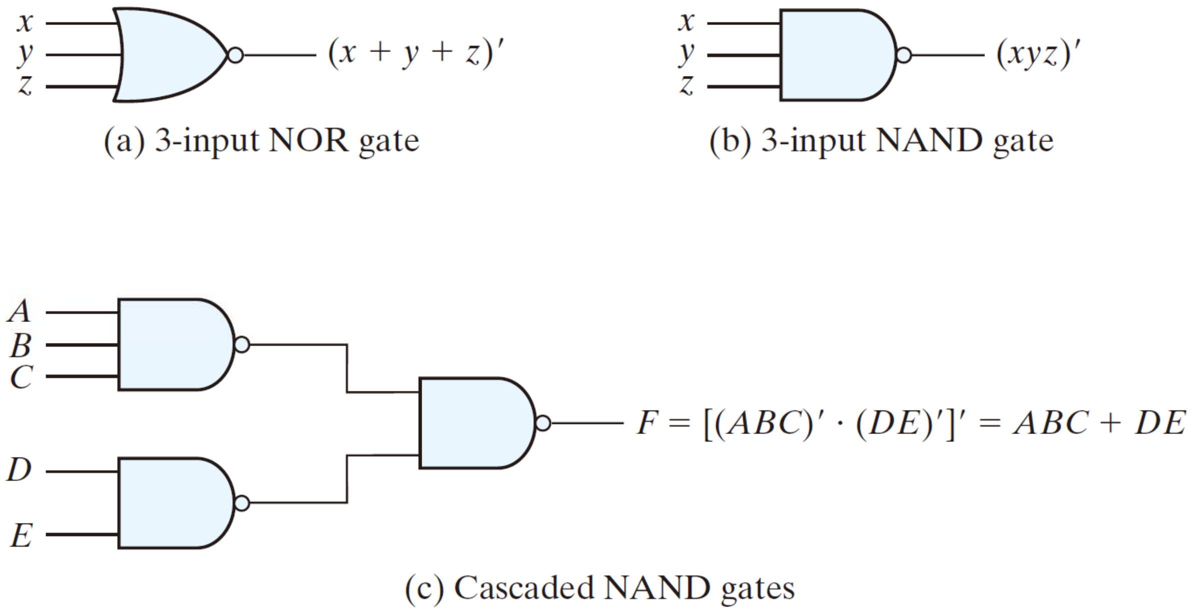

x
y
y
$(x \downarrow y) \downarrow z = (x + y)z'$
x
$x \downarrow (y \downarrow z) = x'\ (y + z)$
y
z

FIGURE 2.6
Demonstrating the nonassociativity of the NOR operator: $(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$

Multiple-input NOR/NAND

- Multiple-input NOR = a complement of OR gate $(x \downarrow y \downarrow z) = (x + y + z)'$
- Multiple-input NAND = a complement of AND $(x \uparrow y \uparrow z) = (x\, y\, z)'$
- The cascaded NAND operations = sum of products
- The cascaded NOR operations = product of sums



(a) 3-input NOR gate

(b) 3-input NAND gate

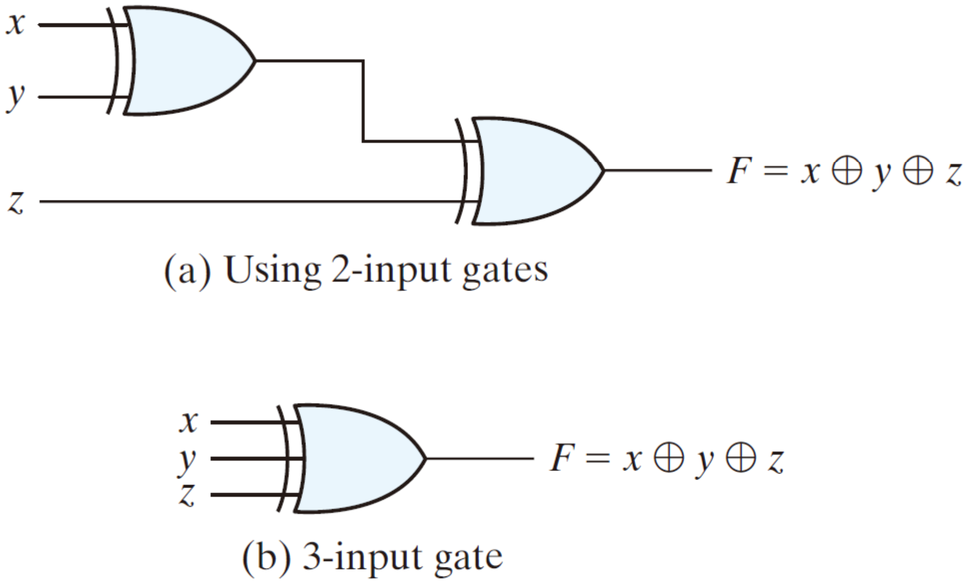
(c) Cascaded NAND gates

FIGURE 2.7
Multiple-input and cascaded NOR and NAND gates

DeMorgan’s theorems are useful here.

Multiple-input XOR/XNOR

- The XOR and XNOR gates are commutative and associative
- Multiple-input XOR gates are uncommon (this is not true anymore!)
- XOR(XNOR) is an odd(even) function: it is equal to 1 if the inputs variables have an odd(even) 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

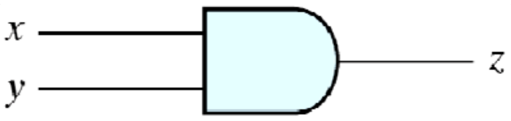
FIGURE 2.8
Three-input exclusive-OR gate

Positive and Negative Logic

- Two signal values (High/Low) <=> two logic values (1/0)
  - positive logic: H = 1; L = 0
  - negative logic: H = 0; L = 1
- Positive logic is commonly used.

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

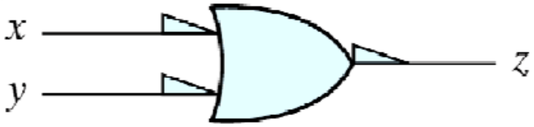
(c) Truth table for positive logic



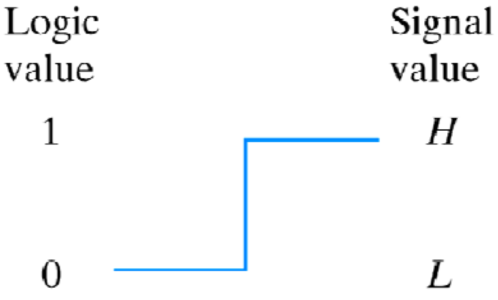
(d) Positive logic AND gate

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

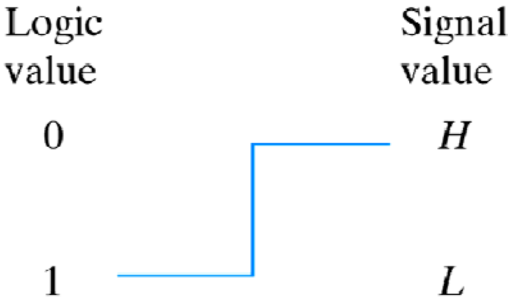
(e) Truth table for negative logic



(f) Negative logic OR gate



(a) Positive logic



(b) Negative logic

Digital Logic Families

- Digital circuit technology:
  - TTL: transistor-transistor logic (dying?)
  - ECL: emitter-coupled logic (high speed, high power consumption)
  - MOS: metal-oxide semiconductor (NMOS, high density)
  - **CMOS**: complementary MOS (low power)
- CMOS technology now dominates the main stream of IC design, it will be taught in *Introduction to VLSI Design* course.

Some Important Parameters of Logic Families

- **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.

Homework #2

- 2.2 (e) (f)
- 2.4 (d) (e)
- 2.9 (c)
- 2.11 (b)
- 2.14 (b) (c)
- 2.22 (b)
- 2.28