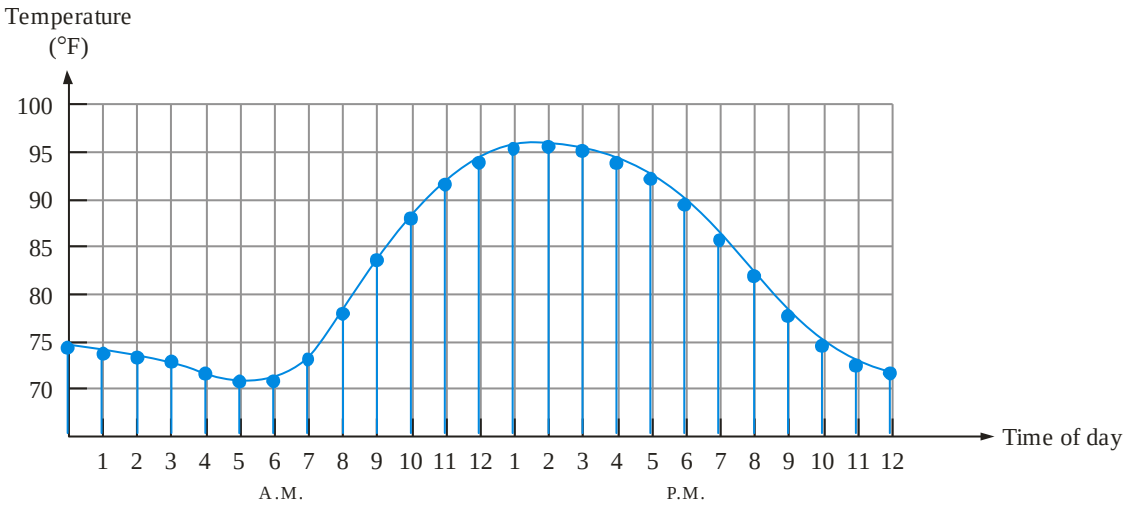
Module 1- Digital Logic

Dr.N.Subhashini

Associate Professor(SENSE)

Analog Quantities

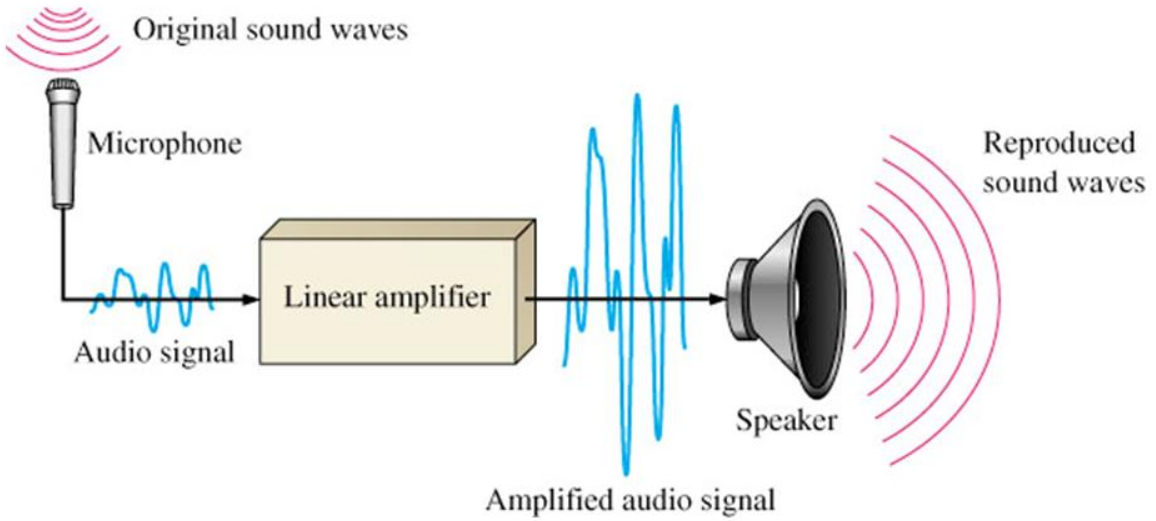
Most natural quantities that we see are **analog** and vary continuously. Analog systems can generally handle higher power than digital systems.



Digital systems can process, store, and transmit data more efficiently but can only assign discrete values to each point.

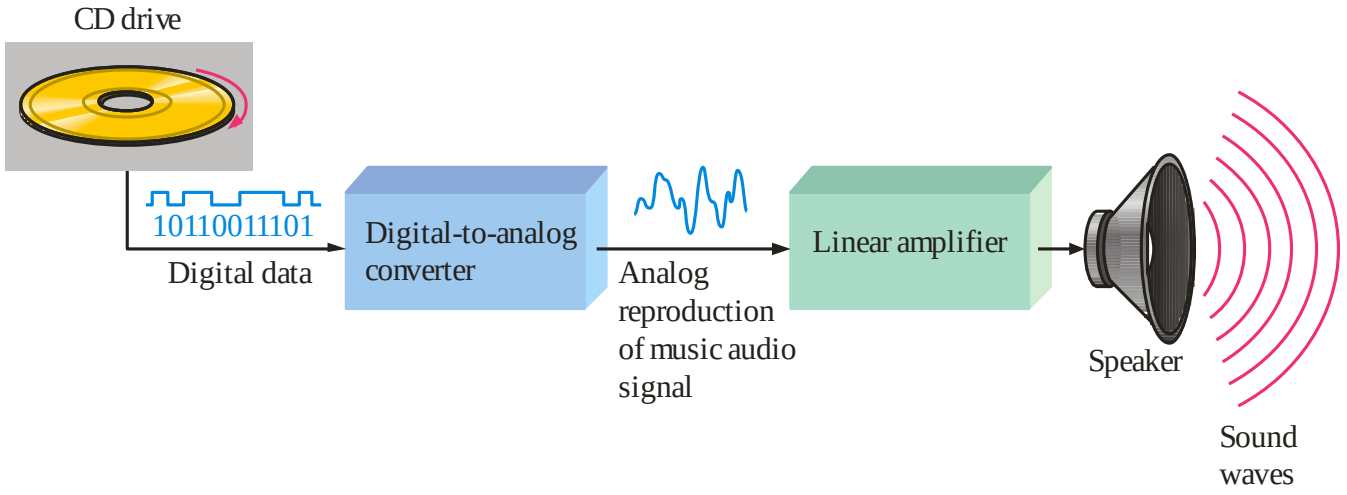
Basic Audio Address System

Figure 1–3 A basic audio public address system.



Analog and Digital Systems

Many systems use a mix of analog and digital electronics to take advantage of each technology. A typical CD player accepts digital data from the CD drive and converts it to an analog signal for amplification.



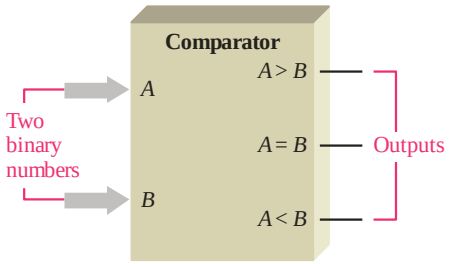
Introduction

- A digital system is an interconnection of digital modules
- To understand the operation of each digital module, it is necessary to have a basic knowledge of digital circuits and their logical function
- Digital circuits – logical circuits- process data by means of binary logic elements(logic gates) using binary signals
- HDL- resembles a programming language , Describe and simulate the functionality of a digital circuit

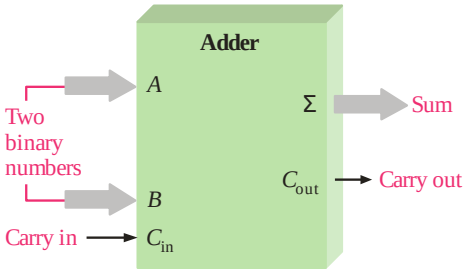
Basic System Functions

And, **or**, and **not** elements can be combined to form various logic functions. A few examples are:

The comparison function

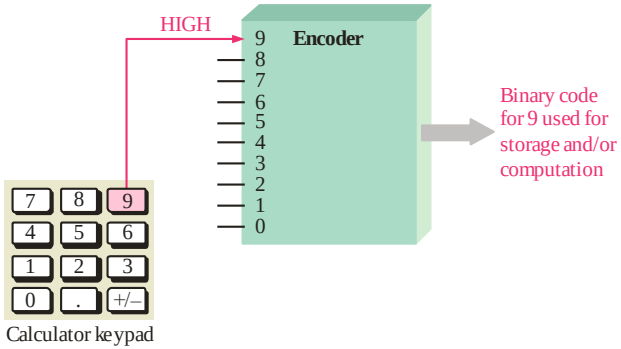


Basic arithmetic functions

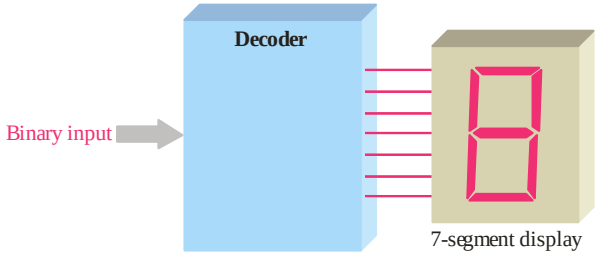


Basic System Functions

The encoding function

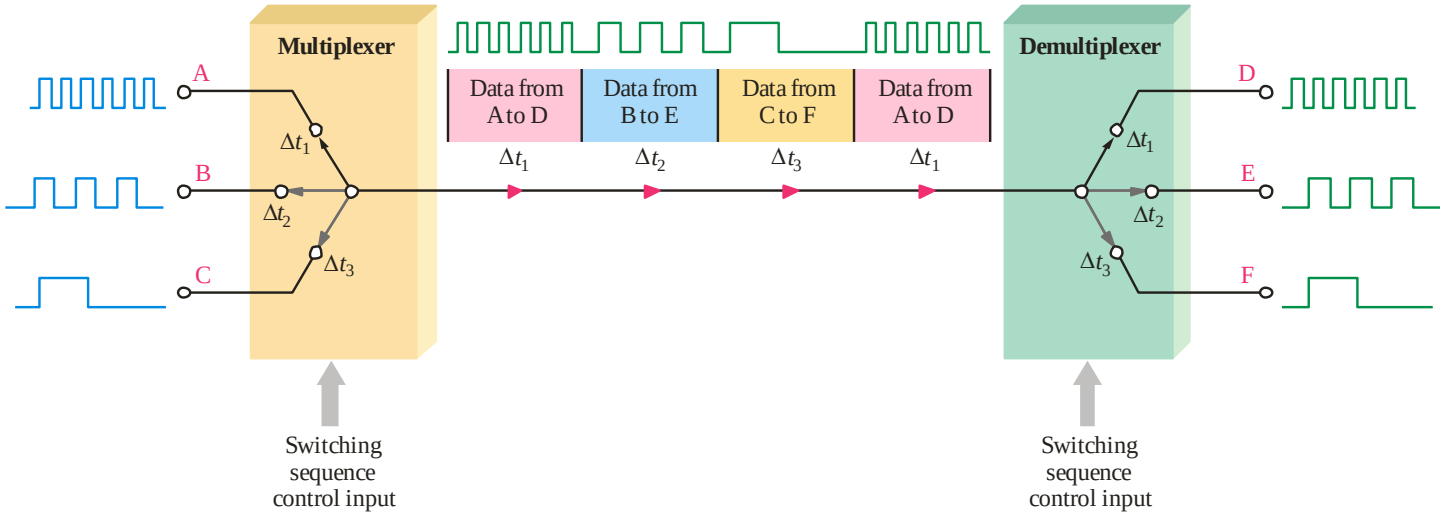


The decoding function



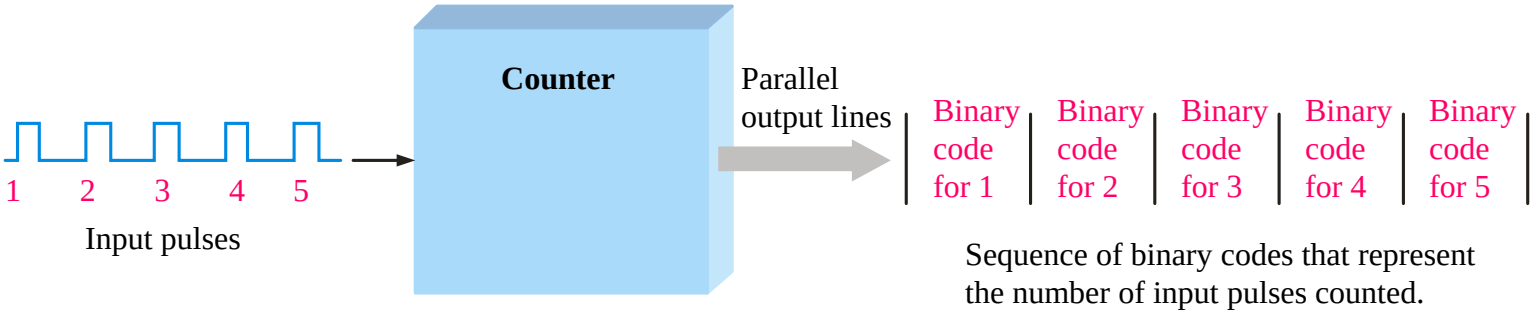
Basic System Functions

The data selection function



Basic System Functions

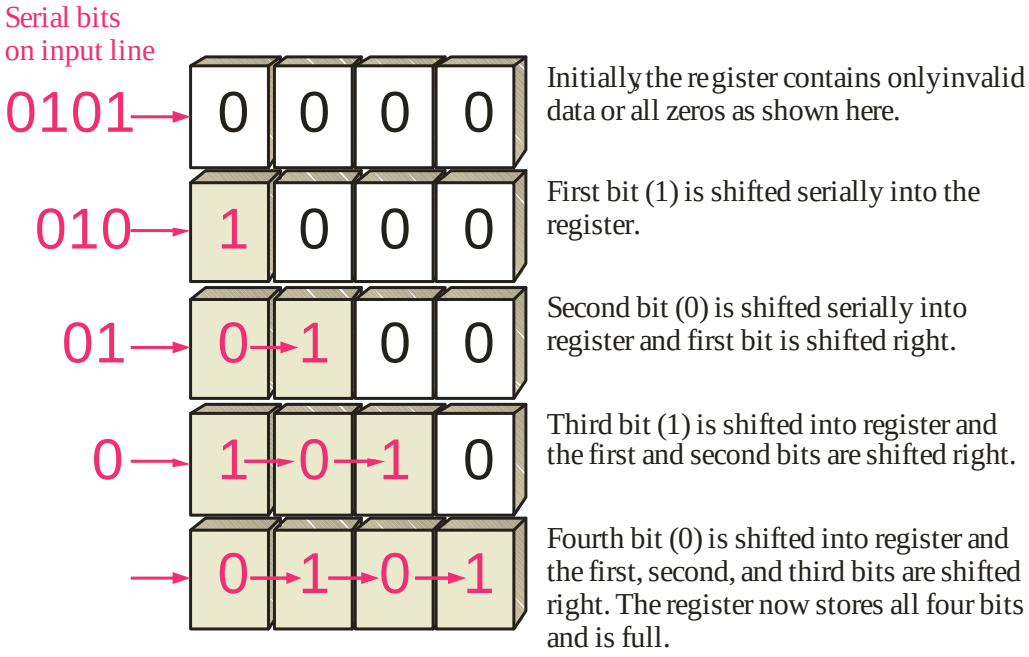
The counting function



…and other functions such as code conversion and storage.

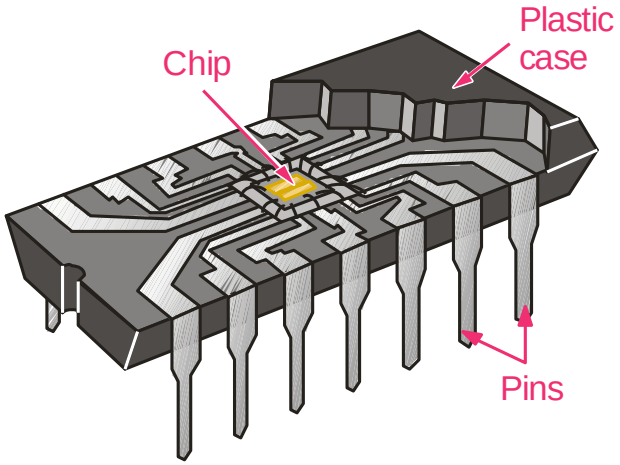
Basic System Functions

One type of storage function is the shift register, that moves and stores data each time it is clocked.



Integrated Circuits

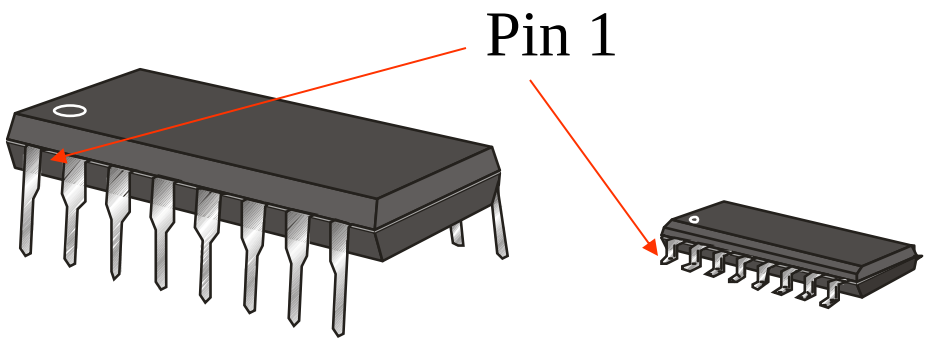
Cutaway view of DIP (Dual-In-line Pins) chip:



The TTL series, available as DIPs are popular for laboratory experiments with logic.

Integrated Circuits

DIP chips and surface mount chips



Dual in-line package

Small outline IC (SOIC)

Introduction to Boolean Algebra

- Because binary logic is used in all of today's digital computers and devices, the cost of the circuits that implement it is an important factor addressed by designers
- Finding simpler and cheaper, but equivalent, realizations of a circuit can reap huge payoffs in reducing the overall cost of the design.
- Mathematical methods that simplify circuits rely primarily on Boolean algebra.
- Help optimize complex circuits involving millions of logic gates.

- In 1854 **George Boole** introduced a systematic logic & developed Boolean Algebra.
- In 1938 **C E Shannon** introduced 2 valued Boolean Algebra called switching algebra
- Postulates of Boolean Algebra are developed by **Huntington** in 1904

- 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, usually having a common property.

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.

As an example,

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 $c \notin S$.

- The postulates of a mathematical system form the basic assumptions from which it is possible to deduce the rules, theorems, and properties of the system.

- **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.

For example, the set of natural numbers $N = \{1, 2, 3, 4, ..,c\}$ is closed with respect to the binary operator $+$ by the rules of arithmetic addition, since, for any $a, b \in N$, there is a unique $c \in N$ such that $a + b = c$.

The set of natural numbers is *not* closed with respect to the binary operator - by the rules of arithmetic subtraction, because $2 - 3 = -1$ and $2, 3 \in N$, but $(-1) \in N$.

- **2.** *Associative law.* A binary operator * on a set S is said to be associative whenever

$(x * y) * z = x * (y * z)$ for all $x, y, z, \in S$

- 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.

link

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

$e * x = x * e = x$ 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, $N=\{1,2,3,4...\}$, has no identity element, since 0 is excluded from the set.

link

- **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

$x * (y \,.\, z) = (x * y) \,.(x * z)$

link

Huntington Postulates

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 $x \neq y$.

Comparing Boolean algebra with arithmetic and ordinary algebra (the field of real numbers), we note the following differences:

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., $x + (y \cdot z) = (x + y) \cdot (x + z)$) 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.

- Basic Theorems of Boolean Algebra
  - Duality Principle
  - Basic Theorems & Postulates

Duality Principle

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

Applying duality principle to Postulates

- $x + 0 = x \qquad x.1 = x$
- Commutataive Law :
  $x + y = y + x \qquad x.y = y.x$
- Distributive law
  $x.(y + z) = (x.y) + (x.z) \qquad x + (y.z) = (x + y).(x + z)$
- $x + x' = 1 \qquad x.x' = 0$

Theorems of Boolean Algebra

1. a) $x + x = x$ b) $x.x = x$
2. a) $x + 1 = 1$ b) $x.0 = 0$
3. Involution $((x')') = x$
4. Associative
   a) $x + (y + z) = (x + y) + z$ b) $x.(y.z) = (x.y).z$
5. DeMorgan's Theorem
   a) $\overline{(x + y)} = (\overline{x}.\overline{y})$ b) $(\overline{x.y}) = (\overline{x} + \overline{y})$
6. absorption
   a) $x + x.y = x$ b) $x.(x + y) = x$

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): $x + x = x.$

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)

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

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)

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

Operator Precedence

- Paranthesis
- Not (‘)
- AND(.)
- OR(+)

Boolean Function

- A Boolean function described by an algebraic expression consists of binary variables, the constants 0 and 1, and the logic operation symbols

- For a given value of the binary variables, the function can be equal to either 1 or 0

- A boolean function can be represented in 2 ways
  - Algebraic expression
  - Truth table

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

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

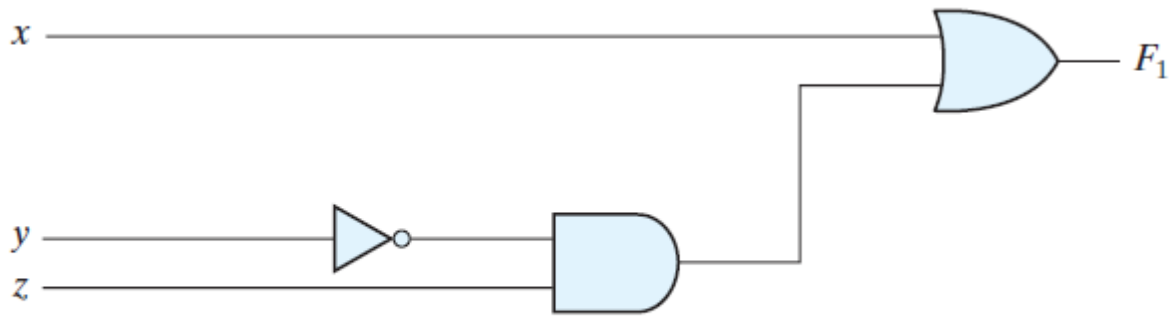
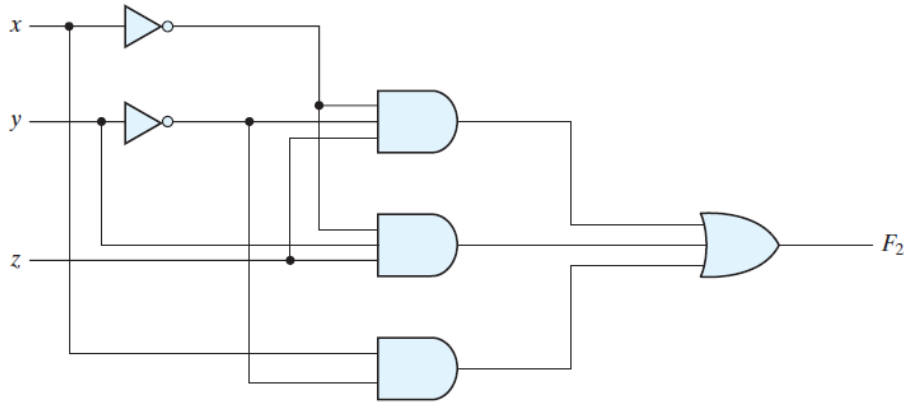


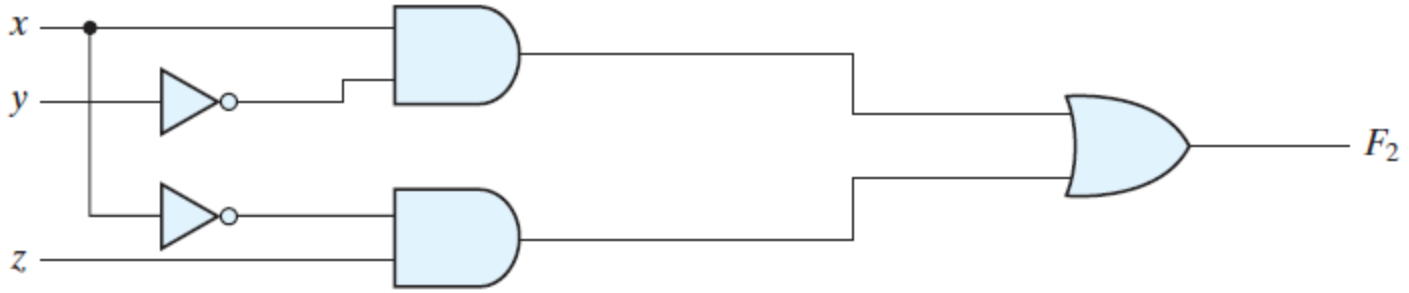
FIGURE 2.1
Gate implementation of $F_1 = x + y'z$

- it is sometimes possible to obtain a simpler expression for the same function and thus reduce the number of gates in the circuit and the number of inputs to the gate. Designers are motivated to reduce the complexity and number of gates because their effort can significantly reduce the cost of a circuit

- $F2 = xyz + xyz + xy$
- $F2 = xyz + xyz + xy = xz(y + y) + xy = xz + xy$



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



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

In general,
there are many equivalent representations of a logic function. Finding the most economic representation of the logic is an important design task.

- Simplify the following boolean expression 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

- 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. The complement of a function may be derived algebraically through DeMorgan's theorems

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

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

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

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:

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

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

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

Minterm

- Each minterm is obtained from an AND term of the *n* variables, with each variable being primed if the corresponding bit of the binary number is a 0 and unprimed if a 1.
- A symbol for each minterm is also shown in the table and is of the form *mj*, where the subscript *j* denotes the decimal equivalent of the binary number of the minterm designated.
- Standard Product

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

Sum of Minterms(Sum of Products)

- **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.**

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

$f1 = x'y'z + xy'z' + xyz = m1 + m4 + m7 = \sum m(1,4,7)$

$f2 = x'yz + xy'z + xyz' + xyz = m3 + m5 + m6 + m7 = \sum m(3,5,6,7)$

Max term(Standard Sum)

- 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 a 1, and
- each maxterm is the complement of its corresponding minterm and vice versa.(Apply Demorgans theorem)

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

Product of Maxterms(Product of Sums)

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

$f1 = xyz + xyz + xyz = m1 + m4 + m7 = \sum m(1,4,7)$

$f2 = xyz + xyz + xyz + xyz = m3 + m5 + m6 + m7 = \sum m(3,5,6,7)$

Write f1'

Write (f1')'

- f1=(x+y+z)(x+y'+z)(x+y'+z')(x'+y+z')(x'+y'+z')=M0M2M3M5M6 =πM(0,2,3,5,6)
- f2=(x+y+z)(x+y+z')(x+y'+z)(x'+y+z)=M0M1M2M4=πM(0,1,2,4)
- POS is the complement of SOP

Canonical Form

- A boolean expression expressed as a sum of minterms(SOP) or product of maxterms(POS) are said to be in canonical form if
- All the variables will be present in primed or unprimed fashion

Standard form

- The terms in the function contain one, two or any number of literals
- 2 types
  - Sum of products form
  - Product of sum form
- F1= x+yz
- F2=(x+y)(x'+y+z)

Non standard form

- Neither in sop or pos form
- F= (x+yz)(yz)

Problems

- Express the given boolean function in sum of minterms form
- Express the given boolean function in product of maxterms form
- Conversion between canonical forms

Karnaugh Map(K Map)

- Two variable K Map

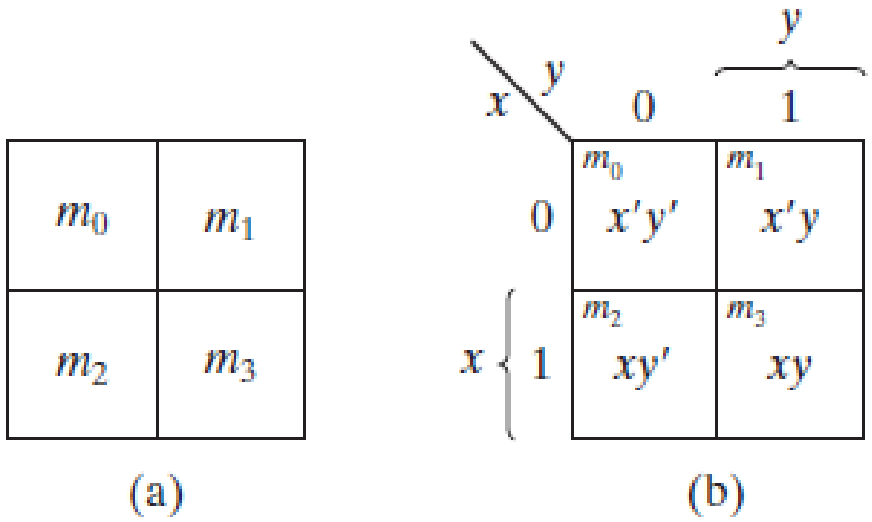
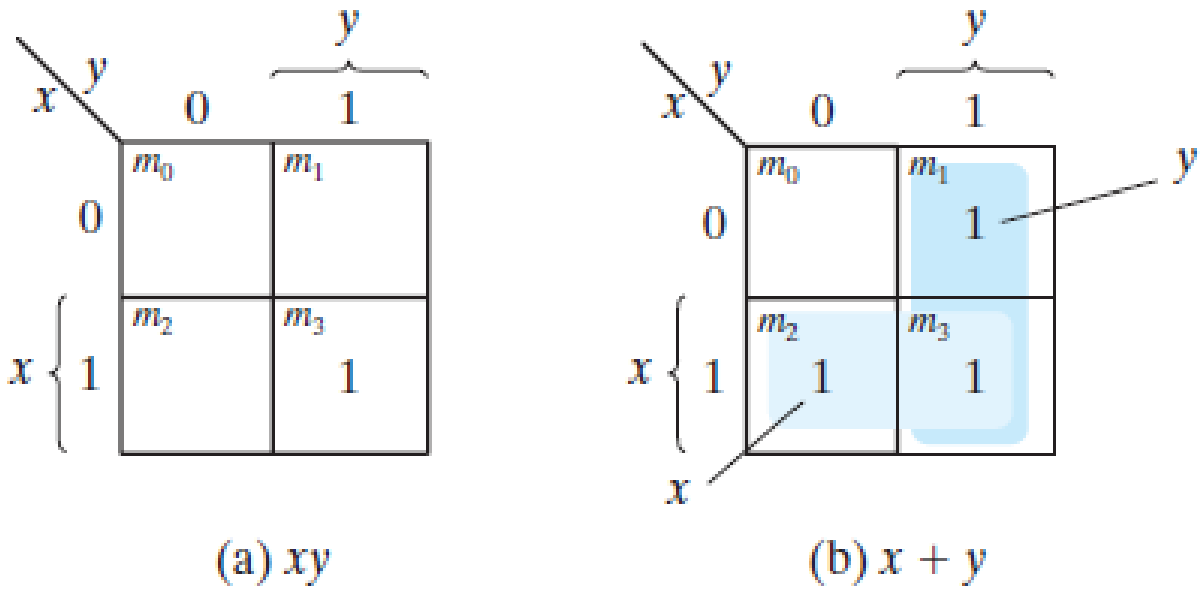


FIGURE 3.1
Two-variable K-map



(a) xy (b) $x + y$

FIGURE 3.2
Representation of functions in the map

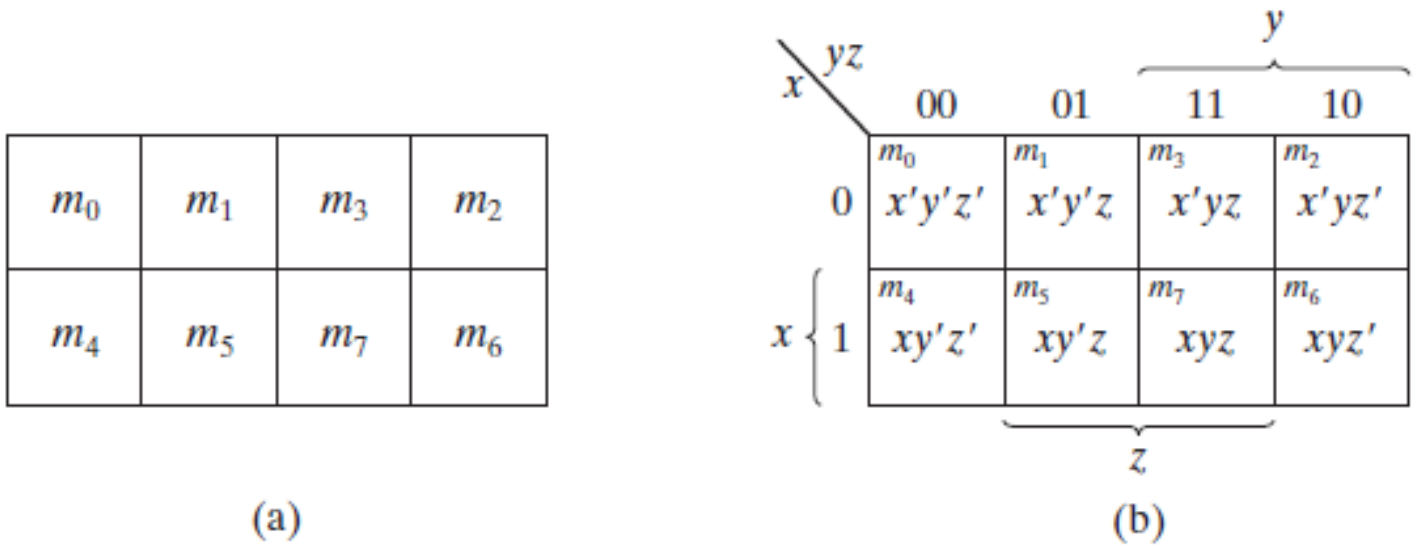


FIGURE 3.3
Three-variable K-map

Simplify the Boolean function

$F(x, y, z) = \sum m(2, 3, 4, 5)$

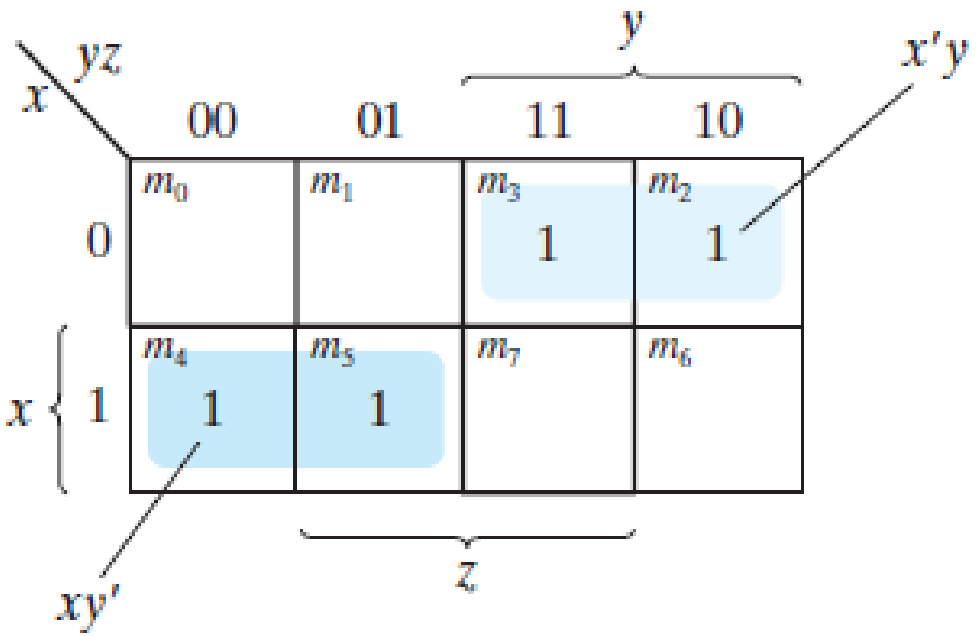


FIGURE 3.4

Map for Example 3.1, $F(x, y, z) = \Sigma(2, 3, 4, 5) = x'y + xy'$

Simplify the Boolean function

$F(x, y, z) = \sum m(3, 4, 6, 7)$

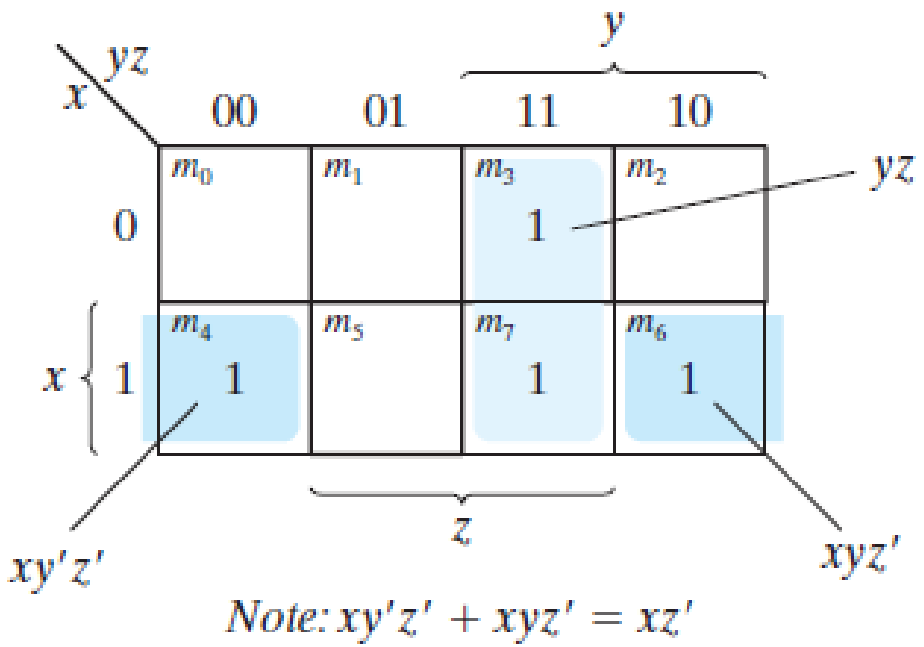


FIGURE 3.5

Map for Example 3.2, $F(x, y, z) = \Sigma(3, 4, 6, 7) = yz + xz'$

Simplify the Boolean function

$F(x, y, z) = \sum m(0, 2, 4, 5, 6)$

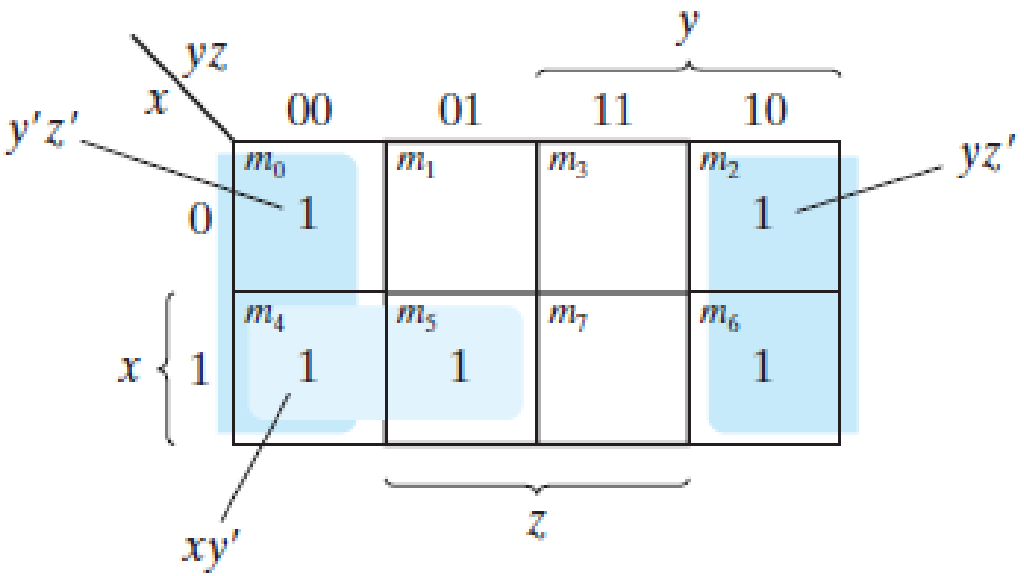


FIGURE 3.6

Map for Example 3.3, $F(x, y, z) = \Sigma(0, 2, 4, 5, 6) = z' + xy'$

For the Boolean function

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

(a) Express this function as a sum of minterms.

(b) Find the minimal sum-of-products expression.

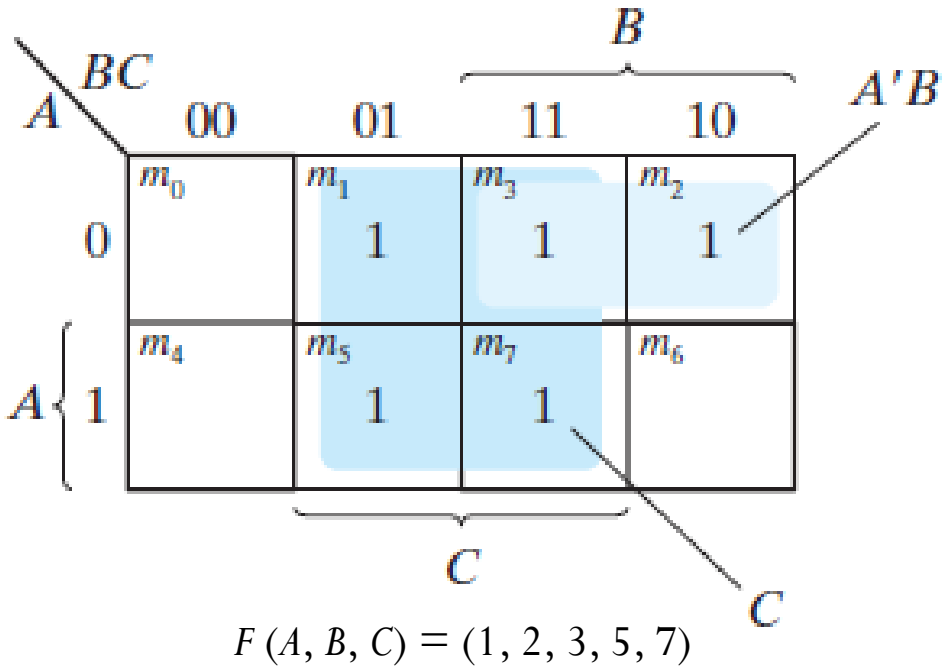


FIGURE 3.7

Map of Example 3.4, $A'C + A'B + AB'C + BC = C + A'B$

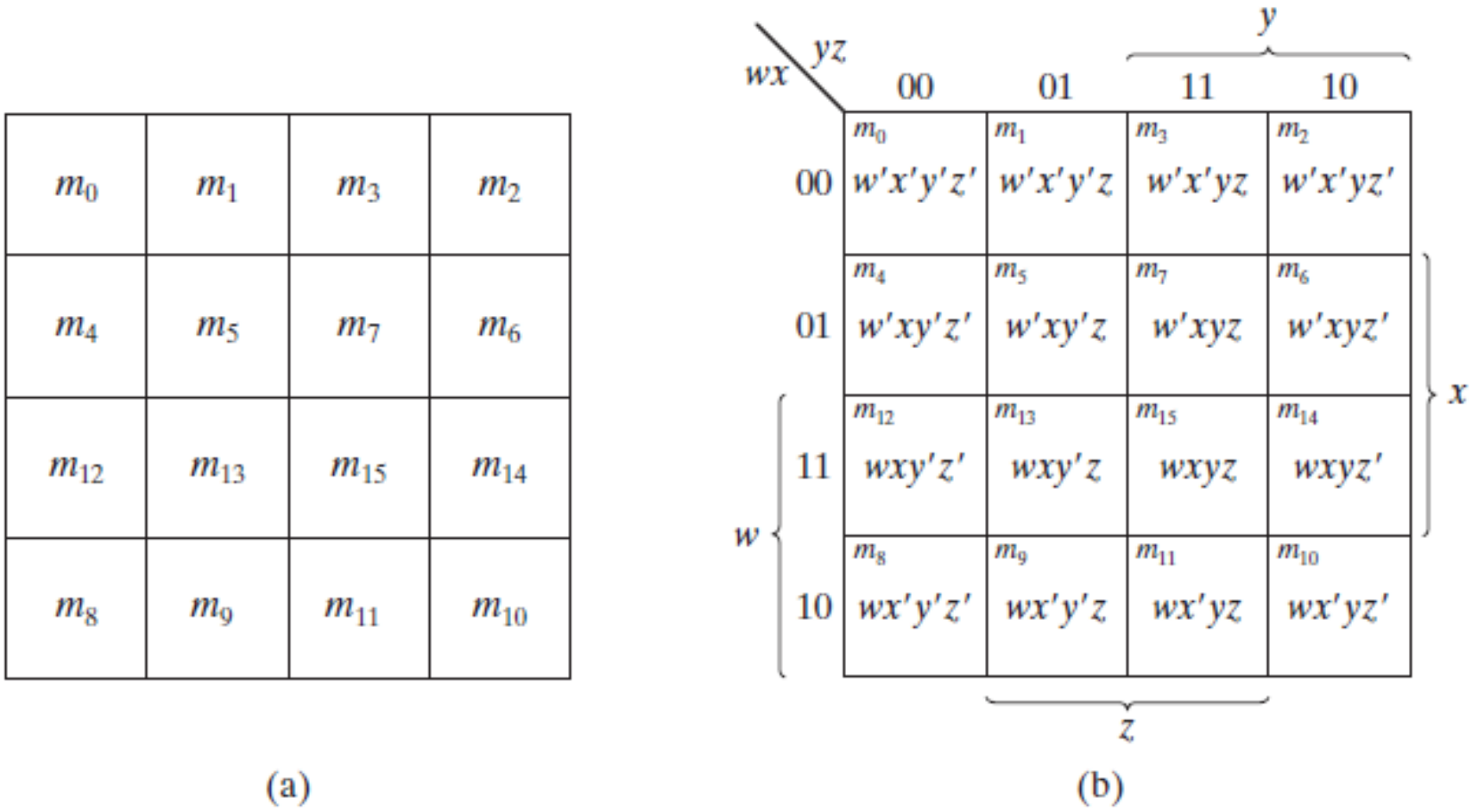


FIGURE 3.8
Four-variable map

Simplify the Boolean function

$F(w, x, y, z) = \sum m(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

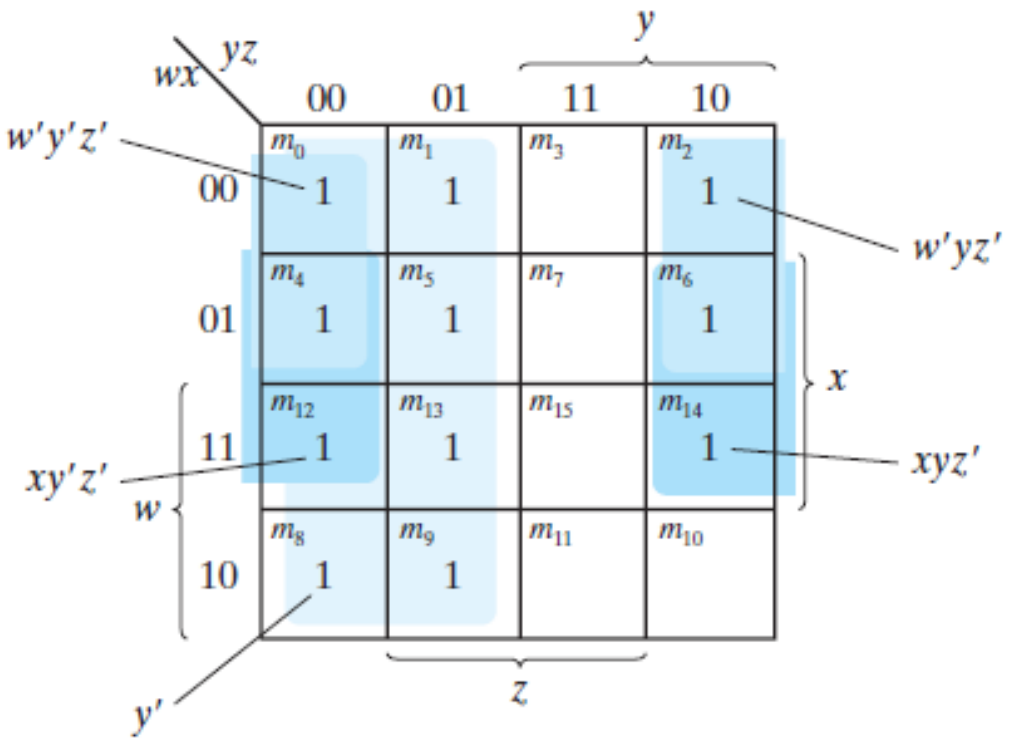


FIGURE 3.9

Map for Example 3.5, $F(w, x, y, z) = \Sigma(0,1, 2, 4, 5, 6, 8, 9, 12, 13, 14) = y' + w'z' + xz'$

Simplify the Boolean function

$F = A'B'C' + B'CD' + A'BCD' + AB'C'$

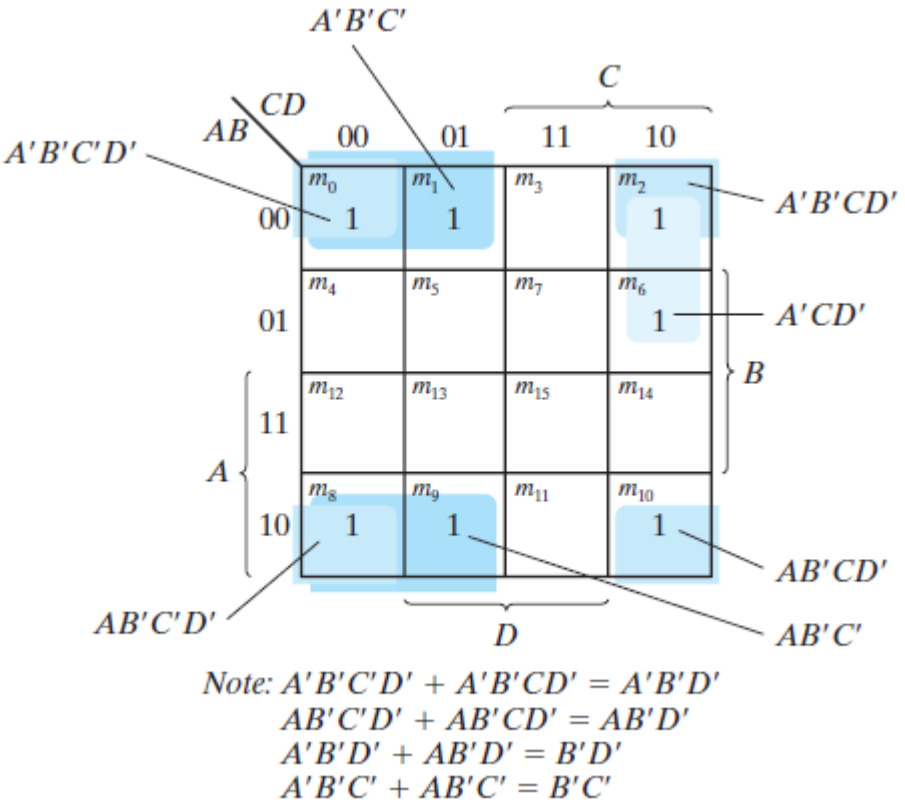


FIGURE 3.10
Map for Example 3.6, $A'B'C' + B'CD' + A'BCD' + AB'C' = B'D' + B'C' + A'CD'$

Implicant, Prime Implicant and Essential prime implicant

- Implicant- A product term is the implicant of the function
- Prime implicant- It is a product term which is obtained by combining the maximum possible number of squares in the map
- Essential prime implicant-If a min term in a square is covered by only one prime implicant, that prime implicant is said to be essential.

$F(A, B, C, D) = (0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$

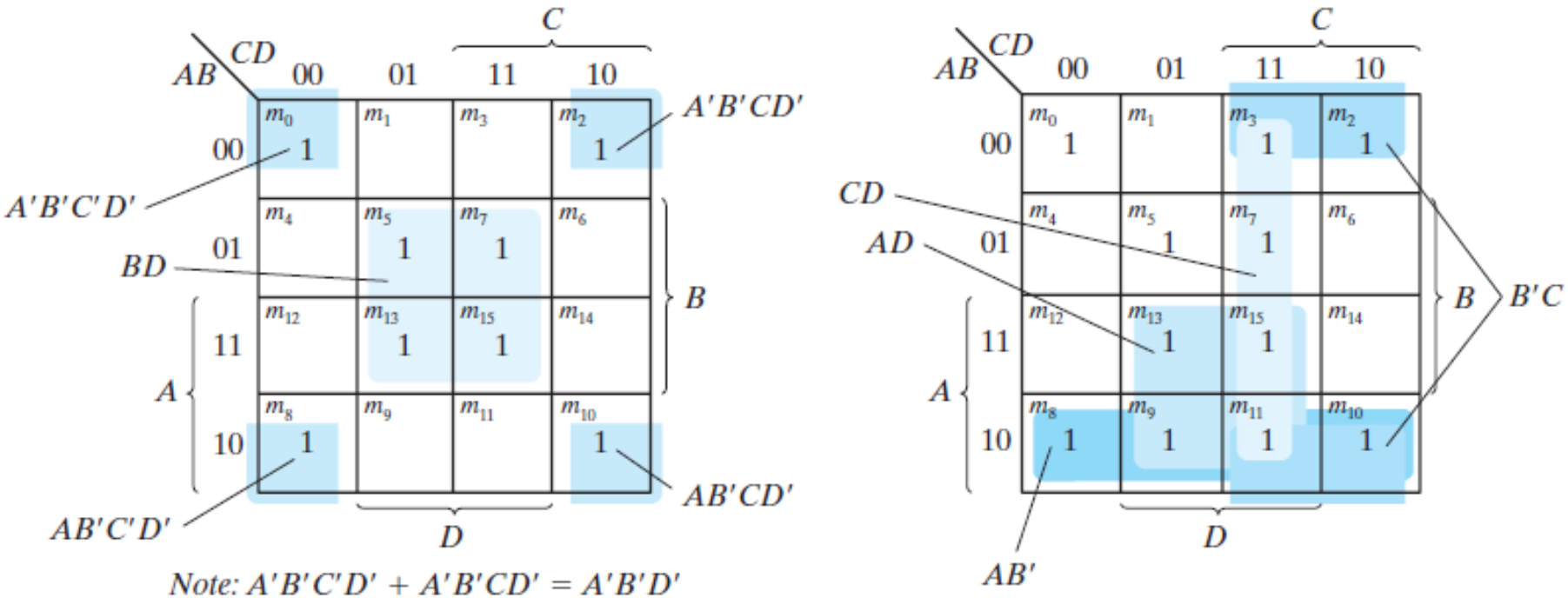
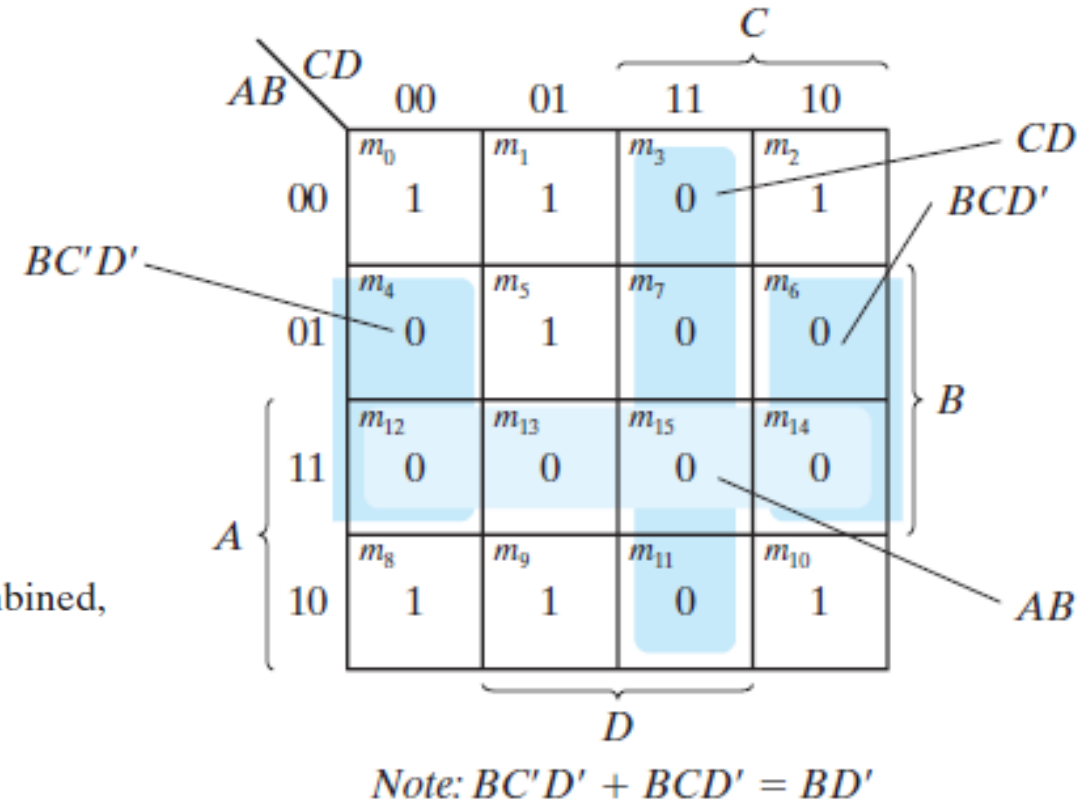


FIGURE 3.11
Simplification using prime implicants

PRODUCT OF SUM FORM

Simplify the following Boolean function into (a) sum-of-products form and (b) product-of-sums form: $F(A, B, C, D) = (0, 1, 2, 5, 8, 9, 10)$



(a) $F = B'D' + B'C' + A'C'D$

If the squares marked with 0's are combined,

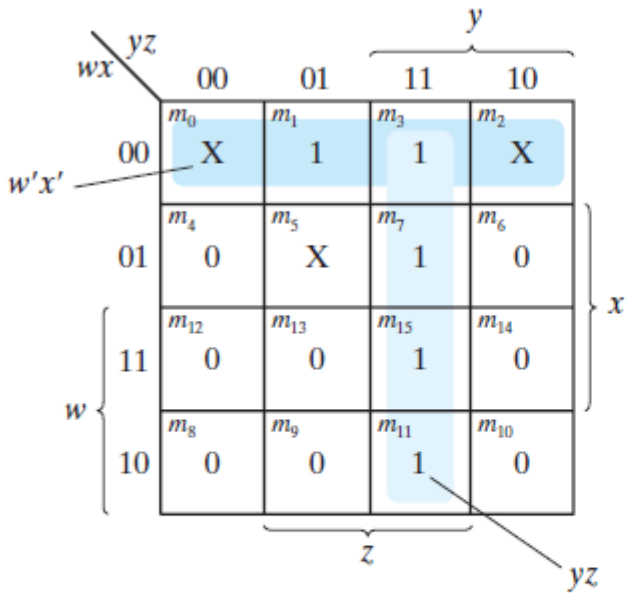
$F' = AB + CD + BD'$

Applying DeMorgan's theorem

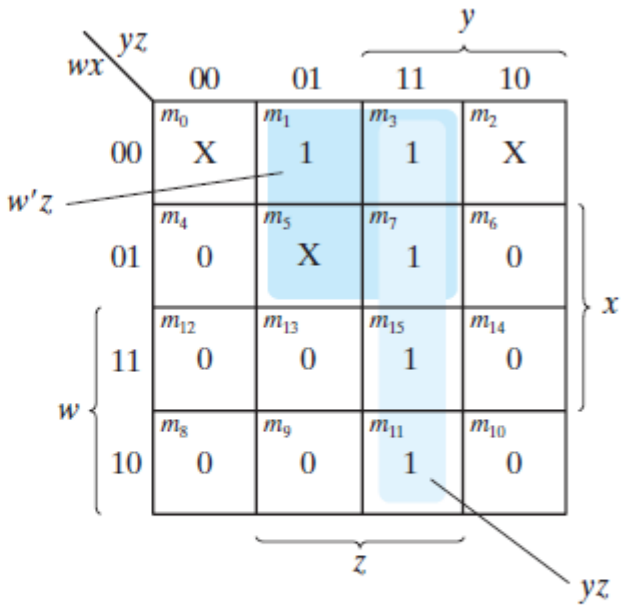
(b) $F = (A' + B')(C' + D')(B' + D)$

DON'T CARE CONDITIONS

- Simplify the Boolean function F $(w, x, y, z) = (1, 3, 7, 11, 15)$ which has the don't-care conditions $d\,(w, x, y, z) = (0, 2, 5)$



(a) $F = yz + w'x'$



(b) $F = yz + w'z$

FIGURE 3.15
Example with don't-care conditions

product-of-sums expression

$F' = z' + wy'$

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

- Simplify $F(w,x,y,z)=\sum(4,5,6,7,12)$;$d=\sum(0,8,13)$
- Soln: W'X+Y'Z'
- or
- W'X+XY'

NAND & NOR Implementation

- Easier to fabricate
- NAND-
- Universal gate

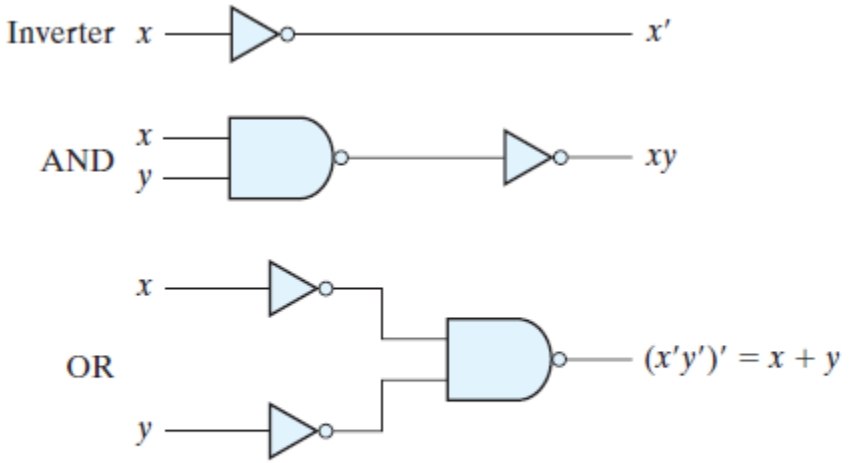


FIGURE 3.16
Logic operations with NAND gates

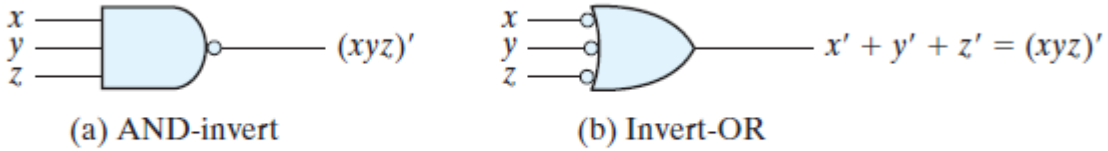


FIGURE 3.17
Two graphic symbols for a three-input NAND gate

$F = AB + CD$

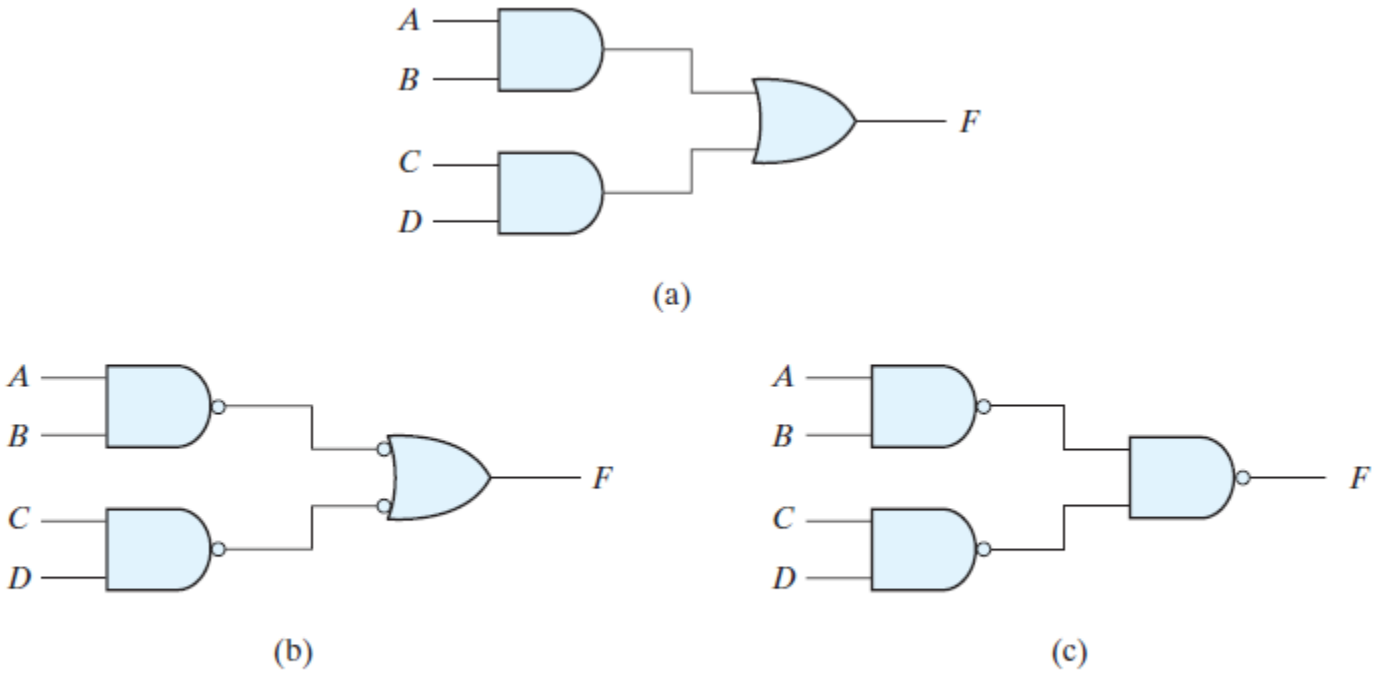
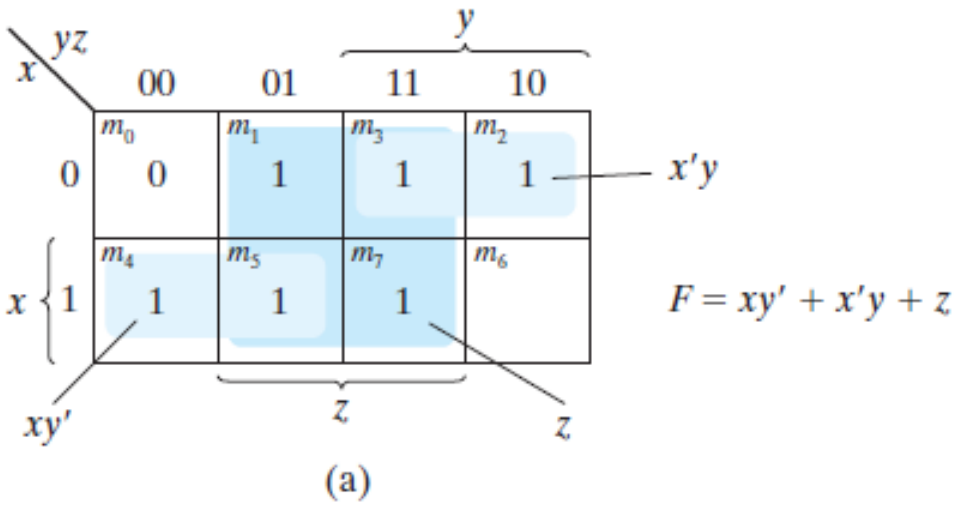


FIGURE 3.18
Three ways to implement $F = AB + CD$

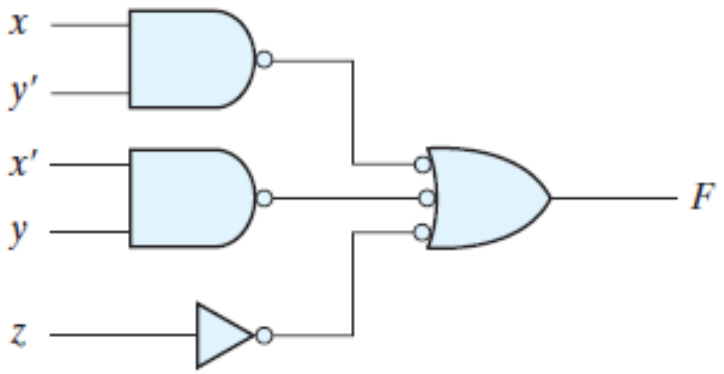
Implement the following Boolean function with NAND gates:

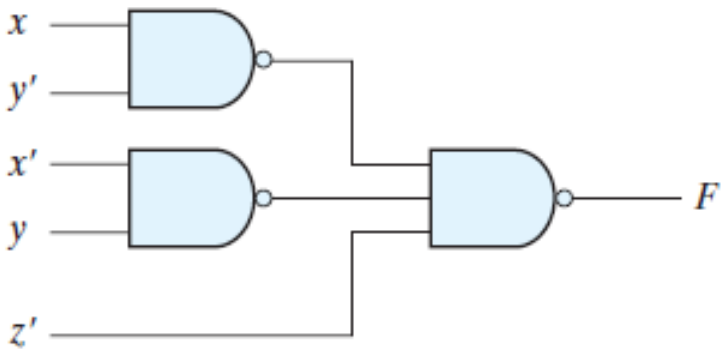
$F(x, y, z) = \sum(1, 2, 3, 4, 5, 7)$

$$F = xy' + x'y + z$$



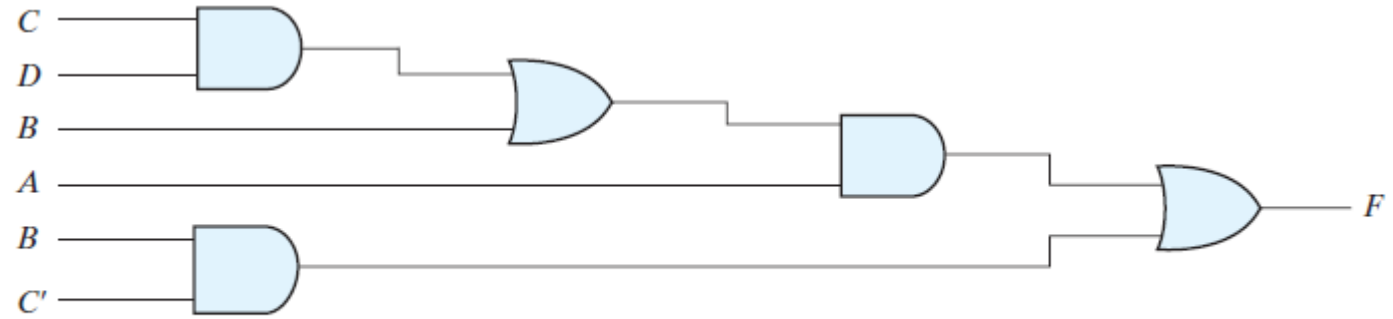
(a)



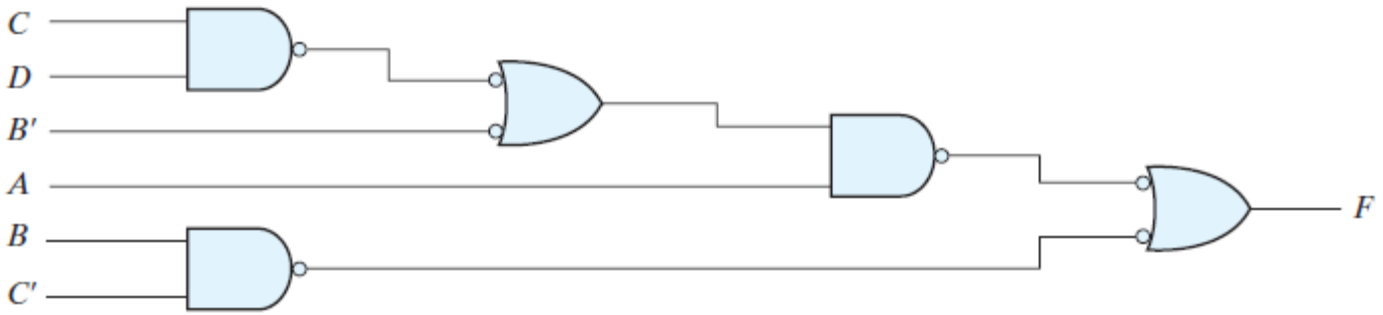


Multilevel NAND Circuits

$F = A\,(CD + B) + BC'$

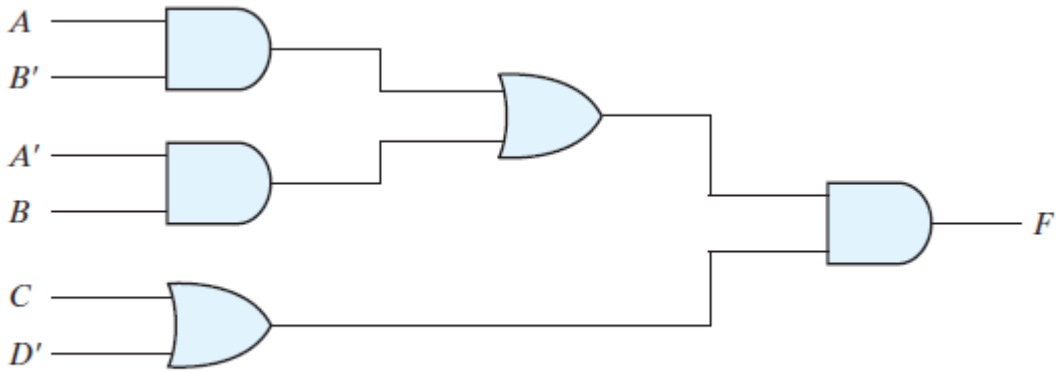


(a) AND–OR gates

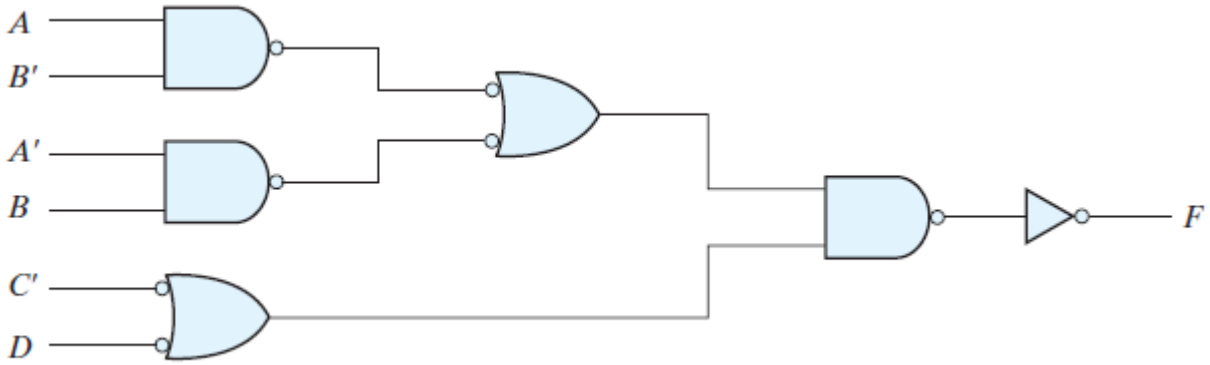


(b) NAND gates

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



(a) AND–OR gates



(b) NAND gates

NOR Implementation

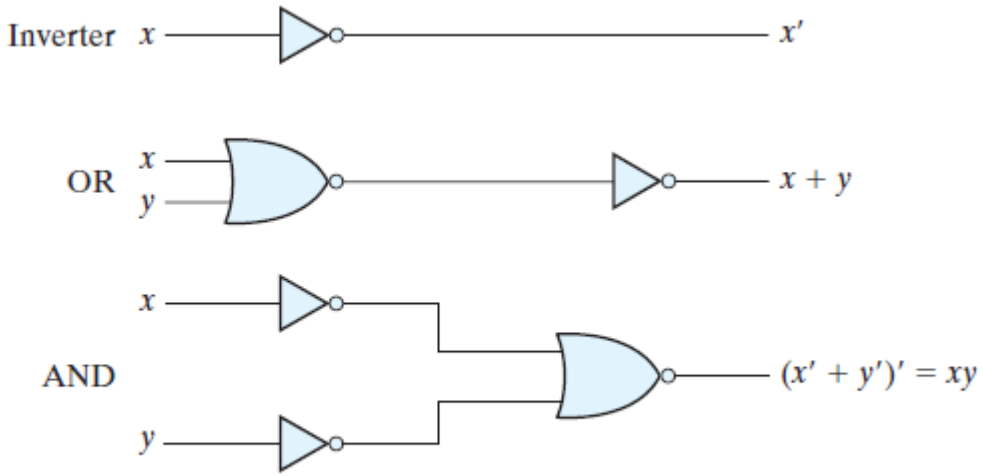


FIGURE 3.22
Logic operations with NOR gates

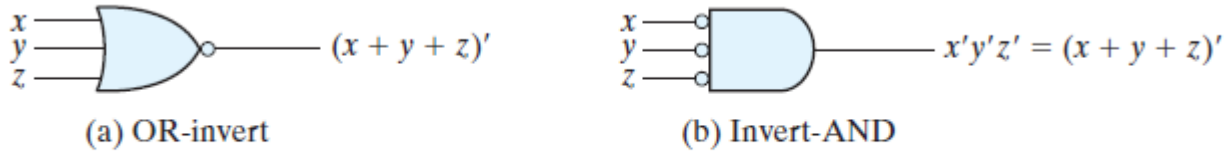


FIGURE 3.23
Two graphic symbols for the NOR gate

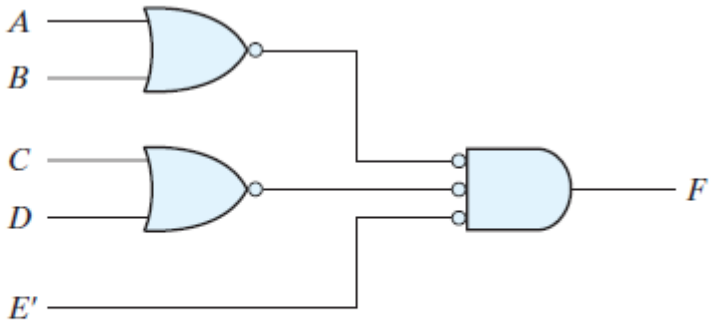


FIGURE 3.24
Implementing $F = (A + B)(C + D)E$

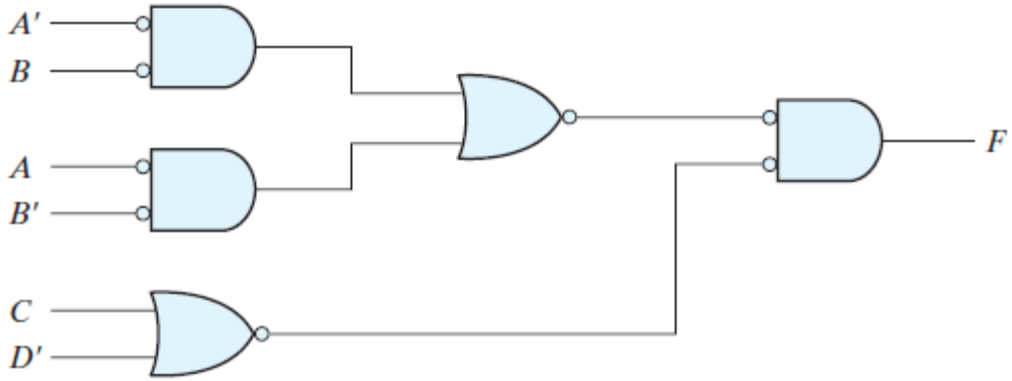


FIGURE 3.25
Implementing $F = (AB' + A'B)(C + D')$ with NOR gates