

# **STLD - Module 2**

# Basic Postulates of a mathematical 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  $N = \{1, 2, 3, 4, \dots\}$  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  $(-1) \notin N$ .
2. *Associative law.* A binary operator  $*$  on a set  $S$  is said to be associative whenever

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

3. *Commutative law.* A binary operator  $*$  on a set  $S$  is said to be commutative whenever

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

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  $I = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$ , since

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

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

# Basic Postulates of a mathematical system

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

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

# Boolean algebra

- In 1854, **George Boole** developed an algebraic system now called Boolean algebra.
- In 1938, **Claude E. Shannon** introduced a two-valued Boolean algebra, i.e., a Boolean algebra with only two elements called **switching algebra** that represented the properties of bistable electrical switching circuits.
- For the formal definition of Boolean algebra, we shall employ the postulates formulated by **E. V. Huntington** in 1904.

# 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:

- The structure is closed with respect to the operator  $+$ .
  - The structure is closed with respect to the operator  $\cdot$ .
- The element 0 is an identity element with respect to  $+$ ; that is,  $x + 0 = 0 + x = x$ .
  - The element 1 is an identity element with respect to  $\cdot$ ; that is,  $x \cdot 1 = 1 \cdot x = x$ .
- The structure is commutative with respect to  $+$ ; that is,  $x + y = y + x$ .
  - The structure is commutative with respect to  $\cdot$ ; that is,  $x \cdot y = y \cdot x$ .
- The operator  $\cdot$  is distributive over  $+$ ; that is,  $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$ .
  - The operator  $+$  is distributive over  $\cdot$ ; that is,  $x + (y \cdot z) = (x + y) \cdot (x + z)$ .
- 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)  $x \cdot x' = 0$ .
- There exist at least two elements  $x, y \in B$  such that  $x \neq y$ .

# Boolean algebra

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  $\cdot$  (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.

# Two-Valued Boolean Algebra (Switching Algebra/ Binary Logic)

A two-valued Boolean algebra is defined on a set of two elements,  $B = \{0, 1\}$ , with rules for the two binary operators  $+$  and  $\cdot$  as shown in the following operator tables (the rule for the complement operator is for verification of postulate 5):

$x$	$y$	$x \cdot y$
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

- ✓ These rules are exactly the same as the **AND**, **OR**, and **NOT** operations, respectively

# Switching Algebra - Verification of Huntington postulates

1. 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  $1, 0 \in B$ .

2. From the tables, we see that

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

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

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.

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).



# Switching Algebra - Verification of Huntington postulates

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  $1 \cdot 1' = 1 \cdot 0 = 0$ .

Thus, postulate 1 is verified.

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

# 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)	$x(x + y) = x$

# Theorems 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)

Note that theorem 1(b) is the dual of theorem 1(a) and that each step of the proof in part (b) is the dual of its counterpart in part (a). Any dual theorem can be similarly derived from the proof of its corresponding theorem.

# Theorems 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  $(x')'$ .

# Theorems of Boolean Algebra

**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.

The theorems of Boolean algebra can be proven by means of truth tables. In truth tables, both sides of the relation are checked to see whether they yield identical results for all possible combinations of the variables involved. The following truth table verifies the first absorption theorem:

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

# Theorems 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. For example, the truth table for the first DeMorgan's theorem,  $(x + y)' = x'y'$ , is as follows:

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

$x'$	$y'$	$x'y'$
1	1	1
1	0	0
0	1	0
0	0	0

# Operator Precedence

**(1) parentheses**

**(2) NOT**

**(3) AND**

**(4) OR**

# Boolean Functions

- ✓ Boolean algebra is an algebra that deals with binary variables and logic operations.
- ✓ 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 a truth table. The number of rows in the truth table is  $2^n$ , 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  $2^n - 1$ .
- ✓ A Boolean function can be transformed from an algebraic expression into a circuit diagram (also called a schematic) composed of logic gates connected in a particular structure.



# Boolean Functions Truth Table - Example

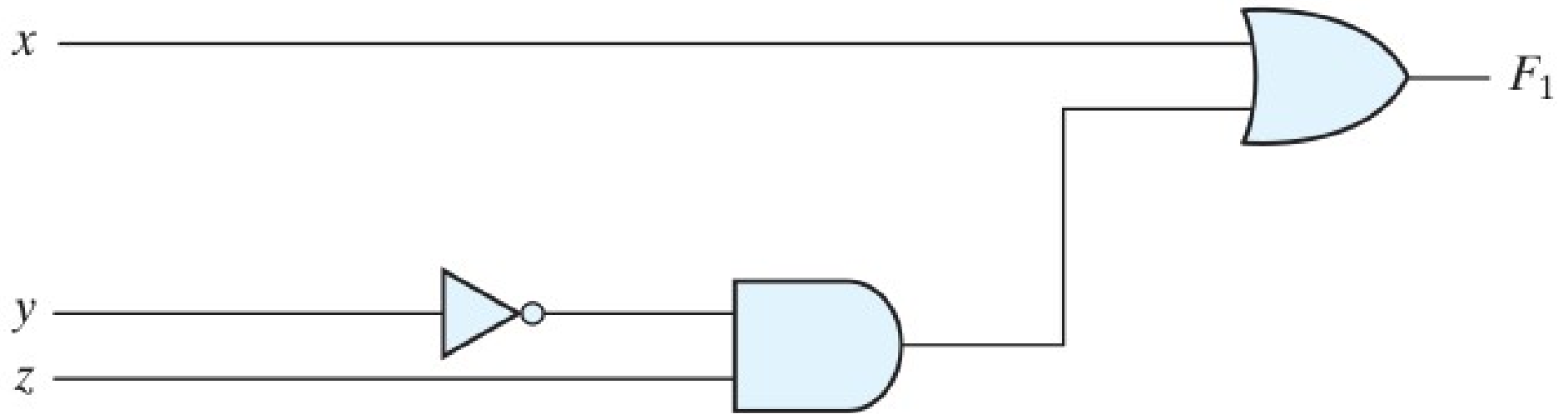
$F1 = x + y'z$  &  $F2 = x'y'z + x'yz + xy'$

$x$	$y$	$z$	$F_1$	$F_2$
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

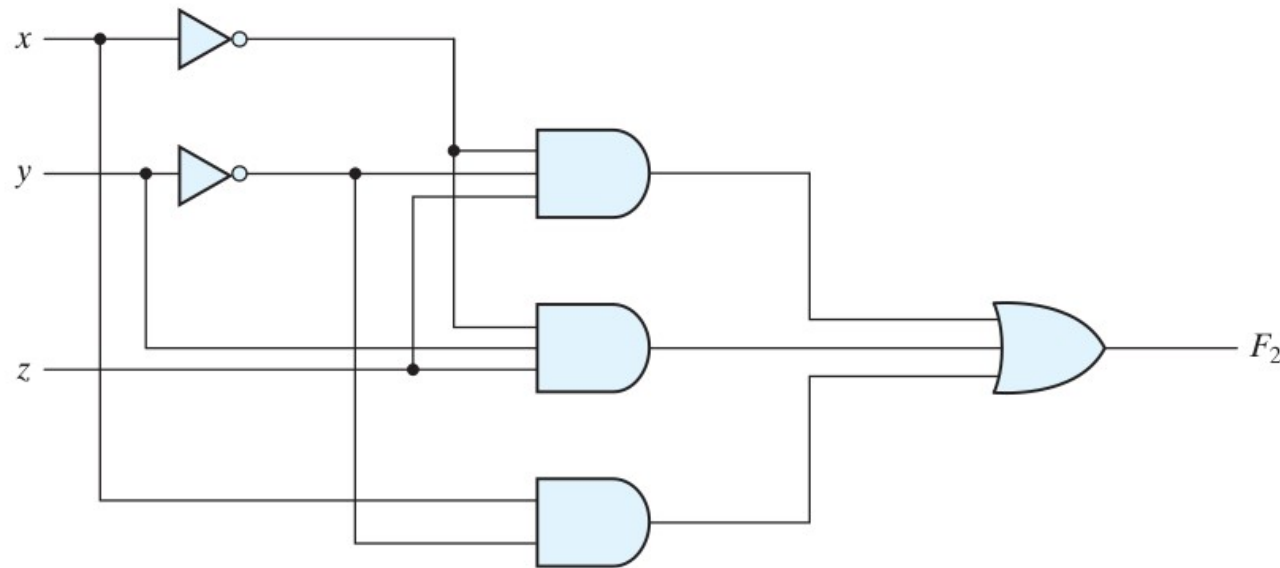
Function F2 can be simplified as:

$$\begin{aligned} F2 &= x'y'z + x'yz + xy' \\ &= x'z(y' + y) + xy' \\ &= x'z + xy' \end{aligned}$$

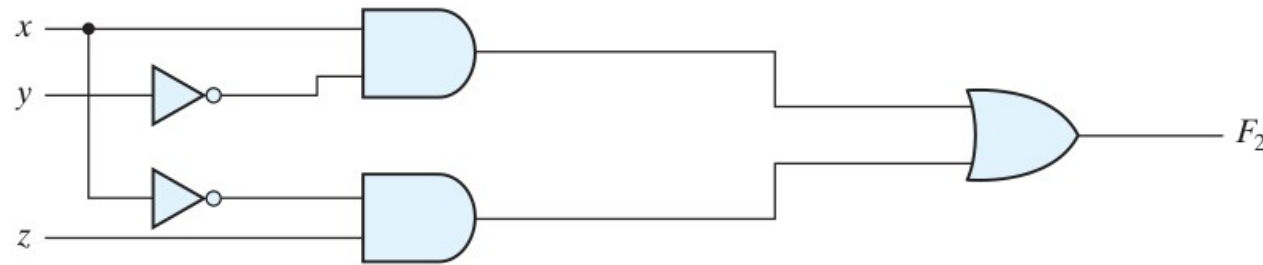
# Gate implementation of $F_1 = x + y'z$



# Implementation of Boolean function F2 with gates



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



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

# 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

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

## Generalized DeMorgan's Theorem

$$\begin{aligned}(A + B + C + D + \cdots + F)' &= A'B'C'D' \dots F' \\ (ABCD \dots F)' &= A' + B' + C' + D' + \cdots + F'\end{aligned}$$

The generalized form of DeMorgan's theorems states that the complement of a function is obtained by interchanging AND and OR operators and complementing each literal.

# Find the complement of the functions

## $F1 = x'yz' + x'y'z$ and $F2 = x(y'z' + yz)$

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

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

**Note:** A simpler procedure for deriving the complement of a function is to take the dual of the function and complement each literal. This method follows from the generalized forms of DeMorgan's theorems. Remember that the **dual of a function** is obtained from the interchange of AND and OR operators and 1's and 0's.

## Find the complement of the functions $F_1$ and $F_2$ 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 form

- Boolean functions expressed as a **sum of minterms** or **product of maxterms** are said to be in **canonical form**.
- A Boolean function can be expressed algebraically from a given truth table by forming a **minterm (maxterm)** for each combination of the variables that produces a **1(0)** in the function and then taking the **OR(AND)** of all those terms.
- A binary variable may appear either in its normal form ( $x$ ) or in its complement (primed) form ( $x'$ ).
- Now consider two binary variables  $x$  and  $y$  combined with an AND operation. Since each variable may appear in either form, there are four possible combinations:  $x'y'$ ,  $x'y$ ,  $xy'$ , and  $xy$ . Each of these four AND terms is called a **minterm**, or a **standard product**. In a similar manner,  **$n$  variables** can be combined to form  **$2^n$  minterms**.
- In a similar fashion,  $n$  variables forming an OR term, with each variable being primed or unprimed, provide  $2^n$  possible combinations, called **maxterms, or standard sums**.



# Minterms and Maxterms

## Minterms and Maxterms for Three Binary Variables

<i>x</i>	<i>y</i>	<i>z</i>	Minterms		Maxterms	
			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

## 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

$$f_1 = x'y'z + xy'z' + xyz$$

$$= m_1 + m_4 + m_7$$

$$f_1' = x'y'z' + x'yz' + x'yz + xy'z + xyz'$$

$$f_1 = (x+y+z)(x+y'+z)(x'+y+z)(x'+y'+z)$$

$$= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6$$

$$f_2 = x'yz + xy'z + xyz' + xyz$$

$$= m_3 + m_5 + m_6 + m_7$$

$$f_2 = (x+y+z)(x+y+z')(x+y'+z)(x'+y+z)$$

$$= M_0.M_1.M_2.M_4$$

# Sum of Minterms - Example

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

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

But  $AB'C$  appears twice, and according to theorem 1 ( $x + x = x$ ), it is possible to remove one of those occurrences. Rearranging the minterms in ascending order, we finally obtain

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

An alternative procedure for deriving the minterms of a Boolean function is to obtain the truth table of the function directly from the algebraic expression and then read the minterms from the truth table.

When a Boolean function is in its sum-of-minterms form, it is sometimes convenient to express the function in the following brief notation:

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

# Product of Maxterms - Example

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,

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

Combining all the terms and removing those which appear more than once, we finally obtain

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

A convenient way to express this function is as follows:

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

# Conversion between Canonical Forms

The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function. This is because the original function is expressed by those minterms which make the function equal to 1, whereas its complement is a 1 for those minterms for which the function is a 0. As an example, consider the function

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

This function has a complement that can be expressed as

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

Now, if we take the complement of  $F'$  by DeMorgan's theorem, we obtain  $F$  in a different form:

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

The last conversion follows from the definition of minterms and maxterms as shown in Table 2.3. From the table, it is clear that the following relation holds:

$$m'_j = M_j$$

That is, the **maxterm with subscript  $j$  is a complement of the minterm with the same subscript  $j$  and vice versa.**

# Conversion between Canonical Forms

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

<b>x</b>	<b>y</b>	<b>z</b>	<b>F</b>
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

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

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

# Standard Forms

- In **standard form**, the terms that form the function may contain one, two, or any number of literals (not like minterm or maxterm which contain, by definition, all the variables, either complemented or pure form).
- There are two types of standard forms:

the sum of products and products of sums.

- 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. An example of a function expressed as a sum of products is

$$F1 = y' + xy + x'yz'$$

- The expression has three product terms, with one, two, and three literals. Their sum is, in effect, an OR operation.
- The logic diagram of a sum-of-products expression consists of a group of AND gates followed by a single OR gate. This circuit configuration is referred to as a **two-level implementation**.

# Standard Forms

➤ 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.

➤ An example of a function expressed as a product of sums is

$$F2 = x(y' + z)(x' + y + z')$$

➤ This expression has three sum terms, with one, two, and three literals. The product is an AND operation.

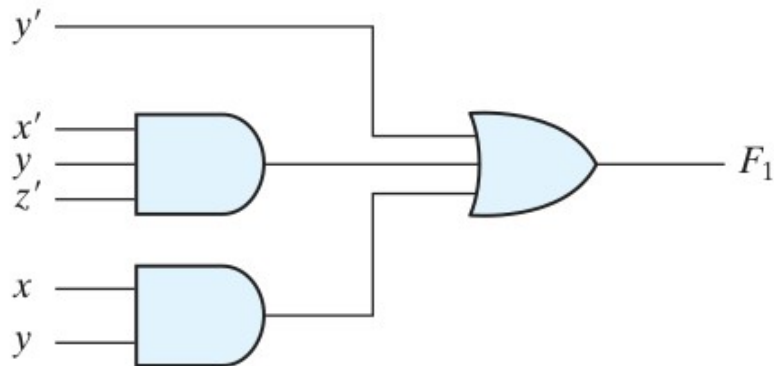
➤ The use of the words product and sum stems from the similarity of the AND operation to the arithmetic product (multiplication) and the similarity of the OR operation to the arithmetic sum (addition).

➤ The gate structure of the product-of-sums expression consists of a group of OR gates for the sum terms (except for a single literal), followed by an AND gate. This standard type of expression results in a two-level structure of gates.

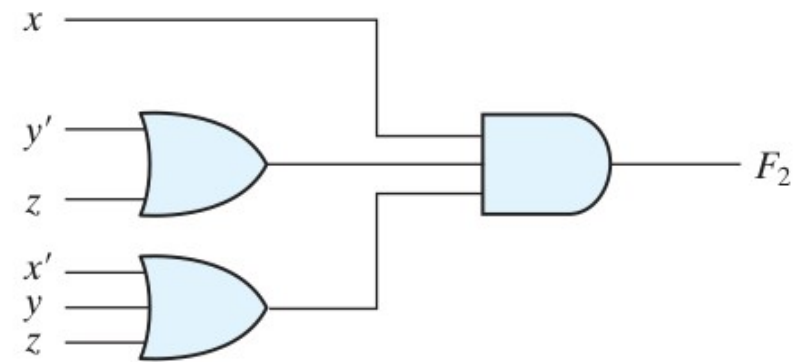


# Standard Forms

## Two-level implementation

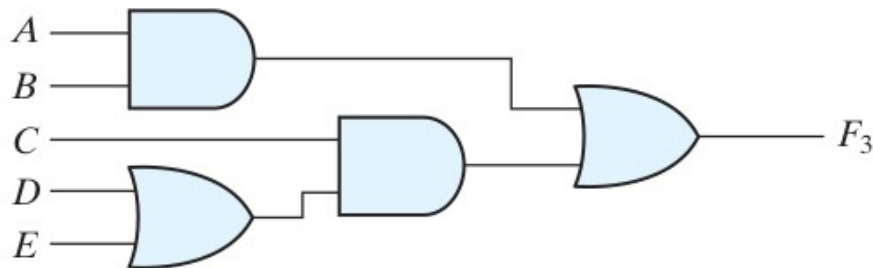


(a) Sum of Products

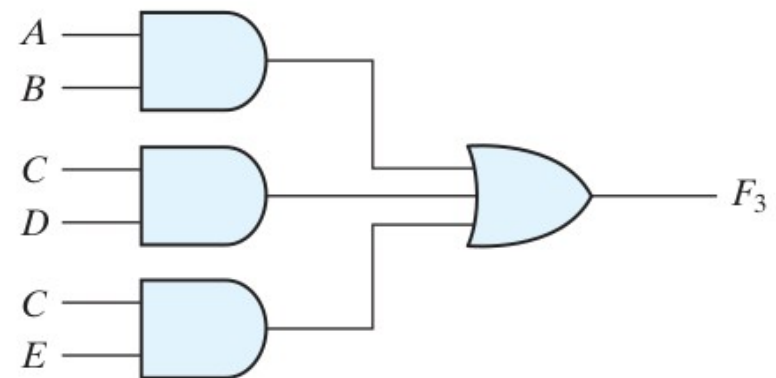


(b) Product of Sums

## Three (nonstandard)- and two-level (standard) implementation



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



(b)  $AB + CD + CE$

# Boolean Functions - other logic operations

There are  $2^{(2^n)}$  functions possible with  $n$  binary variables.

Thus, for two variables,  $n=2$ , and the number of possible Boolean functions is 16.

*Truth Tables for the 16 Functions of Two Binary Variables*

<i>x</i>	<i>y</i>	<i>F</i> <sub>0</sub>	<i>F</i> <sub>1</sub>	<i>F</i> <sub>2</sub>	<i>F</i> <sub>3</sub>	<i>F</i> <sub>4</sub>	<i>F</i> <sub>5</sub>	<i>F</i> <sub>6</sub>	<i>F</i> <sub>7</sub>	<i>F</i> <sub>8</sub>	<i>F</i> <sub>9</sub>	<i>F</i> <sub>10</sub>	<i>F</i> <sub>11</sub>	<i>F</i> <sub>12</sub>	<i>F</i> <sub>13</sub>	<i>F</i> <sub>14</sub>	<i>F</i> <sub>15</sub>
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

The 16 functions listed can be subdivided into three categories:



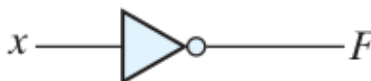

1. Two functions that produce a constant 0 or 1.
2. Four functions with unary operations: complement and transfer.
3. Ten functions with binary operators that define eight different operations: AND, OR, NAND, NOR, exclusive-OR, equivalence, inhibition, and implication.

# Boolean Expressions for the 16 Functions of Two Variables

*Boolean Expressions for the 16 Functions of Two Variables*

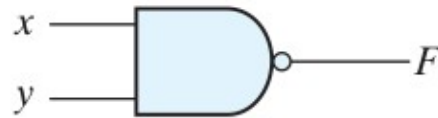
Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	$x$ and $y$
$F_2 = xy'$	$x/y$	Inhibition	$x$ , but not $y$
$F_3 = x$		Transfer	$x$
$F_4 = x'y$	$y/x$	Inhibition	$y$ , but not $x$
$F_5 = y$		Transfer	$y$
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	$x$ 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

Name	Graphic symbol	Algebraic function	Truth table															
AND		$F = x \cdot y$	<table><tr><th><math>x</math></th><th><math>y</math></th><th><math>F</math></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><math>x</math></th><th><math>y</math></th><th><math>F</math></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><math>x</math></th><th><math>F</math></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><math>x</math></th><th><math>F</math></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

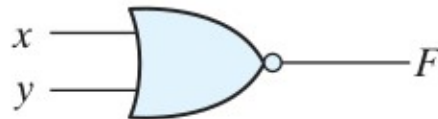
NAND



$$F = (xy)'$$

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

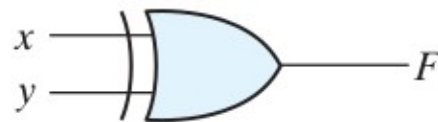
NOR



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

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

Exclusive-OR  
(XOR)



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

$$= x \oplus y$$

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

Exclusive-NOR  
or  
equivalence



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

$$= (x \oplus y)'$$

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

# Gate-Level Minimization

# Karnaugh map or K-map

- Provides a simple, straightforward procedure for minimizing Boolean functions.
- This method may be regarded as a pictorial form of a truth table.
- A K-map is a diagram made up of squares, with each square representing one minterm of the function that is to be minimized.
- The map presents a visual diagram of all possible ways a function may be expressed in standard form.
- By recognizing various patterns, the user can derive alternative algebraic expressions for the same function, from which the simplest can be selected.
- The simplified expressions produced by the map are always in one of the two standard forms: sum of products or product of sums.
- It will be assumed that the **simplest algebraic expression** is an algebraic expression with a minimum number of terms and with the smallest possible number of literals in each term.
- This expression produces a **circuit diagram with a minimum** number of gates and the minimum number of inputs to each gate.
- The **simplest expression is not unique**: It is sometimes possible to find two or more expressions that satisfy the minimization criteria. In that case, either solution is satisfactory.

# Two-Variable K-Map

$m_0$	$m_1$
$m_2$	$m_3$

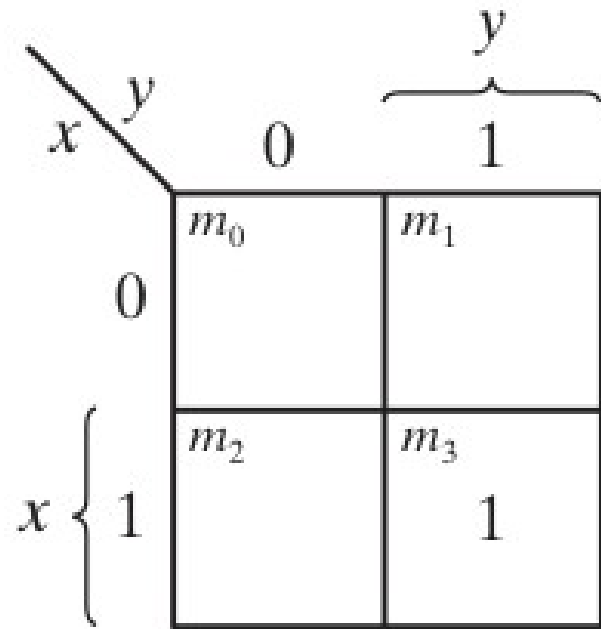
(a)

		$y$	
		$\underbrace{\hspace{1.5cm}}$	
		0	1
$x$	0	$m_0$ $x'y'$	$m_1$ $x'y$
	1	$m_2$ $xy'$	$m_3$ $xy$

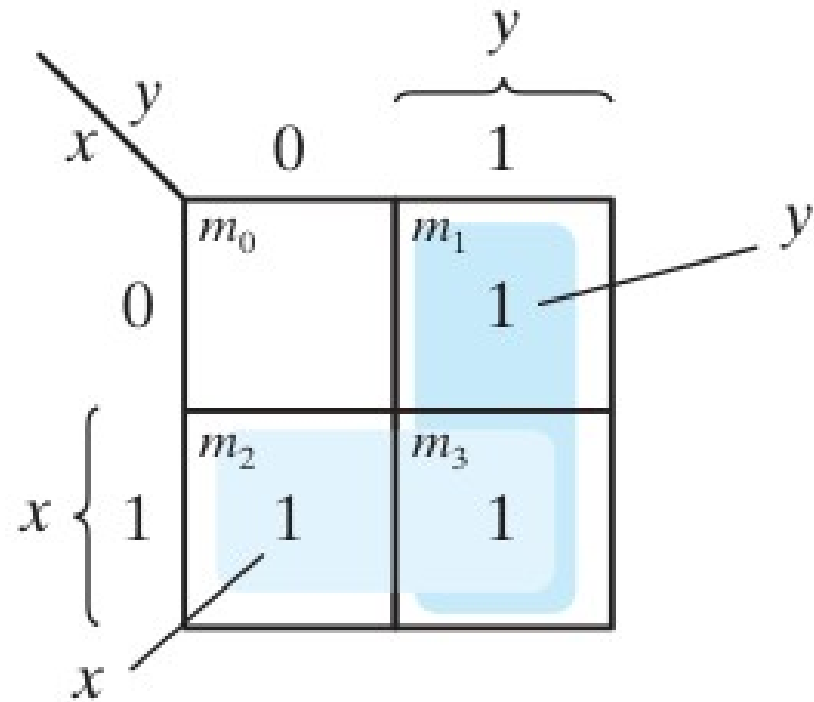
(b)



# Representation of functions in the map



(a)  $xy$



(b)  $x + y$

$$m_1 + m_2 + m_3 = x'y + xy' + xy = x + y$$

# Three-Variable K-Map

$m_0$	$m_1$	$m_3$	$m_2$
$m_4$	$m_5$	$m_7$	$m_6$

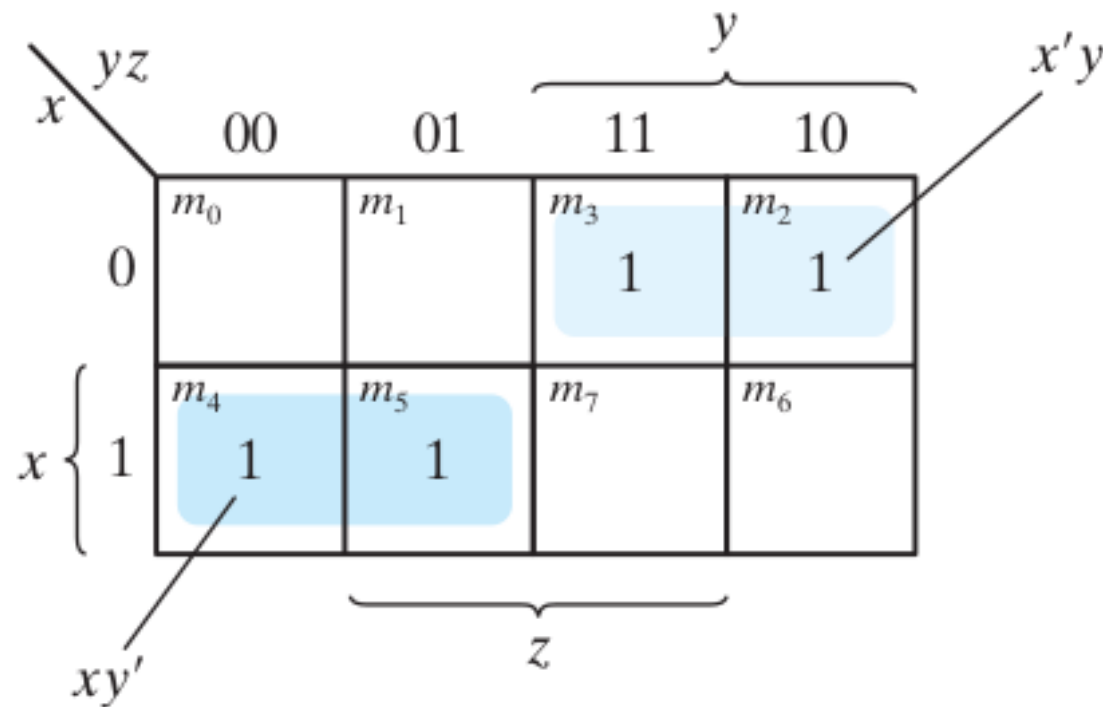
(a)

		$y$			
		$yz$	00	01	11
$x$	0	$m_0$ $x'y'z'$	$m_1$ $x'y'z$	$m_3$ $x'yz$	$m_2$ $x'yz'$
	1	$m_4$ $xy'z'$	$m_5$ $xy'z$	$m_7$ $xyz$	$m_6$ $xyz'$
		$z$			

(b)

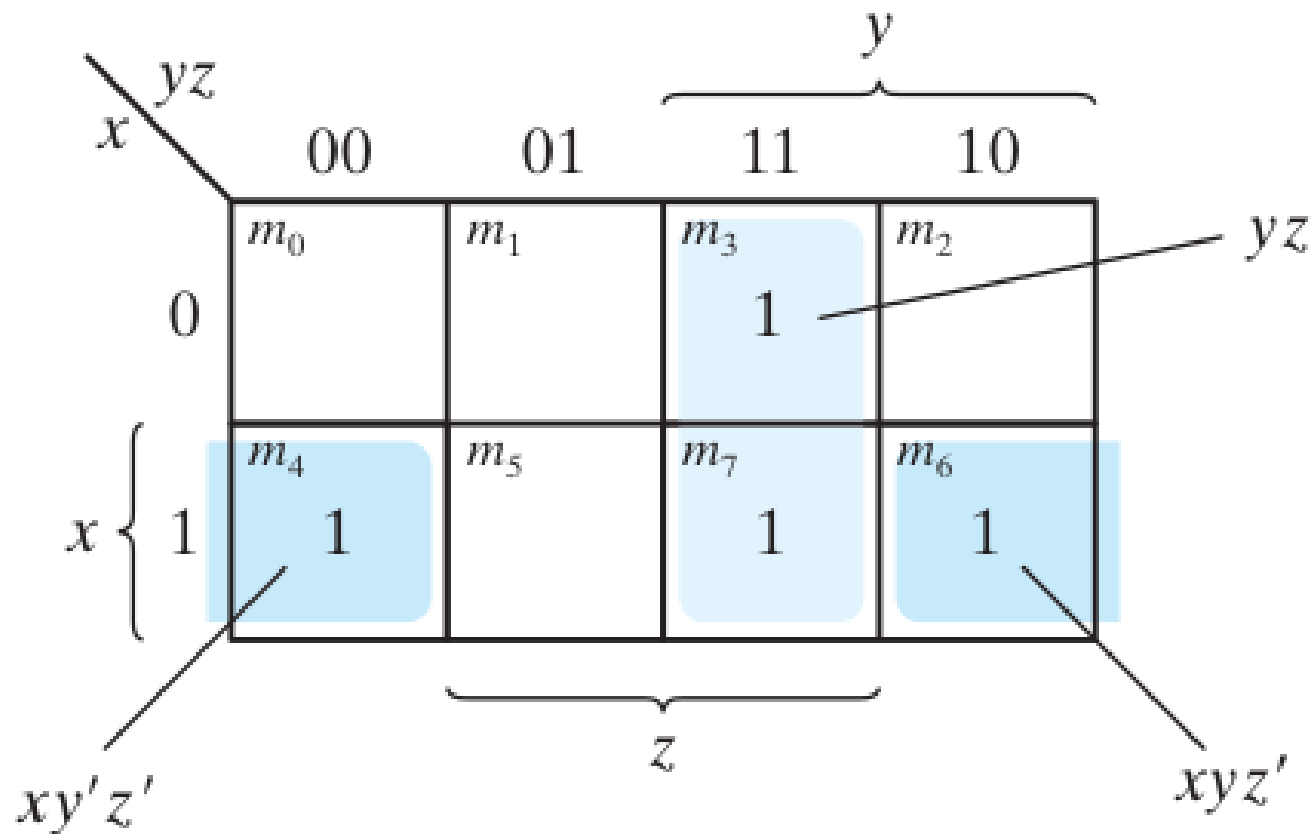
# Simplify the Boolean function

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



# Simplify the Boolean function

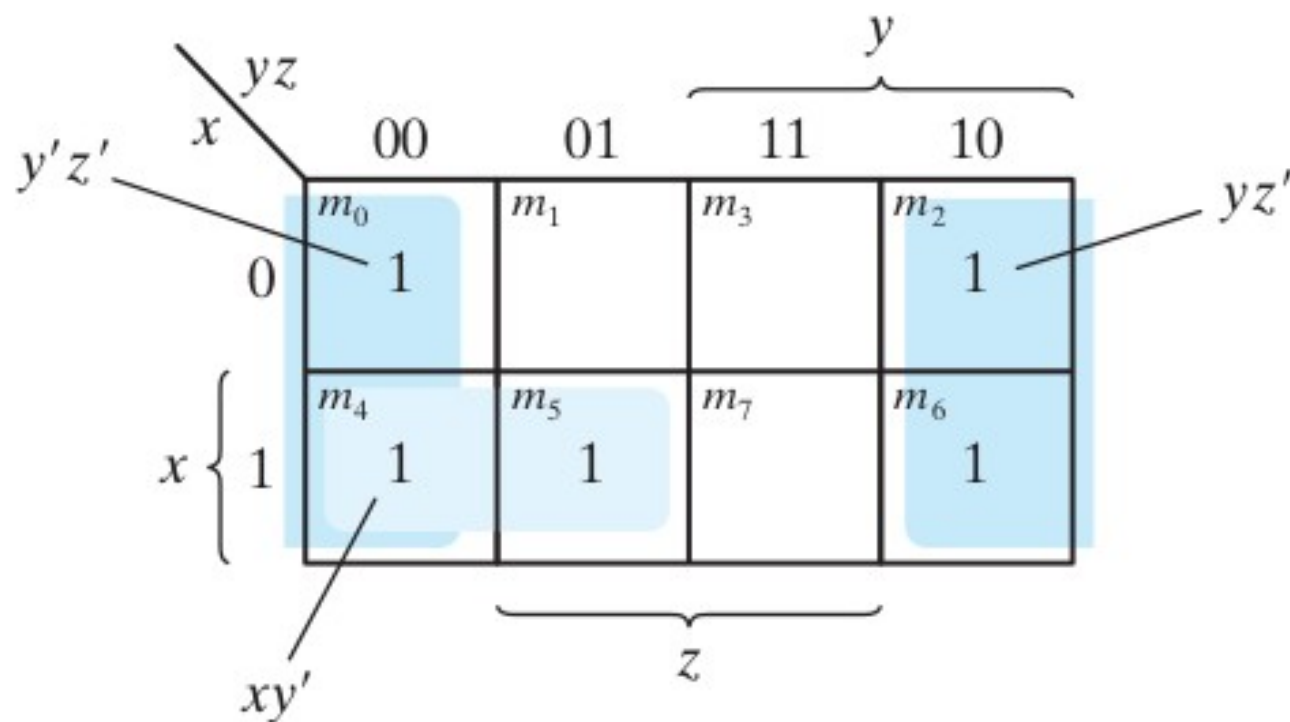
$F(x, y, z) = \Sigma(3, 4, 6, 7)$



Note:  $xy'z' + xyz' = xz'$

# Simplify $F(x, y, z) = \Sigma(0, 2, 4, 5, 6)$

$$F = z' + xy'$$



$$\text{Note: } y'z' + yz' = z'$$

# Question

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.

# Four-Variable K-Map

$m_0$	$m_1$	$m_3$	$m_2$
$m_4$	$m_5$	$m_7$	$m_6$
$m_{12}$	$m_{13}$	$m_{15}$	$m_{14}$
$m_8$	$m_9$	$m_{11}$	$m_{10}$

(a)

		$y$			
		$yz$	00	01	11
$w$	00	$m_0$ $w'x'y'z'$	$m_1$ $w'x'y'z$	$m_3$ $w'x'yz$	$m_2$ $w'x'yz'$
	01	$m_4$ $w'xy'z'$	$m_5$ $w'xy'z$	$m_7$ $w'xyz$	$m_6$ $w'xyz'$
	11	$m_{12}$ $wxy'z'$	$m_{13}$ $wxy'z$	$m_{15}$ $wxyz$	$m_{14}$ $wxyz'$
	10	$m_8$ $wx'y'z'$	$m_9$ $wx'y'z$	$m_{11}$ $wx'yz$	$m_{10}$ $wx'yz'$

(b)

# Problems

Simplify the Boolean function

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

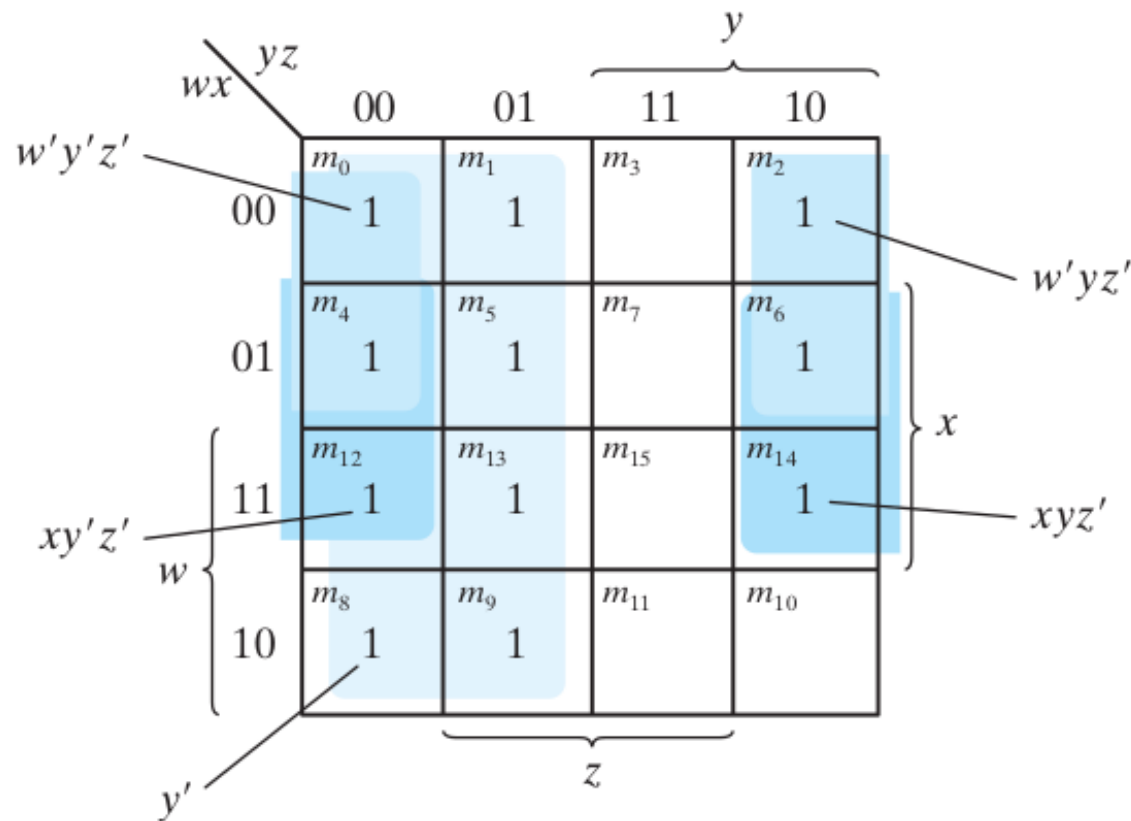
---

Simplify the Boolean function

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

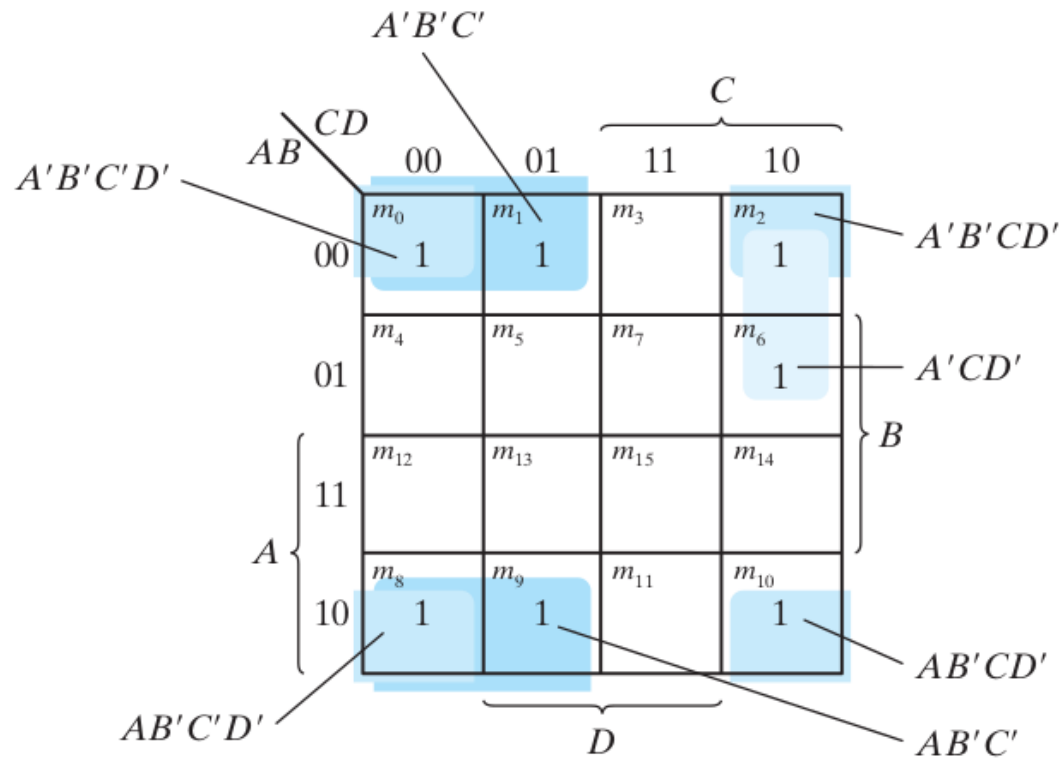


$$F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$$



Note:  $w'y'z' + w'yz' = w'z'$   
 $xy'z' + xyz' = xz'$

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



Note:  $A'B'C'D' + A'B'CD' = A'B'D'$   
 $AB'C'D' + AB'CD' = AB'D'$   
 $A'B'D' + AB'D' = B'D'$   
 $A'B'C' + AB'C' = B'C'$

# Prime Implicants and Essential Prime Implicants

In choosing adjacent squares in a map, we must ensure that

- (1) all the minterms of the function are covered when we combine the squares,
- (2) the number of terms in the expression is minimized, and
- (3) there are no redundant terms (i.e., minterms already covered by other terms).

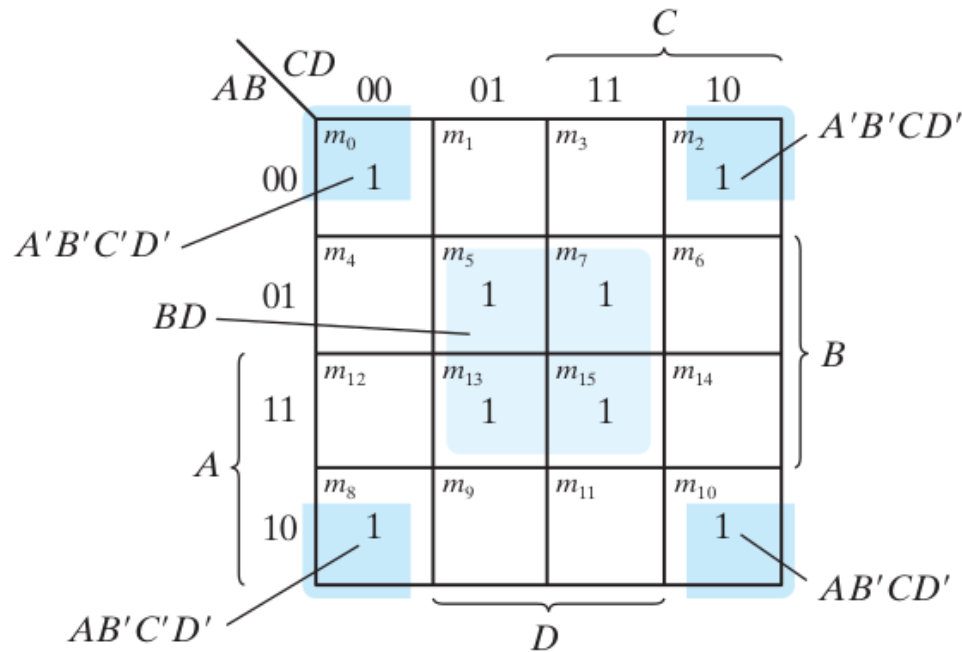
Sometimes there may be two or more expressions that satisfy the simplification criteria. The procedure for combining squares in the map may be made more systematic if we understand the meaning of two special types of terms.

A **prime implicant** is a product term obtained by combining the maximum possible number of adjacent squares in the map. The prime implicants of a function can be obtained from the map by combining **all possible maximum numbers of squares**.

If a minterm in a square is covered by only one prime implicant, that prime implicant is said to be **essential**.

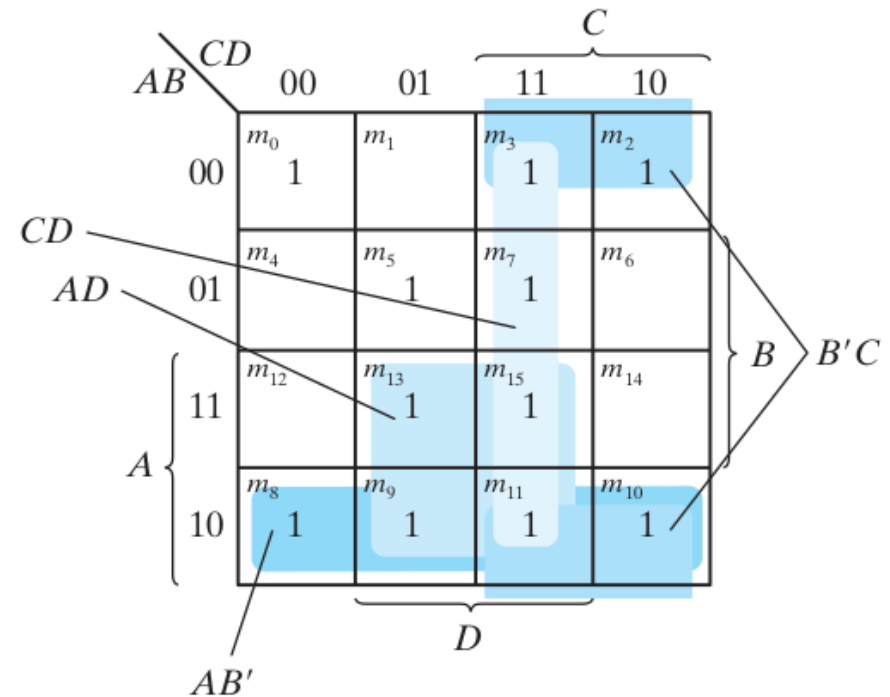
# Example:

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



Note:  $A'B'C'D' + A'B'CD' = A'B'D'$   
 $AB'C'D' + AB'CD' = AB'D'$   
 $A'B'D' + AB'D' = B'D'$

(a) Essential prime implicants  
 $BD$  and  $B'D'$



(b) Prime implicants  $CD$ ,  $B'C$ ,  
 $AD$ , and  $AB'$

## Procedure for finding the simplified expression

The procedure for finding the simplified expression from the map requires that we first determine all the essential prime implicants. The simplified expression is obtained from the logical sum of all the essential prime implicants, plus other prime implicants that may be needed to cover any remaining minterms not covered by the essential prime implicants. Occasionally, there may be more than one way of combining squares, and each combination may produce an equally simplified expression.

$$\begin{aligned} F &= BD + B'D' + CD + AD \\ &= BD + B'D' + CD + AB' \\ &= BD + B'D' + B'C + AD \\ &= BD + B'D' + B'C + AB' \end{aligned}$$

# Product-of-sums minimization

Simplify the following Boolean function into (a) sum-of-products form and (b) product-of-sums form:

$$F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10)$$

**Solution:-**

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

# Don't-care conditions

In practice, in some applications the function is not specified for certain combinations of the variables. As an example, the four-bit binary code for the decimal digits has six combinations that are not used and consequently are considered to be unspecified. Functions that have unspecified outputs for some input combinations are called **incompletely specified functions**. In most applications, we simply don't care what value is assumed by the function for the unspecified minterms. For this reason, it is customary to **call the unspecified minterms of a function don't-care conditions**. These don't-care conditions can be used on a map to provide further simplification of the Boolean expression.

To distinguish the don't-care condition from 1's and 0's, an **X** is used. Thus, an **X** inside a square in the map indicates that we don't care whether the value of 0 or 1 is assigned to F for the particular minterm. In choosing adjacent squares to simplify the function in a map, the don't-care minterms may be assumed to be either 0 or 1. **When simplifying the function, we can choose to include each don't-care minterm with either the 1's or the 0's, depending on which combination gives the simplest expression.**

# Don't-care conditions - Example

Simplify the Boolean function

$$F(w, x, y, z) = \Sigma(1, 3, 7, 11, 15)$$

which has the don't-care conditions

$$d(w, x, y, z) = \Sigma(0, 2, 5)$$



# Don't-care conditions - Solution

Diagram (a) shows a 4x4 Karnaugh map for the function  $F = yz + w'x'$ . The map is labeled with  $wx$  on the left and  $yz$  on the top. The columns are labeled 00, 01, 11, 10, and the rows are labeled 00, 01, 11, 10. The cells are labeled  $m_0$  through  $m_{15}$ . The function is represented by 1s in the cells where  $yz = 11$  (columns 11 and 10) and  $w'x' = 1$  (rows 00 and 01). The cells containing 1s are  $m_1, m_3, m_7, m_{11}$  (all 1s) and  $m_0, m_2, m_4, m_6, m_{12}, m_{14}, m_8, m_{10}$  (all 0s). The cells containing 'X' are  $m_0, m_2, m_4, m_6, m_{12}, m_{14}, m_8, m_{10}$ . The map is grouped into four 2x2 blocks:  $w'x'$  (top-left),  $yz$  (top-right),  $w'z$  (bottom-left), and  $wz$  (bottom-right). The function is  $F = yz + w'x'$ .

$wx \backslash yz$	00	01	11	10
00	$m_0$ X	$m_1$ 1	$m_3$ 1	$m_2$ X
01	$m_4$ 0	$m_5$ X	$m_7$ 1	$m_6$ 0
11	$m_{12}$ 0	$m_{13}$ 0	$m_{15}$ 1	$m_{14}$ 0
10	$m_8$ 0	$m_9$ 0	$m_{11}$ 1	$m_{10}$ 0

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

Diagram (b) shows a 4x4 Karnaugh map for the function  $F = yz + w'z$ . The map is labeled with  $wx$  on the left and  $yz$  on the top. The columns are labeled 00, 01, 11, 10, and the rows are labeled 00, 01, 11, 10. The cells are labeled  $m_0$  through  $m_{15}$ . The function is represented by 1s in the cells where  $yz = 11$  (columns 11 and 10) and  $w'z = 1$  (rows 00 and 01). The cells containing 1s are  $m_1, m_3, m_7, m_{11}$  (all 1s) and  $m_0, m_2, m_4, m_6, m_{12}, m_{14}, m_8, m_{10}$  (all 0s). The cells containing 'X' are  $m_0, m_2, m_4, m_6, m_{12}, m_{14}, m_8, m_{10}$ . The map is grouped into four 2x2 blocks:  $w'z$  (top-left),  $yz$  (top-right),  $w'x$  (bottom-left), and  $wx$  (bottom-right). The function is  $F = yz + w'z$ .

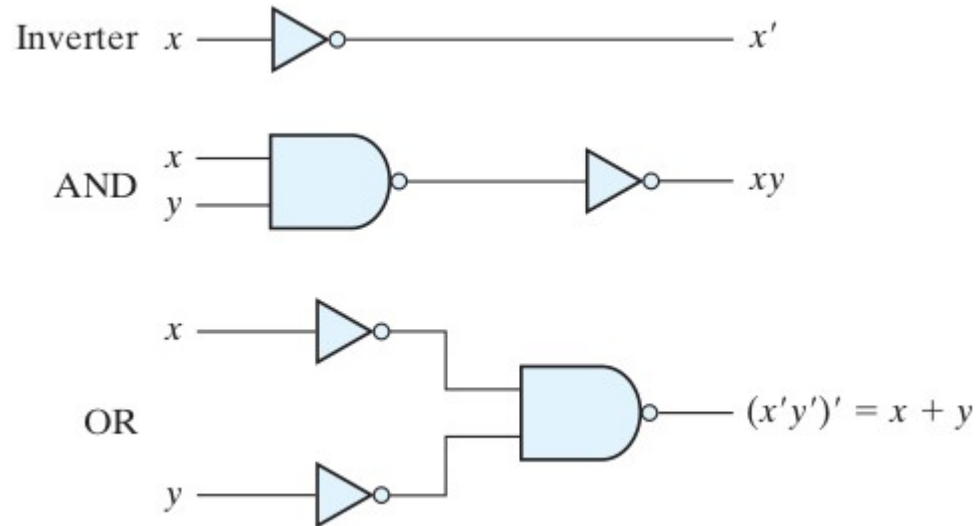
$wx \backslash yz$	00	01	11	10
00	$m_0$ X	$m_1$ 1	$m_3$ 1	$m_2$ X
01	$m_4$ 0	$m_5$ X	$m_7$ 1	$m_6$ 0
11	$m_{12}$ 0	$m_{13}$ 0	$m_{15}$ 1	$m_{14}$ 0
10	$m_8$ 0	$m_9$ 0	$m_{11}$ 1	$m_{10}$ 0

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

# **NAND and NOR Implementation**

# Universal Gate - NAND

NAND gate is said to be **universal gate** because any logic circuit can be implemented with it. To show that any Boolean function can be implemented with NAND gates, we need only show that the logical operations of AND, OR, and complement can be obtained with NAND gates alone. This is indeed shown in following fig:

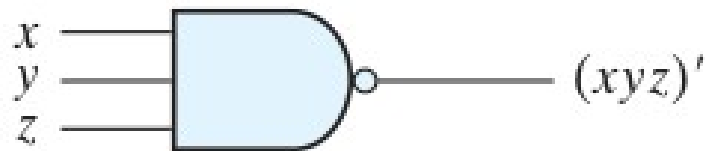


The complement operation is obtained from a one-input NAND gate (inverter).

The AND operation requires two NAND gates. The first produces the NAND operation and the second inverts the logical sense of the signal.

The OR operation is achieved through a NAND gate with additional inverters in each input.

# Graphic symbols for a three-input NAND gate



(a) AND-invert



(b) Invert-OR

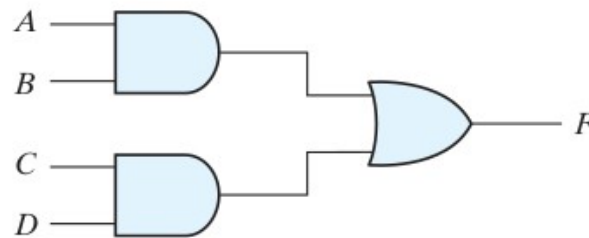
NAND Gate:  
AND-invert /  
Invert-OR /  
Bubbled-OR

# Two-Level NAND gate Implementation

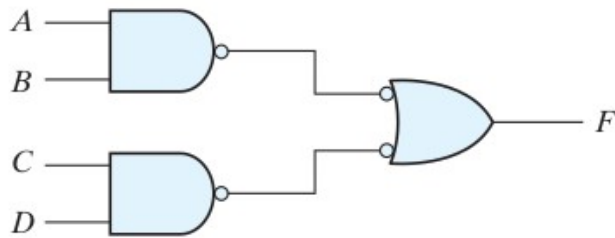
A convenient way to implement a Boolean function with NAND gates is to obtain the simplified Boolean function in terms of Boolean operators and then convert the function to NAND logic.

The implementation of Boolean functions with NAND gates requires that the functions be in sum-of-products form.

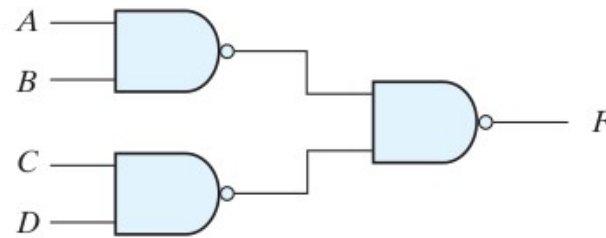
Example:  $F = AB + CD$ ,  $F = ((AB)'(CD)')' = AB + CD$



(a)



(b)



(c)