Digital Design and Computer Organization

Module1 : Introduction to Digital Design

Topics

- Binary Logic,
- Basic Theorems And Properties Of Boolean Algebra,
- Boolean Functions,
- Digital Logic Gates,
- Introduction to the Map Method,
- Four-Variable Map,
- Don’t-Care Conditions,
- NAND and NOR Implementation,
- Other Hardware Description Language – Verilog Model of a simple circuit.

Text book 1: 1.9,2.4, 2.5, 2.8, 3.1, 3.2, 3.3, 3.5, 3.6, 3.9

Binary Logic

Definition of Binary Logic

- Binary logic consists of binary variables and a set of logical operations.
- The variables are designated by letters of the alphabet, such as A, B, C, x, y, z, etc., with each variable having only two distinct possible values: 1 and 0.
- There are three basic logical operations: AND, OR, and NOT. Each operation produces a binary result

AND Operation

- This operation is represented by a dot or by the absence of an operator.
- For example, x·y=z or xy=z is read “x AND y is equal to z.”
- The logical operation AND is interpreted to mean that z=1 if and only if x=1 and y=1; otherwise z=0. (Remember that x, y, and z are binary variables and can be equal either to 1 or 0, and nothing else.)
- The result of the operation x·y is z.

Binary Logic

OR Operation

- This operation is represented by a plus sign.
- For example, x+y=z is read “x OR y is equal to z,” meaning that z=1 if x=1 or if y=1 or if both x=1 and y=1. If both x=0 and y=0, then z=0.

NOT Operation

- This operation is represented by a prime (sometimes by an overbar).
- For example, $x'=z$ (or $\bar{x}=z$) is read “not x is equal to z,” meaning that z is what x is not.
- In other words, if x=1, then z=0, but if x=0, then z=1.
- The NOT operation is also referred to as the complement operation, since it changes a 1 to 0 and a 0 to 1, that is, the result of complementing 1 is 0, and vice versa.

Binary Logic

Truth Table

Definitions of logical operations may be listed in a compact form called truth tables.

A truth table is a table of all possible combinations of the variables, showing the relation between the values that the variables may take and the result of the operation.

The truth tables for the operations AND and OR with variables x and y are obtained by listing all possible values that the variables may have when combined in pairs.

For each combination, the result of the operation is then listed in a separate row. The truth tables for AND, OR, and NOT are given in below Table. These tables clearly demonstrate the definition of the operations.

AND

X	Y	Z= X.Y
0	0	0
0	1	0
1	0	0
1	1	1

OR

X	Y	Z= X+Y
0	0	0
0	1	1
1	0	1
1	1	1

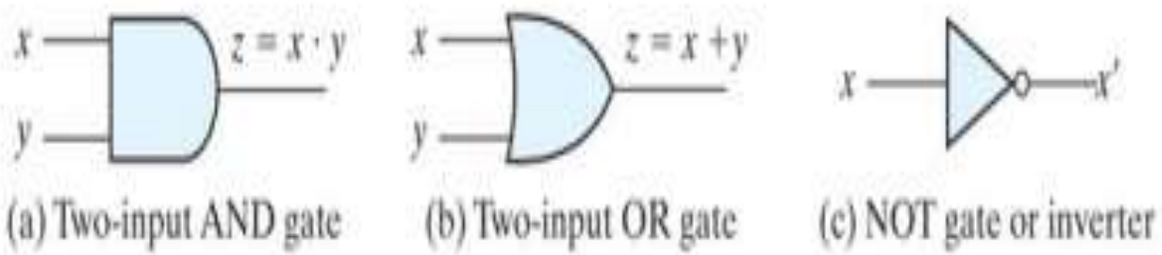
NOT

X	Z=x`
0	1
1	0

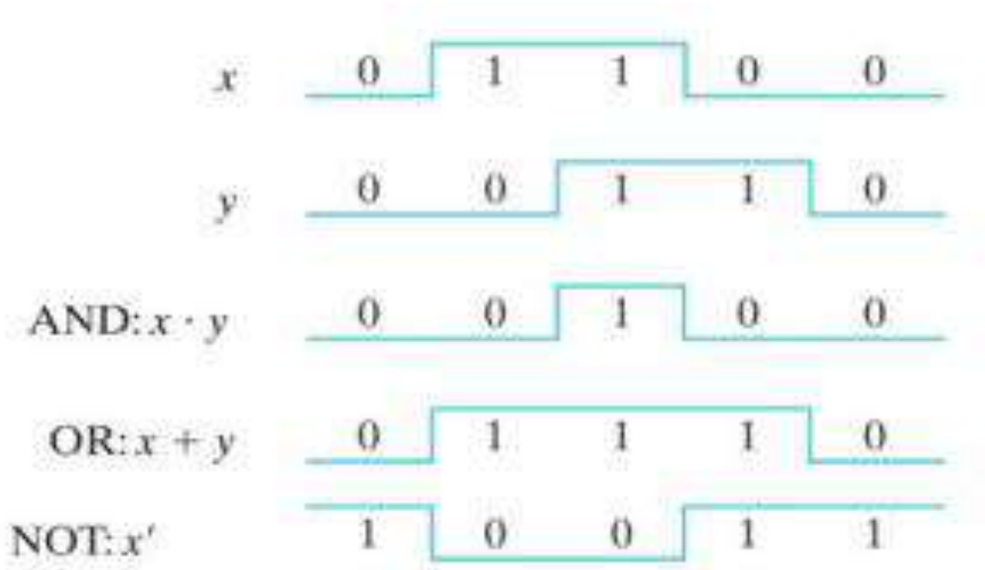
Binary Logic

Logic Gates

Logic gates are electronic circuits that operate on one or more physical input signals to produce an output signal. The graphic symbols used to designate the three types of gates and their input –output signals are shown in Fig.



Symbols for digital logic circuits

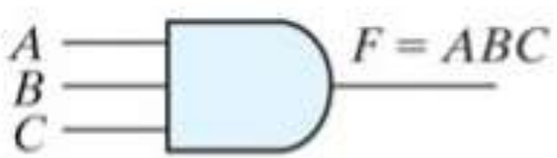


Input–output signals for gates

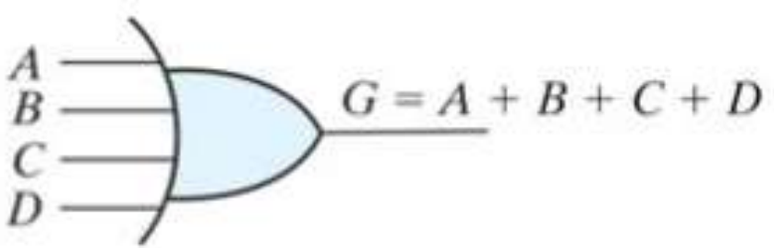
Binary Logic

Gates with multiple inputs

- AND and OR gates may have more than two inputs.
- An AND gate with three inputs and an OR gate with four inputs are shown in Fig..
- The three-input AND gate responds with logic 1 output if all three inputs are logic 1. The output produces logic 0 if any input is logic 0.
- The four-input OR gate responds with logic 1 if any input is logic 1; its output becomes logic 0 only when all inputs are logic 0.



Three-input AND gate



Four-input OR gate

Basic Theorems And Properties Of Boolean Algebra

Table: Postulates and Theorems of Boolean Algebra

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

Basic Theorems And Properties Of Boolean Algebra

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

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

THEOREM 1(b): x . x = x

Statement	Justification
$x \cdot x = x x + 0$	postulate 2(a)
$= x x + x x'$	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).

Basic Theorems And Properties Of Boolean Algebra

THEOREM 2(a): x +1= 1

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

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

THEOREM 6(a): x + x y = x

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

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

Basic Theorems And Properties of Boolean Algebra

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:

THEOREM 6(a): x + x y = x

X	Y	XY	X+XY
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

THEOREM 6(b): x (x + y) = x

X	Y	X+Y	X(X+Y)
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1

Demorgan's Theorem (x+y)'=x'.y'

X	Y	X+Y	(X+Y)`	X'	Y'	X'.Y'
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Basic Theorems And Properties of Boolean Algebra

Operator Precedence

The operator precedence for evaluating Boolean expressions is

(1) parentheses, (2) NOT, (3) AND, and (4) OR.

In other words, expressions inside parentheses must be evaluated before all other operations.

The next operation that holds precedence is the complement, and then follows the AND and, finally, the OR.

As an example, consider the truth table for one of DeMorgan's theorems.

The left side of the expression is $(x + y)'$. Therefore, the expression inside the parentheses is evaluated first and the result then complemented.

The right side of the expression is $x' y'$, so the complement of x and the complement of y are both evaluated first and the result is then ANDed.

Basic Theorems And Properties of Boolean Algebra

Practice Exercises

1. Using the basic theorems and postulates of Boolean algebra, simplify the following Boolean expression: $F = x' y' z + x y z + x' y z + x y' z$.

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

$F= y' z + y z$

$F=(y' + y) z$

$F= z$

Answer: F = z

Basic Theorems And Properties of Boolean Algebra

Practice Exercises

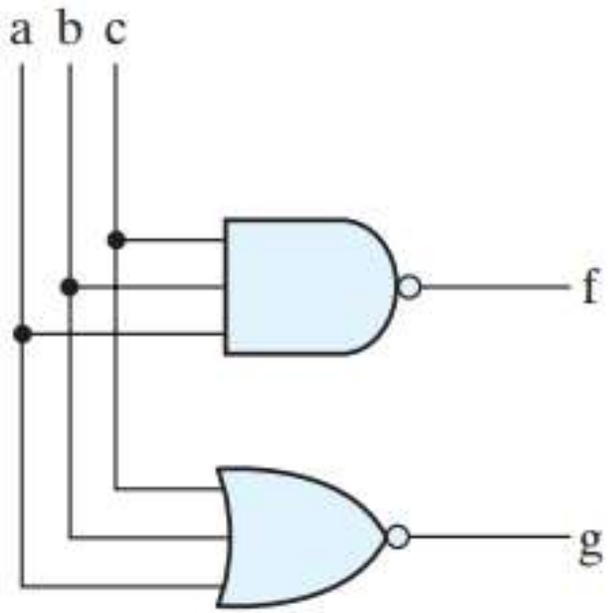
2. Develop a truth table for the Boolean expression $F = x'y'z$.

X	Y	Z	X′	Y′	x′y′	x′y′z
0	0	0	1	1	1	0
0	0	1	1	1	1	1
0	1	0	1	0	0	0
0	1	1	1	0	0	0
1	0	0	0	1	0	0
1	0	1	0	1	0	0
1	1	0	0	0	0	0
1	1	1	0	0	0	0

Basic Theorems And Properties of Boolean Algebra

Practice Exercises

By means of a timing diagram, show the signals of the outputs f and g in Fig as functions of the three inputs a, b, and c. Use all eight possible combinations of a, b, and c.

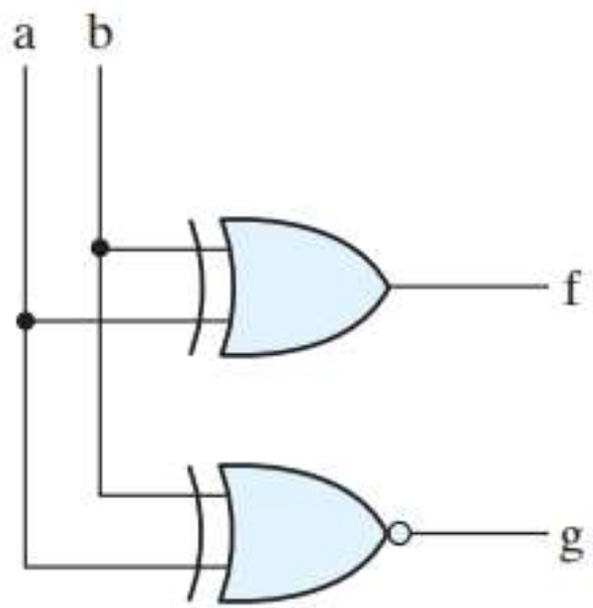


a	b	c	f=(abc)’	g=(a+b+c)’
0	0	0	1	1
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	0

Basic Theorems And Properties of Boolean Algebra

Practice Exercises

By means of a timing diagram, show the signals of the outputs f and g in Fig as functions of the two inputs a and b. Use all four possible combinations of a and b.



a	b	f=a xor b	G=a xnor b
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

BOOLEANFUNCTIONS

- 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.
- As an example, consider the Boolean function

  F1 = x + y'z
- The function F1 is equal to 1 if x is equal to 1 or if both y' and z are equal to 1. F1 is equal to 0 otherwise.
- The complement operation dictates that when y' = 1, y = 0. Therefore, F1 = 1 if x = 1 or if y = 0 and z = 1.
- 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.

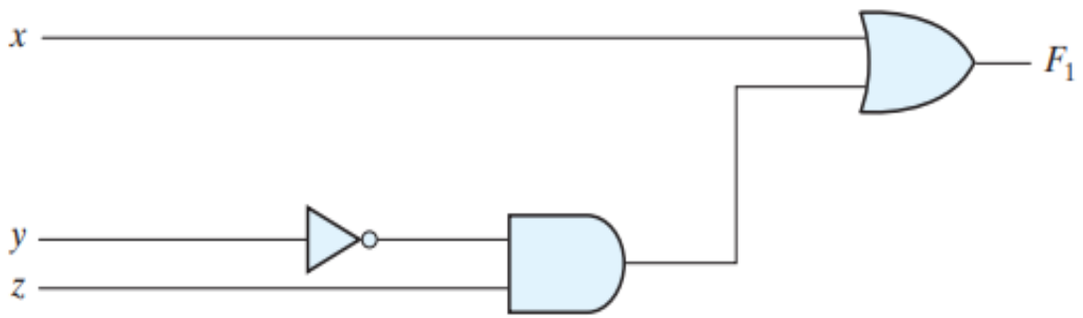
BOOLEAN FUNCTIONS

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 to 2^n - 1.
- Table shows the truth table for the function F1.
- There are eight possible binary combinations for assigning bits to the three variables x, y, and z.
- The column labeled F1 contains either 0 or 1 for each of these combinations.
- The table shows that the function is equal to 1 when x = 1 or when yz = 01 and is equal to 0 otherwise.
- 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

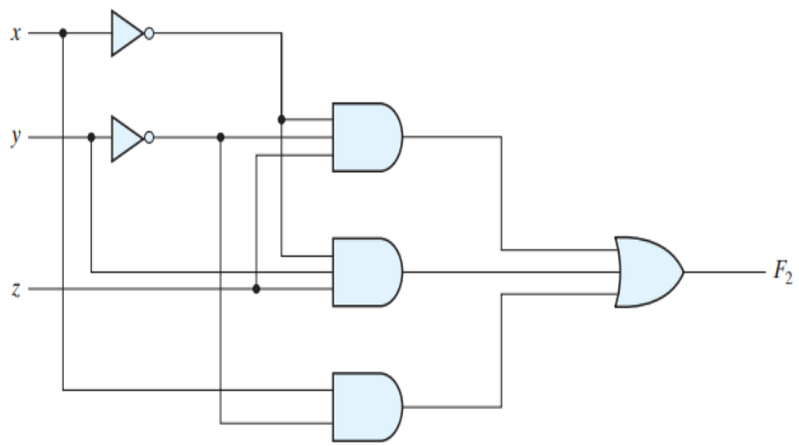


BOOLEAN FUNCTIONS

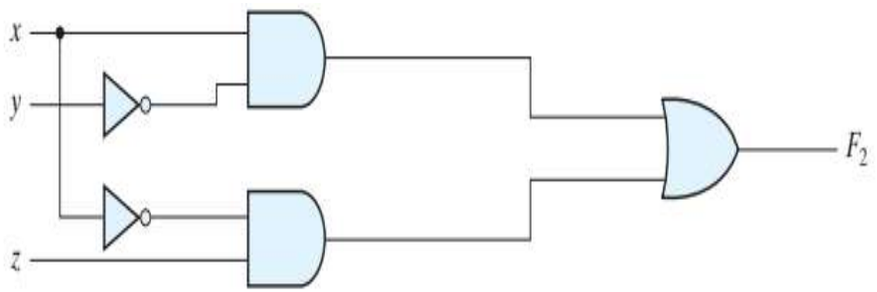
Simplify the function.

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

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



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



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

BOOLEAN FUNCTIONS

Algebraic Manipulation

By reducing the number of terms, the number of literals, or both in a Boolean expression, it is often possible to obtain a simpler circuit.

The manipulation of Boolean algebra consists of reducing an expression for the purpose of obtaining a simpler circuit.

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.

Functions 4 and 5 are together known as the consensus theorem.

BOOLEAN FUNCTIONS

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.

The three-variable form of the first DeMorgan's theorem is derived as follows, from the relevant postulates and theorems:

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

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

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

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.

BOOLEAN FUNCTIONS

Find the complement of the following functions by applying DeMorgan's theorems as many times as necessary.

$F_1 = x'yz' + x'y'z$ and $F_2 = x(y'z' + yz)$.

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

$$F_2' = [x(y'z' + yz)]' = x' + (y'z' + yz)' = x' + (y'z')'(yz)'$$

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

$$= x' + yz' + y'z$$

BOOLEAN FUNCTIONS

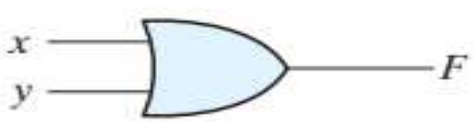
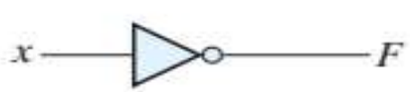
Find the complement of the functions F1 and F2 of previous by taking their duals and complementing each literal.

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

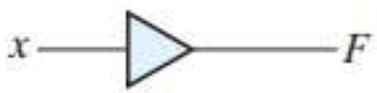
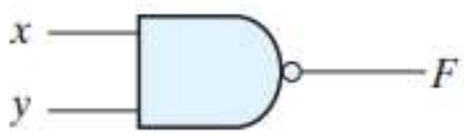
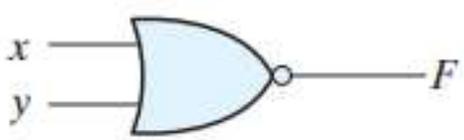
DIGITAL LOGIC GATES

Digital Logic Gates

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

DIGITAL LOGIC GATES

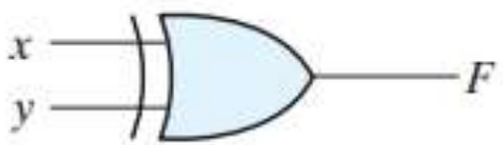
Digital Logic Gates

Buffer		$F = x$	x | F 0 | 0 1 | 1
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

DIGITAL LOGIC GATES

Digital Logic Gates

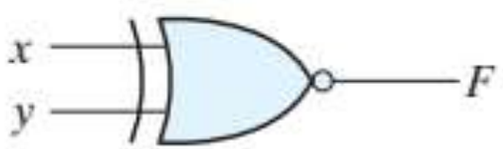
Exclusive-OR (XOR)



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

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

DIGITAL LOGIC GATES

Extension to Multiple Inputs

The gates **except for the inverter and buffer** can be extended to have more than two inputs.

A gate can be extended to have multiple inputs if the binary operation it represents is **commutative and associative.**

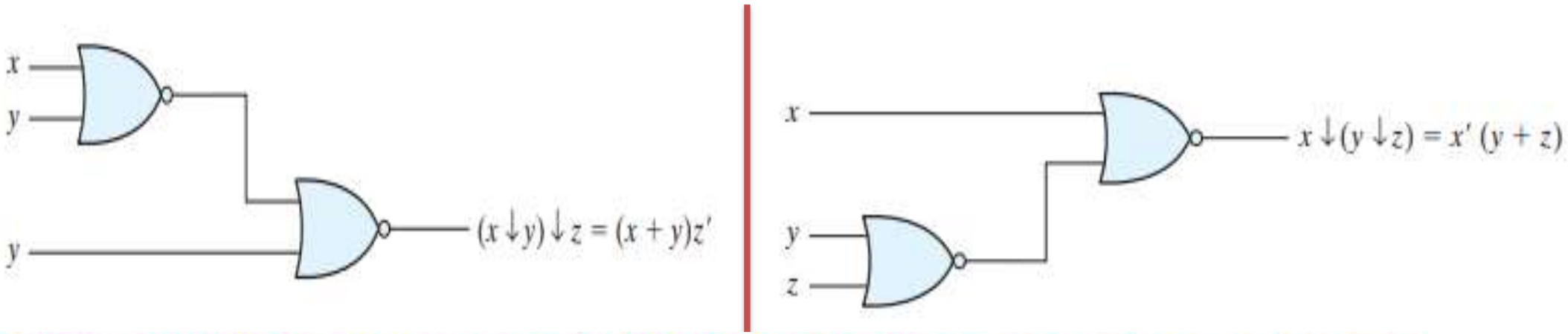
The AND and OR operations, defined in Boolean algebra, possess these two properties.

For the OR function, we have x + y = y + x (commutative) and (x + y) + z = x + (y + z) = x + y + z (associative) which indicates that the gate inputs can be interchanged and that the OR function can be extended to three or more variables.

The NAND and NOR functions are commutative, and their gates can be extended to have more than two inputs, provided that the definition of the operation is **modified slightly**. The difficulty is that the NAND and NOR operators are **not associative**, as shown in Fig. and the following equations:

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

DIGITAL LOGIC GATES



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

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

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

DIGITAL LOGIC GATES

The graphic symbols for the three-input gates are shown in Fig. 2.7.

In writing cascaded NOR and NAND operations, one must use the correct parentheses to signify the proper sequence of the gates. To demonstrate this principle, consider the circuit of Fig. 2.7 (c). The Boolean function for the circuit must be written as $F = [(ABC)'(DE)']' = ABC + DE$

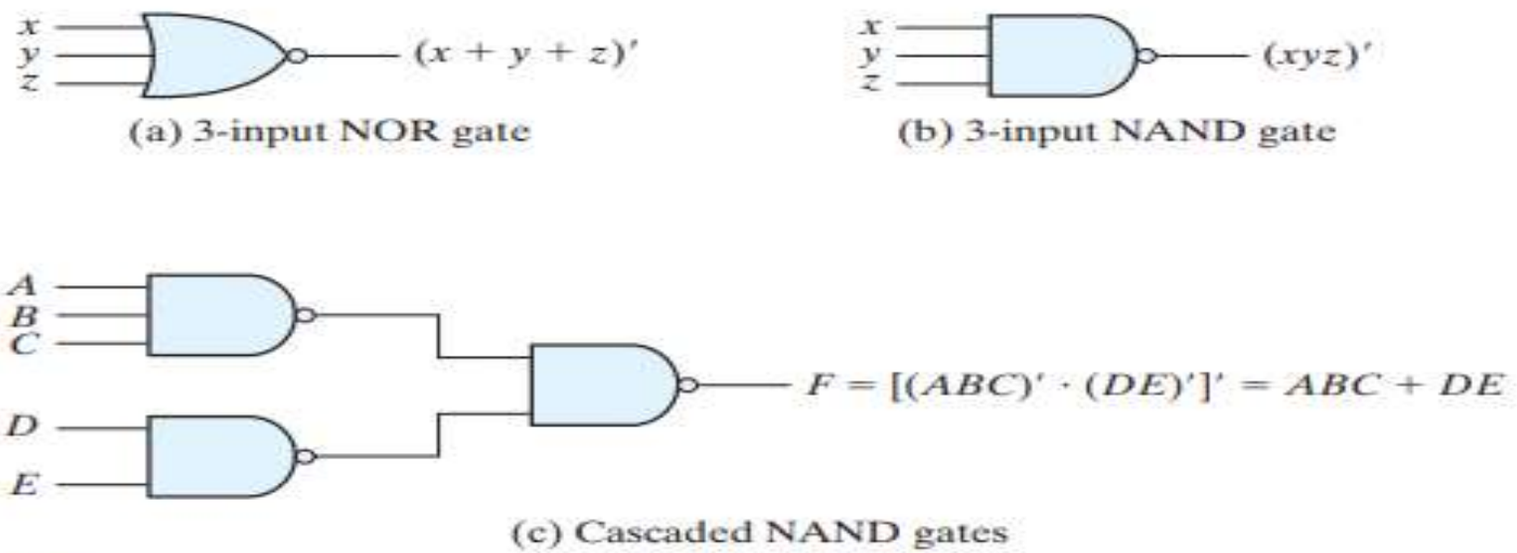
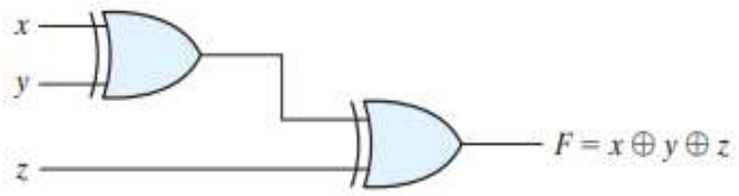


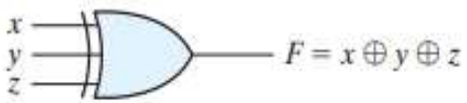
FIGURE 2.7
Multiple-input and cascaded NOR and NAND gates

DIGITAL LOGIC GATES

The **exclusive-OR** and equivalence gates are both commutative and associative and can be extended to more than two inputs. **Exclusive-OR is an odd function (i.e., it is equal to 1 if the input variables have an odd number of 1's).** The construction of a three-input exclusive-OR function is shown in Fig. 2.8 . This function is normally implemented by cascading two-input gates, as shown in (a). Graphically, it can be represented with a single three-input gate, as shown in (b). The truth table in (c) clearly indicates that the output F is equal to 1 if only one input is equal to 1 or if all three inputs are equal to 1 (**i.e., when the total number of 1's in the input variables is odd**).



(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

DIGITAL LOGIC GATES

Positive and Negative Logic

- The binary signal at the inputs and outputs of any gate has one of two values, except during transition.
- One signal value represents logic 1 and the other logic 0. Since two signal values are assigned to two logic values, there exist two different assignments of signal level to logic value, as shown in Fig. 2.9 .
- The higher signal level is designated by H and the lower signal level by L.
- Choosing the high-level H to represent logic 1 defines a positive logic system.
- Choosing the low-level L to represent logic 1 defines a negative logic system.

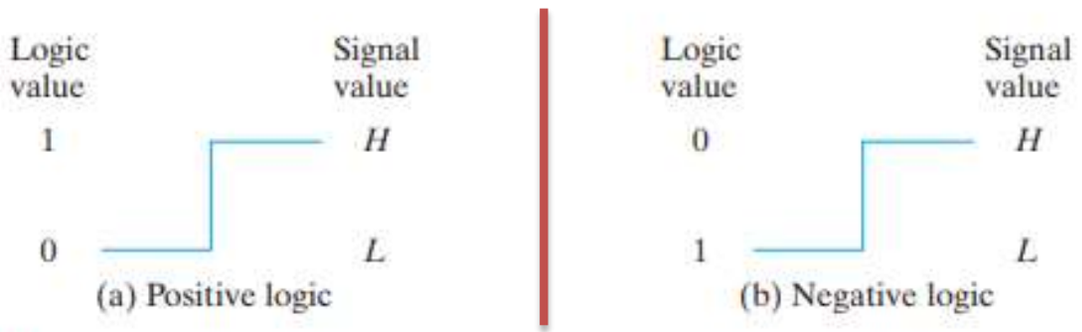


FIGURE 2.9
Signal assignment and logic polarity

DIGITAL LOGIC GATES

Hardware digital gates are defined in terms of signal values such as H and L.

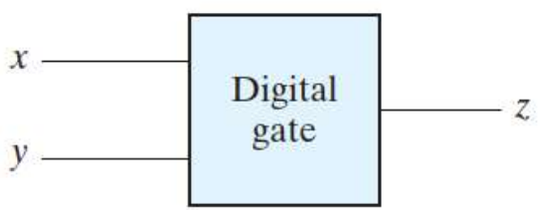
Consider, for example, the electronic gate shown in Fig. (b). The truth table for this gate is listed in Fig. (a).

The truth table of Fig. (c) assumes a positive logic assignment, with H = 1 and L = 0.

This truth table is the same as the one for the AND operation. The graphic symbol for a positive logic AND gate is shown in Fig. (d).

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

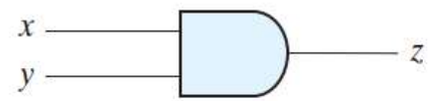
(a) Truth table with *H* and *L*



(b) Gate block diagram

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

Demonstration of positive and negative logic

DIGITAL LOGIC GATES

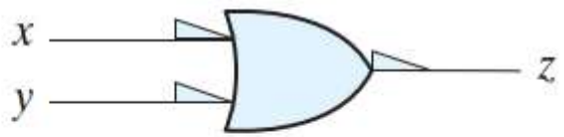
Now consider the **negative logic** assignment for the same physical gate with L = 1 and H = 0. The result is the truth table of Fig. (e). This table represents the OR operation, even though the entries are reversed. The graphic symbol for the negative- logic OR gate is shown in Fig. (f).

The small triangles in the inputs and output designate a polarity indicator(negative logic).

Thus, the same physical gate can operate either as a positive-logic AND gate or as a negative-logic OR 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

Demonstration of positive and negative logic

DIGITAL LOGIC GATES

The conversion from positive logic to negative logic and vice versa is essentially an operation that changes 1's to 0's and 0's to 1's in both the inputs and the output of a gate.

Since this operation produces the dual of a function, the change of all terminals from one polarity to the other results in taking the dual of the function.

All AND operations are converted to OR operations (or graphic symbols) and vice versa. In addition, one must not forget to include the **polarity-indicator** triangle in the graphic symbols when negative logic is assumed.

MAP METHOD

Map Method

- The map method provides a simple, straightforward procedure for minimizing Boolean functions.
- This method may be regarded as a pictorial form of a truth table.
- The map method is also known as the Karnaugh map or K-map .
- A K-map is a diagram made up of squares, with each square representing one minterm of the function that is to be minimized.
- 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.

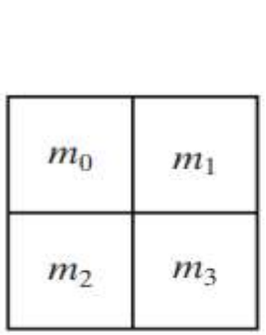
MAP METHOD

Two-Variable K-Map

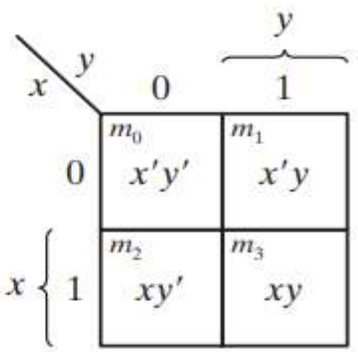
- The two-variable map is shown in Fig(a).
- There are four minterms for two variables; hence, the map consists of four squares, one for each minterm.
- The map is redrawn in (b) to show the relationship between the squares and the two variables x and y .
- The 0 and 1 marked in each row and column designate the values of variables.
- Variable x appears primed in row 0 and unprimed in row 1.
- Similarly, y appears primed in column 0 and unprimed in column 1.

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

Two-variable K-map



(a)



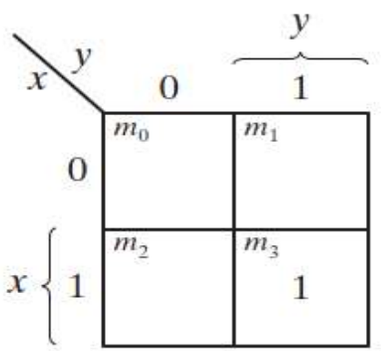
(b)

MAP METHOD

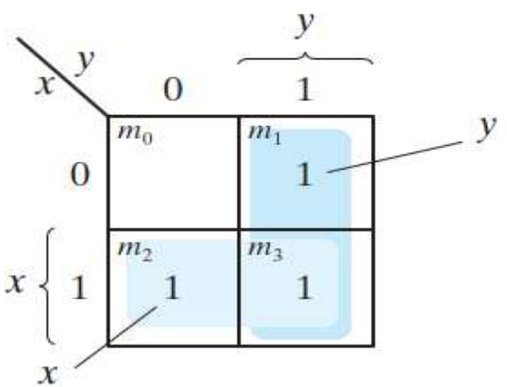
Map Method

If we mark the squares whose minterms belong to a given function, the two-variable map becomes another useful way to represent any one of the 16 Boolean functions of two variables. As an example, the function xy is shown in Fig. (a). Since xy is equal to m3 , a 1 is placed inside the square that belongs to m3 . Similarly, the function x + y is represented in the map of Fig. (b) by three squares marked with 1's. These squares are found from the minterms of the function:

The three squares could also have been determined from the intersection of variable x in the second row and variable y in the second column, which encloses the area belonging to x or y . In each example, the minterms at which the function is asserted are marked with a 1.



(a) xy



(b) $x + y$

MAP METHOD

Three-Variable K-Map

- A three-variable K-map is shown in Fig. 3.
- There are eight minterms for three binary variables; therefore, the map consists of eight squares.
- Note that the minterms are arranged, not in a binary sequence, but in a sequence similar to the Gray code. The characteristic of this sequence is that only one bit changes in value from one adjacent column to the next.
- The map drawn in part (b) is marked with numbers in each row and each column to show the relationship between the squares and the three variables.
- For example, the square assigned to m5 corresponds to row 1 and column 01. When these two numbers are concatenated, they give the binary number 101, whose decimal equivalent is 5.
- Each cell of the map corresponds to a unique minterm, so another way of looking at square m5 = xyz is to consider it to be in the row marked x and the column belonging to yz (column 01). Note that there are four squares in which each variable is equal to 1 and four in which each is equal to 0.
- The variable appears unprimed in the former four squares and primed in the latter. For convenience, we write the variable with its letter symbol under the four squares in which it is unprimed.

MAP METHOD

Three-Variable K-Map

To understand the usefulness of the map in simplifying Boolean functions, we must recognize the basic property possessed by adjacent squares: Any two adjacent squares in the map differ by only one variable, which is primed in one square and unprimed in the other. For example, m5 and m7 lie in two adjacent squares. Variable y is primed in m5 and unprimed in m7 , whereas the other two variables are the same in both squares. From the postulates of Boolean algebra, it follows that the sum of two minterms in adjacent squares can be simplified to a single product term consisting of only two literals. To clarify this concept, consider the sum of two adjacent squares such as m5 and m7 :

$$m_5 + m_7 = xy'z + xyz = xz(y' + y) = xz$$

Here, the two squares differ by the variable y , which can be removed when the sum of the two minterms is formed. Thus, any two minterms in adjacent squares (vertically or horizontally, but not diagonally, adjacent) that are ORed together will cause a removal of the dissimilar variable. The next four examples explain the procedure for minimizing a Boolean function with a K-map.

19/08/2024 Dr. Vinod Kumar P, Associate Professor, CSE-DS ATMECE

MAP METHOD

Three-Variable K-Map

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

(a)

x \ yz	00	01	11	10
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'

(y spans 11 and 10; z spans 01 and 11)

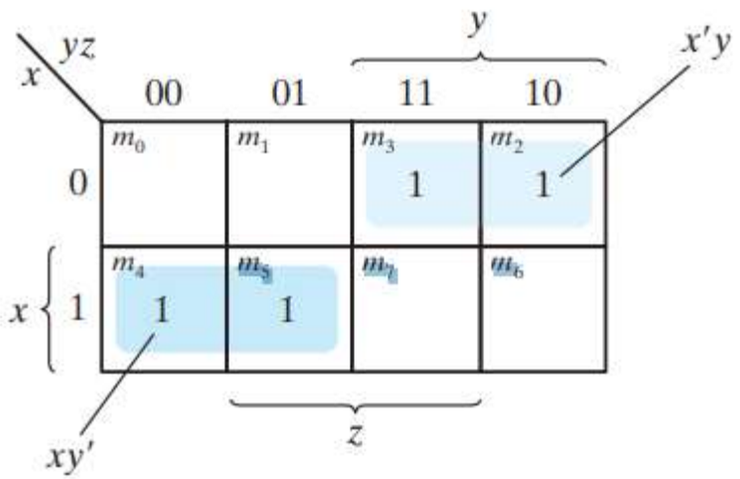
(b)

Three-variable K-map

MAP METHOD

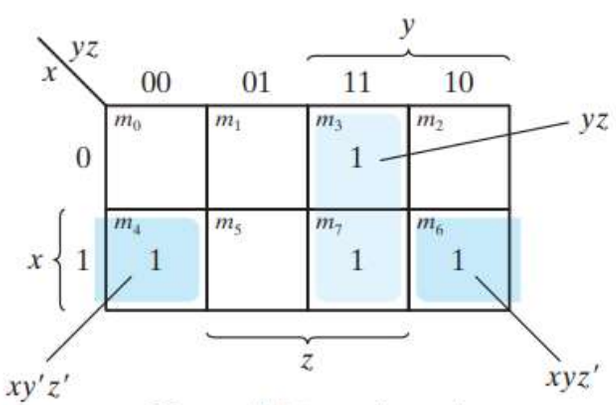
Simplify the Boolean function

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



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

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



$$F = yz + xz'$$

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

MAP METHOD

Three Variable Map Method

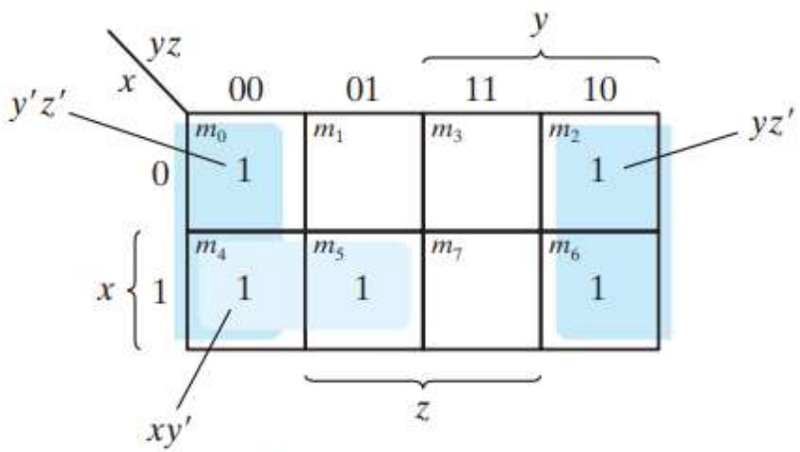
The number of adjacent squares that may be combined must always represent a number that is a power of two, such as 1, 2, 4, and 8. As more adjacent squares are combined, we obtain a product term with fewer literals.

- One square represents one minterm, giving a term with three literals.
- Two adjacent squares represent a term with two literals.
- Four adjacent squares represent a term with one literal.
- Eight adjacent squares encompass the entire map and produce a function that is always equal to 1.

MAP METHOD

Map Method

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



Note: $y'z' + yz' = z'$

$$F = z' + xy'$$

MAP METHOD

Map Method

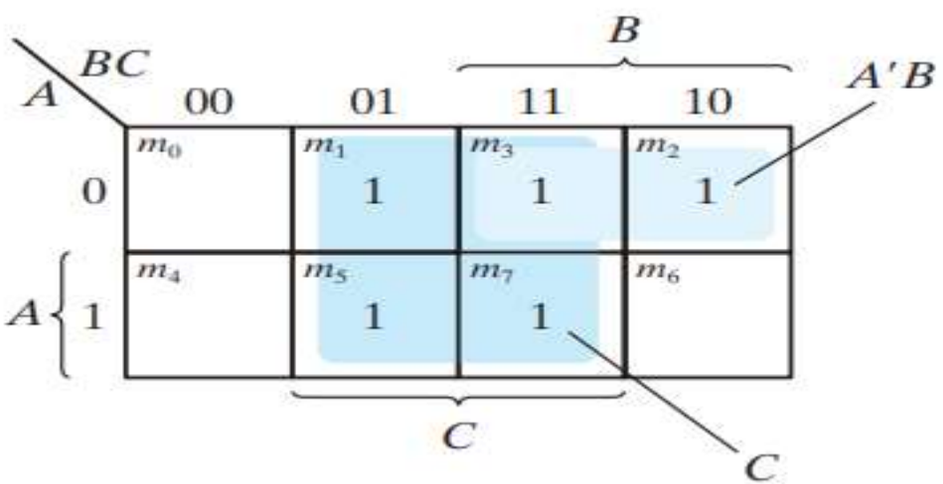
For the Boolean function

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

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

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

$$F(A, B, C) = \Sigma(1, 2, 3, 5, 7)$$



$$F = C + A'B$$

FOUR-VARIABLE MAP

FOUR-VARIABLEK-MAP

- The map for Boolean functions of four binary variables (w, x, y, z) is shown in Fig.
- In Fig. (a) are listed the 16 minterms and the squares assigned to each.
- In Fig. (b), the map is redrawn to show the relationship between the squares and the four variables.
- The rows and columns are numbered in a Gray code sequence, with only one digit changing value between two adjacent rows or columns.
- The minterm corresponding to each square can be obtained from the concatenation of the row number with the column number.
- For example, the numbers of the third row (11) and the second column (01), when concatenated, give the binary number 1101, the binary equivalent of decimal 13.
- Thus, the square in the third row and second column represents minterm m13.

19/08/2024 Dr. Vinod Kumar P, Associate Professor, CSE-DS ATMECE

FOUR-VARIABLE 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)

wx \ yz	00	01	11	10
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} $wx'yz'$

y (11, 10) x (01, 11) w (11, 10) z (00, 01)

(b)

FOUR-VARIABLEMAP

FOUR-VARIABLEK-MAP

- The map minimization of four-variable Boolean functions is similar to the method used to minimize three-variable functions.
- Adjacent squares are defined to be squares next to each other.
- In addition, the map is considered to lie on a surface with the top and bottom edges, as well as the right and left edges, touching each other to form adjacent squares.
- For example, m0 and m2 form adjacent squares, as do m3 and m11.

The combination of adjacent squares that is useful during the simplification process is easily determined from inspection of the four-variable map:

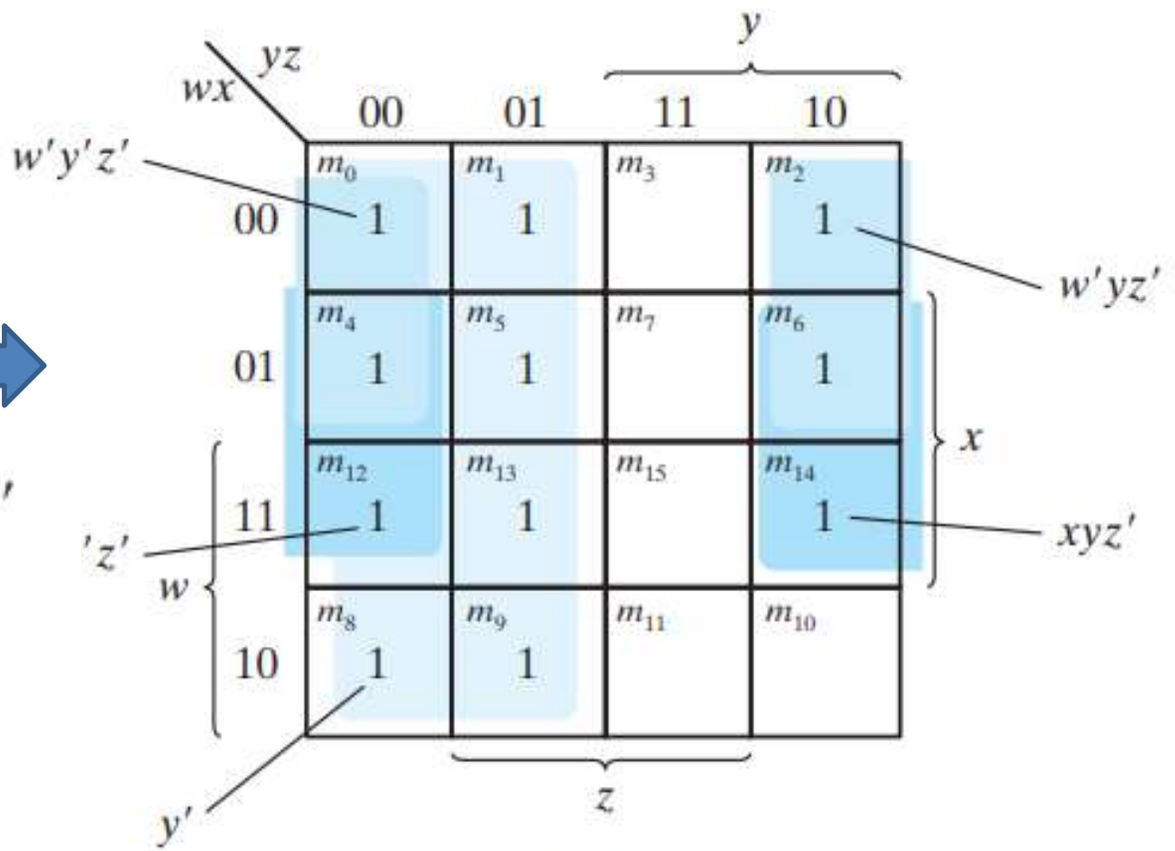
- ✓ One square represents one minterm, giving a term with four literals.
- ✓ Two adjacent squares represent a term with three literals.
- ✓ Four adjacent squares represent a term with two literals.
- ✓ Eight adjacent squares represent a term with one literal.
- ✓ Sixteen adjacent squares produce a function that is always equal to 1.

FOUR-VARIABLE MAP

FOUR-VARIABLE K-MAP

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

$$F = y' + w'z' + xz'$$



Note: $w'y'z' + w'yz' = w'z'$

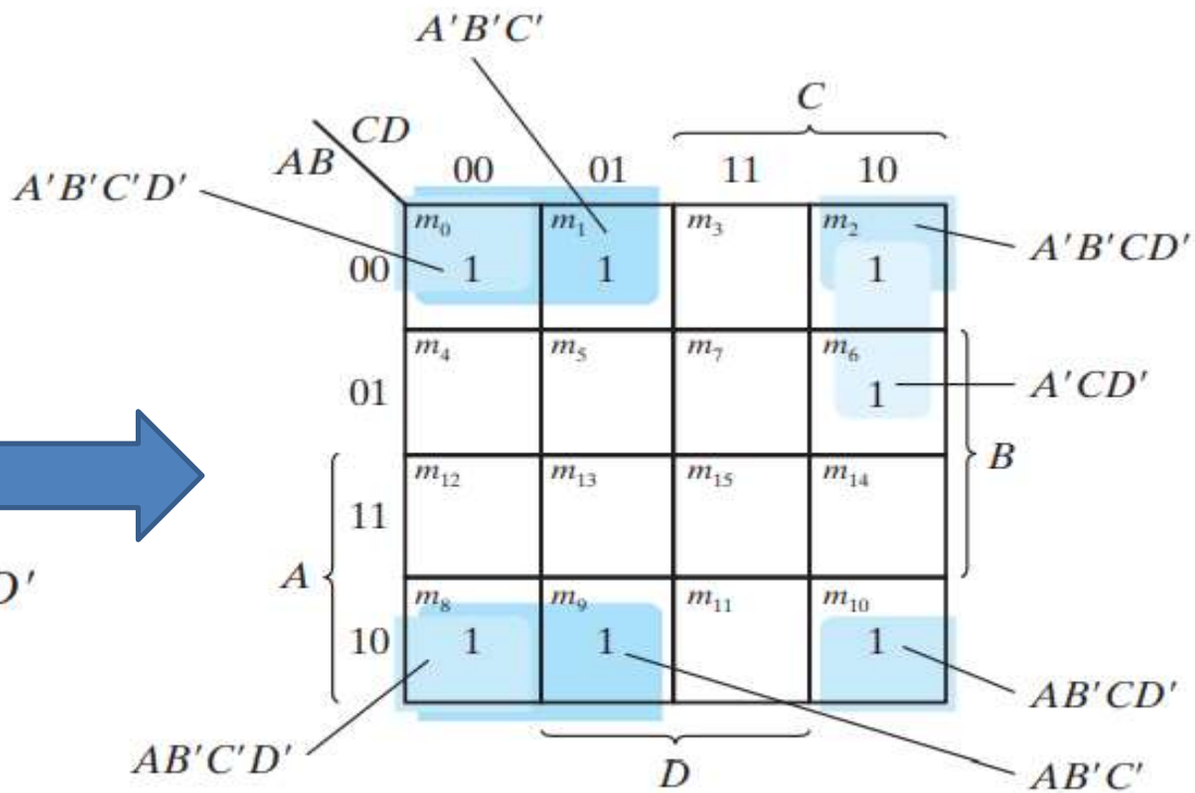
$xy'z' + xyz' = xz'$

FOUR-VARIABLE MAP

FOUR-VARIABLE K-MAP

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

$$F = B'D' + B'C' + A'CD'$$



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

FOUR-VARIABLE MAP

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).

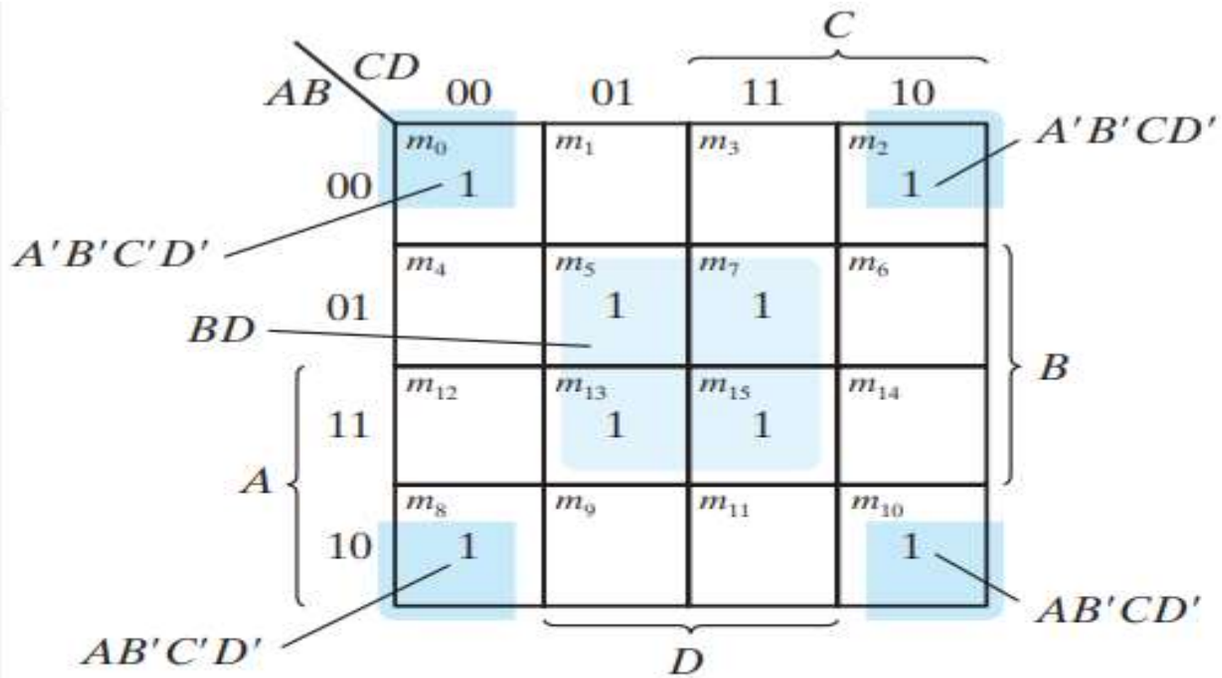
A prime implicant is a product term obtained by combining the maximum possible number of adjacent squares in the map. The prime implicants of a function can be obtained from the map by combining all possible maximum numbers of squares.

- This means that a single 1 on a map represents a prime implicant if it is not adjacent to any other 1's.
- Two adjacent 1's form a prime implicant, provided that they are not within a group of four adjacent squares.
- Four adjacent 1's form a prime implicant if they are not within a group of eight adjacent squares, and so on.
- The essential prime implicants are found by looking at each square marked with a 1 and checking the number of prime implicants that cover it. The prime implicant is essential if it is the only prime implicant that covers the minterm.

FOUR-VARIABLE MAP

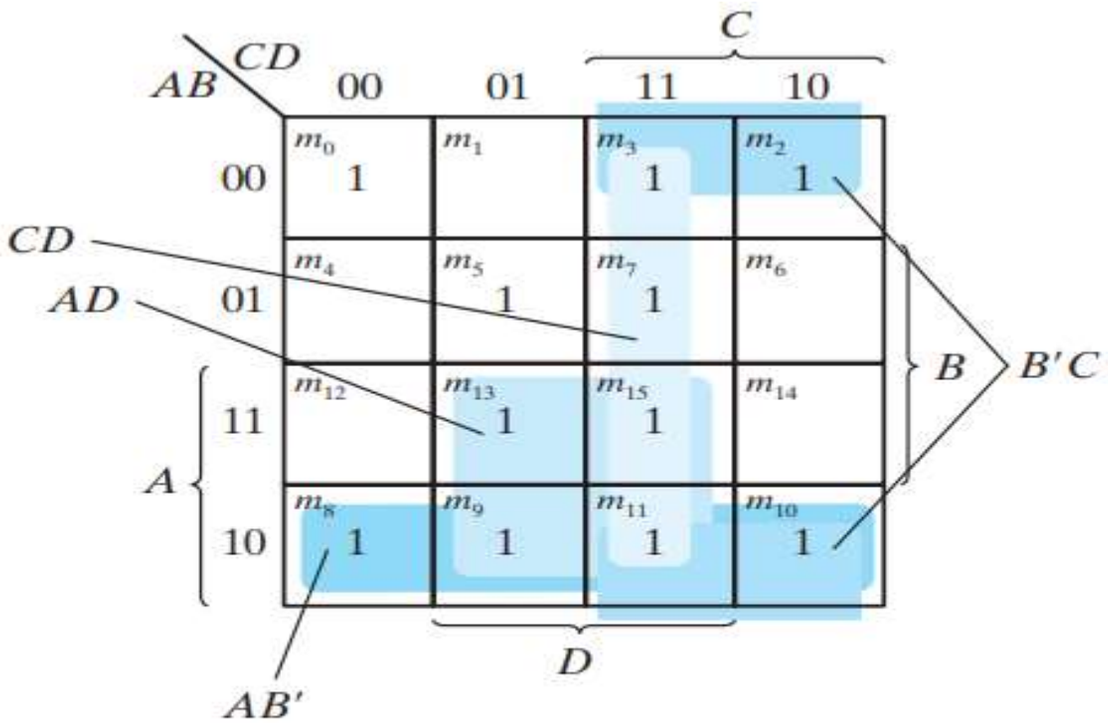
Consider the following four-variable Boolean function:

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



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'

19/08/2024 Dr. Vinod Kumar P, Associate Professor, CSE-DS ATMECE

FOUR-VARIABLE MAP

Simplification using prime implicants

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

DON'T-CARECONDITIONS

DON'T-CARECONDITIONS

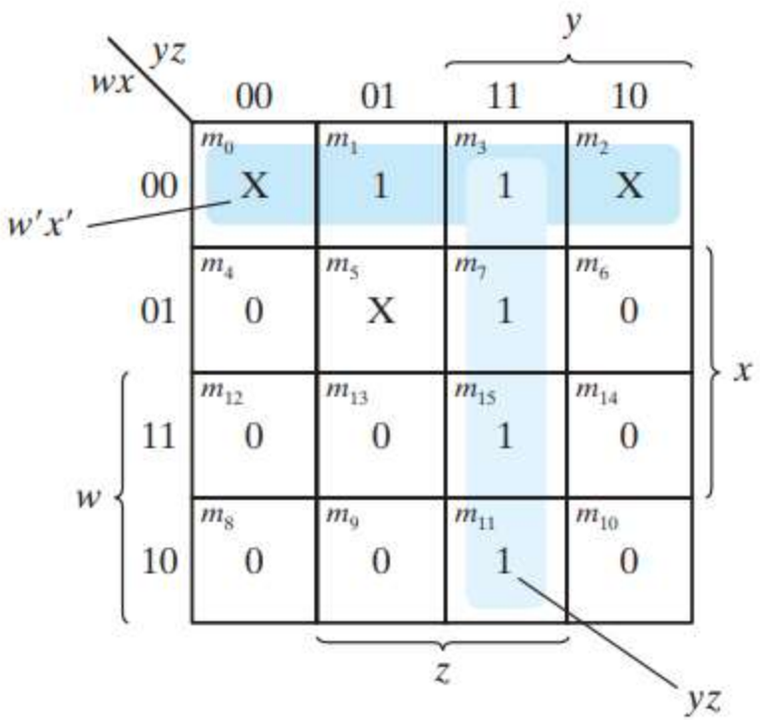
- The logical sum of the minterms associated with a Boolean function specifies the conditions under which the function is equal to 1. The function is equal to 0 for the rest of the minterms. This pair of conditions assumes that all the combinations of the values for the variables of the function are valid.
- Functions that have **unspecified** outputs for some input combinations are called incompletely specified functions . The unspecified minterms of a function are also called as **don't-care conditions** .
- These don't-care conditions can be used on a map to provide further simplification of the Boolean expression.
- A don't-care minterm is a combination of variables whose logical value is not specified. Such a minterm cannot be marked with a 1 in the map, because it would require that the function always be a 1 for such a combination. Likewise, putting a 0 on the square requires the function to be 0. 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

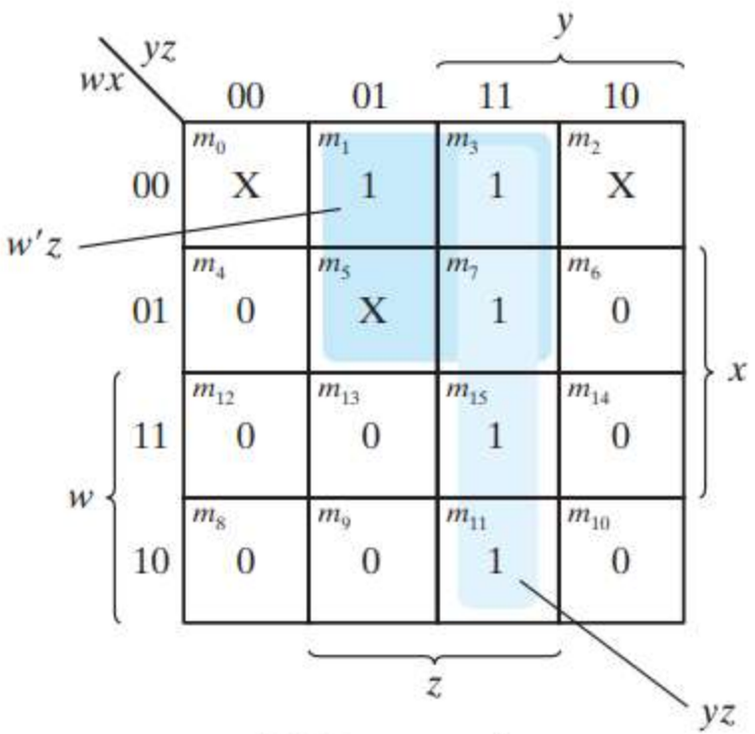
Simplify the Boolean function which has the don't-care conditions

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

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



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



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

Thank You