Analog & Digital Electronics

Course No: PH-218

Lec-29: Combinational Logic Modules

Course Instructor:

❖ **Dr. A. P. VAJPEYI**

Department of Physics,
Indian Institute of Technology Guwahati, India

Combinational Logic Circuits

➢ A Combinational logic is a circuits which employs two or more of the basic gates to form a more useful, complex function.

The Digital circuits are broadly divided in two categories –

➢ Combinational logic
– has no memory
– the present output depends only on the present input

➢ Sequential logic
– has memory
– the present output depends not only on the present input but also on the past sequence of inputs (also called memory)

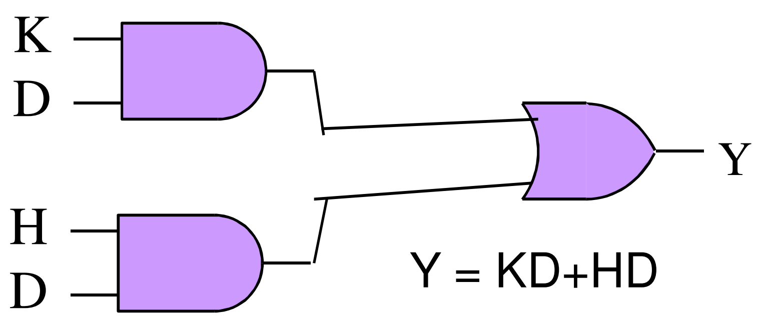
➢ The design requirements of combinational circuits may be described by one of the following ways
–(1) A set of statements, (2) Boolean expressions, (3) Truth table

The objective is to use minimum number of components to ensure low cost, less space, and power requirements.

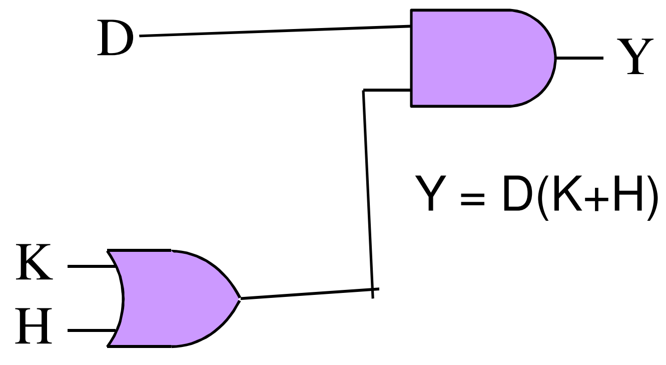
Example of Combinational Logic Circuits

- Let's design a logic for an automobile warning buzzer using combinational logic. The criteria for the activation of the warning buzzer is as follows –
  – The buzzer activates if the headlights are ON and the driver's door is opened,
  or
  – if the key is in the ignition and the door is opened.

- The logic function can be written as a Boolean expression in the form
  Y = (K and D) or (H and D) = KD+HD



Y = KD+HD



Y = D(K+H)

Combinational Logic Circuits

- ➢ Boolean algebra and algebraic simplification
- ➢ Karnaugh maps
- ➢ Designing with NAND and NOR gates

➢ MSI combinational logic circuit components

– Binary adders (Half adder and Full adder)
– Magnitude comparator
– Decoders/demultiplexers
– Data selectors/multiplexers

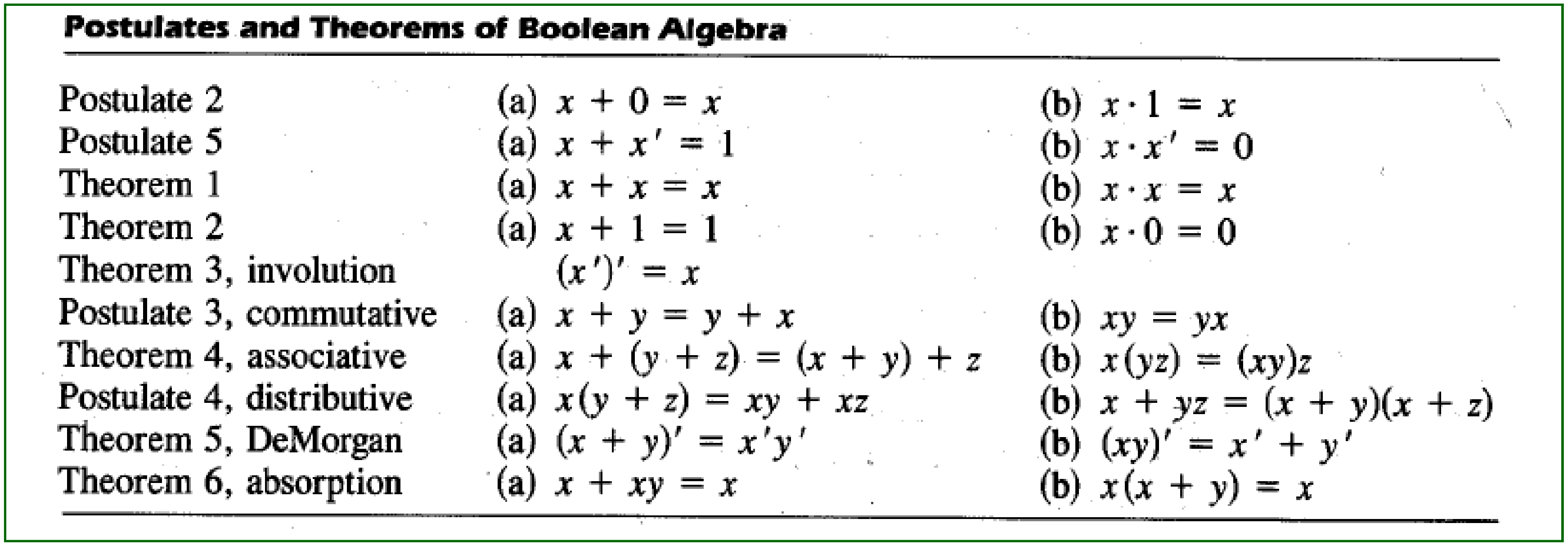
➢ PLD (programmable logic device) components

– Read-only Memory (ROM)
– Programmable Logic Array (PLA)
– Programmable Array Logic (PAL)

Boolean algebra and algebraic simplification

In 1854 George Boole introduced a systematic treatment of logic and developed for this purpose an algebraic system called *Boolean algebra.*

The OR function (X = A+B) is Boolean addition, and the AND function (X = AB) is Boolean multiplication.



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) $x(x + y) = x$

The operator precedence for evaluating Boolean expressions is (1) parentheses, (2) NOT, (3) AND, and (4) OR

De Morgan’s Theorem

Theorem 1: AND gate with inverted output is equivalent to OR gate with inverted inputs. In other words, NAND gate is equivalent to a OR gate with inverted inputs.

Theorem 1: **(A.B)’ = A’+B’**

Theorem 2: OR gate with inverted output is equivalent to AND gate with inverted inputs. In other words, NOR gate is equivalent to a AND gate with inverted inputs

Theorem 2: **(A+B)’ = A’.B’**

Standard Forms

Most Boolean reductions result in an equation in one of the two forms –

- **Product of Sums (POS) form**

Example of POS forms are: (1) X = (A+B’).(B+C) (2) X = (A+C’).(B’+E).(C+B)

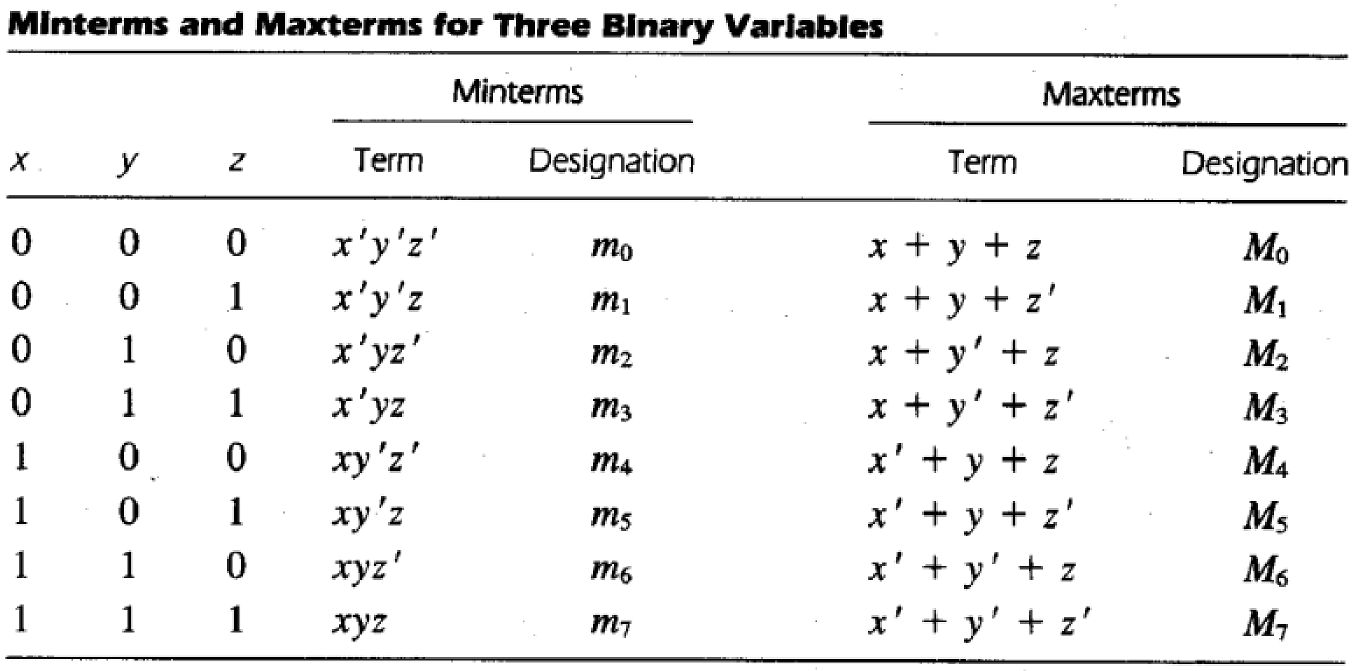
- **Sum of Product (SOP) form**

Example of SOP forms are: (1) X = AB’+BC (2) X = AC’ + B’E + CB

- SOP expression is most common and can be easily constructed using a special combinational logic gate called AND-OR-INVERT gate (AOI).

Minterms and Maxterms

A literal is a primed or unprimed variable. If each term in SOP and POS forms contains all the literals then these are known as canonical SOP and POS, respectively. Each individual term in canonical SOP and POS form is called as minterm and maxterm respectively.



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

m_j is the symbol for each minterm, where j denotes the decimal equivalent of the binary number.

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

Minterms and Maxterms

Any Boolean function can be written in Minterms and Maxterms in the below form
F(A,B,C) = Σ(1,4,5,6,7) = m1+m4++m5+m6+m7

It is sometimes convenient to express the Boolean function in its sum of minterms form. If not in this form, it can be made so by first expanding the expression into a sum of AND terms.

Example: Express the Boolean function *F = A + B´C in a sum* of minterms

Solution1 : 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 is still missing one variable C: A = (AB+AB') (C+C') = ABC+ABC'+AB'C+AB'C'

The second term *B´C is missing one variable: B'C = B'C (A+A') = AB'C+A'B'C*

F = A+B'C = ABC+ABC'+AB'C+AB'C' + *AB'C+A'B'C*
But *AB´C appears twice, and according to theorem 1*
F = ABC+ABC'+AB'C+AB'C'+A'B'C = m7+m6+m5+m4+m1 = Σ(1,4,5,6,7)

Minterms and Maxterms

Example: Express the Boolean function $F = A + B'C$ *in a sum* of minterms

Solution2:

An alternative 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

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

From the truth table, we can then read the five minterms of the function to be 1, 4, 5, 6 and 7

Minterms and Maxterms

Example: Express the Boolean function $F = xy + x'z$ *in a* product of maxterm form.

Solution:

First, convert the function into OR terms using the distributive law:

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

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

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

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

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

Combining all the terms and removing those that appear more than once

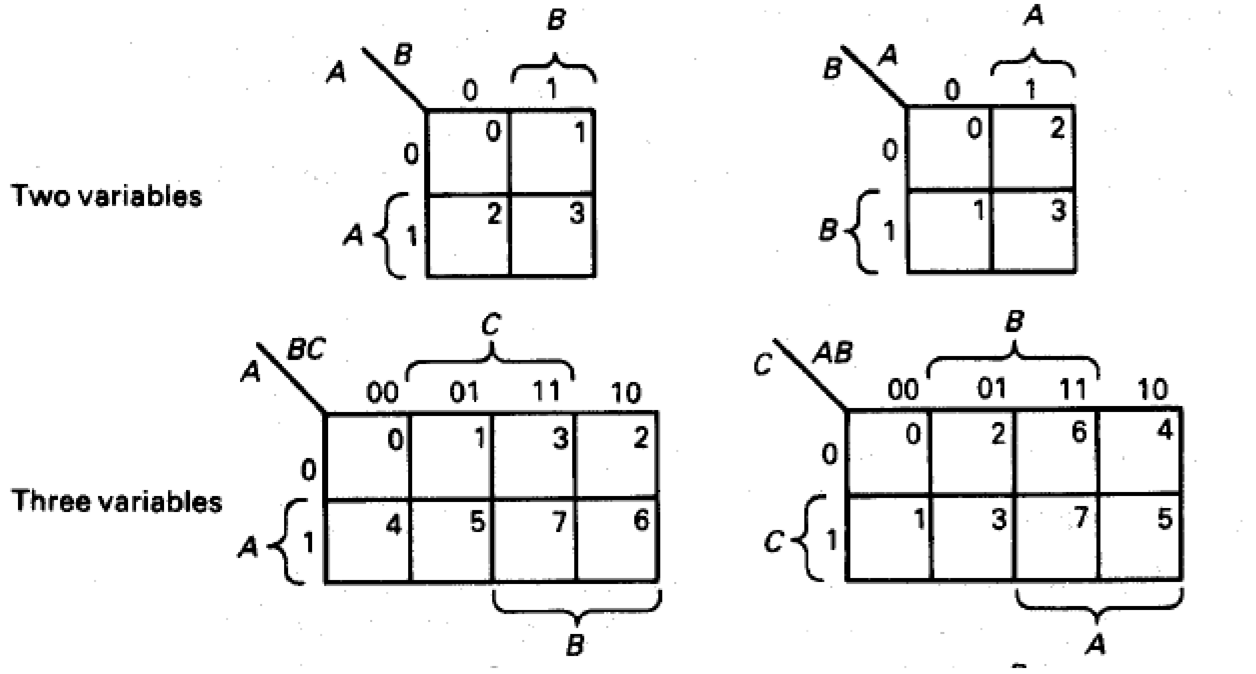
$F = (x+y+z)(x+y'+z)(x'+y+z)(x'+y+z) = MoM2M4M5 = \Pi$ (0,2,4,5)

Karnaugh map minimization method

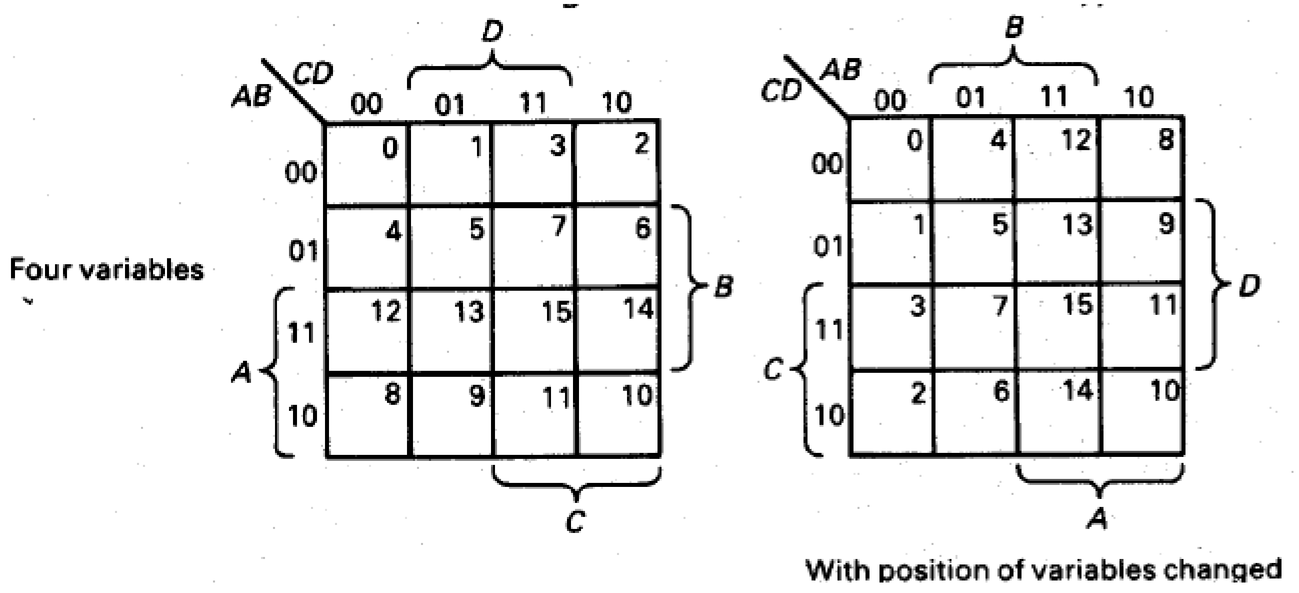
A Karnaugh map consists of a grid of squares, each square representing one canonical minterm combination of the variables or their inverse

– Arranged with squares representing minterms which differ by only one variable to be adjacent both vertically and horizontally.

– Squares on one edge of the map are regarded as adjacent to those on the opposite edge.



Karnaugh map minimization method



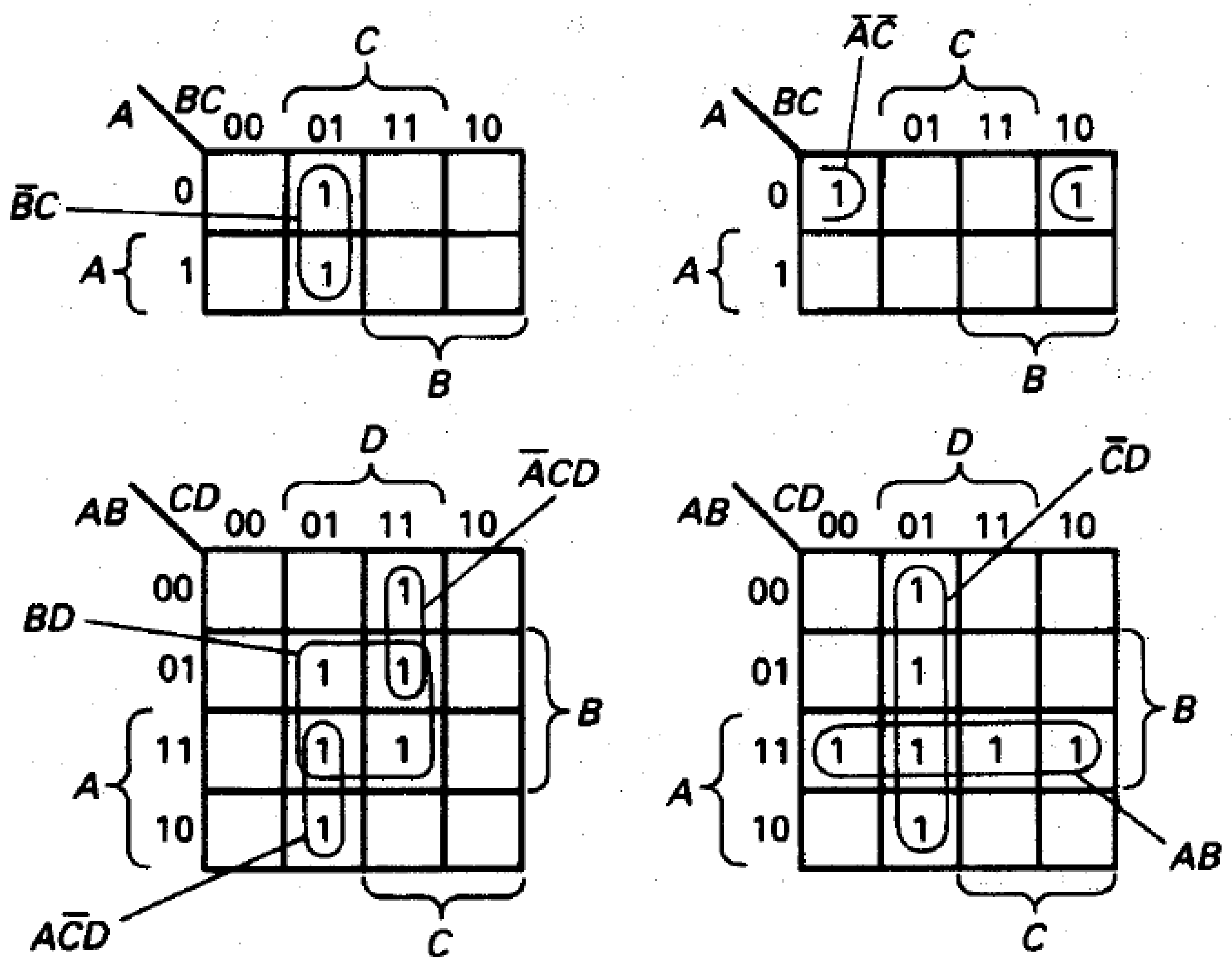
➢ The expression to be minimized should generally be in sum-of-product form.

➢ The function is ‘mapped’ onto the Karnaugh map by marking a 1 in those squares corresponding to the terms in the expression to be simplified.

➢ If two or more pairs are also adjacent, these can also be combined using the same theorem.

➢ The minimization procedure consists of recognizing multiple pairs in terms of 2, 4 or 8 cells.

Karnaugh map minimization method



Examples: Karnaugh map minimization method

Example: Simplify the Boolean function X = *A'B+A'B'C'+ ABC'+AB'C'*

Solution: First write the Boolean function interms of canonical minterm
A'B = A'B(C+C') = A'BC+A'BC'

X = A'BC+A'BC' *+ A'B'C'+ ABC'+AB'C'*

Terms	C'	C
A'B'	1	
A'B	1	1
AB	1	
AB'	1	

X = C'+A'B