Chapter 2

Combinatorial Logic Circuits

Basanta Joshi, PhD

Overview

2-1 Binary Logic and Gates

2-2 Boolean Algebra

2-3 Standard Forms

2-4 Map Simplification

2-5 Universal Gates

2-6 Integrated Circuits

2-1 Binary Logic and Gates

- Digital circuits are hardware components (based on transistors) that manipulate binary information
- We model the transistor-based electronic circuits as logic gates.
  - Designer can ignore the internal electronics of a gate

Binary Logic

- **Binary variables take on one of two values.**
- **Logical operators operate on binary values and binary variables.**
- **Basic logical operators are the logic functions AND, OR and NOT.**
- **Logic gates implement logic functions.**
- **Boolean Algebra: a useful mathematical system for specifying and transforming logic functions.**
- **We study Boolean algebra as a foundation for designing and analyzing digital systems!**

Binary Variables

- **Recall that the two binary values have different names:**
  - **True/False**
  - **On/Off**
  - **Yes/No**
  - **1/0**
- **We use 1 and 0 to denote the two values.**
- **Variable identifier examples:**
  - **A, B, y, z, or X_1 for now**
  - **RESET, START_IT, or ADD1 later**

Logical Operations

- **The three basic logical operations are:**
  - **AND**
  - **OR**
  - **NOT**
- **AND is denoted by a dot (·).**
- **OR is denoted by a plus (+).**
- **NOT is denoted by an overbar (¯), a single quote mark (') after, or (~) before the variable.**

Notation Examples

- **Examples:**
  - **$Y = A \cdot B$ is read "Y is equal to A AND B."**
  - **$z = x + y$ is read "z is equal to x OR y."**
  - **$X = \overline{A}$ is read "X is equal to NOT A."**

- **Note: The statement:**

  1 + 1 = 2 (read "one plus one equals two")

  is not the same as

  1 + 1 = 1 (read "1 or 1 equals 1").

Operator Definitions

- **Operations are defined on the values "0" and "1" for each operator:**

AND	OR	NOT
$0 \cdot 0 = 0$	$0 + 0 = 0$	$\overline{0} = 1$
$0 \cdot 1 = 0$	$0 + 1 = 1$	$\overline{1} = 0$
$1 \cdot 0 = 0$	$1 + 0 = 1$	
$1 \cdot 1 = 1$	$1 + 1 = 1$	

Truth Tables

- ***Truth table*** **− a tabular listing of the values of a function for all possible combinations of values on its arguments**
- **Example: Truth tables for the basic logic operations:**

AND		
X	**Y**	**Z = X·Y**
0	0	0
0	1	0
1	0	0
1	1	1

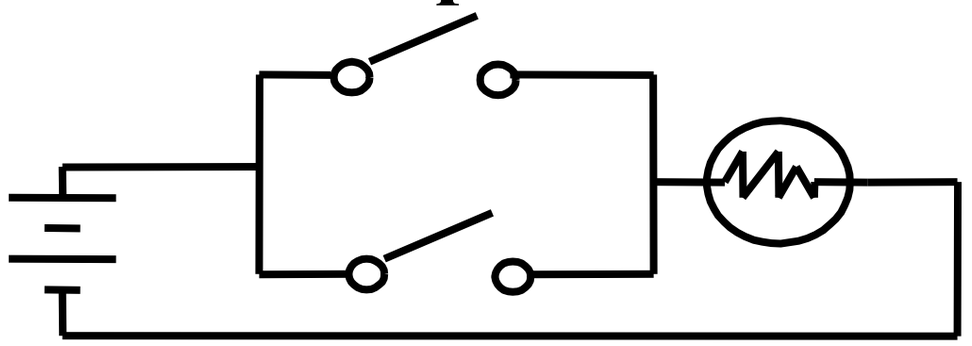
OR		
X	**Y**	**Z = X+Y**
0	0	0
0	1	1
1	0	1
1	1	1

NOT	
X	**$Z = \overline{X}$**
0	1
1	0

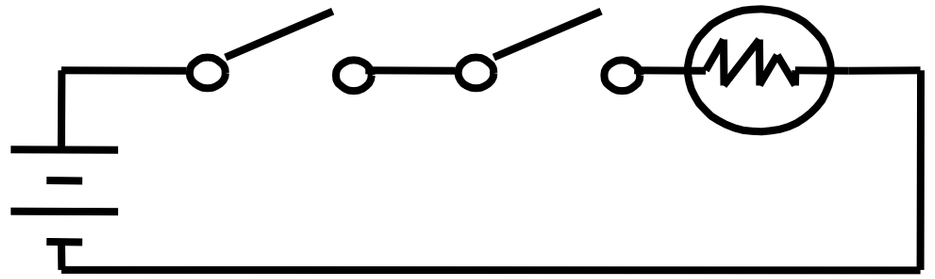
Logic Function Implementation

- **Using Switches**
  - **For inputs:**
    - **logic 1 is switch closed**
    - **logic 0 is switch open**
  - **For outputs:**
    - **logic 1 is light on**
    - **logic 0 is light off.**
- **NOT uses a switch such that:**
  - **logic 1 is switch open**
  - **logic 0 is switch closed**

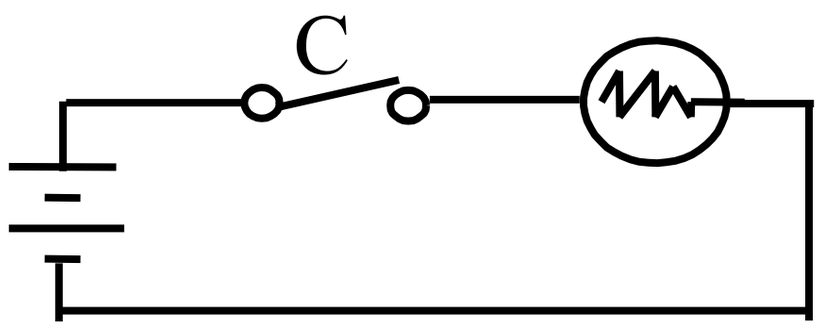
Switches in parallel => OR



Switches in series => AND

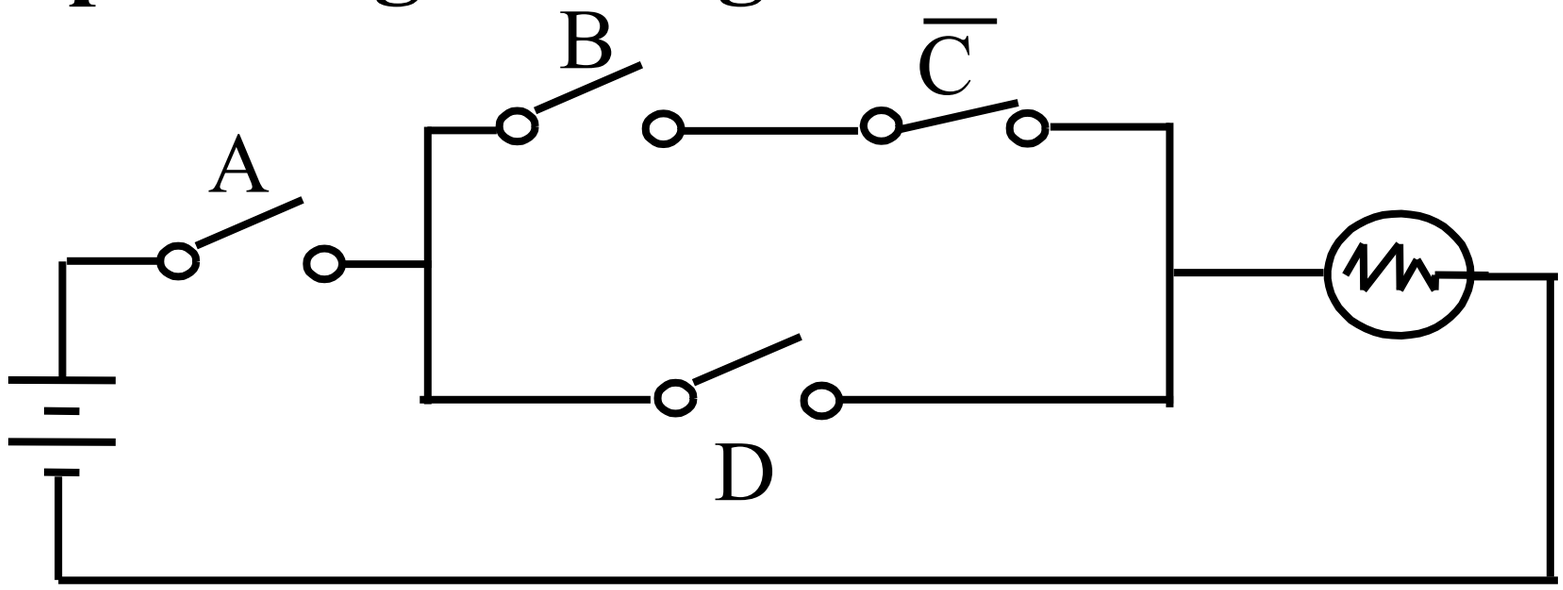


Normally-closed switch => NOT



Logic Function Implementation (Continued)

- **Example: Logic Using Switches**



- **Light is on (L = 1) for**

  $$L(A, B, C, D) =$$

  and off (L = 0), otherwise.

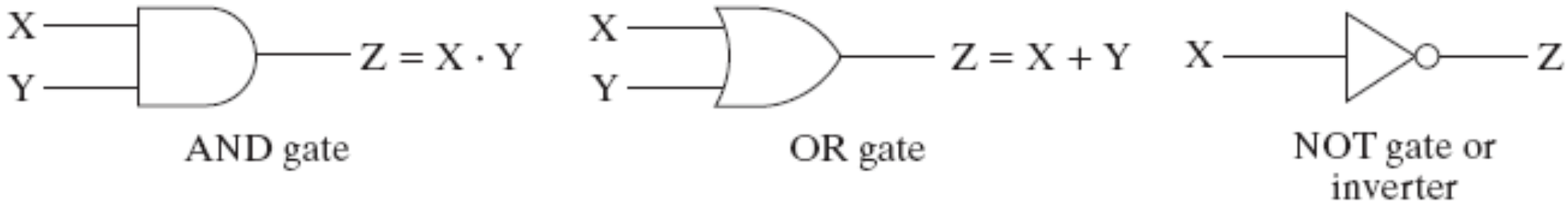
- **Useful model for relay circuits and for CMOS gate circuits, the foundation of current digital logic technology**

Logic Gates

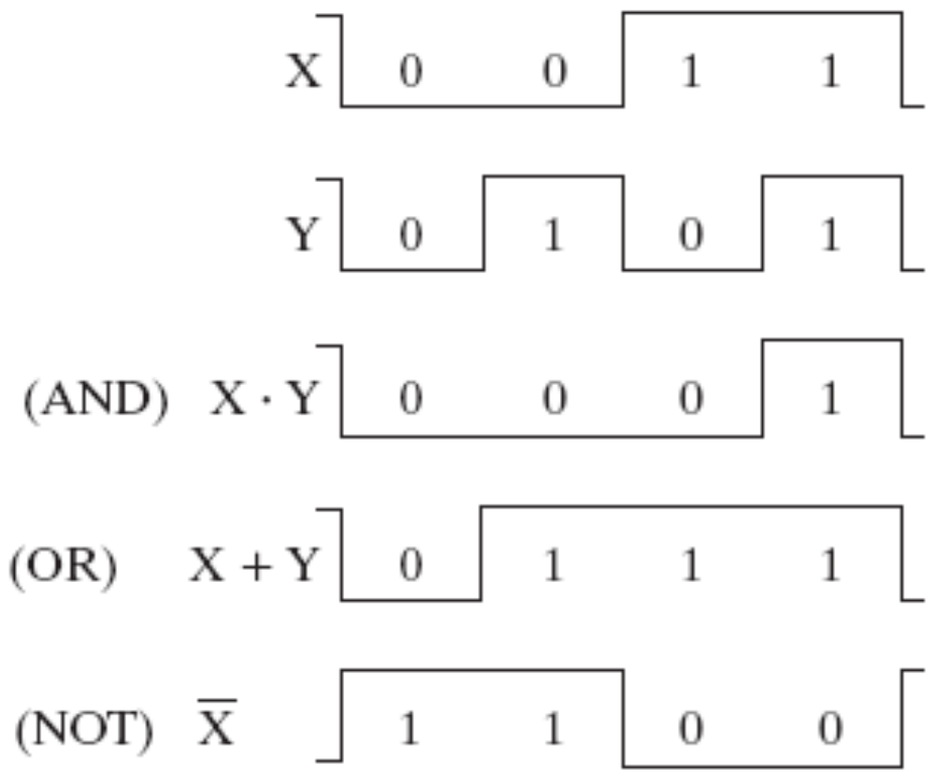
- **In the earliest computers, switches were opened and closed by magnetic fields produced by energizing coils in *relays*. The switches in turn opened and closed the current paths.**
- **Later, *vacuum tubes* that open and close current paths electronically replaced relays.**
- **Today, *transistors* are used as electronic switches that open and close current paths.**
- **Optional: Chapter 6 – Part 1: The Design Space**

Logic Gate Symbols and Behavior

- **Logic gates have special symbols:**



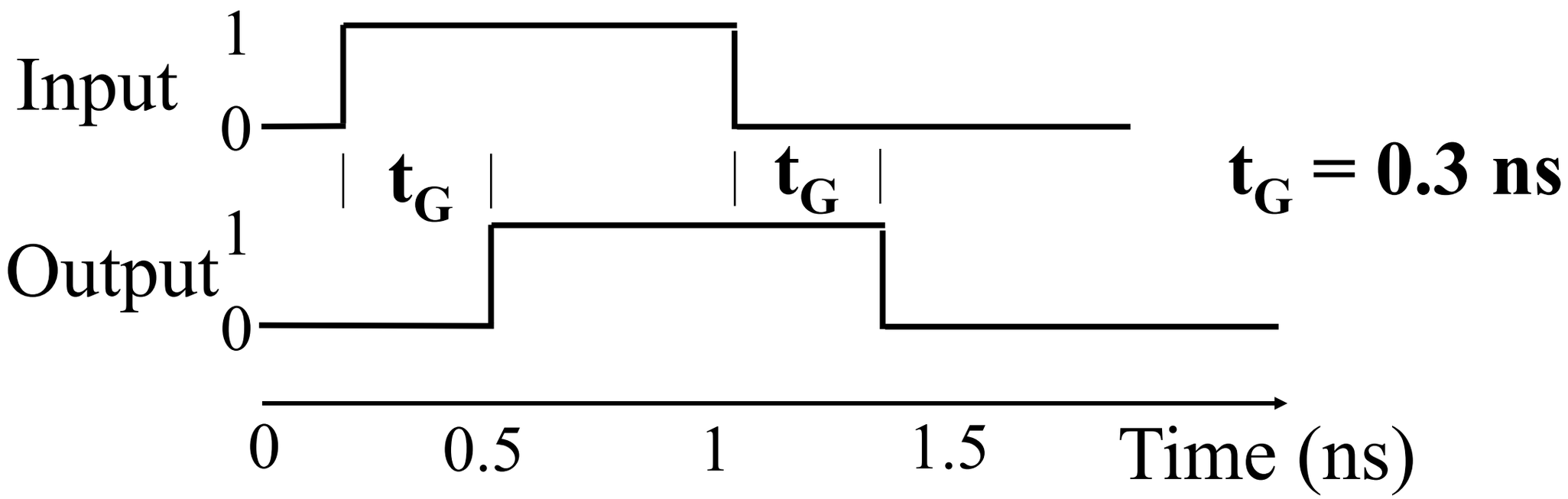
(a) Graphic symbols



(b) Timing diagrams

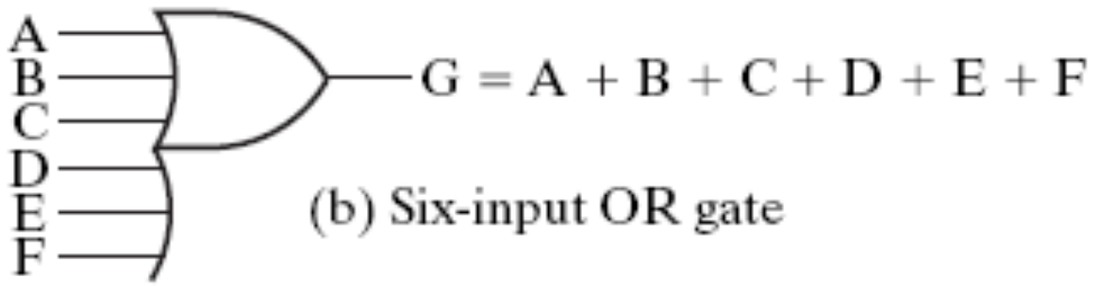
Gate Delay

- **In actual physical gates, if one or more input changes causes the output to change, the output change does not occur instantaneously.**
- **The delay between an input change(s) and the resulting output change is the *gate delay* denoted by t_G:**



AND and OR gates with more than two inputs

(a) Three-input AND gate



(b) Six-input OR gate

2-2 Boolean Algebra

- Boolean expression: a expression formed by binary variables, for example $D\overline{X} + A$
- Boolean function: a binary variable identifying the function followed by an equal sign and a Boolean expression for example

$$L(D, X, A) = D\overline{X} + A$$

Truth table and Logic circuit

For the Boolean function

$$L(D, X, A) = D\overline{X} + A$$

TABLE 2-2
Truth Table for the Function L = D$\overline{X}$ + A

D	X	A	L
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

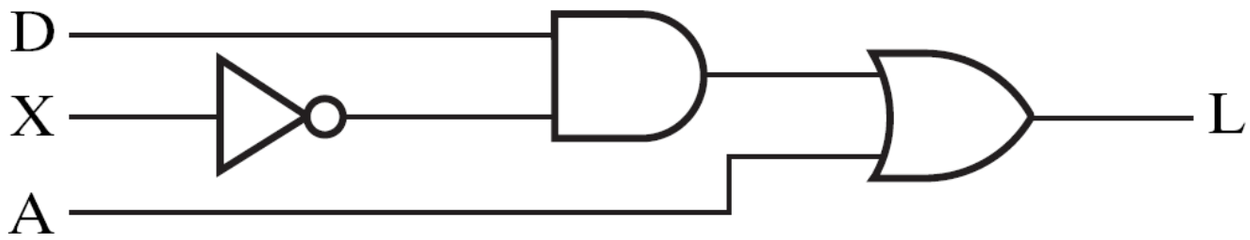


Fig. 2.3 Logic Circuit Diagram

Basic identities of Boolean Algebra

- **An algebraic structure defined on a set of at least two elements together with three binary operators (denoted +, · and -) that satisfies the following basic identities:**

1.	$X + 0 = X$	2.	$X \cdot 1 = X$
3.	$X + 1 = 1$	4.	$X \cdot 0 = 0$
5.	$X + X = X$	6.	$X \cdot X = X$
7.	$X + \overline{X} = 1$	8.	$X \cdot \overline{X} = 0$
9.	$\overline{\overline{X}} = X$		

10.	$X + Y = Y + X$	11.	$XY = YX$ Commutative
12.	$(X + Y) + Z = X + (Y + Z)$	13.	$(XY)Z = X(YZ)$ Associative
14.	$X(Y + Z) = XY + XZ$	15.	$X + YZ = (X + Y)(X + Z)$ Distributive
16.	$\overline{X + Y} = \overline{X} \cdot \overline{Y}$	17.	$\overline{X \cdot Y} = \overline{X} + \overline{Y}$ DeMorgan's

Truth Table to Verify DeMorgan's Theorem

TABLE 2-4
Truth Tables to Verify DeMorgan's Theorem

A) X	Y	X + Y	$\overline{X+Y}$	B) X	Y	$\overline{X}$	$\overline{Y}$	$\overline{X} \cdot \overline{Y}$
0	0	0	1	0	0	1	1	1
0	1	1	0	0	1	1	0	0
1	0	1	0	1	0	0	1	0
1	1	1	0	1	1	0	0	0

Extension of DeMorgan's Theorem:

$$\overline{X_1 + X_2 + \cdots + X_n} = \overline{X}_1 \overline{X}_2 \cdots \overline{X}_n$$

Some Properties of Identities & the Algebra

- **If the meaning is unambiguous, we leave out the symbol "·"**
- **The dual of an algebraic expression is obtained by interchanging + and · and interchanging 0's and 1's.**
- **The identities appear in dual pairs. When there is only one identity on a line the identity is self-dual, i. e., the dual expression = the original expression.**

Some Properties of Identities & the Algebra (Continued)

- **Unless it happens to be self-dual, the dual of an expression does not equal the expression itself.**
- **Example: $F = (A + \overline{C}) \cdot B + 0$**

  dual $F = (A \cdot C + B) \cdot 1 = A \cdot C + B$
- **Example: $G = X \cdot Y + \overline{(W + Z)}$**

  dual G =
- **Example: $H = A \cdot B + A \cdot C + B \cdot C$**

  dual H =
- **Are any of these functions self-dual?**

Boolean Operator Precedence

- **The order of evaluation in a Boolean expression is:**
  1. Parentheses
  2. NOT
  3. AND
  4. OR

- **Example:** F = A(B + C)(C + D)

Boolean Algebraic Manipulation

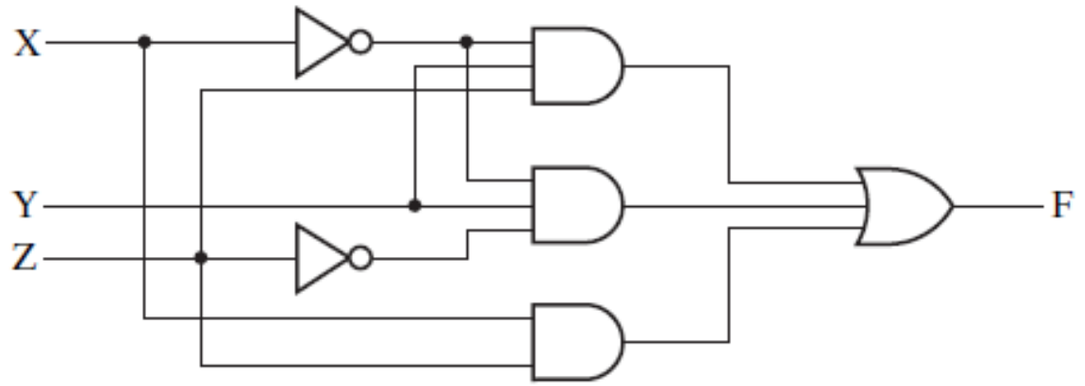
$$F = \bar{X}YZ + \bar{X}Y\bar{Z} + XZ$$

$$= \bar{X}Y(Z + \bar{Z}) + XZ$$

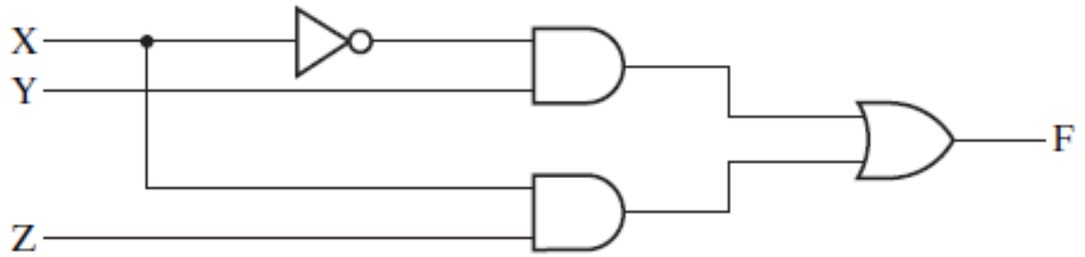
$$= \bar{X}Y + XZ$$

TABLE 2-5
Truth Table for Boolean Function

X	Y	Z	(a) F	(b) F
0	0	0	0	0
0	0	1	0	0
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1



(a) F = $\overline{X}YZ + \overline{X}Y\overline{Z} + XZ$



(b) F = $\overline{X}Y + XZ$

Fig. 2-4

Boolean Algebraic Manipulation

- $AB + \overline{A}C + BC = AB + \overline{A}C$ (Consensus Theorem)

Proof Steps	Justification (identity or theorem)
$AB + \overline{A}C + BC$	
$= AB + \overline{A}C + 1 \cdot BC$	?
$= AB + AC + (A + A) \cdot BC$	?
$=$	

What is the duality?

Example: Complementing Function

$$F_1 = \overline{X}YZ + X\overline{Y}Z$$

$$F_2 = X(\overline{Y}\overline{Z} + YZ)$$

$$\overline{F}_1 = ?$$

$$\overline{F}_2 = ?$$

- By DeMorgan's Theorem (Example 2-2)
- By duality (Example 2-3)

2-3 Canonical Forms

- **It is useful to specify Boolean functions in a form that:**
  - **Allows comparison for equality.**
  - **Has a correspondence to the truth tables**
- **Canonical Forms in common usage:**
  - **Sum of Minterms (SOM)**
  - **Product of Maxterms (POM)**

Minterms

- **Minterms are AND terms with every variable present in either true or complemented form.**
- **Given that each binary variable may appear normal (e.g., x) or complemented (e.g., $\overline{x}$), there are 2^n minterms for *n* variables.**
- **Example: Two variables (X and Y)produce 2 x 2 = 4 combinations:**

  $X\overline{Y}$ (both normal)
  $\overline{X}Y$ (X normal, Y complemented)
  $\overline{X}\overline{Y}$ (X complemented, Y normal)
  XY (both complemented)

- **Thus there are four minterms of two variables.**

Maxterms

- **Maxterms are OR terms with every variable in true or complemented form.**
- **Given that each binary variable may appear normal (e.g., x) or complemented (e.g., $\overline{x}$), there are 2^n maxterms for n variables.**

- **Example: Two variables (X and Y) produce 2 x 2 = 4 combinations:**

  $X + Y$ (both normal)

  $X + \overline{Y}$ (x normal, y complemented)

  $\overline{X} + Y$ (x complemented, y normal)

  $\overline{X} + \overline{Y}$ (both complemented)

Maxterms and Minterms

- **Examples: Two variable minterms and maxterms.**

Index	Minterm	Maxterm
0	$\overline{x}\,\overline{y}$	$x + y$
1	$\overline{x}\,y$	$x + \overline{y}$
2	$x\,\overline{y}$	$\overline{x} + y$
3	$x\,y$	$\overline{x} + \overline{y}$

- **The index above is important for describing which variables in the terms are true and which are complemented.**

Minterms for three variables

TABLE 2-6
Minterms for Three Variables

X	Y	Z	Product Term	Symbol	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
0	0	0	$\overline{X}\,\overline{Y}\,\overline{Z}$	m_0	1	0	0	0	0	0	0	0
0	0	1	$\overline{X}\,\overline{Y}Z$	m_1	0	1	0	0	0	0	0	0
0	1	0	$\overline{X}Y\overline{Z}$	m_2	0	0	1	0	0	0	0	0
0	1	1	$\overline{X}YZ$	m_3	0	0	0	1	0	0	0	0
1	0	0	$X\overline{Y}\,\overline{Z}$	m_4	0	0	0	0	1	0	0	0
1	0	1	$X\overline{Y}Z$	m_5	0	0	0	0	0	1	0	0
1	1	0	$XY\overline{Z}$	m_6	0	0	0	0	0	0	1	0
1	1	1	XYZ	m_7	0	0	0	0	0	0	0	1

Maxterms for three variables

TABLE 2-7
Maxterms for Three Variables

X	Y	Z	Sum Term	Symbol	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7
0	0	0	$X+Y+Z$	M_0	0	1	1	1	1	1	1	1
0	0	1	$X+Y+\overline{Z}$	M_1	1	0	1	1	1	1	1	1
0	1	0	$X+\overline{Y}+Z$	M_2	1	1	0	1	1	1	1	1
0	1	1	$X+\overline{Y}+\overline{Z}$	M_3	1	1	1	0	1	1	1	1
1	0	0	$\overline{X}+Y+Z$	M_4	1	1	1	1	0	1	1	1
1	0	1	$\overline{X}+Y+\overline{Z}$	M_5	1	1	1	1	1	0	1	1
1	1	0	$\overline{X}+\overline{Y}+Z$	M_6	1	1	1	1	1	1	0	1
1	1	1	$\overline{X}+\overline{Y}+\overline{Z}$	M_7	1	1	1	1	1	1	1	0

Minterm and Maxterm Relationship

- **Review: DeMorgan's Theorem**
  $\overline{x \cdot y} = \bar{x} + \bar{y}$ **and** $\overline{x + y} = \bar{x} \cdot \bar{y}$
- **Two-variable example:**
  $M_2 = \bar{x} + y$ **and** $m_2 = x \cdot \bar{y}$
  Thus M_2 **is the complement of** m_2 **and vice-versa.**
- **Since DeMorgan's Theorem holds for *n* variables, the above holds for terms of *n* variables**
- **giving:**

$$M_i = \overline{m}_i \text{ and } m_i = \overline{M}_i$$

Thus M_i **is the complement of** m_i**.**

Function Tables for Both

- **Minterms of 2 variables**

x y	m_0	m_1	m_2	m_3
0 0	1	0	0	0
0 1	0	1	0	0
1 0	0	0	1	0
1 1	0	0	0	1

Maxterms of 2 variables

x y	M_0	M_1	M_2	M_3
0 0	0	1	1	1
0 1	1	0	1	1
1 0	1	1	0	1
1 1	1	1	1	0

- **Each column in the maxterm function table is the complement of the column in the minterm function table since M_i is the complement of m_i.**

Observations

- **In the function tables:**
  - **Each minterm has one and only one 1 present in the 2^n terms (a minimum of 1s). All other entries are 0.**
  - **Each maxterm has one and only one 0 present in the 2^n terms All other entries are 1 (a maximum of 1s).**
- **We can implement any function by "ORing" the minterms corresponding to "1" entries in the function table. These are called the minterms of the function.**
- **We can implement any function by "ANDing" the maxterms corresponding to "0" entries in the function table. These are called the maxterms of the function.**
- **This gives us two canonical forms:**
  - **Sum of Minterms (SOM)**
  - **Product of Maxterms (POM)**

for stating any Boolean function.

Conversion of Minterm and Maxterm

$$F = \overline{X}\overline{Y}\overline{Z} + \overline{X}Y\overline{Z} + X\overline{Y}Z + XYZ = m_0 + m_2 + m_5 + m_7 = \sum m(0, 2, 5, 7)$$

$$\overline{F} = \overline{X}\overline{Y}Z + \overline{X}YZ + X\overline{Y}\overline{Z} + XY\overline{Z} = m_1 + m_3 + m_4 + m_6 = \sum m(1, 3, 4, 6)$$

Conversion of Minterm and Maxterm

$$\overline{F} = m_1 + m_3 + m_4 + m_6$$

$$\Rightarrow F = \overline{m_1 + m_3 + m_4 + m_6} = \overline{m_1} \cdot \overline{m_3} \cdot \overline{m_4} \cdot \overline{m_6}$$

$$\Rightarrow F = M_1 \cdot M_3 \cdot M_4 \cdot M_6 = (X + Y + \overline{Z})(X + \overline{Y} + Z)(\overline{X} + Y + Z)(\overline{X} + \vec{Y} + Z)$$

$$= \prod M(1, 3, 4, 6)$$

Canonical Sum of Minterms

- **Any Boolean function can be expressed as a Sum of Minterms.**
  - **For the function table, the minterms used are the terms corresponding to the 1's**
  - **For expressions, expand all terms first to explicitly list all minterms. Do this by "ANDing" any term missing a variable v with a term $(v + \bar{v})$.**
- **Example: Implement $f = x + \bar{x}\,\bar{y}$ as a sum of minterms.**

  First expand terms: $f = x(y + \bar{y}) + \bar{x}\,\bar{y}$

  Then distribute terms: $f = xy + x\bar{y} + \bar{x}\,\bar{y}$

  Express as sum of minterms: $f = m_3 + m_2 + m_0$

Another SOM Example

Expand by using truth table

$$E = \overline{Y} + \overline{XZ}$$

According to truth table Table 2-8,

$$E = \sum m(0,1,2,4,5) = \prod M(???)$$

TABLE 2-8
Boolean Functions of Three Variables

(a) X	Y	Z	F	$\overline{F}$	(b) X	Y	Z	E
0	0	0	1	0	0	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	1	0	0	1	0	1
0	1	1	0	1	0	1	1	0
1	0	0	0	1	1	0	0	1
1	0	1	1	0	1	0	1	1
1	1	0	0	1	1	1	0	0
1	1	1	1	0	1	1	1	0

Standard Sum-of-Products (SOP)

- A sum of minterms form for *n* variables can be written down directly from a truth table.
  - Implementation of this form is a two-level network of gates such that:
    - The first level consists of *n*-input AND gates, and
    - The second level is a single OR gate (with fewer than 2^n inputs).
- This form often can be simplified so that the corresponding circuit is simpler.

Standard Sum-of-Products (SOP)

Example: $F = \overline{Y} + \overline{X}Y\overline{Z} + XY$

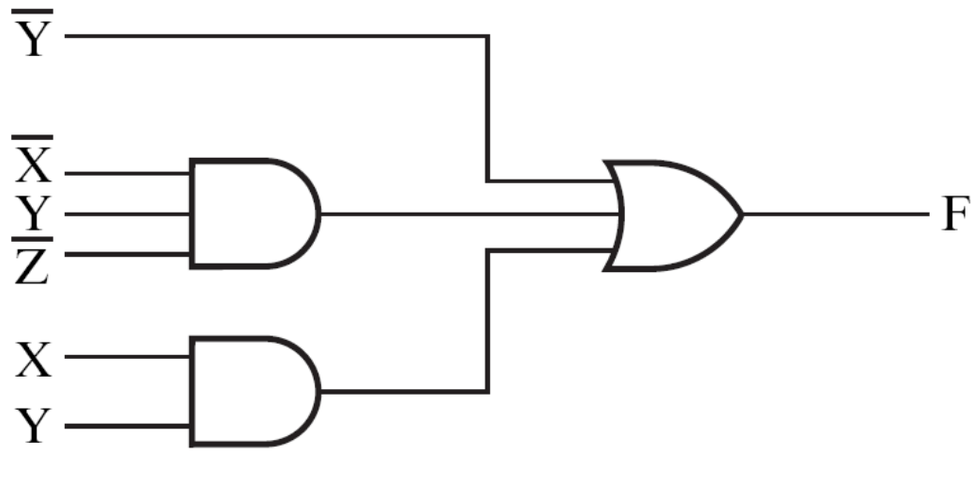


Fig. 2-5

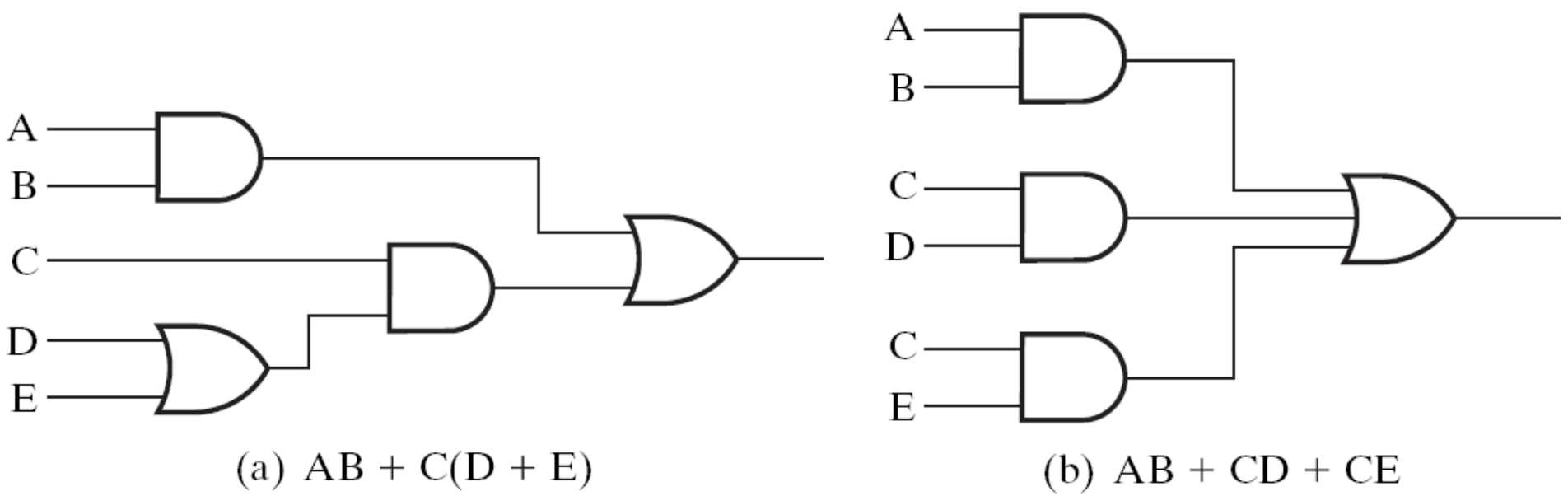
- **a two-level implementation/two-level circuit**

Product-of-Sums (POS): $F = X(\overline{Y} + Z)(X + Y + \overline{Z})$

What's the implementation?

Convert non-SOP expression to SOP expression

$$F = AB + C(D + E) = AB + CD + CE$$

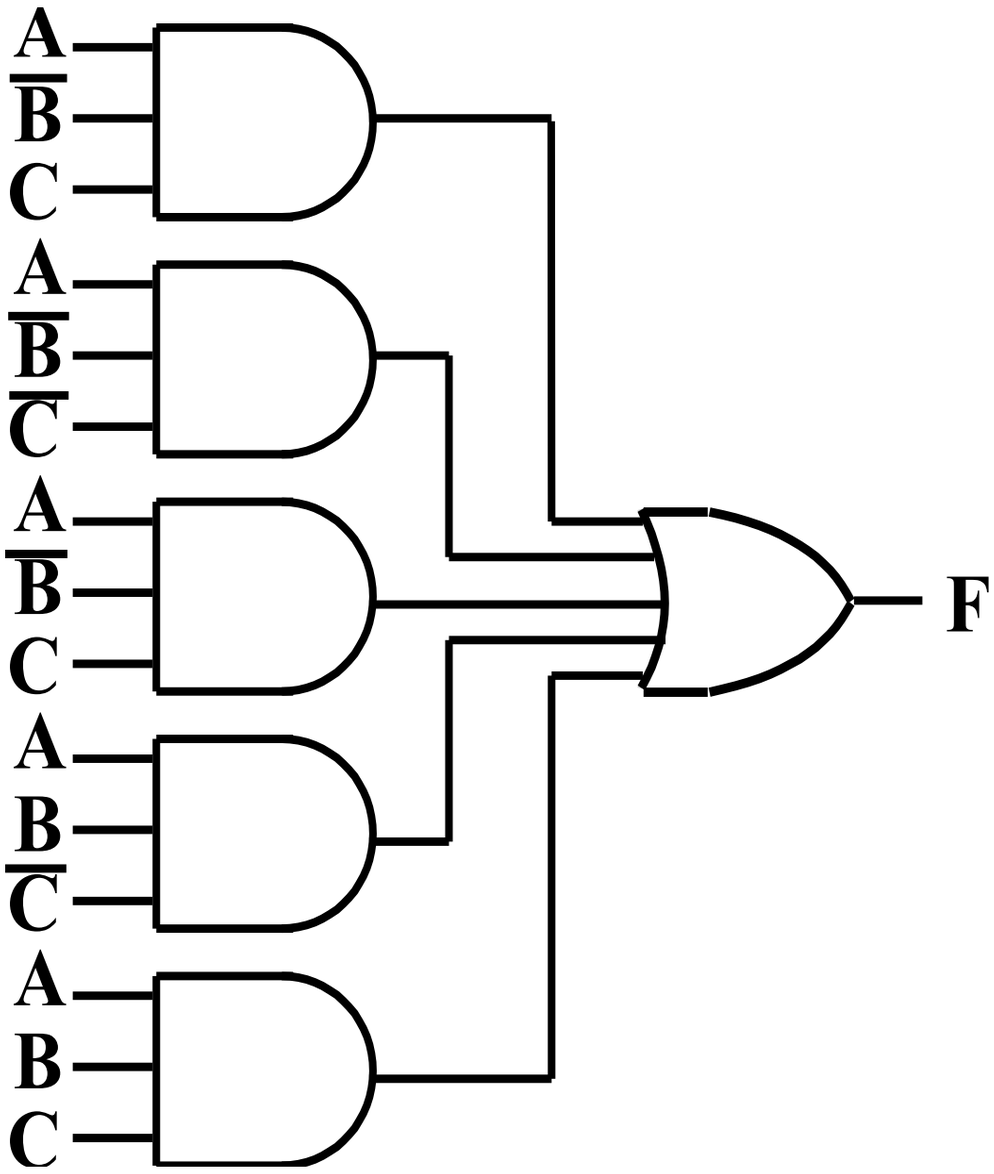


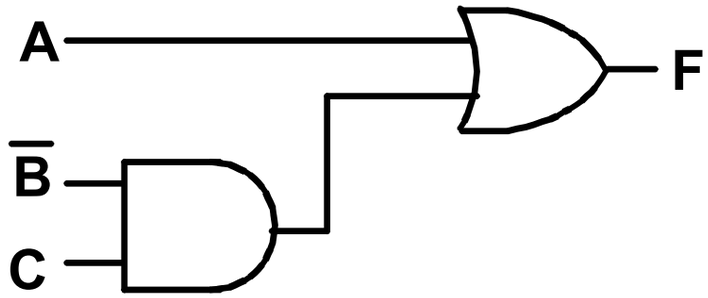
(a) AB + C(D + E) (b) AB + CD + CE

- The decision whether to use a two-level or multiple-level implementation is complex.
  - no. of gates
  - No. of gate inputs
  - amount of time delay

Simplification of two-level implementation of SOP expression

- The two implementations for F are shown below – it is quite apparent which is simpler!





SOP and POS Observations

- **The previous examples show that:**
  - **Canonical Forms (Sum-of-minterms, Product-of-Maxterms), or other standard forms (SOP, POS) differ in complexity**
  - **Boolean algebra can be used to manipulate equations into simpler forms.**
  - **Simpler equations lead to simpler two-level implementations**
- **Questions:**
  - **How can we attain a "simplest" expression?**
  - **Is there only one minimum cost circuit?**
  - **The next part will deal with these issues.**

Circuit Optimization

- **Goal: To obtain the simplest implementation for a given function**
- **Optimization is a more formal approach to simplification that is performed using a specific procedure or algorithm**
- **Optimization requires a cost criterion to measure the simplicity of a circuit**
- **Distinct cost criteria we will use:**
  - **Literal cost (L)**
  - **Gate input cost (G)**
  - **Gate input cost with NOTs (GN)**

Literal Cost

- **Literal – a variable or it complement**
- **Literal cost – the number of literal appearances in a Boolean expression corresponding to the logic circuit diagram**
- **Examples:**
  - **F = AB+C(D+E)** **L = 5**
  - **F = AB+CD+CE** **L = 6**
  - **Which solution is best?**

$$G = ABCD + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \qquad L = 8$$

$$G = (\overline{A} + B)(\overline{B} + C)(\overline{C} + D)(\overline{D} + A) \qquad L = 8$$

Which is best? The same complexity?

Gate Input Cost

- **Gate input costs - the number of inputs to the gates in the implementation corresponding exactly to the given equation or equations. (G - inverters not counted, GN - inverters counted)**
- **For SOP and POS equations, it can be found from the equation(s) by finding the sum of:**
  - **all literal appearances**
  - **the number of terms excluding single literal terms,(G) and**
  - **optionally, the number of distinct complemented single literals (GN).**
- **Example:**

$$G = ABCD + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \qquad G = 8 + 2 = 10, \quad GN = 8 + 2 + 4 = 14$$

$$G = (\overline{A} + B)(\overline{B} + C)(\overline{C} + D)(\overline{D} + A) \qquad G = ? \quad GN = ?$$

  - **Which solution is best?**

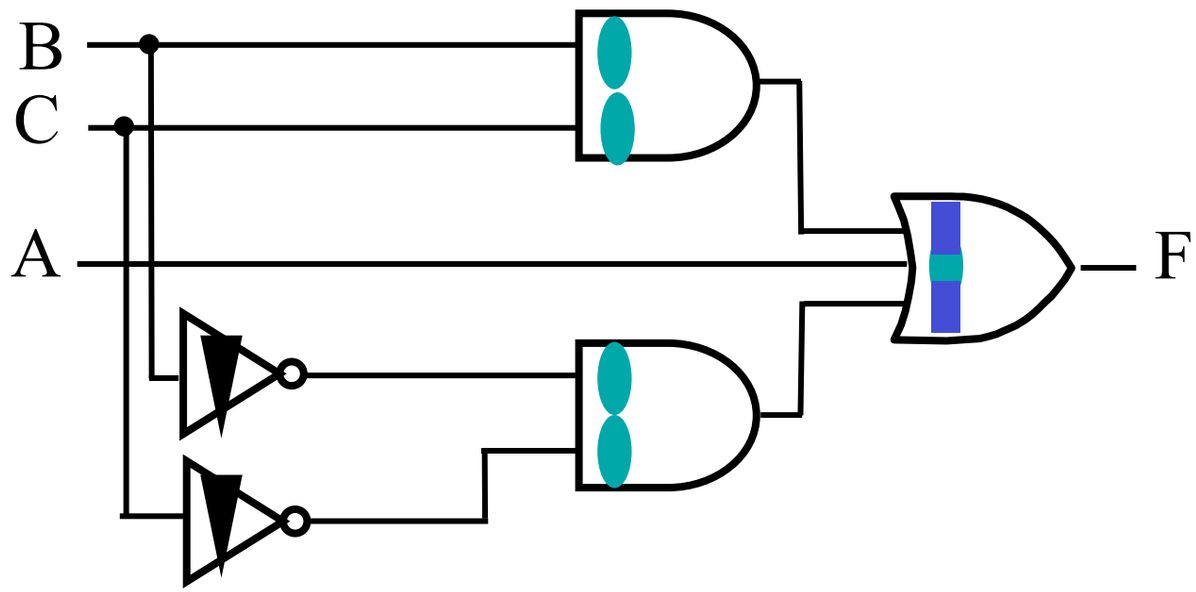
Cost Criteria (continued)

- **Example 1:**
- **$F = A + B\,C + \overline{B}\,\overline{C}$**

GN = G + 2 = 9

L = 5

G = L + 2 = 7

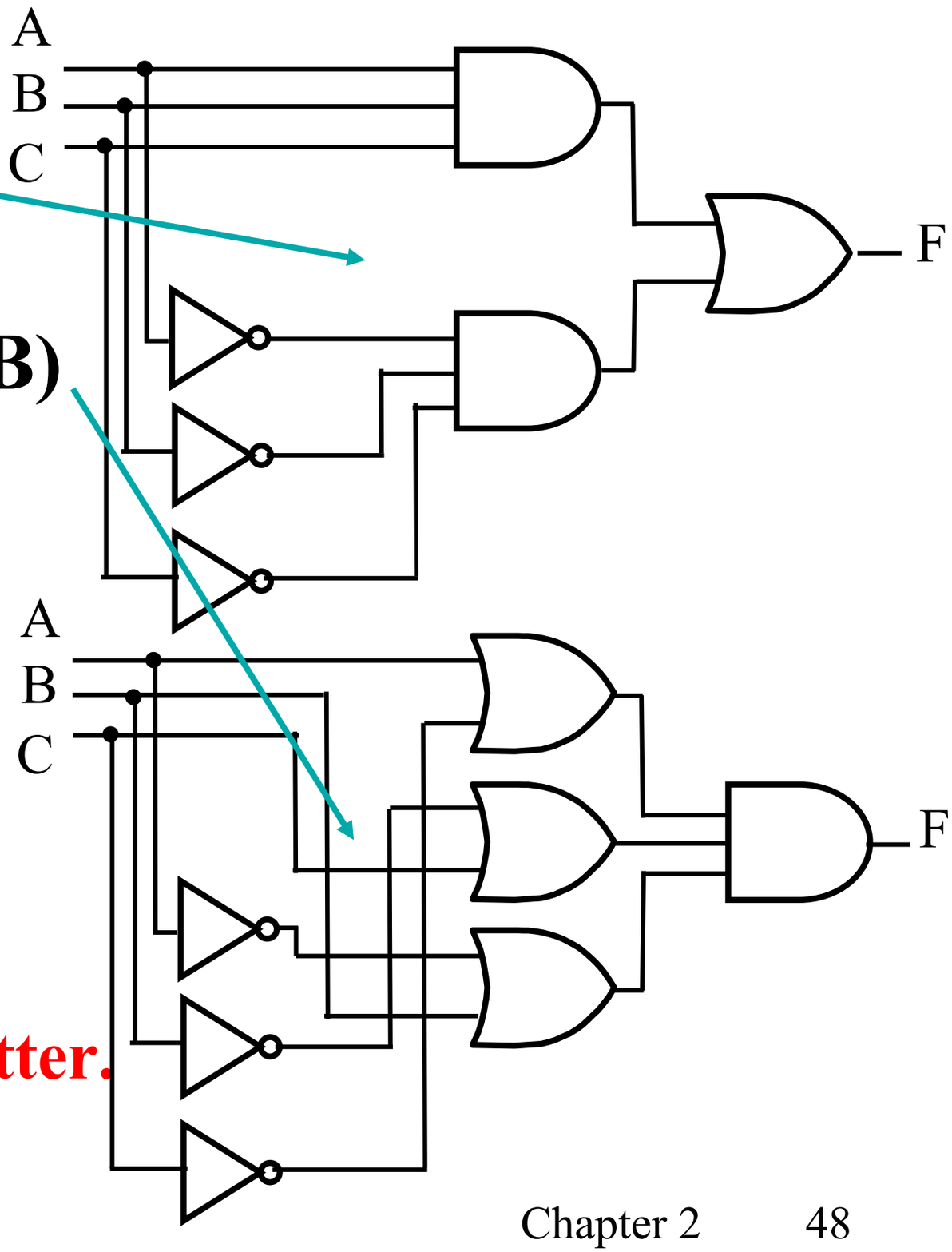


- **L (literal count) counts the AND inputs and the single literal OR input.**
- **G (gate input count) adds the remaining OR gate inputs**
- **GN(gate input count with NOTs) adds the inverter inputs**

Cost Criteria (continued)

- **Example 2:**
- **$F = A\,B\,C + \overline{A}\,\overline{B}\,\overline{C}$**
- **L = 6 G = 8 GN = 11**
- **$F = (A + \overline{C})(\overline{B} + C)(\overline{A} + B)$**
- **L = 6 G = 9 GN = 12**
- **Same function and same literal cost**
- **But first circuit has better gate input count and better gate input count with NOTs**
- **Select it!**

=> Gate input cost performs better.



Boolean Function Optimization

- **Minimizing the gate input (or literal) cost of a (a set of) Boolean equation(s) reduces circuit cost.**
- **We choose gate input cost.**
- **Boolean Algebra and graphical techniques are tools to minimize cost criteria values.**
- **Some important questions:**
  - **When do we stop trying to reduce the cost?**
  - **Do we know when we have a minimum cost?**

  => *Optimization problem*
- **Treat *optimum* or *near-optimum* cost functions for two-level (SOP and POS) circuits first.**
- **Introduce a graphical technique using Karnaugh maps (K-maps, for short)**

Karnaugh Maps (K-map)

- A K-map is a collection of squares
  - Each square represents a minterm
  - The collection of squares is a graphical representation of a Boolean function
  - Adjacent squares differ in the value of one variable
  - Alternative algebraic expressions for the same function are derived by recognizing patterns of squares
- The K-map can be viewed as
  - A reorganized version of the truth table

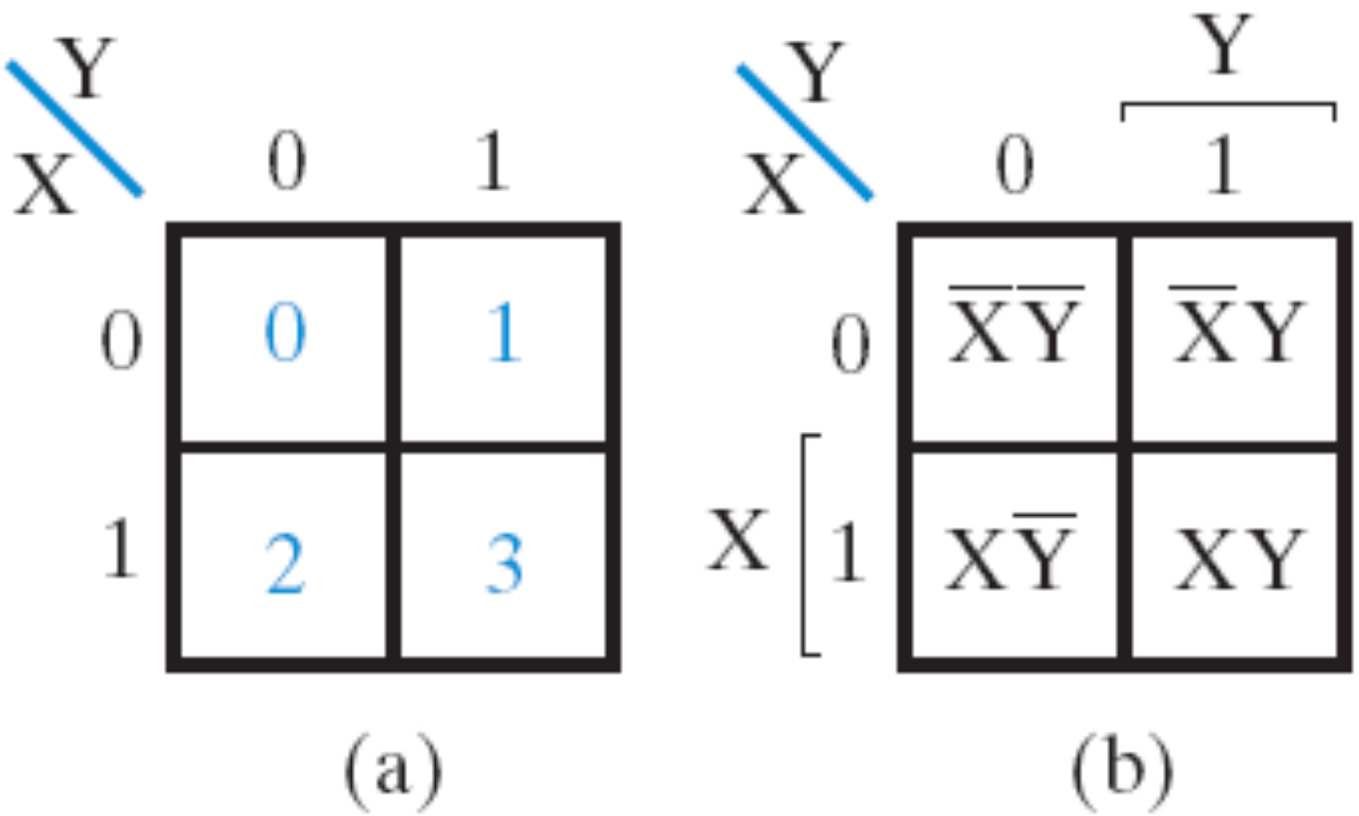
Some Uses of K-Maps

- **Provide a means for:**
  - **Finding optimum or near optimum**
    - **SOP and POS standard forms, and**
    - **two-level AND/OR and OR/AND circuit implementations**

    for functions with small numbers of variables
  - **Visualizing concepts related to manipulating Boolean expressions, and**
  - **Demonstrating concepts used by computer-aided design programs to simplify large circuits**

Two Variable Maps

- **A 2-variable Karnaugh Map:**
  - **Note that all the adjacent minterms are differ in value of only one variable**

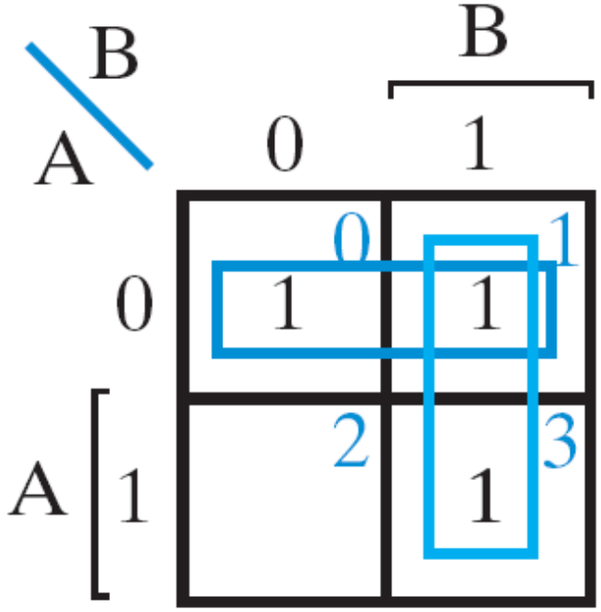


From Truth Tables to K-Map

- **K-Maps can be used to simplify Boolean functions by systematic methods. Terms are selected to cover the 1's cells in the map.**
- **Example – Two variable function:**

TABLE 2-9
Two-Variable Function F(A, B)

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



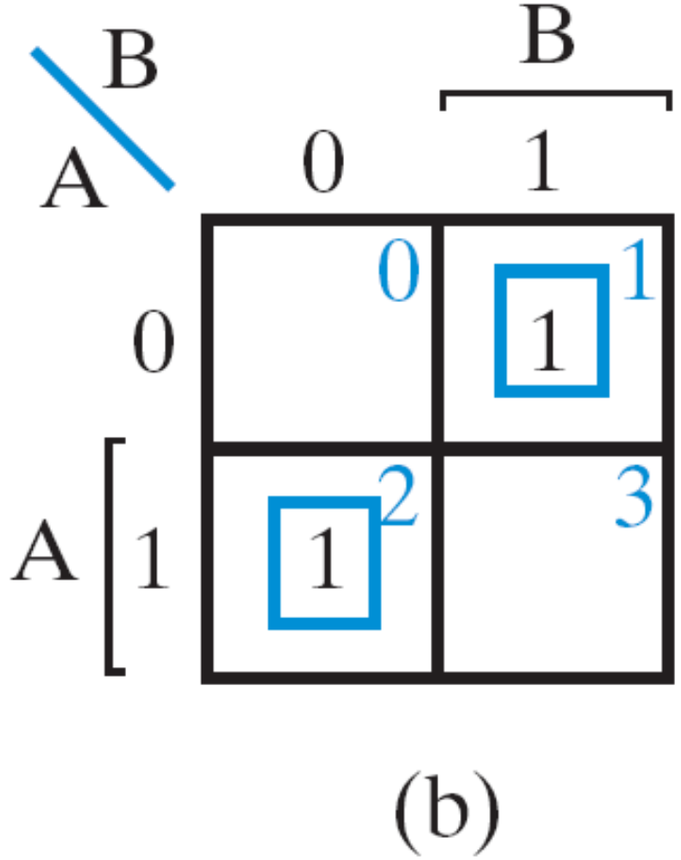
Result of simplification :

$$F = \overline{A} + B$$

K-Map Function Representation

Example 2-4:

$$G(A,B) = \overline{A}B + A\overline{B}$$



Result of simplification=?

K-Map Function Representation

- **Example: G(x,y) = x + y**

G = x+y	y = 0	y = 1
x = 0	0	1
x = 1	1	1

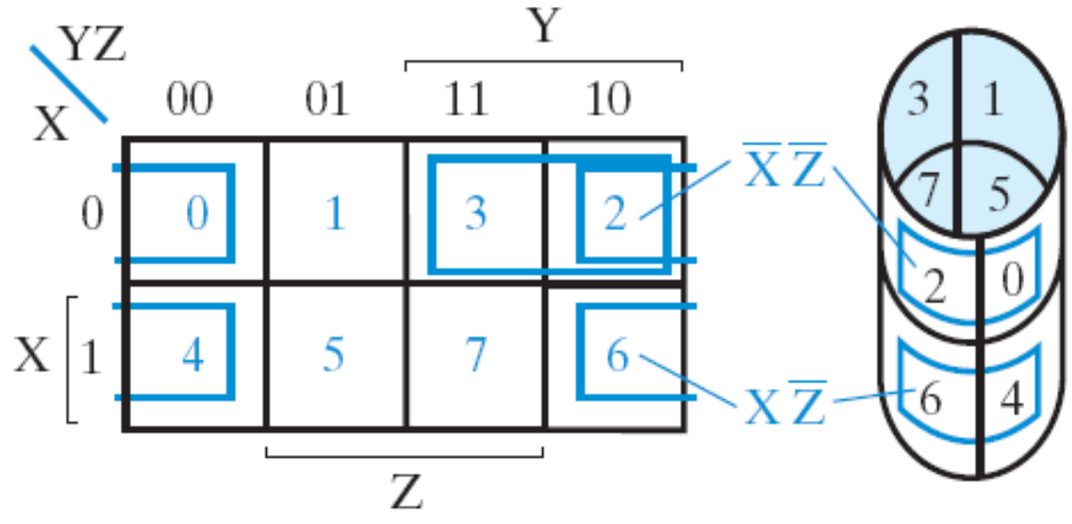
- **For G(x,y), two pairs of adjacent cells containing 1's can be combined using the Minimization Theorem:**

$$G(x,y) = (x\bar{y} + xy) + (xy + \bar{x}y) = x + y$$

Duplicate x y

Three Variable Maps

- **A three-variable K-map:**



- **Where each minterm corresponds to the product terms:**

	yz=00	yz=01	yz=11	yz=10
x=0	$\overline{x}\,\overline{y}\,\overline{z}$	$\overline{x}\,\overline{y}\,z$	$\overline{x}\,y\,z$	$\overline{x}\,y\,\overline{z}$
x=1	$x\,\overline{y}\,\overline{z}$	$x\,\overline{y}\,z$	$x\,y\,z$	$x\,y\,\overline{z}$

- **Note that if the binary value for an index differs in one bit position, the minterms are adjacent on the K-Map**

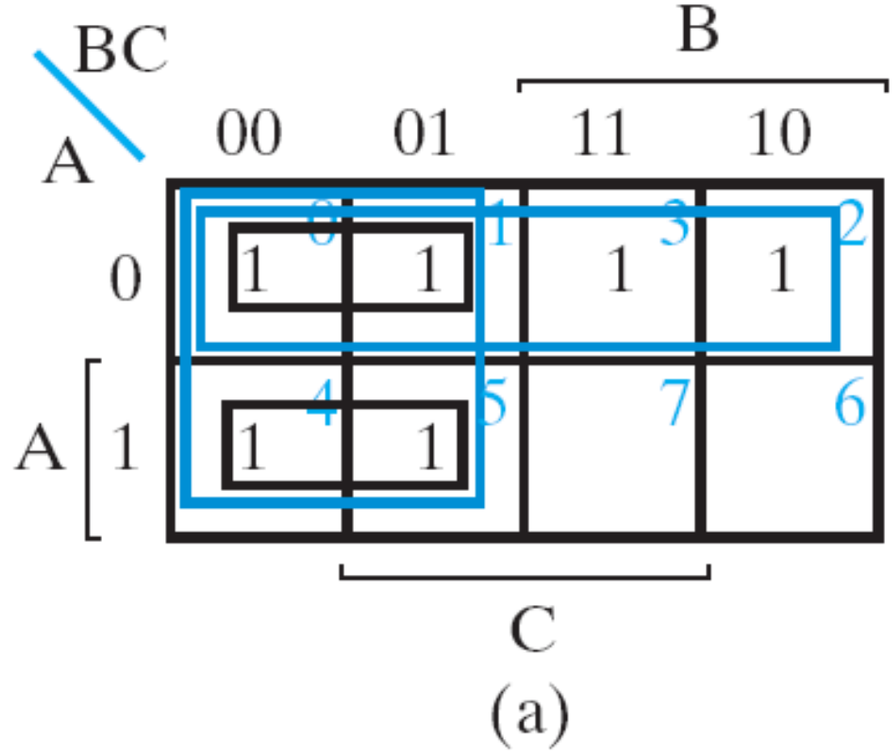
Three-Variable Maps

- **Reduced literal product terms for SOP standard forms correspond to rectangles on K-maps containing cell counts that are powers of 2.**
- **Rectangles of 2 cells represent 2 adjacent minterms; of 4 cells represent 4 minterms that form a "pairwise adjacent" ring.**
  - **2 cells rectangle will cancel 1 variable**
  - **4 (=2^2) cells rectangle will cancel 2 variables**
  - **8 (=2^3) cells rectangle will cancel 3 variables**
    - **What is the result for this case in three variables K-map?**

Three Variable Maps

- **Example 2-5: Simplify**

$$F(A,B,C) = \Sigma_m(0,1,2,3,4,5)$$



(a)

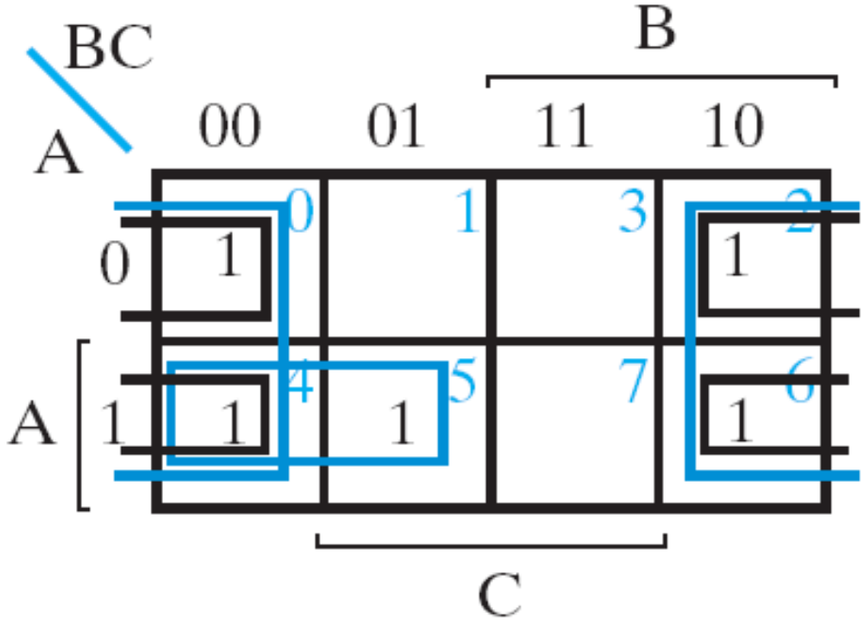
Result:

$$F(A,B,C) = \overline{A} + \overline{B}$$

Three Variable Maps

- **Example 2-6: Simplify**

$$G(A,B,C) = \Sigma_m(0, 2, 4, 5, 6)$$



(b)

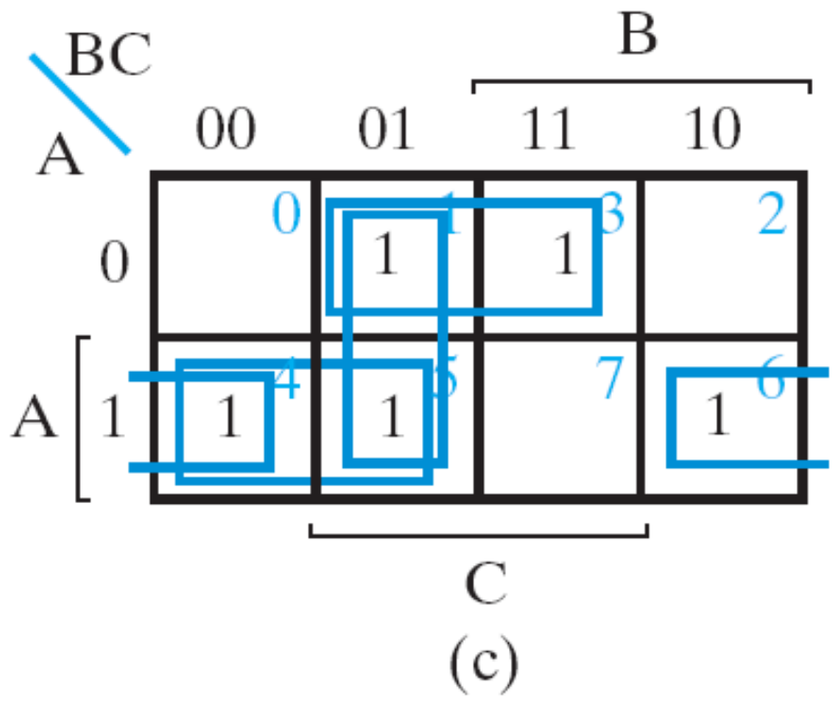
Result:

$$G(A,B,C) = A\overline{B} + \overline{C}$$

Three Variable Maps

- **Example 2-7: Simplify**

$$H(A,B,C) = \Sigma_m(1,3,4,5,6)$$



(c)

Result:

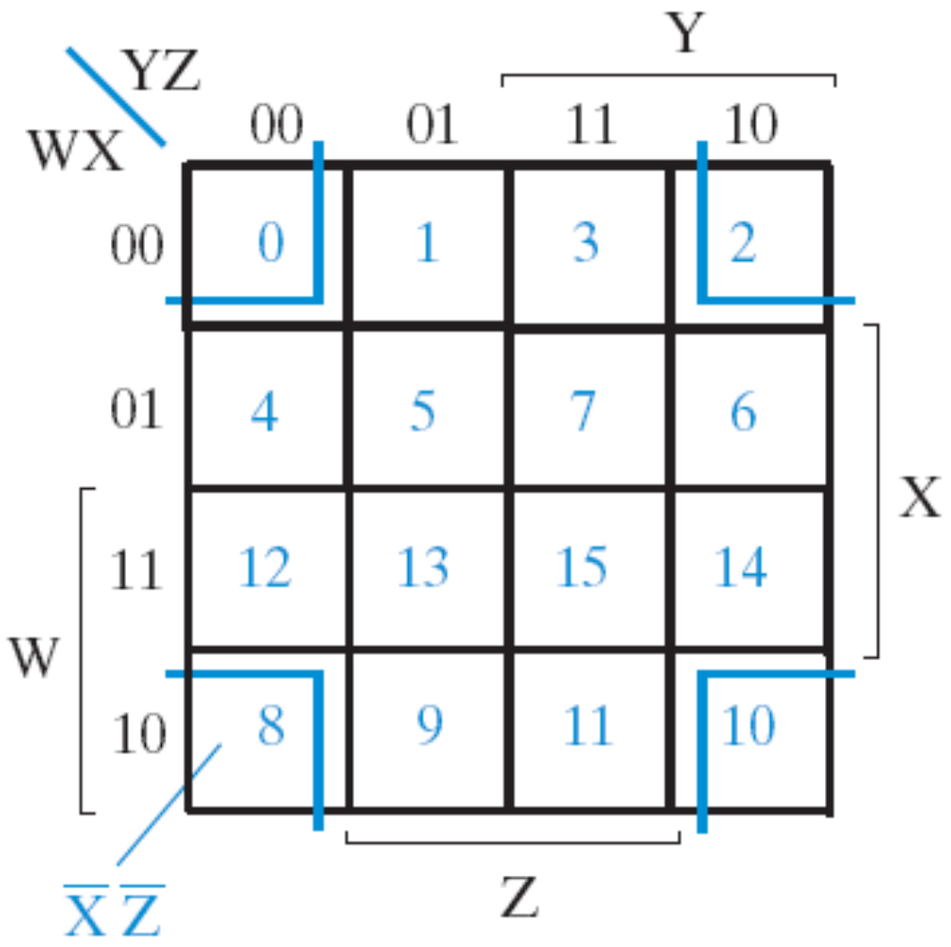
$$H(A,B,C) = \overline{A}C + A\overline{B} + A\overline{C}$$

Three-Variable Map Simplification

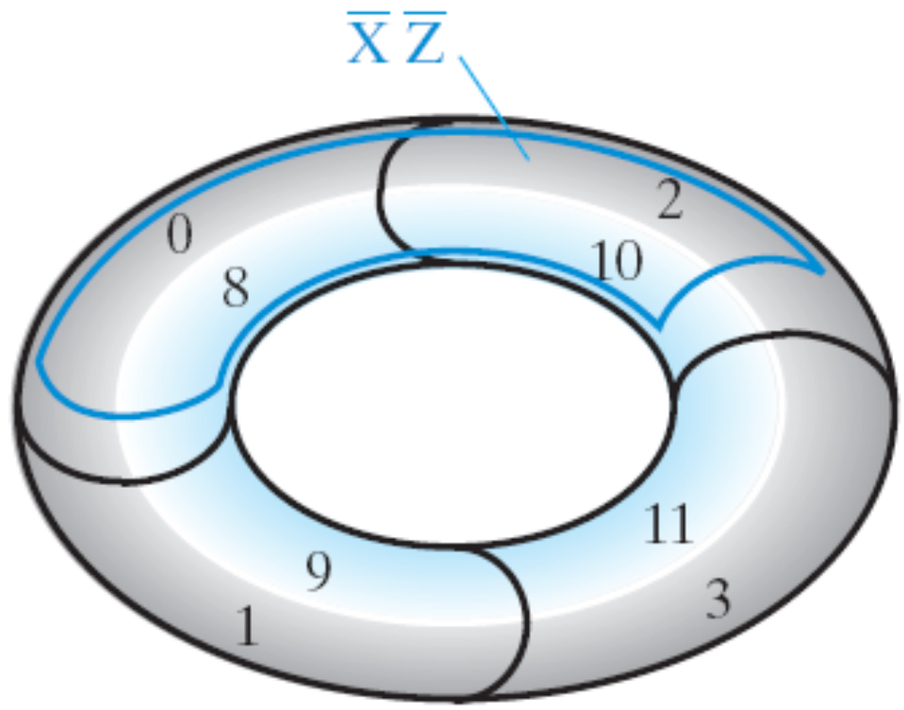
- Use a K-map to find an optimum SOP equation for $F(X, Y, Z) = \Sigma_m(0,1,2,4,6,7)$

Four Variable Maps

- **Map and location of minterms:**



(e)



(f)

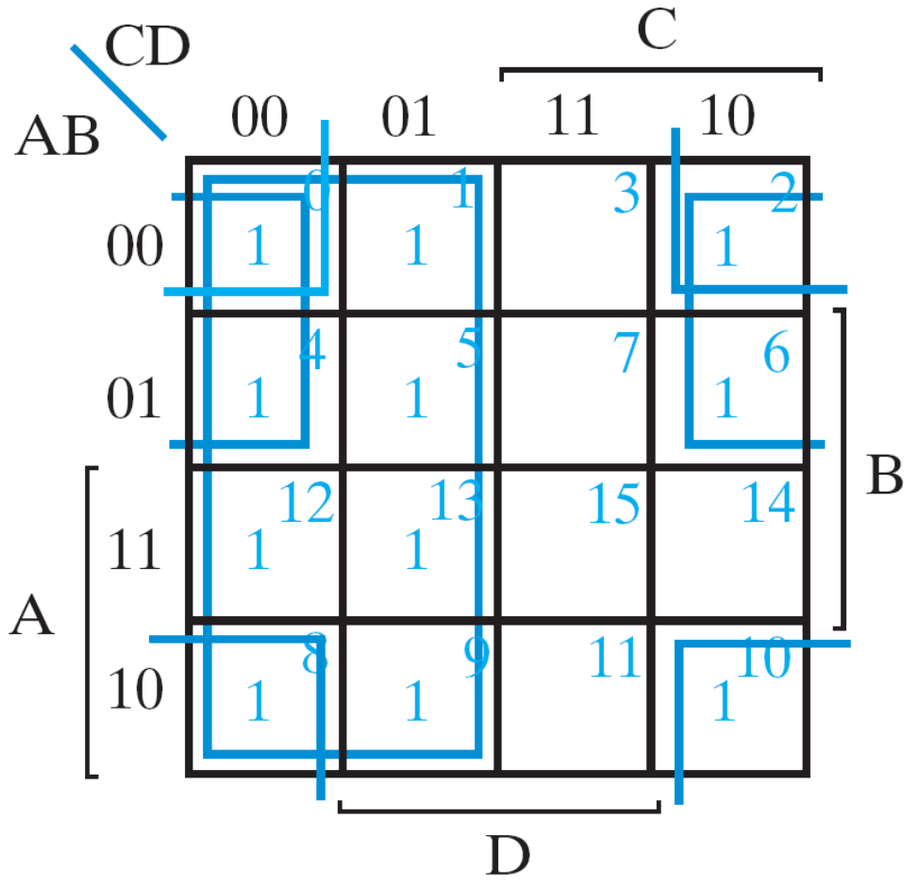
Four Variable Terms

- **Four variable maps can have rectangles corresponding to:**
  - **A single 1 = 4 variables, (i.e. Minterm)**
  - **Two 1s = 3 variables,**
  - **Four 1s = 2 variables**
  - **Eight 1s = 1 variable,**
  - **Sixteen 1s = zero variables (i.e. Constant "1")**

Four-Variable Maps

- **Example 2-8: Simplify**

$$F(A,B,C,D) = \Sigma_m(0,1,2,4,5,6,8,9,10,12,13)$$



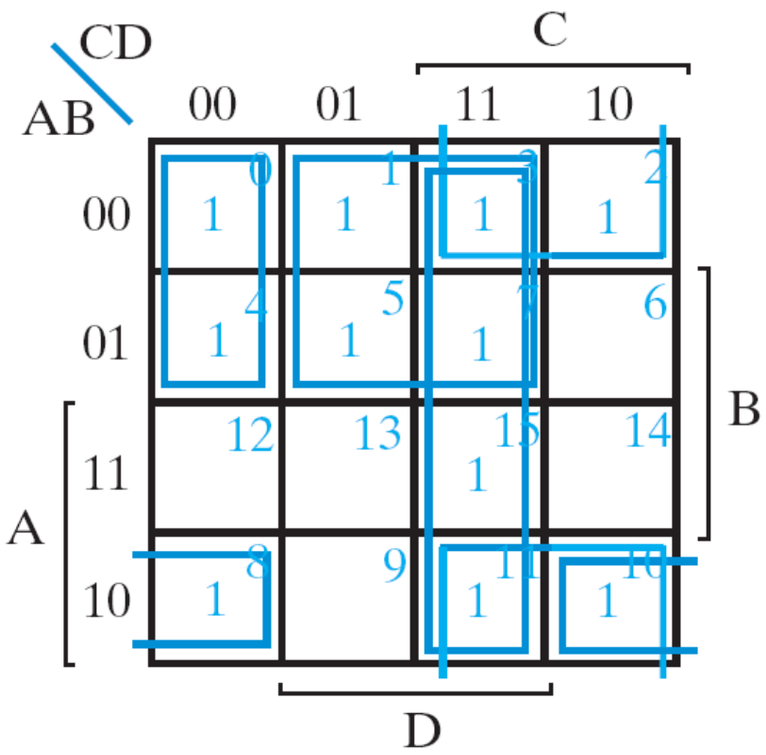
Result:

$$F = \overline{C} + \overline{A}\,\overline{D} + \overline{B}\,\overline{D}$$

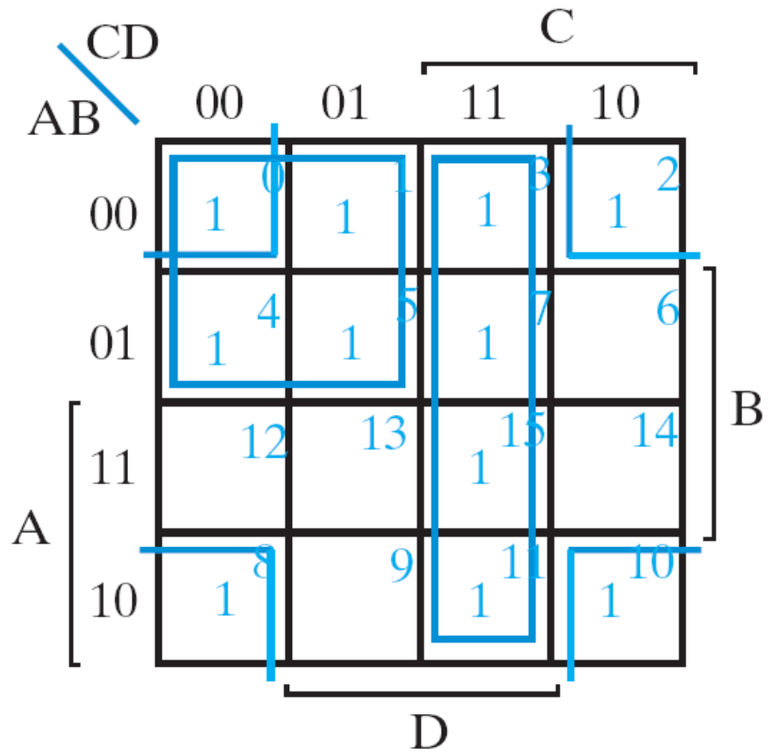
Four-Variable Maps

- **Example 2-9: Simplify**

$$G(A,B,C,D) = \overline{A}\,\overline{C}\,\overline{D} + \overline{A}D + \overline{B}C + CD + A\overline{B}\,\overline{D}$$



(a) K-map for original function G



(b) K-map for simplified function G

Result:

$$G = \overline{B}\,\overline{D} + \overline{A}\,\overline{C} + CD$$

Four-Variable Map

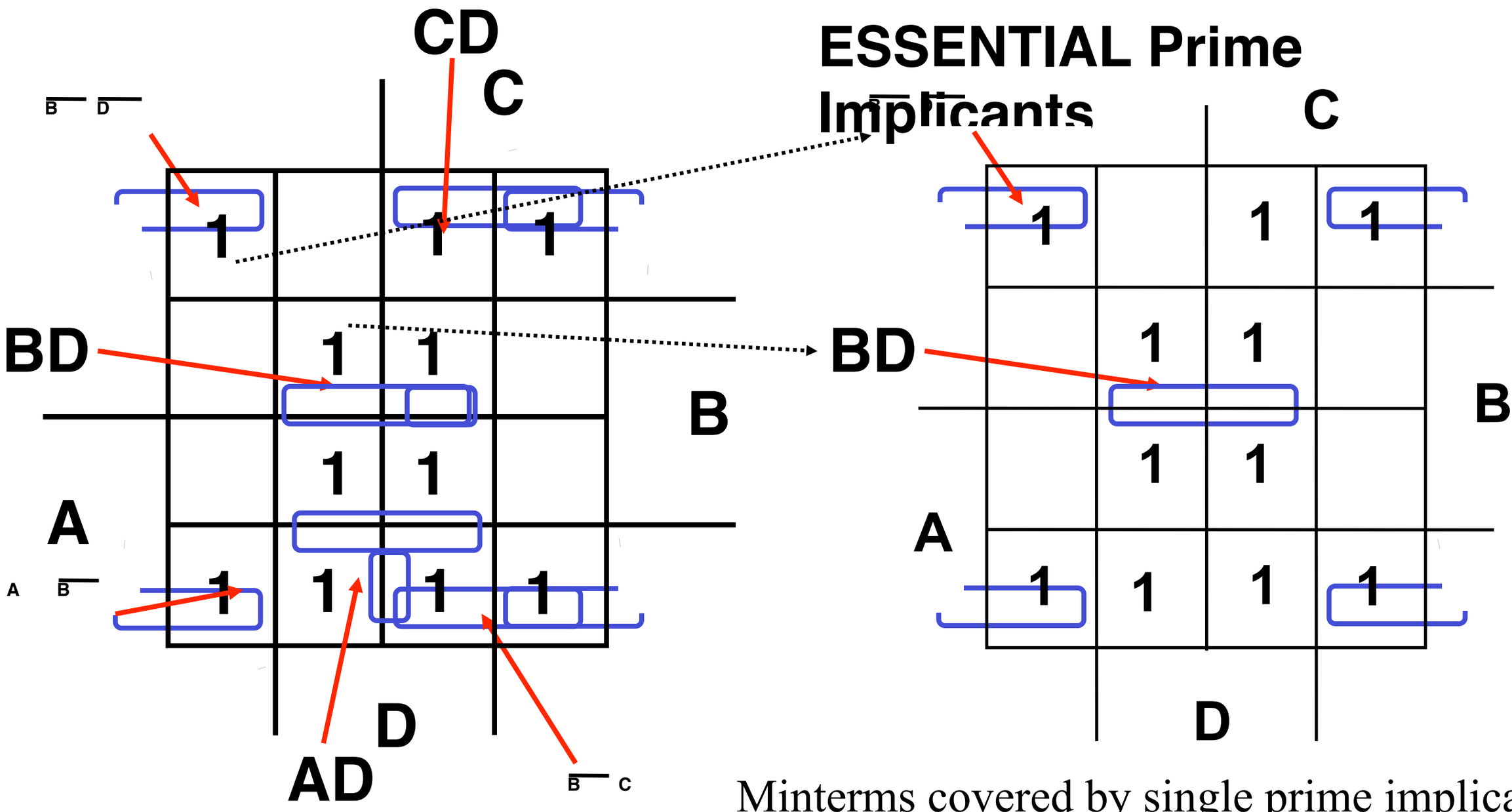
- $F(W, X, Y, Z) = \Sigma_m(3,4,5,7,9,13,14,15)$

2-5 Map Manipulation

- ***Implicant:*** **the rectangles on a map made up of squares containing 1s correspond to implicants.**
- **A *Prime Implicant* is a product term obtained by combining the maximum possible number of adjacent squares in the map into a rectangle with the number of squares a power of 2.**
- **A prime implicant is called an *Essential Prime Implicant* if it is the <u>only</u> prime implicant that covers (includes) one or more minterms.**
- **Prime Implicants and Essential Prime Implicants can be determined by inspection of a K-Map.**
- **A set of prime implicants *"covers all minterms"* if, for each minterm of the function, at least one prime implicant in the set of prime implicants includes the minterm.**

Example of Prime Implicants

- **Find ALL Prime Implicants**



Minterms covered by single prime implicant

Another Example

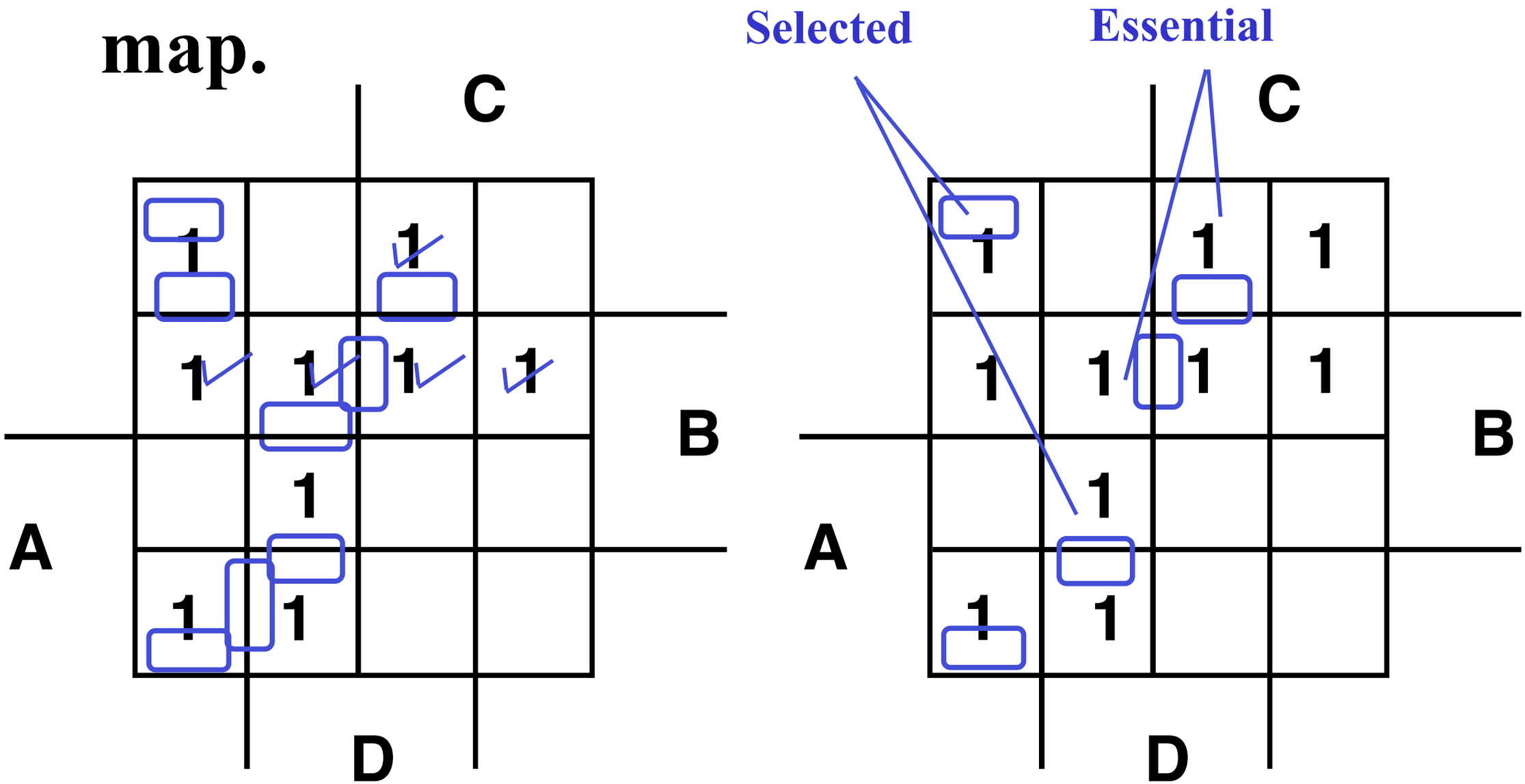
- **Find all prime implicants for:**
  $G(A, B, C, D) = \Sigma_m(0,2,3,4,7,12,13,14,15)$
  - **Hint: There are seven prime implicants!**

Optimization Algorithm

- **Find all prime implicants.**
- **Include all essential prime implicants in the solution**
- **Select a minimum cost set of non-essential prime implicants to cover all minterms not yet covered.**
- **Minimize the overlap among prime implicants as much as possible. In particular, in the final solution, make sure that each prime implicant selected includes at least one minterm not included in any other prime implicant selected.**

Selection Rule Example

- **Simplify F(A, B, C, D) given on the K-map.**

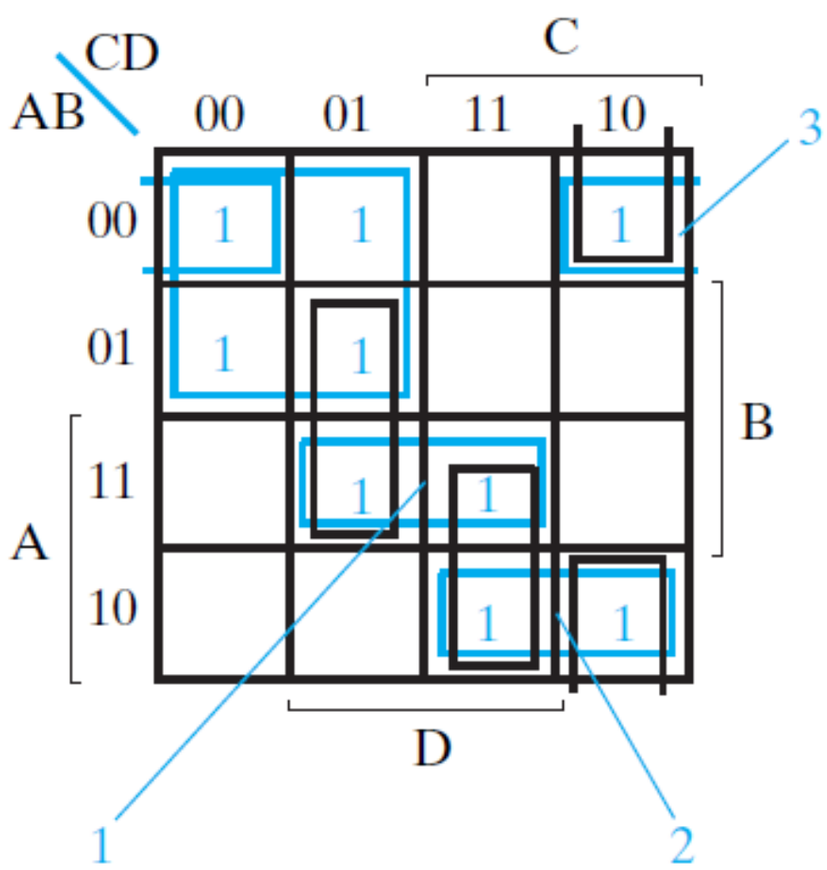


Example 2-12

- **Simplifying a function using the selection rule**

$$F(A,B,C,D) = \Sigma_m(0,1,2,4,5,10,11,13,15)$$

Result:



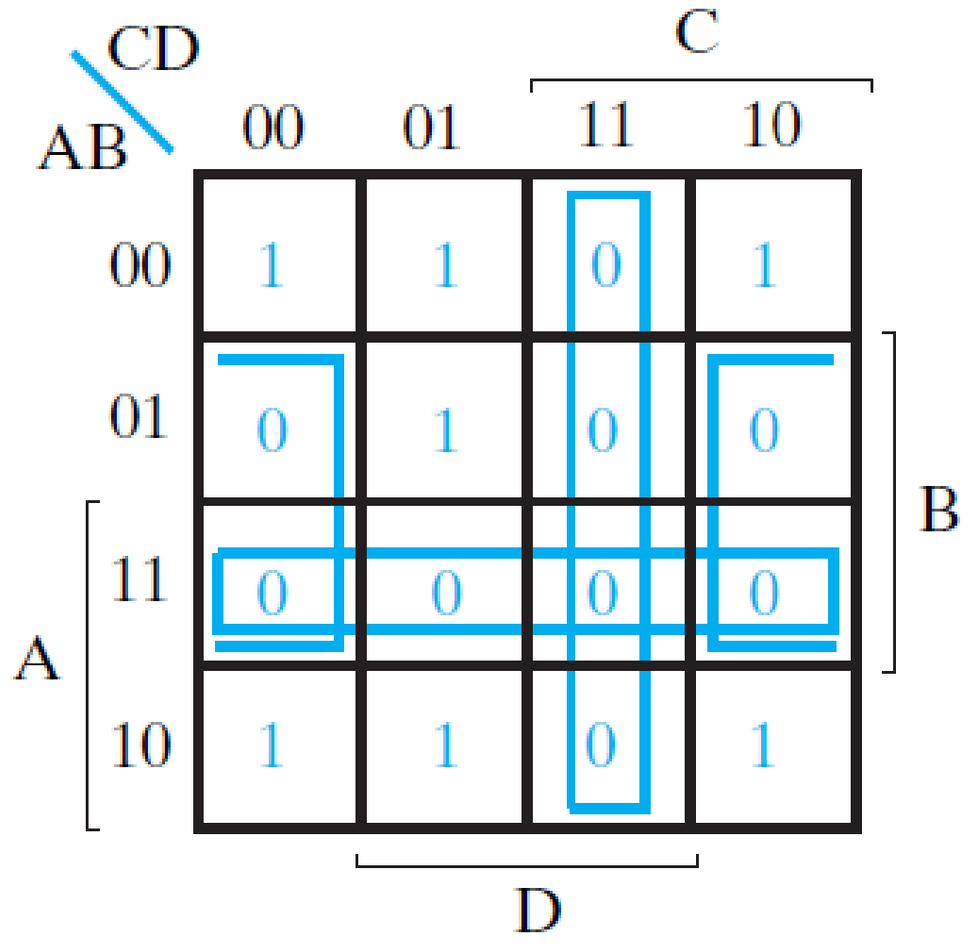
$$F(A,B,C,D) = \overline{A}\,\overline{C} + ABD + A\overline{B}C + \overline{A}\,\overline{B}\overline{D}$$

Product-of-Sums Optimization

- The minterm not included in the function belong to the complement of the function.
- Example 2-13: $F(A,B,C,D) = \Sigma_m(0,1,2,5,8,9,10)$

$$\overline{F} = AB + CD + B\overline{D}$$

$$F = (\overline{A} + \overline{B})(\overline{C} + \overline{D})(\overline{B} + D)$$



Don't Cares in K-Maps

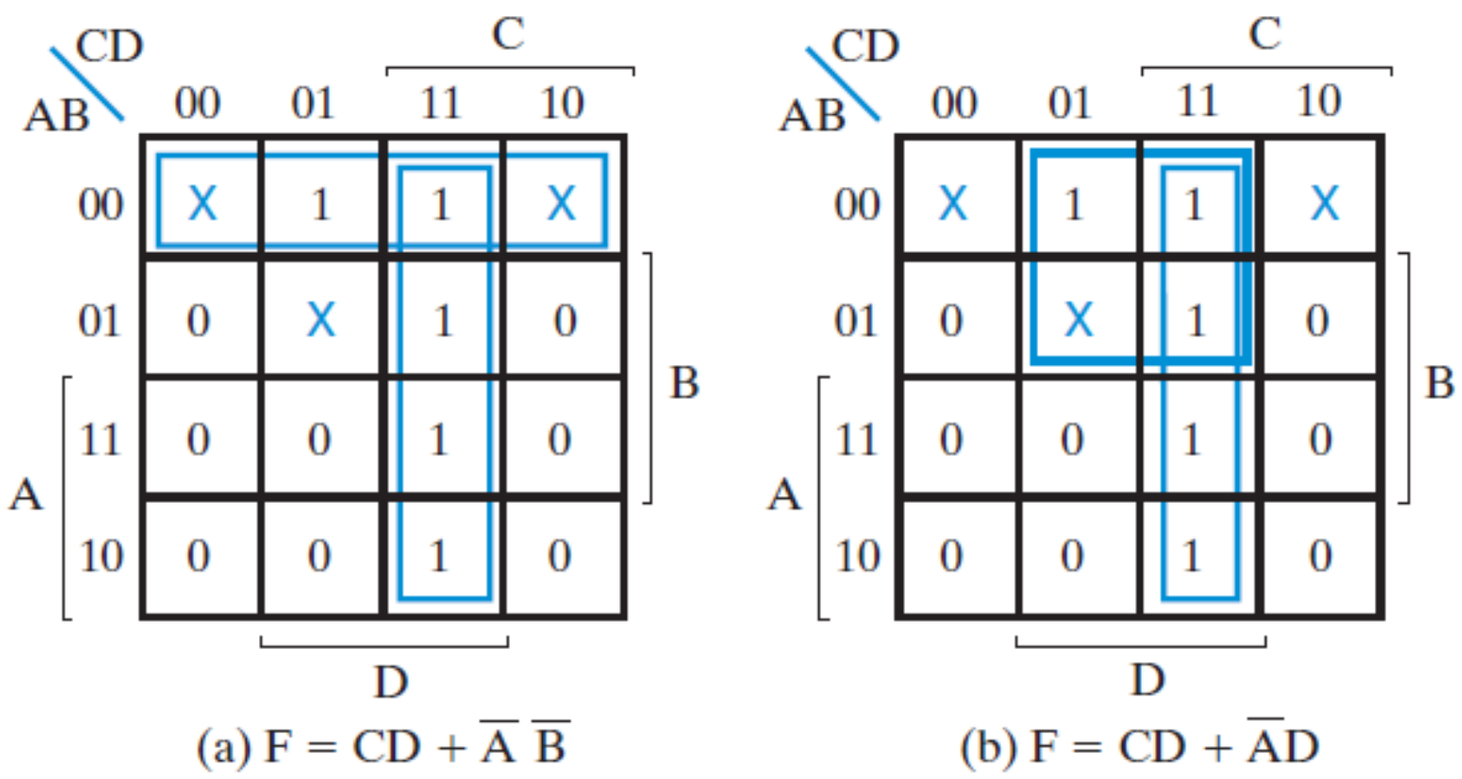
- **Sometimes a function table or map contains entries for which it is known:**
  - **the input values for the minterm will never occur, or**
  - **The output value for the minterm is not used**
- **In these cases, the output value need not be defined**
- **Instead, the output value is defined as a “don't care”**
- **By placing “don't cares” (an “x” entry) in the function table or map, the cost of the logic circuit may be lowered.**
- **Example 1: A logic function having the binary codes for the BCD digits as its inputs. Only the codes for 0 through 9 are used. The six codes, 1010 through 1111 never occur, so the output values for these codes are “x” to represent “don’t cares.”**

Don't Cares in K-Maps

- **Example 2-14: Simplify**

$$F(A,B,C,D) = \sum m(1,3,7,11,15)$$

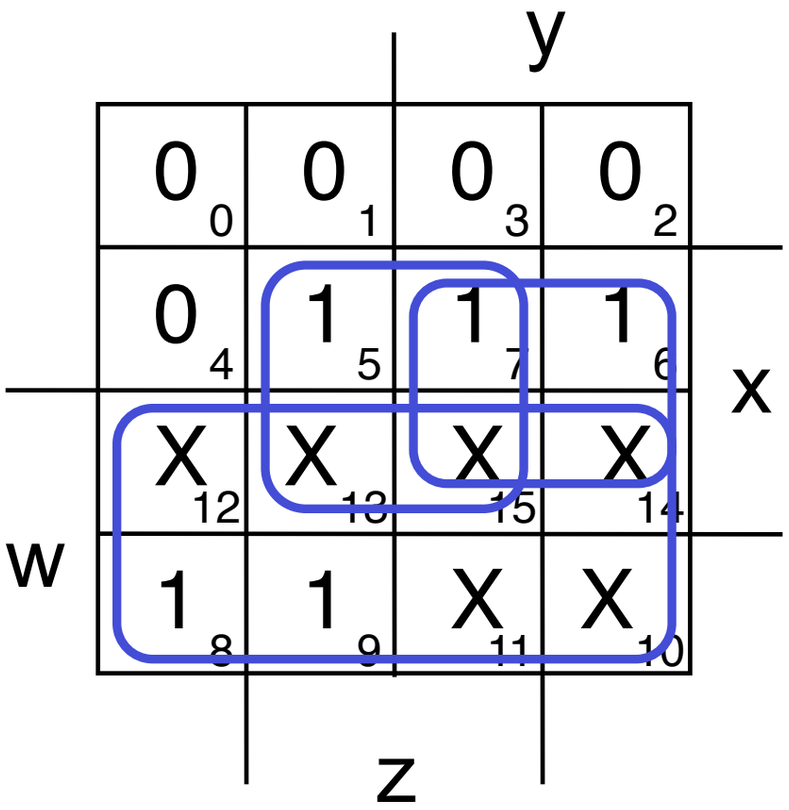
$$d(A,B,C,D) = \sum m(0,2,5)$$



(a) F = CD + $\overline{A}$ $\overline{B}$ (b) F = CD + $\overline{A}$D

Example: BCD “5 or More”

- The map below gives a function F1(w,x,y,z) which is defined as "5 or more" over BCD inputs. With the don't cares used for the 6 non-BCD combinations:



$F_2(w, x, y, z) = \overline{w}\,x\,z + \overline{w}\,x\,y + w\,\overline{x}\,\overline{y}$

F1 (w,x,y,z) = w + x z + x y G = 7

- This is much lower in cost than F2 where the “don't cares” were treated as "0s." **G = 12**
- For this particular function, cost G for the POS solution for $F_1(w,x,y,z)$ is not changed by using the don't cares.