
Digital Logic Basics

Chapter 2

S. Dandamudi

Outline

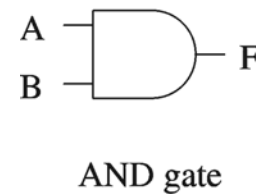
- Basic concepts
 - * Simple gates
 - * Completeness
 - Logic functions
 - * Expressing logic functions
 - * Equivalence
 - Boolean algebra
 - * Boolean identities
 - * Logical equivalence
 - Logic Circuit Design Process
- Deriving logical expressions
 - * Sum-of-products form
 - * Product-of-sums form
 - Simplifying logical expressions
 - * Algebraic manipulation
 - * Karnaugh map method
 - * Quine-McCluskey method
 - Generalized gates
 - Multiple outputs
 - Implementation using other gates (NAND and XOR)

Introduction

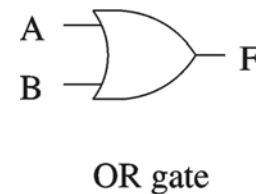
- Hardware consists of a few simple building blocks
 - * These are called *logic gates*
 - » AND, OR, NOT, ...
 - » NAND, NOR, XOR, ...
- Logic gates are built using transistors
 - » NOT gate can be implemented by a single transistor
 - » AND gate requires 3 transistors
- Transistors are the fundamental devices
 - » Pentium consists of 3 million transistors
 - » Compaq Alpha consists of 9 million transistors
 - » Now we can build chips with more than 100 million transistors

Basic Concepts

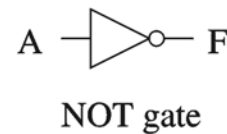
- Simple gates
 - * AND
 - * OR
 - * NOT
- Functionality can be expressed by a truth table
 - * A truth table lists output for each possible input combination
- Other methods
 - * Logic expressions
 - * Logic diagrams



| A | B | F |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



| A | B | F |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |



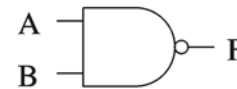
| A | F |
|---|---|
| 0 | 1 |
| 1 | 0 |

Logic symbol

Truth table

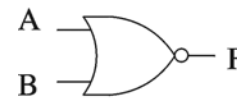
Basic Concepts (cont'd)

- Additional useful gates
 - * NAND
 - * NOR
 - * XOR
- $\text{NAND} = \text{AND} + \text{NOT}$
- $\text{NOR} = \text{OR} + \text{NOT}$
- XOR implements exclusive-OR function
- NAND and NOR gates require only 2 transistors
 - * AND and OR need 3 transistors!



NAND gate

| A | B | F |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



NOR gate

| A | B | F |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |



XOR gate

| A | B | F |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Logic symbol

Truth table

Basic Concepts (cont'd)

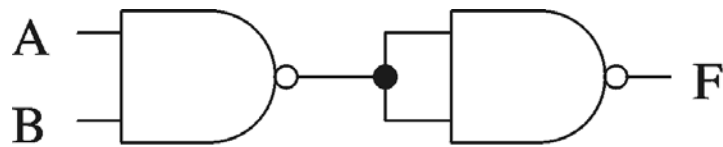
- Number of functions
 - * With N logical variables, we can define 2^{2^N} functions
 - * Some of them are useful
 - » AND, NAND, NOR, XOR, ...
 - * Some are not useful:
 - » Output is always 1
 - » Output is always 0
 - * “Number of functions” definition is useful in proving completeness property

Basic Concepts (cont'd)

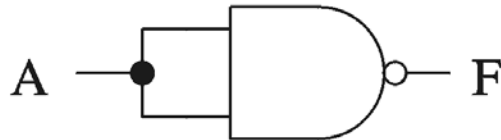
- Complete sets
 - * A set of gates is complete
 - » if we can implement any logical function using only the type of gates in the set
 - You can use as many gates as you want
 - * Some example complete sets
 - » {AND, OR, NOT} ← Not a minimal complete set
 - » {AND, NOT}
 - » {OR, NOT}
 - » {NAND}
 - » {NOR}
 - * Minimal complete set
 - A complete set with no redundant elements.

Basic Concepts (cont'd)

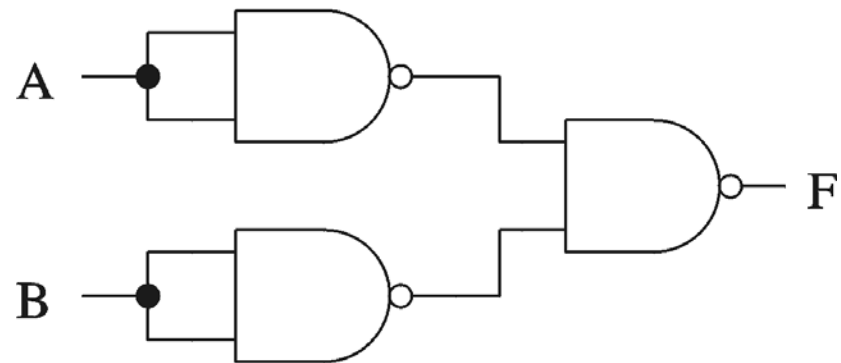
- Proving NAND gate is universal



AND gate



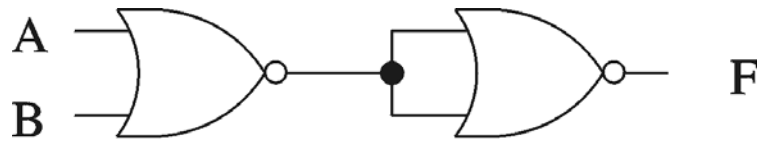
NOT gate



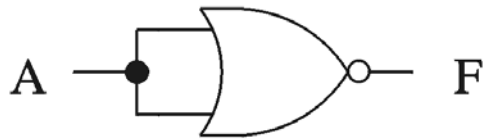
OR gate

Basic Concepts (cont'd)

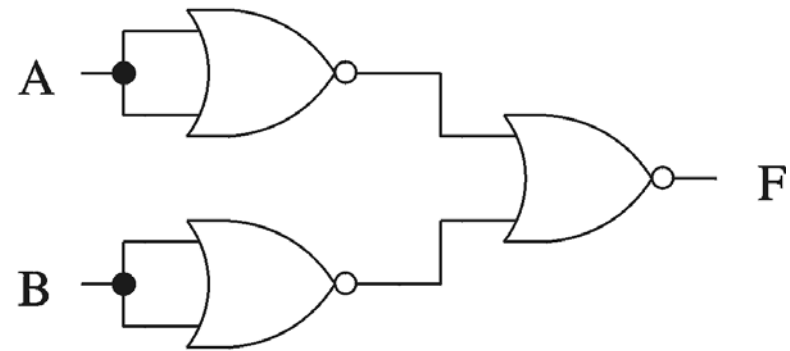
- Proving NOR gate is universal



OR gate



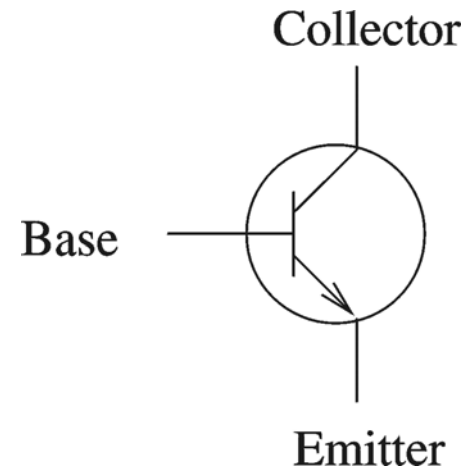
NOT gate



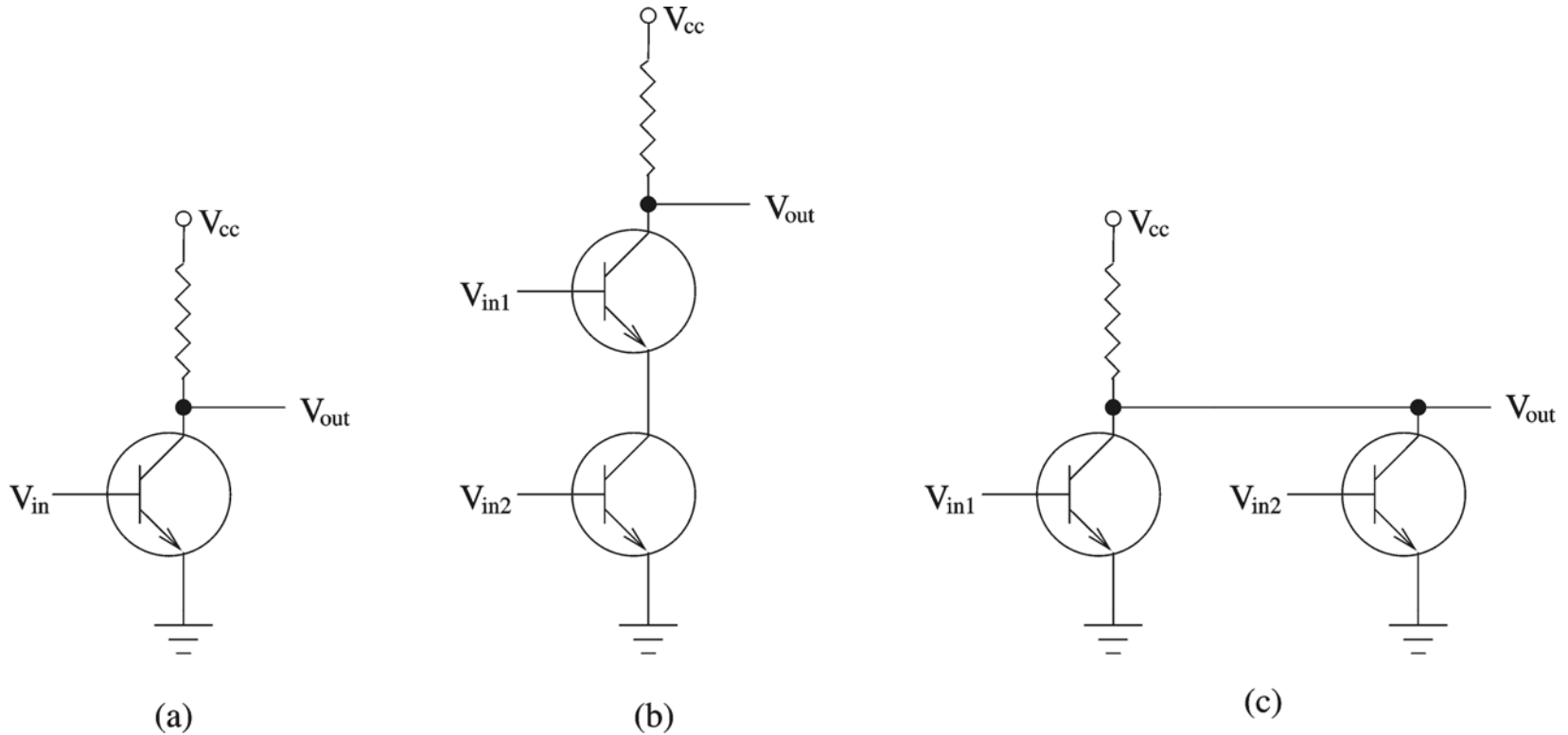
AND gate

Logic Chips

- Basic building block:
 - » Transistor
- Three connection points
 - * Base
 - * Emitter
 - * Collector
- Transistor can operate
 - * Linear mode
 - » Used in amplifiers
 - * Switching mode
 - » Used to implement digital circuits



Logic Chips (cont'd)



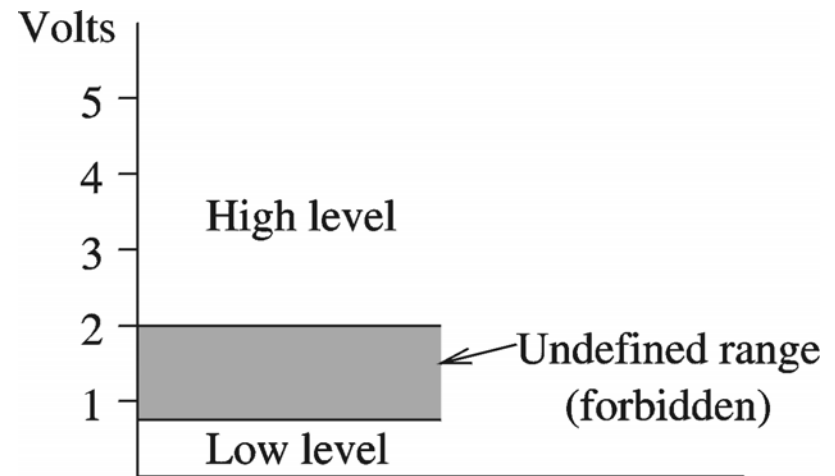
NOT

NAND

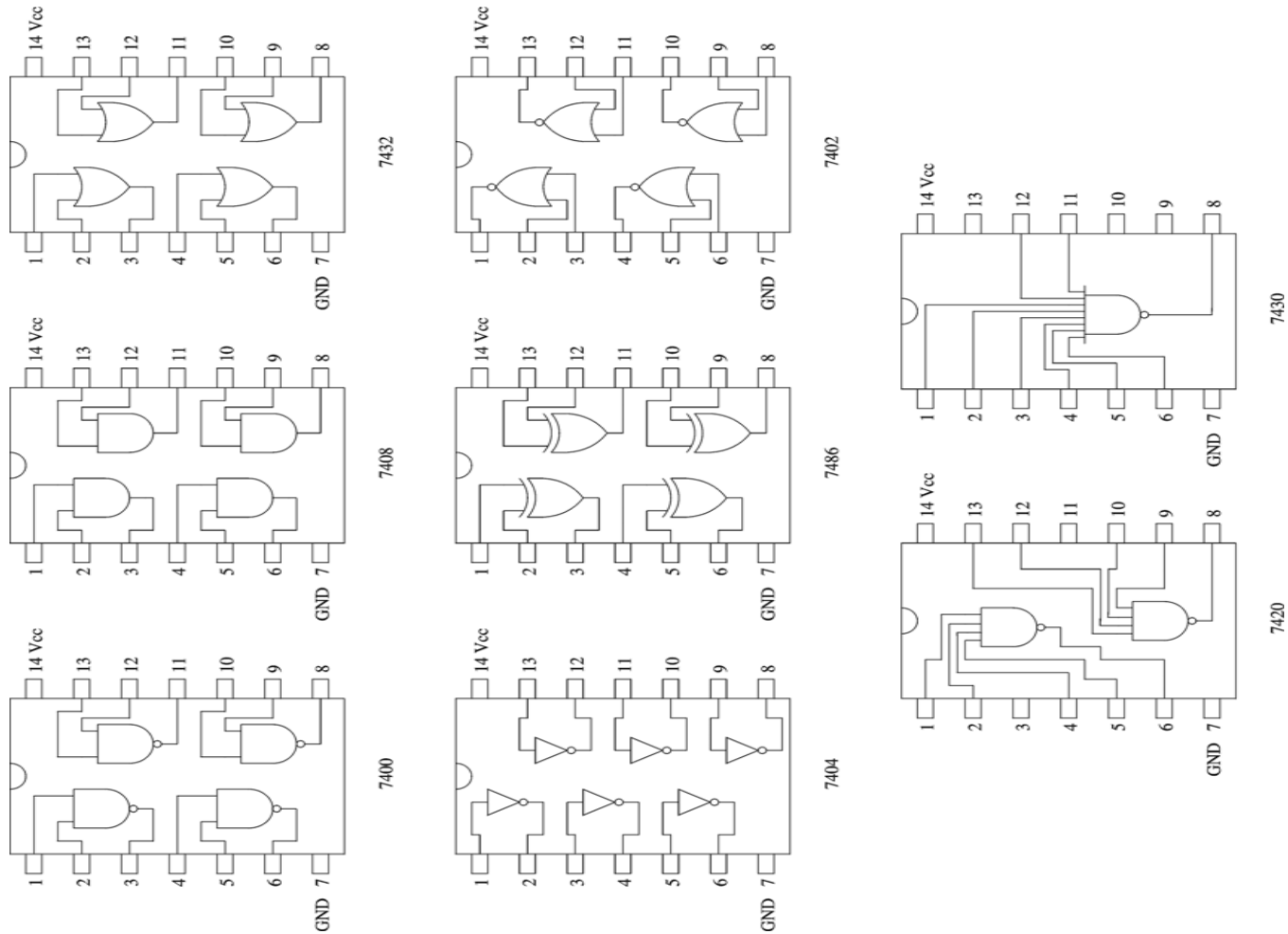
NOR

Logic Chips (cont'd)

- Low voltage level: $< 0.4V$
- High voltage level: $> 2.4V$
- Positive logic:
 - * Low voltage represents 0
 - * High voltage represents 1
- Negative logic:
 - * High voltage represents 0
 - * Low voltage represents 1
- Propagation delay
 - * Delay from input to output
 - * Typical value: 5-10 ns



Logic Chips (cont'd)



Logic Chips (cont'd)

- Integration levels
 - * SSI (small scale integration)
 - » Introduced in late 1960s
 - » 1-10 gates (previous examples)
 - * MSI (medium scale integration)
 - » Introduced in late 1960s
 - » 10-100 gates
 - * LSI (large scale integration)
 - » Introduced in early 1970s
 - » 100-10,000 gates
 - * VLSI (very large scale integration)
 - » Introduced in late 1970s
 - » More than 10,000 gates

Logic Functions

- Logical functions can be expressed in several ways:
 - * Truth table
 - * Logical expressions
 - * Graphical form
- Example:
 - * Majority function
 - » Output is one whenever majority of inputs is 1
 - » We use 3-input majority function

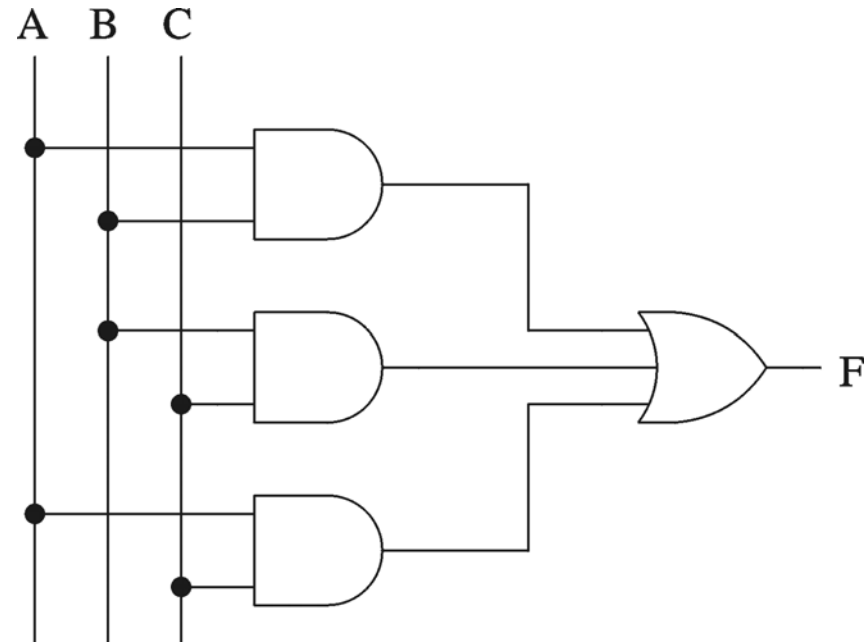
Logic Functions (cont'd)

3-input majority function

- Logical expression form

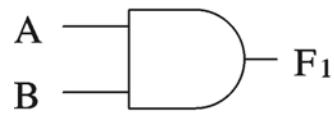
$$F = A B + B C + A C$$

| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

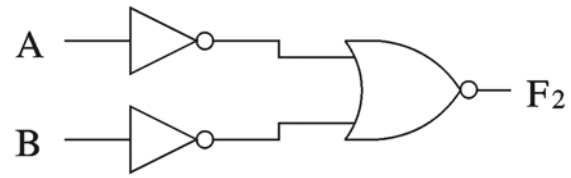


Logical Equivalence

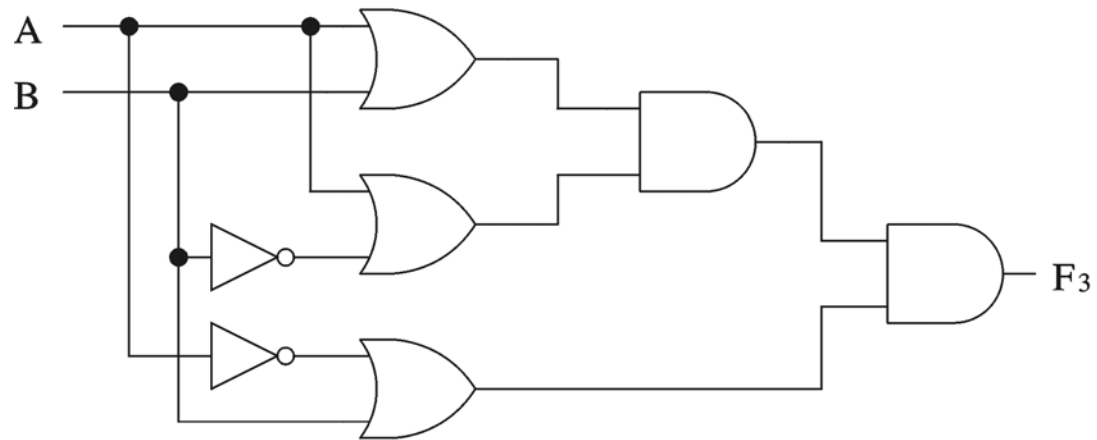
- All three circuits implement $F = A B$ function



(a)



(b)



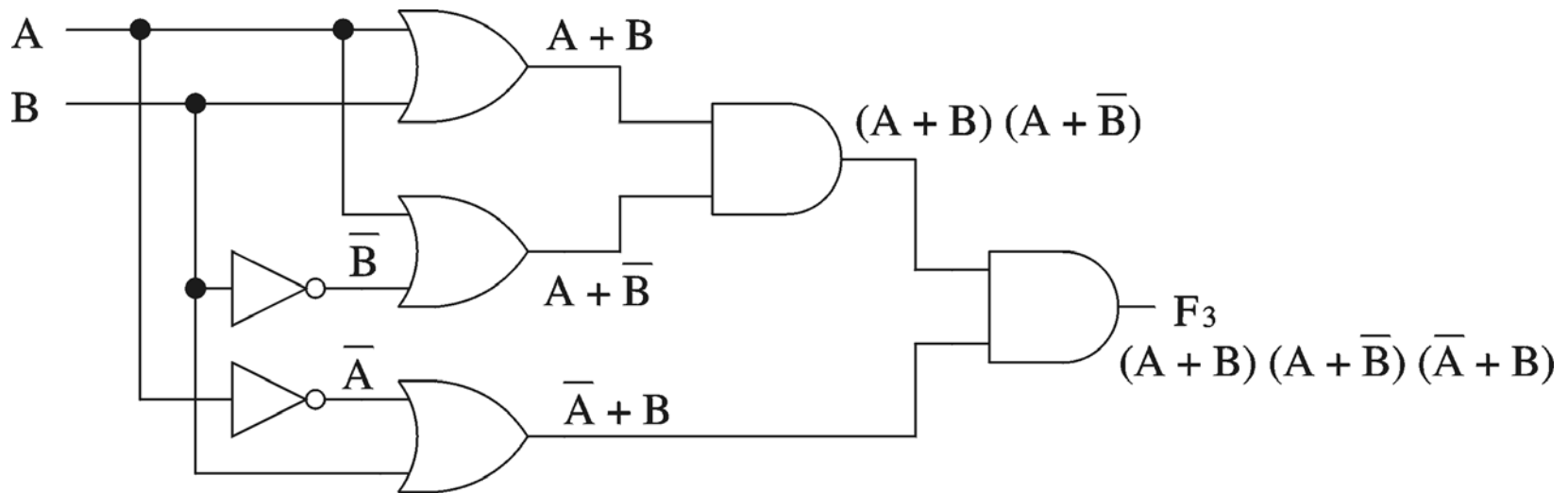
(c)

Logical Equivalence (cont'd)

- Proving logical equivalence of two circuits
 - * Derive the logical expression for the output of each circuit
 - * Show that these two expressions are equivalent
 - » Two ways:
 - You can use the truth table method
 - For every combination of inputs, if both expressions yield the same output, they are equivalent
 - Good for logical expressions with small number of variables
 - You can also use algebraic manipulation
 - Need Boolean identities

Logical Equivalence (cont'd)

- Derivation of logical expression from a circuit
 - * Trace from the input to output
 - » Write down intermediate logical expressions along the path



Logical Equivalence (cont'd)

- Proving logical equivalence: Truth table method

| A | B | F1 = A B | F3 = (A + B) (\bar{A} + B) (A + \bar{B}) |
|----------|----------|-----------------|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |

Boolean Algebra

Boolean identities

| Name | AND version | OR version |
|--------------|-----------------------------|--|
| Identity | $x \cdot 1 = x$ | $x + 0 = x$ |
| Complement | $x \cdot \bar{x} = 0$ | $x + \bar{x} = 1$ |
| Commutative | $x \cdot y = y \cdot x$ | $x + y = y + x$ |
| Distribution | $x \cdot (y + z) = xy + xz$ | $x + (y \cdot z) =$ $(x + y) (x + z)$ |
| Idempotent | $x \cdot x = x$ | $x + x = x$ |
| Null | $x \cdot 0 = 0$ | $x + 1 = 1$ |

Boolean Algebra (cont'd)

- Boolean identities (cont'd)

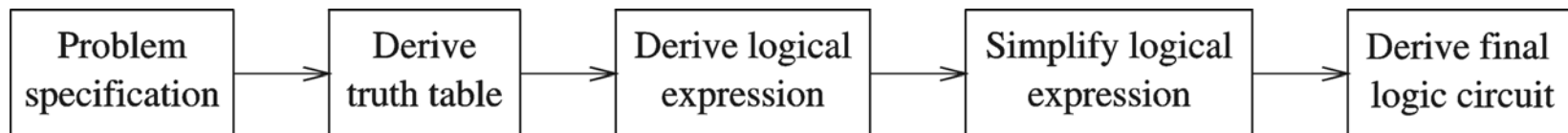
| Name | AND version | OR version |
|-------------|--|--|
| Involution | $\overline{\overline{x}} = x$ | --- |
| Absorption | $x \cdot (x + y) = x$ | $x + (x \cdot y) = x$ |
| Associative | $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ | $x + (y + z) =$ $(x + y) + z$ |
| de Morgan | $\overline{x \cdot y} = \overline{x} + \overline{y}$ | $\overline{x + y} = \overline{x} \cdot \overline{y}$ |

Boolean Algebra (cont'd)

- Proving logical equivalence: Boolean algebra method
 - * To prove that two logical functions $F1$ and $F2$ are equivalent
 - » Start with one function and apply Boolean laws to derive the other function
 - » Needs intuition as to which laws should be applied and when
 - Practice helps
 - » Sometimes it may be convenient to reduce both functions to the same expression
 - * Example: $F1 = A B$ and $F3$ are equivalent

Logic Circuit Design Process

- A simple logic design process involves
 - » Problem specification
 - » Truth table derivation
 - » Derivation of logical expression
 - » Simplification of logical expression
 - » Implementation



Deriving Logical Expressions

- Derivation of logical expressions from truth tables
 - * sum-of-products (SOP) form
 - * product-of-sums (POS) form
- SOP form
 - * Write an AND term for each input combination that produces a 1 output
 - » Write the variable if its value is 1; complement otherwise
 - * OR the AND terms to get the final expression
- POS form
 - * Dual of the SOP form

Deriving Logical Expressions (cont'd)

- 3-input majority function

| A | B | C | F |
|----------|----------|----------|----------|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

- SOP logical expression

- Four product terms

* Because there are 4 rows with a 1 output

$$F = \bar{A} B C + A \bar{B} C + A B \bar{C} + A B C$$

- Sigma notation

$$\Sigma(3, 5, 6, 7)$$

Deriving Logical Expressions (cont'd)

- 3-input majority function

| A | B | C | F |
|----------|----------|----------|----------|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

- POS logical expression

- Four sum terms

* Because there are 4 rows with a 0 output

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

- Pi notation

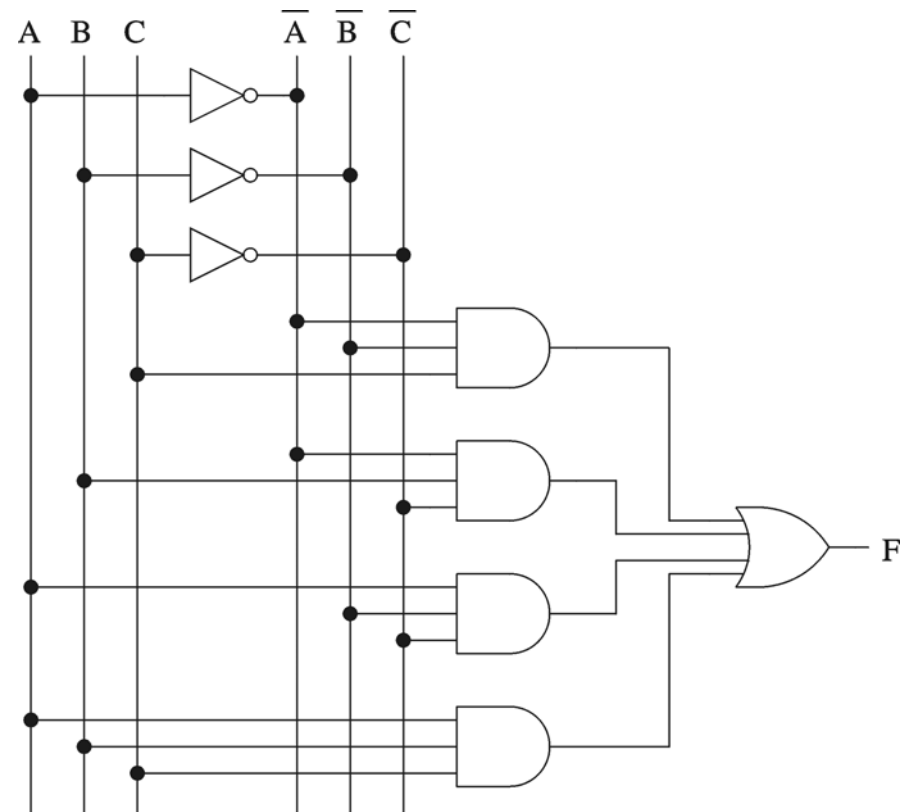
$$\Pi (0, 1, 2, 4)$$

Brute Force Method of Implementation

3-input even-parity function

- SOP implementation

| A | B | C | 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 |

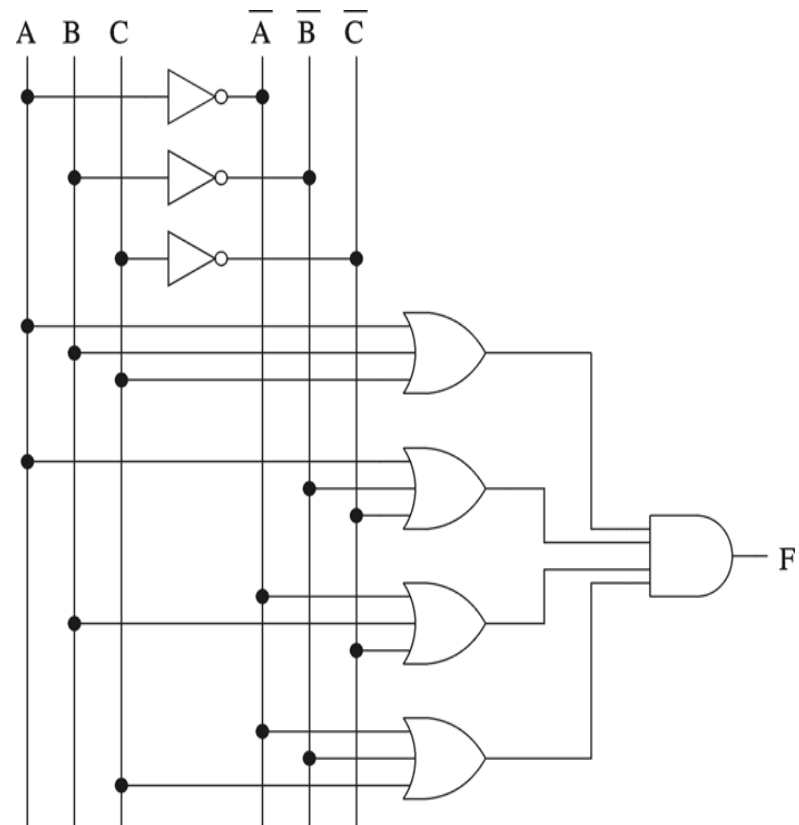


Brute Force Method of Implementation

3-input even-parity function

- POS implementation

| A | B | C | 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 |



Logical Expression Simplification

- Three basic methods
 - * Algebraic manipulation
 - » Use Boolean laws to simplify the expression
 - Difficult to use
 - Don't know if you have the simplified form
 - * Karnaugh map method
 - » Graphical method
 - » Easy to use
 - Can be used to simplify logical expressions with a few variables
 - * Quine-McCluskey method
 - » Tabular method
 - » Can be automated

Algebraic Manipulation

- Majority function example

$$\bar{A} B C + A \bar{B} C + A B \bar{C} + A B C =$$

Added extra

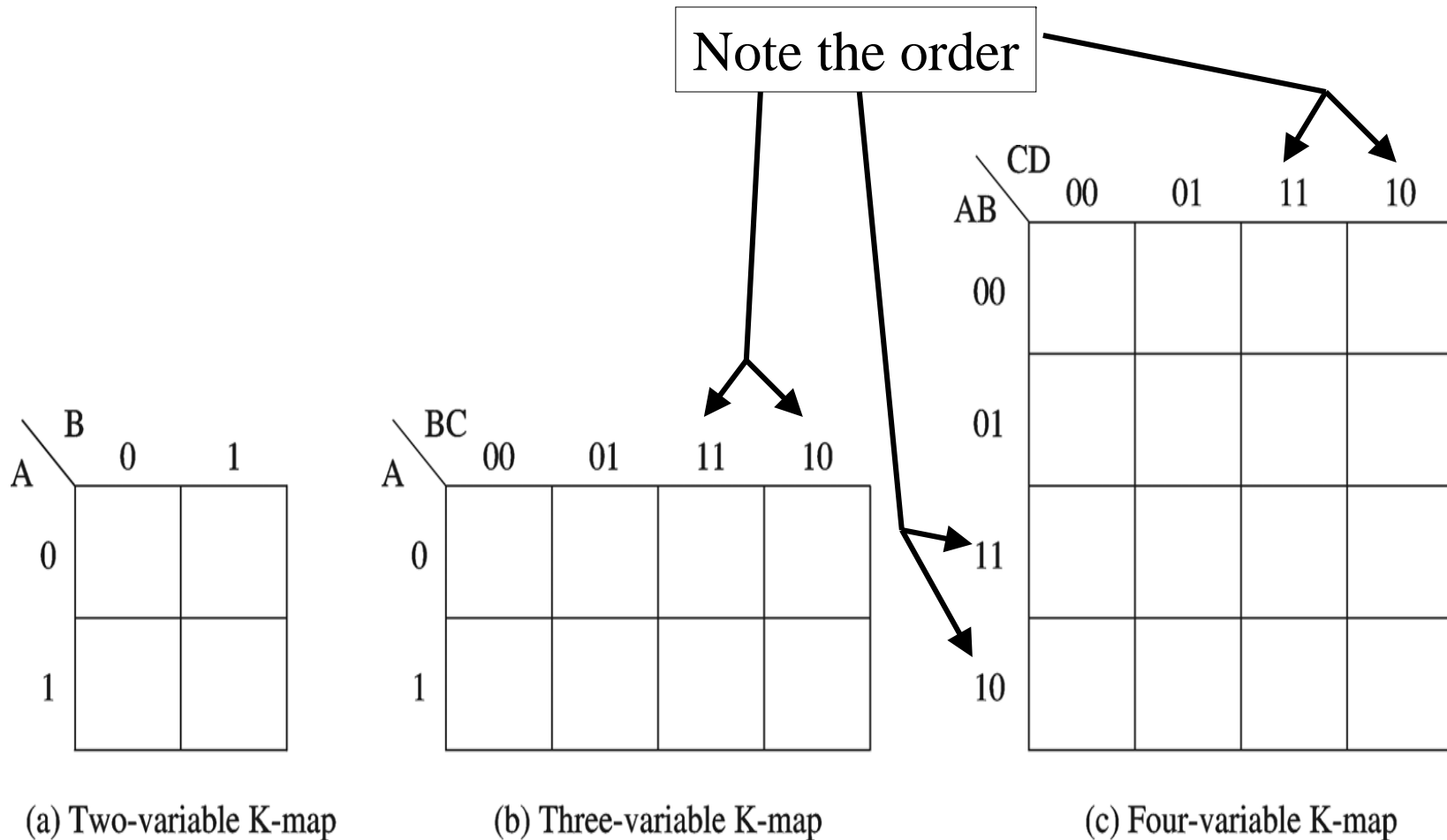
$$\bar{A} B C + A \bar{B} C + A B \bar{C} + A B C + A B C + A B C$$

- We can now simplify this expression as

$$B C + A C + A B$$

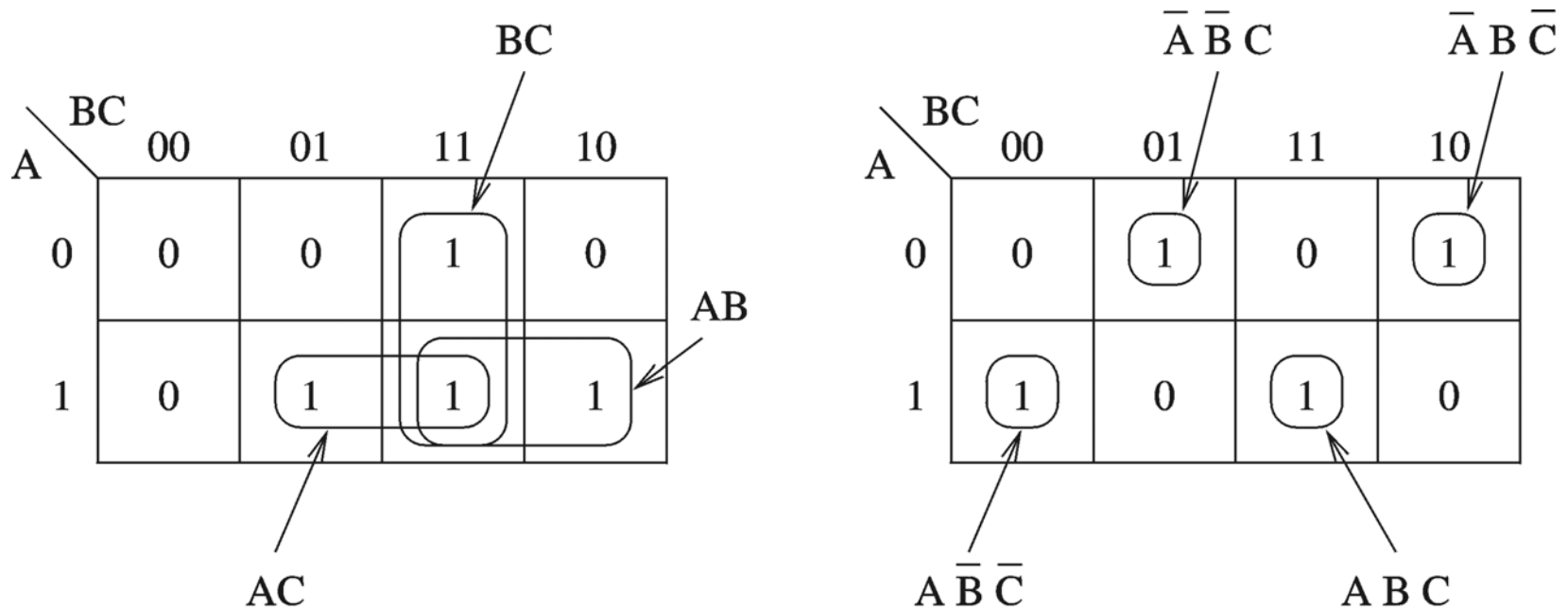
- A difficult method to use for complex expressions

Karnaugh Map Method



Karnaugh Map Method (cont'd)

Simplification examples

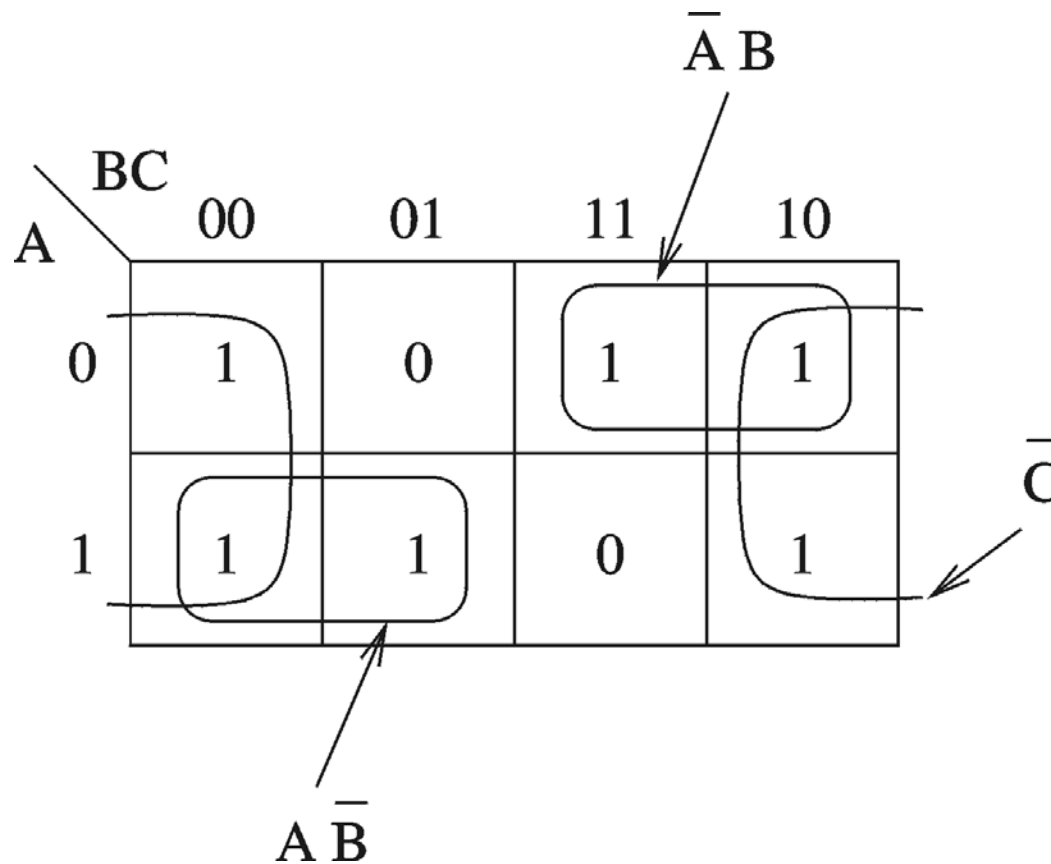


(a) Majority function

(b) Even-parity function

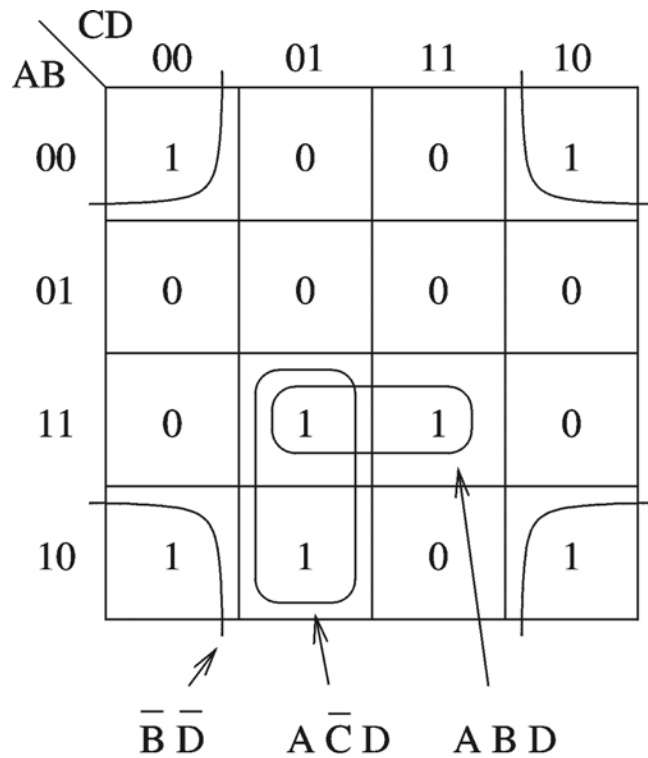
Karnaugh Map Method (cont'd)

First and last columns/rows are adjacent

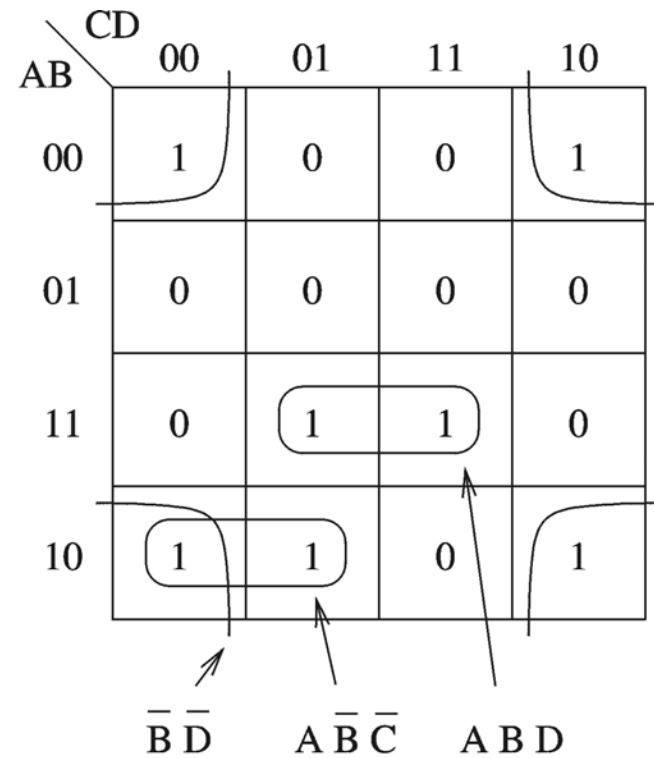


Karnaugh Map Method (cont'd)

Minimal expression depends on groupings



(a)



(b)

Karnaugh Map Method (cont'd)

No redundant groupings

| AB \ CD | 00 | 01 | 11 | 10 |
|---------|----|----|----|----|
| 00 | 0 | 0 | 1 | 0 |
| 01 | 1 | 1 | 1 | 0 |
| 11 | 0 | 1 | 1 | 1 |
| 10 | 0 | 1 | 0 | 0 |

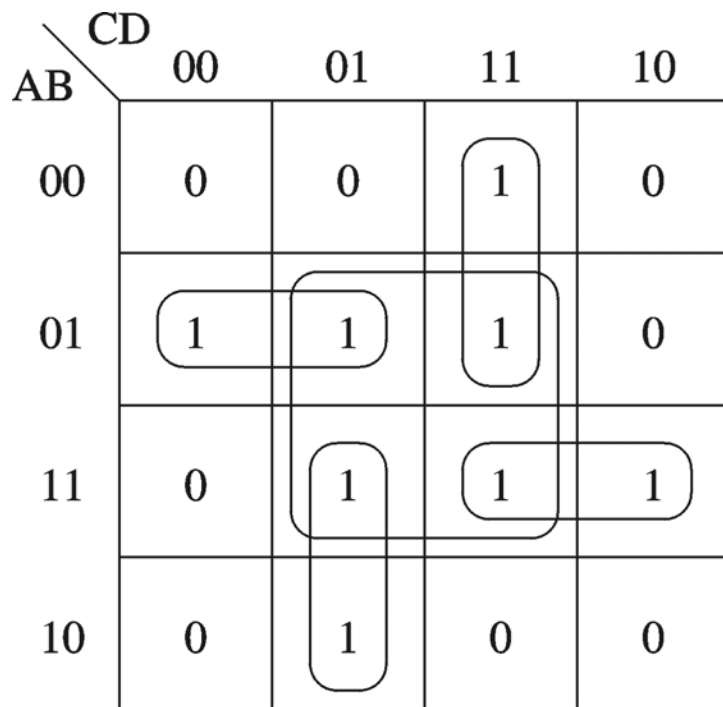


Diagram (a) shows a Karnaugh map with four rows (AB = 00, 01, 11, 10) and four columns (CD = 00, 01, 11, 10). The cells containing '1' are at (00,11), (01,00), (01,01), (01,11), (11,01), (11,11), (11,10), and (10,01). The groups shown are: a vertical group of three 1s in column CD=11 (rows AB=00, 01, 11); a horizontal group of three 1s in row AB=01 (columns CD=00, 01, 11); a vertical group of two 1s in column CD=01 (rows AB=11, 10); a horizontal group of two 1s in row AB=11 (columns CD=11, 10); and a vertical group of two 1s in column CD=01 (rows AB=01, 11). The groups for CD=11 and AB=01 overlap, and the groups for CD=01 and AB=11 overlap, indicating nonminimal simplification.

(a) Nonminimal simplification

| AB \ CD | 00 | 01 | 11 | 10 |
|---------|----|----|----|----|
| 00 | 0 | 0 | 1 | 0 |
| 01 | 1 | 1 | 1 | 0 |
| 11 | 0 | 1 | 1 | 1 |
| 10 | 0 | 1 | 0 | 0 |

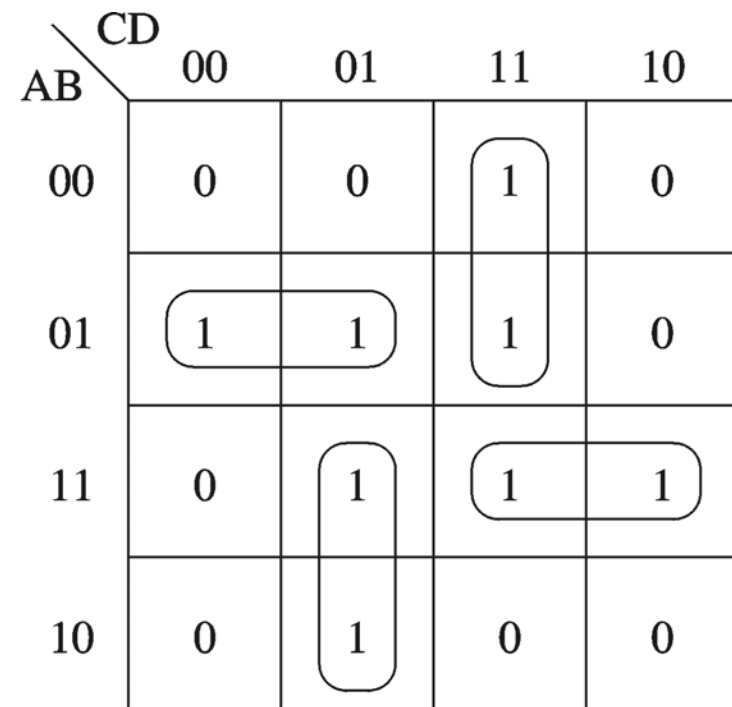
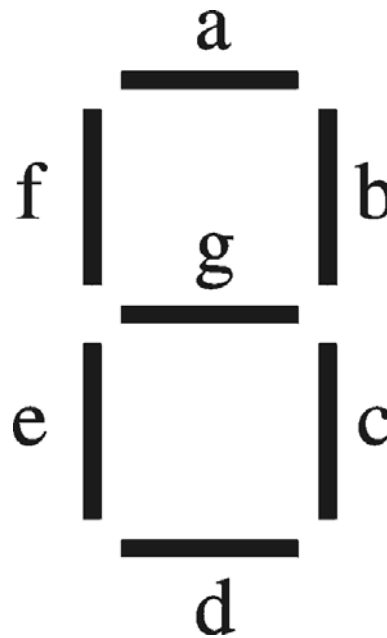


Diagram (b) shows the same Karnaugh map as (a). The groups shown are: a vertical group of three 1s in column CD=11 (rows AB=00, 01, 11); a horizontal group of two 1s in row AB=01 (columns CD=00, 01); a vertical group of two 1s in column CD=01 (rows AB=11, 10); and a horizontal group of two 1s in row AB=11 (columns CD=11, 10). There are no overlapping groups, indicating minimal simplification.

(b) Minimal simplification

Karnaugh Map Method (cont'd)

- Example
 - * Seven-segment display
 - * Need to select the right LEDs to display a digit



Karnaugh Map Method (cont'd)

Truth table for segment d

| No | A | B | C | D | Seg. | No | A | B | C | D | Seg. |
|----|---|---|---|---|------|----|---|---|---|---|------|
| 0 | 0 | 0 | 0 | 0 | 1 | 8 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 9 | 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 1 | 0 | 1 | 10 | 1 | 0 | 1 | 0 | ? |
| 3 | 0 | 0 | 1 | 1 | 1 | 11 | 1 | 0 | 1 | 1 | ? |
| 4 | 0 | 1 | 0 | 0 | 0 | 12 | 1 | 1 | 0 | 0 | ? |
| 5 | 0 | 1 | 0 | 1 | 1 | 13 | 1 | 1 | 0 | 1 | ? |
| 6 | 0 | 1 | 1 | 0 | 1 | 14 | 1 | 1 | 1 | 0 | ? |
| 7 | 0 | 1 | 1 | 1 | 0 | 15 | 1 | 1 | 1 | 1 | ? |

Karnaugh Map Method (cont'd)

Don't cares simplify the expression a lot

| AB \ CD | 00 | 01 | 11 | 10 |
|---------|----|----|----|----|
| 00 | 1 | 0 | 1 | 1 |
| 01 | 0 | 1 | 0 | 1 |
| 11 | 0 | 0 | 0 | 0 |
| 10 | 1 | 1 | 0 | 0 |

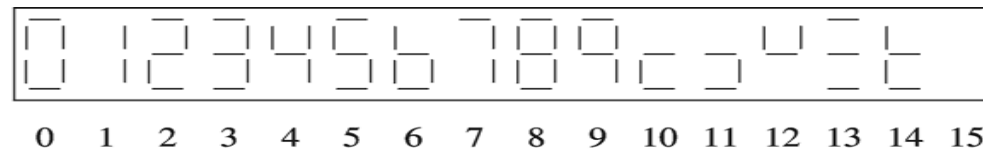
(a) Simplification with no don't cares

| AB \ CD | 00 | 01 | 11 | 10 |
|---------|----|----|----|----|
| 00 | 1 | 0 | 1 | 1 |
| 01 | 0 | 1 | 0 | 1 |
| 11 | d | d | d | d |
| 10 | 1 | 1 | d | d |

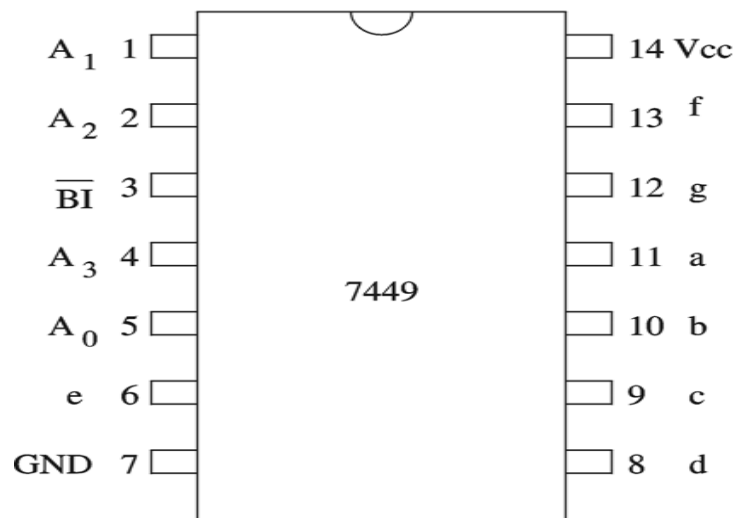
(b) Simplification with don't cares

Karnaugh Map Method (cont'd)

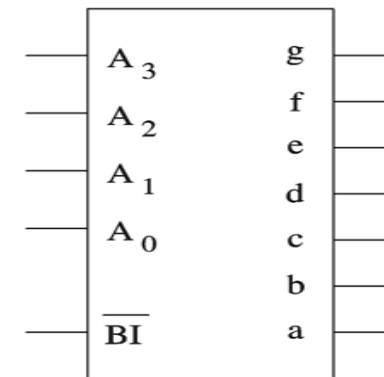
Example 7-segment display driver chip



(a) Display designations



(b) Connection diagram



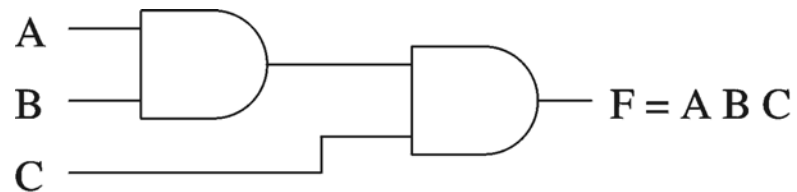
(c) Logic symbol

Quine-McCluskey Method

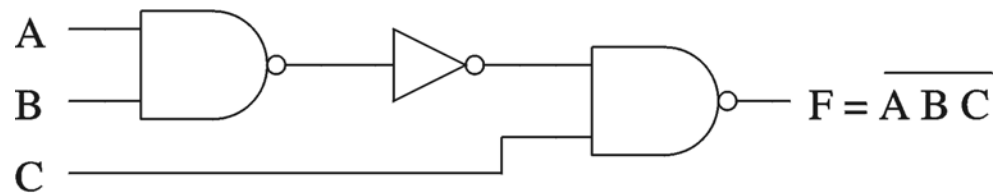
- Simplification involves two steps:
 - * Obtain a simplified expression
 - » Essentially uses the following rule
$$X Y + X \bar{Y} = X$$
 - » This expression need not be minimal
 - Next step eliminates any redundant terms
 - * Eliminate redundant terms from the simplified expression in the last step
 - » This step is needed even in the Karnaugh map method

Generalized Gates

- Multiple input gates can be built using smaller gates



- Some gates like AND are easy to build



- Other gates like NAND are more involved

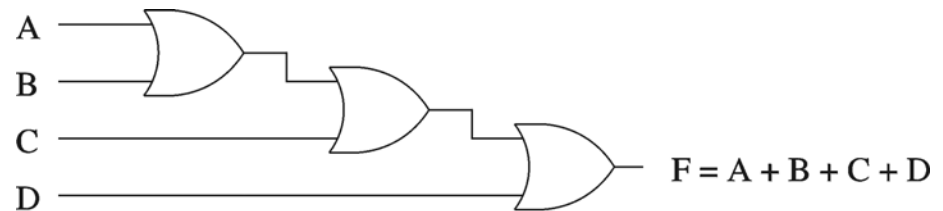
Generalized Gates (cont'd)

- Various ways to build higher-input gates

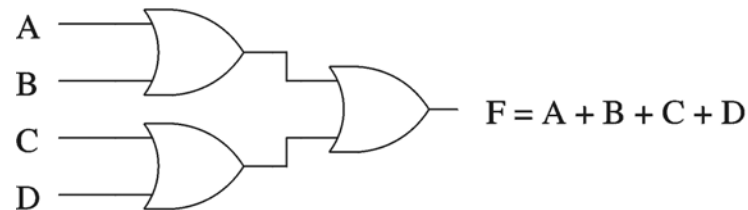
- * Series
- * Series-parallel

- Propagation delay depends on the implementation

- * Series implementation
 - » 3-gate delay
- * Series-parallel implementation
 - » 2-gate delay



(a) Series implementation



(b) Series-parallel implementation

Multiple Outputs

Two-output function

| A | B | C | F1 | F2 |
|----------|----------|----------|-----------|-----------|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

- F1 and F2 are familiar functions

- » F1 = Even-parity function

- » F2 = Majority function

- Another interpretation

- * Full adder

- » F1 = Sum

- » F2 = Carry

Implementation Using Other Gates

- Using NAND gates
 - * Get an equivalent expression

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

- * Using de Morgan's law

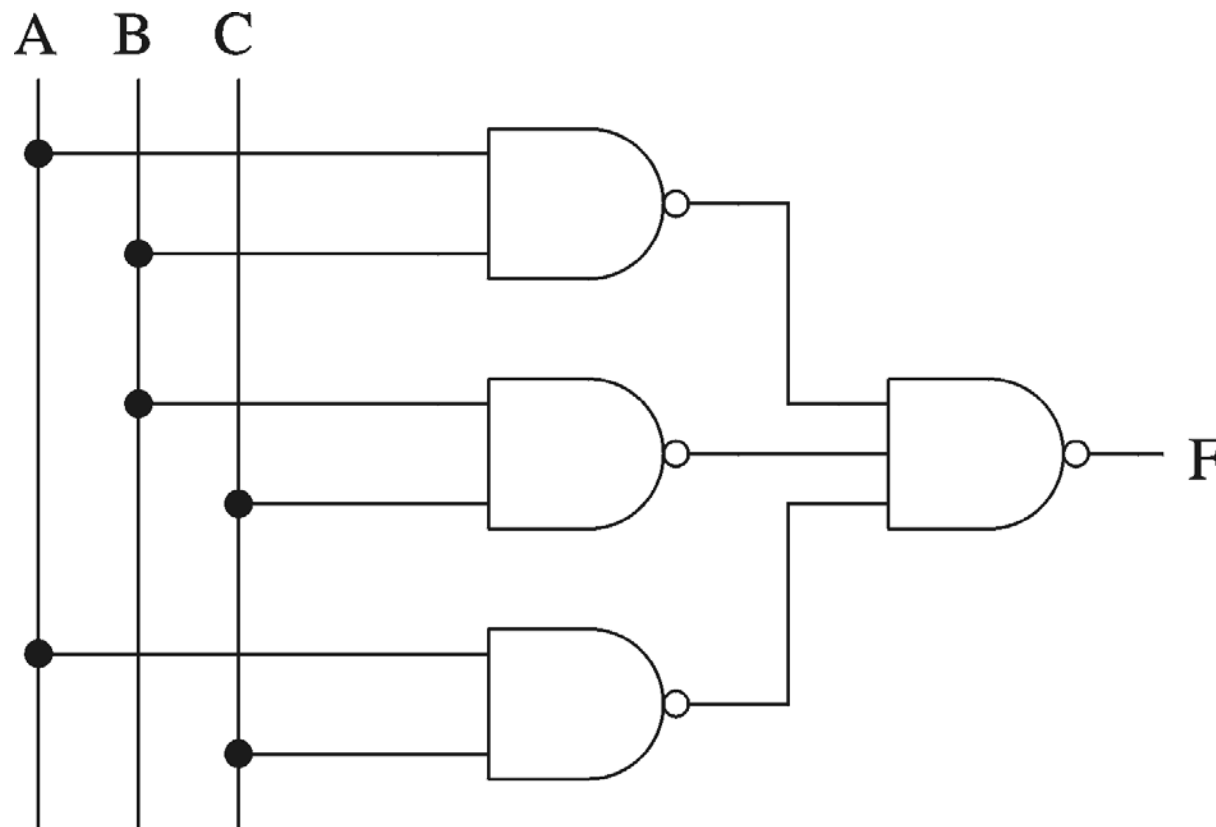
$$A B + C D = \overline{\overline{A B}} \cdot \overline{\overline{C D}}$$

- * Can be generalized
 - » Majority function

$$A B + B C + A C = \overline{\overline{A B}} \cdot \overline{\overline{B C}} \cdot \overline{\overline{A C}}$$

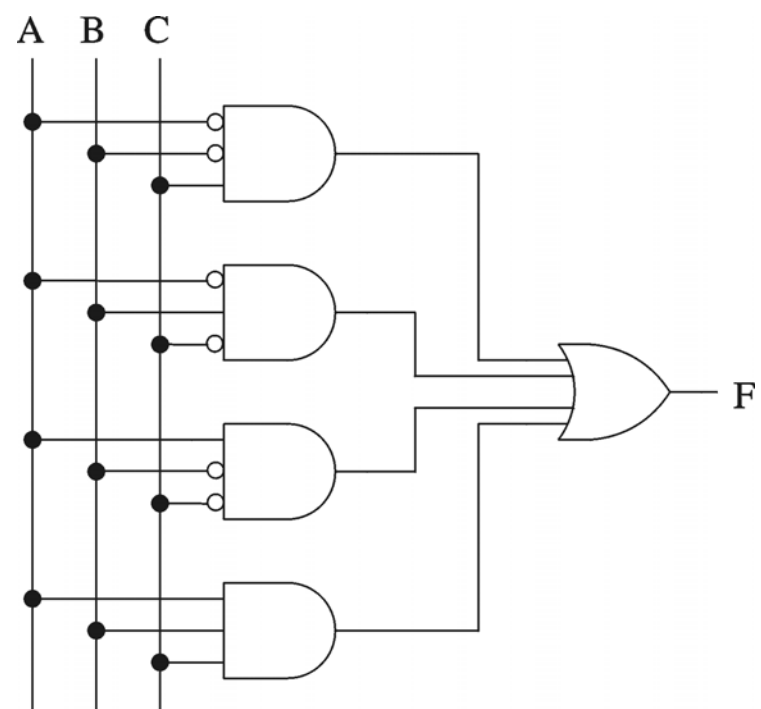
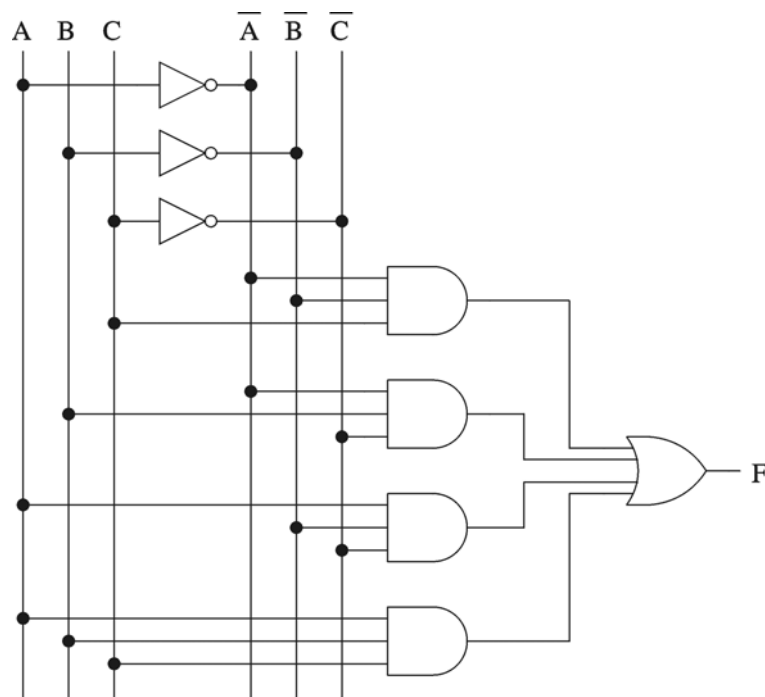
Implementation Using Other Gates (cont'd)

- Majority function



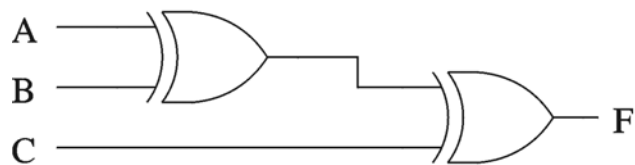
Implementation Using Other Gates (cont'd)

Bubble Notation

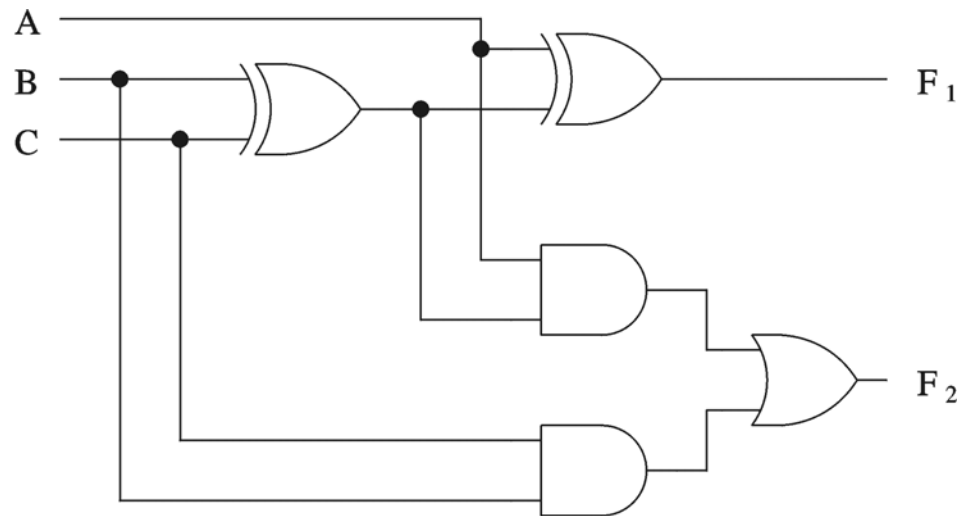


Implementation Using Other Gates (cont'd)

- Using XOR gates
 - * More complicated



(a) Even-parity function



(b) Two-output function

Summary

- Logic gates
 - » AND, OR, NOT
 - » NAND, NOR, XOR
- Logical functions can be represented using
 - » Truth table
 - » Logical expressions
 - » Graphical form
- Logical expressions
 - * Sum-of-products
 - * Product-of-sums

Summary (cont'd)

- Simplifying logical expressions
 - * Boolean algebra
 - * Karnaugh map
 - * Quine-McCluskey
- Implementations
 - * Using AND, OR, NOT
 - » Straightforward
 - * Using NAND
 - * Using XOR

Last slide