Boolean Algebra

Logic Gates

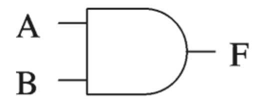
Introduction

- 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 processor consists of 3 million transistors
    - Now we can build chips with more than 100 million transistors

Basic Concepts

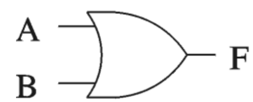
- Simple gates
  - AND
  - OR
  - NOT
- Functionality can be expressed by a truth table
  - A truth table lists output for each possible input combination
- Precedence
  - NOT > AND > OR
  - $F = A\,\overline{B} + \overline{A}\,B$

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



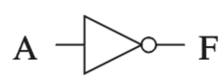
AND gate

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



OR gate

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



NOT gate

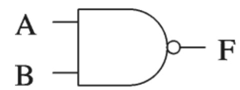
A	F
0	1
1	0

Logic symbol

Truth table

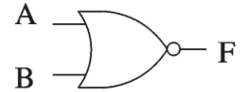
Basic Concepts (cont.)

- Additional useful gates
  - NAND
  - NOR
  - XOR
- NAND = AND + NOT
- NOR = OR + NOT
- XOR implements exclusive-OR function
- NAND and NOR gates require only 2 transistors
  - AND and OR need 3 transistors!



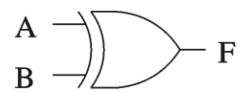
NAND gate

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.)

- Number of functions
  - With N logical variables, we can define

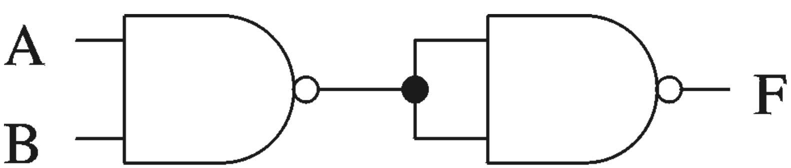
    2^{2^N} functions
  - Some of them are useful
    - AND, NAND, NOR, XOR, …
  - Some are not useful:
    - Output is always 1
    - Output is always 0
  - “Number of functions” definition is useful in proving completeness property

Basic Concepts (cont.)

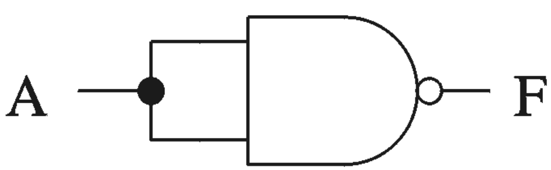
- Complete sets
  - A set of gates is complete
    - If we can implement any logical function using only the type of gates in the set
      - You can uses as many gates as you want
  - Some example complete sets
    - {AND, OR, NOT} ← Not a minimal complete set
    - {AND, NOT}
    - {OR, NOT}
    -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

Basic Concepts (cont.)

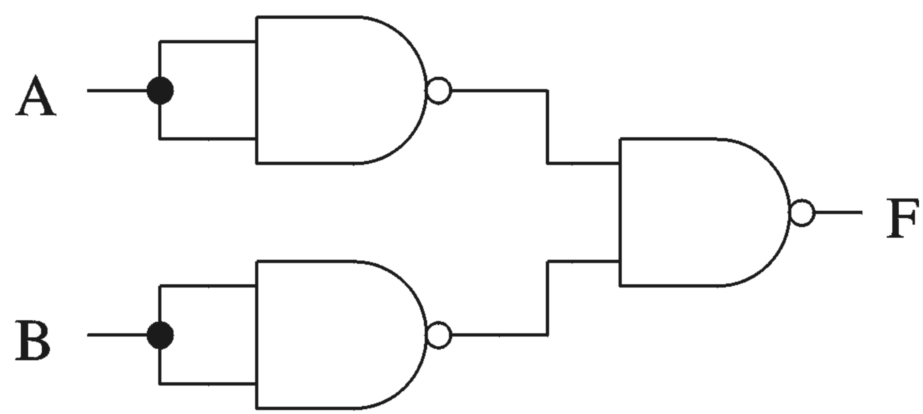
- Proving NAND gate is universal



AND gate



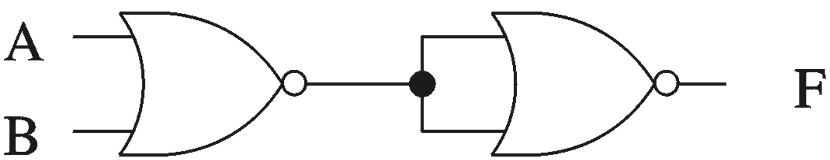
NOT gate



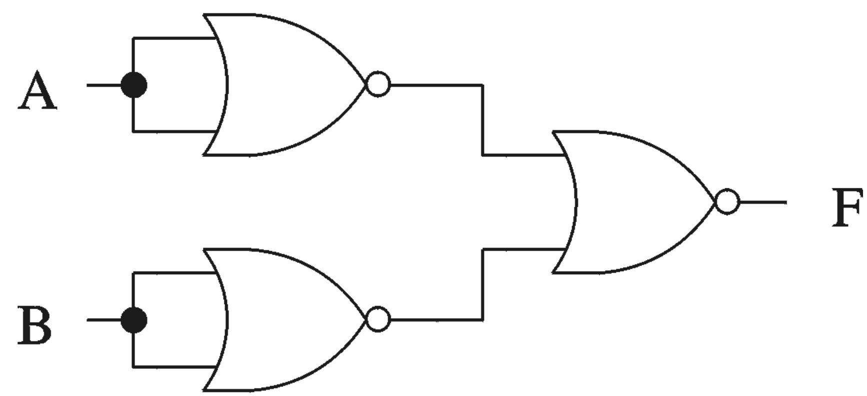
OR gate

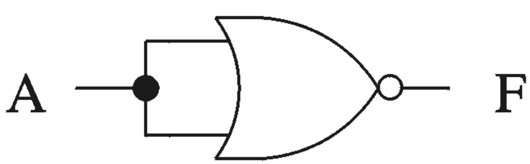
Basic Concepts (cont.)

- Proving NOR gate is universal



OR gate

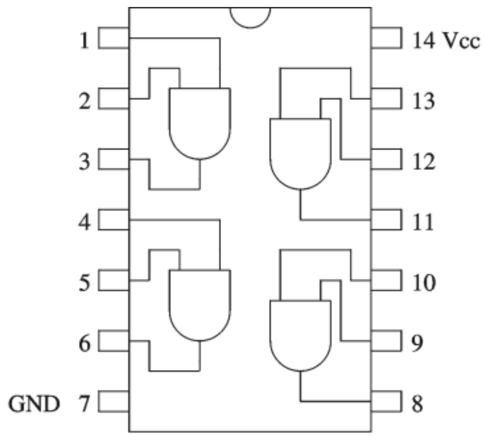




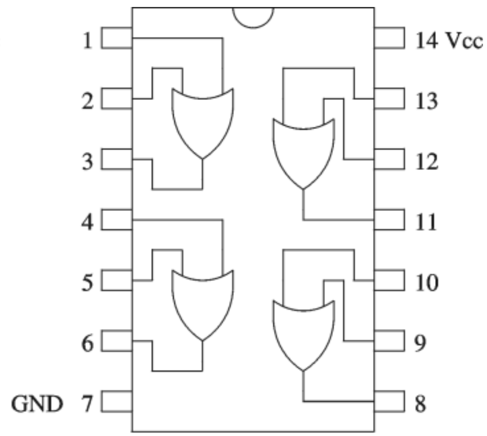
NOT gate

AND gate

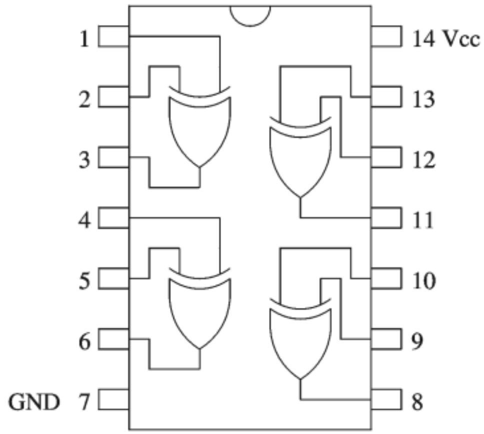
Logic Chips



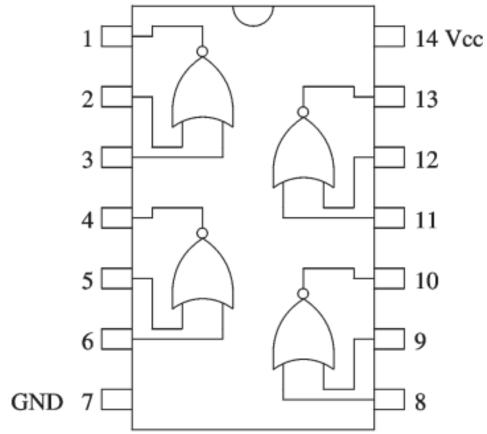
7408



7432



7486



7402

Logic Chips (cont.)

- 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

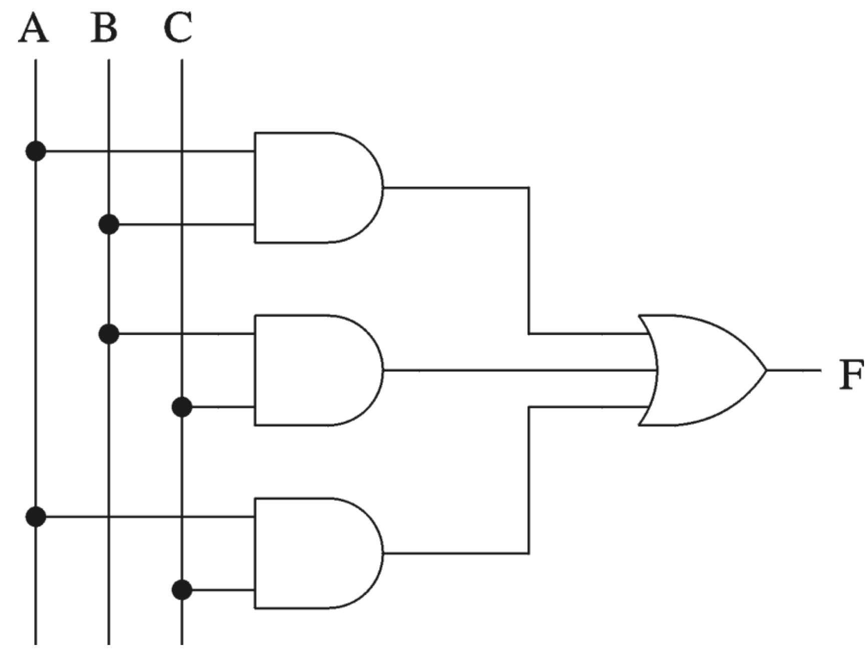
Logic Functions (cont.)

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

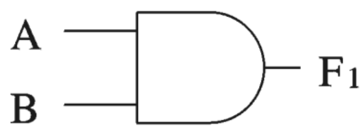
- Logical expression form

  F = A B + B C + A C

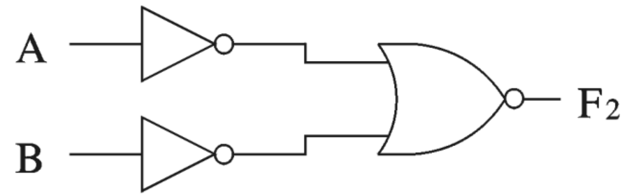


Logical Equivalence

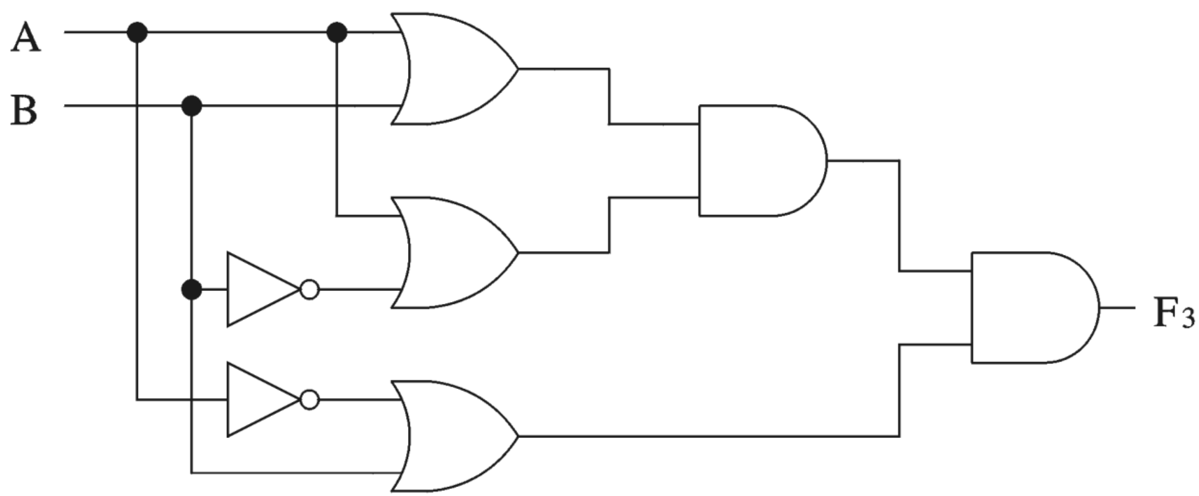
- All three circuits implement F = A B function



(a)



(b)



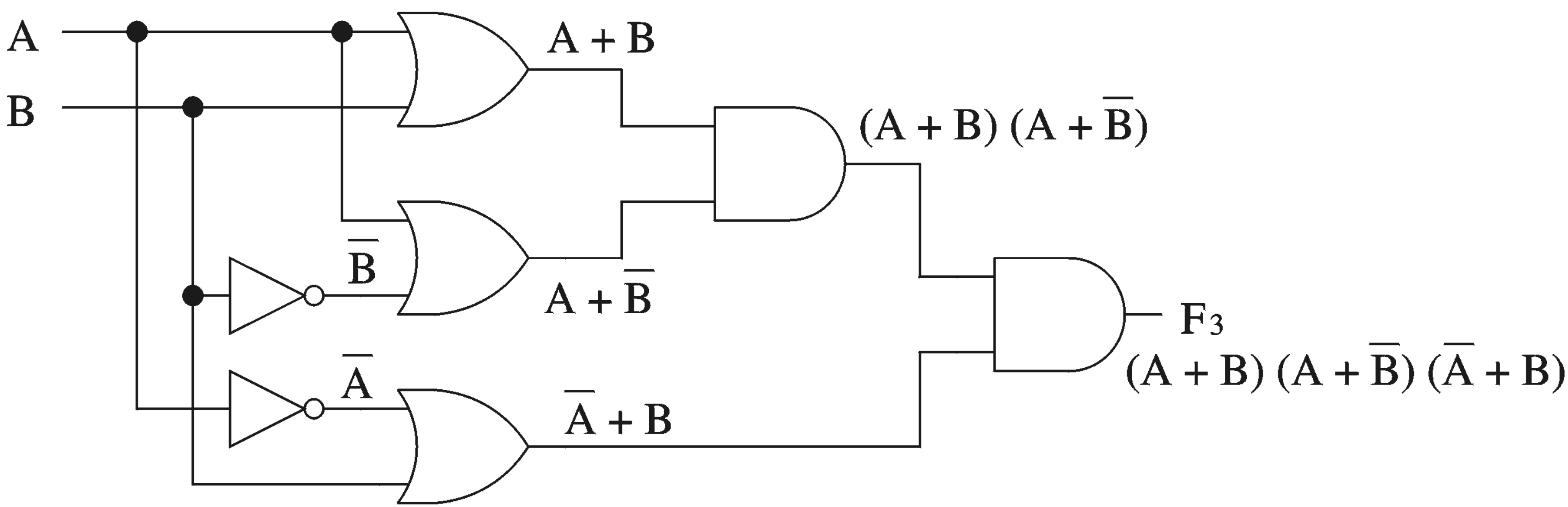
(c)

Logical Equivalence (cont.)

- 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.)

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



Logical Equivalence (cont.)

- Proving logical equivalence: Truth table method

A	B	F1 = A B	F3 = $(A + B)(A + \overline{B})(\overline{A} + B)$
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	$x \cdot \overline{x} = 0$	$x + \overline{x} = 1$
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.)

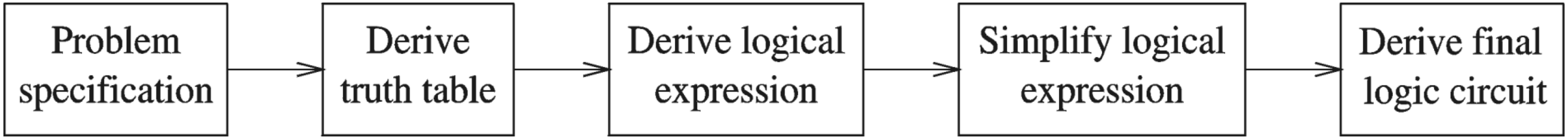
Name	AND version	OR version
Involution	$\overline{\overline{x}} = x$	---
Absorption	$x \cdot (x+y) = x$	$x + (x \cdot y) = x$
Associative	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$	$x + (y + z) = (x + y) + z$
de Morgan	$\overline{x \cdot y} = \overline{x} + \overline{y}$	$\overline{x + y} = \overline{x}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

Boolean Algebra (cont.)

- 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

- 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

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

Deriving Logical Expressions

- 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

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

Logical Expression Simplification

- Algebraic manipulation
  - Use Boolean laws to simplify the expression
    - Difficult to use
    - Don’t know if you have the simplified form

Algebraic Manipulation

- Majority function example

Added extra

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

$$\overline{A}\,B\,C + A\,\overline{B}\,C + A\,B\,\overline{C} + A\,B\,C + A\,B\,C + A\,B\,C$$

- We can now simplify this expression as

$$B\,C + A\,C + A\,B$$

- A difficult method to use for complex expressions

Implementation Using NAND Gates

- Using NAND gates
  - Get an equivalent expression

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

  - Using de Morgan's law

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

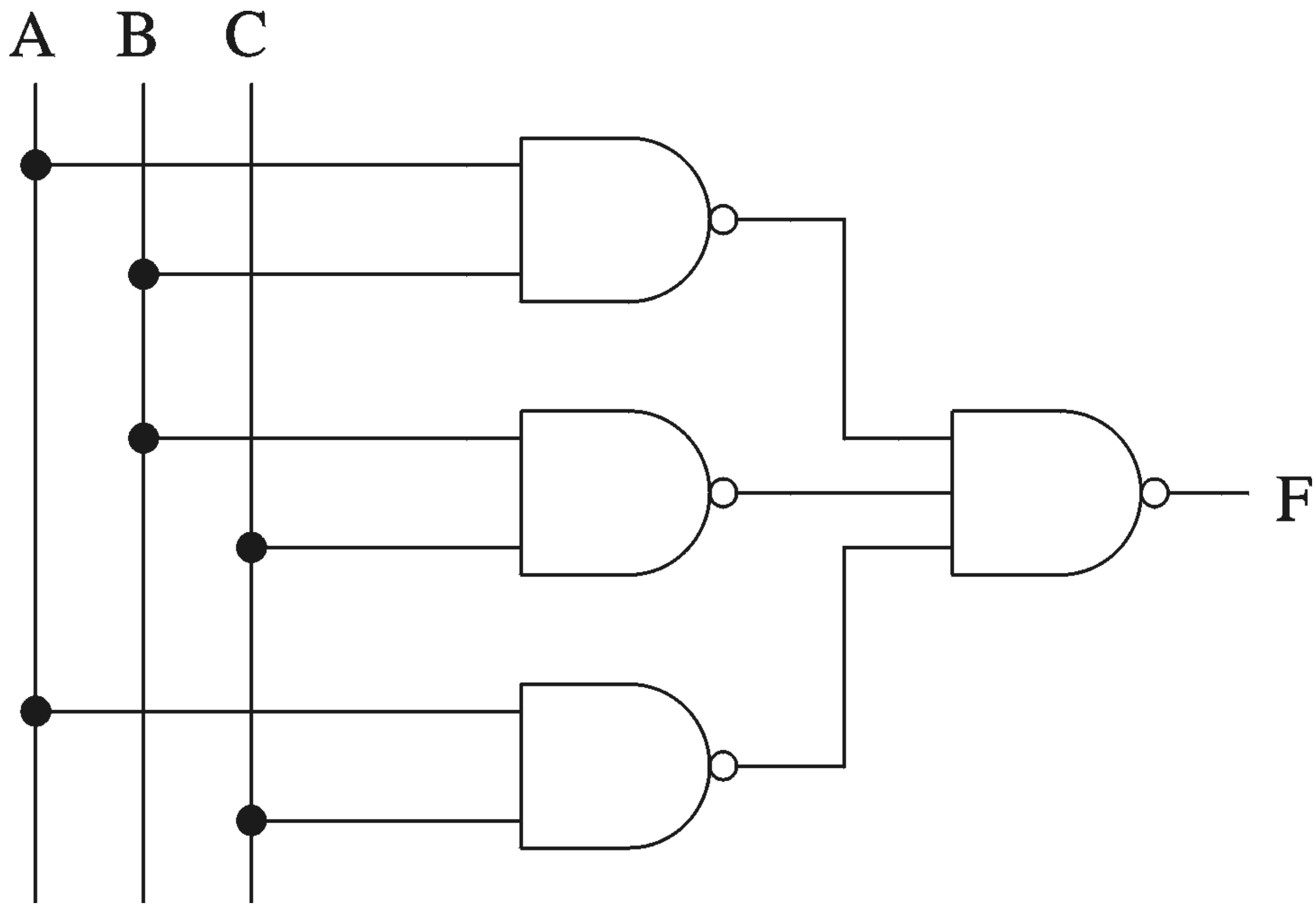
  - Can be generalized
    - Majority function

$$A\,B + B\,C + AC = \overline{\overline{A\,B} \cdot \overline{BC} \cdot \overline{AC}}$$

Idea: NAND Gates: Sum-of-Products, NOR Gates: Product-of-Sums

Implementation Using NAND Gates (cont.)

- Majority function



Introduction to Combinational Circuits

- Combinational circuits
    - Output depends only on the current inputs
- Combinational circuits provide a higher level of abstraction
    - Help in reducing design complexity
    - Reduce chip count
- We look at some useful combinational circuits