Digital Logic Design

- Basics
- Combinational Circuits

Adapted from the slides prepared by S. Dandamudi for the book, Fundamentals of Computer Organization and Design.

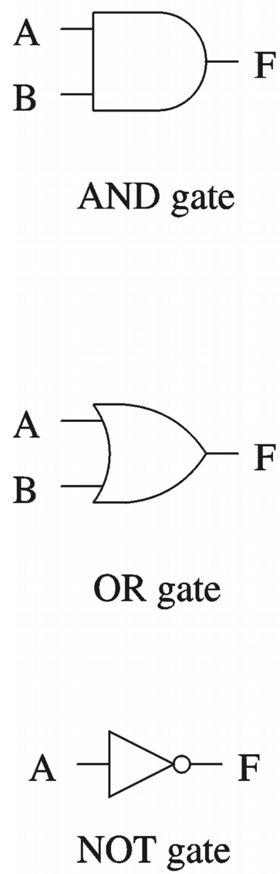
Introduction to Digital Logic Basics

- 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
- Precedence
  - NOT > AND > OR
  - $F = A\,\overline{B} + \overline{A}\,B$

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



Logic symbol

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

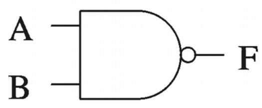
A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

A	F
0	1
1	0

Truth table

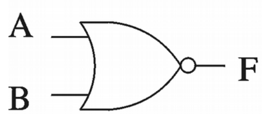
Basic Concepts (cont.)

- 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!



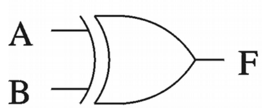
NAND gate

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

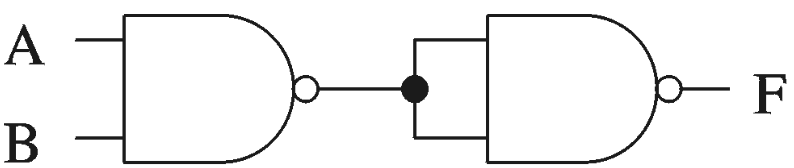
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Logic symbol

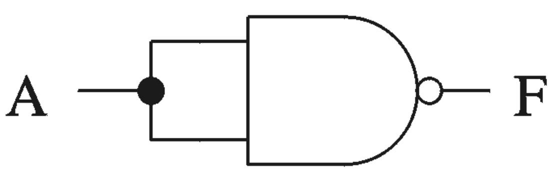
Truth table

Basic Concepts (cont.)

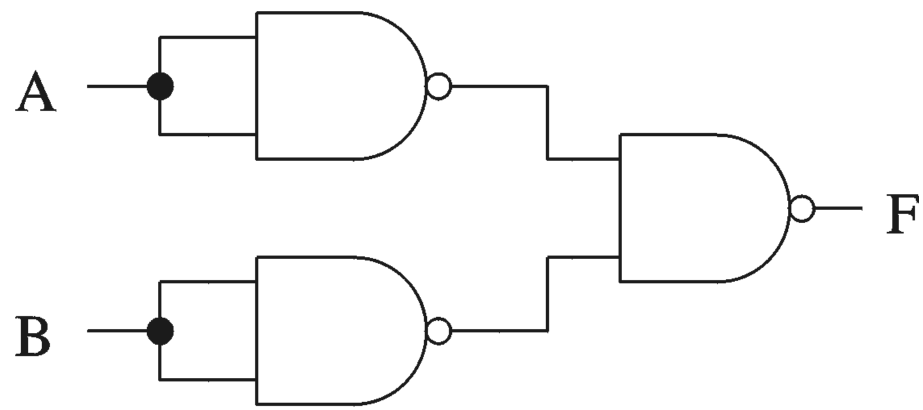
- Proving NAND gate is universal



AND gate



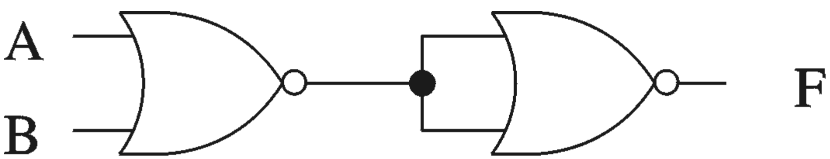
NOT gate



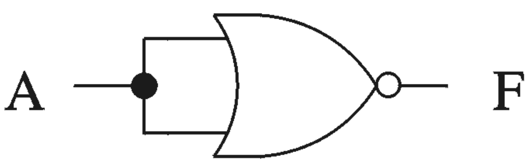
OR gate

Basic Concepts (cont.)

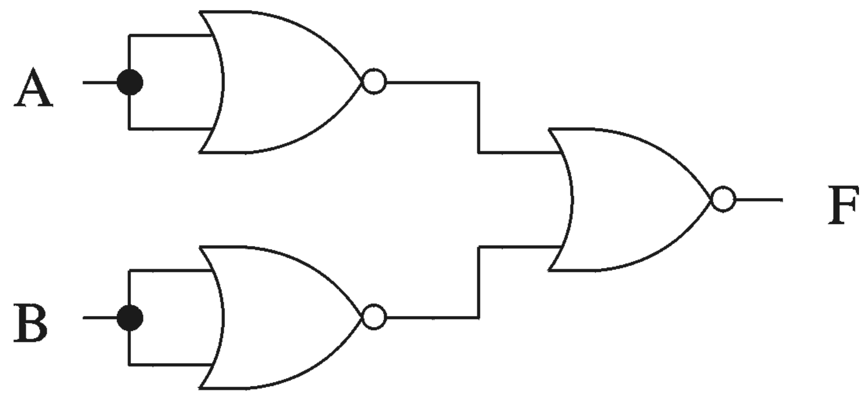
- Proving NOR gate is universal



OR gate

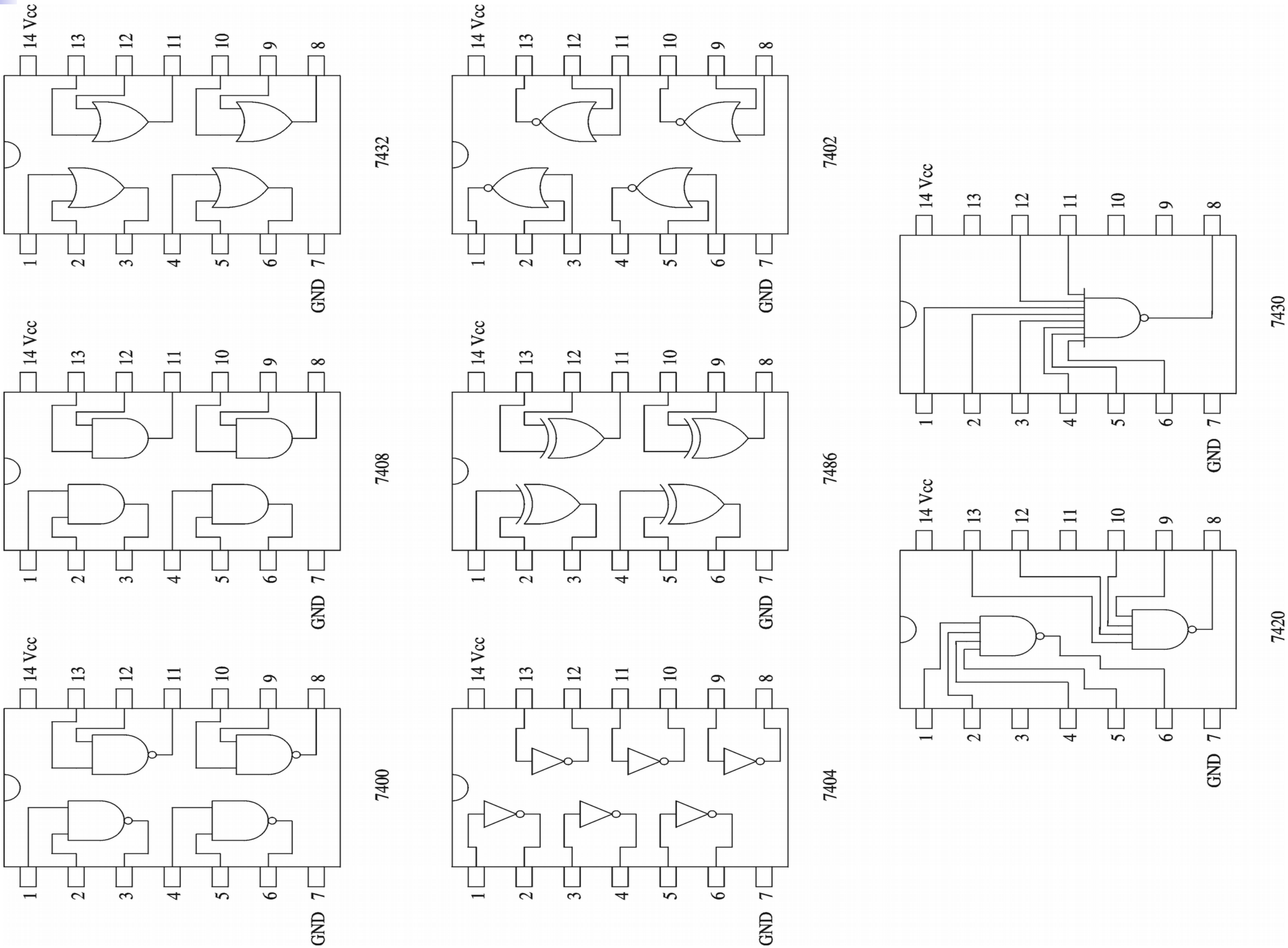


NOT gate



AND gate

Logic Chips (cont.)



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

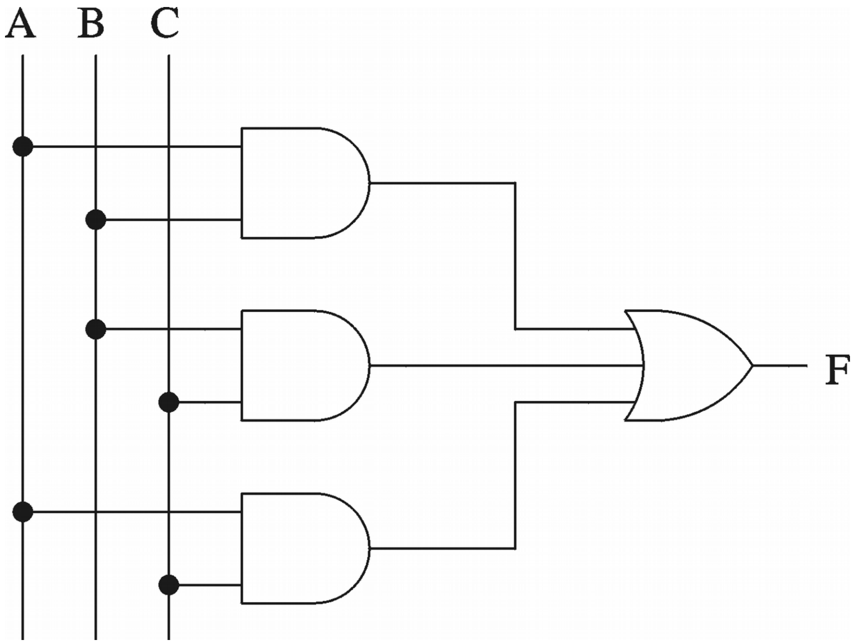
Logic Functions (cont.)

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

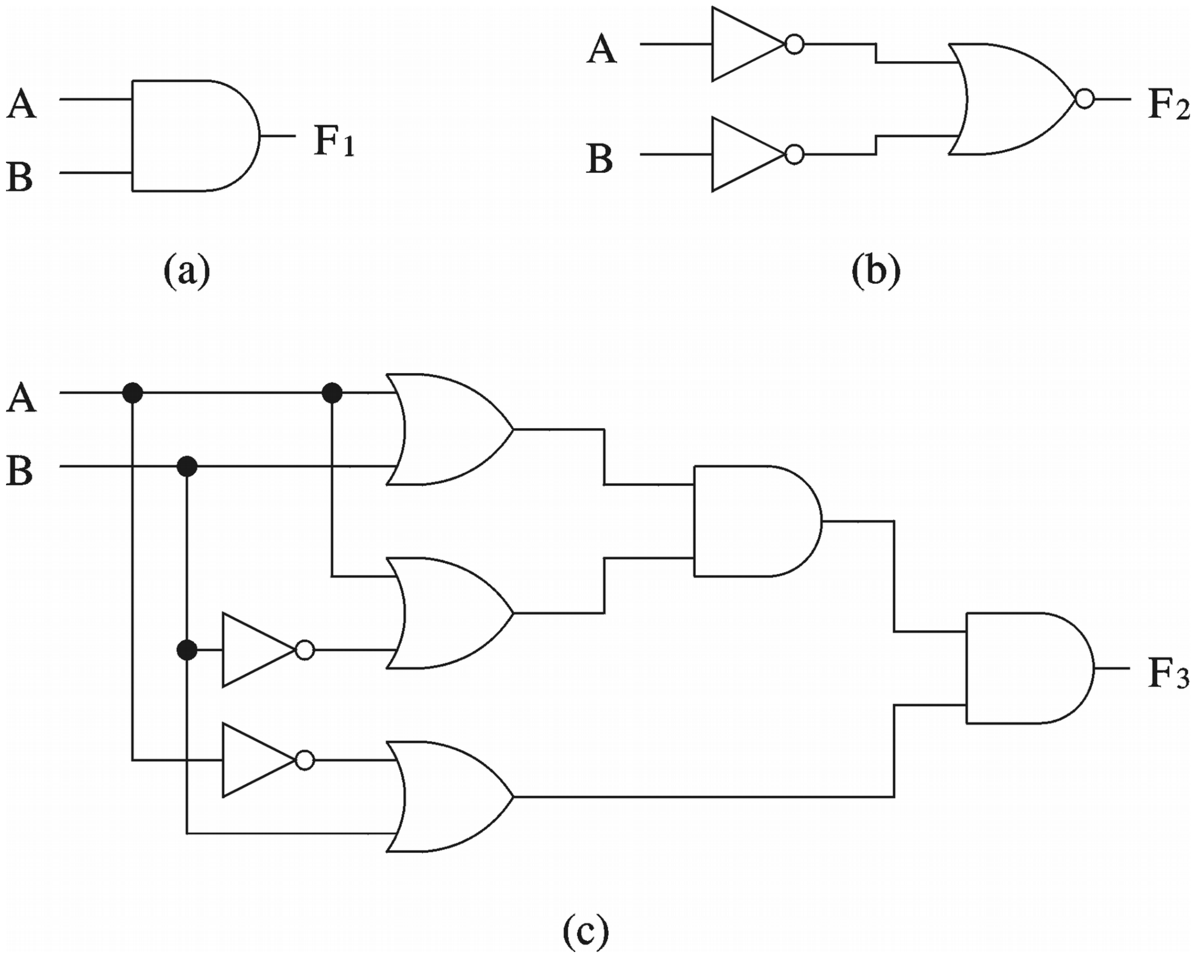
- Logical expression form

$$F = A\,B + B\,C + A\,C$$



Logical Equivalence

- All three circuits implement F = A B function

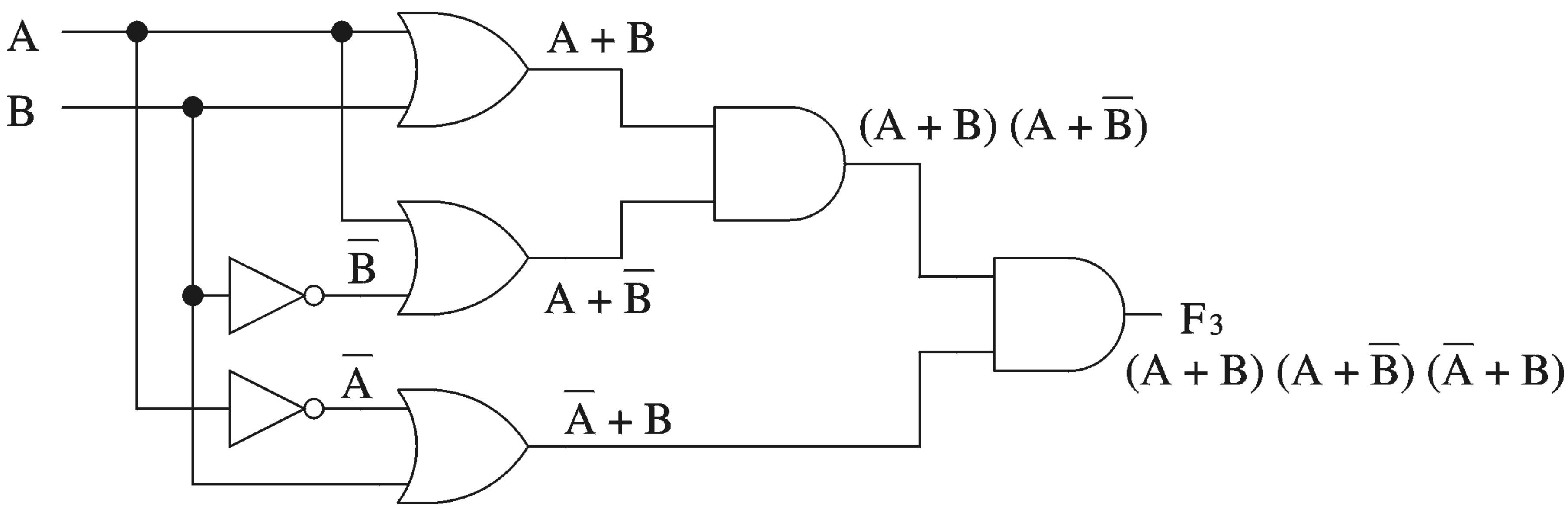


Logical Equivalence (cont.)

- 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	$\overline{x \cdot y} = \overline{x} + \overline{y}$	$\overline{x + y} = \overline{x} \cdot \overline{y}$

Logical Expression Simplification

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

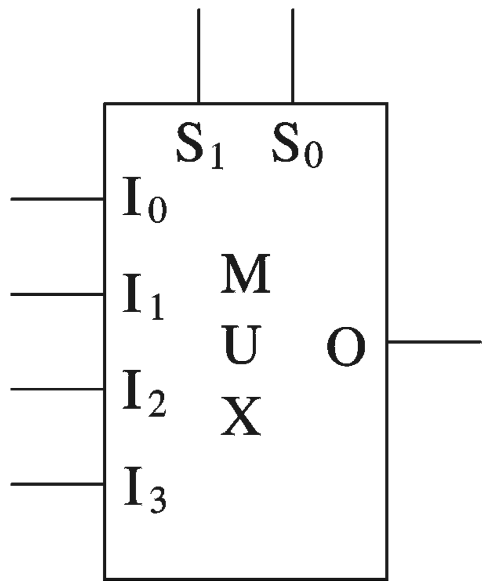
Introduction to Combinational Circuits

- 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

Multiplexers

- Multiplexer
  - 2^n data inputs
  - n selection inputs
  - a single output
- Selection input determines the input that should be connected to the output

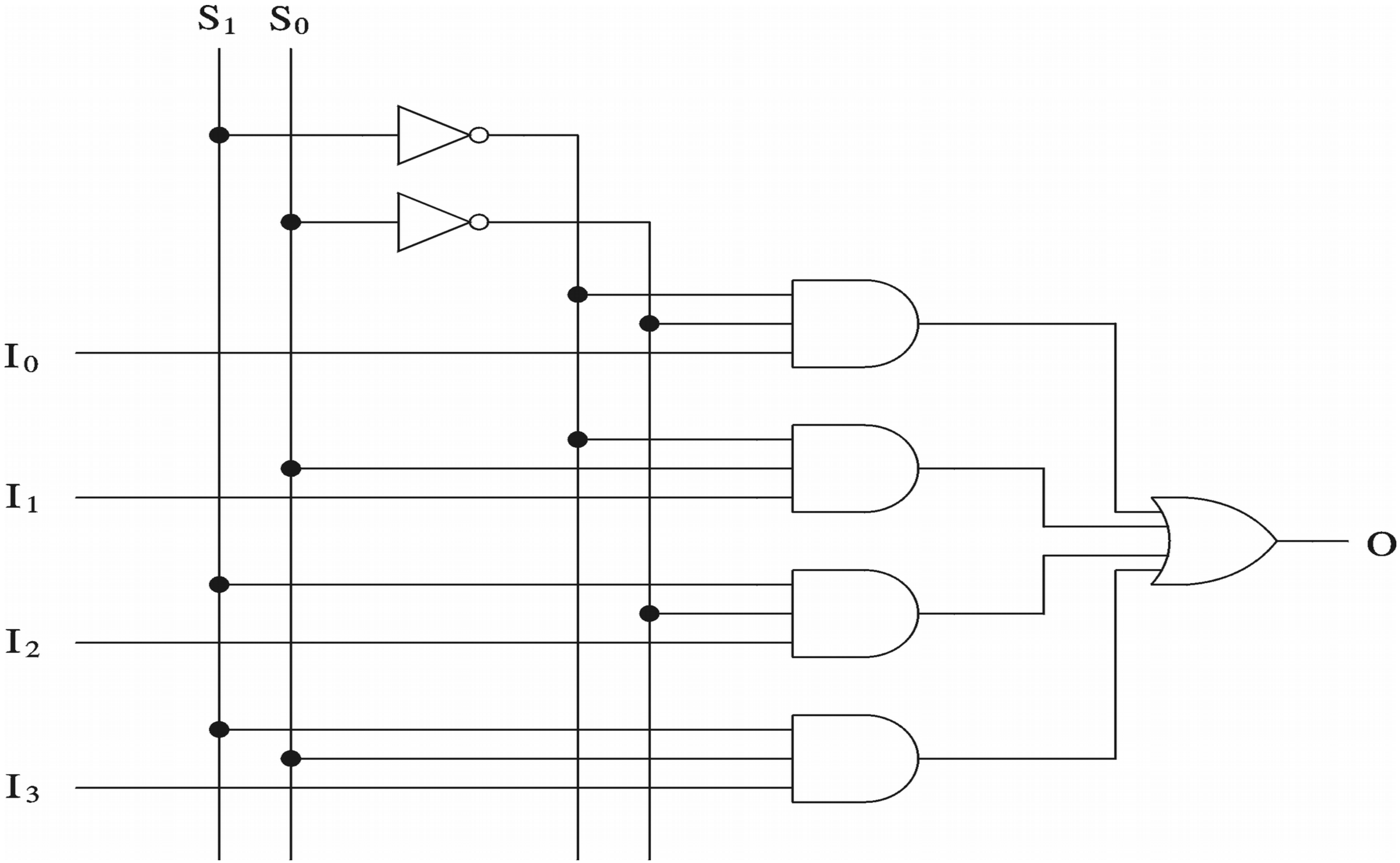
4-data input MUX



S_1	S_0	O
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

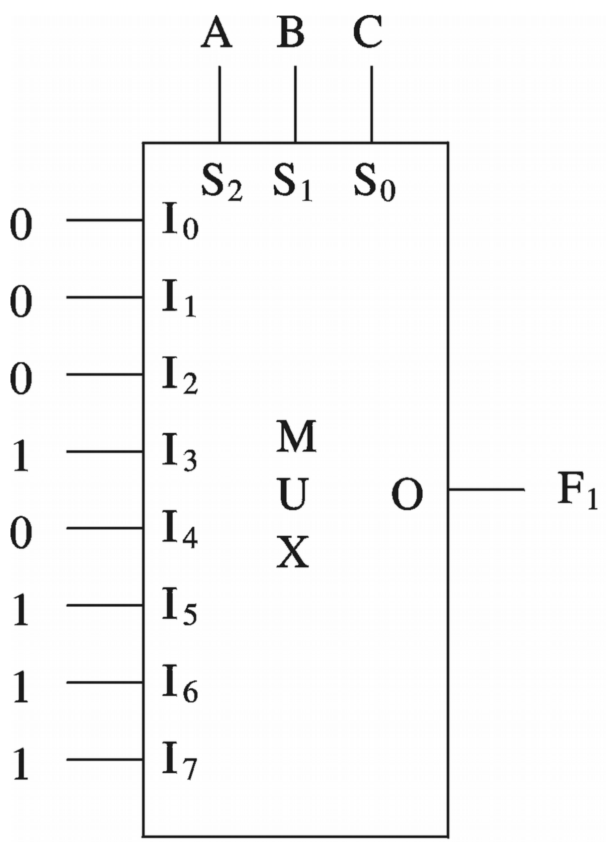
Multiplexers (cont.)

4-data input MUX implementation

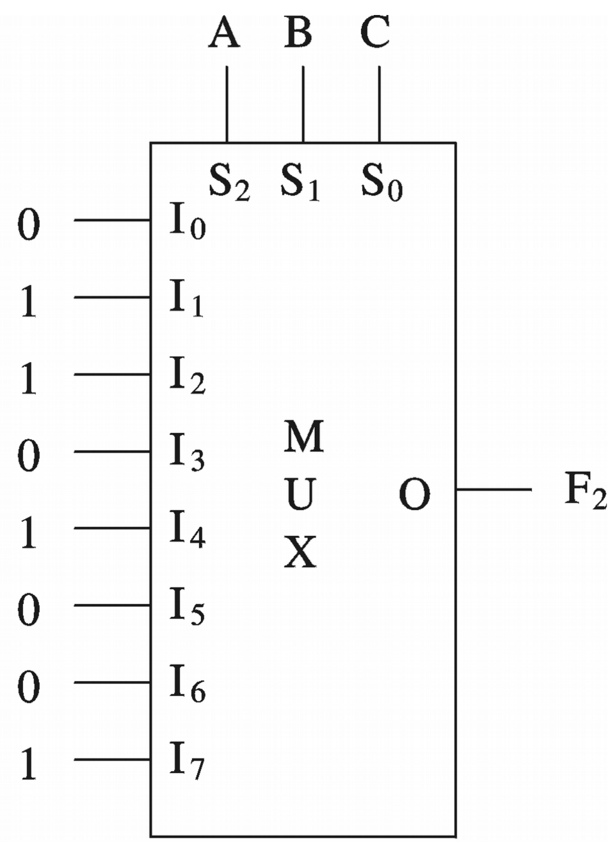


Multiplexers (cont.)

MUX implementations



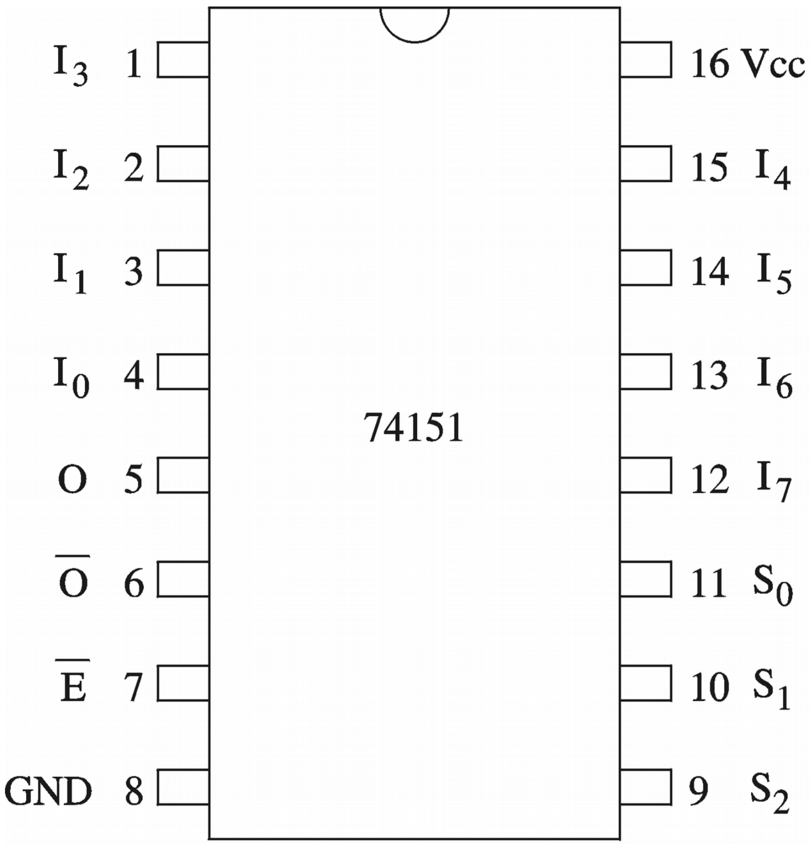
Majority function



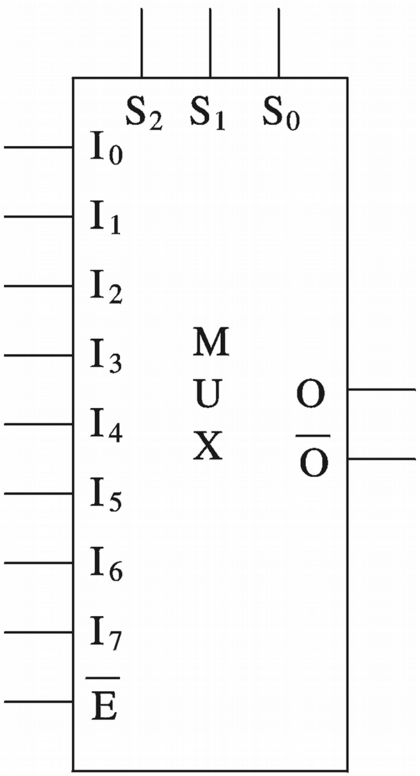
Even-parity function

Multiplexers (cont.)

Example chip: 8-to-1 MUX



(a) Connection diagram



(b) Logic symbol

Multiplexers (cont.)

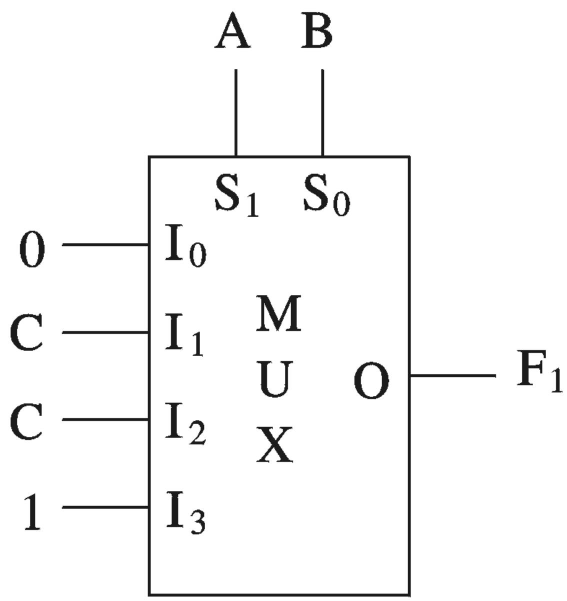
Efficient implementation: Majority function

Original truth table

A	B	C	F_1
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

New truth table

A	B	F_1
0	0	0
0	1	C
1	0	C
1	1	1



Multiplexers (cont.)

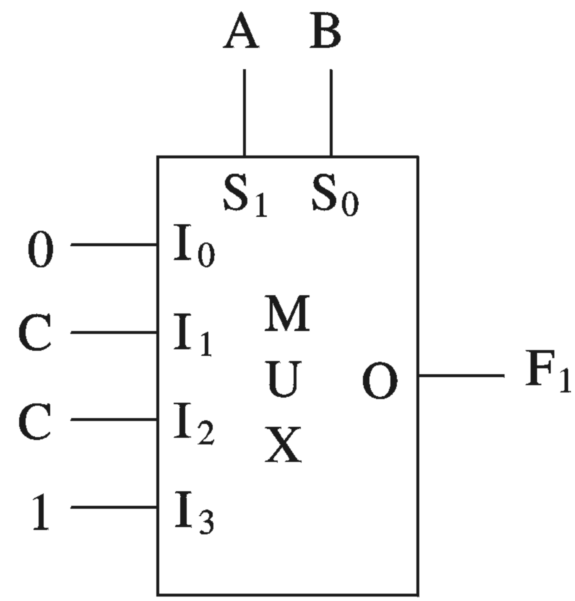
Efficient implementation: Majority function

Original truth table

A	B	C	F_1
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

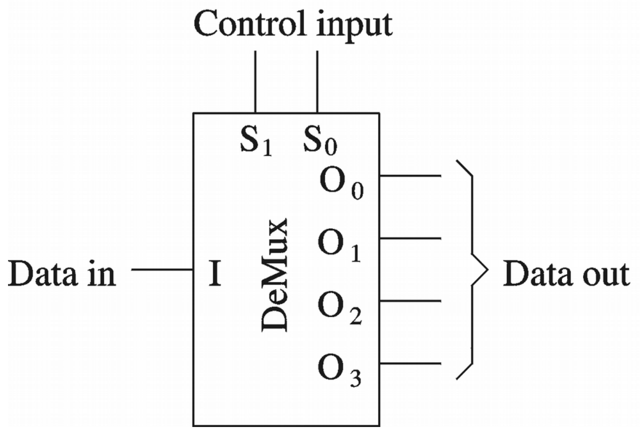
New truth table

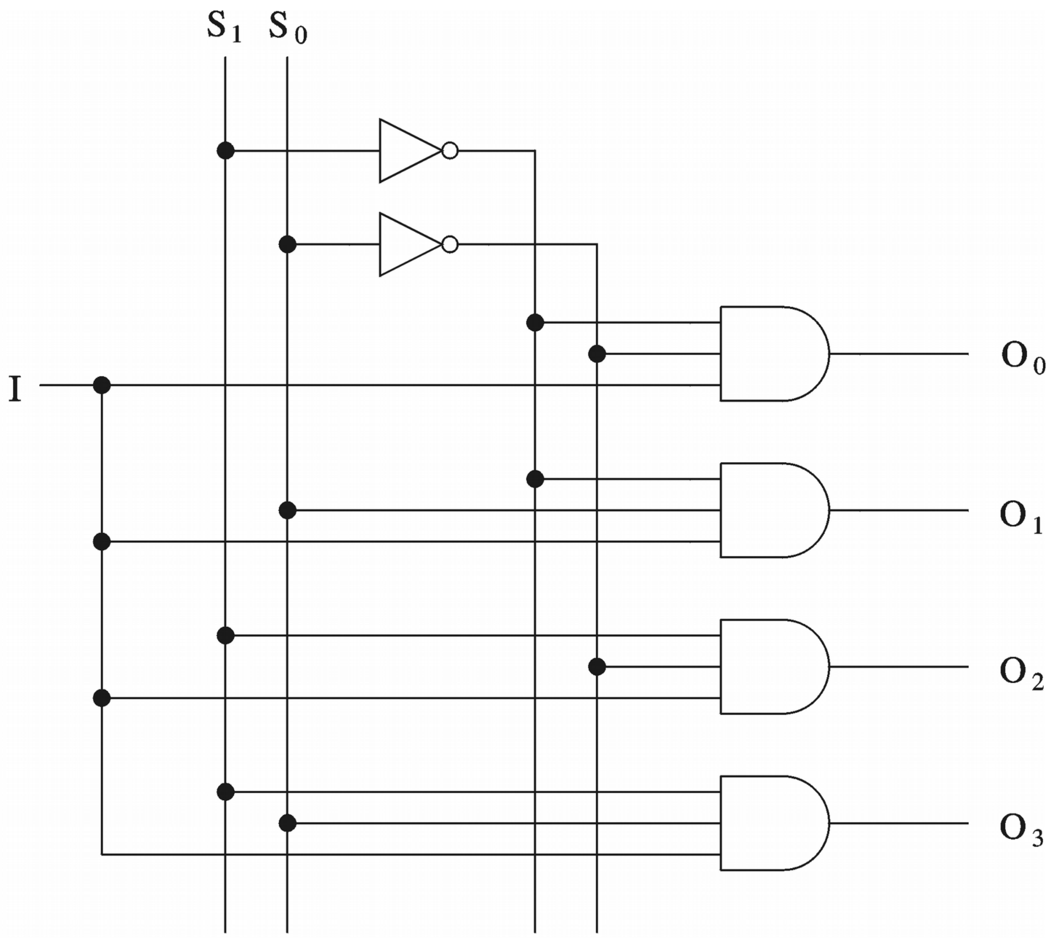
A	B	F_1
0	0	0
0	1	C
1	0	C
1	1	1



Demultiplexers

Demultiplexer (DeMUX)

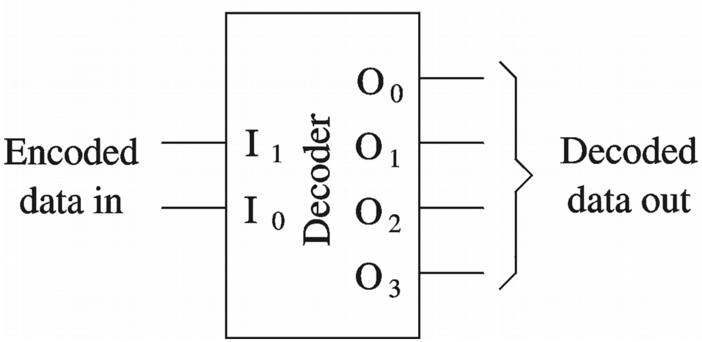


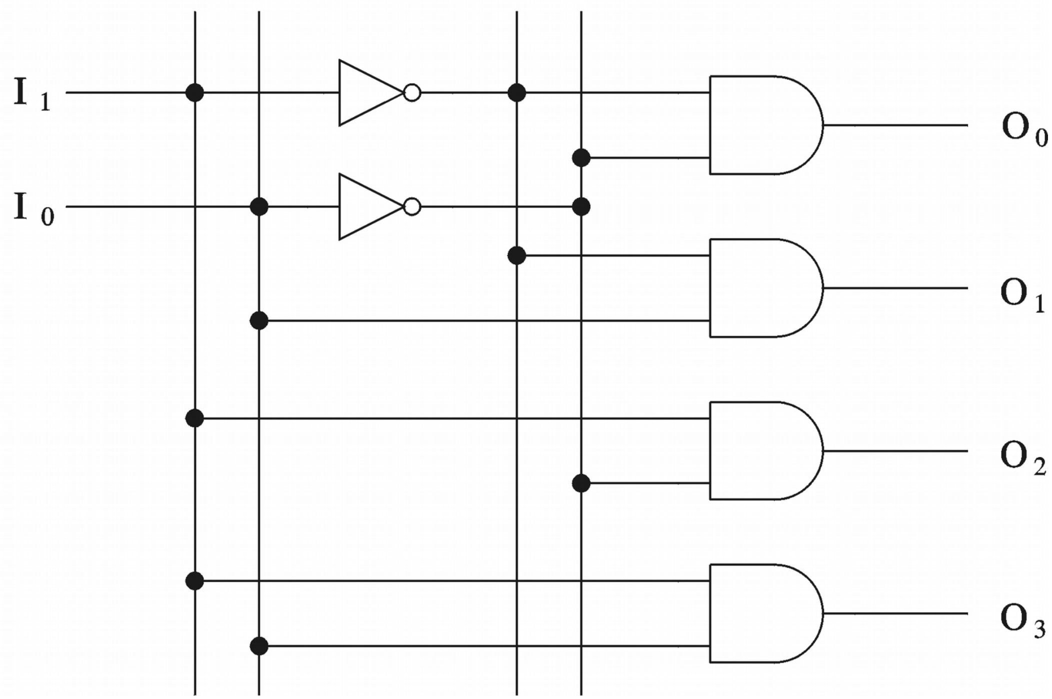


Decoders

Decoder selects one-out-of-N inputs. Decoders are simply a collection of logic gates which are arranged in a specific way so as to breakdown any combination of inputs to a set of terms that are all set to '0' apart from one term.

I_1	I_0	O_3	O_2	O_1	O_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



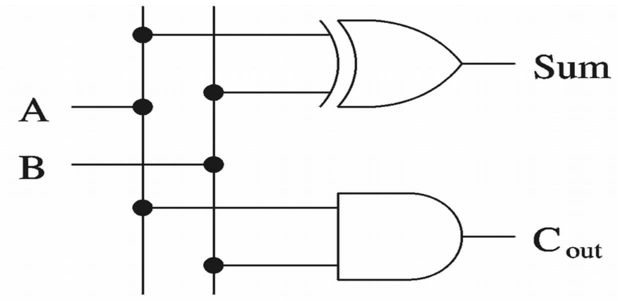


Adders

- Half-adder
  - Adds two bits
    - Produces a *sum* and *carry*
- Full-adder
  - Adds three 1-bit values
    - Like half-adder, produces a *sum* and *carry*
  - Allows building N-bit adders
    - Simple technique
      - Connect C_{out} of one adder to C_{in} of the next
    - These are called *ripple-carry adders*

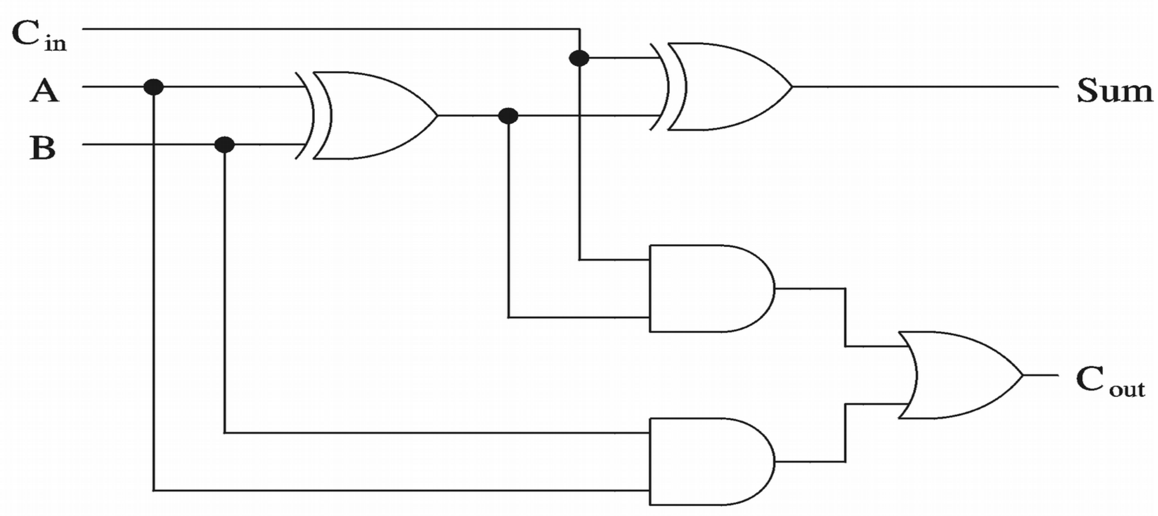
Adders (cont.)

A	B	Sum	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



(a) Half-adder truth table and implementation

A	B	C_{in}	Sum	C_{out}
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



(b) Full-adder truth table and implementation