College of Computer and Information Sciences
Department of Computer Science

CSC 220: Computer Organization

Unit 2

Digital Logic Gates

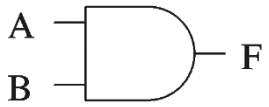
Introduction to Digital Logic Basics

- 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

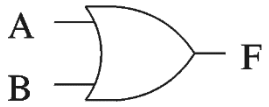
- Simple gates
  - AND
  - OR
  - NOT
- Functionality can be expressed by a truth table
  - A truth table lists output for each possible input combination
- Precedence
  - NOT > AND > OR
  - $F = A\,\overline{B} + \overline{A}\,B$

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



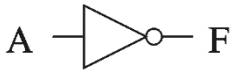
AND gate

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

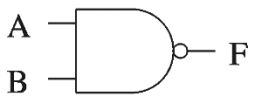
A	F
0	1
1	0

Logic symbol

Truth table

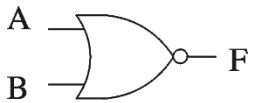
Basic Concepts (cont.)

- 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.)

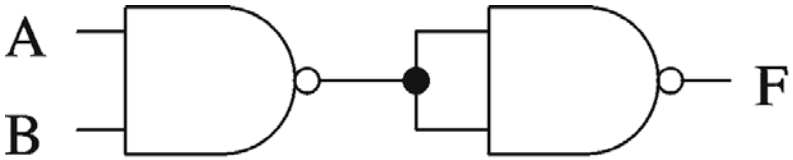
- Complete sets
  - A set of gates is complete
    - If we can implement any logical function using only the type of gates in the set
      - You can uses as many gates as you want
  - Some example complete sets
    - {AND, OR, NOT}
    - {AND, NOT}
    - {OR, NOT}
    - {NAND}
    - {NOR}

Not a minimal complete set

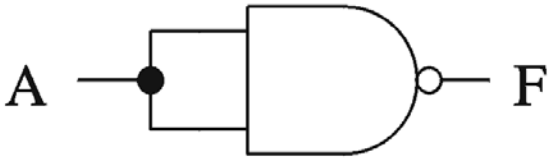
  - Minimal complete set
    - A complete set with no redundant elements.

Basic Concepts (cont.)

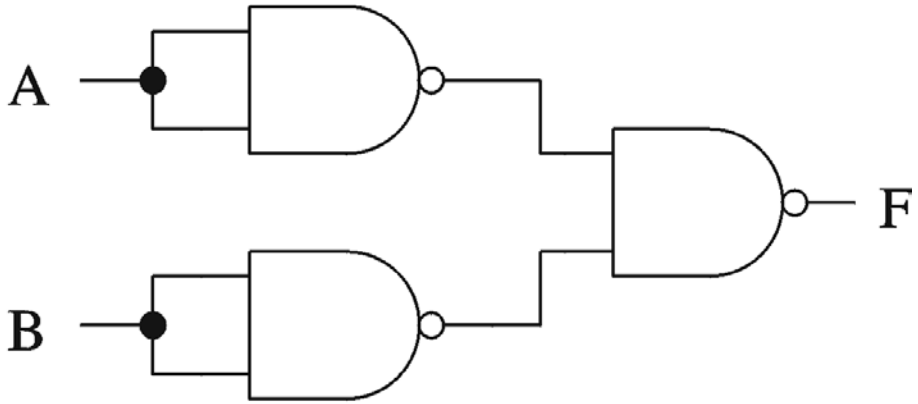
- Proving NAND gate is universal



AND gate



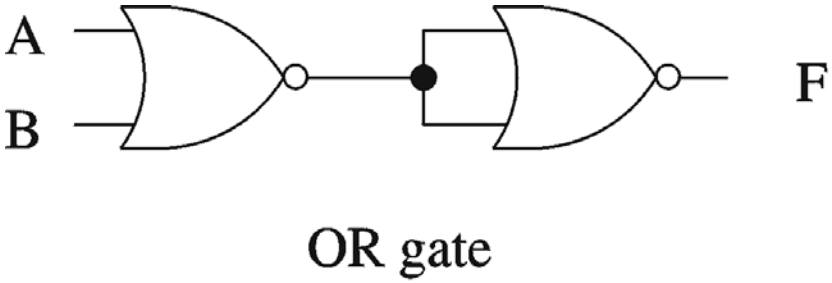
NOT gate



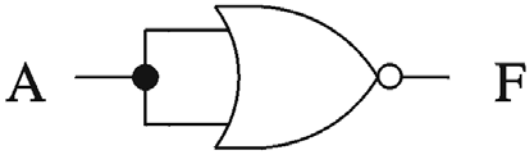
OR gate

Basic Concepts (cont.)

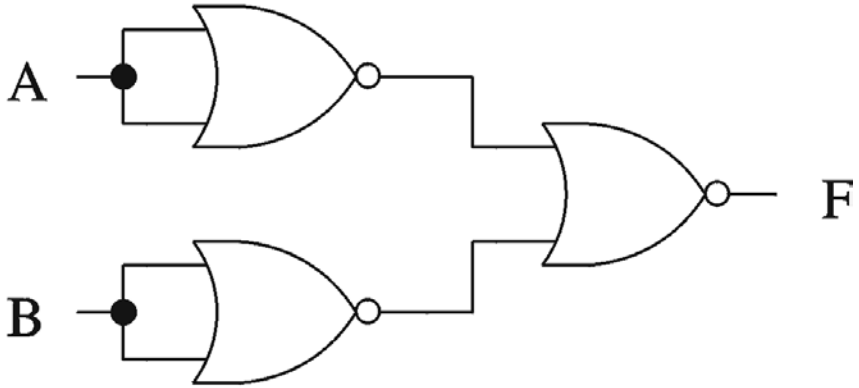
- Proving NOR gate is universal



OR gate

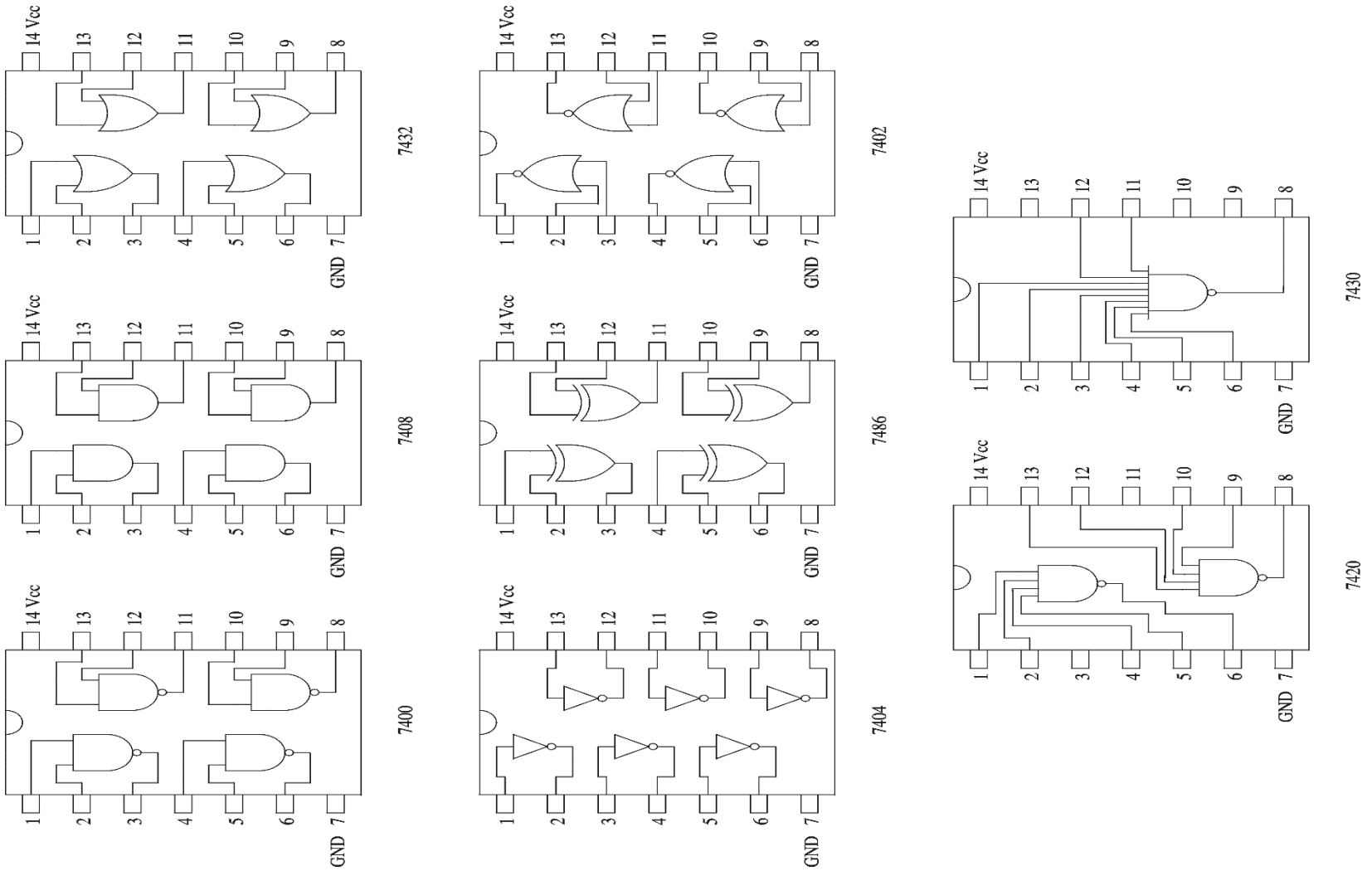


NOT gate



AND gate

Logic Chips (cont.)

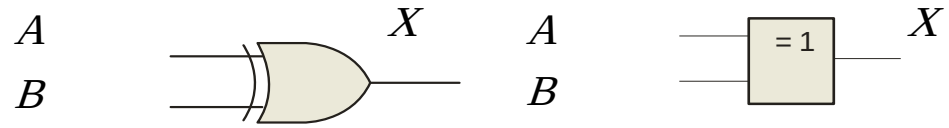


Logic Chips (cont.)

- 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

Applications of XOR and XNOR Gates

The XOR Gate

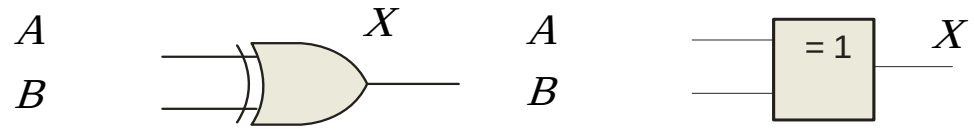


The **XOR gate** produces a HIGH output only when the inputs are at opposite logic levels. The truth table is

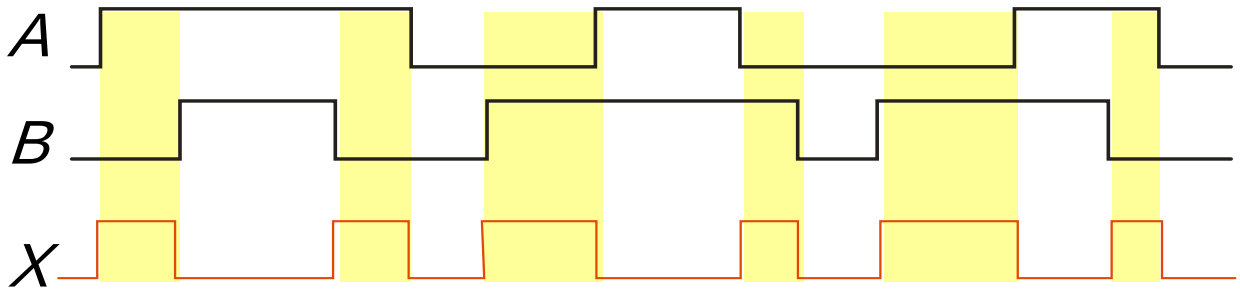
Inputs		Output
A	*B*	*X*
0	0	0
0	1	1
1	0	1
1	1	0

The **XOR** operation is written as $X = A\bar{B} + \bar{A}B$. Alternatively, it can be written with a circled plus sign between the variables as $X = A \oplus B$.

The XOR Gate

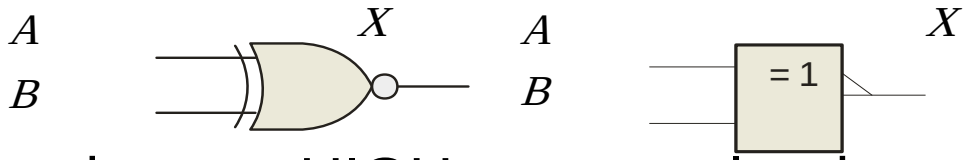


Example waveforms:



Notice that the XOR gate will produce a HIGH only when exactly one input is HIGH.

The XNOR Gate

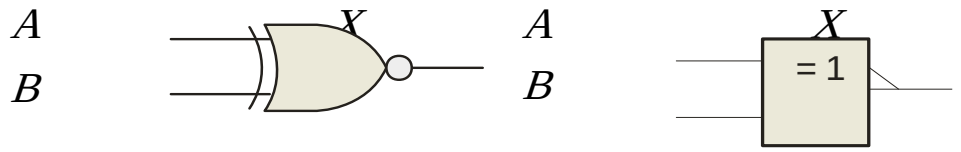


The **XNOR gate** produces a HIGH output only when the inputs are at the same logic level. The truth table is

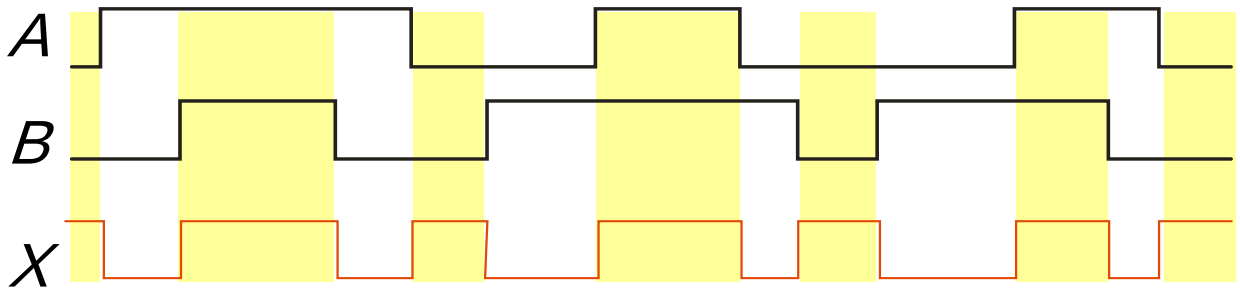
Inputs		Output
A	*B*	*X*
0	0	1
0	1	0
1	0	0
1	1	1

The **XNOR** operation can be shown as $X = AB + \overline{A}\,\overline{B}$.

The XNOR Gate



Example waveforms:



Notice that the XNOR gate will produce a HIGH when both inputs are the same. This makes it useful for comparison functions.

Applications of XOR and XNOR Gates

- Three common applications:
  - Comparators
  - Parity generation and checking

Comparator

- A **comparator** compares two string of bits to see whether they are equal to each other:
  - Example: if string A = 0101 and string B = 0100, then A≠ B.
- Next slide shows how to build a 4-bit comparator from XNOR gates.

Comparator Circuit

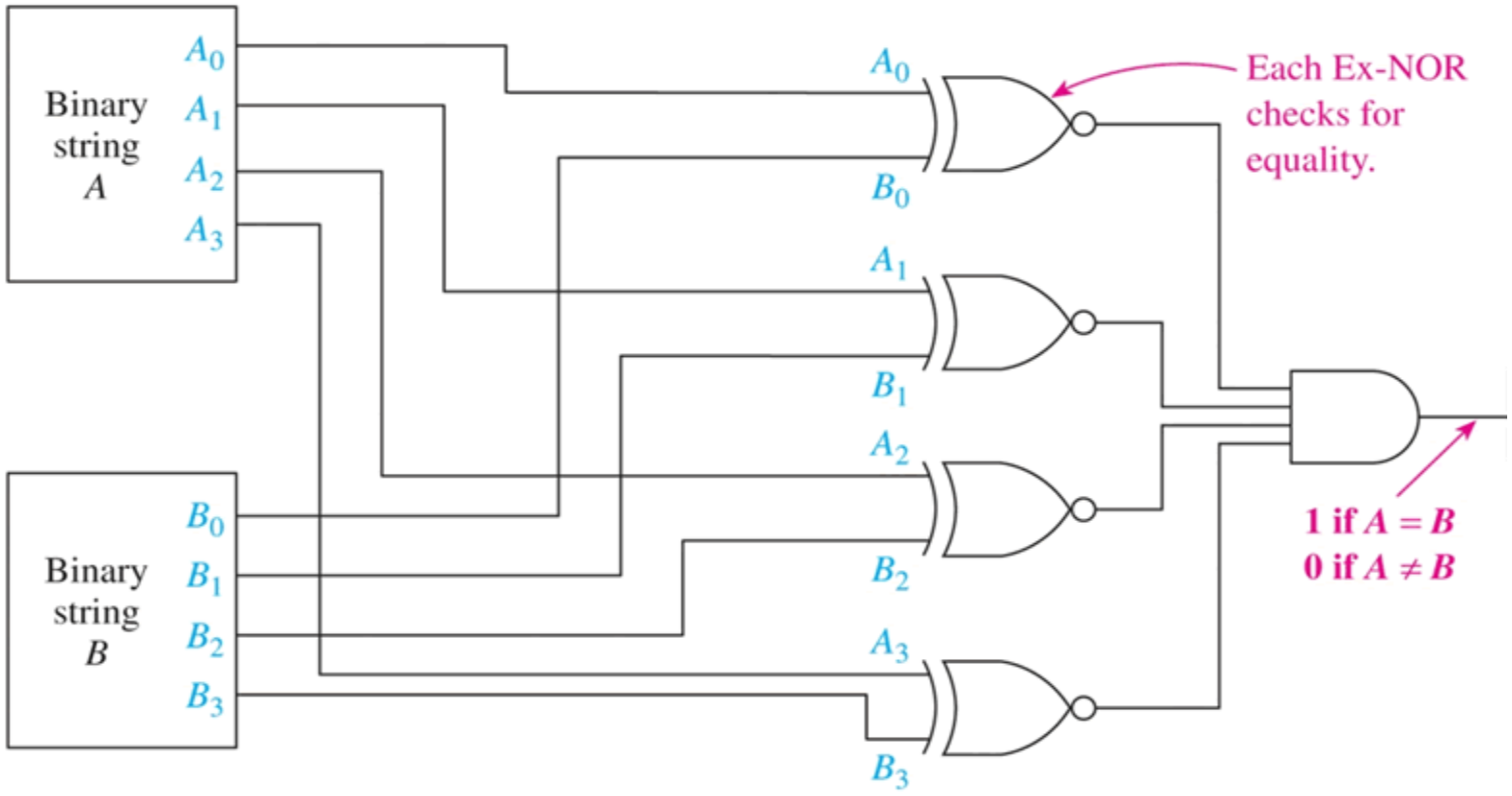


Figure 6–14 Binary comparator system.

Parity Checking

Parity checking is a method of error detection for simple transmission errors involving one bit (or an odd number of bits). A parity bit is an “extra” bit attached to a group of bits to force the number of 1’s to be either even (even parity) or odd (odd parity).

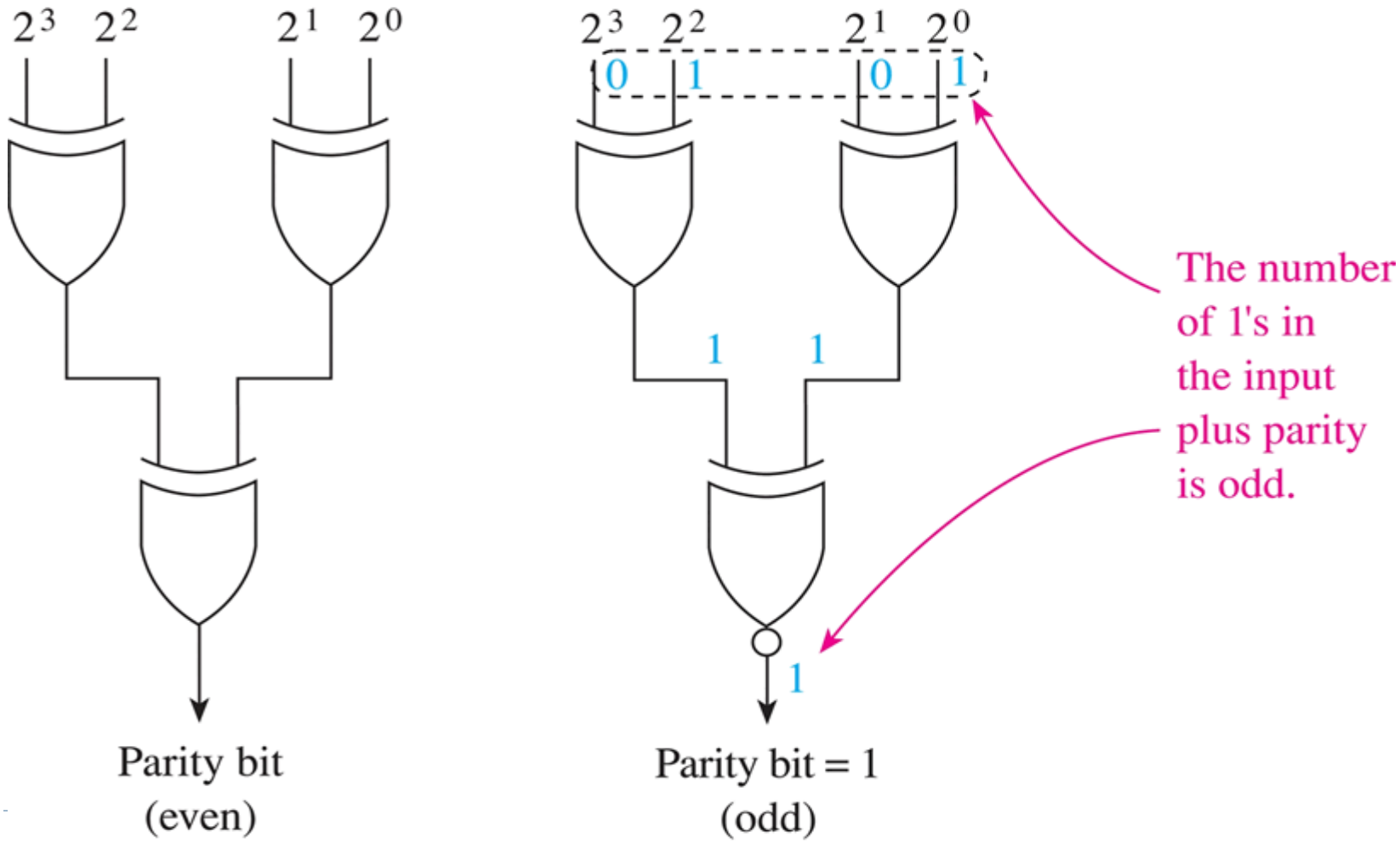
Example The ASCII character for “a” is 1100001 and for “A” is 1000001. What is the correct bit to append to make both of these have odd parity?

Solution The ASCII “a” has an odd number of bits that are equal to 1; therefore the parity bit is 0. The ASCII “A” has an even number of bits that are equal to 1; therefore the parity bit is 1.

Parity Generators

- To implement parity checking, we need circuitry on the sending end that generates the parity bit for each group of bits being sent. This circuitry is called a **parity generator**.
- Next slide shows how to build 4-bit even or odd parity generators.

Parity Generators (Book's Fig. 6-9)

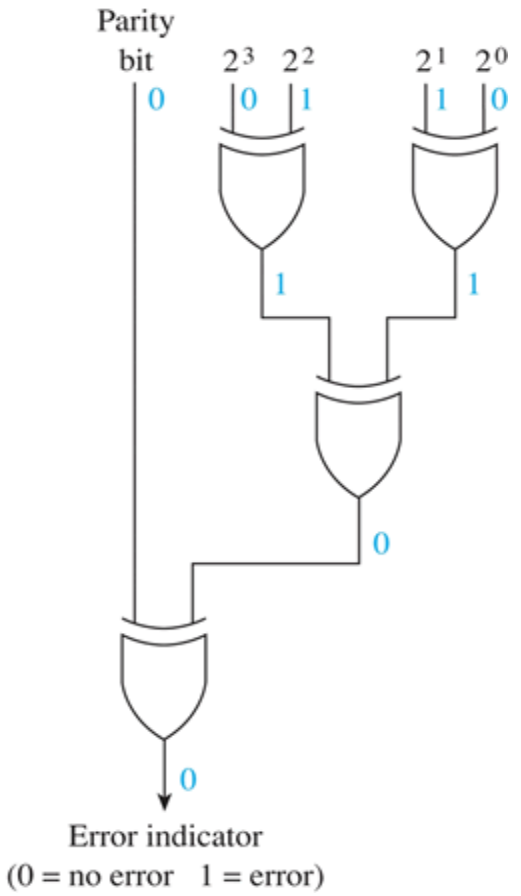


Parity Checkers

- On the receiving end, we need circuitry that checks the data bits and parity bit as they're received to decide whether an error has occurred during transmission. This circuitry is called a **parity checker**.
- Next slide shows how to build a 4-bit-plus-parity even parity checker.

Parity Checker

Figure 6.11 Five-bit even-parity checker.



Parity System

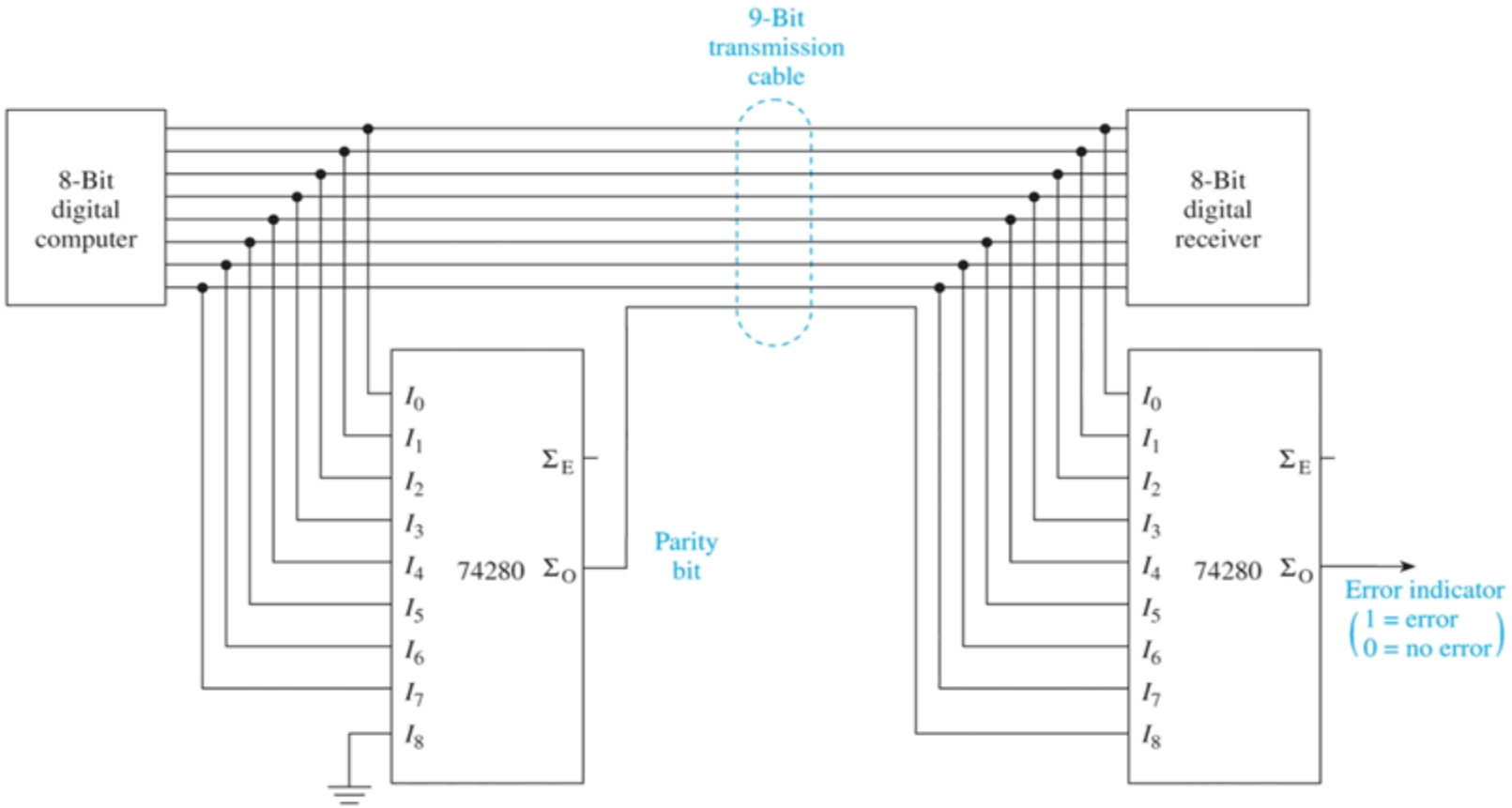


Figure 6–13 Complete 8-bit even-parity error-detection system.