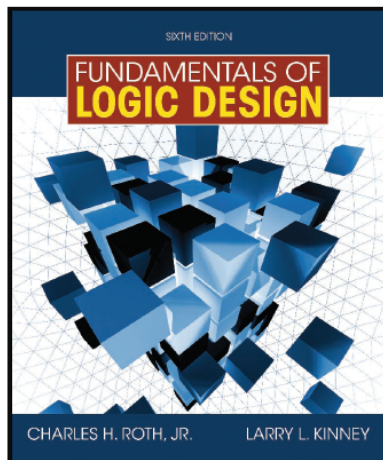


# SLIDES FOR CHAPTER 9

## MULTIPLEXERS, DECODERS, AND PROGRAMMABLE LOGIC DEVICES

*This chapter in the book includes:*

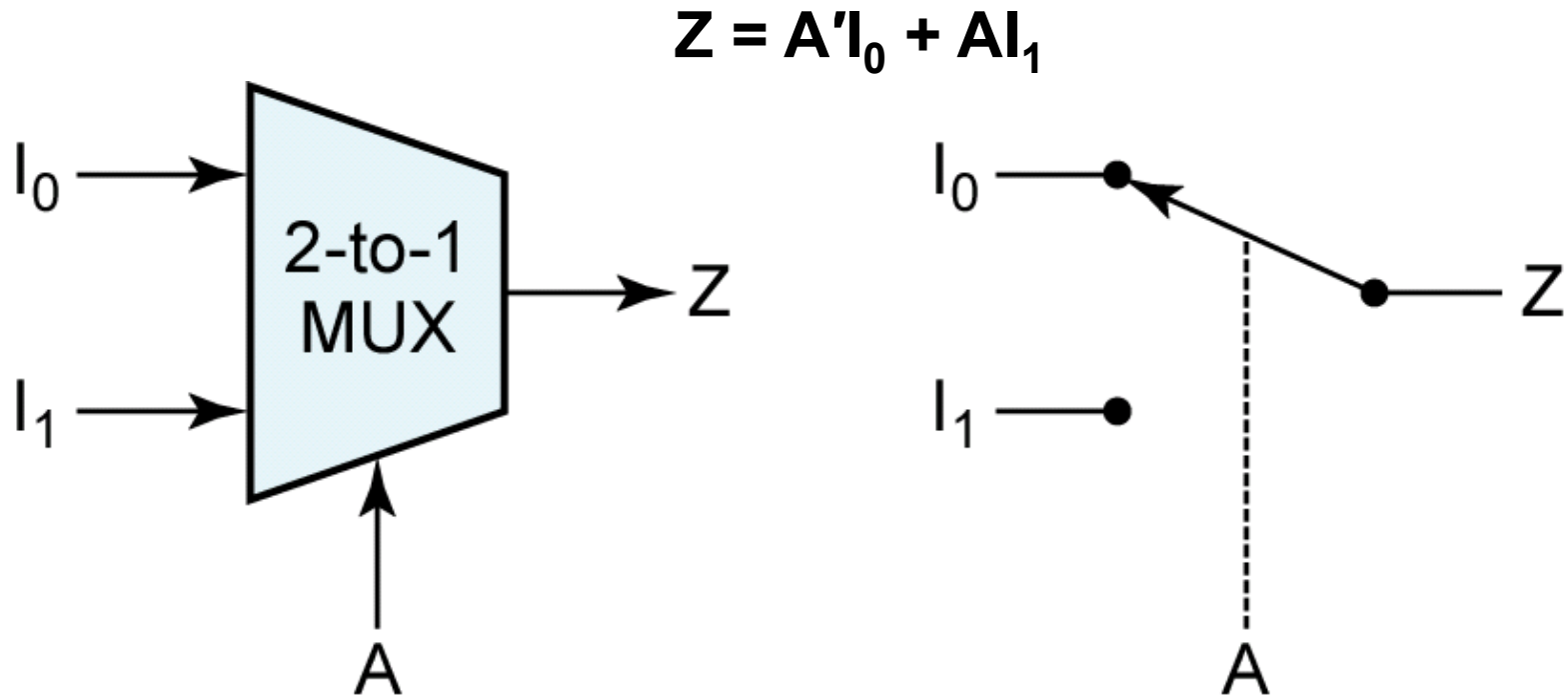


- Objectives
- Study Guide
- 9.1 Introduction
- 9.2 Multiplexers
- 9.3 Three-State Buffers
- 9.4 Decoders and Encoders
- 9.5 Read-Only Memories
- 9.6 Programmable Logic Devices
- 9.7 Complex Programmable Logic Devices
- 9.8 Field Programmable Gate Arrays
- Problems

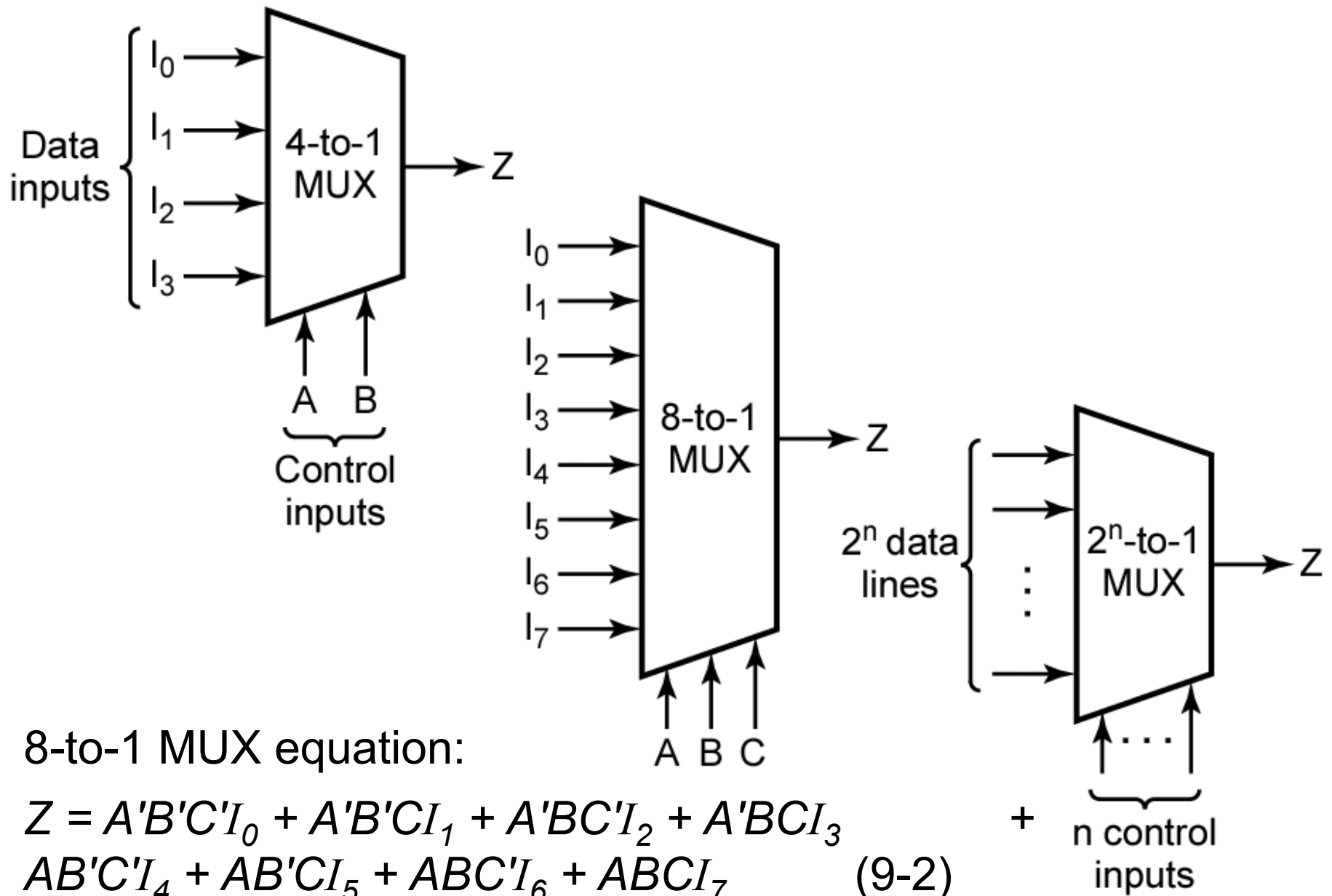
Click the mouse to move to the next page.  
Use the ESC key to exit this chapter.

# Multiplexers

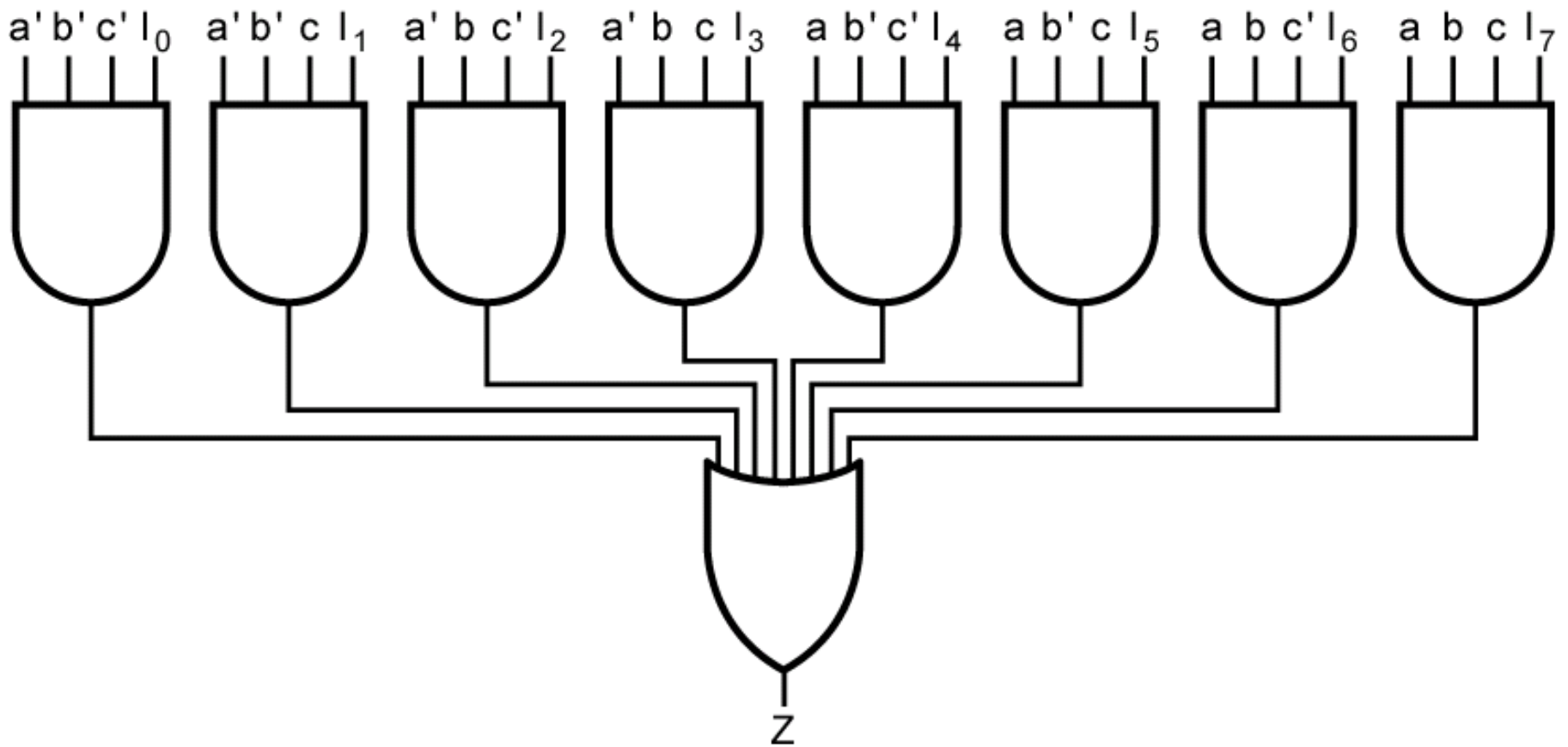
A multiplexer has a group of data inputs and a group of control inputs used to select one of the data inputs and connect it to the output terminal.



**Figure 9-1: 2-to-1 Multiplexer and Switch Analog**

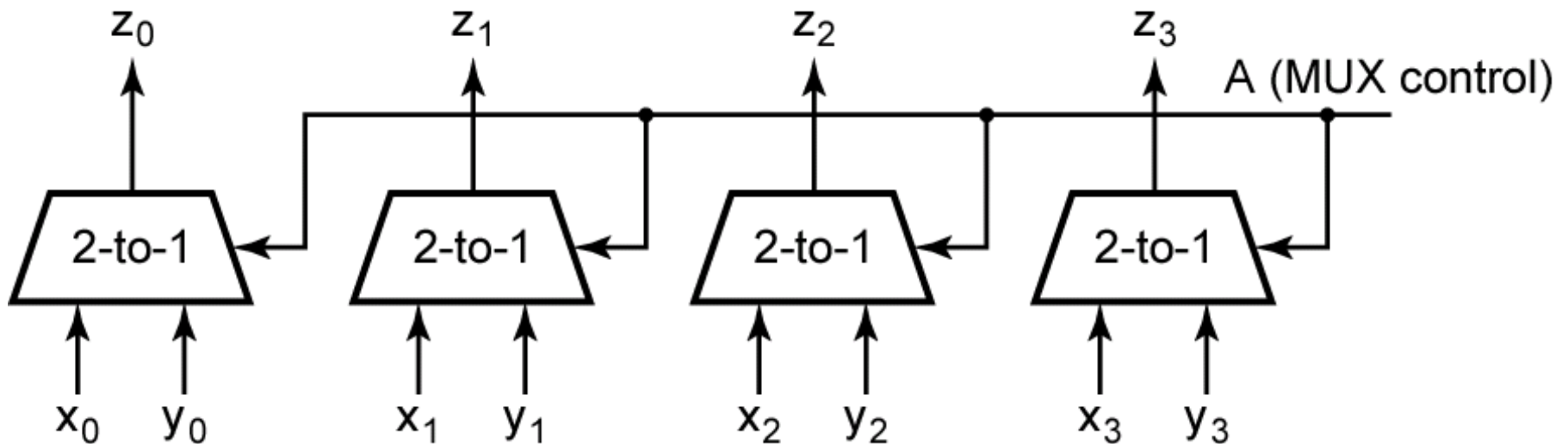


**Figure 9-2: Multiplexers**

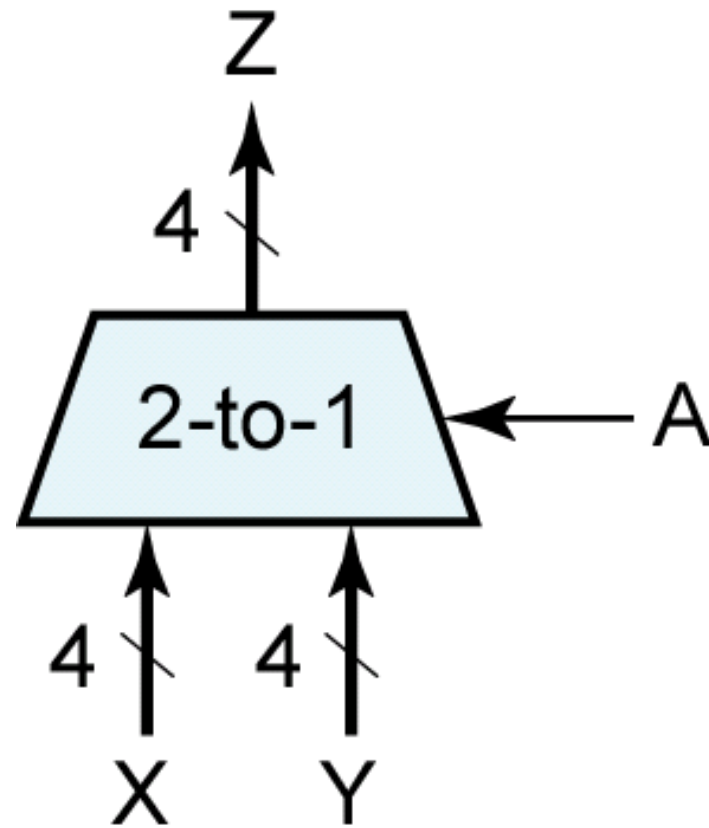


**Figure 9-3: Logic Diagram for 8-to-1 MUX**

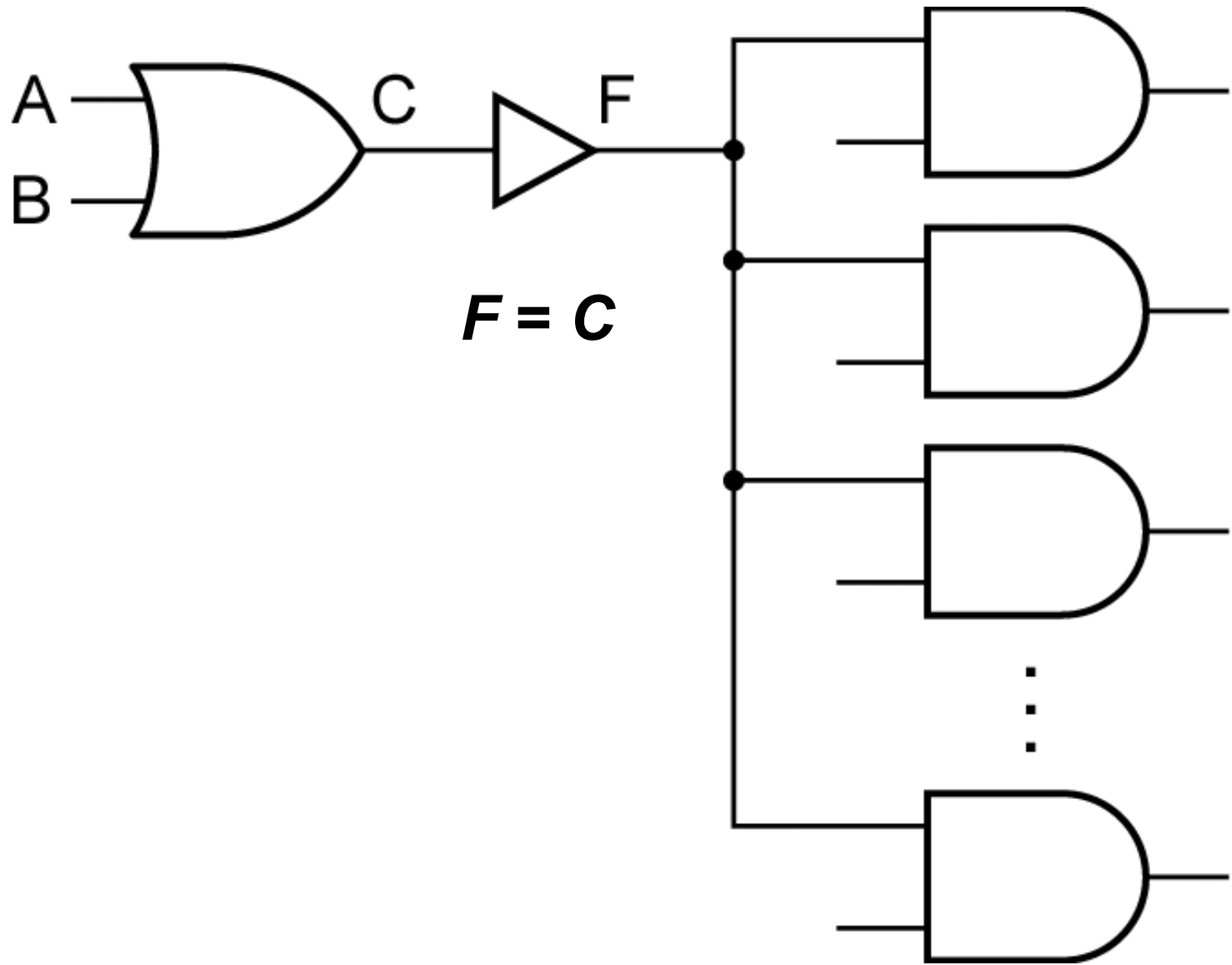
Control Variable  $A$  selects one of two 4-bit data words.



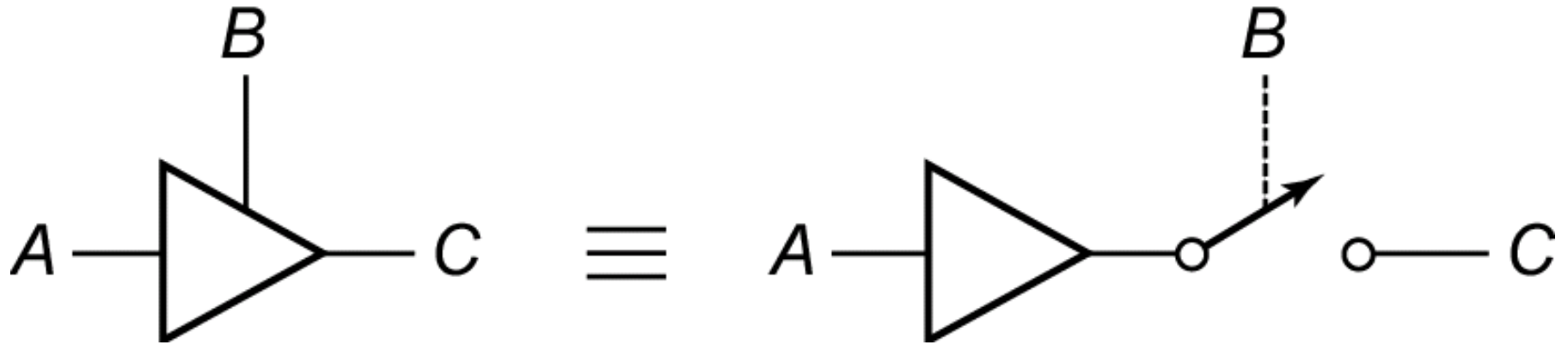
**Figure 9-4: Quad Multiplexer Used to Select Data**



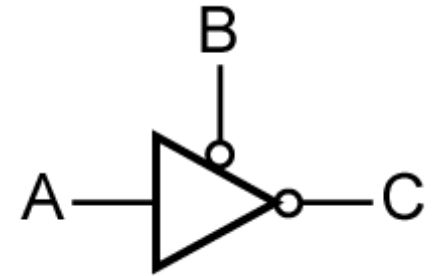
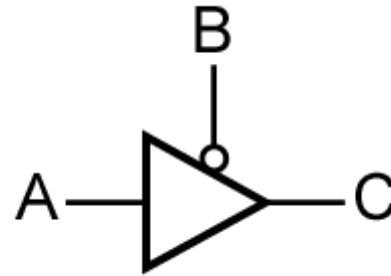
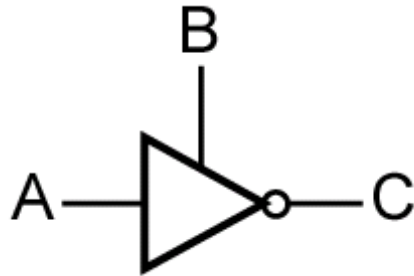
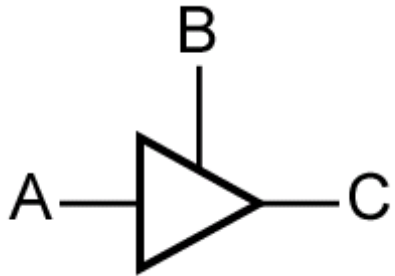
**Figure 9-5: Quad Multiplexer with Bus Inputs and Output**



**Figure 9-6: Gate Circuit with Added Buffer**



**Figure 9-7: Three-State Buffer**



B	A	C
0	0	Z
0	1	Z
1	0	0
1	1	1

(a)

B	A	C
0	0	Z
0	1	Z
1	0	1
1	1	0

(b)

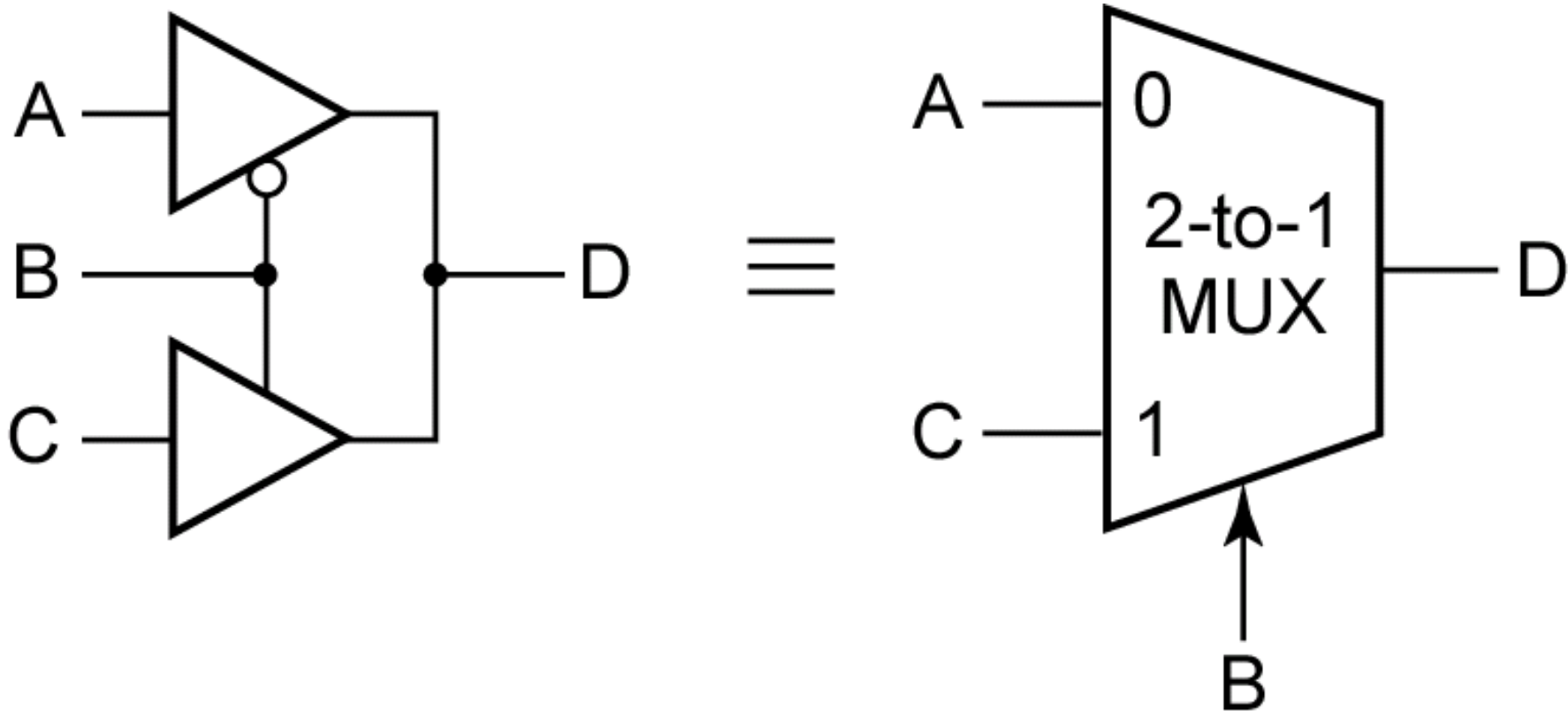
B	A	C
0	0	0
0	1	1
1	0	Z
1	1	Z

(c)

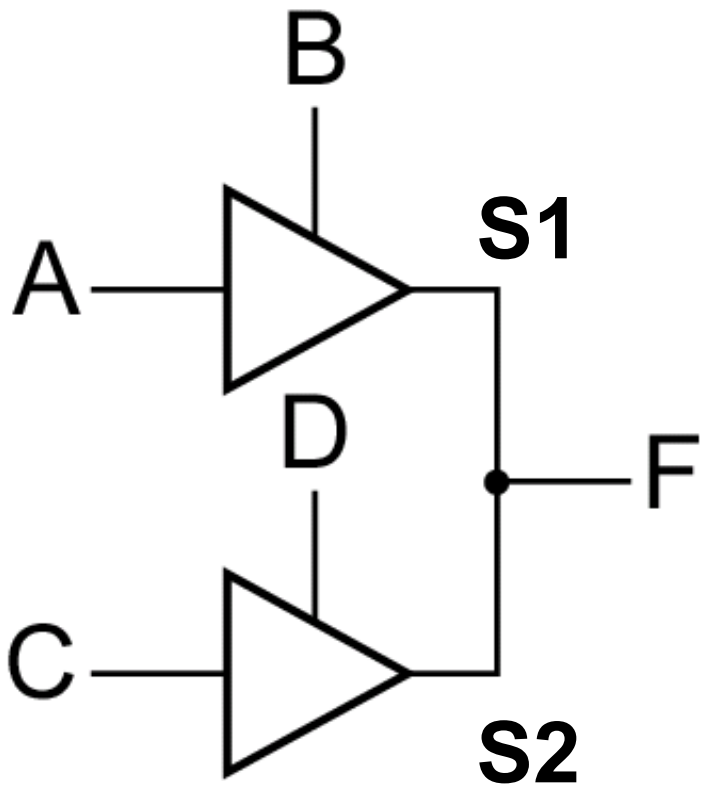
B	A	C
0	0	1
0	1	0
1	0	Z
1	1	Z

(d)

**Figure 9-8: Four Kinds of Three-State Buffers**



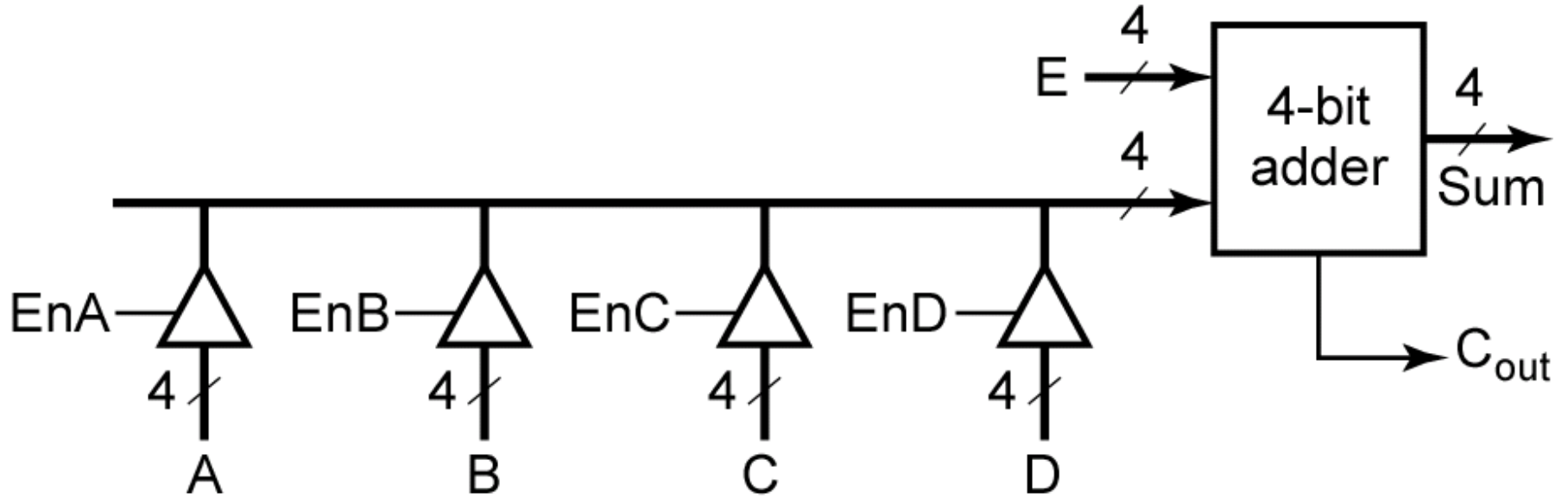
**Figure 9-9: Data Selection Using Three-State Buffers**



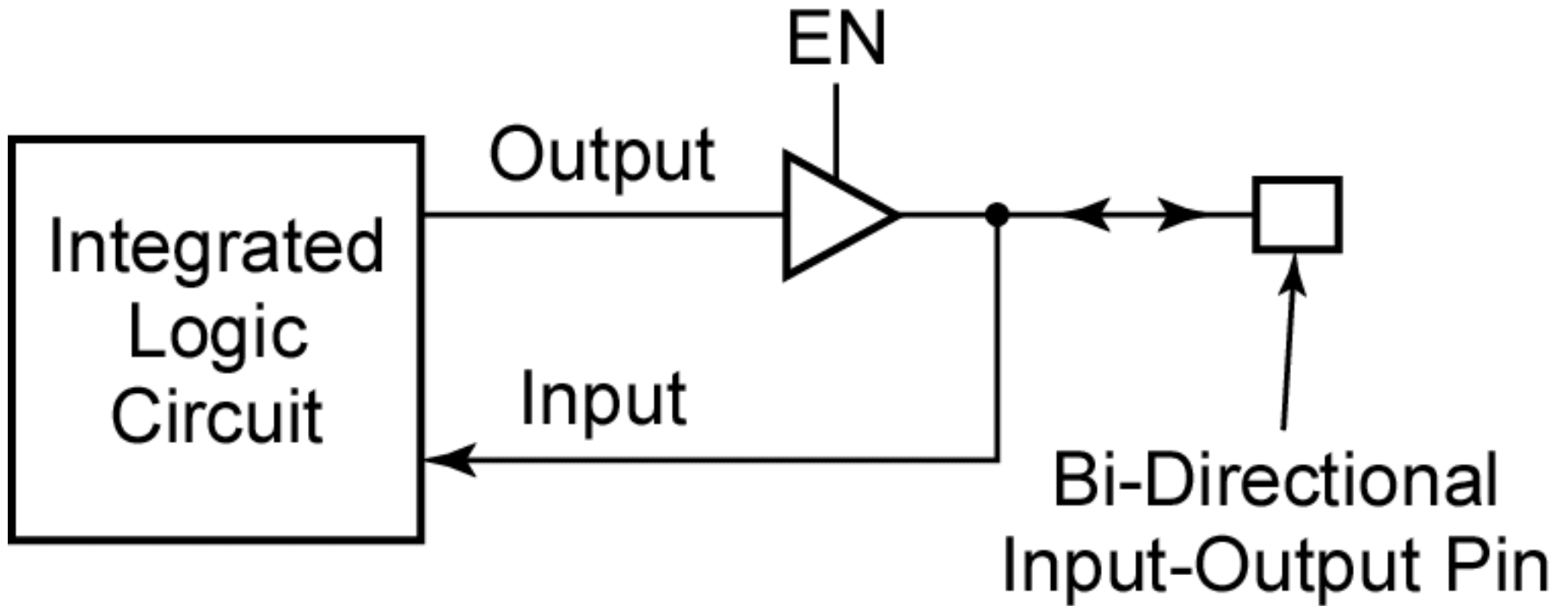
F is determined from the following table:

S1	S2			Z
	X	0	1	
X	X	X	X	X
0	X	0	X	0
1	X	X	1	1
Z	X	0	1	Z

**Figure 9-10: Circuit with Two Three-State Buffers**



**Figure 9-11: 4-Bit Adder with Four Sources for One Operand**

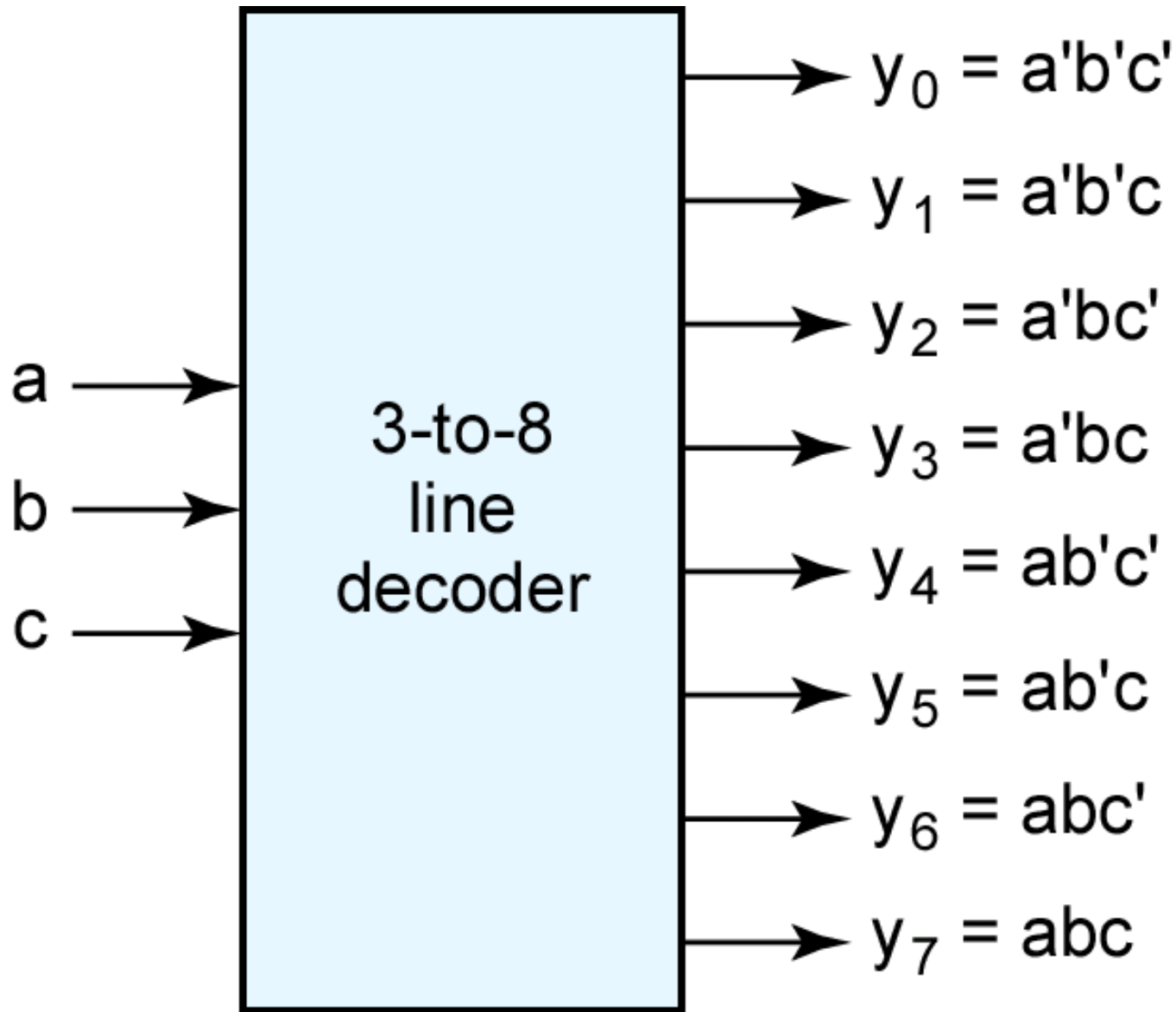


**Figure 9-12: Integrated Circuit with Bi-Directional Input/Output Pin**

# Decoders

The decoder is another commonly used type of integrated circuit. The decoder generates all of the minterms of the three input variables. Exactly one of the output lines will be 1 for each combination of the values of the input variables.

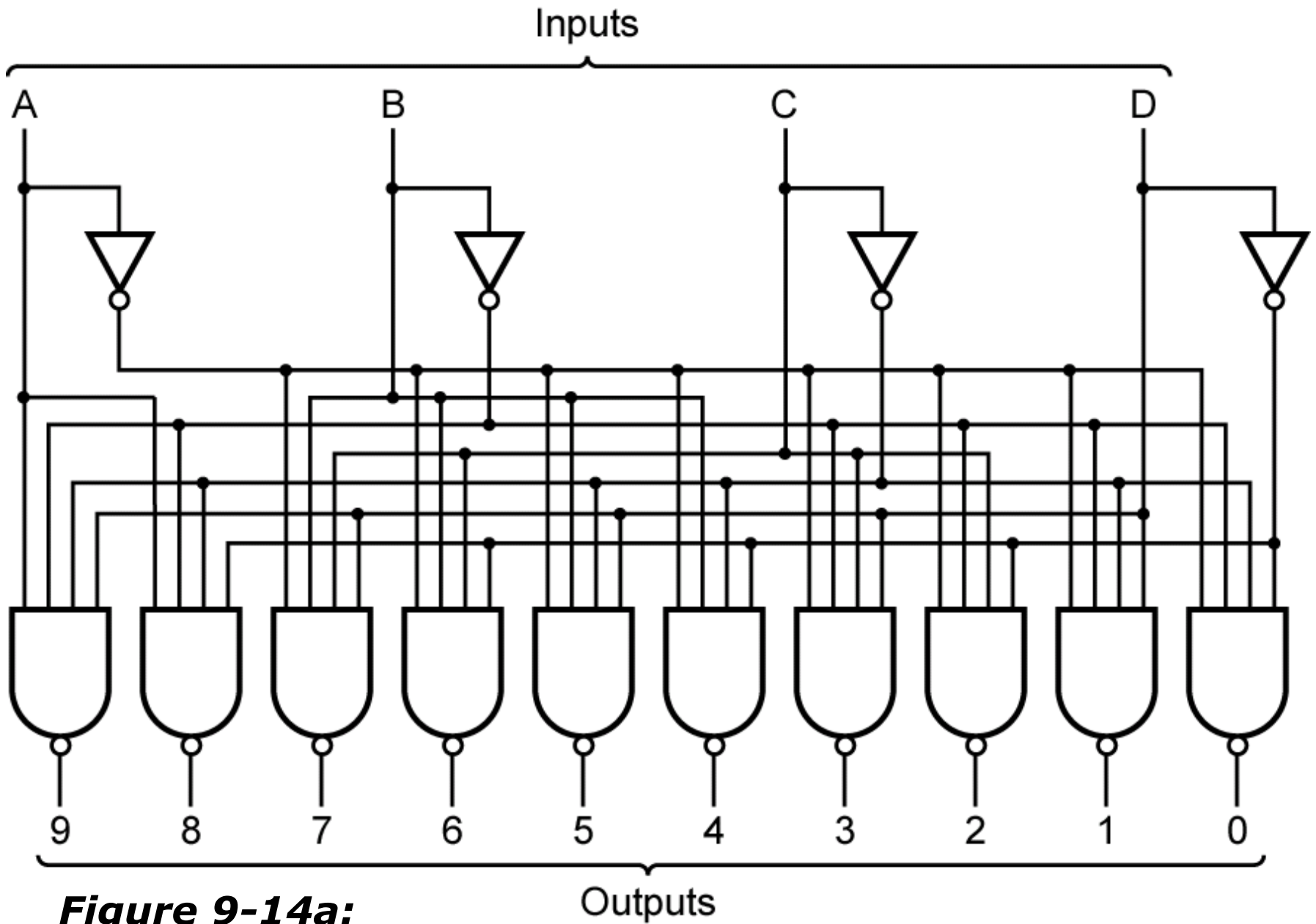
**Section 9.4 (p. 256)**



**Figure 9-13: 3-to-8 Line Decoder**

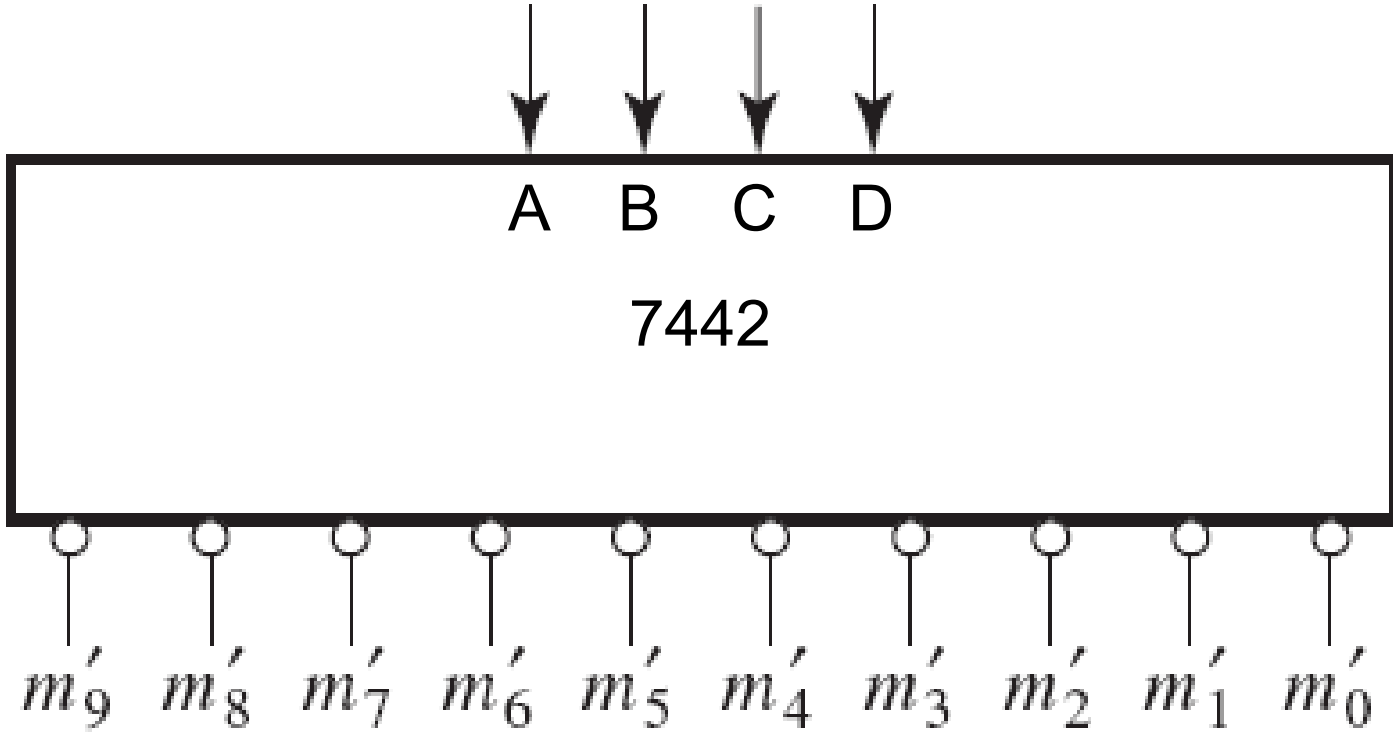
$a$	$b$	$c$	$y_0$	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$	$y_6$	$y_7$
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

**Figure 9-13: 3-to-8 Line Decoder**



**Figure 9-14a:**  
**A 4-to-10 Line**  
**Decoder**

Outputs  
 (a) Logic diagram



**Figure 9-14b:**  
**A 4-to-10 Line Decoder**

(b) Block diagram

BCD Input				Decimal Output									
<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0
1	0	1	0	1	1	1	1	1	1	1	1	1	1

**Figure 9-14c:**  
**A 4-to-10 Line Decoder**

(c) Truth Table

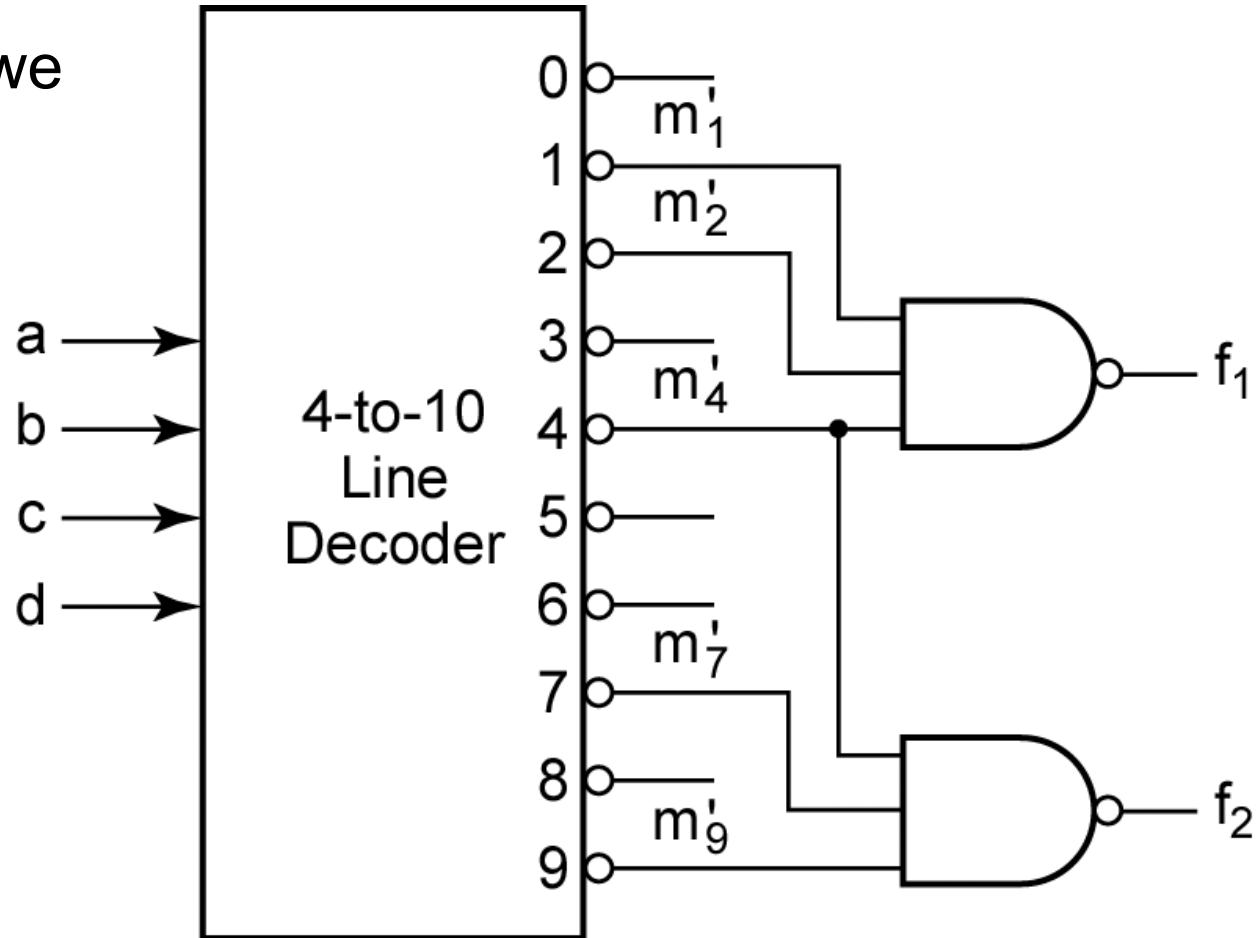
$$f_1(a, b, c, d) = m_1 + m_2 + m_4$$

$$f_2(a, b, c, d) = m_4 + m_7 + m_9$$

Rewriting  $f_1$  and  $f_2$ , we have

$$f_1 = (m_1' m_2' m_4)'$$

$$f_2 = (m_4' m_7' m_9)'$$



**Figure 9-15: Realization of a Multiple-Output Circuit Using a Decoder**

# Encoders

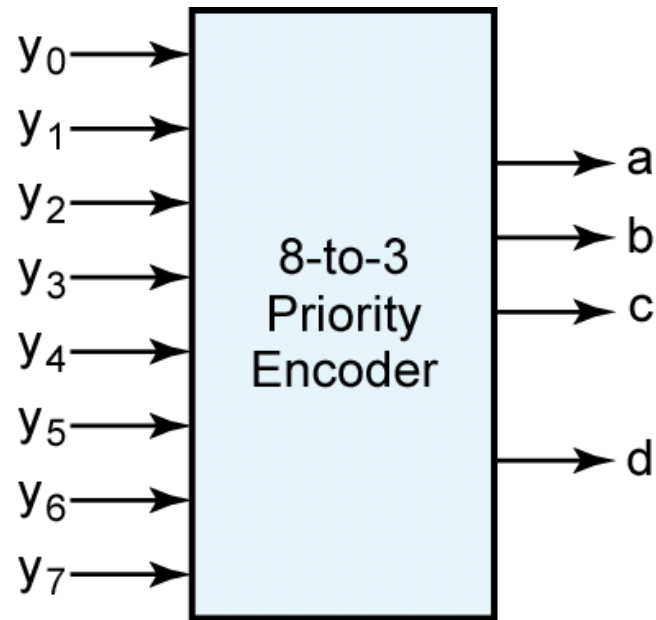
An encoder performs the inverse function of a decoder. If input  $y_i$  is 1 and the other inputs are 0, then  $abc$  outputs represent a binary number equal to  $i$ .

For example, if  $y_3 = 1$ , then  $abc = 011$ .

If more than one input is 1, the highest numbered input determines the output.

An extra output,  $d$ , is 1 if any input is 1, otherwise  $d$  is 0. This signal is needed to distinguish the case of all 0 inputs from the case where only  $y_0$  is 1.

## Section 9.4 (p. 258)



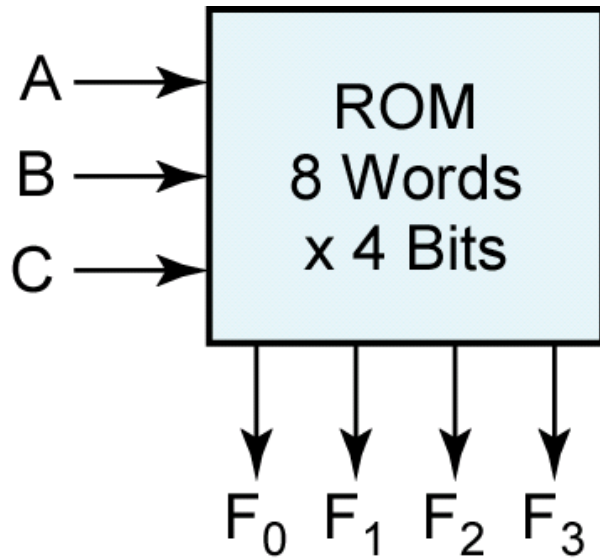
$y_0$	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$	$y_6$	$y_7$	$a$	$b$	$c$	$d$
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
X	1	0	0	0	0	0	0	0	0	1	1
X	X	1	0	0	0	0	0	0	1	0	1
X	X	X	1	0	0	0	0	0	1	1	1
X	X	X	X	1	0	0	0	1	0	0	1
X	X	X	X	X	1	0	0	1	0	1	1
X	X	X	X	X	X	1	0	1	1	0	1
X	X	X	X	X	X	X	1	1	1	1	1

**Figure 9-16: 8-to-3 Priority Coder**

# Read-Only Memories

A read-only memory (ROM) consists of an array of semiconductor devices that are interconnected to store an array of binary data. Once binary data is stored in the ROM, it can be read out whenever desired, but the data that is stored cannot be changed under normal operating conditions.

**Section 9.5 (p. 259)**



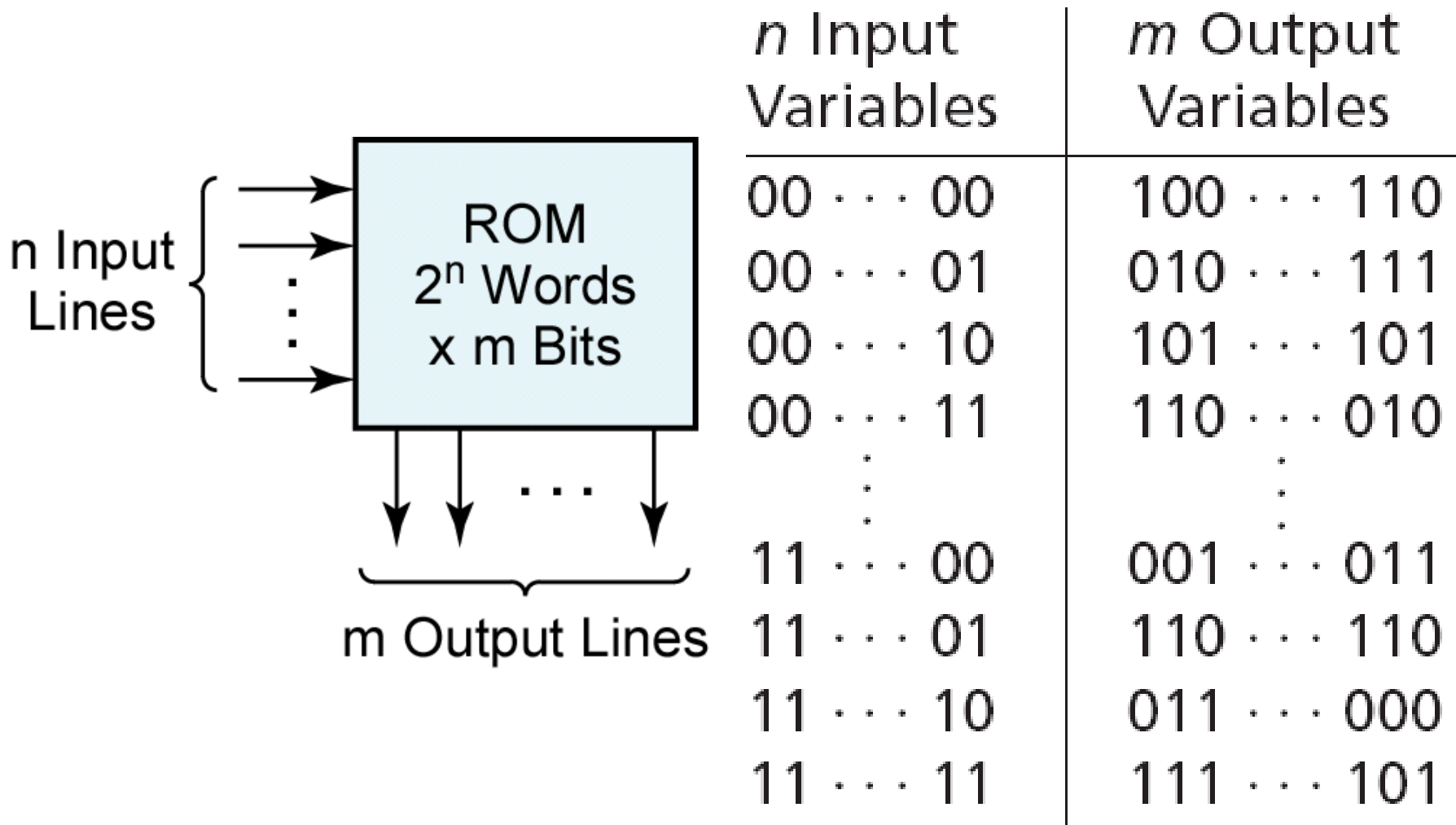
(a) Block diagram

<i>A</i>	<i>B</i>	<i>C</i>	<i>F</i> <sub>0</sub>	<i>F</i> <sub>1</sub>	<i>F</i> <sub>2</sub>	<i>F</i> <sub>3</sub>
0	0	0	1	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	1	1
0	1	1	0	1	0	1
1	0	0	1	1	0	0
1	0	1	0	0	0	1
1	1	0	1	1	1	1
1	1	1	0	1	0	1

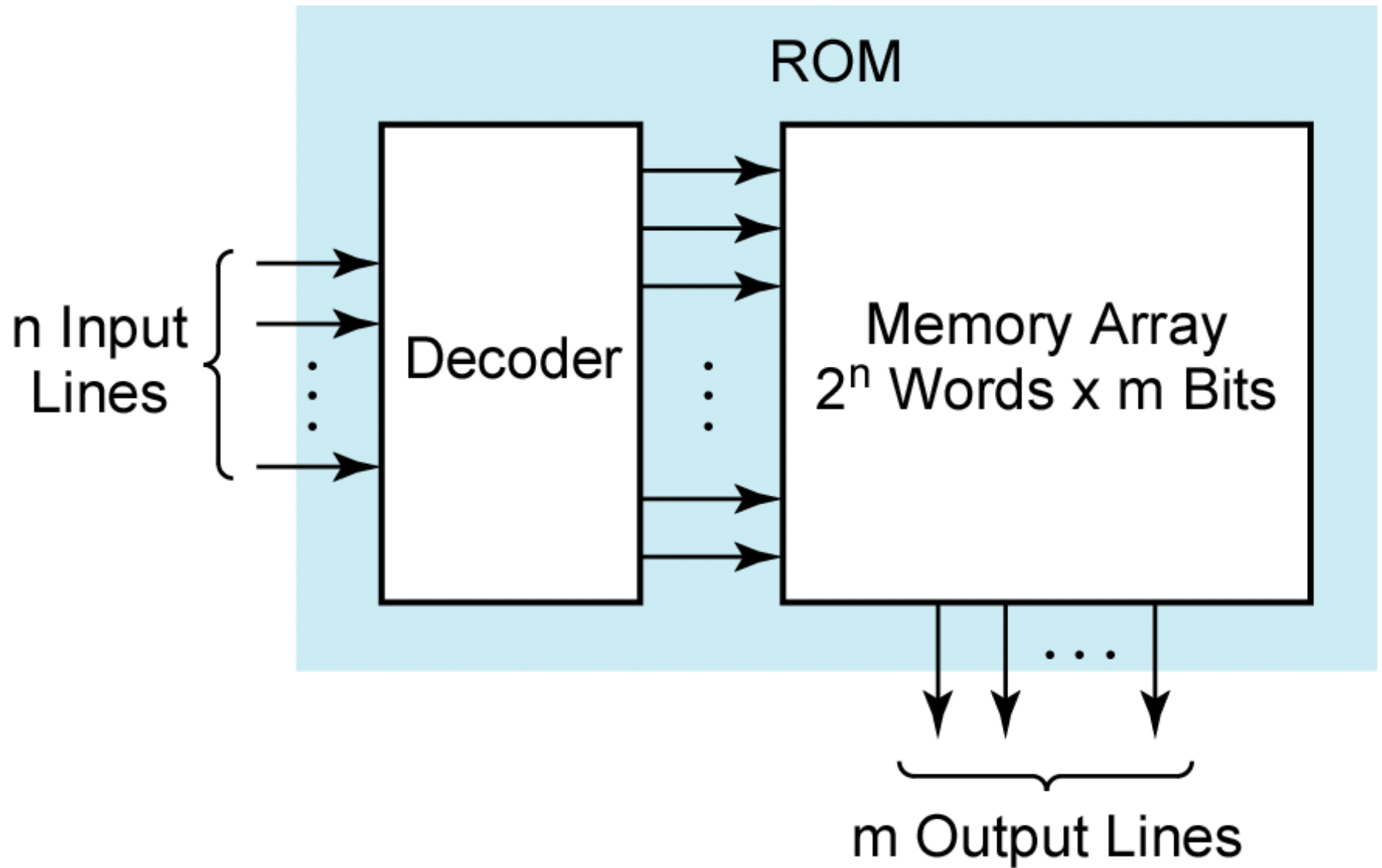
Typical Data  
Stored in  
ROM  
( $2^3$  words of  
4 bits each)

(b) Truth table for ROM

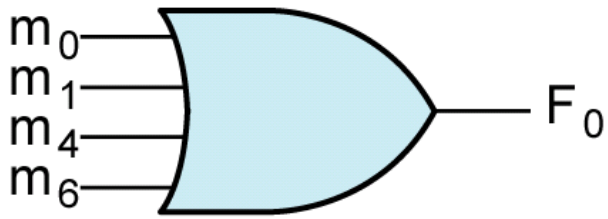
**Figure 9-17: An 8-Word x 4-Bit ROM**



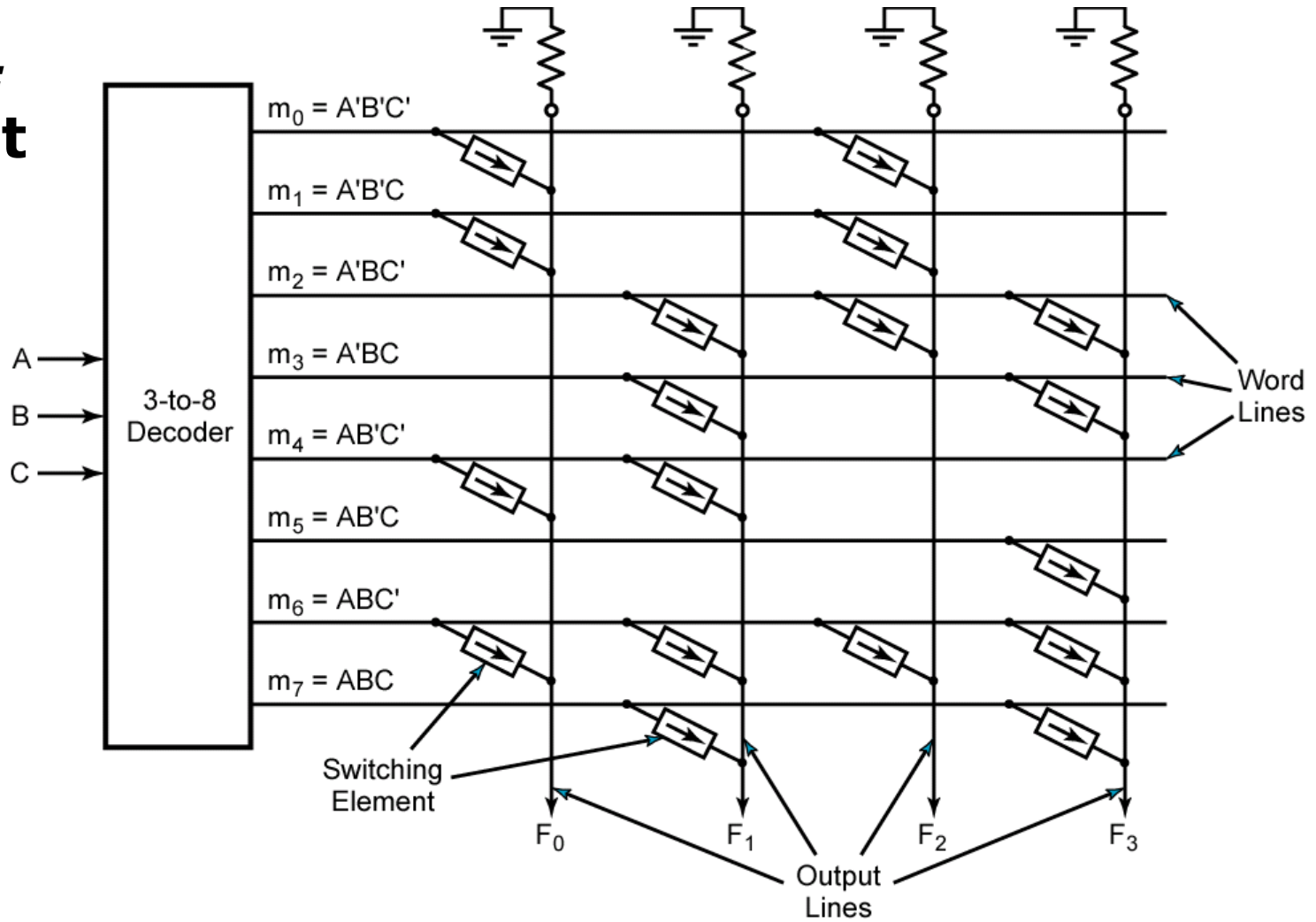
**Figure 9-18: Read-Only Memory with  $n$  Inputs and  $m$  Outputs**



**Figure 9-19: Basic ROM Structure**



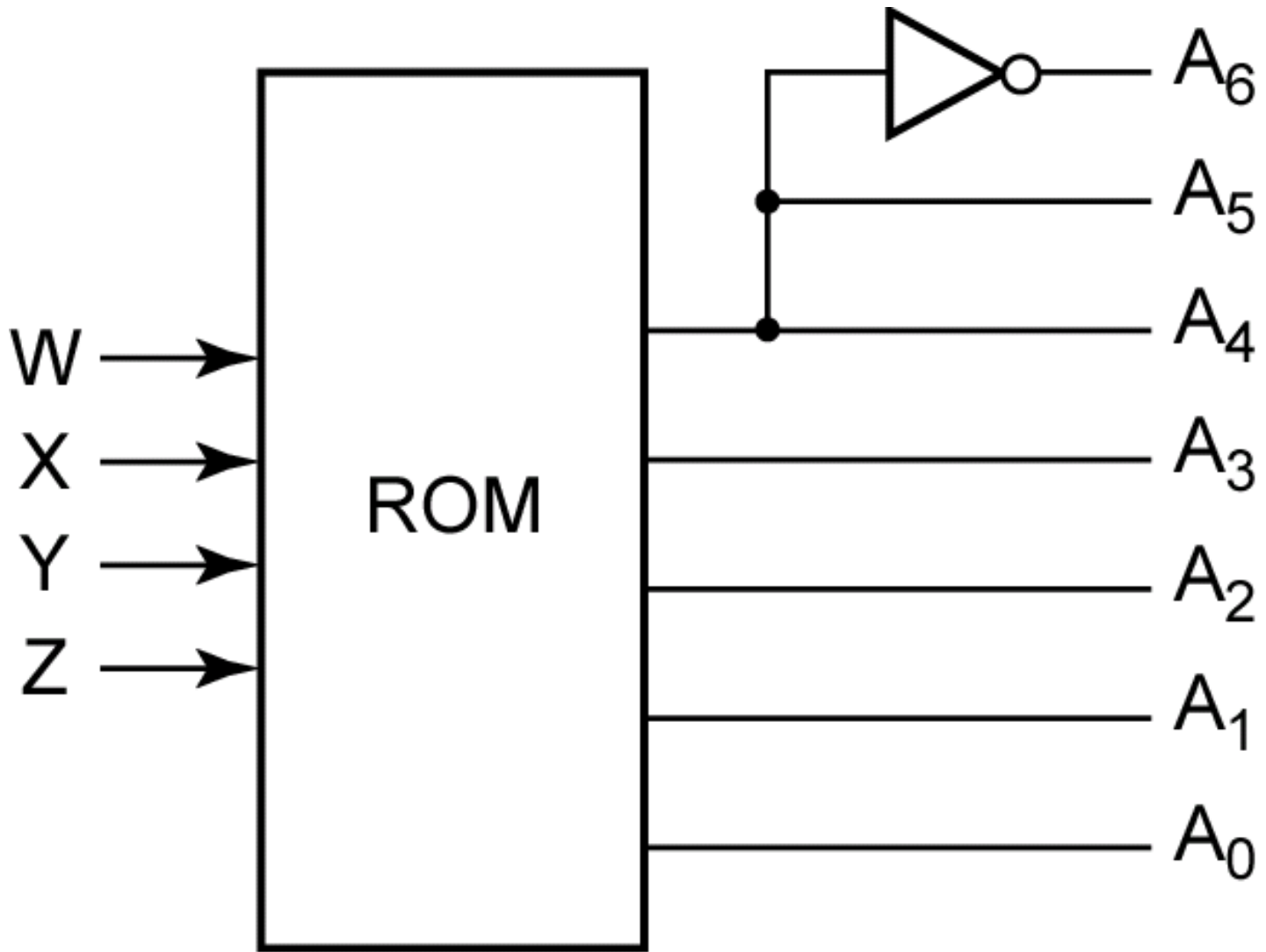
**Figure 9-21:**  
**Equivalent**  
**OR Gate**  
**for  $F_0$**



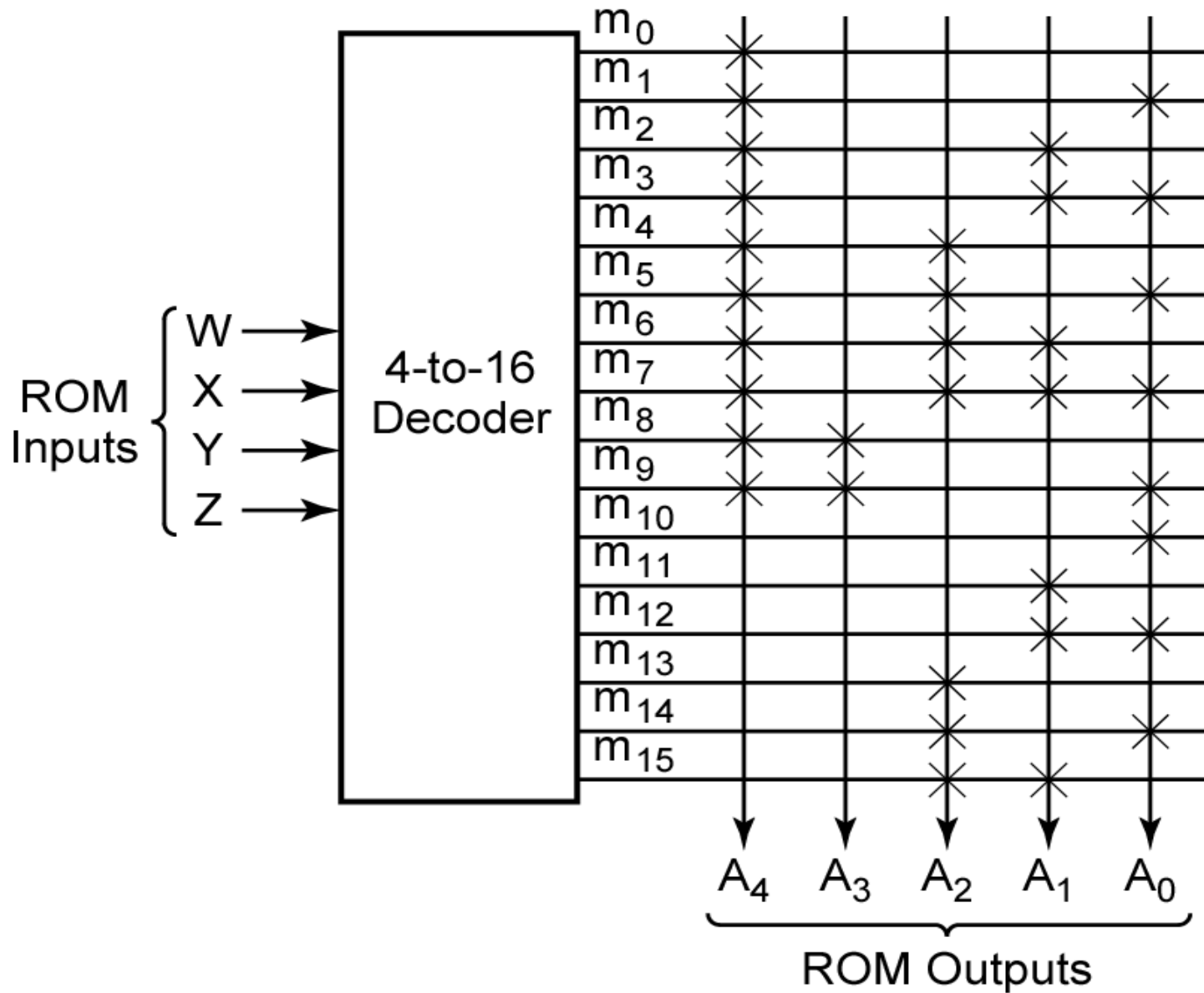
**Figure 9-20: An 8-Word x 4-Bit ROM**

Input				Hex Digit	ASCII Code for Hex Digit						
W	X	Y	Z		A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>
0	0	0	0	0	0	1	1	0	0	0	0
0	0	0	1	1	0	1	1	0	0	0	1
0	0	1	0	2	0	1	1	0	0	1	0
0	0	1	1	3	0	1	1	0	0	1	1
0	1	0	0	4	0	1	1	0	1	0	0
0	1	0	1	5	0	1	1	0	1	0	1
0	1	1	0	6	0	1	1	0	1	1	0
0	1	1	1	7	0	1	1	0	1	1	1
1	0	0	0	8	0	1	1	1	0	0	0
1	0	0	1	9	0	1	1	1	0	0	1
1	0	1	0	A	1	0	0	0	0	0	1
1	0	1	1	B	1	0	0	0	0	1	0
1	1	0	0	C	1	0	0	0	0	1	1
1	1	0	1	D	1	0	0	0	1	0	0
1	1	1	0	E	1	0	0	0	1	0	1
1	1	1	1	F	1	0	0	0	1	1	0

**Figure 9-22: Hexadecimal to ASCII Code Converter**



**Figure 9-22: Hexadecimal to ASCII Code Converter**



**Figure 9-23: ROM Realization of Code Converter**

# Programmable Logic Devices

A programmable logic device (or PLD) is a general name for a digital integrated circuit capable of being programmed to provide a variety of different logic functions.

When a digital system is designed using a PLD, changes in the design can easily be made by changing the programming of the PLD without having to change the wiring in the system.

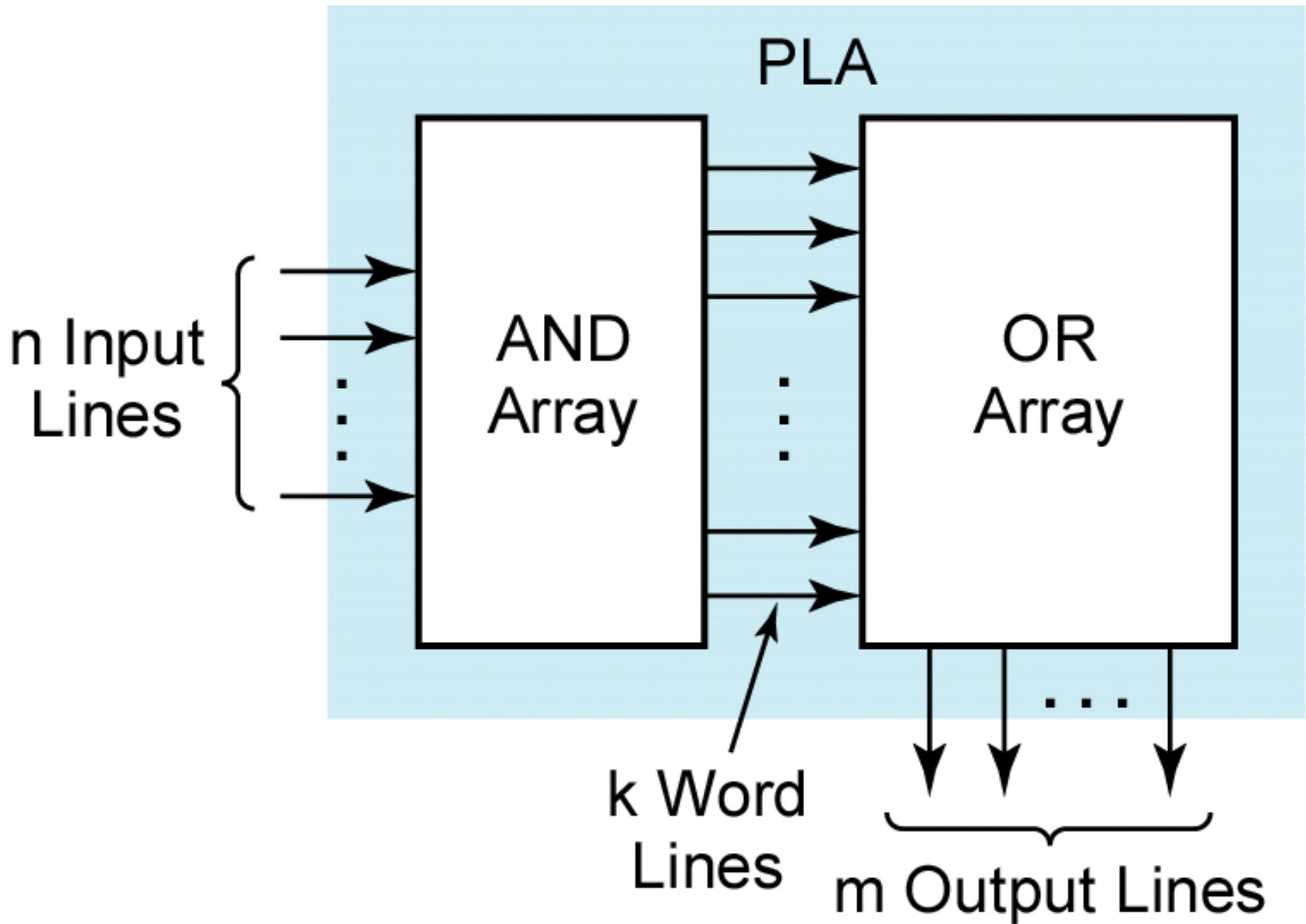
**Section 9.6 (p. 263)**

# Programmable Logic Arrays

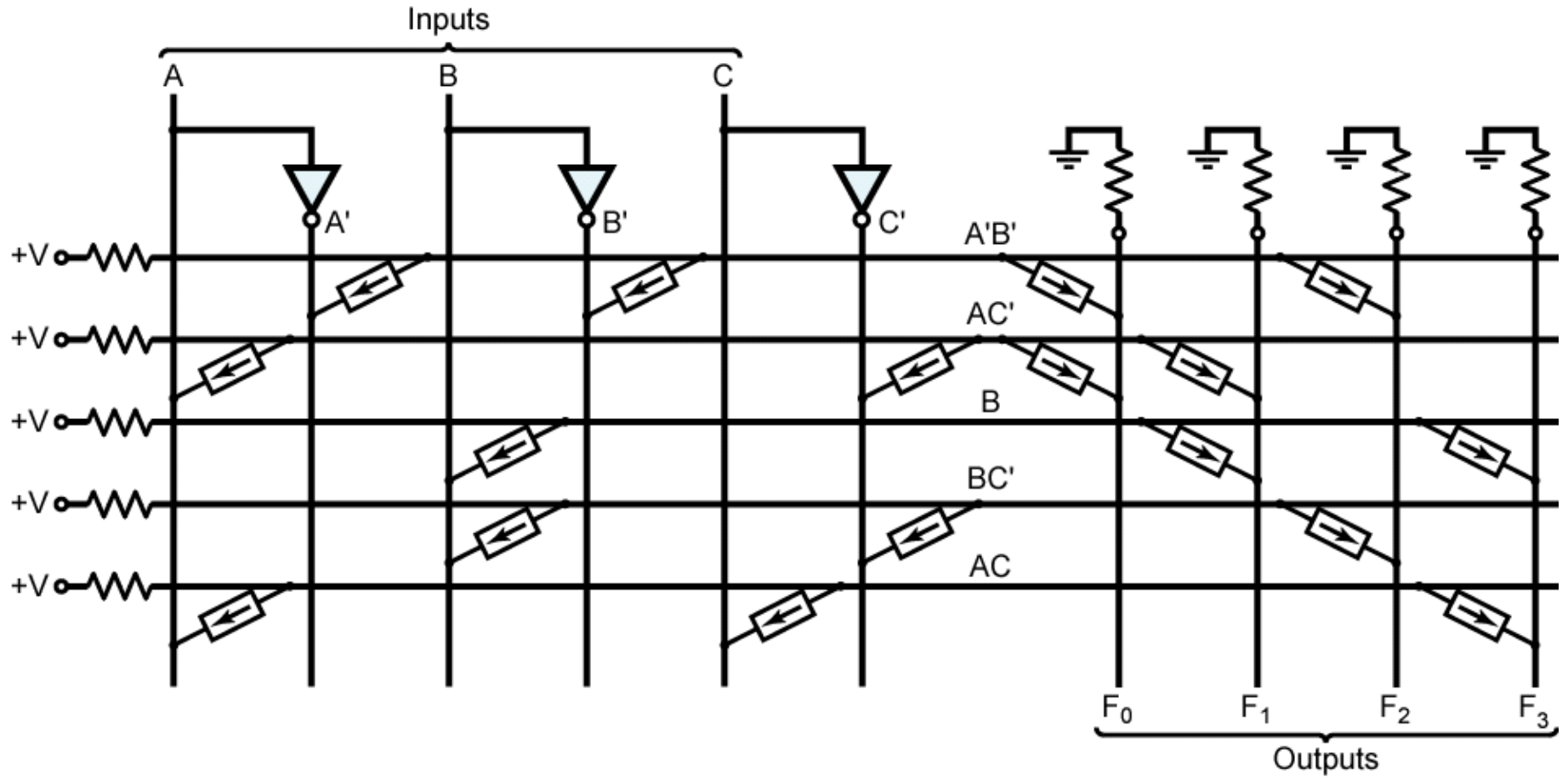
A programmable logic array (PLA) performs the same basic function as a ROM. A PLA with  $n$  inputs and  $m$  outputs can realize  $m$  functions of  $n$  variables.

The internal organization of the PLA is different from that of the ROM in that the decoder is replaced with an AND array which realizes selected product terms of the input variables. The OR array Ors together the product terms needed to form the output functions, so a PLA implements a sum-of-products expression.

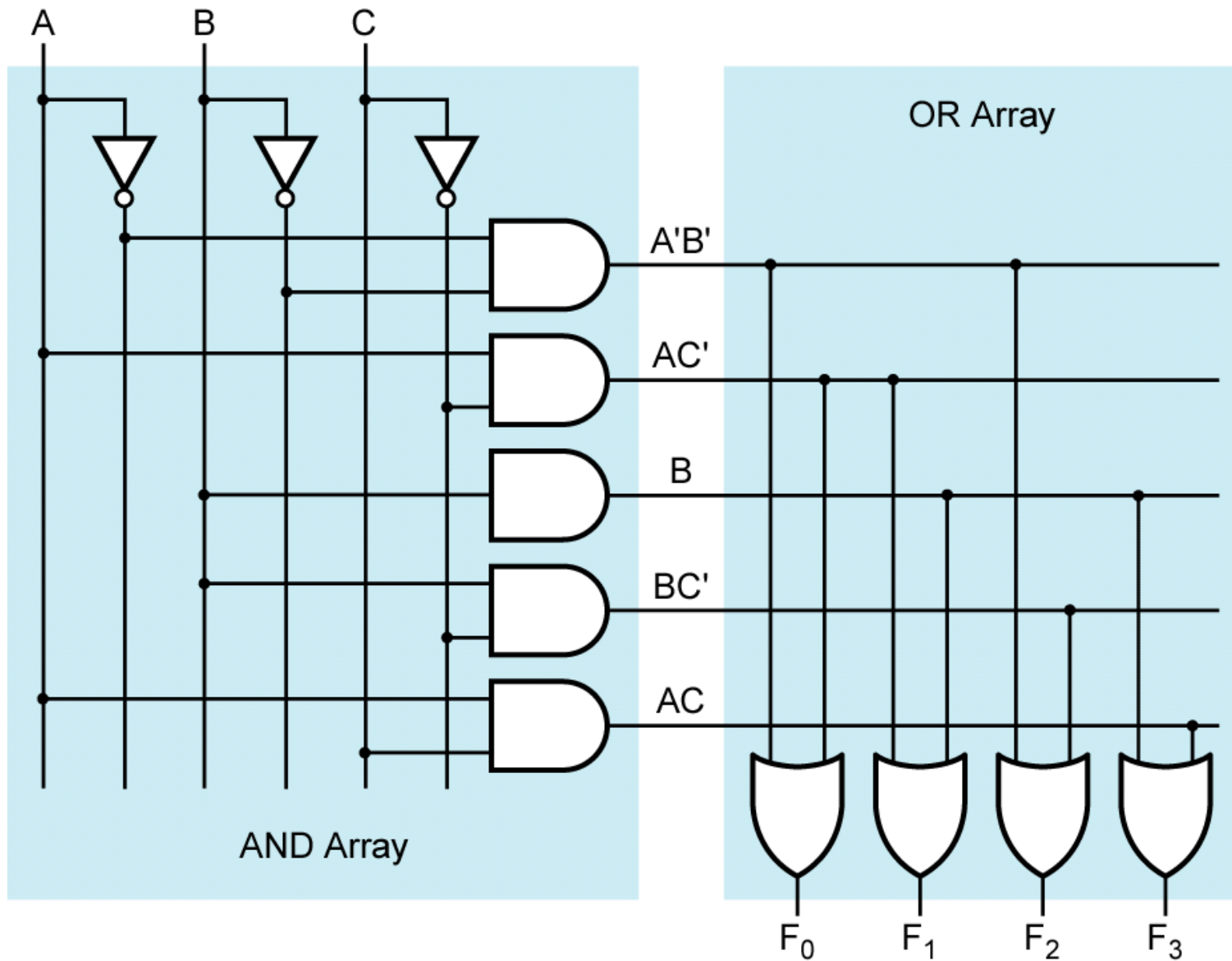
**Section 9.6 (p. 263)**



**Figure 9-24: Programmable Logic Array Structure**



**Figure 9-25: PLA with Three Inputs, Five Product Terms, and Four Outputs**



**Figure 9-26: AND-OR Array Equivalent to Figure 9-25**

**Table 9-1. PLA Table for Figure 9-25**

Product Term	Inputs			Outputs			
	<i>A</i>	<i>B</i>	<i>C</i>	<i>F</i> <sub>0</sub>	<i>F</i> <sub>1</sub>	<i>F</i> <sub>2</sub>	<i>F</i> <sub>3</sub>
<i>A'B'</i>	0	0	–	1	0	1	0
<i>AC'</i>	1	–	0	1	1	0	0
<i>B</i>	–	1	–	0	1	0	1
<i>BC'</i>	–	1	0	0	0	1	0
<i>AC</i>	1	–	1	0	0	0	1

$$F_0 = A'B' + AC'$$

$$F_1 = AC' + B$$

$$F_2 = A'B' + BC'$$

$$F_3 = B + AC$$

# PLA Tables

The input side of a PLA table defines the product terms generated by the AND array:

- 0** indicates a complemented variable
- 1** indicates an uncomplemented variable
- indicates a missing variable

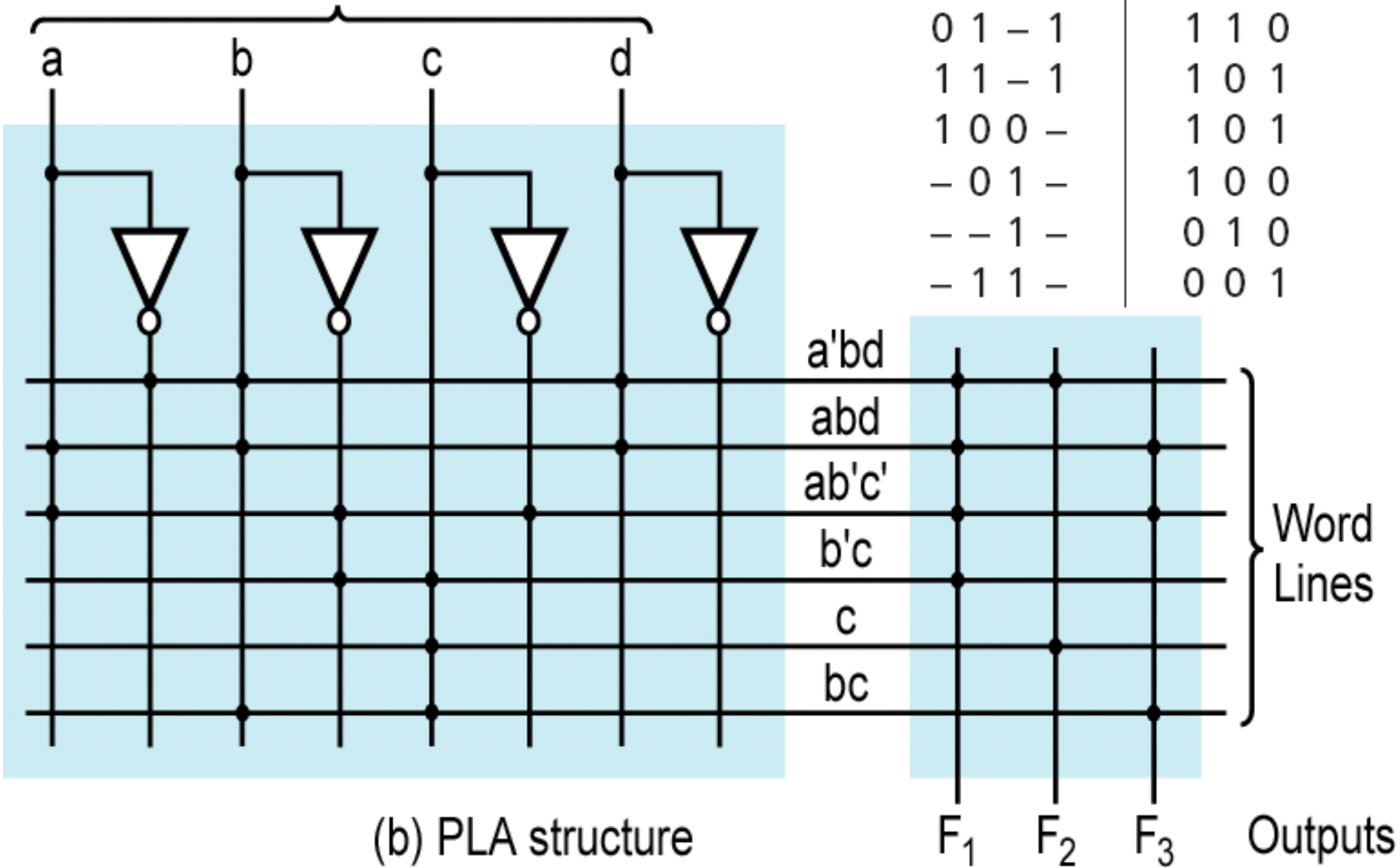
The output side of a PLA table specifies which product terms are ORed together to form the output functions:

- 0** indicates a product term is not present
- 1** indicates a product term is present.

Unlike a truth table, zero, one, or more rows in a PLA table can be selected at the same time.

(a) PLA table

$a$	$b$	$c$	$d$	$f_1$	$f_2$	$f_3$
0	1	-	1	1	1	0
1	1	-	1	1	0	1
1	0	0	-	1	0	1
-	0	1	-	1	0	0
-	-	1	-	0	1	0
-	1	1	-	0	0	1



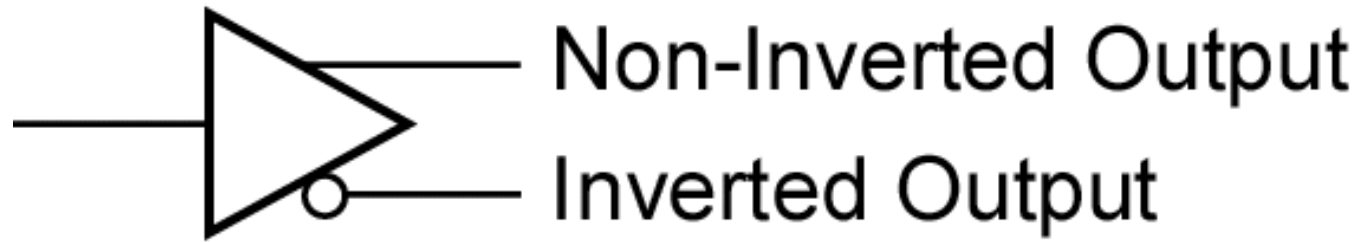
**Figure 9-27b: PLA Realization of Equations**

# Programmable Array Logic

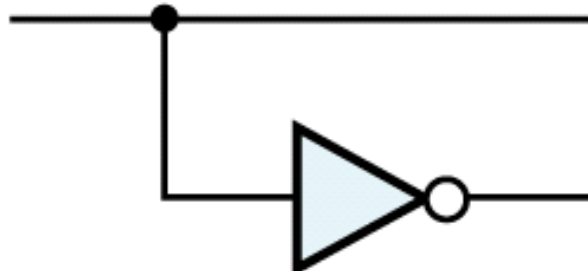
The PAL (programmable array logic) is a special case of the PLA in which the AND array is programmable and the OR array is fixed.

Because only the AND array is programmable, the PAL is less expensive than the more general PLA, and the PAL is easier to program.

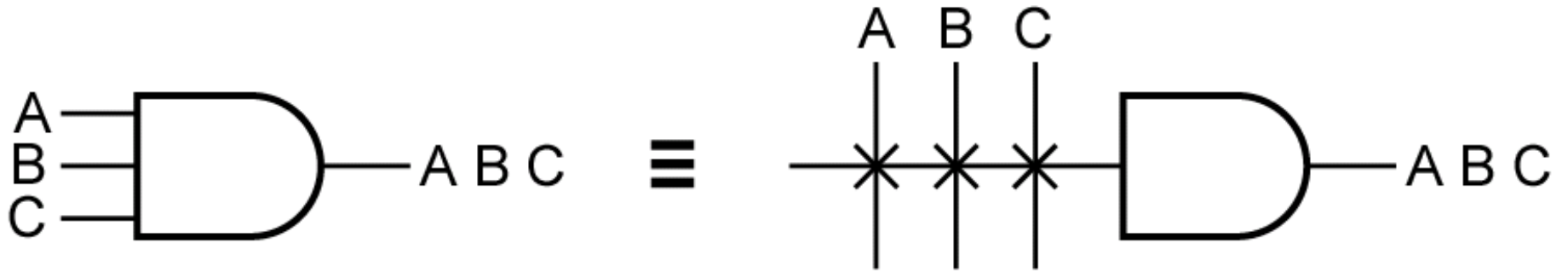
**Section 9.6 (p. 263)**



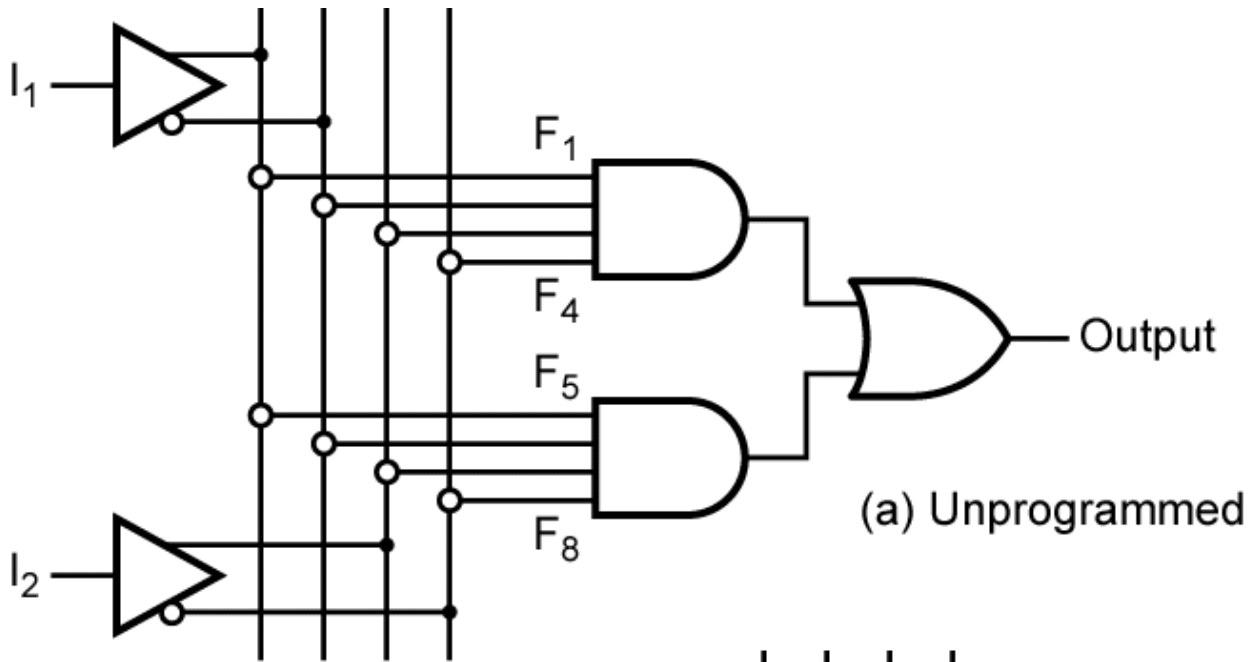
Buffer logically equivalent to



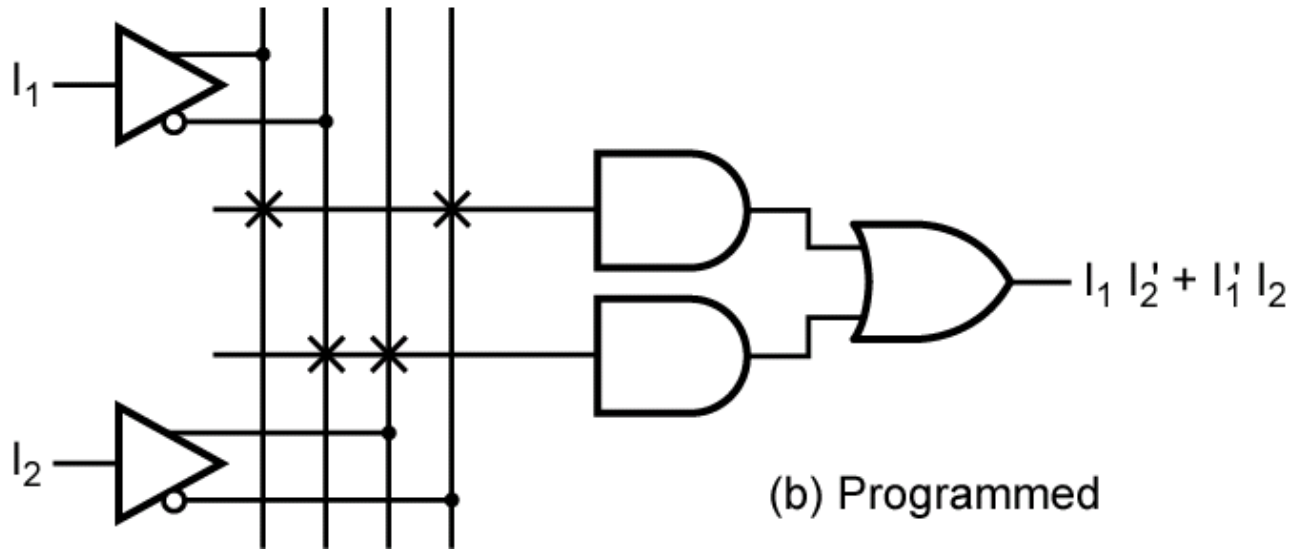
**Section 9.6 (p. 266)**



**Section 9.6 (p. 267)**

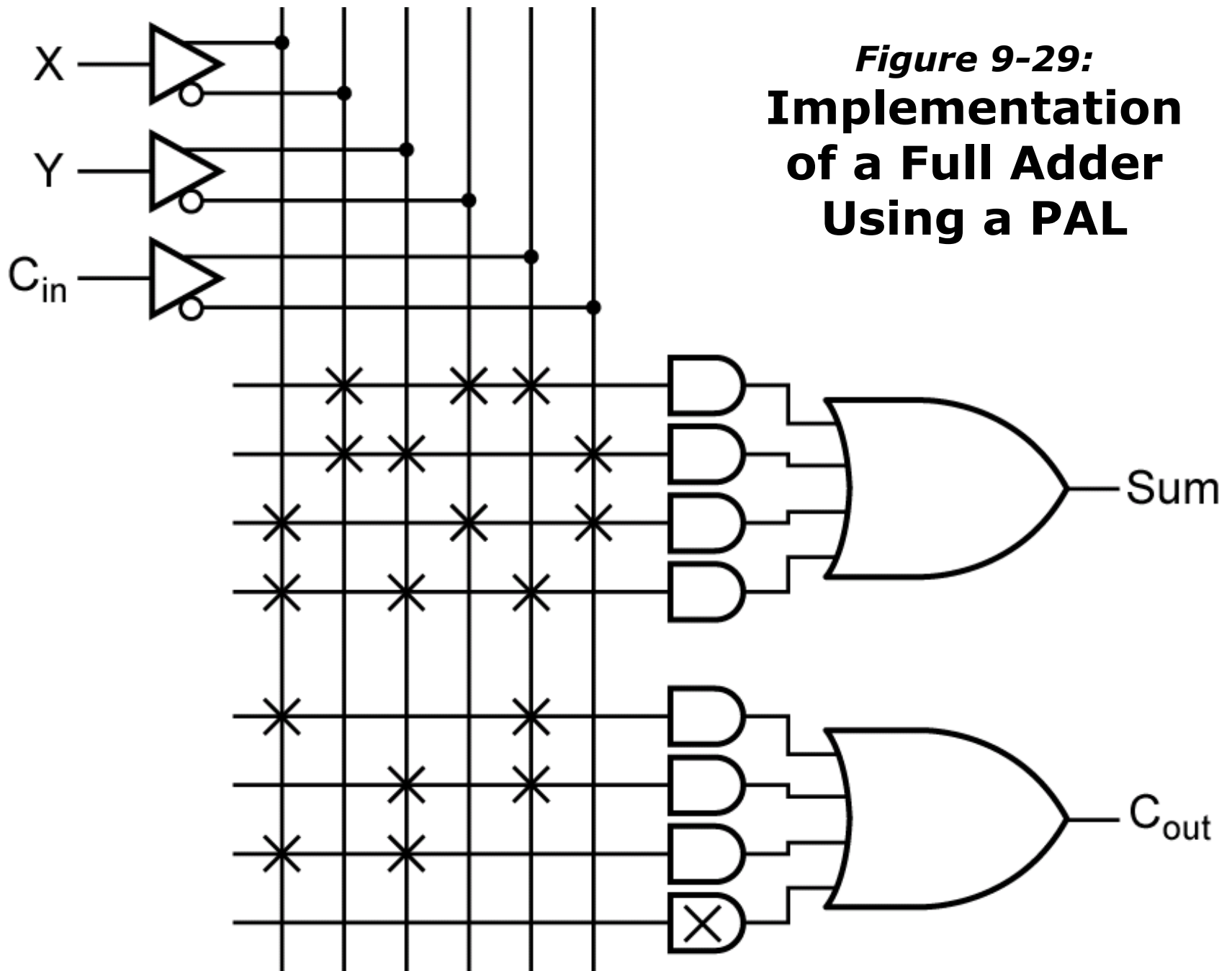


(a) Unprogrammed



(b) Programmed

**Figure 9-28:**  
**PAL Segment**



**Figure 9-29:**  
**Implementation**  
**of a Full Adder**  
**Using a PAL**

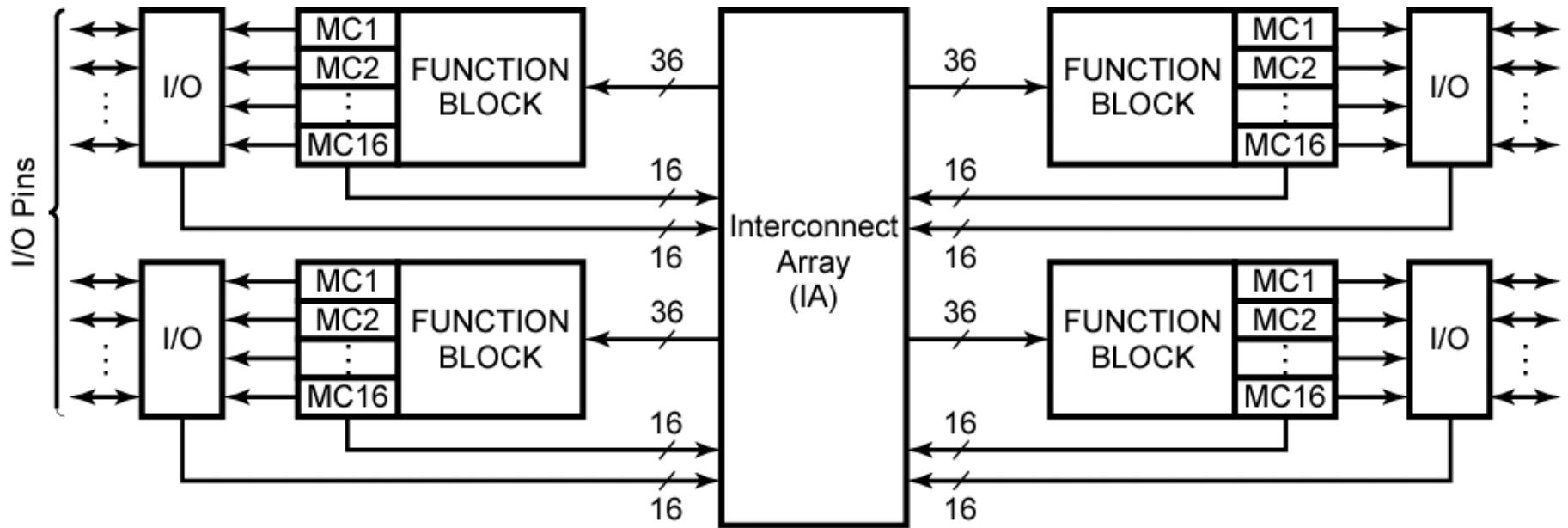
# Complex Programmable Logic Devices

As integrated circuit technology continues to improve, more and more gates can be placed on a single chip. This has allowed the development of complex programmable logic devices (CPLDs).

Instead of a single PAL or PLA on a chip, many PALs or PLAs can be placed on a single CPLD chip and interconnected.

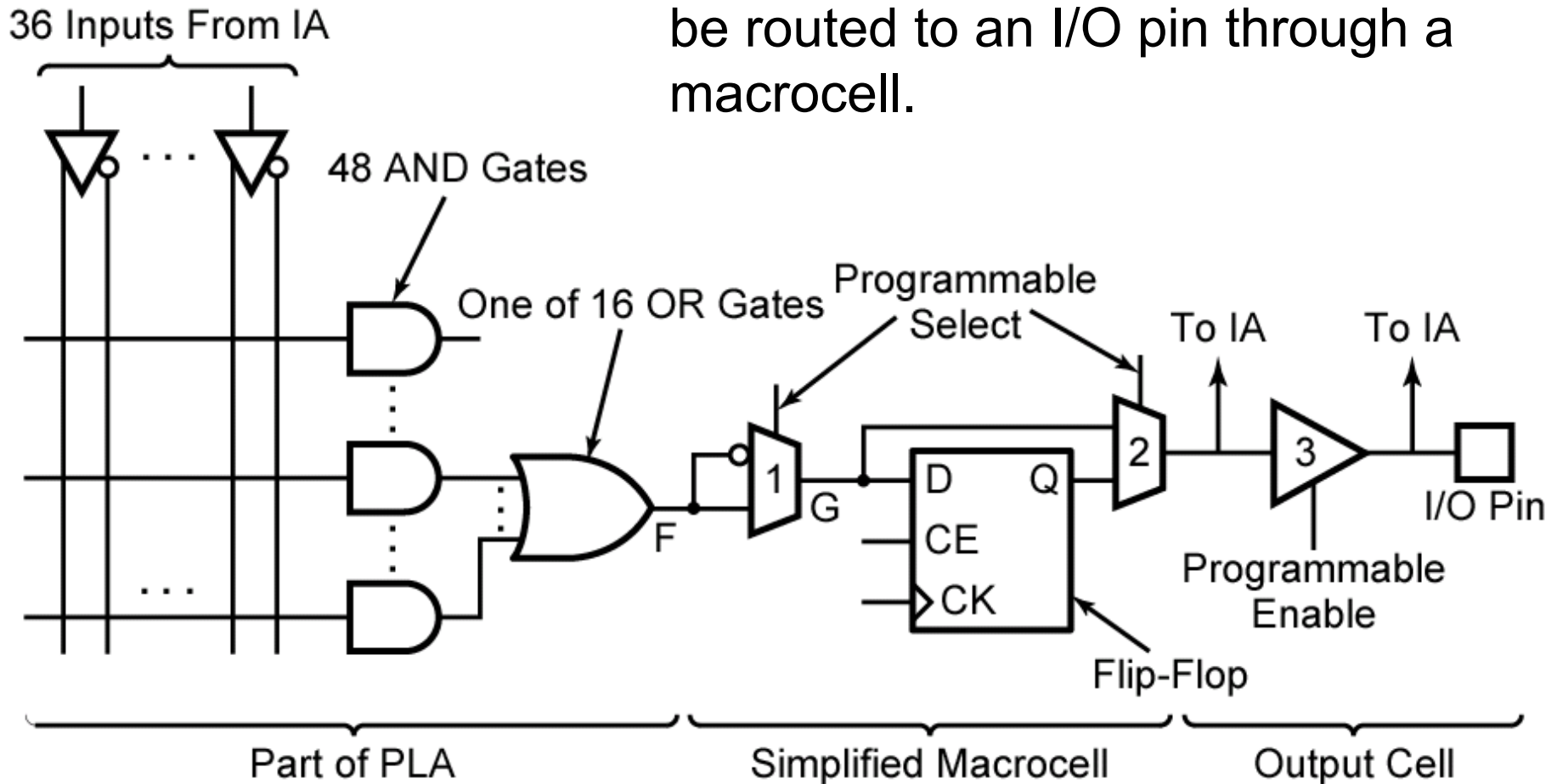
When storage elements such as flip-flops are also included on the same integrated circuit (IC), a small digital system can be implemented with a single CPLD.

**Section 9.7 (p. 268)**



**Figure 9-30: Architecture of Xilinx XCR3064XL CPLD**  
 (Figure based on figures and text owned by Xilinx, Inc., Courtesy of Xilinx, Inc. © Xilinx, Inc. 1999-2003. All rights reserved.)

Signals generated in a PLA can be routed to an I/O pin through a macrocell.



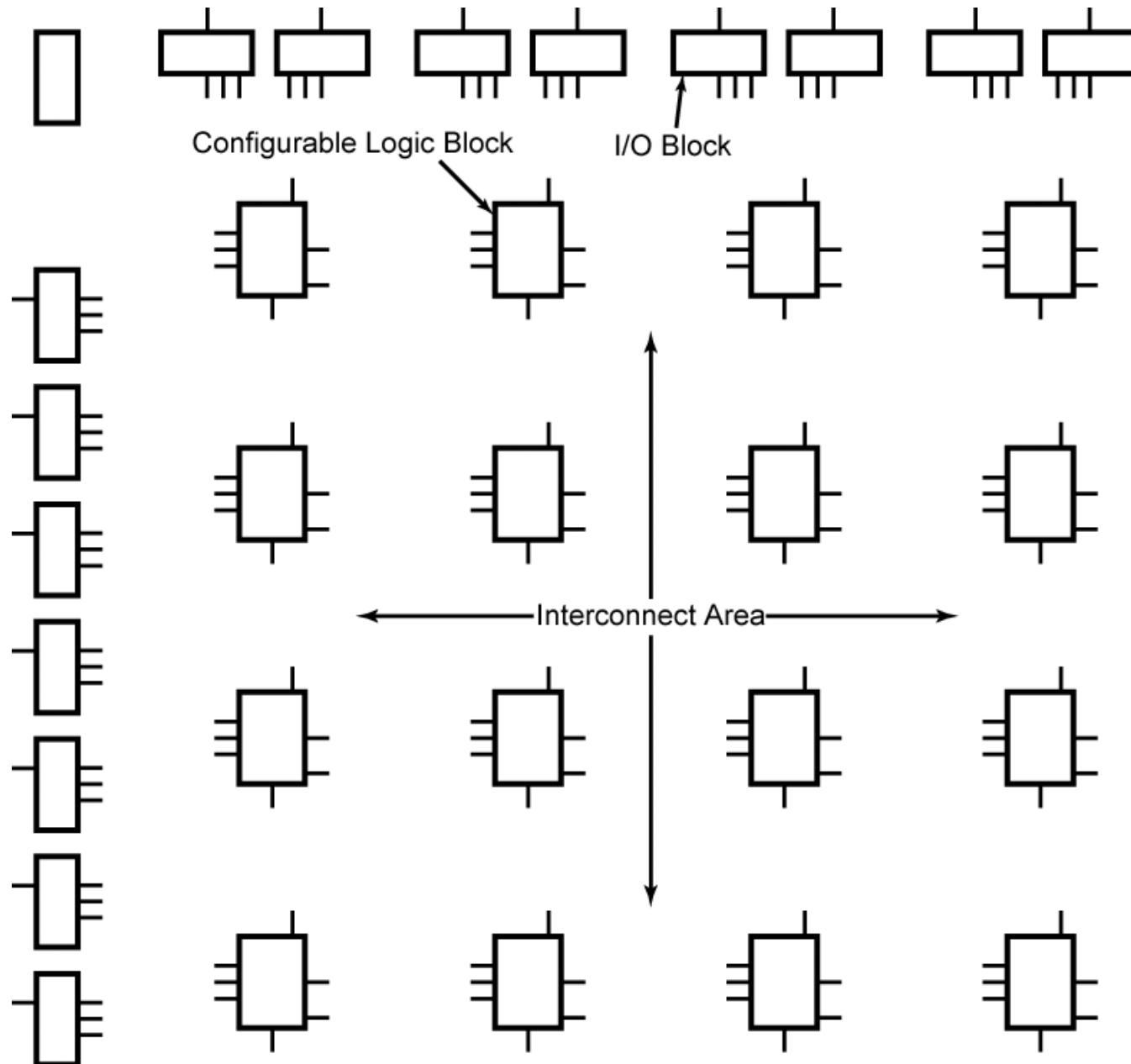
**Figure 9-31: CPLD Function Block and Macrocell (A Simplified Version of XCR3064XL)**

# Field-Programmable Gate Arrays

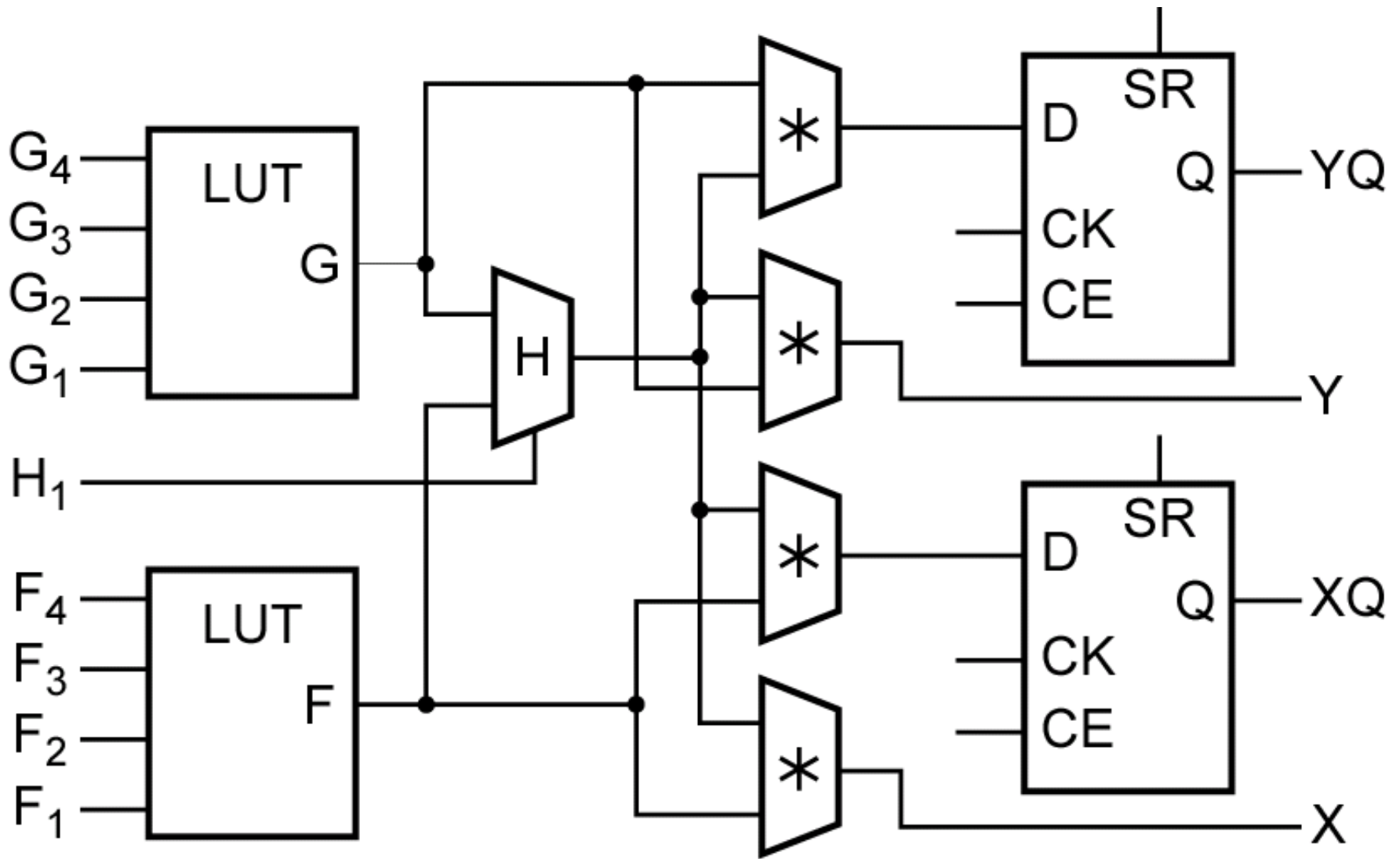
A field-programmable gate array (FPGA) is an IC that contains an array of identical logic cells with programmable interconnections. The user can program the functions realized by each logic cell and the connections between the cells.

The interior of the FPGA consists of an array of logic cells, also called configurable logic blocks (CLBs). The array of CLBs is surrounded by a ring of I/O interface blocks. These I/O blocks connect the CLB signals to IC pins.

**Section 9.8 (p. 270)**



**Figure 9-32: Layout of a Typical FPGA**

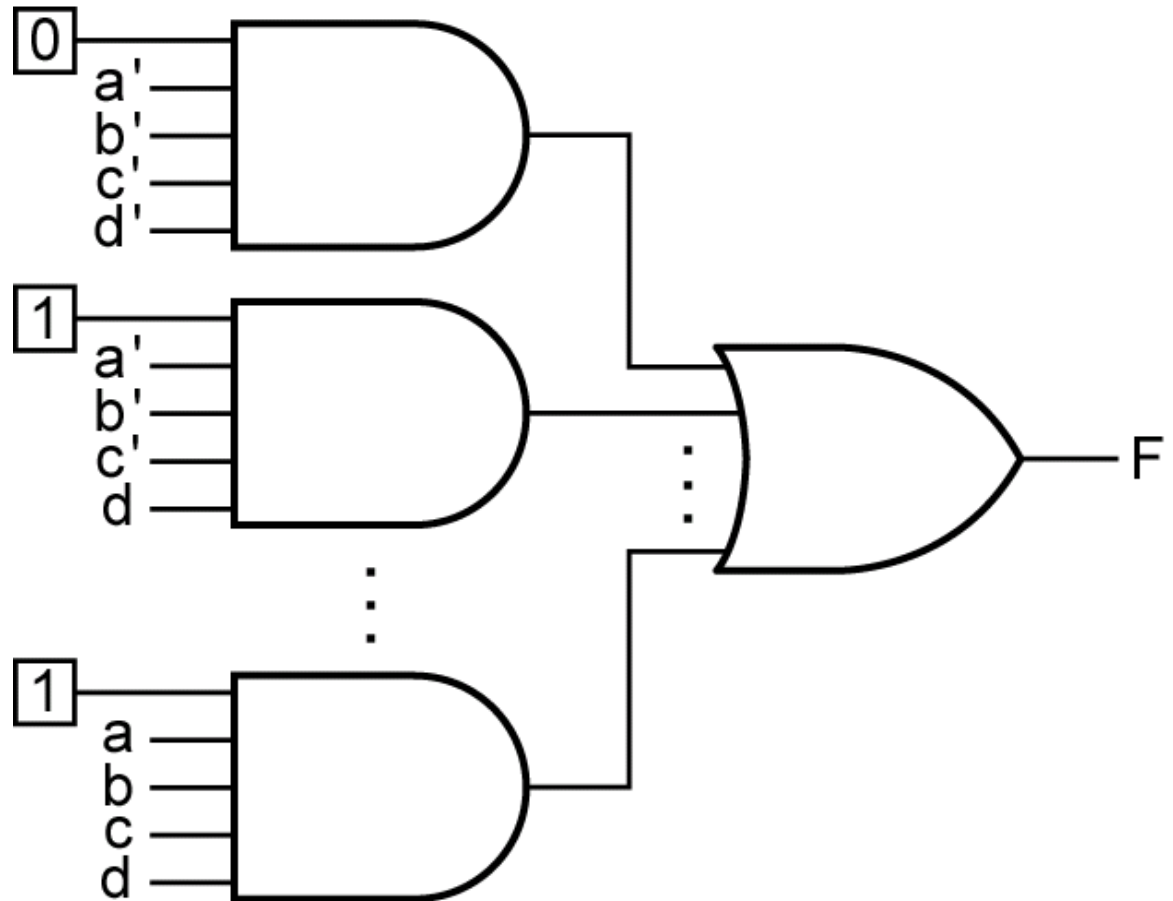


\* = Programmable MUX

**Figure 9-33: Simplified Configurable Logic Block (CLB)**

abcd	F
0000	0
0001	1
⋮	⋮
⋮	⋮
⋮	⋮
1111	1

A four-input LUT is essentially a reprogrammable ROM with 16 1-bit words.



**Figure 9-34:**  
**Implementation of a Lookup Table (LUT)**

# Shannon's Expansion Theorem

In order to implement a switching function of more than three variables using 3-variable function generators, the function must be decomposed into subfunctions where each subfunction requires only three variables.

For example, we can expand a function of the variables  $a$ ,  $b$ ,  $c$ , and  $d$  about the variable  $a$ :

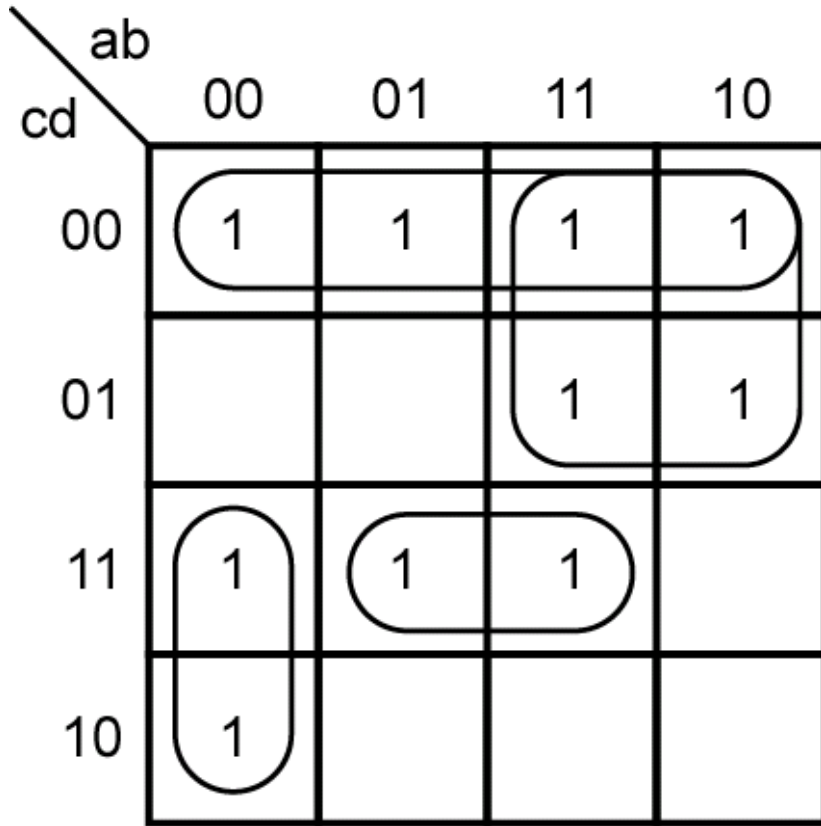
$$f(a, b, c, d) = a'f(0, b, c, d) + af(1, b, c, d) = a'f_0 + af_1 \quad (9-6)$$

$$f(a, b, c, d) = c'd' + a'b'c + bcd + ac' \quad (9-7)$$

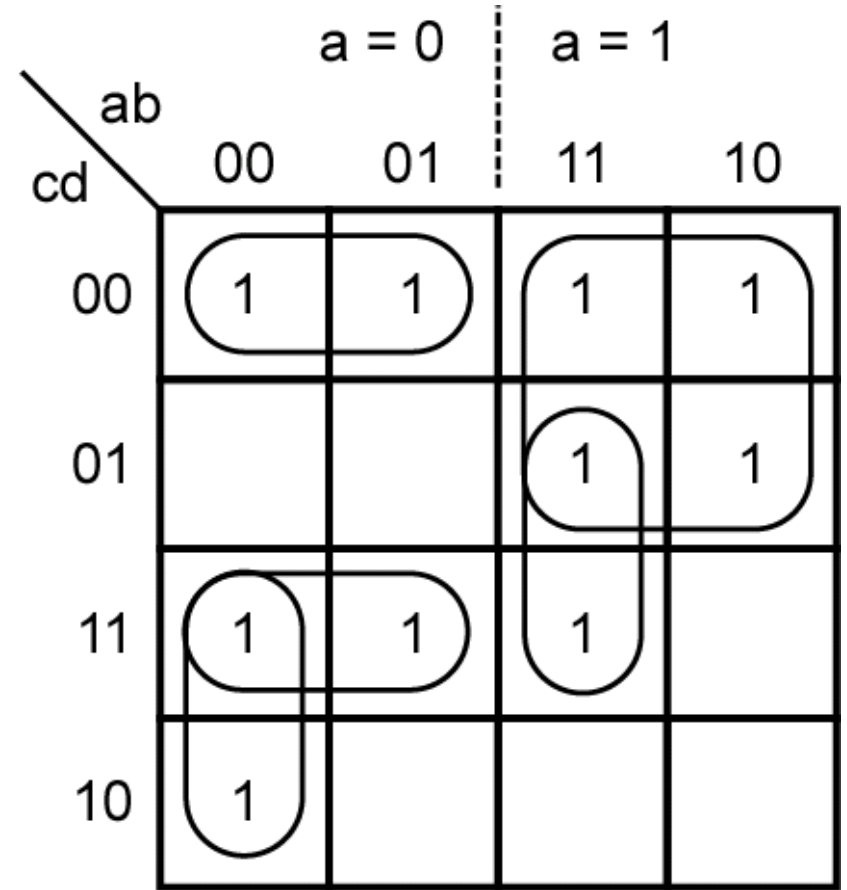
$$= a'(c'd' + b'c + bcd) + a(c'd' + bcd + c')$$

$$= a'(c'd' + b'c + cd) + a(c' + bd) = a'f_0 + af_1$$

## Section 9.8 (p. 271-272)



F



$F_0$

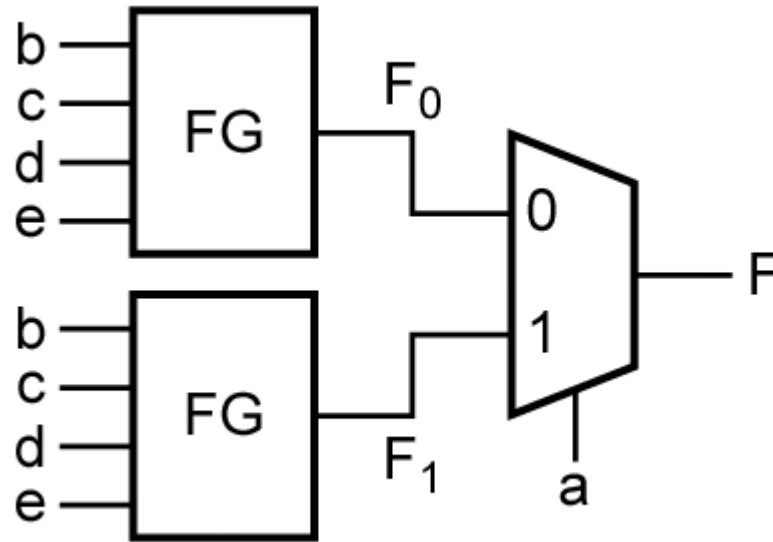
$F_1$

**Figure 9-35: Function Expansion Using a Karnaugh Map**

The general form of Shannon's expansion theorem for expanding an  $n$ -variable function about the variable  $x_i$  is

$$\begin{aligned} & f(x_1, x_2, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n) \\ &= x_i' f(x_1, x_2, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n) + \\ & \quad x_i f(x_1, x_2, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) \\ &= x_i' f_0 + x_i f_1 \end{aligned} \tag{9-8}$$

Where  $f_0$  is the  $(n - 1)$ -variable function obtained by setting  $x_i$  to 0 in the original function and  $f_1$  is the  $(n - 1)$ -variable function obtained by setting  $x_i$  to 1 in the original function.



(a) 5-variable function

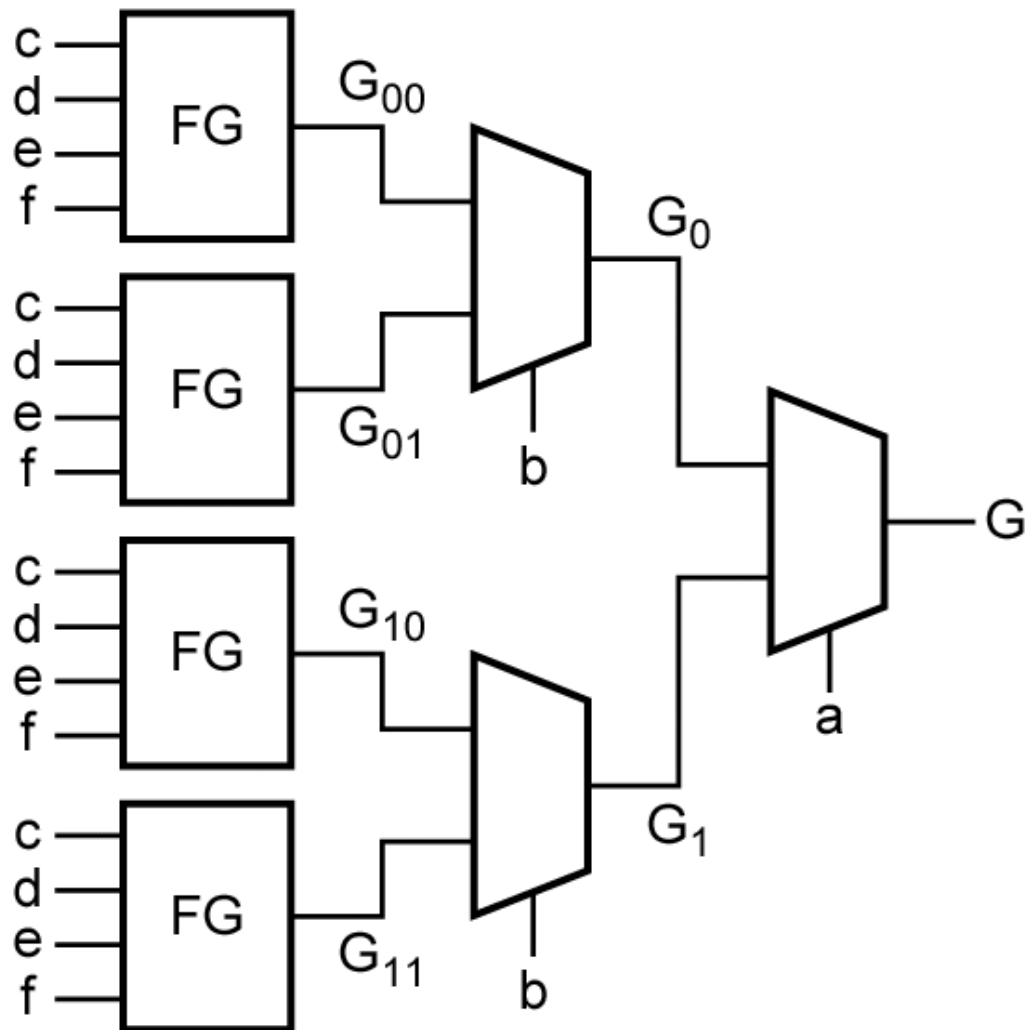
$$\begin{aligned}
 f(a, b, c, d, e) &= \\
 a' f(0, b, c, d, e) &+ a f(1, b, c, d, e) \\
 &= a' f_0 + a f_1 \qquad (9-9)
 \end{aligned}$$

**Figure 9-36: Realization of a 5-Variable Function with Function Generators and a MUX**

$$G(a, b, c, d, e, f) = a'G_0 + aG_1$$

$$G_0 = b'G_{00} + bG_{01}$$

$$G_1 = b'G_{10} + bG_{11}$$



(b) 6-variable function

**Figure 9-36: Realization of a 6-Variable Function with Function Generators and a MUX**