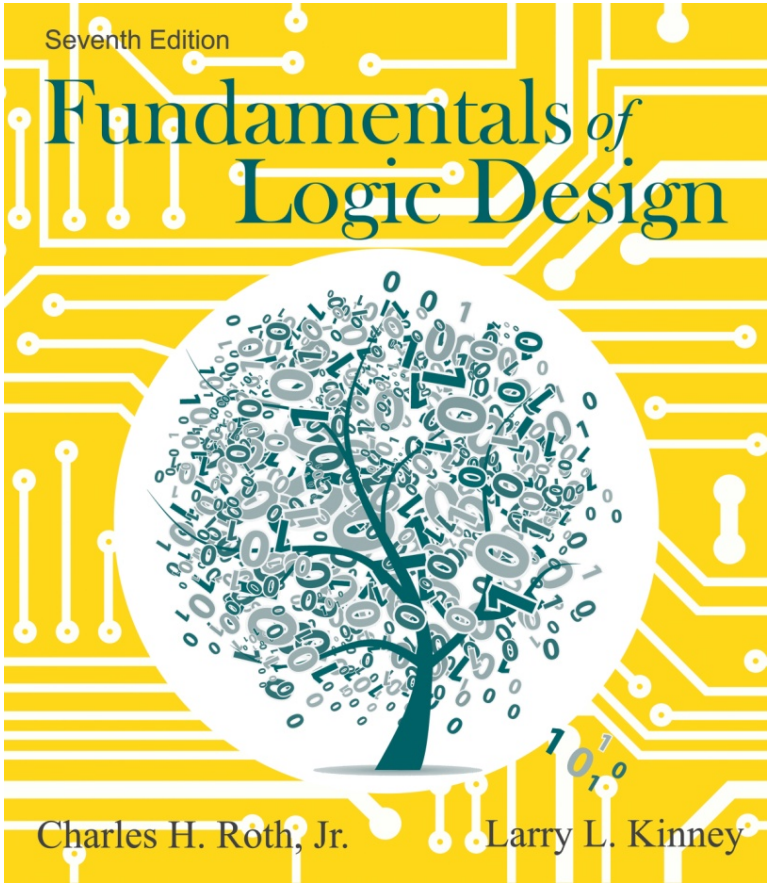


UNIT 9

Multiplexers, Decoders, and Programmable Logic Devices

This chapter includes:

Learning Objectives

1. Explain the function of a multiplexer. Implement a multiplexer using gates.
2. Explain the operation of three-state buffers. Determine the resulting output when three-state buffer outputs are connected together. Use three-state buffers to multiplex signals onto a bus.
3. Explain the operation of a decoder and encoder. Use a decoder with added gates to implement a set of logic functions. Implement a decoder or priority encoder using gates.
4. Explain the operation of a read-only memory (ROM). Use a ROM to implement a set of logic functions.

Learning Objectives

5. Explain the operation of a programmable logic array (PLA). Use a PLA to implement a set of logic functions. Given a PLA table or an internal connection diagram for a PLA, determine the logic functions realized.

6. Explain the operation of a programmable array logic device (PAL). Determine the programming pattern required to realize a set of logic functions with a PAL.

7. Explain the operation of a complex programmable logic device (CPLD) and a field-programmable gate array (FPGA).

8. Use Shannon's expansion theorem to decompose a switching function.

Introduction

Introduction:

- In this unit we introduce the use of more complex integrated circuits (ICs) in logic design.
- Integrated circuits may be classified as small-scale integration (SSI), medium-scale integration (MSI), large-scale integration (LSI), or very-large-scale integration (VLSI), depending on the number of gates in each integrated circuit package and the type of function performed.
- SSI functions include NAND, NOR, AND, and OR gates, inverters, and flip-flops.

Multiplexers

Multiplexers:

- A multiplexer (or data selector, abbreviated as MUX) has a group of data inputs and a group of control inputs.
- The control inputs are used to select one of the data inputs and connect it to the output terminal.
- A 2-1 MUX and its logic equation are shown below:

$$Z = A'I_0 + AI_1$$

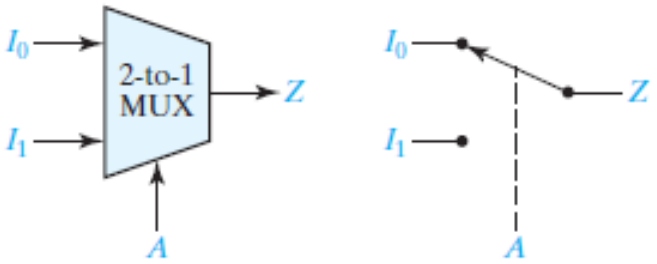
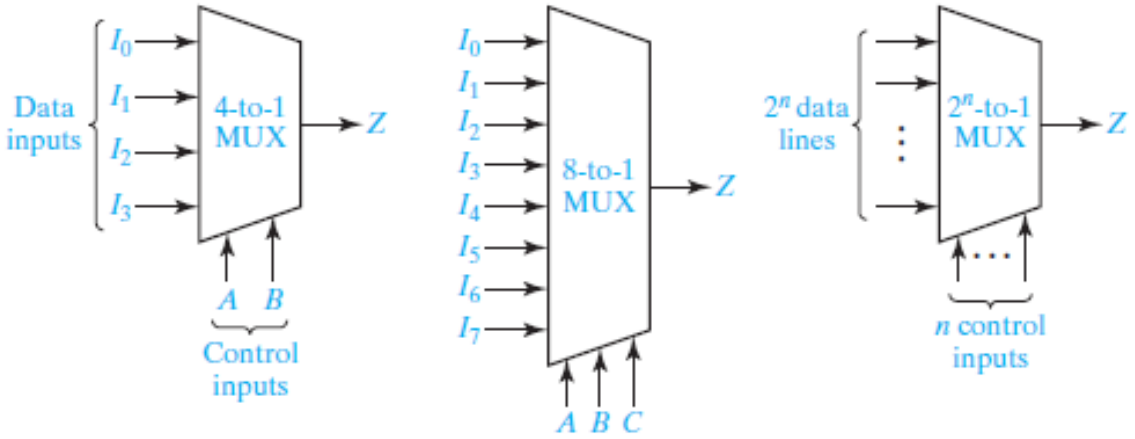


FIGURE 9-1
2-to-1 Multiplexer and Switch Analog

Multiplexers

More Multiplexers:

FIGURE 9-2
Multiplexers



Logic Equation for 4-1 MUX:

$$Z = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3 \quad (9\text{-}1)$$

Logic Equation for 8-1 MUX:

$$Z = A'B'C'I_0 + A'B'CI_1 + A'BC'I_2 + A'BCI_3 + AB'C'I_4 + AB'CI_5 + ABC'I_6 + ABCI_7 \quad (9\text{-}2)$$

Logic Equation for 2^n-1 MUX:

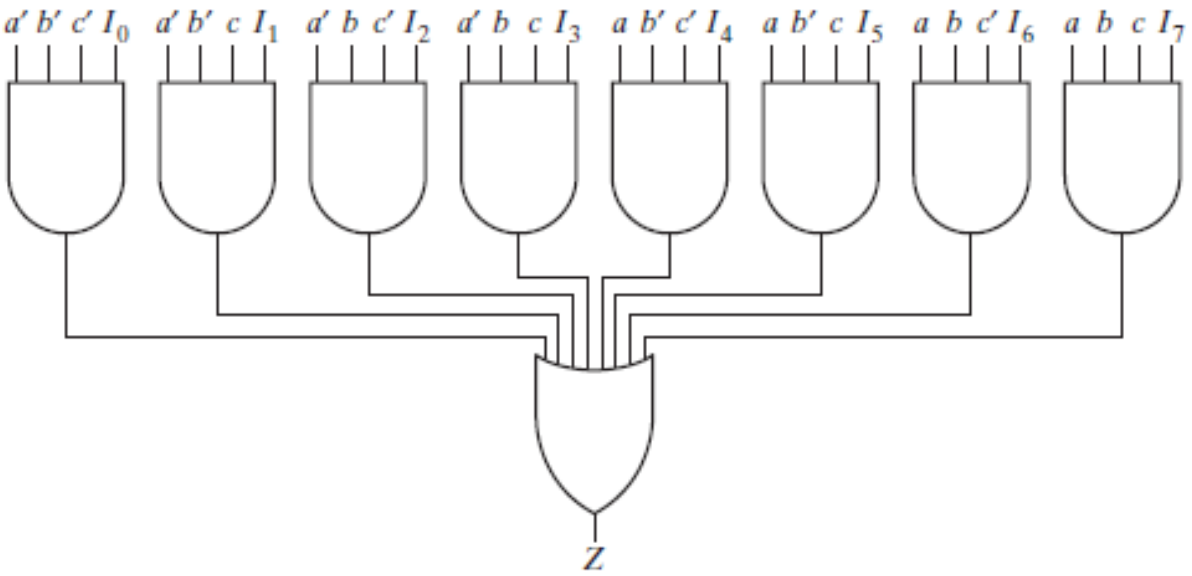
$$Z = \sum_{k=0}^{2^n-1} m_k I_k$$

where m_k is a minterm of the n control variables and I_k is the corresponding data input.

Multiplexers

Logic Diagram for 8-to-1 MUX:

FIGURE 9-3
Logic Diagram for 8-to-1 MUX



Multiplexers

8-to-1 MUX NAND Implementation:

NAND Logic Equation from Factoring Equation 9-2:

$$Z = A'B'(C'I_0 + CI_1) + A'B(C'I_2 + CI_3) + AB'(C'I_4 + CI_5) + AB(C'I_6 + CI_7)$$

FIGURE 9-4
A Multi-Level Implementation of an 8-to-1 MUX

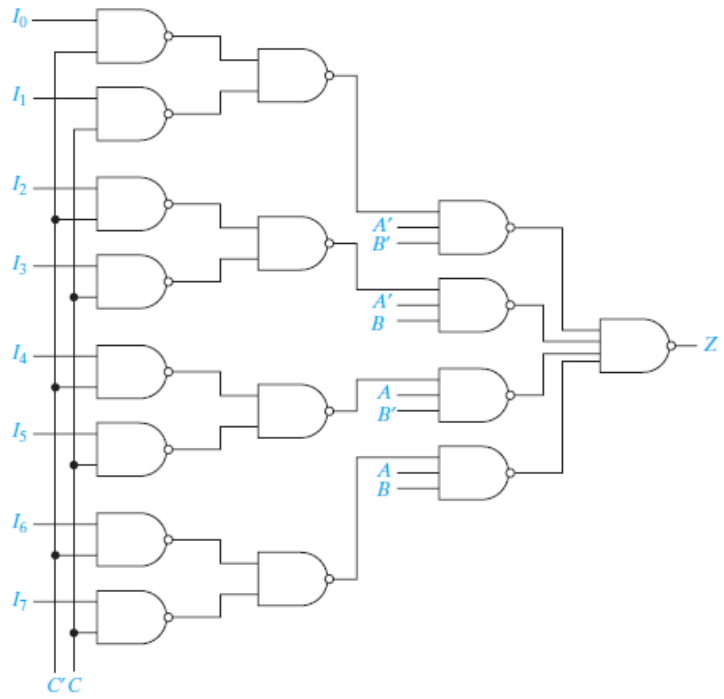
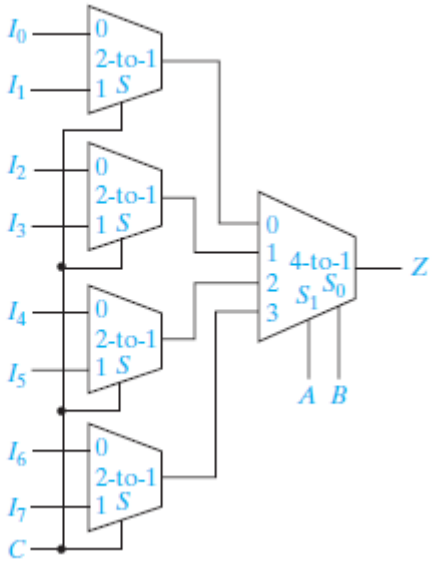


FIGURE 9-5
Component MUXs of Figure 9-4



Multiplexers

Quad Multiplexers to Select Data:

Multiplexers are often used to select data which is to be processed or stored in digital system design.

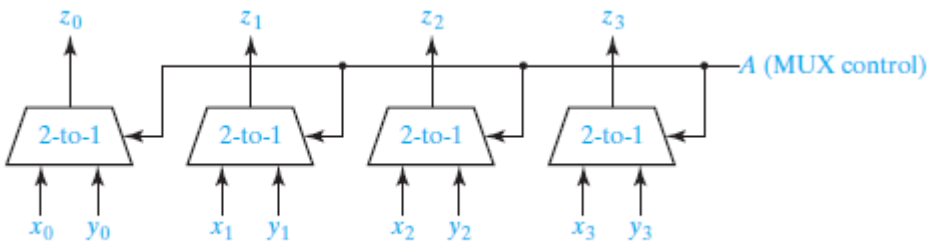


FIGURE 9-6
Quad Multiplexer Used to Select Data

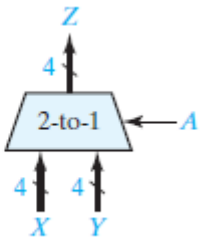


FIGURE 9-7
Quad Multiplexer with Bus Inputs and Output

Multiplexers

Enable:

- Another type of multiplexer has an additional input called an *enable*.
- The 8-to-1 MUX in Figure 9-3 can be modified to include an enable by changing the AND gates to five-input gates.
- The enable signal *E* is connected to the fifth input of each of the AND gates.
- Then, if *E* = 0, *Z* = 0 independent of the gate inputs *Ii* and the select inputs *a*, *b*, and *c*. However, if *E* = 1, then the MUX functions as an ordinary 8-to-1 multiplexer.

Multiplexers

4-1 Multiplexer Combinations and Implemented Functions:

FIGURE 9-8
Active-High, Active-Low Enable and Output Combinations

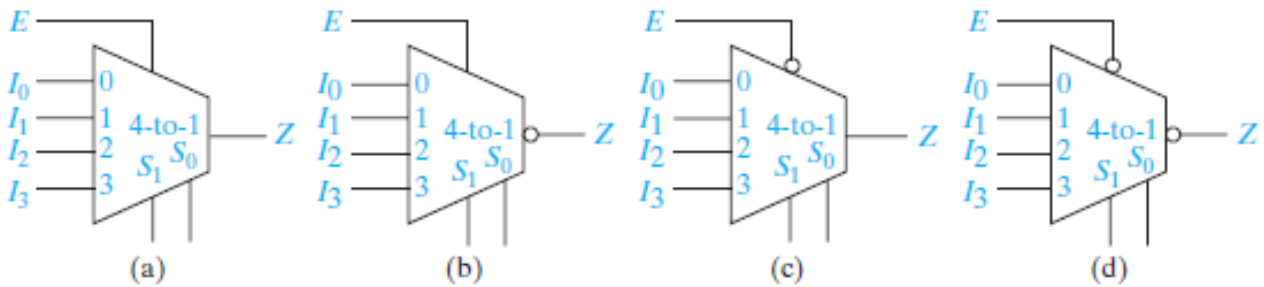
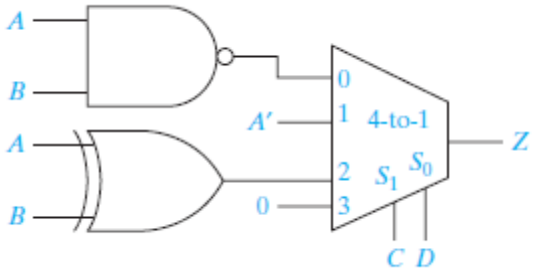


FIGURE 9-9
Four-Variable Function Implemented with a 4-to-1 MUX



Three-State Buffers

Buffers:

A gate output can only be connected to a limited number of other device inputs without degrading the digital system's performance.

A simple buffer may be used to increase the driving capability of a gate output. Figure 9-10 shows a buffer connected between a gate output and several gate inputs.

FIGURE 9-10
Gate Circuit with Added Buffer

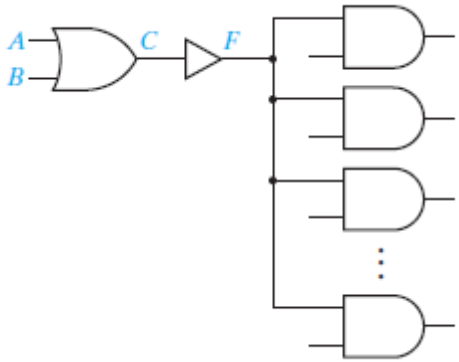
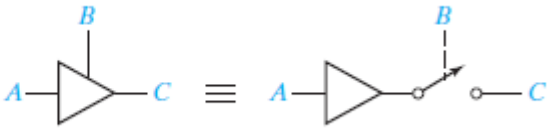


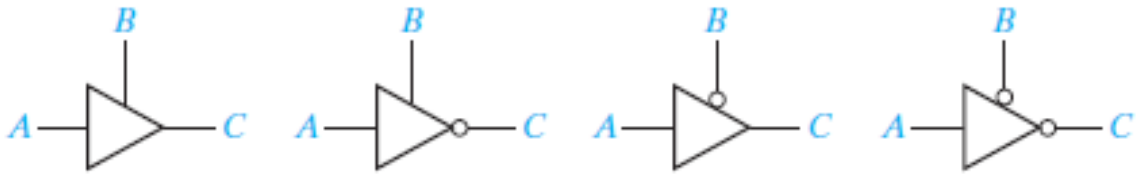
FIGURE 9-11
Three-State Buffer



Three-State Buffers

Four Kinds of Three State- Buffers:

FIGURE 9-12
Four Kinds of Three-State Buffers



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)

Three-State Buffers

Data Selection and Circuits with Three-State Buffers:

FIGURE 9-13
Data Selection Using Three-State Buffers

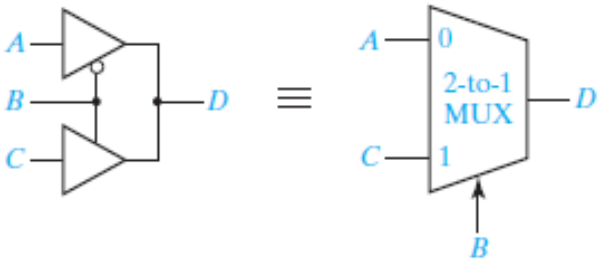
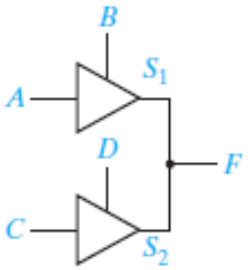


FIGURE 9-14
Circuit with Two Three-State Buffers

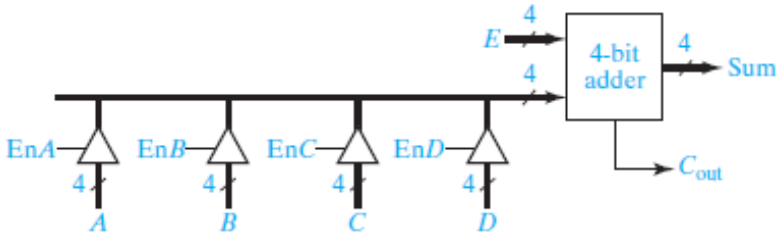


	S_2			
S_1	X	0	1	Z
X	X	X	X	X
0	X	0	X	0
1	X	X	1	1
Z	X	0	1	Z

Three-State Buffers

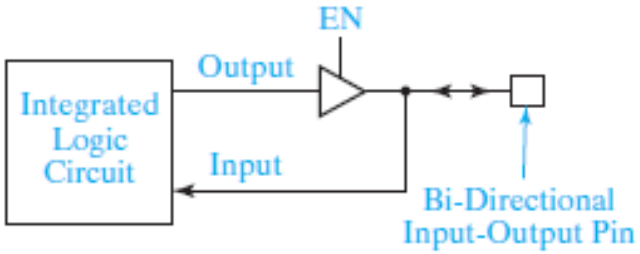
4-Bit Adder and Bidirectional Input-Outputs:

FIGURE 9-15
4-Bit Adder with Four Sources for One Operand



- **Bi-directional** means that the same pin can be used as an input pin and as an output pin, but not both at the same time.

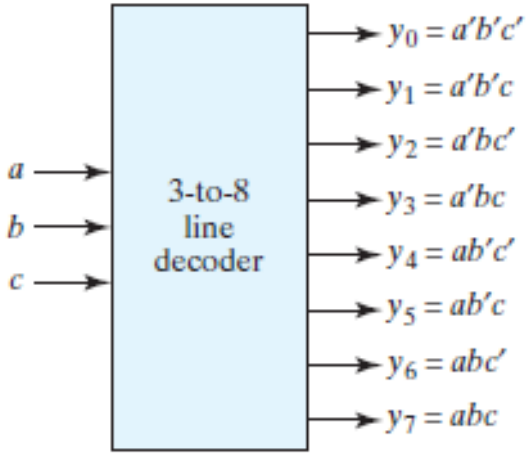
FIGURE 9-16
Integrated Circuit with Bi-Directional Input-Output Pin



Decoders and Encoders

3-8 Decoder Block Diagram and Truth Table:

FIGURE 9-17
A 3-to-8 Line Decoder

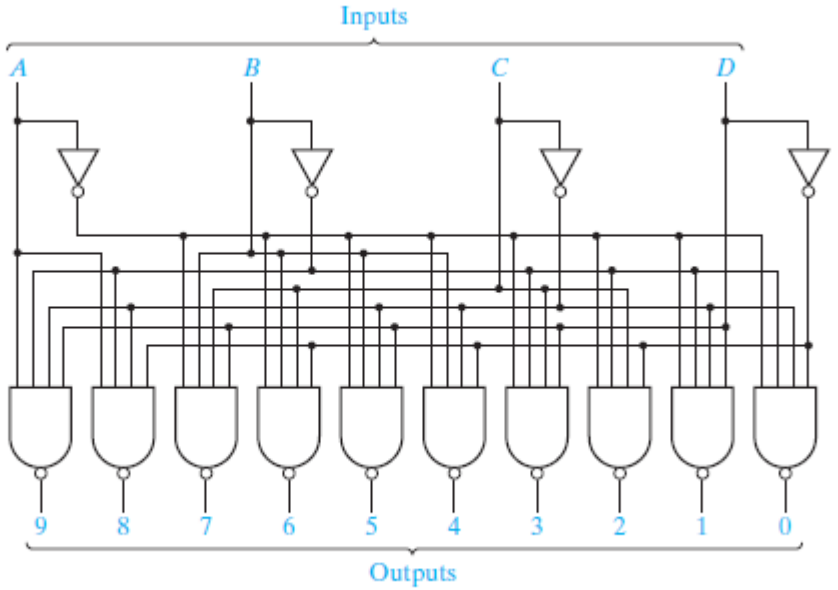


a	b	c	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Decoders and Encoders

4-10 Line Decoder Logic Diagram:

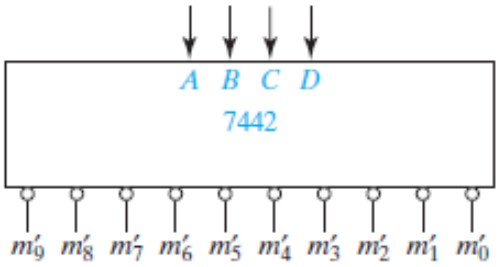
FIGURE 9-18
A 4-to-10 Line Decoder



(a) Logic diagram

Decoders and Encoders

4-10 Decoder Block Diagram and Truth Table:



(b) Block diagram

BCD Input				Decimal Output									
A	B	C	D	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
1	0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

(c) Truth Table

Decoders and Encoders

General Line Decoder Output Equations:

In general, an n-to-2^n line decoder generates all 2^n minterms (or maxterms) of the n input variables. The outputs are defined by the equations

$$y_i = m_i = M_i', \qquad i = 0 \text{ to } 2^n - 1 \qquad \text{(noninverted outputs)} \qquad (9\text{-}5)$$

or

$$y_i = m_i' = M_i, \qquad i = 0 \text{ to } 2^n - 1 \qquad \text{(inverted outputs)} \qquad (9\text{-}6)$$

where m_i is a minterm of the n input variables and M_i is a maxterm.

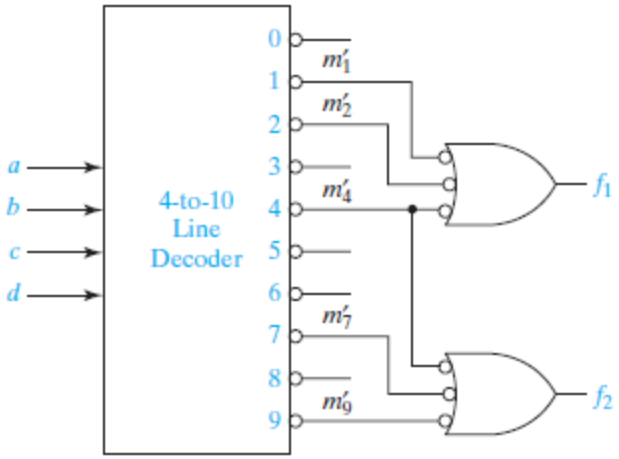
Decoders and Encoders

Realization of Functions Using Decoders:

For functions f1 and f2:

$$f_1(a, b, c, d) = m_1 + m_2 + m_4 \text{ and } f_2(a, b, c, d) = m_4 + m_7 + m_9$$

FIGURE 9-19
Realization of a Multiple-Output Circuit Using a Decoder

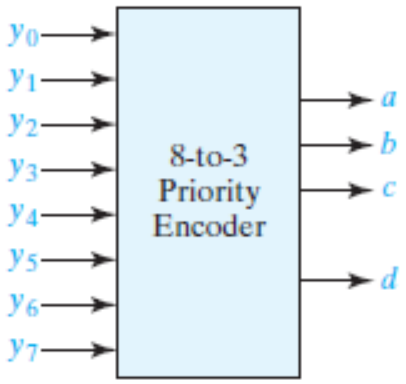


Decoders and Encoders

Encoders:

Encoders have the inverse function of decoders.

FIGURE 9-20
An 8-to-3 Priority Encoder



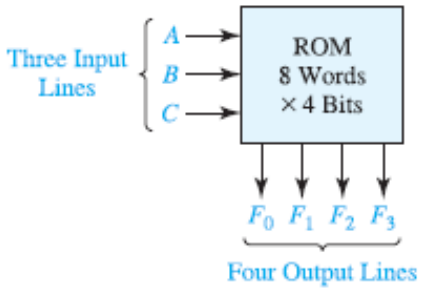
y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	a	b	c	d
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
X	1	0	0	0	0	0	0	0	0	1	1
X	X	1	0	0	0	0	0	0	1	0	1
X	X	X	1	0	0	0	0	0	1	1	1
X	X	X	X	1	0	0	0	1	0	0	1
X	X	X	X	X	1	0	0	1	0	1	1
X	X	X	X	X	X	1	0	1	1	0	1
X	X	X	X	X	X	X	1	1	1	1	1

Read-Only Memories

Read-Only Memory:

- 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.

FIGURE 9-21
An 8-Word × 4-Bit ROM



(a) Block diagram

A B C	F_0 F_1 F_2 F_3
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)

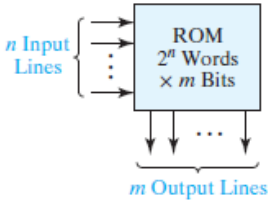
(b) Truth table for ROM

Read-Only Memories

ROM with n-inputs and m-outputs:

- A ROM which has *n* input lines and *m* output lines (Figure 9-22) contains an array of 2^n words, and each word is *m* bits long. The input lines serve as an address to select one of the 2^n words.
- When an input combination is applied to the ROM, the pattern of 0's and 1's which is stored in the corresponding word in the memory appears at the output lines.
- A $2^n \times m$ ROM can realize *m* functions of *n* variables because it can store a truth table with 2^n rows and *m* columns.

FIGURE 9-22
Read-Only Memory with *n* Inputs and *m* Outputs



n Input Variables	*m* Output Variables
00 · · · 00	100 · · · 110
00 · · · 01	010 · · · 111
00 · · · 10	101 · · · 101
00 · · · 11	110 · · · 010
⋮	⋮
11 · · · 00	001 · · · 011
11 · · · 01	110 · · · 110
11 · · · 10	011 · · · 000
11 · · · 11	111 · · · 101

Typical Data Array Stored in ROM (2^n words of *m* bits each)

Read-Only Memories

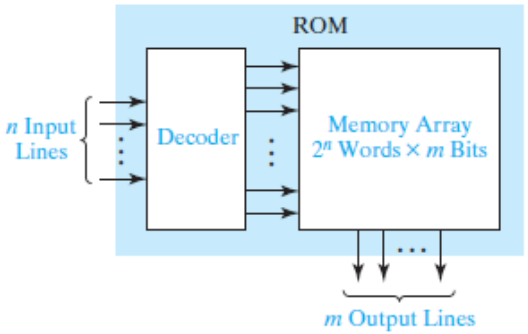
Basic ROM Structure:

A ROM basically consists of a decoder and a memory array, as shown in Figure 9-23.

When a pattern of *n* 0's and 1's is applied to the decoder inputs, exactly one of the 2^n decoder outputs is 1.

This decoder output line selects one of the words in the memory array, and the bit pattern stored in this word is transferred to the memory output lines.

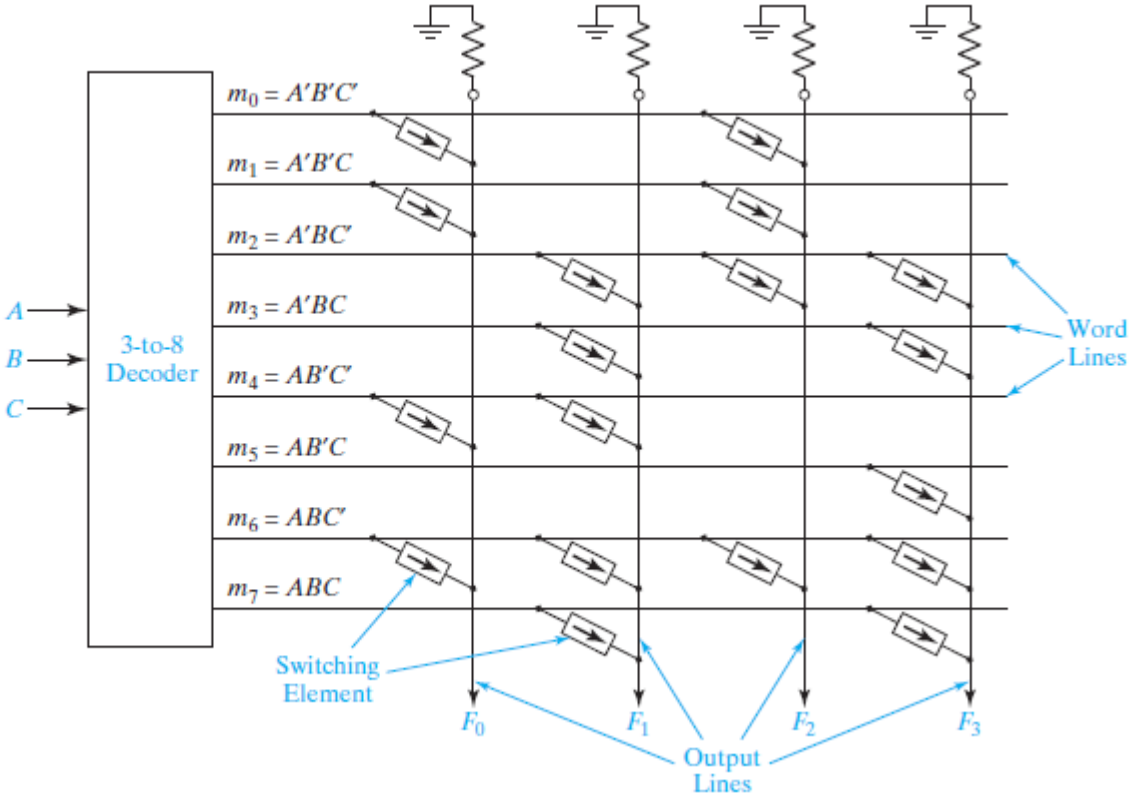
FIGURE 9-23
Basic ROM Structure



Read-Only Memories

8-Word x 4-Bit ROM:

FIGURE 9-24
An 8-Word × 4-Bit ROM



Read-Only Memories

8-Word by 4-Bit ROM Functions:

$$F_0 = \Sigma\, m(0, 1, 4, 6) = A'B' + AC'$$
$$F_1 = \Sigma\, m(2, 3, 4, 6, 7) = B + AC'$$
$$F_2 = \Sigma\, m(0, 1, 2, 6) = A'B' + BC'$$
$$F_3 = \Sigma\, m(2, 3, 5, 6, 7) = AC + B$$

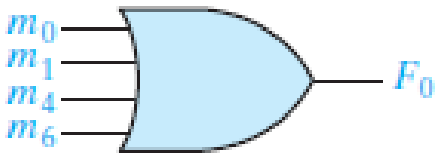


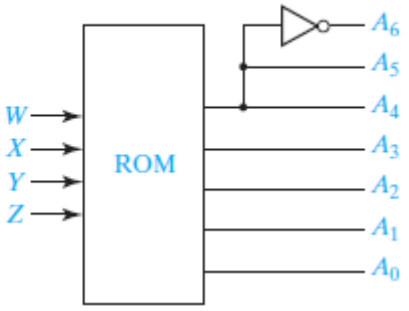
FIGURE 9-25
Equivalent OR Gate for F_0

Read-Only Memories

Hexadecimal-to-ASCII Code Converter:

FIGURE 9-26
Hexadecimal-to-ASCII Code Converter

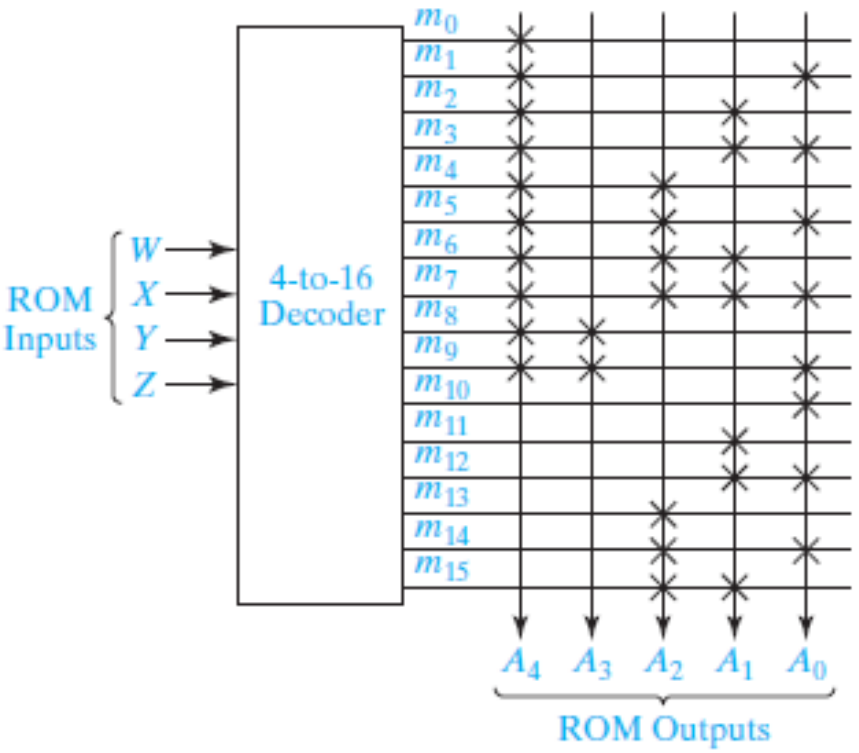
Input W X Y Z	Hex Digit	ASCII Code for Hex Digit A_6 A_5 A_4 A_3 A_2 A_1 A_0
0 0 0 0	0	0 1 1 0 0 0 0
0 0 0 1	1	0 1 1 0 0 0 1
0 0 1 0	2	0 1 1 0 0 1 0
0 0 1 1	3	0 1 1 0 0 1 1
0 1 0 0	4	0 1 1 0 1 0 0
0 1 0 1	5	0 1 1 0 1 0 1
0 1 1 0	6	0 1 1 0 1 1 0
0 1 1 1	7	0 1 1 0 1 1 1
1 0 0 0	8	0 1 1 1 0 0 0
1 0 0 1	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



Read-Only Memories

ROM Realization of Code Converter:

FIGURE 9-27
ROM Realization of Code Converter



Programmable Logic Devices

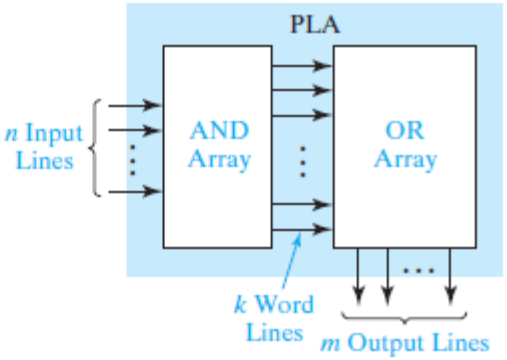
- A **programmable logic device (or PLD)** is a general name for a digital integrated circuit capable of being programmed to provide a variety of different logic functions
- In this chapter we will study:
  - Combinational PLDs
  - Sequential PLDs

Programmable Logic Devices

Programmable Logic Arrays (PLA):

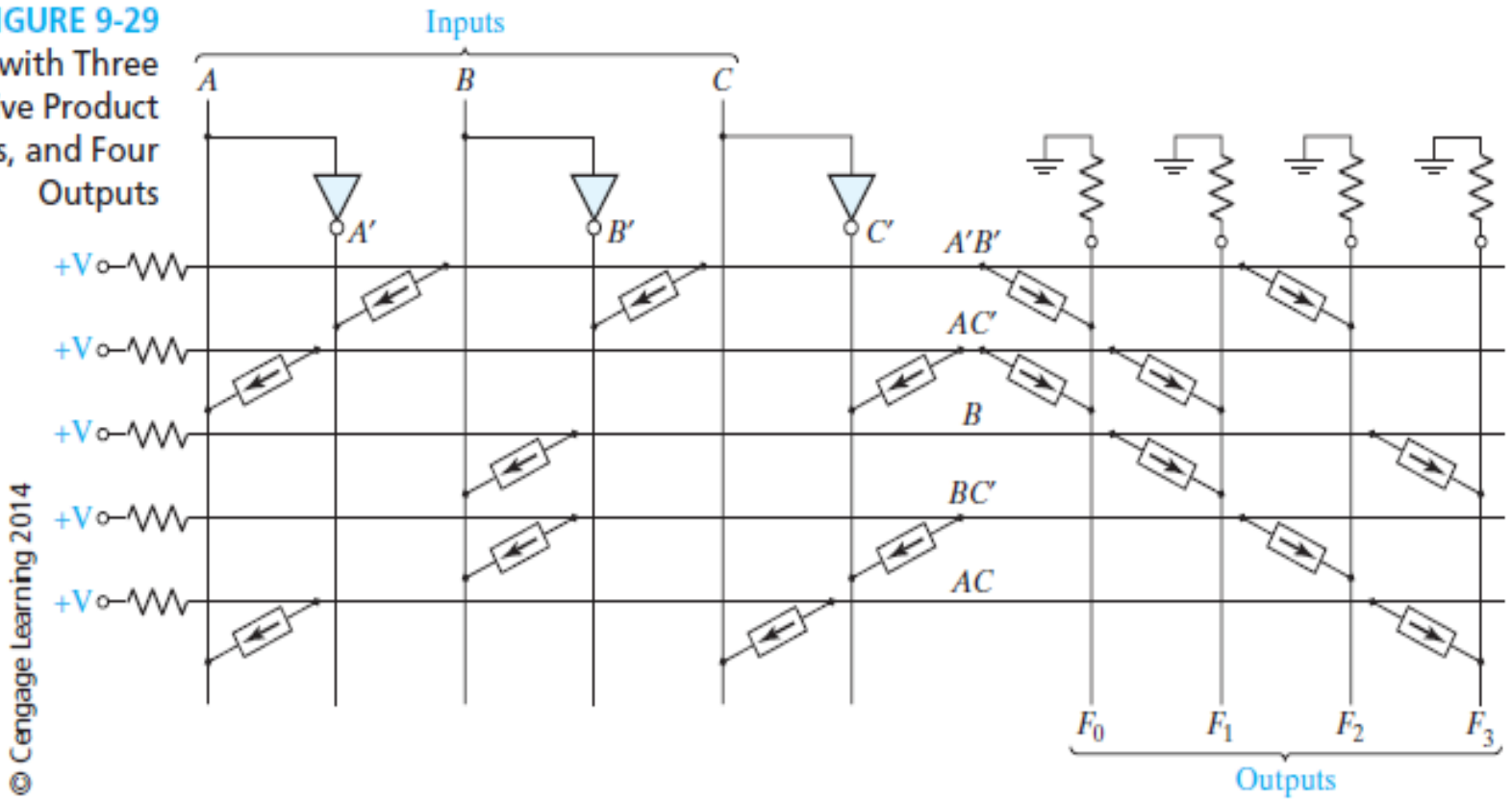
- A PLA performs the same basic function as a ROM.
- A PLA with *n* inputs and *m* outputs (Figure 9-28) can realize *m* functions of *n* variables.

FIGURE 9-28
Programmable Logic Array Structure



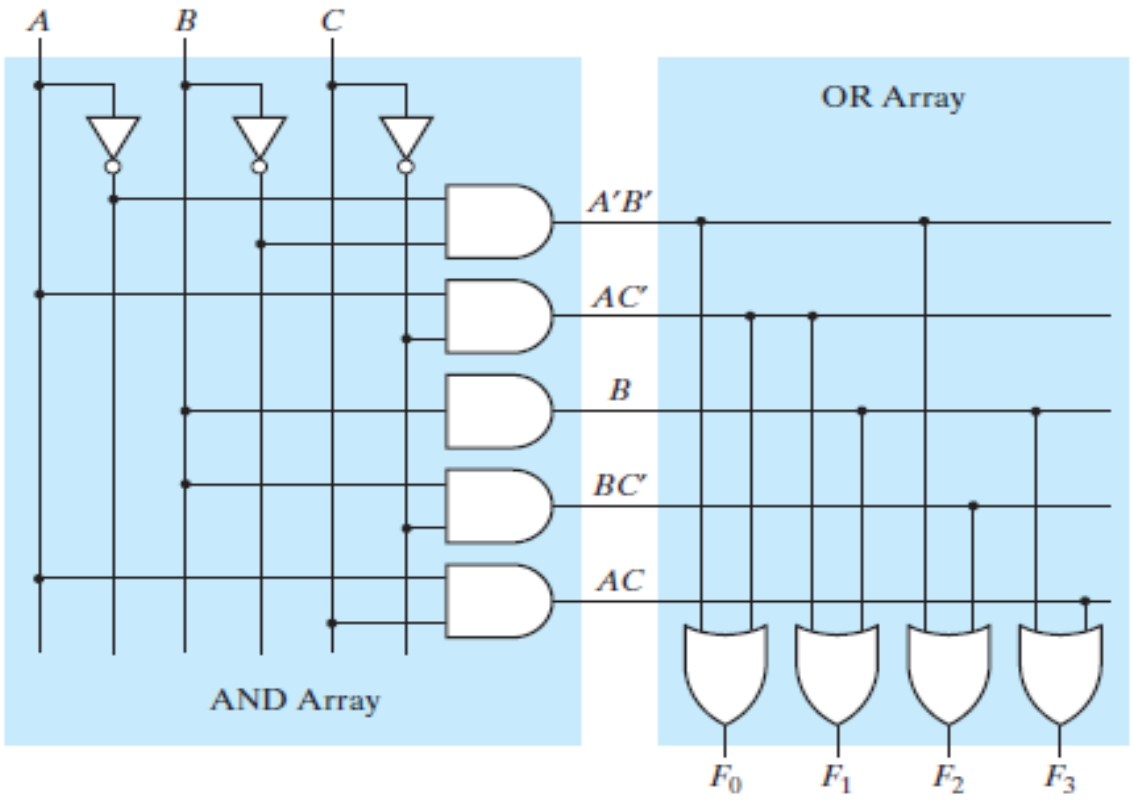
Programmable Logic Devices

FIGURE 9-29
PLA with Three Inputs, Five Product Terms, and Four Outputs



Programmable Logic Devices

FIGURE 9-30
AND-OR Array Equivalent to Figure 9-29



Programmable Logic Devices

PLA Tables:

- The symbols 0, l, and – indicate whether a variable is complemented, not complemented, or not present in the corresponding product term.
- The output side of the table specifies which product terms appear in each output function.
- A 1 or 0 indicates whether a given product term is present or not present in the corresponding output function.

TABLE 9-1
PLA Table for Figure 9-29

Product Term	Inputs A B C	Outputs F_0 F_1 F_2 F_3	
$A'B'$	0 0 –	1 0 1 0	$F_0 = A'B' + AC'$
AC'	1 – 0	1 1 0 0	$F_1 = AC' + B$
B	– 1 –	0 1 0 1	$F_2 = A'B' + BC'$
BC'	– 1 0	0 0 1 0	$F_3 = B + AC$
AC	1 – 1	0 0 0 1	

Programmable Logic Devices

PLA Tables and ROM Truth Tables:

- In a ROM truth table each row represents a minterm; therefore, exactly one row will be selected by each combination of input values.
- The 0's and 1's of the output portion of the selected row determine the corresponding output values.
- On the other hand, each row in a PLA table represents a general product term. Therefore, zero, one, or more rows may be selected by each combination of input values.
- To determine the value of *fi* for a given input combination, the values of *fi* in the selected rows of the PLA table must be ORed together.

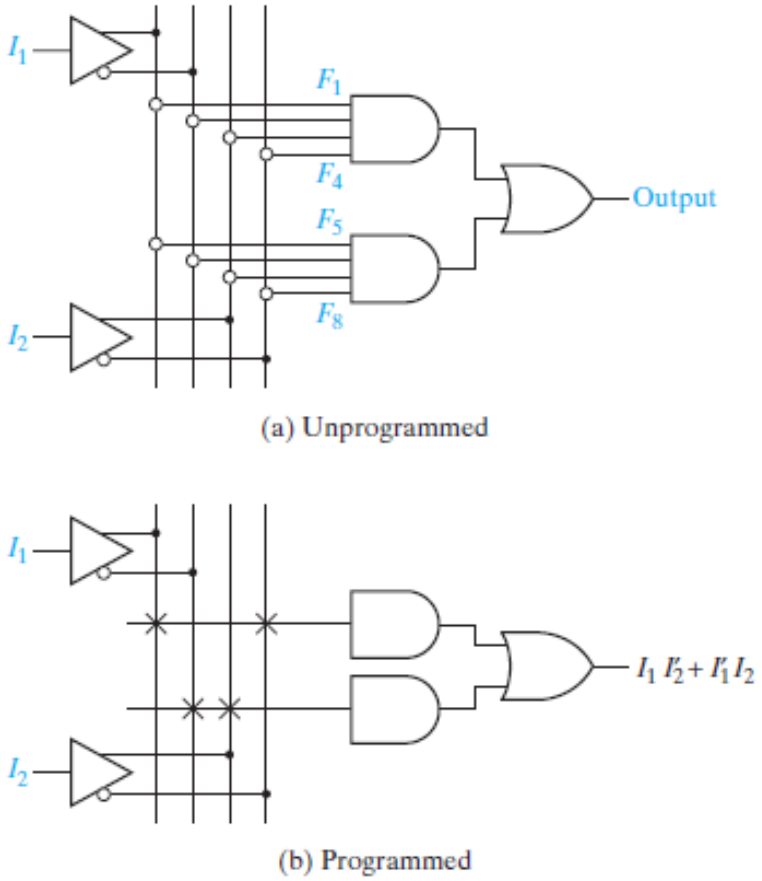
Programmable Logic Devices

Programmable Array Logic (PAL):

- The **PAL (programmable array logic)** is a special case of the programmable logic array in which the AND array is programmable and the OR array is fixed
- PAL has same structure as PLA in Figure 9-28.

Programmable Logic Devices

FIGURE 9-32
PAL Segment



(a) Unprogrammed

(b) Programmed

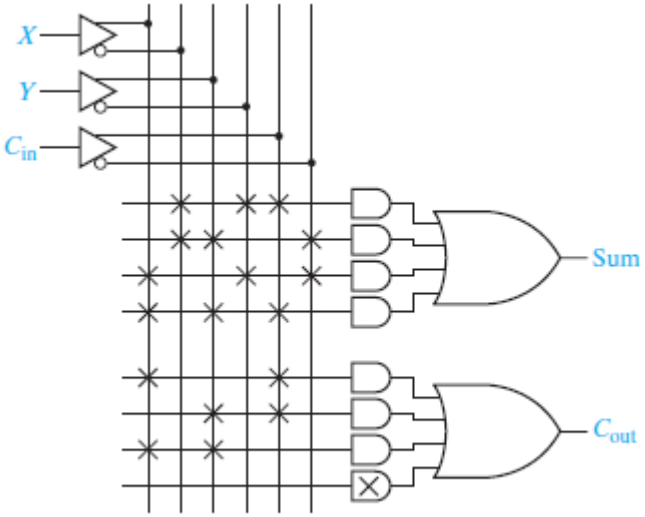
Programmable Logic Devices

Implementing a Full Adder Using PAL:

$$Sum = X'Y'C_{in} + X'YC'_{in} + XY'C'_{in} + XYC_{in}$$

$$C_{out} = XC_{in} + YC_{in} + XY$$

FIGURE 9-33
Implementation of a Full Adder Using a PAL

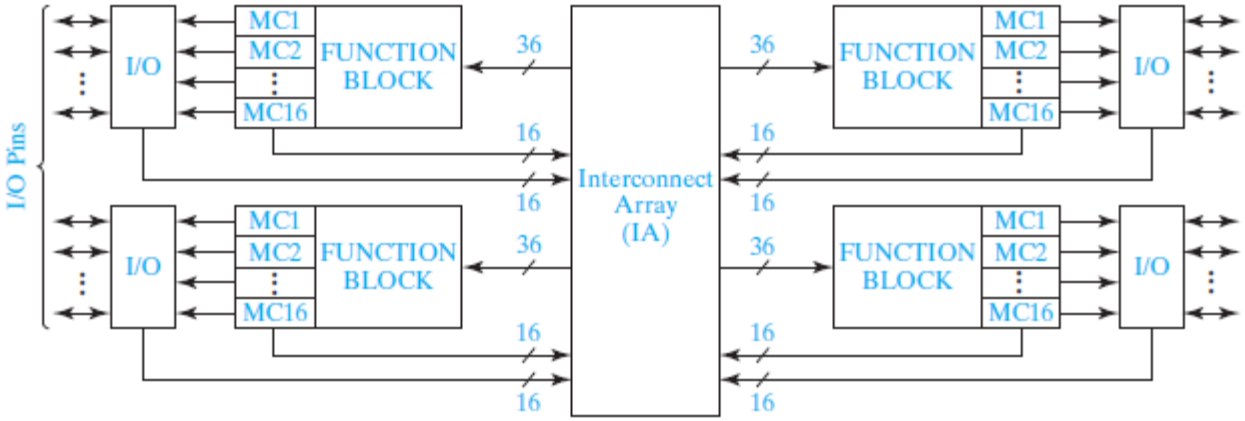


Complex Programmable Logic Devices

Complex Programmable Logic Devices (CPLD):

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

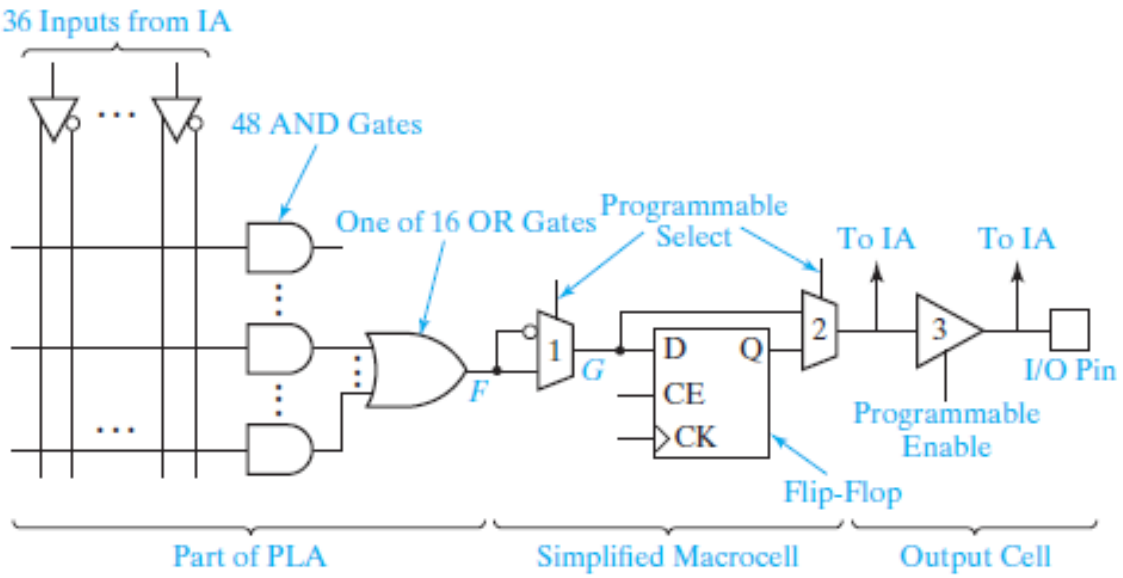
FIGURE 9-34 Architecture of Xilinx XCR3064XL CPLD (Figure based on figures and text owned by Xilinx, Inc., Courtesy of Xilinx, Inc. © Xilinx, Inc. 1999–2003. All rights reserved.)



Complex Programmable Logic Devices

- Figure 9-35 shows how a signal generated in the PLA is routed to an I/O pin through a macrocell.

FIGURE 9-35
CPLD Function Block and Macrocell (A Simplified Version of XCR3064XL)



Field-Programmable Arrays

Field-Programmable Gate Arrays (FPGA):

- An **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 consists of an array of logic cells, also called configurable logic blocks (CLBs).
- The array of CLBs is surrounded by a ring of input-output interface blocks. These I/O blocks connect the CLB signals to IC pins.
- The space between the CLBs is used to route connections between the CLB outputs and inputs.

Field-Programmable Arrays

FPGAs and CLBs:

FIGURE 9-36
Layout of a Typical FPGA

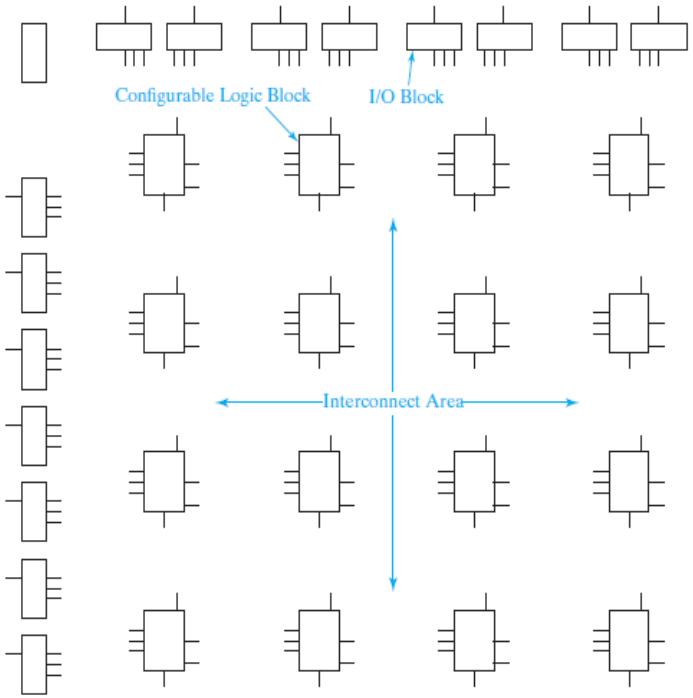
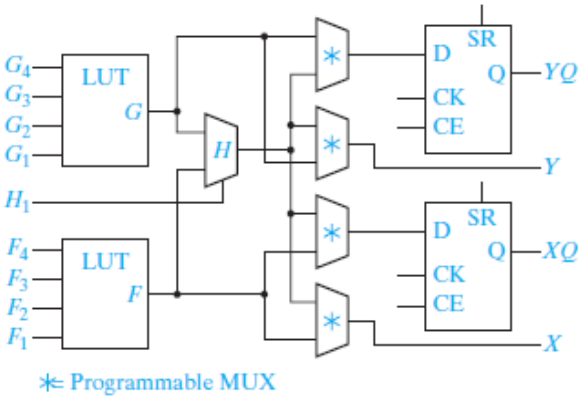


FIGURE 9-37
Simplified Configurable Logic Block (CLB)



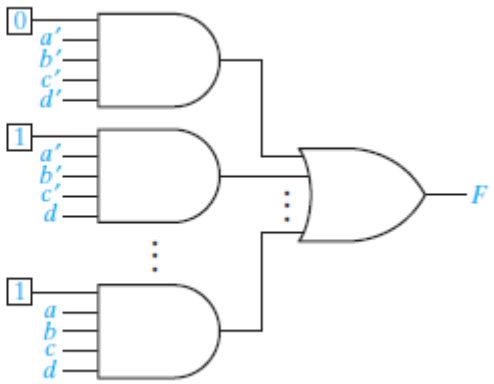
Field-Programmable Arrays

Implementing a Function Generator with Inputs a,b,c,d:

- The numbers in the squares represent the bits stored in the LUT. These bits enable particular minterms.
- Because the function being implemented is stored as a truth table, a function with only one minterm or with as many as 15 minterms requires a single function generator.

FIGURE 9-38
Implementation of a Lookup Table (LUT)

a	b	c	d	F
0	0	0	0	0
0	0	0	1	1
⋮				⋮
1	1	1	1	1



Field-Programmable Arrays

Decomposition of Switching Functions:

- One method of decomposition is **Shannon's Expansion Theorem**.

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

$$f(x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n)$$
$$= x_i' f(x_1, x_2, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) + x_i f(x_1, x_2, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$$
$$= x_i' f_0 + x_i f_1 \qquad (9\text{-}10)$$

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.

Field-Programmable Arrays

Realization of 5- and 6- Variable Functions:

$$f(a, b, c, d, e) = a'\, f(0, b, c, d, e) + a\, f(1, b, c, d, e) = a'\, f_0 + a\, f_1 \qquad (9\text{-}11)$$

$$G(a, b, c, d, e, f) = a'b'G_{00} + a'b\, G_{01} + ab'G_{10} + ab\, G_{11} \qquad (9\text{-}12)$$

FIGURE 9-40
Realization of 5- and 6-Variable Functions with Function Generators