

# CS204: 數位系統設計

## Combinational Logic

---

# Outline of Chapter 4

---

- ▣ 4.1 Introduction
- ▣ 4.2 Combinational Circuits
- ▣ 4.3 Analysis Procedure
- ▣ 4.4 Design Procedure
- ▣ 4.5 Binary Adder-Subtractor
- ▣ 4.6 Decimal Adder
- ▣ 4.7 Binary Multiplier
- ▣ 4.8 Magnitude Comparator
- ▣ 4.9 Decoders
- ▣ 4.10 Encoders
- ▣ 4.11 Multiplexers

# 4.1 Introduction (p.141)

---

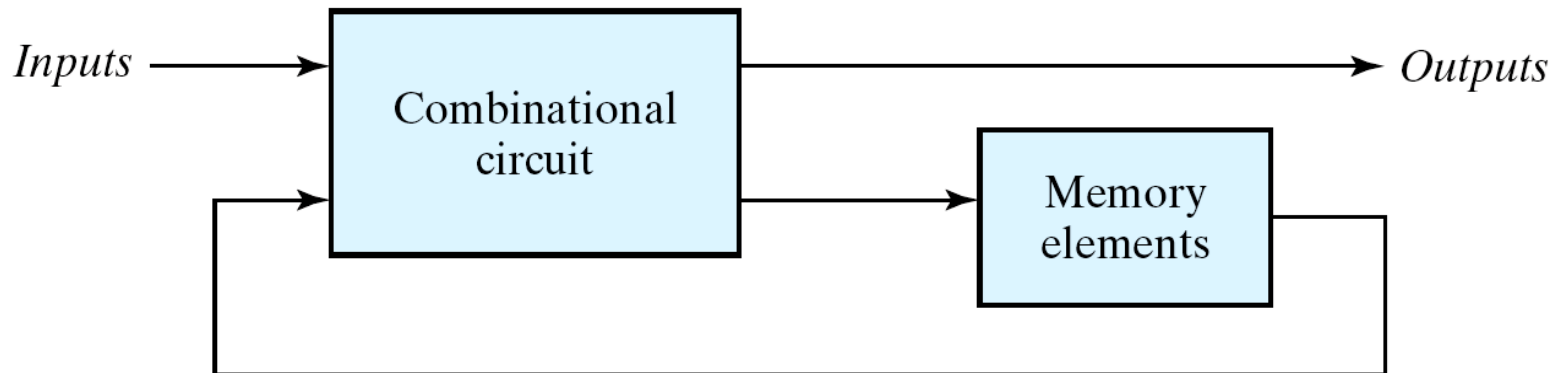
- ▣ Logic circuits for digital systems may be combinational or sequential
- ▣ A combinational circuit consists of logic gates whose **outputs** at any time are determined from only the present **combination of inputs**

# 4.2 Combinational Circuits (p.141)

## ▣ Logic circuits for digital system

### ◆ Sequential circuits

- » Contain memory elements
- » The outputs are a function of the current inputs and the state of the memory elements
- » The outputs also depend on past inputs



# Combinational Circuits (p.142)

## ■ A combinational circuits

- ◆  $2^n$  possible combinations of input values

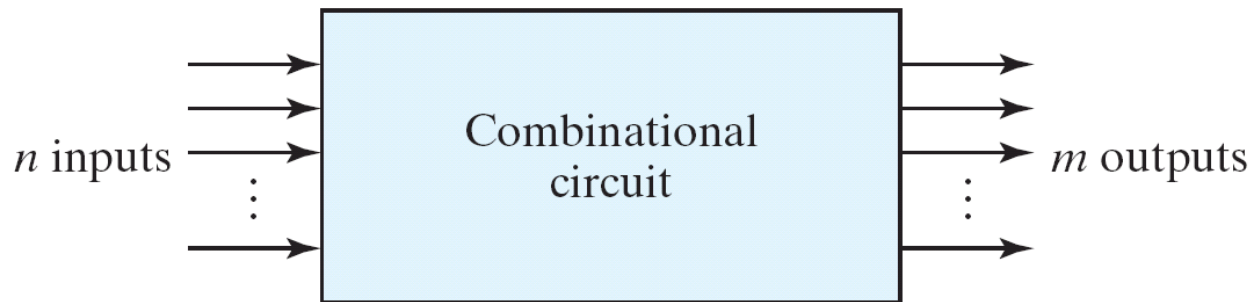


Figure 4.1 Block diagram of combinational circuit

## ◆ Specific functions

- » Adders, subtractors, comparators, decoders, encoders, and multiplexers

# 4-3 Analysis Procedure (p.143)

- Determine the function that the circuit implements
- Given a logic diagram → a set of Boolean functions, a truth table, or possibly an explanation of the circuit operation
- Steps
  - ◆ Make sure that it is combinational not sequential
    - » No feedback path
  - ◆ Derive its Boolean functions (or truth table)
  - ◆ A verbal explanation of its function

# A Straight-forward Procedure (p.143, 144)

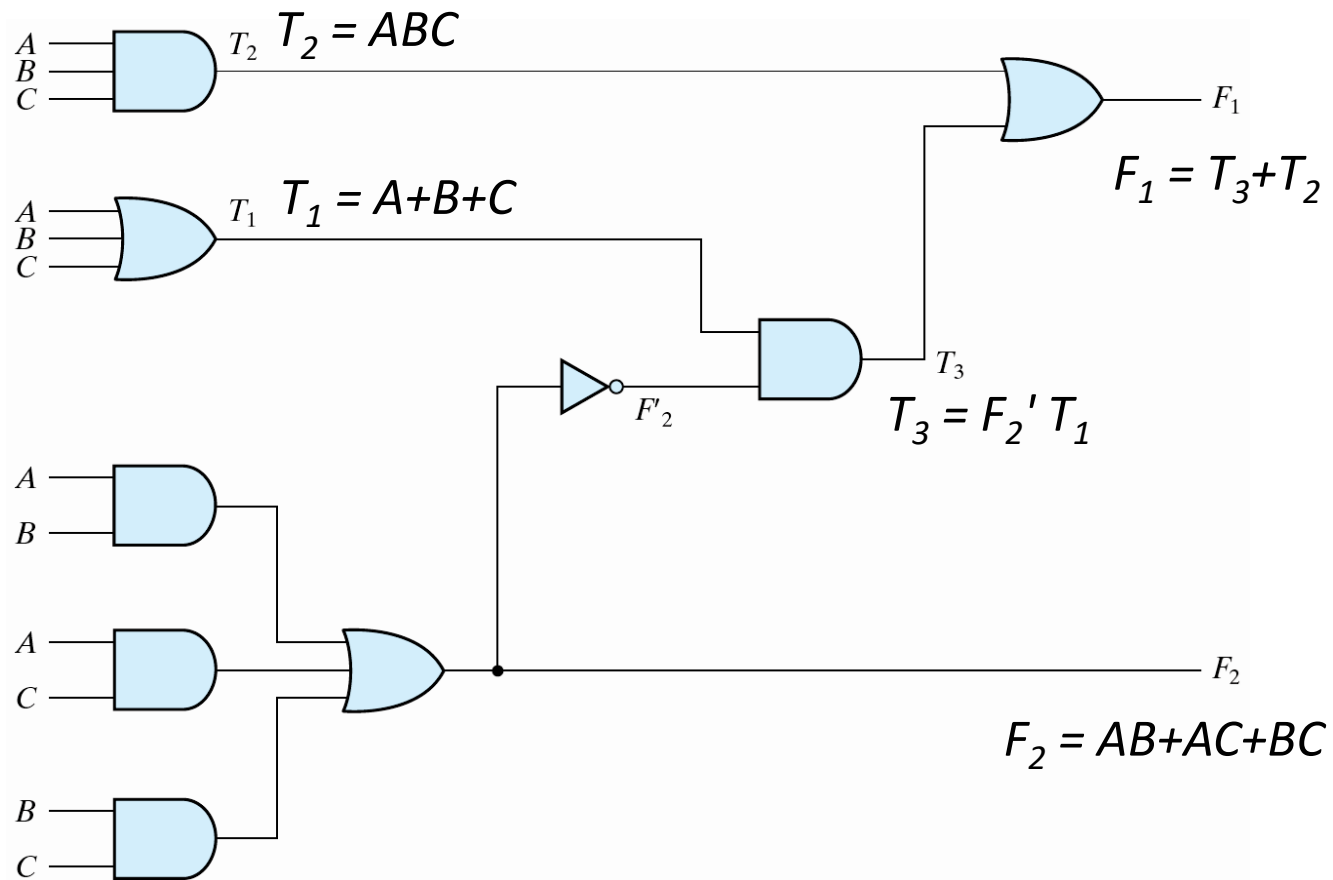
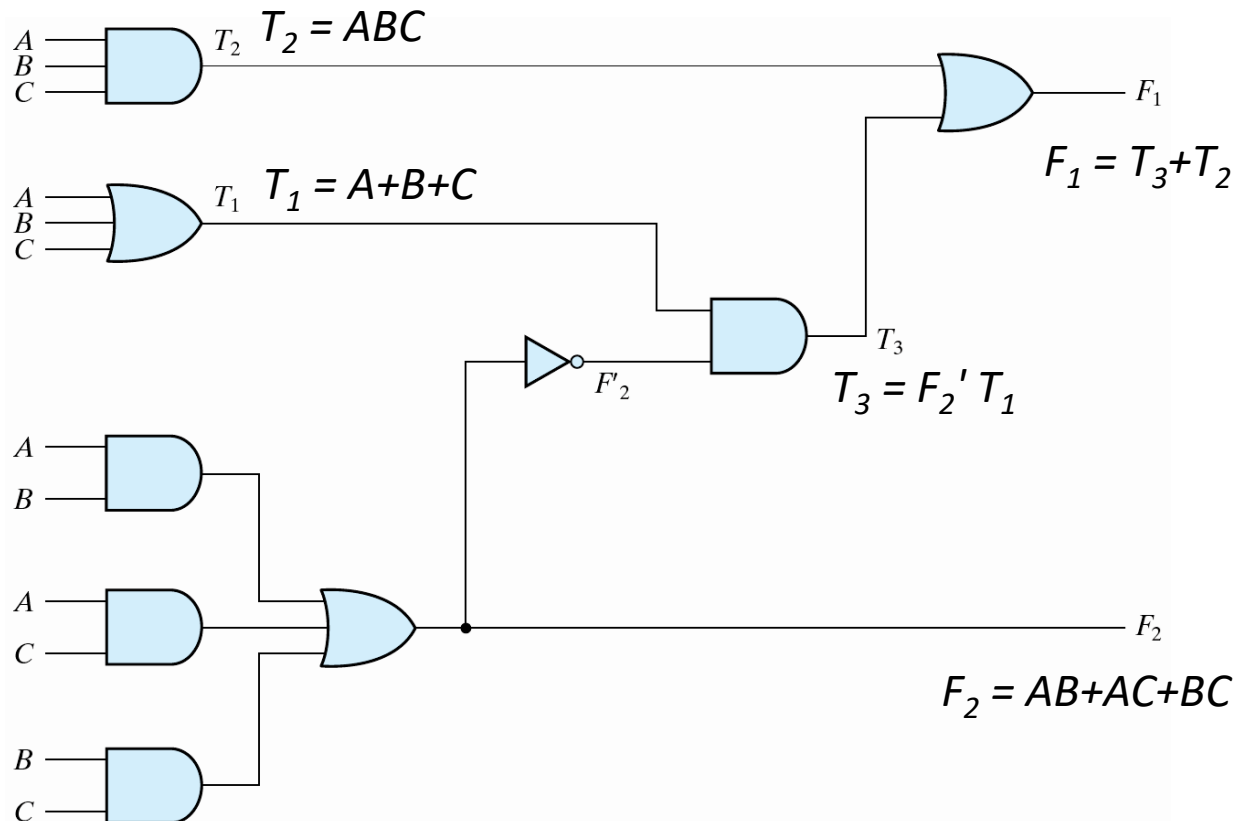


Figure 4.2 Logic Diagram for Analysis Example

# A Straight-forward Procedure (p.143, 144)

■ 
$$F_1 = T_3 + T_2 = F_2' T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC = (A' + B')(A' + C')(B' + C')(A + B + C) + ABC = (A' + B'C')(AB' + AC' + BC' + B'C) + ABC = A'BC' + A'B'C + AB'C' + ABC$$





# Truth Table for Logic Diagram (p.145)

## ■ The truth table

**Table 4.1**

*Truth Table for the Logic Diagram of Fig. 4.2*

<b>A</b>	<b>B</b>	<b>C</b>	<b>F<sub>2</sub></b>	<b>F'<sub>2</sub></b>	<b>T<sub>1</sub></b>	<b>T<sub>2</sub></b>	<b>T<sub>3</sub></b>	<b>F<sub>1</sub></b>
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

# 4-4 Design Procedure (p.145)

- Specification of the design objective → a logic circuit diagram of a set of Boolean functions
- The design procedure of combinational circuits
  - ◆ State the problem (system specification (spec.))
  - ◆ Determine the inputs and outputs
  - ◆ The input and output variables are assigned symbols
  - ◆ Derive the **truth table**
  - ◆ Derive the simplified Boolean functions
  - ◆ Draw the logic diagram and verify the correctness

# Design Procedure (p.146)

## □ Functional description

- ◆ Boolean function
- ◆ HDL (Hardware description language)
  - » Verilog HDL
  - » VHDL
- ◆ Schematic entry

## □ Logic minimization

- ◆ Number of gates
- ◆ Number of inputs to a gate
- ◆ Propagation delay
- ◆ Number of interconnection
- ◆ Limitations of the driving capabilities

# Code Conversion Example (p.146, 147)

## ■ BCD to excess-3 code

### ◆ The truth table

**Table 4.2**

*Truth Table for Code-Conversion Example*

Input BCD				Output Excess-3 Code			
<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>w</i>	<i>x</i>	<i>y</i>	<i>z</i>
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

# BCD Maps (p.148)

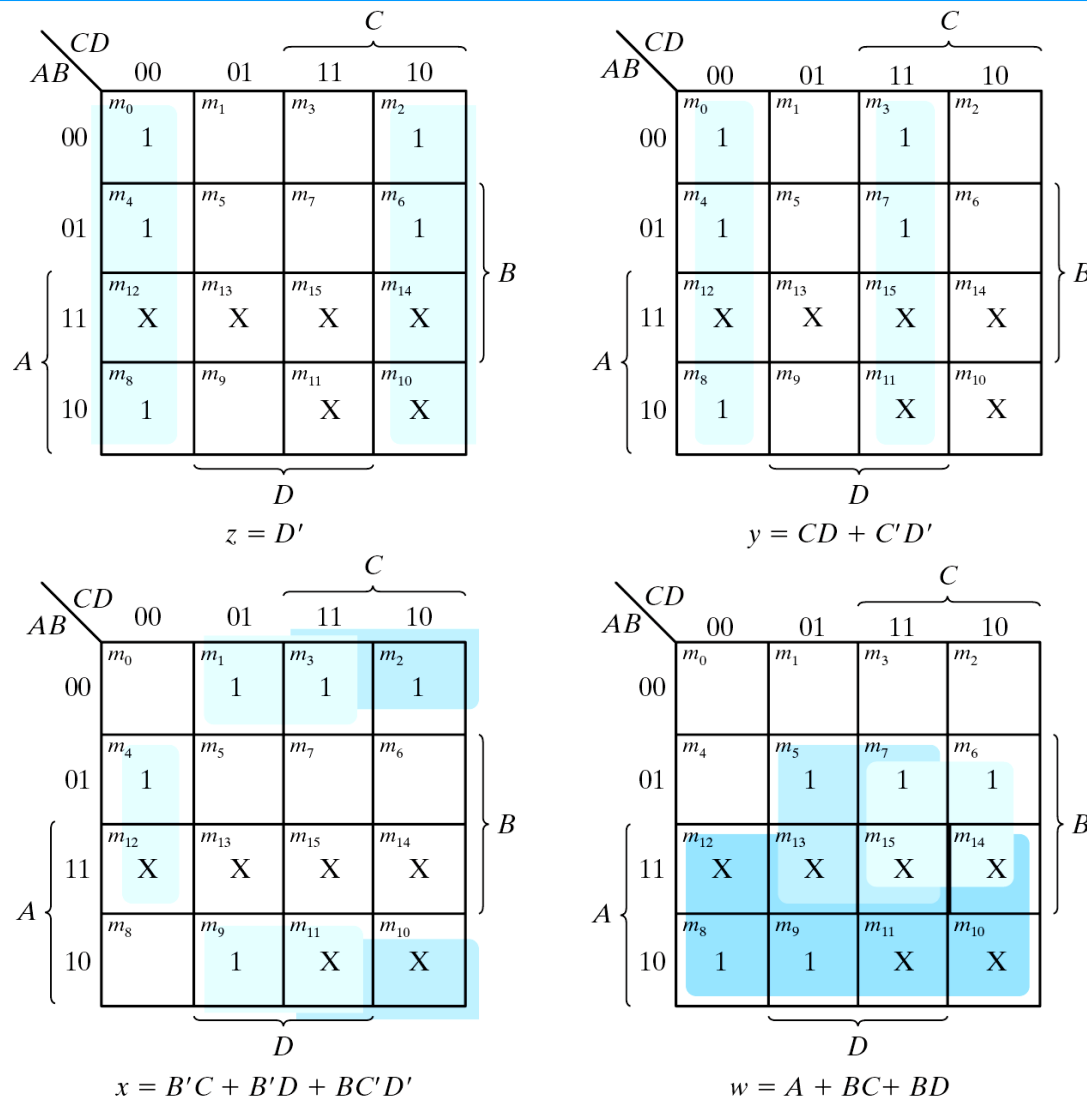


Figure 4.3 Maps for BCE to Excess-3 Code Converter

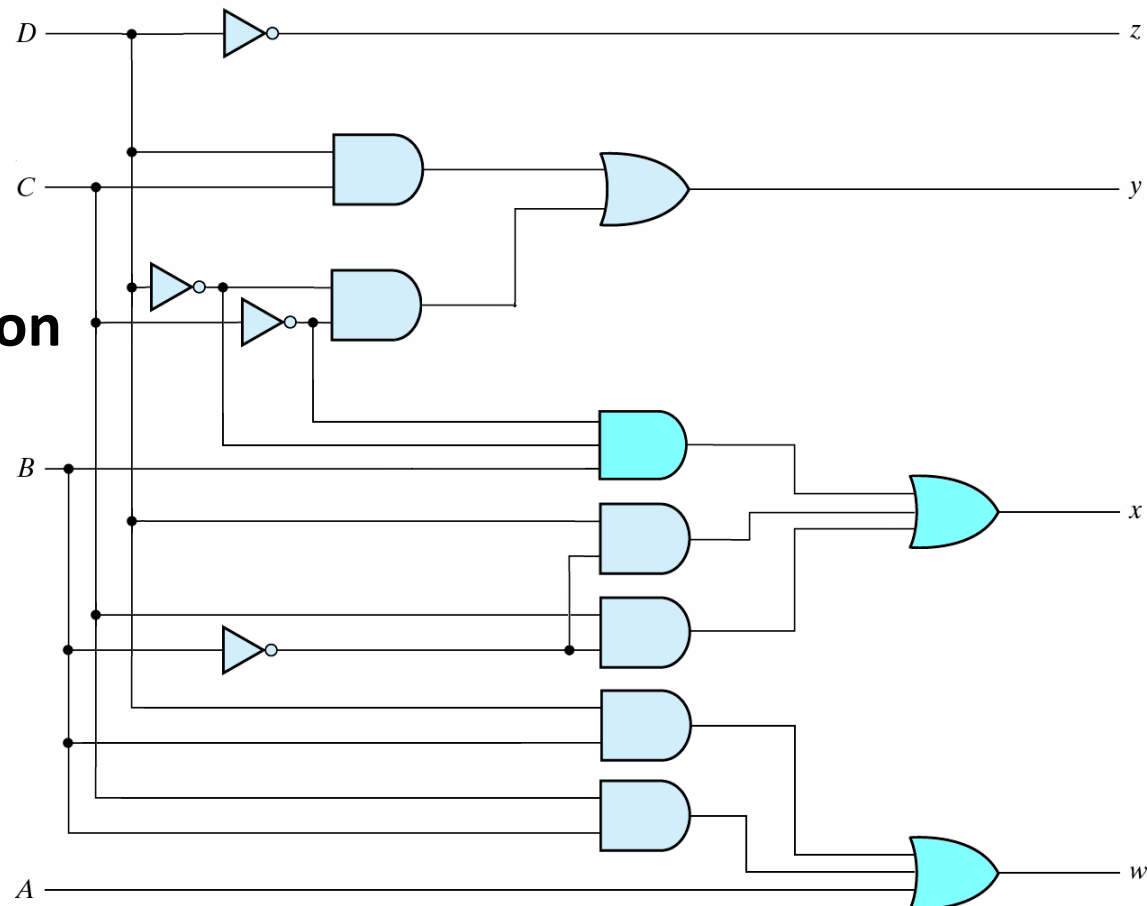
# BCD Functions (p.148)

## ■ The simplified functions

- ◆  $z = D'$
- ◆  $y = CD + C'D'$
- ◆  $x = B'C + B'D + BC'D'$
- ◆  $w = A + BC + BD$

## ■ Another implementation

- ◆  $z = D'$
- ◆  $y = CD + (C + D)'$
- ◆  $x = B'(C + D) + B(C + D)'$
- ◆  $w = A + B(C + D)$



# BCD to Excess-3 (p.149)

- The logic diagram (**fewer gates, no more than two inputs**)

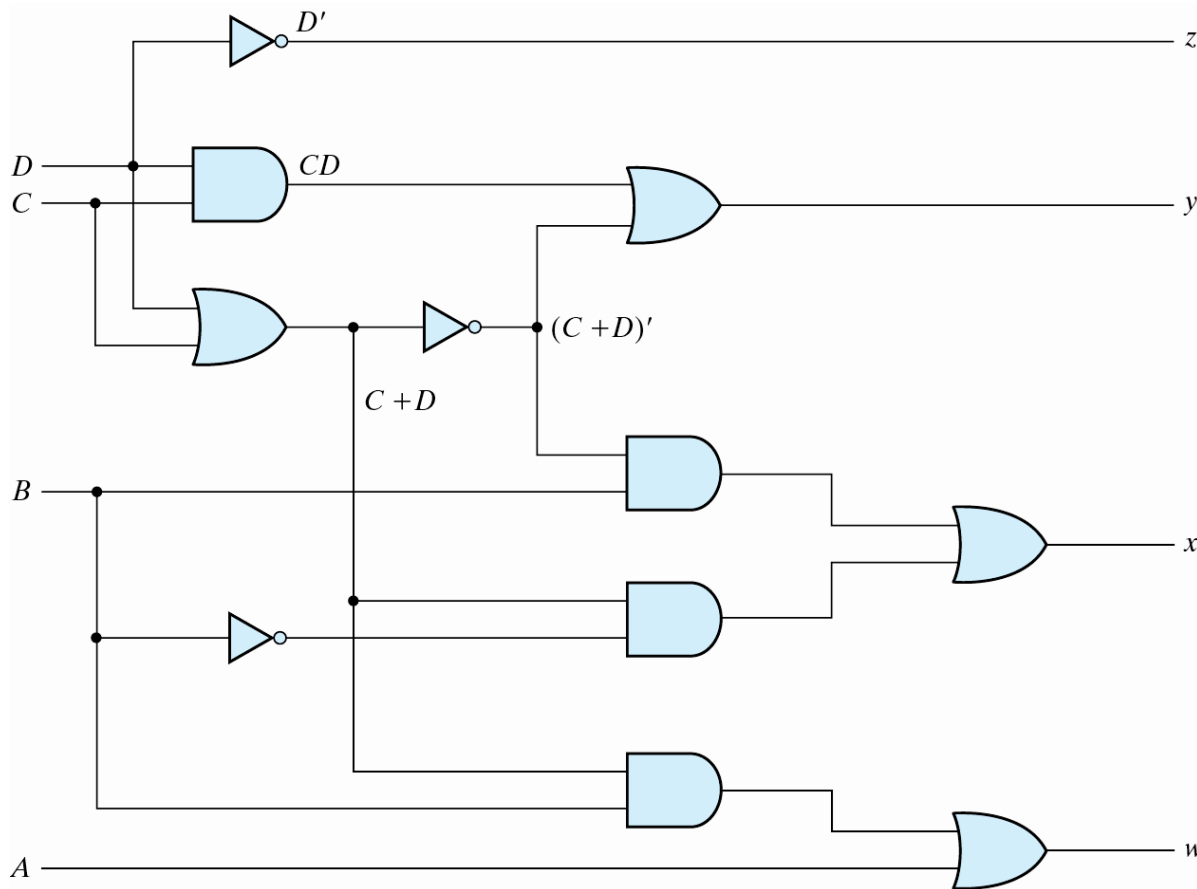


Fig. 4-4 Logic Diagram for BCD to Excess-3 Code Converter

# 4-5 Binary Adder-Subtractor (p.149, 150)

## ■ Half adder

- ◆  $0 + 0 = 0$  ;  $0 + 1 = 1$  ;  $1 + 0 = 1$  ;  $1 + 1 = 10$
- ◆ Two input variables:  $x$ ,  $y$
- ◆ Two output variables:  $C$  (carry),  $S$  (sum)
- ◆ Truth table

**Table 4.3**  
*Half Adder*

$x$	$y$	$C$	$S$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



# Half Adder (p.150)

## ■ Simplified sum-of-products

◆  $S = x'y + xy'$

◆  $C = xy$

## ■ The flexibility for implementation

◆  $S = x \oplus y$

◆  $S = (x + y)(x' + y')$

◆  $S' = xy + x'y'$

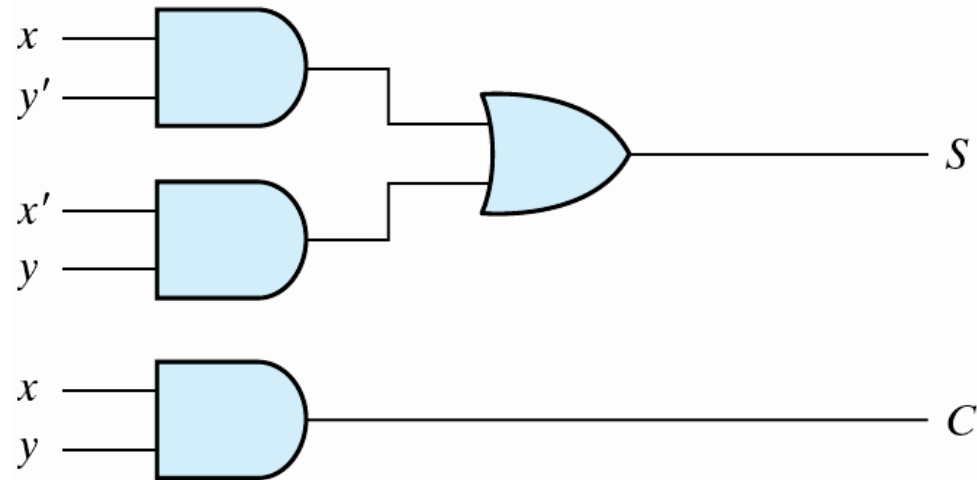
◆  $S = (C + x'y')'$

◆  $C = xy = (x' + y')'$

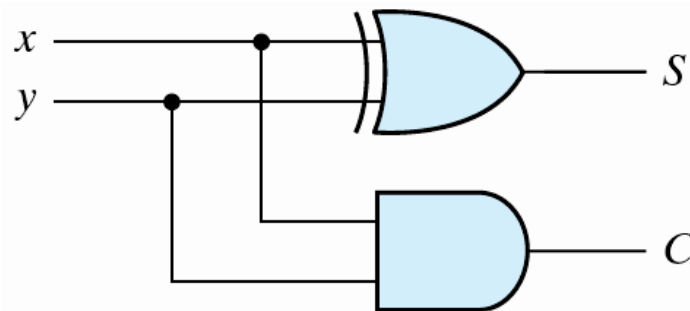
**Table 4.3**  
*Half Adder*

<b>x</b>	<b>y</b>	<b>C</b>	<b>S</b>
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

# Implementation of Half-Adder (p.150)



$$(a) \begin{aligned} S &= xy' + x'y \\ C &= xy \end{aligned}$$



$$(b) \begin{aligned} S &= x \oplus y \\ C &= xy \end{aligned}$$

Figure 4.5 Implementation of Half-Adder

# Full-Adder (p.151)

## ■ Full-Adder

- ◆ The arithmetic sum of three input bits
- ◆ Three input bits
  - »  $x, y$ : two significant bits
  - »  $z$ : the carry bit from the previous lower significant bit
- ◆ Two output bits:  $C, S$

**Table 4.4**  
*Full Adder*

$x$	$y$	$z$	$C$	$S$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

# Full-Adder (p.152)

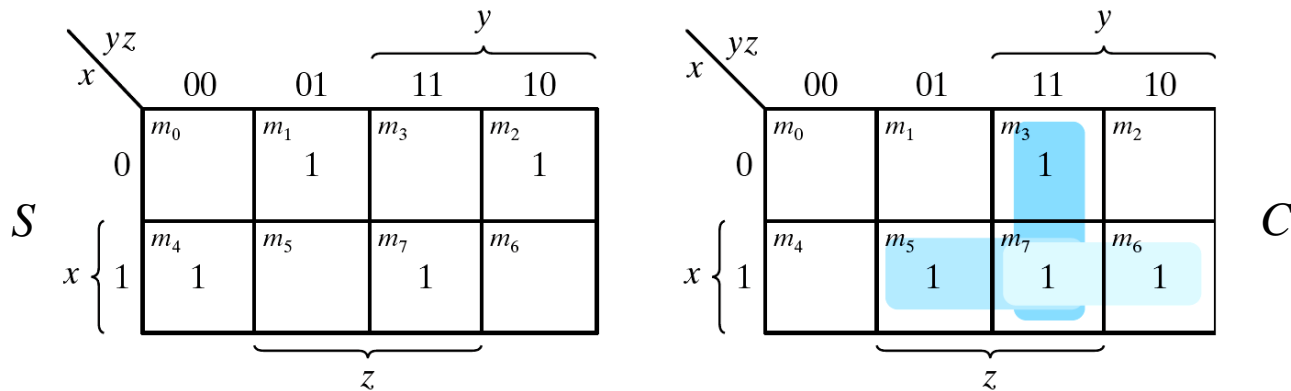


Fig. 4-6 Map for Full Adder

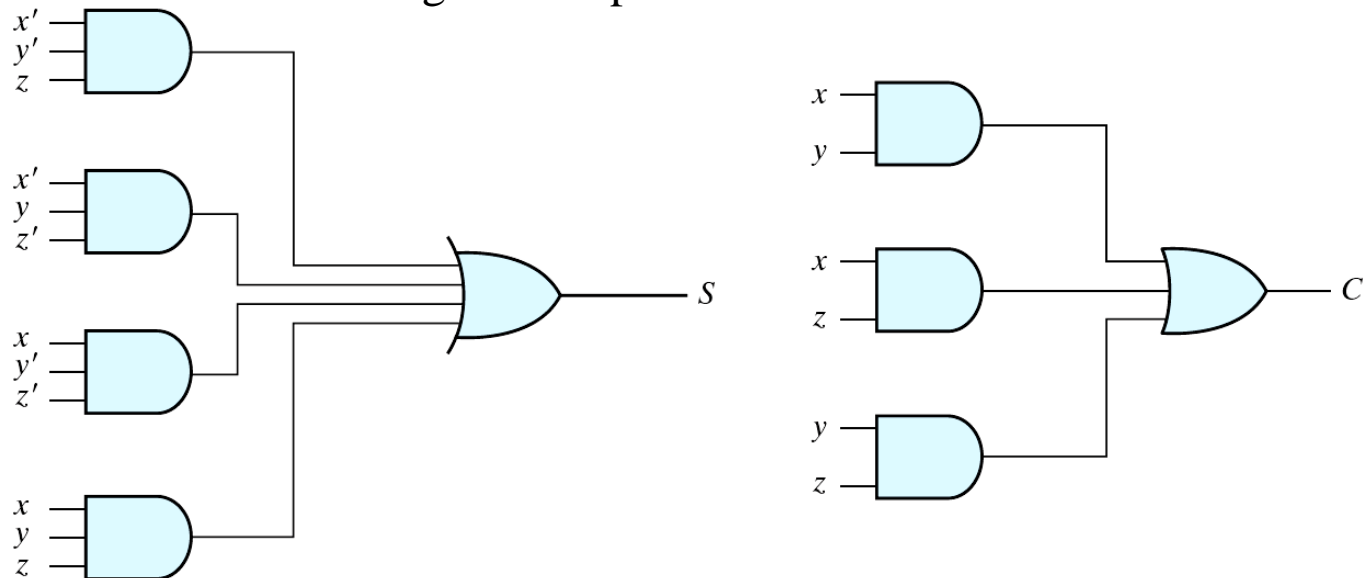


Fig. 4-7 Implementation of Full Adder in Sum of Products

# Full-Adder (p.152, 153)

## ■ Simplified sum-of-products

◆  $S = x'y'z + x'yz' + xy'z' + xyz$

◆  $C = xy + xz + yz$

## ■ Full-adder by using 2 half-adder

◆  $S = z \oplus (x \oplus y) = z'(xy' + x'y) + z(xy' + x'y)' = z'xy' + z'x'y + z((x' + y)(x + y')) = xy'z' + x'yz' + xyz + x'y'z$

◆  $C = z(x \oplus y) + xy = z(xy' + x'y) + xy = xy'z + x'yz + xy = xy'z + x'yz + xy(z+1)$

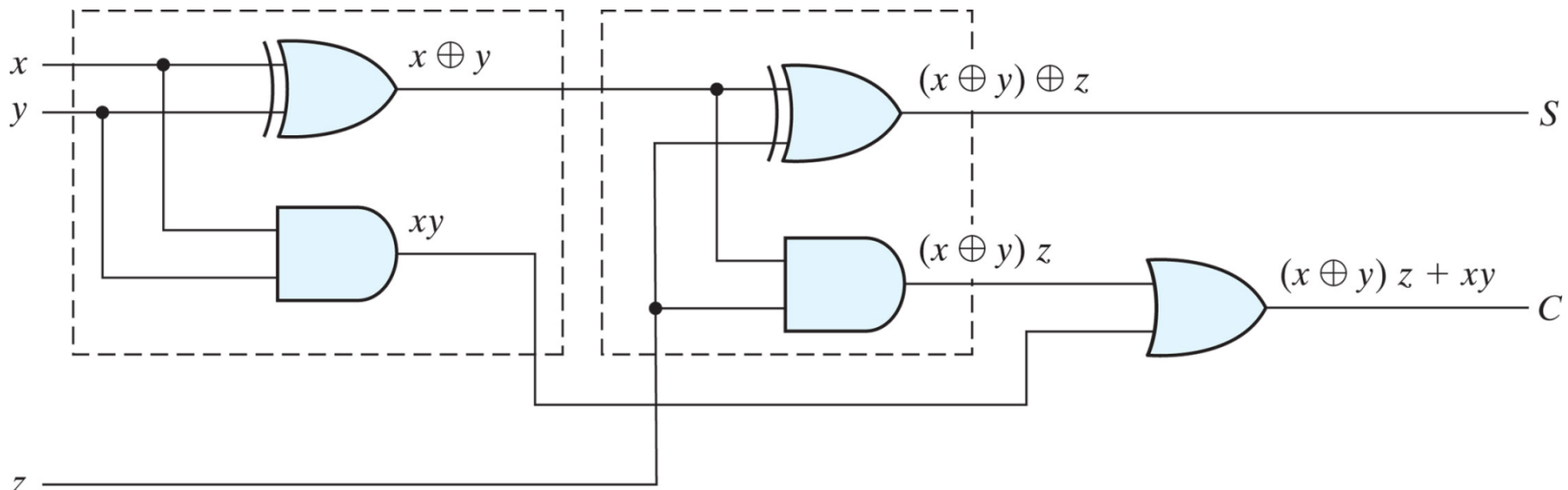


Fig. 4-8 Implementation of Full Adder with Two Half Adders and an OR Gate

# Binary (Ripple-Carry) Adder (p.153, 154)

Subscript i:	3	2	1	0	
Input carry	0	1	1	0	$C_i$
Augend	1	0	1	1	$A_i$
Addend	0	0	1	1	$B_i$
Sum	1	1	1	0	$S_i$
Output carry	0	0	1	1	$C_{i+1}$

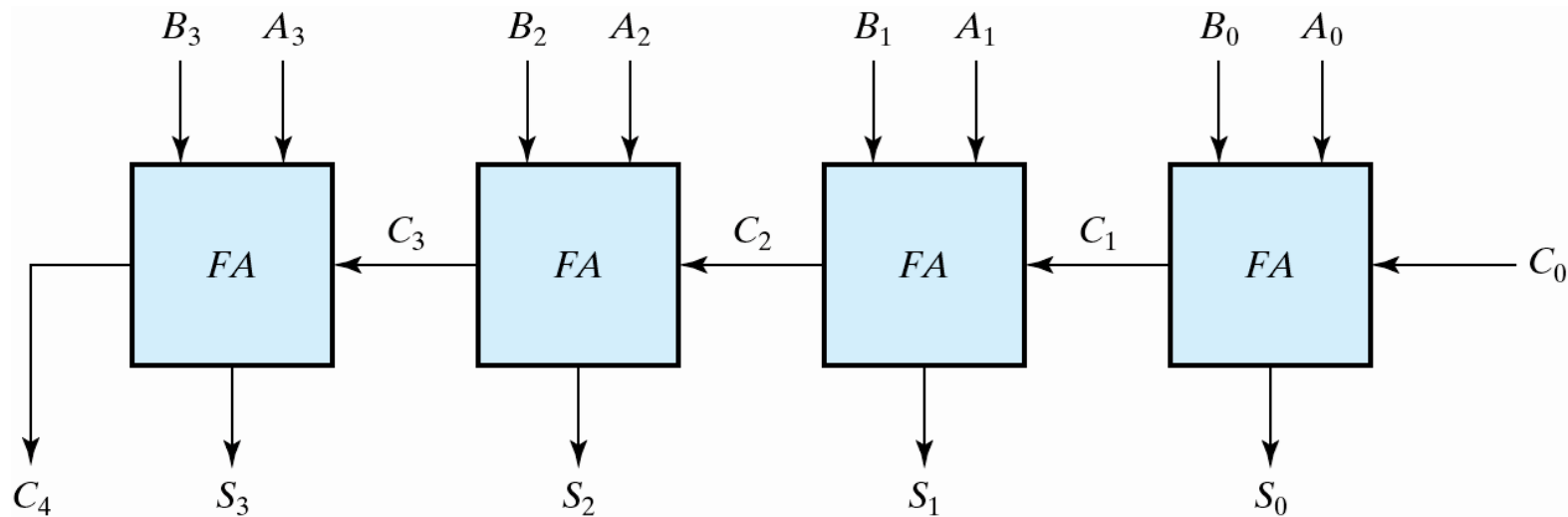


Figure 4.9 Full-bit adder

# Carry Propagation (p.154, 155)

## ■ Carry propagation

- ◆ Total propagation time = gate delay \* number of gate levels
- ◆ The **critical path** (the longest propagation)
- ◆  $(A_0, B_0, C_0) \rightarrow C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow (C_4, S_3)$
- ◆ 4-bits full-adder  $\rightarrow$  8 gate levels ( $n$ -bits:  $2n$  gate levels)

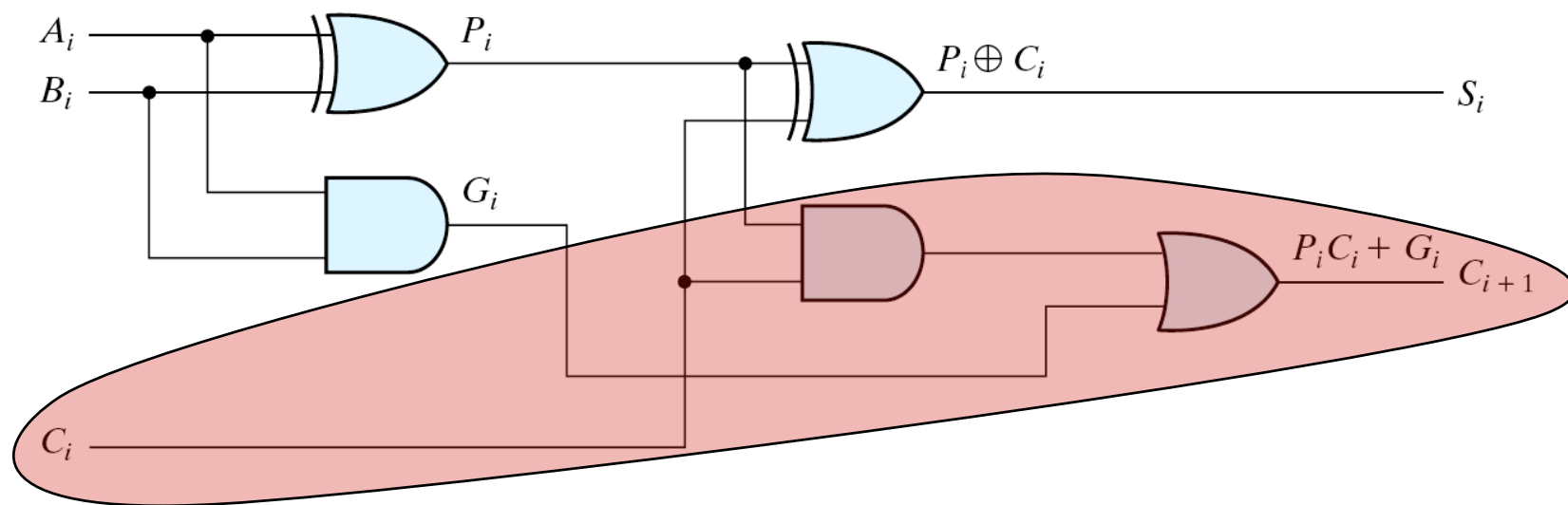


Figure 4.10 Full Adder with  $P$  and  $G$  Shown

# Parallel Adders (p.155, 156)

## ▣ Reduce the carry propagation delay

- ◆ Using faster gates
- ◆ Parallel adders (e.g., **carry look-ahead adder**)

## ▣ Carry look-ahead adder (CLA)

- ◆ Carry propagate:  $P_i = A_i \oplus B_i$
- ◆ Carry generate:  $G_i = A_i B_i$
- ◆ Sum:  $S_i = P_i \oplus C_i$
- ◆ Carry:  **$C_{i+1} = G_i + P_i C_i$**
- ◆  $C_0$  = input carry
- ◆  $C_1 = G_0 + P_0 C_0$
- ◆  $C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$
- ◆  $C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$



# Carry Look-ahead Adder (1/2) (p.156)

## ■ Logic diagram of carry-look-ahead generator

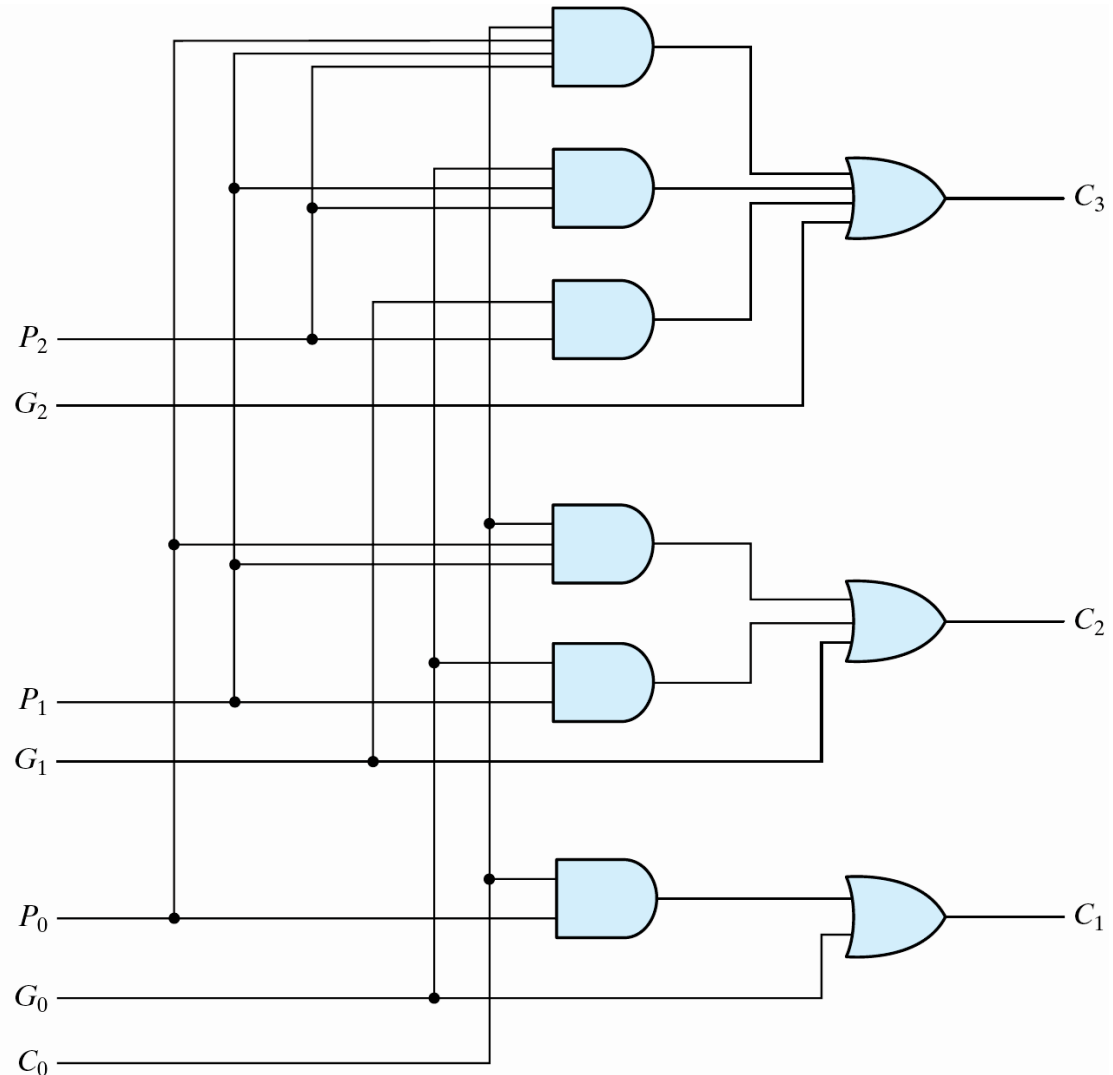


Fig. 4.11 Logic Diagram of Carry Look-ahead Generator

# Carry Look-ahead Adder (2/2) (p.157)

## 4-bit carry-look ahead adder

- ◆ Propagation delay of  $C_3$ ,  $C_2$  and  $C_1$  are equal
- ◆ Propagation delay of  $S_3$ ,  $S_2$  and  $S_1$  are equal

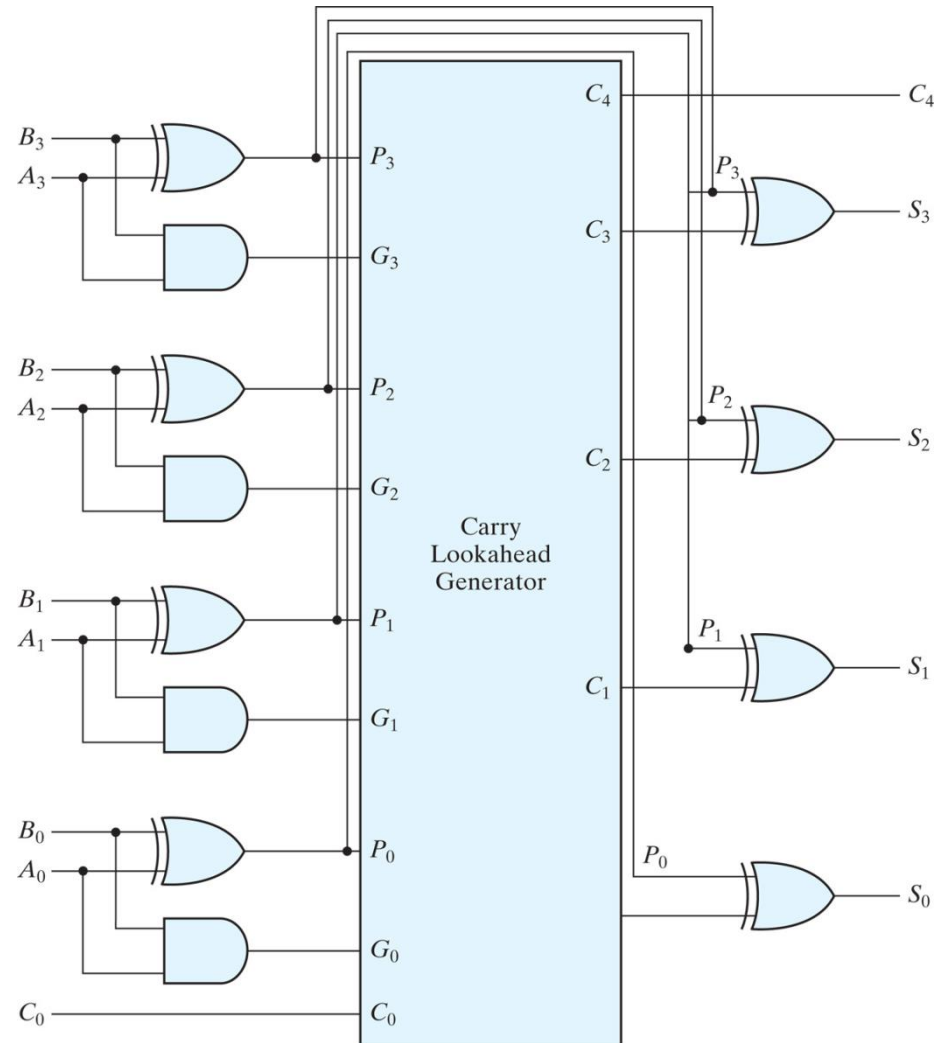
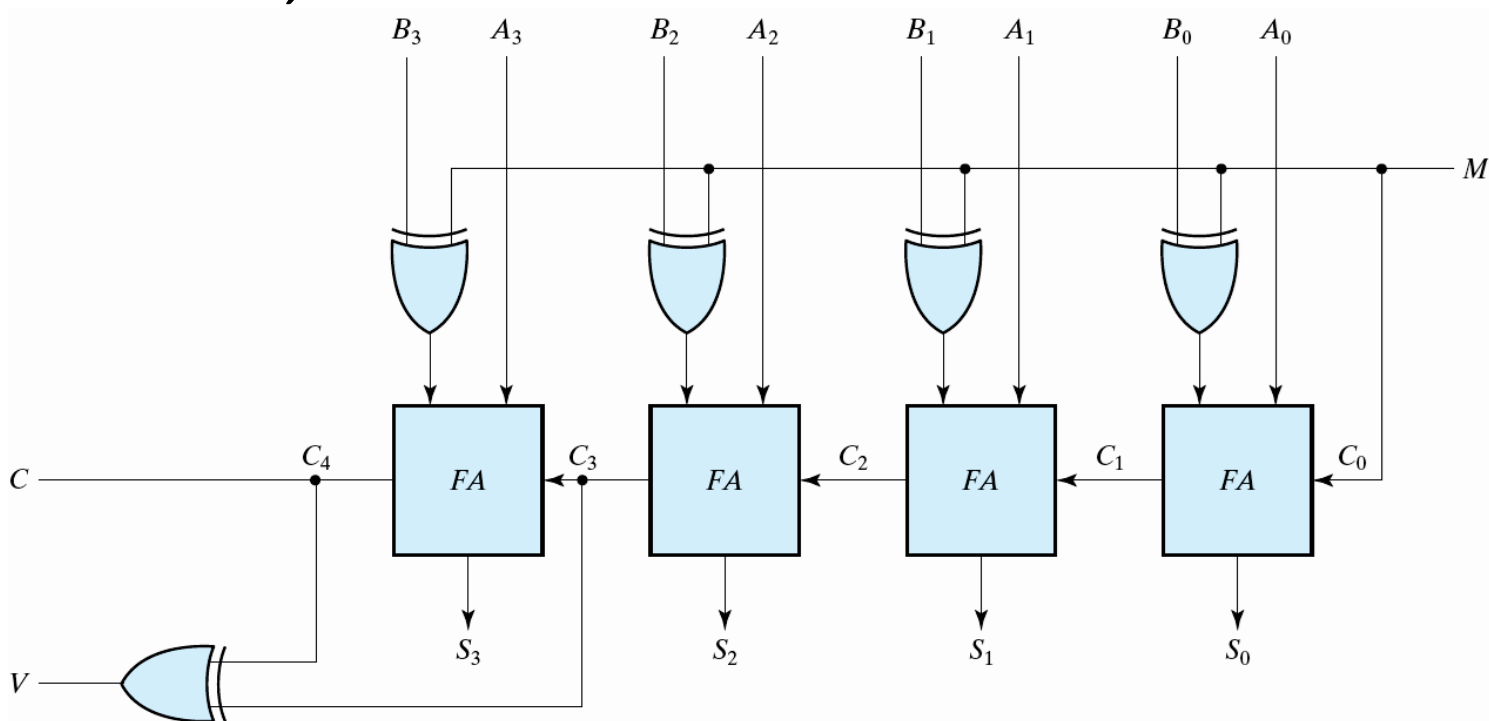


Fig. 4.12 4-Bit Adder with Carry Look-ahead

# Binary Subtractor (p.157, 158)

- Signed binary subtraction is performed by adding the minuend to the 2's complement of the subtrahend
- 4-bit adder-subtractor
  - ◆  $M=0: A + B; A + B + 0$
  - ◆  $M=1: A - B; A + B' + 1$



Overflow detection

Fig. 4.13 4-Bit Adder Subtractor  
Combinational Logic-27

# Overflow (p.159)

## ■ Overflow

- ◆ The number of bits is limited
- ◆ Add two positive numbers and obtain a negative number
- ◆ Add two negative numbers and obtain a positive number
- ◆  $V = 0$ , no overflow;  $V = 1$ , overflow

## ■ Example:

carries:	<div>0 1</div>	carries:	<div>1 0</div>
+70	0 1000110	-70	1 0111010
+80	0 1010000	-80	1 0110000
<u>      </u>	<u>      </u>	<u>      </u>	<u>      </u>
+150	1 0010110	-150	0 1101010

# 4-6 Decimal Adder (p.160)

## ■ Add two BCD's

- ◆ 9 inputs: two BCD's and one carry-in
- ◆ 5 outputs: one BCD and one carry-out

## ■ Design approaches

- ◆ A truth table with  $2^9$  entries
- ◆ Use 4-bit binary adder
  - » The maximum sum  $\leftarrow 9 + 9 + 1 = 19$
  - » Binary to BCD

# BCD Adder (1/3) (p.161)

## ■ BCD Adder: binary sum to BCD sum

Table 4.5  
*Derivation of BCD Adder*

Binary Sum					BCD Sum					Decimal
<i>K</i>	<i>Z</i> <sub>8</sub>	<i>Z</i> <sub>4</sub>	<i>Z</i> <sub>2</sub>	<i>Z</i> <sub>1</sub>	<i>C</i>	<i>S</i> <sub>8</sub>	<i>S</i> <sub>4</sub>	<i>S</i> <sub>2</sub>	<i>S</i> <sub>1</sub>	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

# BCD Adder (2/3) (p.161)

## ▣ Modifications are needed if the sum > 9

◆ If sum > 9, then  $C = 1$

»  $K = 1$ , or

»  $Z_8 Z_4 = 1$  (11xx), or

»  $Z_8 Z_2 = 1$  (1x1x).

◆ Modification:  $-(10)_d$  or  $+6$



$$C = K + Z_8 Z_4 + Z_8 Z_2$$

# BCD Adder (3/3) (p.162)

## ■ Block diagram

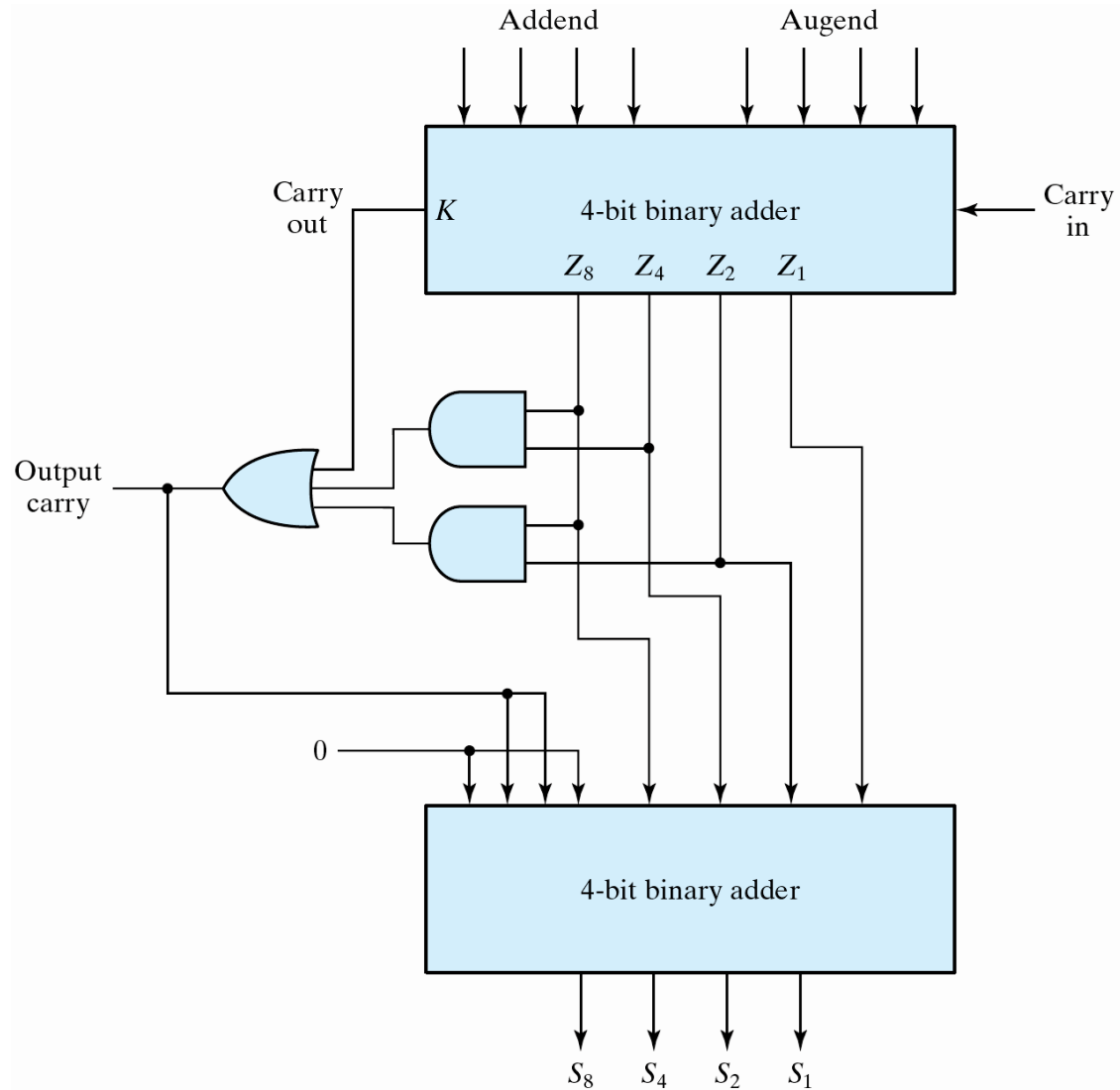


Fig. 4-14 Block Diagram of a BCD Adder



# 4-7 Binary Multiplier (1/3) (p.162)

## ■ Multiplication consists of

- ◆ Generation of partial products
- ◆ Accumulation of shifted partial products

						$y_5$	$y_4$	$y_3$	$y_2$	$y_1$	$y_0$	Multiplicand Multiplier
						$x_5$	$x_4$	$x_3$	$x_2$	$x_1$	$x_0$	
						$x_0 y_5$	$x_0 y_4$	$x_0 y_3$	$x_0 y_2$	$x_0 y_1$	$x_0 y_0$	Partial Products
					$x_1 y_5$	$x_1 y_4$	$x_1 y_3$	$x_1 y_2$	$x_1 y_1$	$x_1 y_0$		
				$x_2 y_5$	$x_2 y_4$	$x_2 y_3$	$x_2 y_2$	$x_2 y_1$	$x_2 y_0$			
			$x_3 y_5$	$x_3 y_4$	$x_3 y_3$	$x_3 y_2$	$x_3 y_1$	$x_3 y_0$				
		$x_4 y_5$	$x_4 y_4$	$x_4 y_3$	$x_4 y_2$	$x_4 y_1$	$x_4 y_0$					
	$x_5 y_5$	$x_5 y_4$	$x_5 y_3$	$x_5 y_2$	$x_5 y_1$	$x_5 y_0$						Product
$p_{11}$	$p_{10}$	$p_9$	$p_8$	$p_7$	$p_6$	$p_5$	$p_4$	$p_3$	$p_2$	$p_1$	$p_0$	

# Binary Multiplier (2/3) (p.163)

## Partial products

### AND operations

$$\begin{array}{r} B_1 \quad B_0 \\ A_1 \quad A_0 \\ \hline A_0 B_1 \quad A_0 B_0 \\ \\ A_1 B_1 \quad A_1 B_0 \\ \hline C_3 \quad C_2 \quad C_1 \quad C_0 \end{array}$$

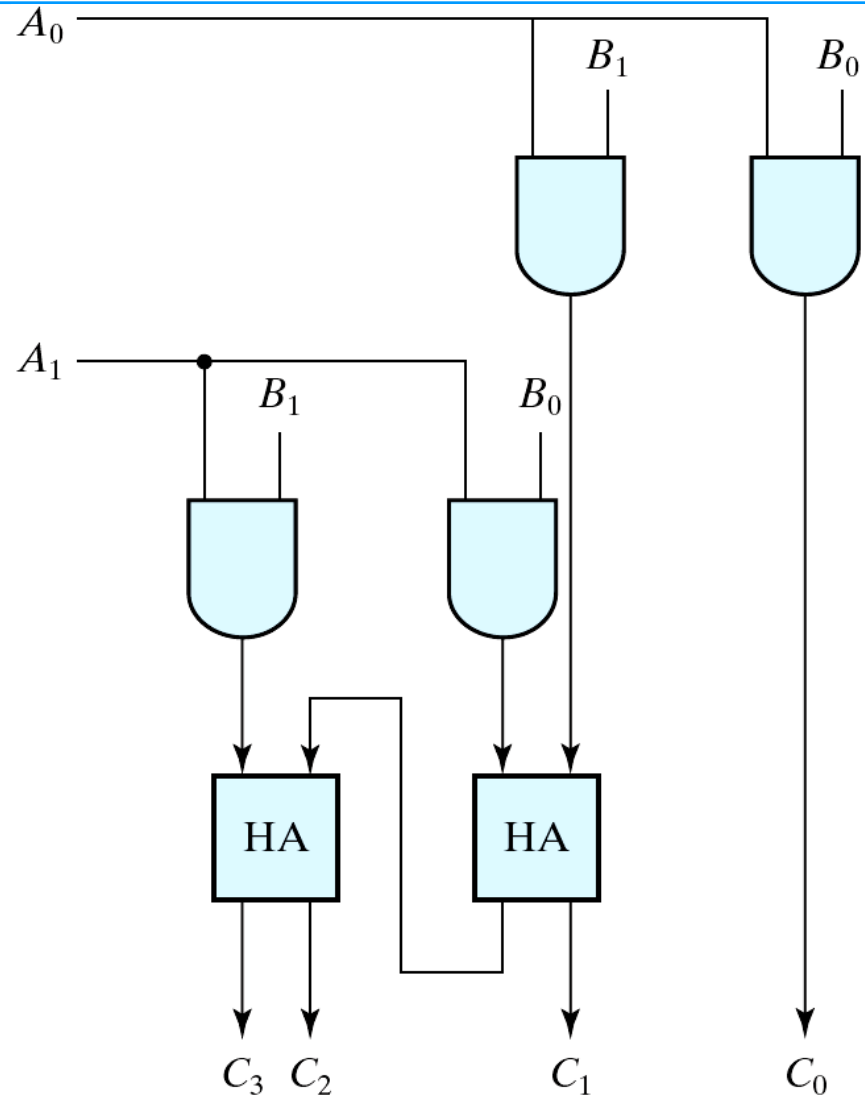


Fig. 4.15 Two-bit by two-bit binary multiplier

# Binary Multiplier (3/3) (p.164)

## 4-bit by 3-bit binary multiplier

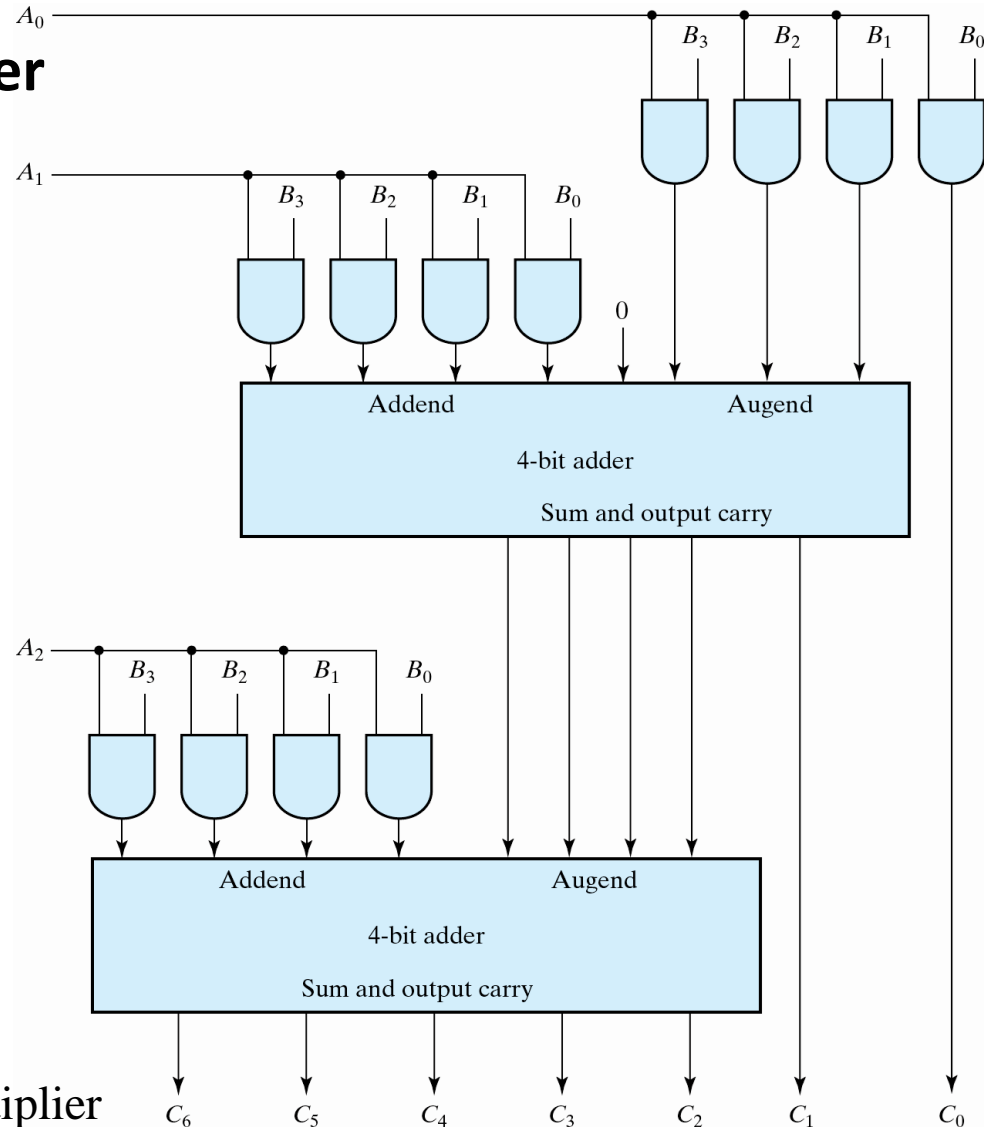
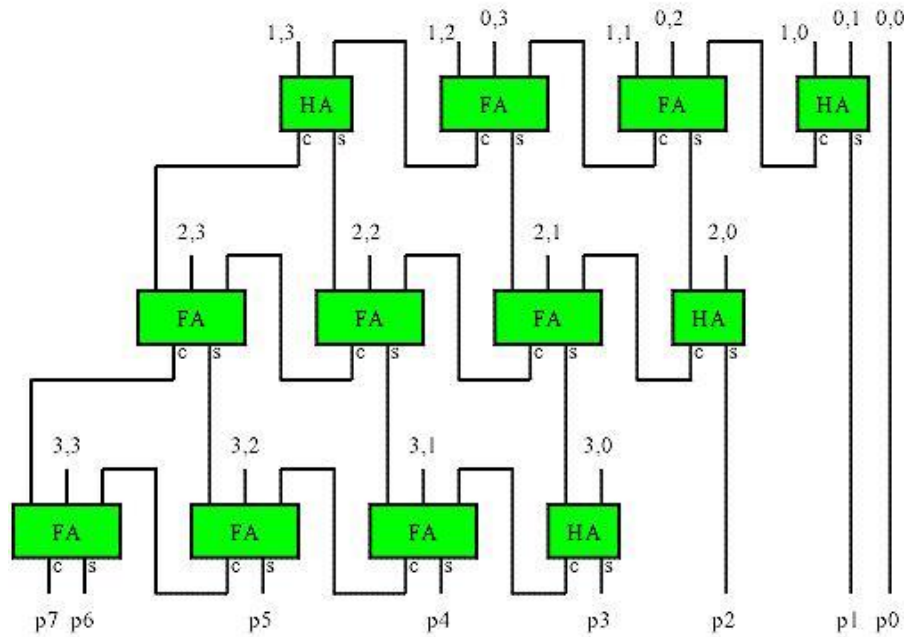
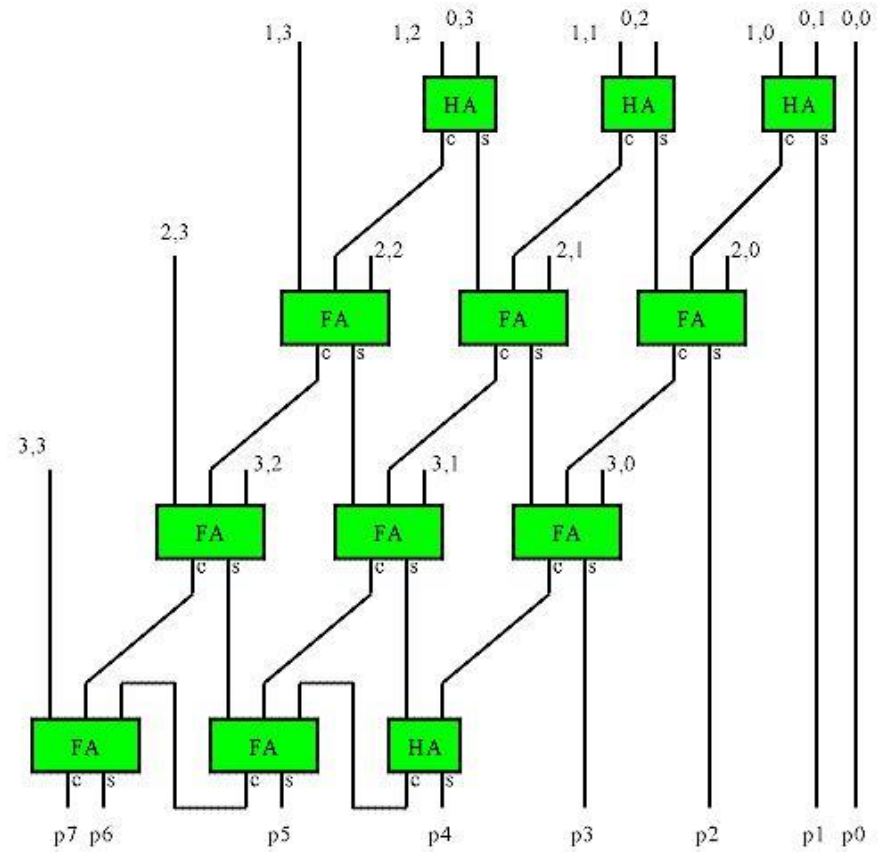


Fig. 4.16 Four-bit by three-bit binary multiplier

# Various Multipliers



	B	3	2	1	0
	A	3	2	1	0
		03	02	01	00
	13	12	11	10	
	23	22	21	20	
33	32	31	30		



# 4-8 Magnitude Comparator (1/3) (p.164)

## ■ The comparison of two unsigned numbers

- ◆ Outputs:  $A > B$ ,  $A = B$ ,  $A < B$

## ■ Design Approaches

- ◆ The truth table of  $2n$ -bit comparator
  - »  $2^{2n}$  entries - too cumbersome for large  $n$
- ◆ Use inherent regularity of the problem
  - » Reduce design efforts
  - » Reduce human errors

# Magnitude Comparator (2/3) (p.165)

## ▣ Algorithm → logic

◆  $A = A_3A_2A_1A_0 ; B = B_3B_2B_1B_0$

◆  $A=B$

»  $A_3=B_3, A_2=B_2, A_1=B_1, \text{ and } A_0=B_0$

– Equality:  $x_i = A_iB_i + A_i'B_i'$  (equivalence)

–  $(A=B) = x_3x_2x_1x_0=1$

◆  $(A>B)$

»  $A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$

◆  $(A<B)$

»  $A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$

## ▣ Implementation

◆  $x_i = (A_iB_i' + A_i'B_i)'$

# Magnitude Comparator (3/3) (p.166)

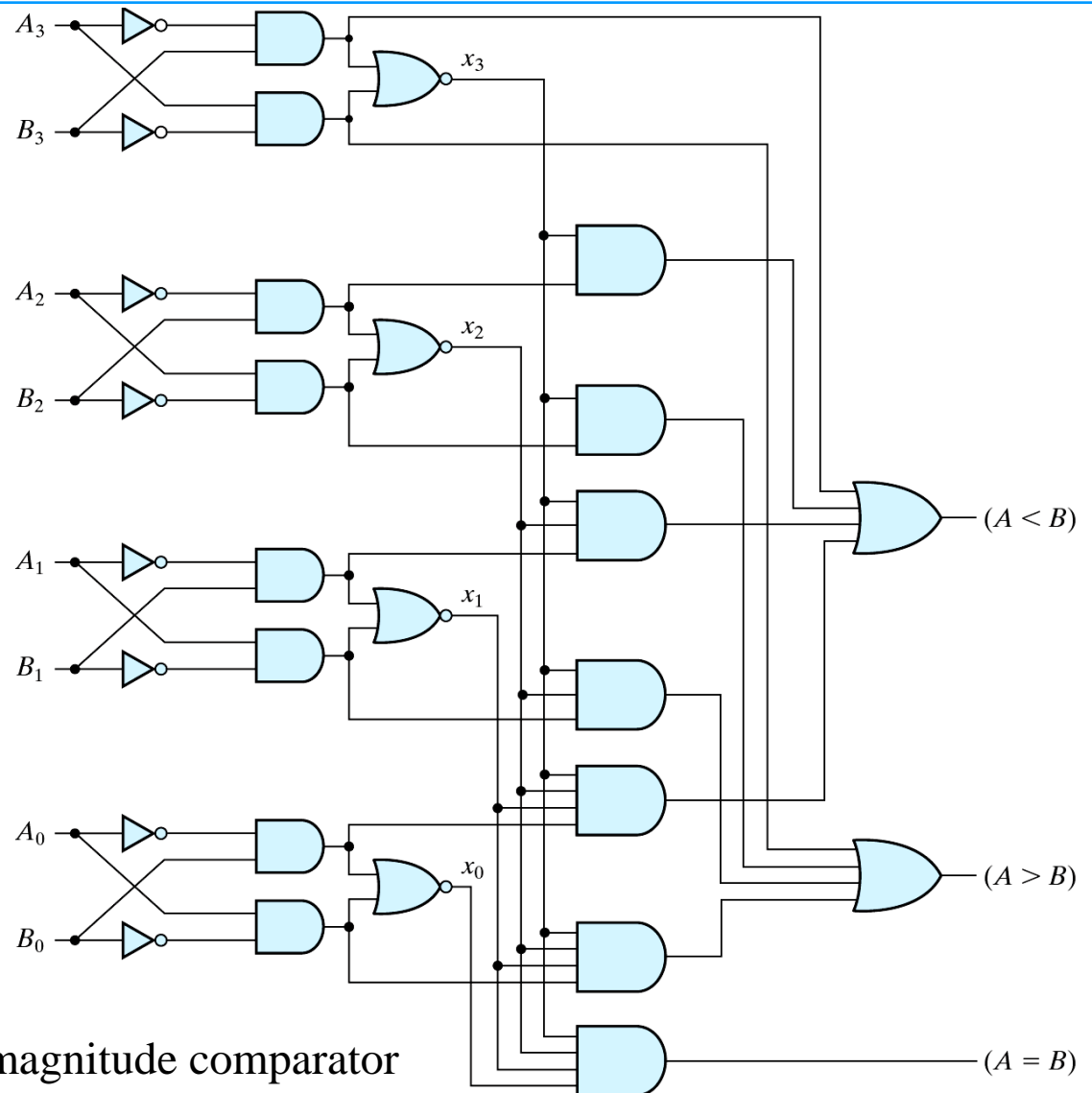


Fig. 4.17 Four-bit magnitude comparator

# 4-9 Decoder (1/2) (p.167, 168)

## ■ A $n$ -to- $m$ decoder

- ◆ A binary code of  $n$  bits =  $2^n$  distinct information
- ◆  $n$  input variables; up to  $2^n$  output lines
- ◆ Only one output can be active (high) at any time

**Table 4.6**

*Truth Table of a Three-to-Eight-Line Decoder*

Inputs			Outputs							
$x$	$y$	$z$	$D_0$	$D_1$	$D_2$	$D_3$	$D_4$	$D_5$	$D_6$	$D_7$
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



# Decoder (2/2)

## ■ An implementation

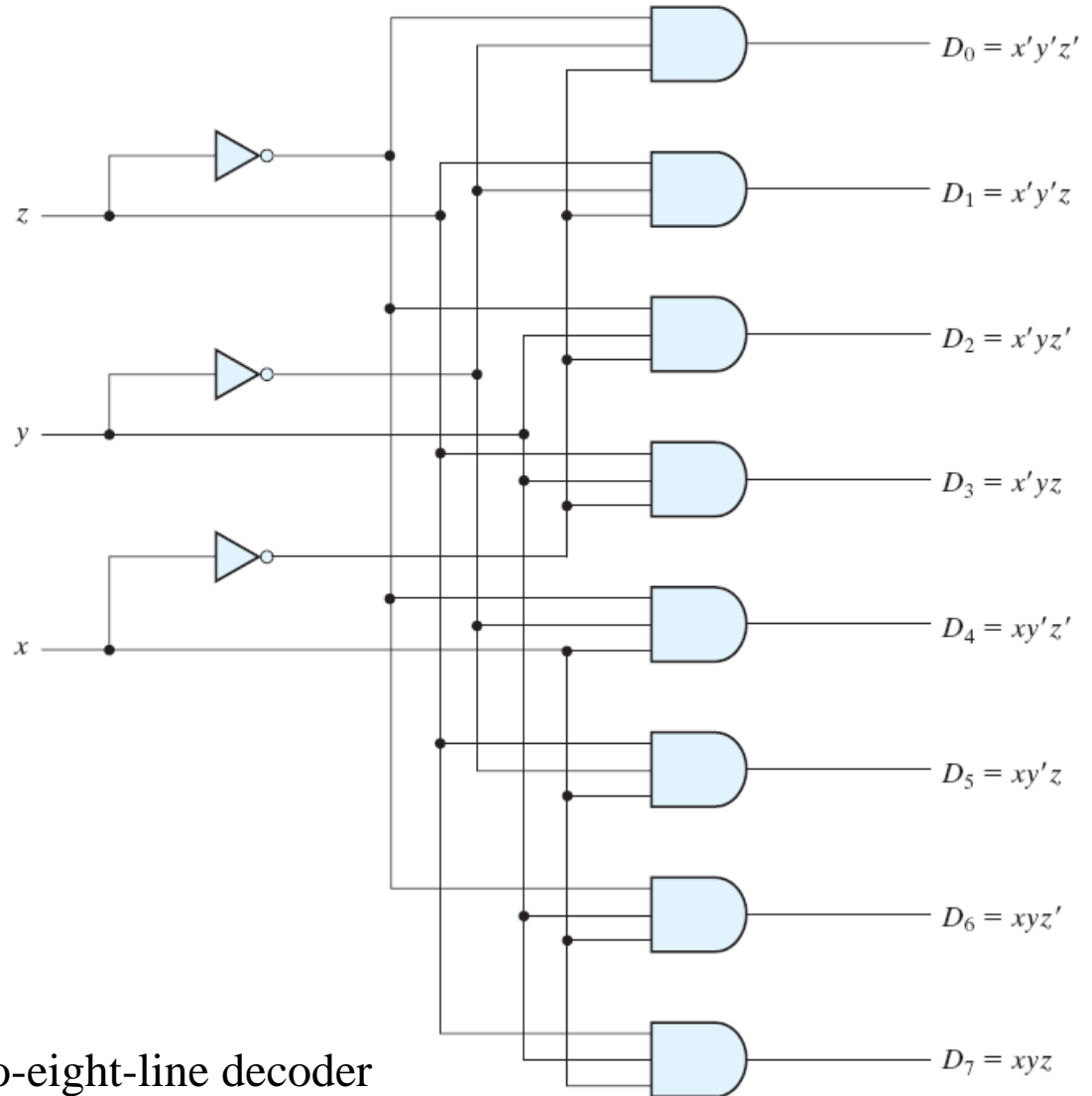


Fig. 4.18 Three-to-eight-line decoder

# Decoder with Enable Input (p.168)

## ■ Decoder with enable

- ◆ A decoder with an enable input
- ◆ Receive information on a single line and transmits it on one of  $2^n$  possible output lines

$E$	$A$	$B$	$D_0$	$D_1$	$D_2$	$D_3$
1	$X$	$X$	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

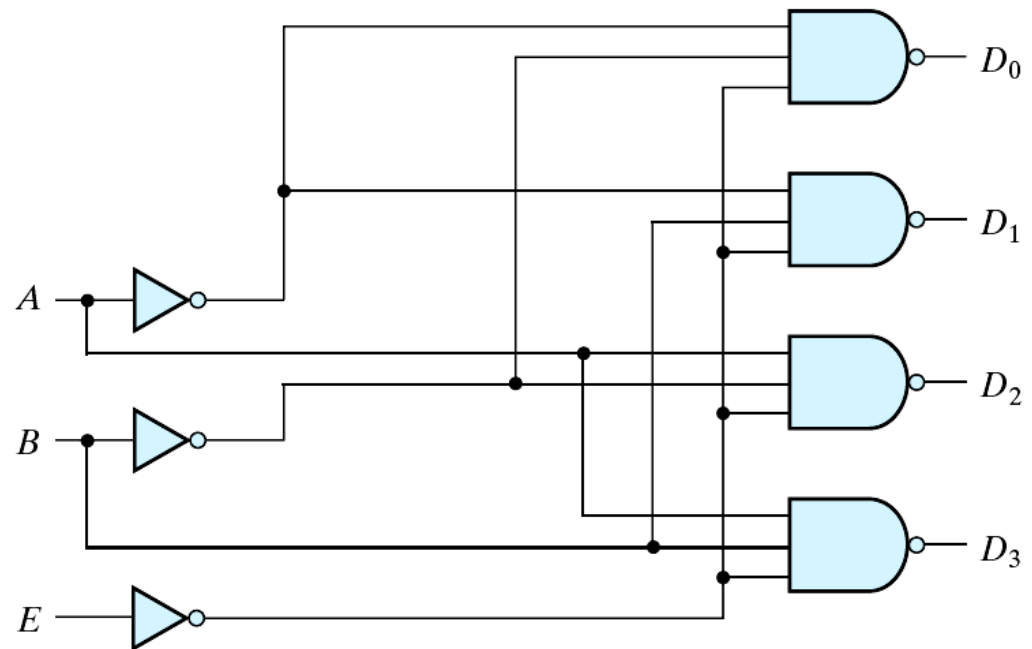
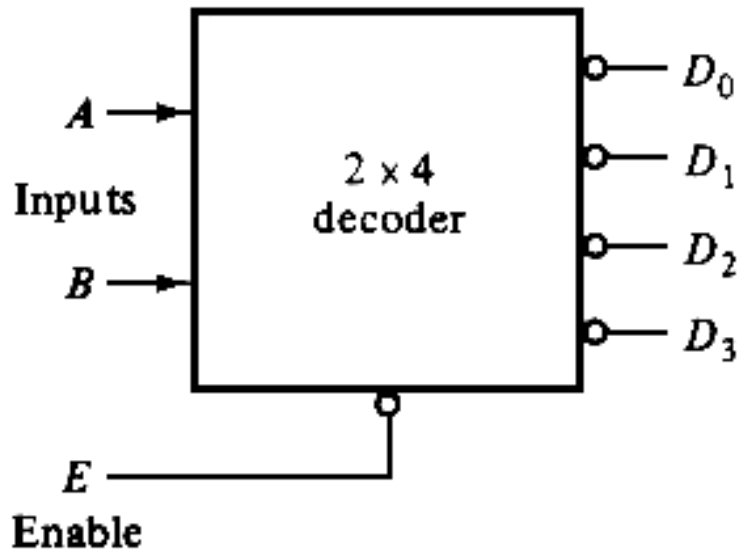


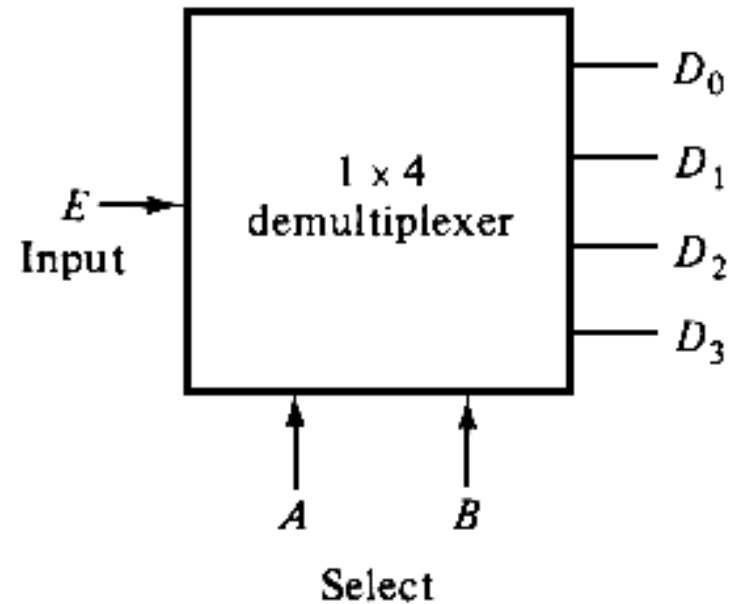
Fig. 4.19 Two-to-four-line decoder with enable input (NAND implementation)

# Decoder/Demultiplexer

## ▣ Decoder/demultiplexer



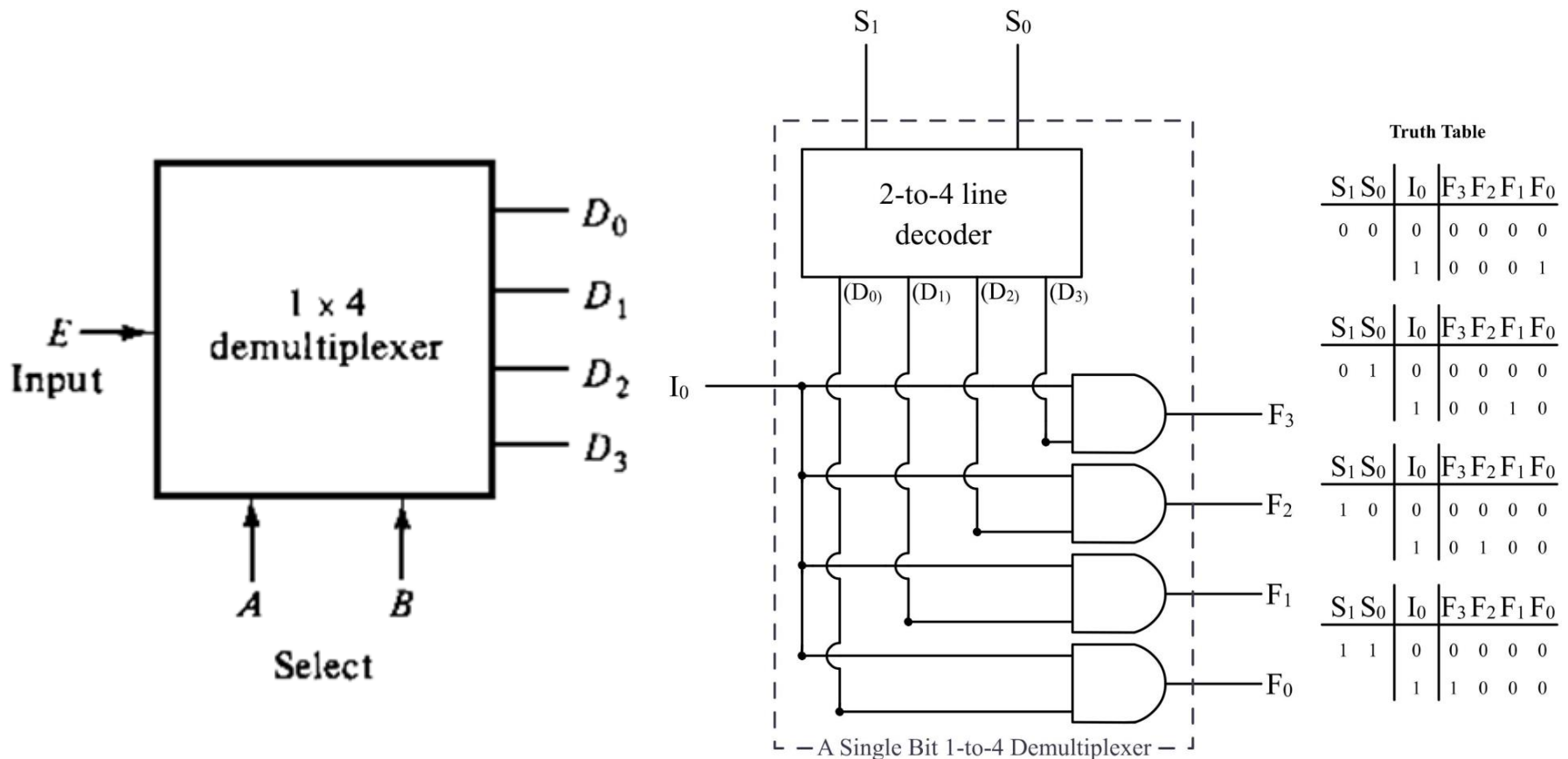
(a) Decoder with enable



(b) Demultiplexer

# Demultiplexer

- Forward the data input to one of the outputs
- 1 input lines,  $n$  selection lines, and  $2^n$  output lines



# Decoder Expansion (p.169)

## ■ Decoder expansion

### ◆ Two 3-to-8 decoder with enable: a 4-to-16 decoder

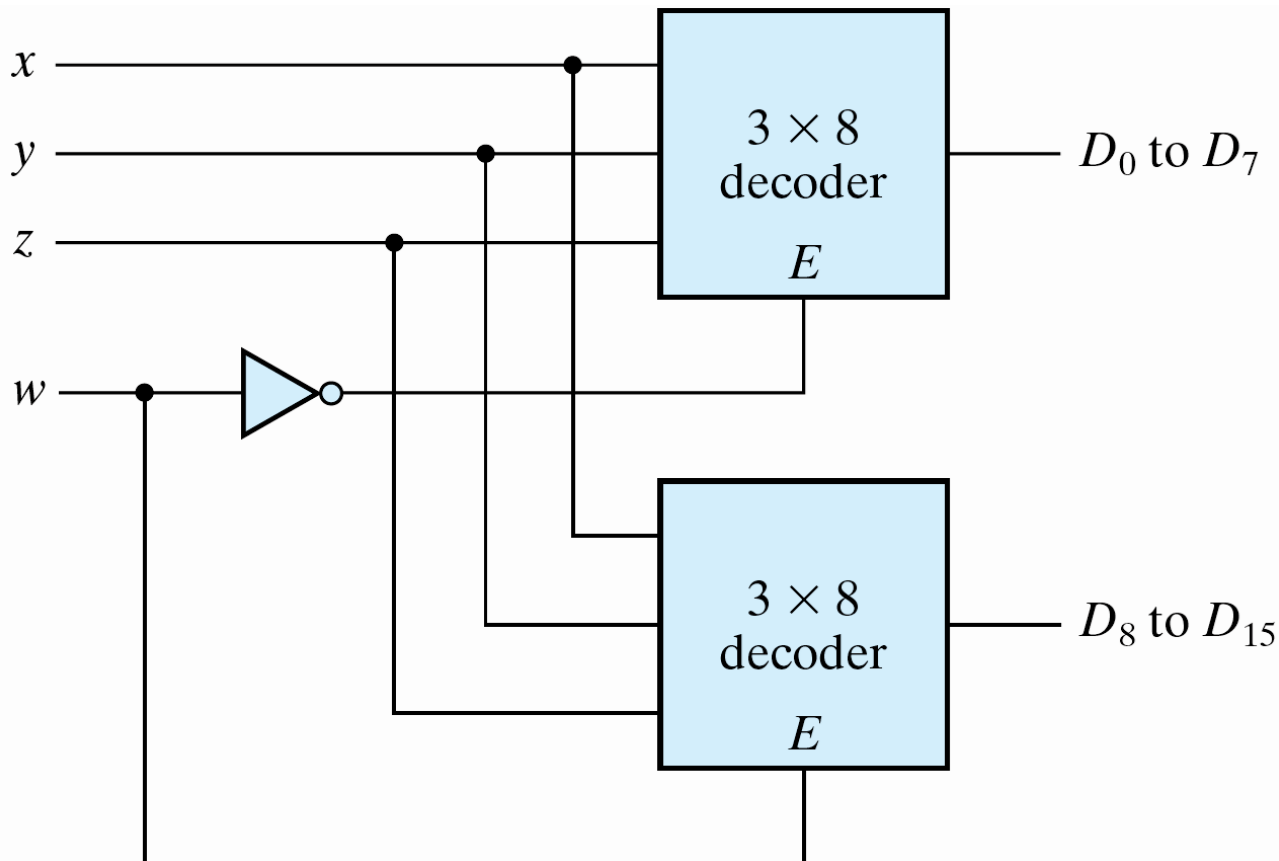


Fig. 4.20  $4 \times 16$  decoder constructed with two  $3 \times 8$  decoders

# SOP Implementation (1/2) (p.170)

## ■ Combinational logic implementation

- ◆ Each output = a minterm
- ◆ Use a decoder and an external OR gate to implement any Boolean function of  $n$  input variables
- ◆ A full-adder
  - »  $S(x, y, z) = \Sigma(1, 2, 4, 7)$
  - »  $C(x, y, z) = \Sigma(3, 5, 6, 7)$

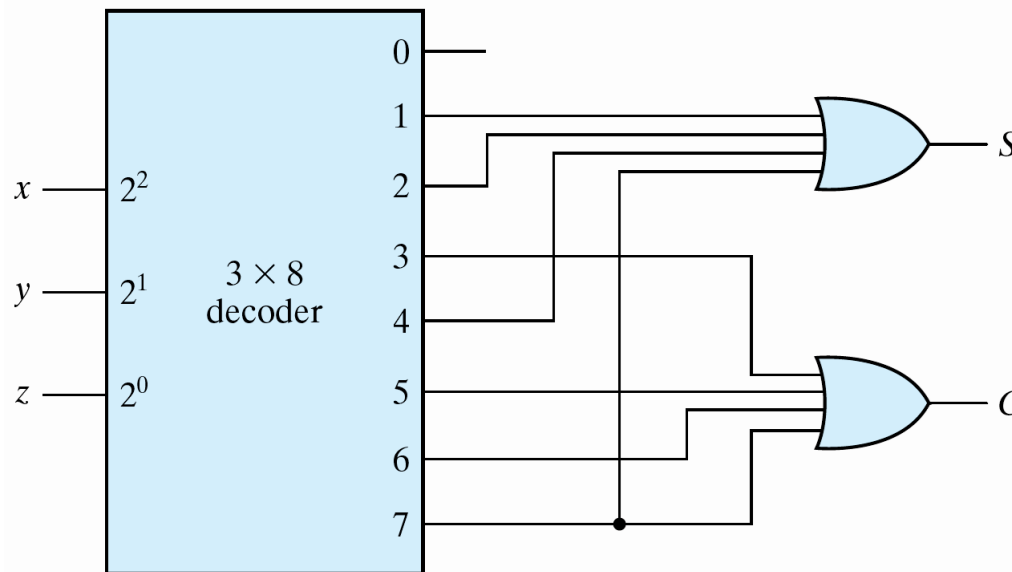


Fig. 4.21 Implementation of a full adder with a decoder

# SOP Implementation (2/2)

- ◆ Two possible approaches using decoder
  - » OR (minterms of  $F$ ):  $k$  inputs ( $k$  minterms)
  - » NOR (minterms of  $F'$ ):  $2^n - k$  inputs
- ◆ In general, it is not a practical implementation for a large design
  - » Too many minterms

# 4-10 Encoders (1/2) (p.171)

## ■ The inverse function of a decoder

**Table 4.7**

*Truth Table of an Octal-to-Binary Encoder*

Inputs								Outputs		
$D_0$	$D_1$	$D_2$	$D_3$	$D_4$	$D_5$	$D_6$	$D_7$	$x$	$y$	$z$
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1


$$z = D_1 + D_3 + D_5 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

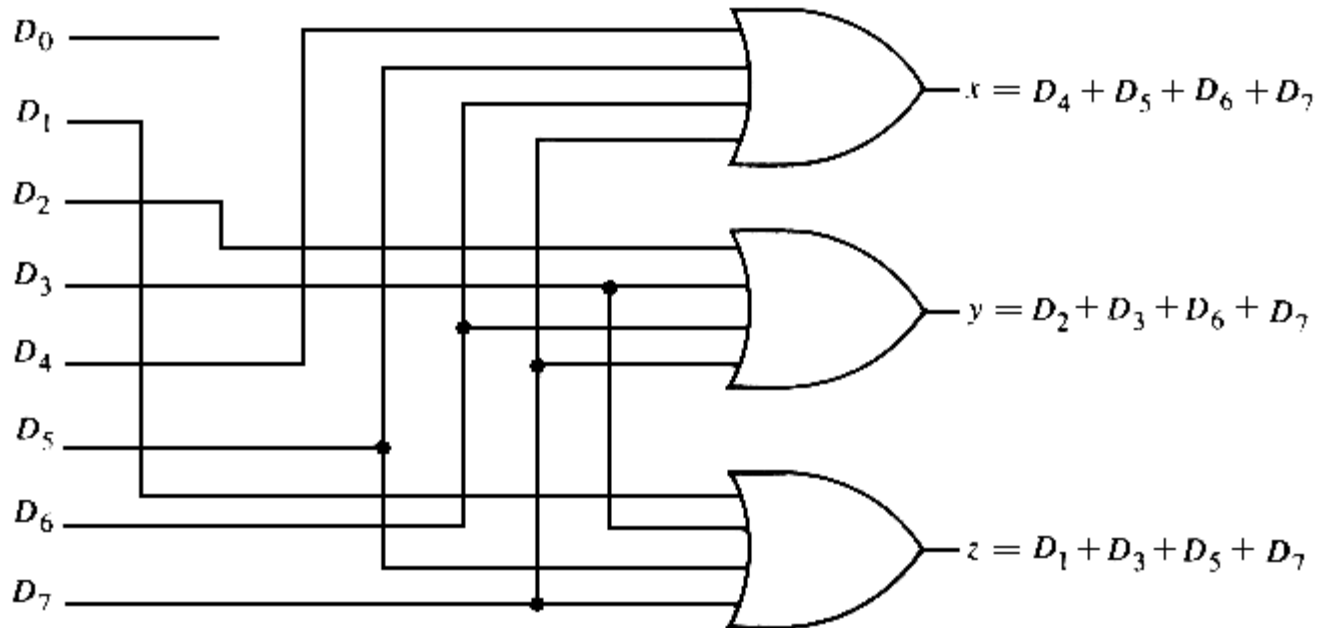
$$x = D_4 + D_5 + D_6 + D_7$$

The encoder can be implemented with three OR gates



# Encoders (2/2)

## ■ An implementation



## ◆ Limitations

- » Illegal input:  $D_3=D_6=1$ 
  - The output = 111 ( $D_7=1$ )
- » Illegal input:  $D_0=D_1=D_2=D_3=D_4=D_5=D_6=D_7=0$ 
  - The output = 000 ( $D_0=1$ )

# Priority Encoder (1/3) (p.172)

- ◆ Resolve the ambiguity of illegal inputs
- ◆ Only one of the input is encoded

**Table 4.8**  
*Truth Table of a Priority Encoder*

Inputs				Outputs		
$D_0$	$D_1$	$D_2$	$D_3$	$x$	$y$	$V$
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

- ◆  $D_3$  has the highest priority
- ◆  $D_0$  has the lowest priority
- ◆ X: don't-care conditions
  - » NOTICE: **input** and **output** don't-cares are different
- ◆  $V$ : valid output indicator
  - » Avoid  $D_0=D_1=D_2=D_3=0$

# Priority Encoder (2/3)

- The maps for simplifying outputs  $x$  and  $y$

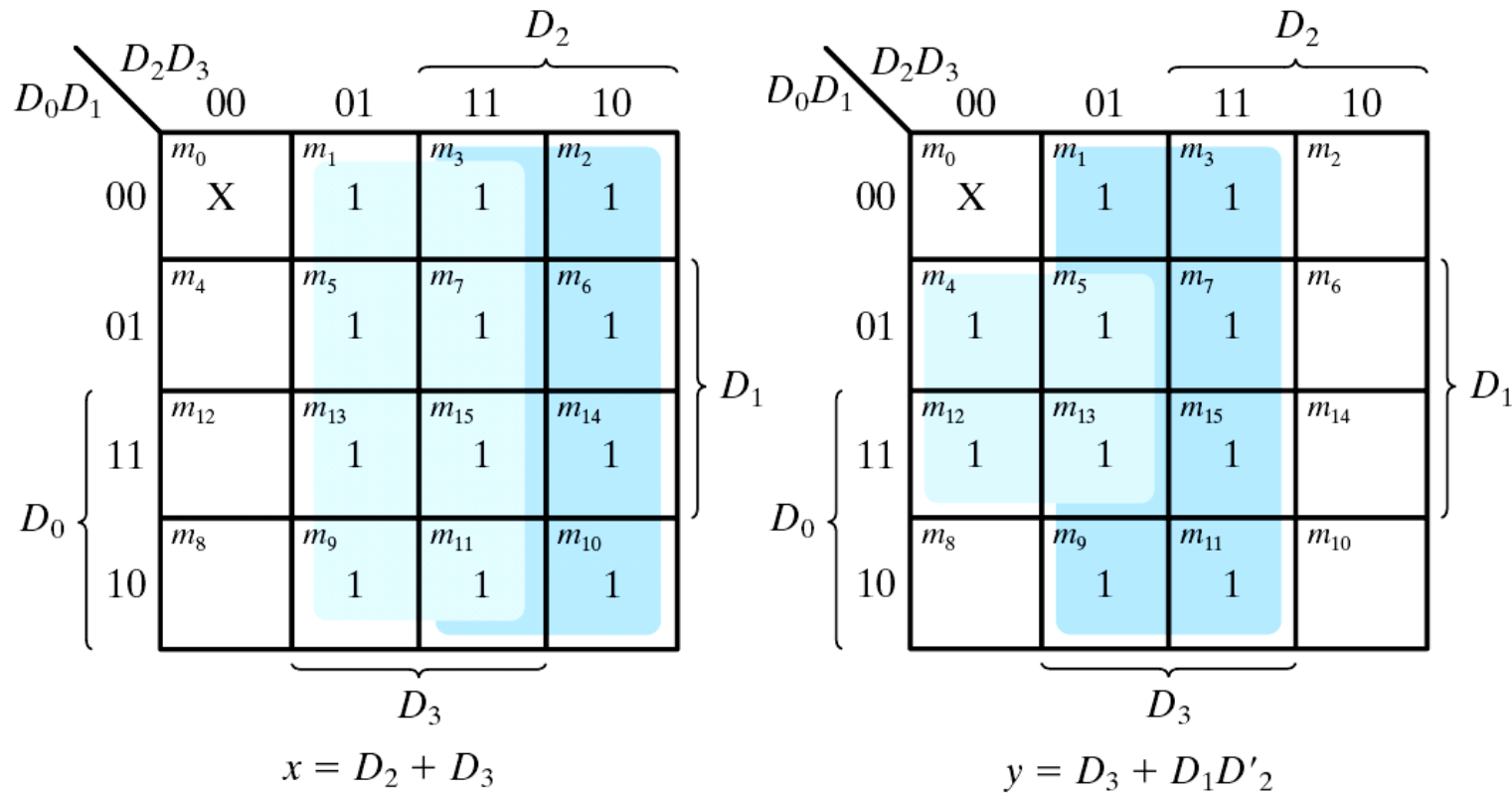


Fig. 4.22 Maps for a priority encoder

# Priority Encoder (3/3)

## Implementation of priority encoder

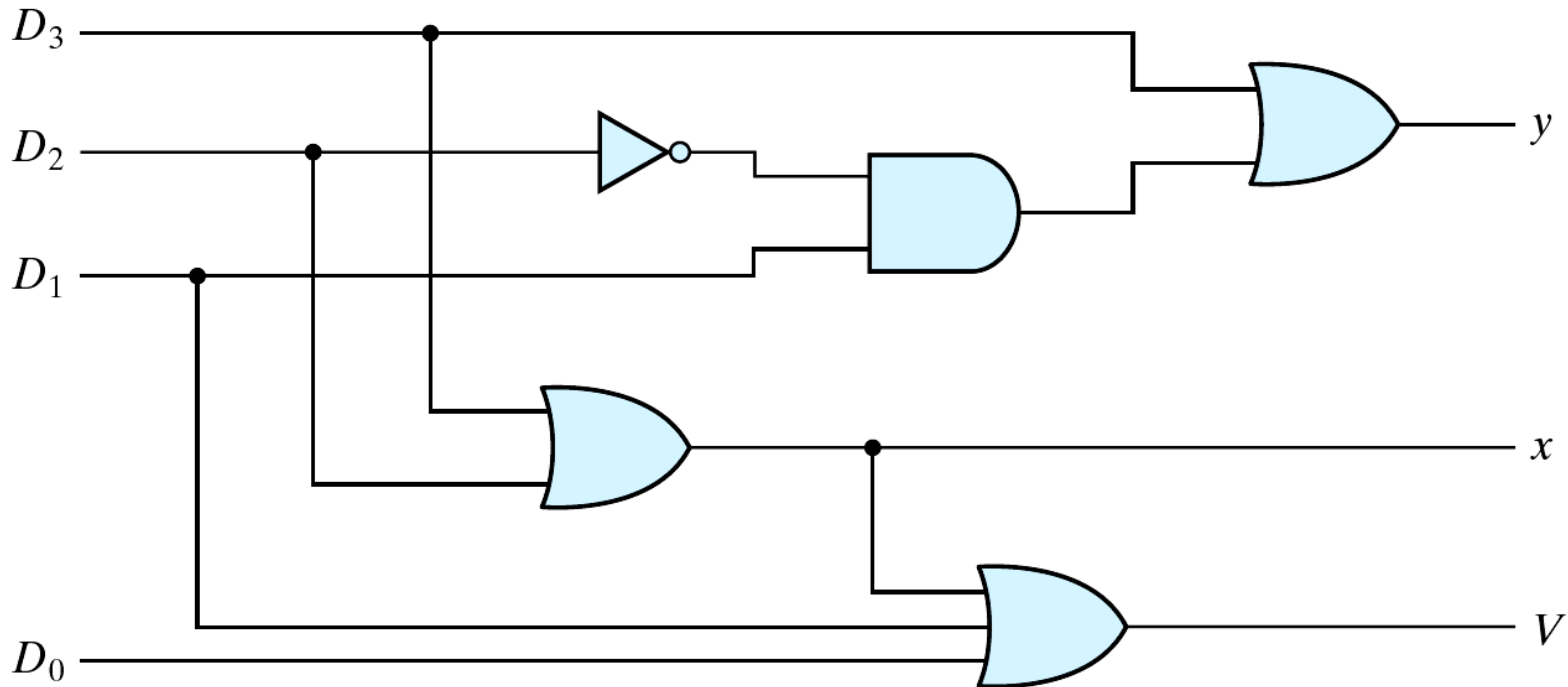


Fig. 4.23 Four-input priority encoder

$$x = D_2 + D_3$$

$$y = D_3 + D_1 D_2'$$

$$V = D_0 + D_1 + D_2 + D_3$$

# 4-11 Multiplexers (p.174)

- Select binary information from one of many input lines and direct it to a single output line
- $2^n$  input lines,  $n$  selection lines, and one output line
- e.g.: 2-to-1-line multiplexer

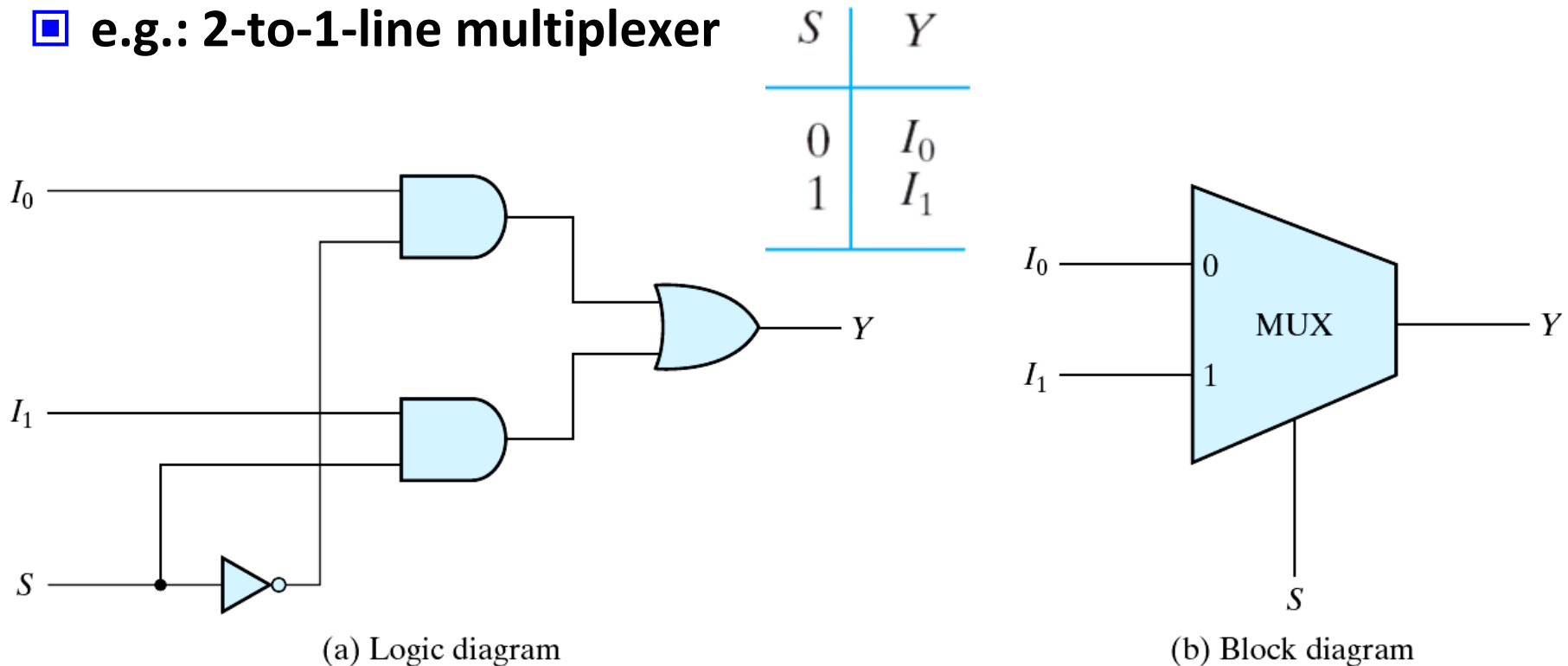


Fig. 4.24 Two-to-one-line multiplexer

# Four-to-one-line Multiplexer (p.175)

## 4-to-1 line multiplexer

$S_1$	$S_0$	$Y$
0	0	$I_0$
0	1	$I_1$
1	0	$I_2$
1	1	$I_3$

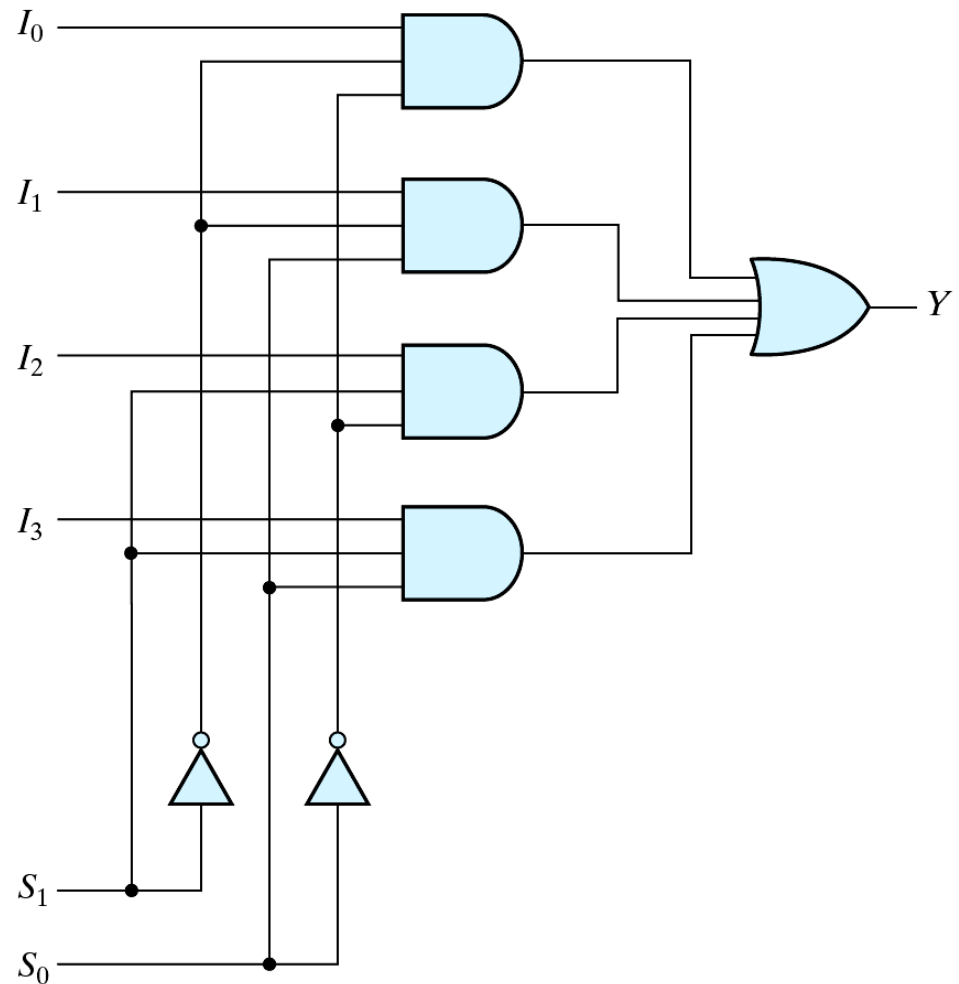
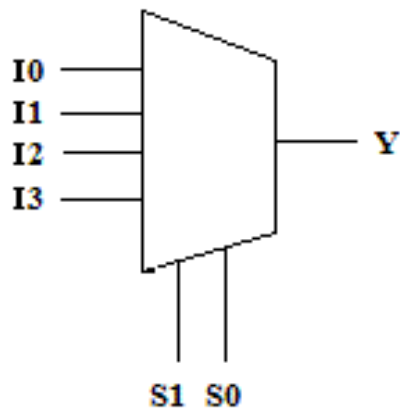


Fig. 4.25 Four-to-one-line multiplexer

# $2^n$ -to-1 Multiplexer

## ■ Note: $2^n$ -to-1 multiplexer

- ◆  $n$ -to- $2^n$  decoder
- ◆ Add the  $2^n$  input lines to each AND gate
- ◆ OR (all AND gates)
- ◆  $n$  selection lines
- ◆ An enable input (optional)

# Quadruple 2-to-1-line MUX with Enable

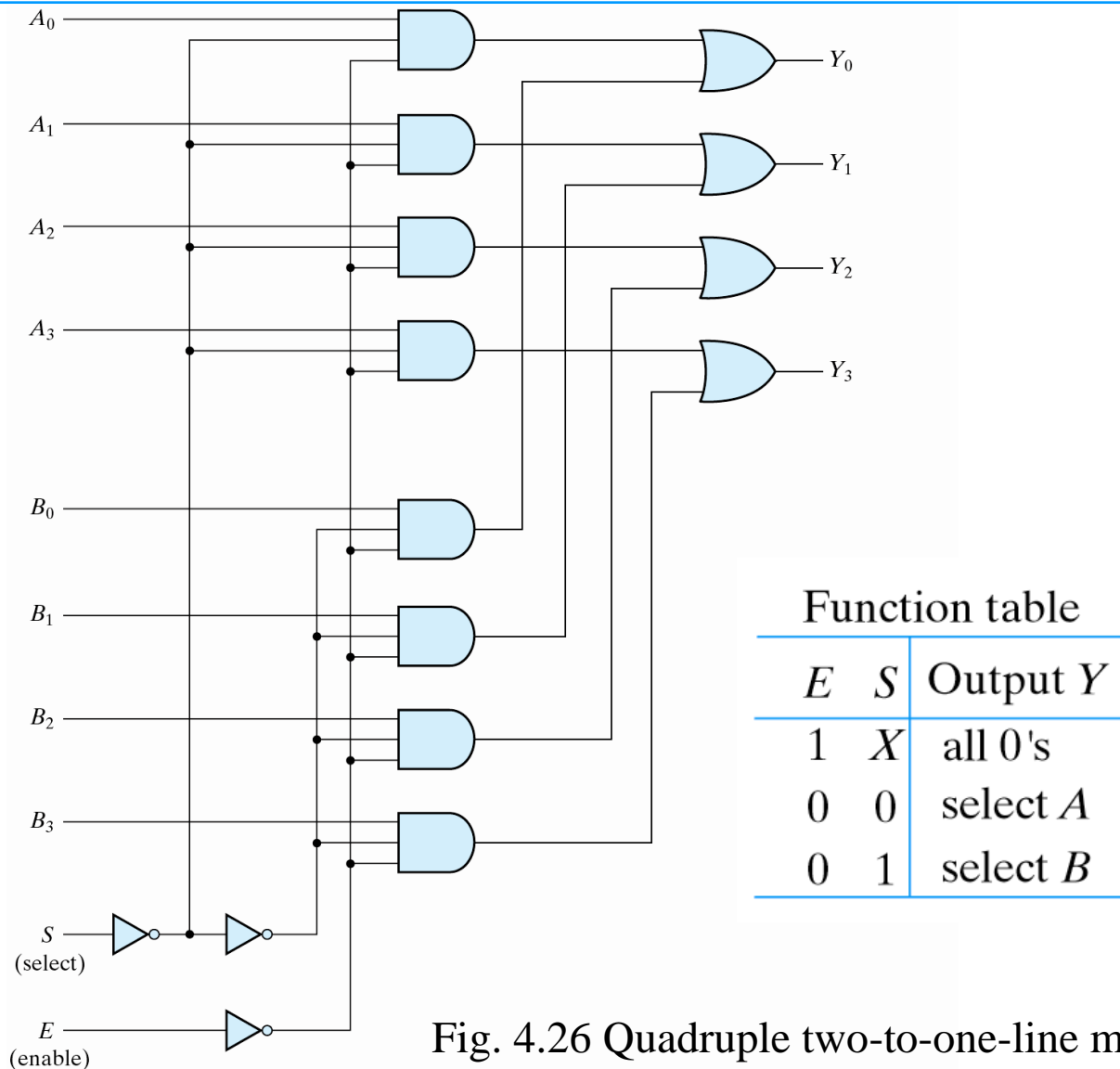


Fig. 4.26 Quadruple two-to-one-line multiplexer



# Boolean Function Implementation (1/4)

(p.176)

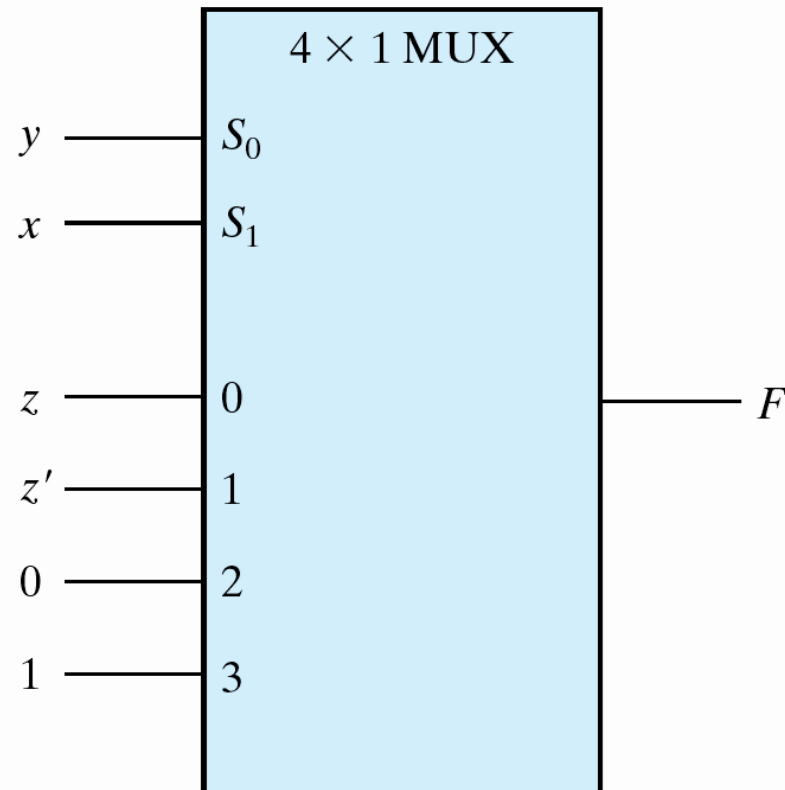
- MUX: a decoder + an OR gate
- $n$  inputs Boolean function implementation
  - ◆  $2^n$ -to-1 MUX: trivial (p.176)
    - » Each minterm value (0 or 1) is assigned to corresponding MUX input line
  - ◆  $2^{n-1}$ -to-1 MUX
    - »  $n-1$  of these variables: MUX selection lines
    - » The last variable: MUX input lines

# Boolean Function Implementation (2/4)

- An example:  $F(x, y, z) = \Sigma(1, 2, 6, 7)$

$x$	$y$	$z$	$F$	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table



(b) Multiplexer implementation

Fig. 4.27 Implementing a Boolean function with a multiplexer

# Boolean Function Implementation (3/4)

## □ Procedure:

- ◆ Assign an ordering sequence of the input variable
- ◆ The rightmost variable will be used for the input lines
- ◆ Assign the remaining  $n-1$  variables to the selection lines w.r.t. their corresponding sequence
- ◆ Construct the truth table
- ◆ Consider a pair of consecutive minterms starting from  $m_0$
- ◆ Determine the input lines

# Boolean Function Implementation (4/4)

(p.178)

■ Example:  $F(A, B, C, D) = \Sigma(1, 3, 4, 11, 12, 13, 14, 15)$

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

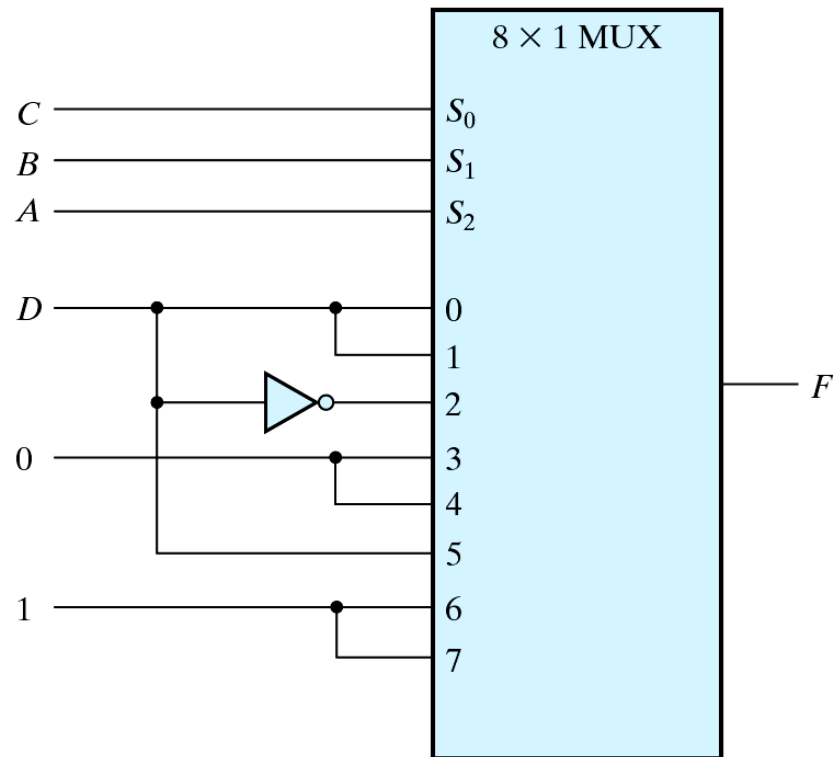
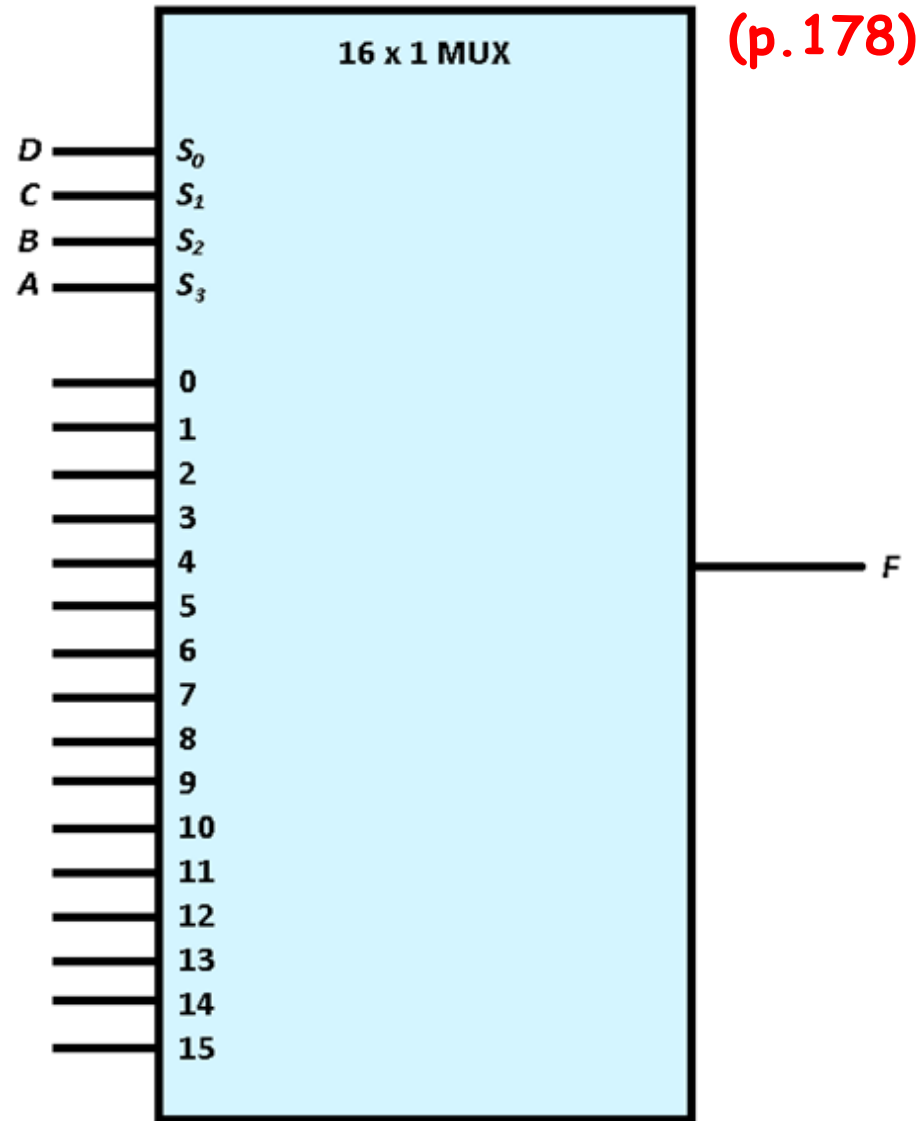


Fig. 4.28 Implementing a four-input function with a multiplexer

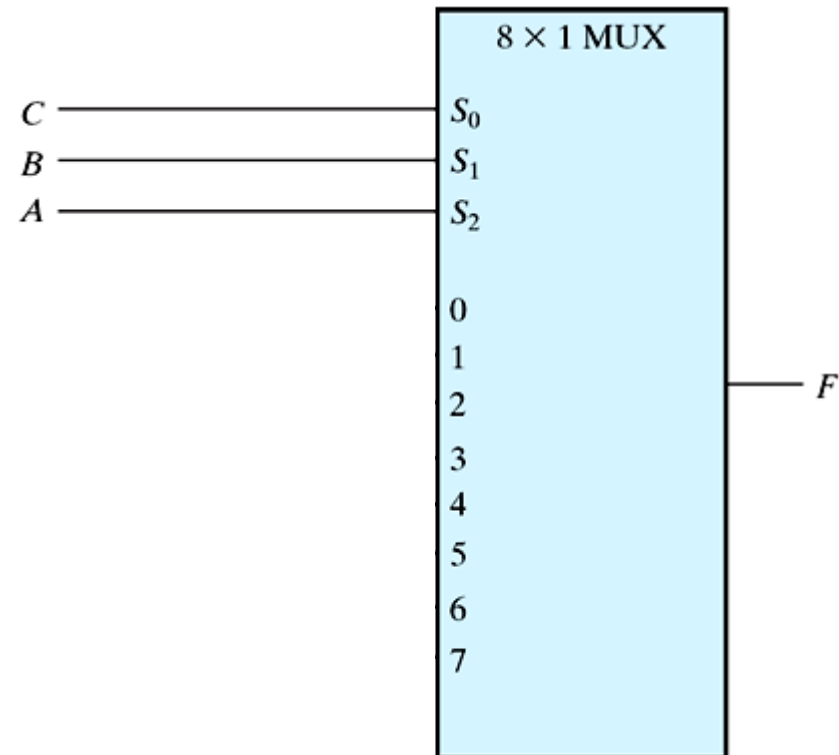
# 16-to-1 MUX Implementation (1/4)

A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



# 8-to-1 MUX Implementation (2/4)

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>F</i>	
0	0	0	0	0	$F_0 = D$
0	0	0	1	1	
0	0	1	0	0	$F_1 = D$
0	0	1	1	1	
0	1	0	0	1	$F_2 = D'$
0	1	0	1	0	
0	1	1	0	0	$F_3 = 0$
0	1	1	1	0	
1	0	0	0	0	$F_4 = 0$
1	0	0	1	0	
1	0	1	0	0	$F_5 = D$
1	0	1	1	1	
1	1	0	0	1	$F_6 = 1$
1	1	0	1	1	
1	1	1	0	1	$F_7 = 1$
1	1	1	1	1	



# 4-to-1 MUX Implementation (3/4)

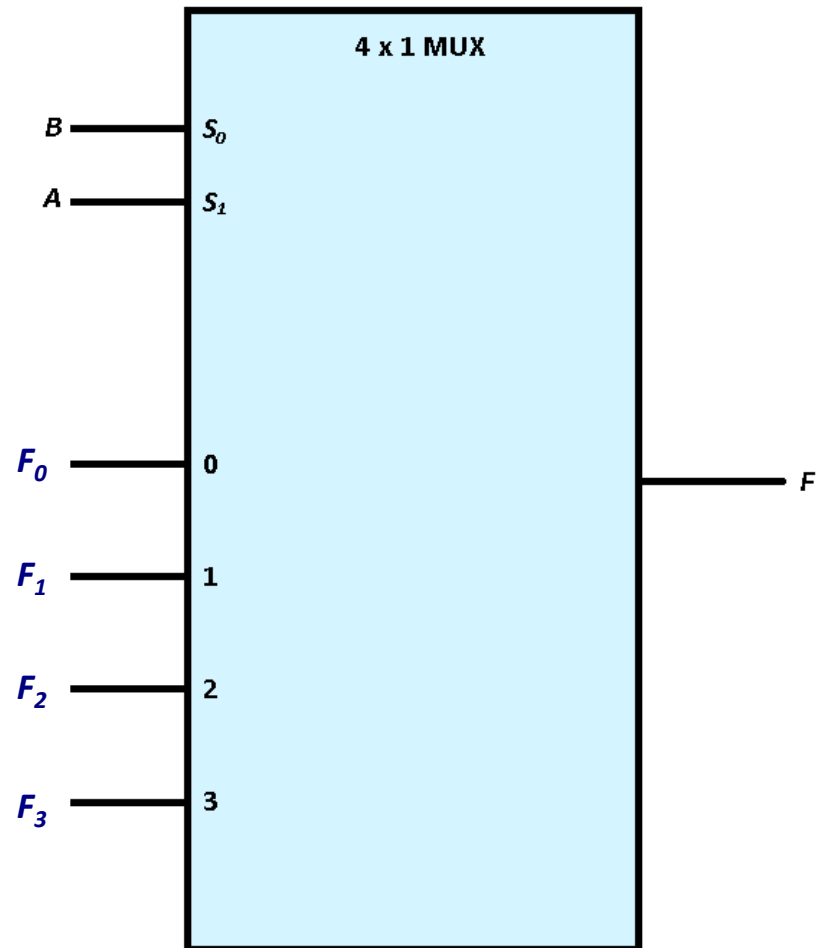
<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>F</i>
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

$$F_0 = D$$

$$F_1 = C'D'$$

$$F_2 = CD$$

$$F_3 = 1$$

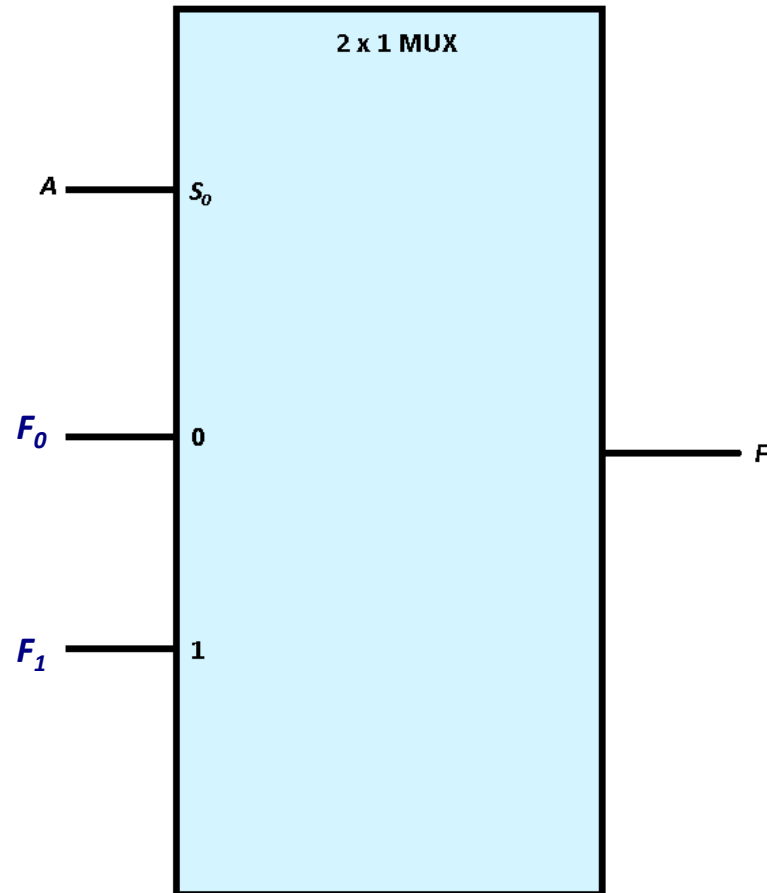


# 2-to-1 MUX Implementation (4/4)

A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

$$F_0 = B'D + BC'D'$$

$$F_1 = B'CD + B$$





# Three-state (Tri-state) Gates (p.178, 179)

- ◆ A multiplexer can be constructed with three-state gates
- ◆ Output state: 0, 1, and high-impedance (open ckts)

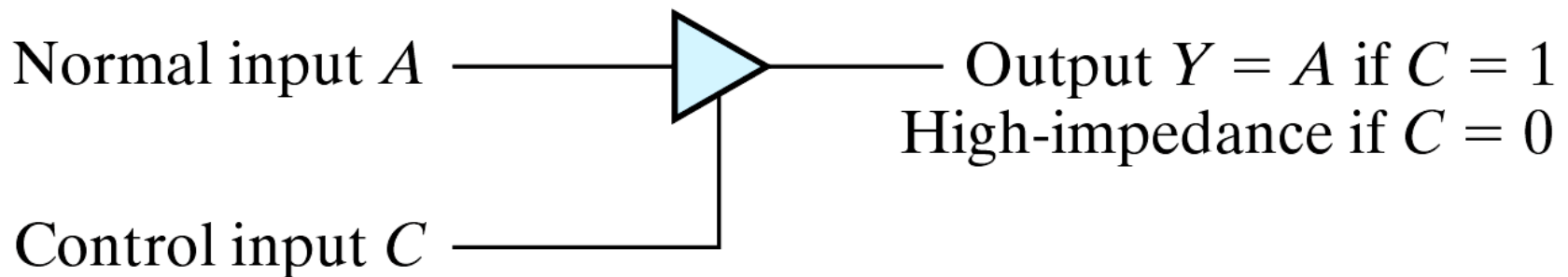
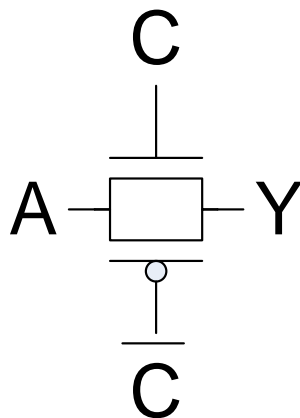


Fig. 4.29 Graphic symbol for a three-state buffer



C	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1

# Multiplexer Examples

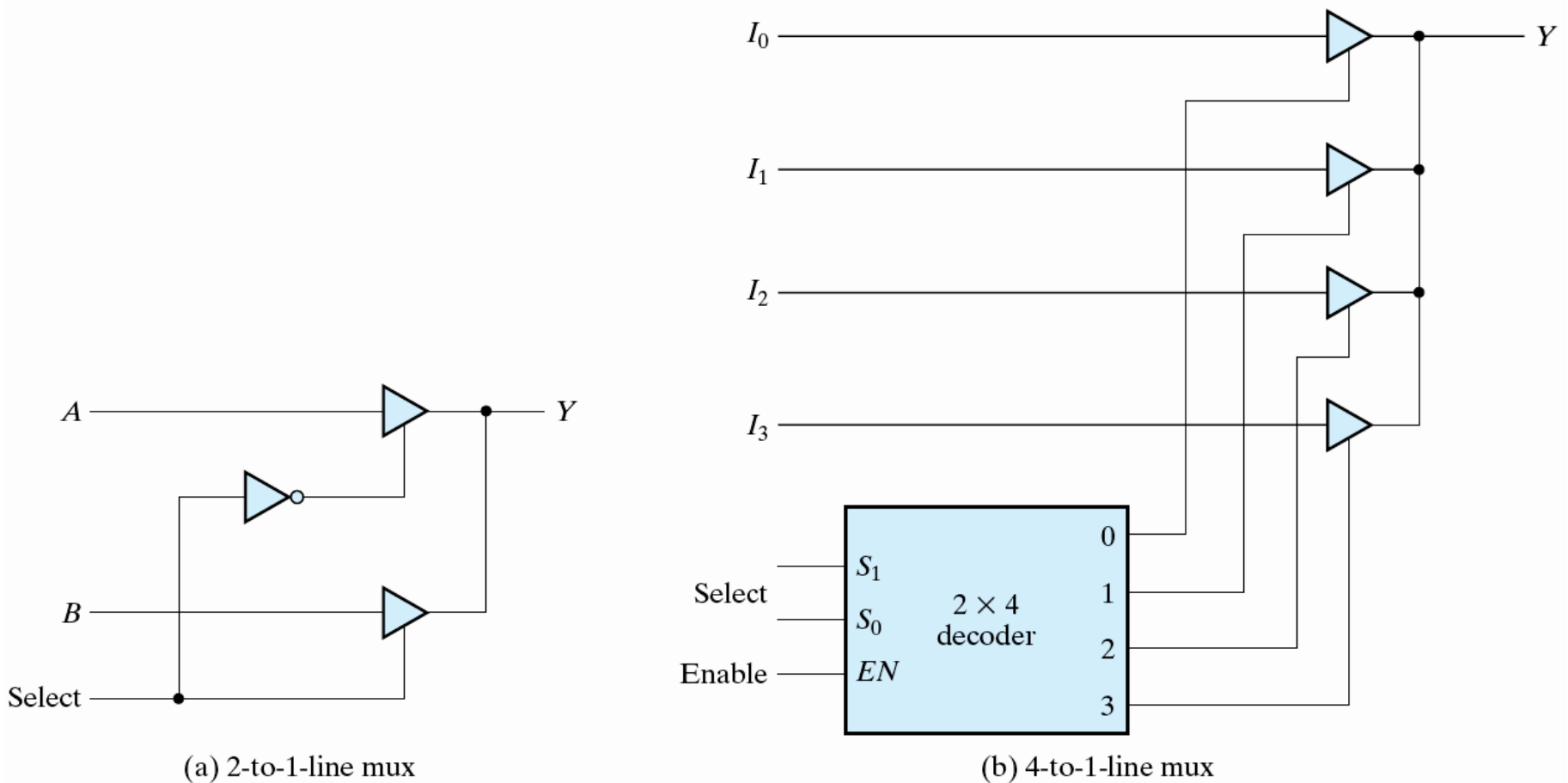


Fig. 4.30 Multiplexer with three-state gates