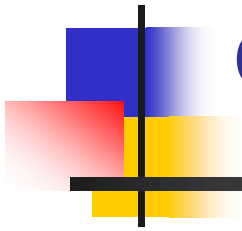
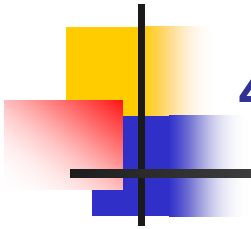


Combinational Logic

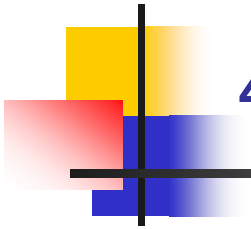
Chapter 4

Reference : Digital Design: With an Introduction to the Verilog HDL, VHDL, and SystemVerilog, 6th Edition
M. Morris R. Mano, Michael D. Ciletti



4.1 Introduction

- 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

- 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

- A combinational circuits
  - 2^n possible combinations of input values

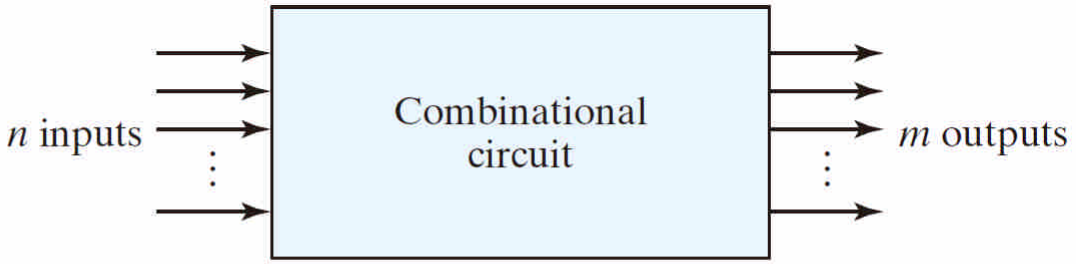
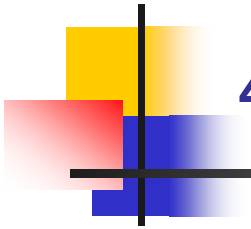


FIGURE 4.1
Block diagram of combinational circuit

  - Specific functions
    - Adders, subtractors, comparators, decoders, encoders, and multiplexers
    - MSI circuits or standard cells



4-3 Analysis Procedure

- A combinational circuit
  - make sure that it is combinational not sequential
    - No feedback path
  - derive its Boolean functions (truth table)
  - design verification
  - a verbal explanation of its function

A straight-forward procedure

$F_2 = AB+AC+BC$

$T_1 = A+B+C$

$T_2 = ABC$

$T_3 = F2'T1$

$F_1 = T3+T2$

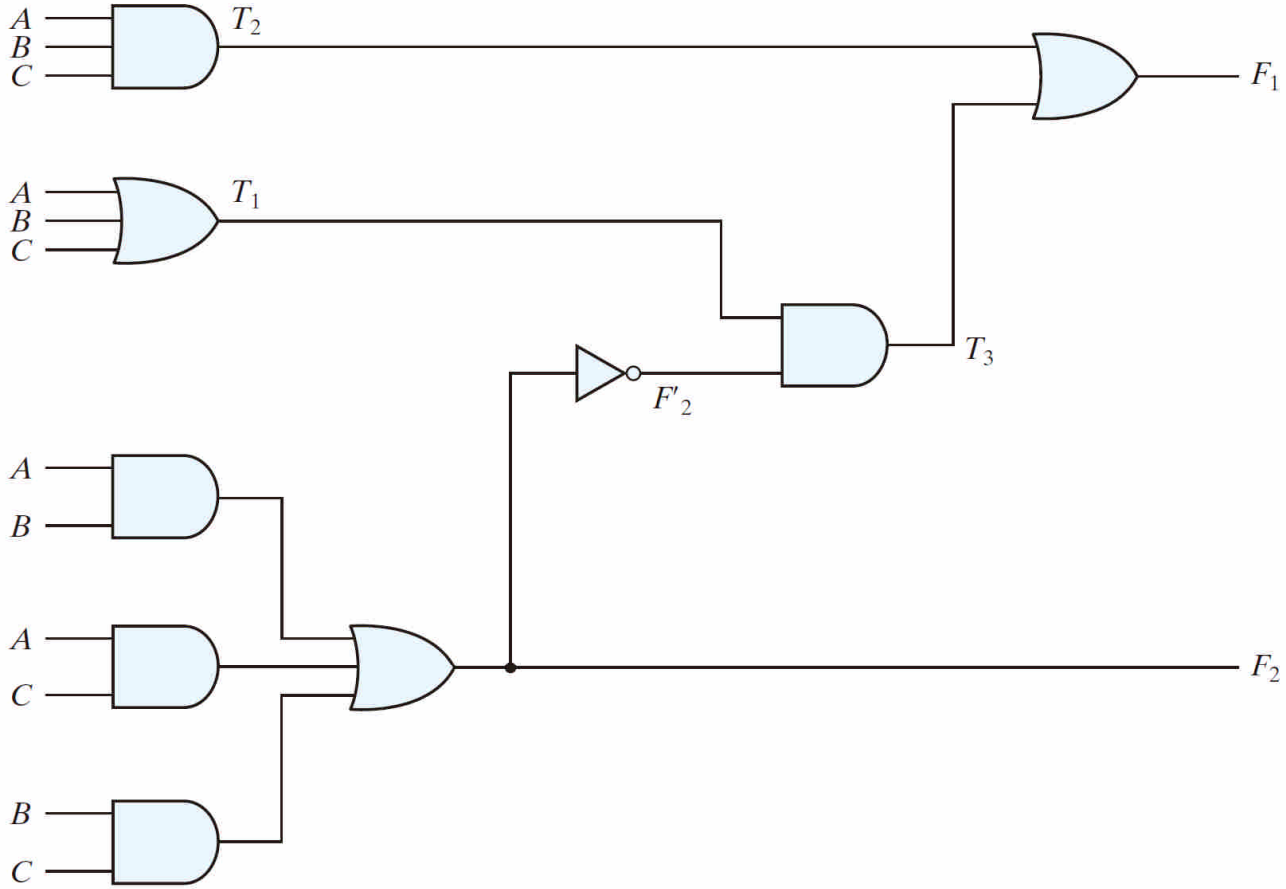
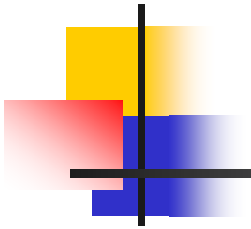


FIGURE 4.2
Logic diagram for analysis example



- $F_1 = T_3+T_2 = F_2'T_1+ABC$
  $= (AB+AC+BC)'(A+B+C)+ABC$
  $= (A'+B')(A'+C')(B'+C')(A+B+C)+ABC$
  $= (A'+B'C')(AB'+AC'+BC'+B'C)+ABC$
  $= A'BC'+A'B'C+AB'C'+ABC$
- A full-adder
  - F_1: the sum
  - F_2: the carry

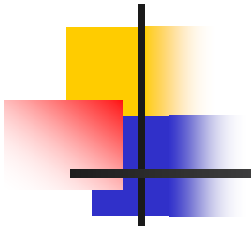
- The truth table

Table 4.1
Truth Table for the Logic Diagram of Fig. 4.2

A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
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

- The design procedure of combinational circuits
  - State the problem (system 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



- 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

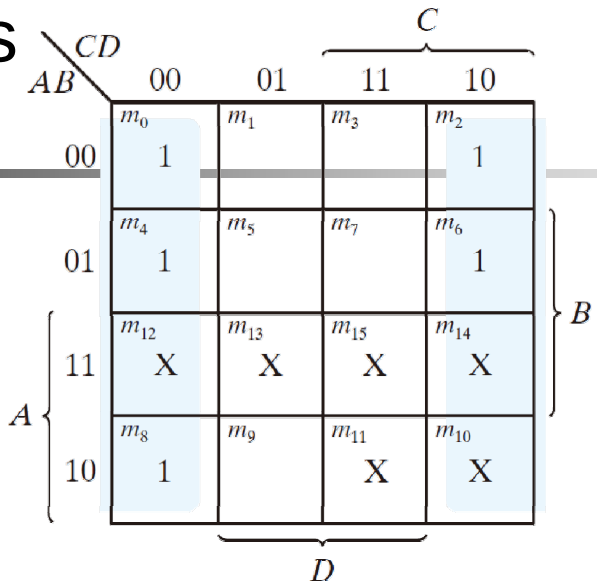
Code conversion example

- BCD to excess-3 code
  - The truth table

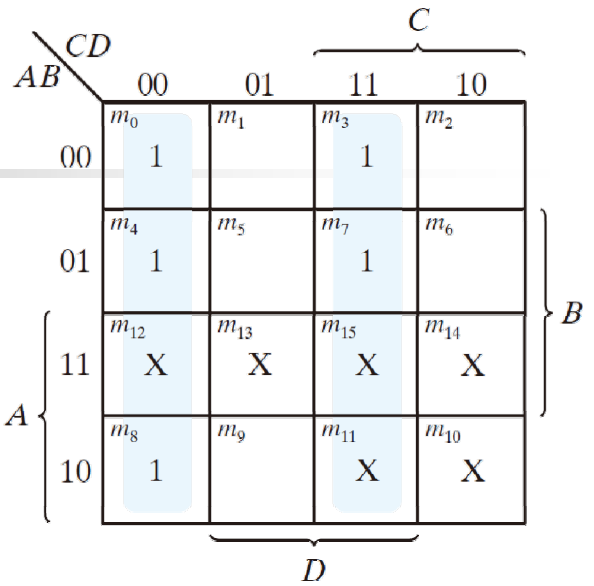
Table 4.2
Truth Table for Code Conversion Example

Input BCD				Output Excess-3 Code			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
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

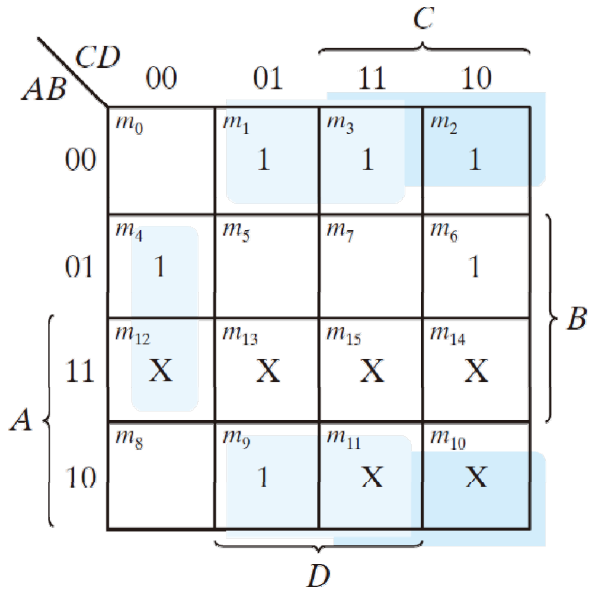
The maps



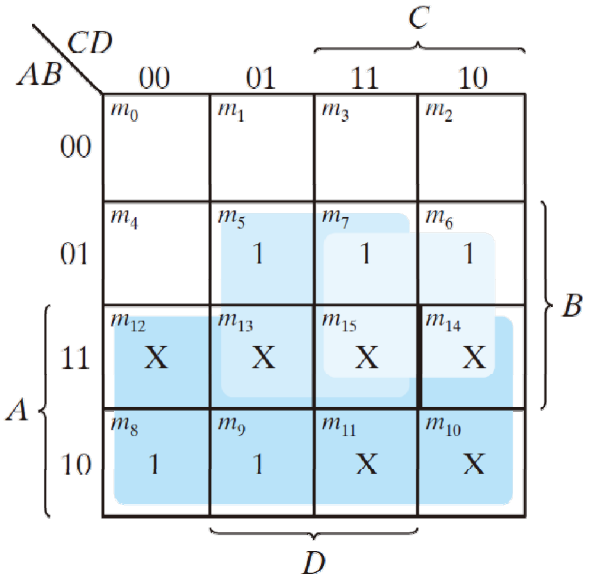
$z = D'$



$y = CD + C'D'$

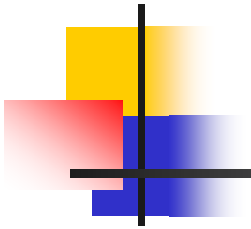


$x = B'C + B'D + BC'D'$



$w = A + BC + BD$

FIGURE 4.3
Maps for BCD-to-excess-3 code converter



- 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' = CD + (C+D)'
    x = B'C + B'D+BC'D‘ = B'(C+D) +B(C+D)'
    w = A+BC+BD

The logic diagram

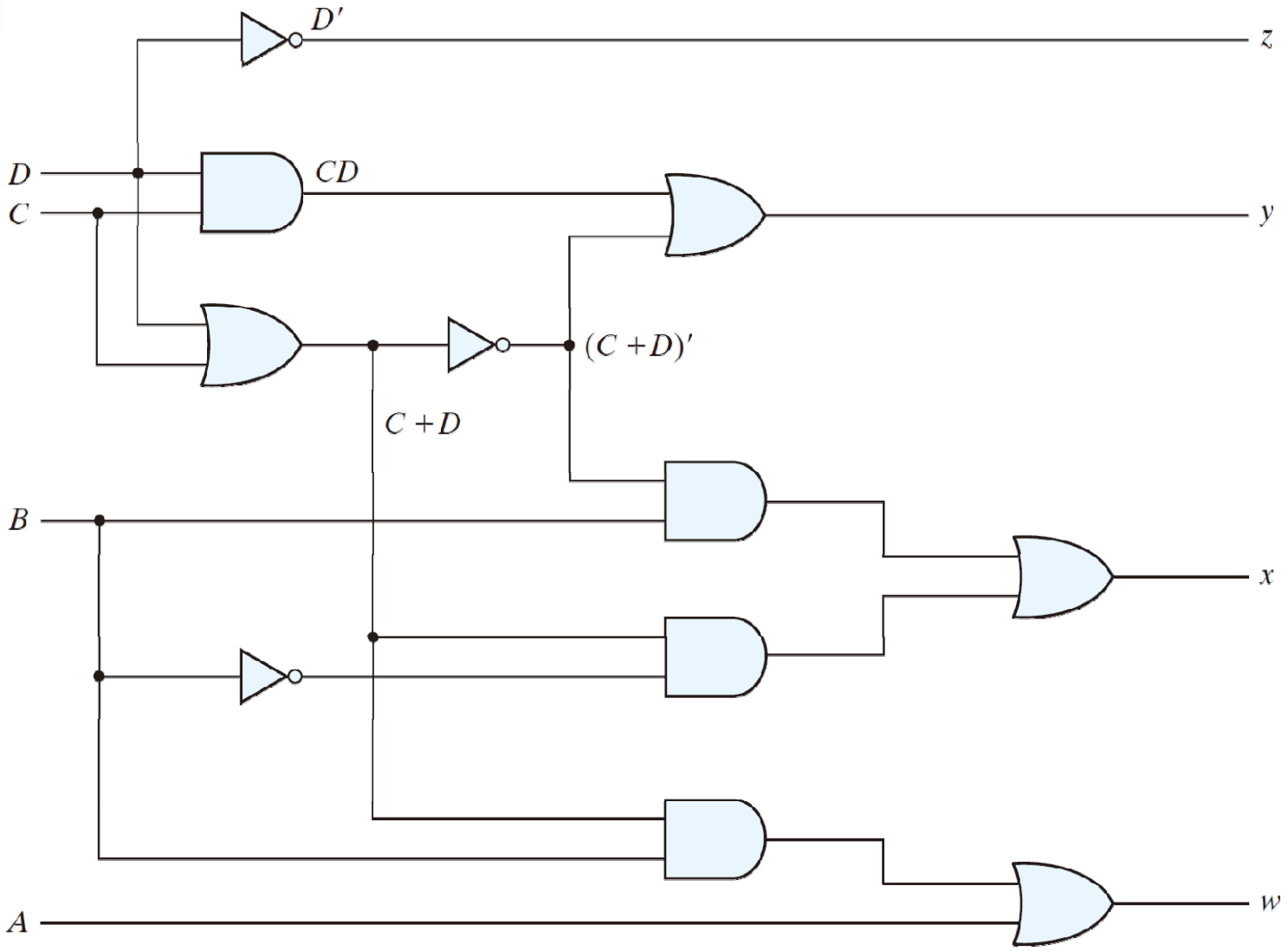


FIGURE 4.4
Logic diagram for BCD-to-excess-3 code converter

4-5 Binary Adder-Subtractor

- 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

- S = x'y+xy'
  C = xy
- the flexibility for implementation
- S=x⊕y
- S = (x+y)(x'+y')
- S' = xy+x'y'
  S = (C+x'y')'
- C = xy = (x'+y')'

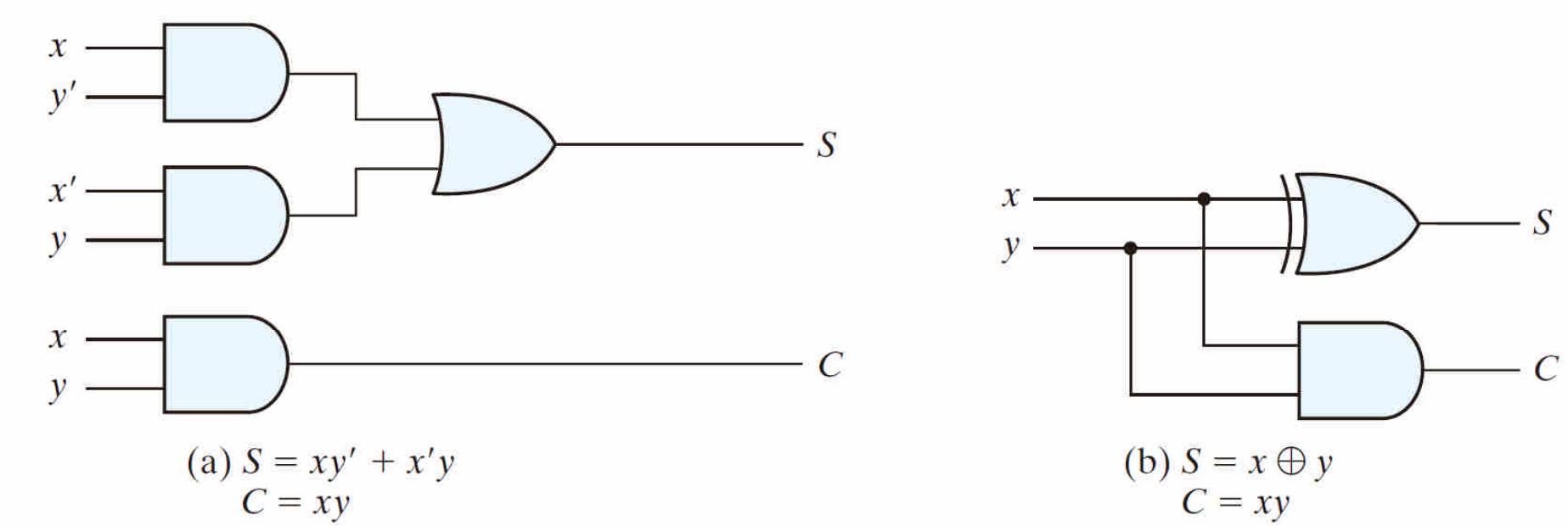


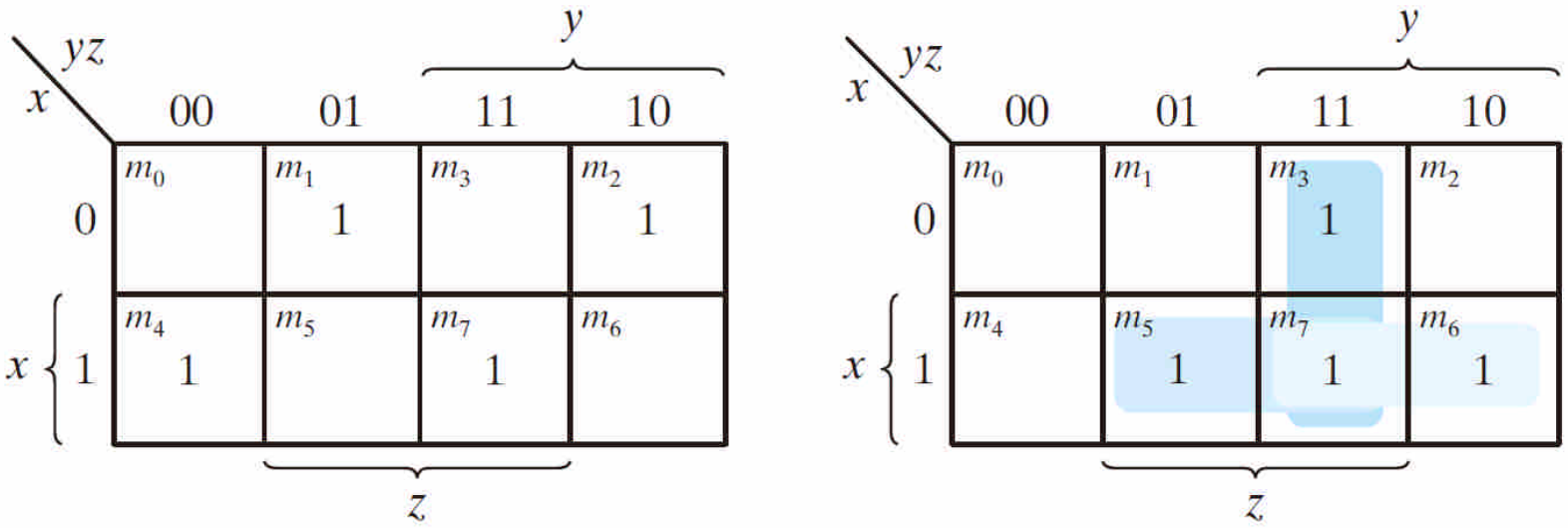
FIGURE 4.5
Implementation of half adder

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



(a) $S = x'y'z + x'yz' + xy'z' + xyz$

(b) $C = xy + xz + yz$

FIGURE 4.6
K-Maps for full adder

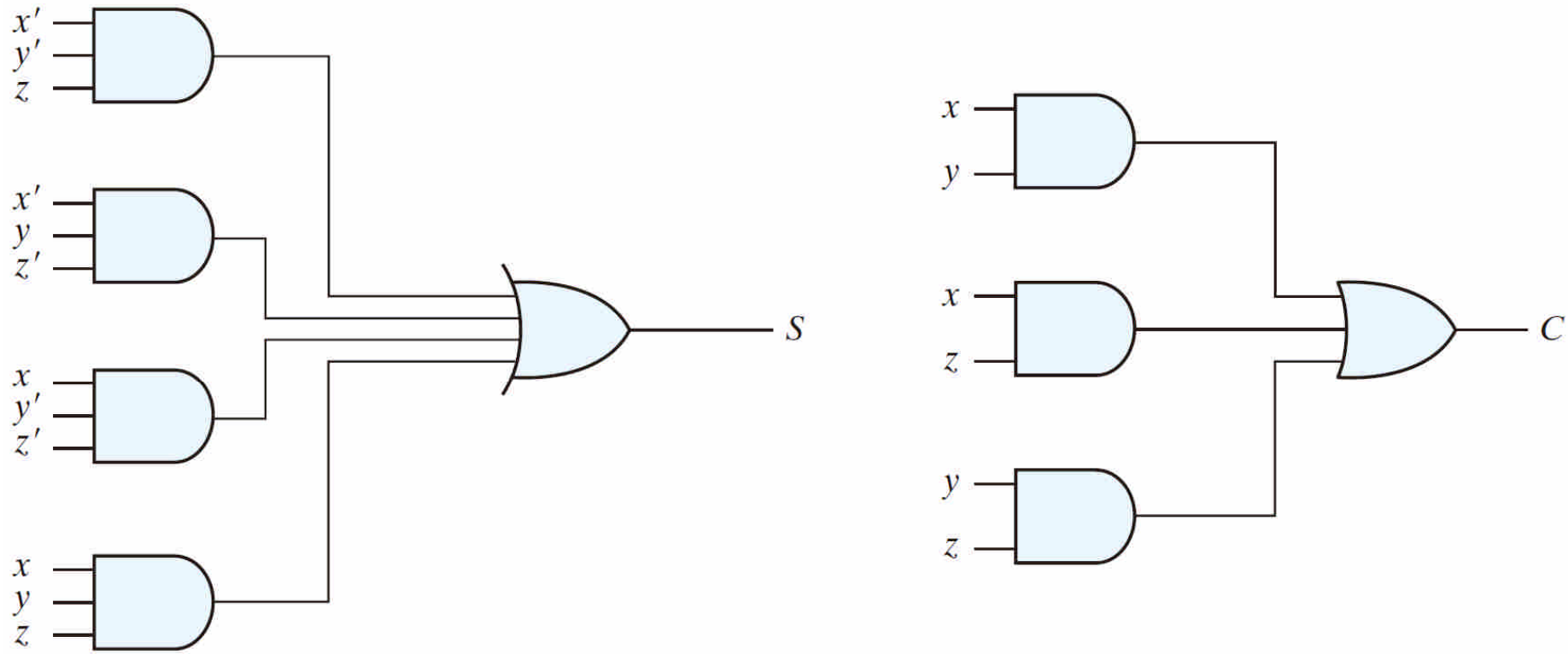


FIGURE 4.7
Implementation of full adder in sum-of-products form

- S = x'y'z+x'yz'+ xy'z'+xyz
  C = xy + xz + yz
- S = z⊕ (x⊕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(xy'+x'y)+xy
  = xy'z+x'yz+ xy

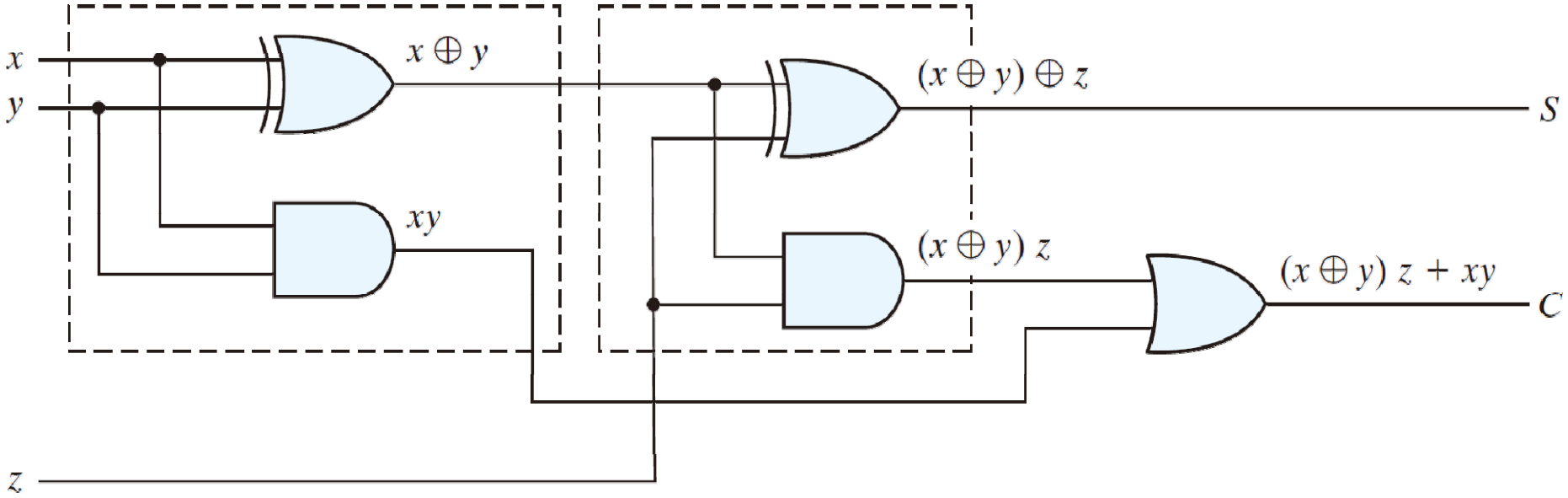


FIGURE 4.8
Implementation of full adder with two half adders and an OR gate

Binary adder

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}

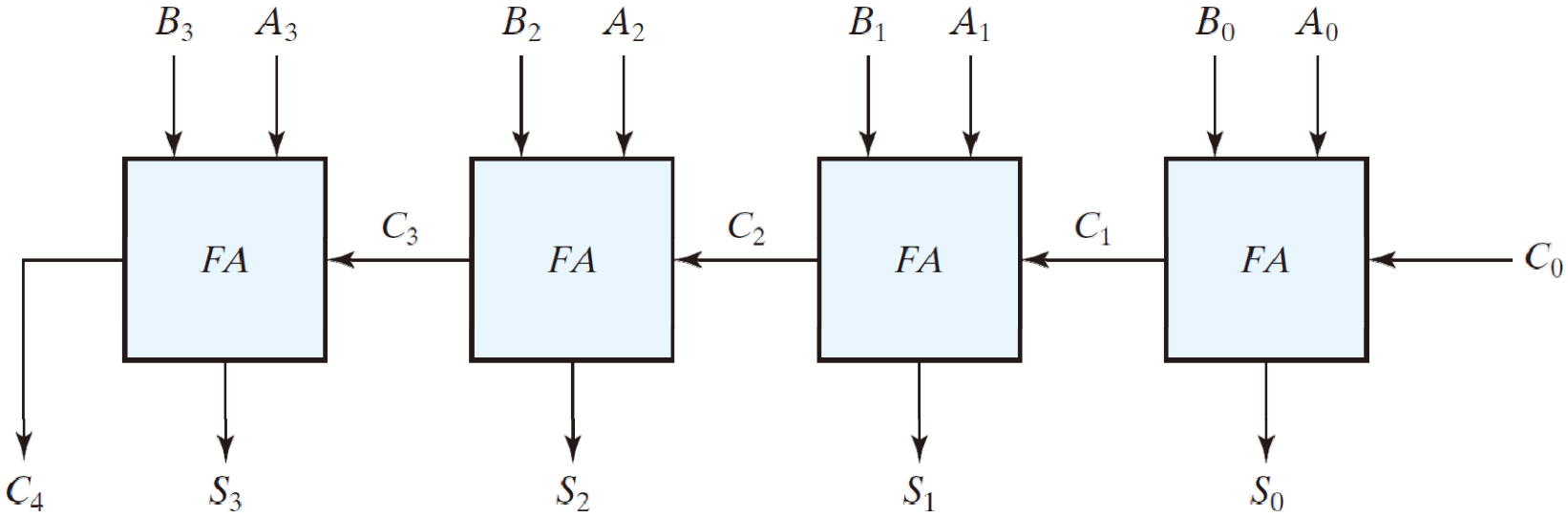


FIGURE 4.9
Four-bit adder

Carry propagation

- when the correct outputs are available
- the critical path counts (the worst case)
- $(A_1,B_1,C_1) > C_2 > C_3 > C_4 > (C_5,S_4)$
- > 8 gate levels

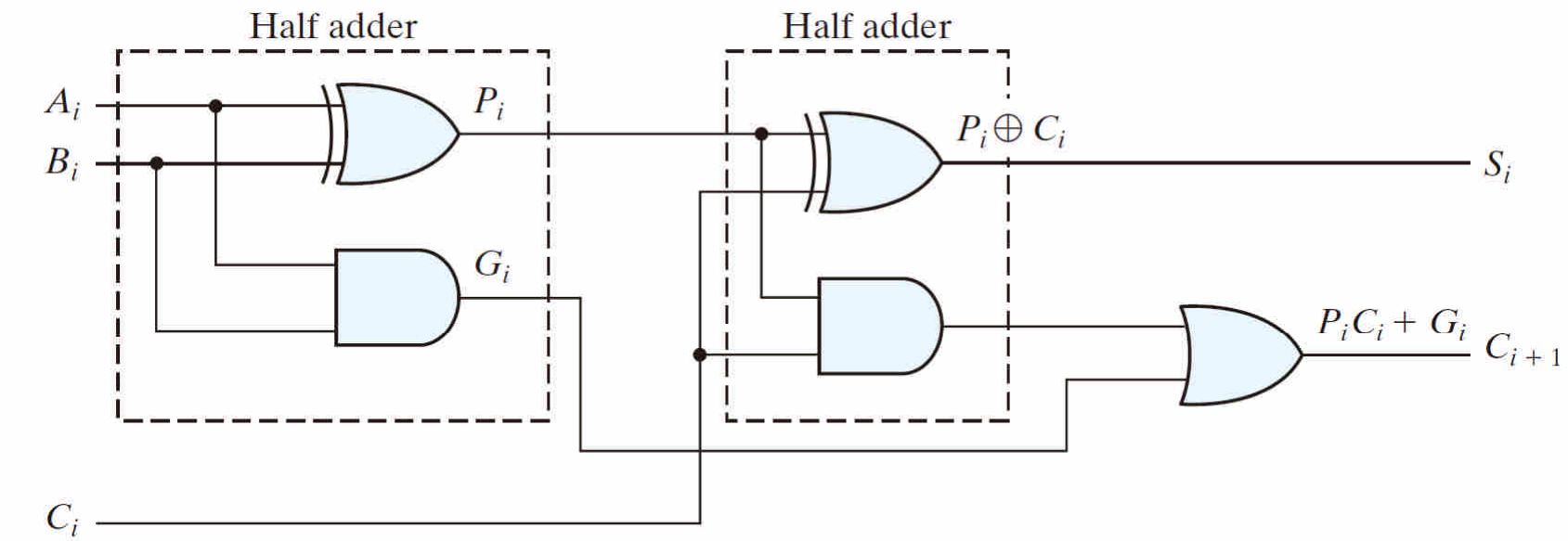
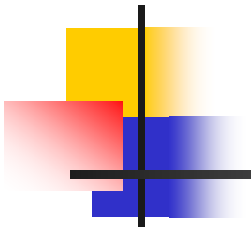


FIGURE 4.10
Full adder with *P* and *G* shown



- Reduce the carry propagation delay
  - employ faster gates
  - look-ahead carry (more complex mechanism, yet faster)
  - 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: $C_{i+1} = G_i + P_i C_i$
  - $C_1 = G_0 + P_0 C_0$
  - $C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0)$
    $= G_1 + P_1 G_0 + P_1 P_0 C_0$
  - $C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

- Logic diagram

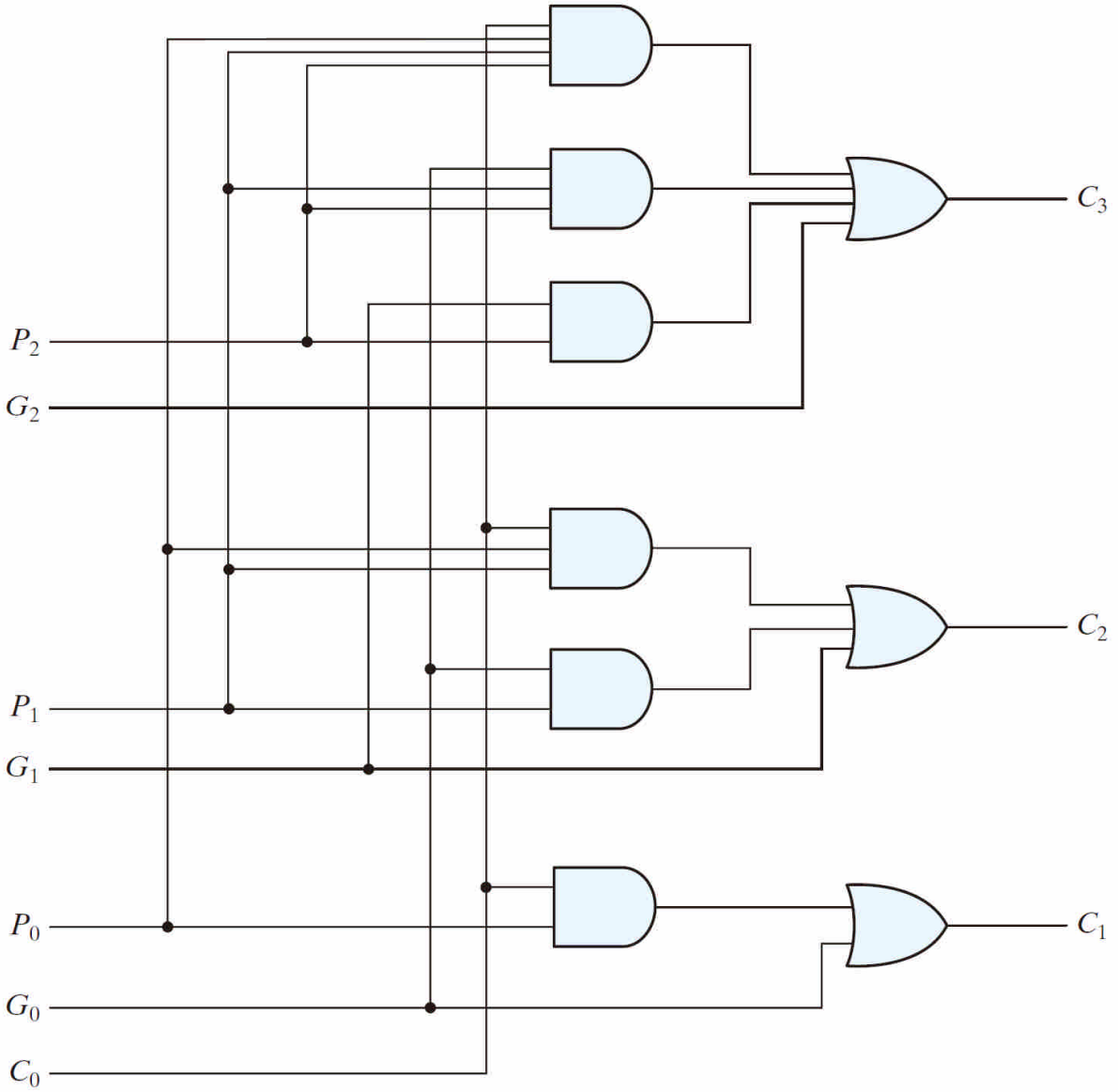


FIGURE 4.11
Logic diagram of carry lookahead generator

4-bit carry-look ahead adder

- propagation delay

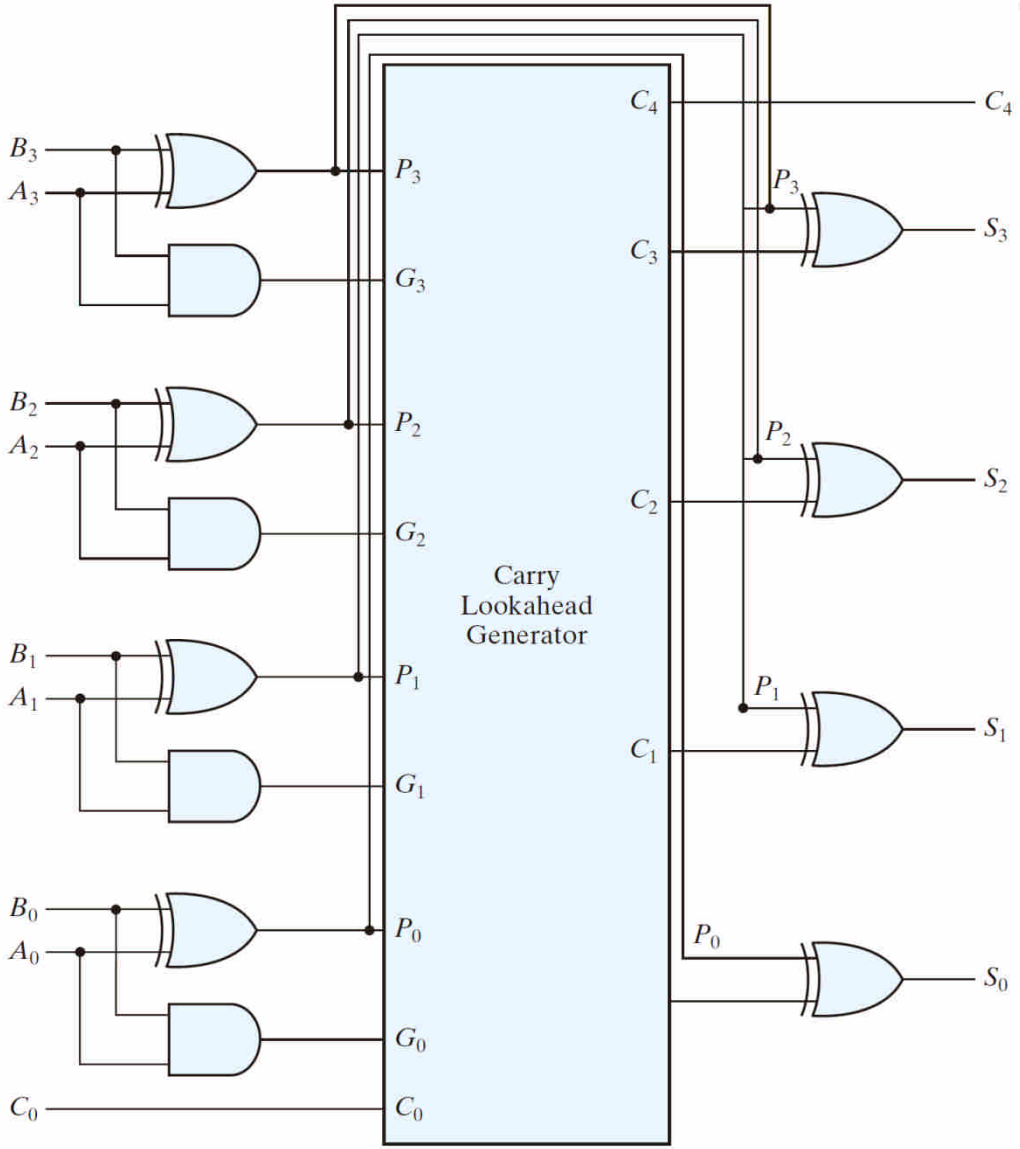


FIGURE 4.12
Four-bit adder with carry lookahead

Binary subtractor

- A-B = A+(2's complement of B)
- 4-bit Adder-subtractor
  - M=0, A+B; M=1, A+B'+1

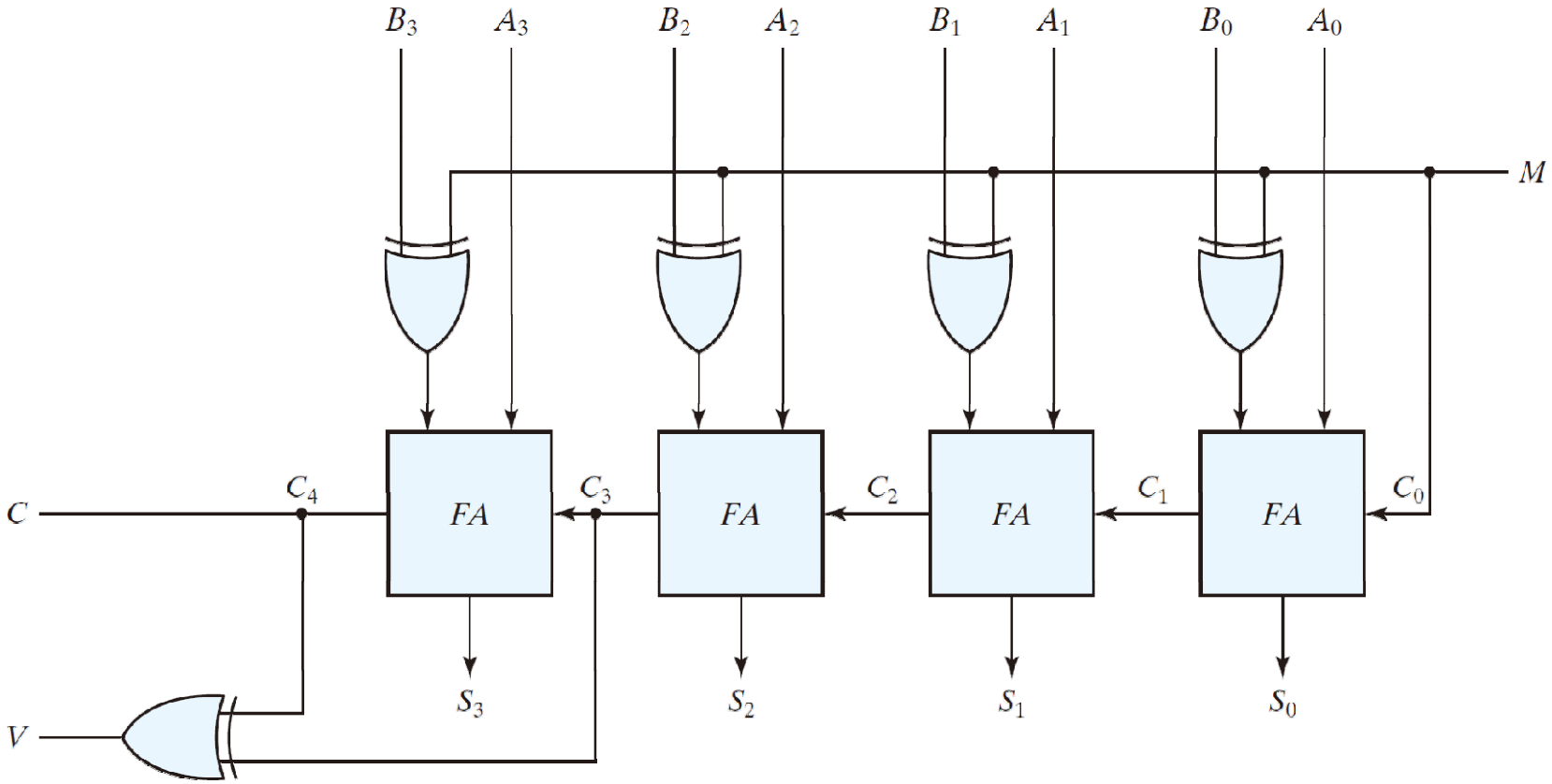


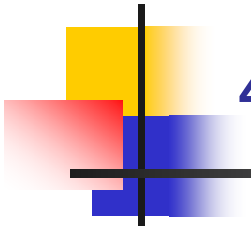
FIGURE 4.13
Four-bit adder–subtractor (with overflow detection)

Overflow

- The storage 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:	0 1	carries:	1 0
+70	0 1000110	−70	1 0111010
+80	0 1010000	−80	1 0110000
+150	1 0010110	−150	0 1101010



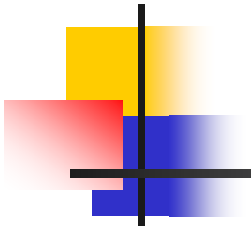
4-6 Decimal Adder

- 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 binary full Adders
    - the sum <= 9 + 9 + 1 = 19
    - binary to BCD

BCD Adder: The truth table

Table 4.5
Derivation of BCD Adder

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
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



- Modifications are needed if the sum > 9
  - C = 1
    - K = 1
    - Z_8Z_4 = 1
    - Z_8Z_2 = 1
  - modification: $-(10)_d$ or +6

$C = K + Z_8Z_4 + Z_8Z_2$

Block diagram

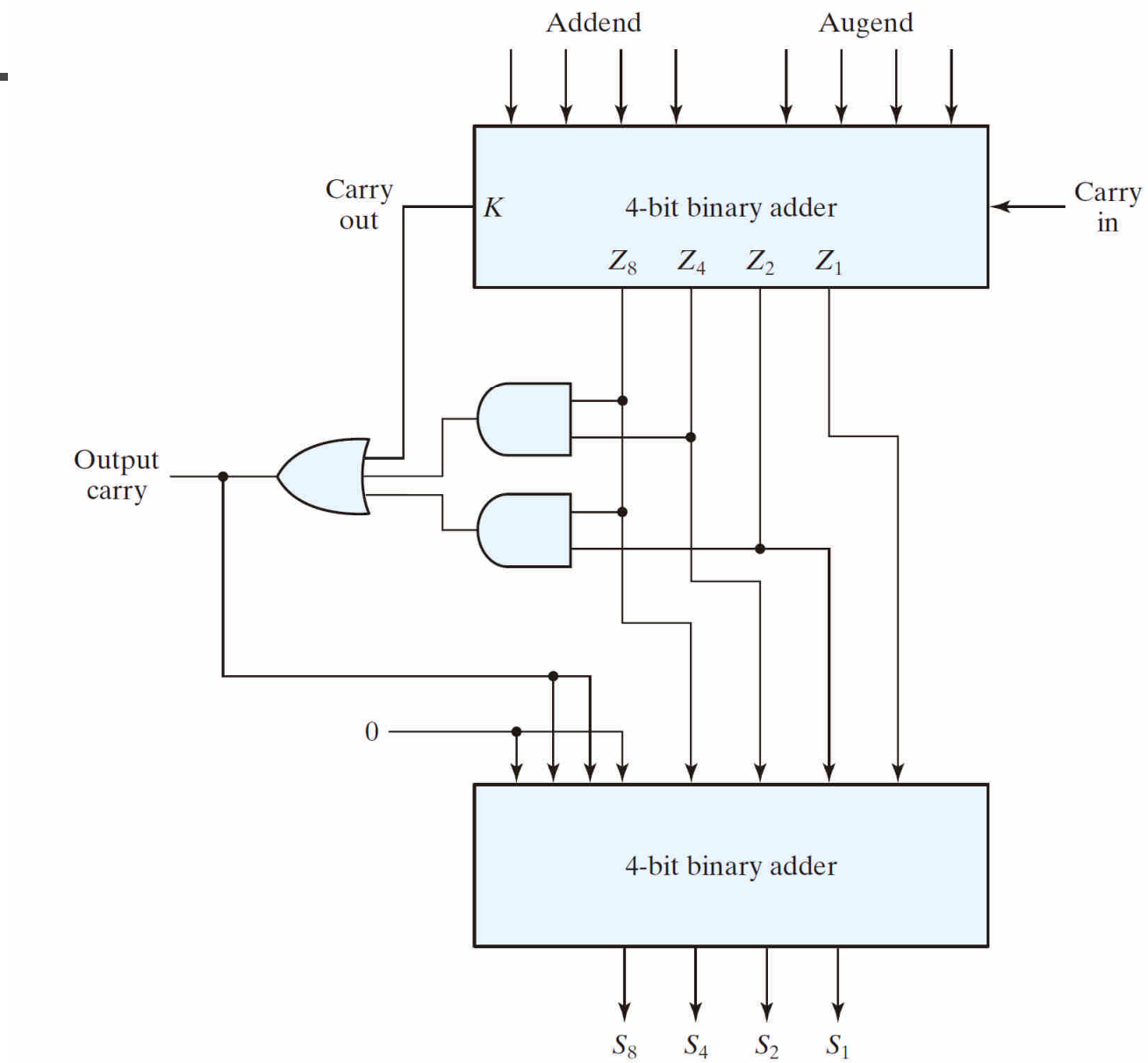


FIGURE 4.14
Block diagram of a BCD adder

Binary Multiplier

- Partial products
  - AND operations

$$
\begin{array}{cccc}
 & & B_1 & B_0 \\
 & & A_1 & A_0 \\
\hline
 & & A_0B_1 & A_0B_0 \\
 & A_1B_1 & A_1B_0 & \\
\hline
C_3 & C_2 & C_1 & C_0
\end{array}
$$

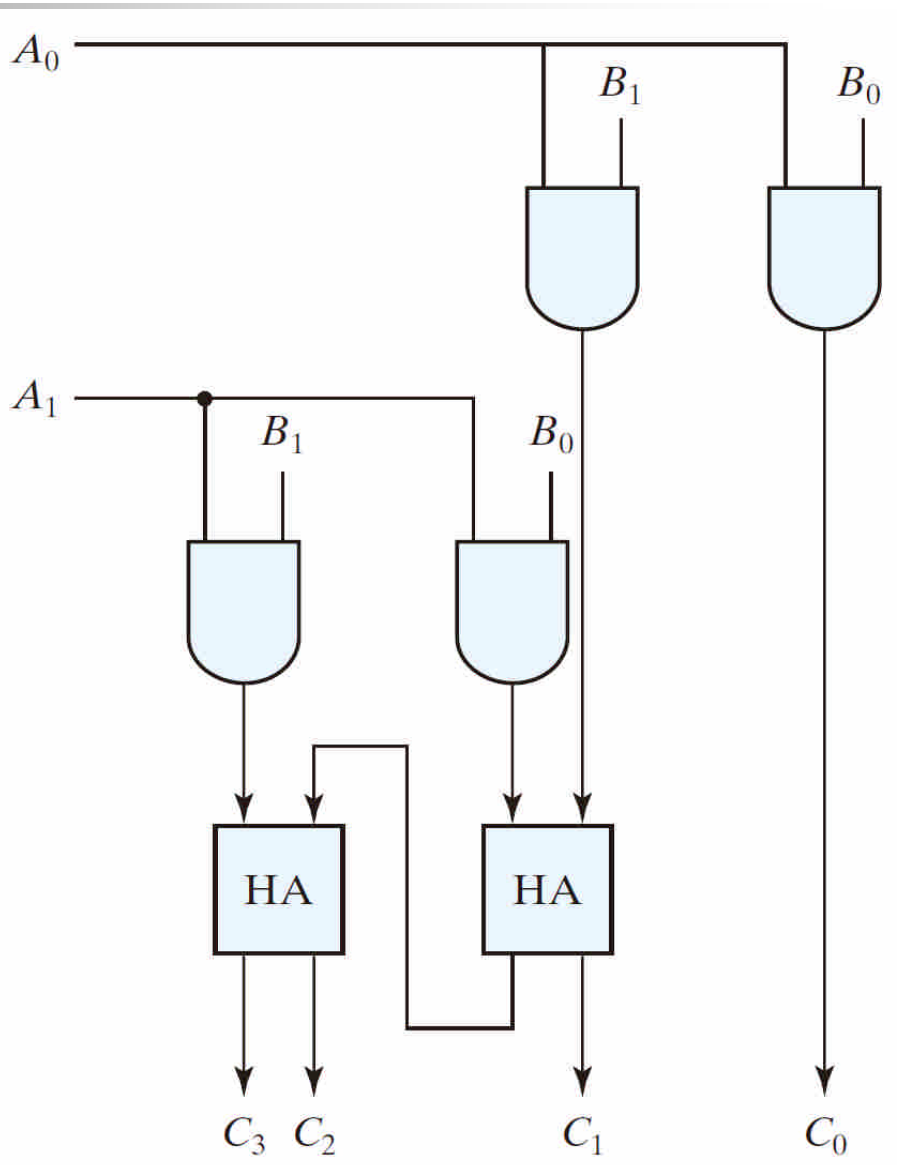


FIGURE 4.15
Two-bit by two-bit binary multiplier

4-bit by 3-bit binary multiplier

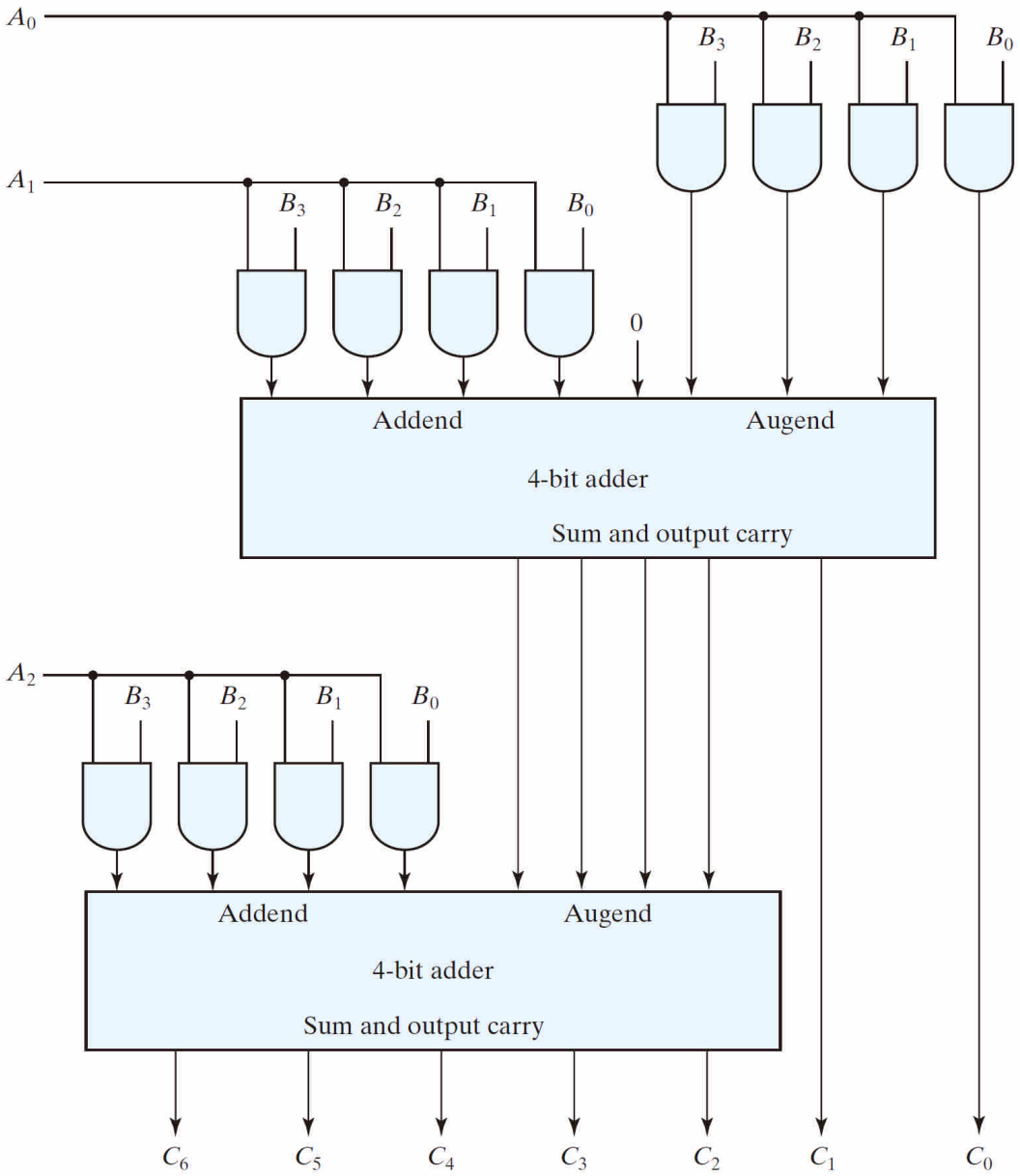
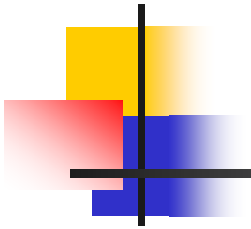


FIGURE 4.16
Four-bit by three-bit binary multiplier

4-8 Magnitude Comparator

- The comparison of two numbers
  - outputs: A>B, A=B, A<B
- Design Approaches
  - the truth table
    - 2^{2n} entries - too cumbersome for large n
  - use inherent regularity of the problem
    - reduce design efforts
    - reduce human errors



- Algorithm -> logic
  - $A = A_3A_2A_1A_0$; $B = B_3B_2B_1B_0$
  - A=B if $A_3=B_3$, $A_2=B_2$, $A_1=B_1$and $A_1=B_1$
    - equality: $x_i= A_iB_i+A_i'B_i'$, for *i* = 0, 1, 2, 3
    - $(A=B) = x_3x_2x_1x_0$
  - $(A>B) = A_3B_3'+x_3A_2B_2'+x_3x_2A_1B_1'+x_3x_2x_1\ A_0B_0'$
  - $(A>B) = A_3'B_3+x_3A_2'B_2+x_3x_2A_1'B_1+x_3x_2x_1\ A_0'B_0$
- Implementation
  - $x_i = (A_iB_i'+A_i'B_i)'$

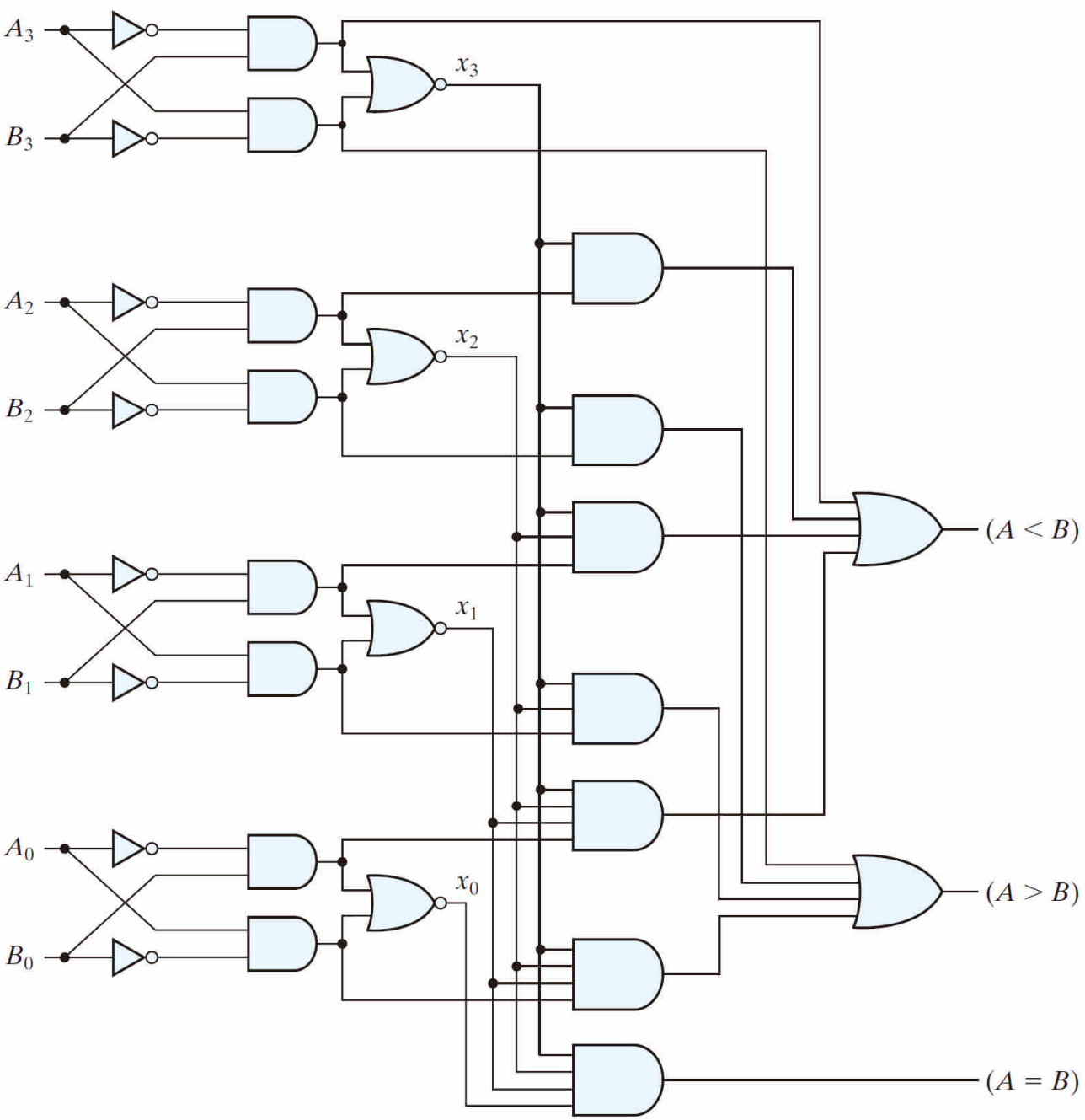


FIGURE 4.17
Four-bit magnitude comparator

4-9 Decoders

- 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

An implementation

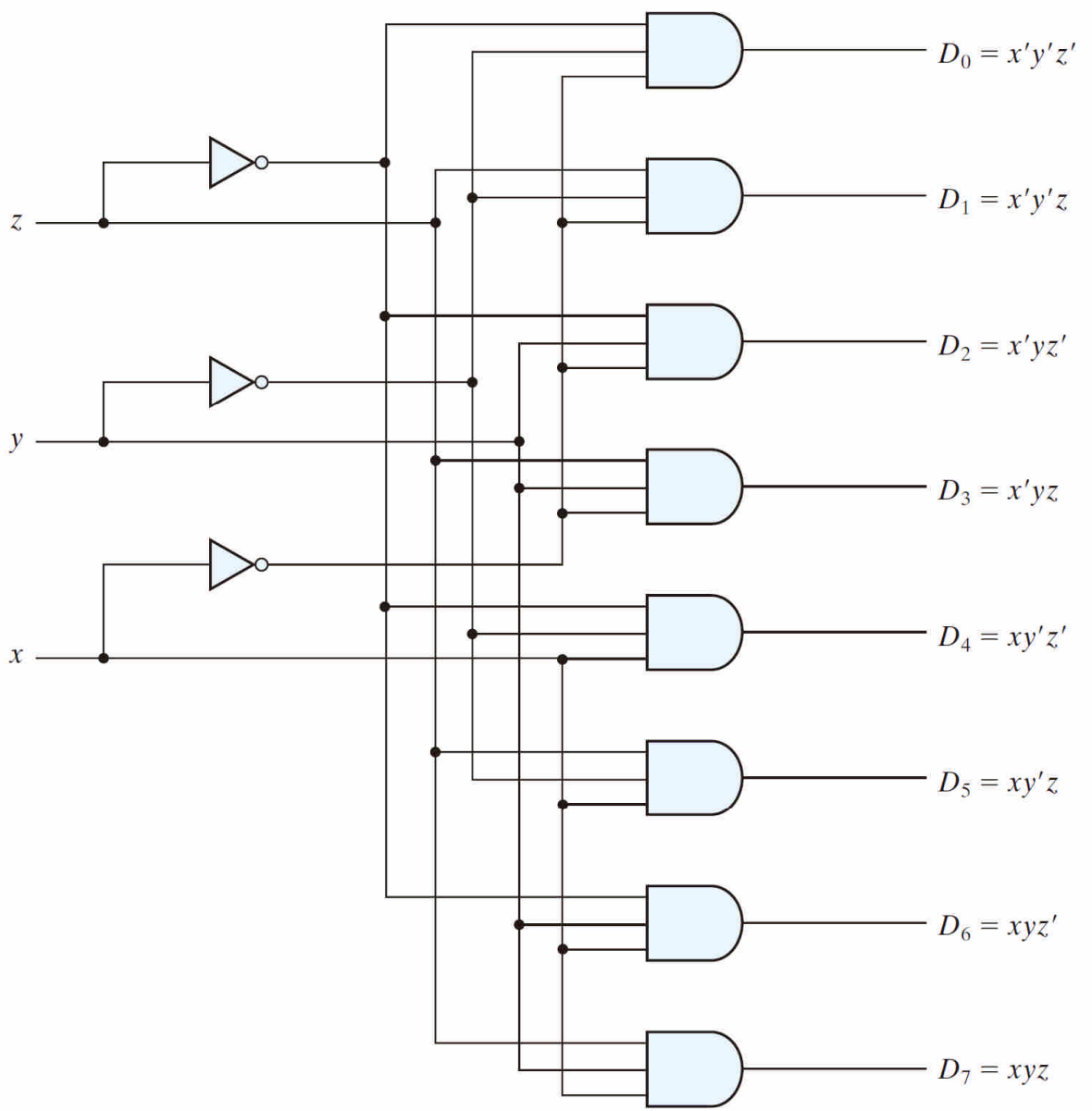
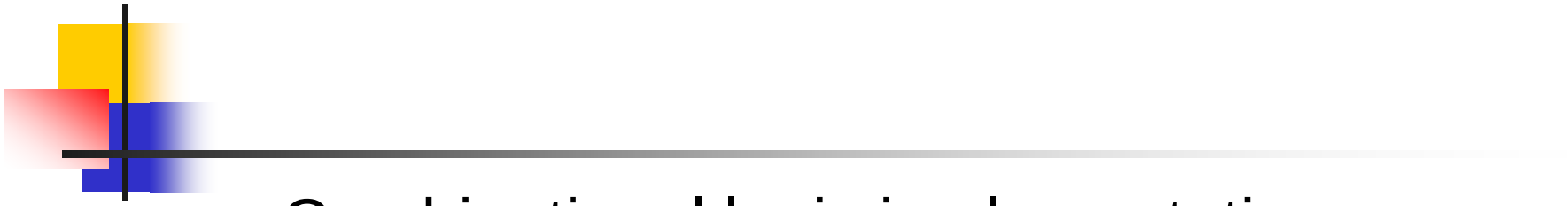


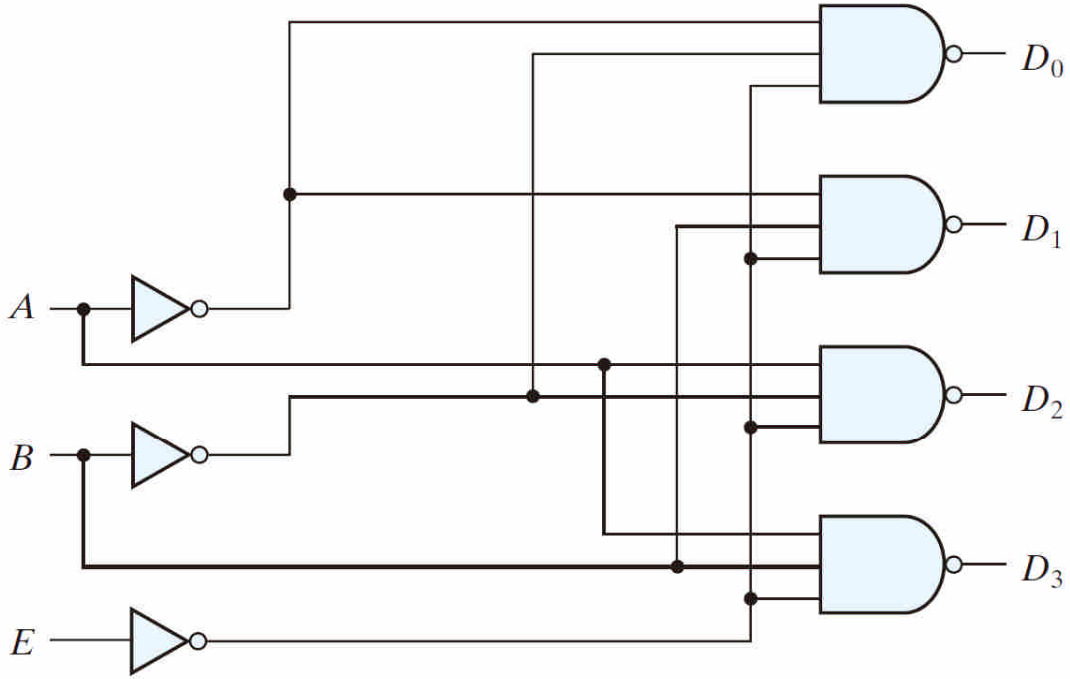
FIGURE 4.18
Three-to-eight-line decoder



- Combinational logic implementation
  - each output = a minterm
  - use a decoder and an external OR gate to implement any Boolean function of n input variables

Demultiplexers

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



(a) Logic diagram

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	1

(b) Truth table

FIGURE 4.19
Two-to-four-line decoder with enable input

Expansion

- two 3-to-8 decoder: a 4-to-16 deocder

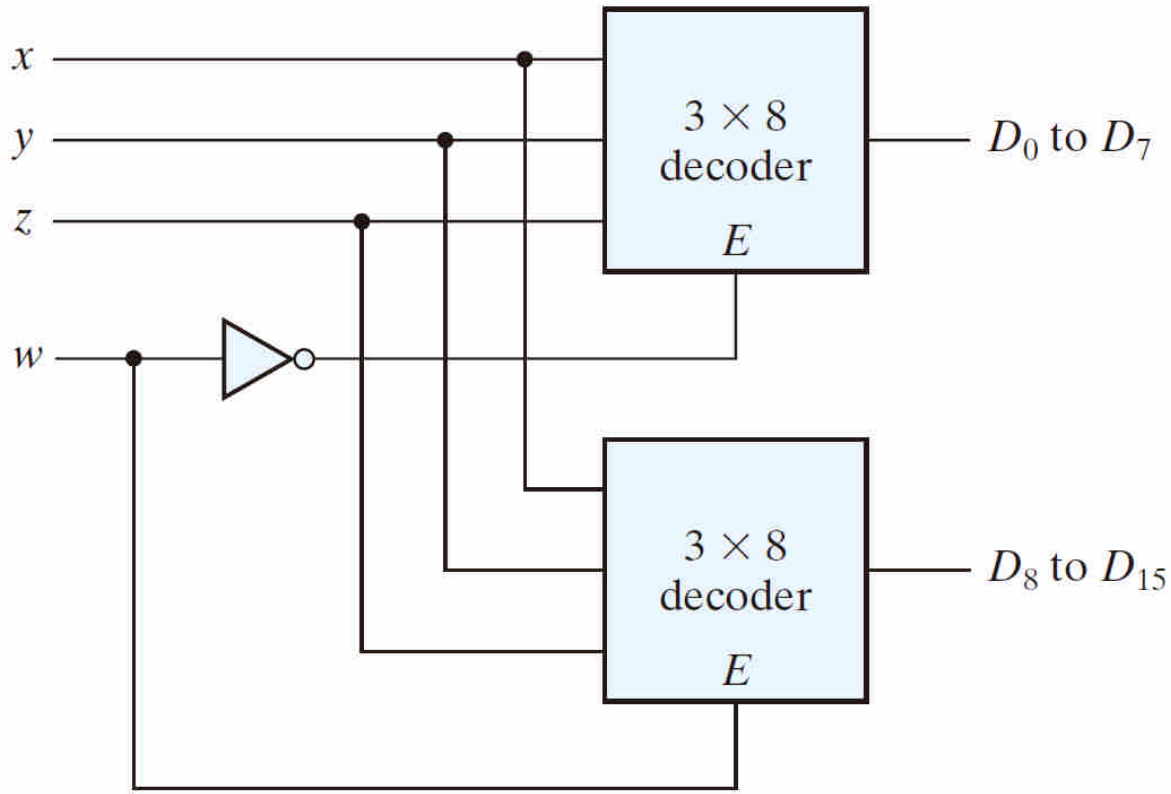


FIGURE 4.20
4 × 16 decoder constructed with two 3 × 8 decoders

- a 5-to-32 decoder?

Combination 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,x)=Σ(1,2,4,7)
    C(x,y,z)= Σ(3,5,6,7)

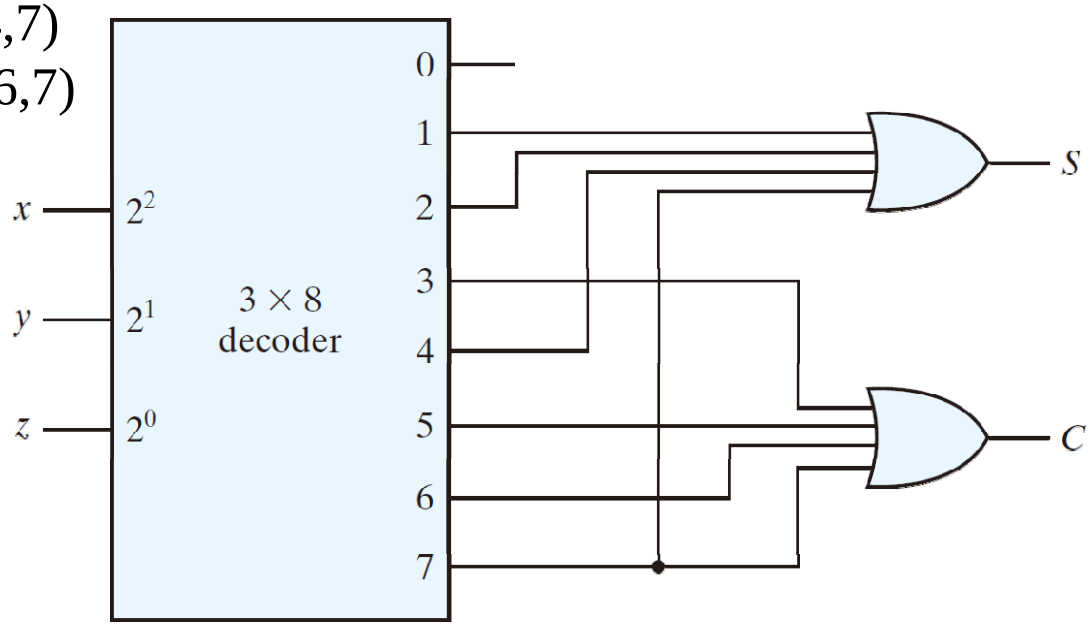
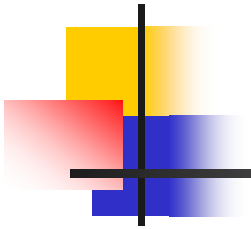


FIGURE 4.21
Implementation of a full adder with a decoder



- two possible approaches using decoder
  - OR(minterms of F): k inputs
  - NOR(minterms of F'): 2^n – k inputs
- In general, it is not a practical implementation

4-10 Encoders

- 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.

Priority Encoder

- 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
- V: valid output indicator

The maps for simplifying outputs x and y

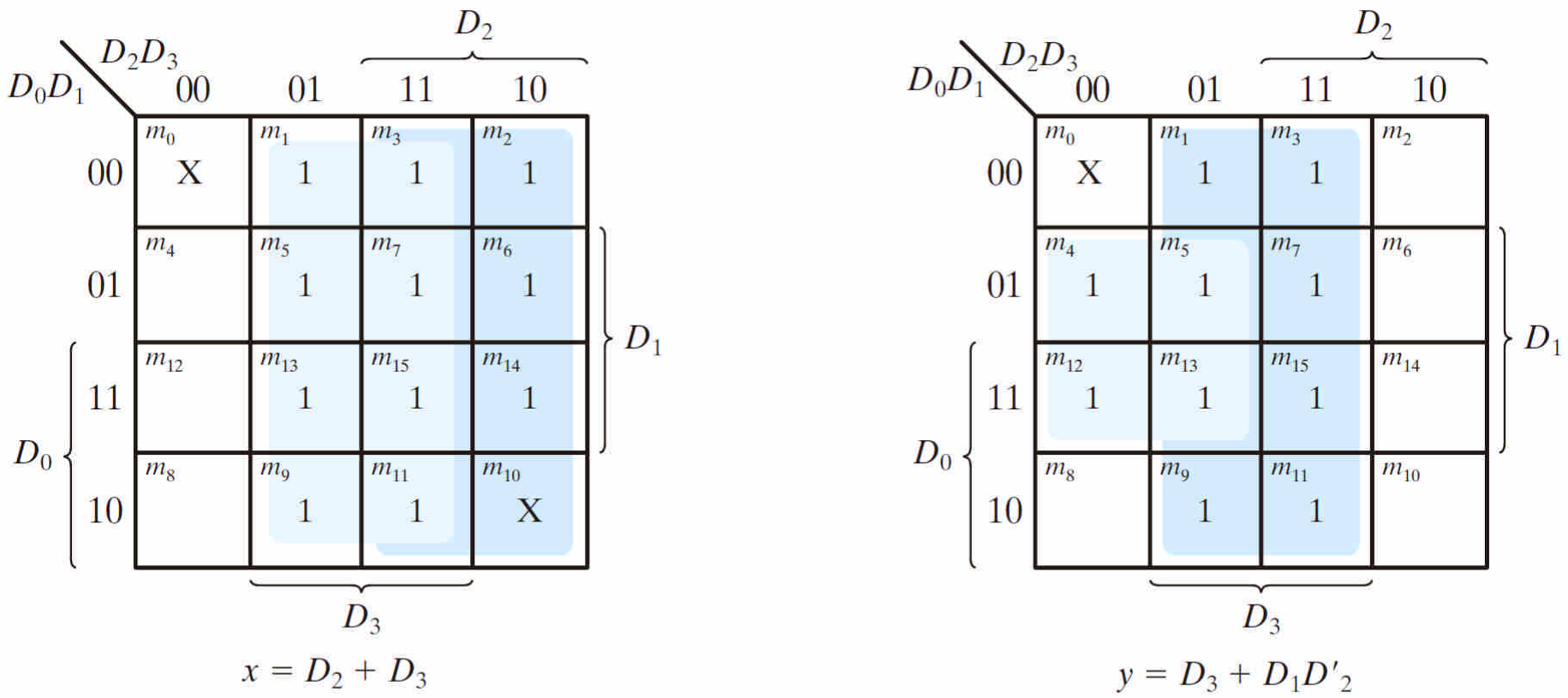


FIGURE 4.22
Maps for a priority encoder

Implementation of priority

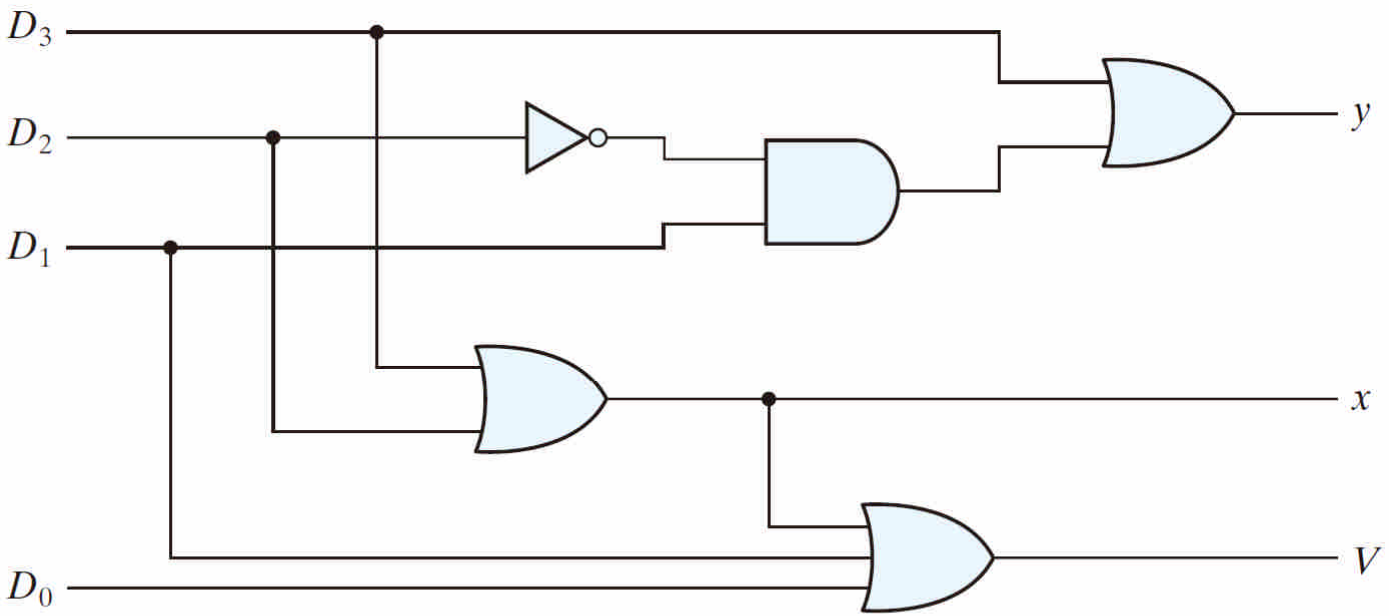
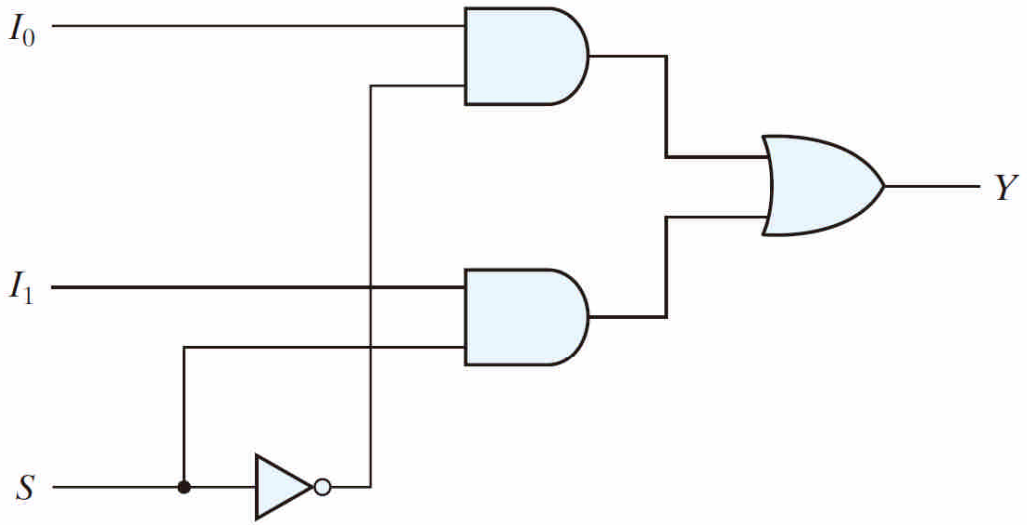


FIGURE 4.23
Four-input priority encoder

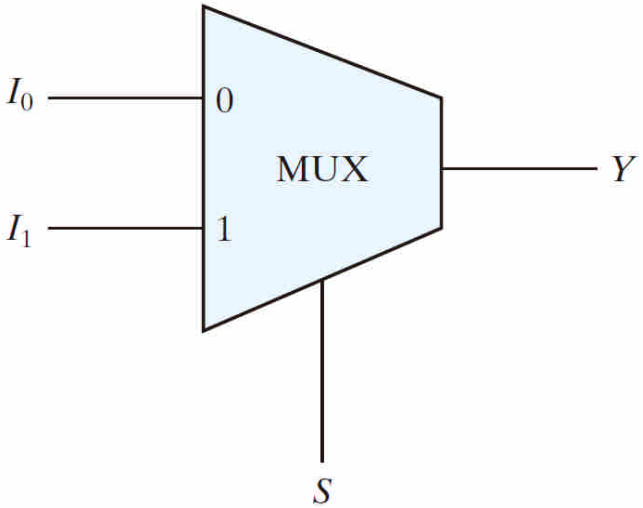
$$x = D_2 + D_3$$
$$y = D_3 + D_1 D_2'$$
$$V = D_0 + D_1 + D_2 + D_3$$

4-11 Multiplexers

- 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



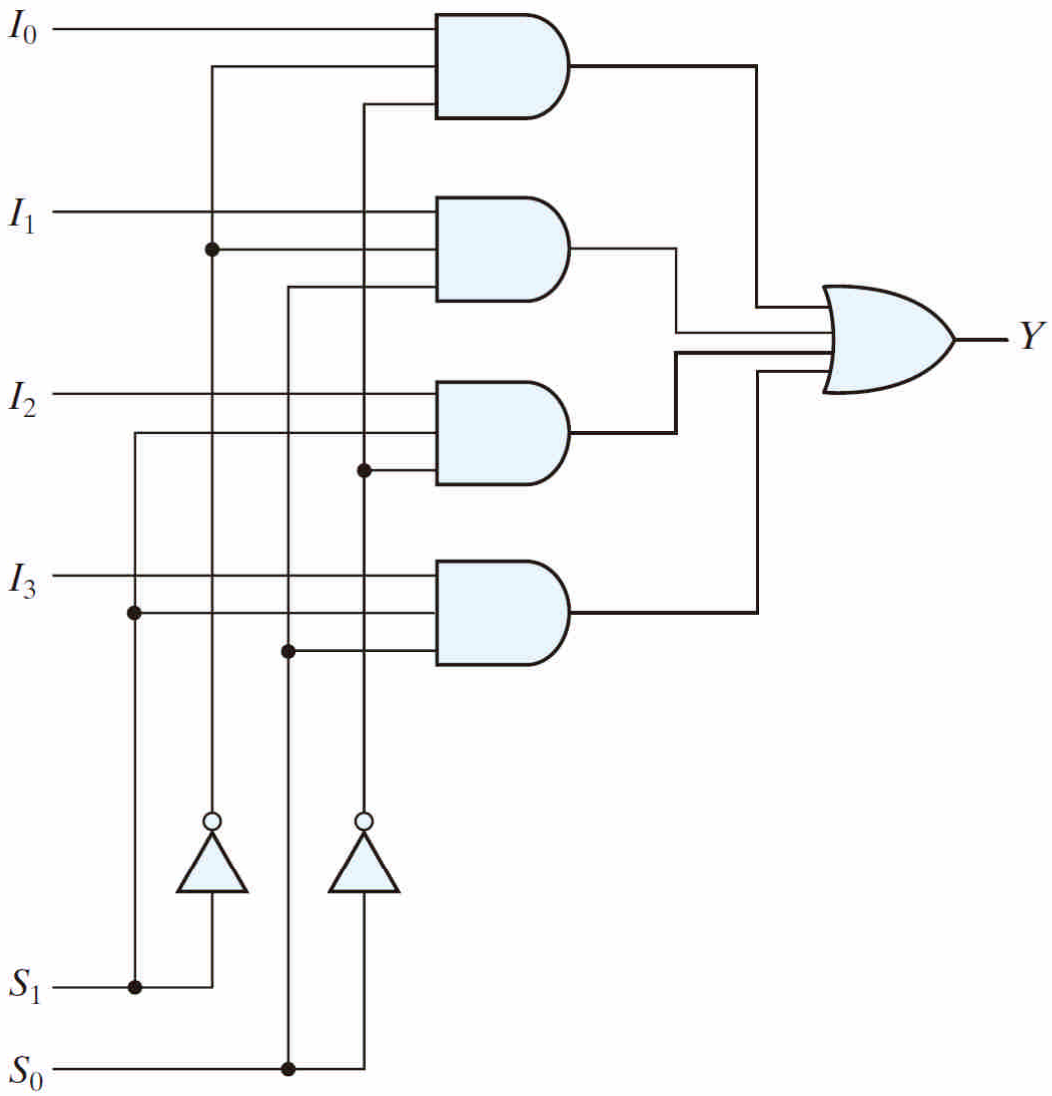
(a) Logic diagram



(b) Block diagram

FIGURE 4.24
Two-to-one-line multiplexer

- 4-to-1-line multiplexer

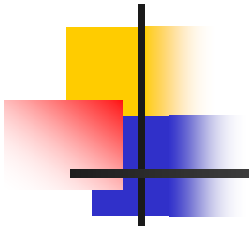


(a) Logic diagram

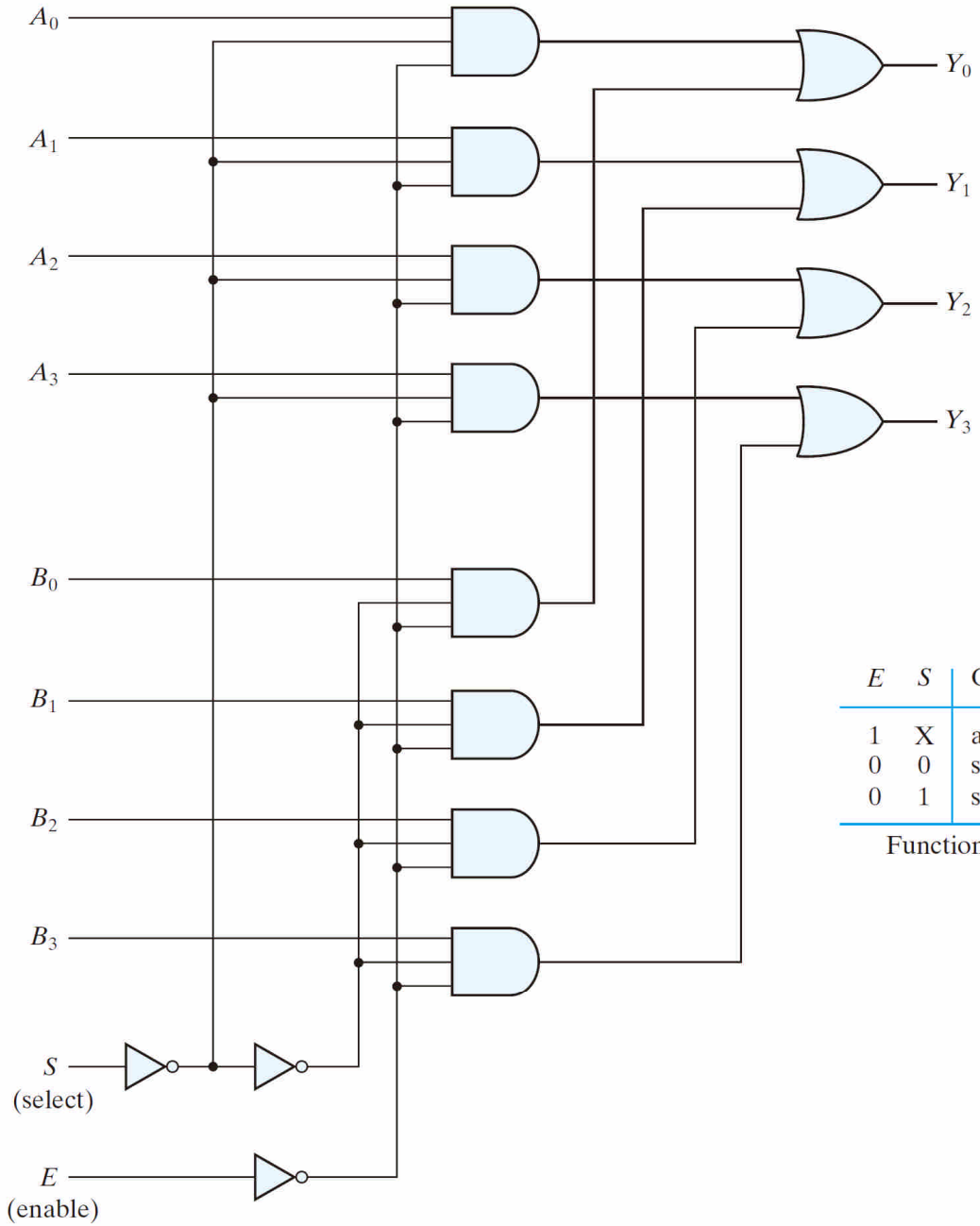
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table

FIGURE 4.25
Four-to-one-line multiplexer



- Note
  - n-to- 2^n decoder
  - add the 2^n input lines to each AND gate
  - OR(all AND gates)
  - an enable input (an option)

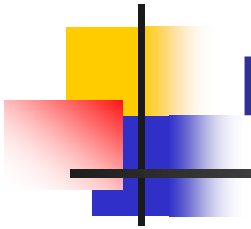


E	S	Output Y
1	X	all 0's
0	0	select A
0	1	select B

Function table

FIGURE 4.26
Quadruple two-to-one-line multiplexer

Boolean function implementation

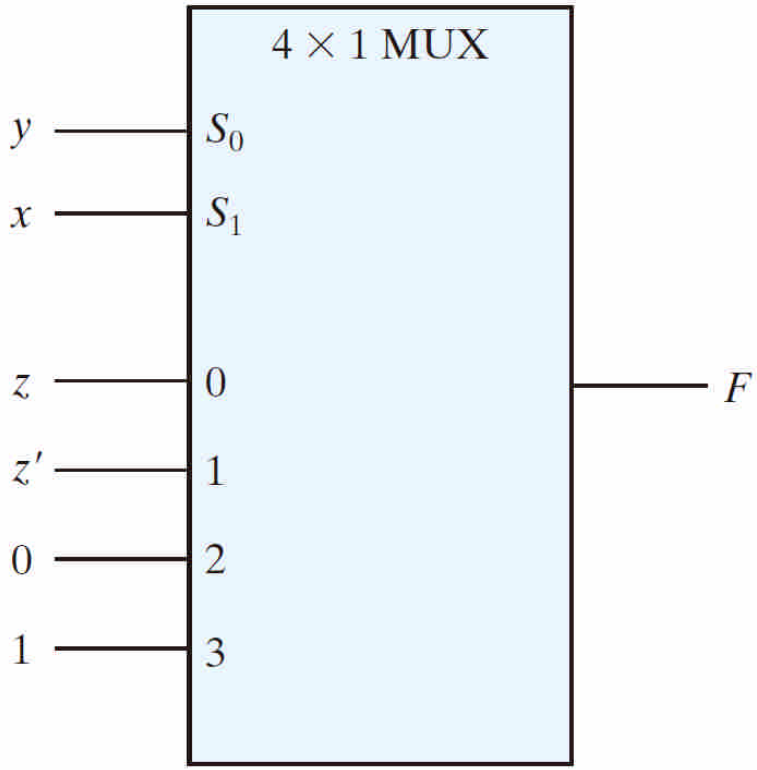


- MUX: a decoder + an OR gate
- 2^n-to-1 MUX can implement any Boolean function of n input variable
- a better solution: implement any Boolean function of n+1 input variable
  - n of these variables: the selection lines
  - the remaining variable: the inputs

- an example: F(A,B,C) = Σ(1,2,6,7)

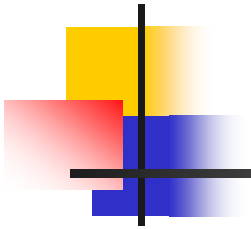
x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table



(b) Multiplexer implementation

FIGURE 4.27
Implementing a Boolean function with a multiplexer



- Procedure:
  - assign an ordering sequence of the input variable
  - the rightmost variable (D) 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

Example: F(A, B, C, D) = Σ(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	

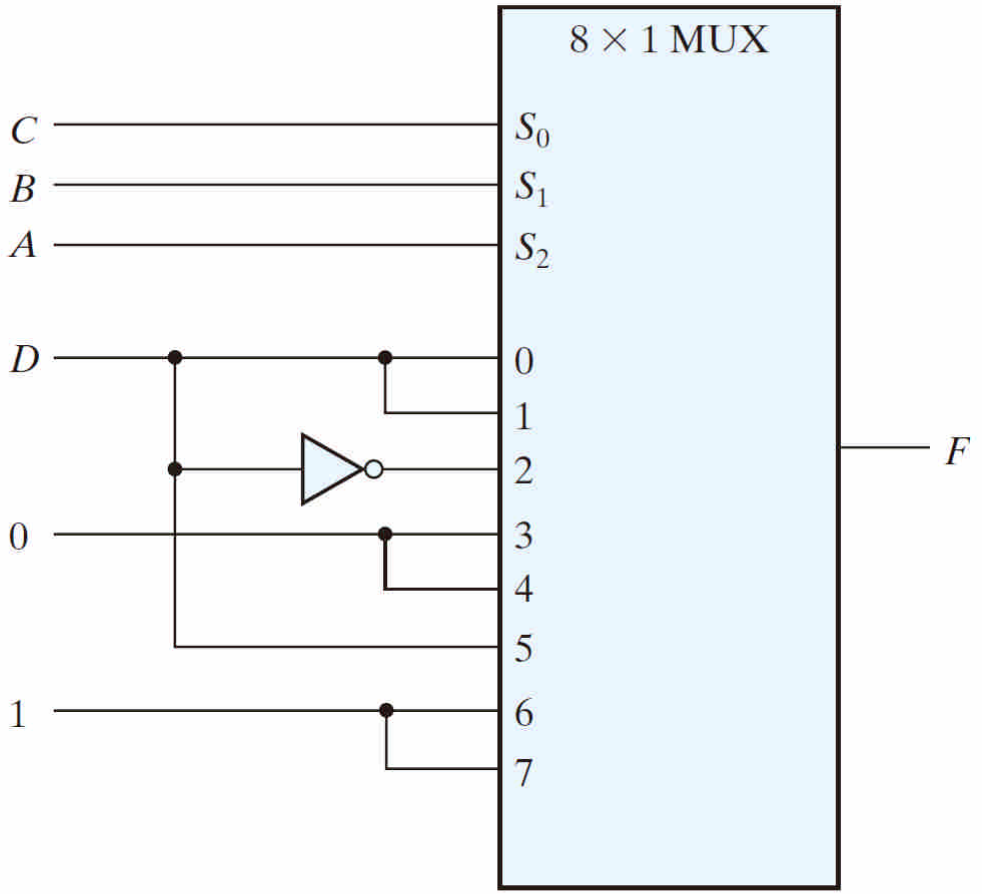


FIGURE 4.28
Implementing a four-input function with a multiplexer

Three-state gates

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

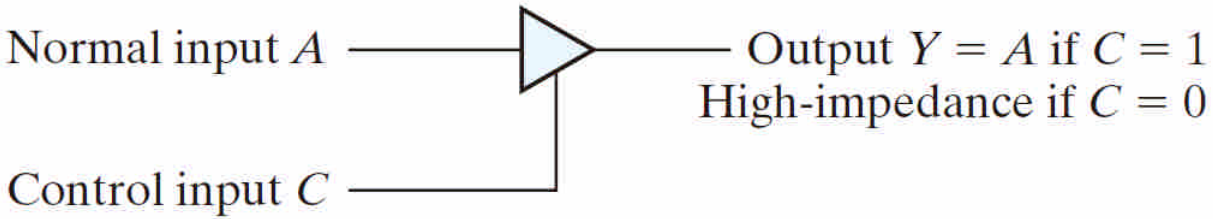


FIGURE 4.29
Graphic symbol for a three-state buffer

Example: Four-to-one-line multiplexer

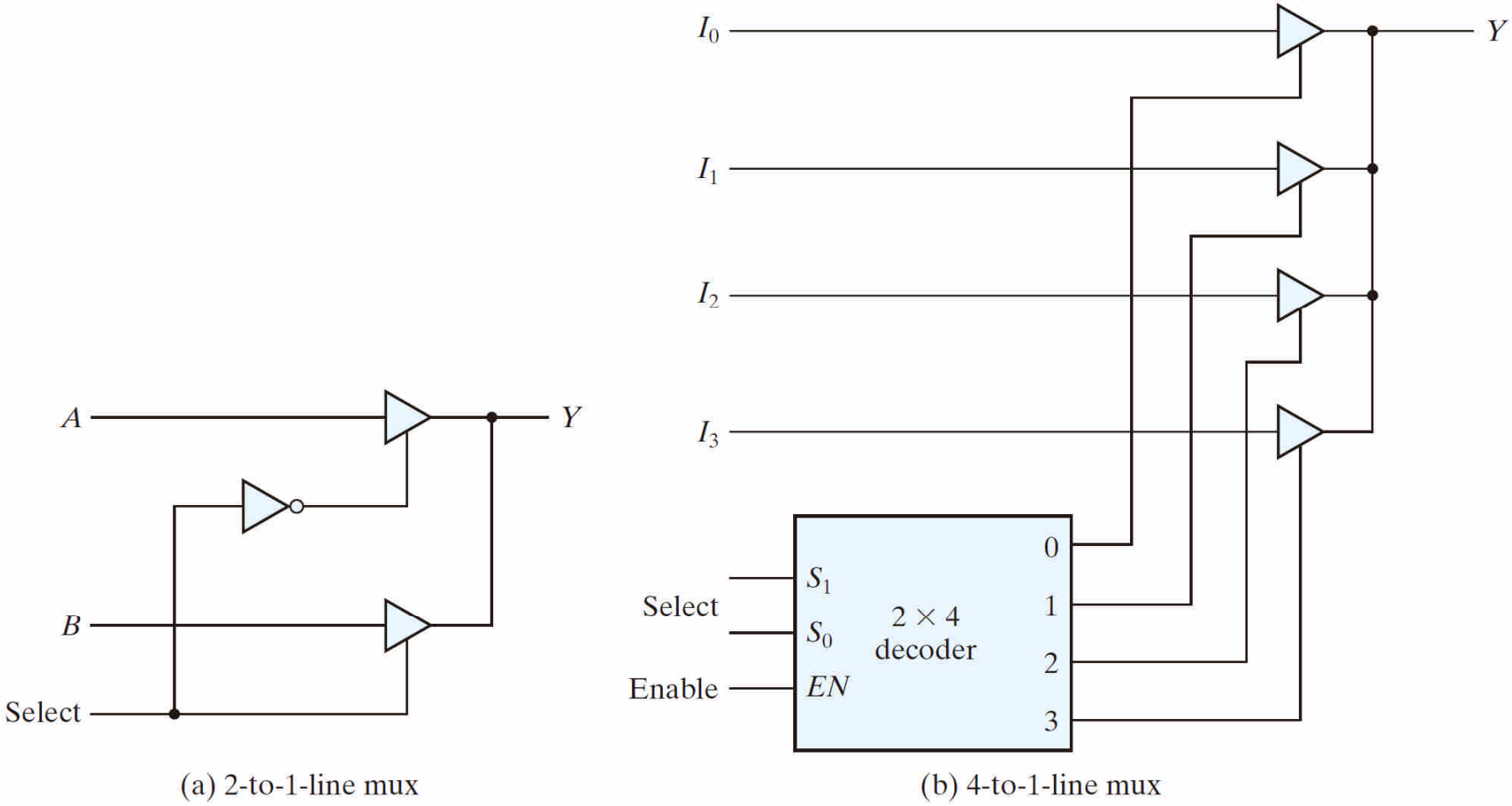


FIGURE 4.30
Multiplexers with three-state gates