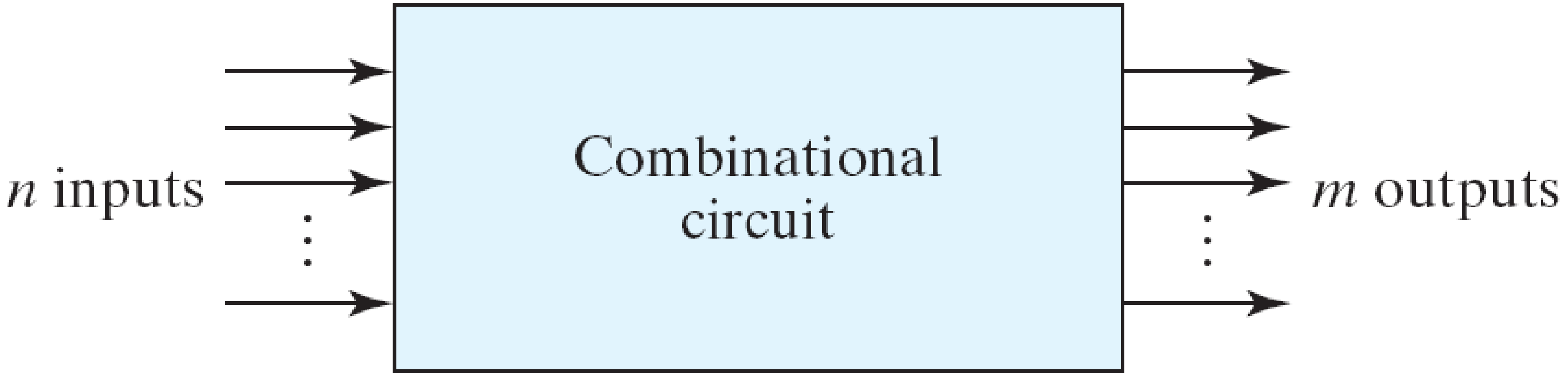
Chapter 4
Combinational Logic

4.1 Introduction

- Logic circuits 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*.
  - It performs an operation that can be specified logically by a set of Boolean functions.
- In contrast, *sequential circuits* employ *storage elements* in addition to *logic gates*.
  - Their outputs are a function of the inputs and the state of storage elements.
  - Because the state of the storage elements is a function of previous inputs, the outputs of a sequential circuit *depend not only on present value of inputs*, but *also on past inputs*, and the circuit behavior must be specified by *a time sequence of inputs and internal states*.

4.2 Combinational Circuits

- A combinational circuit consists of input variables, logic gates, and output variables.
  - *n* inputs and *m* outputs
  - Can be specified by truth table
  - Can be described by *m* Boolean functions



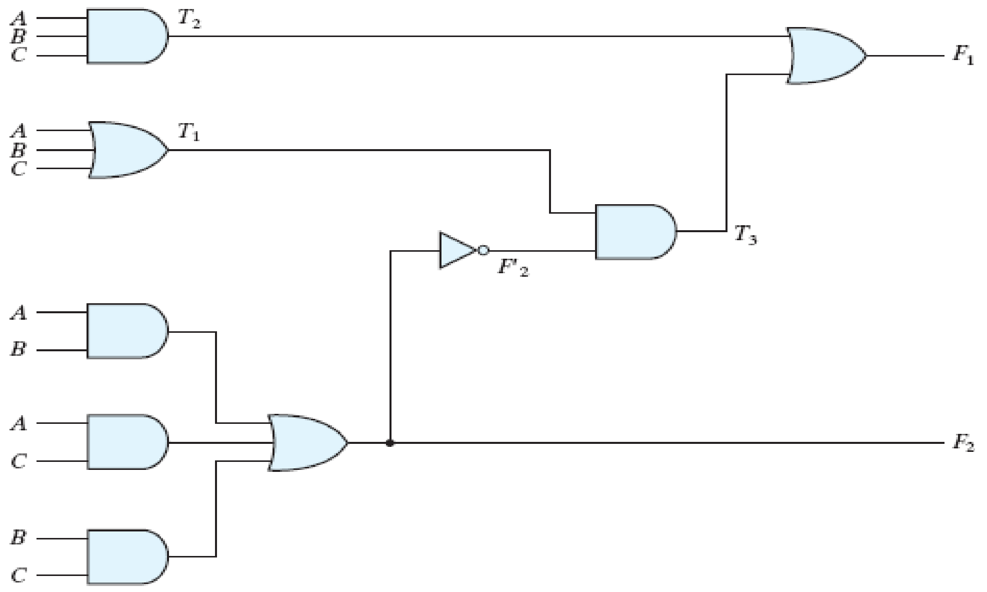
4.3 Analysis Procedure

- The first step in the analysis is to make sure that the given circuit is *combinational* and *not sequential*.
  - The diagram of a combinational circuit has logic gates with no *feedback paths* or *memory elements*.
  - *A feedback path is a connection from the output of one gate to the input of a second gate that forms part of the input to first gate*
- Obtain the output Boolean functions or the truth table

4.3 Analysis Procedure

- $F_1 = T_2 + T_3$
  - But, $T_3 = F_2' T_1$ and $T_2 = ABC$
- So, $F_1 = ABC + F_2' T_1$
  - $T_1 = A + B + C$ and $F_2 = AB + AC + BC$
- $F_1 = ABC + (A + B + C)(AB + AC + BC)'$

  $= A'BC' + A'B'C + AB'C' + ABC$

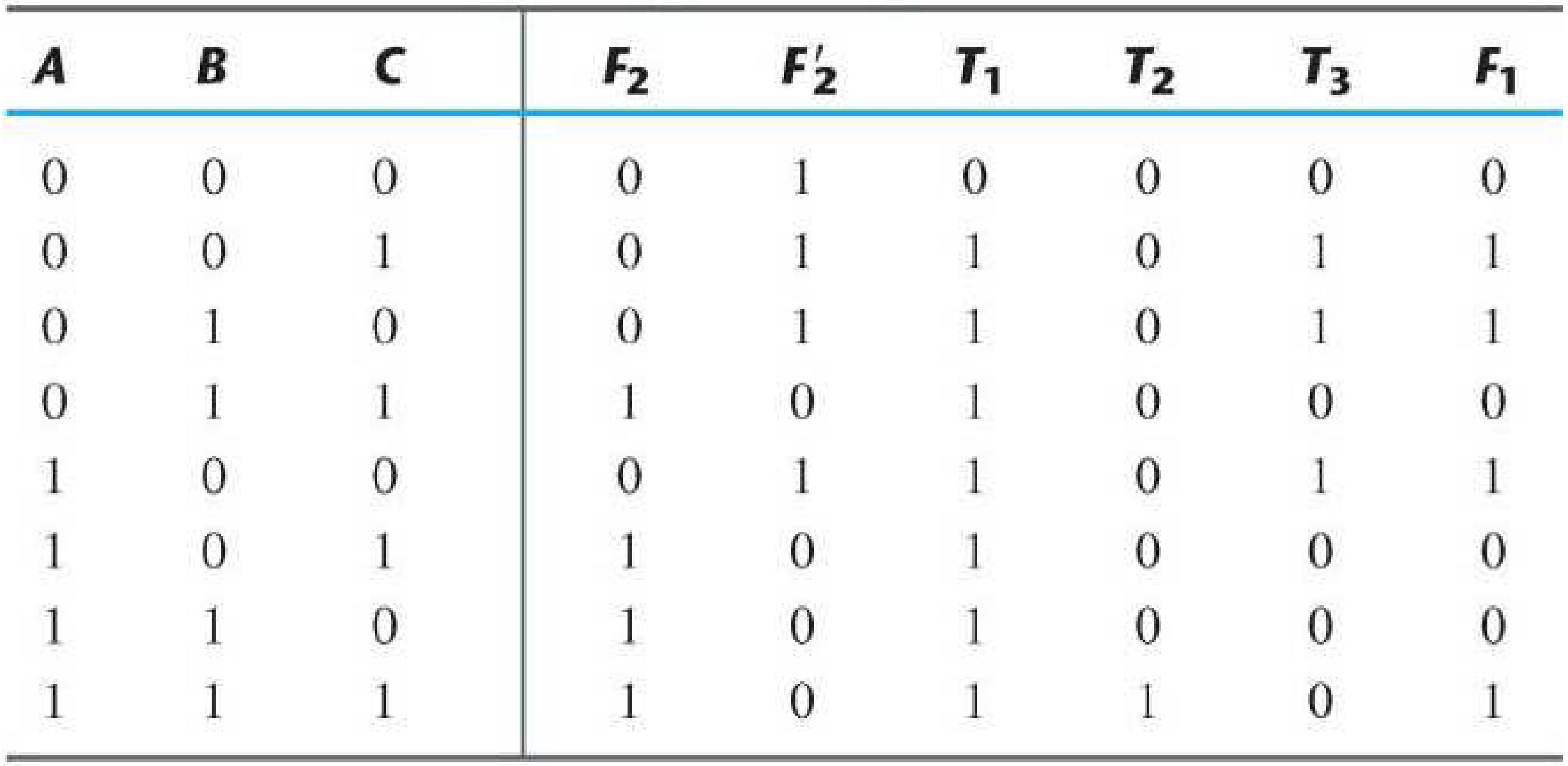


4.3 Analysis Procedure Truth Table

1. Determine the number of input variable in the circuit.
   – For n inputs form the 2^n possible input combinations.
2. Label the outputs of selected gates with arbitrary symbols.
3. Obtain the truth table for the outputs of those gates which are a function of the input variables only.
4. Proceed to obtain the truth table for the outputs of those gates which are a function of previously defined values until the columns for all outputs are determined.

4.3 Analysis Procedure
Truth Table

- Truth table of the previous example



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

4.3 Analysis Procedure

4.1 Consider the combinational circuit shown in Fig. P4.1.

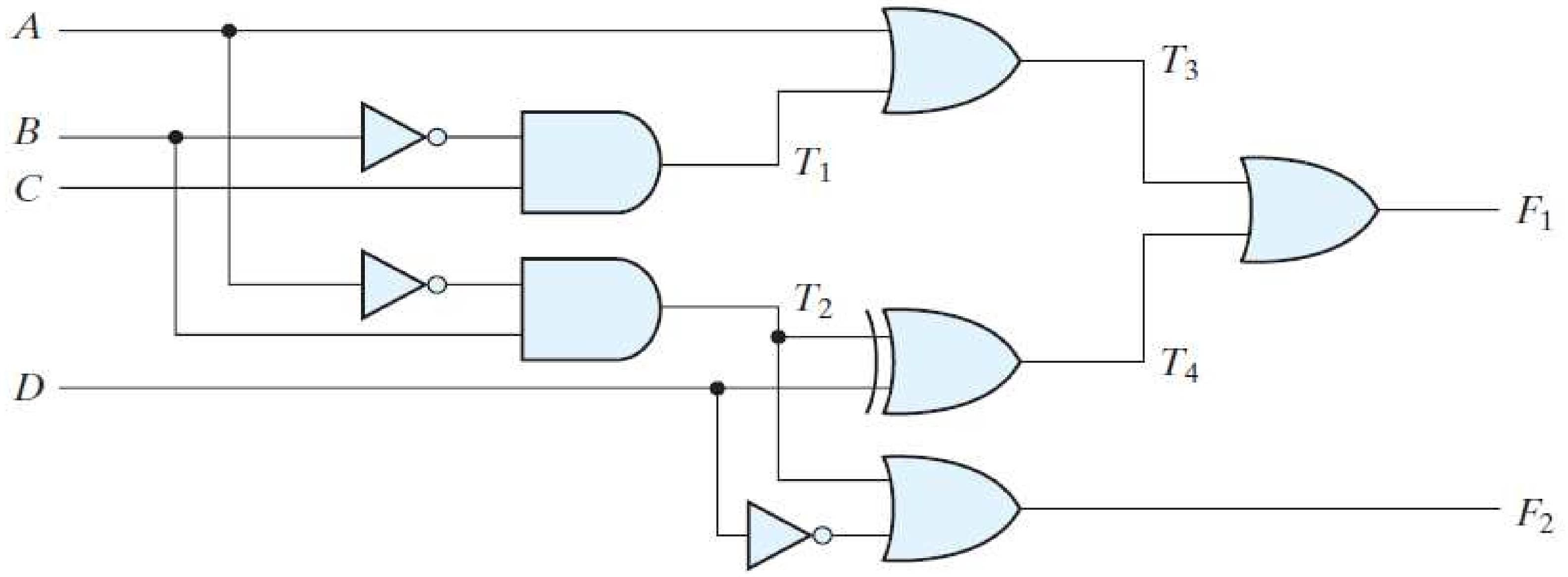


FIGURE P4.1

(a)* Derive the Boolean expressions for T_1 through T_4. Evaluate the outputs F_1 and F_2 as a function of the four inputs.

(b) List the truth table with 16 binary combinations of the four input variables. Then list the binary values for T_1 through T_4 and outputs F_1 and F_2 in the table.

(c) Plot the output Boolean functions obtained in part (b) on maps and show that the simplified Boolean expressions are equivalent to the ones obtained in part (a).

4.3 Analysis Procedure

4.2* Obtain the simplified Boolean expressions for output *F* and *G* in terms of the input variables in the circuit of Fig. P4.2.

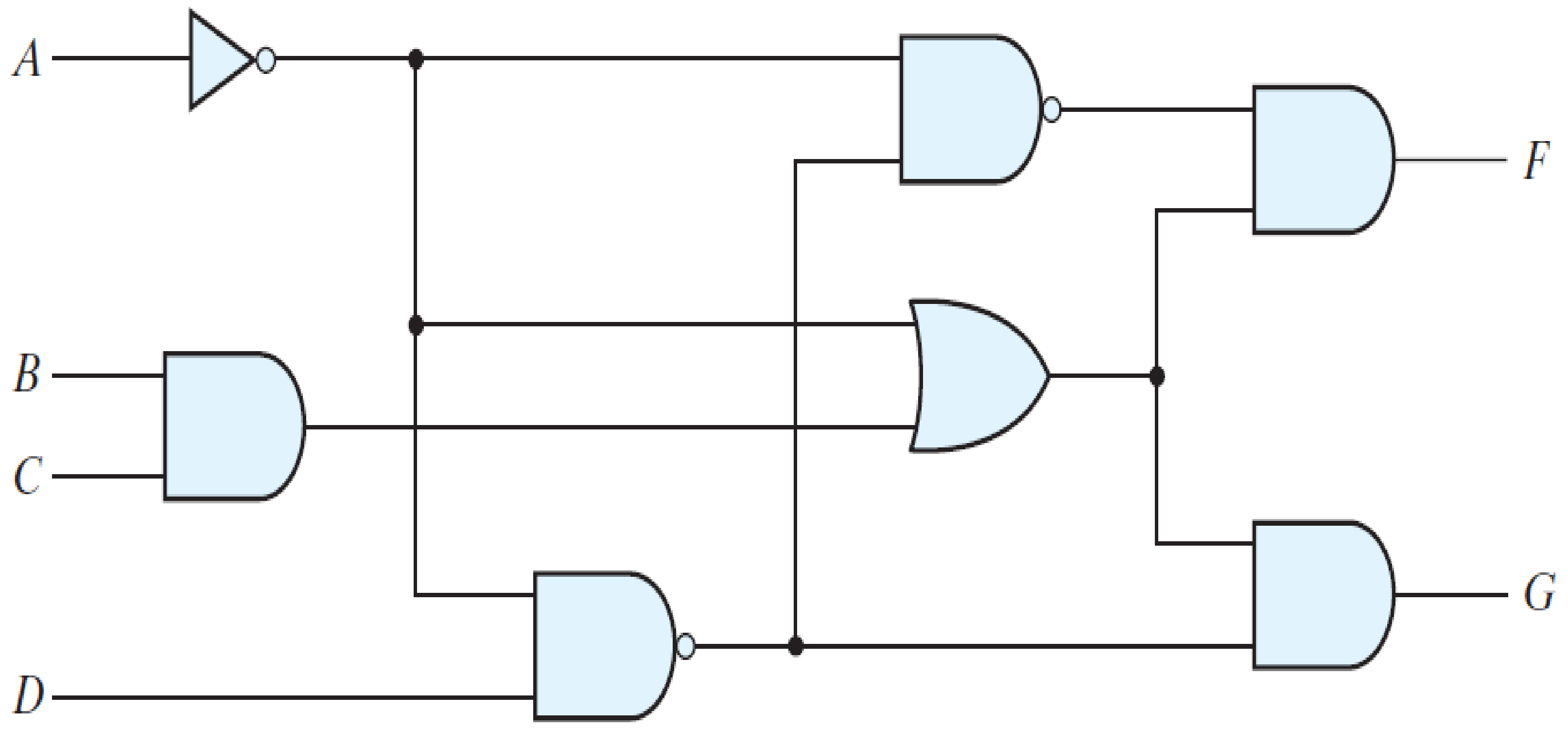


FIGURE P4.2

4.4 Design Procedure

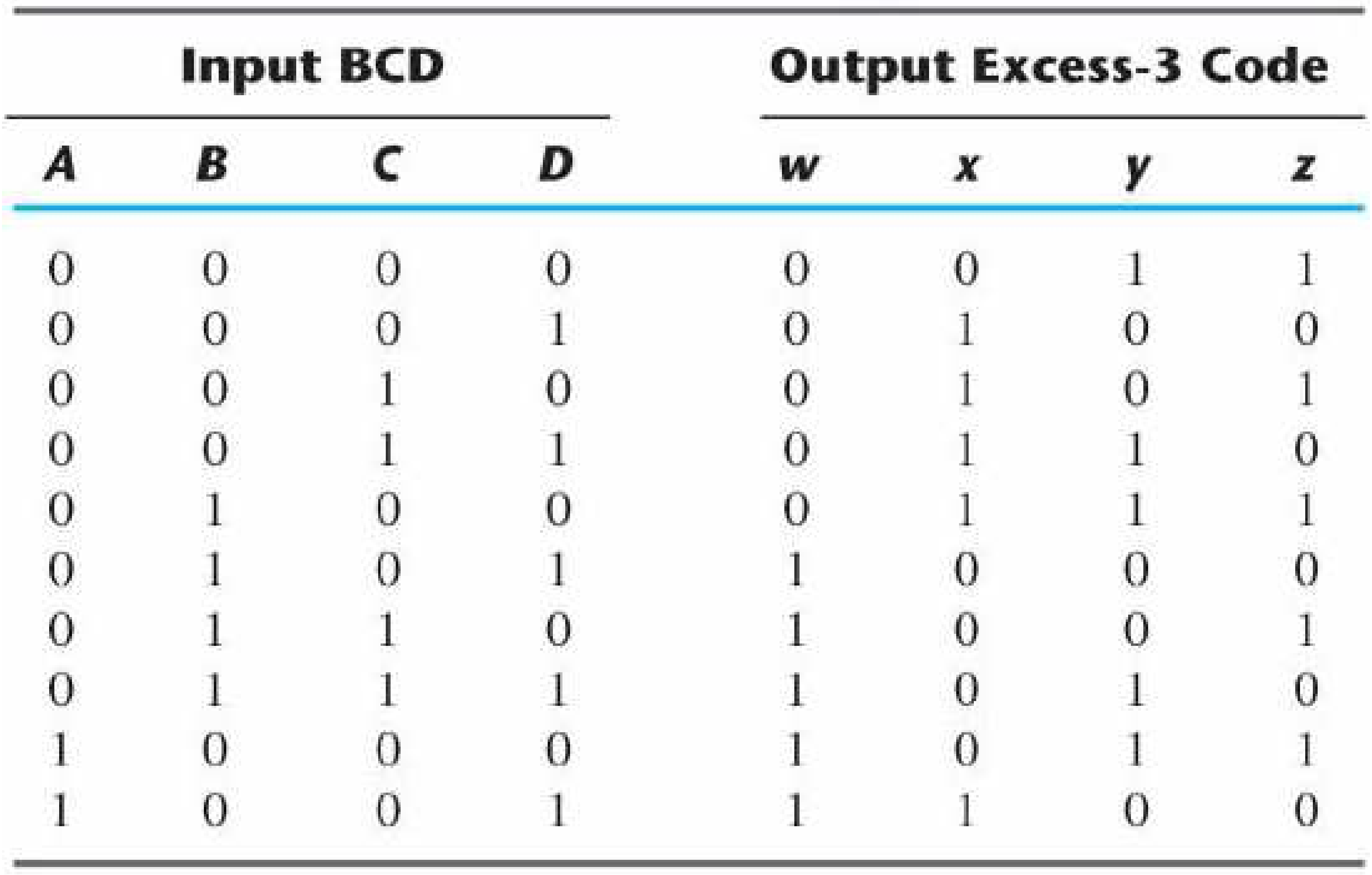
- The design of combinational circuits starts from the specification of the design objective and culminates in a logic circuit diagram or a set of Boolean function from which the logic diagram can be obtained.
- The procedure involves the following steps:

1. Determine the required number of inputs and outputs
2. Derive the truth table
3. Obtain the simplified Boolean functions
4. Draw the logic diagram

4.4 Design Procedure Code Conversion Example

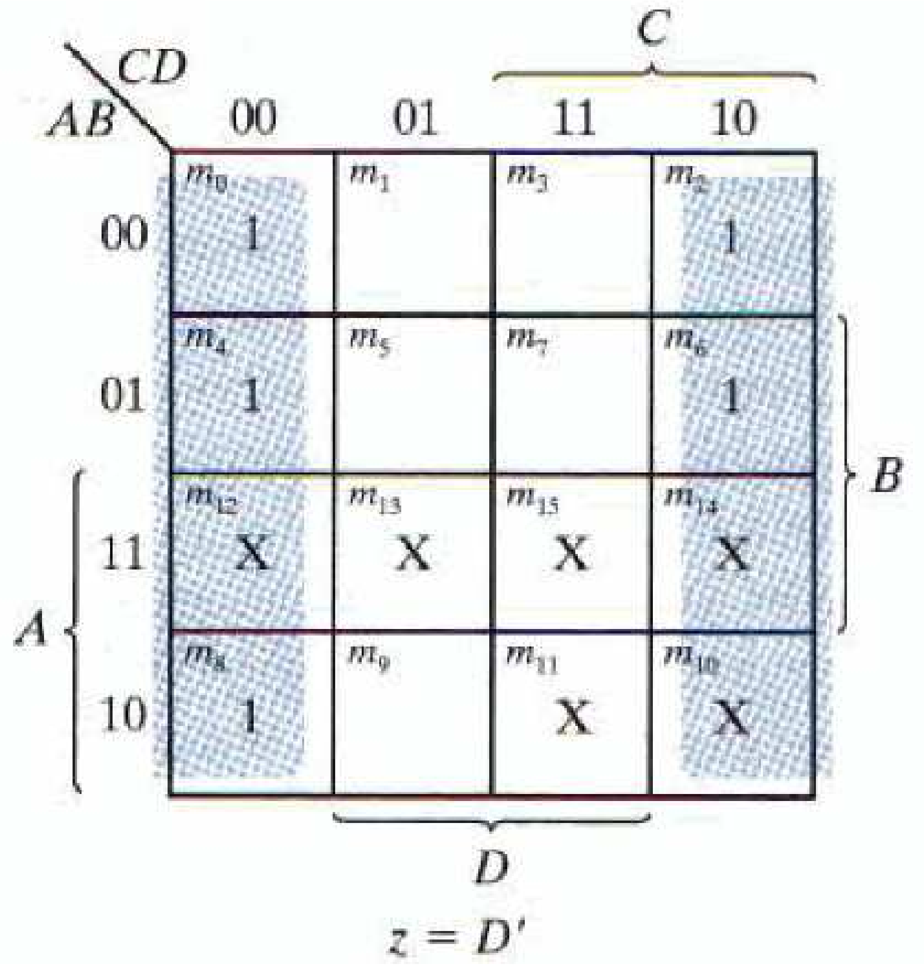
- Convert binary coded decimal (BCD) to the excess-3 code for the decimal digits?
- *Solution:*
- Four bits to represent a decimal digit:
  - Four input binary variables by the symbols *A*, *B*, *C*, and *D*
  - Four output variables by *W*, *X*, *Y*, and ***z***
- Remember, four binary variables may have 16 bit combinations, but only 10 are listed in the truth table.
  - The six bit combinations not listed for the input variables are *don't-care combinations.*

4.4 Design Procedure Code Conversion Example

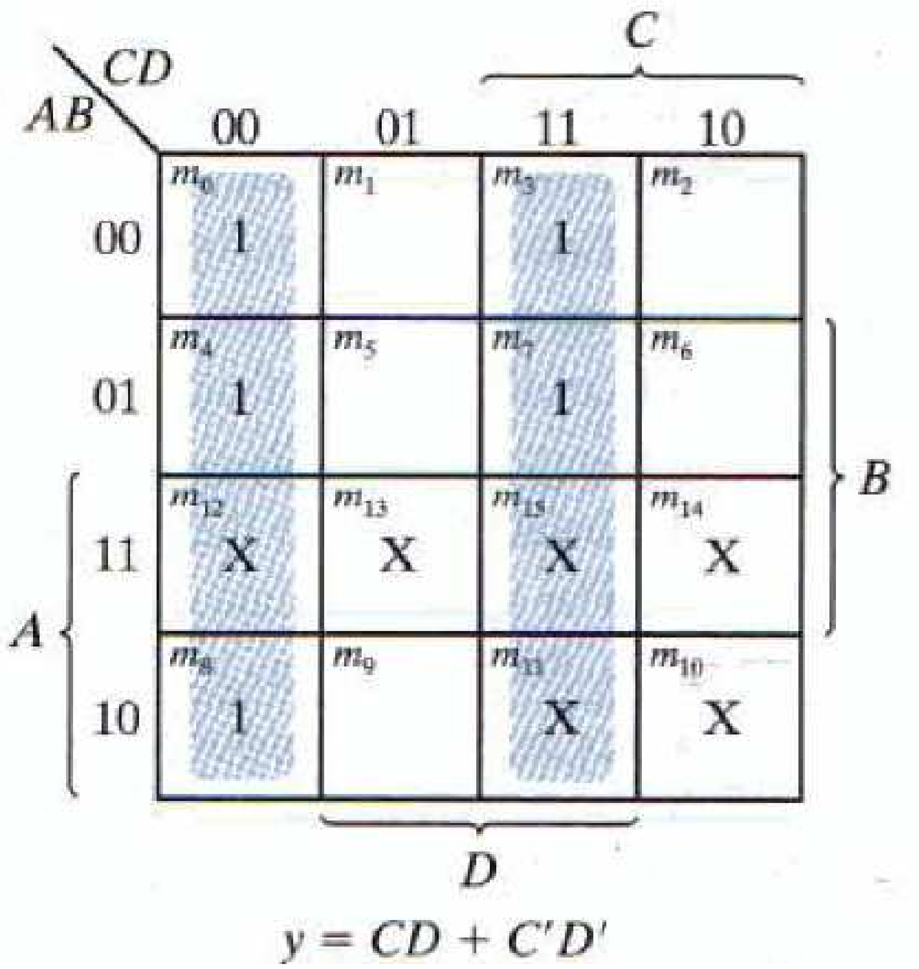


Input BCD				Output Excess-3 Code			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

4.4 Design Procedure
Code Conversion Example

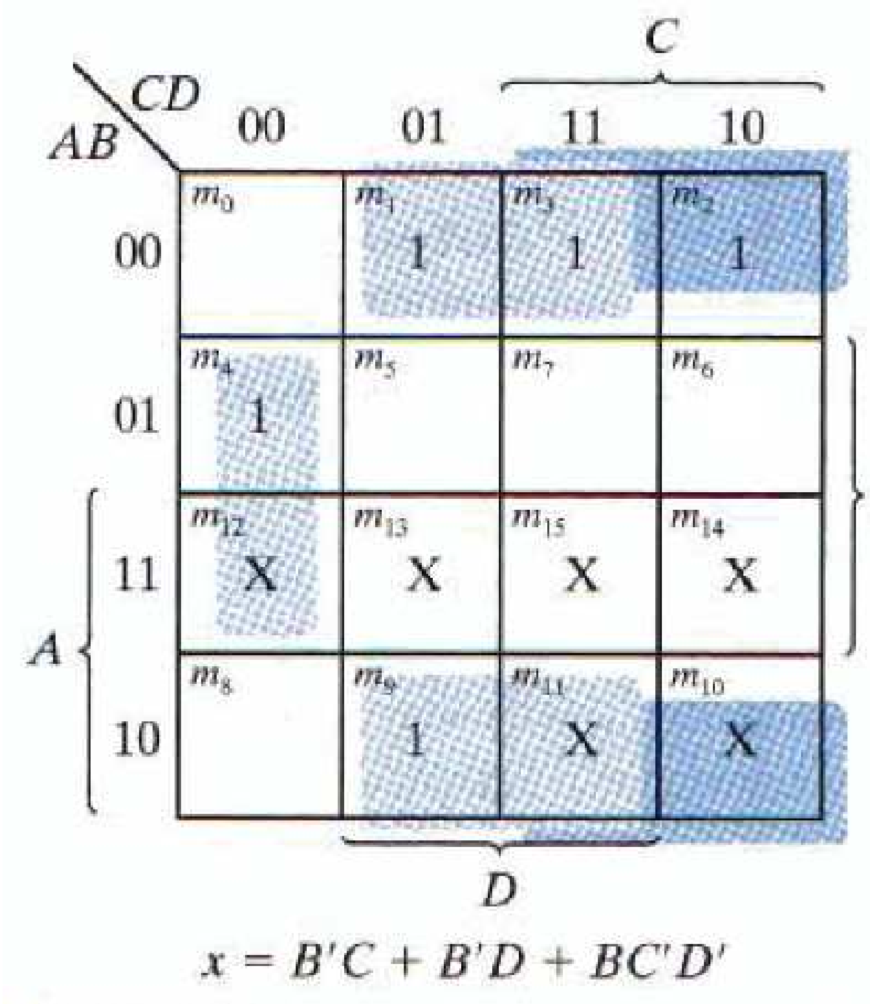


$z = D'$

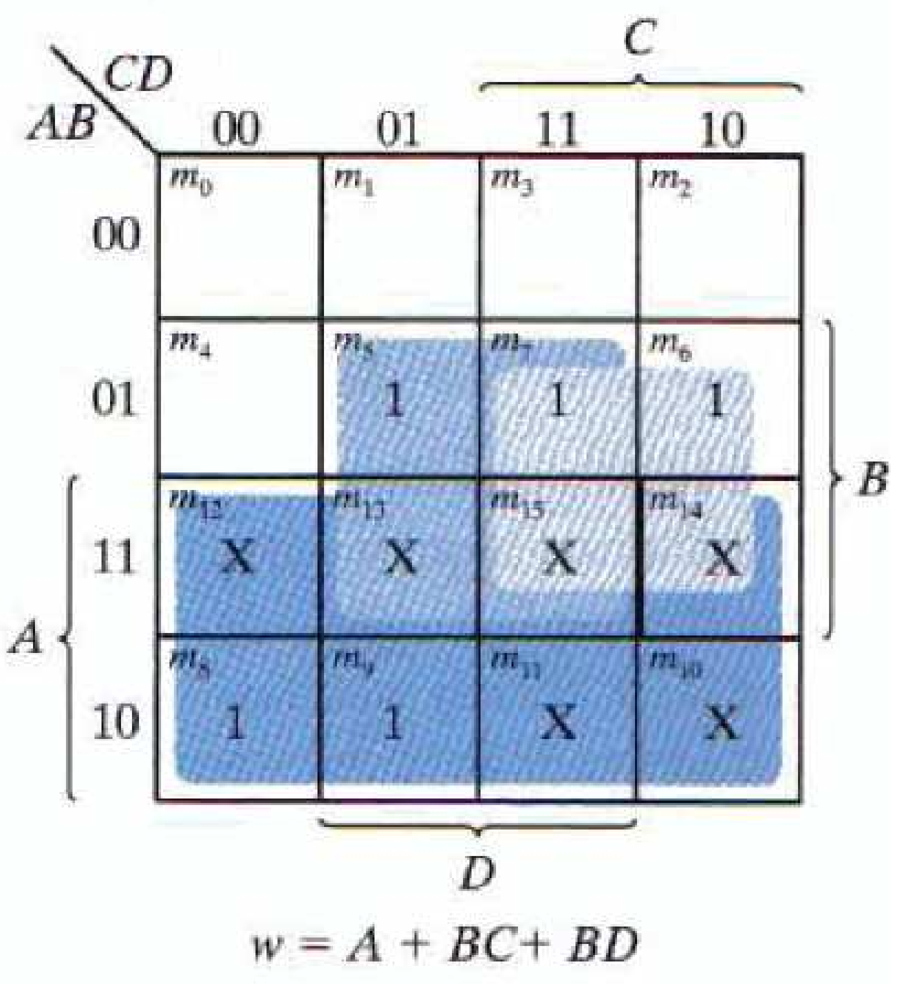


$y = CD + C'D'$

4.4 Design Procedure Code Conversion Example

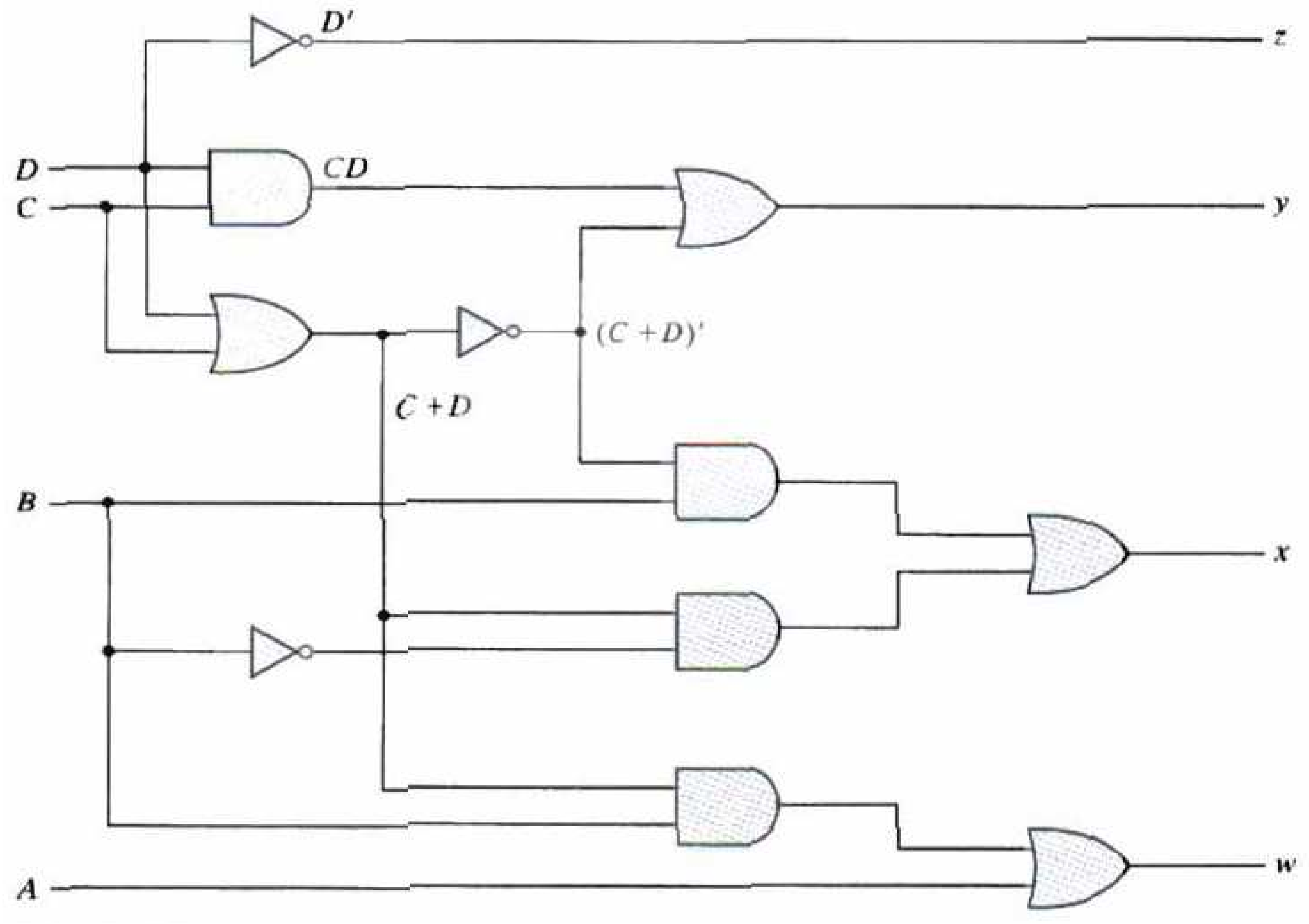


$x = B'C + B'D + BC'D'$



$w = A + BC + BD$

4.4 Design Procedure Code Conversion Example



4.4 Design Procedure

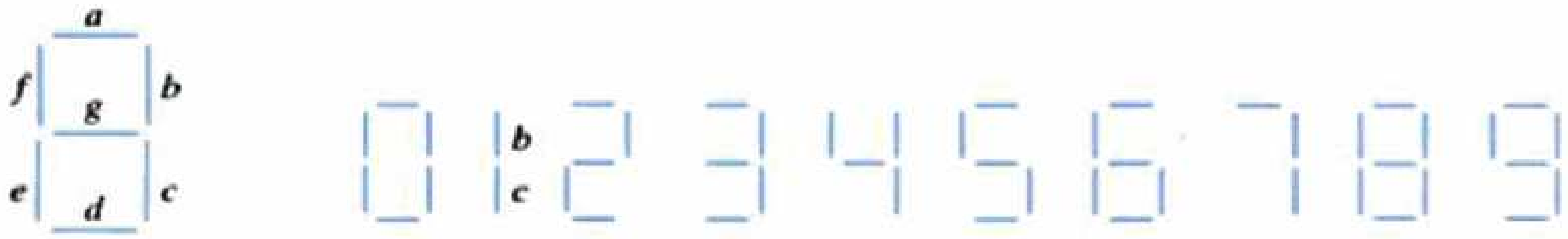
4.4 Design a combinational circuit with three inputs and one output.

(a)* The output is 1 when the binary value of the inputs is less than 3. The output is 0 otherwise.

(b) The output is 1 when the binary value of the inputs is an odd number.

4.5 Design a combinational circuit with three inputs, x, y, and z, and three outputs, A, B, and C. When the binary input is 0, 1, 2, or 3, the binary output is two greater than the input. When the binary input is 4, 5, 6, or 7, the binary output is three less than the input.

4.9 An ABCD-to-seven-segment decoder is a combinational circuit that converts a decimal digit in BCD to an appropriate code for the selection of segments in an indicator used to display the decimal digit in a familiar form. The seven outputs of the decoder (a, b, c, d, e, f, g) select the corresponding segments in the display, as shown in Fig. P4.9(a). The numeric display chosen to represent the decimal digit is shown in Fig. P4.9(b). Using a truth table and Karnaugh maps, design the BCD-to-seven-segment decoder, using a minimum number of gates. The six invalid combinations should result in a blank display.



(a) Segment designation (b) Numerical designation for display

FIGURE P4.9

4.5 Binary Adder-Subtractor

- Digital computers perform a variety of information processing tasks.
  - The most basic arithmetic operation is the *addition of two binary digits*
- A combinational circuit that performs the addition of two bits is called a *half adder*
  - The one that performs the addition of three bits (*two significant bits* and a *previous carry)* is a *full adder*
  - Two half adders can be employed to implement a full adder
- A *binary adder-subtractor* is a combinational circuit that performs the arithmetic operation of *addition* and *subtraction* with *binary numbers*.

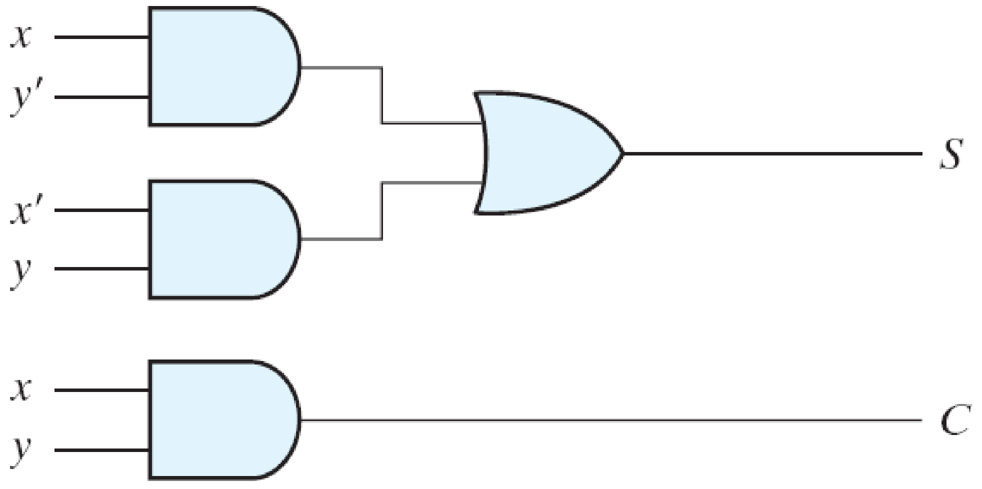
4.5 Binary Adder-Subtractor
Half Adder

- $S = x'y + xy'$

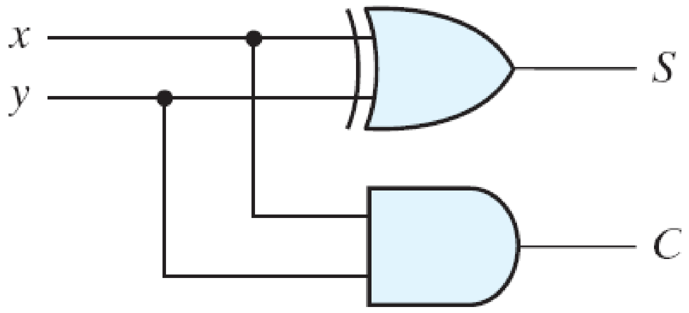
  $S = x \oplus y$
- $C = xy$

Half Adder

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



(a) $S = xy' + x'y$
$C = xy$



(b) $S = x \oplus y$
$C = xy$

4.5 Binary Adder-Subtractor
Full Adder

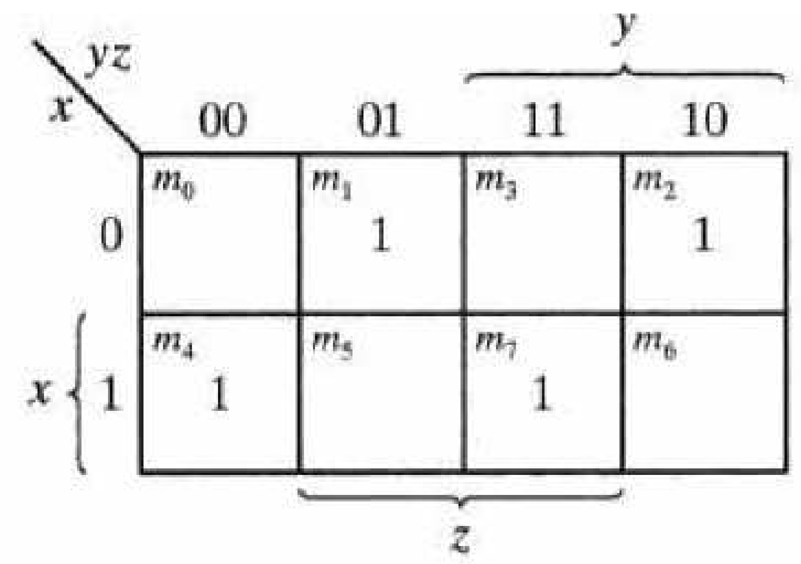
- $S = x'y'z + x'yz' + xy'z' + xyz$

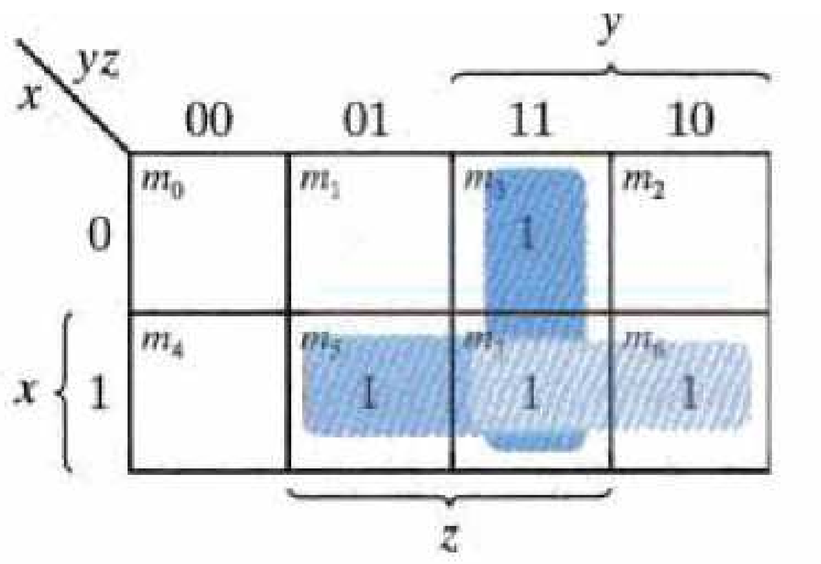
  $S = x \oplus y \oplus z$
- $C = xy + xz + yz$

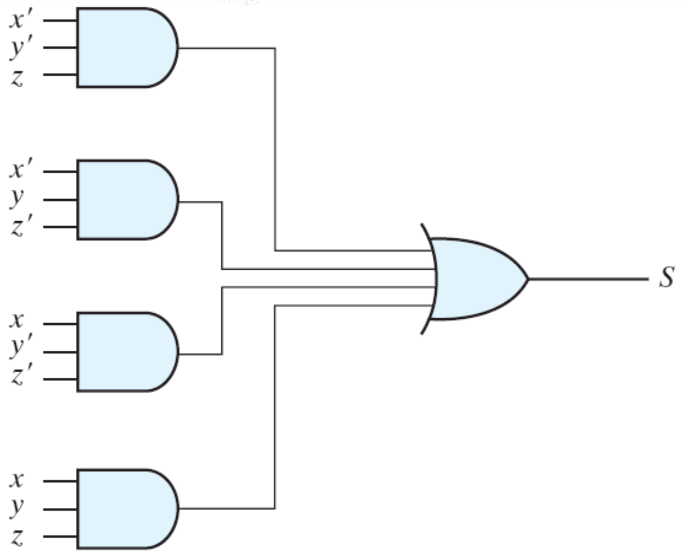
Full Adder

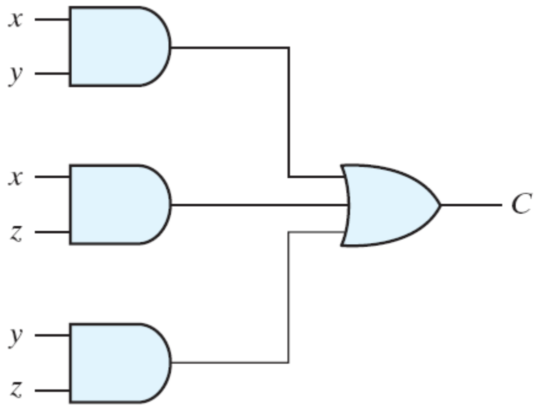
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

4.5 Binary Adder-Subtractor
Full Adder







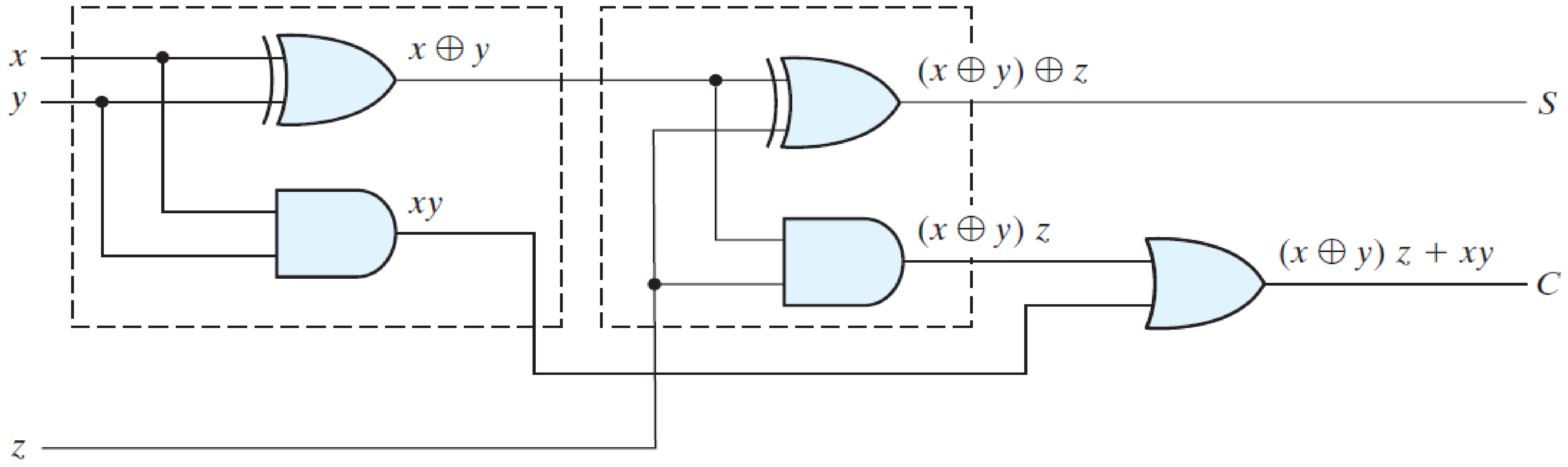


4.5 Binary Adder-Subtractor
Full Adder

- Implement with *2 half adder* and *1 OR*
- From the original function:

$$C = xy'z + x'yz + xyz + xyz'$$

$$= z(xy' + x'y) + xy(z + z') = z(x \oplus y) + xy$$

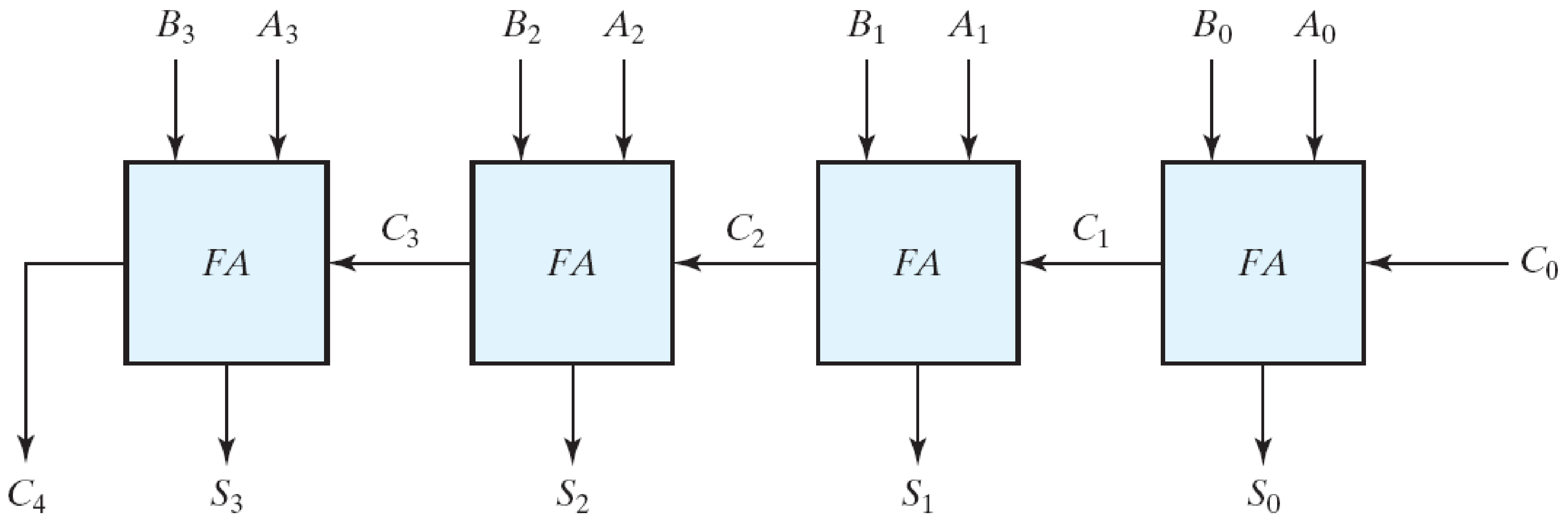


4.5 Binary Adder-Subtractor
Binary Adder

- A digital circuit that produces the arithmetic sum of two binary numbers.
- It can be constructed with *full adders* connected in cascade,
  - The output carry from each full adder connected to the input carry of the next full adder in the chain.

4.5 Binary Adder-Subtractor
Binary Adder

- A four full-adder circuits to provide a four-bit binary ripple carry adder.
  - The augend's bits of A and the addend bits B are designated by subscript numbers from right to left, with subscript 0 denoting the least significant bit.



4.5 Binary Adder-Subtractor
Binary Adder

- Consider the two binary numbers *A*= 1011 and *B*= 0011.
- Their sum *S*= 1110 is formed with the four-bit adder as follows:

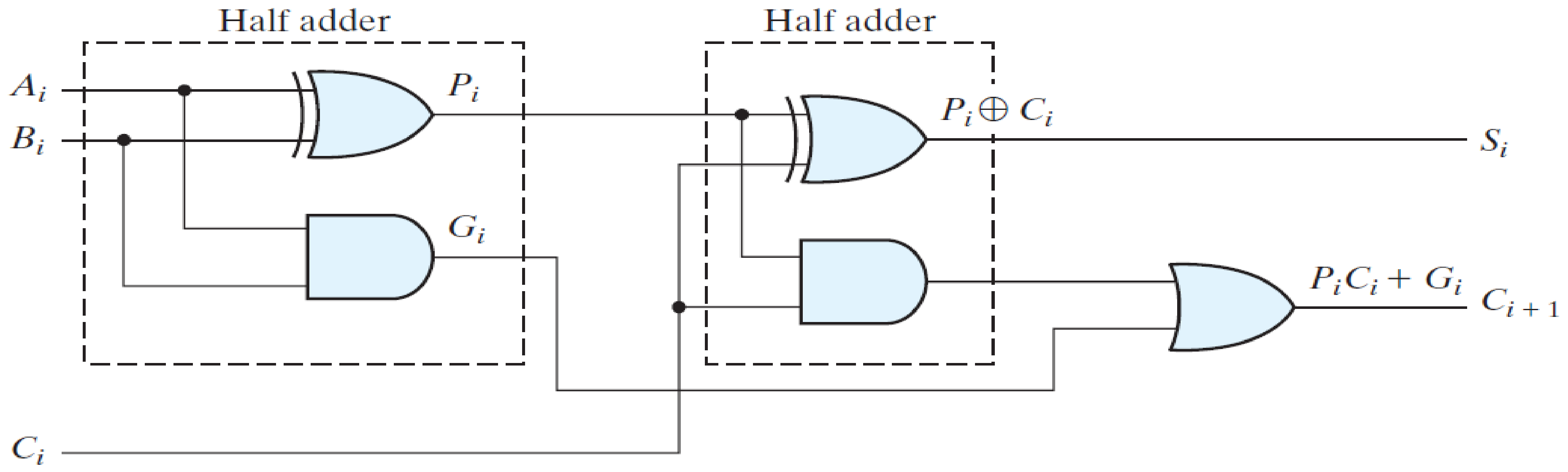
Subscript *i*:	3 2 1 0	
Input carry	0 1 1 0	C_i
Augend	1 0 1 1	A_i
Addend	0 0 1 1	B_i
Sum	1 1 1 0	S_i
Output carry	0 0 1 1	C_{i+1}

4.5 Binary Adder-Subtractor
Carry Propagation

- The addition of two binary numbers in parallel implies that all the bits of the augend and addend are available for *computation at the same time*.
- *As in any combinational circuit*, the signal must propagate through the gates before the correct *output sum* is available in the output terminals.
- The total *propagation time* is equal to the propagation delay of a typical gate, times the number of gate levels in the circuit.
- The longest propagation delay time in an adder is the time it takes the carry to propagate through the full adder.
- Since each bit of the sum output depends on the value of the input carry, the value of S_i at any given stage in the adder will be in its *steady-state final value only after the input carry to that stage has been propagated*.

4.5 Binary Adder-Subtractor Carry Propagation

- The signal from the input carry C_i to the output carry C_{i+1} propagates through an AND gate and an OR gate, which constitute two gate levels.
  - If there are four full adder, the output carry C_4 would have $2 * 4 = 8$ gate level from C_0 to C_4.
- For an n-bit adder, there are $2n$ gate levels for the carry to propagate from input to output.

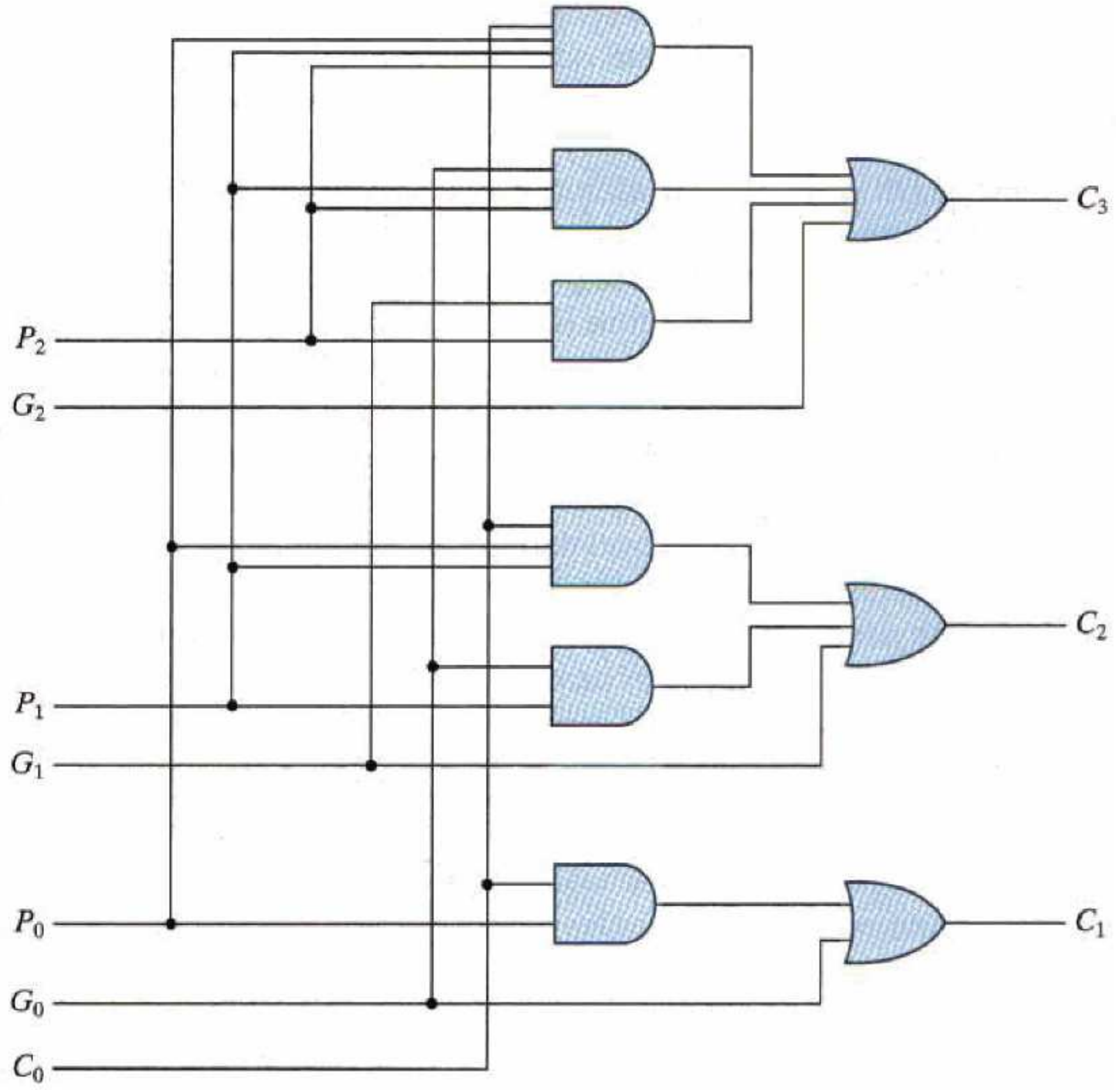


4.5 Binary Adder-Subtractor
Carry Propagation

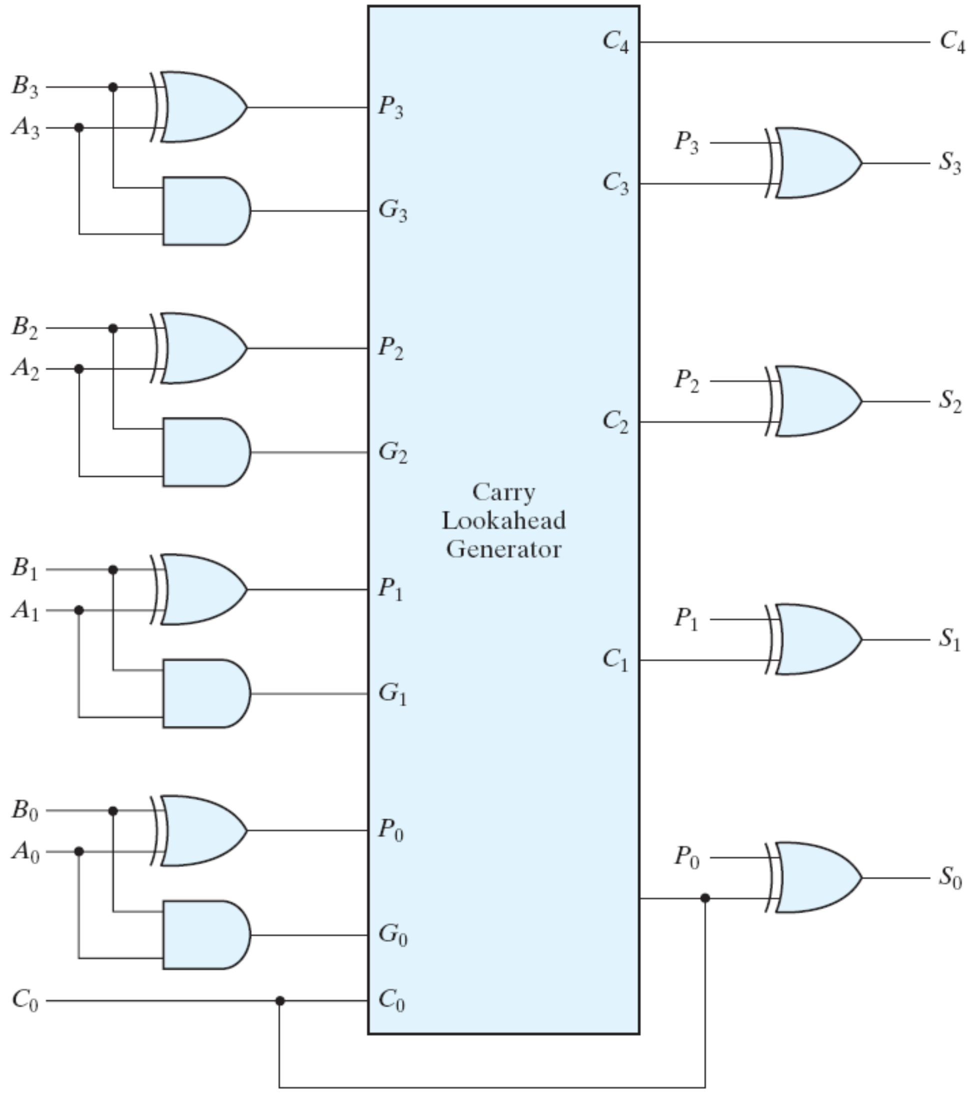
- There are several techniques for reducing the carry propagation time in parallel adder.
- The most widely used technique employs the principle of *carry look-ahead logic*.
- Let $P_i = A_i \oplus B_i$, and $G_i = A_iB_i$

  Then $S_i = P_i \oplus C_i$, and $C_{i+1} = G_i + P_iC_i$
  - G_i is called a *carry generate*
  - P_i is called a *carry propagate*
- C_0 = input carry
- $C_1 = G_0 + P_0 C_0$
- $C_2 = G_1 + P_1 C_1 = G_1+P_1(G_0+P_0C_0) = G_1+P_1G_0+P_1P_0C_0$
- $C_3 = G_2 + P_2 C_2 = G_2+P_2G_1+P_2P_1G_0+P_2P_1P_0C_0$
- Since the Boolean function for each output carry is expressed in sum-of-product form, each function can be implemented with one level of AND gates followed by an OR gate.

4.5 Binary Adder-Subtractor
Carry Propagation – carry look-ahead



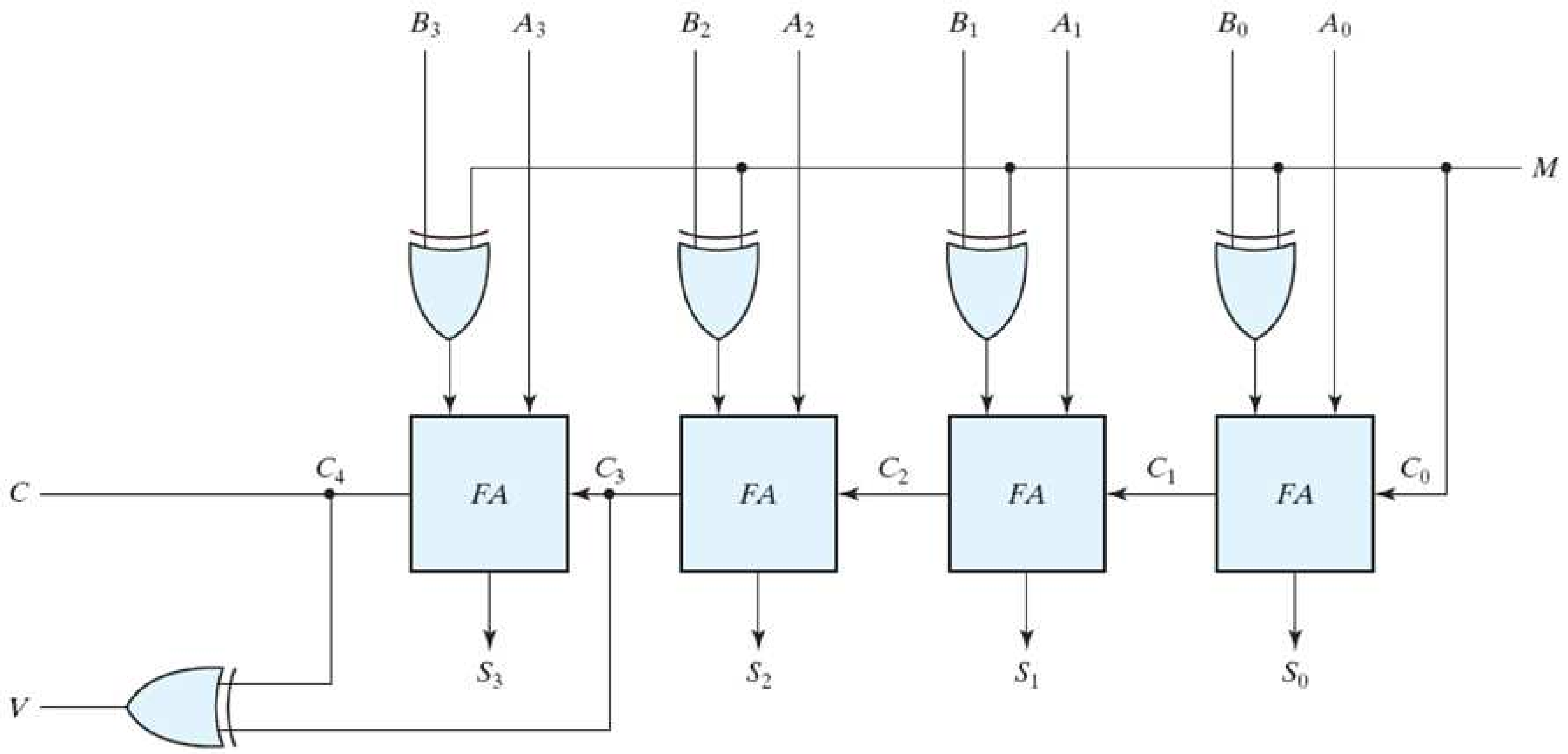
4.5 Binary Adder-Subtractor
Binary Adder with Carry Lookahead



4.5 Binary Adder-Subtractor
Binary Subtractor

- The subtraction $A - B$ can be done by taking the 2's complement of B and adding it to A.
  - Take the 1's complement
  - Add 1 to the least significant pair of bits
- 1's complement can be implemented with *XOR*s
- A 1 can be added to the sum through the *input carry*
- The *addition* and *subtraction* operations can be combined into one circuit
- The mode input M controls the operation:
  - When $M = 0$, the circuit is an adder ($B \oplus 0 = B$, $C_0 = 0$)
  - When $M = 1$, the circuit is a subtractor ($B \oplus 1 = B'$, $C_0 = 1$)

4.5 Binary Adder-Subtractor
Binary Subtractor



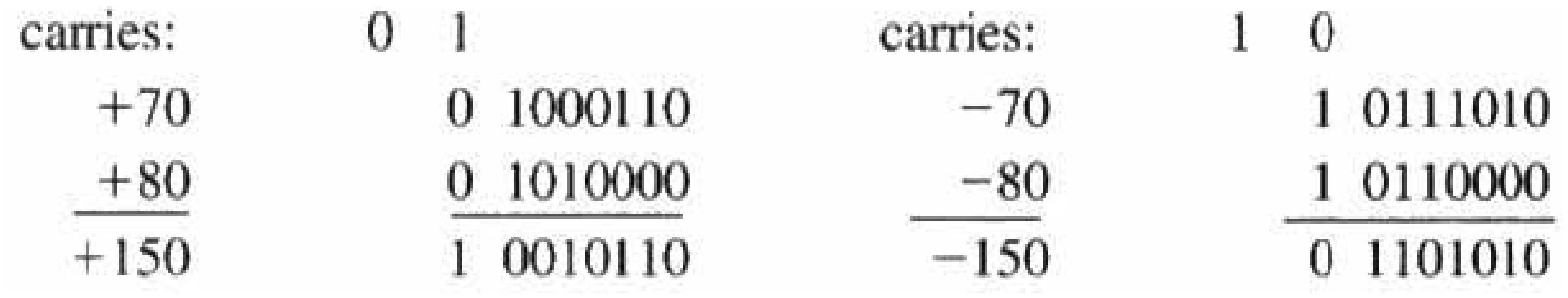
4.5 Binary Adder-Subtractor Overflow

- When two numbers with *n* digits each are added and the sum is a number occupying *n*+1 digits, we say that an *overflow* occurred.
  - Overflow is a problem in digital computers
  - *n*+1 bits cannot be accommodated by an *n*-bit word

1. For *unsigned numbers*, an overflow is detected from the end *carry out of the most significant position* (C_{n+1}).
2. For *signed numbers*, an overflow cannot occur after an addition if one number is positive and the other is negative (V).

4.5 Binary Adder-Subtractor Overflow – Singed Numbers

- An overflow may occur if the two numbers added are both positive or both negative, but the result is an opposite sign
- An overflow can be detected by observing *the carry into the sign bit position* and *the carry out of the sign bit position*.
  – See the output variable $V = C_3 \oplus C_4$ in the previous diagram for four-bit adder-subtractor.

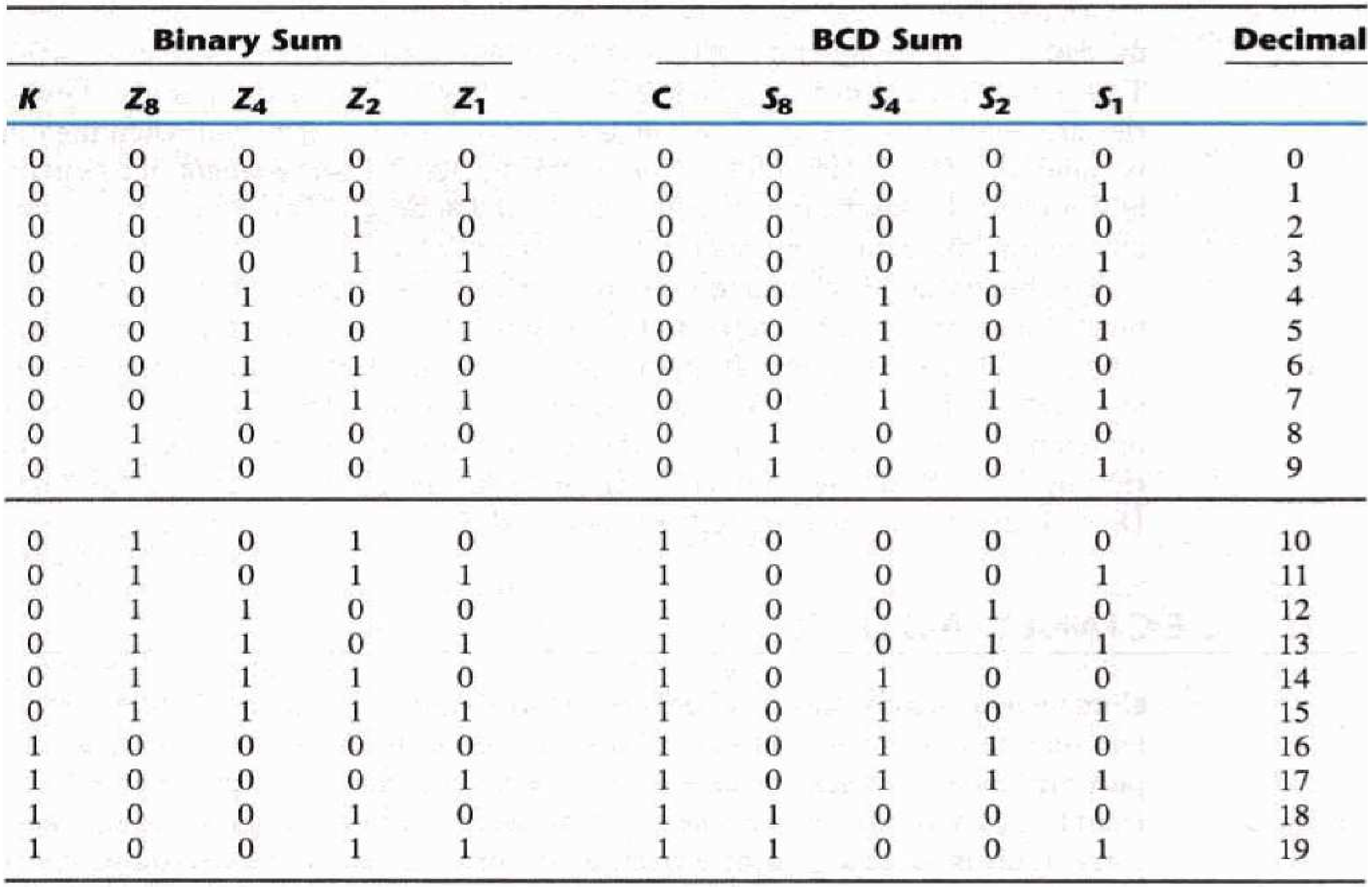


carries:	0 1	carries:	1 0
+70	0 1000110	−70	1 0111010
+80	0 1010000	−80	1 0110000
+150	1 0010110	−150	0 1101010

4.6 Decimal Adder
BCD adder

- Suppose we apply two BCD digits to a four-bit binary adder:
  - The adder will form the sum in binary
  - It will produce a result that ranges from 0 through 19 (9 + 9 + 1 carry).
- The output sum of two decimal digits must be represented in BCD
- Problem: find a rule by which the binary sum is converted to the correct BCD digit representation of the number in the BCD sum?

4.6 Decimal Adder
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

4.6 Decimal Adder
BCD adder

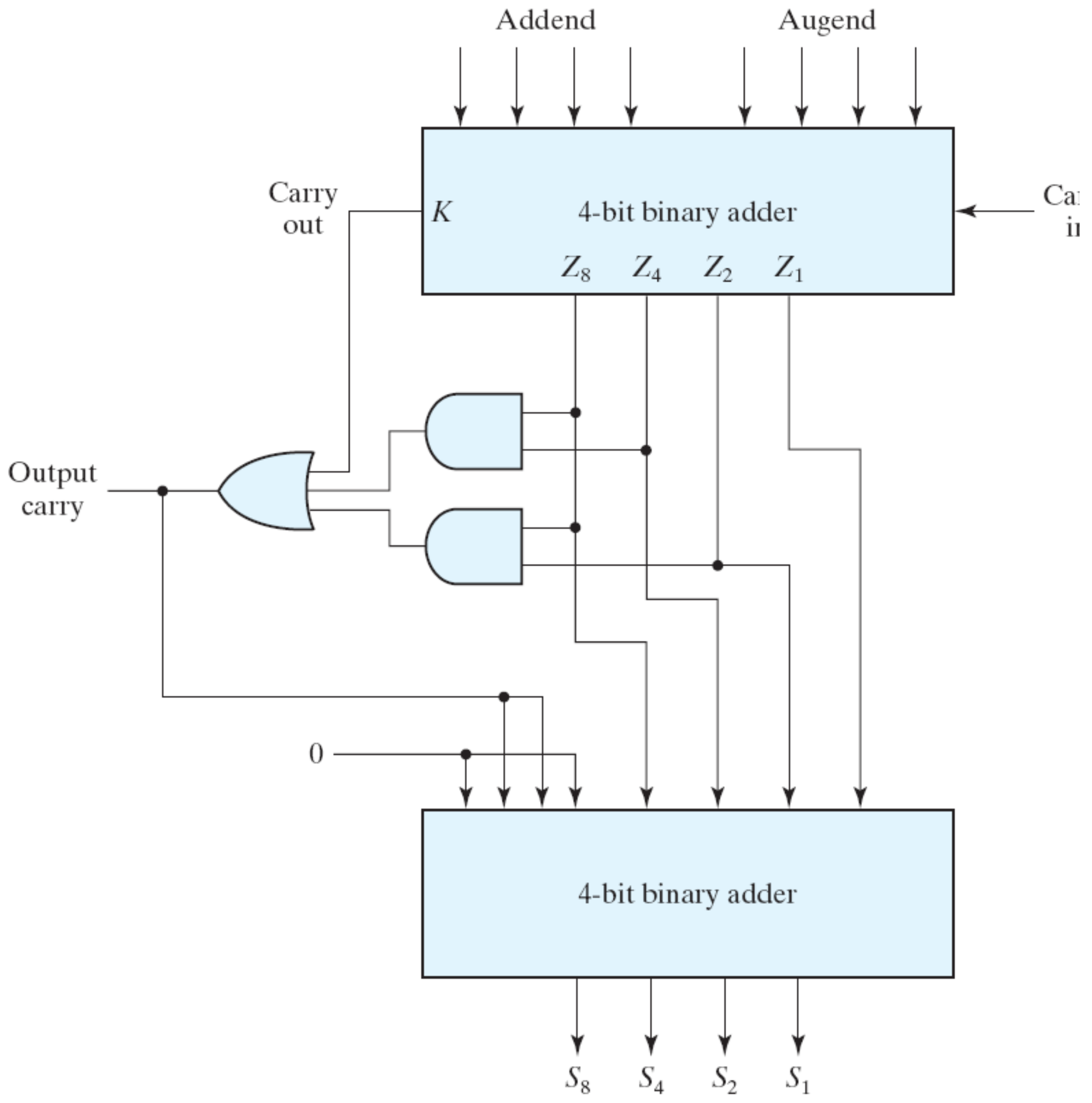
- When carry occurs, the addition of **0110** (**6**) to the binary sum converts it to the correct BCD representation.
- From truth table, output carry:

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

- When C = 1, binary 0110 is added to the binary sum through the bottom four-bit adder

4.6 Decimal Adder
BCD adder

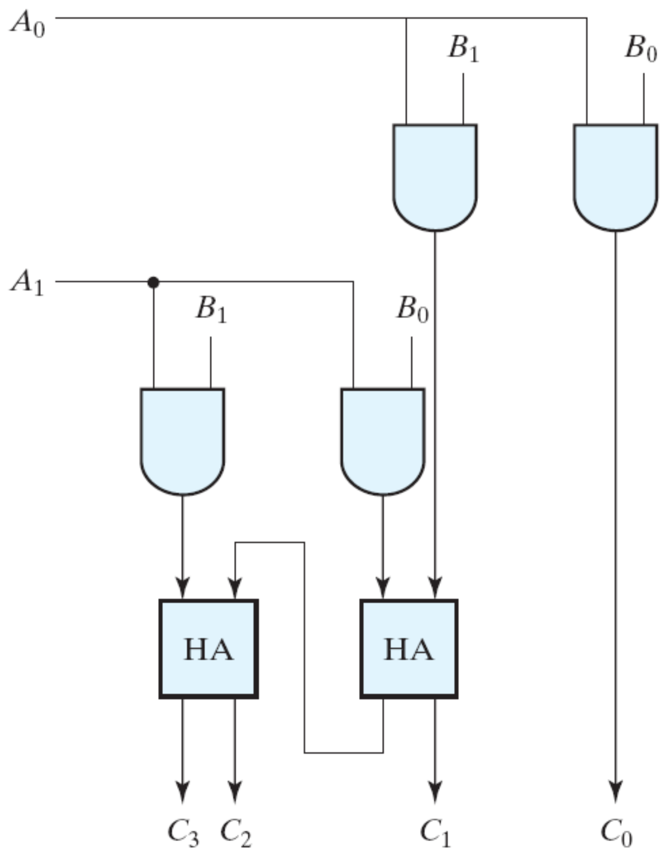
- When C = 1, binary **0110** is added to the binar sum through the bottom four-bit adder



4.7 Binary Multiplier

- Binary multiplication is performed as decimal multiplication

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



- The product is $C_3C_2C_1C_0$

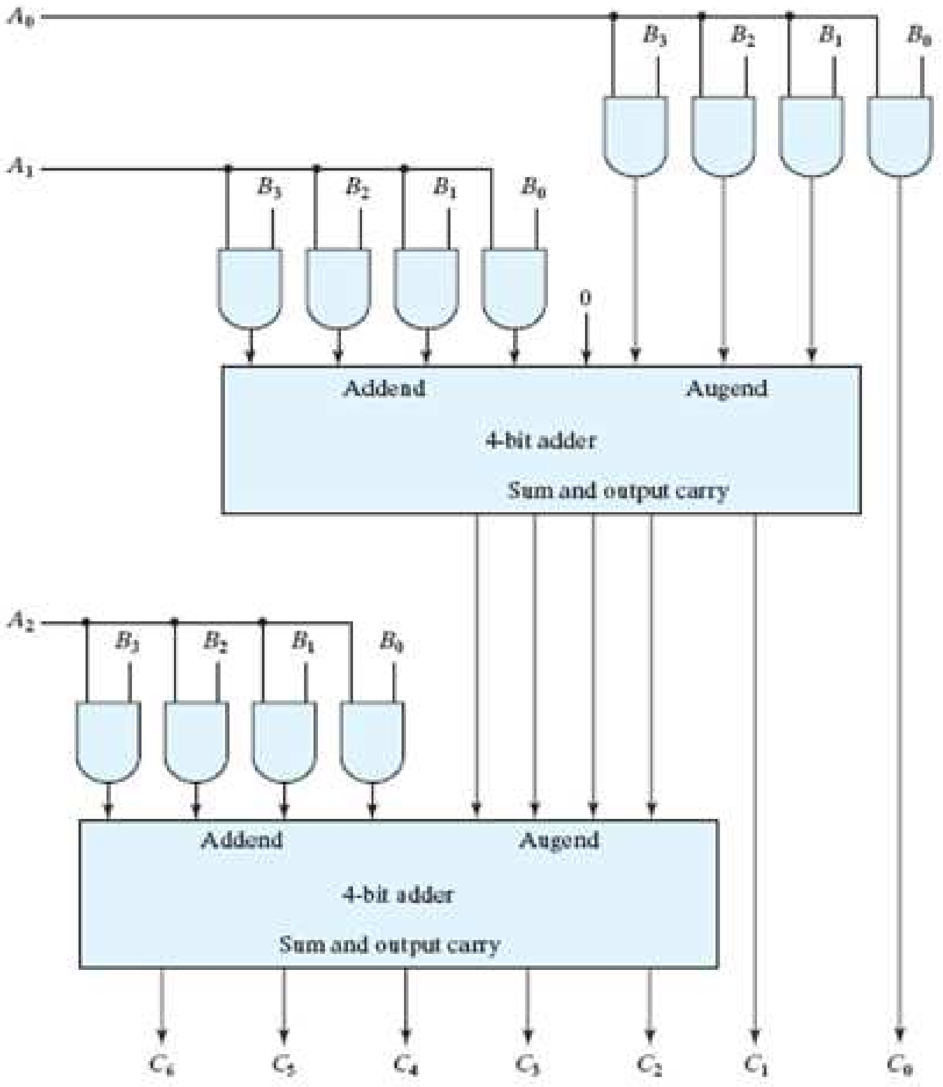
4.7 Binary Multiplier

- Multiplicand: $B_3\ B_2\ B_1\ B_0$
- Multiplier: $\underline{\times\ A_2\ A_1\ A_0}$
- Adder 1: $A_0B_3\ A_0B_2\ A_0B_1\ A_0B_0$

  $A_1B_3\ A_1B_2\ A_1B_1\ A_1B_0$
- Adder 2: $A_2B_3\ A_2B_2\ A_2B_1\ A_2B_0$
- The result will be (4 × 3) bits
  - We need (4 × 3) AND gates and two four-bit adders to produce a product of seven bits.

4.7 Binary Multiplier

- Four-bit by three-bit binary Multiplier $B_3B_2B_1B_0 \times A_2A_1A_0$



4.8 Magnitude Comparator

- The comparison of two numbers is an operation that determines whether one number is *greater than*, *less than* or *equal* to the other number.
- The circuit for comparing two *n-bit* numbers has 2^{2n} entries in the truth table
- Consider two numbers, A and *B*, with four bits each:

  $A = A_3\,A_2\,A_1\,A_0$

  $B = B_3\,B_2\,B_1\,B_0$

  – When the numbers are binary, the digits are either 1 or 0

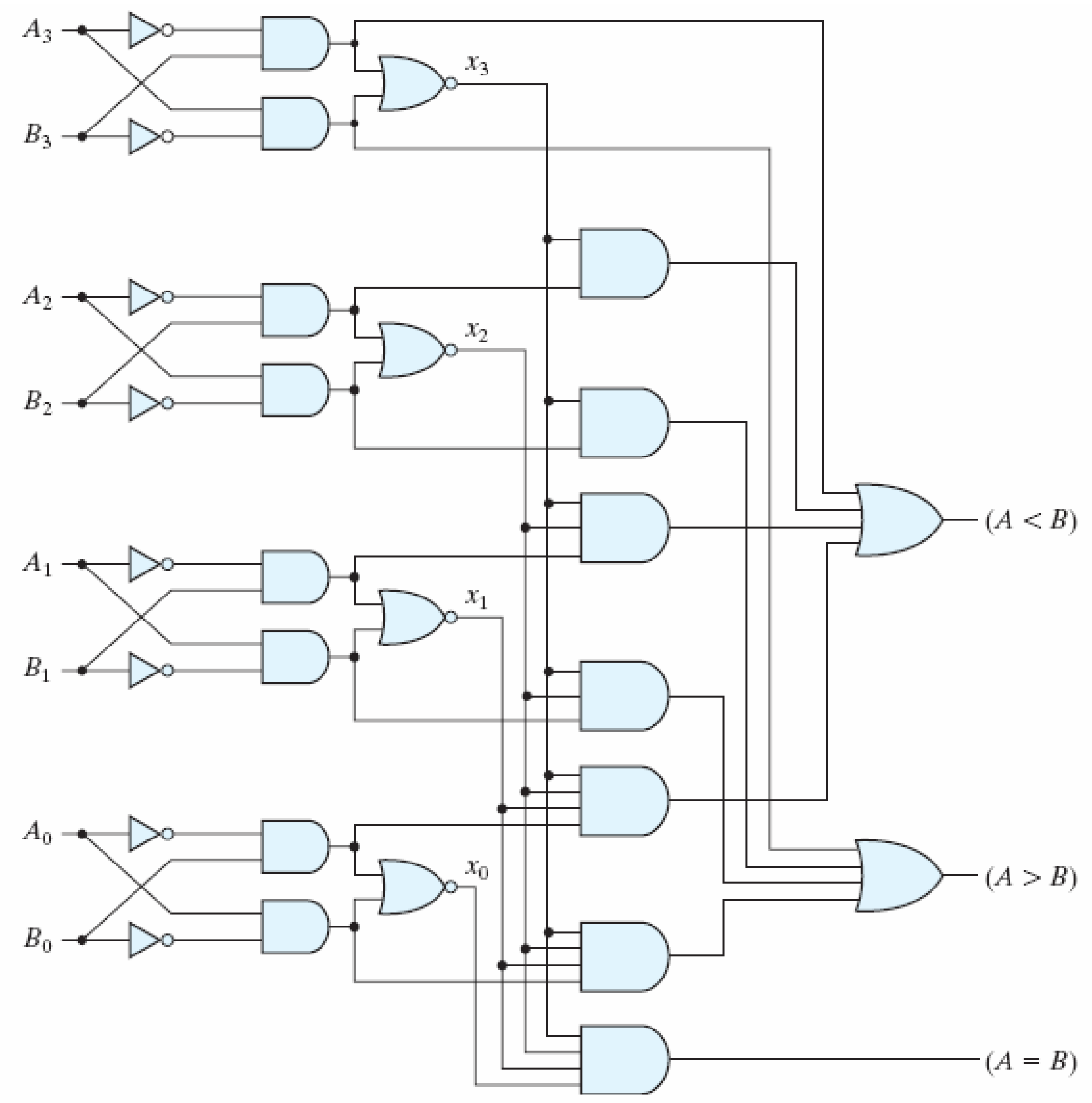
4.8 Magnitude Comparator

- The two numbers are equal if: $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$, *and* $A_0 = B_0$
  - *expressed logically with an exclusive-NOR*
- To determine whether A is *greater* or *less* than B, compare the next lower significant pair of digits
- So, let $x_i = (A_i \oplus B_i)' = A_i B_i + A_i' B_i'$, for $i = 0, 1, 2, 3$

  Then:

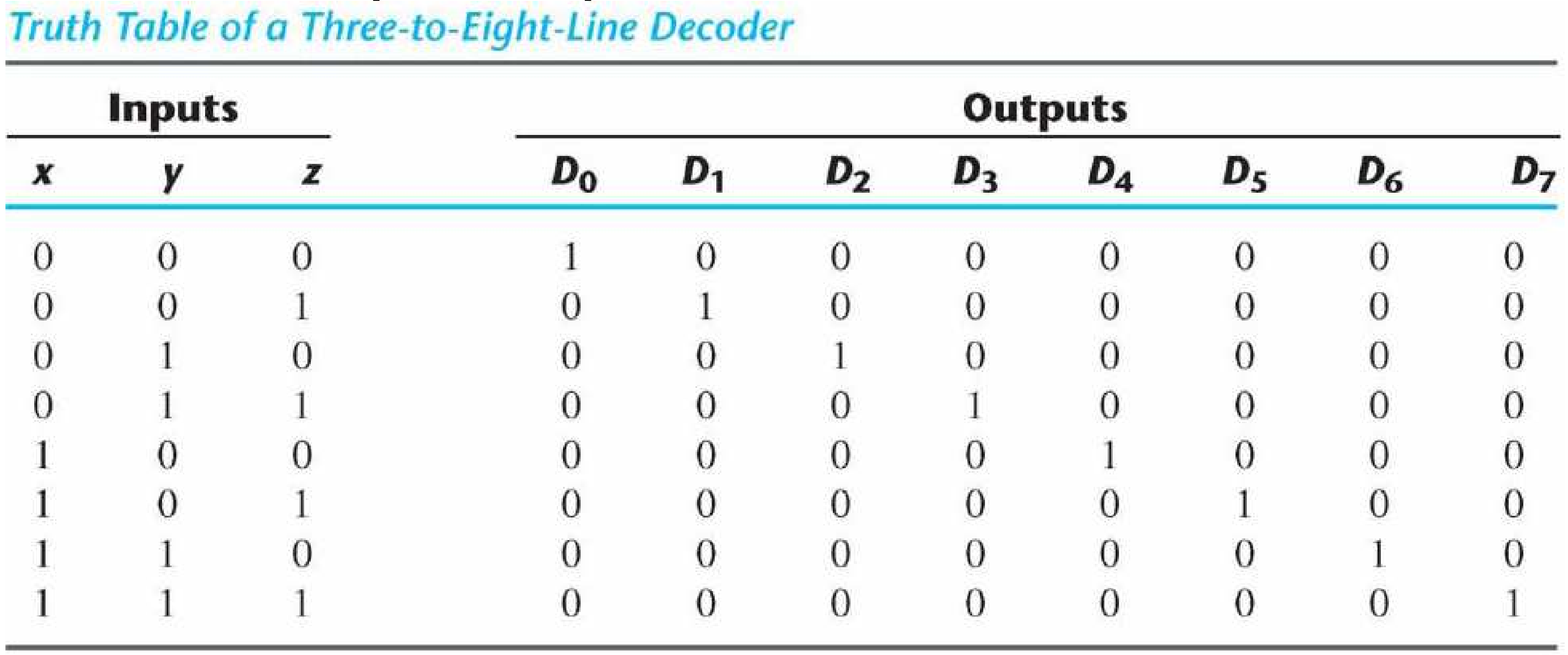
1. $(A = B) = x_3 x_2 x_1 x_0$
2. $(A > B) = A_3 B_3' + x_3 A_2 B_2' + x_3 x_2 A_1 B_1' + x_3 x_2 x_1 A_0 B_0'$
3. $(A < B) = A_3' B_3 + x_3 A_2' B_2 + x_3 x_2 A_1' B_1 + x_3 x_2 x_1 A_0' B_0$

4.8 Magnitude Comparator



4.9 Decoders

- A combinational circuit that *converts* binary information from *n* input lines to a maximum of 2^n unique output lines

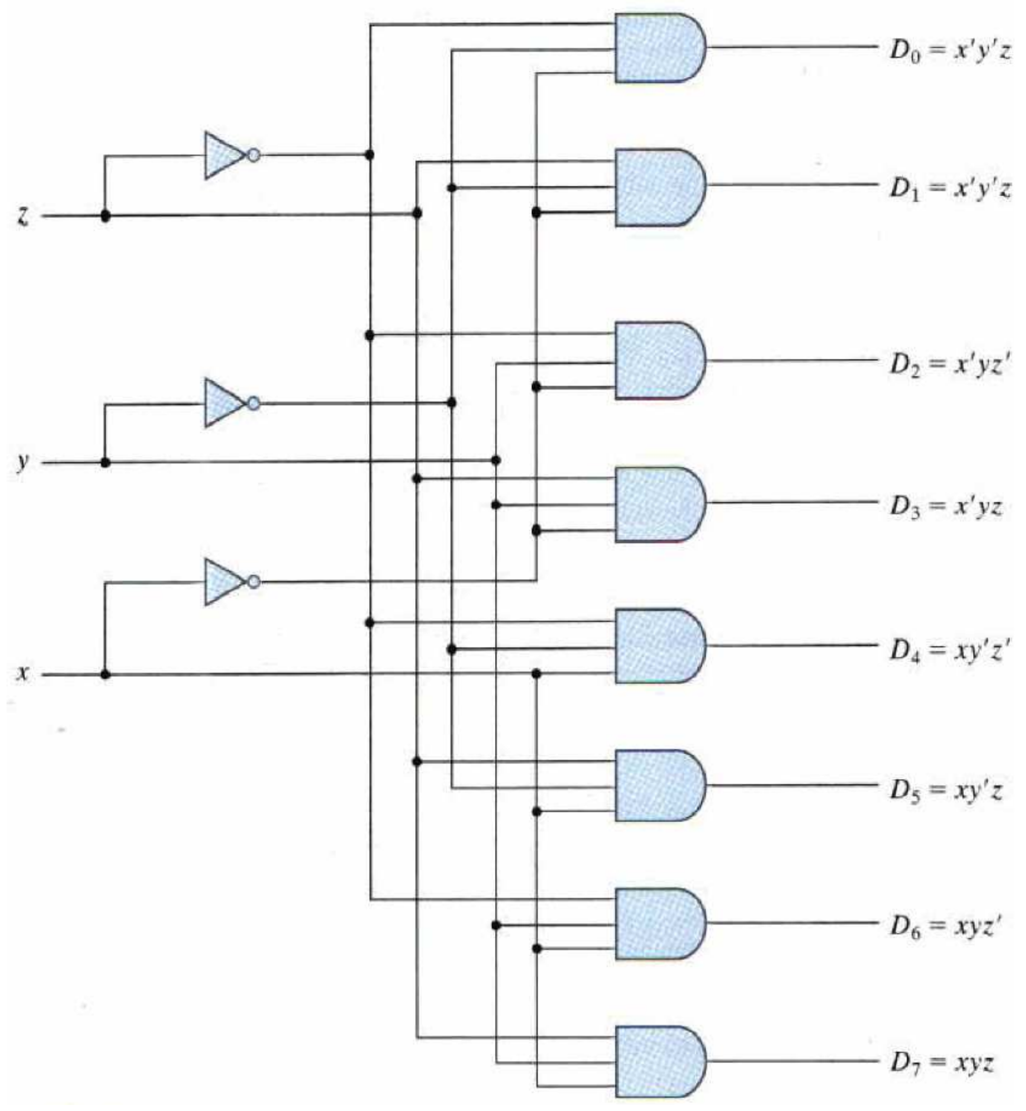


Truth Table of a Three-to-Eight-Line Decoder

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

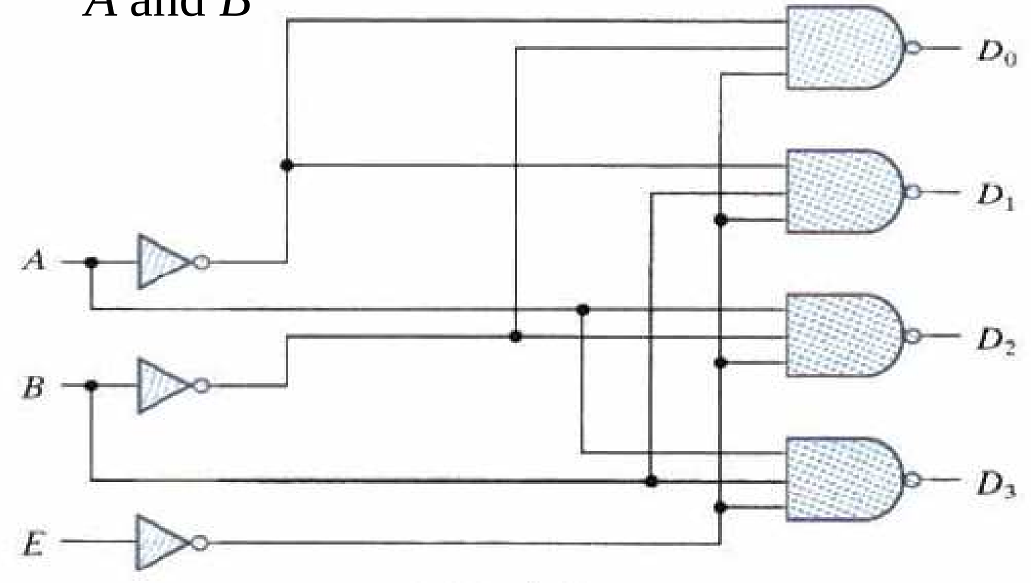
4.9 Decoders

- The three inputs are decoded into eight outputs
  - Each representing one of the minterm of the three input variables
- The input variables represents a binary number



4.9 Decoders
Two-to-four-line Decoder with Enable Input

- Some decoders are constructed with NAND gates
- Some also include one *or* more *enable* inputs to control the circuit operation
  - It operates with complemented outputs and a complemented enable
  - The circuit is disabled when *E* is equal to 1, regardless of the values of the other two inputs
  - The output whose value is equal to 0 represents the minterm selected by inputs *A* and *B*



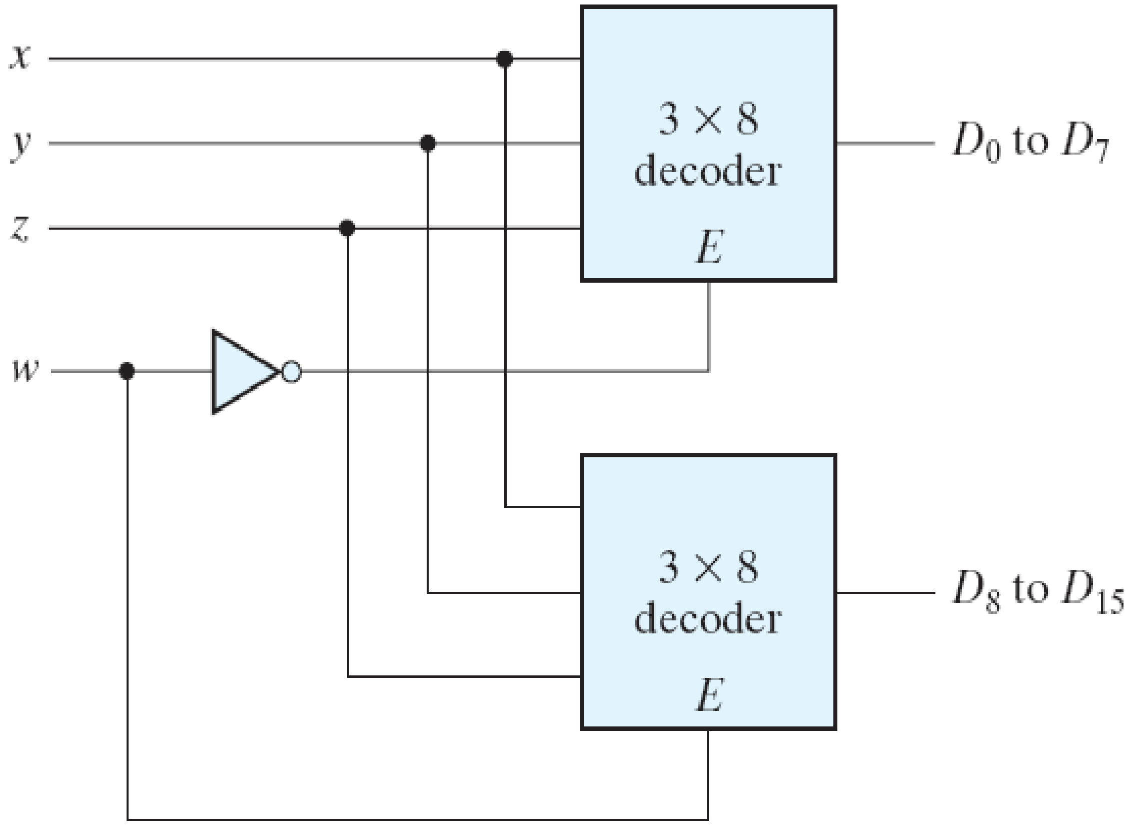
(a) Logic diagram

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	0

(b) Truth table

4.9 Decoders

- Decoders with enable inputs can be connected together to form a larger decoder circuit
- A 4 × 16 decoder can be constructed with two 3 × 8 decoder with enable
  - When *w* = 0, the top decoder is enabled
  - When *w* = 1, the enable conditions are reversed

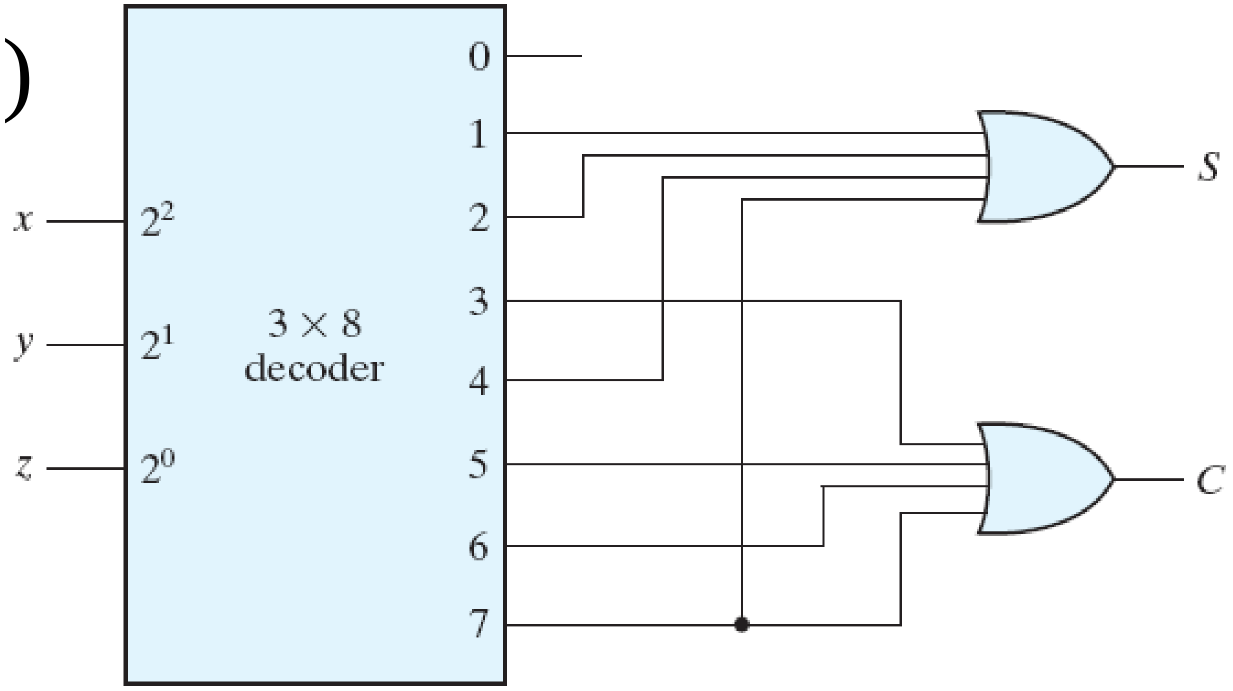


4.9 Decoders
Combinational Logic Implementation

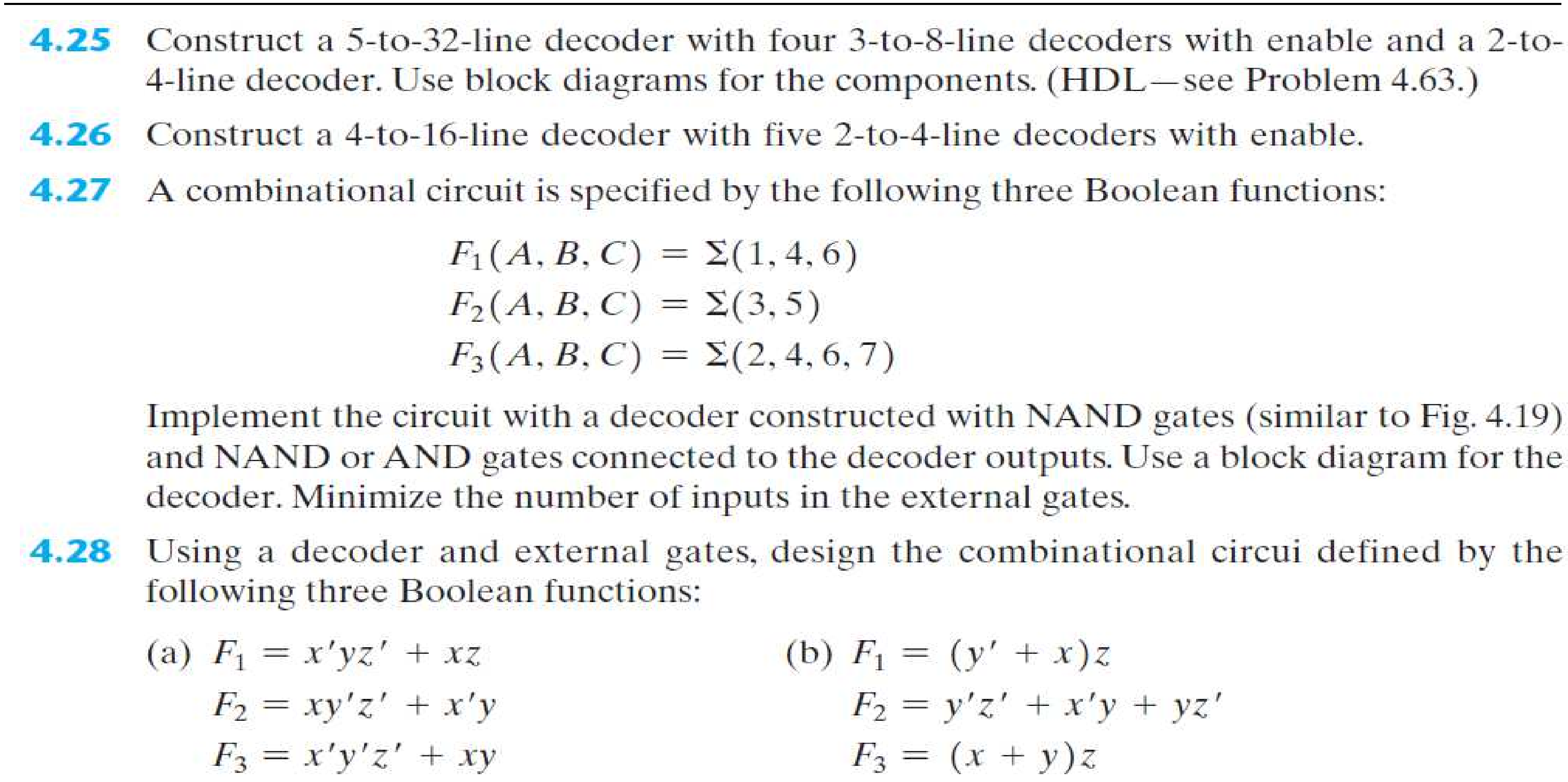
- A decoder provides the 2^n minterms of n input variables
- Any Boolean function can be expressed in *sum-of-minterms* form
- A decoder together with an external OR gate that forms their logical sum, provides a hardware implementation of the function
  - The inputs to each OR gate are selected from the decoder outputs according to the list of minterms of each function

4.9 Decoders
Combinational Logic Implementation

- Implementation of a full adder with a 3 × 8 decoder.
- S = Σ(1, 2, 4, 7)
- C = Σ(3, 5, 6, 7)



4.9 Decoders



4.25 Construct a 5-to-32-line decoder with four 3-to-8-line decoders with enable and a 2-to-4-line decoder. Use block diagrams for the components. (HDL—see Problem 4.63.)

4.26 Construct a 4-to-16-line decoder with five 2-to-4-line decoders with enable.

4.27 A combinational circuit is specified by the following three Boolean functions:

$$F_1(A, B, C) = \Sigma(1, 4, 6)$$
$$F_2(A, B, C) = \Sigma(3, 5)$$
$$F_3(A, B, C) = \Sigma(2, 4, 6, 7)$$

Implement the circuit with a decoder constructed with NAND gates (similar to Fig. 4.19) and NAND or AND gates connected to the decoder outputs. Use a block diagram for the decoder. Minimize the number of inputs in the external gates.

4.28 Using a decoder and external gates, design the combinational circui defined by the following three Boolean functions:

(a) $F_1 = x'yz' + xz$
$F_2 = xy'z' + x'y$
$F_3 = x'y'z' + xy$

(b) $F_1 = (y' + x)z$
$F_2 = y'z' + x'y + yz'$
$F_3 = (x + y)z$

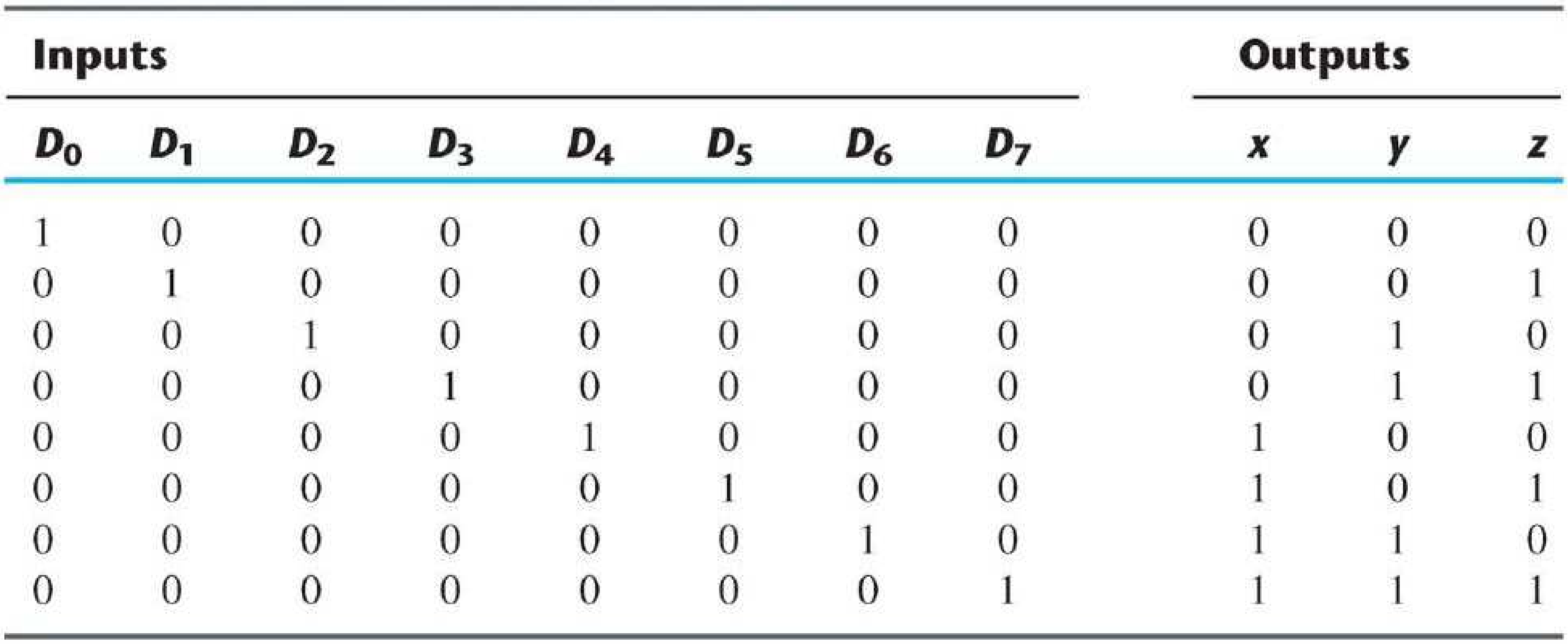
- Given a three-input Boolean function F(A, B, C) = **Σm** (0, 2, 4, 6, 7) + **Σ**d(1). Implement the function using a minimal number of **2-to-4 with enable decoders and a NOR gate**

4.10 Encoders

- A digital circuit that performs the *inverse operation of a decoder*
  - An encoder has 2^n (*or fewer*) input lines and n output lines
- The output lines, generates the binary code corresponding to the input value
- It can be implemented with OR gates
  - Their inputs are determined directly from the truth table

4.10 Encoders
Octal-to-Binary Encoder

- The encoder defined above has the limitation that only one input can be active at any given time.



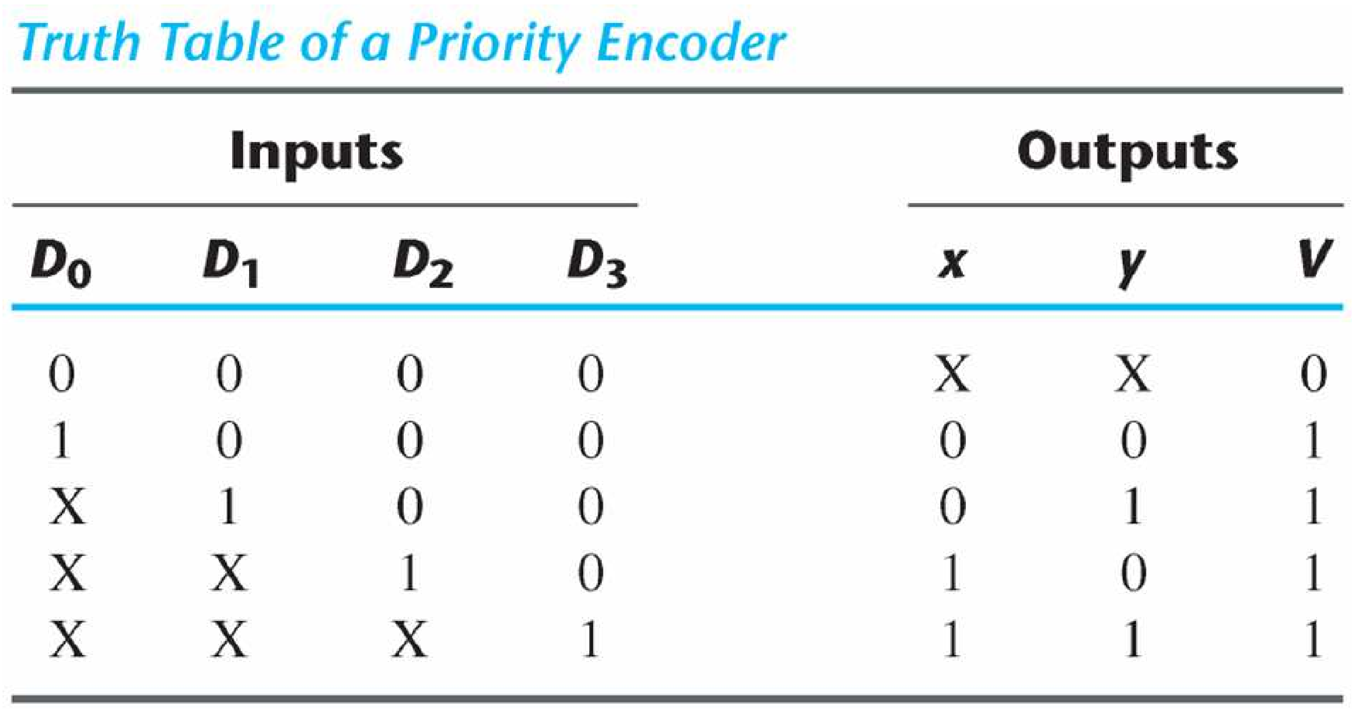
Inputs								**Outputs**		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

4.10 Encoders

- An encoder circuit that includes the priority function
- The operation of the priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

4.10 Encoders
Four-input Priority Encoder

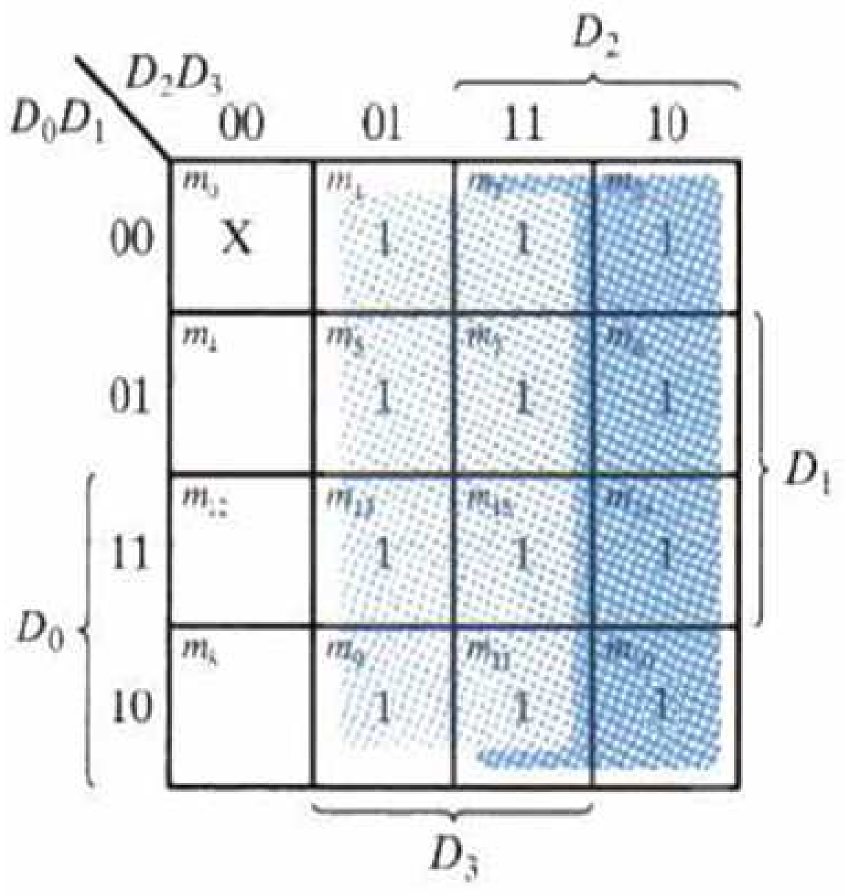
- Output variable *V* indicates whether a *valid input occurs.*
- It can be seen that the input variable D_3 is with the highest priority.



Truth Table of a Priority Encoder

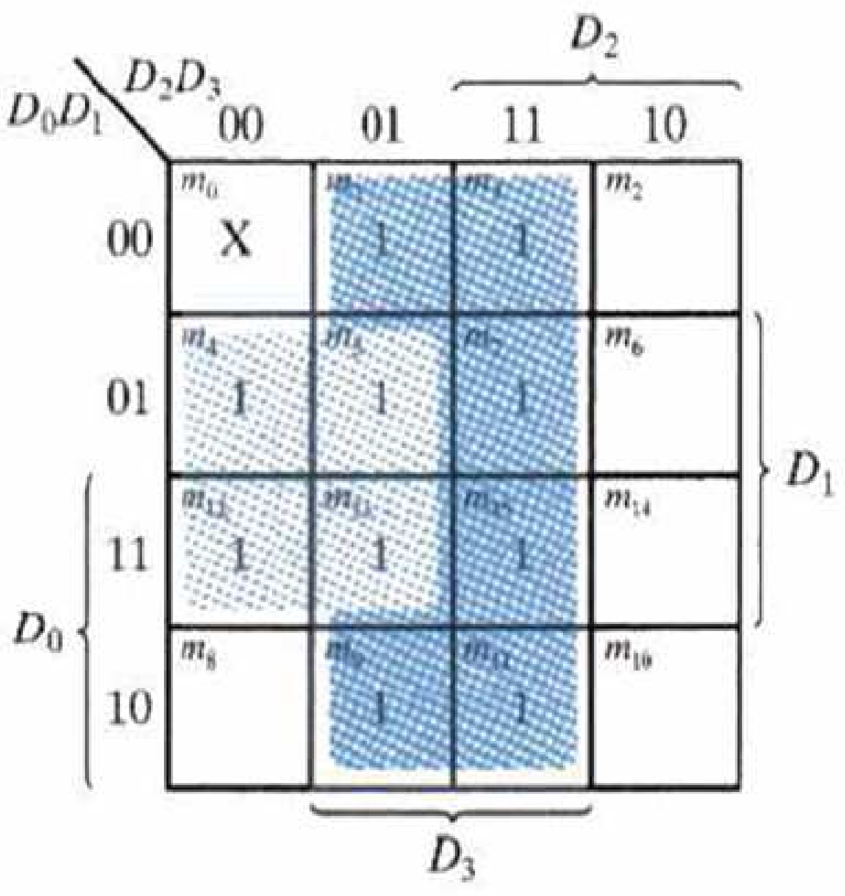
Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

4.10 Encoders
Four-input Priority Encoder



D_2D_3
D_0D_1
00 01 11 10
D_2
D_1
D_0
D_3
m_0 X
m_1 1 m_3 1 m_2 1
m_4 m_5 1 m_7 1 m_6 1
m_{12} m_{13} 1 m_{15} 1 m_{14} 1
m_8 m_9 1 m_{11} 1 m_{10} 1

$x = D_2 + D_3$

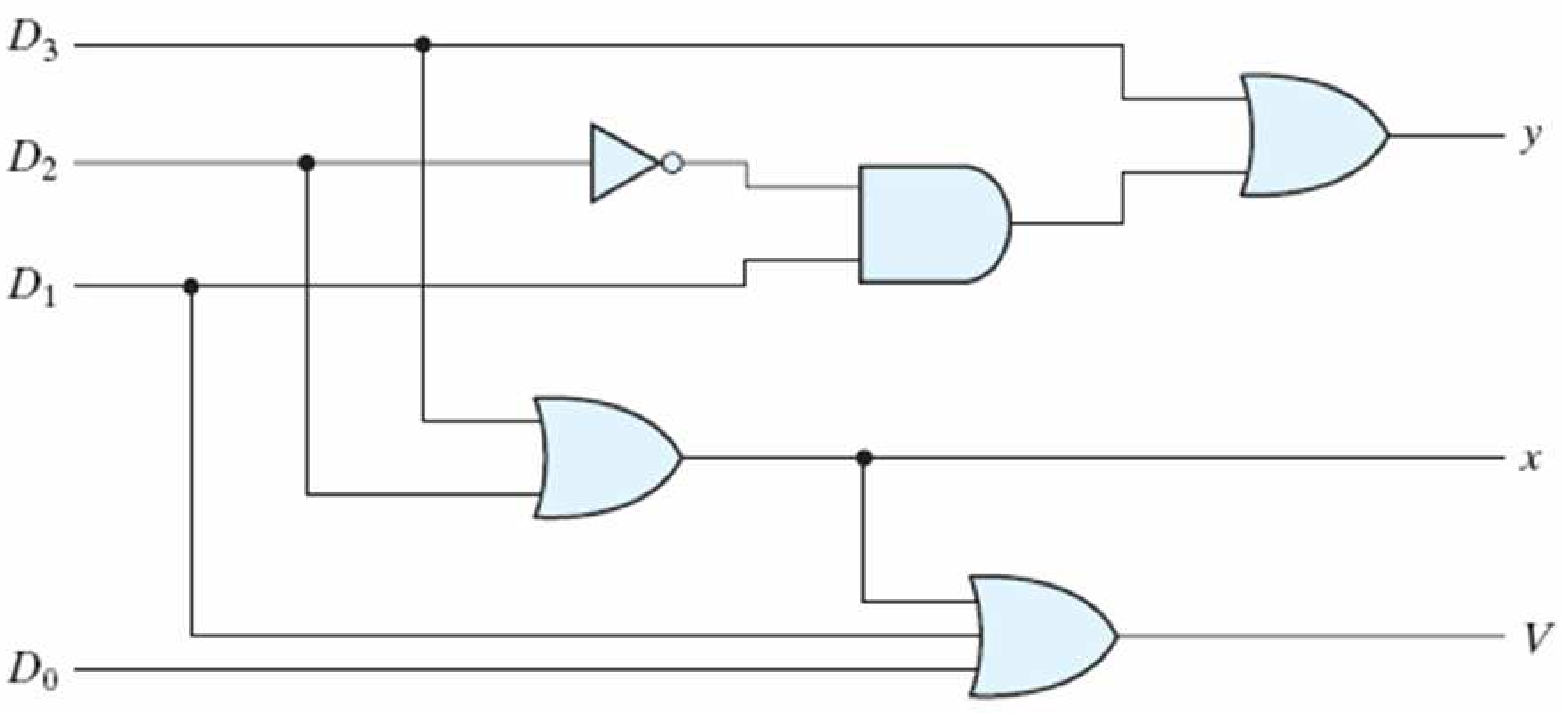


D_2D_3
D_0D_1
00 01 11 10
D_2
D_1
D_0
D_3
m_0 X
m_1 1 m_3 1 m_2
m_4 1 m_5 1 m_7 1 m_6
m_{12} 1 m_{13} 1 m_{15} 1 m_{14}
m_8 m_9 1 m_{11} 1 m_{10}

$y = D_3 + D_1D'_2$

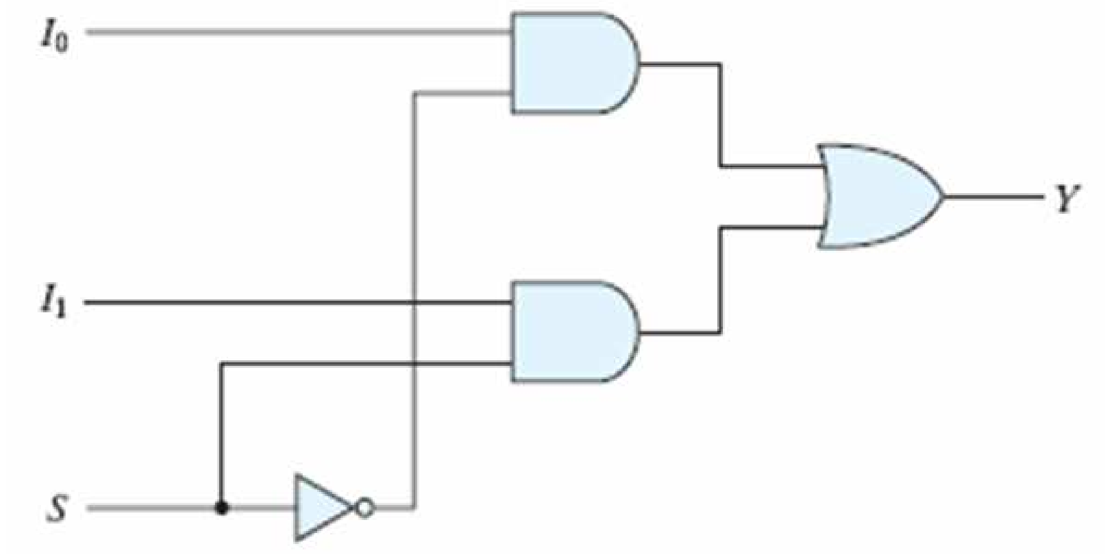
4.10 Encoders
Four-input Priority Encoder

- $V = D_0 + D_1 + D_2 + D_3$
- $x = D_2 + D_3$
- $y = D_3 + D_1D_2'$

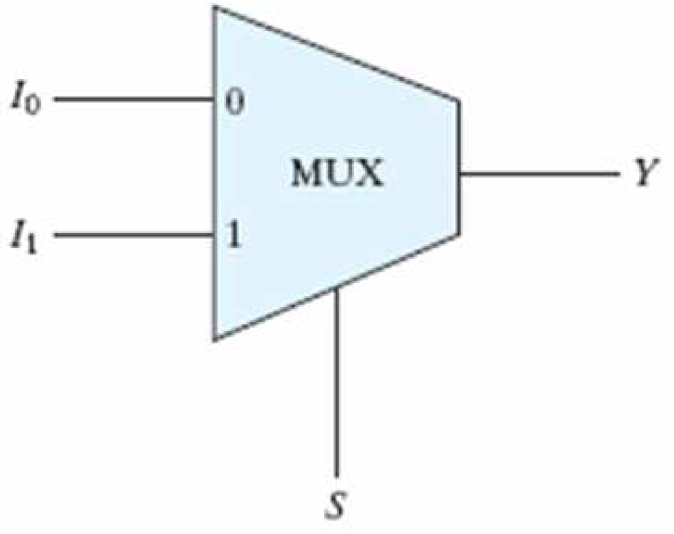


4.11 Multiplexers

- A combinational circuit that selects binary information from *one of many input lines* and directs it to a *single output line*
- The selection of a particular input line is contro1led by a set of *selection* lines
  - 2^n input lines and n selection lines whose bit combinations determine which input is selected

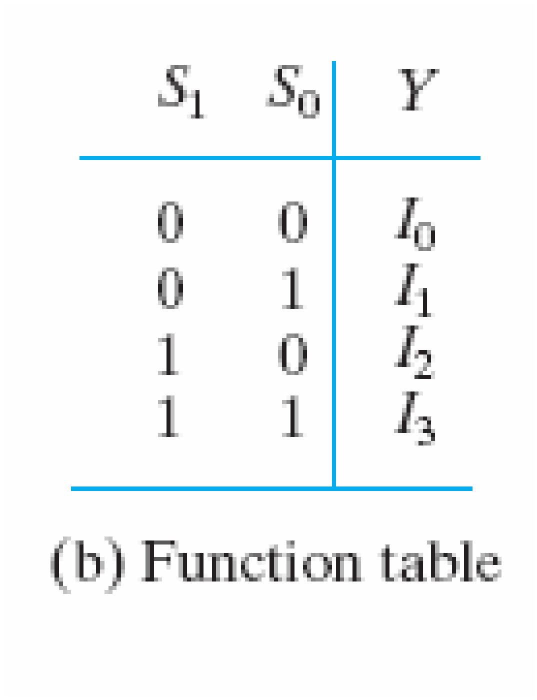


(a) Logic diagram



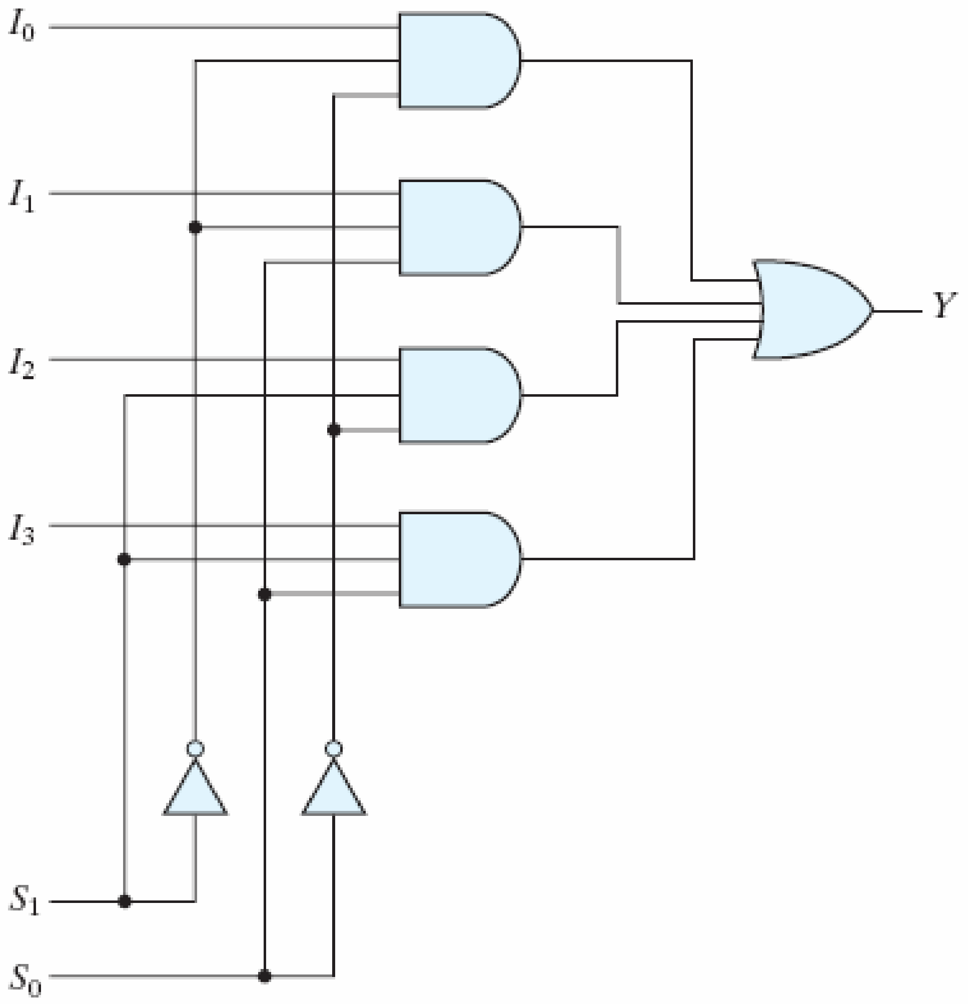
(b) Block diagram

4.11 Multiplexers
Four-to-one-line multiplexer



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

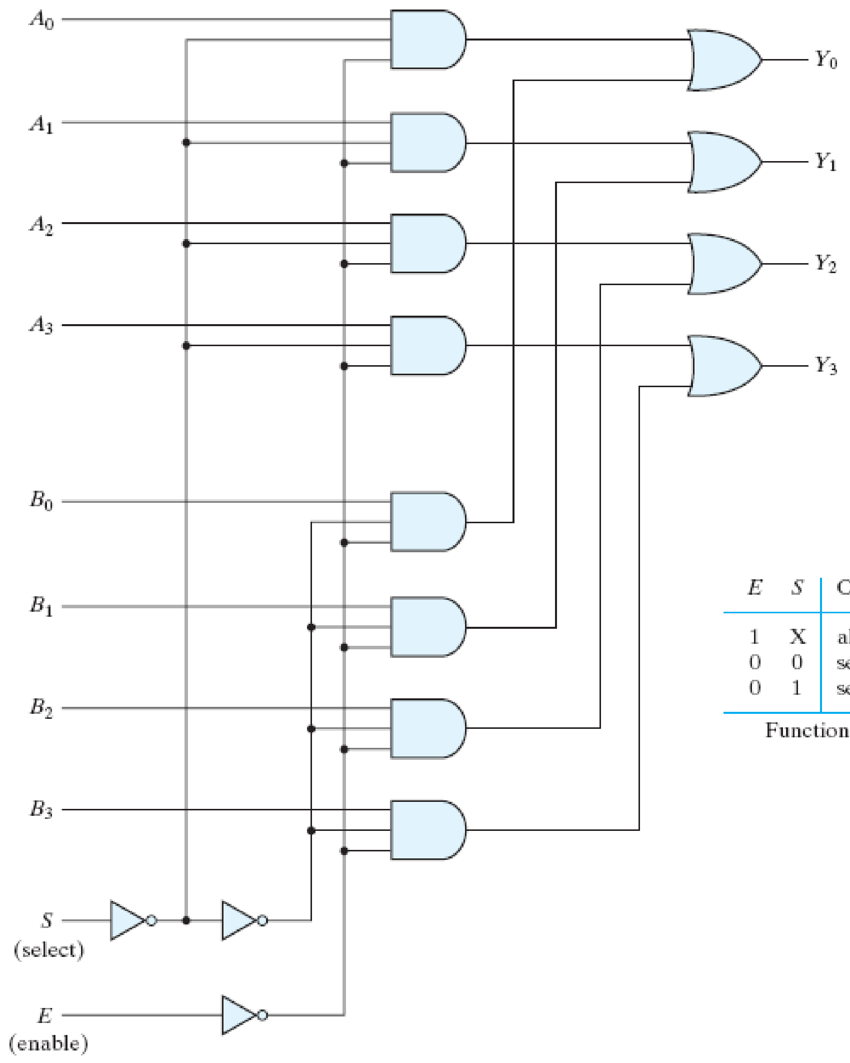
(b) Function table



(a) Logic diagram

4.11 Multiplexers *Quadruple 2-to-1-line multiplexer*

- Multiplexer circuits can be combined with common selection inputs to provide multiple-bit selection logic.
- Select 4 bits



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

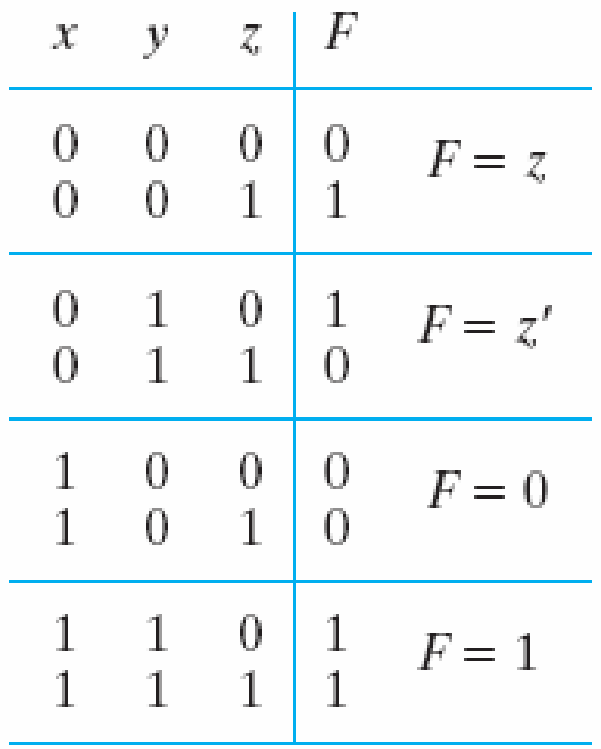
Function table

4.11 Multiplexers Boolean Function Implementation

- The minterms of a function are generated in a multiplexer by the circuit associated with the selection inputs
  - The individual minterms can be selected by the data input
- A Boolean function of *n* variables can be implemented with a multiplexer that has *n*-1 selection input
  - The first *n*-1 variables of the function are connected to the selection inputs of the multiplexer
  - The remaining single variable of the function is used for the data input

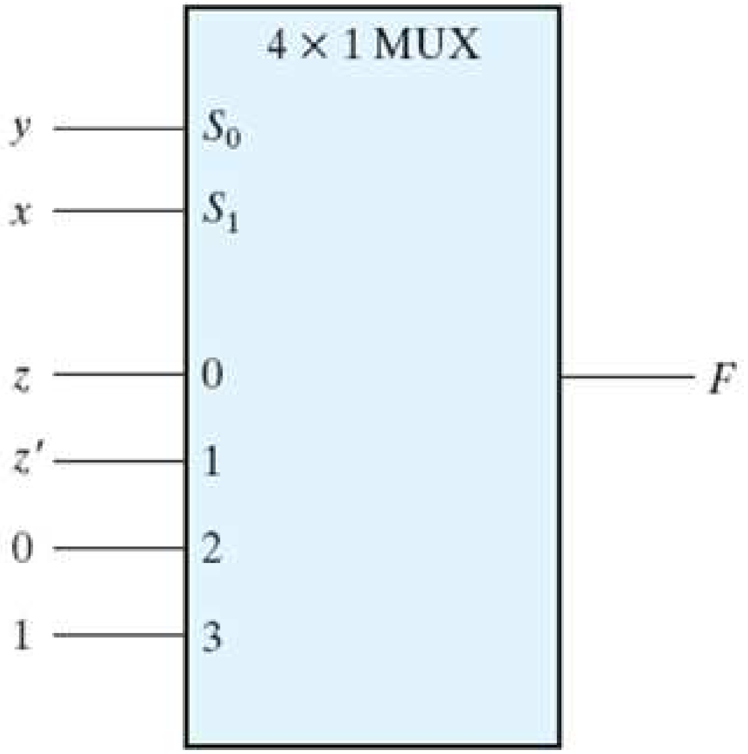
4.11 Multiplexers
Boolean Function Implementation

- Consider the Boolean function $F(x, y, z) = \sum(1,2,6,7)$
  - It can be implemented with a four-to-one-line multiplexer using x and y at the select lines



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

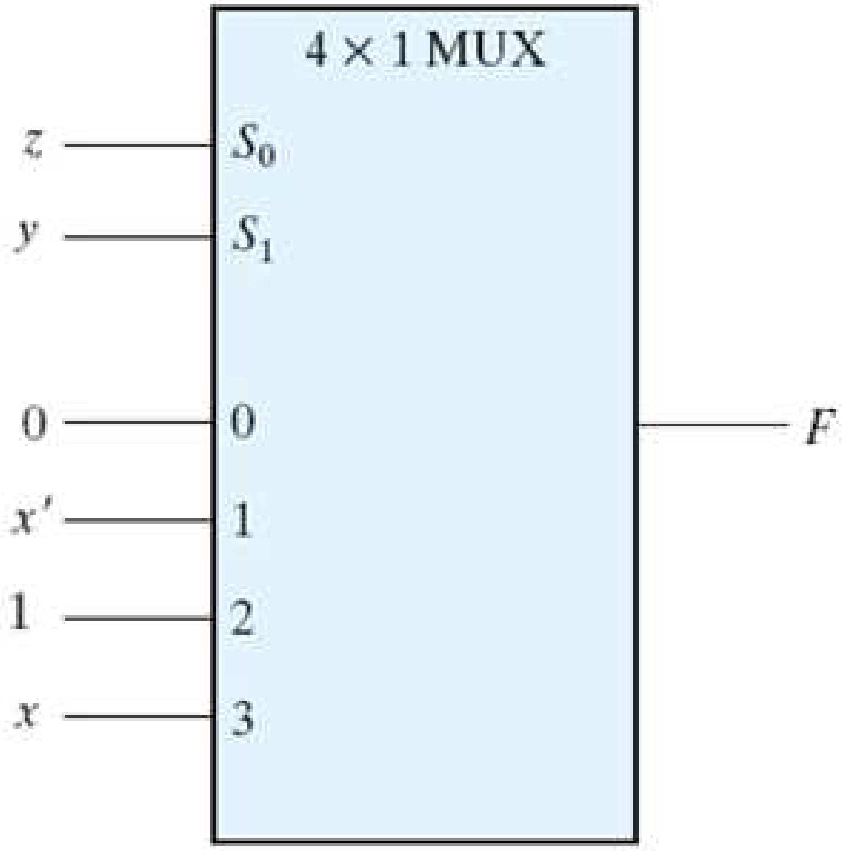


(b) Multiplexer implementation

4.11 Multiplexers
Boolean Function Implementation

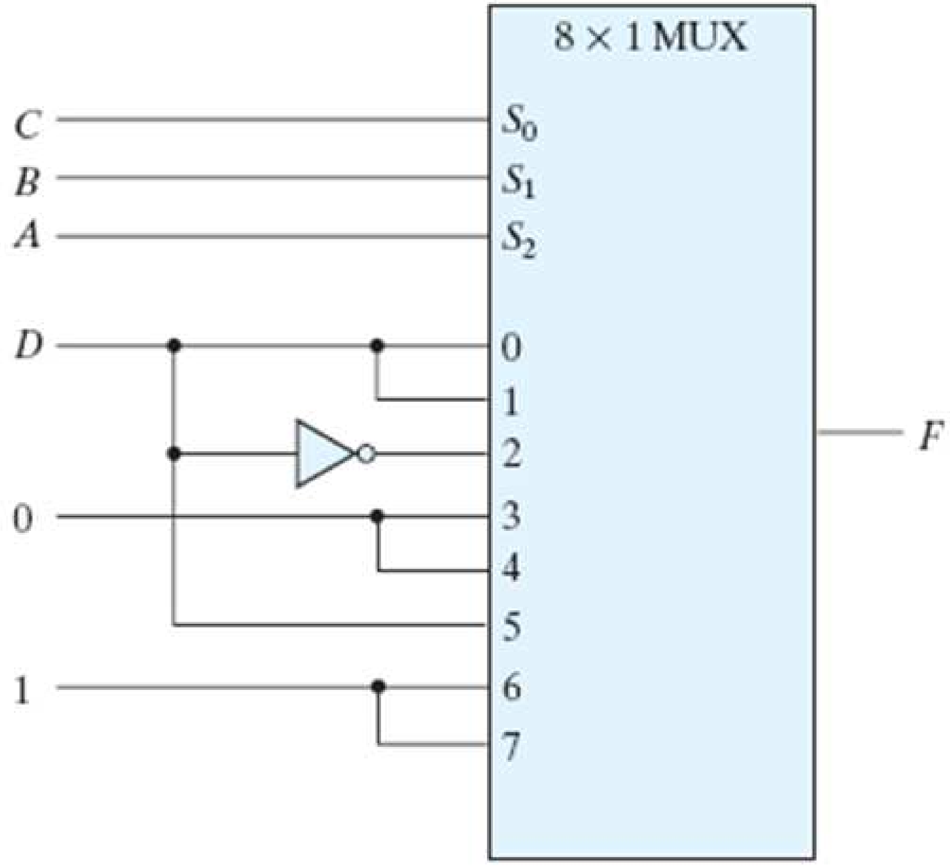
- Consider the Boolean function $F(x, y, z) = \sum(1,2,6,7)$
  - Another solution method using *x* and *y* at the select lines

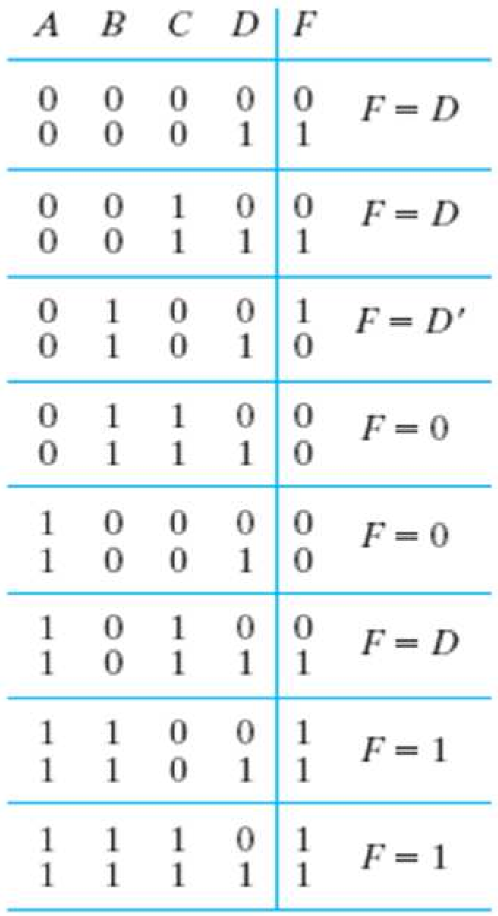
	I_0	I_1	I_2	I_3
x'	0	1	2	3
x	4	5	6	7
	0	x'	1	x



4.11 Multiplexers Boolean Function Implementation

- Consider the Boolean function $F(A, B, C, D) = \sum(1, 3, 4, 11, 12, 13, 14, 15)$ using inputs A, B and C at the select line





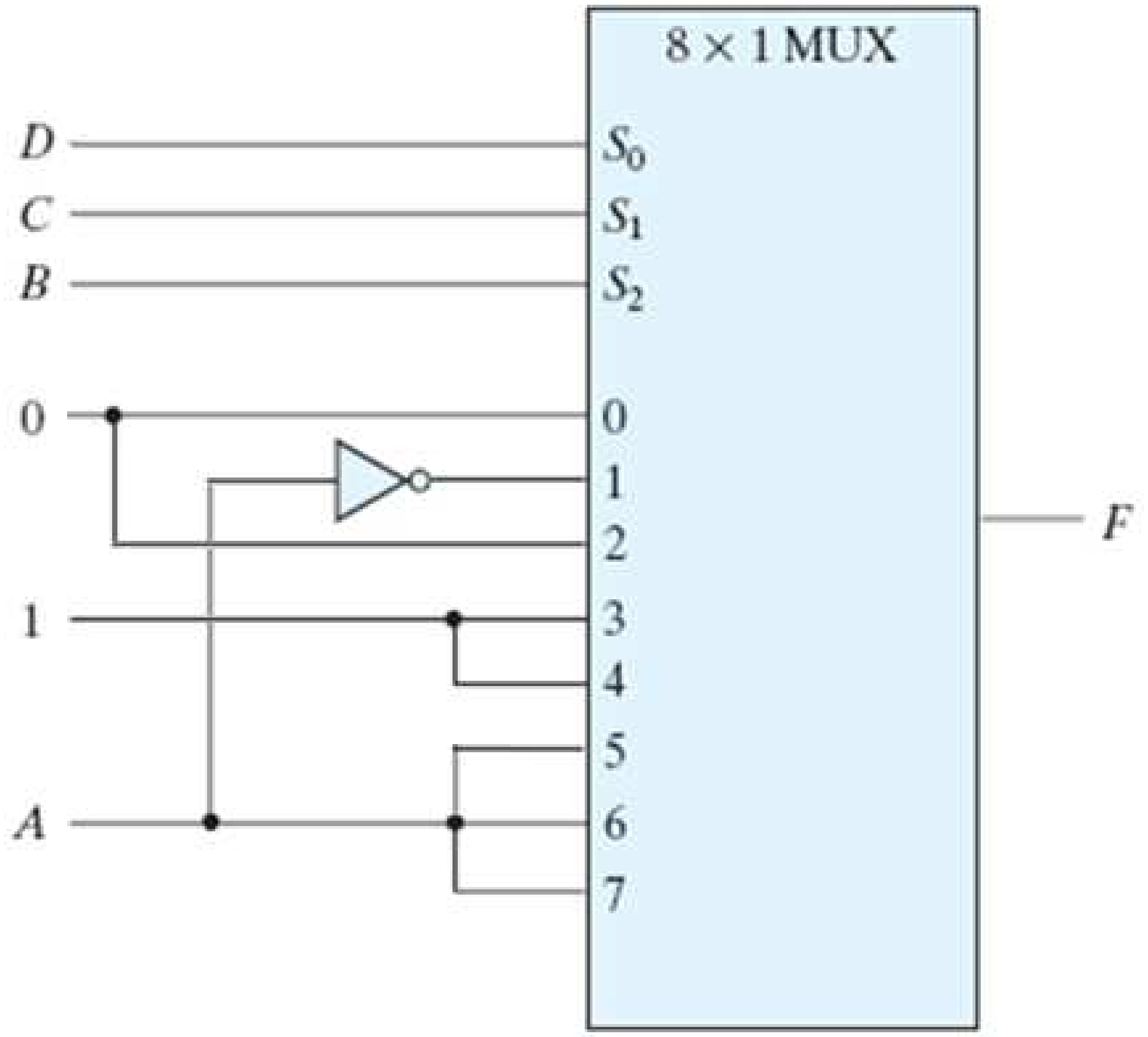
A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

4.11 Multiplexers
Boolean Function Implementation

- Consider the Boolean function
  $F(A, B, C, D) = \sum(1, 3, 4, 11, 12, 13, 14, 15)$
  using inputs B, C and D at the select lines

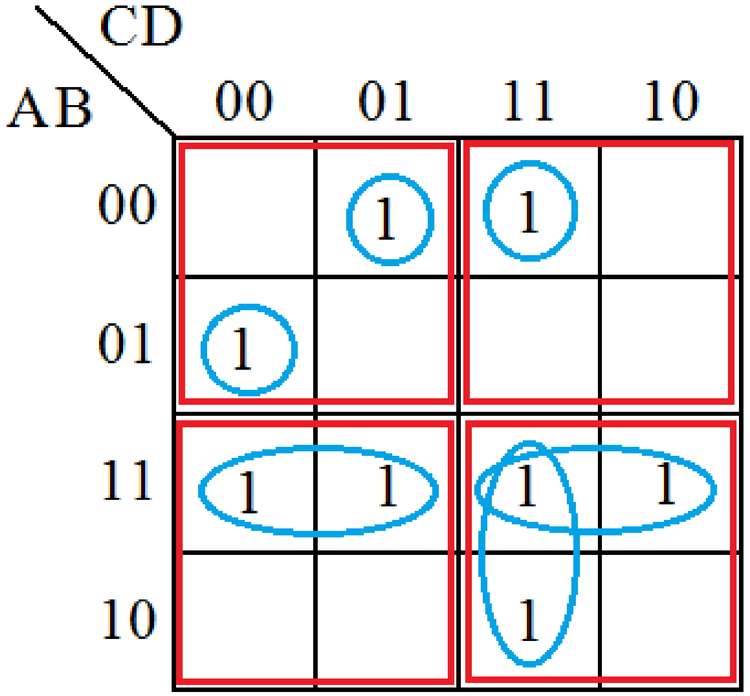
	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
A'	0	$\textcircled{1}$	2	$\textcircled{3}$	$\textcircled{4}$	5	6	7
A	8	9	10	$\textcircled{11}$	$\textcircled{12}$	$\textcircled{13}$	$\textcircled{14}$	$\textcircled{15}$
	0	A'	0	1	1	A	A	A

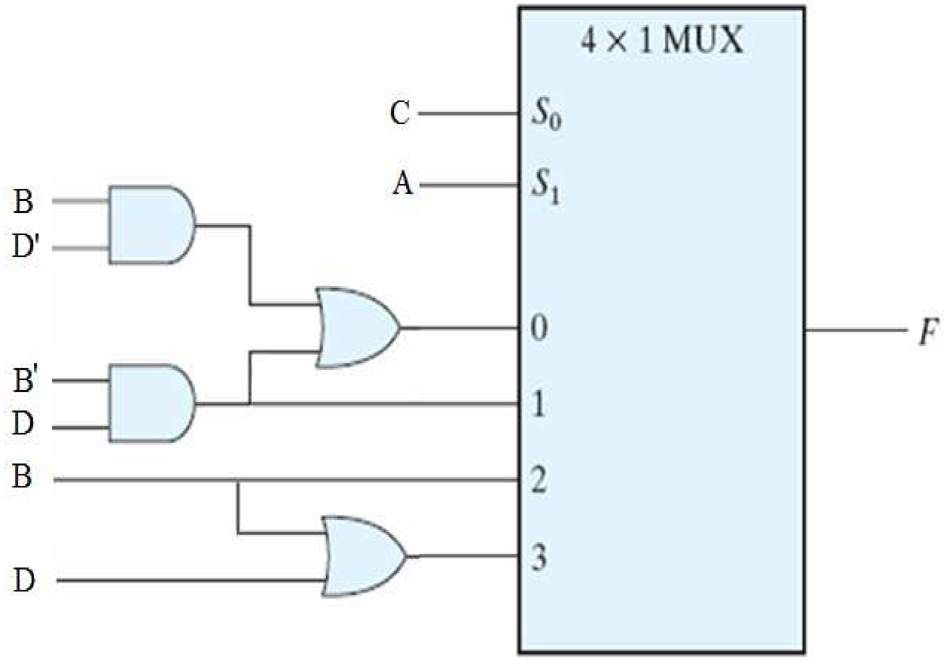
4.11 Multiplexers
Boolean Function Implementation



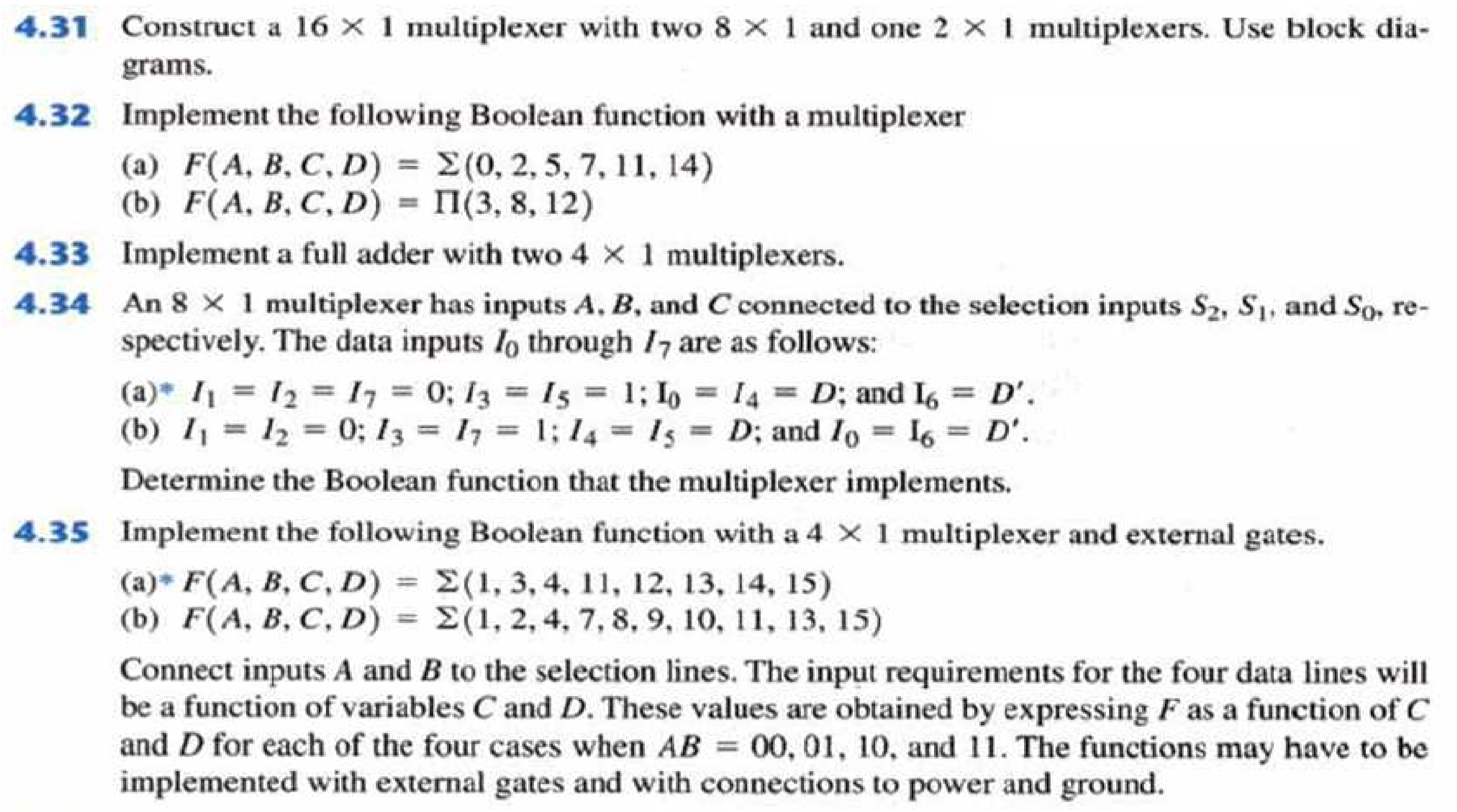
4.11 Multiplexers
Boolean Function Implementation

- Consider the Boolean function $F(A, B, C, D) = \sum(1, 3, 4, 11, 12, 13, 14, 15)$ using inputs ***A*** and ***C*** at the select lines, while inputs ***B*** and ***D*** at the input lines
  - It can be implemented with a four-to-one-line multiplexer
- $I_o = B'D + BD'$ $\quad$ $I_1 = B'D$
- $I_2 = B$ $\quad$ $I_3 = B + D$





4.11 Multiplexers
Boolean Function Implementation



4.31 Construct a 16 × 1 multiplexer with two 8 × 1 and one 2 × 1 multiplexers. Use block diagrams.

4.32 Implement the following Boolean function with a multiplexer

(a) $F(A, B, C, D) = \Sigma(0, 2, 5, 7, 11, 14)$
(b) $F(A, B, C, D) = \Pi(3, 8, 12)$

4.33 Implement a full adder with two 4 × 1 multiplexers.

4.34 An 8 × 1 multiplexer has inputs A, B, and C connected to the selection inputs S_2, S_1, and S_0, respectively. The data inputs I_0 through I_7 are as follows:

(a)* $I_1 = I_2 = I_7 = 0$; $I_3 = I_5 = 1$; $I_0 = I_4 = D$; and $I_6 = D'$.
(b) $I_1 = I_2 = 0$; $I_3 = I_7 = 1$; $I_4 = I_5 = D$; and $I_0 = I_6 = D'$.

Determine the Boolean function that the multiplexer implements.

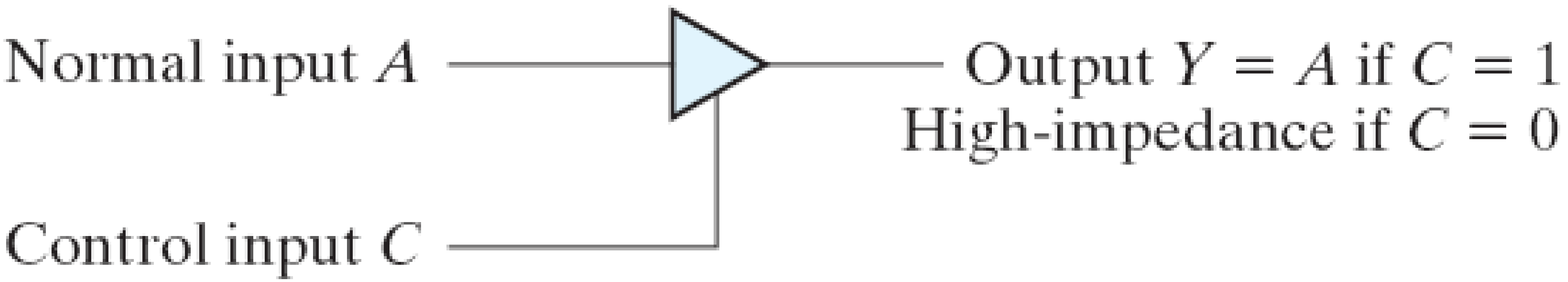
4.35 Implement the following Boolean function with a 4 × 1 multiplexer and external gates.

(a)* $F(A, B, C, D) = \Sigma(1, 3, 4, 11, 12, 13, 14, 15)$
(b) $F(A, B, C, D) = \Sigma(1, 2, 4, 7, 8, 9, 10, 11, 13, 15)$

Connect inputs A and B to the selection lines. The input requirements for the four data lines will be a function of variables C and D. These values are obtained by expressing F as a function of C and D for each of the four cases when $AB = 00, 01, 10$, and 11. The functions may have to be implemented with external gates and with connections to power and ground.

- Design a **full adder** with a minimal number of **2-to-1 multiplexers** (**Do not use any other gate**).

4.11 Multiplexers
Three-State Gates


Normal input A
Control input C
Output $Y = A$ if $C = 1$
High-impedance if $C = 0$