CO
Unit-2

Part-2

Combinational Logic

Introduction

- Logic circuits for digital systems may be combinational or sequential.
- Combinational circuits:
  - Consist of **logic gates only**
  - Outputs are determined from the present values of inputs
  - Operations can be specified by a set of Boolean functions
- Sequential circuits:
  - Consist of **logic gates and storage elements**
  - Outputs are a function of the inputs and the state of the storage elements
    - Depend not only on present inputs, but also on past values
  - Circuit behavior must be specified by a time sequence of inputs and internal states

COMBINATIONAL CIRCUITS

- A combinational circuit consists of
  - Input variables
  - Logic gates
  - Output variables

Fig. 4-1 Block Diagram of Combinational Circuit

COMBINATIONAL CIRCUITS

- Each input and output variable is a binary signal
  - Represent logic 1 and logic 0
- There are 2^n possible binary input combinations for n input variable
- Only one possible output value for each possible input combination
- Can be specified with a truth table
- Can also be described by m Boolean functions, one for each output variable
  - Each output function is expressed in terms of n input variables

ANALYSIS PROCEDURE

- The analysis of a combinational circuit requires that we determine the function that the circuit implements.
- This task starts with a given logic diagram and culminates with a set of Boolean functions, a truth table, or, possibly, an explanation of the circuit operation.
- The analysis can be performed manually by finding the Boolean functions or truth table or by using a computer simulation program.

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.**
- Once the logic diagram is verified to be that of a combinational circuit, one can proceed to obtain the output Boolean functions or the truth table.

ANALYSIS PROCEDURE

- To obtain the output Boolean functions from a logic diagram, proceed as follows:

1. Label all gate outputs that are a function of input variables with arbitrary symbols. Determine the Boolean functions for each gate output.

2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean functions for these gates.

ANALYSIS PROCEDURE

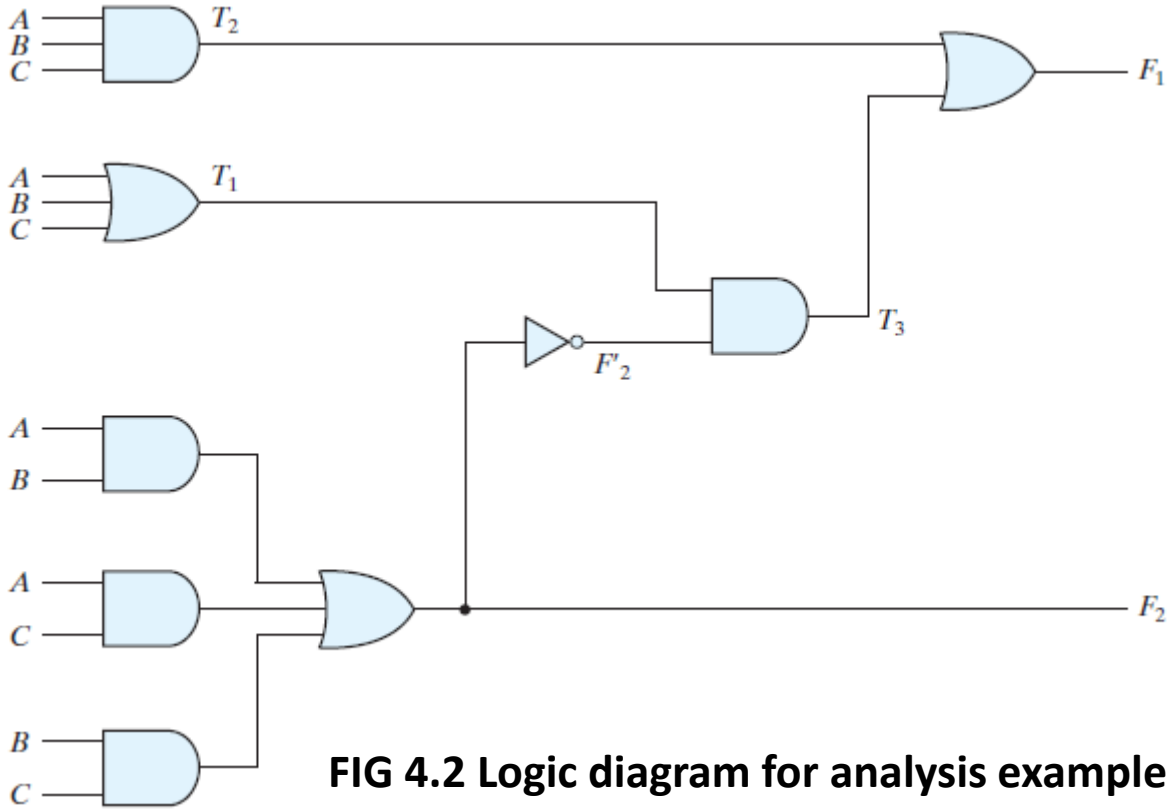
3. Repeat the process outlined in step 2 until the outputs of the circuit are obtained.

4. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.

ANALYSIS PROCEDURE

Step 1:

- Label all gate outputs that are a function of input variables
- Determine Boolean functions for each gate output



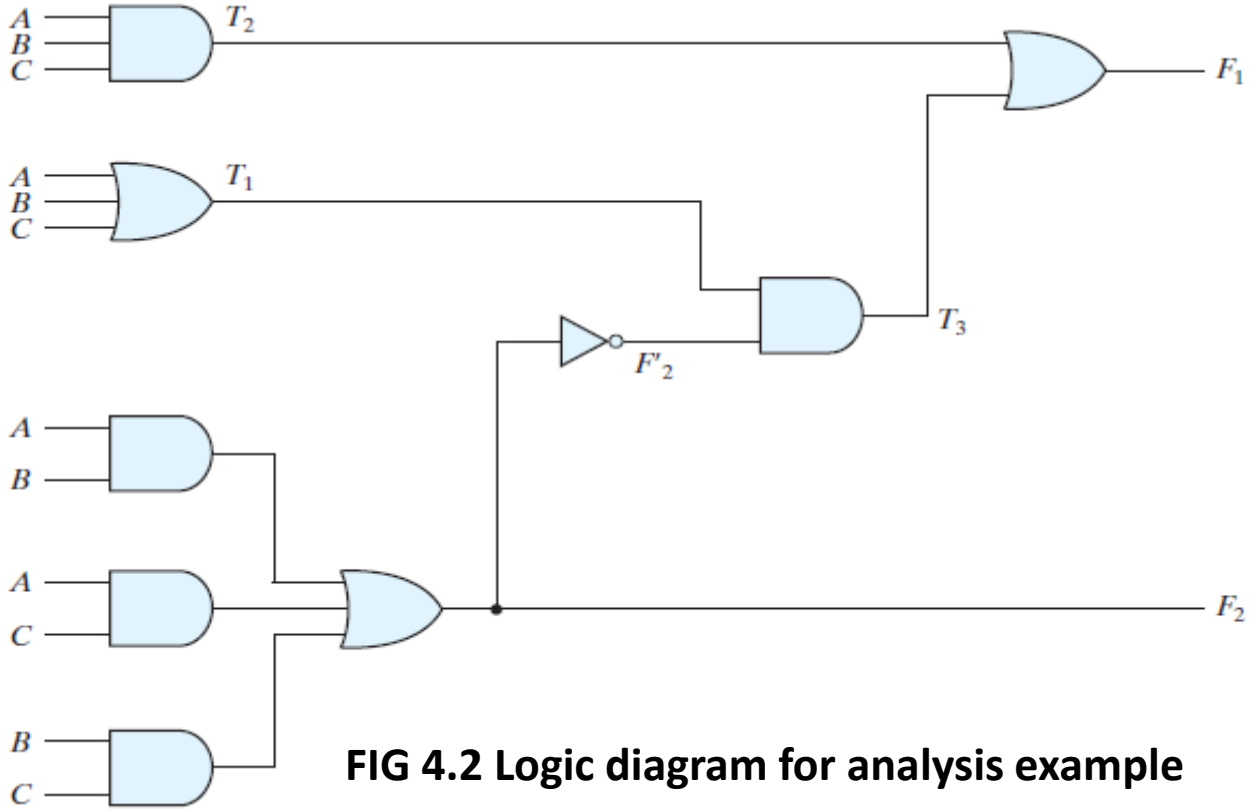
$$F_2 = AB + AC + BC$$
$$T_1 = A + B + C$$
$$T_2 = ABC$$

FIG 4.2 Logic diagram for analysis example

ANALYSIS PROCEDURE

Step 2:

- Label the gates that are a function of input variables a previously labeled gates
- Find the Boolean function for these gates



$$T_3 = F'_2 T_1$$
$$F_1 = T_3 + T_2$$

FIG 4.2 Logic diagram for analysis example

ANALYSIS PROCEDURE

Step 3:

- Obtain the output Boolean function in term of input variables
  - By repeated substitution of previously defined functions

$$F_1 = T_3 + T_2 = F'_2 T_1 + ABC$$

$$= (AB + AC + BC)'(A + B + C) + ABC$$

$$= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC$$

$$= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC$$

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

ANALYSIS PROCEDURE

- To obtain the truth table from the logic diagram:
  1. Determine the number of input variables

     For n inputs:
     - 2^n possible combinations
     - List the binary numbers from 0 to 2^n-1 in a table
  2. Label the outputs of selected gates
  3. Obtain the truth table for the outputs of those gates that are a function of the input variables only
  4. Obtain the truth table for those gates that are a function of previously defined variables at step 3
     - Repeatedly until all outputs are determined

ANALYSIS PROCEDURE

- We can derive the truth table in Table 4-1 by using the circuit of Fig.4-2.

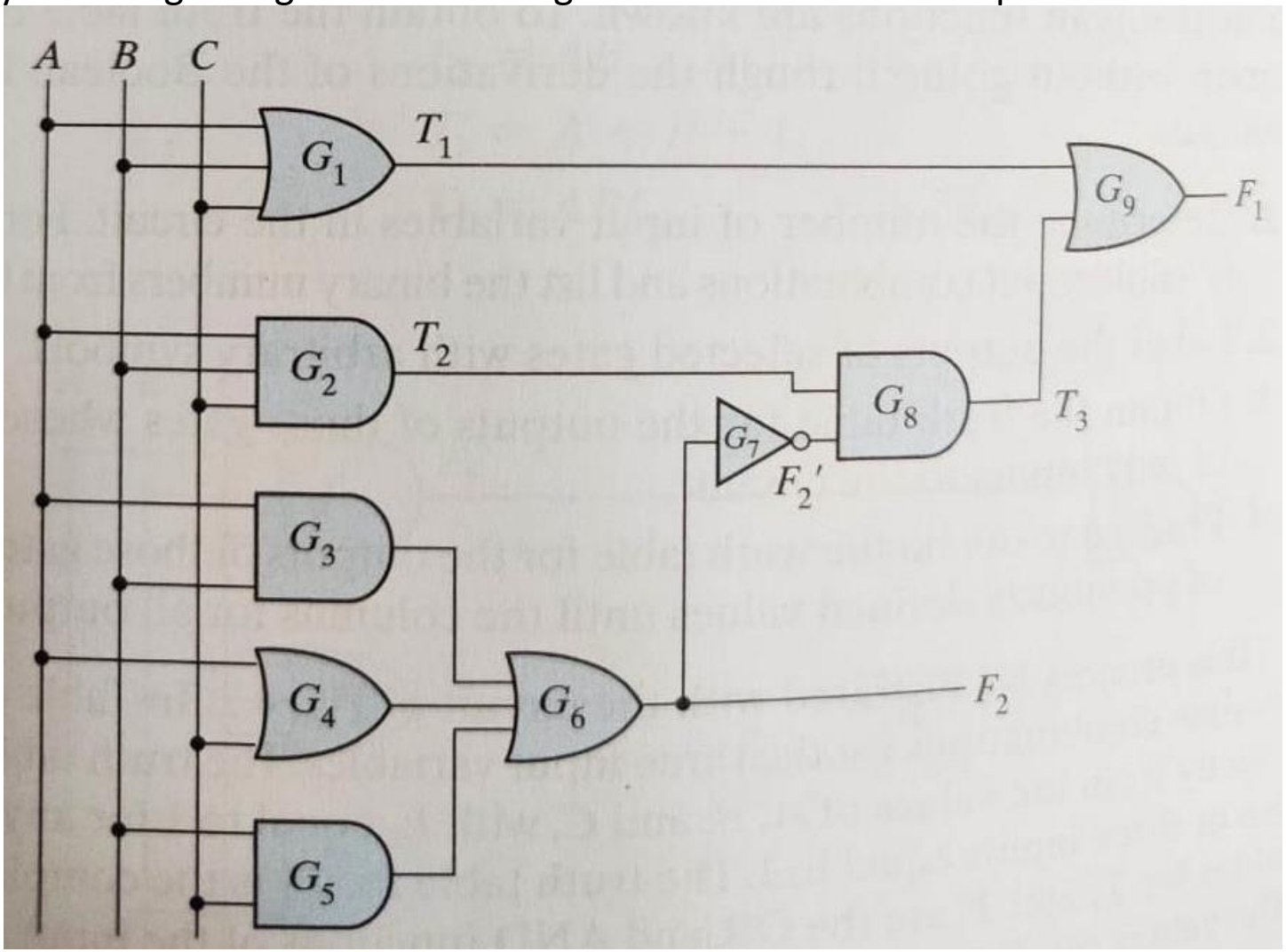
Table 4.1

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

ANALYSIS PROCEDURE

Analyze the logic diagram in below fig and find the boolean expressions for F1 and F2



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 functions from which the logic diagram can be obtained.

(or)

- Design procedure:
  - Input: the specification of the problem
  - Output: the logic circuit diagram (or Boolean functions)

Design Procedure

- The procedure involves the following steps:

1. From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.
2. Derive the truth table that defines the required relationship between inputs and outputs.
3. Obtain the simplified Boolean functions for each output as a function of the input variables.
4. Draw the logic diagram and verify the correctness of the design (manually or by simulation).

Code Conversion Example

- Convert from BCD code to Excess-3 code
- The 6 input combinations not listed are don't cares
- These values have no meaning in BCD
- We can arbitrary assign them to 1 or 0

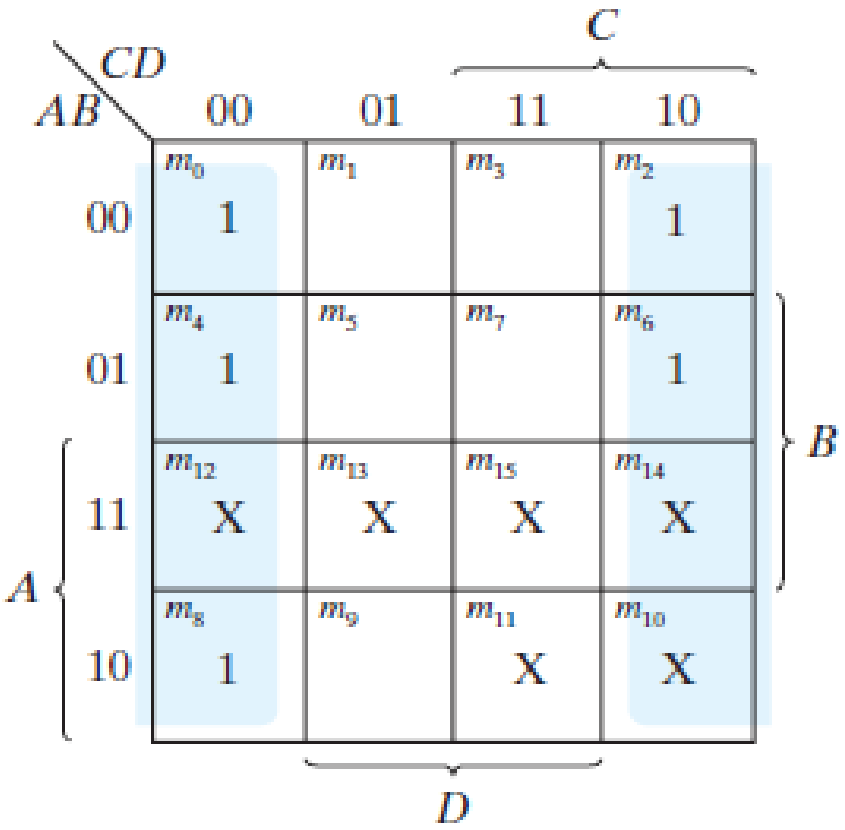
Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

Table 4.2 ***Truth Table for Code Conversion Example***

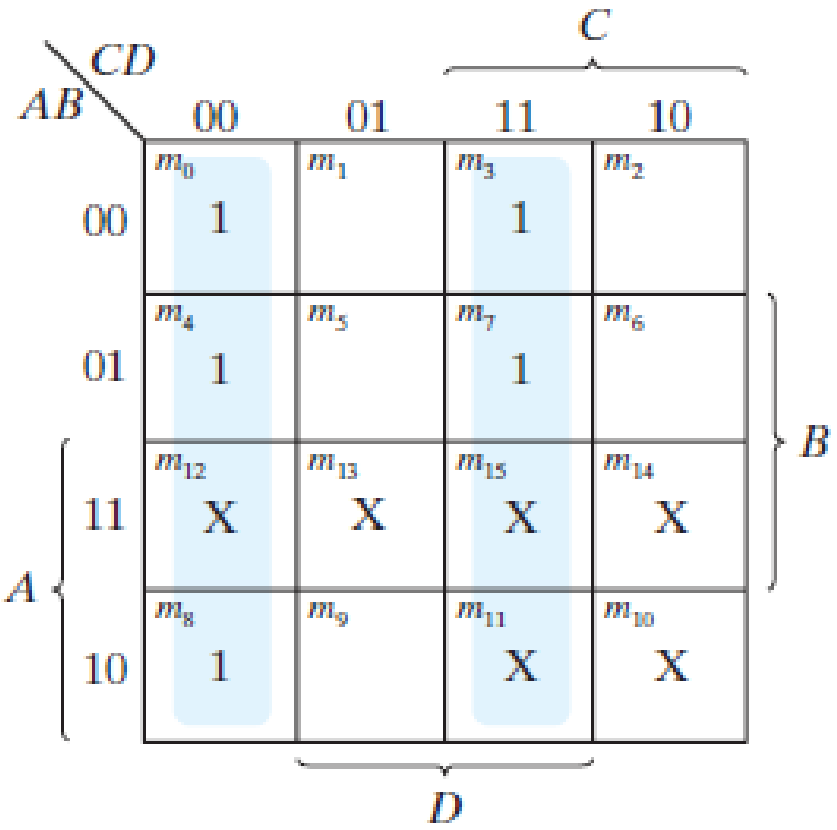
Maps for Code Converter (1/3)

- For each symbol of the Excess-3 code, we use 1's to draw the map for simplifying Boolean function.
- The six don't care minterms 10 through 15 are marked with X.

Maps for Code Converter (2/3)

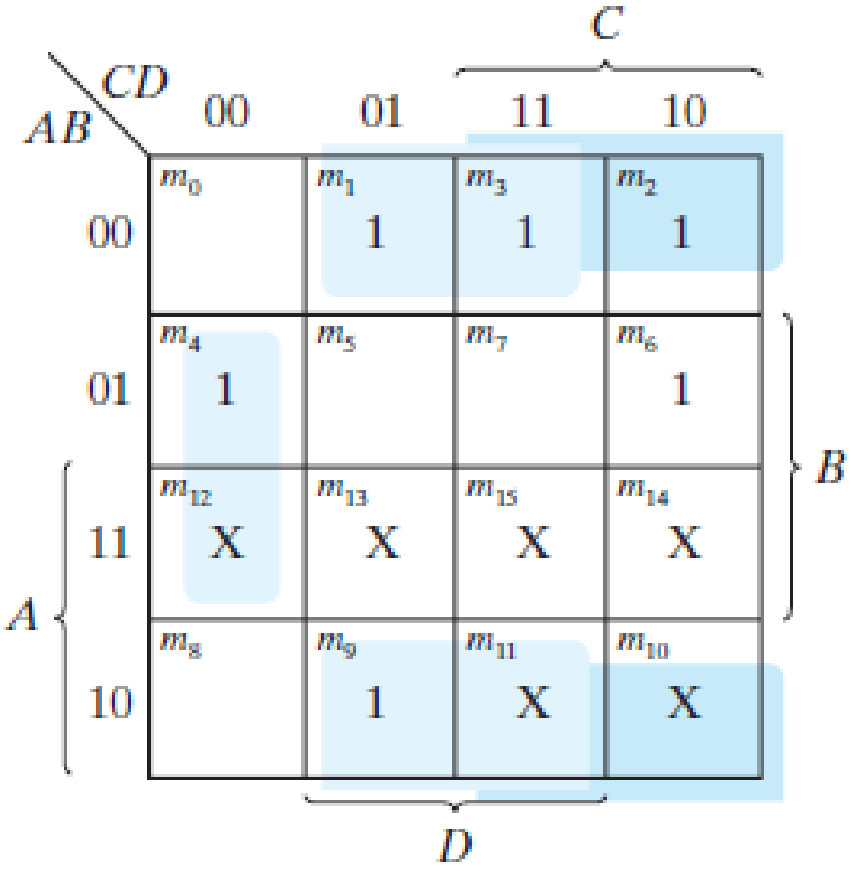


$z = D'$

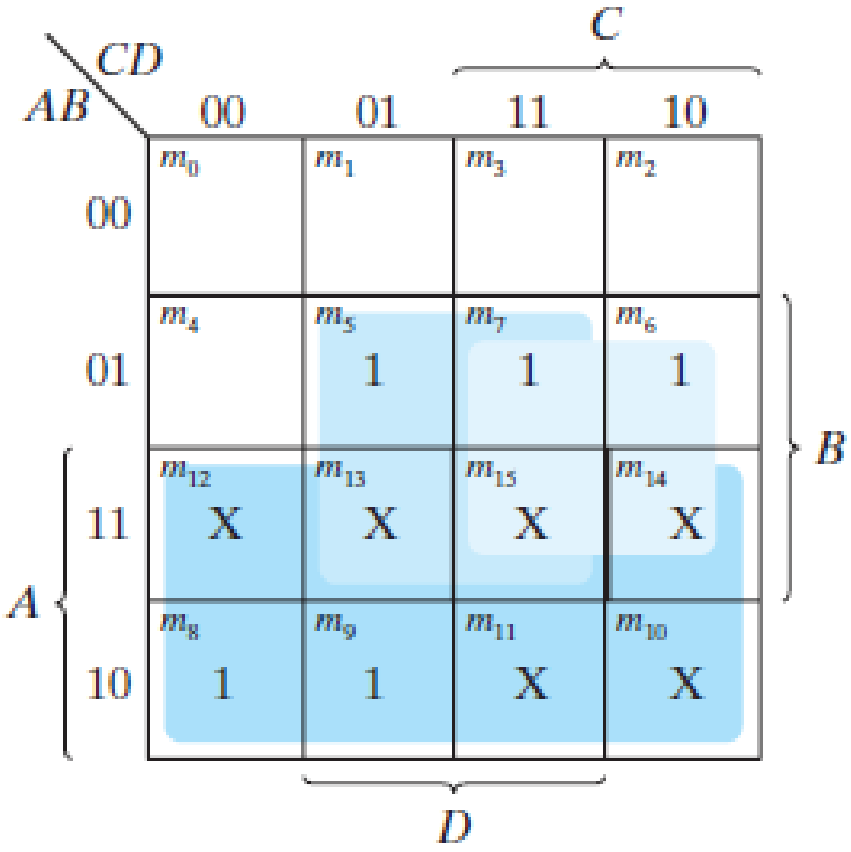


$y = CD + C'D'$

Maps for Code Converter (3/3)



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



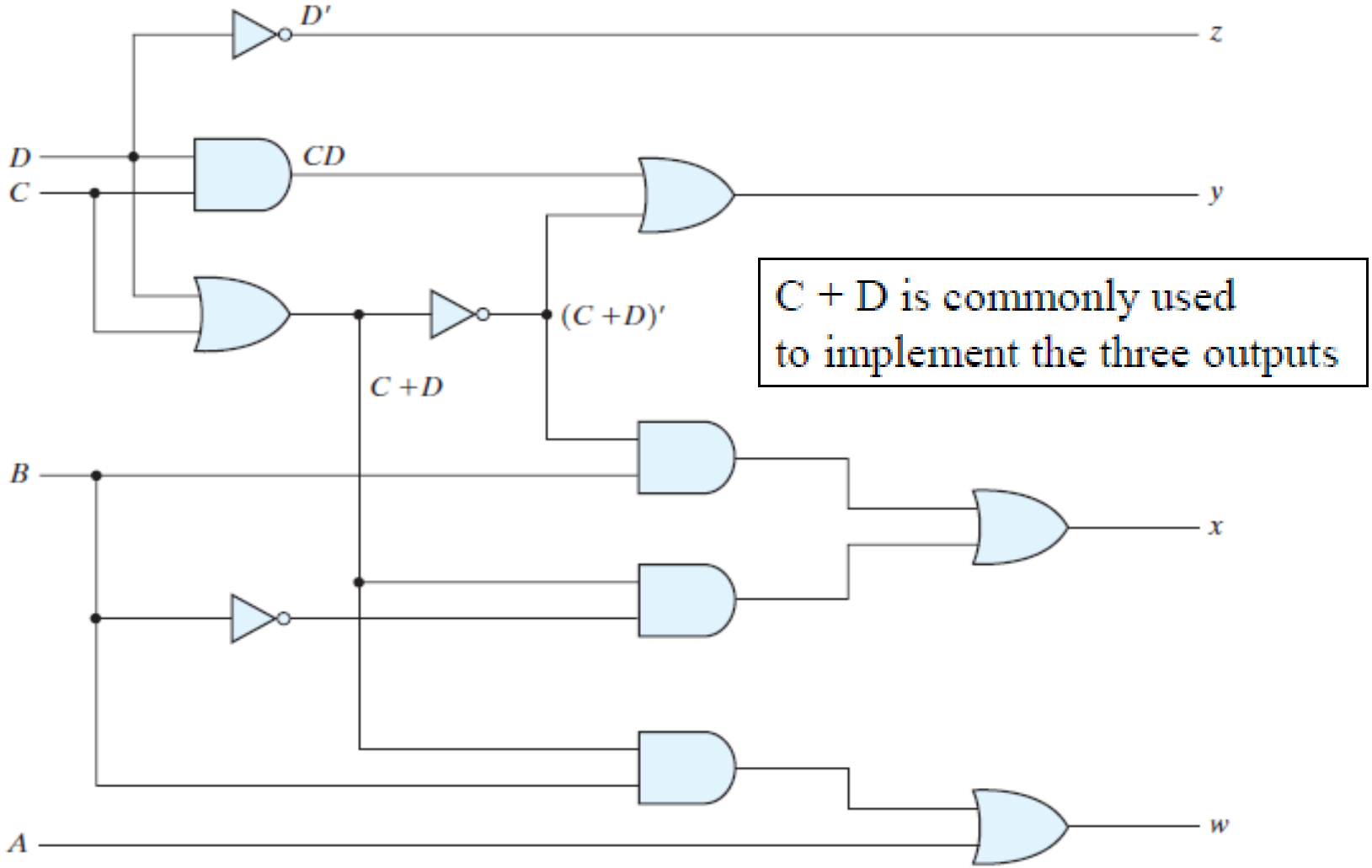
$w = A + BC + BD$

Logic Diagram for the Converter

- There are various possibilities for a logic diagram that implements a circuit
- A two-level logic diagram may be obtained directly from the Boolean expressions derived by the maps
- The expressions may be manipulated algebraically to use common gates for two or more outputs
  - Reduce the number of gates used

$$z = D'$$
$$y = CD + C'D' = CD + (C + D)'$$
$$x = B'C + B'D + BC'D' = B'(C + D) + BC'D'$$
$$= B'(C + D) + B(C + D)'$$
$$w = A + BC + BD = A + B(C + D)$$

Logic Diagram for the Converter



C + D is commonly used to implement the three outputs

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

Home work

- Convert binary to grey code

Input Binary number				Output gray code			
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Binary Adder-Subtractor

- The most basic arithmetic operation is the addition of two binary digits.
- This simple addition consists of four possible elementary operations:

  0 + 0 = 0

  0 + 1 = 1

  1 + 0 = 1

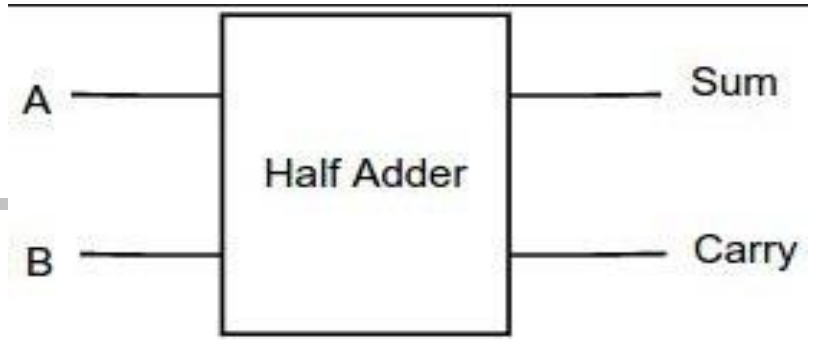
  1 + 1 = 10

- The first three operations produce a sum of one digit, but when both augend and addend bits are equal to 1, the binary sum consists of two digits.
- The higher significant bit of this result is called a carry.

Binary Adder-Subtractor

- When the augend and addend numbers contain more significant digits, the carry obtained from the addition of two bits is added to the next higher order pair of significant bits.
- A combinational circuit that performs the addition of two bits is called a half adder.
- A adder performs the addition of three bits (two significant bits and a previous carry) is a full adder.

Half Adder



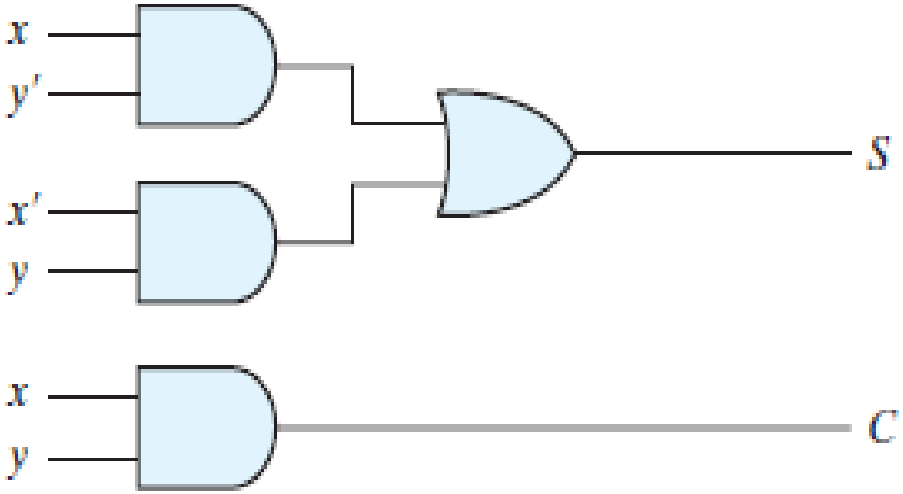
Block Diagram of half adder

- Half adder
  - Inputs: x and y
  - Outputs: S (for sum) and C (for carry)

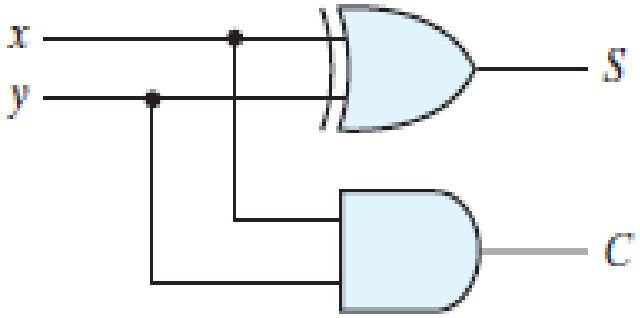
x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = x'y + xy'$$
$$C = xy$$

Implementation of a Half Adder



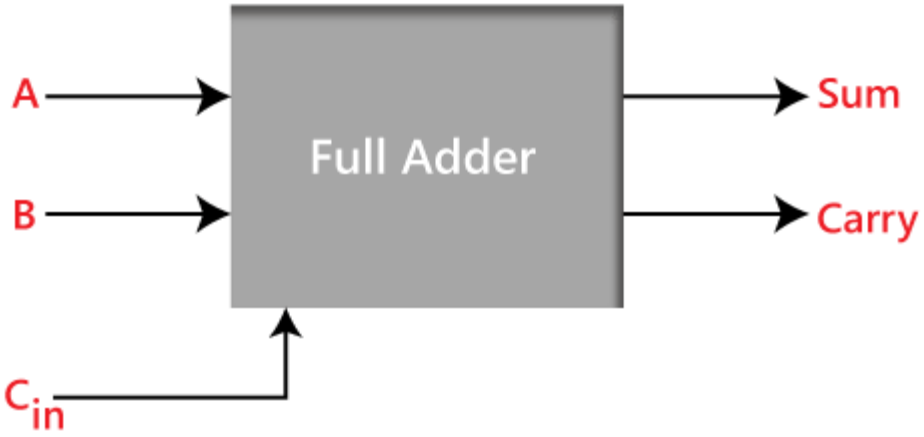
(a) $S = xy' + x'y$
$C = xy$



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

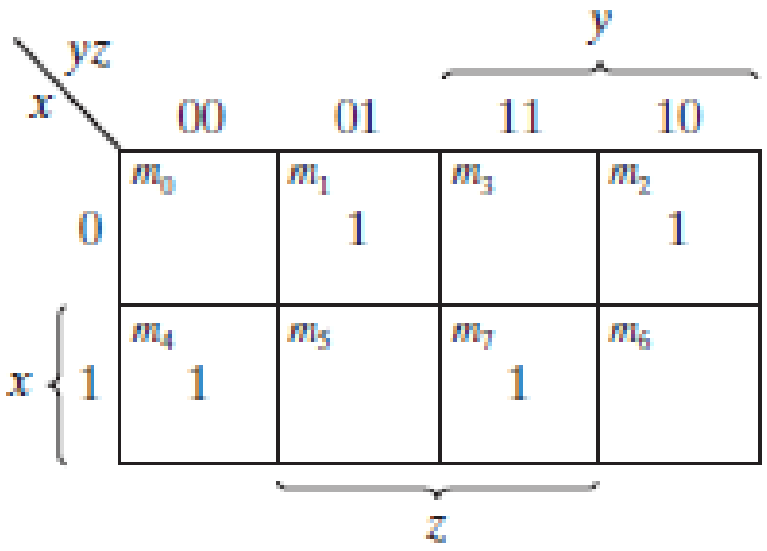
Full Adder

- One that performs the addition of three bits(two significant bits and a previous carry) is a full adder.
- 0+0=0, 0+1=1, 1+0=1, 1+1=10, 10+1=11, 11+1=100

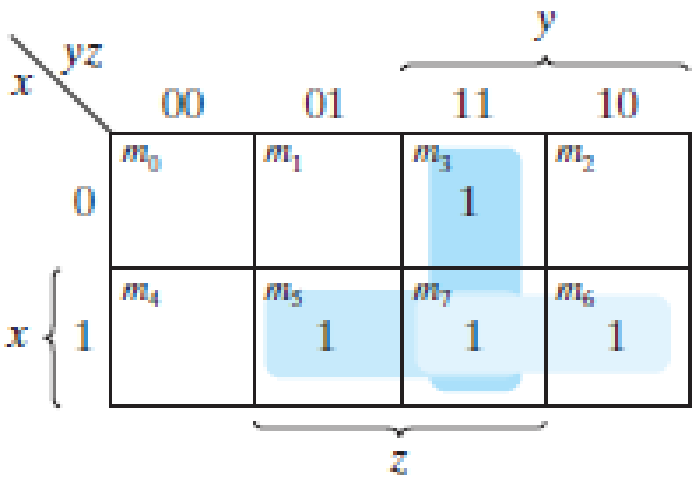


Full Adder

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) $C = xy + xz + yz$

x, y: the two significant bits to be added
z: the carry from the previous position

Implementation of a Full Adder

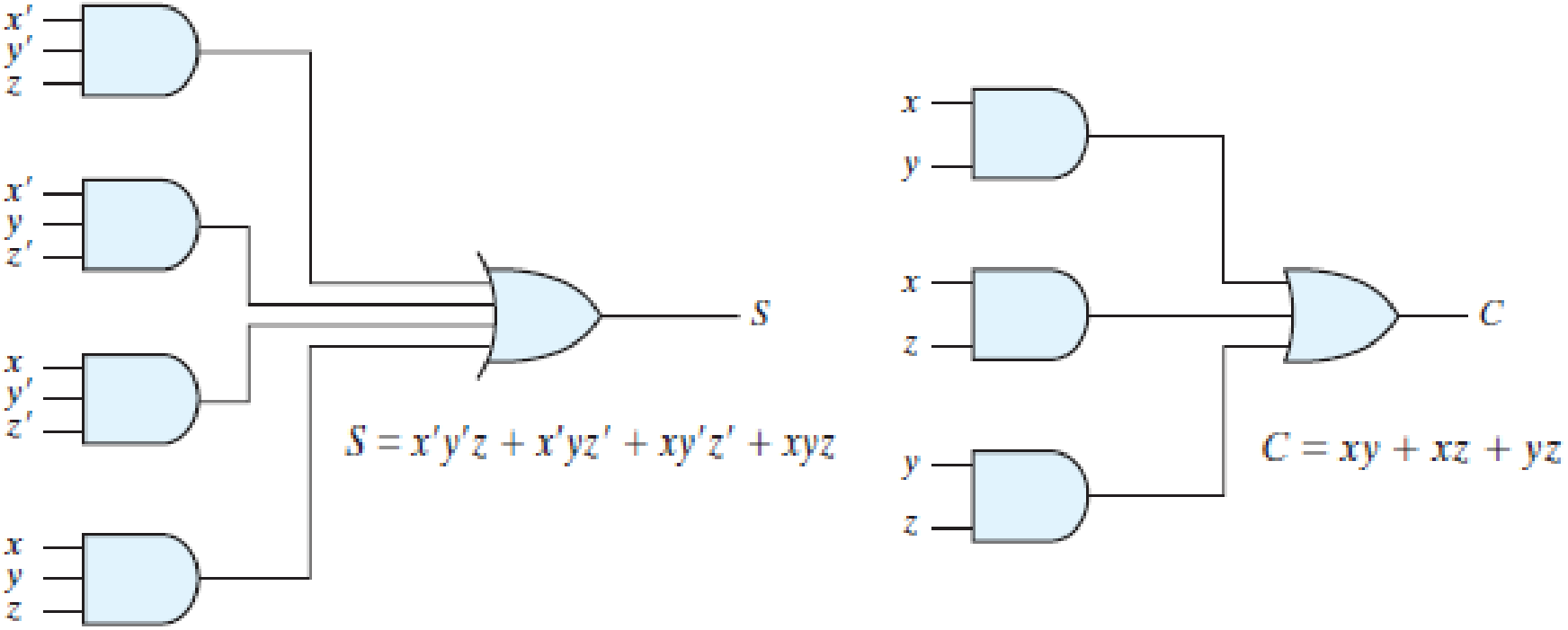


FIGURE 4.7
Implementation of full adder in sum-of-products form

Another implementation

- Full-adder can also implemented with two half adders and one OR gate (Carry Look-Ahead adder).

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

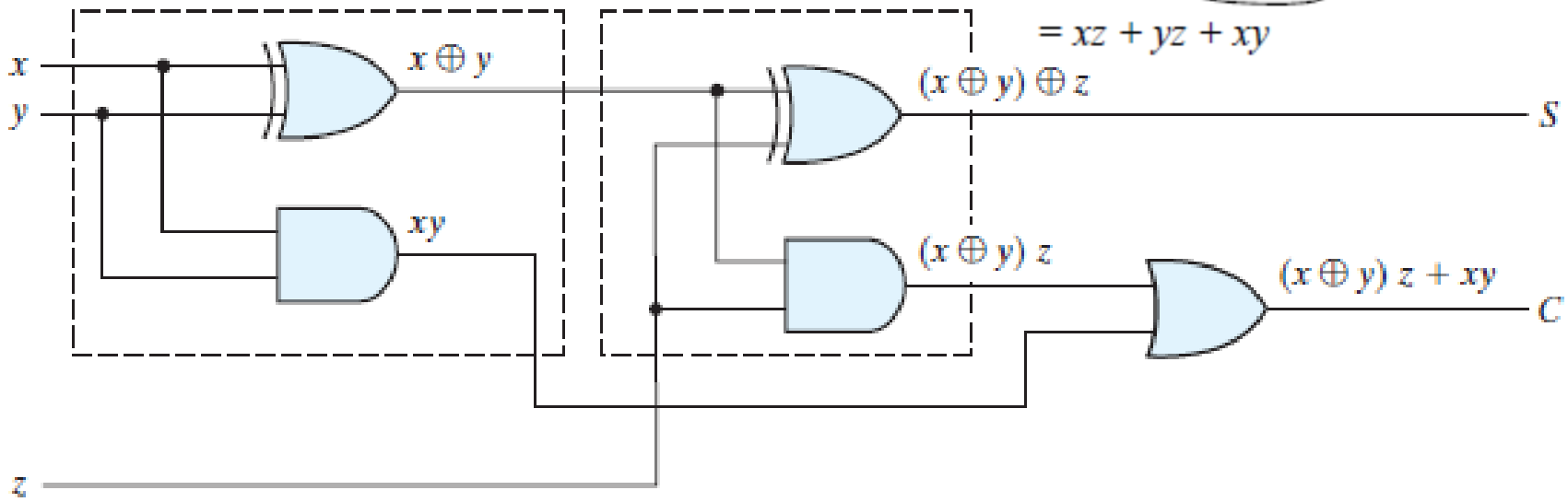


FIGURE 4.8
Implementation of full adder with two half adders and an OR gate

Binary Adder

- A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.
- It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.
- An n -bit adder requires n full adders, with each output carry connected to the input carry of the next higher order full adder.

Binary Adder

- Below figure shows the interconnection of four full-adder (FA) circuits to provide a four-bit binary ripple carry adder.

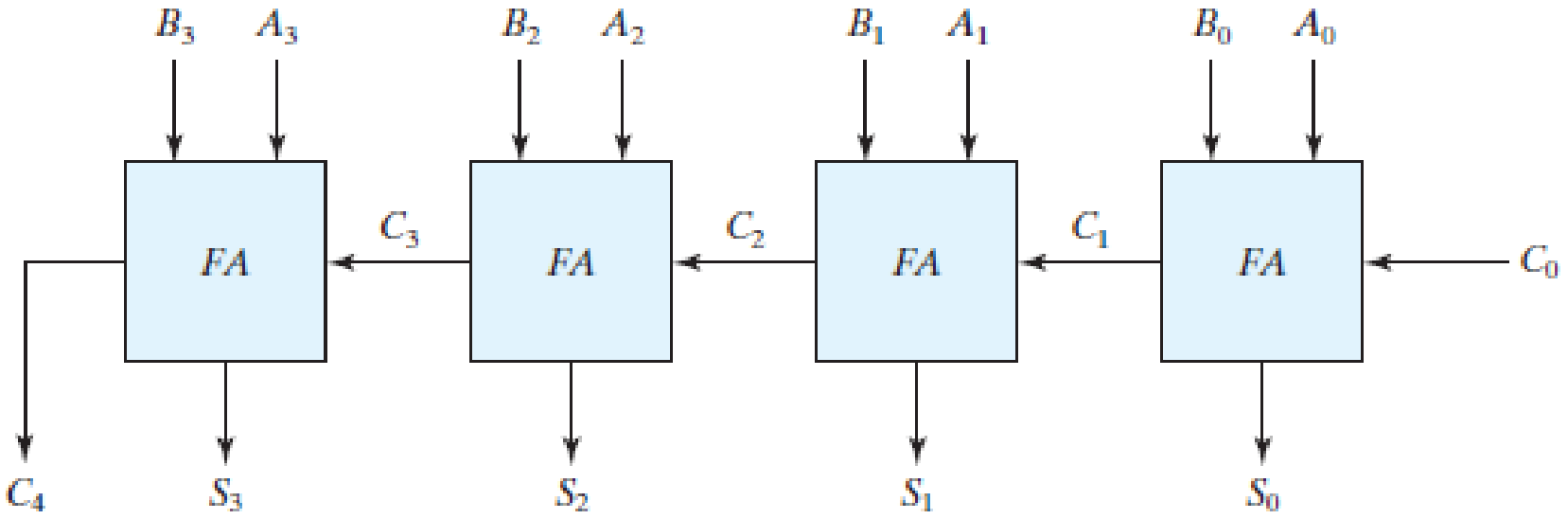


FIGURE 4.9
Four-bit adder

4-bit Adder Example

- Consider two binary number A = 1011 and B = 0011

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-bit Adder Example

- To demonstrate with a specific example, consider the two binary numbers A=1011 and B=0011. Their sum S=1110 is formed with the four-bit adder as follows:

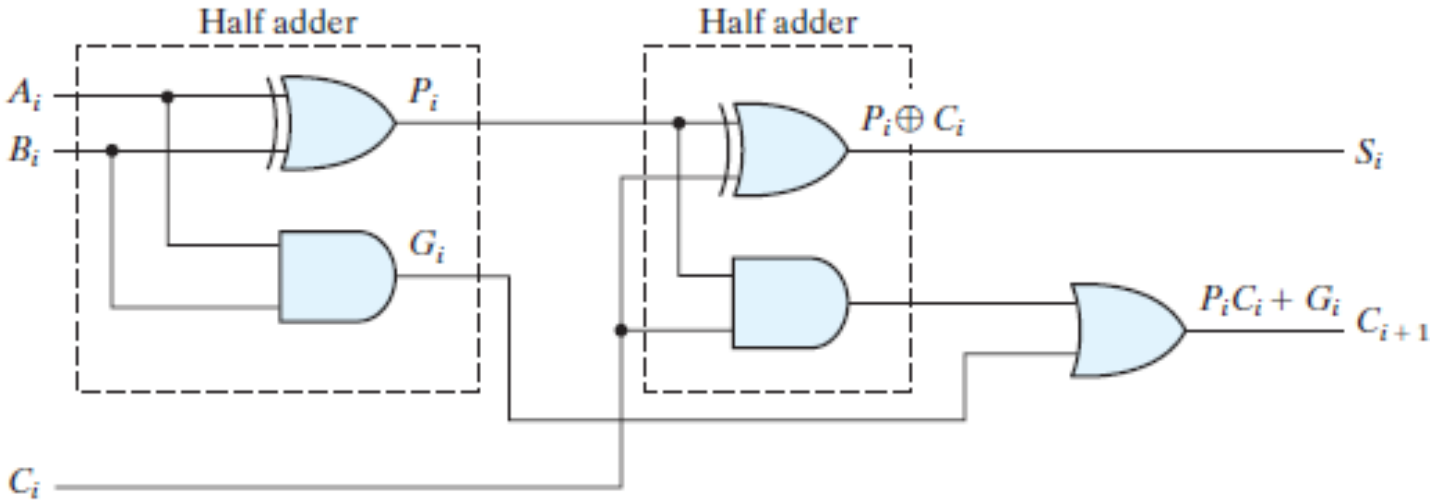
Subscript i :	3	2	1	0	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

Carry Propagation

- 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 adders.
- 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.

Full Adder with P and G

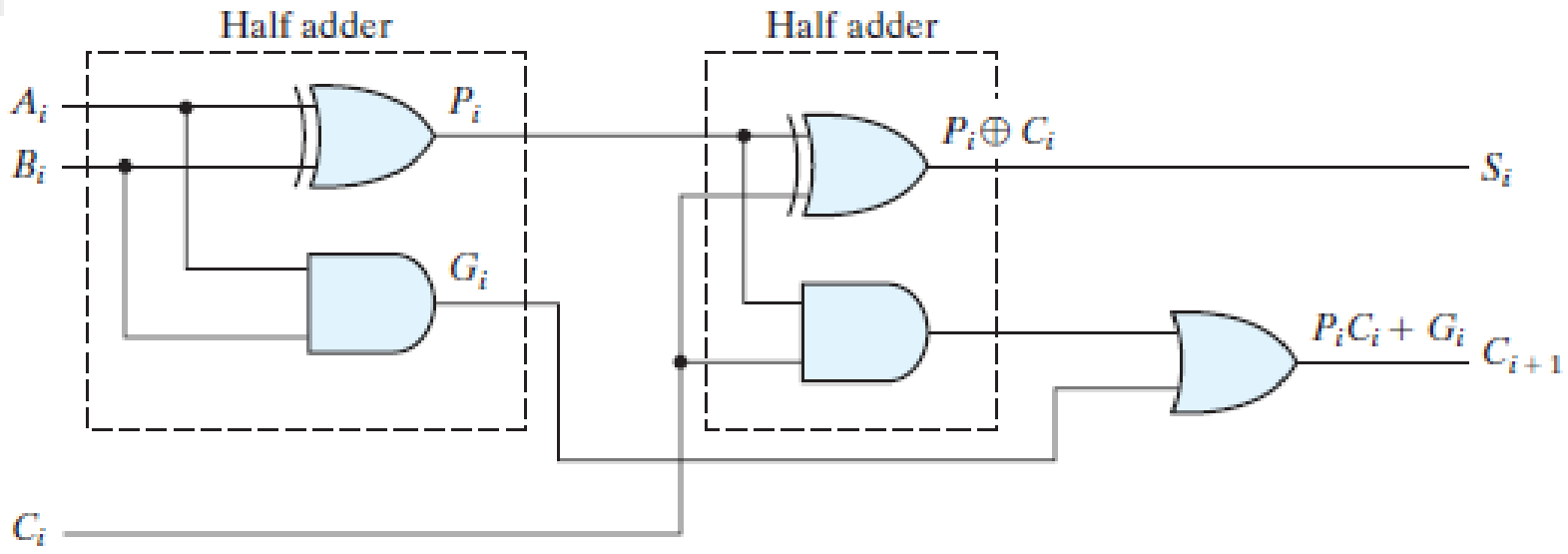
- The full adder can be redrawn with two internal signals P (propagation) and G (generation)
- The signal from input carry C_i to output carry C_{i+1} propagates through an AND and a OR gate (2 gate levels)
  - For n-bit adder, there are 2n gate levels for the carry to propagate from input to output



Carry Propagation

- The ***carry propagation time*** is a limiting factor on the speed with which two numbers are added
- All other arithmetic operations are implemented by successive additions
  - The time consumed during the addition is very critical
- To reduce the carry propagation delay
  - Employ faster gates with reduced delays
  - Increase the equipment complexity
- Several techniques for reducing the carry propagation time in a parallel adder
  - The most widely used technique employs the principle of ***carry lookahead***

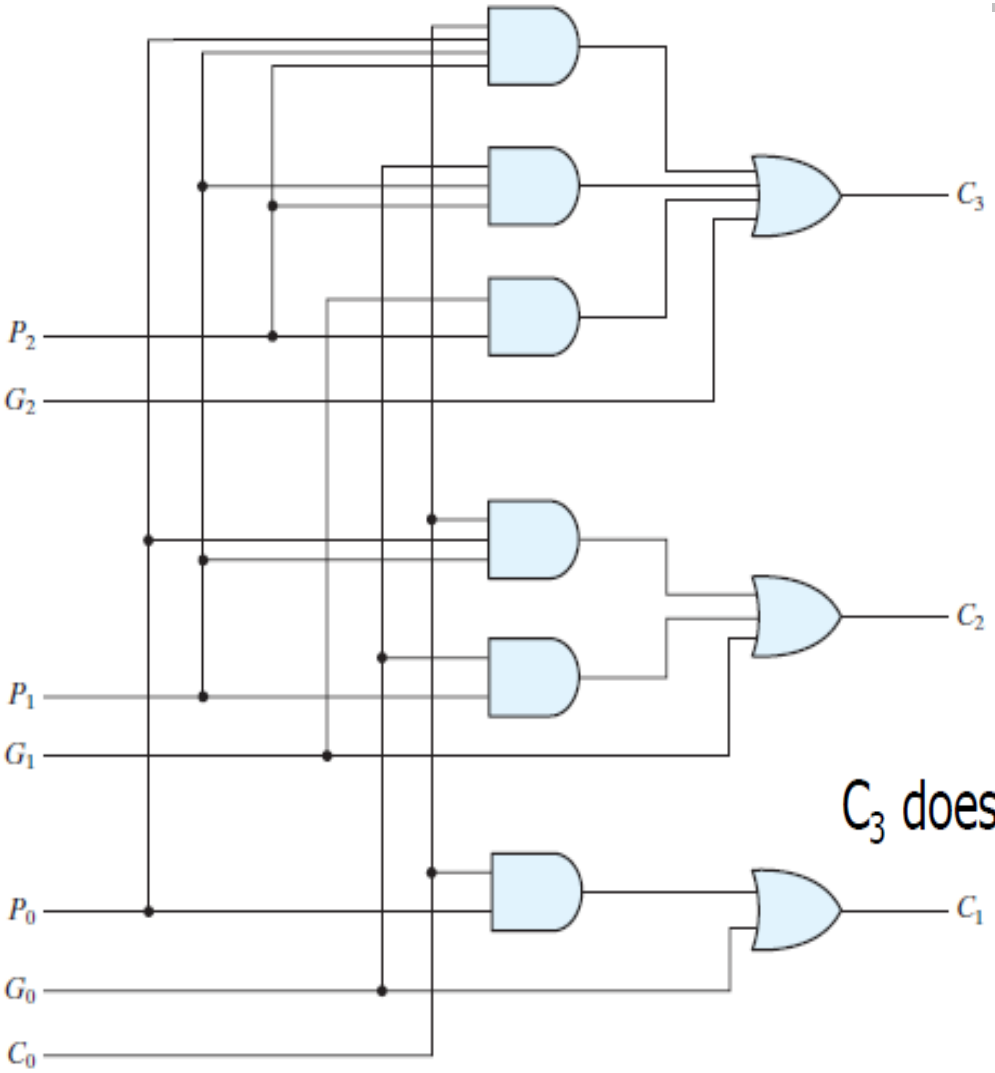
Carry Propagation & Generation



carry propagate : $P_i = A_i \oplus B_i$ $\quad S_i = P_i \oplus C_i$

carry generate : $G_i = A_iB_i$ $\quad C_{i+1} = G_i + P_iC_i$

Carry Lookahead Generator



C_0 = input carry

$C_1 = G_0 + P_0C_0$

$C_2 = G_1 + P_1C_1$

$= G_1 + P_1(G_0 + P_0C_0)$

$= G_1 + P_1G_0 + P_1P_0C_0$

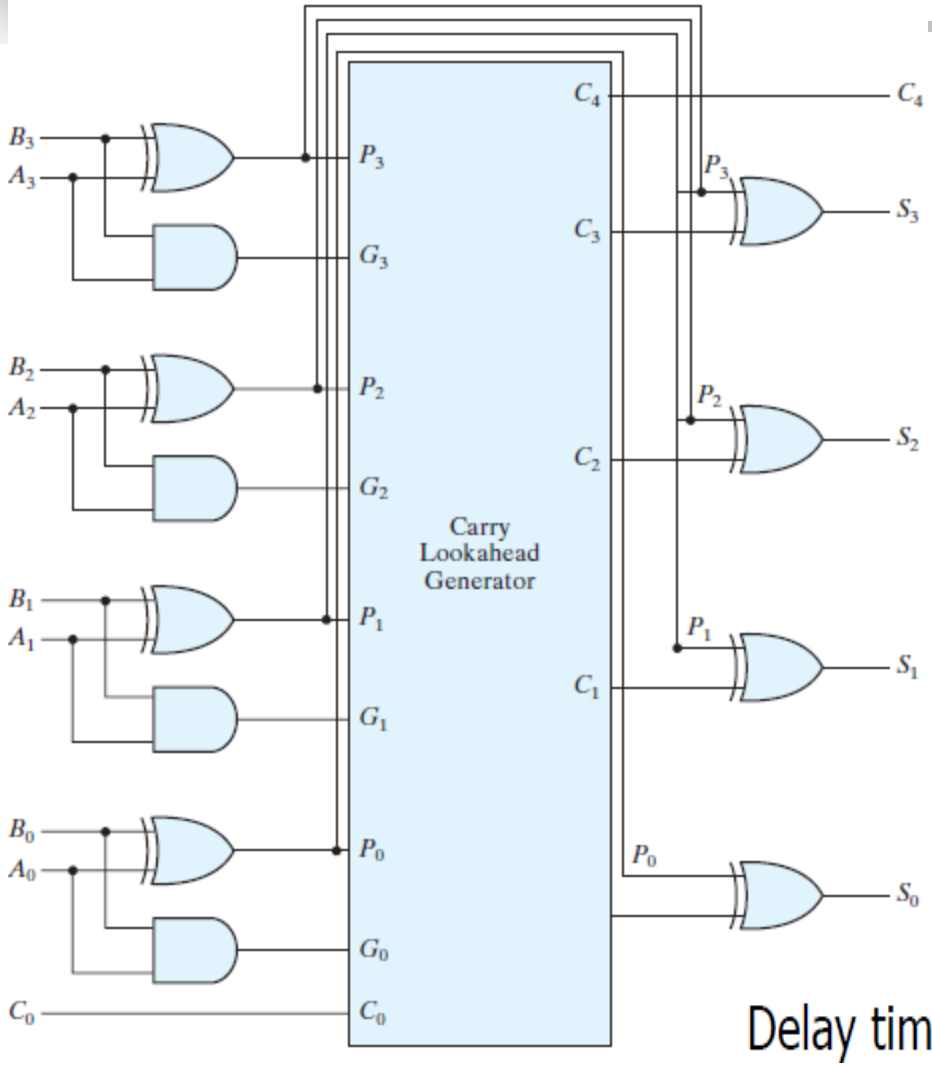
$C_3 = G_2 + P_2C_2$

$= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

C_3 does not have to wait for C_2 and C_1 to propagate

C_3 is propagated at the same time as C_2 and C_1.

Carry Lookahead Adder



- All output carries are generated after a delay through two levels of gates
- Output S1 to S3 can have equal propagation delay times

Delay time of n-bit CLAA = XOR + (AND + OR) + XOR

Binary Subtractor

- A – B can be done by taking the 2's complement of B and adding it to A ---> A – B = A + (-B)
  - 2'complement can be obtained by taking the 1'complement and adding on to the least significant pair of bits
  - A - B = A + (B′ + 1)
- The circuit for subtraction A – B consists of an adder with inverter placed between each data input B and the corresponding input of the full adder
- The input carry C_0 must be equal to 1

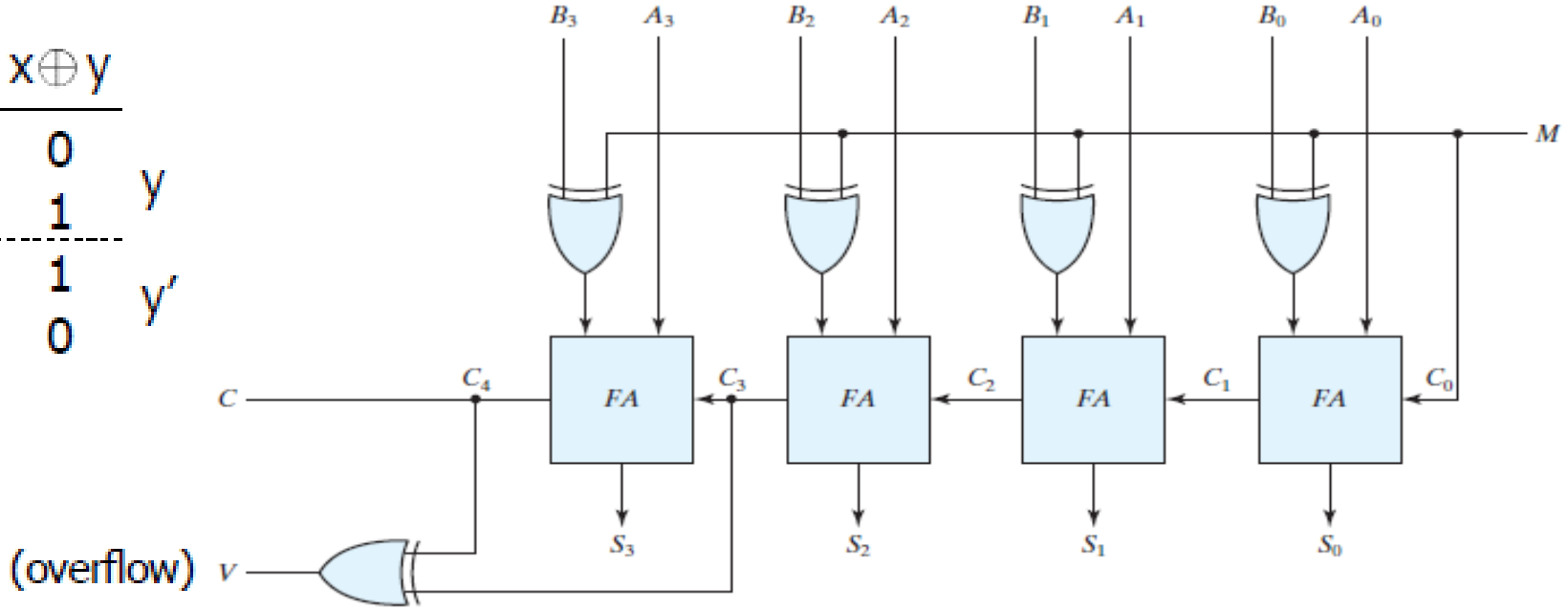
Binary Subtractor

- It is worth noting that binary numbers in the signed-complement system are added and subtracted by the same basic addition and subtraction rules as are unsigned numbers.
- Therefore, computers need only one common hardware circuit to handle both types of arithmetic.
- The user or programmer must interpret the results of such addition or subtraction differently, depending on whether it is assumed that the numbers are signed or unsigned.

4-bit Adder-Subtractor

- M=0 (Adder)
  - Input of FA is A and B ($B \oplus 0 = B$), and C_0 is 0
- M=1 (Subtractor)
  - Input of FA is A and B′ ($B \oplus 1 = B'$), and C_0 is 1

x	y	x⊕y	
0	0	0	y
0	1	1	
1	0	1	y′
1	1	0	



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 because the number of bits that hold the number is finite and a result that contains n+1 bits cannot be accommodated by an n-bit word.
- For this reason, many computers detect the occurrence of an overflow, and when it occurs, a corresponding flip-flop is set that can then be checked by the user.

Overflow

- The detection of an overflow after the addition of two binary numbers depends on whether the numbers are considered to be signed or unsigned.
- When two unsigned numbers are added, an overflow is detected from the end carry out of the most significant position.
- When two signed numbers are added, the sign bit is treated as part of the number and the end carry does not indicate an overflow.
  - Extra overflow detection circuits are required

Overflow

- An overflow cannot occur after an addition if one number is positive and the other is negative
- An overflow may occur if the two numbers added are both positive or both negative.

Overflow Example

Carries :	0	1		Carries :	1	0	
+70		0	1000110	-70		1	0111010
+80		0	1010000	-80		1	0110000
+150		1	0010110 (-106)	-150		0	1101010 (+106)
(010010110)				(101101010)			

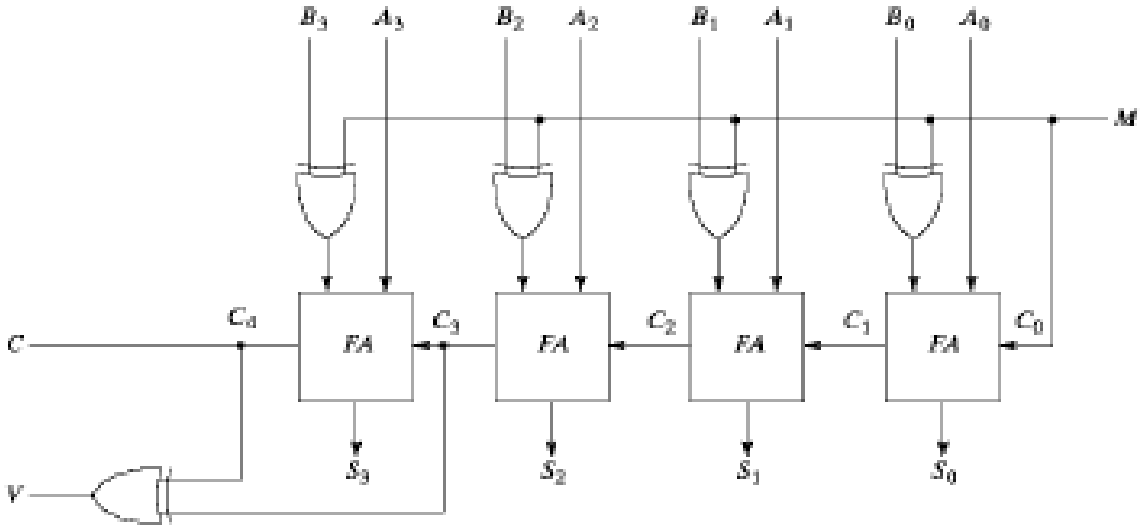
Overflow Detection

- An overflow condition can be detected by observing the carry into the sign bit position and the carry out of the sign bit position
  - If these two carries are not equal, and overflow has occurred
  - If the output V is equal to 1, an overflow is detected

Adder-Subtractor Circuit

- Unsigned
  - C bit detects a ***carry*** after addition or a ***borrow*** after subtraction
- Signed
  - V bit detects an overflow

    0: no overflow; 1: overflow

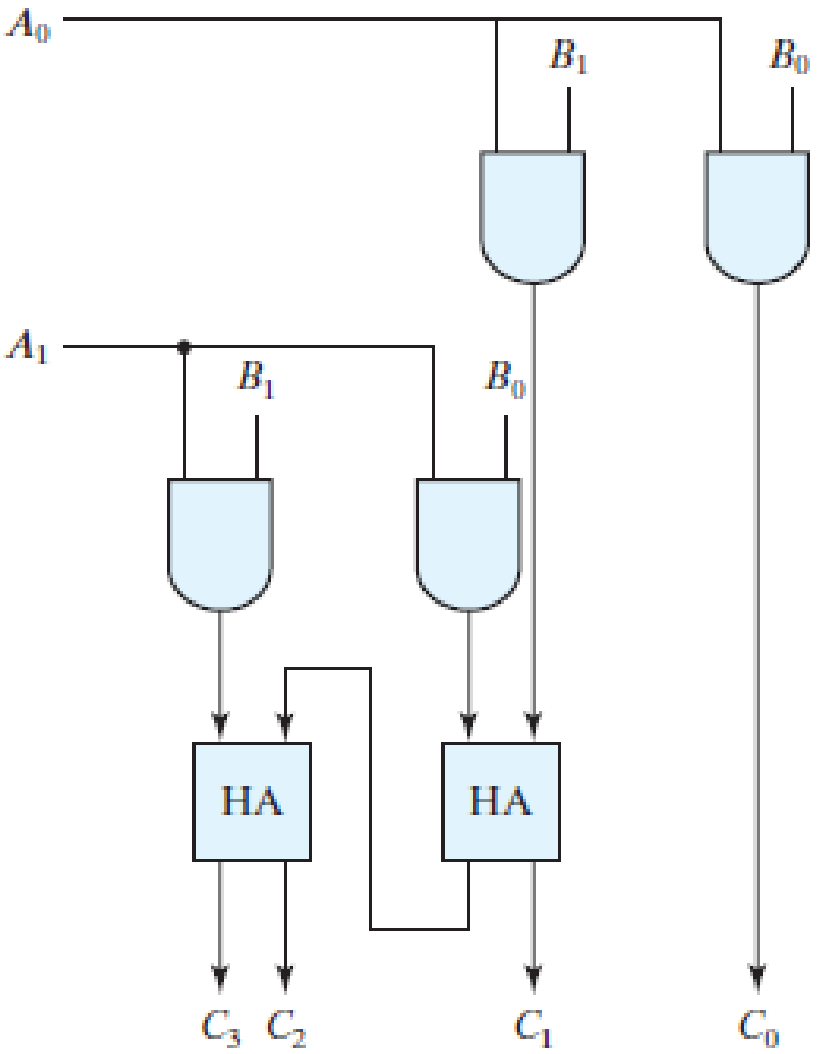


Binary Multiplier

- Multiplication of binary numbers is performed in the same way as multiplication of decimal numbers.
- The multiplicand is multiplied by each bit of the multiplier, starting from the least significant bit.
- Each such multiplication forms a partial product. Successive partial products are shifted one position to the left.
- The final product is obtained from the sum of the partial products.

Two-bit by two-bit binary multiplier

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



Example:-

1 0

1 1 X

---

1 0

1 0

---

1 1 0

Binary Multiplier

- Usually, there are more bits in the partial products and it is necessary to use full adders to produce the sum of the partial products.
- Note:- The least significant bit of the product does not have to go through an adder, since it is formed by the output of the first AND gate.

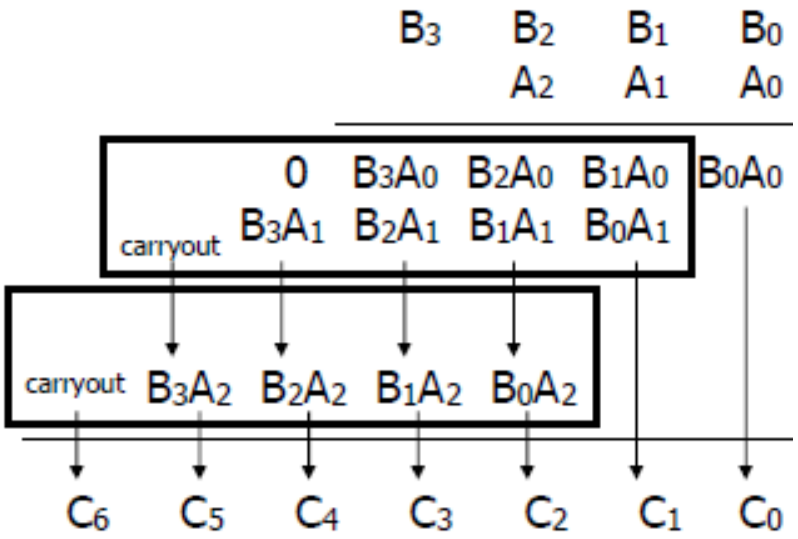
J-bit by K-bit binary multiplier

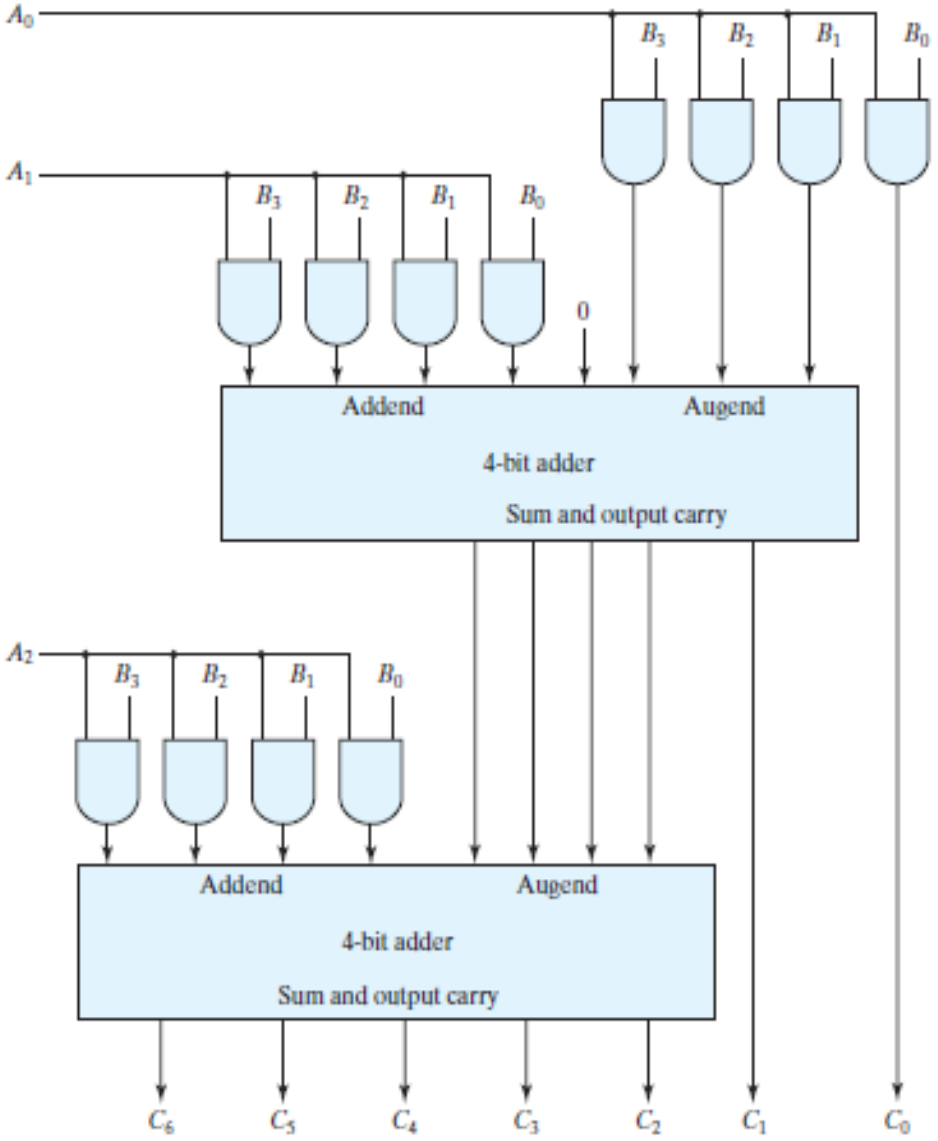
- A combinational circuit binary multiplier with more bits can be constructed in a similar fashion.
- A bit of the multiplier is ANDed with each bit of the multiplicand in as many levels as there are bits in the multiplier.
- The binary output in each level of AND gates is added with the partial product of the previous level to form a new partial product.
- The last level produces the product.
- For J multiplier bits and K multiplicand bits, we need (J*K) AND gates and (J – 1) K-bit adders to produce a product of (J + K) bits.

4-bit by 3-bit binary multiplier

- Second example - Consider a multiplier circuit that multiplies a binary number represented by four bits by a number represented by three bits.
- Let the multiplicand be represented by B3 B2 B1 B0 and the multiplier by A2 A1 A0.
- Since K=4 and J=3, we need 12 AND gates and two 4-bit adders to produce a product of seven bits.
- The logic diagram of the multiplier is shown in below figure.

4-bit by 3-bit binary multiplier





Magnitude Comparator

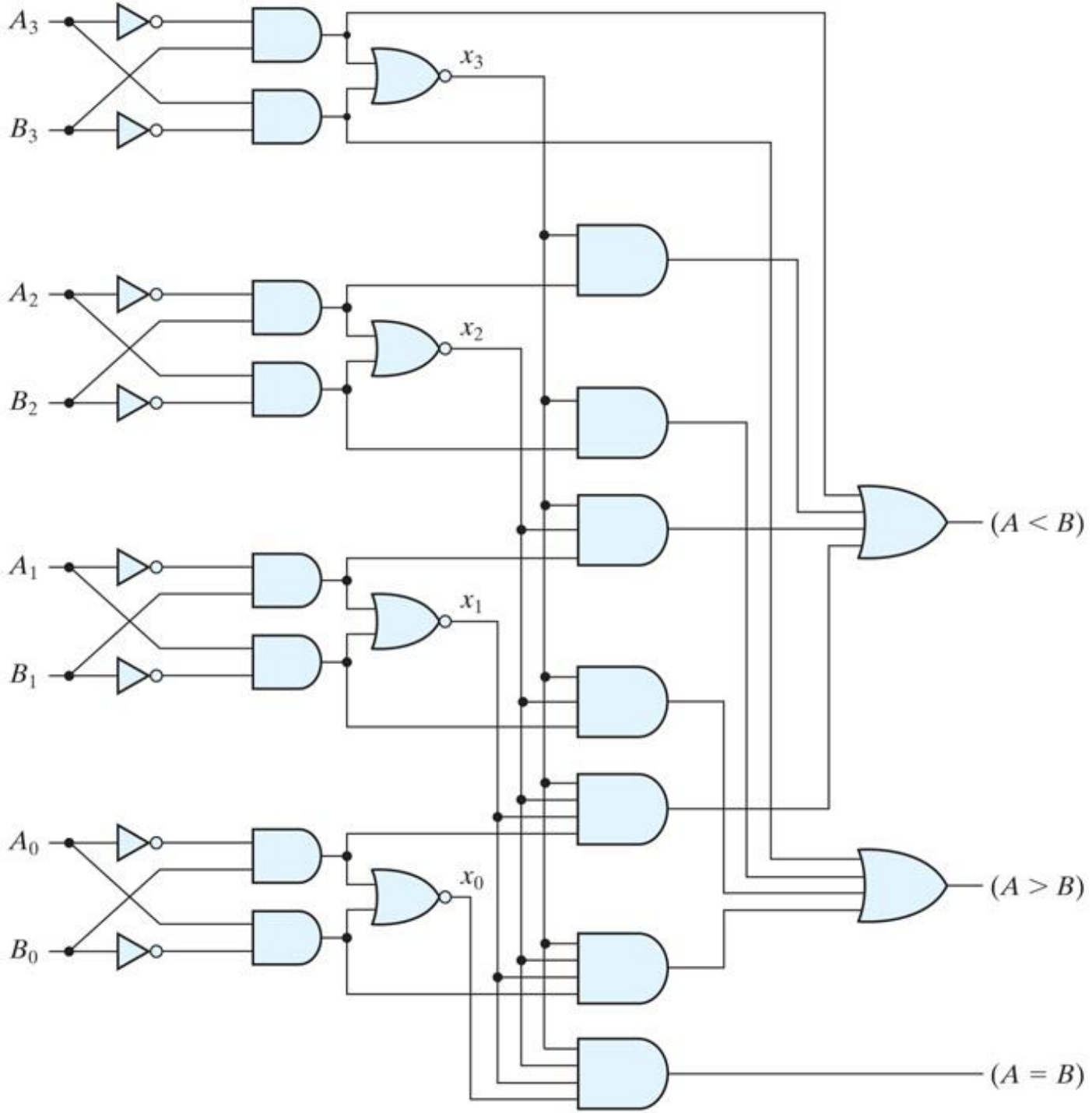
- The comparison of two numbers is an operation that determines whether one number is greater than, less than, or equal to the other number.
- A magnitude comparator is a combinational circuit that compares two numbers A and B and determines their relative magnitudes.
- The outcome of the comparison is specified by three binary variables that indicate whether $A > B$, $A = B$, or $A < B$.

Magnitude Comparator

- The algorithm is a direct application of the procedure a person uses to compare the relative magnitudes of two numbers.
- Consider two numbers, A and B, with four digits each. Write the coefficients of the numbers in descending order of significance:
- A=A3A2A1A0 B=B3B2B1B0
- Each subscripted letter represents one of the digits in the number. The two numbers are equal if all pairs of significant digits are equal: A3=B3, A2=B2, A1=B1 , and A0=B0 . When the numbers are binary, the digits are either 1 or 0, and the equality of each pair of bits can be expressed logically with an exclusive-NOR function as

  $x_i = A_iB_i + A_i'B_i'$ for i = 0, 1, 2, 3

- where xi = 1 only if the pair of bits in position i are equal (i.e., if both are 1 or both are 0).


A_3
B_3
x_3
A_2
B_2
x_2
A_1
B_1
x_1
A_0
B_0
x_0
$(A < B)$
$(A > B)$
$(A = B)$

Decoders

- A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines.
- If the n-bit coded information has unused combinations, the decoder may have fewer than 2^n outputs.

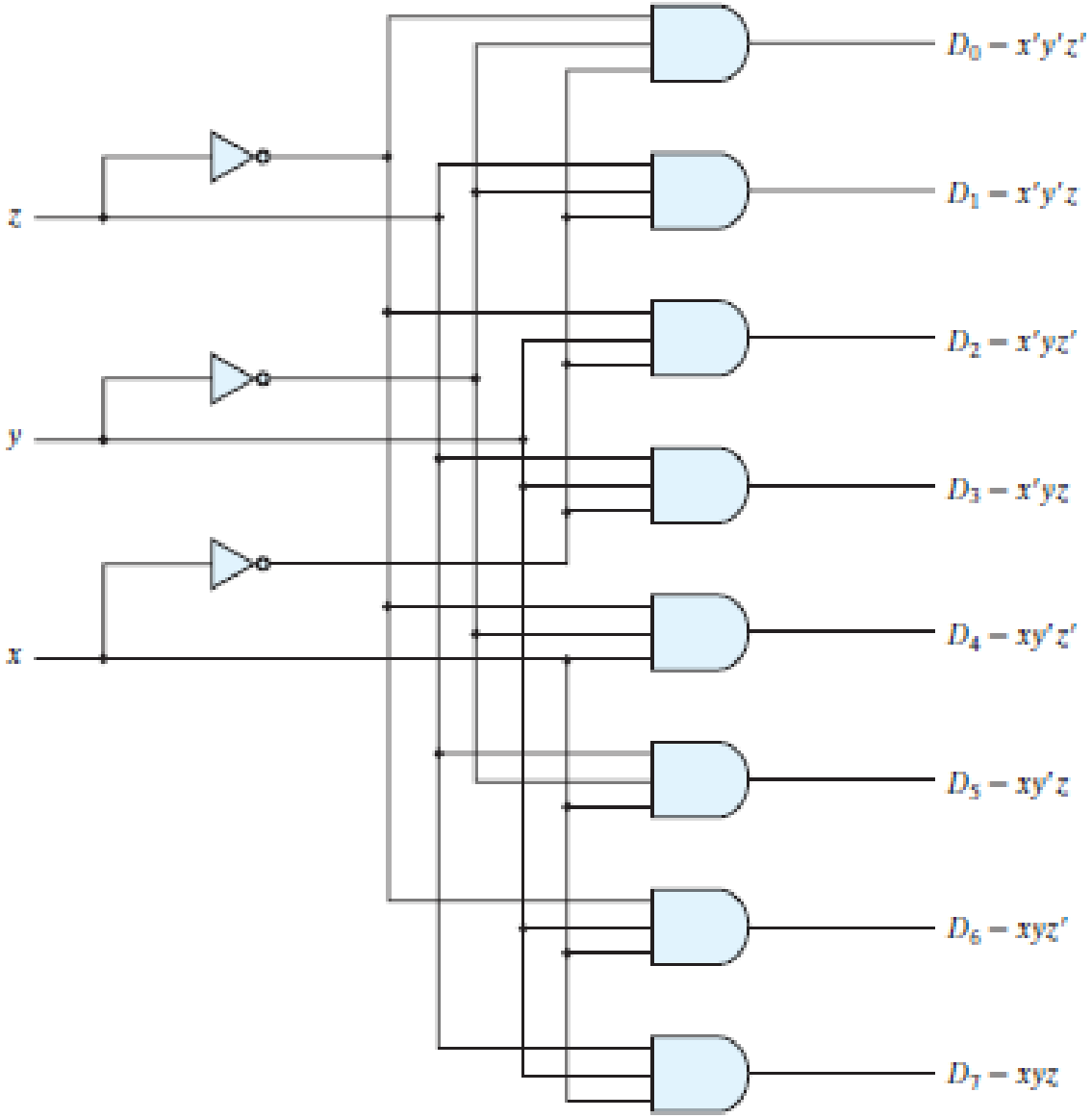
Decoders

- The decoders presented here are called n-to m-line decoders, where m<=2^n.
- Their purpose is to generate the 2^n (or fewer) minterms of n input variables.
- Each combination of inputs will assert a unique output.
- The name decoder is also used in conjunction with other code converters, such as a BCD-to-seven-segment decoder.

3-to-8-Line Decoder

- As an example, consider the three-to-eight-line decoder circuit of below Fig.
- The three inputs are decoded into eight outputs, each representing one of the minterms of the three input variables.
- The operation of the decoder may be clarified by the truth table listed in below Table.
- For each possible input combination, there are seven outputs that are equal to 0 and only one that is equal to 1.
- The output whose value is equal to 1 represents the minterm equivalent of the binary number currently available in the input lines.

3-to-8-Line Decoder



3-to-8-Line Decoder

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

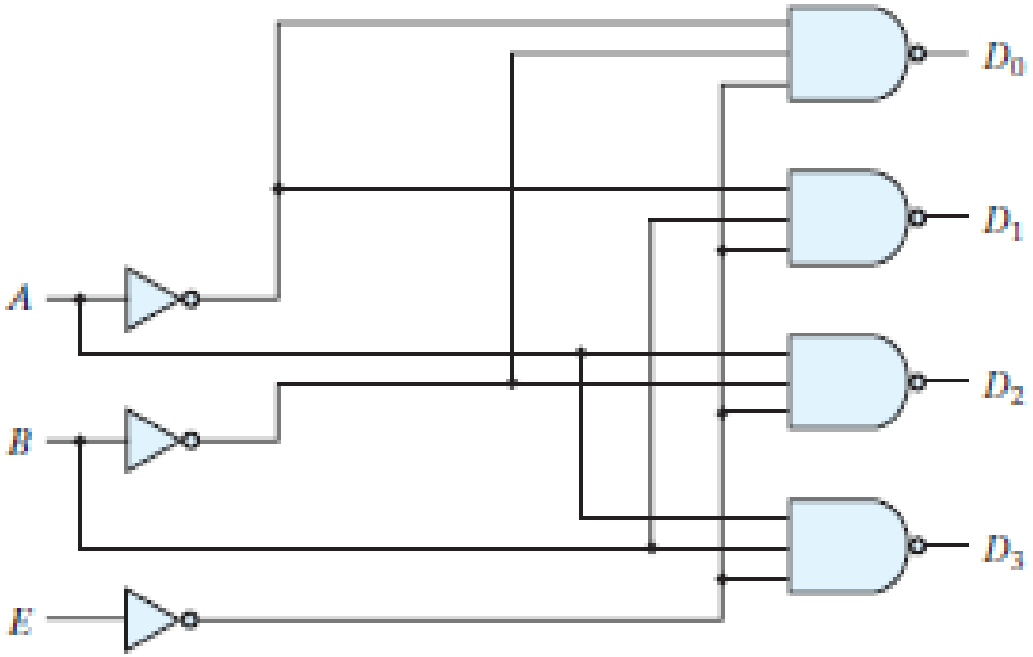
Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

2-to-4-line decoder with enable input

- Some decoders are constructed with NAND gates. Since a NAND gate produces the AND operation with an inverted output, it becomes more economical to generate the decoder minterms in their complemented form.

- An ***enable*** input can be added to control the operation
  - E=1: disabled
  - None of the outputs are equal to 0

2-to-4-line decoder with enable input



(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

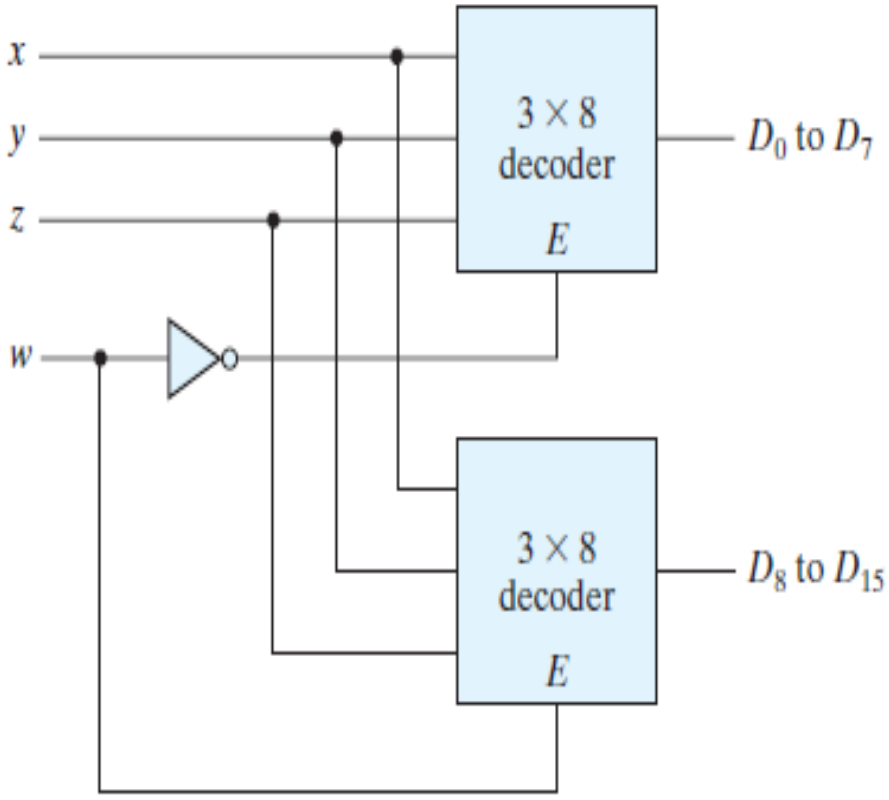
As indicated by the truth table , only one output can be equal to 0 at any given time, all other outputs are equal to 1

Demultiplexer

- A decoder with enable input can function as a demultiplexer — a circuit that receives information from a single line and directs it to one of 2^n possible output lines.
- The selection of a specific output is controlled by the bit combination of n selection lines.
- A decoder with an enable input is referred to as a decoder/demultiplexer.
- Below Figure shows two 3-to-8-line decoders with enable inputs connected to form a 4-to-16-line decoder.

Construct Larger Decoders

- Decoders with enable inputs can be connected together to form a larger decoder
- The enable input is used as the most significant bit of the selection signal
  - w=0: the top decoder is enabled
  - w=1: the bottom one is enabled
- In general, enable inputs are a convenient feature for standard components to expand their numbers of inputs and outputs

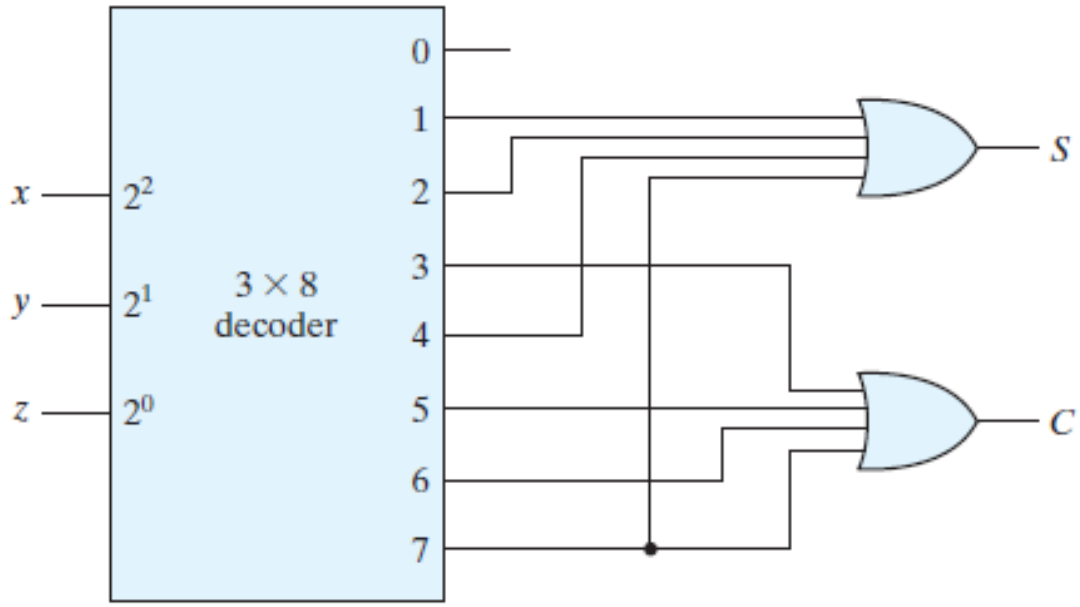


4 × 16 decoder constructed with two 3 × 8 decoders

Combinational Logic Implementation

- A decoder provides the 2^n minterms of n input variables
  - Can be used to form any combinational circuits with extra OR gates (sum of minterms)
- A function having a list of k minterms can be expressed in its complemented form F' with $2^n - k$ minterms
  - If $k > 2^n/2$, F' will have fewer minterms (fewer OR gates)
  - NOR gates are used instead for implementing F'

Combinational Logic Implementation



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

Implementation of a full adder with a decoder

$$S(x,y,z) = \Sigma(1,2,4,7)$$
$$C(x,y,z) = \Sigma(3,5,6,7)$$

Encoder

- A circuit that performs the inverse operation of a decoder
  - Have 2^n (or fewer) input lines and n output lines
  - The output lines generate the binary code of the input positions
- Only one input can be active at any given time
- An extra output may be required to distinguish the cases that $D_0 = 1$ and all inputs are 0

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

Truth Table of an Octal-to-Binary Encoder

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

Priority Encoder

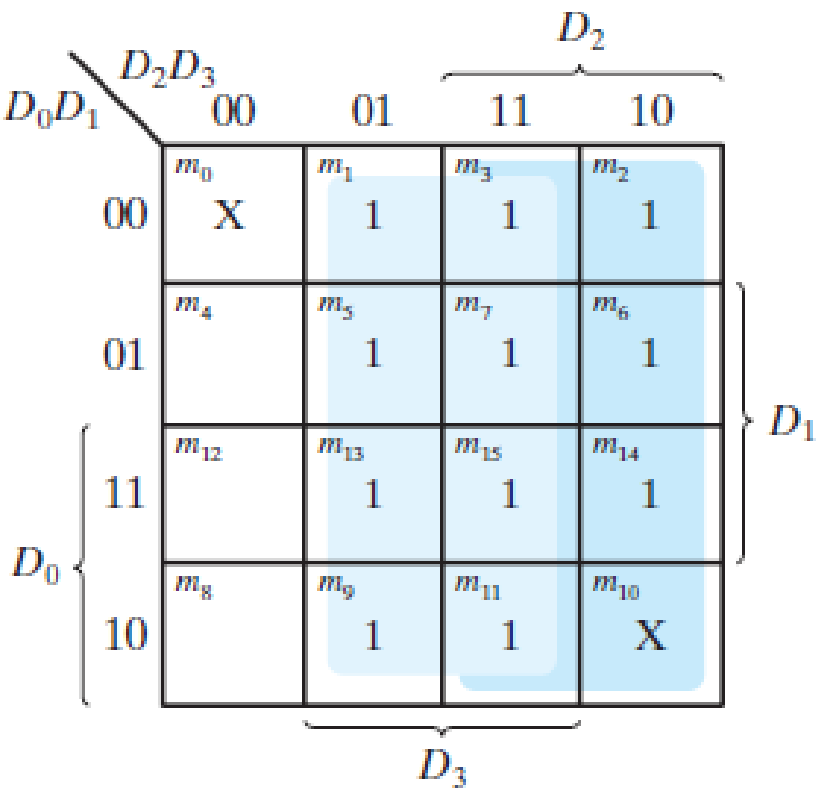
- A priority encoder is an encoder circuit that includes the priority function.
- The operation of the priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

- In the following truth table:
  - $D_3 > D_2 > D_1 > D_0$
  - The X's in output columns represent don't-care conditions
  - The X's in input columns are useful for representing a truth table in condensed form

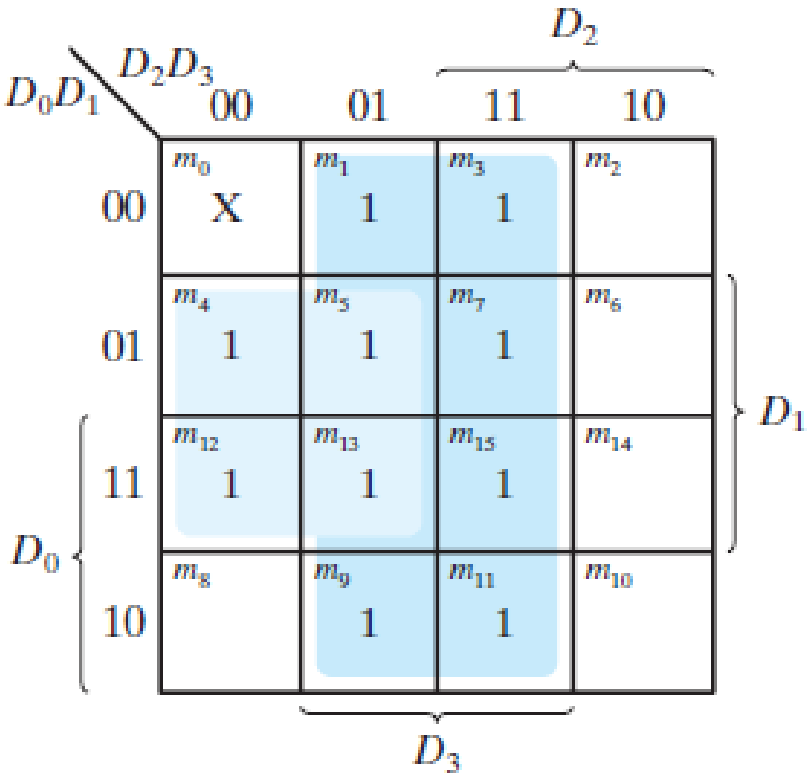
Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

V = 0 :
no valid inputs

Priority Encoder



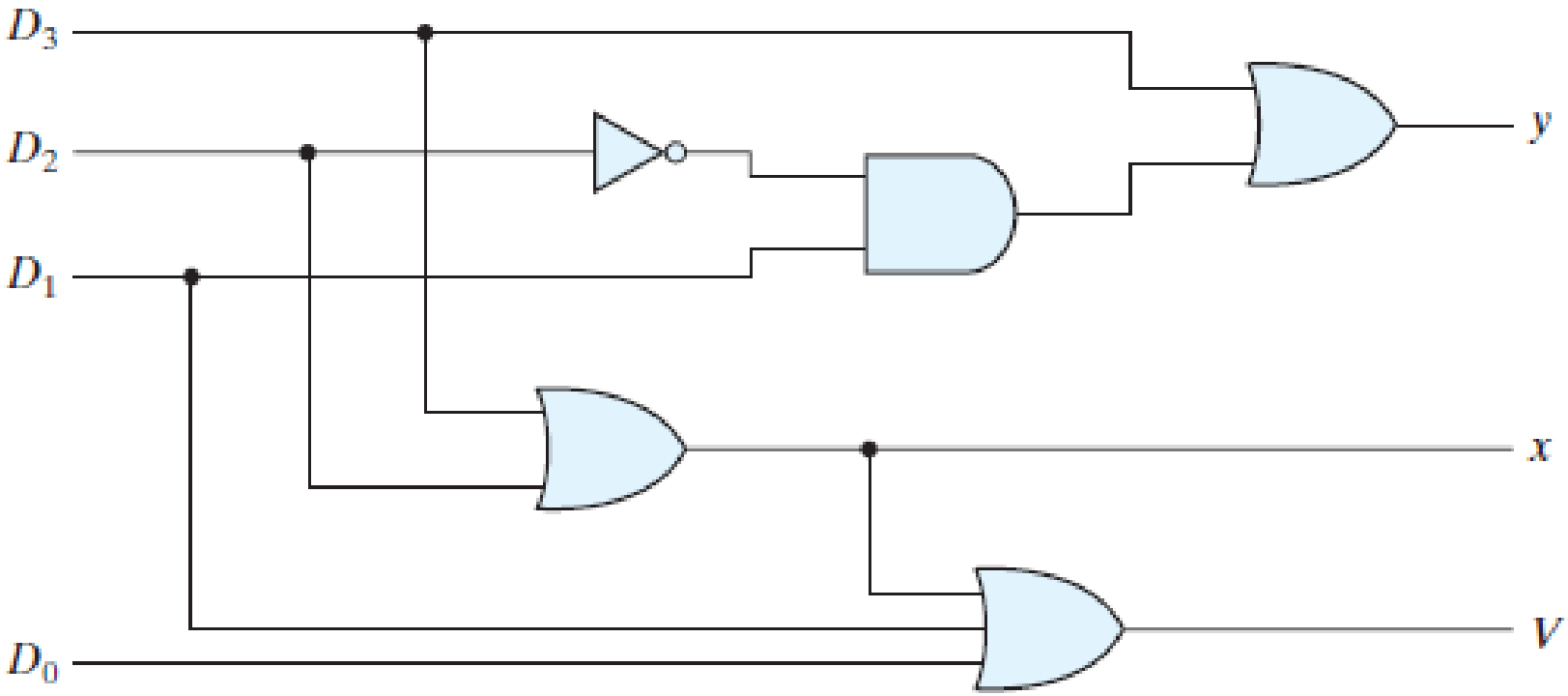
$x = D_2 + D_3$



$y = D_3 + D_1D'_2$

Priority Encoder

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



Four-input priority encoder

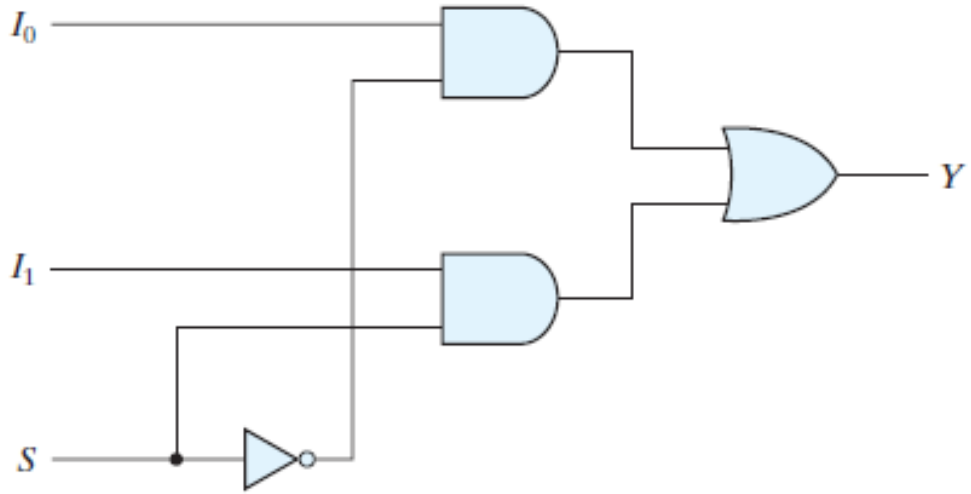
MULTIPLEXERS

- A multiplexer is 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 controlled by a set of selection lines.
- Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

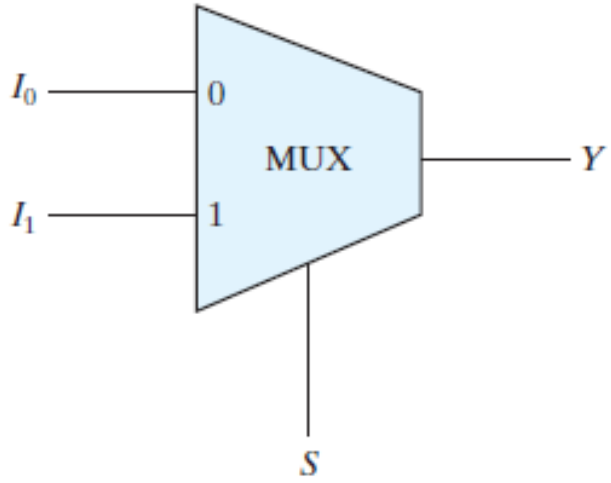
Two-to-one-line multiplexer

- A two-to-one-line multiplexer connects one of two 1-bit sources to a common destination, as shown in below Fig(a).
- The circuit has two data input lines, one output line, and one selection line S.
- When S=0, the upper AND gate is enabled and I_0 has a path to the output.
- When S=1, the lower AND gate is enabled and I_1 has a path to the output.
- The multiplexer acts like an electronic switch that selects one of two sources.
- The block diagram of a multiplexer is sometimes depicted by a wedge-shaped symbol, as shown in Fig (b).
- It suggests visually how a selected one of multiple data sources is directed into a single destination.
- The multiplexer is often labeled "MUX" in block diagrams.

Two-to-one-line multiplexer



(a) Logic diagram



(b) Block diagram

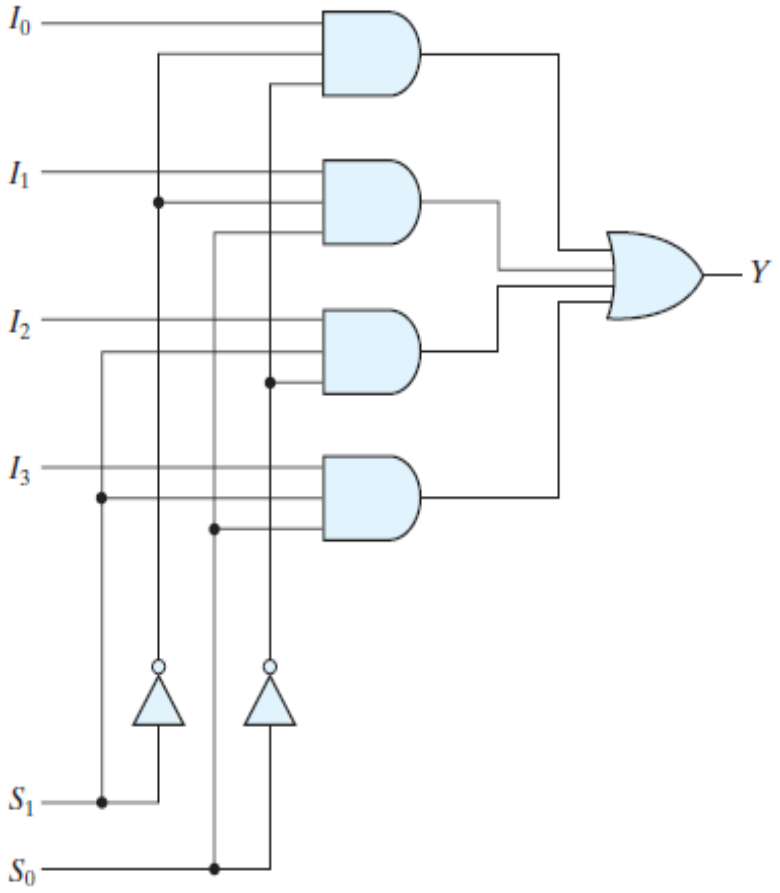
FIGURE 4.24
Two-to-one-line multiplexer

Four-to-one-line multiplexer

- A four-to-one-line multiplexer is shown in below Fig.
- Each of the four inputs, I_0 through I_3, is applied to one input of an AND gate.
- Selection lines S1 and S0 are decoded to select a particular AND gate.
- The outputs of the AND gates are applied to a single OR gate that provides the one-line output.
- The function table lists the input that is passed to the output for each combination of the binary selection values.
- A multiplexer is also called a data selector , since it selects one of many inputs and steers the binary information to the output line.

4-to-1-Line Multiplexer

- The combinations of S0 and S1 control each AND gates
- Part of the multiplexer resembles a decoder
- To construct a multiplexer:
  - Start with an n-to-2^n decoder
  - Add 2^n input lines, one to each AND gate
  - The outputs of the AND gates are applied to a single OR gate



(a) Logic diagram

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

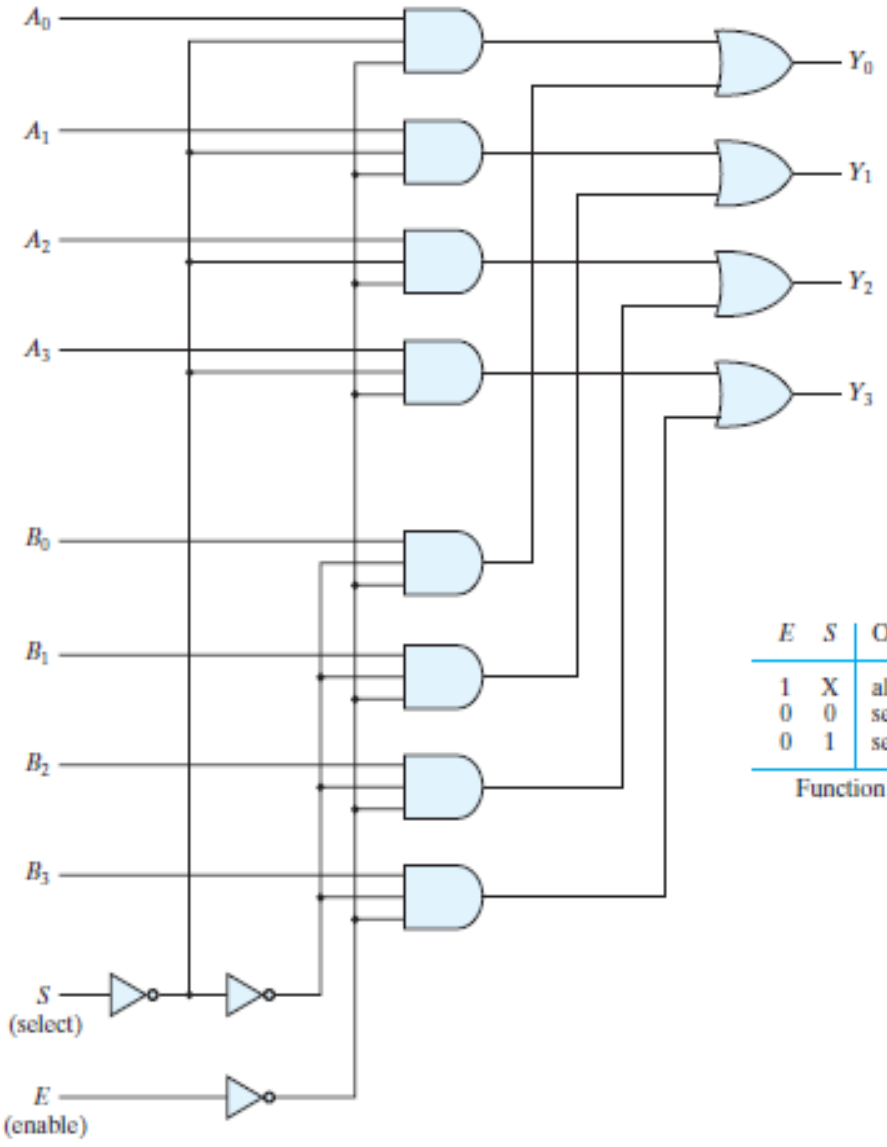
(b) Function table

Quadruple 2-to-1-Line Multiplexer

- Multiplexers can be combined with ***common selection inputs*** to provide multiple-bit selection logic
- Quadruple 2-to-1-line multiplexer:
  - Four 2-to-1-line multiplexers
  - Each capable of selecting one bit of the 2 4-bit inputs
  - E: enable input

    E=1: disable the circuit

    (all outputs are 0)



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

Function table

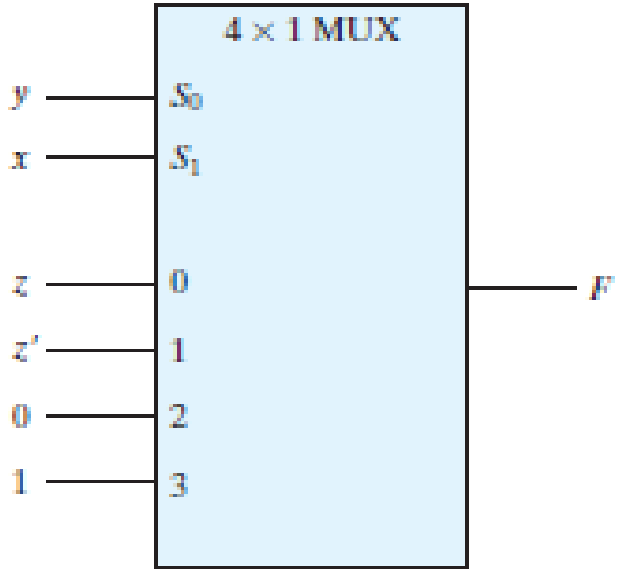
Boolean Function Implementation

- A multiplexer is essentially a decoder with an external OR gate
  - Can be used to implement Boolean functions without extra logic
- To implement a Boolean function of *n* variables:
  - Use a multiplexer with *n* - 1 selection inputs
  - The first *n* – 1 variables are connected to the selection inputs
  - The remaining variable is used for the data inputs

$F(x,y,z) = \sum(1,2,6,7)$

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



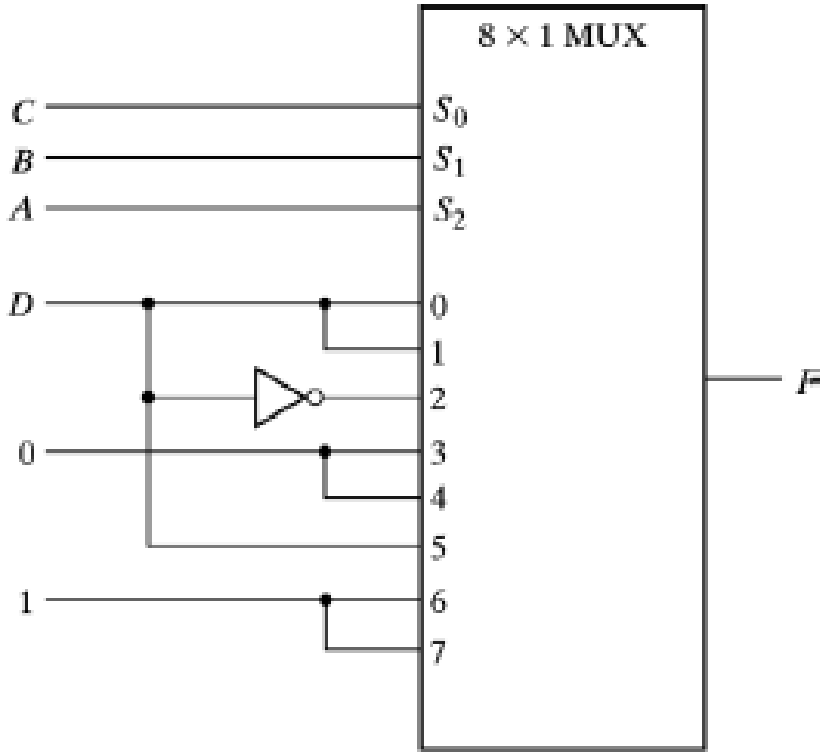
(b) Multiplexer implementation

Implementing a 4-Input Function

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

logic 0: connected to ground
logic 1: connected to Vdd (5V)

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	



can be 0, 1, the variable, or its complement

Three-State Gate

- A circuit that exhibits three states
  - logic 1, logic 0, and ***high-impedance (z)***
- The high-impedance state acts like an open circuit (disconnected)
- The most commonly used three-state gate is the buffer gate
  - C=0 → disabled (high-impedance) ; C=1 → enabled (pass)
  - Can be used at the output of a function without altering the internal implementation

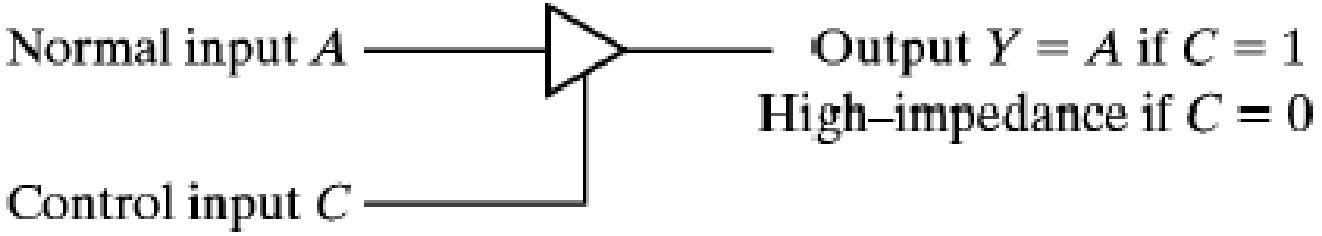
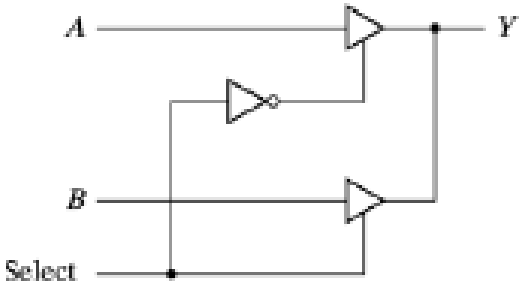


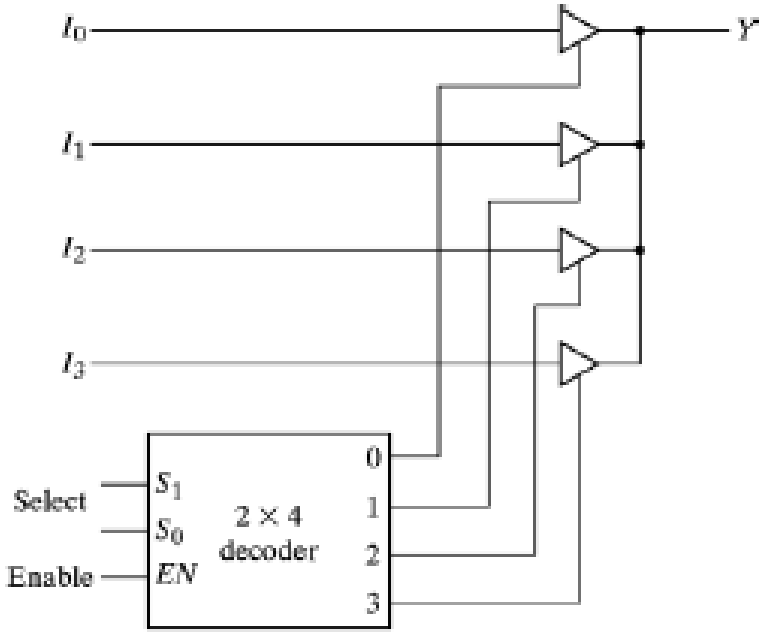
Fig. 4-29 Graphic Symbol for a Three-State Buffer

Implementation with 3-State Gates

- A large number of three-state gate outputs can be connected with wires to form a common line (bus) without logic conflicts
  - Very convenient for implementing some circuits (ex: multiplexer)
  - Only one buffer can be in the active state at any given time
- One way to ensure that no more than one control input is active at any given time is to use a decoder



(a) 2-to-1- line mux



(b) 4 - to - 1 line mux