Chapter 4
Combinational Logic

4-1 Introduction

4-2 Design Procedure

4-3 Adders

4-4 Binary Adder and Subtractor

4-5 Code Conversion

4-6 Analysis Procedure

4-7 Decimal Adders

4-8 Magnitude Comparator

4-9 Decoders and Encoders

4-10 Multiplexers

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 directly from the present combination of inputs without regard to previous inputs.

•A combinational circuit performs a specific information-processing operation fully specified logically by a set of Boolean functions.

•Sequential circuits employ memory elements (binary cells) in addition to logic gates.

•Their outputs are a function of the inputs and the state of the memory elements.

•The state of the memory elements, in turn, is a function of previous inputs.

•As a consequence, the outputs of a sequential circuit depend not only on present inputs, but also on past inputs.

•And the circuit behavior must be specified by a time sequence of inputs and internal states.

•The purpose of this chapter is to use the knowledge acquired in previous chapters and formulate various systematic design and analysis procedures of combinational circuits.

•A combinational circuit consists of input variables, logic gates, and output variables.

•The logic gates accept signals from the inputs and generate signals to the outputs.

•This process transforms binary information from the given input data to the required output data.

•Obviously, both input and output data are represented by binary signals, i.e., they exist in two possible values, one representing logic-1 and the other logic-0.

•A block diagram of combinational circuit is shown in Fig. 4-1

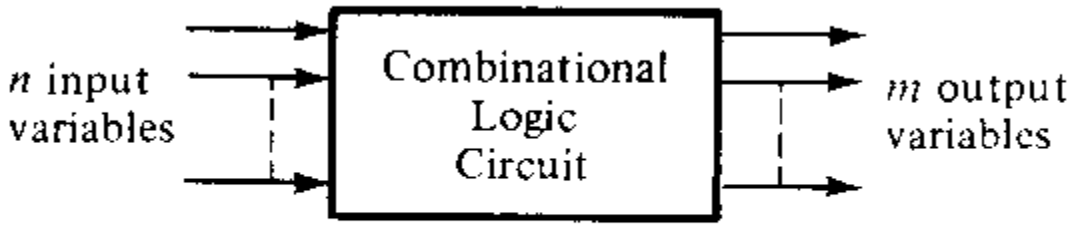


FIGURE 4-1
Block diagram of a combinational circuit

•The n input binary variables come from an external source; the m output variables go to an external destination.

•For n input variables, there are 2^n possible combinations of binary input values.

•For each possible input combination, there is one and only one possible output combination.

•A combinational circuit can be described by m Boolean functions, one for each output variable.

•Each output function is expressed in terms of the n input variables.

4-2 DESIGN PROCEDURE

The design of combinational circuits starts from the verbal outline of the problem and ends in a logic circuit diagram or a set of Boolean functions from which the logic diagram can be easily obtained.

The procedure involves the following steps:

•1. The problem is stated.

•2. The number of available input variables and required output variables is determined.

•3. The input and output variables are assigned letter symbols.

•4. The truth table that defines the required relationships between inputs and outputs is derived.

•5. The simplified Boolean function for each output is obtained.

•6. The logic diagram is drawn

•A truth table for a combinational circuit consists of input columns and output columns.

•The 1’s and 0’s in the input columns are obtained from the 2^n binary combinations available for n input variables.

•The binary values for the outputs are determined from examination of the stated problem.

•An output can be equal to either 0 or 1 for every valid input combination.

•However, the specifications may indicate that some input combinations will not occur.

•These combinations become don’t care conditions.

•The output functions specified in the truth table give the exact definition of the combinational circuit.

•It is important that verbal specifications be interpreted correctly into a truth table.

•Sometimes the designer must use intuition and experience to arrive at the correct interpretation.

•Word specifications are very seldom complete and exact.

•Any wrong interpretation that results in an incorrect truth table produces a combinational circuit that will not fulfill the stated requirements.

•The output Boolean function from the truth table are simplified by any available method, such as algebraic manipulation, the map method, or the tabulation procedure.

•Usually, there will be a variety of simplified expressions from which to choose.

•However, in any particular application, certain restrictions, limitations, and criteria will serve as a guide in the process of choosing a particular algebraic expression.

•A practical design method would have to consider such constraints as

•(1) minimum number of gates

•(2) minimum number of inputs to a gates

•(3) minimum propagation time of the signal through the circuit,

•(4)minimum number of interconnections, and

•(5)limitations of the driving capabilities of each gate.

•All these criteria cannot be satisfied simultaneously.

•The importance of each constraint is dictated by the particular application.

•In most cases, the simplification begins by satisfying an elementary objective, such as producing a simplified Boolean function in a standard form, and from that proceeds to meet any other performance criteria.

•**4-3 ADDERS**

•Digital computers perform arithmetic operation.

•Among these is the addition of two binary digits.

•This simple addition consists of four possible operations, namely 0+0=0, 0 +1=1, 1 +0=1, and 1 +`1=10.

•The first three operations produce a sum whose length is one.

•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.

•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.**

•One that performs the addition of three bits (two significant bits and a previous carry) is a **full-adder.**

•The name of the former stems from the fact that two half-adders can be employed to implement a full-adder.

•The two adder circuits are the first combinational circuits we shall design.

Half-Adder

•From the verbal explanation of a half-adder, we find that this circuit needs two binary inputs and two binary outputs.

•The input variables designate the augend and addend bits.

•The output variables produce the sum and carry.

•It is necessary to specify two output variables because the result may consists of two binary digits.

•We arbitrarily assign symbols x and y to the two inputs and S(for sum) and C(for carry) to the outputs.

•Now that we have established the number and names of the input and output variables, we are ready to formulate a truth table to identify exactly the function of the half-adder.

•This truth table is

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

•The carry output is 0 unless both inputs are 1.

•The S output represents the least significant bit of the sum.

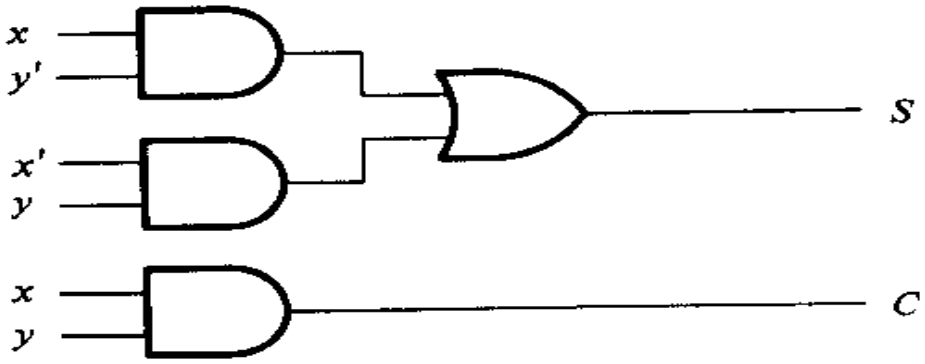
•The simplified Boolean functions for the two outputs can be obtained from the truth table.

- The simplified sum of products expressions are
  - $S = x'y + xy'$
  - $C = xy$

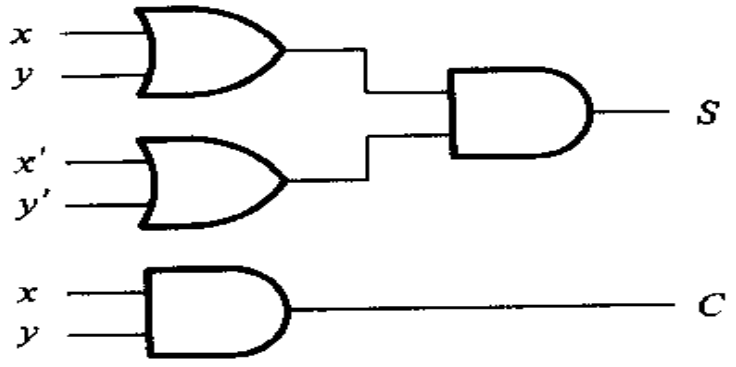
- The logic diagram for this implementation is shown in Fig. 4-2(a), as are four other implementations for a half-adder.

- They all achieve the same result as far as the input-output behavior is concerned.

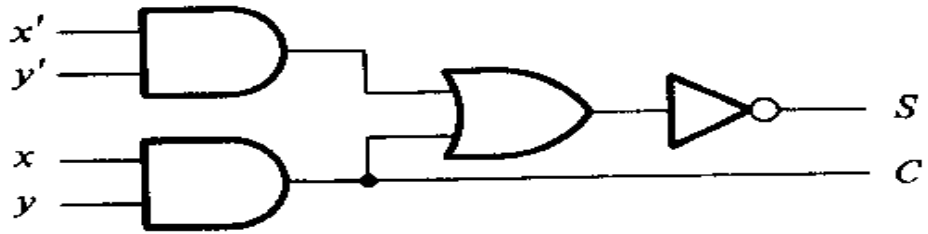
- They illustrate the flexibility available to the designer when implementing even a simple combinational logic function such as this.



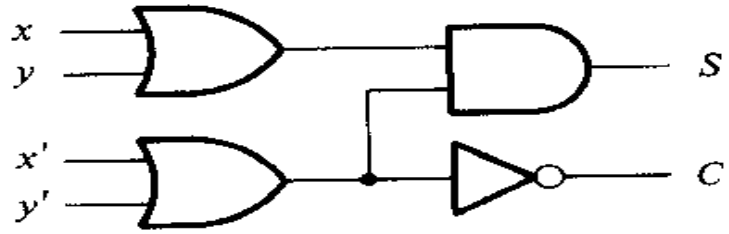
(a) $S = xy' + x'y$
$C = xy$



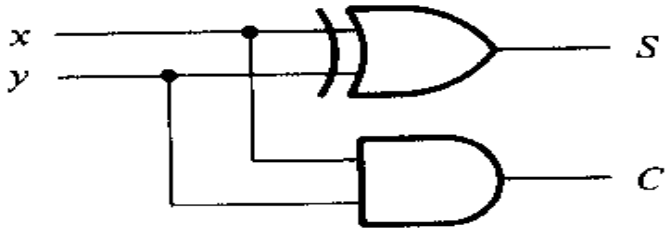
(b) $S = (x + y)(x' + y')$
$C = xy$



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



(d) $S = (x + y)(x' + y')$
$C = (x' + y')'$



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

FIGURE 4-2
Various implementations of a half-adder

•Figure 4-2(a), as mentioned before, is the implementation of the half-adder in sum of products.

•Figure 4-2(b) shows the implementation in product of sums:

- •S = (x + y)(x’ + y’)
- •C = xy

•To obtain the implementation of Fig. 4-2(c), we note that S is the exclusive-OR of x and y.

•The complement of S is the equivalence of x and y:

- •S’ = xy + x’y’

- •But C= xy, and therefore, we have

- •S = (C + x’y’)’

•In Fig. 4-2(d), we use the product of sums implementation with C derived as follows:

- C = xy = (x’ + y’)’

•The half-adder can be implemented with an exclusive-OR and an AND gate, as shown in Fig. 4-2(c).

•This form is used later to show that two half-adder circuits are needed to construct a full-adder circuit.

•**Full-Adder**

•A full adder is a combinational circuit that forms the arithmetic sum of three input bits.

•It consists of three inputs and two outputs.

•The three inputs are denoted by x, y, and z.

- Two outputs are necessary because the arithmetic sum of three binary digits ranges in value from 0 to 3.

- The two outputs are designated by the symbols S for sum and C for carry.

- The binary variable S gives value of the least significant bit of the sum.

- The binary variable C gives the output carry.

•The truth table of the full adder is

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

•The eight rows under the input variables designate all possible combinations of 1's and 0's that these variables may have.

•The 1's and 0's for the output variables are determined from the arithmetic sum of the input bits.

- When all input bits are 0's, the output is 0.

- The S output is equal to 1 when only one input is equal to 1 or when all three inputs are equal to 1.

- The C output has a carry of 1 if two or three inputs are equal to 1.

•It is important to realize that two different interpretations are given to the values of the bits encountered in this circuit.

•The input-output logical relationship of the full-adder circuit may be expressed in two Boolean functions, one for each output variable.

•Each output Boolean function requires a unique map for its simplification.

•Each map must have eight squares, since each output is a function of three input variables.

•The maps of Fig 4-3 are used for simplifying the two output functions.

•The 1’s in the squares for the maps of S and C are determined directly from the truth table.

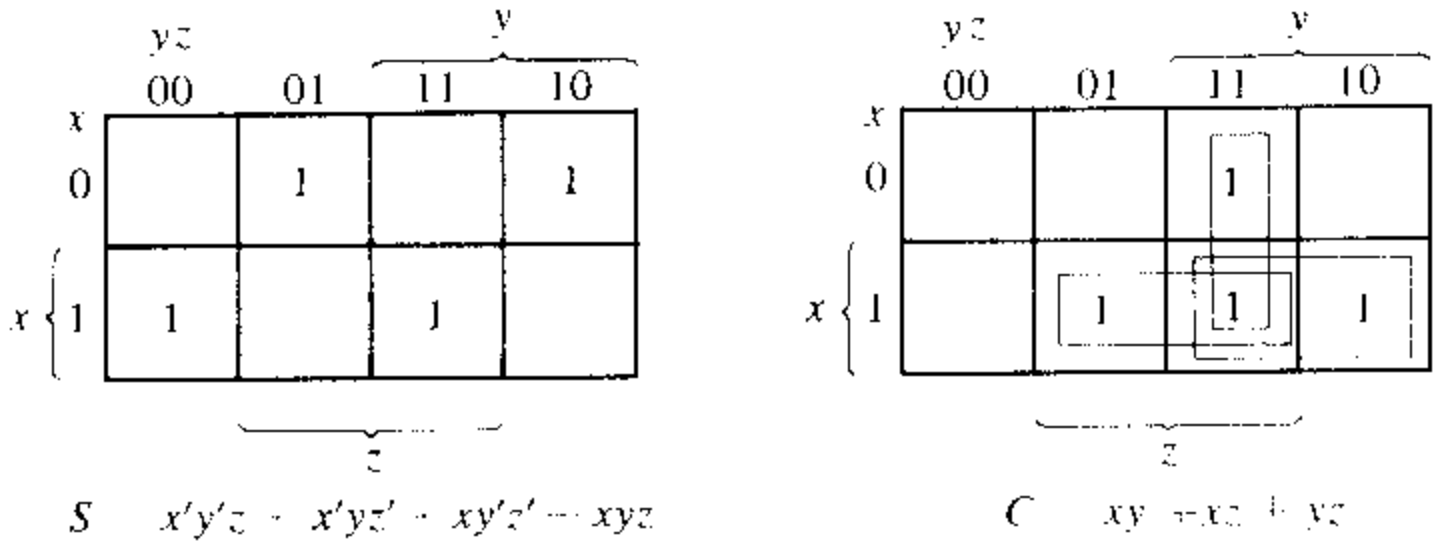


FIGURE 4-3
Maps for a full-adder

- The squares with 1's for S output do not combine in adjacent squares to give a simplified expression in sum of products.

- The C output can be simplified to a six-literal expression

•The logic diagram for the full-adder implemented in sum of products is shown in Fig. 4-4.

•This implementation uses the following Boolean expressions:

- •S = x’y’z + x’yz’ + xy’z’ + xyz
- •C = xy + xz + yz

•A full-adder can be implemented with two half-adders and one OR gate, as shown in Fig 4-5.

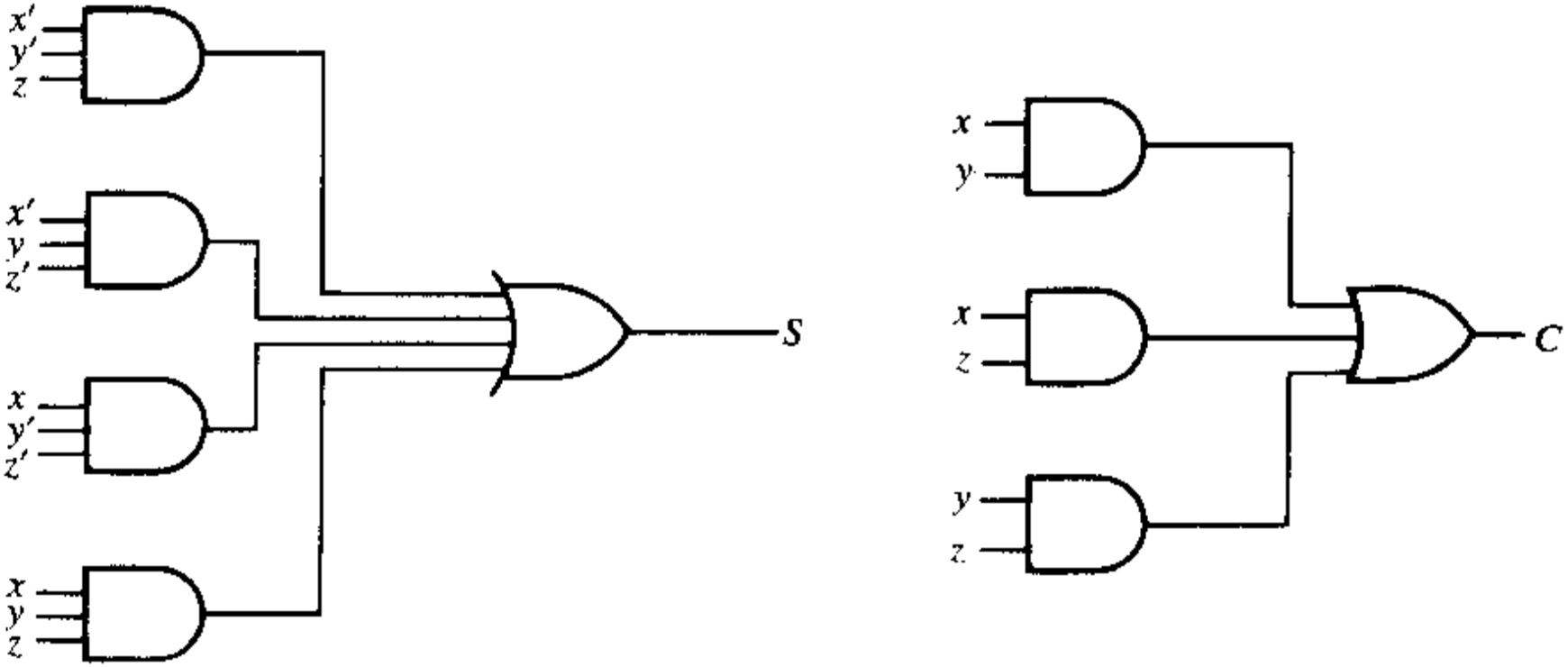


FIGURE 4-4

Implementation of a full-adder in sum of products

•The S output from the second half-adder is the exclusive-OR of z and the output of the first half-adder, giving

$$
\begin{aligned}
S &= z \oplus (x \oplus y) \\
&= z'(xy' + x'y) + z(xy' + x'y)' \\
&= z'(xy' + x'y) + z(xy + x'y') \\
&= xy'z' + x'yz' + xyz + x'y'z
\end{aligned}
$$

•And the carry output is

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

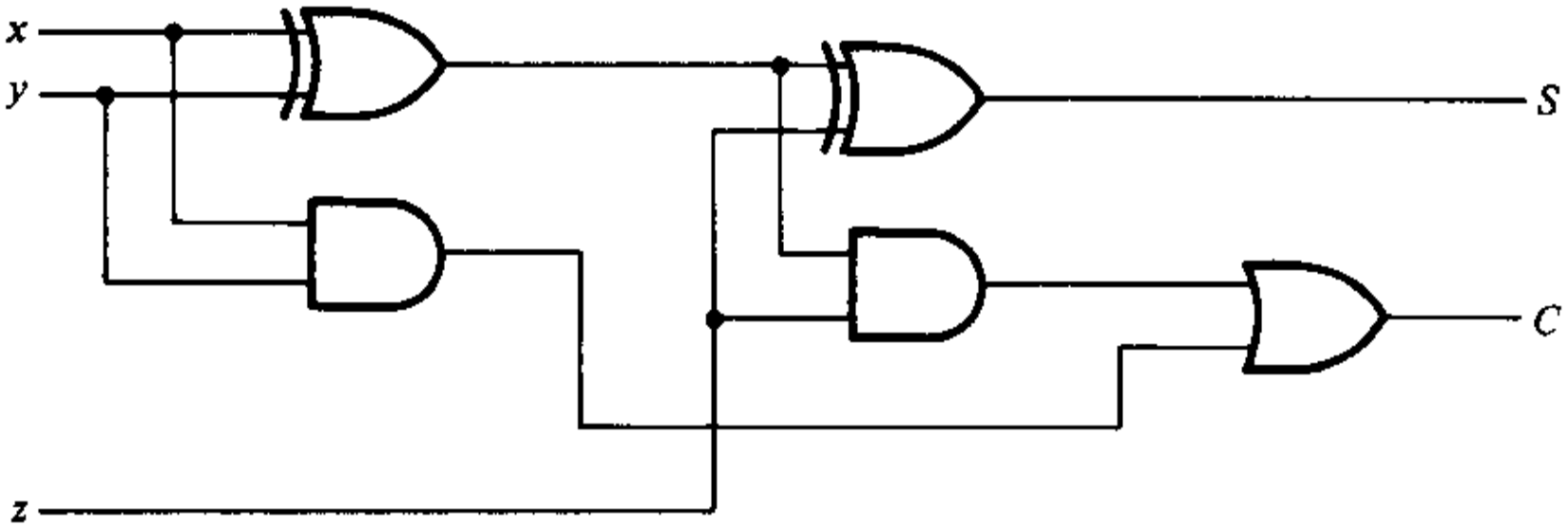


FIGURE 4-5

Implementation of a full-adder with two half-adders and an OR gate

4-4 BINARY ADDER AND SUBTRACTOR

- Two binary numbers of n bits each can be added by means of the full adder.

- To demonstrate with a specific example, consider two binary number A=1011 and B=0011, whose sum is S=1110.

- When a pair of bits are added through a full adder, the circuit produces a carry to be used with the pair of bits one significant position higher.

- This is shown in the following table.

Subscript i	4	3	2	1		Full-adder of Fig. 4-5
Input carry	0	1	1	0	C_i	z
Augend	1	0	1	1	A_i	x
Addend	0	0	1	1	B_i	y
Sum	1	1	1	0	S_i	S
Output carry	0	0	1	1	C_{i+1}	C

- The bits are added with full-adders, starting from the least significant position (subscript 1), to form the sum bit and carry bit.

- The inputs and outputs of the full-adder circuit are also indicated.

- The input carry C1 in the least significant position must be 0.

•The value of Ci + 1 in a given significant position is the output carry of the full adder.

•This value is transferred into the input carry of the full-adder that adds the bits one higher significant position to the left.

•The sum bits are thus generated starting fro m the rightmost position and are available as soon as the corresponding previous carry bit is generated.

•The sum of two n-bit binary number, A and B, can be generated in two ways: either in a serial fashion or in parallel.

•The serial addition method used only one full circuit and a storage device to hold the generated output carry.

•The pair of bits in A and B are transferred serially, one at a time, through the single full-adder to produce a string output bits for the sum.

•The stored output carry from one pair of bits is used as an input carry for the next pair of bits.

•The parallel method uses n full-adder circuits, and all bits of A and B are applied simultaneously.

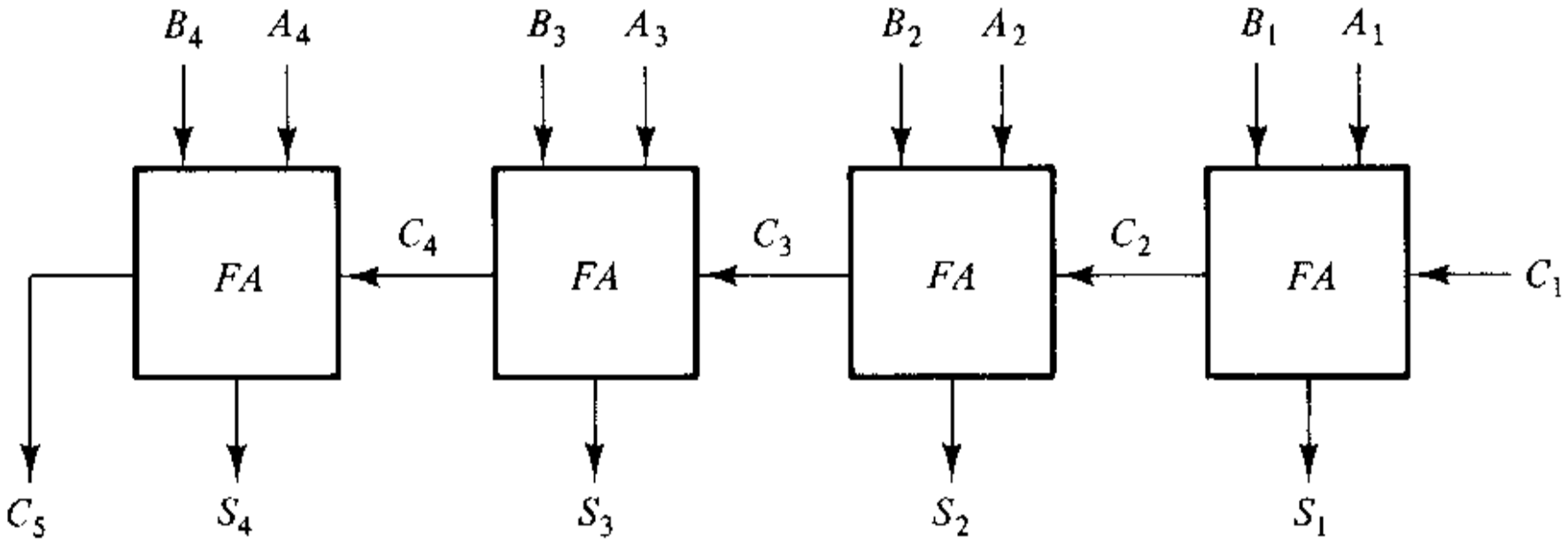
•The output carry from one full-adder is connected to the input carry of the full-adder one position to its left.

•As soon as the carries are generated, the correct sum bits emerge from the sum outputs of all full-adders.

•**Binary Parallel Adder**

•A binary parallel adder is a digital circuit that produces the arithmetic sum of two binary numbers in parallel.

•Figure 5-3(a) shows the interconnection of four full-adder (FA) circuits to provide a 4-bit binary parallel adder



(a) 4-bit parallel adder

Fig.5-2 Binary Parallel Adder

•The augend bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript 1 denoting the low-order bit.

•The carries are connected in chain through the full-adders.

•The input carry to the adder is C_1 and the output carry is C_5.

•The S outputs generate the required sum bits.

•When the 4-bit full-adder circuit is enclosed within an IC package, it has four terminals for the augend bits, and four terminals for the addend bits.

•It also has four terminals for the sum bits, and two terminals for the input and output carries.

•An n bit parallel adder requires n full-adders.

•It can be constructed from 4-bit, 2-bit, and 1-bit full adders by cascading several packages.

•The output carry from one package must be connected to the input carry of the one with the next higher-order bits.

•The 4-bit full-adder is a typical example of an MSI(Middle Scale Integrated Circuit) function.

•**Binary Adder-Subtractor**

•The subtraction of binary numbers can be done most conveniently by means of complements.

•The subtraction A – B can be done by taking the 2's complement of B and adding it to A.

•The 2's complement can be obtained by taking the 1's complement and adding one to the least significant pair of bits.

•The 1's complement can be implemented with inverters and a one can be added through the input carry.

•The circuit for subtracting A – B consists of a parallel adder with inverters placed between each data input B and the corresponding input of the full-adder.

•The input carry C_1 must be equal to 1 when performing subtractions.

•The operation thus performed becomes A plus the 1's complement of B plus 1.

•This is equal to A plus the 2's complement of B.

•For unsigned numbers, this gives A –B if A >= B.

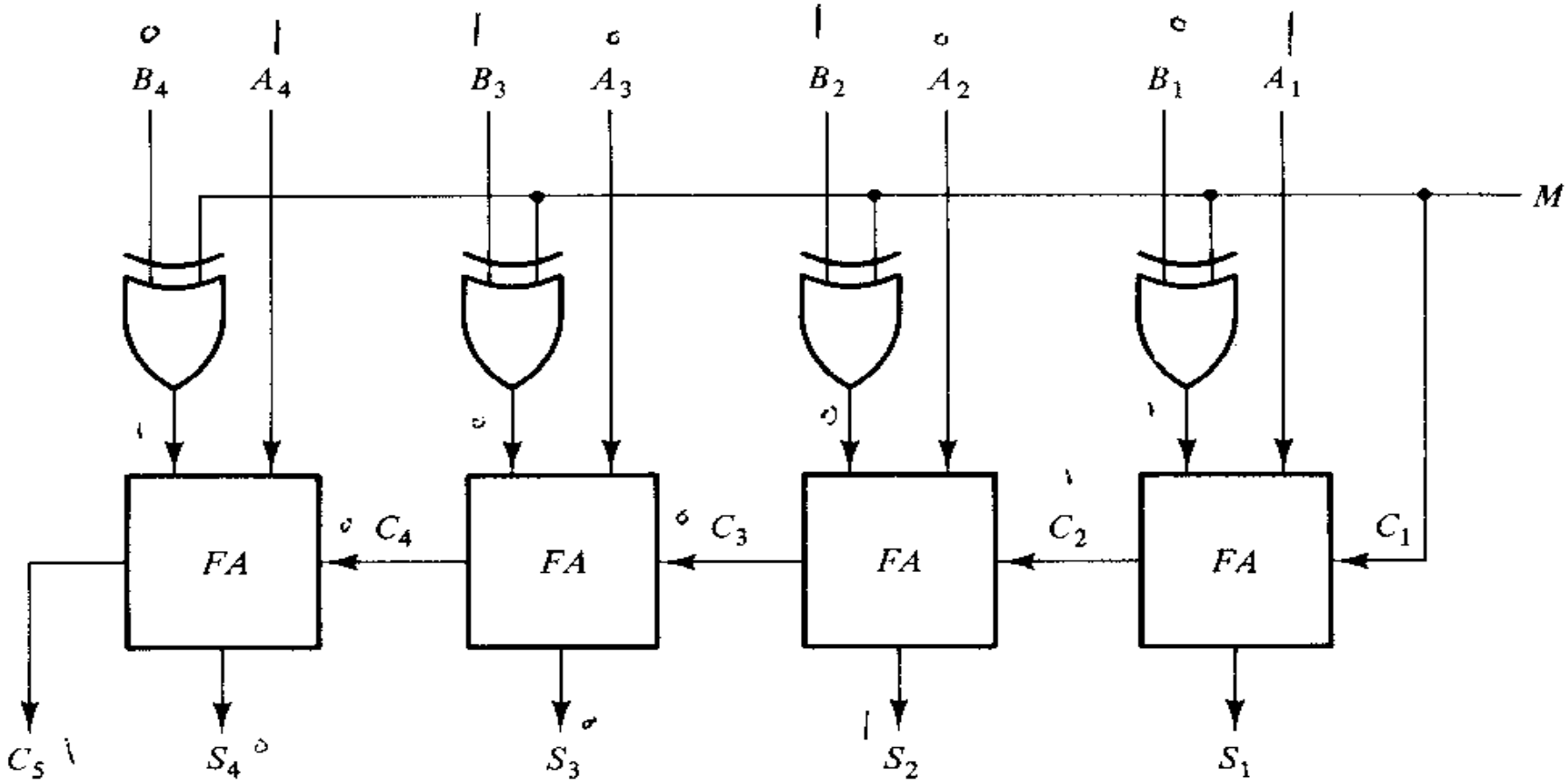
•It also gives the 2's complement of B-A if A<B

•For signed numbers, the result is A-B provided there is no overflow.

•The addition and subtraction can be combined into one circuit with one common binary adder.

•This is done by including an exclusive-0R gate with each full-adder.

•A 4-bit adder-subtractor circuit is shown in Fig. 5-2(b).



(b) 4-bit adder-subtractor

Fig 5-2 4-bit Binary Parallel Adder Subtractor Circuit

•The mode input M control the operation.

•When M =0, the circuit is an adder, and when M=1, the circuit becomes a subtractor.

•Each exclusive-OR gate receives input M and one of the inputs of B.

•When M=0, we have $B \oplus 0 = B$.

•The full-adders receive the value of B, the input carry is 0, and the circuit performs A plus B.

•When M=1, we have $B \oplus 1 = B'$ and C_1 =1.

•The B inputs are all complemented and a 1 is added through the input carry.

•The circuit performs the operation A plus the 2's complement of B.

•**Carry Propagation**

•The addition of two binary numbers in parallel implies that all the bits of the augend and the 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 a parallel 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 Si in 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.

•Consider output S_4 in Fig. 5-2(a).

•Inputs A4 and B4 reach a steady value as soon as input signals are applied to the adder.

•But input carry C_4 does not settle to its final steady-state value until C_3 is available in its steady-state value.

•The number of gate level for the carry propagation can be found from the circuit of the full-adder.

•This circuit was derived in Fig. 4-5 and is redrawn in Fig. 5-3 for convenience.

•The input and output variables use the subscript i to denote a typical stage in the parallel adder.

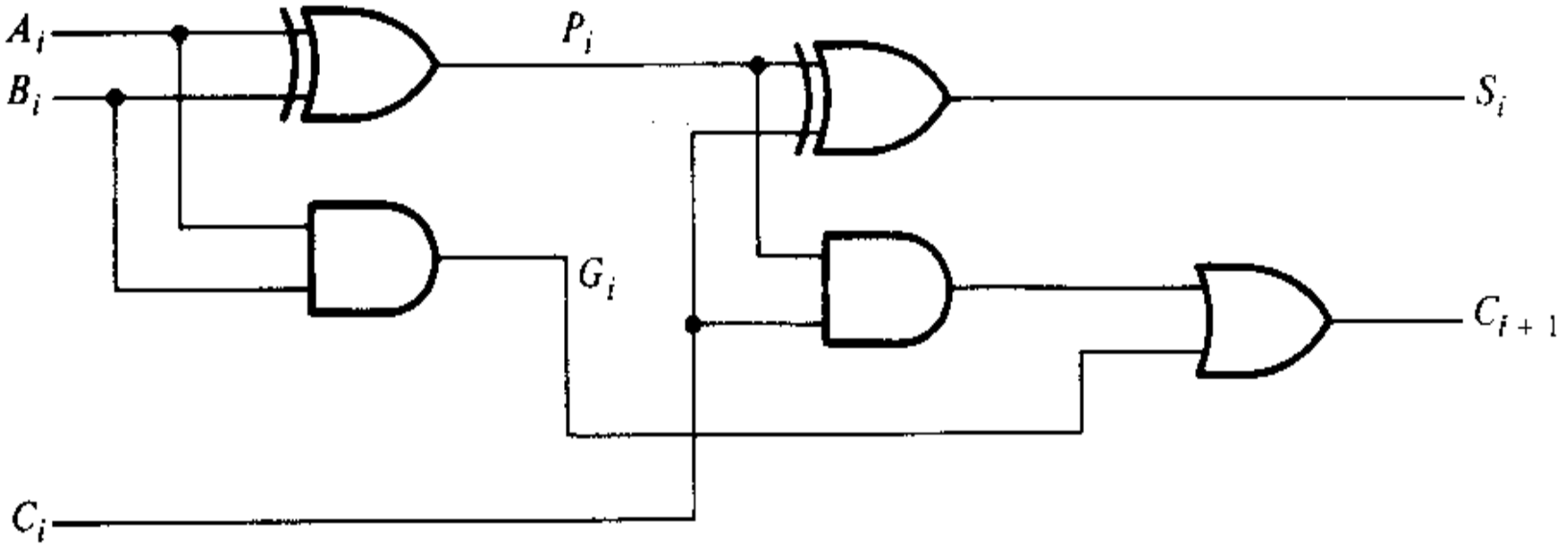


FIGURE 5-3
Full-adder circuit

•The signals at Pi and Gi settle to their steady-state values after the propagation through their respective gates.

•These two signals are common to all full-adders and depend only on the input augend and addend bits.

•The signal from the input carry, Ci, to the output carry, Ci+1, propagates through an AND gate and an OR gate, which constitute two gate levels.

•If there are four full-adders in the parallel adder, the output carry C_5 would have 2x4=8 gate levels from C_1 to C_5.

•For n-bit parallel adder, there are 2n gate levels for the carry to propagate through.

•The carry propagation time is a limiting factor on the speed with which two numbers are added in parallel.

•Although a parallel adder, will always have some value at its output terminals, the outputs will not be correct unless the signals are given enough time to propagate through the gates connected from the inputs to the outputs.

•Since all other arithmetic operations are implemented by successive additions, the time consumed during the addition process is very critical.

•An obvious solution for reducing the carry propagation delay time is to employ faster gates with reduced delays.

•But physical circuits have a limit to their capability.

•Another solution is to increase the equipment complexity in such a way that the carry delay time is reduced

•There are several techniques for reducing the carry propagation time in parallel adder.

•The most widely used technique employs the principle of **look-ahead carry** and is described below.

•Consider the circuit of the full-adder shown in Fig. 5-3.

•If we define two new binary variables:

$$P_i = A_i \oplus B_i$$

$$G_i = A_i B_i$$

•The output sum and carry can be expressed as

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_i C_i$$

•Gi is called a carry generate and it produces an output carry when both Ai and Bi are one, regardless of the input carry.

•Pi is called a carry propagate because it is the term associated with the propagation of the carry from Ci to Ci+1.

•We now write the Boolean function for the carry output of each stage and substitute for each Ci its value from the previous equations:.

$$C_2 = G_1 + P_1 C_1$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2(G_1 + P_1 C_1) = G_2 + P_2 G_1 + P_2 P_1 C_1$$

$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_1$$

•Since the Boolean function for each output carry is expressed in sum of products, each function can be implemented with one level of AND gates followed by an OR gate (or by a two-level NAND).

•The three Boolean functions for C_2, C_3 and C_4 are implemented in the look-ahead carry generator shown in Fig. 5-4.

•Note that C4 does not have to wait for C_3 and C_2 to propagate; in fact, C_4 is propagated at the same time as C_2 and C_3.

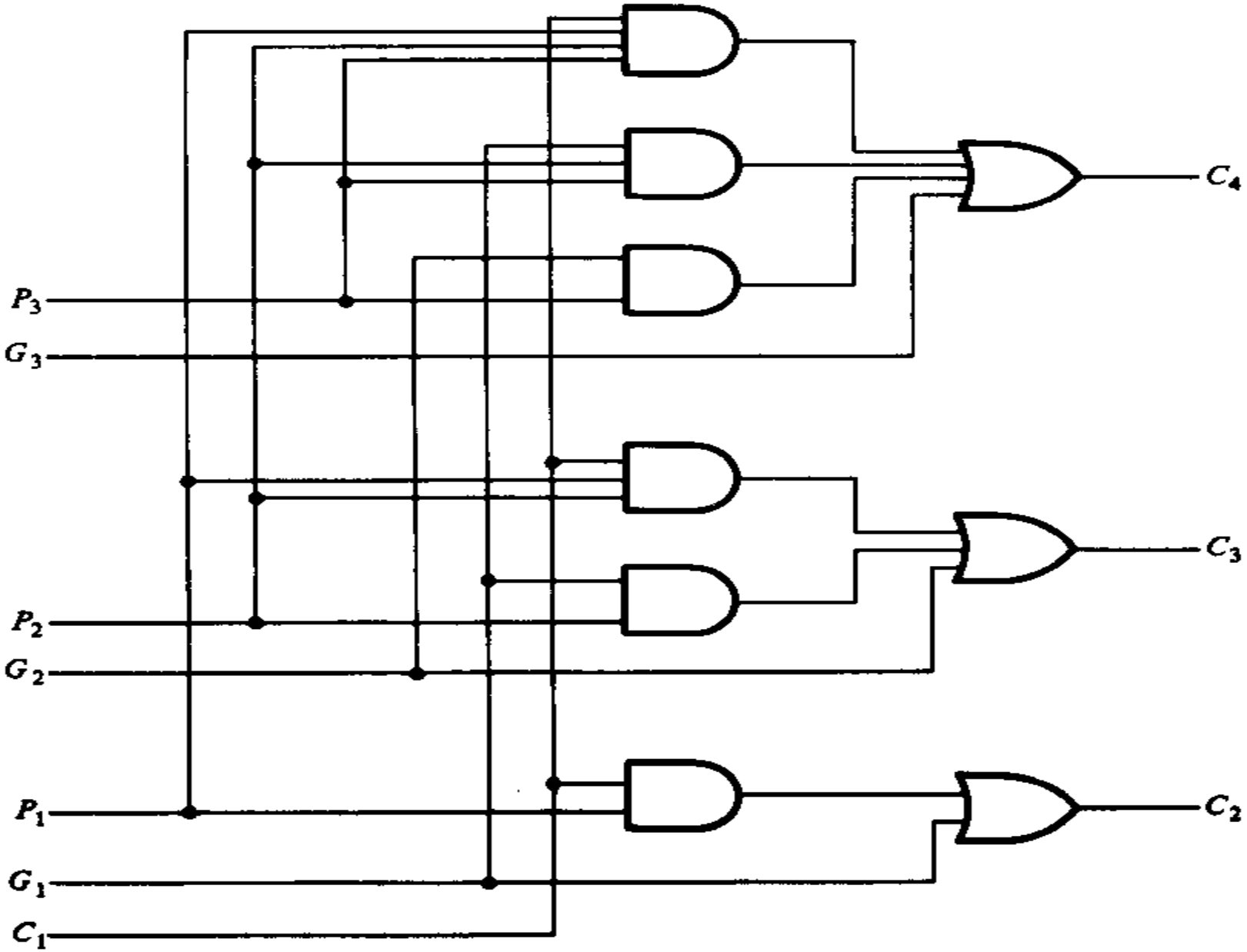


FIGURE 5-4
Logic diagram of a look-ahead carry generator

•The construction of a 4-bit parallel adder with a look-ahead carry scheme is shown in Fig. 5-5.

•Each sum output requires two exclusive-OR gates.

•The output of the first exclusive-OR gate generates the Pi variable, and the AND gate generates the Gi variable.

•All the P's and G's are generated in two gate levels.

•The carries are propagated through the look-ahead carry generator (similar to that in Fig. 5-4) and applied as inputs to the second exclusive-OR gate.

•After the P and G signals settle into their steady-state values, all output carries are generated after a delay of two levels of gates.

•Thus, outputs S2 through S4 have equal propagation delay times.

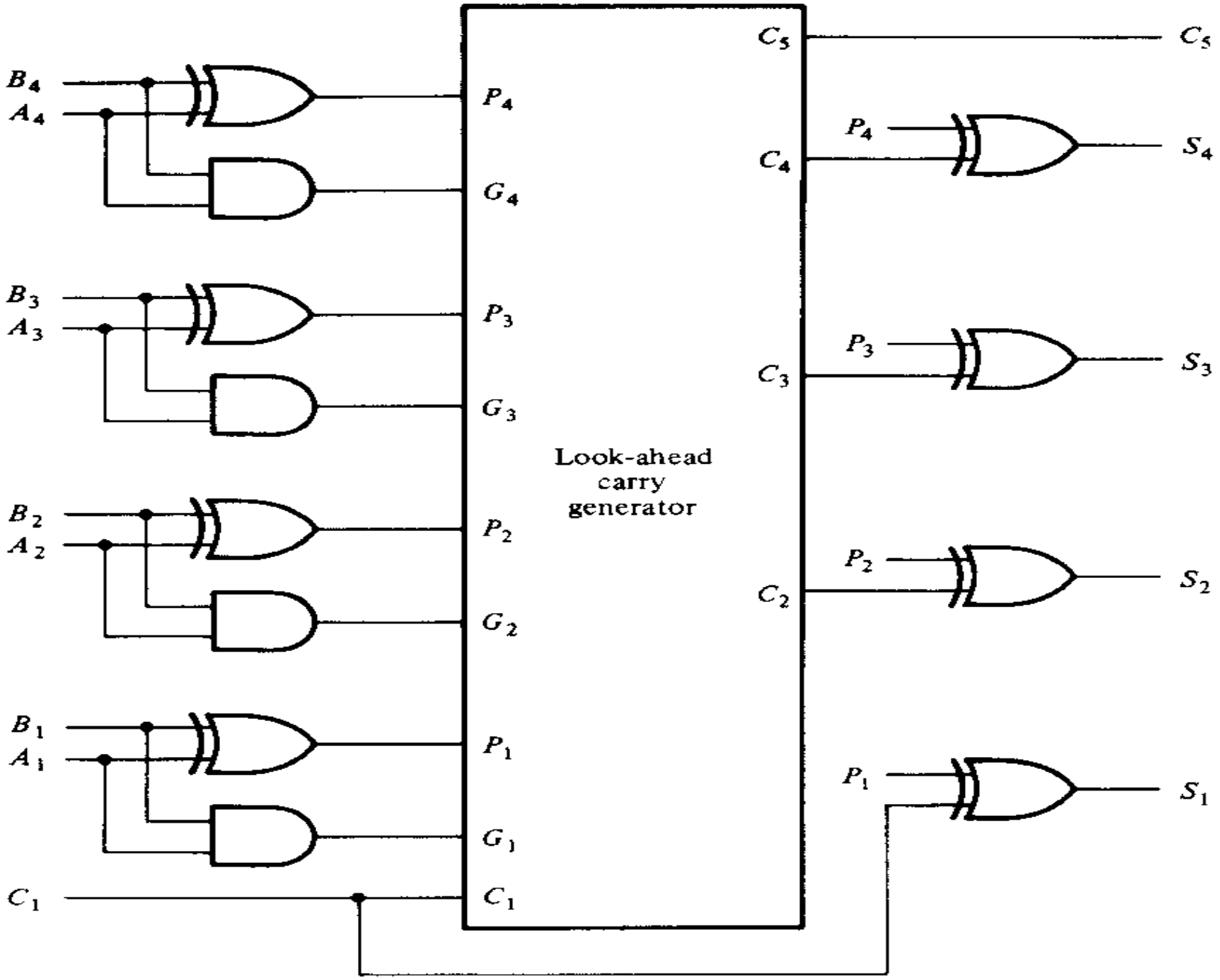


FIGURE 5-5
4-bit full-adders with look-ahead carry

4-5 Code Conversion

•The availability of a large variety of codes for the same discrete elements of information results in the use of different codes by different digital systems.

•It is sometimes necessary to use the output of one system as input to another.

•A conversion circuit must be inserted between the two systems if each use different codes for the same information.

•Thus, a code converter is a circuit that makes the two systems compatible even though each uses a different binary code.

•To convert from binary code A to binary code B, the input lines must supply the bit combination of elements as specified by code A and the output lines must generate the corresponding of code B.

•A combinational circuit performs this transformation by means of logic gates.

•The design procedure of code converters will be illustrated by means of a specific example of conversion from the BCD to the excess-3 code.

•The bit combinations for the BCD and excess-3 codes are listed in Table 1-2 (Section 1-7)

•Since each code uses four bits to represent a decimal digit, there must be four input variables and four output variables.

•Let us designate the four input binary variables by the symbols A, B, C and D, and the four output variables by w, x, y, and z.

•The truth table relating the input and output variables is shown in Table 4-1.

TABLE 4-1
Truth Table for Code-Conversion Example

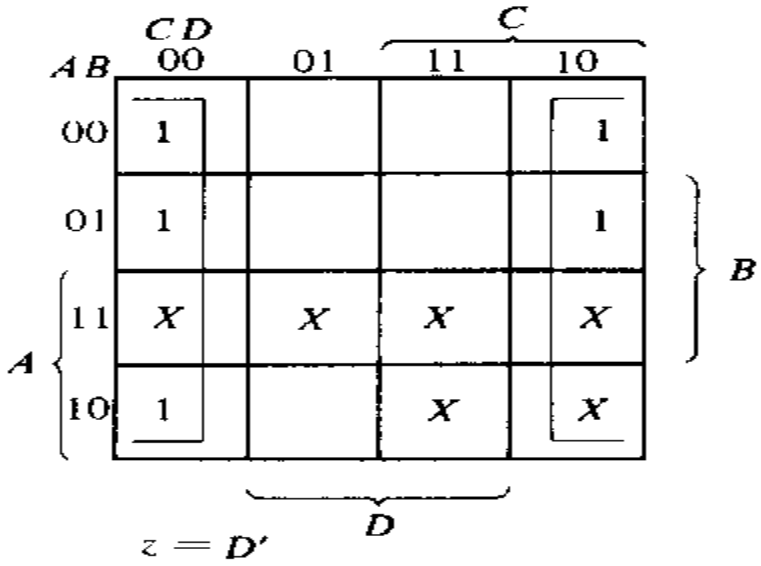
Input BCD				Output Excess-3 Code			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

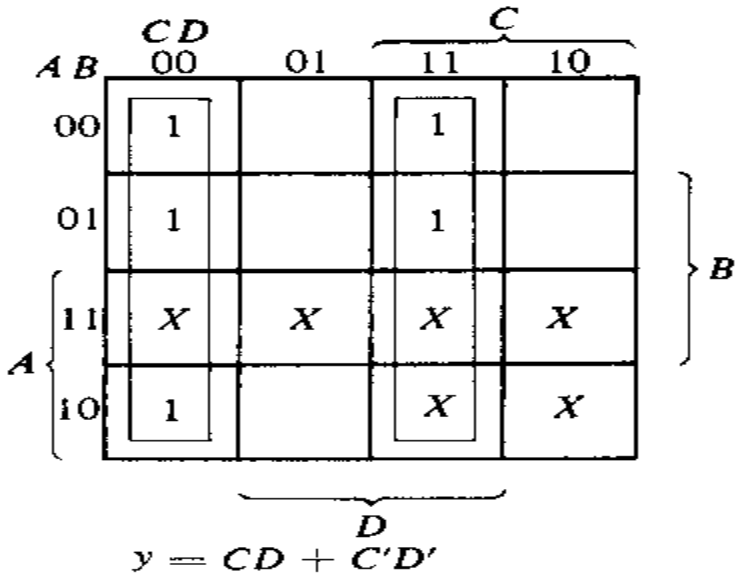
•We note that four binary variables may have 16 bit combinations, only 10 of which are listed in the truth table.

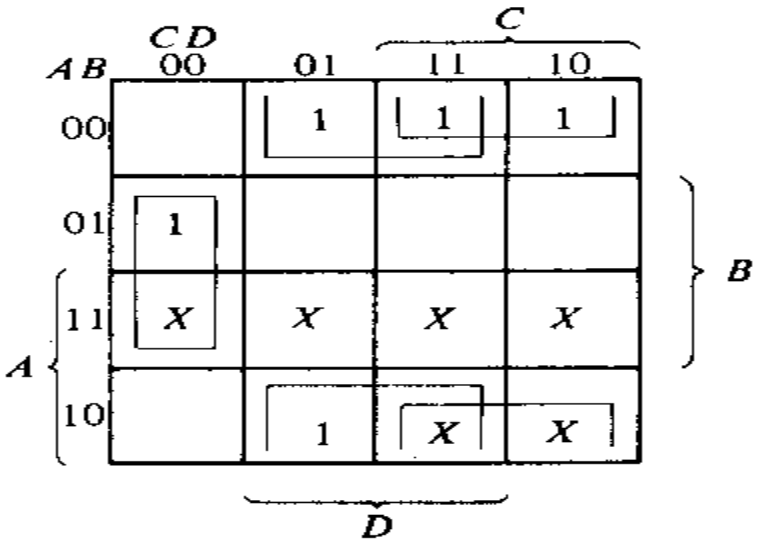
•The six bit combinations not listed for the input variables are don’t-care combinations.

•The maps in Fig 4-7 are drawn to obtain a simplified Boolean function for each output.

•Each of the four maps of Fig. 4-7 represent one of the four outputs of this circuit as a function of the four input variables.







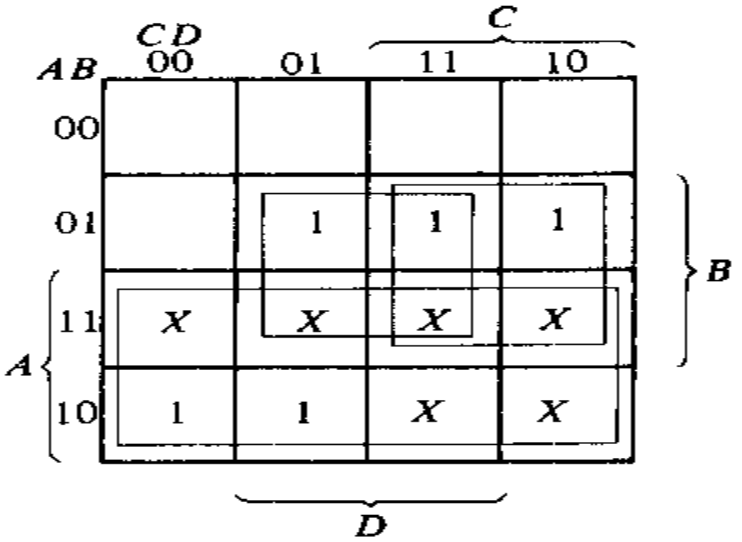


FIGURE 4-7
Maps for a BCD-to-excess-3-code converter

•The 1’s marked inside the squares are obtained from the minterms that make the output equal to 1.

•The 1’s are obtained from the truth table by going over the output columns one at a time.

•For example, the column under output z has five 1’s; therefore, the map for z must have five 1’s, each being in a square corresponding to the minterm that makes z equal to 1.

•The six don’t-care combinations are marked by X’s.

•One possible way to simplify the functions in sum of products is listed under the map of each variable.

•A two-level logic diagram may be obtained directly from the Boolean expressions derived by the maps

•There are various other possibilities for a logic diagram that implements this circuit.

•The expressions obtained in Fig. 4-7 may be manipulated algebraically for the purpose of using common gates for two or more outputs.

•This manipulation, shown below, illustrates the flexibility obtained with multiple-output systems when implemented with three or more levels of gates.

$$z = D'$$

$$y = CD + C'D' = CD + (C + D)'$$

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

$$= B'(C + D) + B(C + D)'$$

$$w = A + BC + BD = A + B(C + D)$$

•The logic diagram that implements these expressions is shown in Fig. 4-8.

•In it we see that the OR gate whose output is C + D has been used to implement partially each of the three outputs.

•Not counting input inverters, the implementation in sum of products requires seven AND and three OR gates.

•The implementation of Fig. 4-8 requires four AND gates, four OR gates, and one inverter.

•If only the normal inputs are available, the first implementation will require inverters for variables B, C, and D, whereas the second implementation requires inverters for variables B and D.

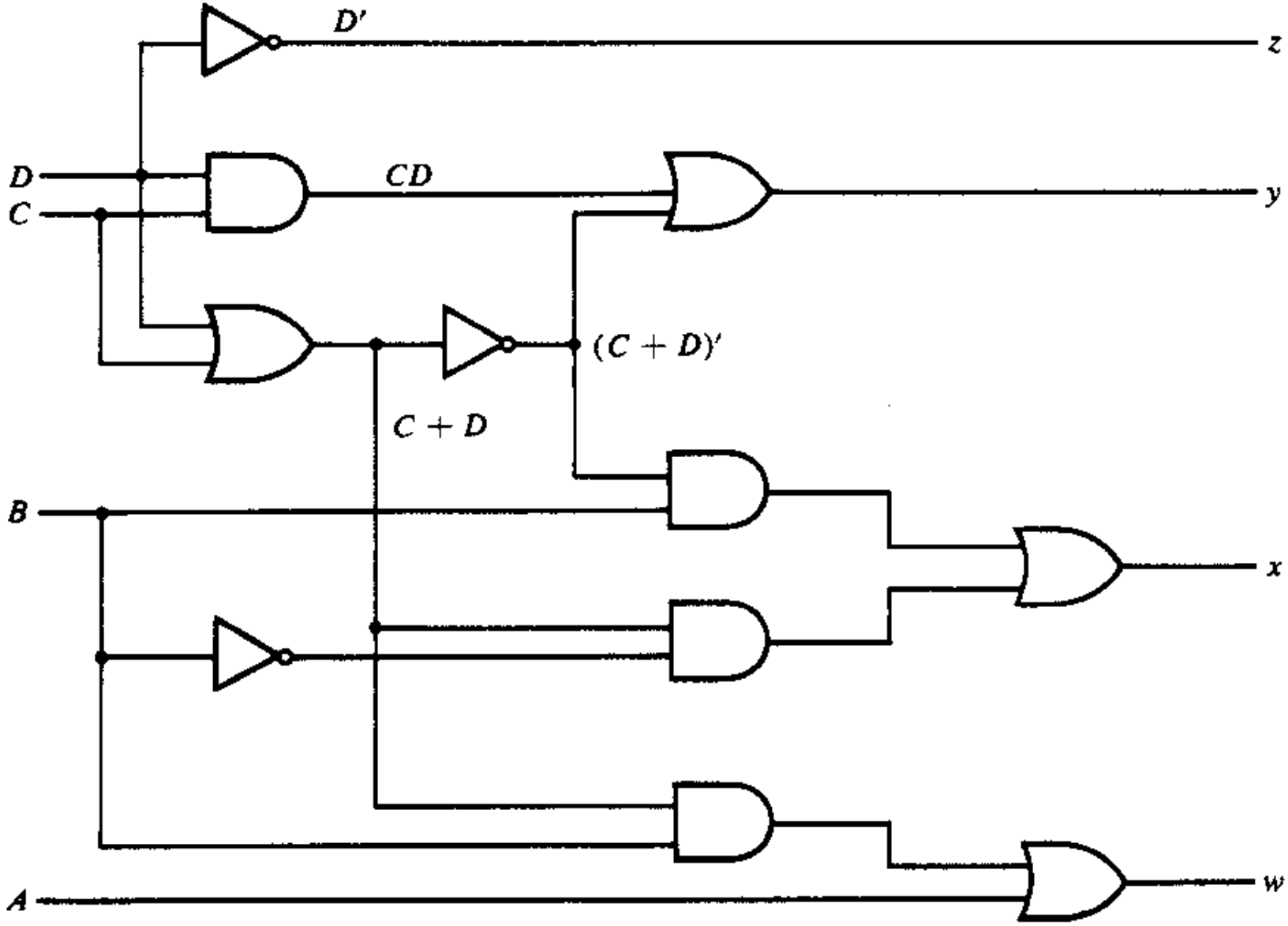


FIGURE 4-8

Logic diagram for a BCD-to-excess-3-code converter

4-6 ANALYSIS PROCEDURE

•The design of combinational circuit starts from the verbal specifications of a required function and culminates with a set of output Boolean functions or a logic diagram

•The **analysis** of a combinational circuit is somewhat the reverse process.

•It starts with a given logic diagram and culminates with a set of Boolean function, a truth table, or a verbal explanation of the circuit operation.

•If the logic diagram to be analyzed is accompanied by function name or an explanation of what it is assumed to accomplish, then the analysis problem reduces to a verification of the stated function.

•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 feed back paths or memory elements.

•A feedback path is a connection from the output of one gate to the input of a second gate that form part of the input to the first gate.

•Feedback paths or memory elements in a digital circuit define a sequential circuit and will be analyzed in later chapters.

•Once the logic diagram is verified as a combinational circuit, one can proceed to obtain the output Boolean functions and /or the truth table.

•If the circuit is accompanied by a verbal explanation of its function, then the Boolean functions or the truth table is sufficient for verification.

•If the function of the circuit is under investigation, then it is necessary to interpret the operation of the circuit from the derived truth table.

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

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

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

•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 only.

•Analysis of the combinational circuit in Fig. 4-9 illustrates the proposed procedure.

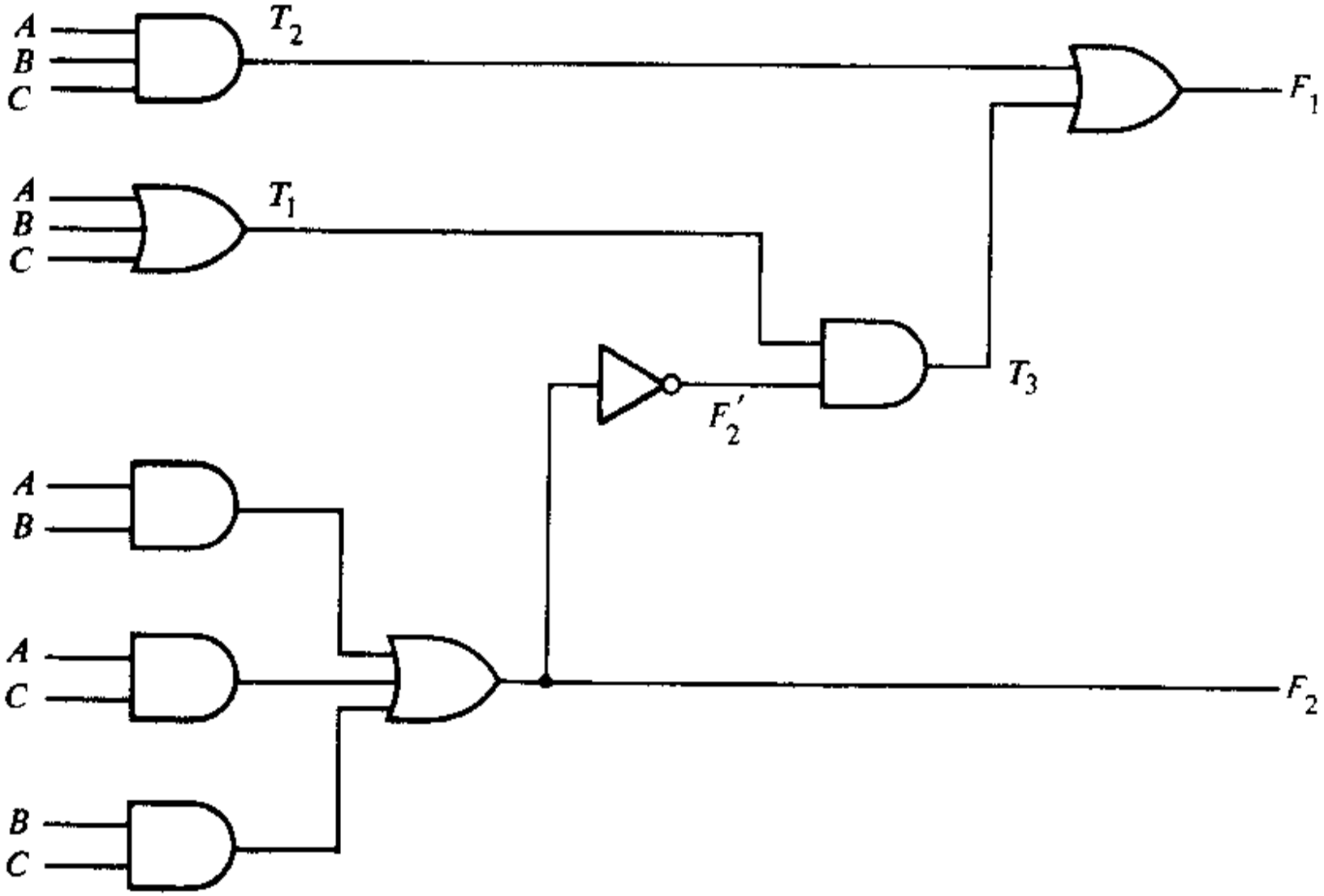


FIGURE 4-9

Logic diagram for analysis example

•We note that the circuit has three binary inputs, A, B, and C, and two binary outputs, F1 and F2.

•The outputs of various gates are labeled with intermediate symbols.

•The outputs of gates that are a function of input variables only are F2, T1, and T2.

•The Boolean functions for these three outputs are

$$F_2 = AB + AC + BC$$

$$T_1 = A + B + C$$

$$T_2 = ABC$$

•Next we consider outputs of gates that are a function of already defined symbols:

$$T_3 = F_2' T_1$$
$$F_1 = T_3 + T_2$$

•The output Boolean function F_2 just expressed is already given as a function of the inputs only.

•To obtain F1 as a function of A, B, and C, form a series of substitution as follows:

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

•If we want to pursue the investigation and determine the information-transformation task achieved by this circuit, we can derive the truth table directly from the Boolean functions and try to recognize a familiar operation.

•For this example, we note that the circuit is a full-adder, with F1 being the sum output and F2 the carry output.

•A, B, and C are the three inputs added arithmetically.

•The derivation of the truth table for the circuit is a straightforward process once the output Boolean functions are known.

•To obtain the truth table directly from the logic diagram without going through the derivations of the Boolean functions, proceed as follows:

1. Determine the number of input variables to the circuit. For n inputs, form the 2^n possible input combinations of 1's and 0's by listing the binary numbers from 0 to 2^n -1.

2. Label the outputs of selected gates with arbitrary symbols.

3. Obtain the truth table for the outputs of those gates that are a function of the input variables only.

4. Proceed to obtain the truth table for the outputs of those gates that are a function of previously defined values until the columns for all outputs are determined.

This process can be illustrated using the circuit of Fig. 4-9.

In table 4-2, we form the eight possible combinations for the thee input variables.

TABLE 4-2
Truth Table for the Logic Diagram of Fig. 4-9

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

•The truth table for F_2 is determined directly from the values of A, B, and C, with F_2 equal to 1.

•The truth table for F_2' is the complement of F_2.

•The truth tables for T_1 and T_2 are the OR and AND functions of the input variables, respectively.

•The values for T_3 are derived from T_1 and F_2'.

•T_3 is equal to 1 when both T_1 and F_2' are equal to 1, and to 0 otherwise.

•Finally, F1 is equal to 1 for those combinations in which either T_2 or T_3 or both are equal to 1.

•Inspection of the truth table combinations A, B,C, F1 and F2 of Table 4-2 shows that it is identical to the truth table of the full-adder given in Section 4-3 for x, y, z, S, and C, respectively.

Consider now a combinational circuit that has don't-care input combinations.

•When such a circuit is designed, the don't-care combinations are marked by X's in the map and assigned an output of either 1 or 0, whichever is more convenient for the simplification of the output Boolean function.

•When a circuit with don't-care combinations is being analyzed, the situation is entirely different.

•Even though we assume that don't-care input combinations will never occur, if any one of these combinations is applied to the inputs, a binary output will be present.

•The value of the output will depend on the choice of X's taken during the design.

•Part of the analysis of such a circuit may involve the determination of the output values for the don't-care input combinations.

•As an example, consider the BCD-to-excess-3 code converter designed in Section 4-5.

•The outputs obtained when the six unused combinations of the BCD code are applied to the inputs are

Unused BCD Inputs				Outputs			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
1	0	1	0	1	1	0	1
1	0	1	1	1	1	1	0
1	1	0	0	1	1	1	1
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	1
1	1	1	1	1	0	1	0

•These outputs may be derived by means of the truth table analysis method as outlined in this section.

•In this particular case, the outputs may be obtained directly from the maps of Fig. 4-7.

•From inspection of the maps, we determine whether the X's in the corresponding minterm squares for each output have been included with 1he 1's or 0's.

•For example, the square for minterm m_{10} (1010) has been included with the 1's for outputs w, x, and z, but not for y.

•Therefore, the outputs for m10 are wxyz= 1101, as listed in the previous table.

• We also note that the first three outputs in the table have no meaning In the excess-3 code, and the last three outputs correspond to decimal 5, 6 and 7 respectively.

•This coincidence is entirely a function of the choice for the X's taken during the design.

4-7 DECIMAL ADDER

•Computers or calculators that perform arithmetic operations directly in the decimal number system represent decimal numbers in binary-coded form.

•An adder for such a computer must employ arithmetic circuits that accept coded decimal numbers and present results in the accepted code.

•For binary addition, it was sufficient to consider a pair of significant bits at a time, together with a previous carry.

•A decimal adder requires a minimum of nine inputs and five outputs, since four bits are required to code each decimal digit and the circuit must have an input carry and output carry.

•The design of a nine-input, five-output combinational circuit by the classical method requires a truth table with 2^9=512 entries.

•Many of the input combinations are don’t care conditions, since each binary code input has six combinations that are invalid.

•The simplified Boolean functions for the circuit may be obtained by a computer-generated tabular method, and the result would probably be a connection of gates forming an irregular pattern.

•An alternate procedure is to add the numbers with full-adder circuits, taking into consideration the fact that six combination in each 4-bit input are not used.

•The output must be modified so that only those binary combinations that are valid combinations of the decimal code are generated.

•**BCD Adder**

•Consider the arithmetic addition of two decimal digits in BCD, together with a possible carry from a pervious stage.

•Since each input digit does not exceed 9, the output sum cannot be greater than 9+9+1 = 19, the 1 in the sum being an input carry.

•Suppose we apply two BCD digits to a 4-bit binary adder.

•The adder will form the sum in binary and produce a result that may range from 0 to 19.

•These binary numbers are listed in Table 5-1 and are labeled by symbols K, Z8, Z4, Z2 and Z1.

•K is the carry, and the subscripts under the letter Z represent the weights 8, 4, 2, and 1 that can be assigned to the four bits in the BCD code.

TABLE 5-1
Derivation of a 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

•The first column in the table lists the binary sums as they appear in the outputs of a 4-bit binary adder.

•The output sum of two decimal digits must be represented in BCD and should appear in the form listed in the second column of the table.

•We need to find a simple rule by which the binary number in the first column can be converted to the correct BCD-digit representation of the number in the second column.

•In examining the contents of the table, it is apparent that when the binary sum is equal t or less than 1001, the corresponding BCD number is identical, and therefore no conversion is needed.

•When the binary sum is greater than 1001, we obtain a non-valid BCD representation.

•The addition of binary 6(0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.

•The logic circuit that detect the necessary correction can be derived from the table entries.

•It is obvious that a correction is needed when binary sum has an output carry K=1.

•The other six combinations from 1010 to 1111 that need correction have a 1 in the position Z8.

•To distinguish them from binary 1000 and 1001, which also have a 1 in position Z8, we specify further that either Z4 or Z2 must have a 1.

•The condition for correction and an output carry can be expressed by the Boolean function

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

•When C=1, it is necessary to add 0110 to the binary sum and provide an output carry for the next stage.

•A BCD adder is a circuit that adds two BCD digits in parallel and produces a sum digit also in BCD.

•A BCD adder must include the correction logic in its internal construction.

•To add 0110 to the binary sum, we use a second 4-bit binary adder, as shown in Fig. 5-6.

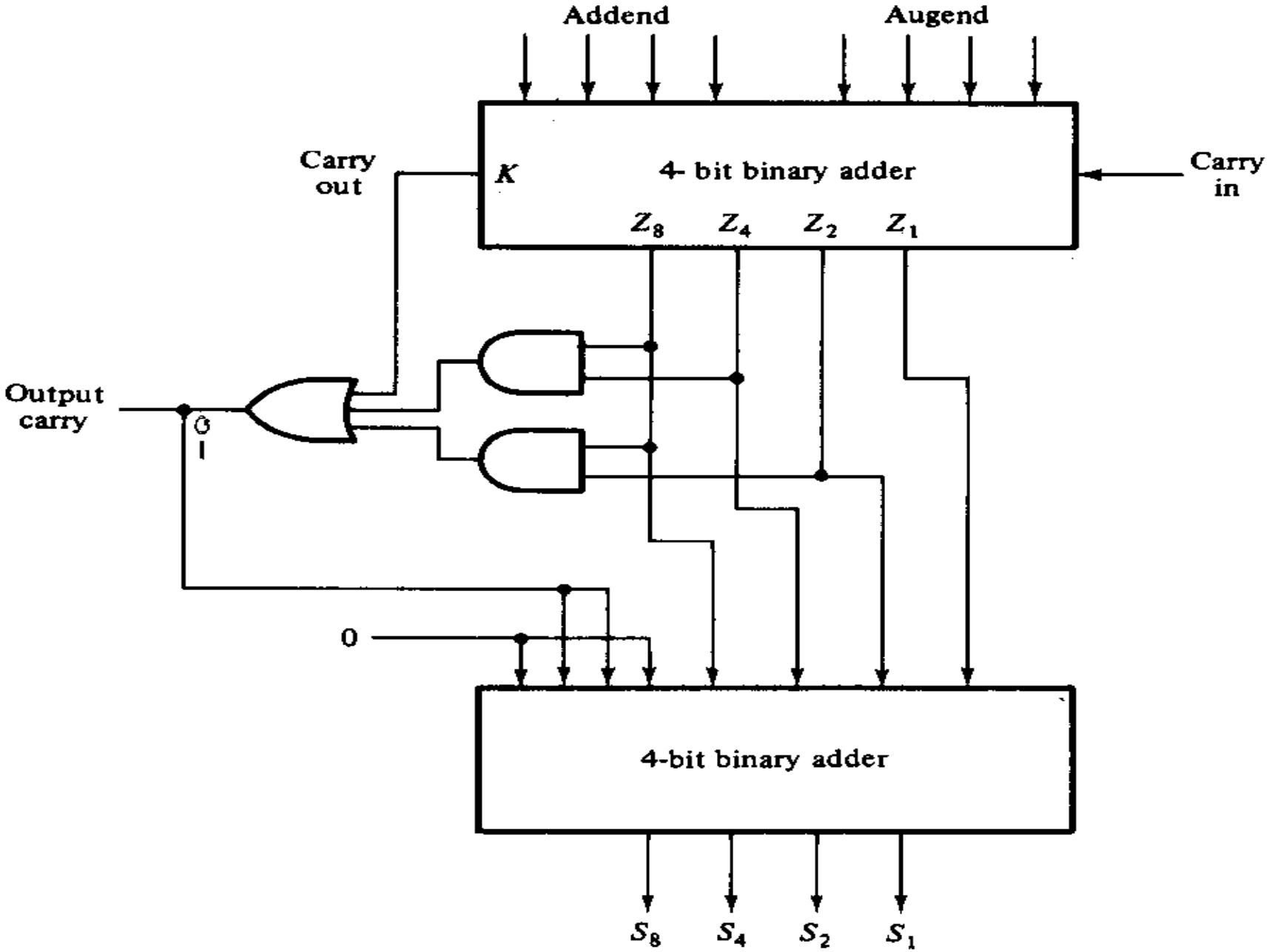


FIGURE 5-6
Block diagram of a BCD adder

•The two decimal digits, together with the input carry, are first added in the top 4-bit binary adder to produce the binary sum.

•When the output carry is equal to zero, nothing is added to the binary sum.

•When the output carry is equal to one, binary 0110 is added to the binary sum through the bottom 4-bit binary adder.

•The output carry generated from the bottom binary adder can be ignored, since it supplies information already available at the output-carry terminal.

•The BCD adder can be constructed with three IC packages.

•Each of the 4-bit adders is an MSI function and the three gates for the correction logic need one SSI package.

•However, the BCD adder is available in one MSI circuit

•To achiever shorter propagation delays, an MSI BCD adder includes the necessary circuits for the look-ahead carries.

•A decimal parallel adder that adds n decimal digits needs n BCD adder stages.

•The output carry from one stage must be connected to the input carry of the next higher-order stage.

•4-8 MAGNITUDE COMPARATOR

•The comparison of two numbers is an operation that determines if one number is greater than, less than, or equal to the other number.

•A **magnitude comparator** is a combinational circuit that compares two number, 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.

• The circuit for comparing two n-bit numbers has 2^2n entries in the truth table and becomes too cumbersome even with n=3.

•On the other hand, as one may suspect, a comparator circuit possesses a certain amount of regularity.

•Digital functions that possess an inherent well-defined regularity can usually be designed by means of an algorithmic procedure if one is found to exist.

•An **algorithm** is a procedure that specifies a finite set of steps that, if followed, give the solution to a problem.

•We will derive the algorithm for the design of a 4-bit magnitude comparator.

•The algorithm is a direct application of the procedure a person uses to compare relative magnitudes of two numbers.

•Consider two numbers; A and B, with four digits each.

•Write the coefficients of the number with descending significance as follows:

$$A = A_3 A_2 A_1 A_0$$
$$B = B_3 B_2 B_1 B_0$$

•Each subscripted letter represents one of the digits in the number.

•The two numbers are equal if all pairs of significant digits are equal.

•Therefore, if $A_3=B_3$ and $A_2=B_2$ and $A_1=B_1$ and $A_0=A_0$, the two numbers are equal.

•When the numbers are binary, the digits are either 1 and 0.

•The equality relation of each pair of bits can be expressed logically with an equivalence function:

$$x_i = A_i B_i + A_i' B_i' \qquad i = 0, 1, 2, 3$$

Where x_i =1 only if the pair of bits in position 1 are equal, i.e, if both are 1's or both are 0's.

The equality of the two numbers, A and B, is displayed in a combinational circuit by an output binary variables that we designate by the symbol (A = B).

This binary variable is equal to 1 if the input numbers, A and B, are equal, and it is equal to 0 otherwise.

For the equality condition to exist, all x_i variables must be equal to 1. this dictates an AND operation of all variables.

$$(A = B) = x_3 x_2 x_1 x_0$$

•The binary variable (A=B) is equal to 1 only if all pairs of digits of the two numbers are equal.

•To determine if A is greater than or less than B, we inspect the relative magnitudes of pairs of significant digits starting from the most significant position.

•If the two digits are equal, we compare the next lower significant pair of digits.

•This comparison continues until a pair of unequal digits is reached.

•If the corresponding digit of A is 1 and that of B is 0, we conclude that A >B.

•If the corresponding digit of A is 0 and that of B is 1, we have that A<B.

•The sequential comparison can be expressed logically by the following two Boolean functions:

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

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

•The gate implementation of the three output variables just derived is simpler than it seems because is involves certain amount of repetition.

•The “unequal” outputs can use the same gates that are needed to generate the “equal” output.

•The logic diagram of the 4-bit magnitude comparator is shown in Fig. 5-7.

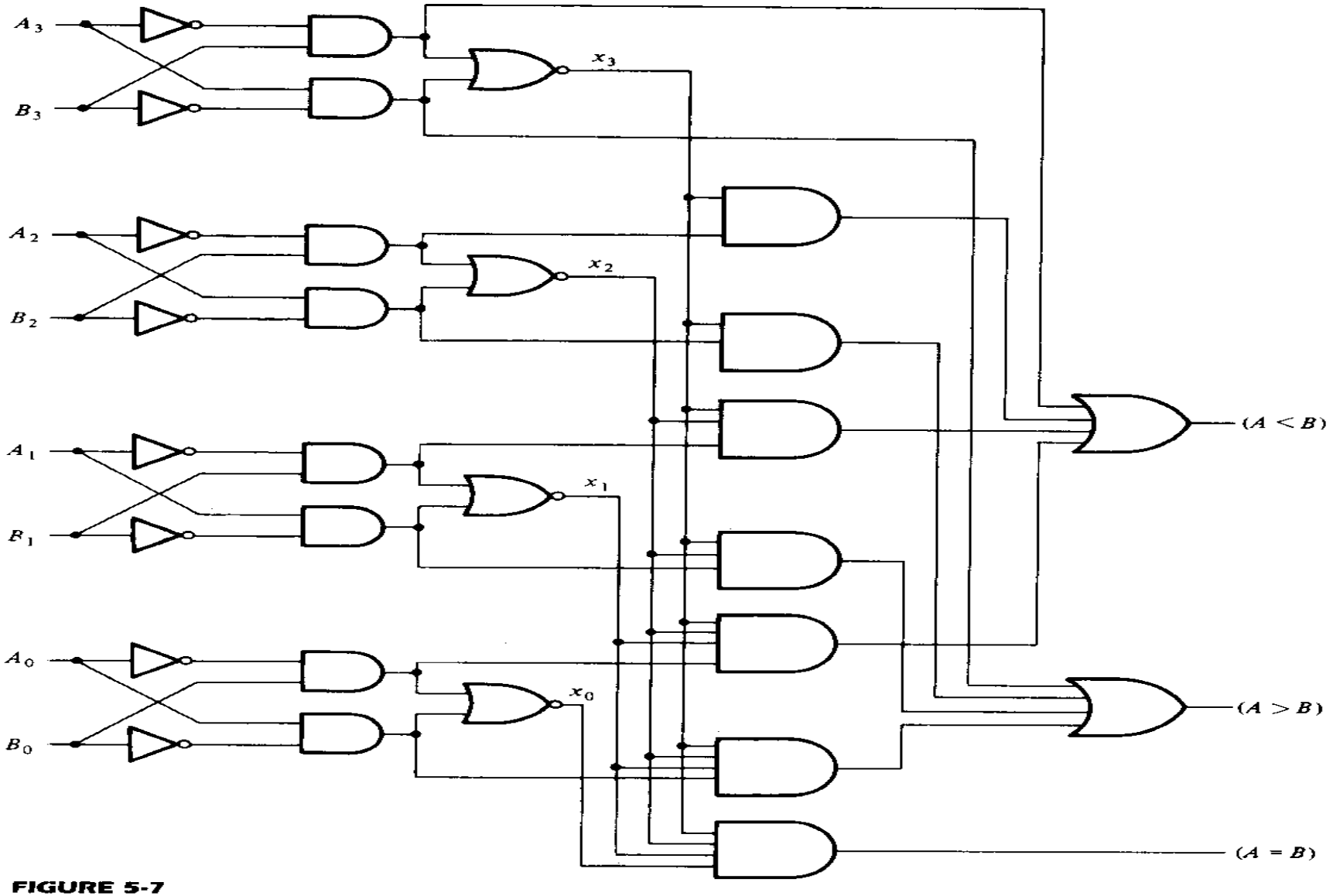


FIGURE 5-7
4-bit magnitude comparator

•The four x outputs are generated with equivalence (exclusive-NOR) circuits and applied to an AND gate to give the output binary variable (A = B).

•The other two outputs use the x variables to generate the Boolean functions listed before.

•This is a multilevel implementation and, as clearly seen, it has a regular pattern.

•The procedure for obtaining magnitude comparator circuit for binary numbers with more than four bits should be obvious from this example.

4-9 DECODERS AND ENCODERS

•Discrete quantities of information are represented in digital systems with binary codes.

•A binary code of n bits is capable of representing up to 2^n distinct elements of coded information.

•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 decoded information has unused or don't care combinations, the decoder output will have fewer than 2^n outputs.

•The decoders presented here are called n-to-m line decoders, where m ≤ 2^n.

•Their purpose is to generate 2^n (or fewer) minterms of n input variables.

•As an example, consider the 3-to -8 line decoder circuit of Fig. 5-8.

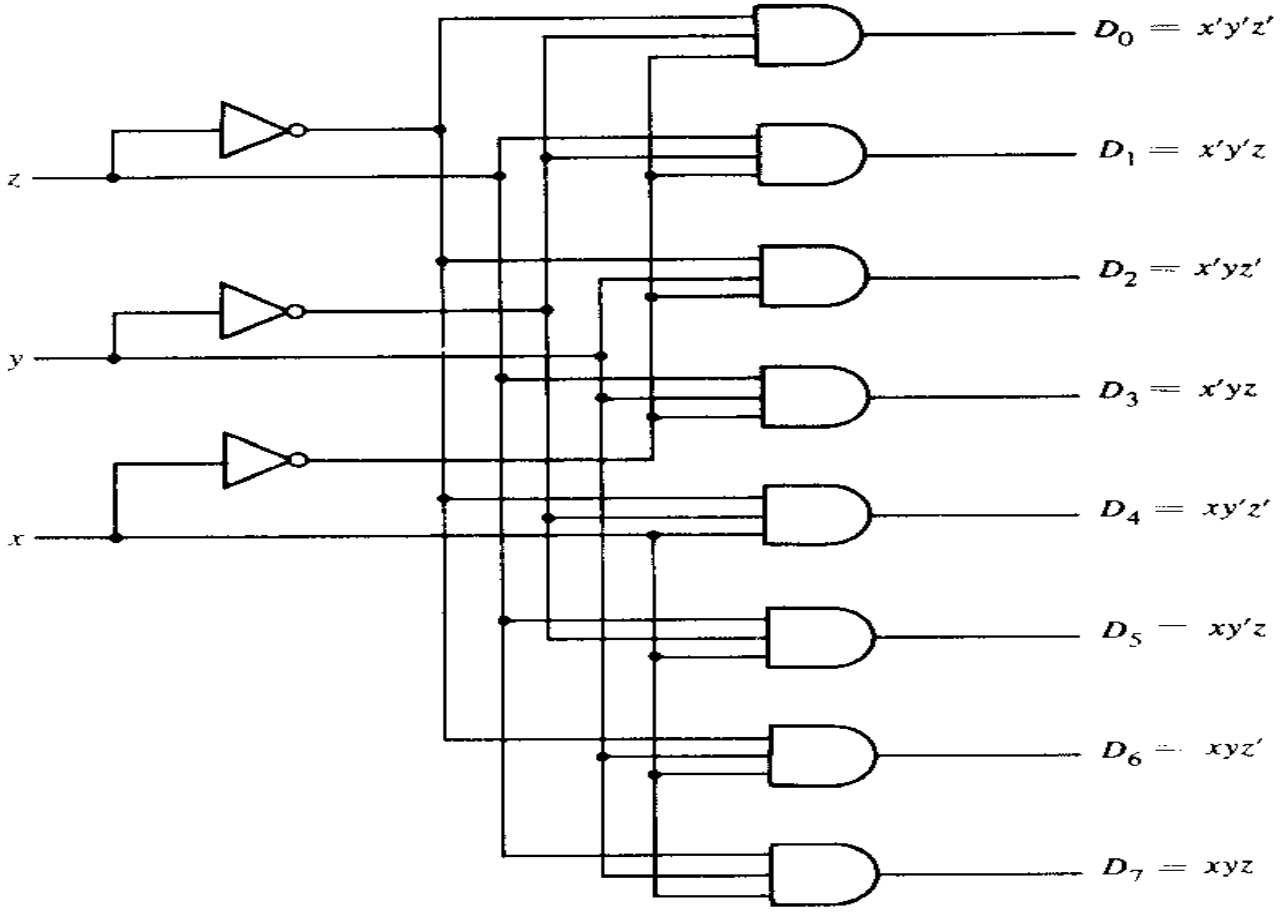


FIGURE 5-8
A 3-to-8 line decoder

•The three inputs are decoded into eight outputs, each output representing one of the minterms of the 3-input variables.

•The three inverters provide the complement of the inputs, and each one of the eight AND gates generates one of the minterms.

•A particular application of this decoder would be binary-to-octal conversion.

•The input variables may represent the binary number, and the outputs will then represent the eight digits in the octal number system.

•However, a 3-to-8 line decoder can be used for decoding any 3-bit code to provide eight outputs, one for each element of the code.

•The operation of the decoder may be further clarified from its input-output relationship, listed in Table 5-2.

TABLE 5-2
Truth Table of a 3-to-8-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

•Observe that in the output line, only one output is equal to 1 at a time.

•The output line whose value is equal to 1 represents the minterm equivalent of the binary number presently available in the input line.

•The output line whose value is equal to 1 also represents the decimal equivalent of the selected of the input values of x, y, and z.

•**Combinational Logic Implementation**

•A decoder provides the 2^n minterm of n input variables.

•Since any Boolean function can be expressed in sum of minterms canonical form, one can use a decoder to generate the minterms and an external OR gate to form the sum.

•In this way, any combinational circuit with n inputs and m outputs can be implemented with an n-to-2^n line decoder and m OR gates.

•The procedure for implementing a combinational circuit by means of a decoder and OR gates requires that the Boolean function for the circuit be expressed in sum of minterms.

•This form can be easily obtained from the truth table or by expanding the function to their sum of minterms.

•A decoder is then chosen that generates all the minterms of the n input variables.

•The inputs to each OR gate are selected from the decoder outputs according to the minterm list in each function.

Example 5-1

•Implement a full-adder circuit with a decoder and two OR gates.

•From the truth table of the full-adder, we obtain the function for this combinational circuit in sum of minterms:

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

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

•Since there are three inputs and a total of eight minterms, we need a 3-to-8 line decoder.

•The implementation is shown in Fig. 5-9.

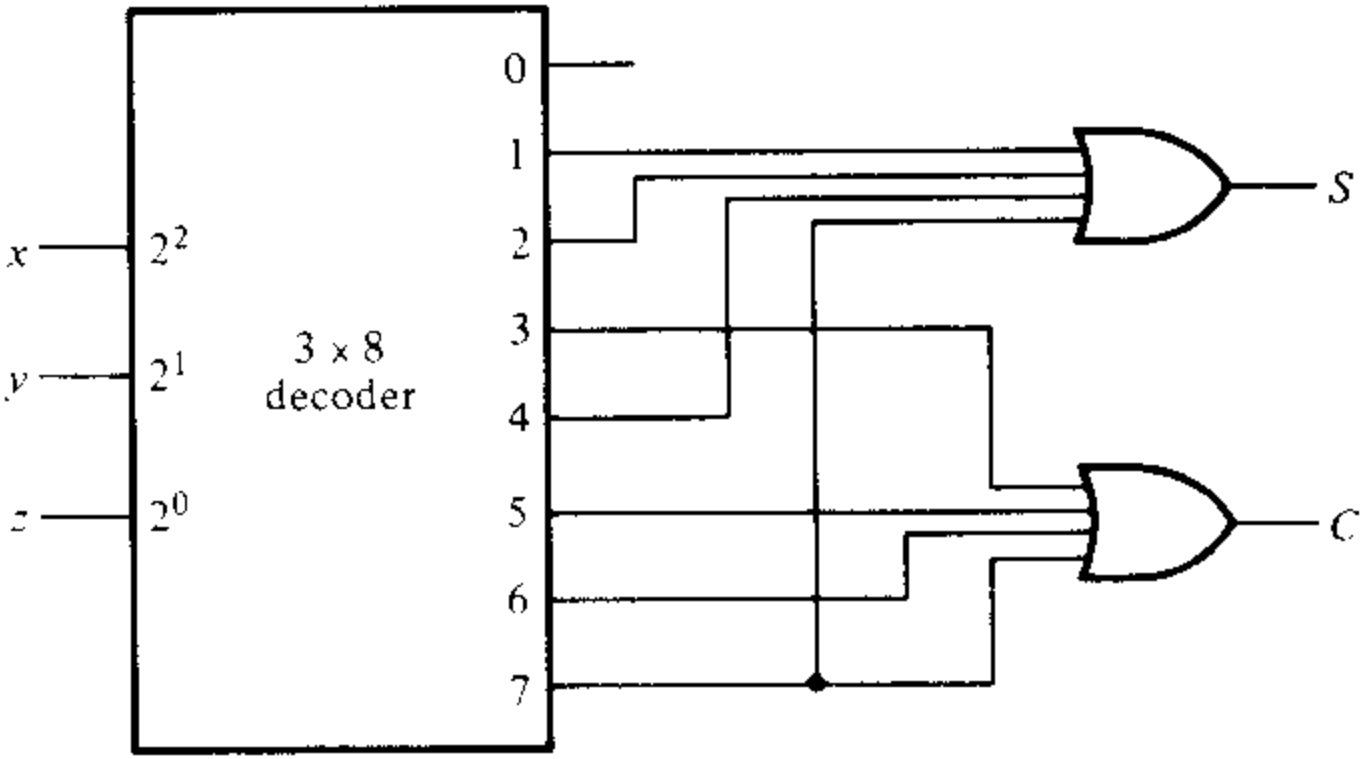


FIGURE 5-9
Implementation of a full-adder with a decoder

•The decoder generates the eight minterms for x, y, z.

•The OR gate for output S forms the sum of minterms 1, 2, 4 and 7.

•The OR gate for output C forms the sum of minterms 3, 5, 6, and 7.

•A function with a long list of minterms requires an OR gate with a large number of inputs.

•A function F having a list of k minterms can be expressed in its complemented form F’ with 2^n – k minterms.

•If the number of minterms in a function is greater than 2^n/2, then F’ can be expressed with fewer minterms than required for F.

•In such a case, it is advantageous to use a NOR gate to sum the minterms of F’.

•The output of the NOR gate will generate the normal output F.

•The decoder method can be used to implement any combinational circuit.

•However, its implementation must be compared with all other possible implementations to determine the best solution.

•In some cases, this method may provide the best implementation, especially if the combinational circuit has many outputs and if each output function (or its complement) is expressed with a small number of minterms.

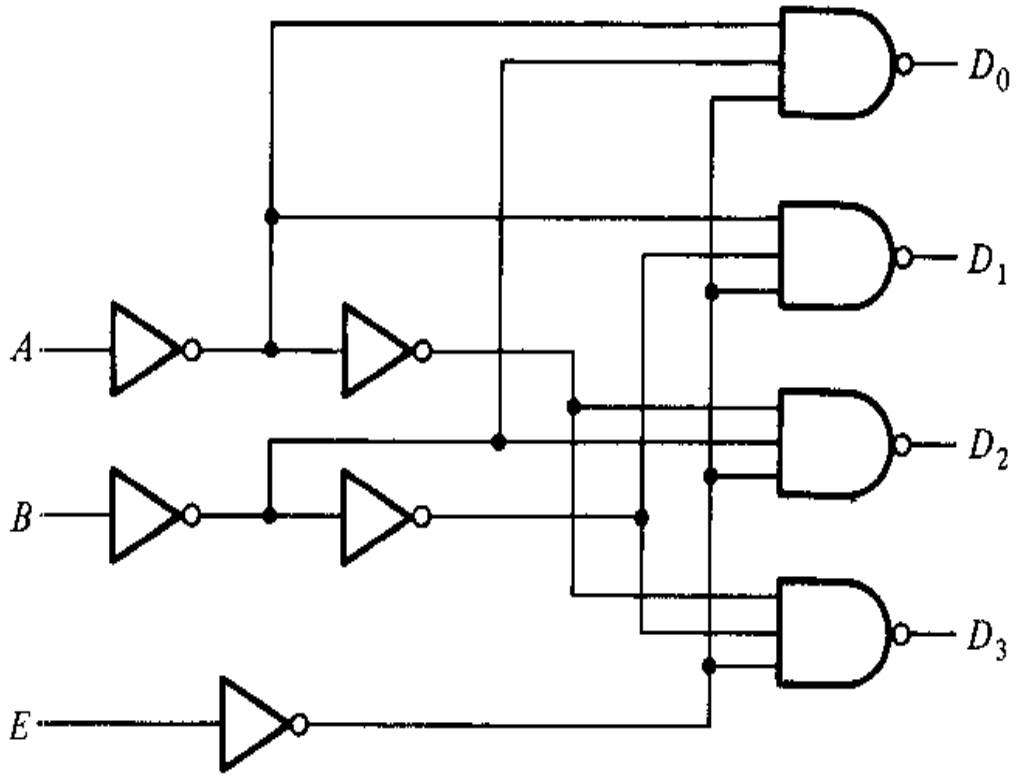
•**Demultiplexer**

•Some IC 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.

•Most, if not all, IC decoders include one or more enable inputs to control the circuit operation.

•A 2-to 4 line decoder with an enable input constructed with NAND gates in shown in Fig. 5-10.



(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

FIGURE 5-10

A 2-to-4-line decoder with enable (E) input

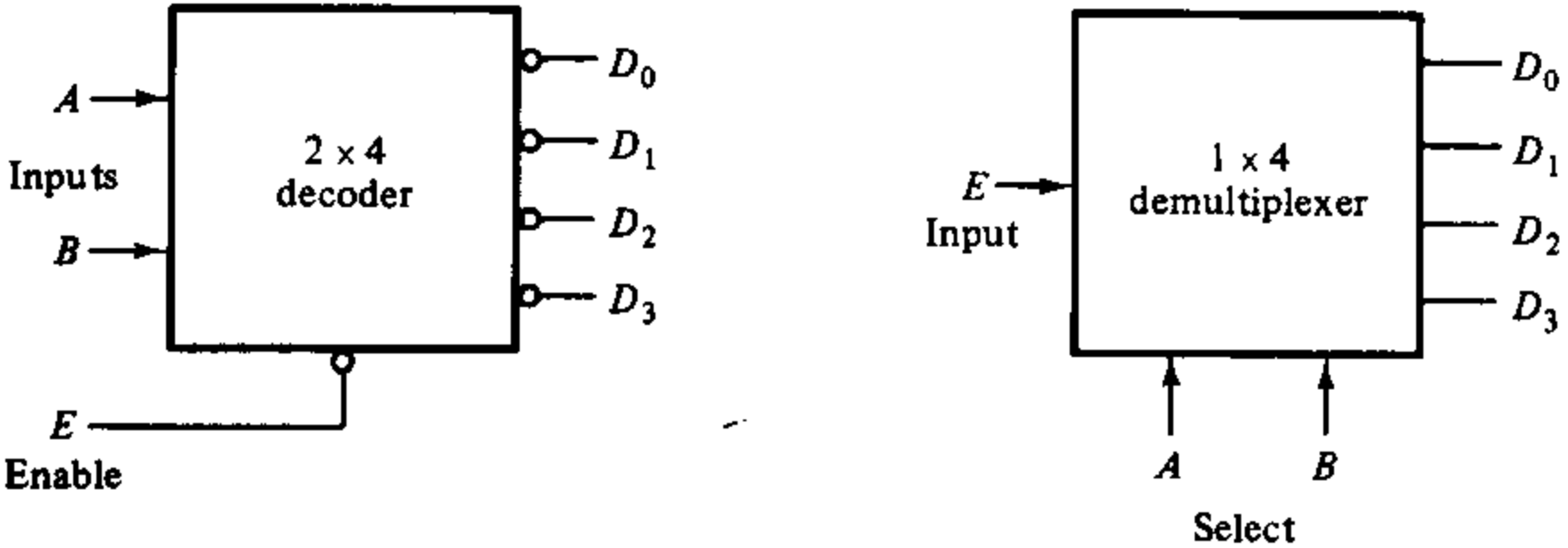
•A decoder with an enable input can function as a demultiplexer.

•A **demultiplexer** is a circuit that receives information on a single line and transmits this information on one of 2^n possible output lines.

•The selection of a specific output line is controlled by the bit values of n selection lines.

•The decoder of Fig. 5-19 can function as a demultiplexer if the E line is taken as a data input line and lines A and B are taken as the selection lines.

•This is shown in Fig. 5-11(b).



(a) Decoder with enable

(b) Demultiplexer

FIGURE 5-11

Block diagrams for the circuit of Fig. 5-10

•The single input variable E has a path to all four outputs, but the input information is directed to only one of the output lines, as specified by the binary value of the two selection line, A and B.

•This can be verified from the truth table of this circuit, shown in Fig. 5-10(b).

•For example, if the selection lines AB=10, output D2 will be the same as the input value E, while all other outputs are maintained at 1.

•It is the enable input that makes the circuit a demultiplexer; the decoder itself can use AND, NAND, or NOR gates.

•Decoder/demultiplexer circuits can be connected together to form a lager decoder circuit.

•Fig. 5-12 shown two 3x8 decoders with enable inputs connected to form a 4x16 decoder.

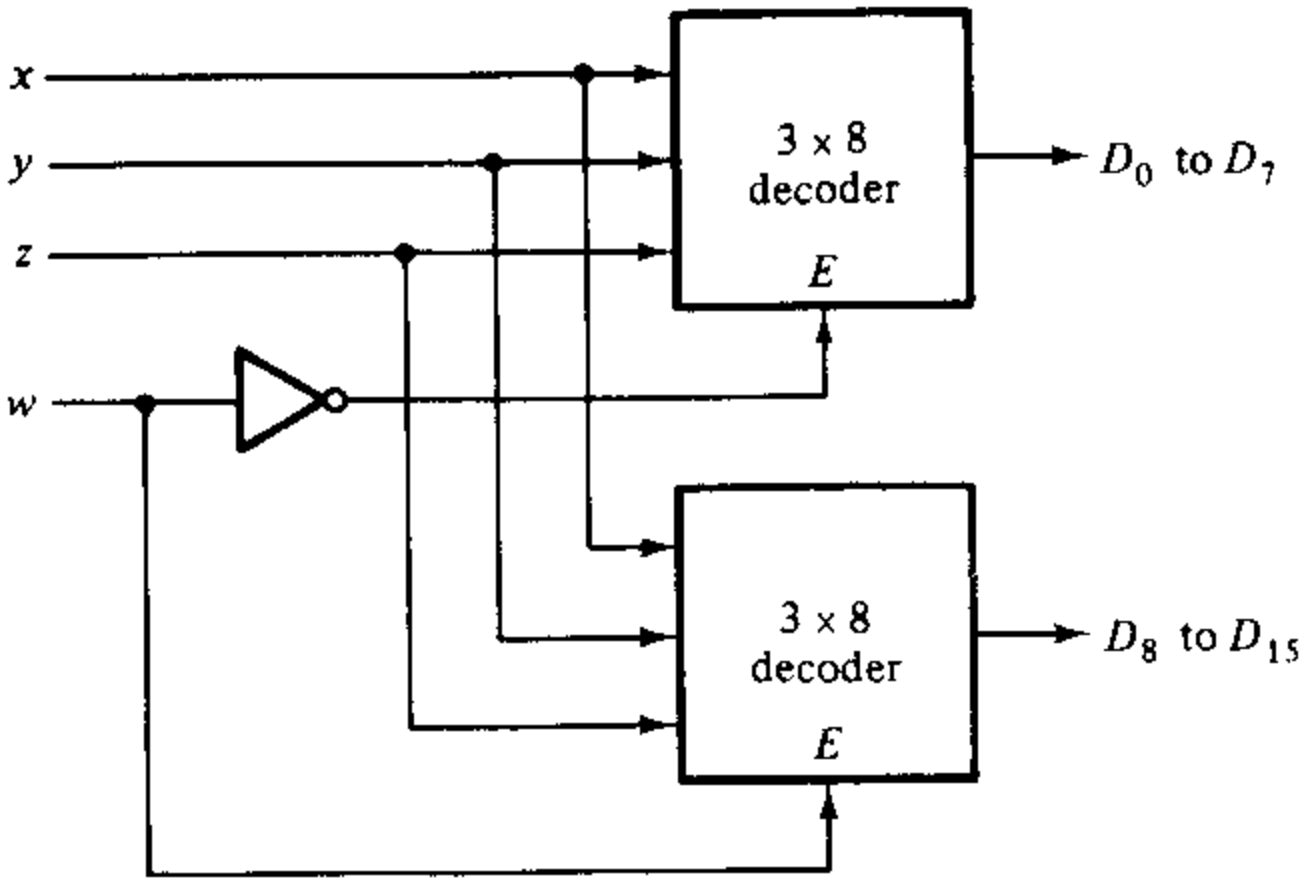


FIGURE 5-12

A 4 × 16 decoder constructed with two 3 × 8 decoders

•When w=0, the top decoder is enabled and the other is disabled.

•The bottom decoder outputs are all 0's, and top eight outputs generate minterms 0000 to 0111.

•When w=1, the enable conditions are reversed; the bottom decoder output generate minterms 1000 to 1111, while the outputs of the top decoder are all 0's.

•This example demonstrates the usefulness of enable inputs in ICs.

•In general, enable lines are a convenient feature for connecting two or more IC packages for the purpose of expanding the digital function into a similar function with more inputs and outputs.

Encoders

•An encoder is 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 line generate the binary code corresponding to the input value.

•As an example of an encoder is the octal-to-binary encoder whose truth table is given in Table 5-3.

•It has eight inputs, one for each of the octal digits, and three outputs that generate the corresponding binary number.

•It is assumed that only one input has a value of 1 at any given time; otherwise the circuit has no meaning.

TABLE 5-3
Truth Table of 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

•The encoder can be implemented with OR gates whose inputs are determined directly fro m the truth table.

•Output z is equal to 1 when the input octal digit is 1 or 3 or 5 or 7.

•Output y is 1 for octal digits 2, 3, 6, 7 and output x is 1 for digits 4,5,6 or 7.

•These conditions can be expressed by the following output Boolean functions:

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

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

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

•The encoder is implemented with three OR gates, as shown in Fig. 5-13

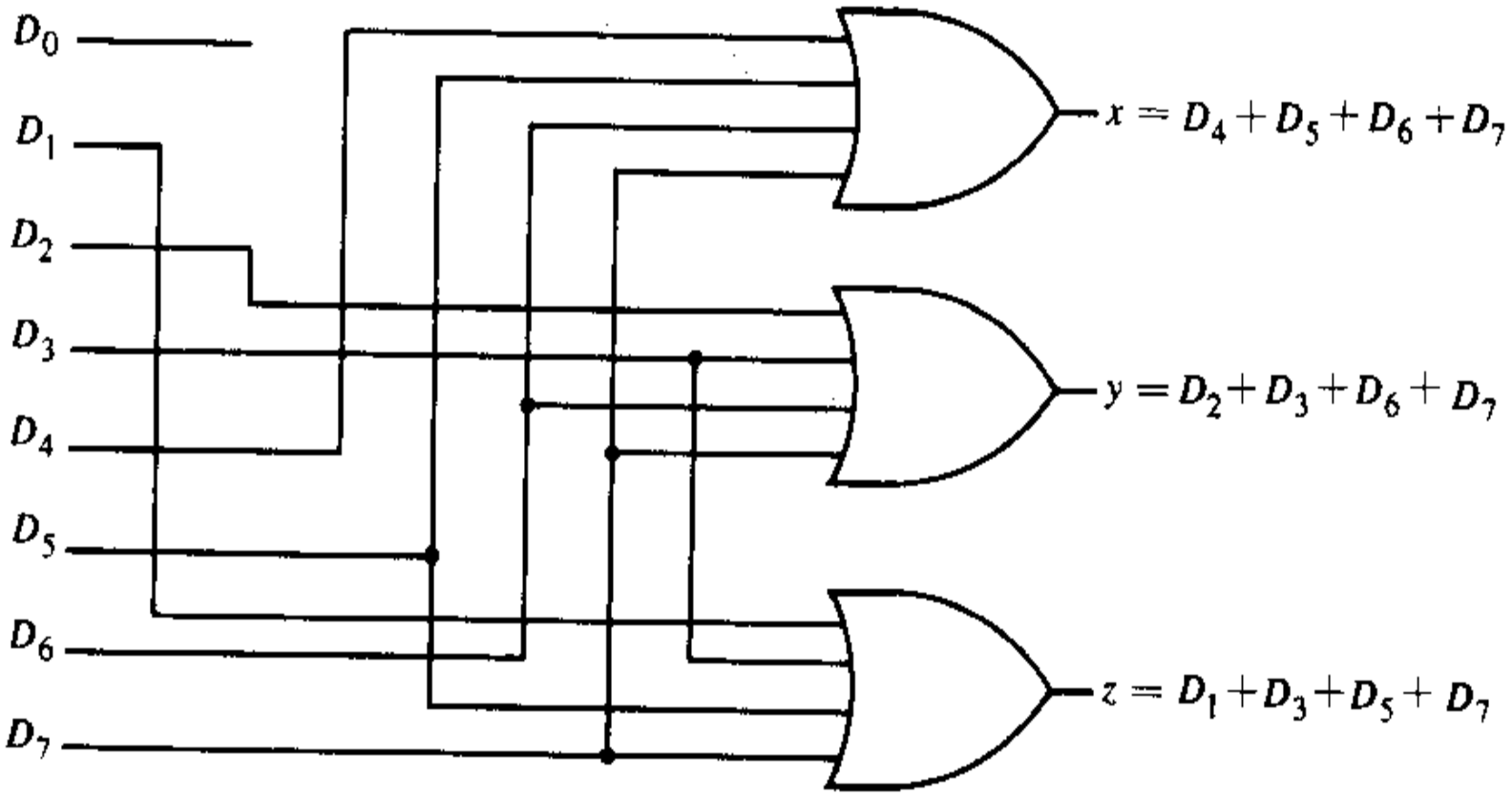


FIGURE 5-13
Octal-to-binary encoder

•The encoder defined in Table 5-3 has the limitation that only one input can be active at any given time.

•If two inputs are active simultaneously, the output produces an undefined combination.

•For example, if D3 and D6 are 1 simultaneously, the output of the encoder will be 111 because all three outputs are equal to 1.

•This does not represent binary 3 nor binary 6.

•To resolve this ambiguity, encoder circuits must establish a priority encoder to ensure that only one input is encoded.

•If we establish a higher priority for the inputs with higher subscript numbers, and if both D3 and D6 are 1 at the same time, the output will be 110 because D6 has higher priority than D3.

•Another ambiguity in the octal-to binary encoder is that an output will all 0's is generated when all the inputs are 0.

•The problem is that an output with all 0's is also, generated when D0 is equal to 1.

•This ambiguity can be resolved by providing an additional output that specifies the condition that none of the inputs are active.

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 to1 at the same time, the input having the highest priority will take precedence.

•The truth table of a four-input priority encoder is given in Table 5-4.

TABLE 5-4
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

•The X's are don't-care conditions that designate the fact that the binary value may be equal either to 0 to 1.

•Input D3 has the highest priority; so regardless of the values of the other inputs, when this input is 1, the output for xy is 11 (binary 3).

•D2 has the next priority level.

•The output is 10 if D2=1 provided that D3=0, regardless of the values of the other two lower-priority inputs.

•The output for D1 is generated only if higher-priority inputs are 0, and so on down the priority level.

•A valid output indicator, designated by V, is set to 1 only when one or more of the inputs are equal to1.

•If all inputs are 0, V is equal to 0, and the other two outputs of the circuit are not used.

•The maps for simplifying outputs x and y are shown in Fig. 5-14.

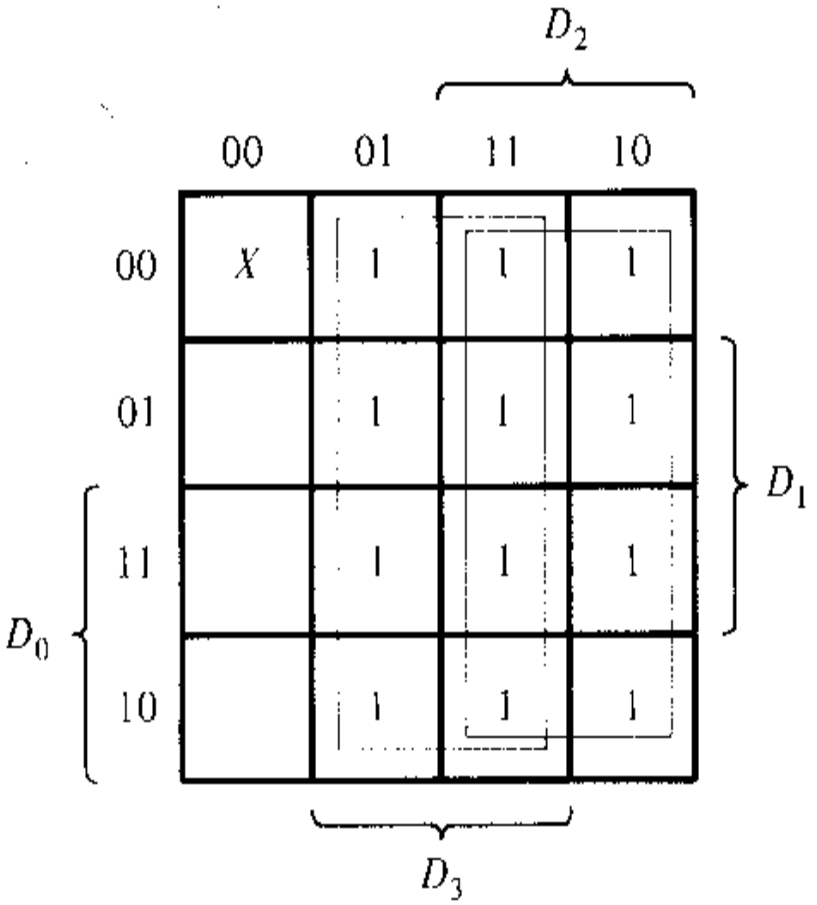
•The minterms for the two functions are derived from Table 5-4.

•Although the table has only five rows, when each don’t-care condition is replaced first by 0 and then by 1, we obtain all 16 possible input combinations.

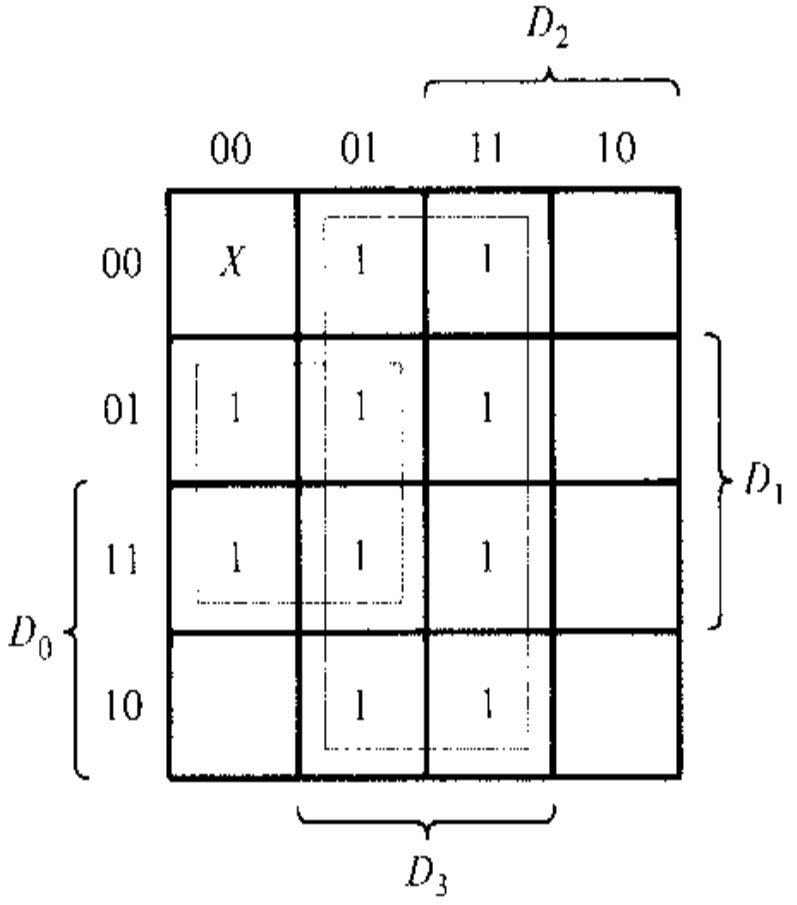
•For example, the third row in the table with X 100 represents minterms 0100 and 1100 since X can be assigned either 0 or 1.

•The simplified Boolean expression for the priority encoder are obtained from the maps.

•The condition for output V is an OR function of all input variables.



$x = D_2 + D_3$



$y = D_3 + D_1 D_2'$

FIGURE 5-14
Maps for a priority encoder

•The priority encoder is implemented in Fig 5-15 according to the following Boolean functions:

$$x = D_2 + D_3$$

$$y = D_3 + D_1 D_2'$$

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

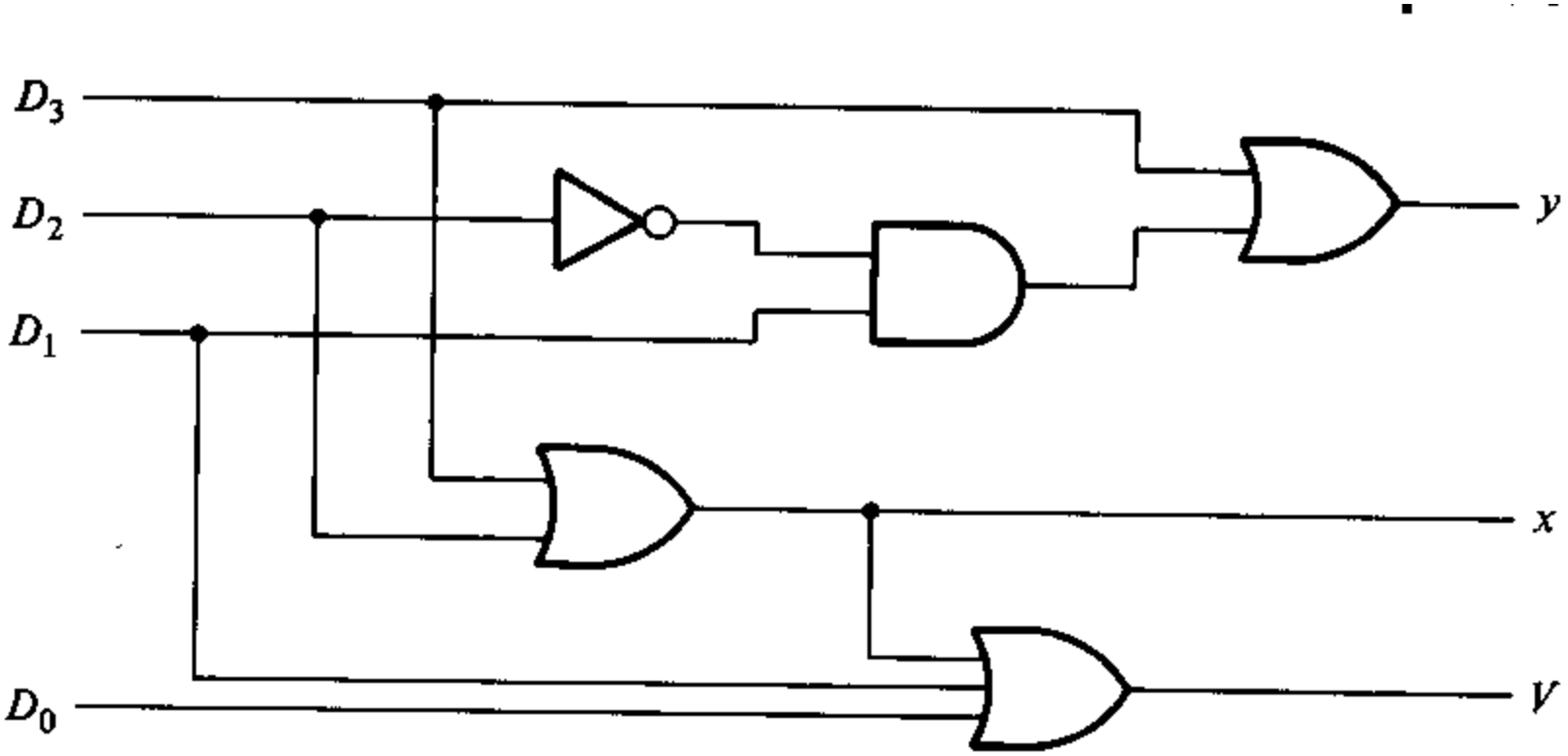


FIGURE 5-15
4-input priority encoder

4-10 MULTIPLEXERS

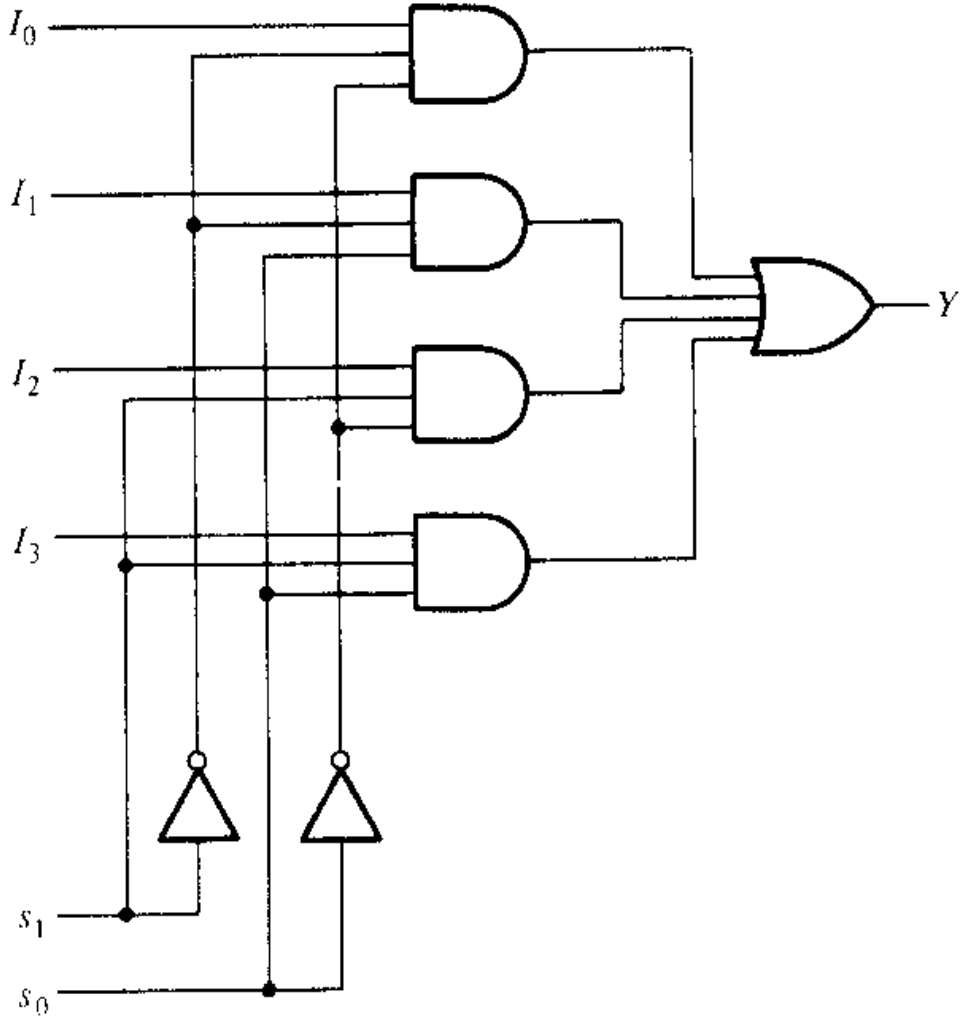
•Multiplexing means transmitting a large number of information units over a smaller number of channel or lines.

•A **digital 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.

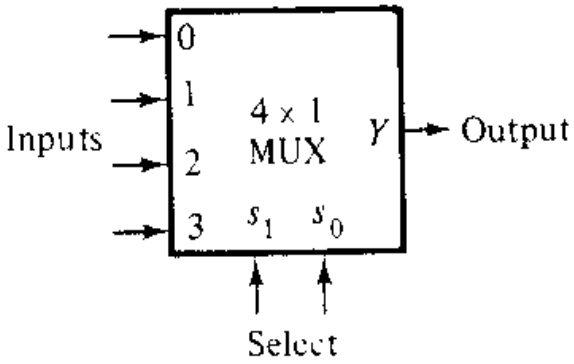
•A 4-to-1 line multiplexer is shown in Fig. 5-16.



(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



(c) Block diagram

FIGURE 5-16
A 4-to-1-line multiplexer

- Each of the four input lines, I_0 to I_3, is applied to one input of an AND gate.

- Selection lines s_1 and s_0 are decoded to select a particular AND gate.

- The function table, Fig 5-16(b), lists the input-to-output path for each possible bit combination of the selection lines.

- When this MSI function is used in the design of a digital system, it is represented in block diagram form, as shown in Fig. 5-16(c).

- To demonstrate the circuit operation, consider the case when $s_1s_0 = 10$.

- The AND gate associated with input I_2 has two of its inputs equal to 1 and the third output connected to I_2.

- The other three AND gates have at least one input equal to 0, which makes their outputs equal to 0.

•The OR gate output is now equal to the value of I_2, thus providing the path from the selected input to the output.

•A multiplexer is also called a **data selector**, since it selects one of the many inputs and steers the binary information to the output line.

•The AND gates and inverters in the multiplexer resemble a decoder circuit and, indeed, they decode the input-selection lines.

•In general, a 2^n-to-1-line multiplexer is constructed from an n-to-2^n decoder by adding it to 2^n input lines, one to each AND gate.

•The outputs of the AND gates are applied to a single OR gate to provide the 1-line output.

•The size of a multiplexer is specified by the number 2^n of its input lines and the single output line.

•It is then implied that it also contains n selection lines.

•A multiplexer is often abbreviated as MUX.

•As in decoders, multiplexer ICs may have an enable input to control the operation of the unit.

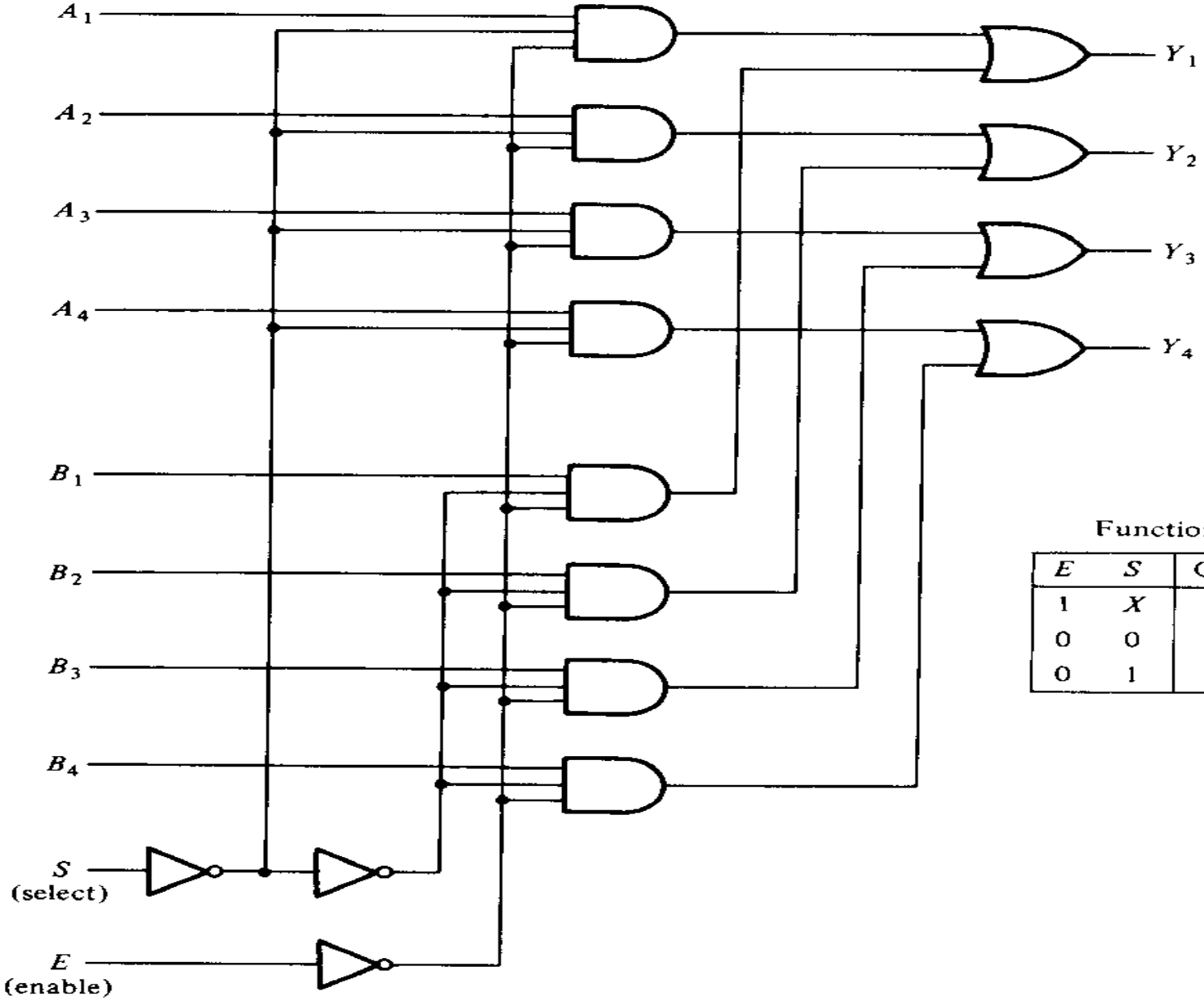
•When the enable input is in a given binary state, the outputs are disabled, and when it is in the other state (the enable state), the circuit functions as a normal multiplexer.

•The enable input (sometime called strobe) can be used to expand two or more multiplexer ICs to a digital multiplexer with a large number of inputs.

•In some cases, two or more multiplexers are enclosed within one IC package.

•The selection and enable inputs in multiple-unit ICs may be common to all multiplexers.

- As an illustration, a quadruple 2-to-1-line multiplexer IC is shown in Fig. 5-17.
- It has four multiplexers, each capable of selecting one of two input lines.
- Out Y_1 can be selected to be equal to either A_1 or B_1.
- Similarly, output Y_2 may have the value of A_2 or B_2, and so on.
- One input selection line, S, suffices to select one of two lines in all four multiplexers.
- The control input E enables the multiplexers in the 0 state and disables them in the 1 state.
- Although the circuit contains four multiplexers, we may think of it as a circuit that selects one in a pair of 4-input lines.



Function table

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

FIGURE 5-17
Quadruple 2-to-1-line multiplexer

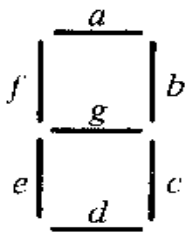
- As shown in the function table, the unit is selected when E=0.
- Then if S=0, the four A inputs have a path to the outputs.
- Otherwise, if S=1, the four B inputs are selected.
- The outputs have all 0's when E=1, regardless of the value of S.

Exercises

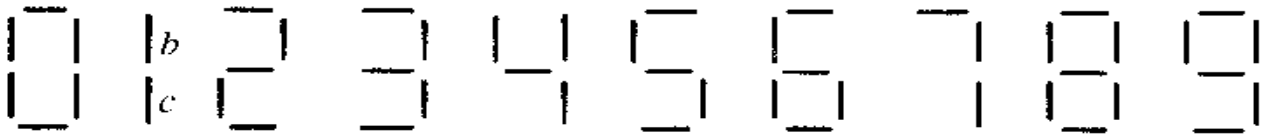
1. A BCD-to-seven –segment decoder is a combinational circuit that converts a decimal digit in BCD to an appropriate code for the selection of segments in a display indicator used for displaying the decimal digit in a familiar form. The seven outputs of the decoder (a, b, c, d, e, f, g) select the corresponding segments in the display, as shown in Fig P4-16(a). The numeric designation chosen to represent the decimal digit is shown in Fig. P4-16(b).

   Design the BCD-to-seven-segment display.

   The six invalid combinations should result in a blank display.



(a) Segment designation



(b) Numerical designation for display

2.A majority function is generated in a combinational circuit when the output is equal to 1 if the input variables have more 1's than 0's. The output is 0 otherwise. Design a three-input majority function and draw the logic diagram for the function.

3. Construct a 5 x 32 decoder with four 3 x 8 decoders/demultiplexers and a 2 x4 decoder. Use block diagrams.

4. Construct a 16 x1 multiplexer with two 8x1 and one 2 x1 multiplexers. Use block diagrams