Chapter 6
Functions of Combinational logic

Nouman M Durrani

BASIC ADDERS

- **Adder** or **summer** is a digital circuit that performs addition of numbers.
- In many computers and other kinds of processors, adders are used not only in the arithmetic logic unit(s), but also in other parts of the processor, where they are used to calculate addresses, table indices, and similar.

The Half-Adder

The half-adder accepts two binary digits on its inputs and produces two binary digits on its outputs, a sum bit and a carry bit.

A half-adder is represented by the logic symbol in Figure 6–1.

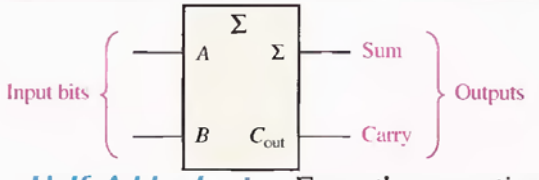


FIGURE 6–1
Logic symbol for a half-adder.

Half-Adder Logic From the operation of the half-adder as stated in Table 6–1, expressions can be derived for the sum and the output carry as functions of the inputs. Notice that the output carry (C_{out}) is a 1 only when both A and B are 1s; therefore, C_{out} can be expressed as the AND of the input variables.

$$C_{out} = AB$$ Equation 6–1

TABLE 6–1
Half-adder truth table.

A	B	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Σ = sum
C_{out} = output carry
A and B = input variables (operands)

- The sum output (Σ) is a 1 only if the input variables, A and B, are not equal.
- The sum can therefore be expressed as the exclusive-OR of the input variables.

$$\Sigma = A \oplus B$$ Equation 6–2

- From Equations 6-1 and 6-2, the logic implementation required for the half-adder function can be developed.

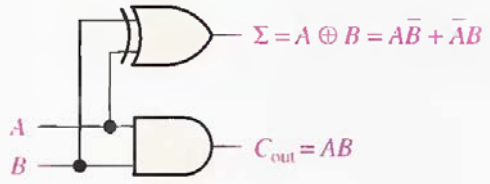


FIGURE 6–2
Half-adder logic diagram.

The Full-Adder

The full-adder accepts two input bits and an input carry and generates a sum output and an output carry.

The basic difference between a full-adder and a half-adder is that the full-adder accepts an input carry. A logic symbol for a full-adder is shown in Figure 6–3, and the truth table in Table 6–2 shows the operation of a full-adder.

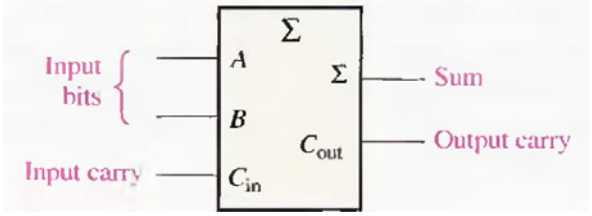


FIGURE 6–3
Logic symbol for a full-adder.

TABLE 6–2
Full-adder truth table.

A	B	C_{in}	C_{out}	Σ
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

C_{in} = input carry, sometimes designated as *CI*

C_{out} = output carry, sometimes designated as *CO*

Σ = sum

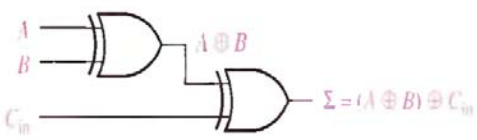
A and B = input variables (operands)

Full-Adder Logic

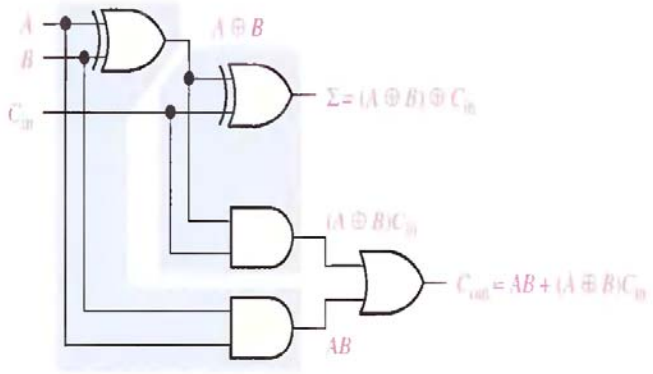
The full-adder must add the two input bits and the input carry.

- To implement the full-adder sum function, two 2-input exclusive-OR gates can be used.
- The first must generate the term, $A \oplus B$, and the second has as its inputs the output of the first XOR gate and the input carry. As illustrated in Figure 6.4(a).

Equation 6–3 $\Sigma = (A \oplus B) \oplus C_{in}$



(a) Logic required to form the sum of three bits

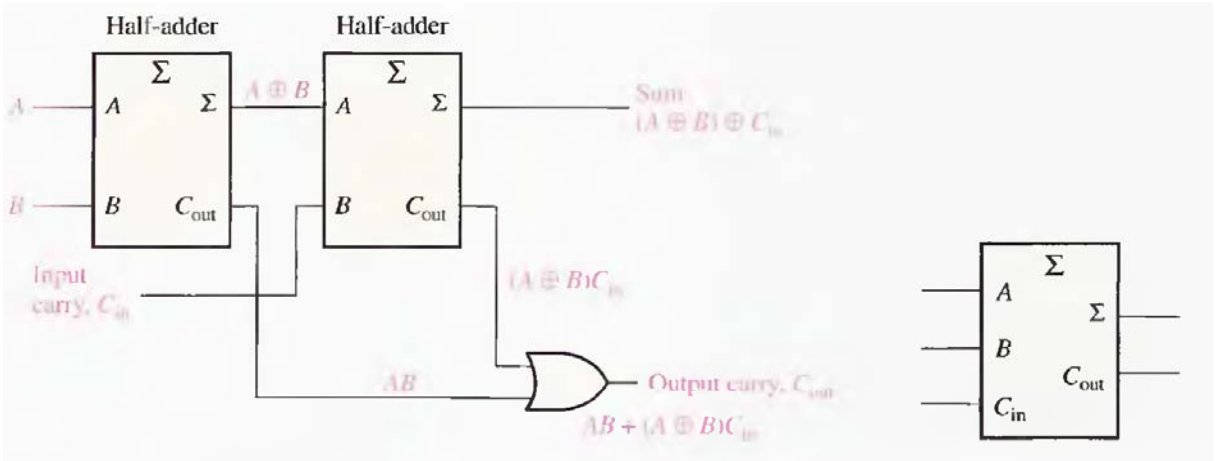


(b) Complete logic circuit for a full-adder (each half-adder is enclosed

- The output carry is a 1 when both inputs to the first XOR gate are 1s or when both inputs to the second XOR gate are 1s.

- The output carry of the full-adder is therefore produced by the inputs A ANDed with B and $A \oplus B$ ANDed with Cin.
- These two terms are ORed, as expressed in Equation 6.4

Equation 6–4 $$C_{out} = AB + (A \oplus B)C_{in}$$



(a) Arrangement of two half-adders to form a full-adder

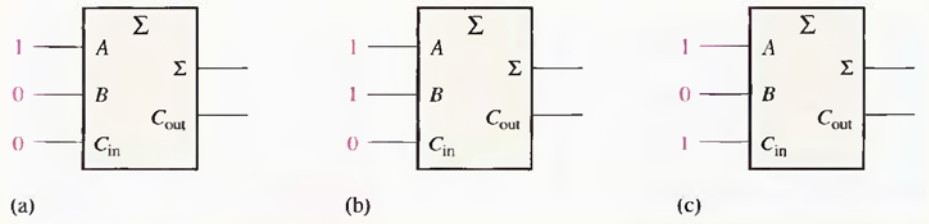
(b) Full-adder logic symbol

FIGURE 6–5

Full-adder implemented with half-adders.

EXAMPLE 6–1

For each of the three full-adders in Figure 6–6, determine the outputs for the inputs shown.



▲ FIGURE 6–6

Solution

(a) The input bits are $A = 1$, $B = 0$, and $C_{in} = 0$.

$$1 + 0 + 0 = 1 \text{ with no carry}$$

Therefore, $\Sigma = 1$ and $C_{out} = 0$.

(b) The input bits are $A = 1$, $B = 1$, and $C_{in} = 0$.

$$1 + 1 + 0 = 0 \text{ with a carry of 1}$$

Therefore, $\Sigma = 0$ and $C_{out} = 1$.

(c) The input bits are $A = 1$, $B = 0$, and $C_{in} = 1$.

$$1 + 0 + 1 = 0 \text{ with a carry of 1}$$

Therefore, $\Sigma = 0$ and $C_{out} = 1$.

Related Problem* What are the full-adder outputs for $A = 1$, $B = 1$, and $C_{in} = 1$?

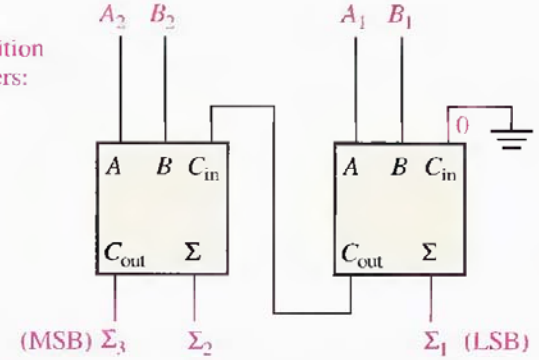
PARALLEL BINARY ADDERS

Two or more full-adders are connected to form parallel binary adders.

To add two binary numbers, a full-adder is required for each bit in the numbers. So for 2-bit numbers, two adders are needed; for 4-bit numbers, four adders are used; and so on. The carry output of each adder is connected to the carry input of the next higher-order adder, as shown in Figure 6–7 for a 2-bit adder. Notice that either a half-adder can be used for the least significant position or the carry input of a full-adder can be made 0 (grounded) because there is no carry input to the least significant bit position.

General format, addition of two 2-bit numbers:

$$\begin{array}{r} A_2A_1 \\ +\,B_2B_1 \\ \hline \Sigma_3\Sigma_2\Sigma_1 \end{array}$$



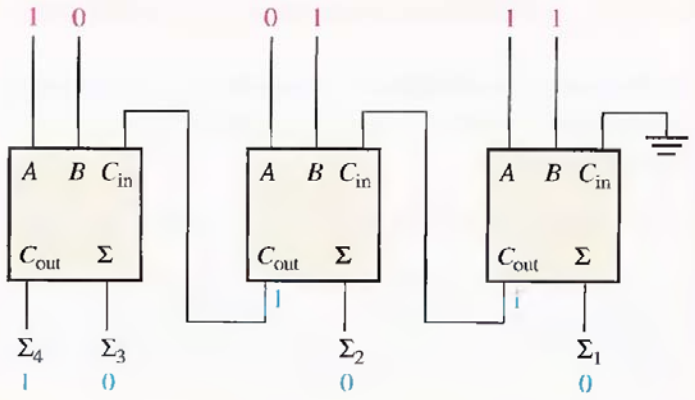
▶ FIGURE 6–7

Block diagram of a basic 2-bit parallel adder using two full-adders.

In Figure 6–7 the least significant bits (LSB) of the two numbers are represented by A_1 and B_1. The next higher-order bits are represented by A_2 and B_2. The three sum bits are Σ_1, Σ_2, and Σ_3. Notice that the output carry from the left-most full-adder becomes the most significant bit (MSB) in the sum, Σ_3.

Determine the sum generated by the 3-bit parallel adder in Figure 6–8 and show the intermediate carries when the binary numbers 101 and 011 are being added.

FIGURE 6–8

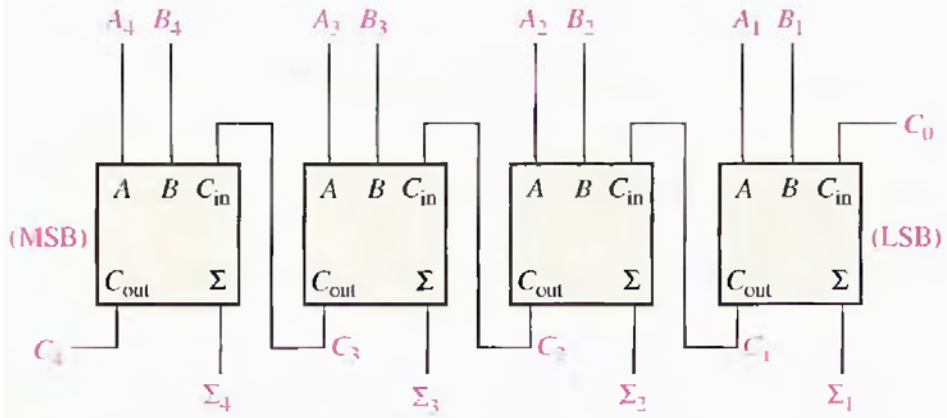


Solution The LSBs of the two numbers are added in the right-most full-adder. The sum bits and the intermediate carries are indicated in blue in Figure 6–8.

Related Problem What are the sum outputs when 111 and 101 are added by the 3-bit parallel adder?

Four-Bit Parallel Adders

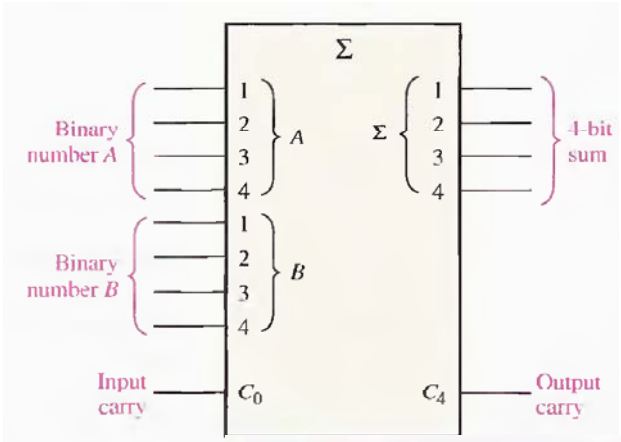
A group of four bits is called a **nibble.** A basic 4-bit parallel adder is implemented with four full-adder stages as shown in Figure 6–9. Again, the LSBs (A_1 and B_1) in each number being added go into the right-most full-adder; the higher-order bits are applied as shown to the successively higher-order adders, with the MSBs (A_4 and B_4) in each number being applied to the left-most full-adder. The carry output of each adder is connected to the carry input of the next higher-order adder as indicated. These are called *internal carries.*



(a) Block diagram

FIGURE 6–9

A 4-bit parallel adder.



(b) Logic symbol

FIGURE 6–9

A 4-bit parallel adder.

TABLE 6–3

Truth table for each stage of a 4-bit parallel adder.

C_{n-1}	A_n	B_n	Σ_n	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

EXAMPLE 6–3

Use the 4-bit parallel adder truth table (Table 6–3) to find the sum and output carry for the addition of the following two 4-bit numbers if the input carry (C_{n-1}) is 0:

Solution

$$A_4A_3A_2A_1 = 1100 \quad \text{and} \quad B_4B_3B_2B_1 = 1100$$

For $n = 1$: $A_1 = 0$, $B_1 = 0$, and $C_{n-1} = 0$. From the 1st row of the table,

$$\Sigma_1 = \mathbf{0} \quad \text{and} \quad C_1 = 0$$

For $n = 2$: $A_2 = 0$, $B_2 = 0$, and $C_{n-1} = 0$. From the 1st row of the table,

$$\Sigma_2 = \mathbf{0} \quad \text{and} \quad C_2 = 0$$

For $n = 3$: $A_3 = 1$, $B_3 = 1$, and $C_{n-1} = 0$. From the 4th row of the table,

$$\Sigma_3 = \mathbf{0} \quad \text{and} \quad C_3 = 1$$

For $n = 4$: $A_4 = 1$, $B_4 = 1$, and $C_{n-1} = 1$. From the last row of the table,

$$\Sigma_4 = \mathbf{1} \quad \text{and} \quad C_4 = \mathbf{1}$$

C_4 becomes the output carry; the sum of 1100 and 1100 is 11000.

Related Problem

Use the truth table (Table 6–3) to find the result of adding the binary numbers 1011 and 1010.

Show how two 74LS283 adders can be connected to form an 8-bit parallel adder. Show output bits for the following 8-bit input numbers:

$$A_8A_7A_6A_5A_4A_3A_2A_1 = 10111001 \quad \text{and} \quad B_8B_7B_6B_5B_4B_3B_2B_1 = 10011110$$

Solution

Two 74LS283 4-bit parallel adders are used to implement the 8-bit adder. The only connection between the two 74LS283s is the carry output (pin 9) of the low-order adder to the carry input (pin 7) of the high-order adder, as shown in Figure 6–13. Pin 7 of the low-order adder is grounded (no carry input).

The sum of the two 8-bit numbers is

$$\Sigma_9\Sigma_8\Sigma_7\Sigma_6\Sigma_5\Sigma_4\Sigma_3\Sigma_2\Sigma_1 = 101010111$$

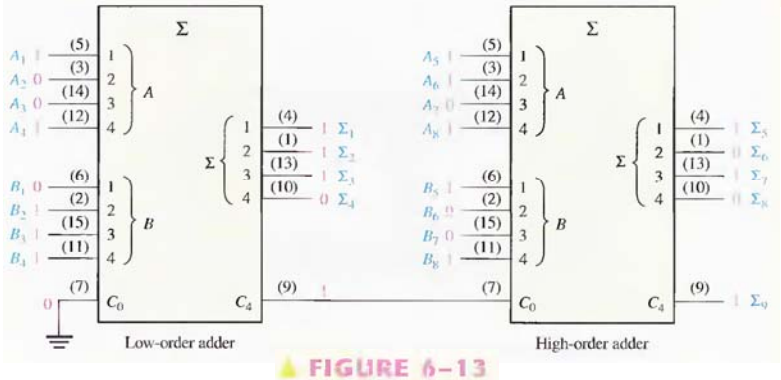


FIGURE 6–13

Two 74LS283 adders connected as an 8-bit parallel adder (pin numbers are in parentheses).

An Application

An example of full-adder and parallel adder application is a simple voting system that can be used to simultaneously provide the number of "yes" votes and the number of "no" votes.

In its simplest form, the system includes a switch for "yes" or "no" selection at each position in the assembly and a digital display for the number of yes votes and one for the number of no votes. The basic system is shown in Figure 6–14 for a 6-position setup, but it can be expanded to any number of positions with additional 6-position modules and additional parallel adder and display circuits.

The two higher-order inputs of the parallel adder are connected to ground (0), because there is never a case where the binary input exceeds 0011 (decimal 3).

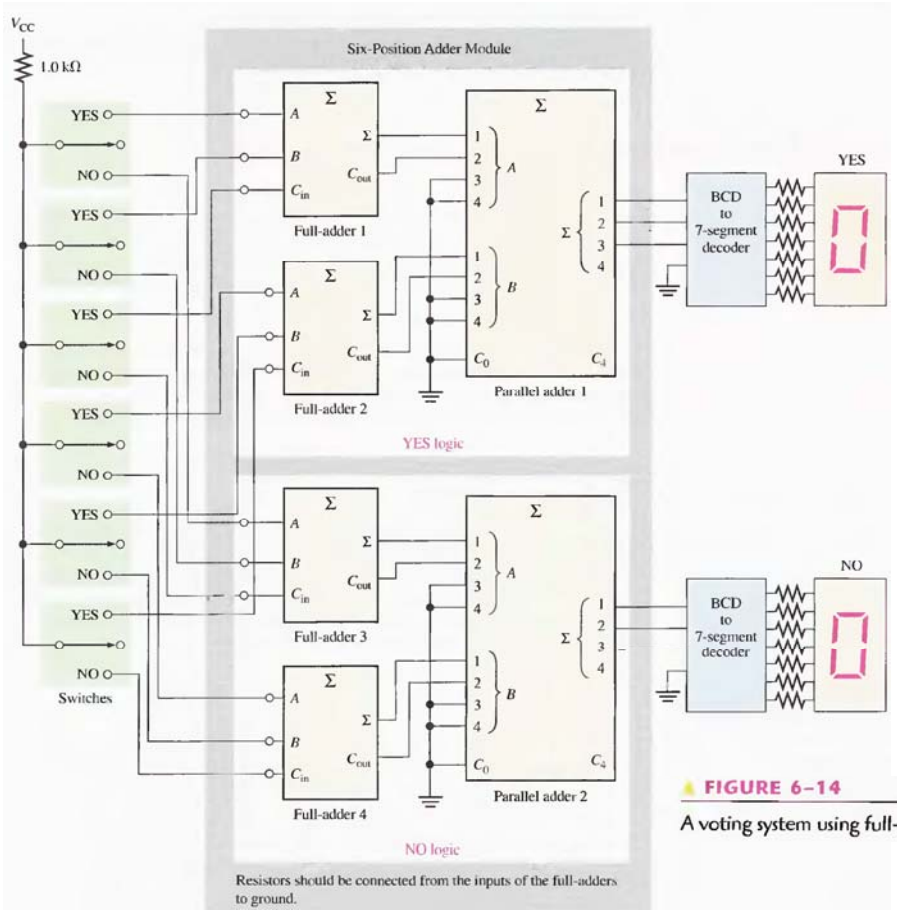


FIGURE 6–14

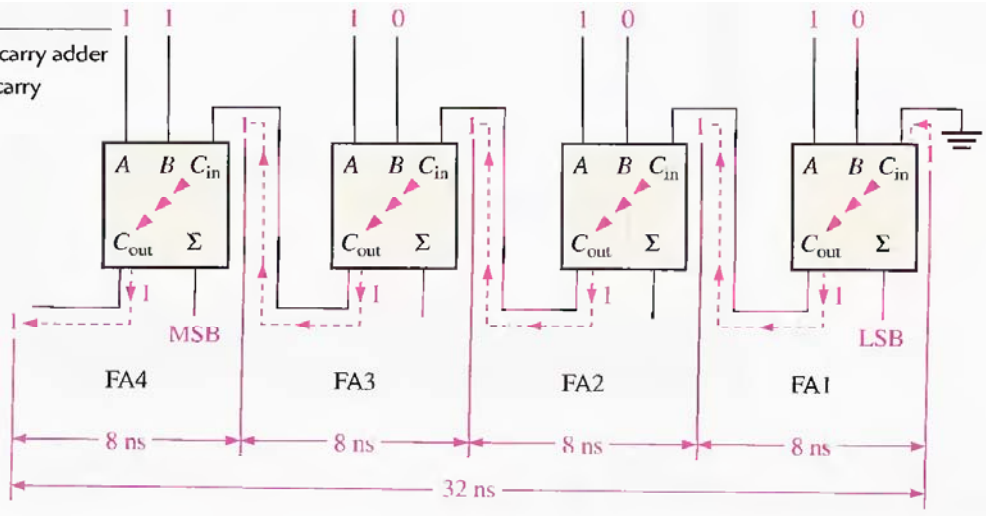
A voting system using full-adders and parallel binary adders.

The Ripple Carry Adder

A **ripple carry** adder is one in which the carry output of each full-adder is connected to the carry input of the next higher-order stage (a stage is one full-adder). The sum and the output carry of any stage cannot be produced until the input carry occurs; this causes a time delay in the addition process, as illustrated in Figure 6–15. The carry propagation delay for each full-adder is the time from the application of the input carry until the output carry occurs, assuming that the *A* and *B* inputs are already present.

FIGURE 6–15

A 4-bit parallel ripple carry adder showing "worst-case" carry propagation delays.



- Full-adder 1 (FA 1) cannot produce a potential output carry until an input carry is applied.
- Full-adder 2 (FA2) cannot produce a potential output carry until full-adder 1 produces an output carry.
- Full-adder 3 (FA3) cannot produce a potential output carry until an output carry is produced by FA 1 followed by an output carry from FA2, and so on.
- In Figure 6-15, the input carry to the least significant stage has to ripple through all the adders before a final sum is produced.
- The total delay can vary, depending on the carry bit produced by each full-adder.
- If two numbers are added such that no carries (0) occur between stages, the addition time is simply the propagation time through a single full-adder from the application of the data bits on the inputs to the occurrence of a sum output.

The look-Ahead Carry Adder

The look-ahead carry adder anticipates the output carry of each stage, and based on the inputs, produces the output carry by either carry generation or carry propagation.

Carry generation occurs when an output carry is produced (generated) internally by the full-adder. A carry is generated only when both input bits are 1s. The generated carry, C_g, is expressed as the AND function of the two input bits, A and B.

$$C_g = AB$$ Equation 6–5

Carry propagation occurs when the input carry is rippled to become the output carry. An input carry may be propagated by the full-adder when either or both of the input bits are 1s. The propagated carry, C_p, is expressed as the OR function of the input bits.

$$C_p = A + B$$ Equation 6–6

The conditions for carry generation and carry propagation are illustrated in Figure 6–16. The three arrowheads symbolize ripple (propagation).

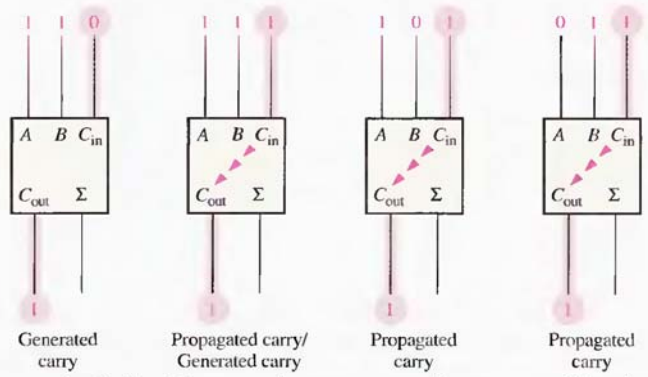
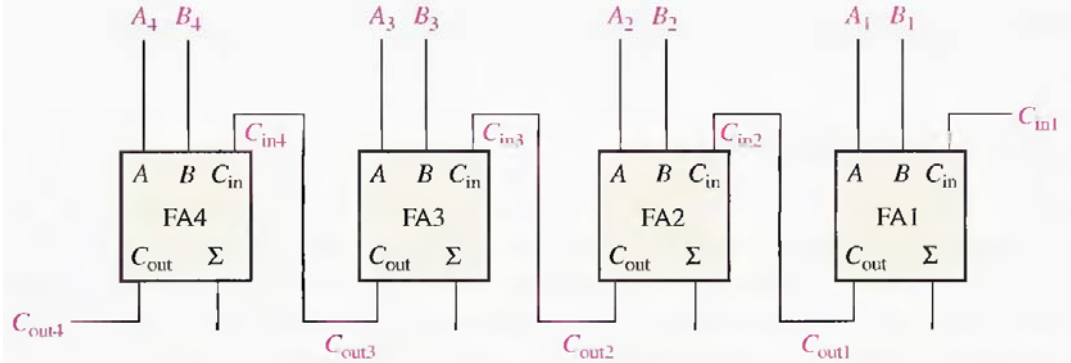


FIGURE 6–16
Illustration of conditions for carry generation and carry propagation.

The output carry of a full-adder can be expressed in terms of both the generated carry (C_g) and the propagated carry (C_p). The output carry (C_{out}) is a 1 if the generated carry is a 1 OR if the propagated carry is a 1 AND the input carry (C_{in}) is a 1. In other words, we get an output carry of 1 if it is generated by the full-adder ($A = 1$ AND $B = 1$) or if the adder propagates the input carry ($A = 1$ OR $B = 1$) AND $C_{in} = 1$. This relationship is expressed as

$$C_{out} = C_g + C_p C_{in}$$ Equation 6–7



Full-adder 4	Full-adder 3	Full-adder 2	Full-adder 1
$C_{g4} = A_4B_4$	$C_{g3} = A_3B_3$	$C_{g2} = A_2B_2$	$C_{g1} = A_1B_1$
$C_{p4} = A_4 + B_4$	$C_{p3} = A_3 + B_3$	$C_{p2} = A_2 + B_2$	$C_{p1} = A_1 + B_1$

Based on this analysis, we can now develop expressions for the output carry, C_{out}, of each full-adder stage for the 4-bit example.

Full-adder 1:

$$C_{out1} = C_{g1} + C_{p1}C_{in1}$$

Full-adder 2:

$$C_{in2} = C_{out1}$$
$$C_{out2} = C_{g2} + C_{p2}C_{in2} = C_{g2} + C_{p2}C_{out1} = C_{g2} + C_{p2}(C_{g1} + C_{p1}C_{in1})$$
$$= C_{g2} + C_{p2}C_{g1} + C_{p2}C_{p1}C_{in1}$$

Full-adder 3:

$$C_{in3} = C_{out2}$$
$$C_{out3} = C_{g3} + C_{p3}C_{in3} = C_{g3} + C_{p3}C_{out2} = C_{g3} + C_{p3}(C_{g2} + C_{p2}C_{g1} + C_{p2}C_{p1}C_{in1})$$
$$= C_{g3} + C_{p3}C_{g2} + C_{p3}C_{p2}C_{g1} + C_{p3}C_{p2}C_{p1}C_{in1}$$

Full-adder 4:

$$C_{in4} = C_{out3}$$
$$C_{out4} = C_{g4} + C_{p4}C_{in4} = C_{g4} + C_{p4}C_{out3}$$
$$= C_{g4} + C_{p4}(C_{g3} + C_{p3}C_{g2} + C_{p3}C_{p2}C_{g1} + C_{p3}C_{p2}C_{p1}C_{in1})$$
$$= C_{g4} + C_{p4}C_{g3} + C_{p4}C_{p3}C_{g2} + C_{p4}C_{p3}C_{p2}C_{g1} + C_{p4}C_{p3}C_{p2}C_{p1}C_{in1}$$

Notice that in each of these expressions, the output carry for each full-adder stage is dependent only on the initial input carry (C_{in1}), the C_g and C_p functions of that stage, and the C_g and C_p functions of the preceding stages. Since each of the C_g and C_p functions can be expressed in terms of the A and B inputs to the full-adders, all the output carries are immediately available (except for gate delays), and you do not have to wait for a carry to ripple through all the stages before a final result is achieved. Thus, the look-ahead carry technique speeds up the addition process.

The C_{out} equations are implemented with logic gates and connected to the full-adders to create a 4-bit look-ahead carry adder, as shown in Figure 6–18.

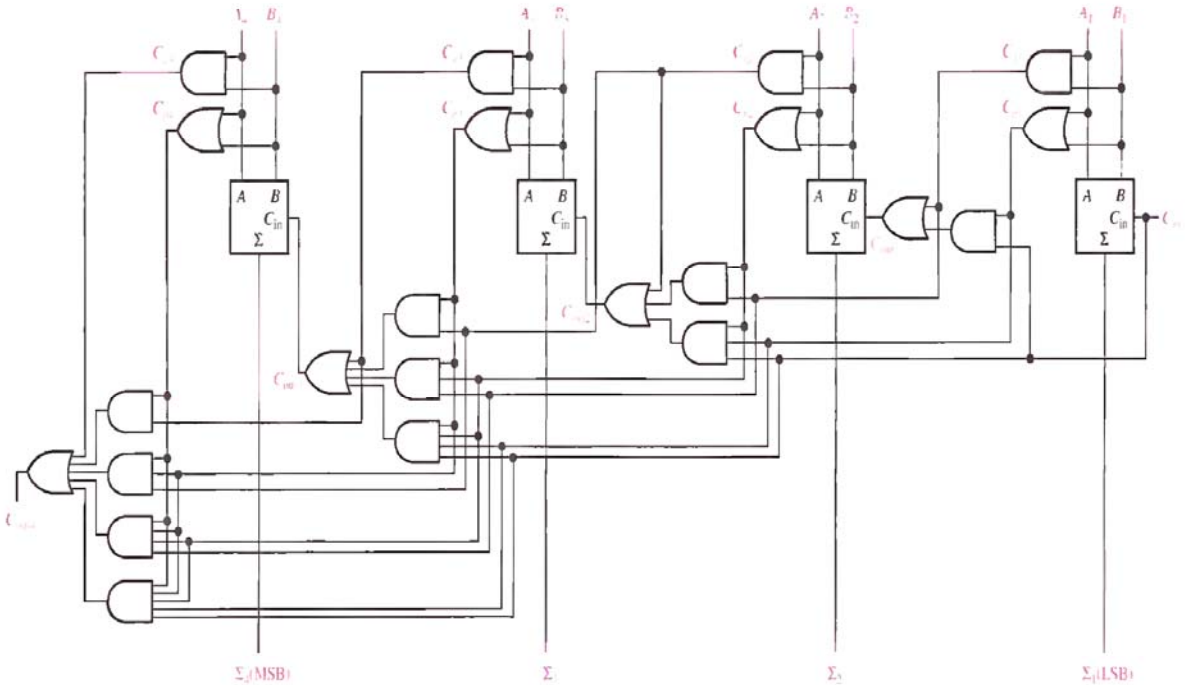


FIGURE 6–18

Logic diagram for a 4-stage look-ahead carry adder.

COMPARATORS

The basic function of a comparator is to compare the magnitudes of two binary quantities to determine the relationship of those quantities. In its simplest form, a comparator circuit determines whether two numbers are equal.

Equality

- The exclusive-OR gate can be used as a basic comparator because its output is a 1 if the two input bits are not equal and a 0 if the input bits are equal.
- Figure 6-19 shows the exclusive-OR gate as a 2-bit comparator.

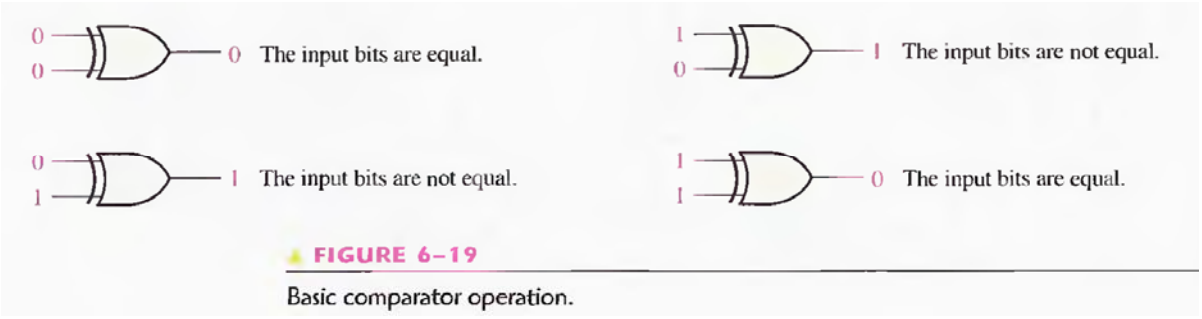
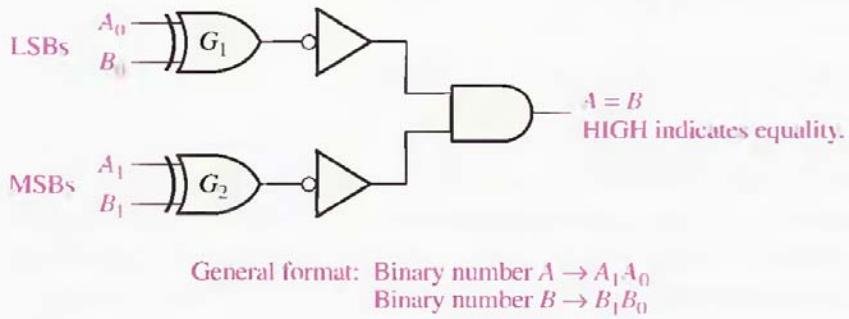


FIGURE 6–19
Basic comparator operation.

In order to compare binary numbers containing two bits each, an additional exclusive-OR gate is necessary. The two least significant bits (LSBs) of the two numbers are compared by gate G_1, and the two most significant bits (MSBs) are compared by gate G_2, as shown in Figure 6–20. If the two numbers are equal, their corresponding bits are the same, and the output of each exclusive-OR gate is a 0. If the corresponding sets of bits are not equal, a 1 occurs on that exclusive-OR gate output.



- In order to produce a single output indicating an equality or inequality of two numbers, two inverters and an AND gate can be used, as shown in Figure 6-20.
- When the two numbers are not equal, one or both sets of corresponding bits are unequal, and a 0 appears on at least one input to the AND gate to produce a 0 on its output.
- The output of the AND gate indicates equality (1) or inequality (0) of the two numbers.

EXAMPLE 6–5

Apply each of the following sets of binary numbers to the comparator inputs in Figure 6–21, and determine the output by following the logic levels through the circuit.

(a) 10 and 10 **(b)** 11 and 10

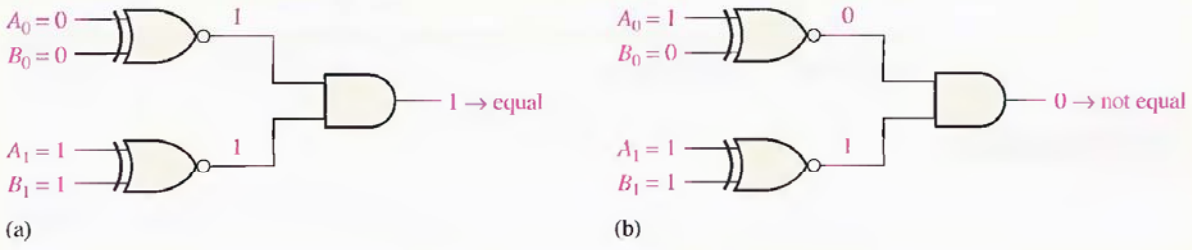


FIGURE 6–21

Solution **(a)** The output is **1** for inputs 10 and 10, as shown in Figure 6–21(a).

(b) The output is **0** for inputs 11 and 10, as shown in Figure 6–21(b).

Related Problem Repeat the process for binary inputs of 01 and 10.

Inequality

In addition to the equality output, many IC comparators provide additional outputs that indicate which of the two binary numbers being compared is the larger. That is, there is an output that indicates when number A is greater than number B ($A > B$) and an output that indicates when number A is less than number B ($A < B$), as shown in the logic symbol for a 4-bit comparator in Figure 6–22.

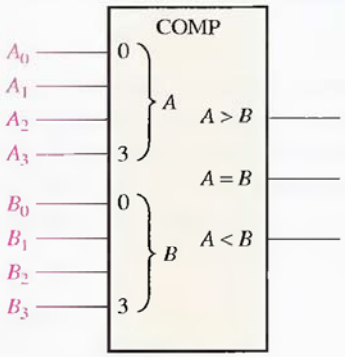


FIGURE 6–22

Logic symbol for a 4-bit comparator with inequality indication.

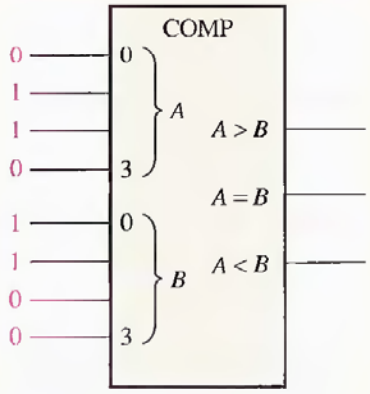
To determine an inequality of binary numbers A and B, you first examine the highest-order bit in each number. The following conditions are possible:

1. If $A_3 = 1$ and $B_3 = 0$, number A is greater than number B.
2. If $A_3 = 0$ and $B_3 = 1$, number A is less than number B.
3. If $A_3 = B_3$, then you must examine the next lower bit position for an inequality.

EXAMPLE 6–6

Determine the $A = B$, $A > B$, and $A < B$ outputs for the input numbers shown on the comparator in Figure 6–23.

▶ **FIGURE 6–23**



Solution The number on the A inputs is 0110 and the number on the B inputs is 0011. The **$A > B$ output is HIGH and the other outputs are LOW.**

Related Problem

What are the comparator outputs when $A_3A_2A_1A_0 = 1001$ and $B_3B_2B_1B_0 = 1010$?

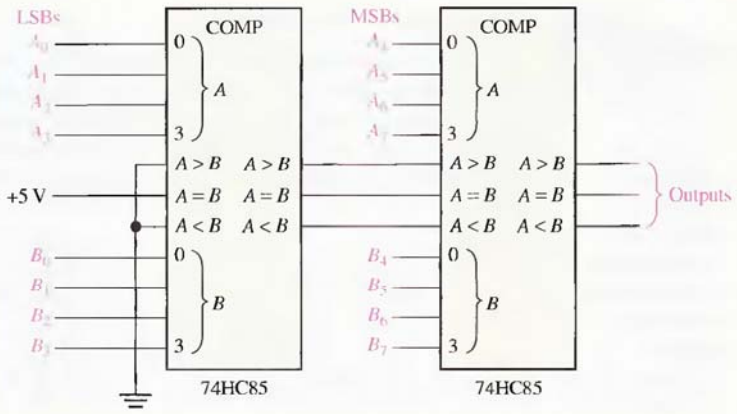
EXAMPLE 6–7

Use 74HC85 comparators to compare the magnitudes of two 8-bit numbers. Show the comparators with proper interconnections.

Solution Two 74HC85s are required to compare two 8-bit numbers. They are connected as shown in Figure 6–25 in a cascaded arrangement.

▶ **FIGURE 6–25**

An 8-bit magnitude comparator using two 74HC85s.



Related Problem Expand the circuit in Figure 6–25 to a 16-bit comparator.

DECODER

- A decoder is a digital circuit that detects the presence of a specified combination of bits (code) on its inputs and indicates the presence of that code by a specified output level.

- In its general form, a decoder has n input lines to handle n bits and from one to 2^n output lines to indicate the presence of one or more n-bit combinations.

The Basic Binary Decoder

An AND gate can be used as the basic decoding element because it produces a HIGH output only when all of its inputs are HIGH.

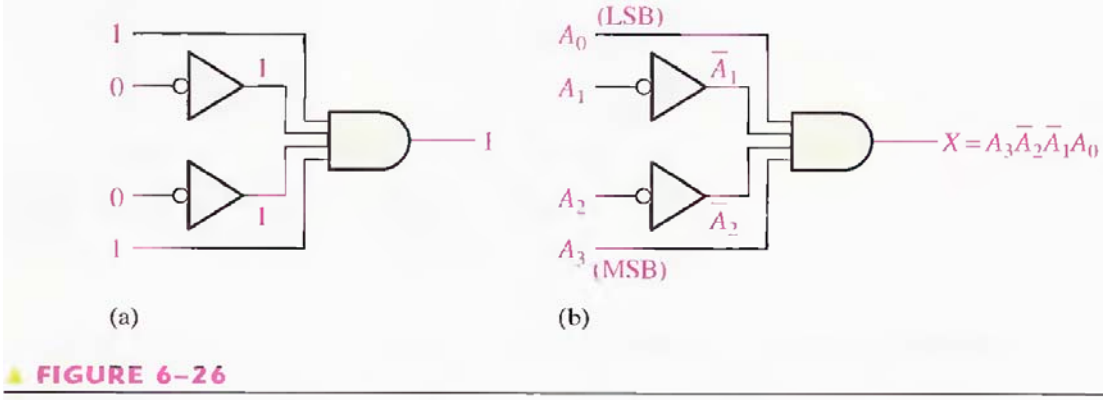


FIGURE 6–26

Decoding logic for the binary code 1001 with an active-HIGH output.

If a NAND gate is used in place of the AND gate in Figure 6-26, a LOW output will indicate the presence of the proper binary code, which is 1001 in this case.

EXAMPLE 6–8

Determine the logic required to decode the binary number 1011 by producing a HIGH level on the output.

Solution

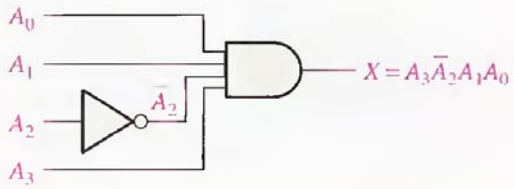
The decoding function can be formed by complementing only the variables that appear as 0 in the desired binary number, as follows:

$$X = A_3\overline{A}_2A_1A_0 \qquad (1011)$$

This function can be implemented by connecting the true (uncomplemented) variables A_0, A_1, and A_3 directly to the inputs of an AND gate, and inverting the variable A_2 before applying it to the AND gate input. The decoding logic is shown in Figure 6–27.

FIGURE 6–27

Decoding logic for producing a HIGH output when 1011 is on the inputs.



The 4-Bit Decoder

In order to decode all possible combinations of four bits, sixteen decoding gates are required ($2^4 = 16$). This type of decoder is commonly called either a *4-line-to-16-line decoder* because there are four inputs and sixteen outputs or a *1-of-16 decoder* because for any given code on the inputs, one of the sixteen outputs is activated. A list of the sixteen binary codes and their corresponding decoding functions is given in Table 6–4.

If an active-LOW output is required for each decoded number, the entire decoder can be implemented with NAND gates and inverters. In order to decode each of the sixteen binary codes, sixteen NAND gates are required (AND gates can be used to produce active-HIGH outputs).

TABLE 6–4

Decoding functions and truth table for a 4-line-to-16-line (1-of-16) decoder with active-LOW outputs.

DECIMAL DIGIT	BINARY INPUTS A_3	A_2	A_1	A_0	DECODING FUNCTION	OUTPUTS 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	$\overline{A}_3\overline{A}_2\overline{A}_1\overline{A}_0$	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	1	$\overline{A}_3\overline{A}_2\overline{A}_1A_0$	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	1	0	$\overline{A}_3\overline{A}_2A_1\overline{A}_0$	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	1	1	$\overline{A}_3\overline{A}_2A_1A_0$	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
4	0	1	0	0	$\overline{A}_3A_2\overline{A}_1\overline{A}_0$	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
5	0	1	0	1	$\overline{A}_3A_2\overline{A}_1A_0$	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
6	0	1	1	0	$\overline{A}_3A_2A_1\overline{A}_0$	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
7	0	1	1	1	$\overline{A}_3A_2A_1A_0$	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
8	1	0	0	0	$A_3\overline{A}_2\overline{A}_1\overline{A}_0$	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
9	1	0	0	1	$A_3\overline{A}_2\overline{A}_1A_0$	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
10	1	0	1	0	$A_3\overline{A}_2A_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
11	1	0	1	1	$A_3\overline{A}_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
12	1	1	0	0	$A_3A_2\overline{A}_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
13	1	1	0	1	$A_3A_2\overline{A}_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
14	1	1	1	0	$A_3A_2A_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
15	1	1	1	1	$A_3A_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

A logic symbol for a 4-line-to-16-line (1-of-16) decoder with active-LOW outputs is shown in Figure 6–28. The BIN/DEC label indicates that a binary input makes the corresponding decimal output active. The input labels 8, 4, 2, and 1 represent the binary weights of the input bits ($2^3 2^2 2^1 2^0$).

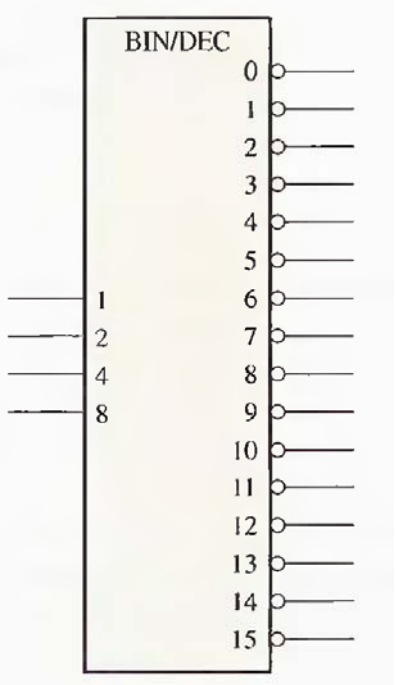
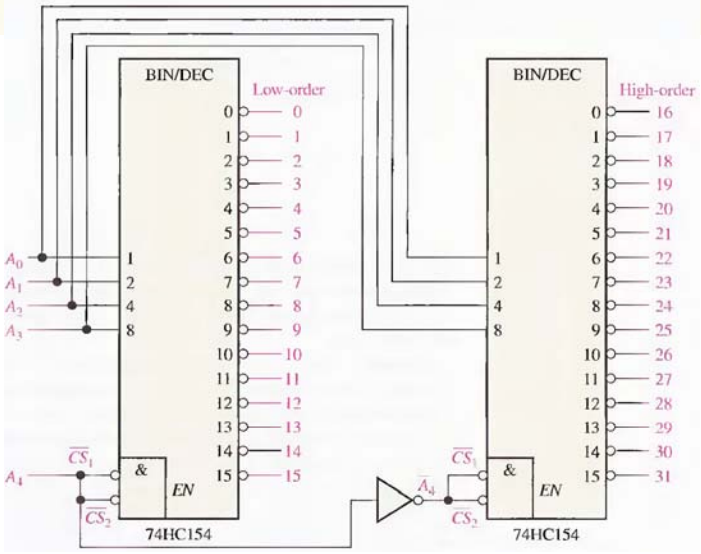


FIGURE 6-28
logic symbol for a 4-line-to-16-line (1-of-1 6) decoder.

A certain application requires that a 5-bit number be decoded. Use 74HC154 decoders to implement the logic. The binary number is represented by the format $A_4A_3A_2A_1A_0$.

Since the 74HC154 can handle only four bits, two decoders must be used to decode five bits. The fifth bit, A_4, is connected to the chip select inputs, $\overline{CS}_1$ and $\overline{CS}_2$, of one decoder, and $\overline{A}_4$ is connected to the $\overline{CS}_1$ and $\overline{CS}_2$ inputs of the other decoder, as shown in Figure 6–30. When the decimal number is 15 or less, $A_4 = 0$, the low-order decoder is enabled, and the high-order decoder is disabled. When the decimal number is greater than 15, $A_4 = 1$ so $\overline{A}_4 = 0$, the high-order decoder is enabled, and the low-order decoder is disabled.



An Application

Decoders are used in many types of applications. One example is in computers for input/output selection as depicted in the general diagram of Figure 6–31.

Computers must communicate with a variety of external devices called *peripherals* by sending and/or receiving data through what is known as input/output (I/O) ports. These

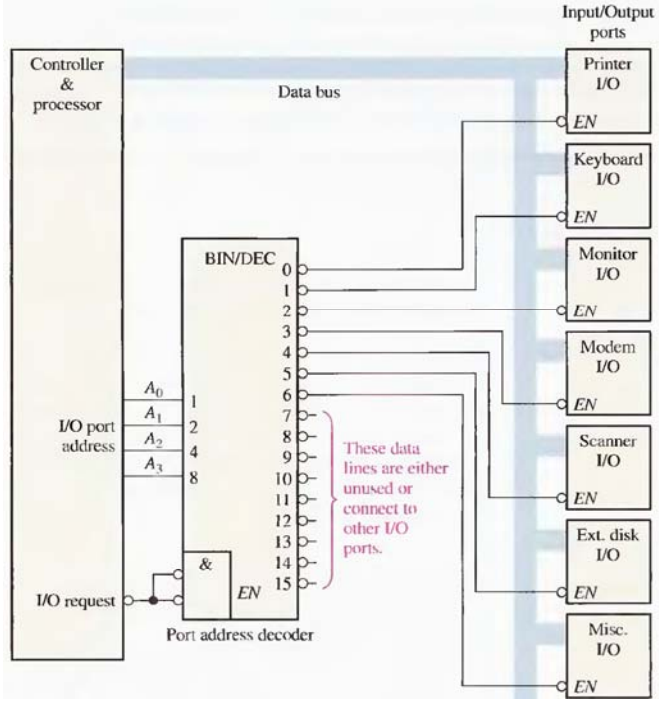


FIGURE 6–31

A simplified computer I/O port system with a port address decoder with only four address lines shown.

external devices include printers, modems, scanners, external disk drives, keyboard, video monitors, and other computers. As indicated in Figure 6–31, a decoder is used to select the I/O port as determined by the computer so that data can be sent or received from a specific external device.

Each I/O port has a number, called an address, which uniquely identifies it. When the computer wants to communicate with a particular device, it issues the appropriate address code for the I/O port to which that particular device is connected. This binary port address is decoded and the appropriate decoder output is activated to enable the I/O port.

As shown in Figure 6–31, binary data are transferred within the computer on a data bus, which is a set of parallel lines. For example, an 8-bit bus consists of eight parallel lines that can carry one byte of data at a time. The data bus goes to all of the I/O ports, but any data coming in or going out will only pass through the port that is enabled by the port address decoder.

The BCD-to-Decimal Decoder

The BCD-to-decimal decoder converts each BCD code (8421 code) into one of ten possible decimal digit indications. It is frequently referred to as a *4-line-to-10-line decoder* or a *1-of-10 decoder*.

TABLE 6–5

BCD decoding functions.

DECIMAL DIGIT	BCD CODE A_3	A_2	A_1	A_0	DECODING FUNCTION
0	0	0	0	0	$\overline{A}_3\overline{A}_2\overline{A}_1\overline{A}_0$
1	0	0	0	1	$\overline{A}_3\overline{A}_2\overline{A}_1A_0$
2	0	0	1	0	$\overline{A}_3\overline{A}_2A_1\overline{A}_0$
3	0	0	1	1	$\overline{A}_3\overline{A}_2A_1A_0$
4	0	1	0	0	$\overline{A}_3A_2\overline{A}_1\overline{A}_0$
5	0	1	0	1	$\overline{A}_3A_2\overline{A}_1A_0$
6	0	1	1	0	$\overline{A}_3A_2A_1\overline{A}_0$
7	0	1	1	1	$\overline{A}_3A_2A_1A_0$
8	1	0	0	0	$A_3\overline{A}_2\overline{A}_1\overline{A}_0$
9	1	0	0	1	$A_3\overline{A}_2\overline{A}_1A_0$

Each of these decoding functions is implemented with NAND gates to provide active-LOW outputs. If an active-HIGH output is required, AND gates are used for decoding.

EXAMPLE 6–10

The 74HC42 is an integrated circuit BCD-to-decimal decoder. The logic symbol is shown in Figure 6–32. If the input waveforms in Figure 6–33(a) are applied to the inputs of the 74HC42, show the output waveforms.

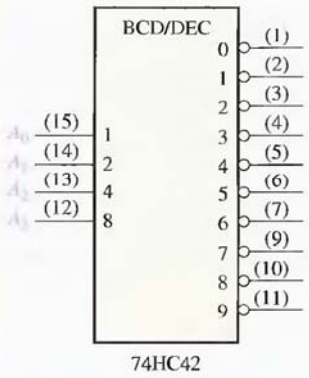
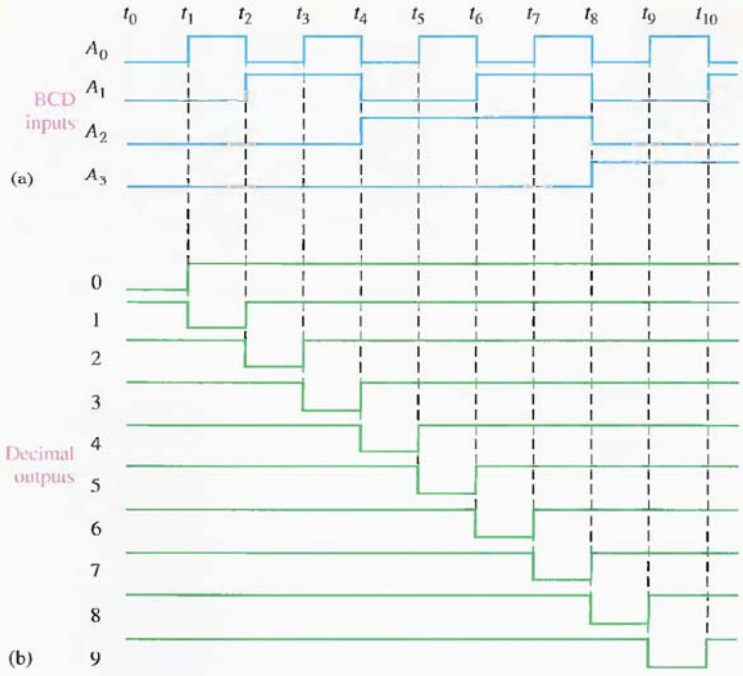


FIGURE 6–32

The 74HC42 BCD-to-decimal decoder.



The BCD-to-7-Segment Decoder

The BCD-to-7-segment decoder accepts the BCD code on its inputs and provides outputs to drive 7-segment display devices to produce a decimal readout. The logic diagram for a basic 7-segment decoder is shown in Figure 6–34.

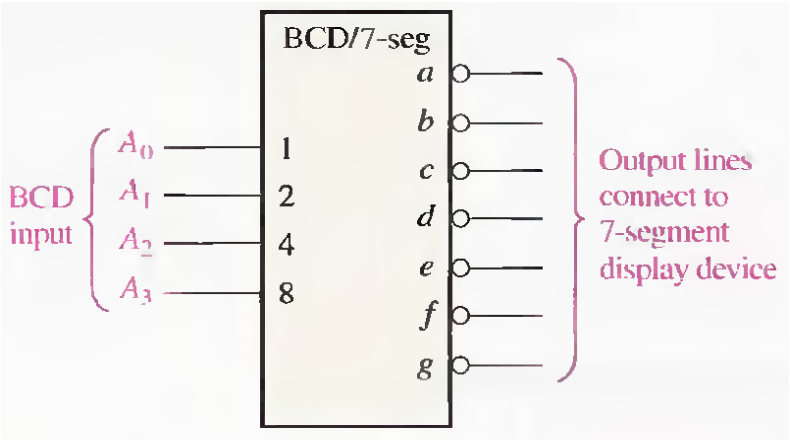


FIGURE 6-34
Logic symbol for a BCD-to- 7 -segment decoder driver with active-lOW outputs.

Encoder

- An encoder accepts an active level on one of its inputs representing a digit, such as a decimal or octal digit, and converts it to a coded output, such as BCD or binary.
- Encoders can also be devised to encode various symbols and alphabetic characters.
- The process of converting from symbols or numbers to a coded format is called encoding.

The Decimal-to-BCD Encoder

This type of encoder has ten inputs—one for each decimal digit—and four outputs corresponding to the BCD code, as shown in Figure 6–37. This is a basic 10-line-to-4-line encoder.

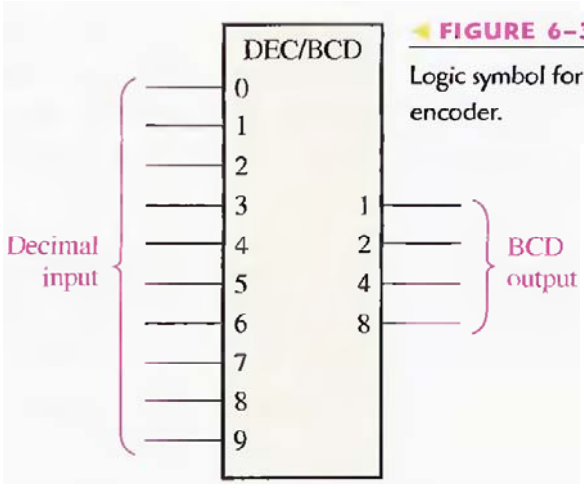


FIGURE 6–37

Logic symbol for a decimal-to-BCD encoder.

TABLE 6–6

DECIMAL DIGIT	BCD CODE A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

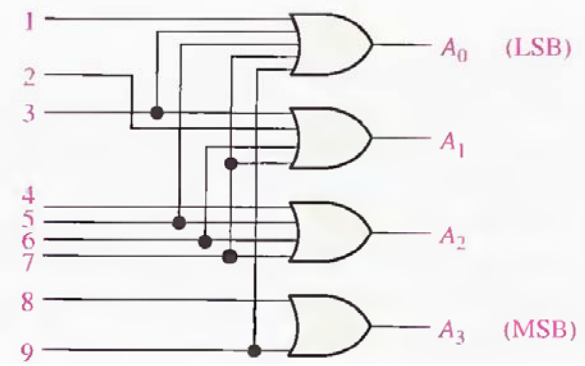


FIGURE 6–38

Basic logic diagram of a decimal-to-BCD encoder. A 0-digit input is not needed because the BCD outputs are all LOW when there are no HIGH inputs.

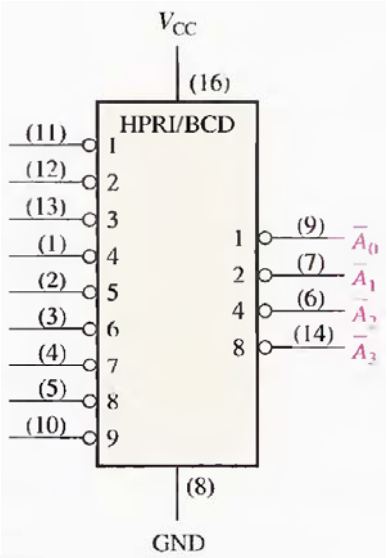
The basic operation of the circuit in Figure 6–38 is as follows: When a HIGH appears on *one* of the decimal digit input lines, the appropriate levels occur on the four BCD output lines. For instance, if input line 9 is HIGH (assuming all other input lines are LOW), this condition will produce a HIGH on outputs A_0 and A_3 and LOWs on outputs A_1 and A_2, which is the BCD code (1001) for decimal 9.

The Decimal-to-BCD Priority Encoder

- A priority encoder also offers additional flexibility in that it can be used in applications that require priority detection.
- The priority function means that the encoder will produce a BCD output corresponding to the highest-order decimal digit input that is active and will ignore any other lower-order active inputs.
- For instance, if the 6 and the 3 inputs are both active, the BCD output is 0110 (which represents decimal 6).

THE 74HC147 DECIMAL-TO-BCD ENCODER

The 74HC147 is a priority encoder with active-LOW inputs (0) for decimal digits 1 through 9 and active-LOW BCD outputs as indicated in the logic symbol in Figure 6–39. A BCD zero output is represented when none of the inputs is active.



▶ **FIGURE 6–39** Logic diagram

THE 74LS148 8-LINE-TO-3-lINE ENCODER

- The 74LSl48 is a priority encoder that has eight active-LOW inputs and three active-LOW binary outputs, as shown in Figure 6-40.
- This device can be used for converting octal inputs to a 3-bit binary code.
- To enable the device, the EI (enable input) must be LOW.
- It also has the EO (enable output) and GS output for expansion purposes.
- The EO is LOW when the EI is LOW and none of the inputs (0 through 7) is active.
- GS is LOW when EI is LOW and any of the inputs is active.

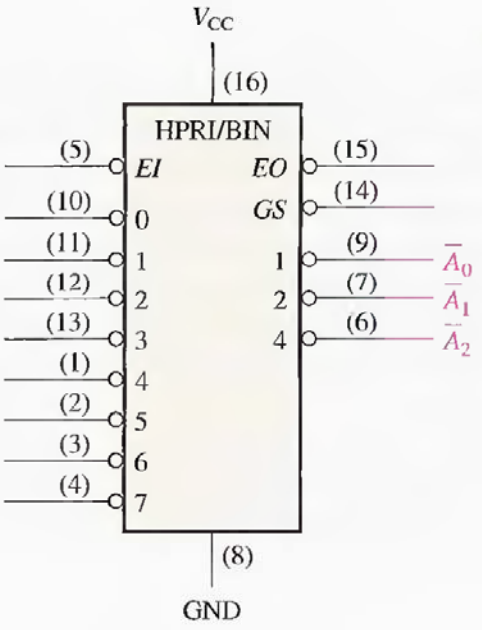


FIGURE 6–40

Logic symbol for the 74LS148 8-line-to-3-line encoder.

The 74LS148 can be expanded to a 16-line-to-4-line encoder by connecting the *EO* of the higher-order encoder to the *EI* of the lower-order encoder and negative-ORing the corresponding binary outputs as shown in Figure 6–41. The *EO* is used as the fourth and most-significant bit. This particular configuration produces active-HIGH outputs for the 4-bit binary number.

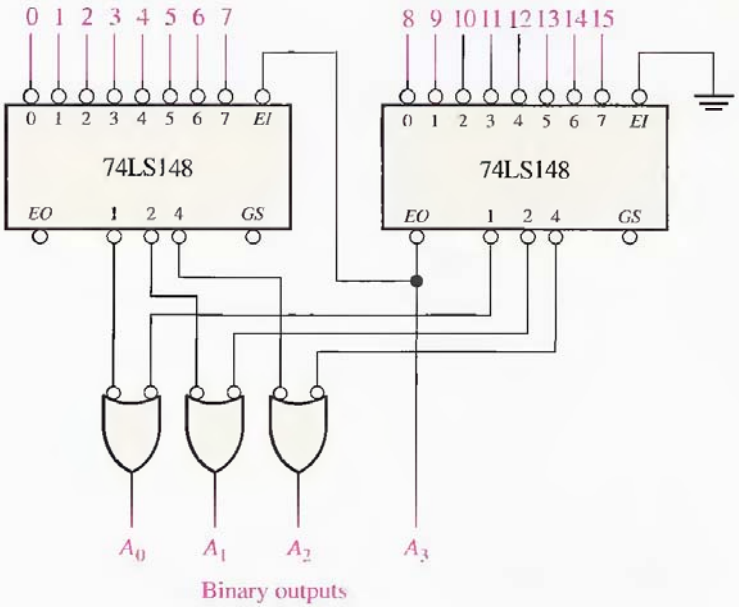


FIGURE 6–41

A 16-line-to-4 line encoder using 74LS148s and external logic.

EXAMPLE 6–11

If LOW levels appear on pins, 1, 4, and 13 of the 74HC147 shown in Figure 6–39, indicate the state of the four outputs. All other inputs are HIGH.

Solution Pin 4 is the highest-order decimal digit input having a LOW level and represents decimal 7. Therefore, the output levels indicate the BCD code for decimal 7 where $\overline{A}_0$ is the LSB and $\overline{A}_3$ is the MSB. Output $\overline{A}_0$ is LOW, $\overline{A}_1$ is LOW, $\overline{A}_2$ is LOW, and $\overline{A}_3$ is HIGH.

Related Problem What are the outputs of the 74HC147 if all its inputs are LOW? If all its inputs are HIGH?

MULTIPLEXERS (DATA SELECTORS)

A **multiplexer (MUX)** is a device that allows digital information from several sources to be routed onto a single line for transmission over that line to a common destination. The basic multiplexer has several data-input lines and a single output line. It also has data-select inputs, which permit digital data on any one of the inputs to be switched to the output line. Multiplexers are also known as data selectors.

- A logic symbol for a 4-input multiplexer (MUX) is shown in Figure 6-46.
- There are two data-select lines because with two select bits. anyone of the four data-input lines can be selected.

FIGURE 6–46

Logic symbol for a 1-of-4 data selector/multiplexer.

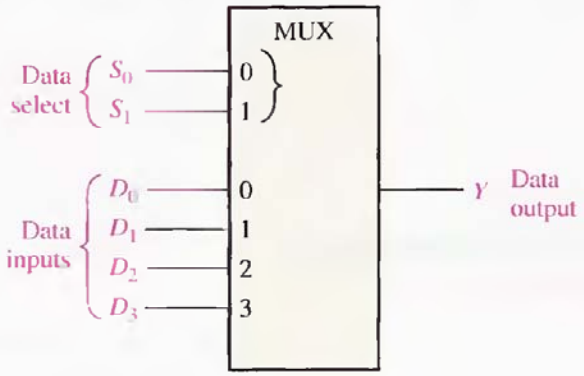


TABLE 6–8

Data selection for a 1-of-4-multiplexer.

DATA-SELECT INPUTS S_1	S_0	INPUT SELECTED
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

- The data output is equal to the state of the selected data input.
- We can therefore, derive a logic expression for the output in terms of the data input and the select inputs.

The data output is equal to D_0 only if $S_1 = 0$ and $S_0 = 0$: $Y = D_0\overline{S}_1\overline{S}_0$.

The data output is equal to D_1 only if $S_1 = 0$ and $S_0 = 1$: $Y = D_1\overline{S}_1S_0$.

The data output is equal to D_2 only if $S_1 = 1$ and $S_0 = 0$: $Y = D_2S_1\overline{S}_0$.

The data output is equal to D_3 only if $S_1 = 1$ and $S_0 = 1$: $Y = D_3S_1S_0$.

When these terms are ORed, the total expression for the data output is

$$Y = D_0\overline{S}_1\overline{S}_0 + D_1\overline{S}_1S_0 + D_2S_1\overline{S}_0 + D_3S_1S_0$$

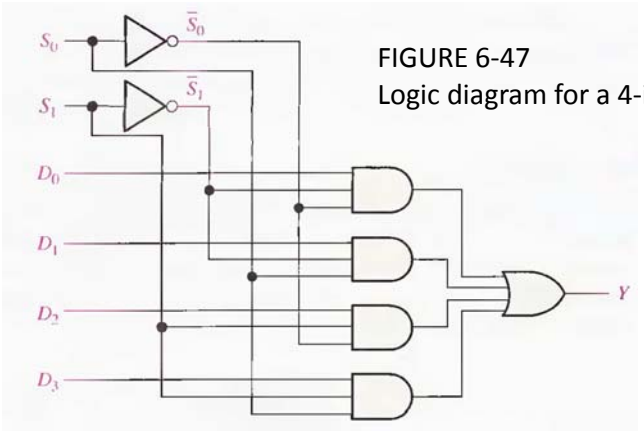
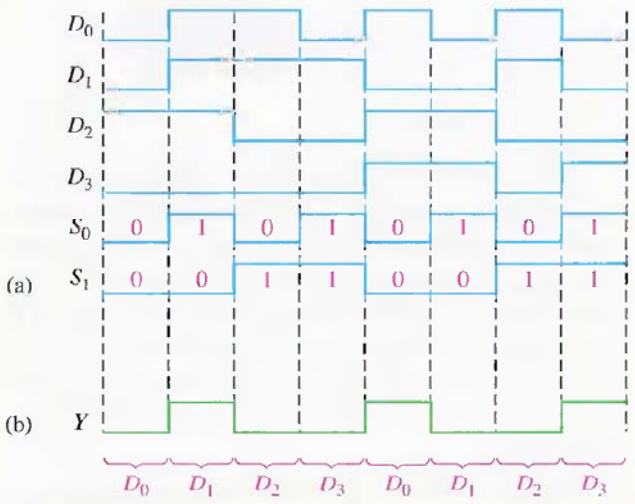


FIGURE 6-47
Logic diagram for a 4-input multiplexer

EXAMPLE 6–14

The data-input and data-select waveforms in Figure 6–48(a) are applied to the multiplexer in Figure 6–47. Determine the output waveform in relation to the inputs.

Solution



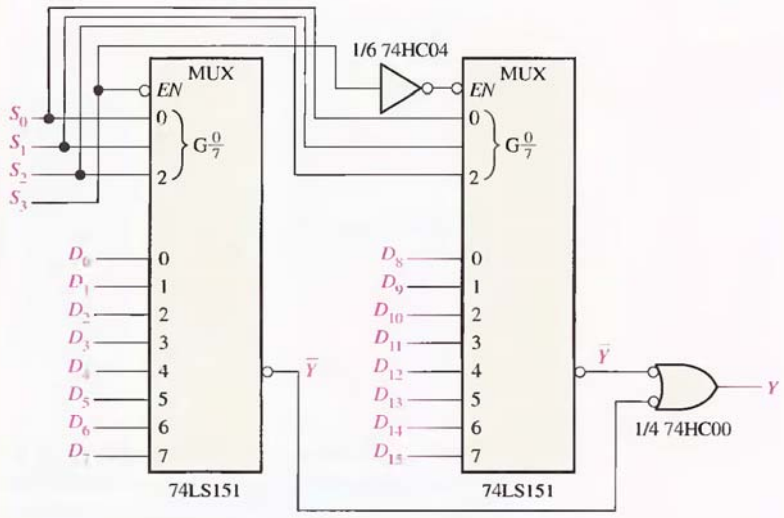
The binary state of the data-select inputs during each interval determines which data input is selected. Notice that the data-select inputs go through a repetitive binary sequence 00, 01, 10. 11, 00. 01, 10. 11. and so on. The resulting output waveform is shown in Figure 6–48(b).

EXAMPLE 6–15

Use 74LS151s and any other logic necessary to multiplex 16 data lines onto a single data-output line.

Solution

An implementation of this system is shown in Figure 6–51. Four bits are required to select one of 16 data inputs ($2^4 = 16$). In this application the *Enable* input is used as the most significant data-select bit. When the MSB in the data-select code is LOW, the left 74LS151 is enabled, and one of the data inputs (D_0 through D_7) is selected by the other three data-select bits. When the data-select MSB is HIGH, the right 74LS151 is enabled, and one of the data inputs (D_8 through D_{15}) is selected. The selected input data are then passed through to the negative-OR gate and onto the single output line.



▶ **FIGURE 6–51**

A 16-input multiplexer.

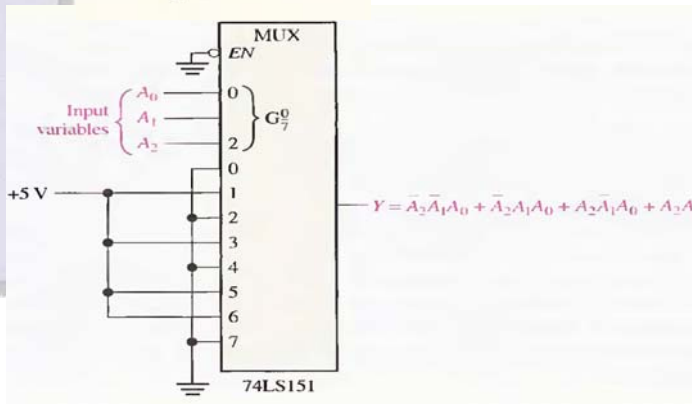
Implement the logic function specified in Table 6–9 by using a 74LS151 8-input data selector/multiplexer. Compare this method with a discrete logic gate implementation.

TABLE 6–9

INPUTS			OUTPUT
A_2	A_1	A_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

FIGURE 6–53

Data selector/multiplexer connected as a 3-variable logic function generator.



The implementation of this function with logic gates would require four 3-input AND gates, one 4-input OR gate, and three inverters unless the expression can be simplified.

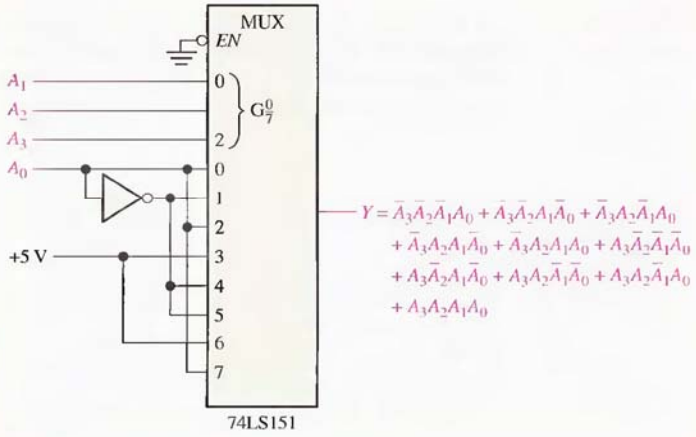
Implement the logic function in Table 6–10 by using a 74LS151 8-input data selector/multiplexer. Compare this method with a discrete logic gate implementation.

TABLE 6–10

DECIMAL DIGIT	INPUTS				OUTPUT
	A_3	A_2	A_1	A_0	Y
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

FIGURE 6–54

Data selector/multiplexer connected as a 4-variable logic function generator.

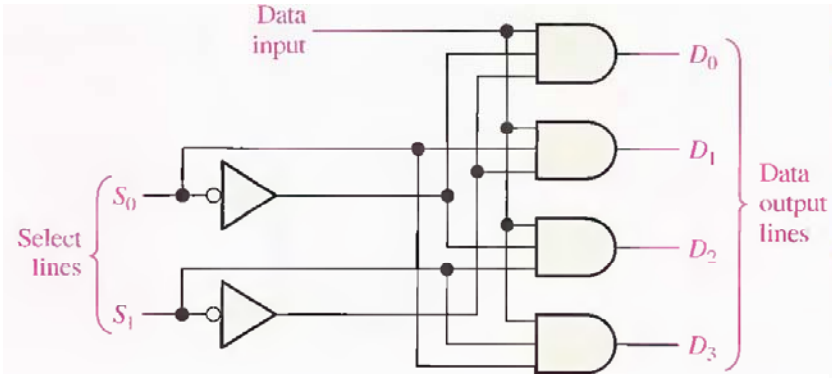


If implemented with logic gates, the function would require as many as ten 4-input AND gates, one 10-input OR gate, and four inverters, although possible simplification would reduce this requirement.

DEMULTIPLEXERS

- A demultiplexer (DEMUX) takes digital information from one line and distribute it to a given number of output lines. For this reason, the demultiplexer is also known as a data distributor.
- Decoders can also be used as demultiplexer.

Figure 6–55 shows a 1-line-to-4-line demultiplexer (DEMUX) circuit. The data-input line goes to all of the AND gates. The two data-select lines enable only one gate at a time, and the data appearing on the data-input line will pass through the selected gate to the associated data-output line.



In a demultiplexer, data goes from one line to several lines.

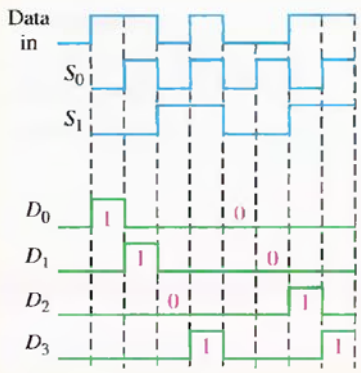
FIGURE 6–55

A 1-line-to-4-line demultiplexer.

EXAMPLE 6–18

The serial data-input waveform (Data in) and data-select inputs (S_0 and S_1) are shown in Figure 6–56. Determine the data-output waveforms on D_0 through D_3 for the demultiplexer in Figure 6–55.

FIGURE 6–56



Solution Notice that the select lines go through a binary sequence so that each successive input bit is routed to D_0, D_1, D_2, and D_3 in sequence, as shown by the output waveforms in Figure 6–56.

Related Problem Develop the timing diagram for the demultiplexer if the S_0 and S_1 waveforms are both inverted.

Encoder and decoder applications

- The radio frequency spectrum is filled with noise and other signals, especially those frequencies where unlicensed transmitter operation under FCC part 15 rules is allowed. When using a wireless remote control system it is desirable to have a way of filtering out or ignoring those unwanted signals to prevent false data from being received.

- A simple way to accomplish this is to use an encoder IC at the transmitter and a decoder IC at the receiver. The encoder generates serial codes that are automatically sent three times and must be received at least twice before data is accepted as valid by the decoder circuit.

CODE CONVERTERS

BCD-to-Binary Conversion

decimal number 87 can be expressed in BCD as

$$\underbrace{1000}_{8}\ \underbrace{0111}_{7}$$

The left-most 4-bit group represents 80, and the right-most 4-bit group represents 7. That is, the left-most group has a weight of 10, and the right-most group has a weight of 1. Within each group, the binary weight of each bit is as follows:

	Tens Digit				Units Digit			
Weight:	80	40	20	10	8	4	2	1
Bit designation:	B_3	B_2	B_1	B_0	A_3	A_2	A_1	A_0

Convert the BCD numbers 00100111 (decimal 27) and 10011000 (decimal 98) to binary.

Write the binary representations of the weights of all 1s appearing in the numbers, and then add them together.

```
80  40  20  10  8  4  2  1
 0   0   1   0  0  1  1  1
```

```
0000001    1
0000010    2
0000100    4
+0010100   20
0011011    Binary number for decimal 27
```

```
80  40  20  10  8  4  2  1
 1   0   0   1  1  0  0  0
```

```
0001000    8
0001010   10
+1010000  80
1100010    Binary number for decimal 98
```

Binary-to-Gray and Gray-to-Binary Conversion

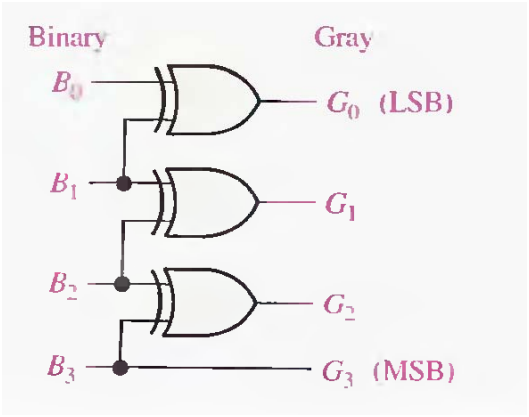


FIGURE 6–43

Four-bit binary-to-Gray conversion

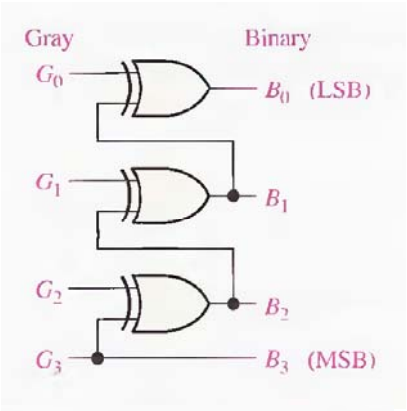


FIGURE 6–44

Four-bit Gray-to-binary conversion

(a) Convert the binary number 0101 to Gray code with exclusive-OR gates.

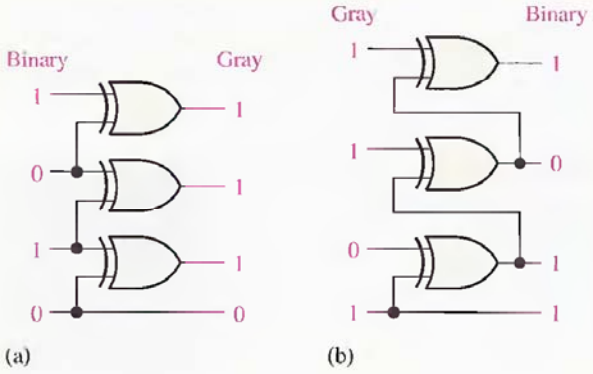
(b) Convert the Gray code 1011 to binary with exclusive-OR gates.

Solution

(a) 0101_2 is 0111 Gray. See Figure 6–45(a).

(b) 1011 Gray is 1101_2. See Figure 6–45(b).

FIGURE 6–45



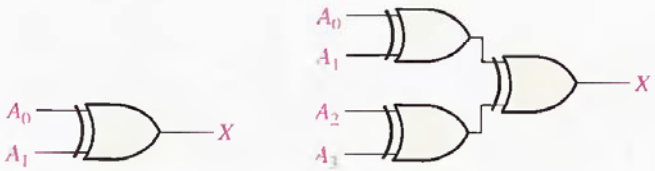
PARITY GENERATORS/CHECKERS

Basic Parity Logic

A parity bit indicates if the number of 1s in a code is even or odd for the purpose of error detection.

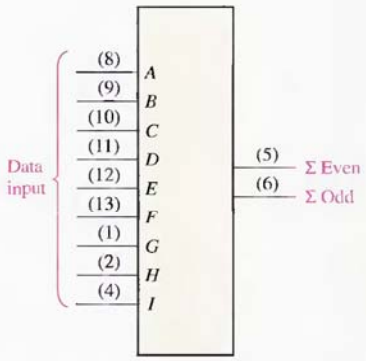
When the number of 1s on the inputs is even, the output X is 0 (LOW). When the number of 1s is odd, the output X is 1 (HIGH).

FIGURE 6–58



(a) Summing of two bits (b) Summing of four bits

THE 74LS280 9-BIT PARITY GENERATOR/CHECKER



(a) Traditional logic symbol

Number of Inputs A–I That Are HIGH	Outputs Σ Even	Outputs Σ Odd
0, 2, 4, 6, 8	H	L
1, 3, 5, 7, 9	L	H

(b) Function table

FIGURE 6–59

The 74LS280 9-bit parity generator/checker.

Parity Checker When this device is used as an even parity checker, the number of input bits should always be even; and when a parity error occurs, the Σ Even output goes LOW and the Σ Odd output goes HIGH. When it is used as an odd parity checker, the number of input bits should always bé odd; and when a parity error occurs, the Σ Odd output goes LOW and the Σ Even output goes HIGH.

Parity Generator If this device is used as an even parity generator. the parity bit is taken at the Σ Odd output because this output is a 0 if there is an even number of input bits and it is a 1 if there is an odd number. When used as an odd parity generator, the parity bit is taken at the Σ Even output because it is a 0 when the number of inputs bits is odd.