

# Functions of Combinational Logic

## CHAPTER OUTLINE

- 6–1 Half and Full Adders
- 6–2 Parallel Binary Adders
- 6–3 Ripple Carry and Look-Ahead Carry Adders
- 6–4 Comparators
- 6–5 Decoders
- 6–6 Encoders
- 6–7 Code Converters
- 6–8 Multiplexers (Data Selectors)
- 6–9 Demultiplexers
- 6–10 Parity Generators/Checkers

## INTRODUCTION

In this chapter, several types of combinational logic functions are introduced including adders, comparators, decoders, encoders, code converters, multiplexers (data selectors), demultiplexers, and parity generators/checkers. VHDL implementation of each logic function is provided, and examples of fixed-function IC devices are included. Each device introduced may also be available in other logic families.

### 6–1 Half and Full Adders

Adders are important in computers and also in other types of digital systems in which numerical data are processed. An understanding of the basic adder operation is fundamental to the study of digital systems. In this section, the half-adder and the full-adder are introduced.

## The Half-Adder

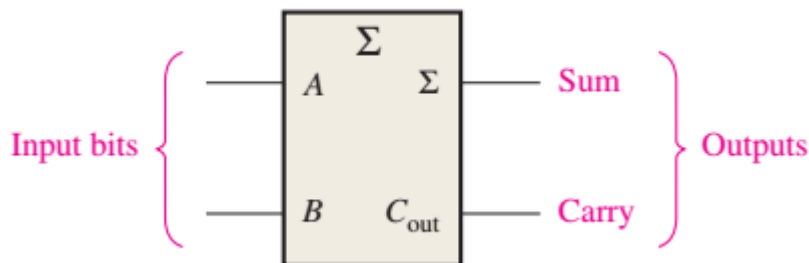
Recall the basic rules for binary addition as stated in Chapter 2.

$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 10$

The operations are performed by a logic circuit called a **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.



**A half-adder adds two bits and produces a sum and an output carry.**

### 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.

$$C_{out} = AB \quad \text{Equation 6–1}$$

Now observe that the sum output ( $\Sigma$ ) 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.

$$\Sigma = A \oplus B \quad \text{Equation 6–2}$$

From Equations 6–1 and 6–2, the logic implementation required for the half-adder function can be developed. The output carry is produced with an AND gate with  $A$  and  $B$  on the

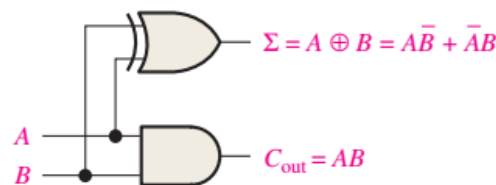
**TABLE 6-1**

Half-adder truth table.

<i>A</i>	<i>B</i>	<i>C</i> <sub>out</sub>	$\Sigma$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

 $\Sigma$  = sum*C*<sub>out</sub> = output carry*A* and *B* = input variables (operands)

inputs, and the sum output is generated with an exclusive-OR gate, as shown in Figure 6-2. Remember that the exclusive-OR can be implemented with AND gates, an OR gate, and inverters.

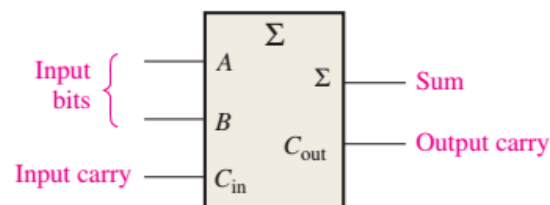
**FIGURE 6-2** Half-adder logic diagram.

## The Full-Adder

The second category of adder is 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. Open file F06-03 to verify operation.

**TABLE 6-2**

Full-adder truth table.

$A$	$B$	$C_{in}$	$C_{out}$	$\Sigma$
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  $CO$  $\Sigma$  = sum $A$  and  $B$  = input variables (operands)

### Full-Adder Logic

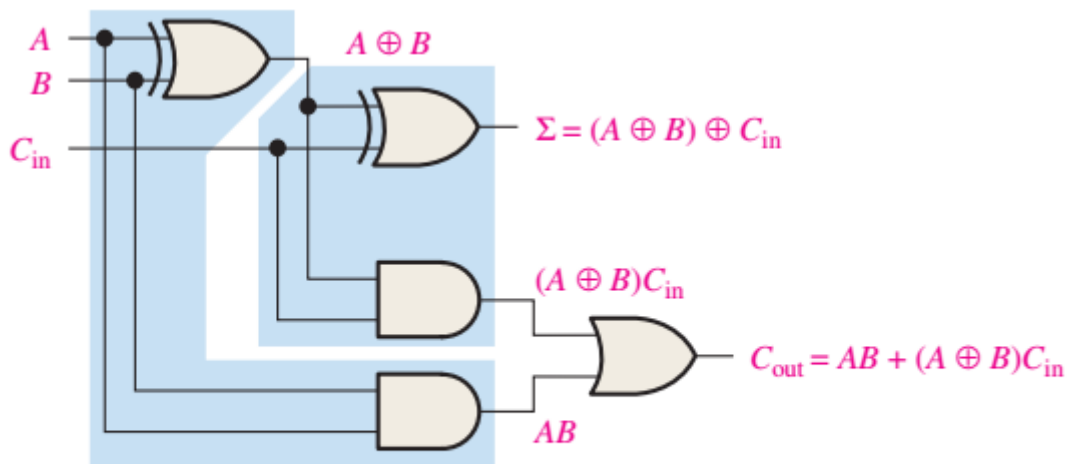
The full-adder must add the two input bits and the input carry. From the half-adder you know that the sum of the input bits  $A$  and  $B$  is the exclusive-OR of those two variables,  $A \oplus B$ . For the input carry ( $C_{in}$ ) to be added to the input bits, it must be exclusive-ORed with  $A \oplus B$ , yielding the equation for the sum output of the full-adder.

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

This means that 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).



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



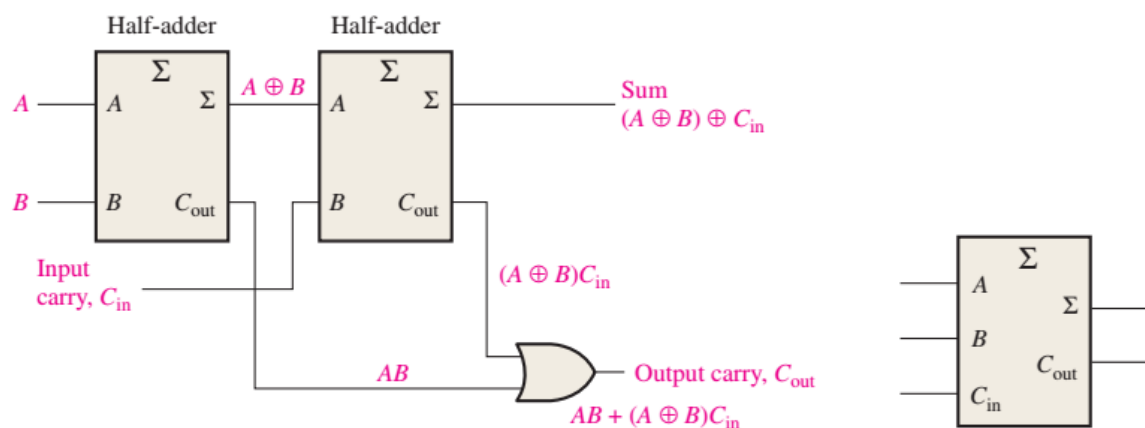
(b) Complete logic circuit for a full-adder (each half-adder is enclosed by a shaded area)

**FIGURE 6-4** Full-adder logic.

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. You can verify this fact by studying Table 6-2. The output carry of the full-adder is therefore produced by input  $A$  ANDed with input  $B$  and  $A \oplus B$  ANDed with  $C_{in}$ . These two terms are ORed, as expressed in Equation 6-4. This function is implemented and combined with the sum logic to form a complete full-adder circuit, as shown in Figure 6-4(b).

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

Notice in Figure 6-4(b) there are two half-adders, connected as shown in the block diagram of Figure 6-5(a), with their output carries ORed. The logic symbol shown in Figure 6-5(b) will normally be used to represent the full-adder.



(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.

## SECTION 6-1 CHECKUP

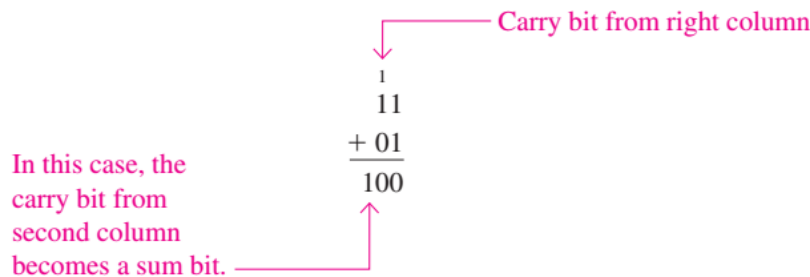
Answers are at the end of the chapter.

1. Determine the sum ( $\Sigma$ ) and the output carry ( $C_{out}$ ) of a half-adder for each set of input bits:  
(a) 01      (b) 00      (c) 10      (d) 11
2. A full-adder has  $C_{in} = 1$ . What are the sum ( $\Sigma$ ) and the output carry ( $C_{out}$ ) when  $A = 1$  and  $B = 1$ ?

## 6-2 Parallel Binary Adders

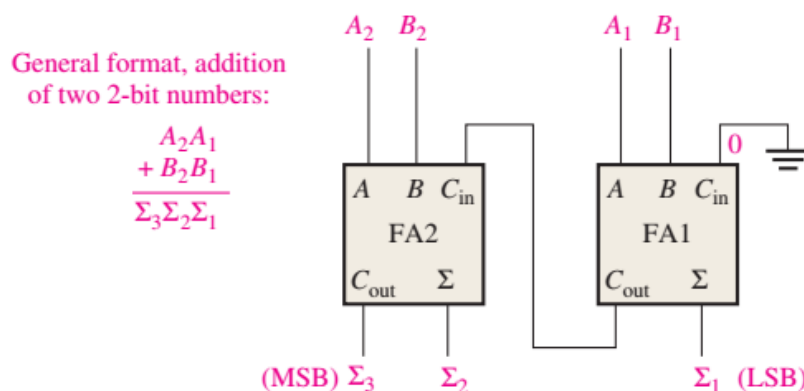
Two or more full-adders are connected to form parallel binary adders. In this section, you will learn the basic operation of this type of adder and its associated input and output functions.

As you learned in Section 6-1, a single full-adder is capable of adding two 1-bit numbers and an input carry. To add binary numbers with more than one bit, you must use additional full-adders. When one binary number is added to another, each column generates a sum bit and a 1 or 0 carry bit to the next column to the left, as illustrated here with 2-bit numbers.



To add two binary numbers, a full-adder (FA) 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.



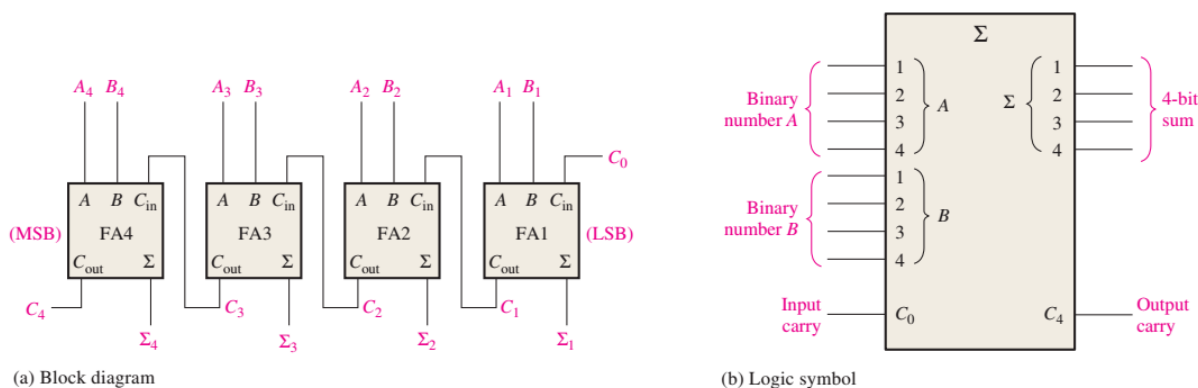


**FIGURE 6-7** Block diagram of a basic 2-bit parallel adder using two full-adders.

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  $\Sigma_1$ ,  $\Sigma_2$ , and  $\Sigma_3$ . Notice that the output carry from the left-most full-adder becomes the most significant bit (MSB) in the sum,  $\Sigma_3$ .

## 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*.



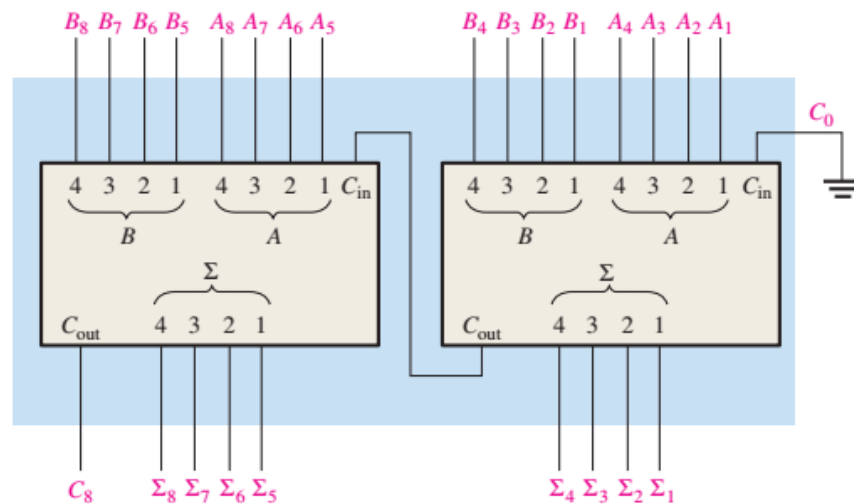
**FIGURE 6-9** A 4-bit parallel adder.

In keeping with most manufacturers' data sheets, the input labeled  $C_0$  is the input carry to the least significant bit adder;  $C_4$ , in the case of four bits, is the output carry of the most significant bit adder; and  $\Sigma_1$  (LSB) through  $\Sigma_4$  (MSB) are the sum outputs. The logic symbol is shown in Figure 6-9(b).

In terms of the method used to handle carries in a parallel adder, there are two types: the *ripple carry* adder and the *carry look-ahead* adder. These are discussed in Section 6-3.

## Adder Expansion

The 4-bit parallel adder can be expanded to handle the addition of two 8-bit numbers by using two 4-bit adders. The carry input of the low-order adder ( $C_0$ ) is connected to ground because there is no carry into the least significant bit position, and the carry output of the low-order adder is connected to the carry input of the high-order adder, as shown in Figure 6–11. This process is known as **cascading**. Notice that, in this case, the output carry is designated  $C_8$  because it is generated from the eighth bit position. The low-order adder is



**FIGURE 6–11** Cascading of two 4-bit adders to form an 8-bit adder.

the one that adds the lower or less significant four bits in the numbers, and the high-order adder is the one that adds the higher or more significant four bits in the 8-bit numbers. Similarly, four 4-bit adders can be cascaded to handle two 16-bit numbers.

## 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. This type of system can be used where a group of people are assembled and there is a need for immediately determining opinions (for or against), making decisions, or voting on certain issues or other matters.

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 6–13 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.



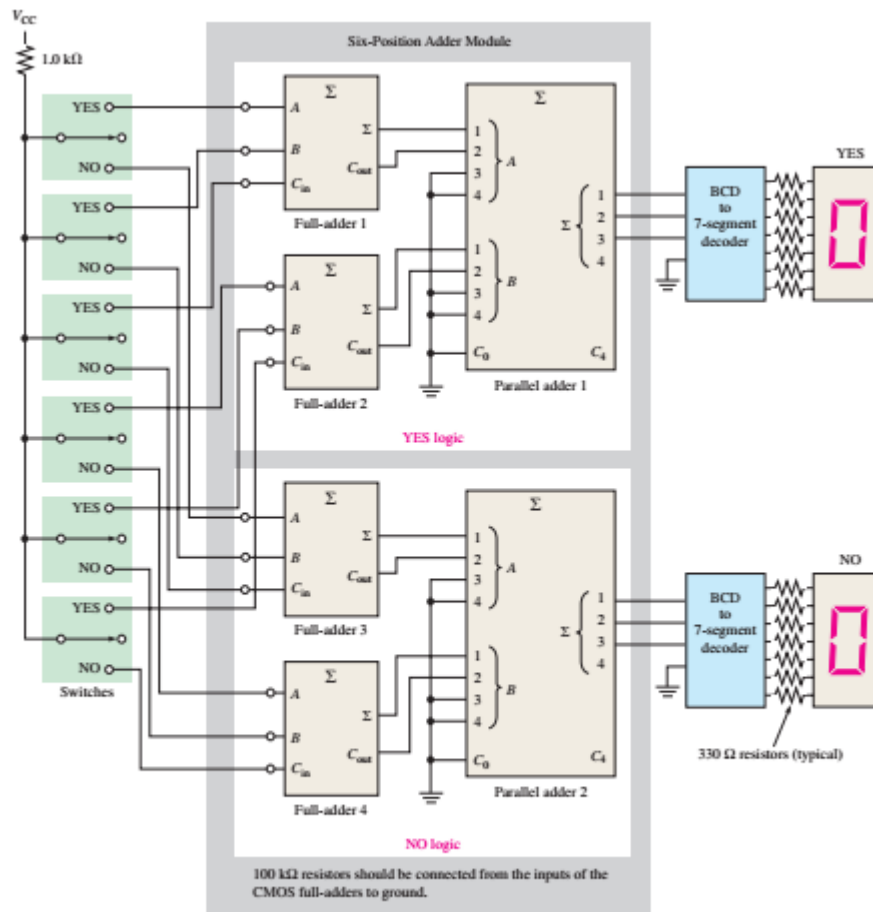


FIGURE 6-13 A voting system using full-adders and parallel binary adders.

In Figure 6-13 each full-adder can produce the sum of up to three votes. The sum and output carry of each full-adder then goes to the two lower-order inputs of a parallel binary adder. 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). For this basic 6-position system, the outputs of the parallel adder go to a BCD-to-7-segment decoder that drives the 7-segment display. As mentioned, additional circuits must be included when the system is expanded.

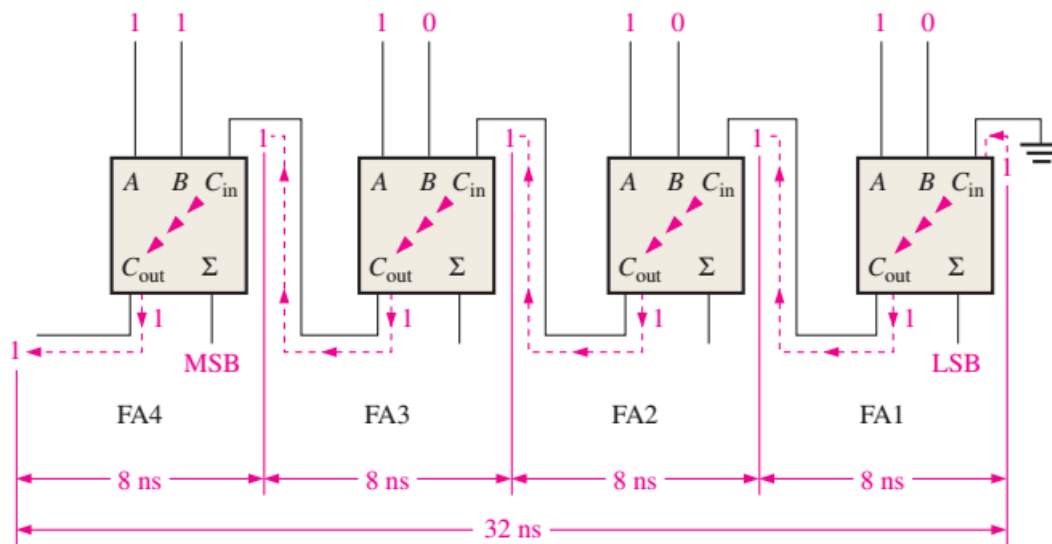
The resistors from the inputs of each full-adder to ground assure that each input is LOW when the switch is in the neutral position (CMOS logic is used). When a switch is moved to the “yes” or to the “no” position, a HIGH level ( $V_{CC}$ ) is applied to the associated full-adder input.

## 6-3 Ripple Carry and Look-Ahead Carry Adders

As mentioned in the last section, parallel adders can be placed into two categories based on the way in which internal carries from stage to stage are handled. Those categories are ripple carry and look-ahead carry. Externally, both types of adders are the same in terms of inputs and outputs. The difference is the speed at which they can add numbers. The look-ahead carry adder is much faster than the ripple carry adder.

## 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 6–14. 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 6–14** A 4-bit parallel ripple carry adder showing “worst-case” carry propagation delays.

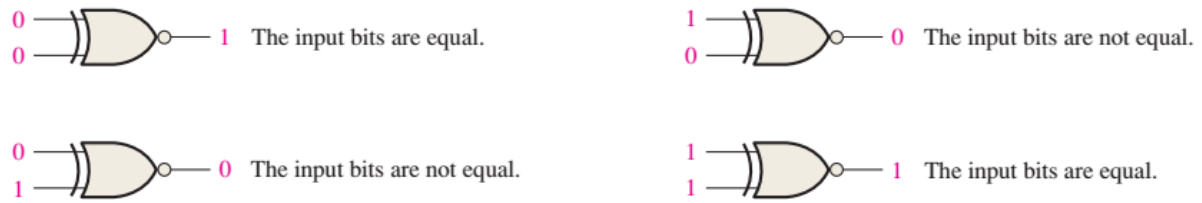
Full-adder 1 (FA1) cannot produce a potential output carry until an input carry is applied. Full-adder 2 (FA2) cannot produce a potential output carry until FA1 produces an output carry. Full-adder 3 (FA3) cannot produce a potential output carry until an output carry is produced by FA1 followed by an output carry from FA2, and so on. As you can see in Figure 6–14, the input carry to the least significant stage has to ripple through all the adders before a final sum is produced. The cumulative delay through all the adder stages is a “worst-case” addition time. 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; however, worst-case addition time must always be assumed.

## 6–4 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.

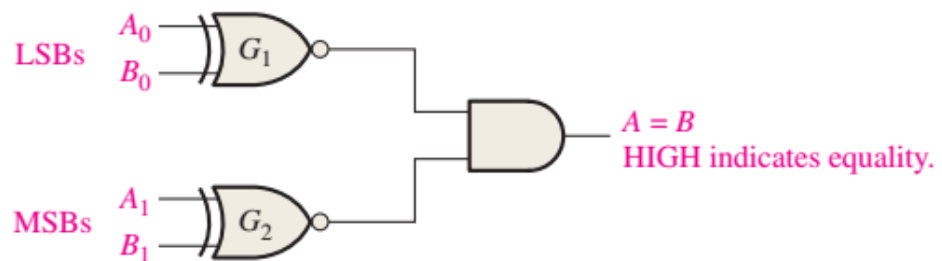
### Equality

As you learned in Chapter 3, the exclusive-NOR gate can be used as a basic comparator because its output is a 0 if the two input bits are not equal and a 1 if the input bits are equal. Figure 6–18 shows the exclusive-NOR gate as a 2-bit comparator.



**FIGURE 6-18** Basic comparator operation.

In order to compare binary numbers containing two bits each, an additional exclusive-NOR 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 6-19. If the two numbers are equal, their corresponding bits are the same, and the output of each exclusive-NOR gate is a 1. If the corresponding sets of bits are not equal, a 0 occurs on that exclusive-NOR gate output.



General format: Binary number  $A \rightarrow A_1A_0$   
 Binary number  $B \rightarrow B_1B_0$

**FIGURE 6-19** Logic diagram for equality comparison of two 2-bit numbers.

In order to produce a single output indicating an equality or inequality of two numbers, an AND gate can be combined with XNOR gates, as shown in Figure 6-19. The output of each exclusive-NOR gate is applied to the AND gate input. When the two input bits for each exclusive-NOR are equal, the corresponding bits of the numbers are equal, producing a 1 on both inputs to the AND gate and thus a 1 on the output. 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. Thus, the output of the AND gate indicates equality (1) or inequality (0) of the two numbers. Example 6-5 illustrates this operation for two specific cases.

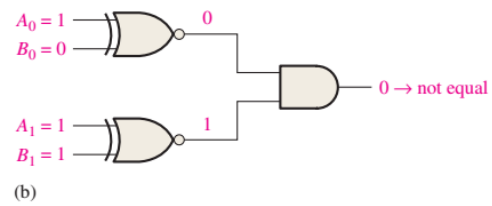
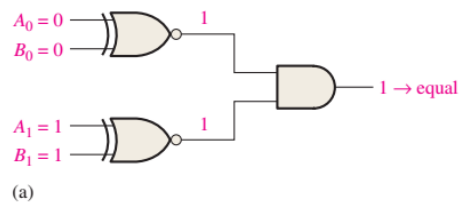
As you know from Chapter 3, the basic comparator can be expanded to any number of bits. The AND gate sets the condition that all corresponding bits of the two numbers must be equal if the two numbers themselves are equal.

**EXAMPLE 6-5**

Apply each of the following sets of binary numbers to the comparator inputs in Figure 6-20, and determine the output by following the logic levels through the circuit.

(a) 10 and 10

(b) 11 and 10



**FIGURE 6-20**

## Solution

(a) The output is **1** for inputs 10 and 10, as shown in Figure 6-20(a).

(b) The output is **0** for inputs 11 and 10, as shown in Figure 6-20(b).

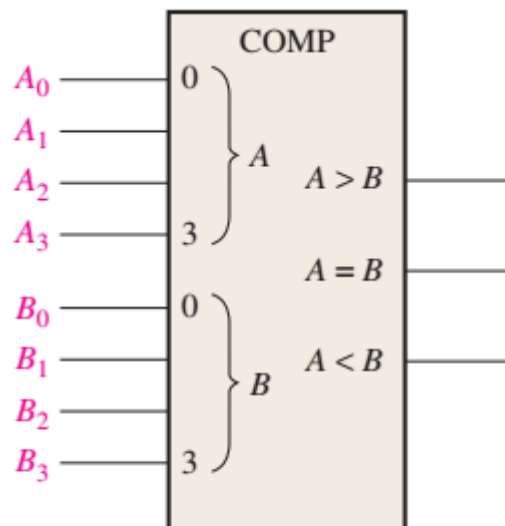
## Inequality

In addition to the equality output, fixed-function comparators can 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 6–21.

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.

These three operations are valid for each bit position in the numbers. The general procedure used in a comparator is to check for an inequality in a bit position, starting with the highest-order bits (MSBs). When such an inequality is found, the relationship of the two numbers is established, and any other inequalities in lower-order bit positions must be ignored because it is possible for an opposite indication to occur; *the highest-order indication must take precedence*.



**FIGURE 6–21** Logic symbol for a 4-bit comparator with inequality indication.

### EXAMPLE 6-6

Determine the  $A = B$ ,  $A > B$ , and  $A < B$  outputs for the input numbers shown on the comparator in Figure 6-22.

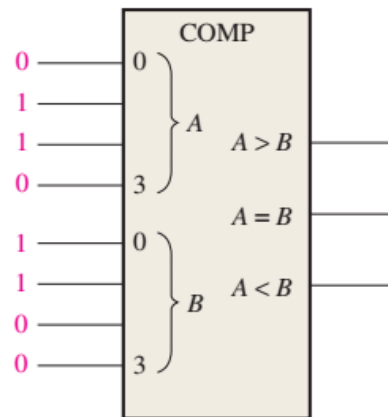


FIGURE 6-22

#### 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**.

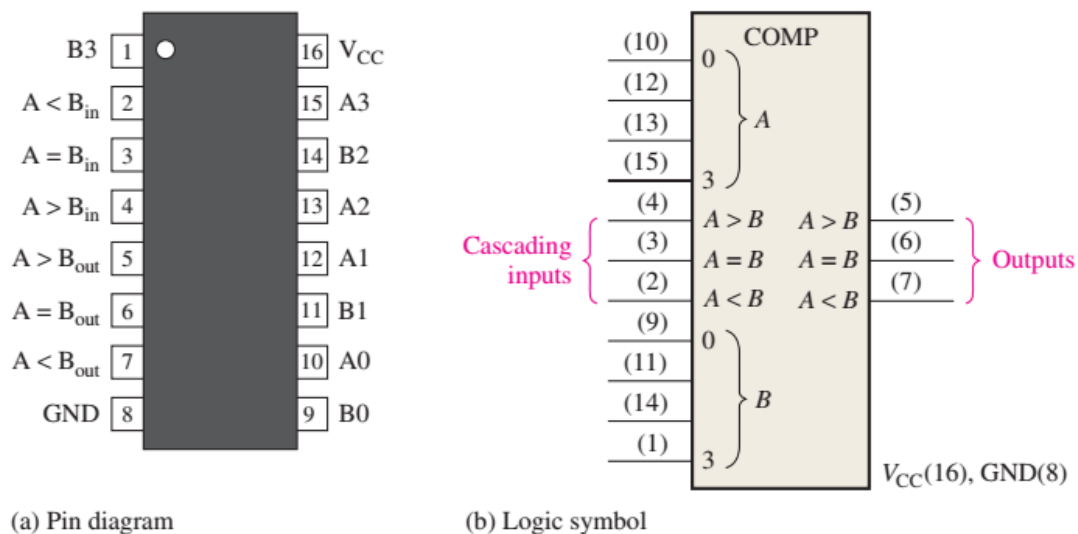
#### Related Problem

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

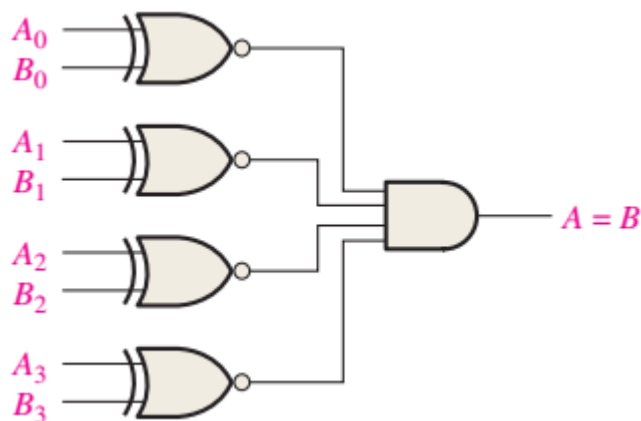


## 4-BIT MAGNITUDE COMPARATOR

**Fixed-Function Device** The 74HC85/74LS85 pin diagram and logic symbol are shown in Figure 6–23. Notice that this device has all the inputs and outputs of the generalized comparator previously discussed and, in addition, has three cascading inputs:  $A < B$ ,  $A = B$ ,  $A > B$ . These inputs allow several comparators to be cascaded for comparison of any number of bits greater than four. To expand the comparator, the  $A < B$ ,  $A = B$ , and  $A > B$  outputs of the lower-order comparator are connected to the corresponding cascading inputs of the next higher-order comparator. The lowest-order comparator must have a HIGH on the  $A = B$  input and LOWs on the  $A < B$  and  $A > B$  inputs.



**FIGURE 6–23** The 74HC85/74LS85 4-bit magnitude comparator.



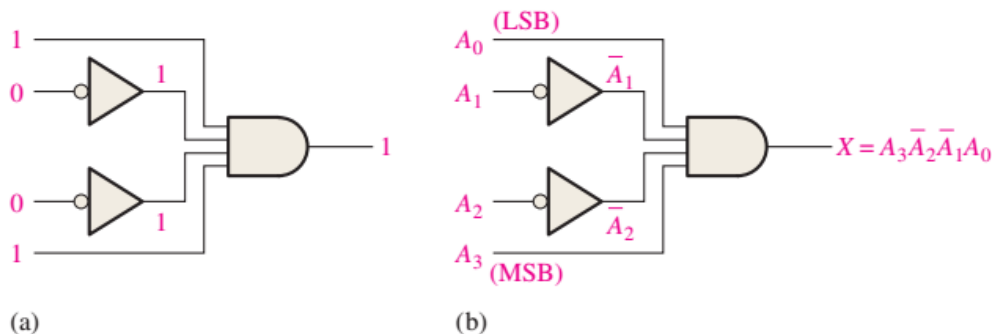
**FIGURE 6–24**

## 6-5 Decoders

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. In this section, three fixed-function IC decoders are introduced. The basic principles can be extended to other types of decoders.

### The Basic Binary Decoder

Suppose you need to determine when a binary 1001 occurs on the inputs of a digital circuit. 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. Therefore, you must make sure that all of the inputs to the AND gate are HIGH when the binary number 1001 occurs; this can be done by inverting the two middle bits (the 0s), as shown in Figure 6-26.



**FIGURE 6-26** Decoding logic for the binary code 1001 with an active-HIGH output.

The logic equation for the decoder of Figure 6-26(a) is developed as illustrated in Figure 6-26(b). You should verify that the output is 0 except when  $A_0 = 1$ ,  $A_1 = 0$ ,  $A_2 = 0$ , and  $A_3 = 1$  are applied to the inputs.  $A_0$  is the LSB and  $A_3$  is the MSB. *In the representation of a binary number or other weighted code in this book, the LSB is the right-most bit in a horizontal arrangement and the topmost bit in a vertical arrangement, unless specified otherwise.*

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

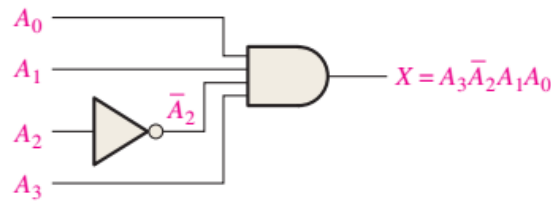
Determine the logic required to decode the binary number 1011 by producing a HIGH level on the output.

#### Solution

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

$$X = A_3\bar{A}_2A_1A_0 \quad (1011)$$

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.

### Related Problem

Develop the logic required to detect the binary code 10010 and produce an active-LOW output.

## 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.

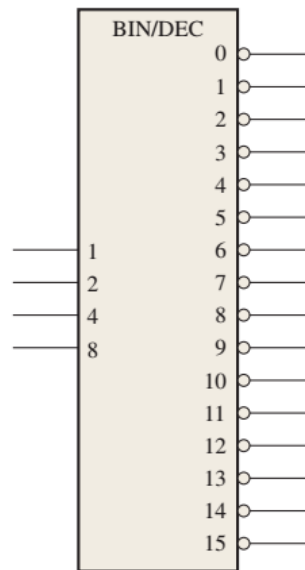
**TABLE 6-4**

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

Decimal Digit	Binary Inputs				Decoding Function	Outputs															
	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>		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	$\overline{A_3}\overline{A_2}\overline{A_1}A_0$	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	1	0	$\overline{A_3}\overline{A_2}A_1\overline{A_0}$	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	1	1	$\overline{A_3}\overline{A_2}A_1A_0$	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
4	0	1	0	0	$\overline{A_3}A_2\overline{A_1}\overline{A_0}$	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
5	0	1	0	1	$\overline{A_3}A_2\overline{A_1}A_0$	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
6	0	1	1	0	$\overline{A_3}A_2A_1\overline{A_0}$	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
7	0	1	1	1	$\overline{A_3}A_2A_1A_0$	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	$A_3\overline{A_2}\overline{A_1}A_0$	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
10	1	0	1	0	$A_3\overline{A_2}A_1\overline{A_0}$	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
11	1	0	1	1	$A_3\overline{A_2}A_1A_0$	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	$A_3A_2\overline{A_1}A_0$	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

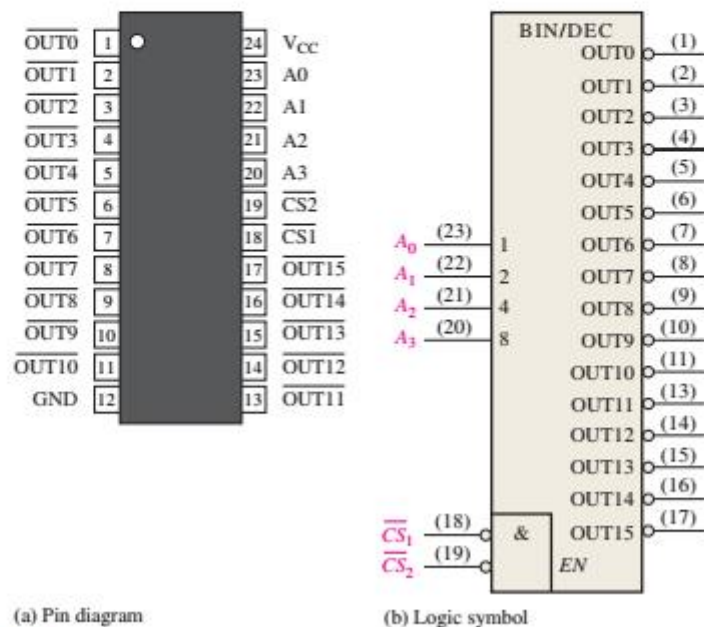
If an active-LOW output is required for each decoded number, the entire decoder can be implemented with NAND gates and inverters. In order to decode each of the sixteen binary codes, sixteen NAND gates are required (AND gates can be used to produce active-HIGH outputs).

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 ( $2^32^22^12^0$ ).



**FIGURE 6-28** Logic symbol for a 4-line-to-16-line (1-of-16) decoder.  
**1-OF-16 DECODER**

**Fixed-Function Device** The 74HC154 is a good example of a fixed-function IC decoder. The logic symbol is shown in Figure 6-29. There is an enable function (*EN*) provided on this device, which is implemented with a NOR gate used as a negative-AND. A LOW level on each chip select input,  $\overline{CS}_1$  and  $\overline{CS}_2$ , is required in order to make the enable gate output (*EN*) HIGH. The enable gate output is connected to an input of *each* NAND gate in the decoder, so it must be HIGH for the NAND gates to be enabled. If the enable gate is not activated by a LOW on both inputs, then all sixteen decoder outputs (*OUT*) will be HIGH regardless of the states of the four input variables,  $A_0$ ,  $A_1$ ,  $A_2$ , and  $A_3$ .



**FIGURE 6-29** The 74HC154 1-of-16 decoder.

### EXAMPLE 6-9

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  $A_4A_3A_2A_1A_0$ .

### Solution

Since the 74HC154 can handle only four bits, two decoders must be used to form a 5-bit expansion. 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.

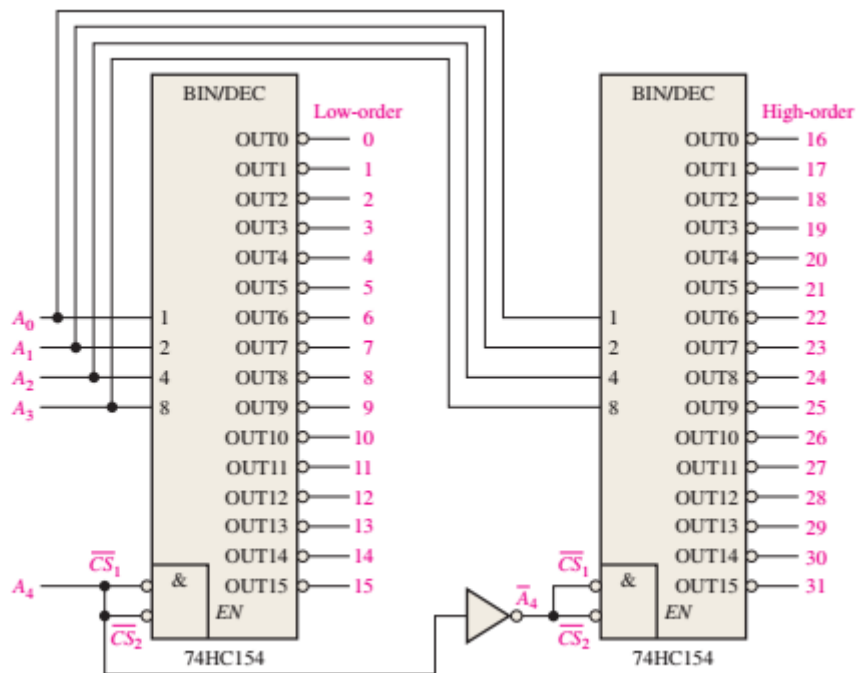


FIGURE 6-30 A 5-bit decoder using 74HC154s.

## 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*.

The method of implementation is the same as for the 1-of-16 decoder previously discussed, except that only ten decoding gates are required because the BCD code represents only the ten decimal digits 0 through 9. A list of the ten BCD codes and their corresponding decoding functions is given in Table 6-5. 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. The logic is identical to that of the first ten decoding gates in the 1-of-16 decoder (see Table 6-4).



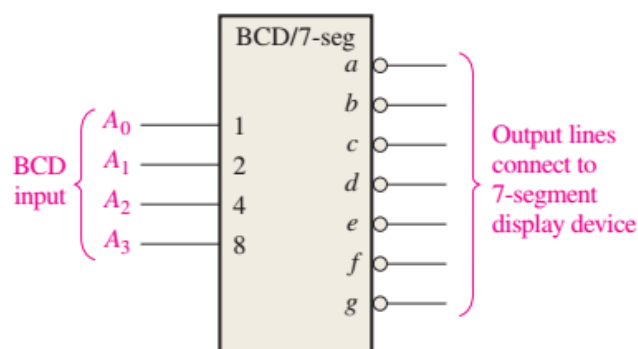
**TABLE 6-5**

BCD decoding functions.

Decimal Digit	BCD Code				Decoding Function
	$A_3$	$A_2$	$A_1$	$A_0$	
0	0	0	0	0	$\bar{A}_3\bar{A}_2\bar{A}_1\bar{A}_0$
1	0	0	0	1	$\bar{A}_3\bar{A}_2\bar{A}_1A_0$
2	0	0	1	0	$\bar{A}_3\bar{A}_2A_1\bar{A}_0$
3	0	0	1	1	$\bar{A}_3\bar{A}_2A_1A_0$
4	0	1	0	0	$\bar{A}_3A_2\bar{A}_1\bar{A}_0$
5	0	1	0	1	$\bar{A}_3A_2\bar{A}_1A_0$
6	0	1	1	0	$\bar{A}_3A_2A_1\bar{A}_0$
7	0	1	1	1	$\bar{A}_3A_2A_1A_0$
8	1	0	0	0	$A_3\bar{A}_2\bar{A}_1\bar{A}_0$
9	1	0	0	1	$A_3\bar{A}_2\bar{A}_1A_0$

## The BCD-to-7-Segment Decoder

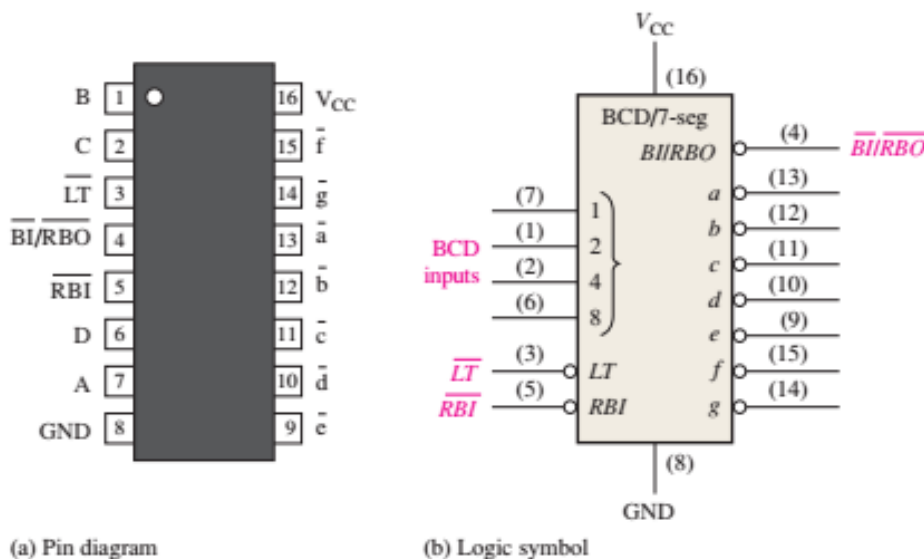
The BCD-to-7-segment decoder accepts the BCD code on its inputs and provides outputs to drive 7-segment display devices to produce a decimal readout. The logic diagram for a basic 7-segment decoder is shown in Figure 6-33.

**FIGURE 6-33** Logic symbol for a BCD-to-7-segment decoder/driver with active-LOW



## BCD-TO-7-SEGMENT DECODER/DRIVER

**Fixed-Function Device** The 74HC47 is an example of an IC device that decodes a BCD input and drives a 7-segment display. In addition to its decoding and segment drive capability, the 74HC47 has several additional features as indicated by the  $\overline{LT}$ ,  $\overline{RBI}$ ,  $\overline{BI}/\overline{RBO}$  functions in the logic symbol of Figure 6-34. As indicated by the bubbles on the logic symbol, all of the outputs ( $a$  through  $g$ ) are active-LOW as are the  $\overline{LT}$  (lamp test),  $\overline{RBI}$  (ripple blanking input), and  $\overline{BI}/\overline{RBO}$  (blanking input/ripple blanking output) functions. The outputs can drive a common-anode 7-segment display directly. Recall that 7-segment displays were discussed in Chapter 4. In addition to decoding a BCD input and producing the appropriate 7-segment outputs, the 74HC47 has lamp test and zero suppression capability.



**FIGURE 6-34** The 74HC47 BCD-to-7-segment decoder/driver.

**Lamp Test** When a LOW is applied to the  $\overline{LT}$  input and the  $\overline{BI}/\overline{RBO}$  is HIGH, all of the seven segments in the display are turned on. Lamp test is used to verify that no segments are burned out.

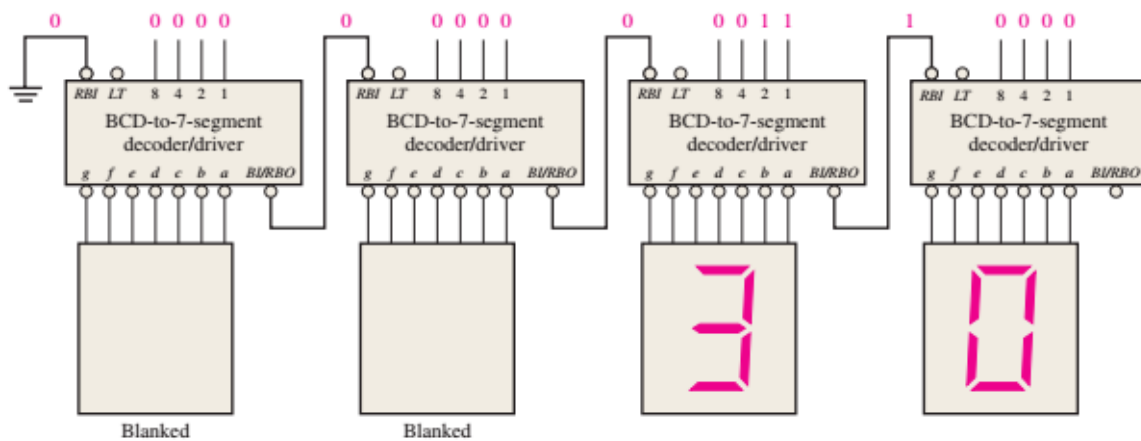
**Zero Suppression** **Zero suppression** is a feature used for multidigit displays to blank out unnecessary zeros. For example, in a 6-digit display the number 6.4 may be displayed as 006.400 if the zeros are not blanked out. Blanking the zeros at the front of a number is called *leading zero suppression* and blanking the zeros at the back of the number is called *trailing zero suppression*. Keep in mind that only nonessential zeros are blanked. With zero suppression, the number 030.080 will be displayed as 30.08 (the essential zeros remain).

Zero suppression in the 74HC47 is accomplished using the  $\overline{RBI}$  and  $\overline{BI}/\overline{RBO}$  functions.  $\overline{RBI}$  is the ripple blanking input and  $\overline{RBO}$  is the ripple blanking output on the 74HC47; these are used for zero suppression.  $\overline{BI}$  is the blanking input that shares the same pin with  $\overline{RBO}$ ; in other words, the  $\overline{BI}/\overline{RBO}$  pin can be used as an input or an output. When used as a  $\overline{BI}$  (blanking input), all segment outputs are HIGH (nonactive) when  $\overline{BI}$  is LOW, which overrides all other inputs. The  $\overline{BI}$  function is not part of the zero suppression capability of the device.

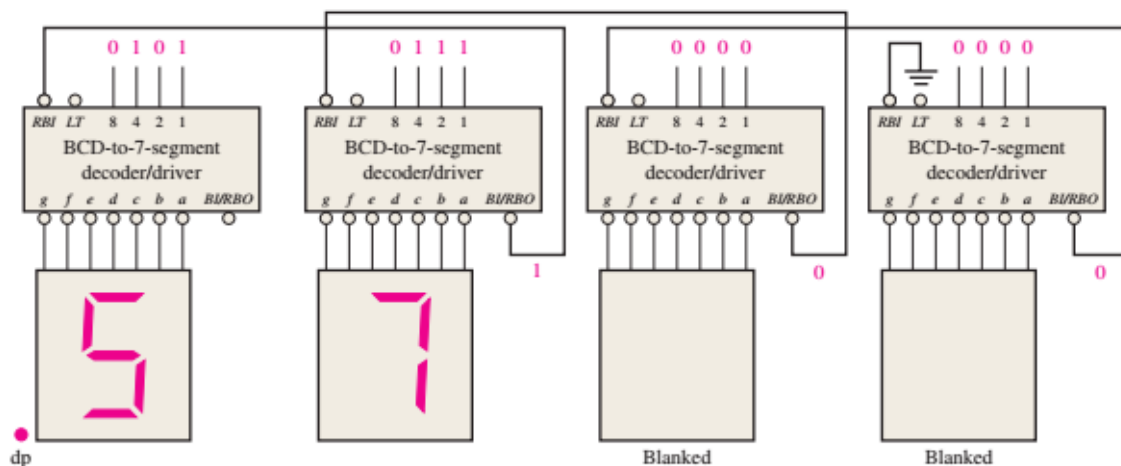
All of the segment outputs of the decoder are nonactive (HIGH) if a zero code (0000) is on its BCD inputs and if its  $\overline{RBI}$  is LOW. This causes the display to be blank and produces a LOW  $\overline{RBO}$ .

## Zero Suppression for a 4-Digit Display

The logic diagram in Figure 6–35(a) illustrates leading zero suppression for a whole number. The highest-order digit position (left-most) is always blanked if a zero code is on its BCD inputs because the  $\overline{RBI}$  of the most-significant decoder is made LOW by connecting it to ground. The  $\overline{RBO}$  of each decoder is connected to the  $\overline{RBI}$  of the next lowest-order decoder so that all zeros to the left of the first nonzero digit are blanked. For example, in part (a) of the figure the two highest-order digits are zeros and therefore are blanked. The remaining two digits, 3 and 0 are displayed.



(a) Illustration of leading zero suppression



(b) Illustration of trailing zero suppression

**FIGURE 6–35** Examples of zero suppression using a BCD-to-7-segment decoder/driver

The logic diagram in Figure 6–35(b) illustrates trailing zero suppression for a fractional number. The lowest-order digit (right-most) is always blanked if a zero code is on its BCD inputs because the  $\overline{RBI}$  is connected to ground. The  $\overline{RBO}$  of each decoder is connected to the  $\overline{RBI}$  of the next highest-order decoder so that all zeros to the right of the first nonzero digit are blanked. In part (b) of the figure, the two lowest-order digits are zeros and therefore are blanked. The remaining two digits, 5 and 7 are displayed. To combine both leading and trailing zero suppression in one display and to have decimal point capability, additional logic is required.

## 6-6 Encoders

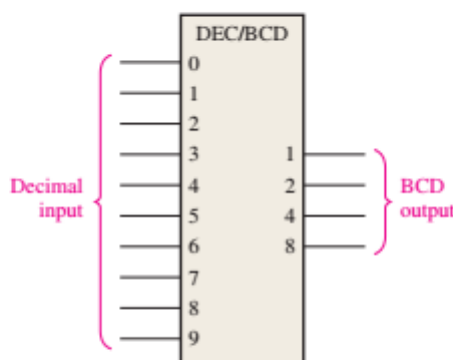
An **encoder** is a combinational logic circuit that essentially performs a “reverse” decoder function. An encoder accepts an active level on one of its inputs representing a digit, such as a decimal or octal digit, and converts it to a coded output, such as BCD or binary. Encoders can also be devised to encode various symbols and alphabetic characters. The process of converting from familiar symbols or numbers to a coded format is called *encoding*.

**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

### 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 6-36. This is a basic 10-line-to-4-line encoder.



**FIGURE 6-36** Logic symbol for a decimal-to-BCD encoder.

The BCD (8421) code is listed in Table 6-6. From this table you can determine the relationship between each BCD bit and the decimal digits in order to analyze the logic. For instance, the most significant bit of the BCD code,  $A_3$ , is always a 1 for decimal digit 8 or 9. An OR expression for bit  $A_3$  in terms of the decimal digits can therefore be written as

$$A_3 = 8 + 9$$

Bit  $A_2$  is always a 1 for decimal digit 4, 5, 6 or 7 and can be expressed as an OR function as follows:

$$A_2 = 4 + 5 + 6 + 7$$

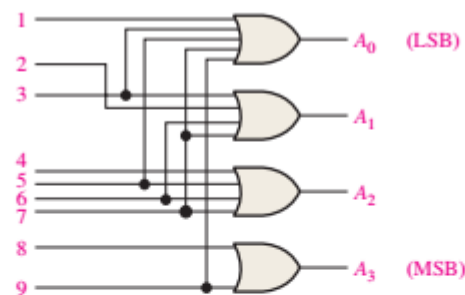
Bit  $A_1$  is always a 1 for decimal digit 2, 3, 6, or 7 and can be expressed as

$$A_1 = 2 + 3 + 6 + 7$$

Finally,  $A_0$  is always a 1 for decimal digit 1, 3, 5, 7, or 9. The expression for  $A_0$  is

$$A_0 = 1 + 3 + 5 + 7 + 9$$

Now let's implement the logic circuitry required for encoding each decimal digit to a BCD code by using the logic expressions just developed. It is simply a matter of ORing the appropriate decimal digit input lines to form each BCD output. The basic encoder logic resulting from these expressions is shown in Figure 6-37.



**FIGURE 6-37** 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 6-37 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.

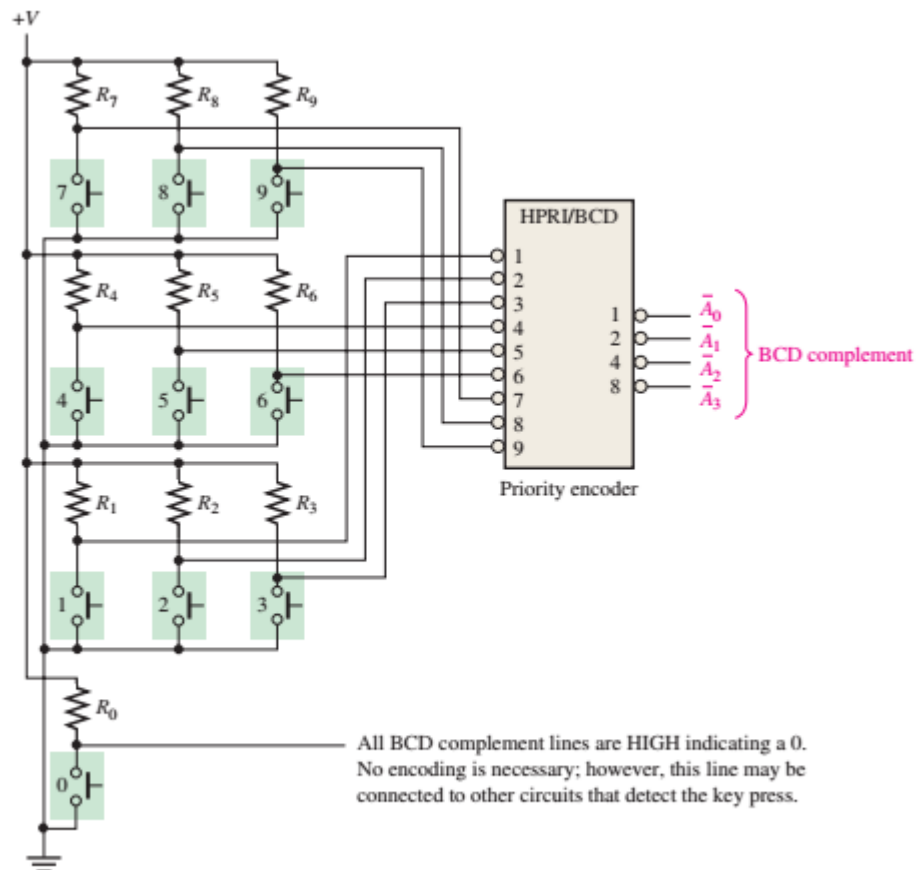
### The Decimal-to-BCD Priority Encoder

This type of encoder performs the same basic encoding function as previously discussed. 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).

### An Application

The ten decimal digits on a numeric keypad must be encoded for processing by the logic circuitry. In this example, when one of the keys is pressed, the decimal digit is encoded to the corresponding BCD code. Figure 6-39 shows a simple keyboard encoder arrangement using a priority encoder. The keys are represented by ten push-button switches, each with a **pull-up resistor** to +V. The pull-up resistor ensures that the line is HIGH when a key is not depressed. When a key is depressed, the line is connected to ground, and a LOW is applied to the corresponding encoder input. The zero key is not connected because the BCD output represents zero when none of the other keys is depressed.

The BCD complement output of the encoder goes into a storage device, and each successive BCD code is stored until the entire number has been entered. Methods of storing BCD numbers and binary data are covered in Chapter 11.



**FIGURE 6-39** A simplified keyboard encoder.

## 6-7 Code Converters

In this section, we will examine some methods of using combinational logic circuits to convert from one code to another.



## BCD-to-Binary Conversion

One method of BCD-to-binary code conversion uses adder circuits. The basic conversion process is as follows:

1. The value, or weight, of each bit in the BCD number is represented by a binary number.
2. All of the binary representations of the weights of bits that are 1s in the BCD number are added.
3. The result of this addition is the binary equivalent of the BCD number.

A more concise statement of this operation is

**The binary numbers representing the weights of the BCD bits are summed to produce the total binary number.**

Let's examine an 8-bit BCD code (one that represents a 2-digit decimal number) to understand the relationship between BCD and binary. For instance, you already know that the decimal number 87 can be expressed in BCD as

$$\begin{array}{r} \underline{1000} \quad \underline{0111} \\ 8 \quad \quad 7 \end{array}$$

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:	$B_3$	$B_2$	$B_1$	$B_0$	$A_3$	$A_2$	$A_1$	$A_0$

The binary equivalent of each BCD bit is a binary number representing the weight of that bit within the total BCD number. This representation is given in Table 6-7.

**TABLE 6-7**

Binary representations of BCD bit weights.

BCD Bit	BCD Weight	Binary Representation						
		(MSB) 64	32	16	8	4	2	(LSB) 1
$A_0$	1	0	0	0	0	0	0	1
$A_1$	2	0	0	0	0	0	1	0
$A_2$	4	0	0	0	0	1	0	0
$A_3$	8	0	0	0	1	0	0	0
$B_0$	10	0	0	0	1	0	1	0
$B_1$	20	0	0	1	0	1	0	0
$B_2$	40	0	1	0	1	0	0	0
$B_3$	80	1	0	1	0	0	0	0

If the binary representations for the weights of all the 1s in the BCD number are added, the result is the binary number that corresponds to the BCD number. Example 6-12 illustrates this.



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

The basic process for Gray-binary conversions was covered in Chapter 2. Exclusive-OR gates can be used for these conversions. Programmable logic devices (PLDs) can also be programmed for these code conversions. Figure 6–40 shows a 4-bit binary-to-Gray code converter, and Figure 6–41 illustrates a 4-bit Gray-to-binary converter.



### EXAMPLE 6-13

- Convert the binary number 0101 to Gray code with exclusive-OR gates.
- Convert the Gray code 1011 to binary with exclusive-OR gates.

#### Solution

- 0101<sub>2</sub> is 0111 Gray. See Figure 6–42(a).
- 1011 Gray is 1101<sub>2</sub>. See Figure 6–42(b).

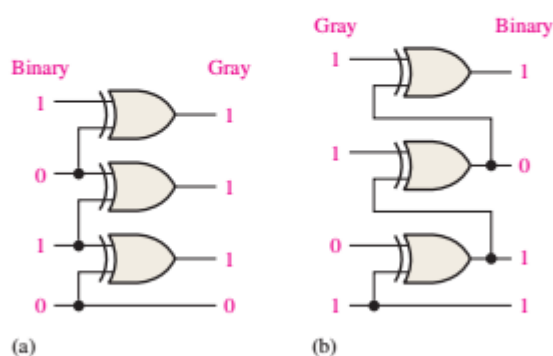


FIGURE 6-42

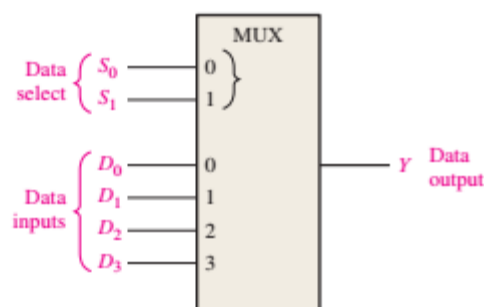
#### Related Problem

How many exclusive-OR gates are required to convert 8-bit binary to Gray?

## 6-8 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 6-43. Notice that there are two data-select lines because with two select bits, any one of the four data-input lines can be selected.



**FIGURE 6-43** Logic symbol for a 1-of-4 data selector/multiplexer.

In Figure 6-43, a 2-bit code on the data-select ( $S$ ) inputs will allow the data on the selected data input to pass through to the data output. If a binary 0 ( $S_1 = 0$  and  $S_0 = 0$ ) is applied to the data-select lines, the data on input  $D_0$  appear on the data-output line. If a binary 1 ( $S_1 = 0$  and  $S_0 = 1$ ) is applied to the data-select lines, the data on input  $D_1$  appear on the data output. If a binary 2 ( $S_1 = 1$  and  $S_0 = 0$ ) is applied, the data on  $D_2$  appear on the output. If a binary 3 ( $S_1 = 1$  and  $S_0 = 1$ ) is applied, the data on  $D_3$  are switched to the output line. A summary of this operation is given in Table 6-8.

**TABLE 6-8**

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

Data-Select Inputs		Input Selected
$S_1$	$S_0$	
0	0	$D_0$
0	1	$D_1$
1	0	$D_2$
1	1	$D_3$

In Figure 6-43, a 2-bit code on the data-select ( $S$ ) inputs will allow the data on the selected data input to pass through to the data output. If a binary 0 ( $S_1 = 0$  and  $S_0 = 0$ ) is applied to the data-select lines, the data on input  $D_0$  appear on the data-output line. If a binary 1 ( $S_1 = 0$  and  $S_0 = 1$ ) is applied to the data-select lines, the data on input  $D_1$  appear on the data output. If a binary 2 ( $S_1 = 1$  and  $S_0 = 0$ ) is applied, the data on  $D_2$  appear on the output. If a binary 3 ( $S_1 = 1$  and  $S_0 = 1$ ) is applied, the data on  $D_3$  are switched to the output line. A summary of this operation is given in Table 6-8.

Now let's look at the logic circuitry required to perform this multiplexing operation. The data output is equal to the state of the *selected* data input. You 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  $S_0 = 0$ :  $Y = D_0\bar{S}_1\bar{S}_0$ .

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

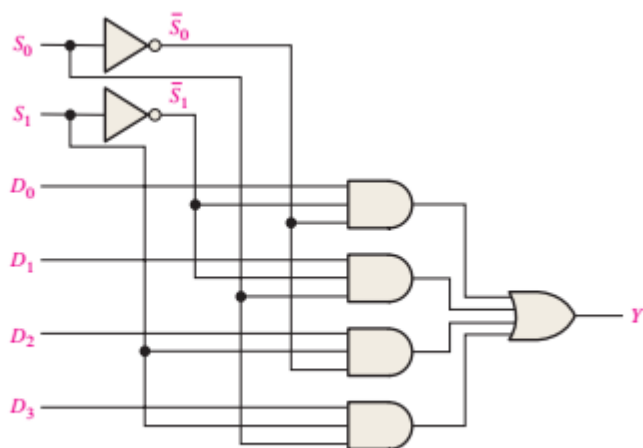
The data output is equal to  $D_2$  only if  $S_1 = 1$  and  $S_0 = 0$ :  $Y = D_2S_1\bar{S}_0$ .

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

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

The implementation of this equation requires four 3-input AND gates, a 4-input OR gate, and two inverters to generate the complements of  $S_1$  and  $S_0$ , as shown in Figure 6–44. Because data can be selected from any one of the input lines, this circuit is also referred to as a **data selector**.



**FIGURE 6–44** Logic diagram for a 4-input multiplexer. Open file F06-44 to

#### EXAMPLE 6-14

The data-input and data-select waveforms in Figure 6-45(a) are applied to the multiplexer in Figure 6-44. Determine the output waveform in relation to the inputs.

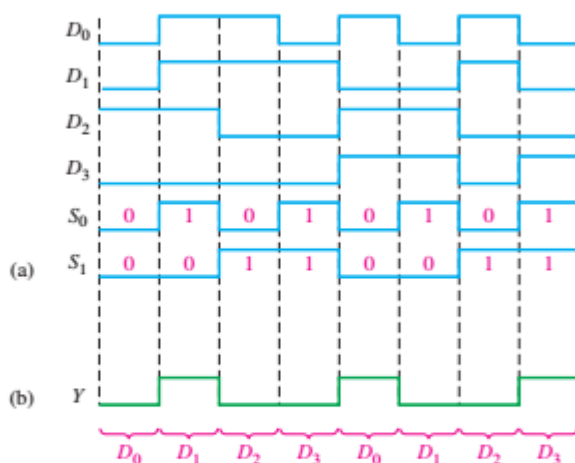


FIGURE 6-45

#### 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 6-45(b).

When the data-select line goes HIGH, the  $B$  bits ( $B_3B_2B_1B_0$ ) are passed through to the inputs of the BCD-to-7-segment decoder. Also, the 74HC139 decoder's 1 output is activated, thus enabling the  $B$ -digit display. The  $B$  digit is now *on* and the  $A$  digit is *off*. The cycle repeats at the frequency of the data-select square wave. This frequency must be high enough to prevent visual flicker as the digit displays are multiplexed.

### A Logic Function Generator

A useful application of the data selector/multiplexer is in the generation of combinational logic functions in sum-of-products form. When used in this way, the device can replace discrete gates, can often greatly reduce the number of ICs, and can make design changes much easier.

To illustrate, a 74HC151 8-input data selector/multiplexer can be used to implement any specified 3-variable logic function if the variables are connected to the data-select inputs and each data input is set to the logic level required in the truth table for that function. For example, if the function is a 1 when the variable combination is  $\bar{A}_2A_1\bar{A}_0$ , the 2 input (selected by 010) is connected to a HIGH. This HIGH is passed through to the output when this particular combination of variables occurs on the data-select lines. Example 6-16 will help clarify this application.

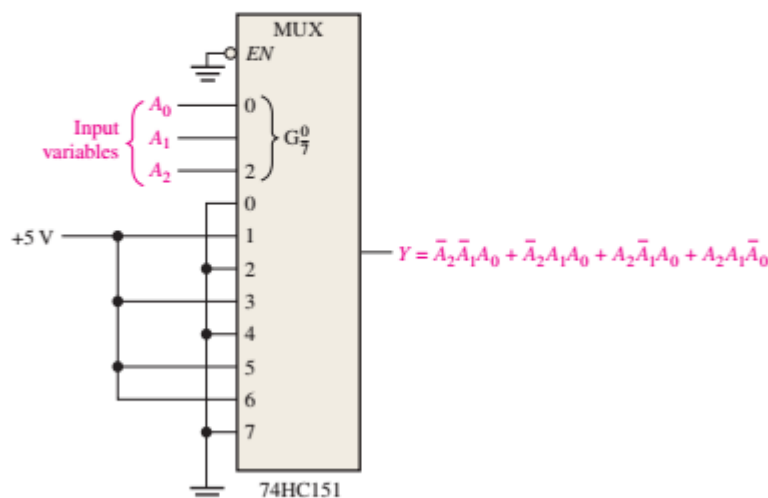
**EXAMPLE 6-16**

Implement the logic function specified in Table 6-9 by using a 74HC151 8-input data selector/multiplexer. Compare this method with a discrete logic gate implementation.

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

**Solution**

Notice from the truth table that  $Y$  is a 1 for the following input variable combinations: 001, 011, 101, and 110. For all other combinations,  $Y$  is 0. For this function to be implemented with the data selector, the data input selected by each of the above-mentioned combinations must be connected to a HIGH (5 V). All the other data inputs must be connected to a LOW (ground), as shown in Figure 6-50.



**FIGURE 6-50** 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.

Example 6–16 illustrated how the 8-input data selector can be used as a logic function generator for three variables. Actually, this device can be also used as a 4-variable logic function generator by the utilization of one of the bits ( $A_0$ ) in conjunction with the data inputs.

A 4-variable truth table has sixteen combinations of input variables. When an 8-bit data selector is used, each input is selected twice: the first time when  $A_0$  is 0 and the second time when  $A_0$  is 1. With this in mind, the following rules can be applied ( $Y$  is the output, and  $A_0$  is the least significant bit):

1. If  $Y = 0$  both times a given data input is selected by a certain combination of the input variables,  $A_3A_2A_1$ , connect that data input to ground (0).
2. If  $Y = 1$  both times a given data input is selected by a certain combination of the input variables,  $A_3A_2A_1$ , connect the data input to  $+V(1)$ .
3. If  $Y$  is different the two times a given data input is selected by a certain combination of the input variables,  $A_3A_2A_1$ , and if  $Y = A_0$ , connect that data input to  $A_0$ .
4. If  $Y$  is different the two times a given data input is selected by a certain combination of the input variables,  $A_3A_2A_1$ , and if  $Y = \bar{A}_0$ , connect that data input to  $\bar{A}_0$ .

#### EXAMPLE 6-17

Implement the logic function in Table 6–10 by using a 74HC151 8-input data selector/multiplexer. Compare this method with a discrete logic gate implementation.

**TABLE 6-10**

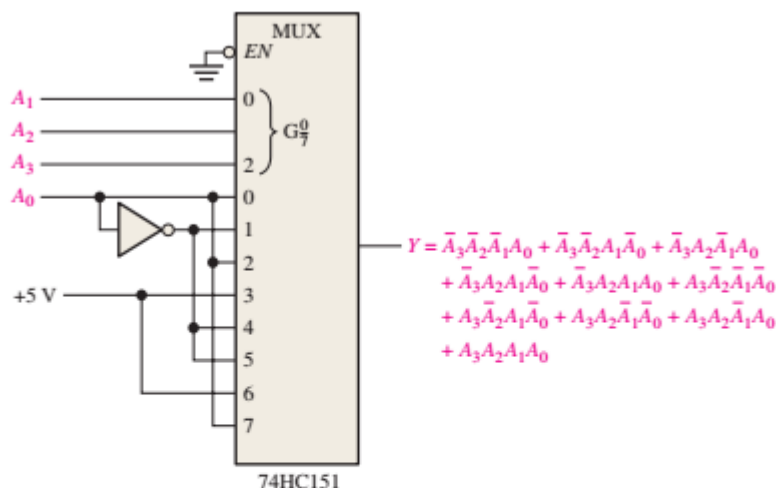
Decimal Digit	Inputs				Output
	$A_3$	$A_2$	$A_1$	$A_0$	$Y$
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

#### Solution

The data-select inputs are  $A_3A_2A_1$ . In the first row of the table,  $A_3A_2A_1 = 000$  and  $Y = A_0$ . In the second row, where  $A_3A_2A_1$  again is 000,  $Y = \bar{A}_0$ . Thus,  $A_0$  is connected to the 0 input. In the third row of the table,  $A_3A_2A_1 = 001$  and  $Y = \bar{A}_0$ . Also, in the fourth row, when  $A_3A_2A_1$  again is 001,  $Y = A_0$ . Thus,  $A_0$  is inverted and connected to the 1 input. This analysis is continued until each input is properly connected according to the specified rules. The implementation is shown in Figure 6–51.

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.





**FIGURE 6-51** Data selector/multiplexer connected as a 4-variable logic function generator.

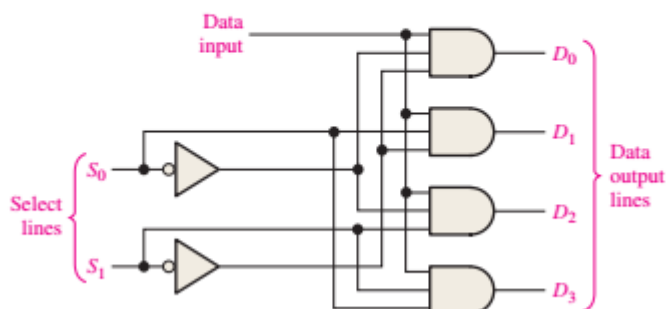
### Related Problem

In Table 6-10, if  $Y = 0$  when the inputs are all zeros and is alternately a 1 and a 0 for the remaining rows in the table, use a 74HC151 to implement the resulting logic function.

## 6-9 Demultiplexers

A **demultiplexer (DEMUX)** basically reverses the multiplexing function. It takes digital information from one line and distributes it to a given number of output lines. For this reason, the demultiplexer is also known as a data distributor. As you will learn, decoders can also be used as demultiplexers.

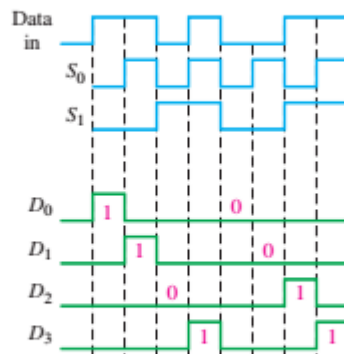
Figure 6-52 shows a 1-line-to-4-line demultiplexer (DEMUX) circuit. The data-input line goes to all of the AND gates. The two data-select lines enable only one gate at a time, and the data appearing on the data-input line will pass through the selected gate to the associated data-output line.



**FIGURE 6-52** A 1-line-to-4-line demultiplexer.

**EXAMPLE 6-18**

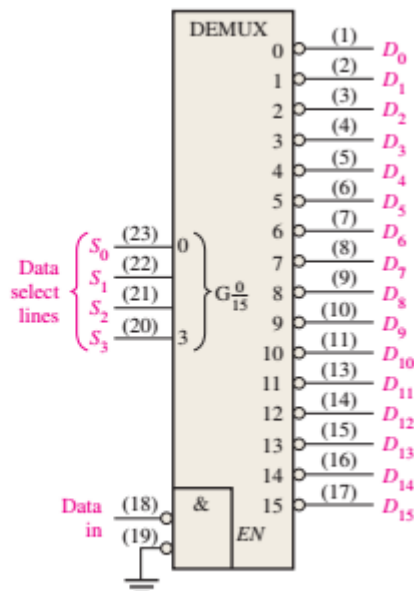
The serial data-input waveform (Data in) and data-select inputs ( $S_0$  and  $S_1$ ) are shown in Figure 6-53. Determine the data-output waveforms on  $D_0$  through  $D_3$  for the demultiplexer in Figure 6-52.

**FIGURE 6-53****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 6-53.

**4-Line-to-16-Line Decoder as a Demultiplexer**

We have already discussed a 4-line-to-16-line decoder (Section 6-5). This device and other decoders can also be used in demultiplexing applications. The logic symbol for this device when used as a demultiplexer is shown in Figure 6-54. In demultiplexer applications, the input lines are used as the data-select lines. One of the chip select inputs is used as the data-input line, with the other chip select input held LOW to enable the internal negative-AND gate at the bottom of the diagram.

**FIGURE 6-54** The decoder used as a demultiplexer.

## 6-10 Parity Generators/Checkers

Errors can occur as digital codes are being transferred from one point to another within a digital system or while codes are being transmitted from one system to another. The errors take the form of undesired changes in the bits that make up the coded information; that is, a 1 can change to a 0, or a 0 to a 1, because of component malfunctions or electrical noise. In most digital systems, the probability that even a single bit error will occur is very small, and the likelihood that more than one will occur is even smaller. Nevertheless, when an error occurs undetected, it can cause serious problems in a digital system.

The parity method of error detection in which a **parity bit** is attached to a group of information bits in order to make the total number of 1s either even or odd (depending on the system) was covered in Chapter 2. In addition to parity bits, several specific codes also provide inherent error detection.

### Basic Parity Logic

In order to check for or to generate the proper parity in a given code, a basic principle can be used:

**The sum (disregarding carries) of an even number of 1s is always 0, and the sum of an odd number of 1s is always 1.**

Therefore, to determine if a given code has **even parity** or **odd parity**, all the bits in that code are summed. As you know, the modulo-2 sum of two bits can be generated by an exclusive-OR gate, as shown in Figure 6-55(a); the modulo-2 sum of four bits can be formed by three exclusive-OR gates connected as shown in Figure 6-55(b); and so on. 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).

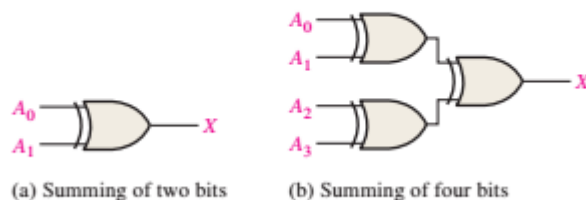
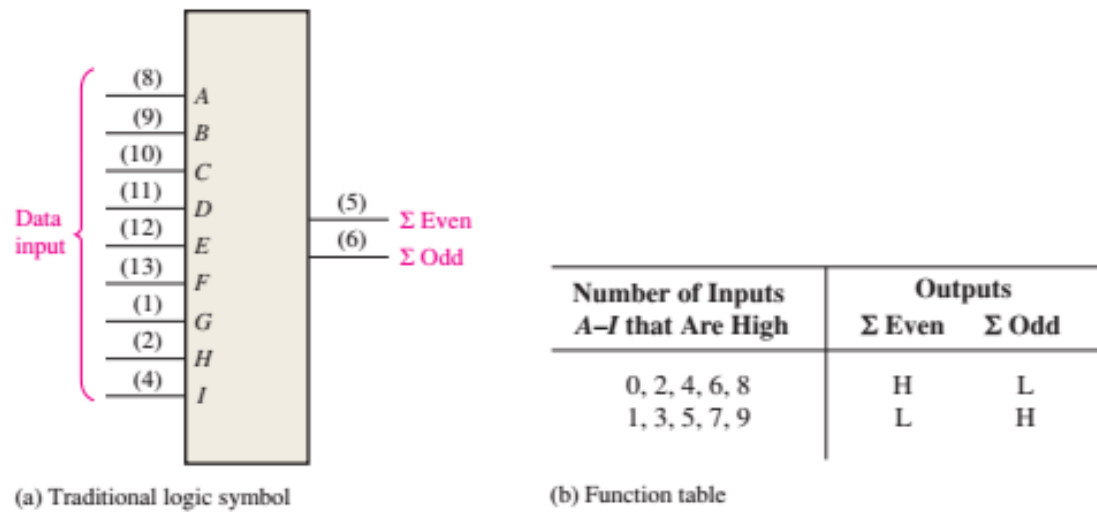


FIGURE 6-55

# 9-BIT PARITY GENERATOR/CHECKER

**Fixed-Function Device** The logic symbol and function table for a 74HC280 are shown in Figure 6-56. This particular device can be used to check for odd or even parity on a 9-bit code (eight data bits and one parity bit), or it can be used to generate a parity bit for a binary code with up to nine bits. The inputs are *A* through *I*; when there is an even number of 1s on the inputs, the  $\Sigma$  Even output is HIGH and the  $\Sigma$  Odd output is LOW.



**FIGURE 6-56** The 74HC280 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  $\Sigma$  Even output goes LOW and the  $\Sigma$  Odd output goes HIGH. When it is used as an odd parity checker, the number of input bits should always be odd; and when a parity error occurs, the  $\Sigma$  Odd output goes LOW and the  $\Sigma$  Even output goes HIGH.

**Parity Generator** If this device is used as an even parity generator, the parity bit is taken at the  $\Sigma$  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  $\Sigma$  Even output because it is a 0 when the number of inputs bits is odd.

# Latches, Flip-Flops, and Timers

## CHAPTER OUTLINE

- 7-1 Latches
  - 7-2 Flip-Flops
  - 7-3 Flip-Flop Operating Characteristics
  - 7-4 Flip-Flop Applications
  - 7-5 One-Shots
  - 7-6 The Astable Multivibrator
  - 7-7 Troubleshooting
- Applied Logic

## INTRODUCTION

This chapter begins a study of the fundamentals of sequential logic. Bistable, monostable, and astable logic devices called *multivibrators* are covered. Two categories of bistable devices are the latch and the flip-flop. Bistable devices have two stable states, called SET and RESET; they can retain either of these states indefinitely, making them useful as storage devices. The basic difference between latches and flip-flops is the way in which they are changed from one state to the other. The flip-flop is a basic building block for counters, registers, and other sequential control logic and is used in certain types of memories. The monostable multivibrator, commonly known as the one-shot, has only one stable state. A one-shot produces a single controlled-width pulse when activated or triggered. The astable multivibrator has no stable state and is used primarily as an oscillator, which is a self-sustained waveform generator. Pulse oscillators are used as the sources for timing waveforms in digital systems.

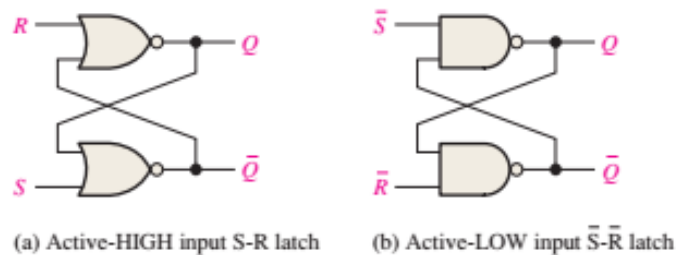
### 7-1 Latches

The **latch** is a type of temporary storage device that has two stable states (bistable) and is normally placed in a category separate from that of flip-flops. Latches are similar to flip-flops because they are bistable devices that can reside in either of two states using a feedback arrangement, in which the outputs are connected back to the opposite inputs. The main difference between latches and flip-flops is in the method used for changing their state.



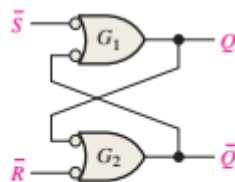
## The S-R (SET-RESET) Latch

A latch is a type of **bistable** logic device or **multivibrator**. An active-HIGH input S-R (SET-RESET) latch is formed with two cross-coupled NOR gates, as shown in Figure 7-1(a); an active-LOW input  $\bar{S}$ - $\bar{R}$  latch is formed with two cross-coupled NAND gates, as shown in Figure 7-1(b). Notice that the output of each gate is connected to an input of the opposite gate. This produces the regenerative **feedback** that is characteristic of all latches and flip-flops.



**FIGURE 7-1** Two versions of SET-RESET (S-R) latches. Open files F07-01(a) and (b) and verify the operation of both latches. A Multisim tutorial is available on the website.

To explain the operation of the latch, we will use the NAND gate  $\bar{S}$ - $\bar{R}$  latch in Figure 7-1(b). This latch is redrawn in Figure 7-2 with the negative-OR equivalent symbols used for the NAND gates. This is done because LOWs on the  $\bar{S}$  and  $\bar{R}$  lines are the activating inputs.



**FIGURE 7-2** Negative-OR equivalent of the NAND gate  $\bar{S}$ - $\bar{R}$  latch in Figure 7-1(b).

The latch in Figure 7-2 has two inputs,  $\bar{S}$  and  $\bar{R}$ , and two outputs,  $Q$  and  $\bar{Q}$ . Let's start by assuming that both inputs and the  $Q$  output are HIGH, which is the normal latched state. Since the  $Q$  output is connected back to an input of gate  $G_2$ , and the  $\bar{R}$  input is HIGH, the output of  $G_2$  must be LOW. This LOW output is coupled back to an input of gate  $G_1$ , ensuring that its output is HIGH.

When the  $Q$  output is HIGH, the latch is in the **SET** state. It will remain in this state indefinitely until a LOW is temporarily applied to the  $\bar{R}$  input. With a LOW on the  $\bar{R}$  input and a HIGH on  $\bar{S}$ , the output of gate  $G_2$  is forced HIGH. This HIGH on the  $\bar{Q}$  output is coupled back to an input of  $G_1$ , and since the  $\bar{S}$  input is HIGH, the output of  $G_1$  goes LOW. This LOW on the  $Q$  output is then coupled back to an input of  $G_2$ , ensuring that the  $\bar{Q}$  output remains HIGH even when the LOW on the  $\bar{R}$  input is removed. When the  $Q$  output is LOW, the latch is in the **RESET** state. Now the latch remains indefinitely in the RESET state until a momentary LOW is applied to the  $\bar{S}$  input.

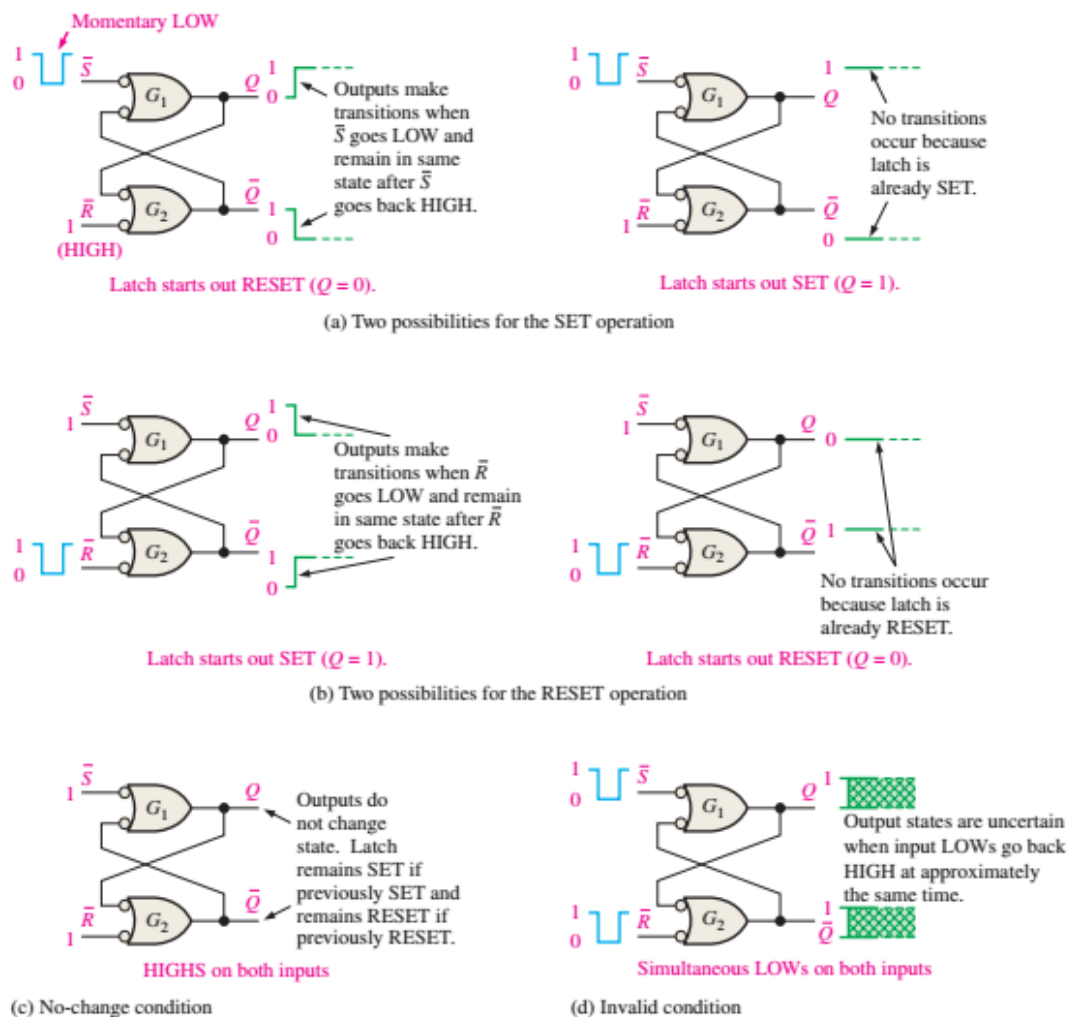


In normal operation, the outputs of a latch are always complements of each other.

**When  $Q$  is HIGH,  $\bar{Q}$  is LOW, and when  $Q$  is LOW,  $\bar{Q}$  is HIGH.**

An invalid condition in the operation of an active-LOW input  $\bar{S}$ - $\bar{R}$  latch occurs when LOWs are applied to both  $\bar{S}$  and  $\bar{R}$  at the same time. As long as the LOW levels are simultaneously held on the inputs, both the  $Q$  and  $\bar{Q}$  outputs are forced HIGH, thus violating the basic complementary operation of the outputs. Also, if the LOWs are released simultaneously, both outputs will attempt to go LOW. Since there is always some small difference in the propagation delay time of the gates, one of the gates will dominate in its transition to the LOW output state. This, in turn, forces the output of the slower gate to remain HIGH. In this situation, you cannot reliably predict the next state of the latch.

Figure 7-3 illustrates the active-LOW input  $\bar{S}$ - $\bar{R}$  latch operation for each of the four possible combinations of levels on the inputs. (The first three combinations are valid, but the last is not.) Table 7-1 summarizes the logic operation in truth table form. Operation of the active-HIGH input NOR gate latch in Figure 7-1(a) is similar but requires the use of opposite logic levels.

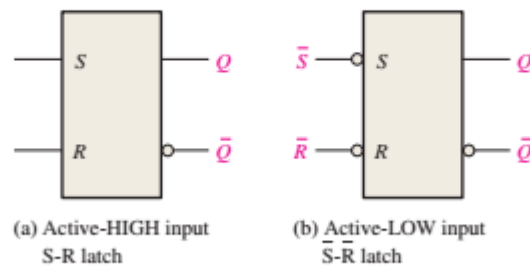


**FIGURE 7-3** The three modes of basic  $\bar{S}$ - $\bar{R}$  latch operation (SET, RESET, no-change) and the invalid condition.

**TABLE 7-1**Truth table for an active-LOW input  $\bar{S}\text{-}\bar{R}$  latch.

Inputs		Outputs		Comments
$\bar{S}$	$\bar{R}$	$Q$	$\bar{Q}$	
1	1	NC	NC	No change. Latch remains in present state.
0	1	1	0	Latch SET.
1	0	0	1	Latch RESET.
0	0	1	1	Invalid condition

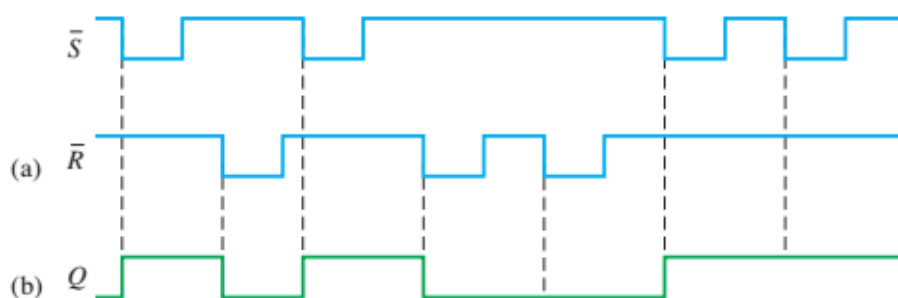
Logic symbols for both the active-HIGH input and the active-LOW input latches are shown in Figure 7-4.

**FIGURE 7-4** Logic symbols for the S-R and  $\bar{S}\text{-}\bar{R}$  latch.

Example 7-1 illustrates how an active-LOW input  $\bar{S}\text{-}\bar{R}$  latch responds to conditions on its inputs. LOW levels are pulsed on each input in a certain sequence and the resulting  $Q$  output waveform is observed. The  $\bar{S} = 0, \bar{R} = 0$  condition is avoided because it results in an invalid mode of operation and is a major drawback of any SET-RESET type of latch.

**EXAMPLE 7-1**

If the  $\bar{S}$  and  $\bar{R}$  waveforms in Figure 7-5(a) are applied to the inputs of the latch in Figure 7-4(b), determine the waveform that will be observed on the  $Q$  output. Assume that  $Q$  is initially LOW.

**FIGURE 7-5****Solution**

See Figure 7-5(b).

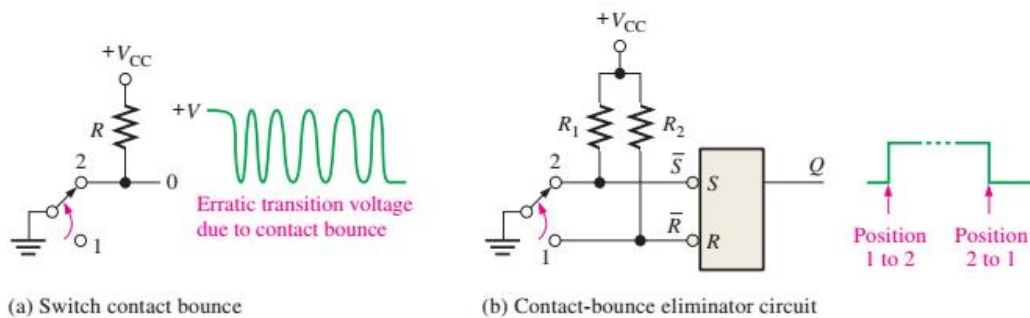
**Related Problem\***

Determine the  $Q$  output of an active-HIGH input S-R latch if the waveforms in Figure 7-5(a) are inverted and applied to the inputs.

## An Application

### The Latch as a Contact-Bounce Eliminator

A good example of an application of an  $\bar{S}$ - $\bar{R}$  latch is in the elimination of mechanical switch contact “bounce.” When the pole of a switch strikes the contact upon switch closure, it physically vibrates or bounces several times before finally making a solid contact. Although these bounces are very short in duration, they produce voltage spikes that are often not acceptable in a digital system. This situation is illustrated in Figure 7-6(a).

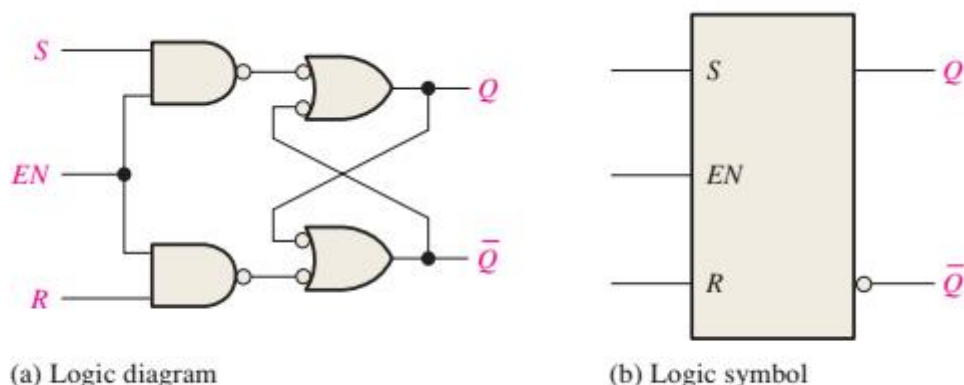


**FIGURE 7-6** The  $\bar{S}$ - $\bar{R}$  latch used to eliminate switch contact bounce.

An  $\bar{S}$ - $\bar{R}$  latch can be used to eliminate the effects of switch bounce as shown in Figure 7-6(b). The switch is normally in position 1, keeping the  $\bar{R}$  input LOW and the latch RESET. When the switch is thrown to position 2,  $\bar{R}$  goes HIGH because of the pull-up resistor to  $V_{CC}$ , and  $\bar{S}$  goes LOW on the first contact. Although  $\bar{S}$  remains LOW for only a very short time before the switch bounces, this is sufficient to set the latch. Any further voltage spikes on the  $\bar{S}$  input due to switch bounce do not affect the latch, and it remains SET. Notice that the  $Q$  output of the latch provides a clean transition from LOW to HIGH, thus eliminating the voltage spikes caused by contact bounce. Similarly, a clean transition from HIGH to LOW is made when the switch is thrown back to position 1.

### The Gated S-R Latch

A gated latch requires an enable input,  $EN$  ( $G$  is also used to designate an enable input). The logic diagram and logic symbol for a gated S-R latch are shown in Figure 7-8. The  $S$  and  $R$  inputs control the state to which the latch will go when a HIGH level is applied to the  $EN$  input. The latch will not change until  $EN$  is HIGH; but as long as it remains HIGH, the output is controlled by the state of the  $S$  and  $R$  inputs. The gated latch is a *level-sensitive* device. In this circuit, the invalid state occurs when both  $S$  and  $R$  are simultaneously HIGH and  $EN$  is also HIGH.



**FIGURE 7-8** A gated S-R latch.

### EXAMPLE 7-2

Determine the  $Q$  output waveform if the inputs shown in Figure 7-9(a) are applied to a gated S-R latch that is initially RESET.

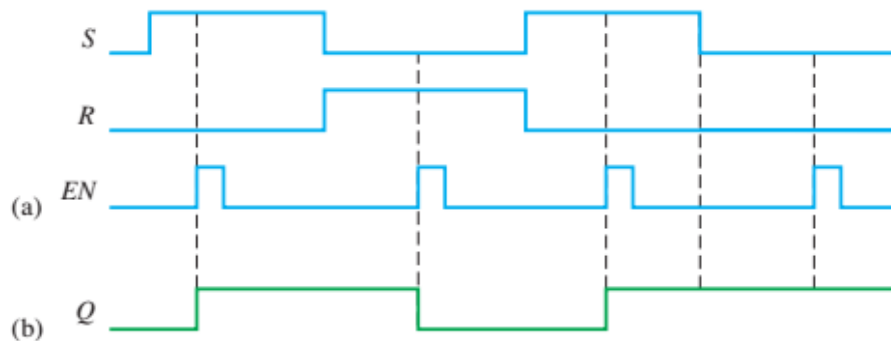


FIGURE 7-9

#### Solution

The  $Q$  waveform is shown in Figure 7-9(b). When  $S$  is HIGH and  $R$  is LOW, a HIGH on the  $EN$  input sets the latch. When  $S$  is LOW and  $R$  is HIGH, a HIGH on the  $EN$  input resets the latch. When both  $S$  and  $R$  are LOW, the  $Q$  output does not change from its present state.

#### Related Problem

Determine the  $Q$  output of a gated S-R latch if the  $S$  and  $R$  inputs in Figure 7-9(a) are inverted.

### The Gated D Latch

Another type of gated latch is called the D latch. It differs from the S-R latch because it has only one input in addition to  $EN$ . This input is called the  $D$  (data) input. Figure 7-10 contains a logic diagram and logic symbol of a D latch. When the  $D$  input is HIGH and the  $EN$  input is HIGH, the latch will set. When the  $D$  input is LOW and  $EN$  is HIGH, the latch will reset. Stated another way, the output  $Q$  follows the input  $D$  when  $EN$  is HIGH.

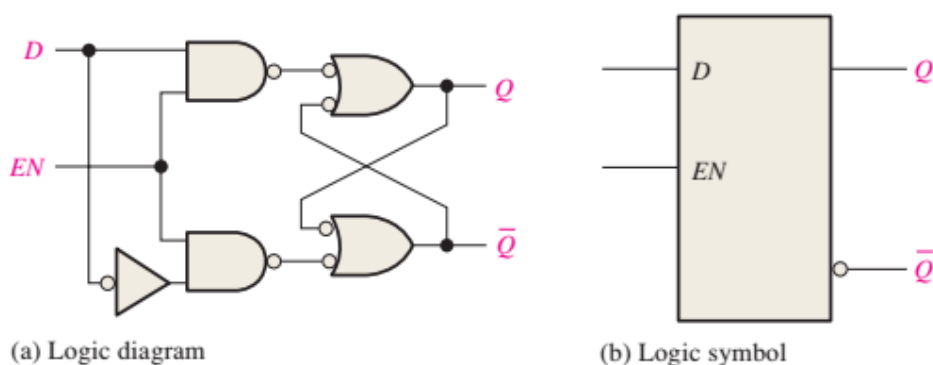


FIGURE 7-10 A gated D latch. Open file F07-10 and verify the operation.

### EXAMPLE 7-3

Determine the  $Q$  output waveform if the inputs shown in Figure 7-11(a) are applied to a gated D latch, which is initially RESET.

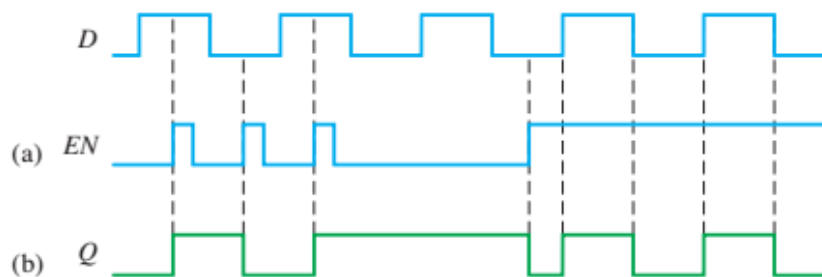


FIGURE 7-11

### Solution

The  $Q$  waveform is shown in Figure 7-11(b). When  $D$  is HIGH and  $EN$  is HIGH,  $Q$  goes HIGH. When  $D$  is LOW and  $EN$  is HIGH,  $Q$  goes LOW. When  $EN$  is LOW, the state of the latch is not affected by the  $D$  input.

Inputs		Outputs		Comments
$D$	$EN$	$Q$	$\bar{Q}$	
0	1	0	1	RESET
1	1	1	0	SET
X	0	$Q_0$	$\bar{Q}_0$	No change

Note:  $Q_0$  is the prior output level before the indicated input conditions were established.

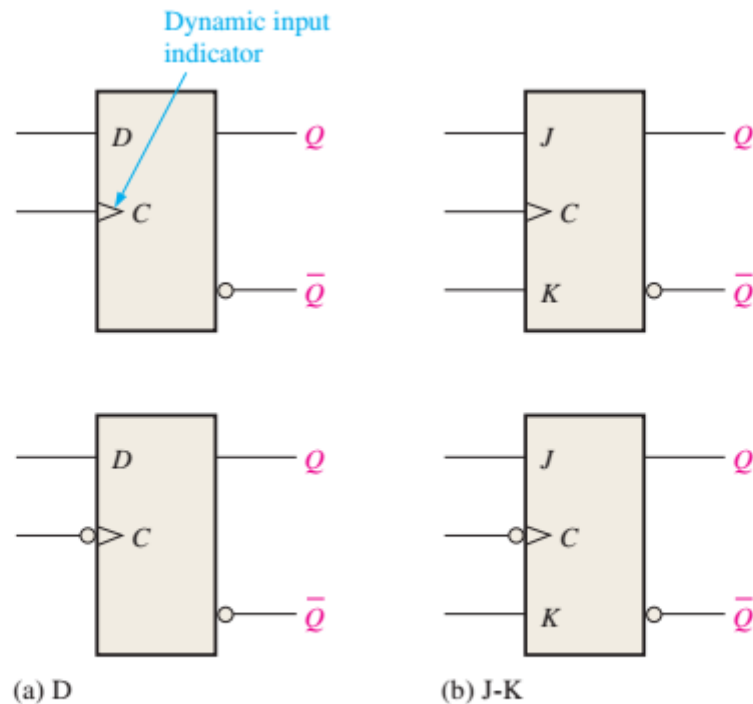
### SECTION 7-1 CHECKUP

Answers are at the end of the chapter.

1. List three types of latches.
2. Develop the truth table for the active-HIGH input S-R latch in Figure 7-1(a).
3. What is the  $Q$  output of a D latch when  $EN = 1$  and  $D = 1$ ?

An **edge-triggered flip-flop** changes state either at the positive edge (rising edge) or at the negative edge (falling edge) of the clock pulse and is sensitive to its inputs only at this transition of the clock. Two types of edge-triggered flip-flops are covered in this section: D and J-K. The logic symbols for these flip-flops are shown in Figure 7-13. Notice that each type can be either positive edge-triggered (no bubble at  $C$  input) or negative edge-triggered (bubble at  $C$  input). The key to identifying an edge-triggered flip-flop by its logic symbol is the small triangle inside the block at the clock ( $C$ ) input. This triangle is called the *dynamic input indicator*.



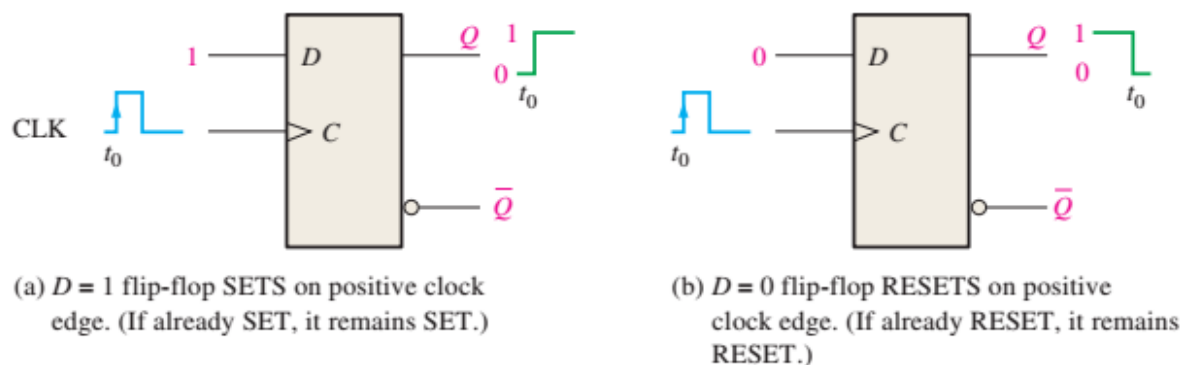


**FIGURE 7-13** Edge-triggered flip-flop logic symbols (top: positive edge-triggered; bottom: negative edge-triggered).

## The D Flip-Flop

The  $D$  input of the **D flip-flop** is a **synchronous** input because data on the input are transferred to the flip-flop's output only on the triggering edge of the clock pulse. When  $D$  is HIGH, the  $Q$  output goes HIGH on the triggering edge of the clock pulse, and the flip-flop is SET. When  $D$  is LOW, the  $Q$  output goes LOW on the triggering edge of the clock pulse, and the flip-flop is RESET.

This basic operation of a positive edge-triggered D flip-flop is illustrated in Figure 7-14, and Table 7-2 is the truth table for this type of flip-flop. Remember, *the flip-flop cannot change state except on the triggering edge of a clock pulse*. The  $D$  input can be changed at any time when the clock input is LOW or HIGH (except for a very short interval around the triggering transition of the clock) without affecting the output. Just remember,  $Q$  follows  $D$  at the triggering edge of the clock.



**FIGURE 7-14** Operation of a positive edge-triggered D flip-flop.



**TABLE 7-2**

Truth table for a positive edge-triggered D flip-flop.

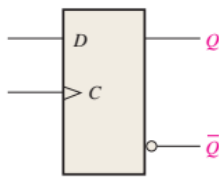
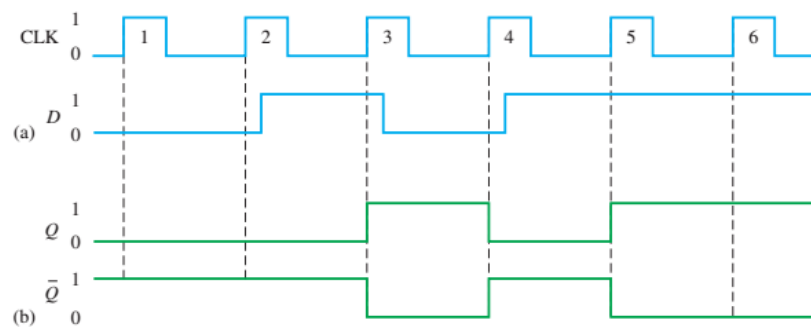
Inputs		Outputs		Comments
$D$	CLK	$Q$	$\bar{Q}$	
0	↑	0	1	RESET
1	↑	1	0	SET

↑ = clock transition LOW to HIGH

The operation and truth table for a negative edge-triggered D flip-flop are the same as those for a positive edge-triggered device except that the falling edge of the clock pulse is the triggering edge.

**EXAMPLE 7-4**

Determine the  $Q$  and  $\bar{Q}$  output waveforms of the flip-flop in Figure 7-15 for the  $D$  and CLK inputs in Figure 7-16(a). Assume that the positive edge-triggered flip-flop is initially RESET.

**FIGURE 7-15****FIGURE 7-16****Solution**

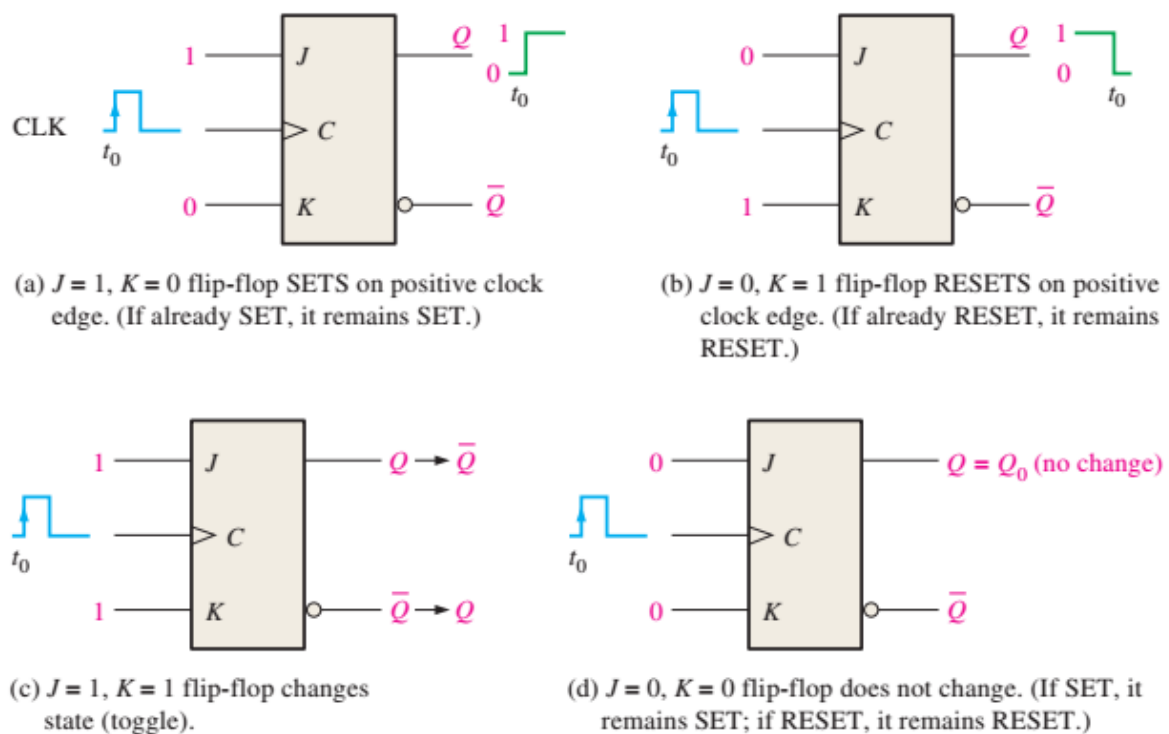
1. At clock pulse 1,  $D$  is LOW, so  $Q$  remains LOW (RESET).
2. At clock pulse 2,  $D$  is LOW, so  $Q$  remains LOW (RESET).
3. At clock pulse 3,  $D$  is HIGH, so  $Q$  goes HIGH (SET).
4. At clock pulse 4,  $D$  is LOW, so  $Q$  goes LOW (RESET).
5. At clock pulse 5,  $D$  is HIGH, so  $Q$  goes HIGH (SET).
6. At clock pulse 6,  $D$  is HIGH, so  $Q$  remains HIGH (SET).

Once  $Q$  is determined,  $\bar{Q}$  is easily found since it is simply the complement of  $Q$ , shown in Figure 7-16(b) for the input waveforms in part (a).

## The J-K Flip-Flop

The  $J$  and  $K$  inputs of the **J-K flip-flop** are synchronous inputs because data on these inputs are transferred to the flip-flop's output only on the triggering edge of the clock pulse. When  $J$  is HIGH and  $K$  is LOW, the  $Q$  output goes HIGH on the triggering edge of the clock pulse, and the flip-flop is SET. When  $J$  is LOW and  $K$  is HIGH, the  $Q$  output goes LOW on the triggering edge of the clock pulse, and the flip-flop is RESET. When both  $J$  and  $K$  are LOW, the output does not change from its prior state. When  $J$  and  $K$  are both HIGH, the flip-flop changes state. This called the **toggle** mode.

This basic operation of a positive edge-triggered flip-flop is illustrated in Figure 7–17, and Table 7–3 is the truth table for this type of flip-flop. Remember, *the flip-flop cannot change state except on the triggering edge of a clock pulse*. The  $J$  and  $K$  inputs can be changed at any time when the clock input is LOW or HIGH (except for a very short interval around the triggering transition of the clock) without affecting the output.



**FIGURE 7-17** Operation of a positive edge-triggered J-K flip-flop.

**TABLE 7-3**

Truth table for a positive edge-triggered J-K flip-flop.

Inputs			Outputs		Comments
$J$	$K$	CLK	$Q$	$\bar{Q}$	
0	0	↑	$Q_0$	$\bar{Q}_0$	No change
0	1	↑	0	1	RESET
1	0	↑	1	0	SET
1	1	↑	$\bar{Q}_0$	$Q_0$	Toggle

↑ = clock transition LOW to HIGH

$Q_0$  = output level prior to clock transition

### EXAMPLE 7-5

The waveforms in Figure 7-18(a) are applied to the  $J$ ,  $K$ , and clock inputs as indicated. Determine the  $Q$  output, assuming that the flip-flop is initially RESET.

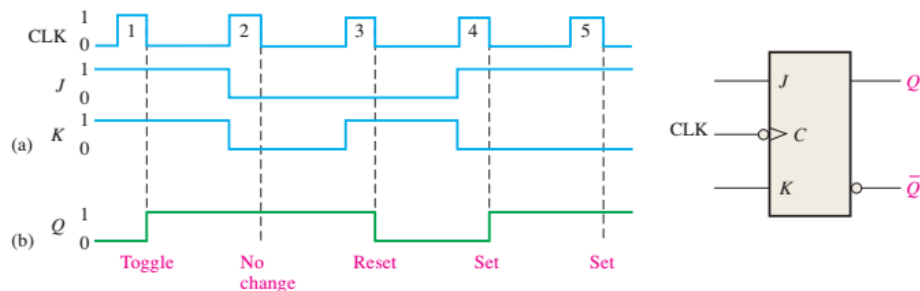


FIGURE 7-18

#### Solution

Since this is a negative edge-triggered flip-flop, as indicated by the “bubble” at the clock input, the  $Q$  output will change only on the negative-going edge of the clock pulse.

1. At the first clock pulse, both  $J$  and  $K$  are HIGH; and because this is a toggle condition,  $Q$  goes HIGH.
2. At clock pulse 2, a no-change condition exists on the inputs, keeping  $Q$  at a HIGH level.
3. When clock pulse 3 occurs,  $J$  is LOW and  $K$  is HIGH, resulting in a RESET condition;  $Q$  goes LOW.
4. At clock pulse 4,  $J$  is HIGH and  $K$  is LOW, resulting in a SET condition;  $Q$  goes HIGH.
5. A SET condition still exists on  $J$  and  $K$  when clock pulse 5 occurs, so  $Q$  will remain HIGH.

The resulting  $Q$  waveform is indicated in Figure 7-18(b).

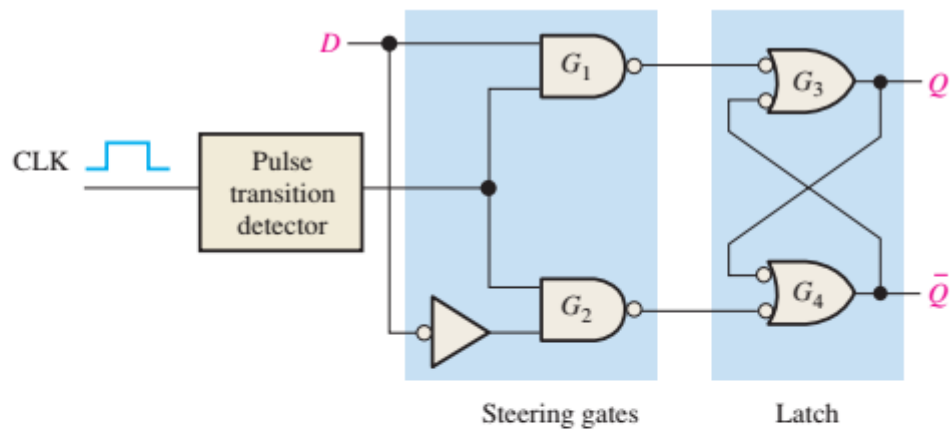
## Edge-Triggered Operation

### D Flip-Flop

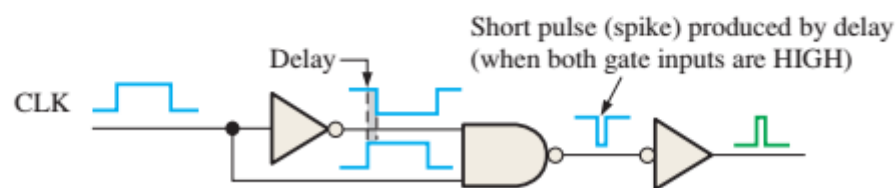
A simplified implementation of an edge-triggered D flip-flop is illustrated in Figure 7-19(a) and is used to demonstrate the concept of edge-triggering. Notice that the basic D flip-flop differs from the gated D latch only in that it has a pulse transition detector.

One basic type of pulse transition detector is shown in Figure 7-19(b). As you can see, there is a small delay through the inverter on one input to the NAND gate so that the inverted clock pulse arrives at the gate input a few nanoseconds after the true clock pulse. This circuit produces a very short-duration spike on the positive-going transition of the clock pulse. In a negative edge-triggered flip-flop the clock pulse is inverted first, thus producing a narrow spike on the negative-going edge.

The circuit in Figure 7-19(a) is partitioned into two sections, one labeled Steering gates and the other labeled Latch. The steering gates direct, or steer, the clock spike either to the input to gate  $G_3$  or to the input to gate  $G_4$ , depending on the state of the  $D$  input. To understand the operation of this flip-flop, begin with the assumptions that it is in the RESET state ( $Q = 0$ ) and that the  $D$  and CLK inputs are LOW. For this condition, the outputs of gate  $G_1$  and gate  $G_2$  are both HIGH. The LOW on the  $Q$  output is coupled back into one input of gate  $G_4$ , making the  $\bar{Q}$  output HIGH. Because  $\bar{Q}$  is HIGH, both inputs to gate  $G_3$  are HIGH (remember, the output of gate  $G_1$  is HIGH), holding the  $Q$  output LOW. If a pulse is applied to the CLK input, the outputs of gates  $G_1$  and  $G_2$  remain HIGH because they are disabled by the LOW on the  $D$  input; therefore, there is no change in the state of the flip-flop—it remains in the RESET state.



(a) A simplified logic diagram for a positive edge-triggered D flip-flop

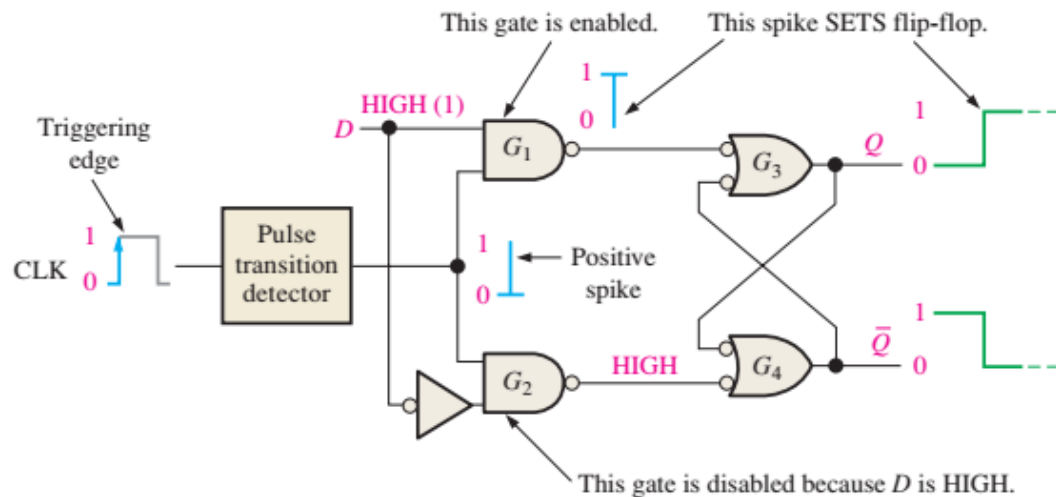


(b) A type of pulse transition detector

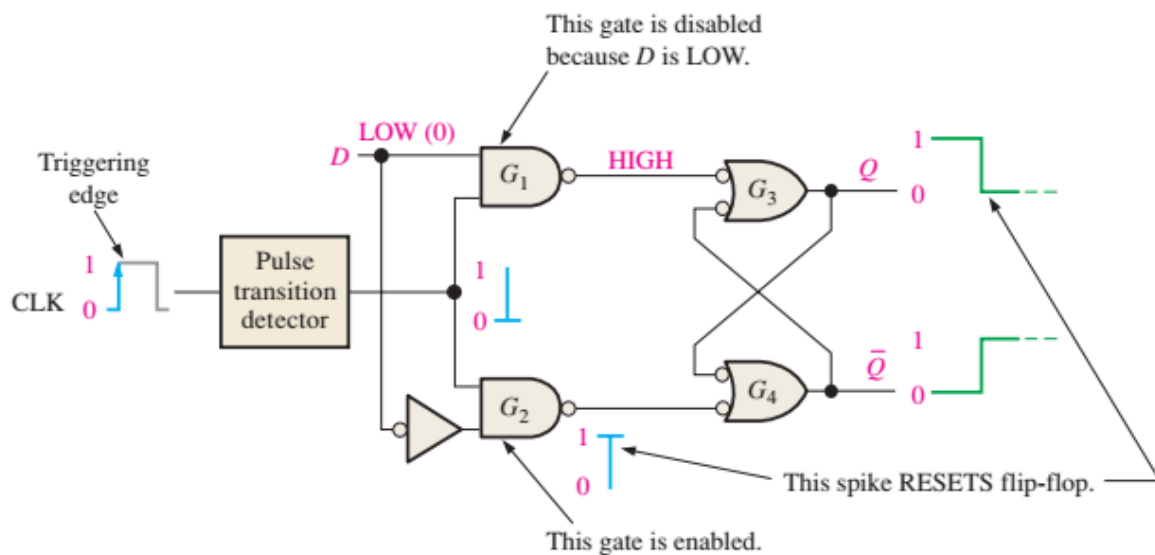
**FIGURE 7-19** Edge triggering.

Let's now make  $D$  HIGH and apply a clock pulse. Because the  $D$  input to gate  $G_1$  is now HIGH, the output of gate  $G_1$  goes LOW for a very short time (spike) when CLK goes HIGH, causing the  $Q$  output to go HIGH. Both inputs to gate  $G_4$  are now HIGH (remember, gate  $G_2$  output is HIGH because  $D$  is HIGH), forcing the  $\bar{Q}$  output LOW. This LOW on  $\bar{Q}$  is coupled back into one input of gate  $G_3$ , ensuring that the  $Q$  output will remain HIGH. The flip-flop is now in the SET state. Figure 7-20 illustrates the logic level transitions that take place within the flip-flop for this condition.

Next, let's make  $D$  LOW and apply a clock pulse. The positive-going edge of the clock produces a negative-going spike on the output of gate  $G_2$ , causing the  $\bar{Q}$  output to go HIGH. Because of this HIGH on  $\bar{Q}$ , both inputs to gate  $G_3$  are now HIGH (remember, the output of gate  $G_1$  is HIGH because of the LOW on  $D$ ), forcing the  $Q$  output to go LOW. This LOW on  $Q$  is coupled back into one input of gate  $G_4$ , ensuring that  $\bar{Q}$  will remain HIGH. The flip-flop is now in the RESET state. Figure 7-21 illustrates the logic level transitions that occur within the flip-flop for this condition.



**FIGURE 7-20** Flip-flop making a transition from the RESET state to the SET state on the positive-going edge of the clock pulse.



**FIGURE 7-21** Flip-flop making a transition from the SET state to the RESET state on the positive-going edge of the clock pulse.



### EXAMPLE 7-6

Given the waveforms in Figure 7-22(a) for the  $D$  input and the clock, determine the  $Q$  output waveform if the flip-flop starts out RESET.

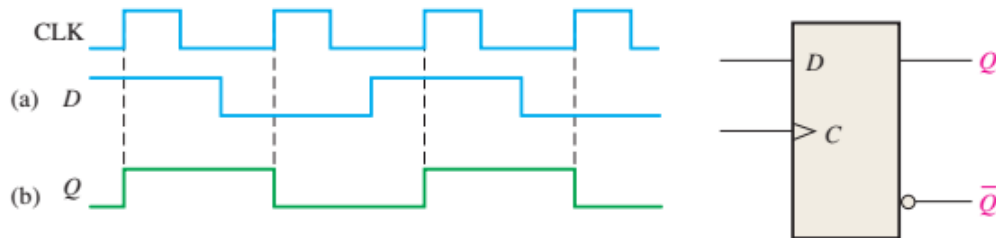


FIGURE 7-22

### Solution

The  $Q$  output goes to the state of the  $D$  input at the time of the positive-going clock edge. The resulting output is shown in Figure 7-22(b).

## J-K Flip-Flop

Figure 7-23 shows the basic internal logic for a positive edge-triggered J-K flip-flop. The  $Q$  output is connected back to the input of gate  $G_2$ , and the  $\bar{Q}$  output is connected back to the input of gate  $G_1$ . The two control inputs are labeled  $J$  and  $K$  in honor of Jack Kilby, who invented the integrated circuit. A J-K flip-flop can also be of the negative edge-triggered type, in which case the clock input is inverted.

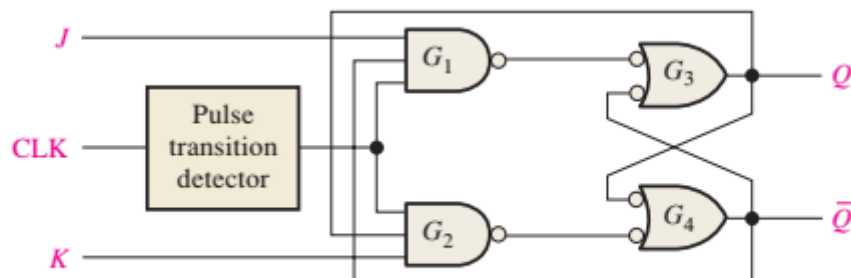
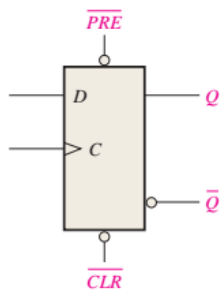


FIGURE 7-23 A simplified logic diagram for a positive edge-triggered J-K flip-flop.

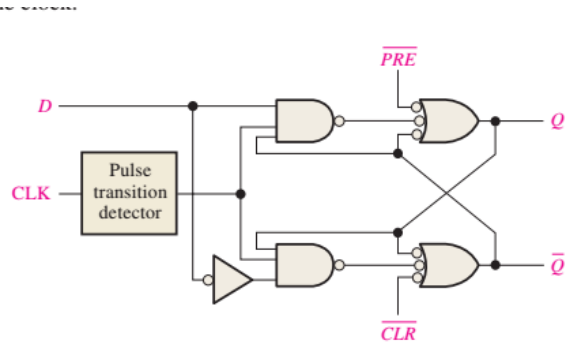
Let's assume that the flip-flop in Figure 7-24 is RESET and that the  $J$  input is HIGH and the  $K$  input is LOW rather than as shown. When a clock pulse occurs, a leading-edge spike indicated by ① is passed through gate  $G_1$  because  $\bar{Q}$  is HIGH and  $J$  is HIGH. This will cause the latch portion of the flip-flop to change to the SET state. The flip-flop is now SET.







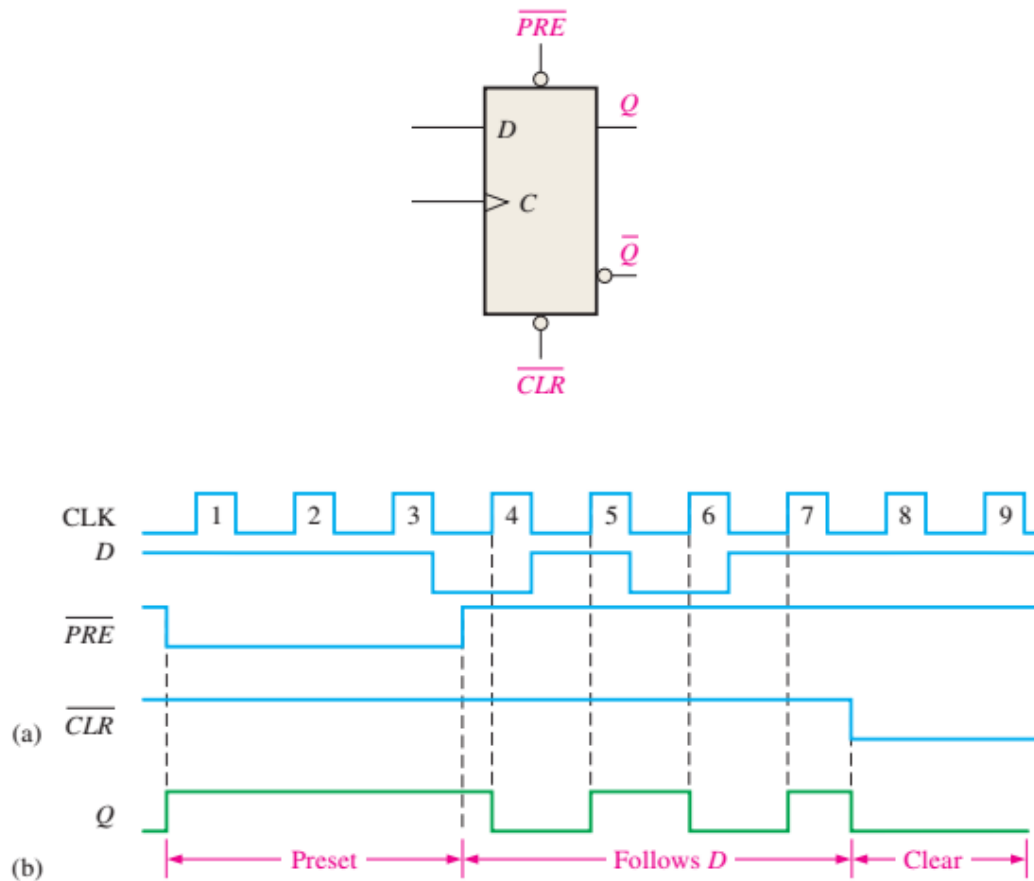
**FIGURE 7-25** Logic symbol for a D flip-flop with active-LOW preset and clear inputs.



**FIGURE 7-26** Logic diagram for a basic D flip-flop with active-LOW preset and clear inputs.

### EXAMPLE 7-7

For the positive edge-triggered D flip-flop with preset and clear inputs in Figure 7-27, determine the  $Q$  output for the inputs shown in the timing diagram in part (a) if  $Q$  is initially LOW.



**FIGURE 7-27** Open file F07-27 to verify the operation

### Solution

1. During clock pulses 1, 2, and 3, the preset ( $\overline{PRE}$ ) is LOW, keeping the flip-flop SET regardless of the synchronous  $D$  input.
2. For clock pulses 4, 5, 6, and 7, the output follows the input on the clock pulse because both  $\overline{PRE}$  and  $\overline{CLR}$  are HIGH.
3. For clock pulses 8 and 9, the clear ( $\overline{CLR}$ ) input is LOW, keeping the flip-flop RESET regardless of the synchronous inputs.

The resulting  $Q$  output is shown in Figure 7–27(b).

Let's look at two specific edge-triggered flip-flops. They are representative of the various types of flip-flops available in fixed-function IC form and, like most other devices, are available in CMOS and in bipolar (TTL) logic families.

Also, you will learn how VHDL is used to describe the types of flip-flops.

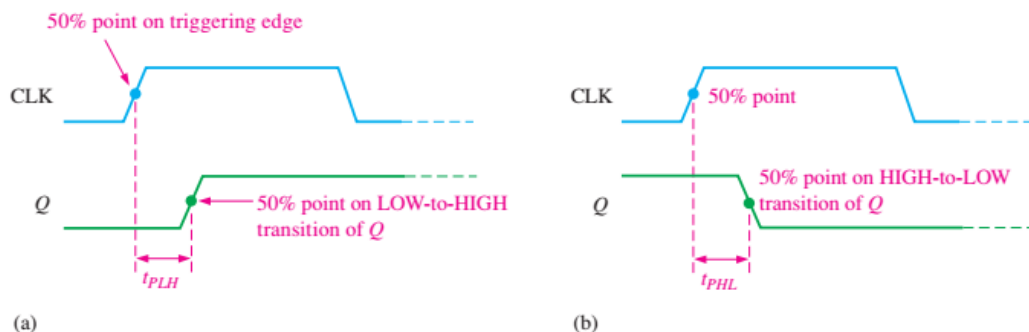
## 7–3 Flip-Flop Operating Characteristics

The performance, operating requirements, and limitations of flip-flops are specified by several operating characteristics or parameters found on the data sheet for the device. Generally, the specifications are applicable to all CMOS and bipolar (TTL) flip-flops.

### Propagation Delay Times

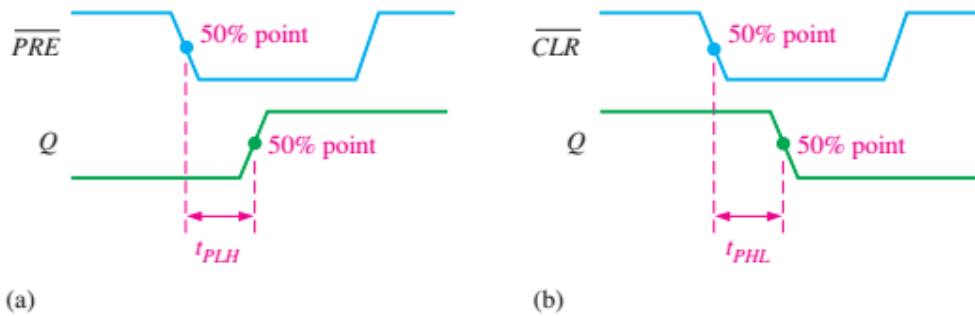
A **propagation delay time** is the interval of time required after an input signal has been applied for the resulting output change to occur. Four categories of propagation delay times are important in the operation of a flip-flop:

1. Propagation delay  $t_{PLH}$  as measured from the triggering edge of the clock pulse to the LOW-to-HIGH transition of the output. This delay is illustrated in Figure 7–31(a).
2. Propagation delay  $t_{PHL}$  as measured from the triggering edge of the clock pulse to the HIGH-to-LOW transition of the output. This delay is illustrated in Figure 7–31(b).



**FIGURE 7–31** Propagation delays, clock to output.

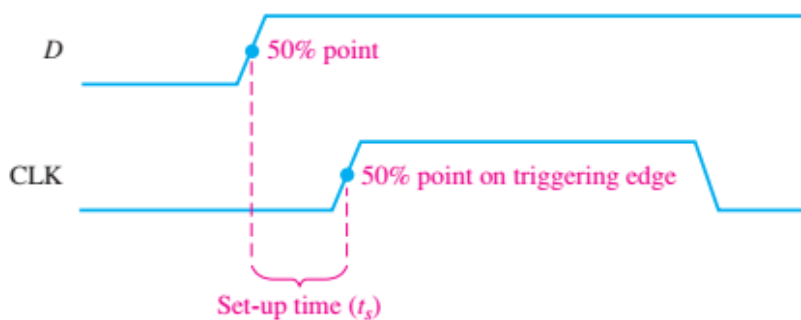
3. Propagation delay  $t_{PLH}$  as measured from the leading edge of the preset input to the LOW-to-HIGH transition of the output. This delay is illustrated in Figure 7–32(a) for an active-LOW preset input.
4. Propagation delay  $t_{PHL}$  as measured from the leading edge of the clear input to the HIGH-to-LOW transition of the output. This delay is illustrated in Figure 7–32(b) for an active-LOW clear input.



**FIGURE 7–32** Propagation delays, preset input to output and clear input to output.

### Set-up Time

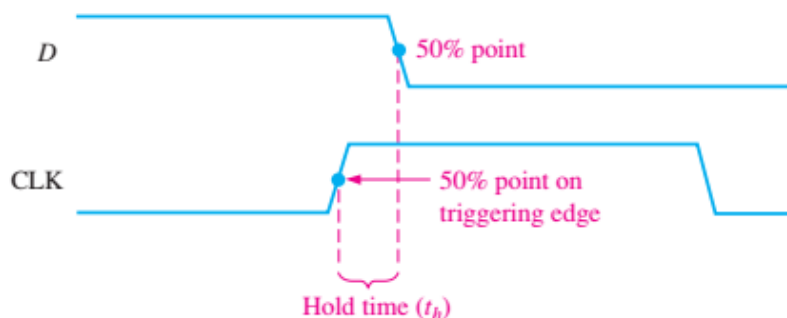
The **set-up time** ( $t_s$ ) is the minimum interval required for the logic levels to be maintained constantly on the inputs ( $J$  and  $K$ , or  $D$ ) prior to the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip-flop. This interval is illustrated in Figure 7–33 for a D flip-flop.



**FIGURE 7–33** Set-up time ( $t_s$ ). The logic level must be present on the  $D$  input for a time equal to or greater than  $t_s$  before the triggering edge of the clock pulse for reliable data entry.

## Hold Time

The **hold time** ( $t_h$ ) is the minimum interval required for the logic levels to remain on the inputs after the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip-flop. This is illustrated in Figure 7–34 for a D flip-flop.



**FIGURE 7–34** Hold time ( $t_h$ ). The logic level must remain on the  $D$  input for a time equal to or greater than  $t_h$  after the triggering edge of the clock pulse for reliable data entry.

## Maximum Clock Frequency

The maximum clock frequency ( $f_{\max}$ ) is the highest rate at which a flip-flop can be reliably triggered. At clock frequencies above the maximum, the flip-flop would be unable to respond quickly enough, and its operation would be impaired.

## Pulse Widths

Minimum pulse widths ( $t_W$ ) for reliable operation are usually specified by the manufacturer for the clock, preset, and clear inputs. Typically, the clock is specified by its minimum HIGH time and its minimum LOW time.

## Power Dissipation

The **power dissipation** of any digital circuit is the total power consumption of the device. For example, if the flip-flop operates on a +5 V dc source and draws 5 mA of current, the power dissipation is

$$P = V_{CC} \times I_{CC} = 5 \text{ V} \times 5 \text{ mA} = 25 \text{ mW}$$

The power dissipation is very important in most applications in which the capacity of the dc supply is a concern. As an example, let's assume that you have a digital system that requires a total of ten flip-flops, and each flip-flop dissipates 25 mW of power. The total power requirement is

$$P_T = 10 \times 25 \text{ mW} = 250 \text{ mW} = 0.25 \text{ W}$$

This tells you the output capacity required of the dc supply. If the flip-flops operate on +5 V dc, then the amount of current that the supply must provide is

$$I = \frac{250 \text{ mW}}{5 \text{ V}} = 50 \text{ mA}$$

You must use a +5 V dc supply that is capable of providing at least 50 mA of current.



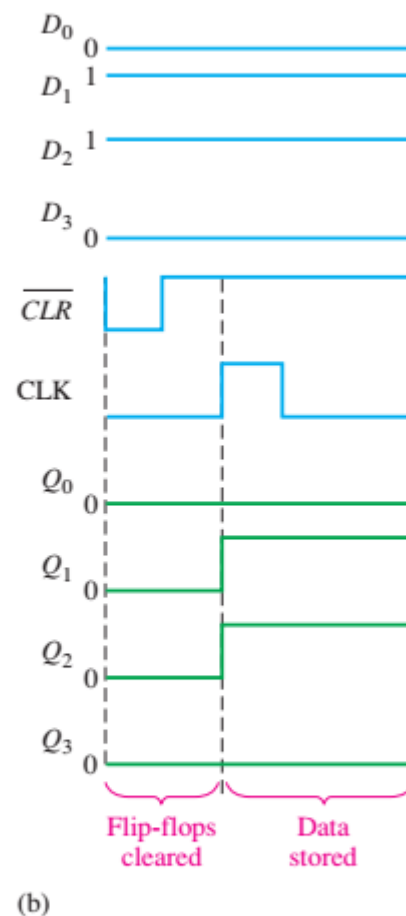
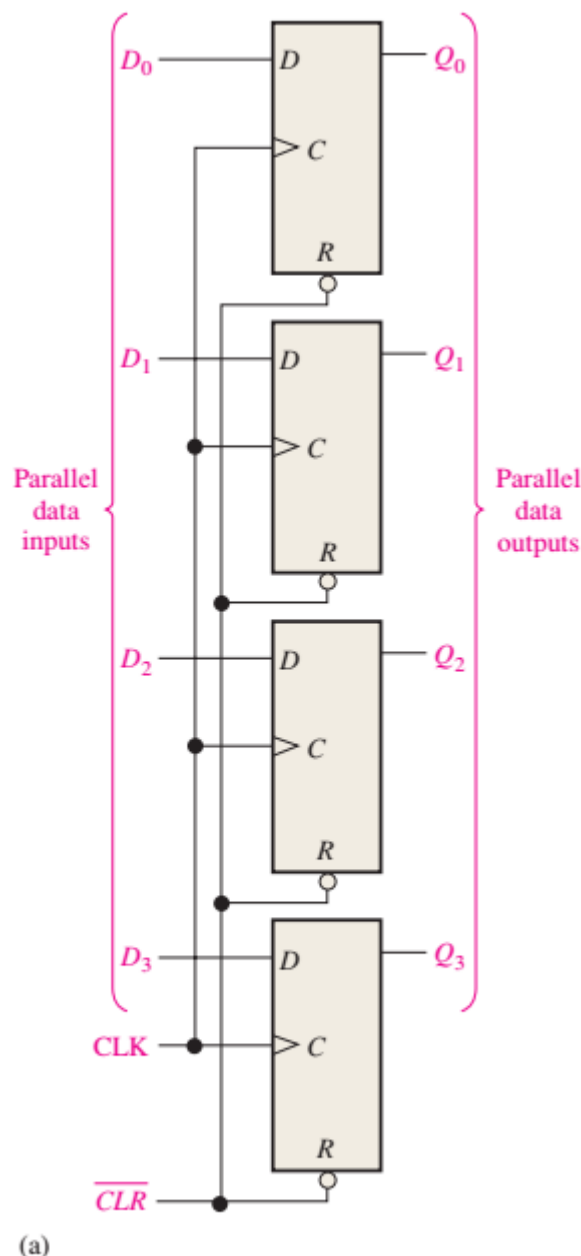
## 7-4 Flip-Flop Applications

In this section, three general applications of flip-flops are discussed to give you an idea of how they can be used. In Chapters 8 and 9, flip-flop applications in registers and counters are covered in detail.

### Parallel Data Storage

A common requirement in digital systems is to store several bits of data from parallel lines simultaneously in a group of flip-flops. This operation is illustrated in Figure 7-35(a) using four flip-flops. Each of the four parallel data lines is connected to the  $D$  input of a flip-flop. The clock inputs of the flip-flops are connected together, so that each flip-flop is triggered by the same clock pulse. In this example, positive edge-triggered flip-flops are used, so the data on the  $D$  inputs are stored simultaneously by the flip-flops on the positive edge of the clock, as indicated in the timing diagram in Figure 7-35(b). Also, the asynchronous reset ( $R$ ) inputs are connected to a common  $\overline{CLR}$  line, which initially resets all the flip-flops.



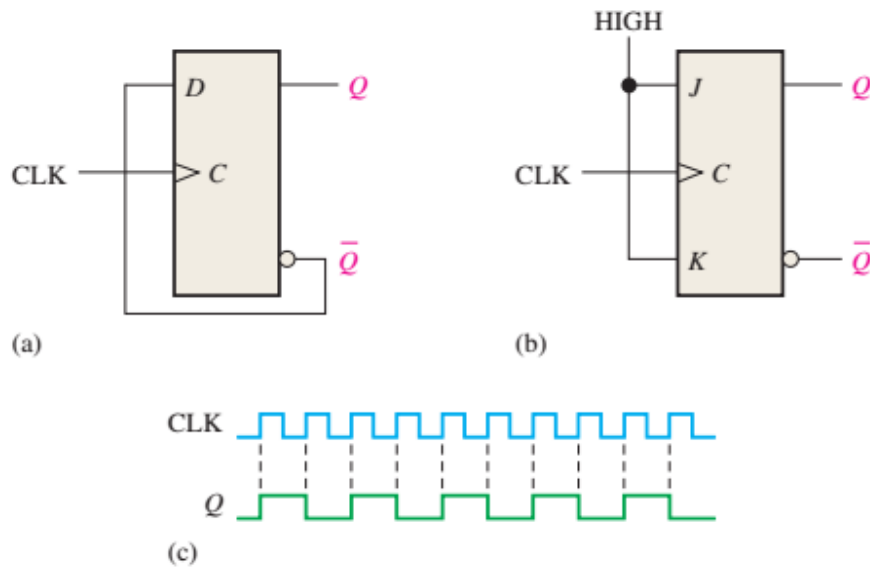


**FIGURE 7-35** Example of flip-flops used in a basic register for parallel data storage.

This group of four flip-flops is an example of a basic register used for data storage. In digital systems, data are normally stored in groups of bits (usually eight or multiples thereof) that represent numbers, codes, or other information. Registers are covered in Chapter 8.

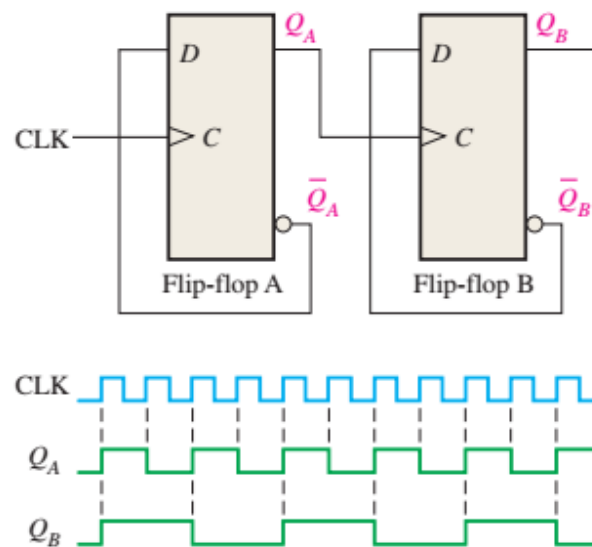
## Frequency Division

Another application of a flip-flop is dividing (reducing) the frequency of a periodic waveform. When a pulse waveform is applied to the clock input of a D or J-K flip-flop that is connected to toggle ( $D = \overline{Q}$  or  $J = K = 1$ ), the  $Q$  output is a square wave with one-half the frequency of the clock input. Thus, a single flip-flop can be applied as a divide-by-2 device, as is illustrated in Figure 7-36 for both a D and a J-K flip-flop. As you can see in part (c), the flip-flop changes state on each triggering clock edge (positive edge-triggered in this case). This results in an output that changes at half the frequency of the clock waveform.



**FIGURE 7-36** The D flip-flop and J-K flip-flop as a divide-by-2 device.  $Q$  is one-half the frequency of CLK. Open file F07-36 and verify the operation.

Further division of a clock frequency can be achieved by using the output of one flip-flop as the clock input to a second flip-flop, as shown in Figure 7-37. The frequency of the  $Q_A$  output is divided by 2 by flip-flop B. The  $Q_B$  output is, therefore, one-fourth the frequency of the original clock input. Propagation delay times are not shown on the timing diagrams.

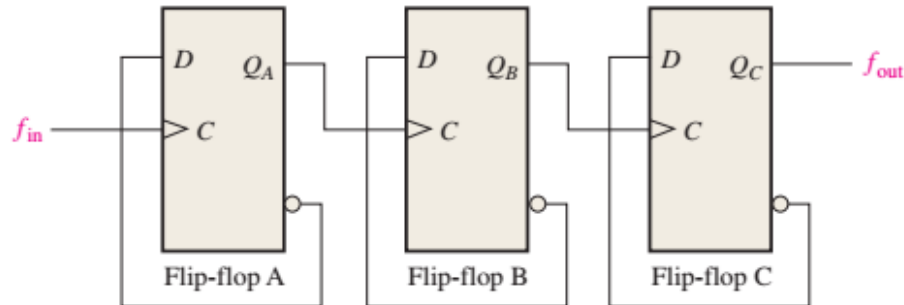


**FIGURE 7-37** Example of two D flip-flops used to divide the clock frequency by 4.  $Q_A$  is one-half and  $Q_B$  is one-fourth the frequency of CLK. Open file F07-37 and verify the operation.

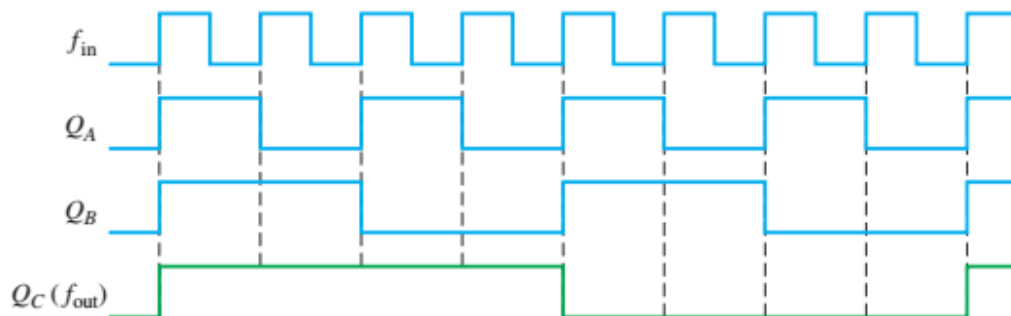
By connecting flip-flops in this way, a frequency division of  $2^n$  is achieved, where  $n$  is the number of flip-flops. For example, three flip-flops divide the clock frequency by  $2^3 = 8$ ; four flip-flops divide the clock frequency by  $2^4 = 16$ ; and so on.

**EXAMPLE 7-9**

Develop the  $f_{\text{out}}$  waveform for the circuit in Figure 7-38 when an 8 kHz square wave input is applied to the clock input of flip-flop A.

**FIGURE 7-38****Solution**

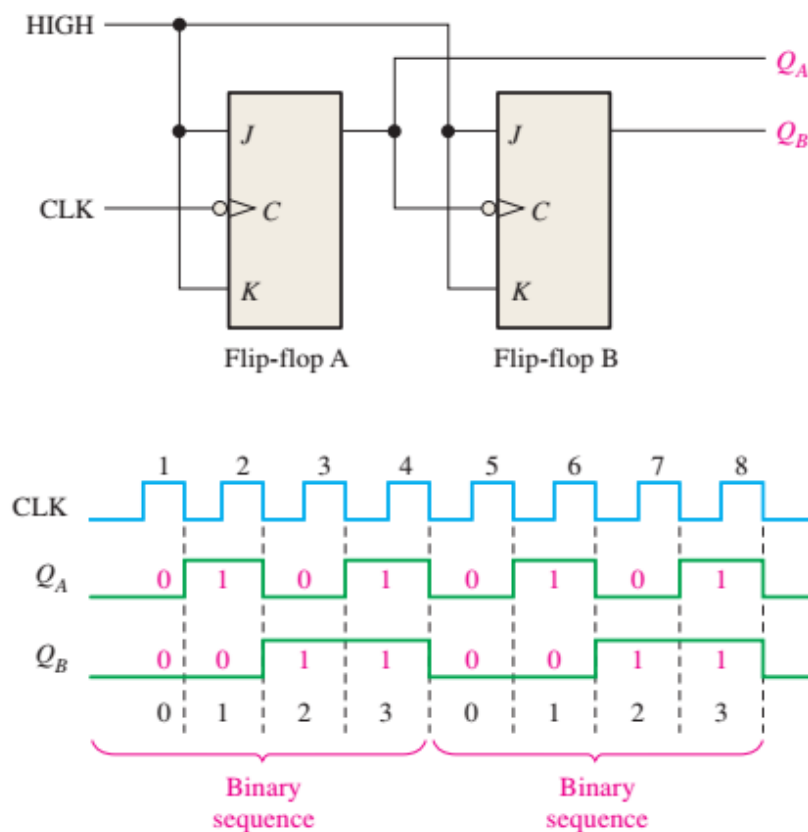
The three flip-flops are connected to divide the input frequency by eight ( $2^3 = 8$ ) and the  $Q_C$  ( $f_{\text{out}}$ ) waveform is shown in Figure 7-39. Since these are positive edge-triggered flip-flops, the outputs change on the positive-going clock edge. There is one output pulse for every eight input pulses, so the output frequency is 1 kHz. Waveforms of  $Q_A$  and  $Q_B$  are also shown.

**FIGURE 7-39**

## Counting

Another important application of flip-flops is in digital counters, which are covered in detail in Chapter 9. The concept is illustrated in Figure 7–40. Negative edge-triggered J-K flip-flops are used for illustration. Both flip-flops are initially RESET. Flip-flop A toggles on the negative-going transition of each clock pulse. The  $Q$  output of flip-flop A clocks flip-flop B, so each time  $Q_A$  makes a HIGH-to-LOW transition, flip-flop B toggles. The resulting  $Q_A$  and  $Q_B$  waveforms are shown in the figure.

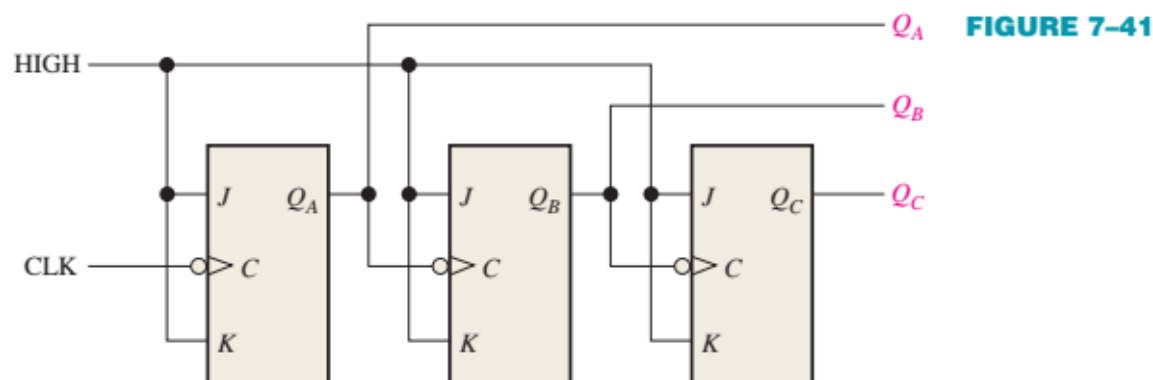
Observe the sequence of  $Q_A$  and  $Q_B$  in Figure 7–40. Prior to clock pulse 1,  $Q_A = 0$  and  $Q_B = 0$ ; after clock pulse 1,  $Q_A = 1$  and  $Q_B = 0$ ; after clock pulse 2,  $Q_A = 0$  and  $Q_B = 1$ ; and after clock pulse 3,  $Q_A = 1$  and  $Q_B = 1$ . If we take  $Q_A$  as the least significant bit, a 2-bit sequence is produced as the flip-flops are clocked. This binary sequence repeats every four clock pulses, as shown in the timing diagram of Figure 7–40. Thus, the flip-flops are counting in sequence from 0 to 3 (00, 01, 10, 11) and then recycling back to 0 to begin the sequence again.



**FIGURE 7–40** J-K flip-flops used to generate a binary count sequence (00, 01, 10, 11). Two repetitions are shown.

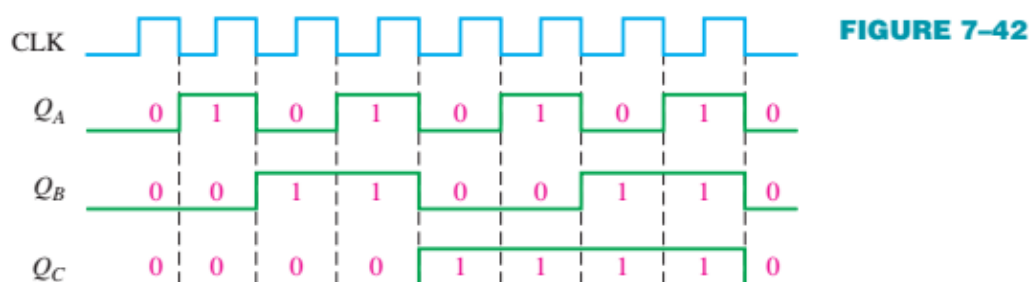
### EXAMPLE 7-10

Determine the output waveforms in relation to the clock for  $Q_A$ ,  $Q_B$ , and  $Q_C$  in the circuit of Figure 7-41 and show the binary sequence represented by these waveforms.



### Solution

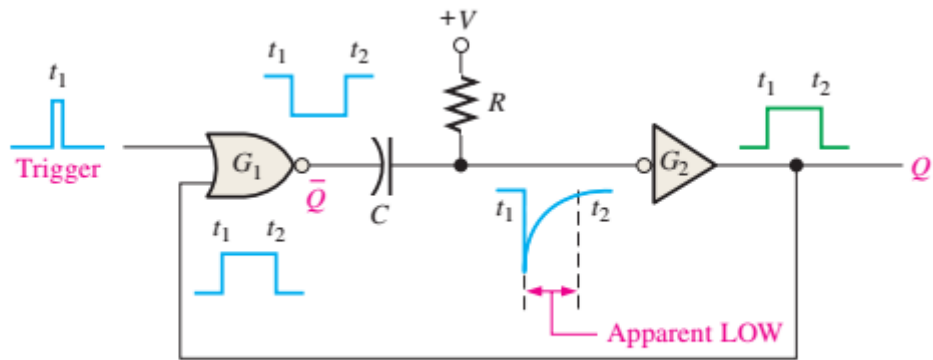
The output timing diagram is shown in Figure 7-42. Notice that the outputs change on the negative-going edge of the clock pulses. The outputs go through the binary sequence 000, 001, 010, 011, 100, 101, 110, and 111 as indicated.



## 7-5 One-Shots

The **one-shot**, also known as a **monostable** multivibrator, is a device with only one stable state. A one-shot is normally in its stable state and will change to its unstable state only when triggered. Once it is triggered, the one-shot remains in its unstable state for a predetermined length of time and then automatically returns to its stable state. The time that the device stays in its unstable state determines the pulse width of its output.

Figure 7-43 shows a basic one-shot (monostable multivibrator) that is composed of a logic gate and an inverter. When a pulse is applied to the **trigger** input, the output of gate  $G_1$  goes LOW. This HIGH-to-LOW transition is coupled through the capacitor to the input of inverter  $G_2$ . The apparent LOW on  $G_2$  makes its output go HIGH. This HIGH is connected back into  $G_1$ , keeping its output LOW. Up to this point the trigger pulse has caused the output of the one-shot,  $Q$ , to go HIGH.

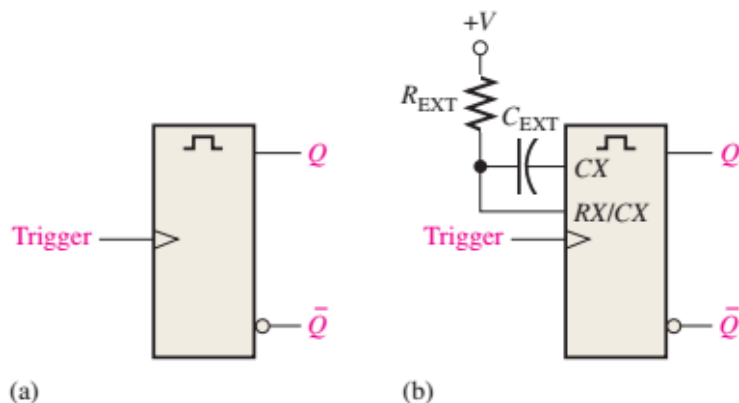


**FIGURE 7-43** A simple one-shot circuit.

The capacitor immediately begins to charge through  $R$  toward the high voltage level. The rate at which it charges is determined by the  $RC$  time constant. When the capacitor charges to a certain level, which appears as a HIGH to  $G_2$ , the output goes back LOW.

To summarize, the output of inverter  $G_2$  goes HIGH in response to the trigger input. It remains HIGH for a time set by the  $RC$  time constant. At the end of this time, it goes LOW. A single narrow trigger pulse produces a single output pulse whose time duration is controlled by the  $RC$  time constant. This operation is illustrated in Figure 7-43.

A typical one-shot logic symbol is shown in Figure 7-44(a), and the same symbol with an external  $R$  and  $C$  is shown in Figure 7-44(b). The two basic types of IC one-shots are nonretriggerable and retriggerable.

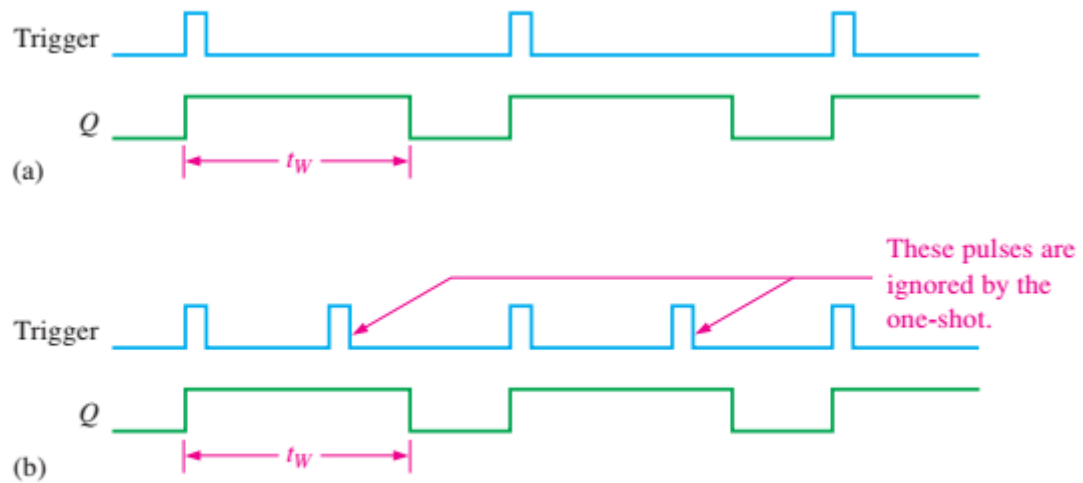


**FIGURE 7-44** Basic one-shot logic symbols.  $CX$  and  $RX$  stand for external components.

A nonretriggerable one-shot will not respond to any additional trigger pulses from the time it is triggered into its unstable state until it returns to its stable state. In other words, it will ignore any trigger pulses occurring before it times out. The time that the one-shot remains in its unstable state is the pulse width of the output.

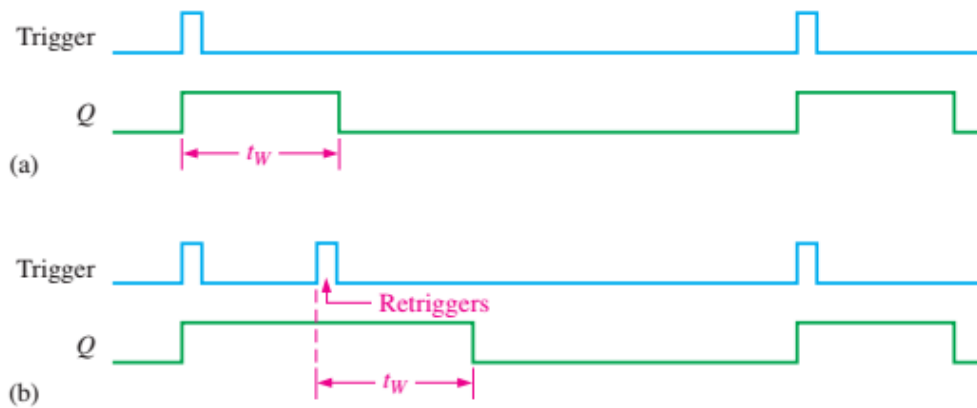
Figure 7-45 shows the nonretriggerable one-shot being triggered at intervals greater than its pulse width and at intervals less than the pulse width. Notice that in the second case, the additional pulses are ignored.





**FIGURE 7-45** Nonretriggerable one-shot action.

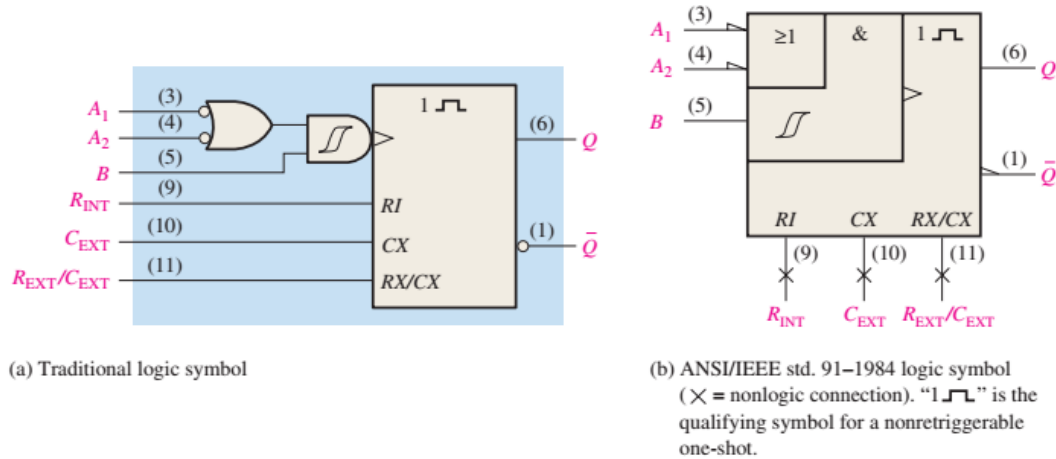
A retriggerable one-shot can be triggered before it times out. The result of retriggering is an extension of the pulse width as illustrated in Figure 7-46.



**FIGURE 7-46** Retriggerable one-shot action.

## Nonretriggerable One-Shot

The 74121 is an example of a nonretriggerable IC one-shot. It has provisions for external  $R$  and  $C$ , as shown in Figure 7-47. The inputs labeled  $A_1$ ,  $A_2$ , and  $B$  are gated trigger inputs. The  $R_{INT}$  input connects to a 2 k $\Omega$  internal timing resistor.



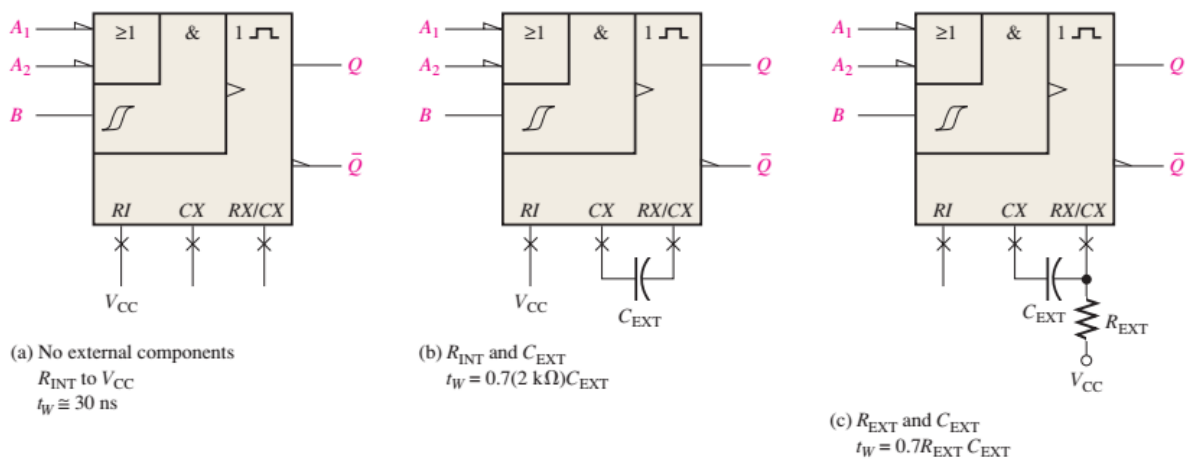
**FIGURE 7-47** Logic symbols for the 74121 nonretriggerable one-shot.

## Setting the Pulse Width

A typical pulse width of about 30 ns is produced when no external timing components are used and the internal timing resistor ( $R_{INT}$ ) is connected to  $V_{CC}$ , as shown in Figure 7-48(a). The pulse width can be set anywhere between about 30 ns and 28 s by the use of external components. Figure 7-48(b) shows the configuration using the internal resistor (2 k $\Omega$ ) and an external capacitor. Part (c) shows the configuration using an external resistor and an external capacitor. The output pulse width is set by the values of the resistor ( $R_{INT} = 2$  k $\Omega$ , and  $R_{EXT}$  is selected) and the capacitor according to the following formula:

$$t_W = 0.7RC_{EXT} \quad \text{Equation 7-1}$$

where  $R$  is either  $R_{INT}$  or  $R_{EXT}$ . When  $R$  is in kilohms (k $\Omega$ ) and  $C_{EXT}$  is in picofarads (pF), the output pulse width  $t_W$  is in nanoseconds (ns).



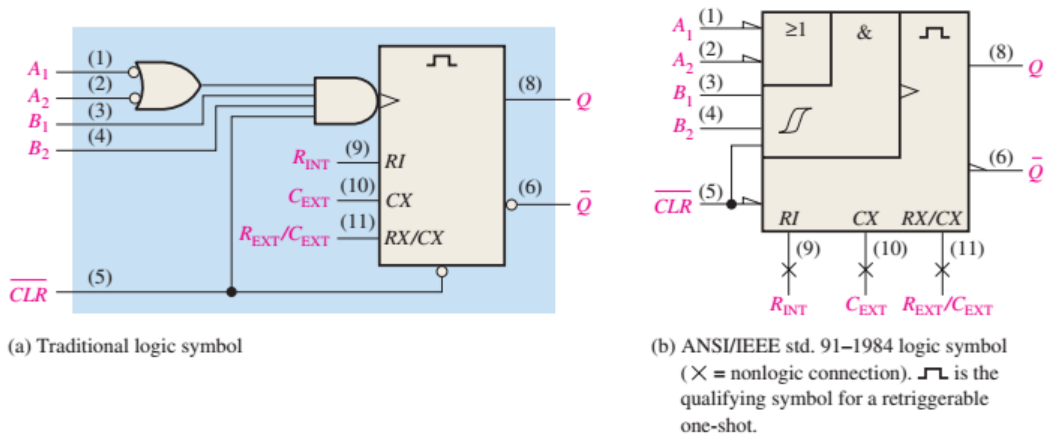
**FIGURE 7-48** Three ways to set the pulse width of a 74121.

## The Schmitt-Trigger Symbol

The symbol  $\int$  indicates a Schmitt-trigger input. This type of input uses a special threshold circuit that produces **hysteresis**, a characteristic that prevents erratic switching between states when a slow-changing trigger voltage hovers around the critical input level. This allows reliable triggering to occur even when the input is changing as slowly as 1 volt/second.

### Retriggerable One-Shot

The 74LS122 is an example of a retriggerable IC one-shot with a clear input. It also has provisions for external  $R$  and  $C$ , as shown in Figure 7–49. The inputs labeled  $A_1$ ,  $A_2$ ,  $B_1$ , and  $B_2$  are the gated trigger inputs.



**FIGURE 7–49** Logic symbol for the 74LS122 retriggerable one-shot.

A minimum pulse width of approximately 45 ns is obtained with no external components. Wider pulse widths are achieved by using external components. A general formula for calculating the values of these components for a specified pulse width ( $t_W$ ) is

$$t_W = 0.32RC_{EXT} \left( 1 + \frac{0.7}{R} \right) \quad \text{Equation 7-2}$$

where 0.32 is a constant determined by the particular type of one-shot,  $R$  is in  $k\Omega$  and is either the internal or the external resistor,  $C_{EXT}$  is in pF, and  $t_W$  is in ns. The internal resistance is 10  $k\Omega$  and can be used instead of an external resistor. (Notice the difference between this formula and that for the 74121, shown in Equation 7–1.)



### EXAMPLE 7-12

Determine the values of  $R_{EXT}$  and  $C_{EXT}$  that will produce a pulse width of  $1\ \mu\text{s}$  when connected to a 74LS122.

### Solution

Assume a value of  $C_{EXT} = 560\ \text{pF}$  and then solve for  $R_{EXT}$ . The pulse width must be expressed in ns and  $C_{EXT}$  in pF.  $R_{EXT}$  will be in  $\text{k}\Omega$ .

$$\begin{aligned}t_w &= 0.32R_{EXT}C_{EXT}\left(1 + \frac{0.7}{R_{EXT}}\right) = 0.32R_{EXT}C_{EXT} + 0.7\left(\frac{0.32R_{EXT}C_{EXT}}{R_{EXT}}\right) \\&= 0.32R_{EXT}C_{EXT} + (0.7)(0.32)C_{EXT} \\R_{EXT} &= \frac{t_w - (0.7)(0.32)C_{EXT}}{0.32C_{EXT}} = \frac{t_w}{0.32C_{EXT}} - 0.7 \\&= \frac{1000\ \text{ns}}{(0.32)560\ \text{pF}} - 0.7 = \mathbf{4.88\ \text{k}\Omega}\end{aligned}$$

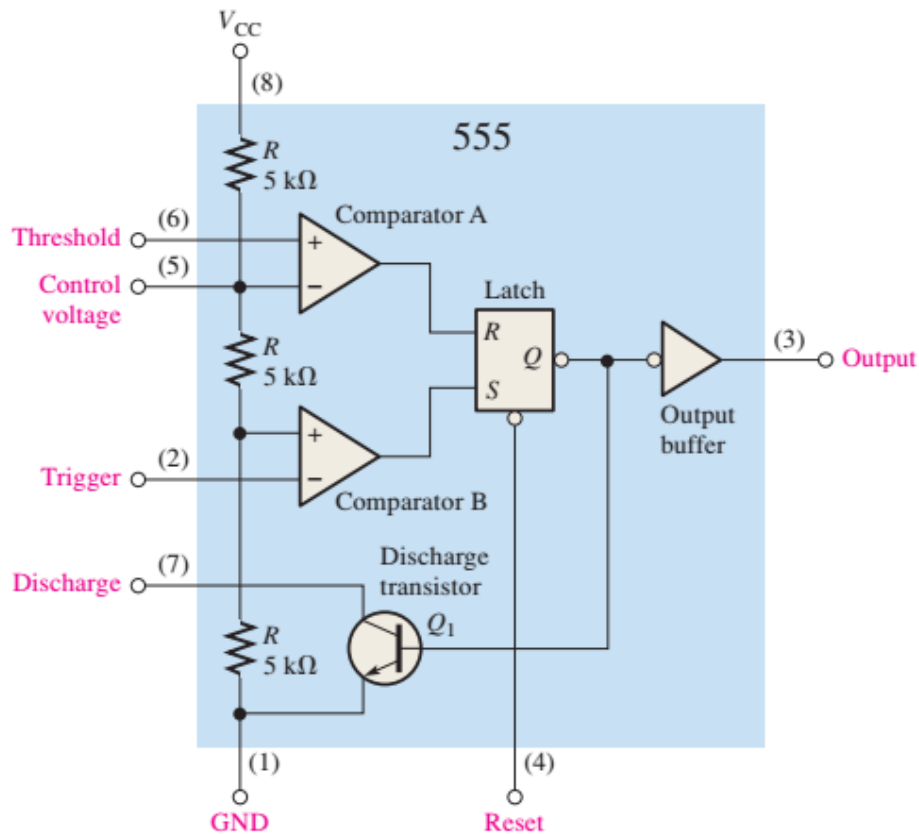
Use a standard value of  $4.7\ \text{k}\Omega$ .

### The 555 Timer as a One-Shot

The 555 **timer** is a versatile and widely used IC device because it can be configured in two different modes as either a monostable multivibrator (one-shot) or as an astable multivibrator (pulse oscillator). The astable multivibrator is discussed in Section 7-6.

### The 555 Timer Operation

A functional diagram showing the internal components of a 555 timer is shown in Figure 7-52. The comparators are devices whose outputs are HIGH when the voltage on the positive (+) input is greater than the voltage on the negative (−) input and LOW when the − input voltage is greater than the + input voltage. The voltage divider consisting of three  $5\ \text{k}\Omega$  resistors provides a trigger level of  $\frac{1}{3}V_{CC}$  and a threshold level of  $\frac{2}{3}V_{CC}$ . The control voltage input (pin 5) can be used to externally adjust the trigger and threshold levels to other values if necessary. When the normally HIGH trigger input momentarily goes below  $\frac{1}{3}V_{CC}$ , the output of comparator B switches from LOW to HIGH and sets the S-R latch, causing the output (pin 3) to go HIGH and turning the discharge transistor  $Q_1$  off. The output will stay HIGH until the normally LOW threshold input goes above  $\frac{2}{3}V_{CC}$  and causes the output of comparator A to switch from LOW to HIGH. This resets the latch, causing the output to go back LOW and turning the discharge transistor on. The external reset input can be used to reset the latch independent of the threshold circuit. The trigger and threshold inputs (pins 2 and 6) are controlled by external components connected to produce either monostable or astable action.



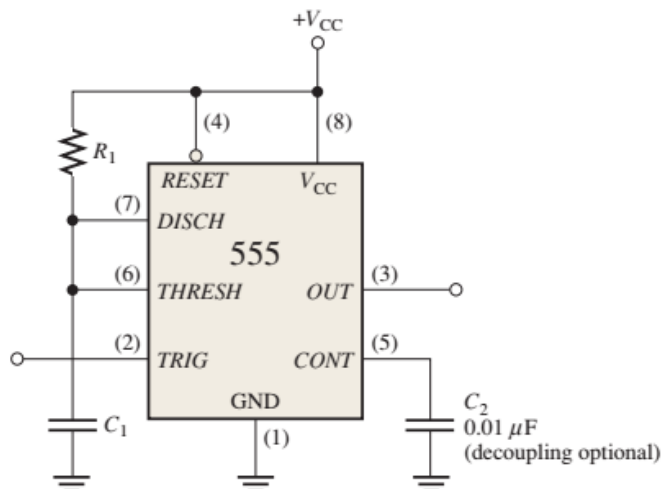
**FIGURE 7-52** Internal functional diagram of a 555 timer (pin numbers are in parentheses).

### Monostable (One-Shot) Operation

An external resistor and capacitor connected as shown in Figure 7-53 are used to set up the 555 timer as a nonretriggerable one-shot. The pulse width of the output is determined by the time constant of  $R_1$  and  $C_1$  according to the following formula:

$$t_W = 1.1R_1C_1 \quad \text{Equation 7-3}$$

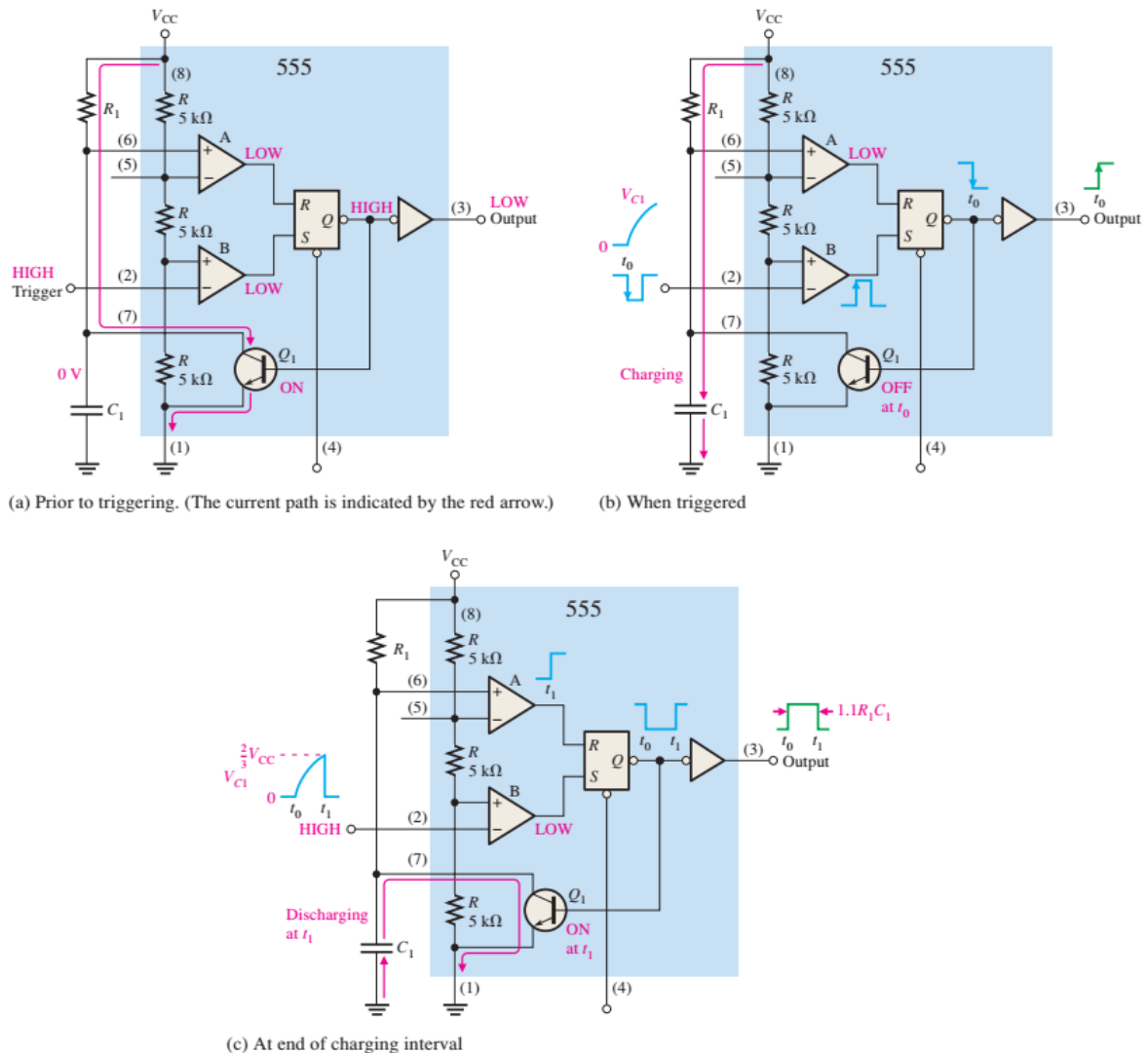
The control voltage input is not used and is connected to a decoupling capacitor  $C_2$  to prevent noise from affecting the trigger and threshold levels.



**FIGURE 7-53** The 555 timer connected as a one-shot.



Before a trigger pulse is applied, the output is LOW and the discharge transistor  $Q_1$  is *on*, keeping  $C_1$  discharged as shown in Figure 7–54(a). When a negative-going trigger pulse is applied at  $t_0$ , the output goes HIGH and the discharge transistor turns *off*, allowing capacitor  $C_1$  to begin charging through  $R_1$  as shown in part (b). When  $C_1$  charges to  $\frac{1}{3} V_{CC}$ , the output goes back LOW at  $t_1$  and  $Q_1$  turns *on* immediately, discharging  $C_1$  as shown in part (c). As you can see, the charging rate of  $C_1$  determines how long the output is HIGH.



**FIGURE 7–54** One-shot operation of the 555 timer.

### EXAMPLE 7–13

What is the output pulse width for a 555 monostable circuit with  $R_1 = 2.2 \text{ k}\Omega$  and  $C_1 = 0.01 \text{ }\mu\text{F}$ ?

#### Solution

From Equation 7–3 the pulse width is

$$t_W = 1.1R_1C_1 = 1.1(2.2 \text{ k}\Omega)(0.01 \text{ }\mu\text{F}) = \mathbf{24.2 \text{ }\mu\text{s}}$$

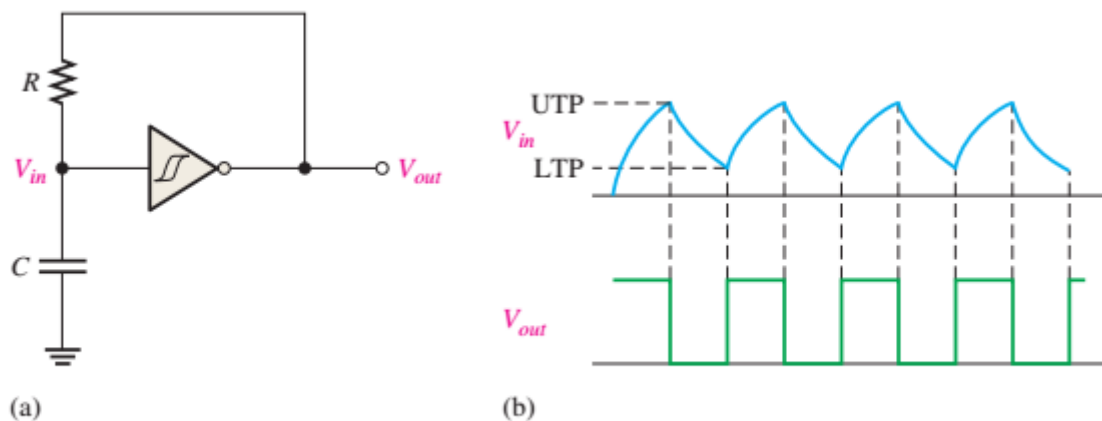
#### Related Problem

For  $C_1 = 0.01 \text{ }\mu\text{F}$ , determine the value of  $R_1$  for a pulse width of 1 ms.

## 7-6 The Astable Multivibrator

An **astable** multivibrator is a device that has no stable states; it changes back and forth (oscillates) between two unstable states without any external triggering. The resulting output is typically a square wave that is used as a clock signal in many types of sequential logic circuits. Astable multivibrators are also known as pulse **oscillators**.

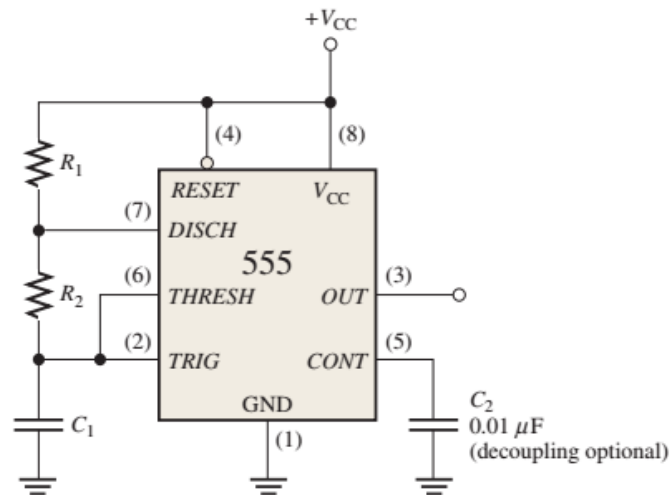
Figure 7-55(a) shows a simple form of astable multivibrator using an inverter with hysteresis (Schmitt trigger) and an  $RC$  circuit connected in a feedback arrangement. When power is first applied, the capacitor has no charge; so the input to the Schmitt trigger inverter is LOW and the output is HIGH. The capacitor charges through  $R$  until the inverter input voltage reaches the upper trigger point (UTP), as shown in Figure 7-55(b). At this point, the inverter output goes LOW, causing the capacitor to discharge back through  $R$ , shown in part (b). When the inverter input voltage decreases to the lower trigger point (LTP), its output goes HIGH and the capacitor charges again. This charging/discharging cycle continues to repeat as long as power is applied to the circuit, and the resulting output is a pulse waveform, as indicated.



**FIGURE 7-55** Basic astable multivibrator using a Schmitt trigger.

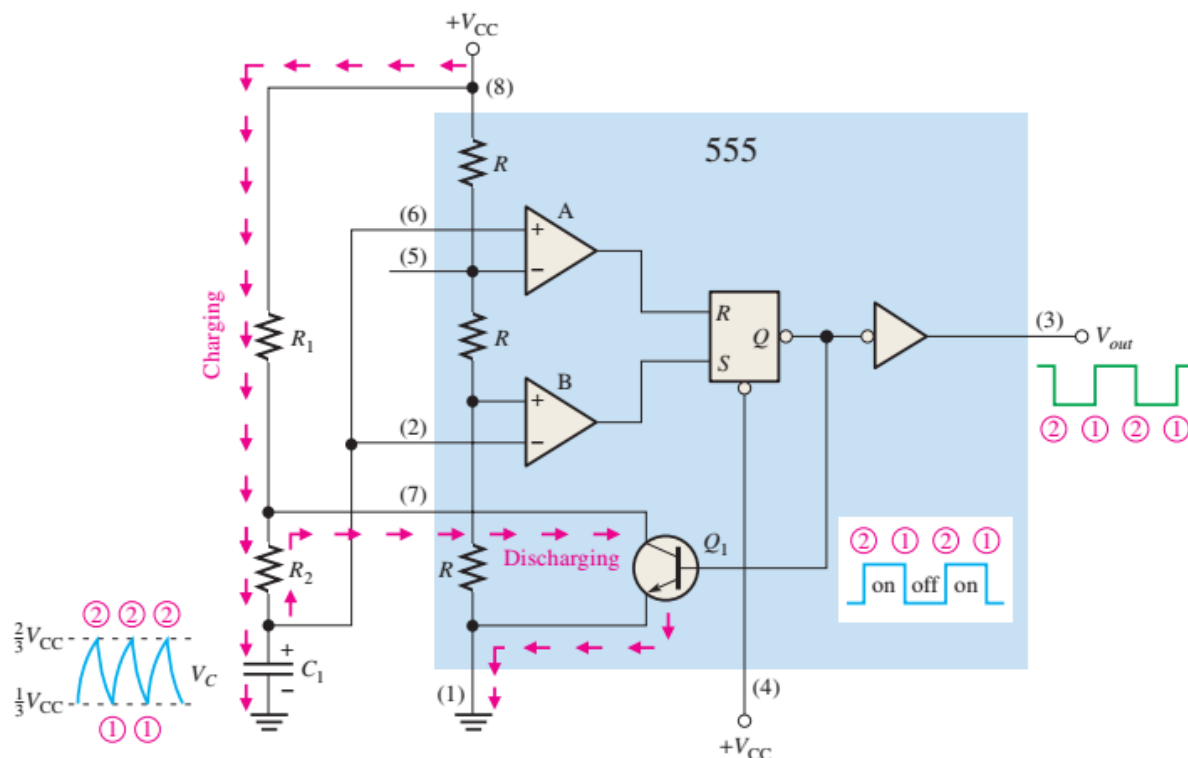
## The 555 Timer as an Astable Multivibrator

A 555 timer connected to operate as an astable multivibrator is shown in Figure 7–56. Notice that the threshold input (*THRESH*) is now connected to the trigger input (*TRIG*). The external components  $R_1$ ,  $R_2$ , and  $C_1$  form the timing network that sets the frequency of oscillation. The  $0.01\ \mu\text{F}$  capacitor,  $C_2$ , connected to the control (*CONT*) input is strictly for decoupling and has no effect on the operation; in some cases it can be left off.



**FIGURE 7–56** The 555 timer connected as an astable multivibrator (oscillator).

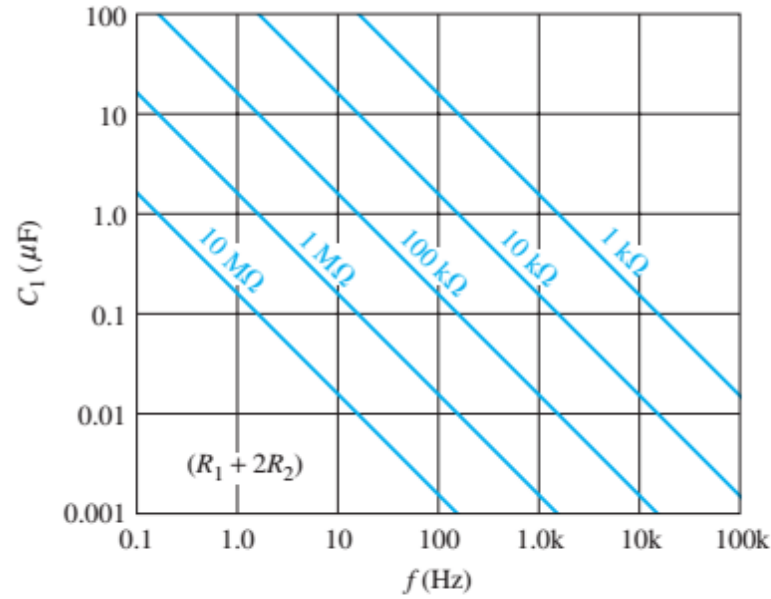
Initially, when the power is turned on, the capacitor ( $C_1$ ) is uncharged and thus the trigger voltage (pin 2) is at 0 V. This causes the output of comparator B to be HIGH and the output of comparator A to be LOW, forcing the output of the latch, and thus the base of  $Q_1$ , LOW and keeping the transistor off. Now,  $C_1$  begins charging through  $R_1$  and  $R_2$ , as indicated in Figure 7-57. When the capacitor voltage reaches  $\frac{1}{3} V_{CC}$ , comparator B switches to its LOW output state; and when the capacitor voltage reaches  $\frac{2}{3} V_{CC}$ , comparator A switches to its HIGH output state. This resets the latch, causing the base of  $Q_1$  to go HIGH and turning on the transistor. This sequence creates a discharge path for the capacitor through  $R_2$  and the transistor, as indicated. The capacitor now begins to discharge, causing comparator A to go LOW. At the point where the capacitor discharges down to  $\frac{1}{3} V_{CC}$ , comparator B switches HIGH; this sets the latch, making the base of  $Q_1$  LOW and turning off the transistor. Another charging cycle begins, and the entire process repeats. The



**FIGURE 7-57** Operation of the 555 timer in the astable mode.

result is a rectangular wave output whose duty cycle depends on the values of  $R_1$  and  $R_2$ . The frequency of oscillation is given by the following formula, or it can be found using the graph in Figure 7-58.

$$f = \frac{1.44}{(R_1 + 2R_2)C_1} \quad \text{Equation 7-4}$$



**FIGURE 7-58** Frequency of oscillation as a function of  $C_1$  and  $R_1 + 2R_2$ . The sloped lines are values of  $R_1 + 2R_2$ .

By selecting  $R_1$  and  $R_2$ , the duty cycle of the output can be adjusted. Since  $C_1$  charges through  $R_1 + R_2$  and discharges only through  $R_2$ , duty cycles approaching a minimum of 50 percent can be achieved if  $R_2 \gg R_1$  so that the charging and discharging times are approximately equal.

An expression for the duty cycle is developed as follows. The time that the output is HIGH ( $t_H$ ) is how long it takes  $C_1$  to charge from  $\frac{1}{3} V_{CC}$  to  $\frac{2}{3} V_{CC}$ . It is expressed as

$$t_H = 0.7(R_1 + R_2)C_1 \quad \text{Equation 7-5}$$

The time that the output is LOW ( $t_L$ ) is how long it takes  $C_1$  to discharge from  $\frac{1}{3} V_{CC}$  to  $\frac{2}{3} V_{CC}$ . It is expressed as

$$t_L = 0.7R_2C_1 \quad \text{Equation 7-6}$$

The period,  $T$ , of the output waveform is the sum of  $t_H$  and  $t_L$ . This is the reciprocal of  $f$  in Equation 7-4.

$$T = t_H + t_L = 0.7(R_1 + 2R_2)C_1$$

Finally, the duty cycle is

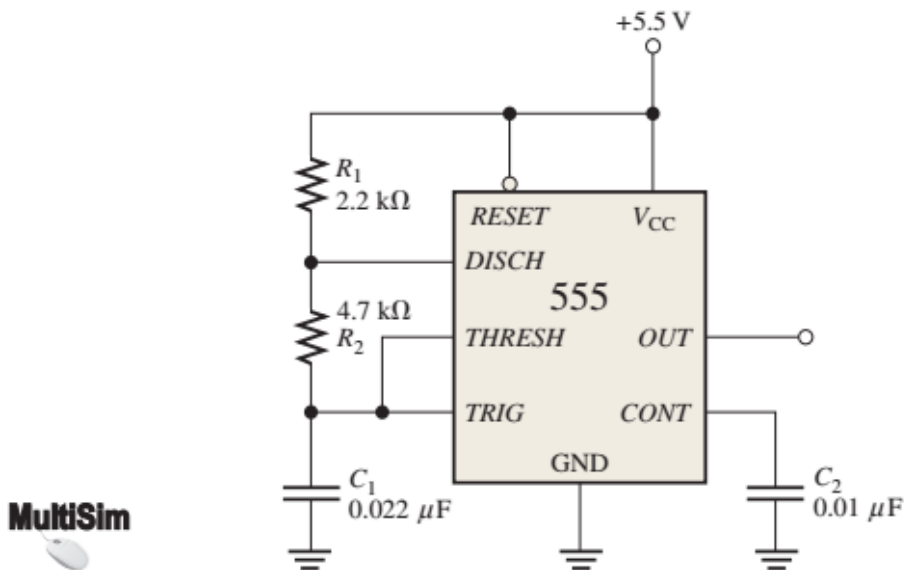
$$\begin{aligned} \text{Duty cycle} &= \frac{t_H}{T} = \frac{t_H}{t_H + t_L} \\ \text{Duty cycle} &= \left( \frac{R_1 + R_2}{R_1 + 2R_2} \right) 100\% \end{aligned} \quad \text{Equation 7-7}$$

To achieve duty cycles of less than 50 percent, the circuit in Figure 7–56 can be modified so that  $C_1$  charges through only  $R_1$  and discharges through  $R_2$ . This is achieved with a diode,  $D_1$ , placed as shown in Figure 7–59. The duty cycle can be made less than 50 percent by making  $R_1$  less than  $R_2$ . Under this condition, the expression for the duty cycle is

$$\text{Duty cycle} = \left( \frac{R_1}{R_1 + R_2} \right) 100\% \quad \text{Equation 7-8}$$

#### EXAMPLE 7-14

A 555 timer configured to run in the astable mode (pulse oscillator) is shown in Figure 7–60. Determine the frequency of the output and the duty cycle.



**FIGURE 7-60** Open file F07-60 to verify operation.

#### Solution

Use Equations 7–4 and 7–7.

$$f = \frac{1.44}{(R_1 + 2R_2)C_1} = \frac{1.44}{(2.2 \text{ k}\Omega + 9.4 \text{ k}\Omega)0.022 \text{ }\mu\text{F}} = 5.64 \text{ kHz}$$

$$\text{Duty cycle} = \left( \frac{R_1 + R_2}{R_1 + 2R_2} \right) 100\% = \left( \frac{2.2 \text{ k}\Omega + 4.7 \text{ k}\Omega}{2.2 \text{ k}\Omega + 9.4 \text{ k}\Omega} \right) 100\% = 59.5\%$$