Second Semester

Second Year

MANDALAY TECHNOLOGICAL UNIVERSITY

DEPARTMENT OF MECHATRONIC ENGINEERING

DIGITAL ELECTRONICS

McE 22046

Motto: Creative , Innovative , Mechatronics

CHAPTER 7

Latches, Flip-Flops, and Timers

CHAPTER OUTLINE

CHAPTER OBJECTIVES

- Use logic gates to construct basic latches
- Explain the difference between an S-R latch and a D latch
- Recognize the difference between a latch and a flip-flop
- Explain how D and J-K flip-flops differ
- Understand the significance of propagation delays, set-up time, hold time, maximum operating frequency, minimum clock pulse widths, and power dissipation in the application of flip-flops
- Apply flip-flops in basic applications
- Explain how retriggerable and nonretriggerable one-shots differ
- Connect a 555 timer to operate as either an astable multivibrator or a one-shot
- Describe latches, flip-flops, and timers using VHDL
- Troubleshoot basic flip-flop circuits

KEY TERMS

Key terms are in order of appearance in the chapter.

- Latch
- Bistable
- SET
- RESET
- Clock
- Edge-triggered flip-flop
- D flip-flop
- Synchronous
- J-K flip-flop
- Toggle
- Preset
- Clear
- Propagation delay time
- Set-up time
- Hold time
- Power dissipation
- One-shot
- Monostable
- Timer
- Astable

VISIT THE WEBSITE

Study aids for this chapter are available at http://www.pearsonglobaleditions.com/floyd

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.

After completing this section, you should be able to

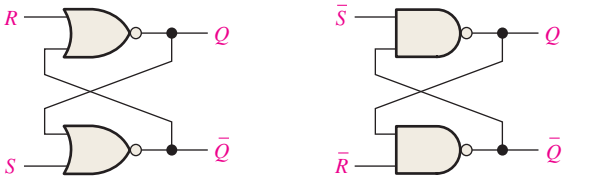
- Explain the operation of a basic S-R latch
- Explain the operation of a gated S-R latch
- Explain the operation of a gated D latch
- Implement an S-R or D latch with logic gates
- Describe the 74HC279A and 74HC75 quad latches

InfoNote

Latches are sometimes used for multiplexing data onto a bus. For example, data being input to a computer from an external source have to share the data bus with data from other sources. When the data bus becomes unavailable to the external source, the existing data must be temporarily stored, and latches placed between the external source and the data bus may be used to do this.

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 $\overline{S}$-$\overline{R}$ 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.



(a) Active-HIGH input S-R latch (b) Active-LOW input $\overline{S}$-$\overline{R}$ latch

MultiSim

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 $\overline{S}$-$\overline{R}$ 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 $\overline{S}$ and $\overline{R}$ lines are the activating inputs.

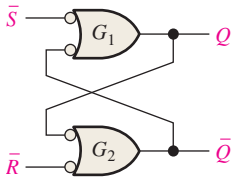


FIGURE 7–2 Negative-OR equivalent of the NAND gate $\overline{S}$-$\overline{R}$ latch in Figure 7–1(b).

A latch can reside in either of its two states, SET or RESET.

The latch in Figure 7–2 has two inputs, $\overline{S}$ and $\overline{R}$, and two outputs, Q and $\overline{Q}$. 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 $\overline{R}$ 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 $\overline{R}$ input. With a LOW on the $\overline{R}$ input and a HIGH on $\overline{S}$, the output of gate G_2 is forced HIGH. This HIGH on the $\overline{Q}$ output is coupled back to an input of G_1, and since the $\overline{S}$ 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 $\overline{Q}$ output remains HIGH even when the LOW on the $\overline{R}$ 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 $\overline{S}$ input.

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

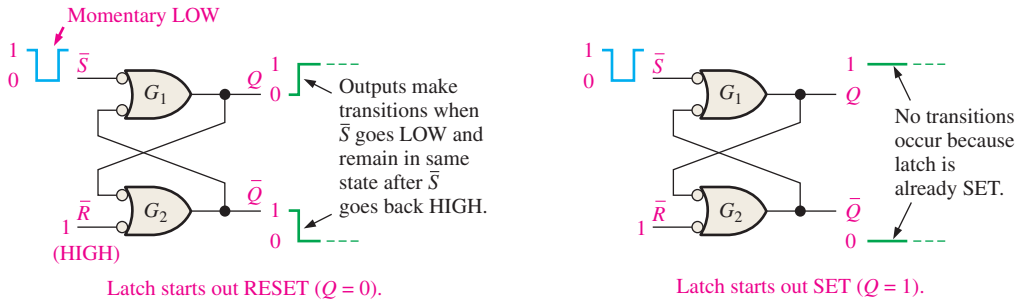
When Q is HIGH, $\overline{Q}$ is LOW, and when Q is LOW, $\overline{Q}$ is HIGH.

SET means that the Q output is HIGH.

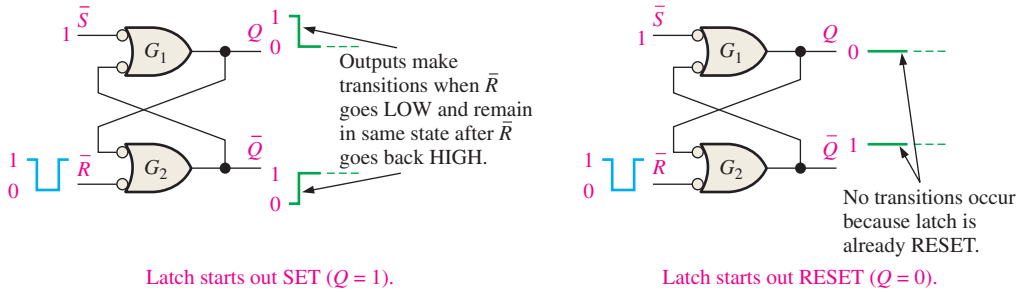
An invalid condition in the operation of an active-LOW input $\overline{S}$-$\overline{R}$ latch occurs when LOWs are applied to both $\overline{S}$ and $\overline{R}$ at the same time. As long as the LOW levels are simultaneously held on the inputs, both the Q and $\overline{Q}$ outputs are forced HIGH, 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.

RESET means that the Q output is LOW.

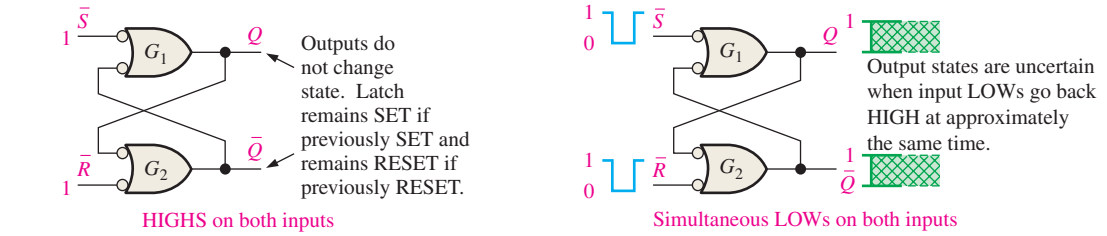
Figure 7–3 illustrates the active-LOW input $\overline{S}$-$\overline{R}$ latch operation for each of the four possible combinations of levels on the inputs. (The first three combinations are valid, but the last is not.) Table 7–1 summarizes the logic operation in truth table form. Operation of the active-HIGH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



(a) Two possibilities for the SET operation



(b) Two possibilities for the RESET operation



(c) No-change condition

(d) Invalid condition

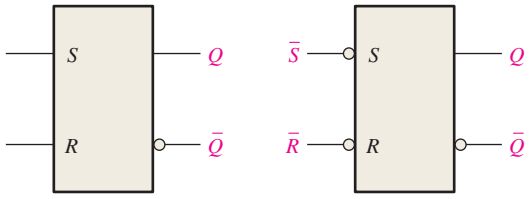
FIGURE 7–3 The three modes of basic $\overline{S}$-$\overline{R}$ latch operation (SET, RESET, no-change) and the invalid condition.

TABLE 7–1

Truth table for an active-LOW input $\overline{S}$-$\overline{R}$ latch.

Inputs		Outputs		
$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	Comments
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.



(a) Active-HIGH input S-R latch (b) Active-LOW input $\overline{S}$-$\overline{R}$ latch

FIGURE 7–4 Logic symbols for the S-R and $\overline{S}$-$\overline{R}$ latch.

Example 7–1 illustrates how an active-LOW input $\overline{S}$-$\overline{R}$ 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 $\overline{S} = 0, \overline{R} = 0$ 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 $\overline{S}$ and $\overline{R}$ 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.

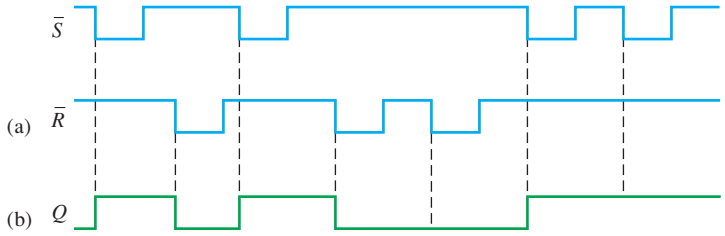


FIGURE 7–5

Solution

See Figure 7–5(b).

Related Problem*

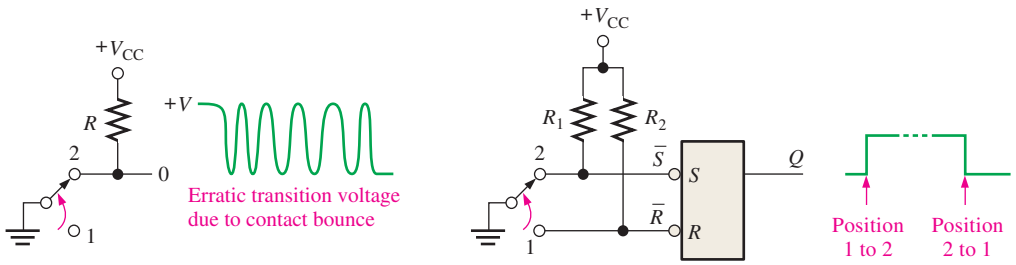
Determine the Q output of an active-HIGH input S-R latch if the waveforms in Figure 7–5(a) are inverted and applied to the inputs.

*Answers are at the end of the chapter.

An Application

The Latch as a Contact-Bounce Eliminator

A good example of an application of an $\overline{S}$-$\overline{R}$ 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).

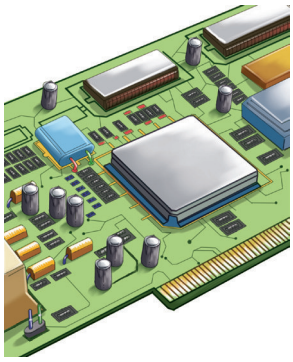


(a) Switch contact bounce (b) Contact-bounce eliminator circuit

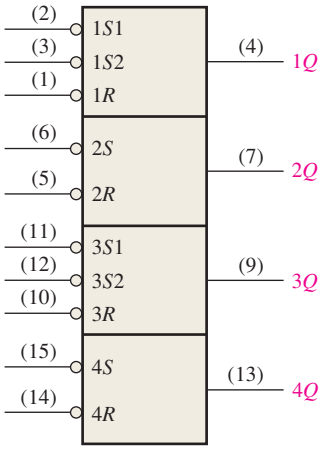
FIGURE 7–6 The $\overline{S}$-$\overline{R}$ latch used to eliminate switch contact bounce.

An $\overline{S}$-$\overline{R}$ 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 $\overline{R}$ input LOW and the latch RESET. When the switch is thrown to position 2, $\overline{R}$ goes HIGH because of the pull-up resistor to V_{CC}, and $\overline{S}$ goes LOW on the first contact. Although $\overline{S}$ 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 $\overline{S}$ 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.

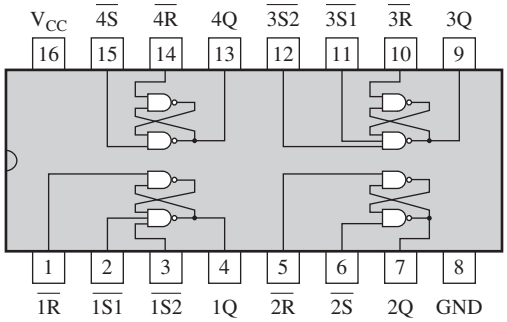
IMPLEMENTATION: $\overline{S}$-$\overline{R}$ LATCH



Fixed-Function Device The 74HC279A is a quad $\overline{S}$-$\overline{R}$ latch represented by the logic diagram of Figure 7–7(a) and the pin diagram in part (b). Notice that two of the latches each have two $\overline{S}$ inputs.



(a) Logic diagram



(b) Pin diagram

FIGURE 7–7 The 74HC279A quad $\overline{S}$-$\overline{R}$ latch.

Programmable Logic Device (PLD) An $\overline{S}$-$\overline{R}$ latch can be described using VHDL and implemented as hardware in a PLD. VHDL statements and keywords not used in previous chapters are introduced in this chapter. These are **library, use, std_logic, all,** and **inout**. The data flow approach is used in this program to describe a single $\overline{S}$-$\overline{R}$ latch. (The blue comments are not part of the program.)

```
entity SRLatch is
   port (SNot, RNot: in std_logic; Q, QNot: inout std_logic);
end entity SRLatch;

architecture LogicOperation of SRLatch is
begin
   Q <= QNot nand SNot;
   QNot <= Q nand RNot;
end architecture LogicOperation;
```

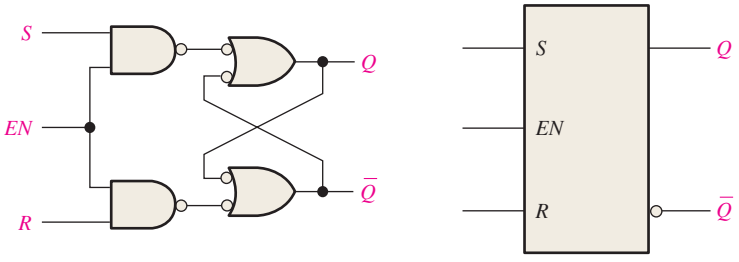
SNot: SET complement
RNot: RESET complement
Q: Latch output
QNot: Latch output complement

Boolean expressions define the outputs

The two inputs SNot and RNot are defined as **std_logic** from the IEEE library. The **inout** keyword allows the Q and QNot outputs of the latch to be used also as inputs for cross-coupling.

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.



(a) Logic diagram (b) Logic symbol

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.

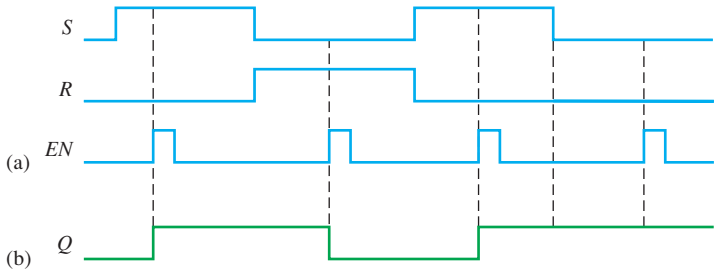


FIGURE 7–9

Solution

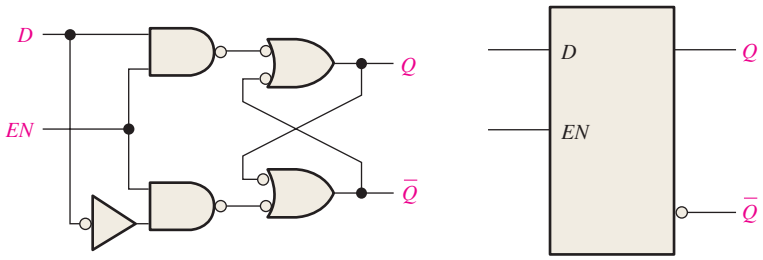
The Q waveform is shown in Figure 7–9(b). When S is HIGH and R is LOW, a HIGH on the EN input sets the latch. When S is LOW and R is HIGH, a HIGH on the EN input resets the latch. When both S and R are LOW, the Q output does not change from its present state.

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.



(a) Logic diagram (b) Logic symbol

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

MultiSim

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.

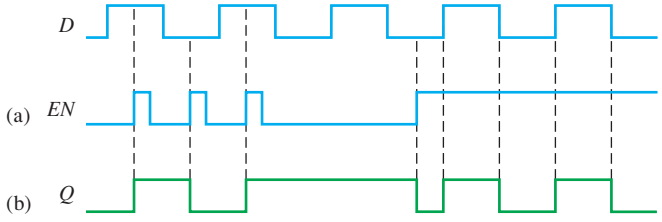


FIGURE 7–11

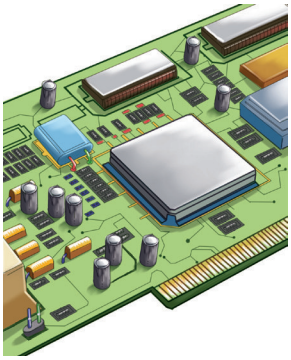
Solution

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.

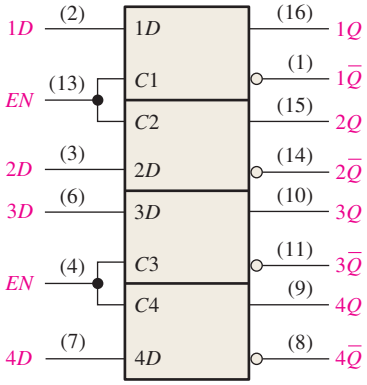
Related Problem

Determine the Q output of the gated D latch if the D input in Figure 7–11(a) is inverted.

IMPLEMENTATION: GATED D LATCH



Fixed-Function Device An example of a gated D latch is the 74HC75 represented by the logic symbol in Figure 7–12(a). The device has four latches. Notice that each active-HIGH *EN* input is shared by two latches and is designated as a control input (*C*). The truth table for each latch is shown in Figure 7–12(b). The X in the truth table represents a "don't care" condition. In this case, when the *EN* input is LOW, it does not matter what the *D* input is because the outputs are unaffected and remain in their prior states.



(a) Logic symbol

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

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

(b) Truth table (each latch)

FIGURE 7–12 The 74HC75 quad D latch.

Programmable Logic Device (PLD) The gated D latch can be described using VHDL and implemented as hardware in a PLD. The data flow approach is used in this program to describe a single D latch.

```
library ieee;
use ieee.std_logic_1164.all;

entity DLatch1 is
   port (D, EN: in std_logic; Q, QNot: inout std_logic);
end entity DLatch1;

architecture LogicOperation of DLatch1 is
begin
   Q <= QNot nand (D nand EN);
   QNot <= Q nand (not D nand EN);
end architecture LogicOperation;
```

D: Data input
EN: Enable
Q: Latch output
QNot: Latch output complement

Boolean expressions define the outputs

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 $D = 1$?

7–2 Flip-Flops

Flip-flops are synchronous bistable devices, also known as *bistable multivibrators.* In this case, the term *synchronous* means that the output changes state only at a specified point (leading or trailing edge) on the triggering input called the **clock** (CLK), which is designated as a control input, *C*; that is, changes in the output occur in synchronization with the clock. Flip-flops are edge-triggered or edge-sensitive whereas gated latches are level-sensitive.

After completing this section, you should be able to

- Define *clock*
- Define *edge-triggered flip-flop*
- Explain the difference between a flip-flop and a latch
- Identify an edge-triggered flip-flop by its logic symbol
- Discuss the difference between a positive and a negative edge-triggered flip-flop
- Discuss and compare the operation of D and J-K edge-triggered flip-flops and explain the differences in their truth tables
- Discuss the asynchronous inputs of a flip-flop

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.*

The dynamic input indicator ▷ means the flip-flop changes state only on the edge of a clock pulse.

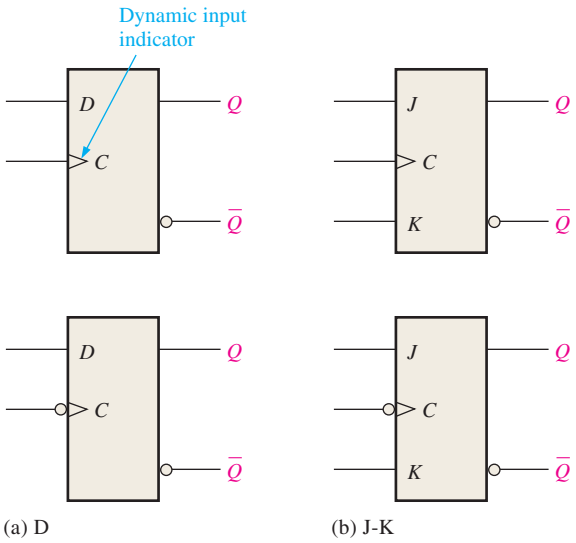


FIGURE 7–13 Edge-triggered flip-flop logic symbols (top: positive edge-triggered; bottom: negative edge-triggered).

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

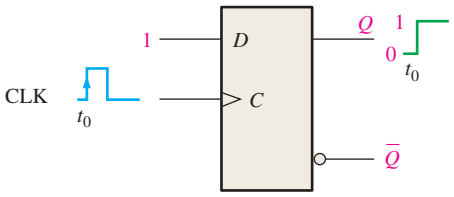
D flip-flop but *D* as variable.

InfoNote

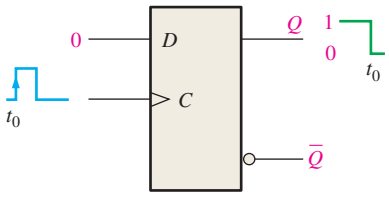
Semiconductor memories consist of large numbers of individual cells. Each storage cell holds a 1 or a 0. One type of memory is the Static Random Access Memory or SRAM, which uses flip-flops for the storage cells because a flip-flop will retain either of its two states indefinitely as long as dc power is applied, thus the term *static*. This type of memory is classified as a *volatile* memory because all the stored data are lost when power is turned off. Another type of memory, the Dynamic Random Access Memory or DRAM, uses capacitance rather than flip-flops as the basic storage element and must be periodically refreshed in order to maintain the stored data.

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.



(a) $D = 1$ flip-flop SETS on positive clock edge. (If already SET, it remains SET.)



(b) $D = 0$ flip-flop RESETS on positive clock edge. (If already RESET, it remains RESET.)

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		
D	CLK	Q	$\overline{Q}$	Comments
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 $\overline{Q}$ 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.

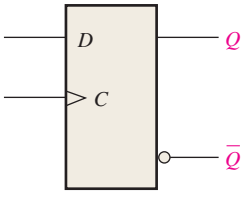


FIGURE 7–15

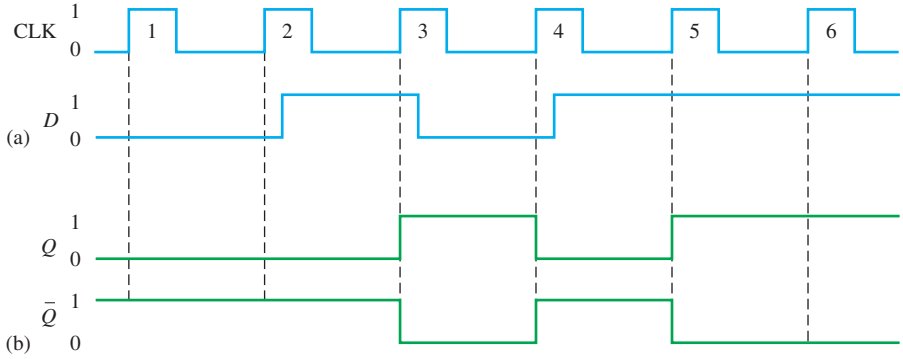


FIGURE 7–16

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, $\overline{Q}$ is easily found since it is simply the complement of Q. The resulting waveforms for Q and $\overline{Q}$ are shown in Figure 7–16(b) for the input waveforms in part (a).

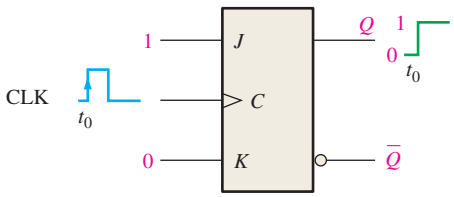
Related Problem

Determine Q and $\overline{Q}$ for the D input in Figure 7–16(a) if the flip-flop is a negative edge-triggered device.

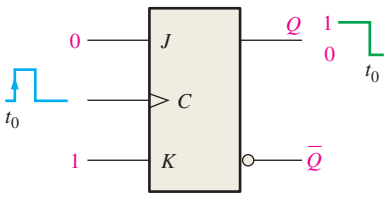
The J-K Flip-Flop

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.

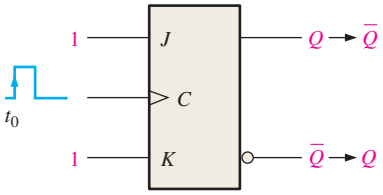
This basic operation of a positive edge-triggered flip-flop is illustrated in Figure 7–17, and Table 7–3 is the truth table for this type of flip-flop. Remember, *the flip-flop cannot change state except on the triggering edge of a clock pulse.* The J and K inputs can be changed at any time when the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



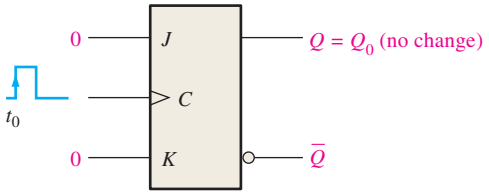
(a) $J = 1$, $K = 0$ flip-flop SETS on positive clock edge. (If already SET, it remains SET.)



(b) $J = 0$, $K = 1$ flip-flop RESETS on positive clock edge. (If already RESET, it remains RESET.)



(c) $J = 1$, $K = 1$ flip-flop changes state (toggle).



(d) $J = 0$, $K = 0$ flip-flop does not change. (If SET, it remains SET; if RESET, it remains RESET.)

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		
J	K	CLK	Q	$\overline{Q}$	Comments
0	0	↑	Q_0	$\overline{Q}_0$	No change
0	1	↑	0	1	RESET
1	0	↑	1	0	SET
1	1	↑	$\overline{Q}_0$	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.

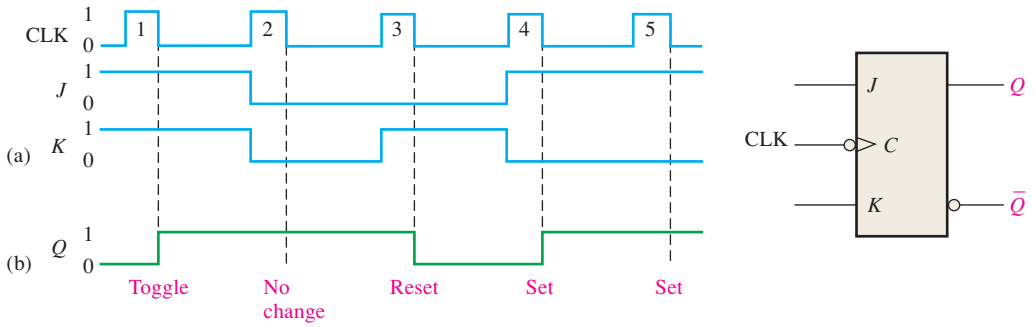


FIGURE 7–18

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).

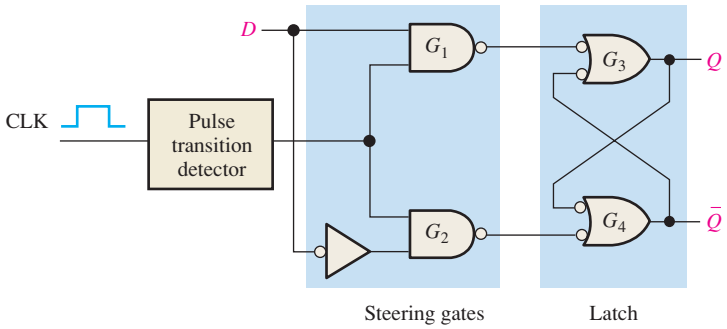
Related Problem

Determine the Q output of the J-K flip-flop if the J and K inputs in Figure 7–18(a) are inverted.

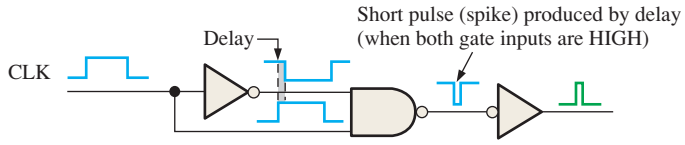
Edge-Triggered Operation

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.



(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.

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 $\overline{Q}$ output HIGH. Because $\overline{Q}$ 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.

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 $\overline{Q}$ output LOW. This LOW on $\overline{Q}$ 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 $\overline{Q}$ output to go HIGH. Because of this HIGH on $\overline{Q}$, 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 $\overline{Q}$ 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.

InfoNote

All logic operations that are performed with hardware can also be implemented in software. For example, the operation of a J-K flip-flop can be performed with specific computer instructions. If two bits were used to represent the *J* and *K* inputs, the computer would do nothing for 00, a data bit representing the *Q* output would be set (1) for 10, the *Q* data bit would be cleared (0) for 01, and the *Q* data bit would be complemented for 11. Although it may be unusual to use a computer to simulate a flip-flop, the point is that all hardware operations can be simulated using software.

The *Q* output of a D flip-flop assumes the state of the *D* input on the triggering edge of the clock.

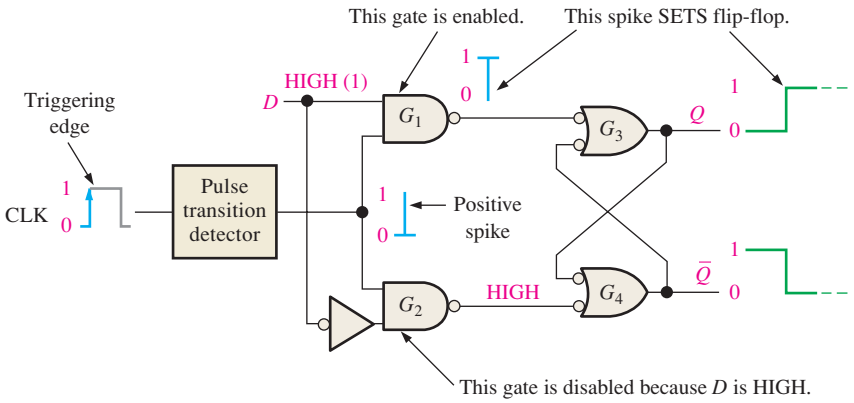


FIGURE 7–20 Flip-flop making a transition from the RESET state to the SET state on the positive-going edge of the clock pulse.

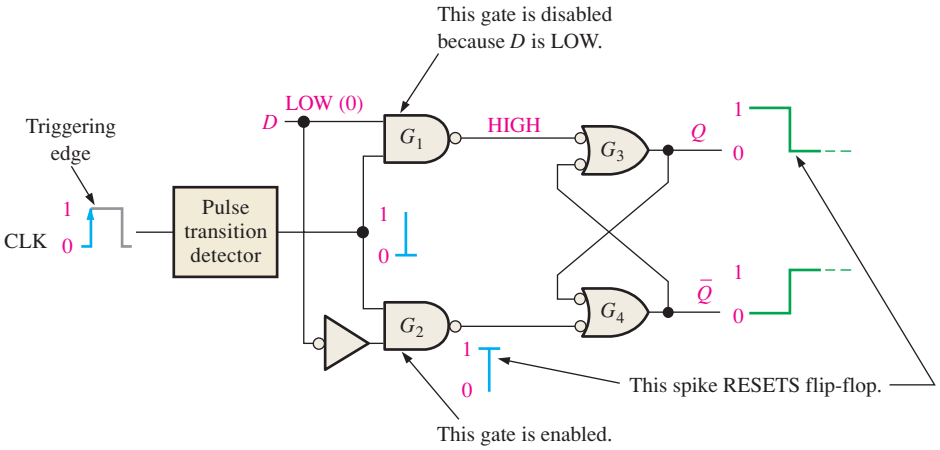


FIGURE 7–21 Flip-flop making a transition from the SET state to the RESET state on the positive-going edge of the clock pulse.

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.

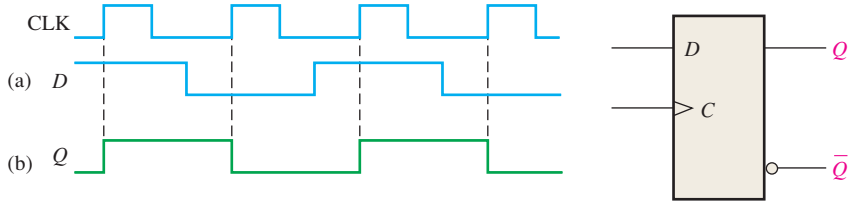


FIGURE 7–22

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).

Related Problem

Determine the *Q* output for the D flip-flop if the *D* input in Figure 7–22(a) is inverted.

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 $\overline{Q}$ 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.

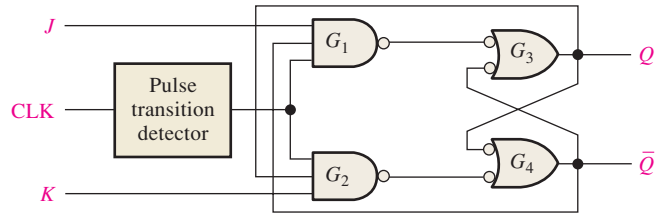


FIGURE 7–23 A simplified logic diagram for a positive edge-triggered J-K flip-flop.

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 $\overline{Q}$ 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.

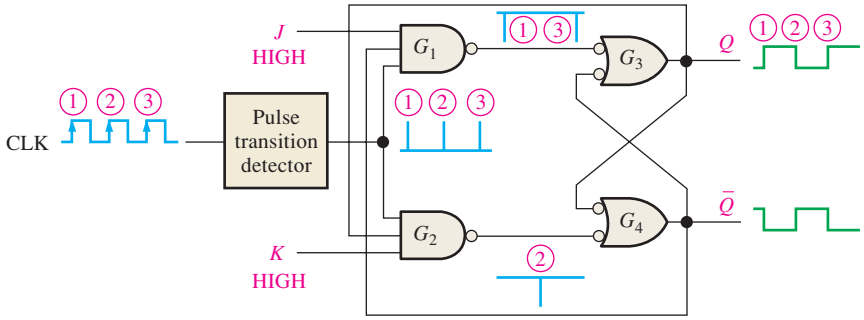


FIGURE 7–24 Transitions illustrating flip-flop operation.

If you make J LOW and K HIGH, the next clock spike indicated by ② will pass through gate G_2 because Q is HIGH and K is HIGH. This will cause the latch portion of the flip-flop to change to the RESET state.

In the toggle mode, a J-K flip-flop changes state on every clock pulse.

If you apply a LOW to both the J and K inputs, the flip-flop will stay in its present state when a clock pulse occurs. A LOW on both J and K results in a *no-change* condition.

When both the J and K inputs are HIGH and the flip-flop is RESET, the HIGH on the $\overline{Q}$ enables gate G_1; so the clock spike indicated by ③ passes through to set the flip-flop. Now there is a HIGH on Q, which allows the next clock spike to pass through gate G_2 and reset the flip-flop.

As you can see, on each successive clock spike, the flip-flop toggles to the opposite state. Figure 7–24 illustrates the transitions when the flip-flop is in the toggle mode. A J-K flip-flop connected for toggle operation is sometimes called a *T flip-flop*.

Asynchronous Preset and Clear Inputs

For the flip-flops just discussed, the D and J-K inputs are called *synchronous inputs* because data on these inputs are transferred to the flip-flop's output only on the triggering edge of the clock pulse; that is, the data are transferred synchronously with the clock.

An active preset input makes the Q output HIGH (SET).

An active clear input makes the *Q* output LOW (RESET).

Most integrated circuit flip-flops also have **asynchronous** inputs. These are inputs that affect the state of the flip-flop *independent of the clock.* They are normally labeled **preset** (*PRE*) and **clear** (*CLR*), or *direct set* (S_D) and *direct reset* (R_D) by some manufacturers. An active level on the preset input will set the flip-flop, and an active level on the clear input will reset it. A logic symbol for a D flip-flop with preset and clear inputs is shown in Figure 7–25. These inputs are active-LOW, as indicated by the bubbles. These preset and clear inputs must both be kept HIGH for synchronous operation. In normal operation, preset and clear would not be LOW at the same time.

Figure 7–26 shows the logic diagram for an edge-triggered D flip-flop with active-LOW preset ($\overline{PRE}$) and clear ($\overline{CLR}$) inputs. This figure illustrates basically how these inputs work. As you can see, they are connected so that they override the effect of the synchronous input, *D* and the clock.

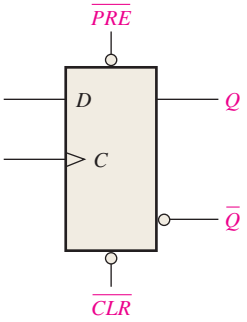


FIGURE 7–25 Logic symbol for a D flip-flop with active-LOW preset and clear inputs.

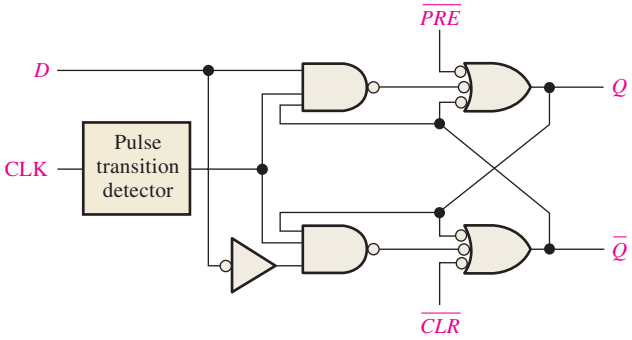
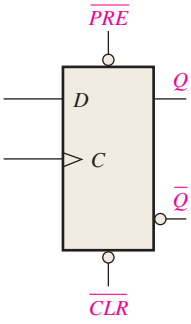
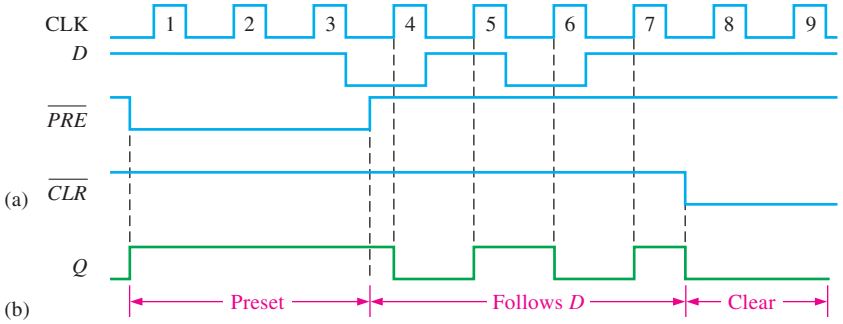


FIGURE 7–26 Logic diagram for a basic D flip-flop with active-LOW preset and clear inputs.

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.





MultiSim

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).

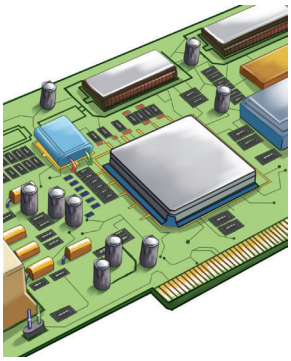
Related Problem

If you interchange the $\overline{PRE}$ and $\overline{CLR}$ waveforms in Figure 7–27(a), what will the Q output look like?

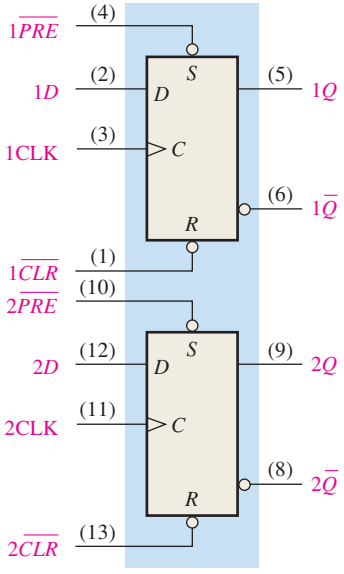
Let's look at two specific edge-triggered flip-flops. They are representative of the various types of flip-flops available in fixed-function IC form and, like most other devices, are available in CMOS and in bipolar (TTL) logic families.

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

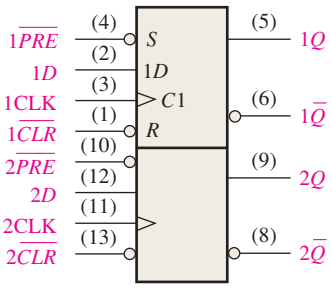
IMPLEMENTATION: D FLIP-FLOP



Fixed-Function Device The 74HC74 dual D flip-flop contains two identical D flip-flops that are independent of each other except for sharing V_{CC} and ground. The flip-flops are positive edge-triggered and have active-LOW asynchronous preset and clear inputs. The logic symbols for the individual flip-flops within the package are shown in Figure 7–28(a), and an ANSI/IEEE standard single block symbol that represents the entire device is shown in part (b). The pin numbers are shown in parentheses.



(a) Individual logic symbols



(b) Single block logic symbol
Note: The S and R inside the block indicate that $\overline{PRE}$ SETS and $\overline{CLR}$ RESETS.

FIGURE 7–28 The 74HC74 dual positive edge-triggered D flip-flop.

Programmable Logic Device (PLD) The positive edge-triggered D flip-flop can be described using VHDL and implemented as hardware in a PLD. In this program, the behavioral approach will be used for the first time because it lends itself to describing sequential operations. A new VHDL statement, **wait until rising_edge**, is introduced. This statement allows the program to wait for the rising edge of a clock pulse to process the *D* input to create the desired results. Also the **if then else** statement is introduced. The keyword **process** is a block of code placed between the **begin** and **end** statements of the architecture to allow statements to be sequentially processed. The program code for a single D flip-flop is as follows:

```
library ieee;
use ieee.std_logic_1164.all;

entity dffl is
   port (D, Clock, Pre, Clr: in std_logic; Q: inout std_logic);
end entity dffl;

architecture LogicOperation of dffl is
begin
process
   begin
      wait until rising_edge (Clock);
         if Clr = '1' then
            if Pre = '1' then
               if D = '1' then
                  Q <= '1';
               else
                  Q <= '0';
               end if;
            else
               Q <= '1';
            end if;
         else
            Q <= '0';
      end if;
   end process;
end architecture LogicOperation;
```

D: Flip-flop input
Clock: System clock
Pre: Preset input
Clr: Clear input
Q: Flip-flop output

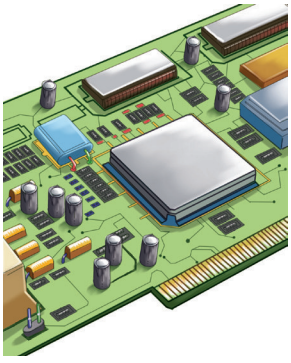
Check for Preset and Clear conditions

Q input follows D input when Clr and Pre inputs are HIGH.

Q is set HIGH when Pre input is LOW.

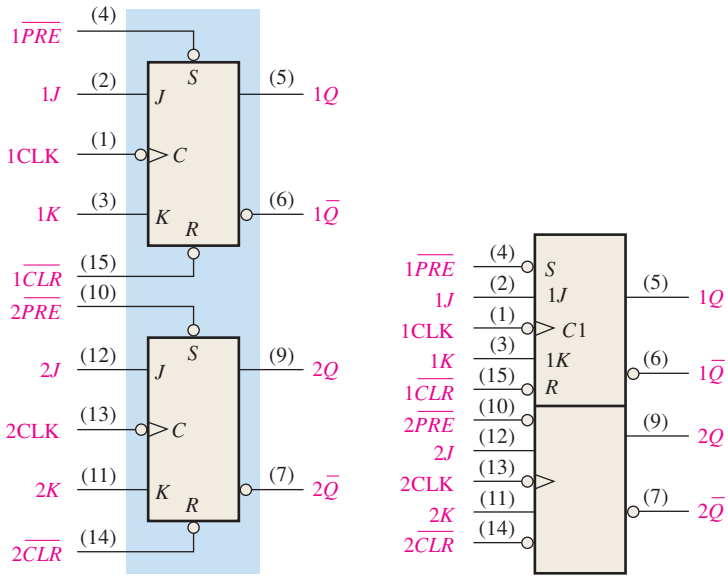
Q is set LOW when Clr input is LOW.

IMPLEMENTATION: J-K FLIP-FLOP



Fixed-Function Device The 74HC112 dual J-K flip-flop has two identical flip-flops that are negative edge-triggered and have active-LOW asynchronous preset and clear inputs. The logic symbols are shown in Figure 7–29.

Programmable Logic Device (PLD) The negative edge-triggered J-K flip-flop can be described using VHDL and implemented as hardware in a PLD. In this program, the behavioral approach will be used. A new VHDL statement, **if falling edge then,** is introduced. This statement allows the program to wait for the falling edge of a clock pulse



(a) Individual logic symbols (b) Single block logic symbol

FIGURE 7–29 The 74HC112 dual negative edge-triggered J-K flip-flop.

to process the *J* and *K* inputs to create the desired results. The following program code describes a single J-K flip-flop with no preset or clear inputs.

```
library ieee;
use ieee.std_logic_1164.all;
entity JKFlipFlop is
   port (J, K, Clock: in std_logic; Q, QNot: inout std_logic);
end entity JKFlipFlop;

architecture LogicOperation of JKFlipFlop is
signal J1, K1: std_logic;

begin
process (J, K, Clock, J1, K1, Q, QNot)
   begin
      if falling_edge(Clock) and Clock = '0' then
         J1 <= not (J and not Clock and QNot);
         K1 <= not (K and not Clock and Q);
      end if;
         Q <= J1 nand QNot;
      QNot <= K1 nand Q;
end process;
end architecture LogicOperation;
```

Inputs and outputs declared

Identifies with Boolean expressions the inputs (J1 and K1) to the latch portion of the flip-flop

Defines the outputs in terms of J1 and K1 with Boolean expressions

EXAMPLE 7–8

The 1*J*, 1*K*, 1CLK, 1$\overline{PRE}$, and 1$\overline{CLR}$ waveforms in Figure 7–30(a) are applied to one of the negative edge-triggered flip-flops in a 74HC112 package. Determine the 1*Q* output waveform.

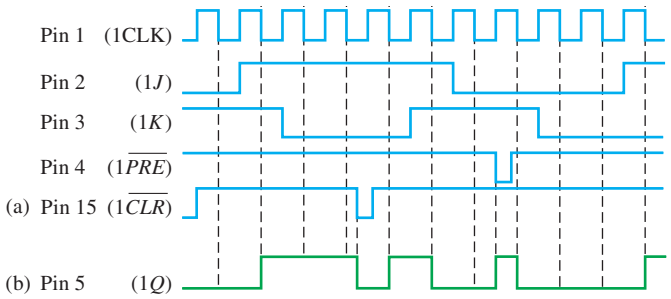


FIGURE 7–30

Solution

The resulting 1Q waveform is shown in Figure 7–30(b). Notice that each time a LOW is applied to the 1$\overline{PRE}$ or 1$\overline{CLR}$, the flip-flop is set or reset regardless of the states of the other inputs.

Related Problem

Determine the 1Q output waveform if the waveforms for 1$\overline{PRE}$ and 1$\overline{CLR}$ are interchanged.

SECTION 7–2 CHECKUP

1. Describe the main difference between a gated D latch and an edge-triggered D flip-flop.
2. How does a J-K flip-flop differ from a D flip-flop in its basic operation?
3. Assume that the flip-flop in Figure 7–22 is negative edge-triggered. Describe the output waveform for the same CLK and D waveforms.

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.

After completing this section, you should be able to

- Define *propagation delay time*
- Explain the various propagation delay time specifications
- Define *set-up time* and discuss how it limits flip-flop operation
- Define *hold time* and discuss how it limits flip-flop operation
- Discuss the significance of maximum clock frequency
- Discuss the various pulse width specifications
- Define *power dissipation* and calculate its value for a specific device
- Compare various series of flip-flops in terms of their operating parameters

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).

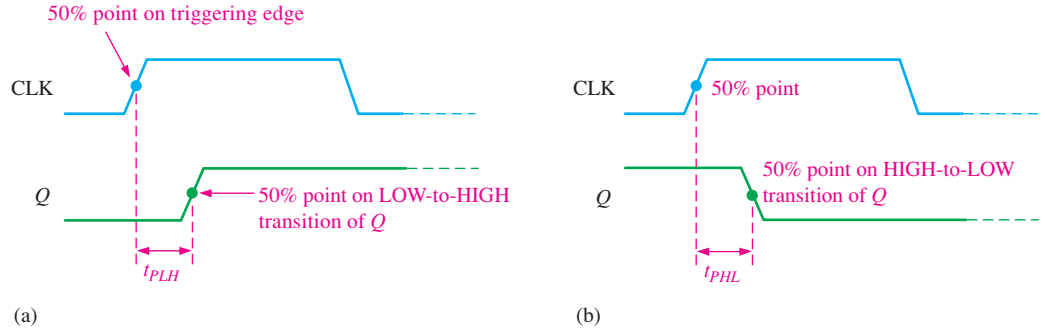


FIGURE 7–31 Propagation delays, clock to output.

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.

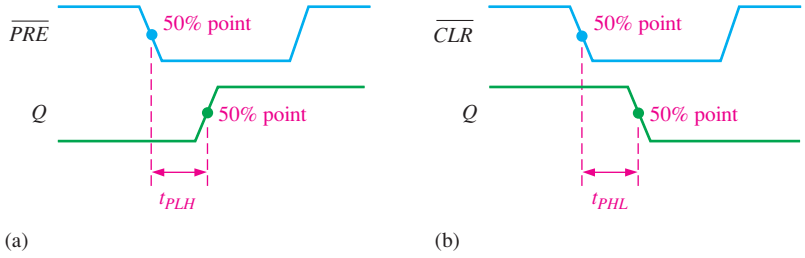


FIGURE 7–32 Propagation delays, preset input to output and clear input to output.

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.

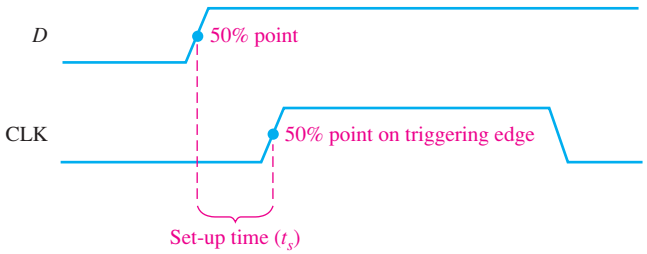


FIGURE 7–33 Set-up time (t_s). The logic level must be present on the *D* input for a time equal to or greater than t_s before the triggering edge of the clock pulse for reliable data entry.

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.

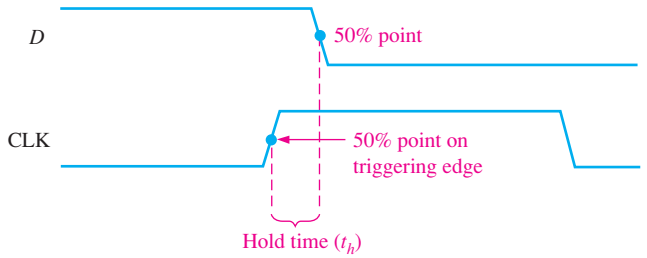


FIGURE 7–34 Hold time (t_h). The logic level must remain on the *D* input for a time equal to or greater than t_h after the triggering edge of the clock pulse for reliable data entry.

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

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

An advantage of CMOS is that it can operate over a wider range of dc supply voltages (typically 2 V to 6 V) than bipolar and, therefore, less expensive power supplies that do not have precise regulation can be used. Also, batteries can be used as secondary or primary sources for CMOS circuits. In addition, lower voltages mean that the IC dissipates less power. The drawback is that the performance of CMOS is degraded with lower supply voltages. For example, the guaranteed maximum clock frequency of a CMOS flip-flop is much less at $V_{CC} = 2$ V than at $V_{CC} = 6$ V.

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.

Comparison of Specific Flip-Flops

Table 7–4 provides a comparison, in terms of the operating parameters discussed in this section, of four CMOS and bipolar (TTL) flip-flops of the same type but with different IC families (HC, AHC, LS, and F).

TABLE 7–4

Comparison of operating parameters for four IC families of flip-flops of the same type at 25°C.

	CMOS		Bipolar (TTL)	
Parameter	**74HC74A**	**74AHC74**	**74LS74A**	**74F74**
t_{PHL} (CLK to Q)	17 ns	4.6 ns	40 ns	6.8 ns
t_{PLH} (CLK to Q)	17 ns	4.6 ns	25 ns	8.0 ns
t_{PHL}($\overline{CLR}$ to Q)	18 ns	4.8 ns	40 ns	9.0 ns
t_{PLH}($\overline{PRE}$ to Q)	18 ns	4.8 ns	25 ns	6.1 ns
t_s (set-up time)	14 ns	5.0 ns	20 ns	2.0 ns
t_h (hold time)	3.0 ns	0.5 ns	5 ns	1.0 ns
t_W (CLK HIGH)	10 ns	5.0 ns	25 ns	4.0 ns
t_W (CLK LOW)	10 ns	5.0 ns	25 ns	5.0 ns
t_W($\overline{CLR}$/$\overline{PRE}$)	10 ns	5.0 ns	25 ns	4.0 ns
f_{max}	35 MHz	170 MHz	25 MHz	100 MHz
Power, quiescent	0.012 mW	1.1 mW		
Power, 50% duty cycle			44 mW	88 mW

SECTION 7–3 CHECKUP

1. Define the following:
   (a) set-up time **(b)** hold time
2. Which specific flip-flop in Table 7–4 can be operated at the highest frequency?

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.

After completing this section, you should be able to

- Discuss the application of flip-flops in data storage
- Describe how flip-flops are used for frequency division
- Explain how flip-flops are used in basic counter applications

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.

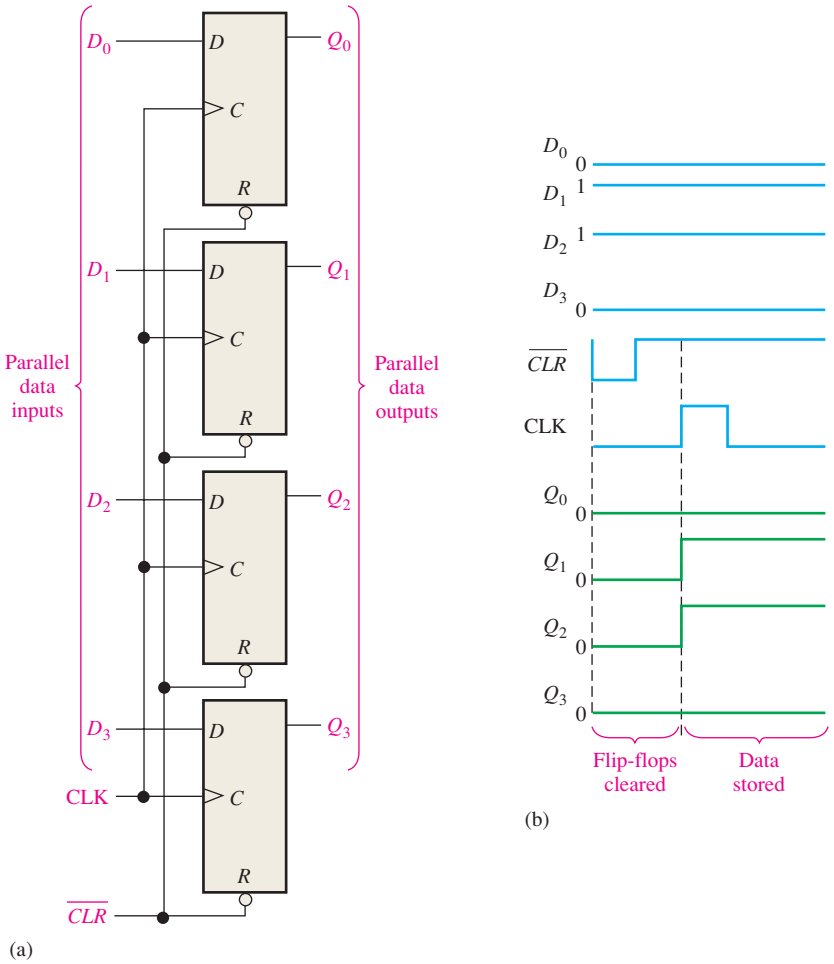


FIGURE 7–35 Example of flip-flops used in a basic register for parallel data storage.

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.

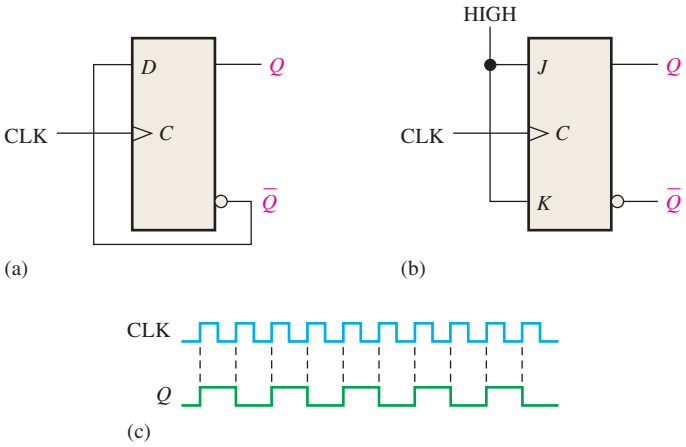


FIGURE 7–36 The D flip-flop and J-K flip-flop as a divide-by-2 device. Q is one-half the frequency of CLK. Open file F07-36 and verify the operation.

MultiSim

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.

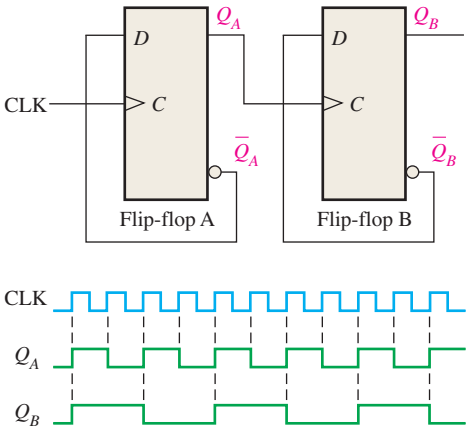


FIGURE 7–37 Example of two D flip-flops used to divide the clock frequency by 4. Q_A is one-half and Q_B is one-fourth the frequency of CLK. Open file F07-37 and verify the operation.

MultiSim

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_{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.

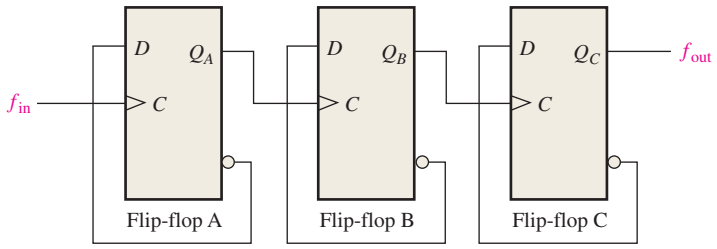


FIGURE 7–38

Solution

The three flip-flops are connected to divide the input frequency by eight ($2^3 = 8$) and the Q_C (f_{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.

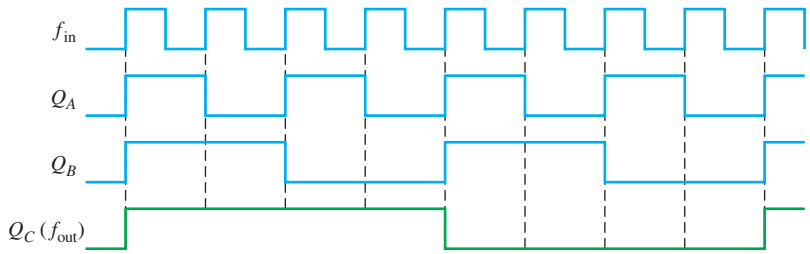


FIGURE 7–39

Related Problem

How many flip-flops are required to divide a frequency by thirty-two?

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.

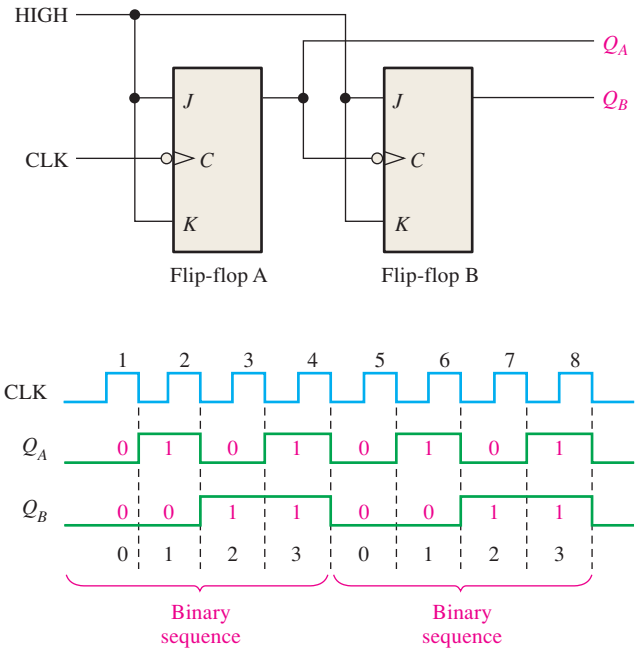


FIGURE 7–40 J-K flip-flops used to generate a binary count sequence (00, 01, 10, 11). Two repetitions are shown.

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.

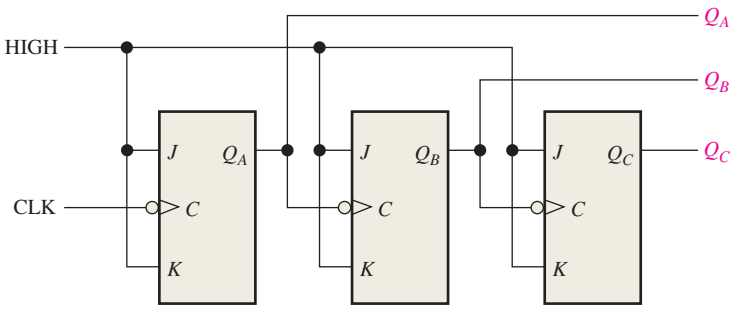


FIGURE 7–41

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.

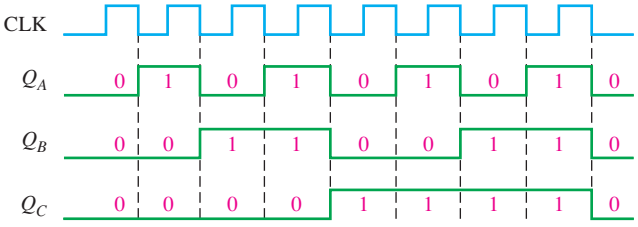


FIGURE 7–42

Related Problem

How many flip-flops are required to produce a binary sequence representing decimal numbers 0 through 15?

SECTION 7–4 CHECKUP

1. What is a group of flip-flops used for data storage called?
2. How must a D flip-flop be connected to function as a divide-by-2 device?
3. How many flip-flops are required to produce a divide-by-64 device?

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.

After completing this section, you should be able to

- Describe the basic operation of a one-shot
- Explain how a nonretriggerable one-shot works
- Explain how a retriggerable one-shot works
- Set up the 74121 and the 74LS122 one-shots to obtain a specified output pulse width
- Recognize a Schmitt trigger symbol and explain basically what it means
- Describe the basic elements of a 555 timer
- Set up a 555 timer as a one-shot

A one-shot produces a single pulse each time it is triggered.

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.

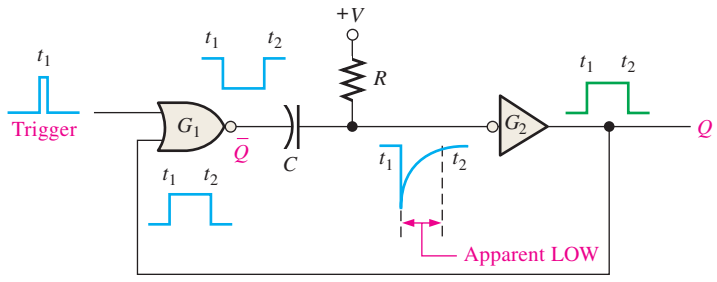


FIGURE 7–43 A simple one-shot circuit.

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.

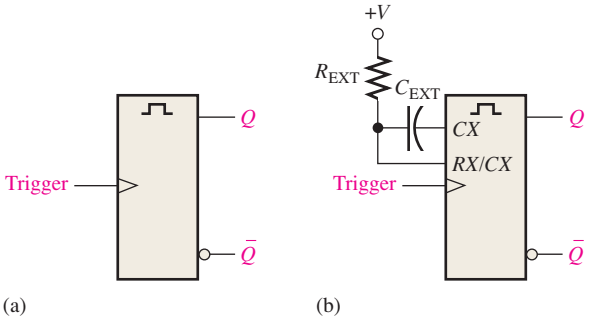


FIGURE 7–44 Basic one-shot logic symbols. *CX* and *RX* stand for external components.

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.

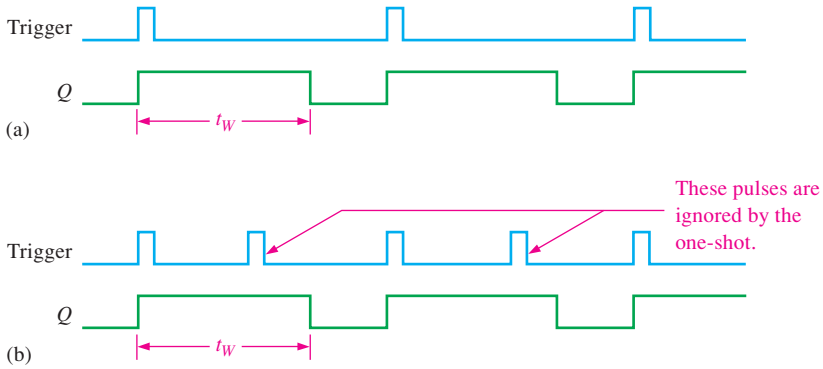


FIGURE 7–45 Nonretriggerable one-shot action.

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.

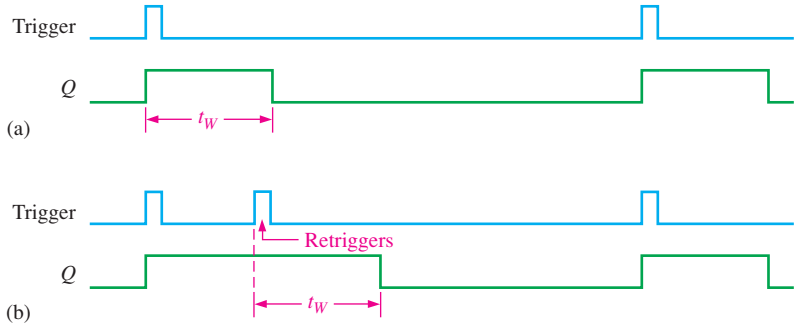
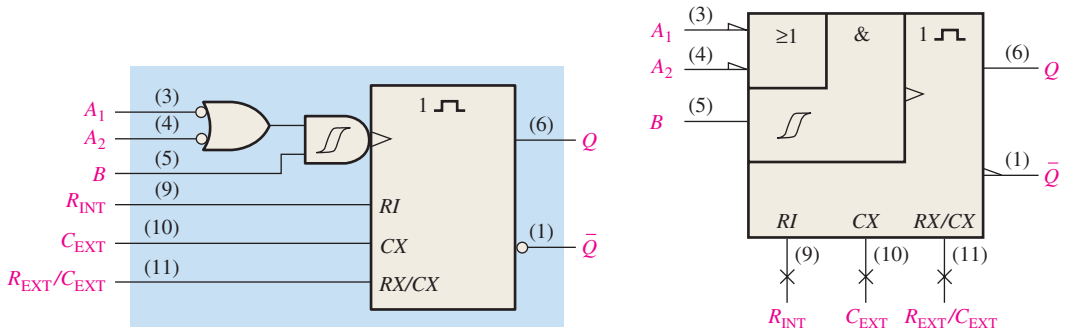


FIGURE 7–46 Retriggerable one-shot action.

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 kΩ internal timing resistor.



(a) Traditional logic symbol

(b) ANSI/IEEE std. 91–1984 logic symbol (× = nonlogic connection). "1⊓" is the qualifying symbol for a nonretriggerable one-shot.

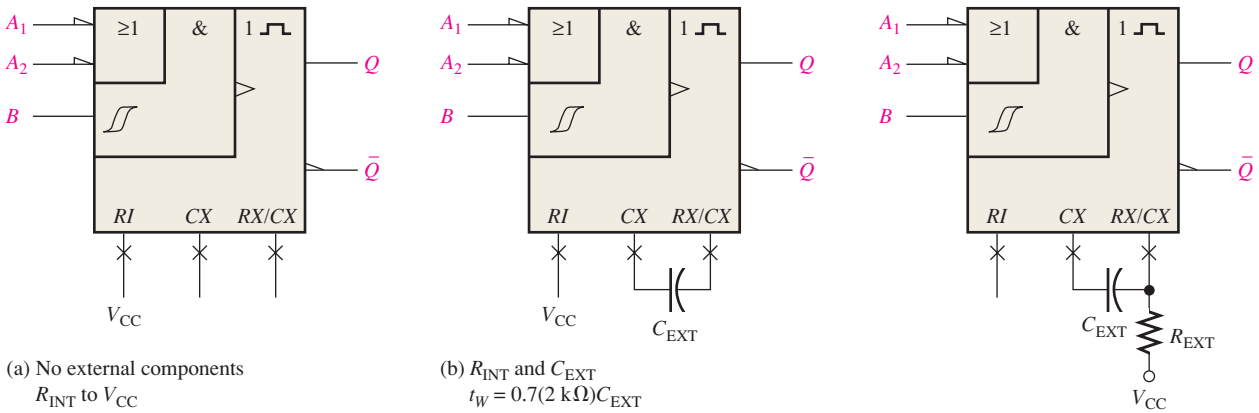
FIGURE 7–47 Logic symbols for the 74121 nonretriggerable one-shot.

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 kΩ) 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 kΩ, and R_{EXT} is selected) and the capacitor according to the following formula:

$$t_W = 0.7RC_{EXT} \qquad \text{Equation 7–1}$$

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



(a) No external components
R_{INT} to V_{CC}
$t_W \cong 30$ ns

(b) R_{INT} and C_{EXT}
$t_W = 0.7(2\ \text{k}\Omega)C_{EXT}$

(c) R_{EXT} and C_{EXT}
$t_W = 0.7R_{EXT}\,C_{EXT}$

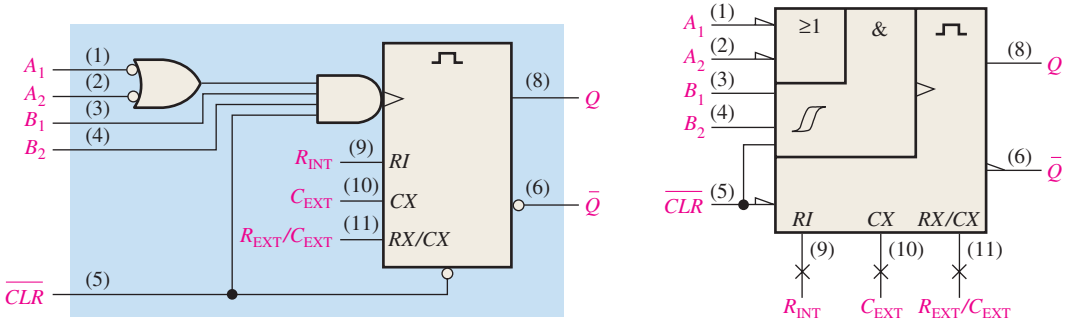
FIGURE 7–48 Three ways to set the pulse width of a 74121.

The Schmitt-Trigger Symbol

The symbol ⨌ 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.



(a) Traditional logic symbol

(b) ANSI/IEEE std. 91–1984 logic symbol (× = nonlogic connection). ⨍ is the qualifying symbol for a retriggerable one-shot.

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

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

where 0.32 is a constant determined by the particular type of one-shot, R is in kΩ 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 kΩ 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 7–11

A certain application requires a one-shot with a pulse width of approximately 100 ms. Using a 74121, show the connections and the component values.

Solution

Arbitrarily select $R_{EXT} = \mathbf{39\ k\Omega}$ and calculate the necessary capacitance.

$$t_W = 0.7R_{EXT}C_{EXT}$$

$$C_{EXT} = \frac{t_W}{0.7R_{EXT}}$$

where C_{EXT} is in pF, R_{EXT} is in kΩ, and t_W is in ns. Since 100 ms $= 1 \times 10^8$ ns,

$$C_{EXT} = \frac{1 \times 10^8\ \text{ns}}{0.7(39\ \text{k}\Omega)} = 3.66 \times 10^{-6}\ \text{pF} = \mathbf{3.66\ \mu F}$$

A standard 3.3 μF capacitor will give an output pulse width of 91 ms. The proper connections are shown in Figure 7–50. To achieve a pulse width closer to 100 ms, other combinations of values for R_{EXT} and C_{EXT} can be tried. For example, $R_{EXT} = 68\ k\Omega$ and $C_{EXT} = 2.2\ \mu F$ gives a pulse width of 105 ms.

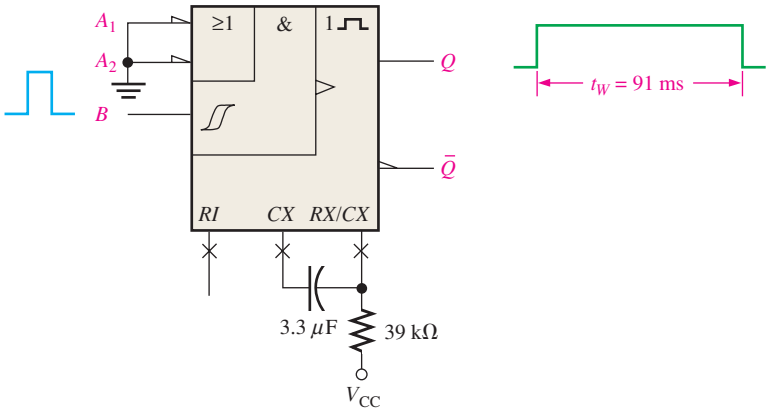


FIGURE 7–50

Related Problem

Use an external capacitor in conjunction with R_{INT} to produce an output pulse width of 10 μs from the 74121.

EXAMPLE 7–12

Determine the values of R_{EXT} and C_{EXT} that will produce a pulse width of 1 μs when connected to a 74LS122.

Solution

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

$$t_w = 0.32R_{EXT}C_{EXT}\left(1 + \frac{0.7}{R_{EXT}}\right) = 0.32R_{EXT}C_{EXT} + 0.7\left(\frac{0.32R_{EXT}C_{EXT}}{R_{EXT}}\right)$$

$$= 0.32R_{EXT}C_{EXT} + (0.7)(0.32)C_{EXT}$$

$$R_{EXT} = \frac{t_W - (0.7)(0.32)C_{EXT}}{0.32C_{EXT}} = \frac{t_W}{0.32C_{EXT}} - 0.7$$

$$= \frac{1000\ ns}{(0.32)560\ pF} - 0.7 = \mathbf{4.88\ k\Omega}$$

Use a standard value of **4.7 kΩ.**

Related Problem

Show the connections and component values for a 74LS122 one-shot with an output pulse width of 5 μs. Assume $C_{EXT} = 560$ pF.

An Application

One practical one-shot application is a sequential timer that can be used to illuminate a series of lights. This type of circuit can be used, for example, in a lane change directional indicator for highway construction projects or in sequential turn signals on automobiles.

Figure 7–51 shows three 74LS122 one-shots connected as a sequential timer. This particular circuit produces a sequence of three 1 s pulses. The first one-shot is triggered by a switch closure or a low-frequency pulse input, producing a 1 s output pulse. When the first one-shot (OS 1) times out and the 1 s pulse goes LOW, the second one-shot (OS 2) is triggered, also producing a 1 s output pulse. When this second pulse goes LOW, the third one-shot (OS 3) is triggered and the third 1 s pulse is produced. The output timing is illustrated in the figure. Variations of this basic arrangement can be used to produce a variety of timed outputs.

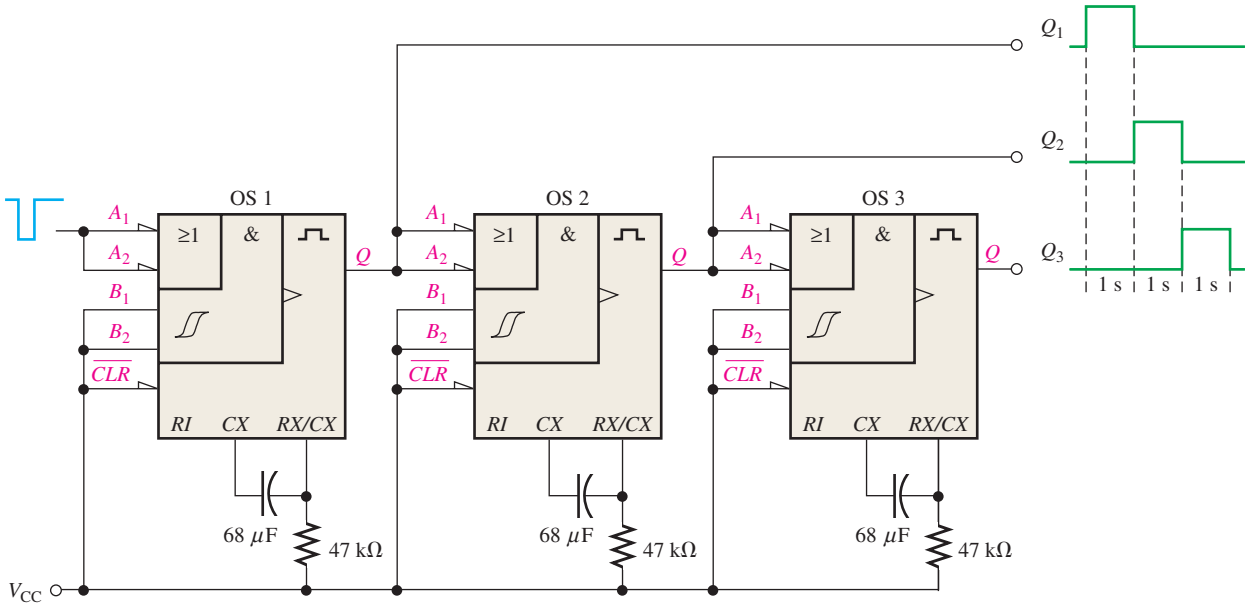


FIGURE 7–51 A sequential timing circuit using three 74LS122 one-shots.

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 kΩ resistors provides a trigger level of $^1/_3\ V_{CC}$ and a threshold level of $^2/_3\ V_{CC}$. 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 $^1/_3\ V_{CC}$, 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 $^2/_3\ V_{CC}$ 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.

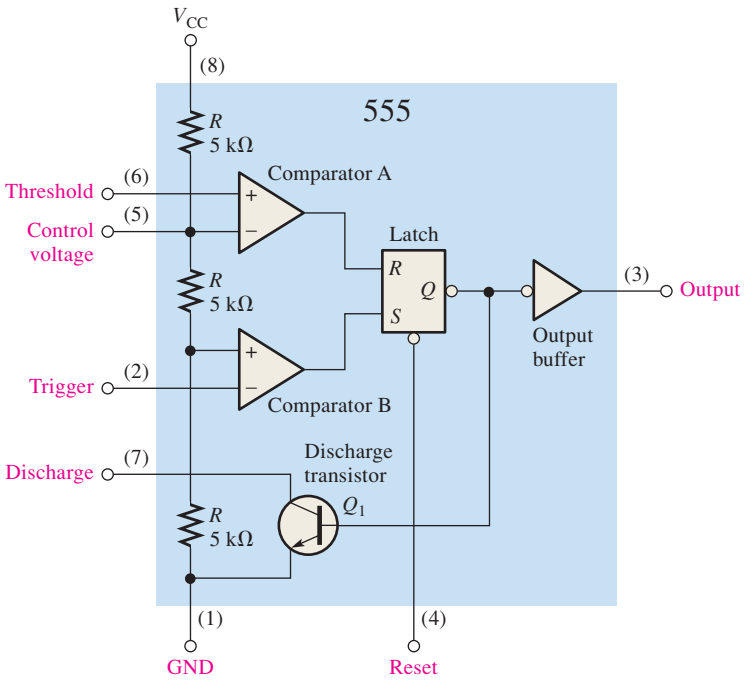


FIGURE 7–52 Internal functional diagram of a 555 timer (pin numbers are in parentheses).

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:

$$t_W = 1.1R_1C_1 \qquad \text{Equation 7–3}$$

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.

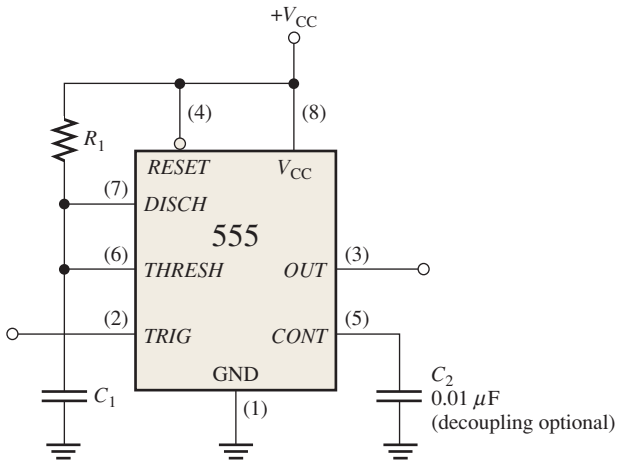
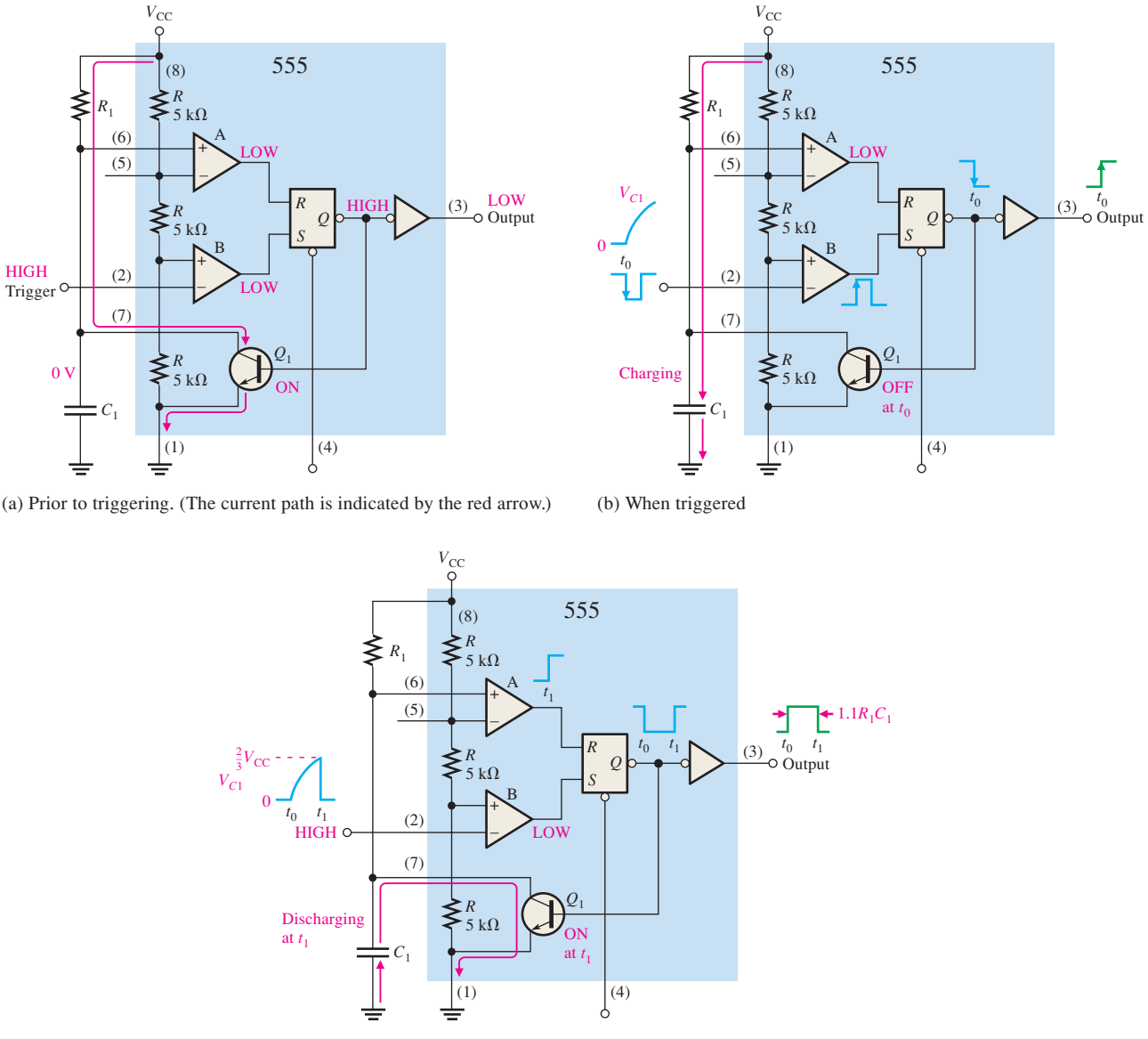


FIGURE 7–53 The 555 timer connected as a one-shot.

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 $^1/_3$ 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.



(a) Prior to triggering. (The current path is indicated by the red arrow.)

(b) When triggered

(c) At end of charging interval

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 $C_1 = 0.01\ \mu\text{F}$?

Solution

From Equation 7–3 the pulse width is

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

Related Problem

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

One-Shot with VHDL

An example of a VHDL program code for a one-shot is as follows:

```
library ieee;
use ieee.std_logic_1164.all;

entity OneShot is
   port (Enable, Clk: in std_logic;
         Duration: in integer range 0 to 25;
         QOut: buffer std_logic);
end entity OneShot;

architecture OneShotBehavior of OneShot is
begin
   Counter: process (Enable, Clk, Duration)
   variable Flag        : boolean := true;
   variable Cnt         : integer range 0 to 25;
   variable SetCount : integer range 0 to 25;
   begin
        SetCount := Duration;
        if (Clk'EVENT and Clk = '1') then
        if Enable = '0' then
              Flag := true;
           end if;

           if Enable = '1' and Flag then
           Cnt := 1;
                Flag :=False;
        end if;

           if cnt = SetCount then
         Qout <= '0';
                Cnt := 0;
                     Flag := false;
        else
                if Cnt > 0 then
                Cnt := Cnt + 1;
                     Qout <= '1';
           end if;
              end if;
        end if;
   end process;
end architecture OneShotBehavior;
```

In normal operation, a one-shot produces only a single pulse, which can be difficult to measure on an oscilloscope because the pulse does not occur regularly. To obtain a stable display for test purposes, it is useful to trigger the one-shot from a pulse generator that is set to a longer period than the expected pulse width and trigger the oscilloscope from the same pulse. For very long pulses, either store the waveform using a digital storage oscilloscope or shorten the time constant by some known factor. For example, replace a 1000 μF capacitor with a 1 μF capacitor to shorten the time by a factor of 1000. A faster pulse is easier to see and measure with an oscilloscope.

SECTION 7–5 CHECKUP

1. Describe the difference between a nonretriggerable and a retriggerable one-shot.
2. How is the output pulse width set in most IC one-shots?
3. What is the pulse width of a 555 timer one-shot when $C = 1\ \mu F$ and $R = 10\ k\Omega$?

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**.

After completing this section, you should be able to

- Describe the operation of a simple astable multivibrator using a Schmitt trigger circuit.
- Set up a 555 timer as an astable multivibrator.

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.

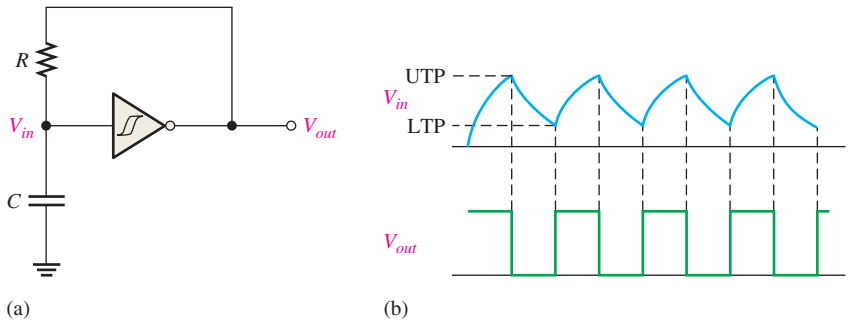


FIGURE 7–55 Basic astable multivibrator using a Schmitt trigger.

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 μF 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.

InfoNote

Most systems require a timing source to provide accurate clock waveforms. The timing section controls all system timing and is responsible for the proper operation of the system hardware. The timing section usually consists of a crystal-controlled oscillator and counters for frequency division. Using a high-frequency oscillator divided down to a lower frequency provides for greater accuracy and frequency stability.

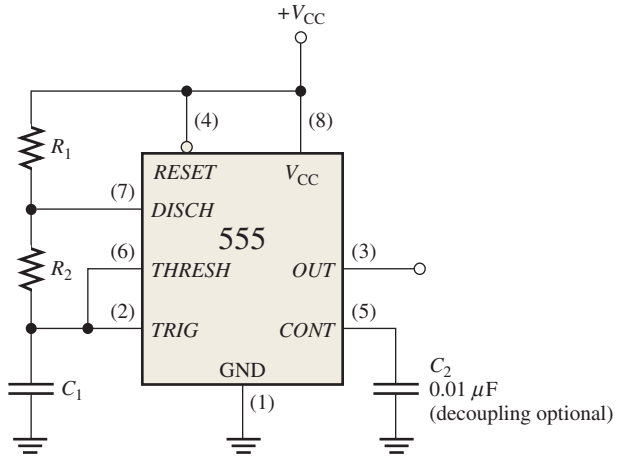


FIGURE 7–56 The 555 timer connected as an astable multivibrator (oscillator).

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 $^1/_3\,V_{CC}$, comparator B switches to its LOW output state; and when the capacitor voltage reaches $^2/_3\,V_{CC}$, 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 $^1/_3\,V_{CC}$, 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

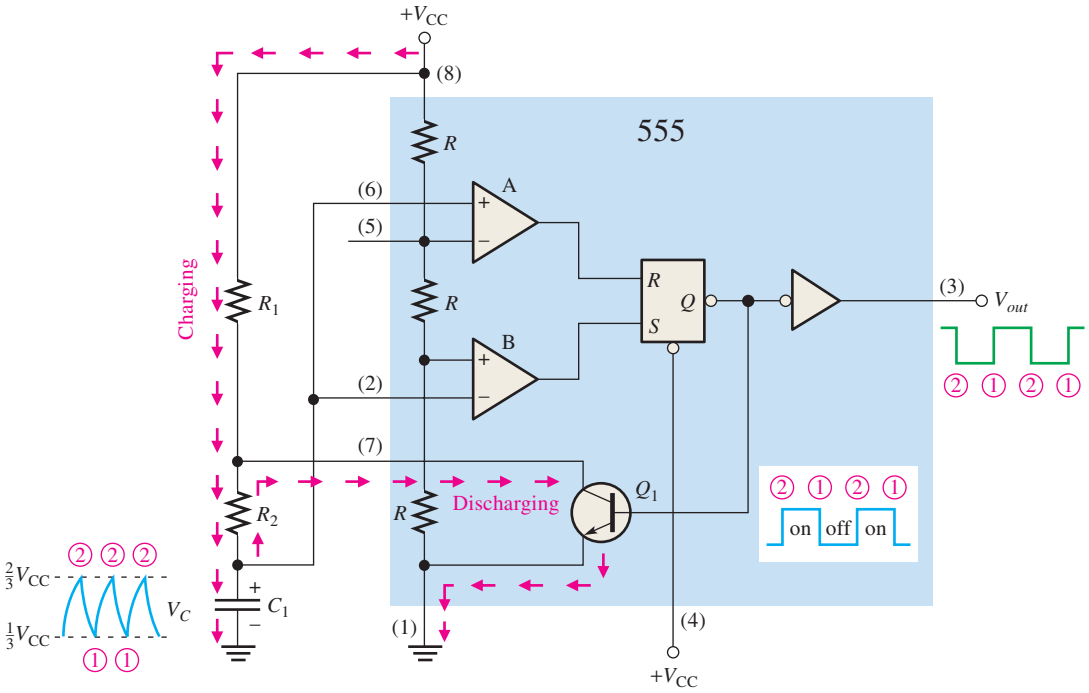


FIGURE 7–57 Operation of the 555 timer in the astable mode.

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.

$$f = \frac{\mathbf{1.44}}{\mathbf{(R_1 + 2R_2)C_1}} \qquad \text{Equation 7–4}$$

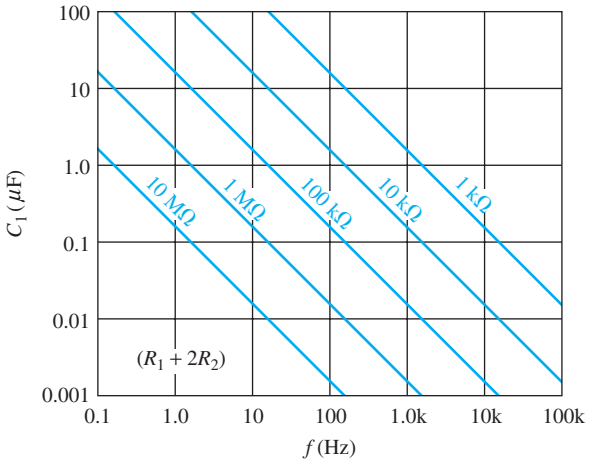


FIGURE 7–58 Frequency of oscillation as a function of C_1 and $R_1 + 2R_2$. The sloped lines are values of $R_1 + 2R_2$.

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 $^1/_3\ V_{CC}$ to $^2/_3\ V_{CC}$. It is expressed as

$$\mathbf{t_H = 0.7(R_1 + R_2)C_1} \qquad \text{Equation 7–5}$$

The time that the output is LOW (t_L) is how long it takes C_1 to discharge from $^1/_3\ V_{CC}$ to $^2/_3\ V_{CC}$. It is expressed as

$$\mathbf{t_L = 0.7R_2C_1} \qquad \text{Equation 7–6}$$

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

$$\text{Duty cycle} = \frac{t_H}{T} = \frac{t_H}{t_H + t_L}$$

$$\mathbf{Duty\ cycle} = \left(\frac{\mathbf{R_1 + R_2}}{\mathbf{R_1 + 2R_2}}\right)\mathbf{100\%} \qquad \text{Equation 7–7}$$

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

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

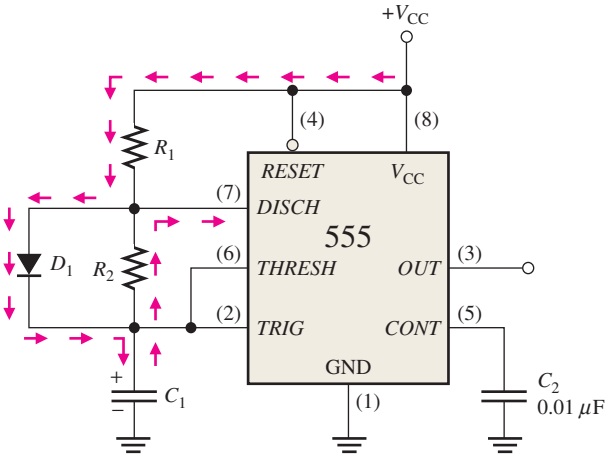
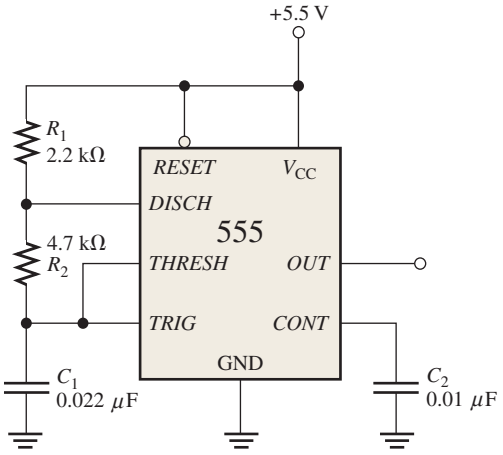


FIGURE 7–59 The addition of diode D_1 allows the duty cycle of the output to be adjusted to less than 50 percent by making $R_1 < R_2$.

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.



MultiSim

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

Solution

Use Equations 7–4 and 7–7.

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

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

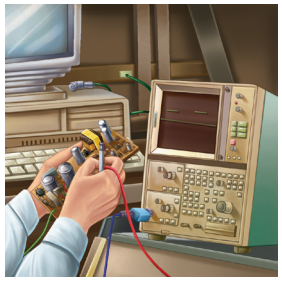
Related Problem

Determine the duty cycle in Figure 7–60 if a diode is connected across R_2 as indicated in Figure 7–59.

SECTION 7–6 CHECKUP

1. Explain the difference in operation between an astable multivibrator and a monostable multivibrator.
2. For a certain astable multivibrator, $t_H = 15$ ms and $T = 20$ ms. What is the duty cycle of the output?

7–7 Troubleshooting



It is standard practice to test a new circuit design to be sure that it is operating as specified. New fixed-function designs are "breadboarded" and tested before the design is finalized. The term *breadboard* refers to a method of temporarily hooking up a circuit so that its operation can be verified and any design flaws worked out before a prototype unit is built.

After completing this section, you should be able to

- Describe how the timing of a circuit can produce erroneous glitches
- Approach the troubleshooting of a new design with greater insight and awareness of potential problems

The circuit shown in Figure 7–61(a) generates two clock waveforms (CLK A and CLK B) that have an alternating occurrence of pulses. Each waveform is to be one-half the frequency of the original clock (CLK), as shown in the ideal timing diagram in part (b).

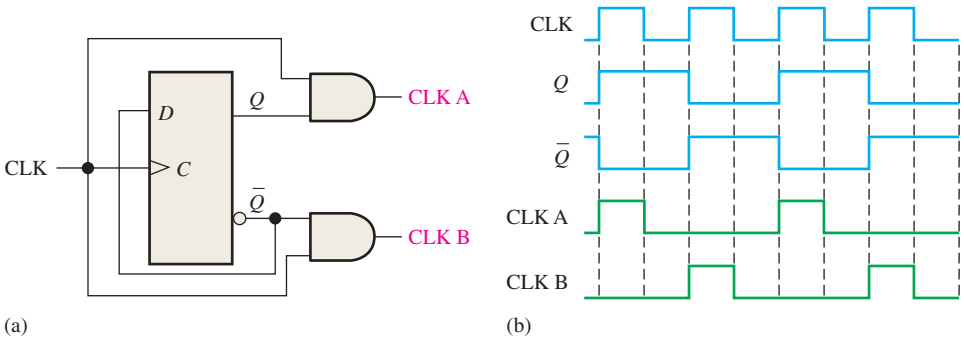
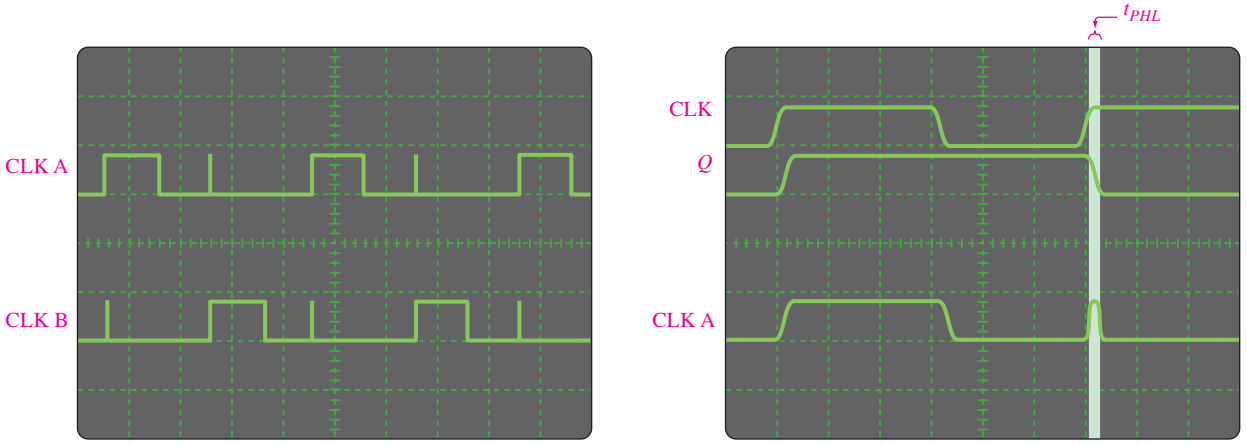


FIGURE 7–61 Two-phase clock generator with ideal waveforms. Open file F07-61 and verify the operation.

MultiSim

When the circuit is tested with an oscilloscope or logic analyzer, the CLK A and CLK B waveforms appear on the display screen as shown in Figure 7–62(a). Since glitches occur on both waveforms, something is wrong with the circuit either in its basic design or in the way it is connected. Further investigation reveals that the glitches are caused by a **race** condition between the CLK signal and the Q and $\overline{Q}$ signals at the inputs of the AND gates. As displayed in Figure 7–62(b), the propagation delays between CLK and Q and $\overline{Q}$ create a short-duration coincidence of HIGH levels at the leading edges of alternate clock pulses. Thus, there is a basic design flaw.

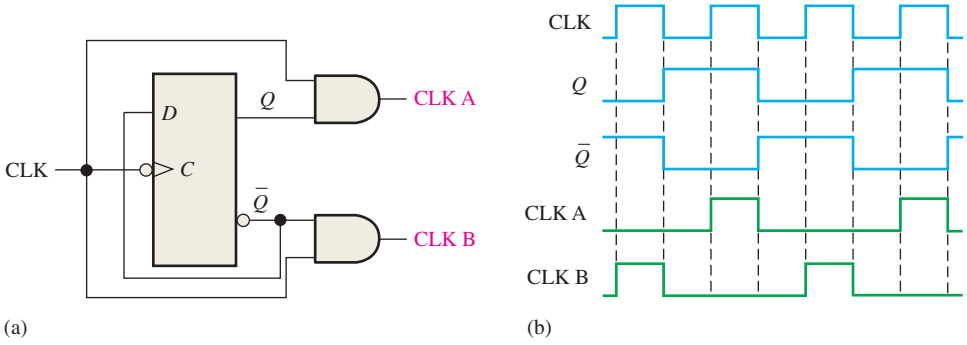
The problem can be corrected by using a negative edge-triggered flip-flop in place of the positive edge-triggered device, as shown in Figure 7–63(a). Although the propagation delays between CLK and Q and $\overline{Q}$ still exist, they are initiated on the trailing edges of the clock (CLK), thus eliminating the glitches, as shown in the timing diagram of Figure 7–63(b).



(a) Oscilloscope display of CLK A and CLK B waveforms with glitches indicated by the "spikes".

(b) Oscilloscope display showing propagation delay that creates glitch on CLK A waveform

FIGURE 7–62 Oscilloscope displays for the circuit in Figure 7–61.



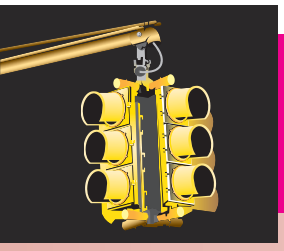
MultiSim

FIGURE 7–63 Two-phase clock generator using negative edge-triggered flip-flop to eliminate glitches. Open file F07-63 and verify the operation.

Glitches that occur in digital systems are very fast (extremely short in duration) and can be difficult to see on an oscilloscope, particularly at lower sweep rates. A logic analyzer, however, can show a glitch easily. To look for glitches using a logic analyzer, select "latch" mode or (if available) transitional sampling. In the latch mode, the analyzer looks for a voltage level change. When a change occurs, even if it is of extremely short duration (a few nanoseconds), the information is "latched" into the analyzer's memory as another sampled data point. When the data are displayed, the glitch will show as an obvious change in the sampled data, making it easy to identify.

SECTION 7–7 CHECKUP

1. Can a negative edge-triggered J-K flip-flop be used in the circuit of Figure 7–63?
2. What device can be used to provide the clock for the circuit in Figure 7–63?



Applied Logic

Traffic Signal Controller: Part 2

The combinational logic unit of the traffic signal controller was completed in Chapter 6. Now, the timing circuits and sequential logic are developed. Recall that the timing circuits produce a 25 s time interval for the red and green lights and a 4 s interval for the yellow caution light. These outputs will be used by the sequential logic. The block diagram of the complete traffic signal controller is shown in Figure 7–64.

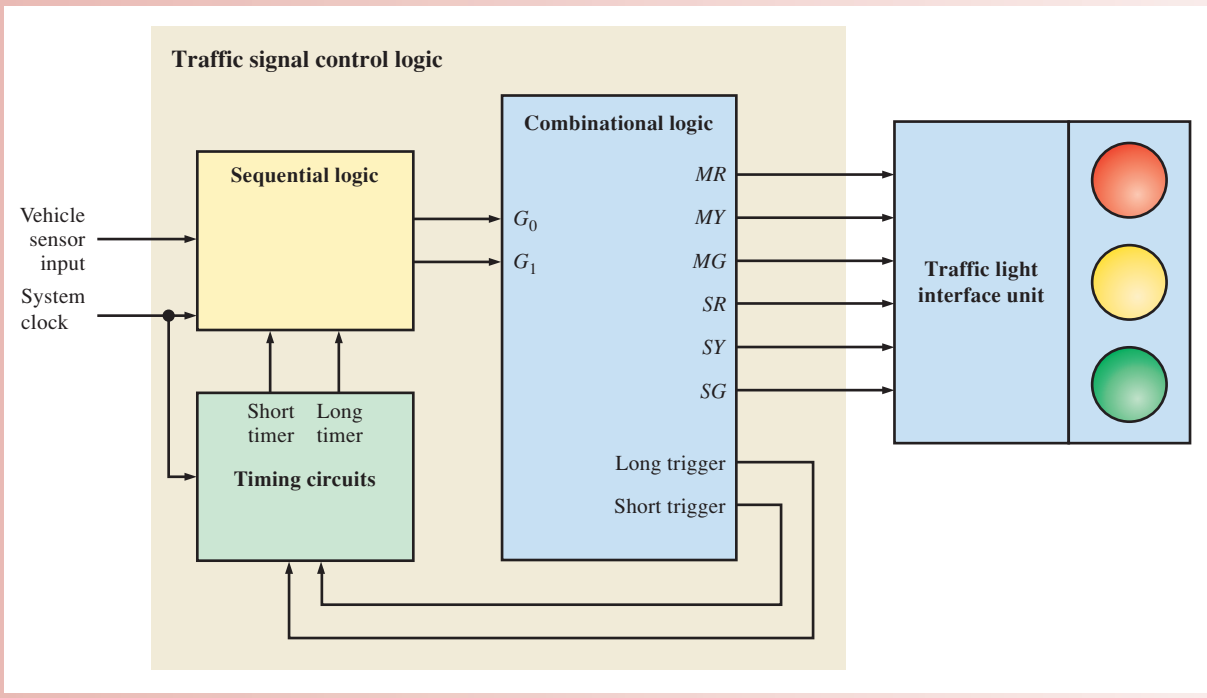


FIGURE 7–64 Block diagram of the traffic signal controller.

Timing Circuits

The timing circuits unit of the traffic signal controller consists of a 25 s timer and a 4 s timer and a clock generator. One way to implement this unit is with two 555 timers configured as one-shots and one 555 timer configured as an astable multivibrator (oscillator), as discussed earlier in this chapter. Component values are calculated based on the formulas given.

Another way to implement the timing circuits is shown in Figure 7–65. An external 24 MHz system clock (arbitrary value) is divided down to an accurate 1 Hz clock by the frequency divider. The 1 Hz clock is then used to establish the 25 s and the 4 s intervals by counting the 1 Hz pulses. This approach lends itself better to a VHDL description.

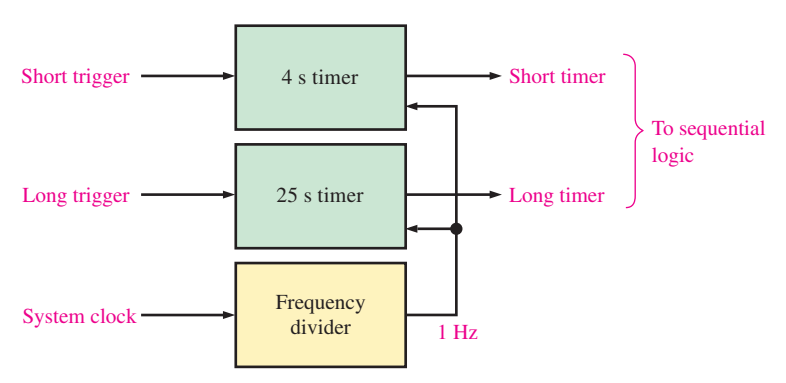


FIGURE 7–65 Block diagram of the timing circuits unit.

Exercise

1. Determine the values for the resistor and capacitor in a 25 s 555 timer.
2. Determine the values for the resistor and capacitor in a 4 s 555 timer.
3. What is the purpose of the frequency divider?

Controller Programming with VHDL

A programming model for the traffic signal controller is shown in Figure 7–66, where all the input and output labels are given. Notice that the Timing circuits block is split into two parts; the Frequency divider and the Timer circuits; and the Combinational logic block is divided into the State decoder and two logic sections (Light output logic and Trigger logic). This model will be used to develop the VHDL program codes.

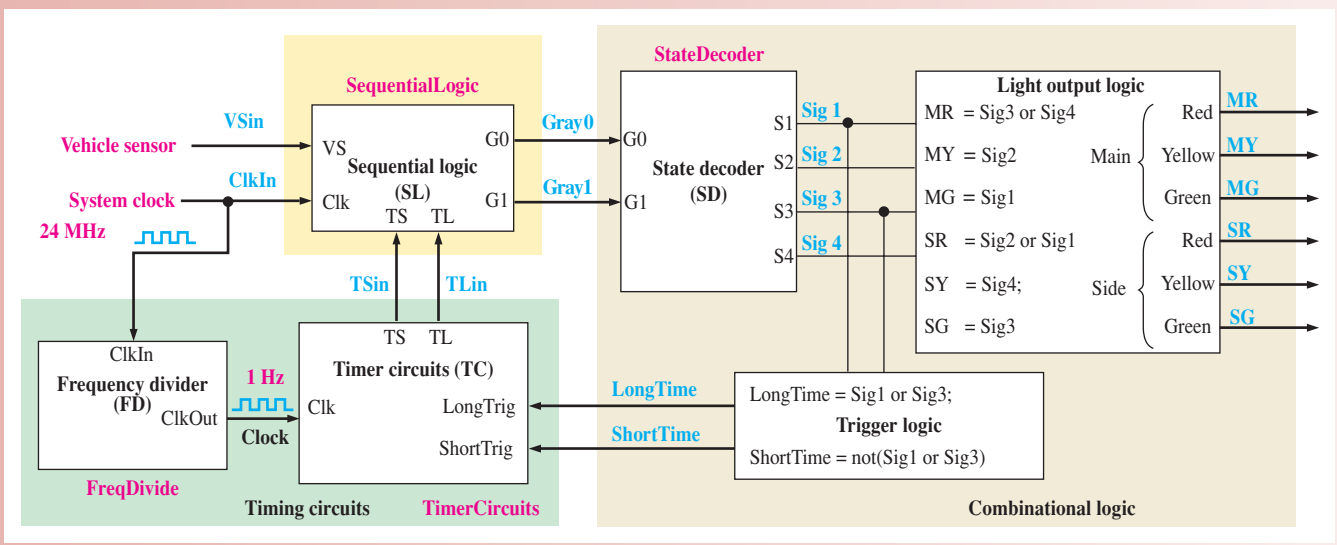


FIGURE 7–66 Programming model for the traffic signal controller.

Frequency Divider The purpose of the frequency divider is to produce a 1 Hz clock for the timer circuits. The input ClkIn in this application is a 24.00 MHz oscillator that drives the program code. SetCount is used to initialize the count for a 1 Hz interval. The program

FreqDivide counts up from zero to the value assigned to SetCount (one-half the oscillator speed) and inverts the output identifier ClkOut.

The integer value Cnt is set to zero prior to operation. The clock pulses are counted and compared to the value assigned to SetCount. When the number of pulses counted reaches the value in SetCount, the output ClkOut is checked to see if it is currently set to a 1 or 0. If ClkOut is currently 0, ClkOut is assigned a 1; otherwise, ClkIn is set to 1. Cnt is assigned a value of 0 and the process repeats. Toggling the output ClkOut each time the value of SetCount is reached creates a 1 Hz clock output with a 50% duty cycle.

The VHDL program code for the frequency divider is as follows:

```
library ieee;                                          Clkln: 24.00 MHz clock driver
use ieee.std_logic_1164.all;                           ClkOut: Output at 1 Hz

entity FreqDivide is
port(Clkln, in std_logic;                              Cnt: Counts up to value in SetCount
     ClkOut: buffer std_logic);                        SetCount: Holds 1/2 timer interval value
end entity FreqDivide;

architecture FreqDivide Behavior of FreqDivide is
begin
  FreqDivide: process(Clkln)
  variable Cnt: integer := 0;
  variable SetCount: integer;

begin
  SetCount := 12000000; -- 1/2 duty cycle              SetCount is assigned a value equal to half the system clock to produce a 1 Hz output. In this case, a 24 MHz system clock is used.
  if (ClkIn'EVENT and ClkIn = '1') then                The if statement causes program to wait for a clock event and clock = 1 to start operation.
    if (Cnt = SetCount) then                           Check that the terminal value in SetCount has been reached at which time ClkOut is toggled and Cnt is reset to 0.
      if ClkOut = '0' then
        ClkOut <= '1'; --Output high 50%
      else
        ClkOut <= '0'; --Output Low 50%
      end if;
      Cnt := 0;
    else
      Cnt := Cnt + 1;                                  If terminal value has not been reached, Cnt is incremented.
    end if;
  end if;
end process;
end architecture FreqDivideBehavior;
```

Timer Circuits The program TimerCircuits uses two one-shot instances consisting of a 25 s timer (TLong) and a 4 s timer (TShort). The 25 s and the 4 s timers are triggered by long trigger (LongTrig) and short trigger (ShortTrig). In the VHDL program, countdown timers driven by a 1 Hz clock input (Clk) replicate the one-shot components TLong and TShort. The values stored in SetCountLong and SetCountShort are assigned to the Duration inputs of one-shot components TLong and TShort, setting the 25-second and 4-second timeouts. When Enable is set LOW, the one-shot timer is initiated and output QOut is set HIGH. When the one-shot timers time out, QOut is set LOW. The output of one-shot component TLong is sent to TimerCircuits identifier TL. The output of one-shot component TShort is sent to TimerCircuits identifier TS.

The VHDL program code for the timing circuits is as follows:

```
library ieee;
use ieee.std_logic_1164.all;

entity TimerCircuits is
   port(LongTrig, ShortTrig, Clk: in std_logic;
        TS, TL: buffer std_logic);
end entity TimerCircuits;

architecture TimerBehavior of TimerCircuits is
component OneShot is
   port(Enable, Clk: in std_logic;
        Duration :in integer range 0 to 25;
        QOut     :buffer std_logic);
end component OneShot;

signal SetCountLong, SetCountShort: integer range 0 to 25;
begin
   SetCountLong <= 25;
   SetCountShort <= 4;
   TLong:OneShot port map(Enable=>LongTrig, Clk=>Clk, Duration=>SetCountLong, QOut=>TL);
   TShort:OneShotport map(Enable=>ShortTrig, Clk=>Clk, Duration=>SetCountShort, QOut=>TS);
end architecture TimerBehavior;
```

LongTrig: Long timeout timer enable input
ShortTrig: Short timeout timer enable input
Clk: 1 Hz Clock input
TS: Short timer timeout signal
TL: Long timer timeout signal

Component declaration for OneShot.

SetCountLong: Holds long timer duration
SetCountShort: Holds short timer duration

Long and short count times are hard-coded to 25 and 4 based on a 1 Hz clock.

← Instantiation TLong

← Instantiation TShort

Sequential Logic

The sequential logic unit controls the sequencing of the traffic lights, based on inputs from the timing circuits and the side street vehicle sensor. The sequential logic produces a 2-bit Gray code sequence for each of the four states that were described in Chapter 6.

The Counter The sequential logic consists of a 2-bit Gray code counter and the associated input logic, as shown in Figure 7–67. The counter produces the four-state sequence on outputs G_0 and G_1. Transitions from one state to the next are determined by the short timer (T_S), the long timer (T_L), and vehicle sensor (V_s) inputs.

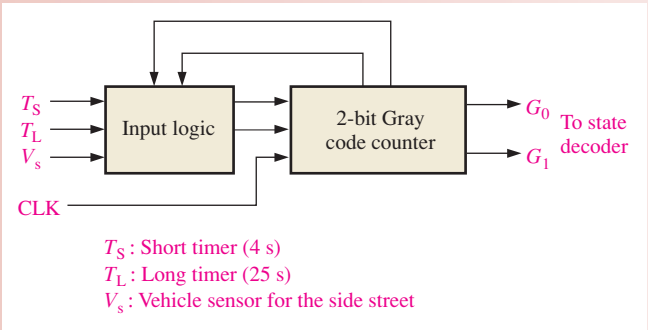


FIGURE 7–67 Block diagram of the sequential logic.

The diagram in Figure 7–68 shows how two D flip-flops can be used to implement the Gray code counter. Outputs from the input logic provide the *D* inputs to the flip-flops so they sequence through the proper states.

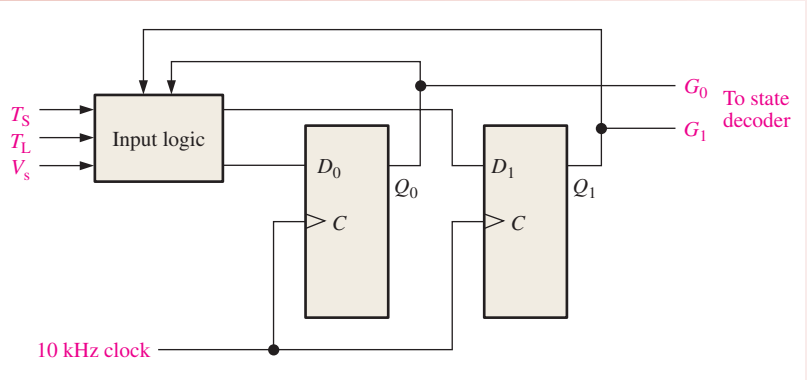


FIGURE 7–68 Sequential logic diagram with two *D* flip-flops used to implement the 2-bit Gray code counter.

The D flip-flop transition table is shown in Table 7–5. A next-state table developed from the state diagram in Chapter 6 Applied Logic is shown in Table 7–6. The subject of counter design is covered further in Chapter 8.

TABLE 7–5

D flip-flop transition table. Q_N is the output before clock pulse. Q_{N+1} is output after clock pulse.

Output Transitions			Flip-Flop Input
Q_N		Q_{N+1}	D
0	⟶	0	0
0	⟶	1	1
1	⟶	0	0
1	⟶	1	1

TABLE 7–6

Next-state table for the counter.

Present State		Next State			FF Inputs	
Q_1	Q_0	Q_1	Q_0	Input Conditions	D_1	D_0
0	0	0	0	$T_L + \overline{V}_s$	0	0
0	0	0	1	$\overline{T}_L V_s$	0	1
0	1	0	1	T_S	0	1
0	1	1	1	$\overline{T}_S$	1	1
1	1	1	1	$T_L V_s$	1	1
1	1	1	0	$\overline{T}_L + \overline{V}_s$	1	0
1	0	1	0	T_S	1	0
1	0	0	0	$\overline{T}_S$	0	0

The Input Logic Using Tables 7–5 and 7–6, the conditions required for each flip-flop to go to the 1 state can be determined. For example, G_0 goes from 0 to 1 when the present state is 00 and the condition on input D_0 is $\overline{T}_L V_s$, as indicated on the second row of Table 7–6. D_0 must be a 1 to make G_0 go to a 1 or to remain a 1 on the next clock pulse. A Boolean expression describing the conditions that make D_0 a 1 is derived from Table 7–6 as follows:

$$D_0 = \overline{G}_1\overline{G}_0\overline{T}_L V_s + \overline{G}_1 G_0 T_S + \overline{G}_1 G_0 \overline{T}_S + G_1 G_0 T_L V_s$$

In the two middle terms, the T_S and the $\overline{T}_S$ variables cancel, leaving the expression

$$D_0 = \overline{G}_1\overline{G}_0\overline{T}_L V_s + \overline{G}_1 G_0 + G_1 G_0 T_L V_s$$

Also, from Table 7–6, an expression for D_1 can be developed as follows:

$$D_1 = \overline{G}_1 G_0 \overline{T}_S + G_1 G_0 T_L V_s + G_1 G_0 \overline{T}_L + G_1 G_0 \overline{V}_s + G_1 \overline{G}_0 T_S$$

Based on the minimized expression for D_0 and D_1, the complete sequential logic diagram is shown in Figure 7–69.

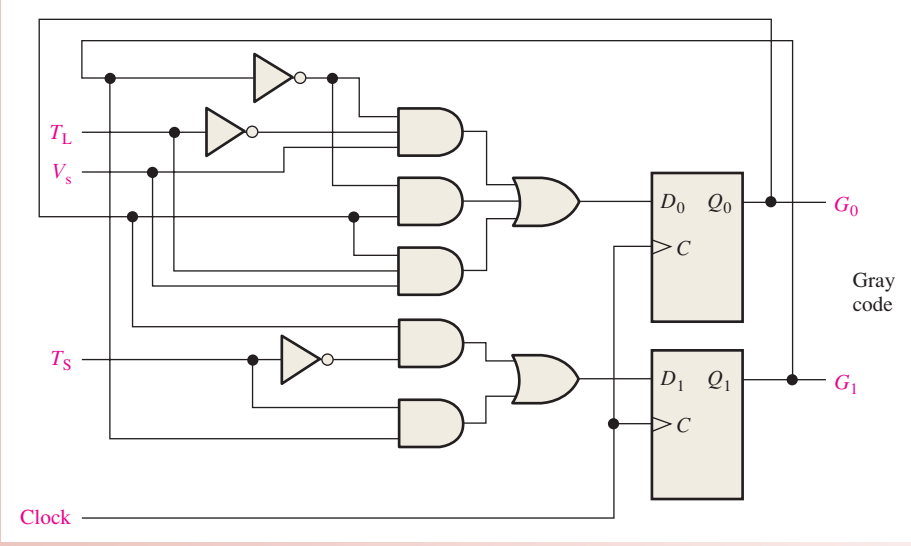


FIGURE 7–69 Complete diagram for the sequential logic.

Exercise

4. State the Boolean law and rule that permits the cancellation of T_S and $\overline{T}_S$ in the expression for D_0.
5. Use the Karnaugh map to reduce the D_0 expression further to a minimum form.
6. Use Boolean laws, rules, and/or the Karnaugh map to reduce the D_1 expression to a minimum form.
7. Do your minimized expressions for D_0 and D_1 agree with the logic shown in Figure 7–69?

The Sequential Logic with VHDL

The program SequentialLogic describes the Gray code logic needed to drive the traffic signal controller based on input from the timing circuits and the side street vehicle sensor. The sequential logic code produces a 2-bit Gray code sequence for each of the

four sequence states. The component definition dff is used to instantiate two D flip-flop instances DFF0 and DFF1. DFF0 and DFF1 produce the two-bit Gray code. The Gray code output sequences the traffic signal controller through each of four states. Internal variables D0 and D1 store the results of the D0 and D1 Boolean expressions developed in this chapter. The stored results in D0 and D1 are assigned to D flip-flops DFF0 and DFF1 along with the system clock to drive outputs G0 and G1 from the D flip-flop *Q* outputs.

The VHDL program code for the sequential logic is as follows:

```
library ieee;
use ieee.std_logic_1164.all;

entity SequentialLogic is
port(VS, TL, TS, Clk: in std_logic; G0, G1: inout std_logic);
end entity SequentialLogic;

architecture SequenceBehavior of SequentialLogic is

component dff is
port (D, Clk: in std_logic; Q: out std_logic);
end component dff;

signal D0, D1: std_logic;
begin
D1 <= (G0 and not TS) or (G1 and TS);
D0 <= (not G1 and not TL and VS) or (not G1 and G0)
      or (G0 and TL and VS);

DFF0: dff port map(D=> D0, Clk => Clk, Q => G0);
DFF1: dff port map(D=> D1, Clk => Clk, Q => G1);

end architecture SequenceBehavior;
```

VS: Vehicle sensor input
TL: Long timer input
TS: Short timer input
Clk: System clock
G0: Gray code output bit 0
G1: Gray code output bit 1
D0: Logic for DFlipFlop DFF0
D1: Logic for DFlipFlop DFF1

Component declaration for D flip-flop (dff)

Logic definitions for D flip-flop inputs D0 and D1 derived from Boolean expressions developed in this chapter.

Component instantiations

The Complete Traffic Signal Controller

The program TrafficLights completes the traffic signal controller. Components FreqDivide, TimerCircuits, SequentialLogic, and StateDecoder are used to compose the completed system. Signal CLKin from the TrafficLights program source code is the clock input to the FreqDivide component. The frequency divided output ClkOut is stored as local variable Clock and is the divided clock input to the TimerCircuits and SequentialLogic components. TimerCircuits is controlled by local variables LongTime and ShortTime, which are controlled by the outputs Sig1 and Sig3 from component StateDecoder. StateDecoder also provides outputs Sig1 through Sig4 to control the traffic lights MG, SG, MY, SY, MR, and SR. TimerCircuit timeout signals TS and TL are stored in variables TLin (timer long in) and TSin (timer short in).

Signals TSin and TLin from TimerCircuits are used along with vehicle sensor VSin as inputs to the SequentialLogic component. The outputs from SequentialLogic G0 and G1 are stored in variables Gray0 and Gray1 as inputs to component StateDecoder. Component StateDecoder returns signals S1 through S4 which are in turn passed to variables Sig1 through Sig4. The light output logic and trigger logic developed in Chapter 6 are not used as components in this program, but are stated as logic expressions. The values stored in variables Sig1 through Sig4 provide the logic for outputs MG, SG, MY, SY, MR, SR; and local timer triggers LongTime and ShortTime are sent to TimerCircuits.

The VHDL program code for the traffic signal controller is as follows:

```
library ieee;
use ieee.std_logic_1164.all;

entity TrafficLights is
port(VSin, ClkIn: in std_logic; MR, SR, MY, SY, MG, SG: out std_logic);
end entity TrafficLights;

architecture TrafficLightsBehavior of TrafficLights is

component StateDecoder is
port(G0, G1: in std_logic; S1, S2, S3, S4: out std_logic);
end component StateDecoder;

component SequentialLogic is
port(VS, TL, TS, Clk: in std_logic; G0, G1: inout std_logic);
end component SequentialLogic;

component TimerCircuits is
port(LongTrig, ShortTrig, Clk: In std_logic; TS, TL: buffer std_logic);
end component TimerCircuits;

component FreqDivide is
port(Clkin: in std_logic; ClkOut: buffer std_logic);
end component FreqDivide;

signal Sig1, Sig2, Sig3, Sig4, Gray0, Gray1: std_logic;
signal LongTime, ShortTime, TLin, TSin, Clock: std_logic;

begin
MR <= Sig3 or Sig4;
SR <= Sig2 or Sig1;
MY <= Sig2;
SY <= Sig4;
MG <= Sig1;
SG <= Sig3;

LongTime <= Sig1 or Sig3;
ShortTime <= not(Sig1 or Sig3);

SD: StateDecoder     port map (G0 => Gray0, G1 => Gray1, S1 => Sig1, S2 => Sig2, S3 => Sig3, S4 => Sig4);
SL: SequentialLogic  port map (VS => VSin, TL => TLin, TS => TSin, Clk => Fout, G0 => Gray0, G1 => Gray1);
TC: TimerCircuits    port map (LongTrig=>LongTime, ShortTrig=>ShortTime, Clk=>Clock, TS=>TSin, TL=>TLin);
FD: FreqDivide       port map (Clkln => CLKin, ClkOut =>-Clock);

end architecture TrafficLightsBehavior;
```

VSin : Vehicle sensor input
CLKin : System Clock
MR : Main red light output
SR : Side red light output
MY : Main yellow light output
SY : Side yellow light output
MG : Main green light output
SG : Side green light output

Component declaration for StateDecoder

Component declaration for SequentialLogic

Component declaration for TimerCircuits

Component declaration for FreqDivider

Sig1-4 : Return values from StateDecoder
Gray0-1 : SequentialLogic Gray code return
LongTime : Trigger input to TimerCircuits
ShortTime : Trigger input to TimerCircuits
TLin : Store TimerCircuits long timeout
TSin : Store TimerCircuits Short timeout
Clock : Divided clock from FreqDivide

Logic definitions for the light output logic

Logic definitions for the trigger logic

Component instantiations

Simulation

MultiSim

Open file AL07 in the Applied Logic folder on the website. Run the traffic signal controller simulation using your Multisim software and observe the operation. Lights will appear randomly when first turned on. Simulation times may vary.

Putting Your Knowledge to Work

Add your modification for the pedestrian input developed in Chapter 6 and run a simulation.

SUMMARY

- Latches are bistable devices whose state normally depends on asynchronous inputs.
- Edge-triggered flip-flops are bistable devices with synchronous inputs whose state depends on the inputs only at the triggering transition of a clock pulse. Changes in the outputs occur at the triggering transition of the clock.
- Monostable multivibrators (one-shots) have one stable state. When the one-shot is triggered, the output goes to its unstable state for a time determined by an *RC* circuit.
- Astable multivibrators have no stable states and are used as oscillators to generate timing waveforms in digital systems.

KEY TERMS

Key terms and other bold terms in the chapter are defined in the end-of-book glossary.

Astable Having no stable state. An astable multivibrator oscillates between two quasi-stable states.

Bistable Having two stable states. Flip-flops and latches are bistable multivibrators.

Clear An asynchronous input used to reset a flip-flop (make the *Q* output 0).

Clock The triggering input of a flip-flop.

D flip-flop A type of bistable multivibrator in which the output assumes the state of the *D* input on the triggering edge of a clock pulse.

Edge-triggered flip-flop A type of flip-flop in which the data are entered and appear on the output on the same clock edge.

Hold time The time interval required for the control levels to remain on the inputs to a flip-flop after the triggering edge of the clock in order to reliably activate the device.

J-K flip-flop A type of flip-flop that can operate in the SET, RESET, no-change, and toggle modes.

Latch A bistable digital circuit used for storing a bit.

Monostable Having only one stable state. A monostable multivibrator, commonly called a *one-shot,* produces a single pulse in response to a triggering input.

One-shot A monostable multivibrator.

Power dissipation The amount of power required by a circuit.

Preset An asynchronous input used to set a flip-flop (make the *Q* output 1).

Propagation delay time The interval of time required after an input signal has been applied for the resulting output change to occur.

RESET The state of a flip-flop or latch when the output is 0; the action of producing a RESET state.

SET The state of a flip-flop or latch when the output is 1; the action of producing a SET state.

Set-up time The time interval required for the control levels to be on the inputs to a digital circuit, such as a flip-flop, prior to the triggering edge of a clock pulse.

Synchronous Having a fixed time relationship.

Timer A circuit that can be used as a one-shot or as an oscillator.

Toggle The action of a flip-flop when it changes state on each clock pulse.

TRUE/FALSE QUIZ

Answers are at the end of the chapter.

1. A latch has one stable state.

2. A latch is considered to be in the RESET state when the *Q* output is low.

3. A gated D latch cannot be used to change state.
4. Flip-flops and latches are both bistable devices.
5. An edge-triggered D flip-flop changes state whenever the *D* input changes.
6. A clock input is necessary for an edge-triggered flip-flop.
7. When both the *J* and *K* inputs are HIGH, an edge-triggered J-K flip-flop changes state on each clock pulse.
8. A one-shot is also known as an astable multivibrator.
9. When triggered, a one-shot produces a single pulse.
10. The 555 timer cannot be used as a pulse oscillator.

SELF-TEST

Answers are at the end of the chapter.

1. An active HIGH input S-R latch is formed by the cross-coupling of
   (a) two NOR gates **(b)** two NAND gates **(c)** two OR gates **(d)** two AND gates
2. Which of the following is not true for an active LOW input $\overline{S}$–$\overline{R}$ latch?
   (a) $\overline{S} = 1, \overline{R} = 1, Q = NC, \overline{Q} = NC$ **(b)** $\overline{S} = 0, \overline{R} = 1, Q = 1, \overline{Q} = 0$
   (c) $\overline{S} = 1, \overline{R} = 0, Q = 1, \overline{Q} = 0$ **(d)** $\overline{S} = 0, \overline{R} = 0, Q = 1, \overline{Q} = 1$
3. For what combinations of the inputs *D* and *EN* will a *D* latch reset?
   (a) D = LOW, *EN* = LOW
   (b) *D* = LOW, *EN* = HIGH
   (c) *D* = HIGH, *EN* = LOW
   (d) *D* = HIGH, *EN* = HIGH
4. A flip-flop changes its state during the
   (a) complete operational cycle
   (b) falling edge of the clock pulse
   (c) rising edge of the clock pulse
   (d) both answers (b) and (c)
5. The purpose of the clock input to a flip-flop is to
   (a) clear the device
   (b) set the device
   (c) always cause the output to change states
   (d) cause the output to assume a state dependent on the controlling (*J*-*K* or *D*) inputs.
6. For an edge-triggered D flip-flop,
   (a) a change in the state of the flip-flop can occur only at a clock pulse edge
   (b) the state that the flip-flop goes to depends on the *D* input
   (c) the output follows the input at each clock pulse
   (d) all of these answers
7. A feature that distinguishes the J-K flip-flop from the D flip-flop is the
   (a) toggle condition **(b)** preset input
   (c) type of clock **(d)** clear input
8. A flip-flop is SET when
   (a) $J = 0, K = 0$ **(b)** $J = 0, K = 1$
   (c) $J = 1, K = 0$ **(d)** $J = 1, K = 1$
9. A J-K flip-flop with $J = 1$ and $K = 1$ has a 10 kHz clock input. The Q output is
   (a) constantly HIGH **(b)** constantly LOW
   (c) a 10 kHz square wave **(d)** a 5 kHz square wave
10. A one-shot is a type of
    (a) monostable multivibrator **(b)** astable multivibrator
    (c) timer **(d)** answers (a) and (c)
    (e) answers (b) and (c)

11. The output pulse width of a nonretriggerable one-shot depends on
(a) the trigger intervals **(b)** the supply voltage
(c) a resistor and capacitor **(d)** the threshold voltage

12. An astable multivibrator
(a) requires a periodic trigger input **(b)** has no stable state
(c) is an oscillator **(d)** produces a periodic pulse output
(e) answers (a), (b), (c), and (d) **(f)** answers (b), (c), and (d) only

PROBLEMS

Answers to odd-numbered problems are at the end of the book.

Section 7–1 Latches

1. If the waveforms in Figure 7–70 are applied to an active-HIGH S-R latch, draw the resulting Q output waveform in relation to the inputs. Assume that Q starts LOW.

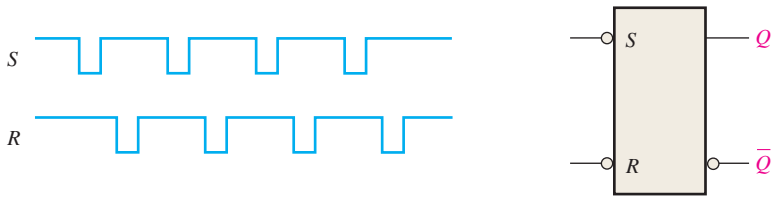


FIGURE 7–70

2. Solve Problem 1 for the input waveforms in Figure 7–71 applied to an active-LOW $\overline{S} - \overline{R}$ latch.

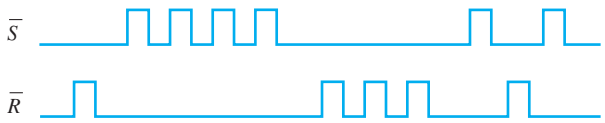


FIGURE 7–71

3. Solve Problem 1 for the input waveform in Figure 7–72.

FIGURE 7–72

4. For a gated S-R latch, determine the Q and $\overline{Q}$ outputs for the inputs in Figure 7–73. Show them in proper relation to the enable input. Assume that Q starts LOW.

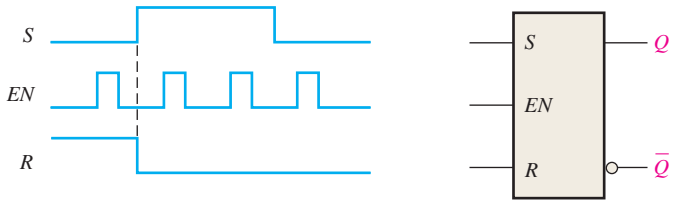


FIGURE 7–73

5. Determine the output of a gated D latch for the inputs in Figure 7–74.

FIGURE 7–74

6. Determine the output of a gated D latch for the inputs in Figure 7–75.

FIGURE 7–75

7. For a gated D latch, the waveforms shown in Figure 7–76 are observed on its inputs. Draw the timing diagram showing the output waveform you would expect to see at *Q* if the latch is initially RESET.

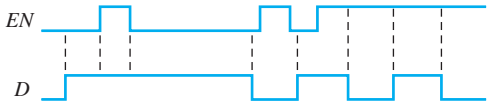


FIGURE 7–76

Section 7–2 Flip-Flops

8. Two edge-triggered J-K flip-flops are shown in Figure 7–77. If the inputs are as shown, draw the *Q* output of each flip-flop relative to the clock, and explain the difference between the two. The flip-flops are initially RESET.

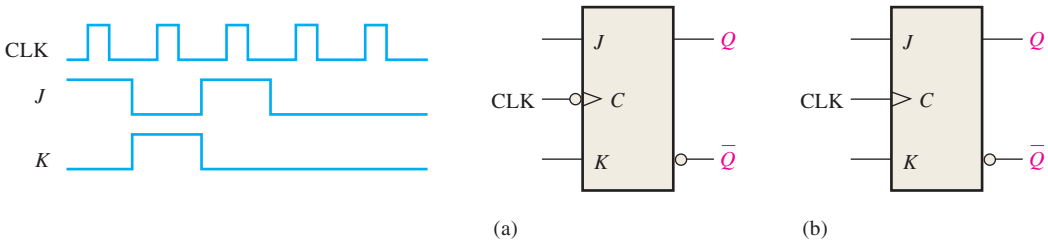


FIGURE 7–77

9. The *Q* output of an edge-triggered D flip-flop is shown in relation to the clock signal in Figure 7–78. Determine the input waveform on the D input that is required to produce this output if the flip-flop is a positive edge-triggered type.

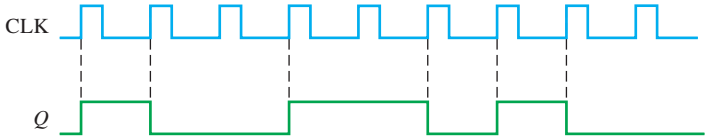


FIGURE 7–78

10. Draw the *Q* output relative to the clock for a D flip-flop with the inputs as shown in Figure 7–79. Assume positive edge-triggering and *Q* initially LOW.

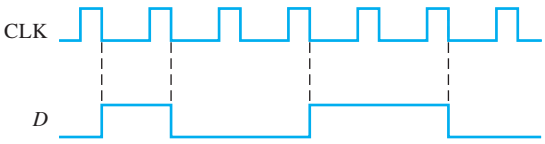


FIGURE 7–79

11. Solve Problem 10 for the inputs in Figure 7–80.

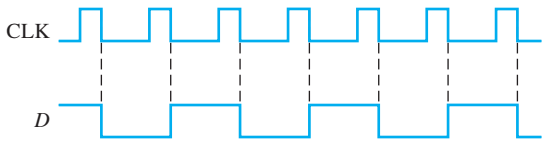


FIGURE 7–80

12. For a positive edge-triggered D flip-flop with the input as shown in Figure 7–81, determine the *Q* output relative to the clock. Assume that *Q* starts LOW.

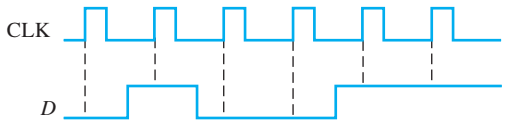


FIGURE 7–81

13. Solve Problem 12 for the input in Figure 7–82.

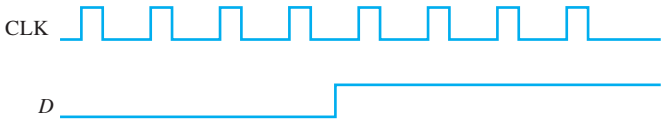


FIGURE 7–82

14. Determine the *Q* waveform relative to the clock if the signals shown in Figure 7–83 are applied to the inputs of the J-K flip-flop. Assume that *Q* is initially LOW.

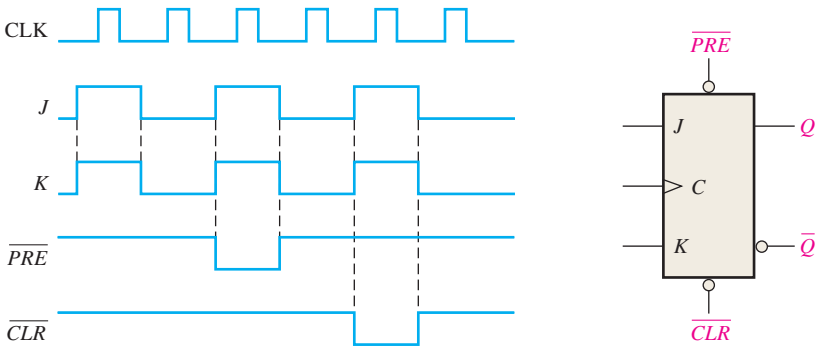


FIGURE 7–83

15. For a negative edge-triggered J-K flip-flop with the inputs in Figure 7–84, develop the *Q* output waveform relative to the clock. Assume that *Q* is initially LOW.

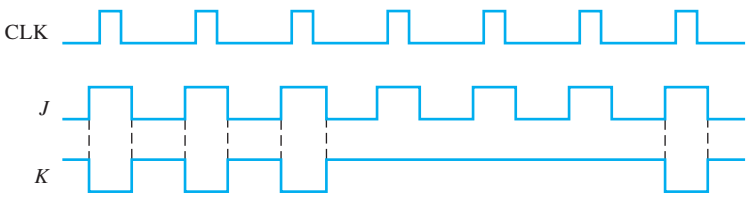


FIGURE 7–84

16. The following serial data are applied to the flip-flop through the AND gates as indicated in Figure 7–85. Determine the resulting serial data that appear on the Q output. There is one clock pulse for each bit time. Assume that Q is initially 0 and that $\overline{PRE}$ and $\overline{CLR}$ are HIGH. Rightmost bits are applied first.

J_1: 1 0 1 0 0 1 1; J_2: 0 1 1 1 0 1 0; J_3: 1 1 1 1 0 0 0; K_1: 0 0 0 1 1 1 0; K_2: 1 1 0 1 1 0 0; K_3: 1 0 1 0 1 0 1

17. For the circuit in Figure 7–85, complete the timing diagram in Figure 7–86 by showing the Q output (which is initially LOW). Assume $\overline{PRE}$ and $\overline{CLR}$ remain HIGH.

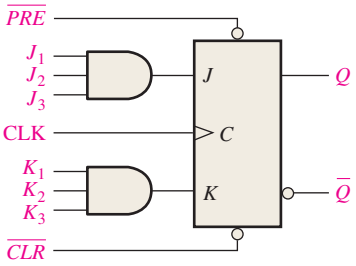


FIGURE 7–85

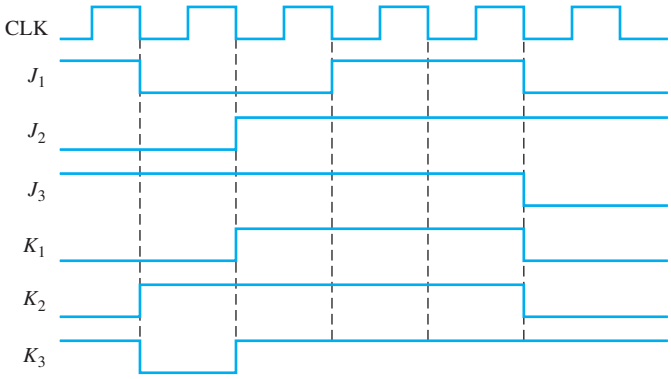


FIGURE 7–86

18. Solve Problem 17 with the same J and K inputs but with the $\overline{PRE}$ and $\overline{CLR}$ inputs as shown in Figure 7–87 in relation to the clock.

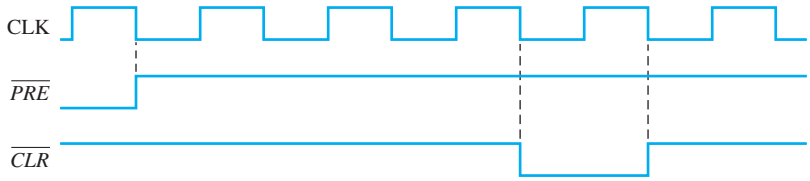


FIGURE 7–87

Section 7–3 Flip-Flop Operating Characteristics

19. What determines the power dissipation of a flip-flop?

20. Typically, a flip-flop is limited in its operation due to hold time and setup time. Explain how.

21. The datasheet of a certain flip-flop specified that the minimum HIGH time for the clock pulse is 20 ns and the minimum LOW time is 40 ns. What is the maximum operating frequency?

22. The flip-flop in Figure 7–88 is initially RESET. Show the relation between Q output and the clock pulse if the propagation delay t_{PLH} (clock to Q) is 5 ns.

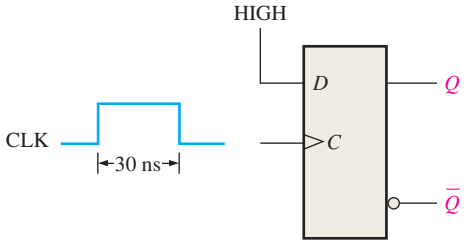


FIGURE 7–88

23. The direct current required by a particular flip-flop that operates on a +4 V dc source is found to be 8 mA. A certain digital device uses 16 of these flip-flops. Determine the current capacity required for the +4 V dc supply and the total power dissipation of the system.

24. For the circuit in Figure 7–89, determine the maximum frequency of the clock signal for reliable operation if the set-up time for each flip-flop is 3 ns and the propagation delays (t_{PLH} and t_{PHL}) from clock to output are 6 ns for each flip-flop.

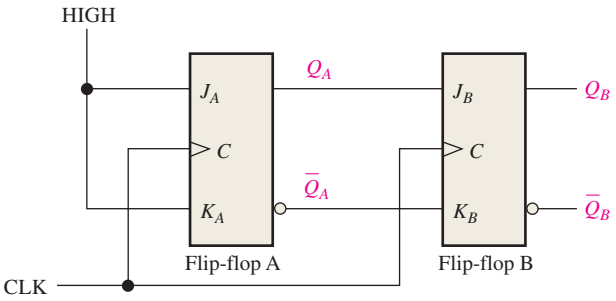


FIGURE 7–89

Section 7–4 Flip-Flop Applications

25. A D flip-flop is connected as shown in Figure 7–90. Determine the Q output in relation to the clock. What specific function does this device perform?

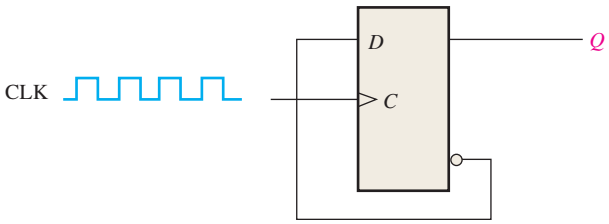


FIGURE 7–90

26. For the circuit in Figure 7–89, develop a timing diagram for eight clock pulses, showing the Q_A and Q_B outputs in relation to the clock.

Section 7–5 One-Shots

27. Determine the pulse width of a 74121 one-shot if the external resistor is 1 kΩ and the external capacitor is 1 pF.

28. An output pulse of 3 μs duration is to be generated by a 74LS122 one-shot. Using a capacitor of 50,000 pF, determine the value of external resistance required.

29. Create a one-shot using a 555 timer that will produce a 0.5 s output pulse.

Section 7–6 The Astable Multivibrator

30. A 555 timer is configured to run as an astable multivibrator as shown in Figure 7–91. Determine its frequency.

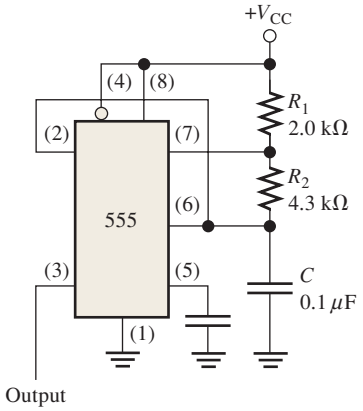


FIGURE 7–91

31. Determine the values of the external resistors for a 555 timer used as an astable multivibrator with an output frequency of 10 kHz, if the external capacitor C is 0.004 μF and the duty cycle is to be approximately 80%.

Section 7–7 Troubleshooting

32. The flip-flop in Figure 7–92 is tested under all input conditions as shown. Is it operating properly? If not, what is the most likely fault?

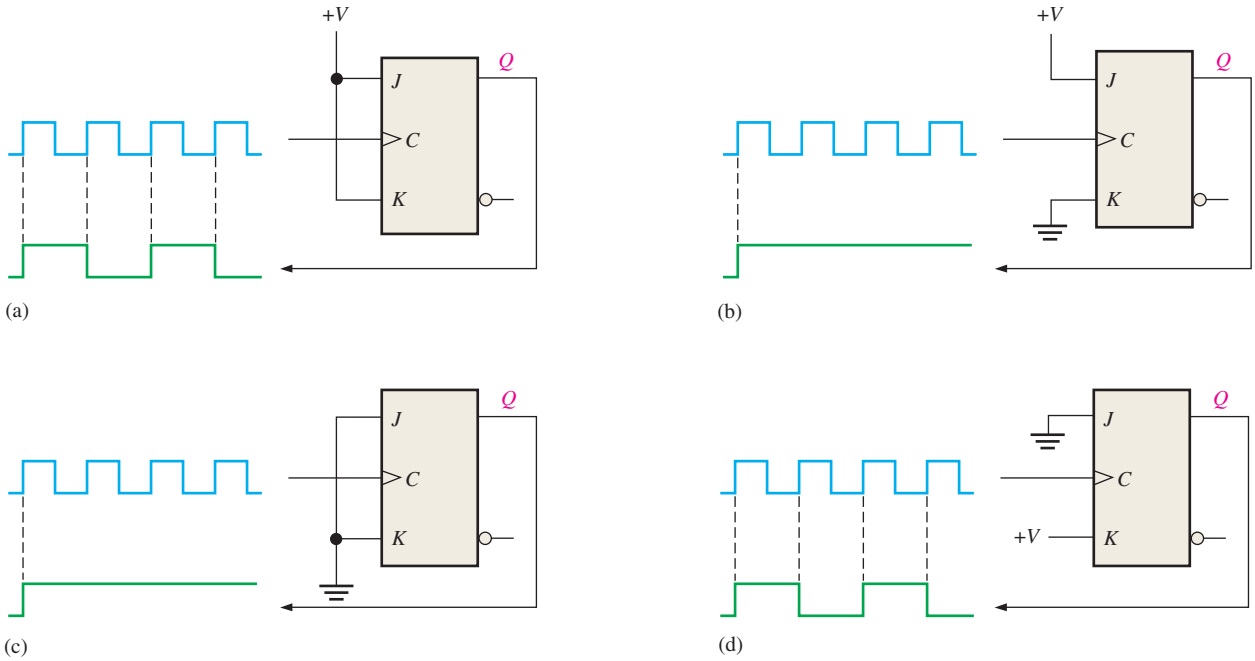


FIGURE 7–92

33. A 74HC00 quad NAND gate IC is used to construct a gated S-R latch on a protoboard in the lab as shown in Figure 7–93. The schematic in part (a) is used to connect the circuit in part (b). When you try to operate the latch, you find that the Q output stays HIGH no matter what the inputs are. Determine the problem.

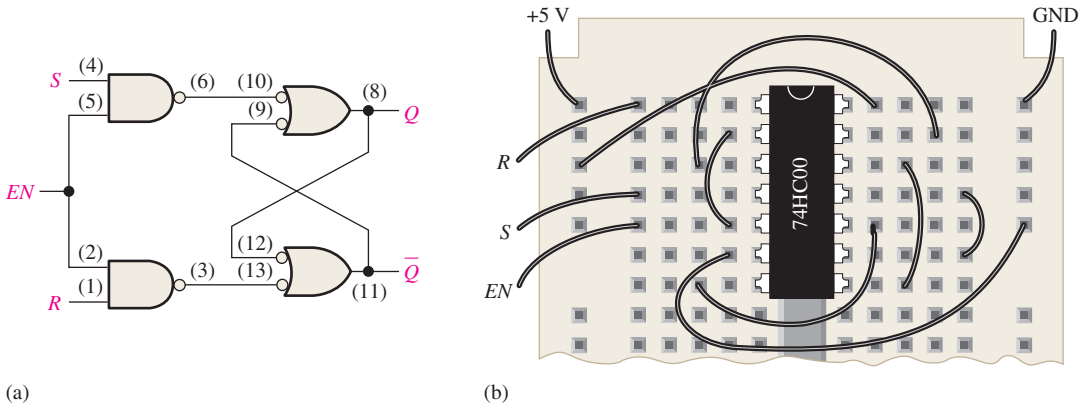


FIGURE 7–93

34. Determine if the flip-flop in Figure 7–94 is operating properly, and if not, identify the most probable fault.

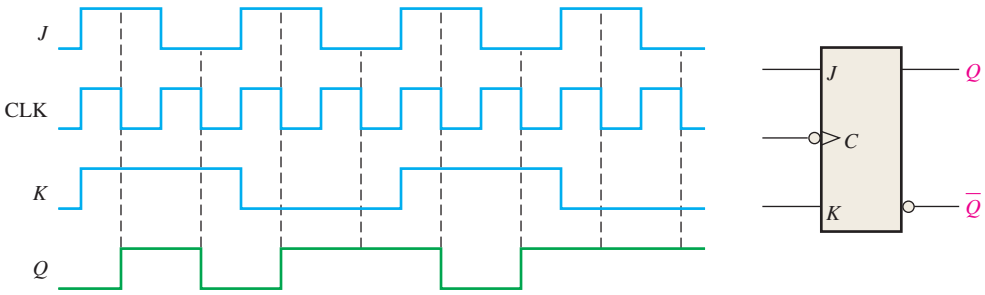
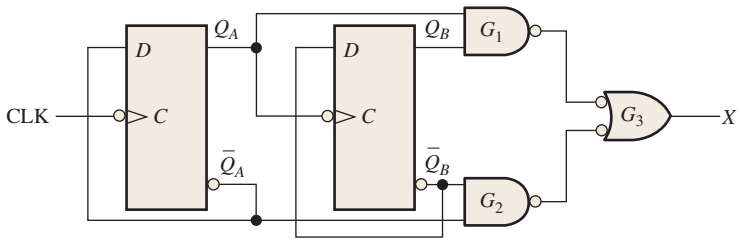


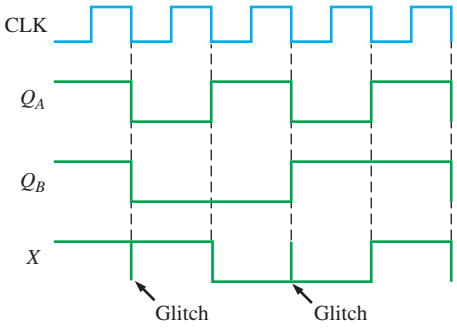
FIGURE 7–94

35. The parallel data storage circuit in Figure 7–35 does not operate properly. To check it out, you first make sure that V_{CC} and ground are connected, and then you apply LOW levels to all the D inputs and pulse the clock line. You check the Q outputs and find them all to be LOW; so far, so good. Next you apply HIGHs to all the D inputs and again pulse the clock line. When you check the Q outputs, they are still all LOW. What is the problem, and what procedure will you use to isolate the fault to a single device?

36. The flip-flop circuit in Figure 7–95(a) is used to generate a binary count sequence. The gates form a decoder that is supposed to produce a HIGH when a binary zero or a binary three state occurs (00 or 11). When you check the Q_A and Q_B outputs, you get the display shown in part (b), which reveals glitches on the decoder output (X) in addition to the correct pulses. What is causing these glitches, and how can you eliminate them?



(a)



(b)

FIGURE 7–95

37. Determine the Q_A, Q_B and X outputs over six clock pulses in Figure 7–95(a) for each of the following faults in the bipolar (TTL) circuits. Start with both Q_A and Q_B LOW.

(a) D input open
(b) Q_B output open
(c) clock input to flip-flop B shorted
(d) gate G_2 output open

38. Two 74121 one-shots are connected on a circuit board as shown in Figure 7–96. After observing the oscilloscope display, do you conclude that the circuit is operating properly? If not, what is the most likely problem?

Applied Logic

39. Using 555 timers, redesign the timing circuits portion of the traffic signal controller for an approximate 5 s caution light and 30 s red and green lights.

40. Repeat Problem 39 using 74121 one-shots.

41. Repeat Problem 39 using 74122 one-shots.

42. Implement the input logic in the sequential circuit unit of the traffic signal controller using only NAND gates.

43. Specify how you would change the time interval for the green light from 25 s to 60 s.

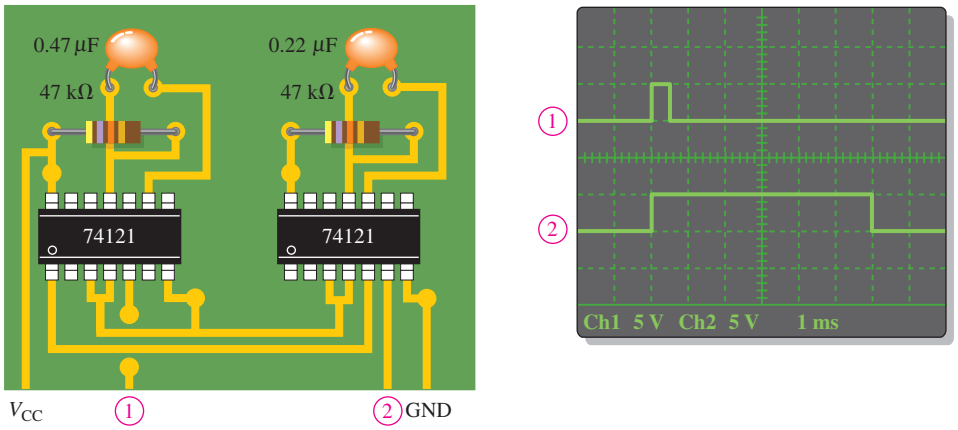


FIGURE 7–96

Special Design Problems

44. Design a basic counting circuit that produces a binary sequence from zero through seven by using negative edge-triggered J-K flip-flops.

45. In the shipping department of a softball factory, the balls roll down a conveyor and through a chute single file into boxes for shipment. Each ball passing through the chute activates a switch circuit that produces an electrical pulse. The capacity of each box is 32 balls. Design a logic circuit to indicate when a box is full so that an empty box can be moved into position.

46. List the design changes that would be necessary in the traffic signal controller to add a 15 s left turn arrow for the main street. The turn arrow will occur after the red light and prior to the green light. Modify the state diagram from Chapter 6 to show these changes.

Multisim Troubleshooting Practice

47. Open file P07-47. For the specified fault, predict the effect on the circuit. Then introduce the fault and verify whether your prediction is correct.

48. Open file P07-48. For the specified fault, predict the effect on the circuit. Then introduce the fault and verify whether your prediction is correct.

49. Open file P07-49. For the observed behavior indicated, predict the fault in the circuit. Then introduce the suspected fault and verify whether your prediction is correct.

50. Open file P07-50. For the observed behavior indicated, predict the fault in the circuit. Then introduce the suspected fault and verify whether your prediction is correct.

51. Open file P07-51. For the observed behavior indicated, predict the fault in the circuit. Then introduce the suspected fault and verify whether your prediction is correct.

ANSWERS

SECTION CHECKUPS

Section 7–1 Latches

1. Three types of latches are S-R, gated S-R, and gated D.

2. $SR = 00$, NC; $SR = 01$, $Q = 0$; $SR = 10$, $Q = 1$; $SR = 11$, invalid

3. $Q = 1$

Section 7–2 Flip-Flops

1. The output of a gated D latch can change any time the gate enable (*EN*) input is active. The output of an edge-triggered D flip-flop can change only on the triggering edge of a clock pulse.

2. The output of a J-K flip-flop is determined by the state of its two inputs whereas the output of a D flip-flop follows the input.

3. Output Q goes HIGH on the trailing edge of the first clock pulse, LOW on the trailing edge of the second pulse, HIGH on the trailing edge of the third pulse, and LOW on the trailing edge of the fourth pulse.

Section 7–3 Flip-Flop Operating Characteristics

1. **(a)** Set-up time is the time required for input data to be present before the triggering edge of the clock pulse.
   (b) Hold time is the time required for data to remain on the inputs after the triggering edge of the clock pulse.
2. The 74AHC74 can be operated at the highest frequency, according to Table 7–4.

Section 7–4 Flip-Flop Applications

1. A group of data storage flip-flops is a register.
2. For divide-by-2 operation, the flip-flop must toggle ($D = \overline{Q}$).
3. Six flip-flops are used in a divide-by-64 device.

Section 7–5 One-Shots

1. A nonretriggerable one-shot times out before it can respond to another trigger input. A retriggerable one-shot responds to each trigger input.
2. Pulse width is set with external R and C components.
3. 11 ms.

Section 7–6 The Astable Multivibrator

1. An astable multivibrator has no stable state. A monostable multivibrator has one stable state.
2. Duty cycle $=$ (15 ms/20 ms)100% $=$ 75%

Section 7–7 Troubleshooting

1. Yes, a negative edge-triggered J-K flip-flop can be used.
2. An astable multivibrator using a 555 timer can be used to provide the clock.

RELATED PROBLEMS FOR EXAMPLES

7–1 The Q output is the same as shown in Figure 7–5(b).

7–2 See Figure 7–97.

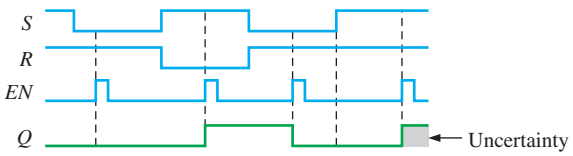


FIGURE 7–97

7–3 See Figure 7–98.

7–4 See Figure 7–99.

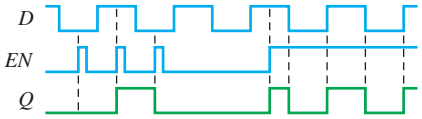


FIGURE 7–98

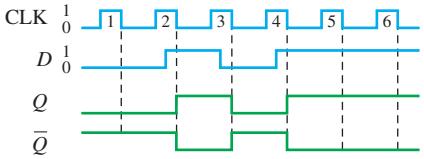


FIGURE 7–99

7–5 See Figure 7–100.

7–6 See Figure 7–101.

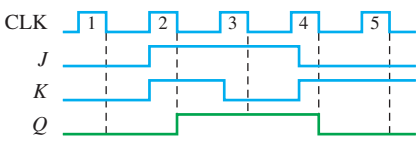


FIGURE 7–100

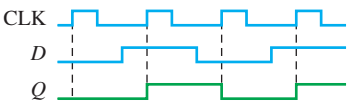


FIGURE 7–101

7–7 See Figure 7–102.

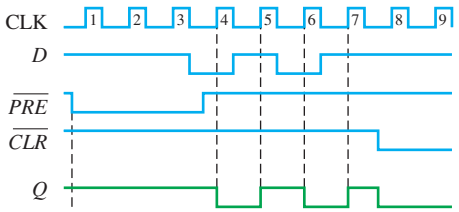


FIGURE 7–102

7–8 See Figure 7–103.

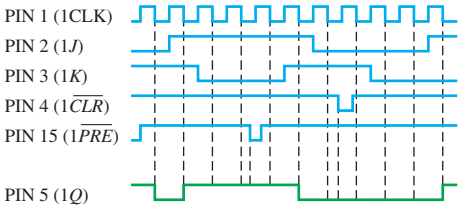


FIGURE 7–103

7–9 $2^5 = 32$. Five flip-flops are required.

7–10 Sixteen states require four flip-flops ($2^4 = 16$).

7–11 $C_{EXT} = 7143$ pF connected from *CX* to *RX/CX* of the 74121 with no external resistor.

7–12 $C_{EXT} = 560$ pF, $R_{EXT} = 27$ kΩ. See Figure 7–104.

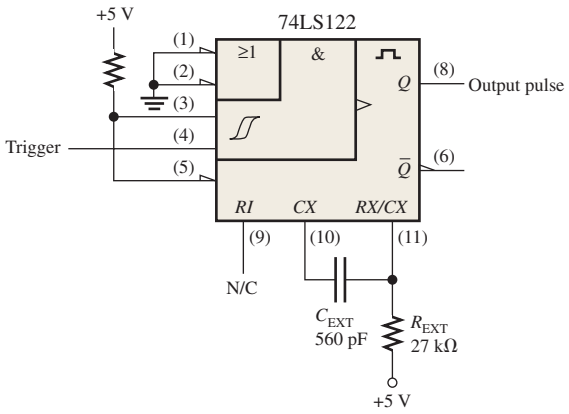


FIGURE 7–104

7–13 $R_1 = 91$ kΩ

7–14 Duty cycle ≅ 32%

TRUE/FALSE QUIZ

1. F **2.** T **3.** F **4.** T **5.** F **6.** T **7.** T **8.** F **9.** T **10.** F

SELF-TEST

1. (a) **2.** (c) **3.** (b) **4.** (d) **5.** (d) **6.** (d)
7. (a) **8.** (c) **9.** (d) **10.** (d) **11.** (c) **12.** (f)

CHAPTER 8

Shift Registers

CHAPTER OUTLINE

CHAPTER OBJECTIVES

- Identify the basic forms of data movement in shift registers
- Explain how serial in/serial out, serial in/parallel out, parallel in/serial out, and parallel in/parallel out shift registers operate
- Describe how a bidirectional shift register operates
- Determine the sequence of a Johnson counter
- Set up a ring counter to produce a specified sequence
- Construct a ring counter from a shift register
- Use a shift register as a time-delay device
- Use a shift register to implement a serial-to-parallel data converter
- Implement a basic shift-register-controlled keyboard encoder
- Interpret ANSI/IEEE Standard 91-1984 shift register symbols with dependency notation
- Use shift registers in a system application

KEY TERMS

Key terms are in order of appearance in the chapter.

- Register
- Stage
- Load
- Bidirectional

VISIT THE WEBSITE

Study aids for this chapter are available at http://www.pearsonglobaleditions.com/floyd

INTRODUCTION

Shift registers are a type of sequential logic circuit used primarily for the storage of digital data and typically do not possess a characteristic internal sequence of states. There are exceptions, however, and these are covered in Section 8–4.

In this chapter, the basic types of shift registers are studied and several applications are presented. Also, a troubleshooting method is introduced.

8–1 Shift Register Operations

Shift registers consist of arrangements of flip-flops and are important in applications involving the storage and transfer of data in a digital system. A register has no specified sequence of states, except in certain very specialized applications. A register, in general, is used solely for storing and shifting data (1s and 0s) entered into it from an external source and typically possesses no characteristic internal sequence of states.

After completing this section, you should be able to

- Explain how a flip-flop stores a data bit
- Define the storage capacity of a shift register
- Describe the shift capability of a register

A register can consist of one or more flip-flops used to store and shift data.

A **register** is a digital circuit with two basic functions: data storage and data movement. The storage capability of a register makes it an important type of memory device. Figure 8–1 illustrates the concept of storing a 1 or a 0 in a D flip-flop. A 1 is applied to the data input as shown, and a clock pulse is applied that stores the 1 by *setting* the flip-flop. When the 1 on the input is removed, the flip-flop remains in the SET state, thereby storing the 1. A similar procedure applies to the storage of a 0 by *resetting* the flip-flop, as also illustrated in Figure 8–1.

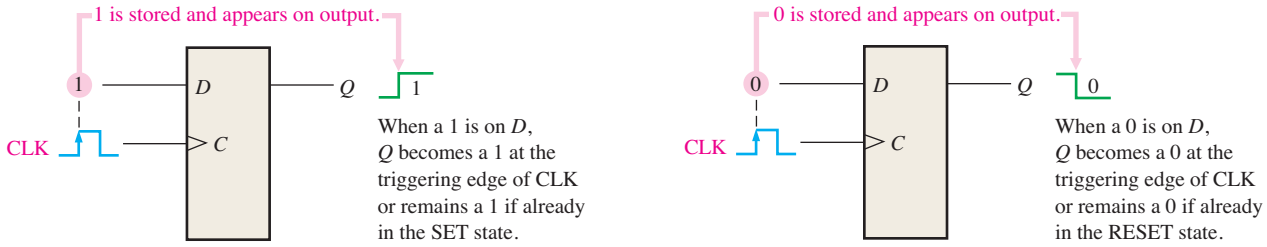


FIGURE 8–1 The flip-flop as a storage element.

The *storage capacity* of a register is the total number of bits (1s and 0s) of digital data it can retain. Each **stage** (flip-flop) in a shift register represents one bit of storage capacity; therefore, the number of stages in a register determines its storage capacity.

The *shift capability* of a register permits the movement of data from stage to stage within the register or into or out of the register upon application of clock pulses. Figure 8–2

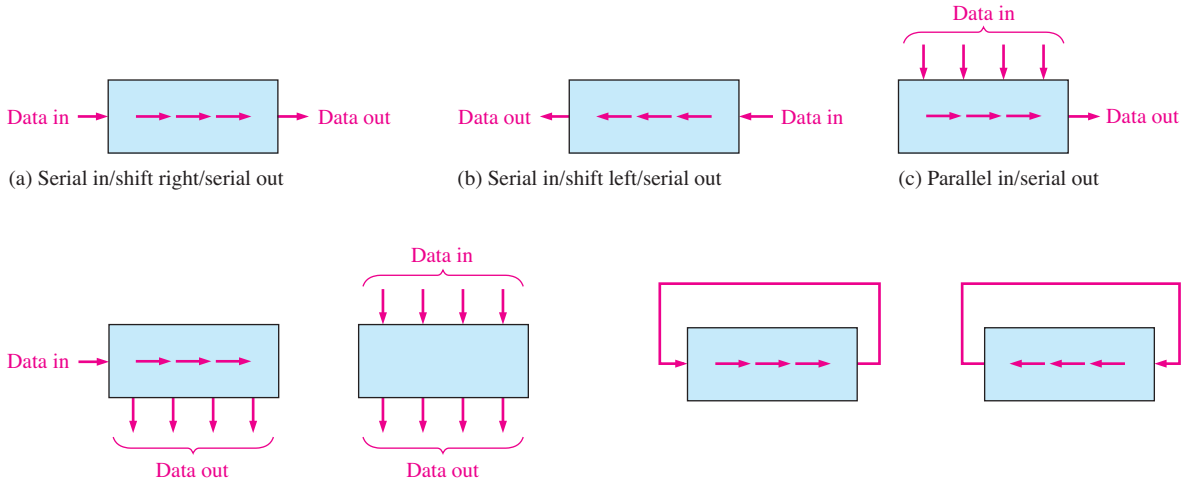


FIGURE 8–2 Basic data movement in shift registers. (Four bits are used for illustration. The bits move in the direction of the arrows.)

illustrates the types of data movement in shift registers. The block represents any arbitrary 4-bit register, and the arrows indicate the direction of data movement.

SECTION 8–1 CHECKUP

Answers are at the end of the chapter.

1. What determines the storage capacity of a shift register?
2. What two principal functions are performed by a shift register?

8–2 Types of Shift Register Data I/Os

In this section, four types of shift registers based on data input and output (inputs/outputs) are discussed: serial in/serial out, serial in/parallel out, parallel in/serial out, and parallel in/parallel out.

After completing this section, you should be able to

- Describe the operation of four types of shift registers
- Explain how data bits are entered into a shift register
- Describe how data bits are shifted through a register
- Explain how data bits are taken out of a shift register
- Develop and analyze timing diagrams for shift registers

Serial In/Serial Out Shift Registers

The serial in/serial out shift register accepts data serially—that is, one bit at a time on a single line. It produces the stored information on its output also in serial form. Let's first look at the serial entry of data into a typical shift register. Figure 8–3 shows a 4-bit device implemented with D flip-flops. With four stages, this register can store up to four bits of data.

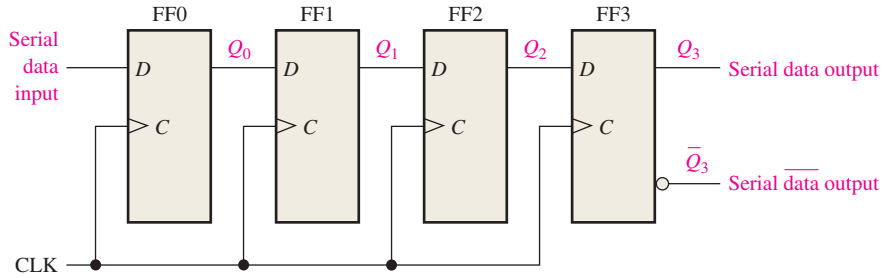


FIGURE 8–3 Serial in/serial out shift register.

InfoNote

Frequently, it is necessary to *clear* an internal register in a processor. For example, a register may be cleared prior to an arithmetic or other operation. One way that registers in a processor are cleared is using software to subtract the contents of the register from itself. The result, of course, will always be zero. For example, a processor instruction that performs this operation is SUB AL,AL. with this instruction, the register named AL is cleared.

Table 8–1 shows the entry of the four bits 1010 into the register in Figure 8–3, beginning with the least significant bit. The register is initially clear. The 0 is put onto the data input line, making $D = 0$ for FF0. When the first clock pulse is applied, FF0 is reset, thus storing the 0.

TABLE 8–1

Shifting a 4-bit code into the shift register in Figure 8–3. Data bits are indicated by a beige screen.

CLK	FF0 (Q_0)	FF1 (Q_1)	FF2 (Q_2)	FF3 (Q_3)
Initial	0	0	0	0
1	**0**	0	0	0
2	**1**	**0**	0	0
3	**0**	**1**	**0**	0
4	**1**	**0**	**1**	**0**

Next the second bit, which is a 1, is applied to the data input, making $D = 1$ for FF0 and $D = 0$ for FF1 because the D input of FF1 is connected to the Q_0 output. When the second clock pulse occurs, the 1 on the data input is shifted into FF0, causing FF0 to set; and the 0 that was in FF0 is shifted into FF1.

The third bit, a 0, is now put onto the data-input line, and a clock pulse is applied. The 0 is entered into FF0, the 1 stored in FF0 is shifted into FF1, and the 0 stored in FF1 is shifted into FF2.

For serial data, one bit at a time is transferred.

The last bit, a 1, is now applied to the data input, and a clock pulse is applied. This time the 1 is entered into FF0, the 0 stored in FF0 is shifted into FF1, the 1 stored in FF1 is shifted into FF2, and the 0 stored in FF2 is shifted into FF3. This completes the serial entry of the four bits into the shift register, where they can be stored for any length of time as long as the flip-flops have dc power.

If you want to get the data out of the register, the bits must be shifted out serially to the Q_3 output, as Table 8–2 illustrates. After CLK4 in the data-entry operation just described, the LSB, 0, appears on the Q_3 output. When clock pulse CLK5 is applied, the second bit appears on the Q_3 output. Clock pulse CLK6 shifts the third bit to the output, and CLK7 shifts the fourth bit to the output. While the original four bits are being shifted out, more bits can be shifted in. All zeros are shown being shifted in, after CLK8.

TABLE 8–2

Shifting a 4-bit code out of the shift register in Figure 8–3. Data bits are indicated by a beige screen.

CLK	FF0 (Q_0)	FF1 (Q_1)	FF2 (Q_2)	FF3 (Q_3)
Initial	**1**	**0**	**1**	**0**
5	0	**1**	**0**	**1**
6	0	0	**1**	**0**
7	0	0	0	**1**
8	0	0	0	0

EXAMPLE 8–1

Show the states of the 5-bit register in Figure 8–4(a) for the specified data input and clock waveforms. Assume that the register is initially cleared (all 0s).

Solution

The first data bit (1) is entered into the register on the first clock pulse and then shifted from left to right as the remaining bits are entered and shifted. The register contains $Q_4Q_3Q_2Q_1Q_0 = 11010$ after five clock pulses. See Figure 8–4(b).

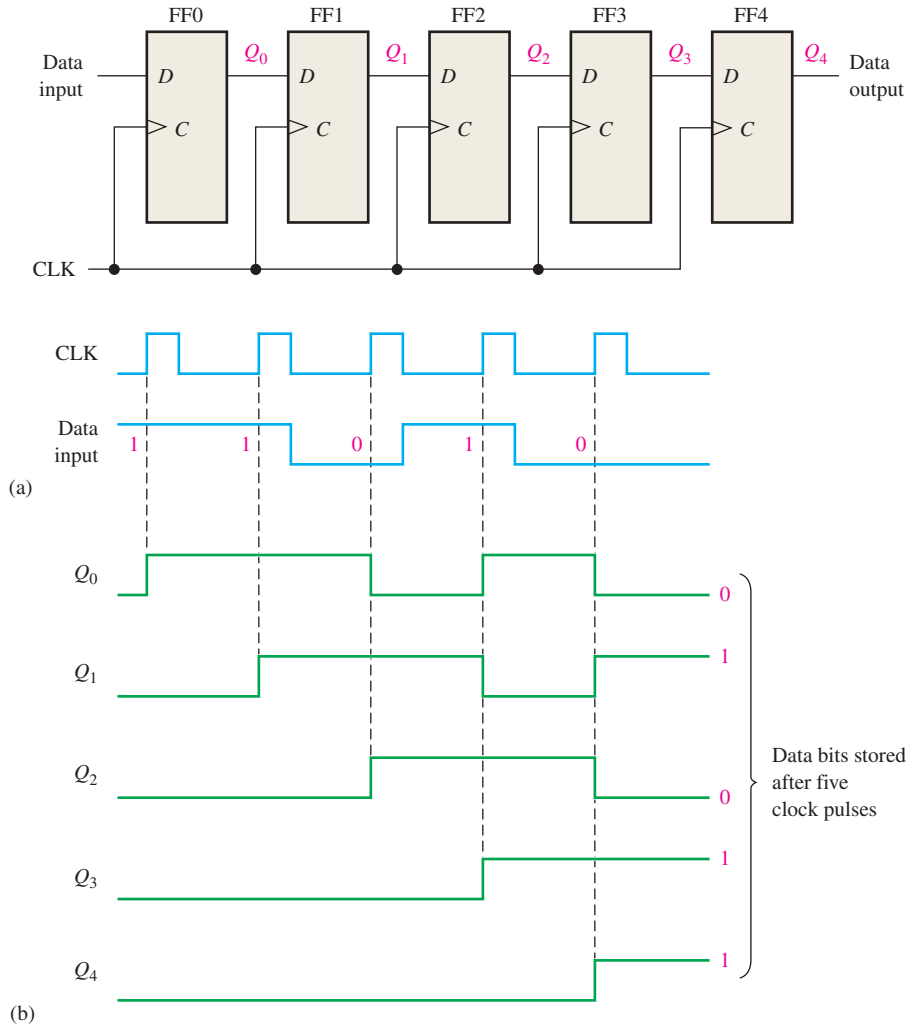


FIGURE 8–4 Open file F08-04 to verify operation. *A Multisim tutorial is available on the website.*

MultiSim

Related Problem*

Show the states of the register if the data input is inverted. The register is initially cleared.

*Answers are at the end of the chapter.

A traditional logic block symbol for an 8-bit serial in/serial out shift register is shown in Figure 8–5. The "SRG 8" designation indicates a shift register (SRG) with an 8-bit capacity.

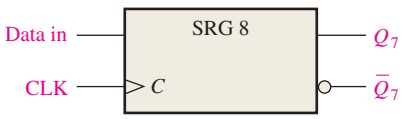


FIGURE 8–5 Logic symbol for an 8-bit serial in/serial out shift register.

Serial In/Parallel Out Shift Registers

Data bits are entered serially (least-significant bit first) into a serial in/parallel out shift register in the same manner as in serial in/serial out registers. The difference is the way in which the data bits are taken out of the register; in the parallel output register, the output of each stage is available. Once the data are stored, each bit appears on its respective output line, and all bits are available simultaneously, rather than on a bit-by-bit basis as with the serial output. Figure 8–6 shows a 4-bit serial in/parallel out shift register and its logic block symbol.

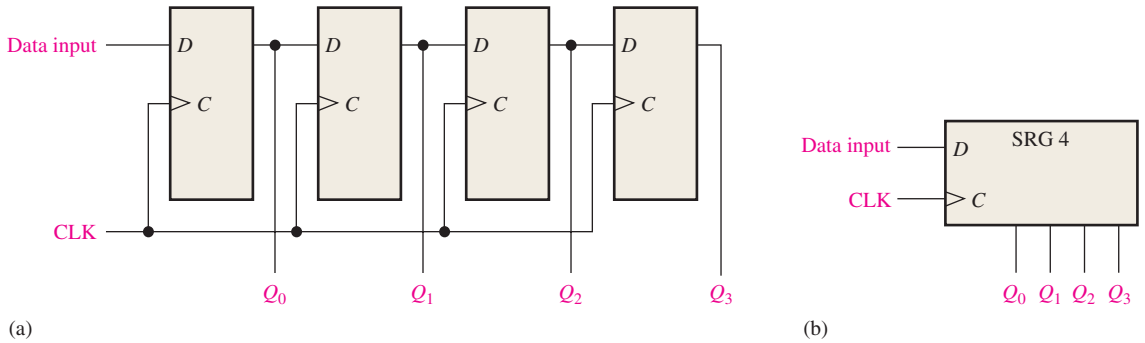


FIGURE 8–6 A serial in/parallel out shift register.

EXAMPLE 8–2

Show the states of the 4-bit register (SRG 4) for the data input and clock waveforms in Figure 8–7(a). The register initially contains all 1s.

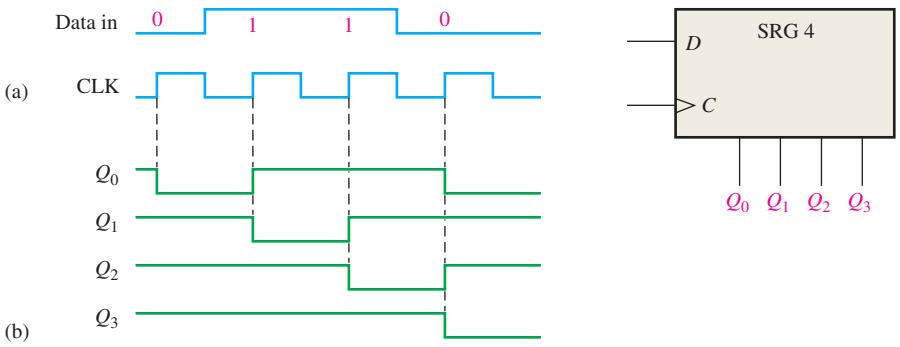


FIGURE 8–7

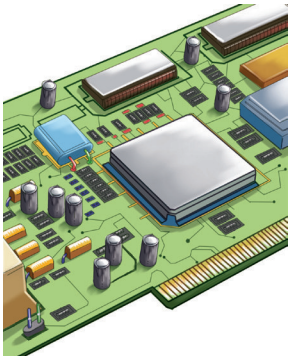
Solution

The register contains 0110 after four clock pulses. See Figure 8–7(b).

Related Problem

If the data input remains 0 after the fourth clock pulse, what is the state of the register after three additional clock pulses?

IMPLEMENTATION: 8-BIT SERIAL IN/PARALLEL OUT SHIFT REGISTER



Fixed-Function Device The 74HC164 is an example of a fixed-function IC shift register having serial in/parallel out operation. The logic block symbol is shown in Figure 8–8. This device has two gated serial inputs, *A* and *B*, and an asynchronous clear ($\overline{CLR}$) input that is active-LOW. The parallel outputs are Q_0 through Q_7.

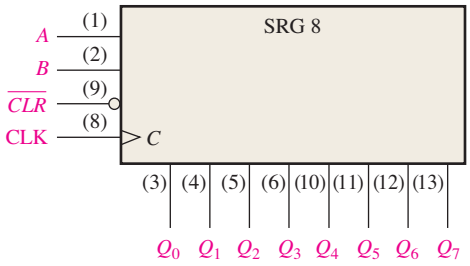


FIGURE 8–8 The 74HC164 8-bit serial in/parallel out shift register.

A sample timing diagram for the 74HC164 is shown in Figure 8–9. Notice that the serial input data on input *A* are shifted into and through the register after input *B* goes HIGH.

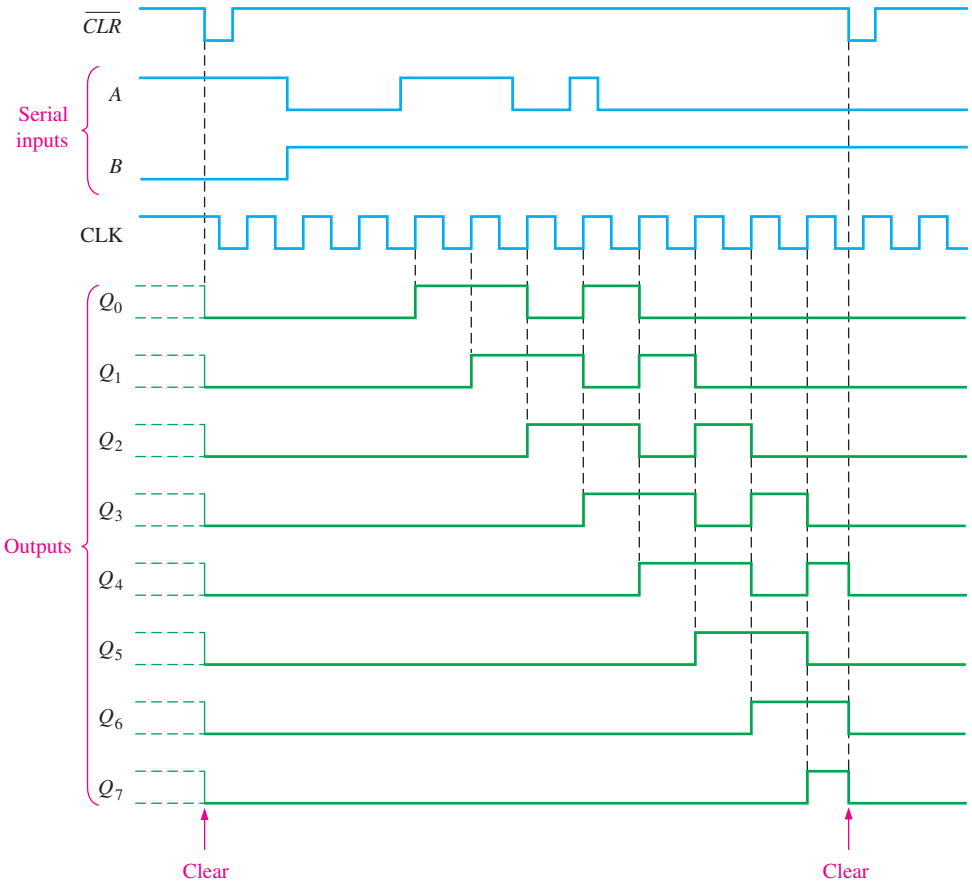


FIGURE 8–9 Sample timing diagram for a 74HC164 shift register.

Programmable Logic Device (PLD) The 8-bit serial in/parallel out shift register can be described using VHDL and implemented as hardware in a PLD. The program code is as follows. (Blue comments are not part of the program.)

```
library ieee;
use ieee.std_logic_1164.all;
entity SerInParOutShift is
   port (D0, Clock, Clr: in std_logic; Q0, Q1, Q2, Q3,
   Q4, Q5, Q6, Q7: inout std_logic);
end entity SerInParOutShift;
architecture LogicOperation of SerInParOutShift is
component dffl is
   port (D, Clock: in std_logic; Q: inout std_logic);
end component dff1;
begin
      FF0: dff1 port map(D=>D0 and Clr, Clock=>Clock, Q=>Q0);
      FF1: dff1 port map(D=>Q0 and Clr, Clock=>Clock, Q=>Q1);
      FF2: dff1 port map(D=>Q1 and Clr, Clock=>Clock, Q=>Q2);
      FF3: dff1 port map(D=>Q2 and Clr, Clock=>Clock, Q=>Q3);
      FF4: dff1 port map(D=>Q3 and Clr, Clock=>Clock, Q=>Q4);
      FF5: dff1 port map(D=>Q4 and Clr, Clock=>Clock, Q=>Q5);
      FF6: dff1 port map(D=>Q5 and Clr, Clock=>Clock, Q=>Q6);
      FF7: dff1 port map(D=>Q6 and Clr, Clock=>Clock, Q=>Q7);
   end architecture LogicOperation;
```

D0: Data input
Clock: System clock
Clr: Clear
Q0–Q7: Register outputs

D flip-flop with preset and clear inputs was described in Chapter 7 and is used as a component.

Instantiations describe how the flip-flops are connected to form the register.

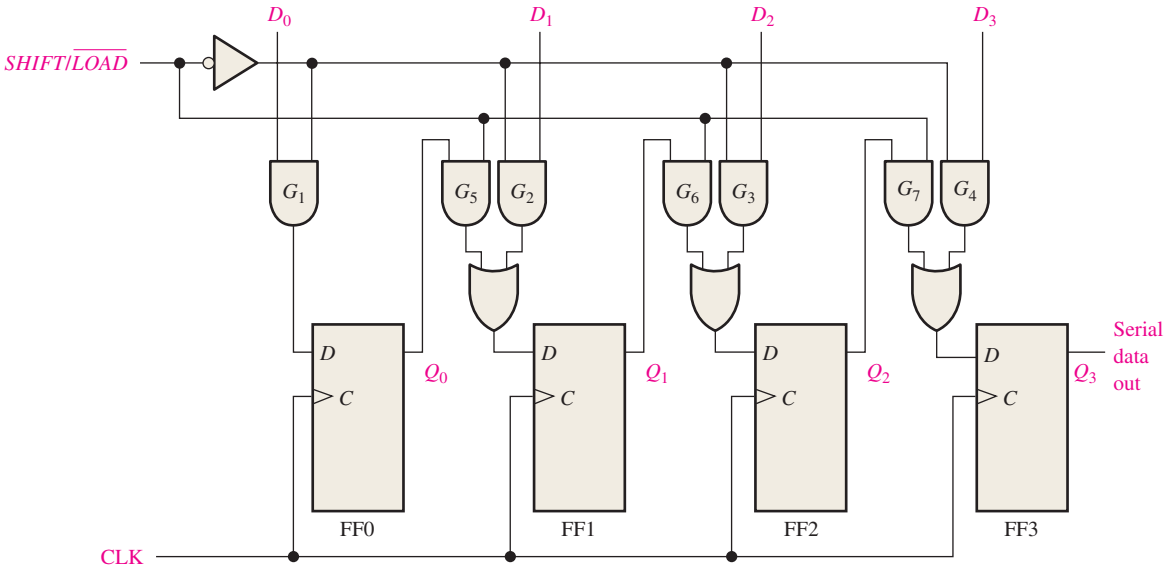
Parallel In/Serial Out Shift Registers

For a register with parallel data inputs, the bits are entered simultaneously into their respective stages on parallel lines rather than on a bit-by-bit basis on one line as with serial data inputs. The serial output is the same as in serial in/serial out shift registers, once the data are completely stored in the register.

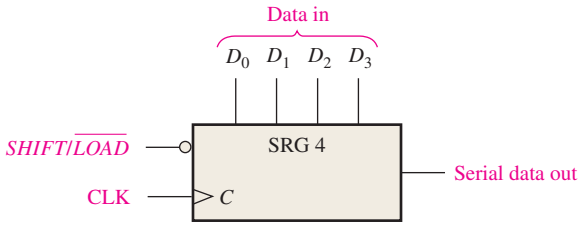
For parallel data, multiple bits are transferred at one time.

Figure 8–10 illustrates a 4-bit parallel in/serial out shift register and a typical logic symbol. There are four data-input lines, D_0, D_1, D_2, and D_3, and a $SHIFT/\overline{LOAD}$ input, which allows four bits of data to **load** in parallel into the register. When $SHIFT/\overline{LOAD}$ is LOW, gates G_1 through G_4 are enabled, allowing each data bit to be applied to the D input of its respective flip-flop. When a clock pulse is applied, the flip-flops with $D = 1$ will set and those with $D = 0$ will reset, thereby storing all four bits simultaneously.

When $SHIFT/\overline{LOAD}$ is HIGH, gates G_1 through G_4 are disabled and gates G_5 through G_7 are enabled, allowing the data bits to shift right from one stage to the next. The OR gates allow either the normal shifting operation or the parallel data-entry operation, depending on which AND gates are enabled by the level on the $SHIFT/\overline{LOAD}$ input. Notice that FF0 has a single AND to disable the parallel input, D_0. It does not require an AND/OR arrangement because there is no serial data in.



(a) Logic diagram



(b) Logic symbol

FIGURE 8–10 A 4-bit parallel in/serial out shift register. Open file F08-10 to verify operation.

MultiSim

EXAMPLE 8–3

Show the data-output waveform for a 4-bit register with the parallel input data and the clock and *SHIFT*/$\overline{LOAD}$ waveforms given in Figure 8–11(a). Refer to Figure 8–10(a) for the logic diagram.

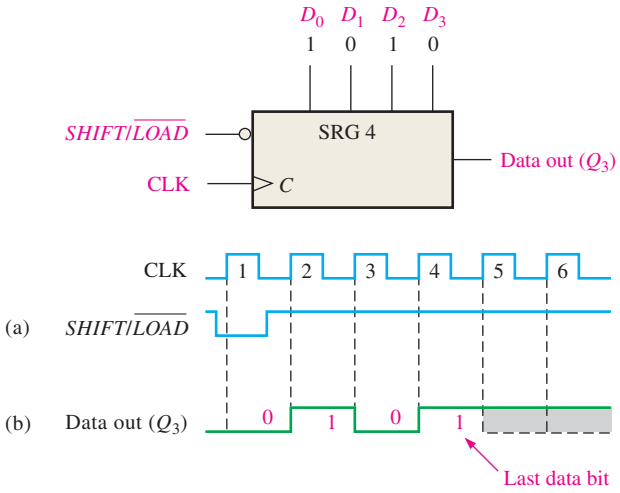


FIGURE 8–11

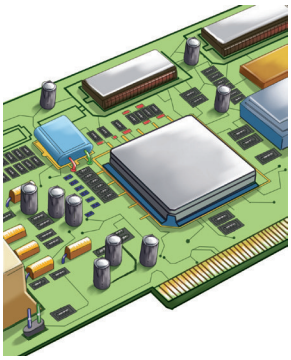
Solution

On clock pulse 1, the parallel data ($D_0D_1D_2D_3 = 1010$) are loaded into the register, making Q_3 a 0. On clock pulse 2 the 1 from Q_2 is shifted onto Q_3; on clock pulse 3 the 0 is shifted onto Q_3; on clock pulse 4 the last data bit (1) is shifted onto Q_3; and on clock pulse 5, all data bits have been shifted out, and only 1s remain in the register (assuming the D_0 input remains a 1). See Figure 8–11(b).

Related Problem

Show the data-output waveform for the clock and *SHIFT*/$\overline{LOAD}$ inputs shown in Figure 8–11(a) if the parallel data are $D_0D_1D_2D_3 = 0101$.

IMPLEMENTATION: 8-BIT PARALLEL LOAD SHIFT REGISTER



Fixed-Function Device The 74HC165 is an example of a fixed-function IC shift register that has a parallel in/serial out operation (it can also be operated as serial in/serial out). Figure 8–12 shows a typical logic block symbol. A LOW on the *SHIFT*/$\overline{LOAD}$ input (*SH*/$\overline{LD}$) enables asynchronous parallel loading. Data can be entered serially on the *SER* input. Also, the clock can be inhibited anytime with a HIGH on the *CLK INH* input. The serial data outputs of the register are Q_7 and its complement $\overline{Q}_7$. This implementation is different from the synchronous method of parallel loading previously discussed, demonstrating that there are usually several ways to accomplish the same function.

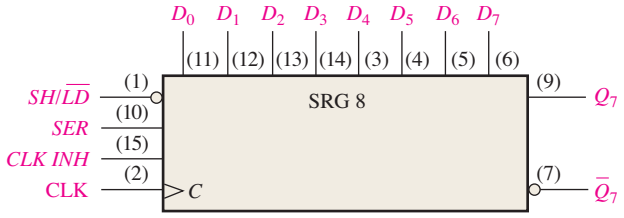


FIGURE 8–12 The 74HC165 8-bit parallel load shift register.

Figure 8–13 is a timing diagram showing an example of the operation of a 74HC165 shift register.

Programmable Logic Device (PLD) The 8-bit parallel load shift register is a parallel in/serial out device and can be implemented in a PLD with the following VHDL code:

```
library ieee:
use ieee.std_logic_1164.all;

entity ParSerShift is
    port (D0, D1, D2, D3, D4, D5, D6, D7, SHLD, Clock:
          in std_logic; Q, QNot: inout std_logic);
end entity ParSerShift;

architecture LogicOperation of ParSerShift is
    signal S1, S2, S3, S4, S5, S6, S7,
           Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7: std_logic;
function ShiftLoad (A,B,C: in std_logic)return std_logic is
begin
    return ((A and B) or (not B and C));
end function ShiftLoad;
```

D0-D7: Parallel input
SHLD: Shift Load input
Clock: System clock
Q: Serial output
QNot: Inverted serial output

S1-S7: Shift load signals from function ShiftLoad
Q0-Q7: Intermediate variables for flip-flop stages

Function ShiftLoad provides the AND-OR function shown in Figure 8–10 to allow the parallel load of data or data shift from one flip-flop stage to the next.

```
component dff1 is
port (D, Clock: in std_logic;
      Q: inout std_logic);
end component dff1;
```

D flip-flop component used as storage for shift register

```
begin
   SL1:S1 <=ShiftLoad(Q0, SHLD, D1);
   SL2:S2 <=ShiftLoad(Q1, SHLD, D2);
   SL3:S3 <=ShiftLoad(Q2, SHLD, D3);
   SL4:S4 <=ShiftLoad(Q3, SHLD, D4);
   SL5:S5 <=ShiftLoad(Q4, SHLD, D5);
   SL6:S6 <=ShiftLoad(Q5, SHLD, D6);
   SL7:S7 <=ShiftLoad(Q6, SHLD, D7);
   FF0: dff1 port map(D=>D0 and not SHLD, Clock=>Clock, Q=>Q0);
   FF1: dff1 port map(D=>S1, Clock=>Clock, Q=>Q1);
   FF2: dff1 port map(D=>S2, Clock=>Clock, Q=>Q2);
   FF3: dff1 port map(D=>S3, Clock=>Clock, Q=>Q3);
   FF4: dff1 port map(D=>S4, Clock=>Clock, Q=>Q4);
   FF5: dff1 port map(D=>S5, Clock=>Clock, Q=>Q5);
   FF6: dff1 port map(D=>S6, Clock=>Clock, Q=>Q6);
   FF7: dff1 port map(D=>S7, Clock=>Clock, Q=>Q);
   QNot <=not Q;
end architecture LogicOperation;
```

ShiftLoad instances SL1–SL7 allow eight bits of data to load into flip-flop stages FF0–FF7 or to shift through the register providing the parallel load serial out function.

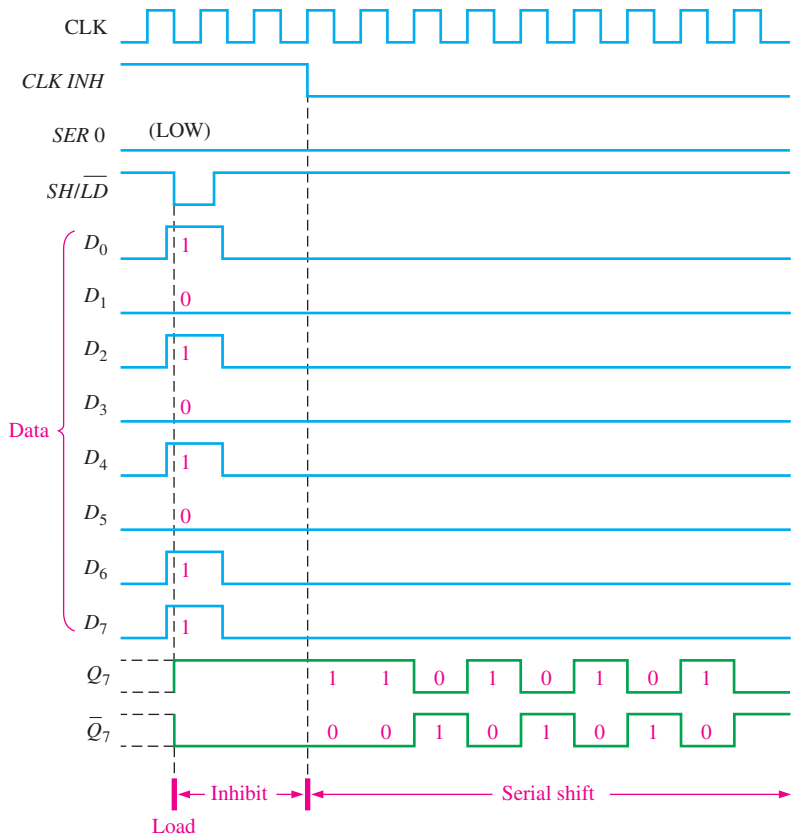


FIGURE 8–13 Sample timing diagram for a 74HC165 shift register.

Parallel In/Parallel Out Shift Registers

Parallel entry and parallel output of data have been discussed. The parallel in/parallel out register employs both methods. Immediately following the simultaneous entry of all data bits, the bits appear on the parallel outputs. Figure 8–14 shows a parallel in/parallel out shift register.

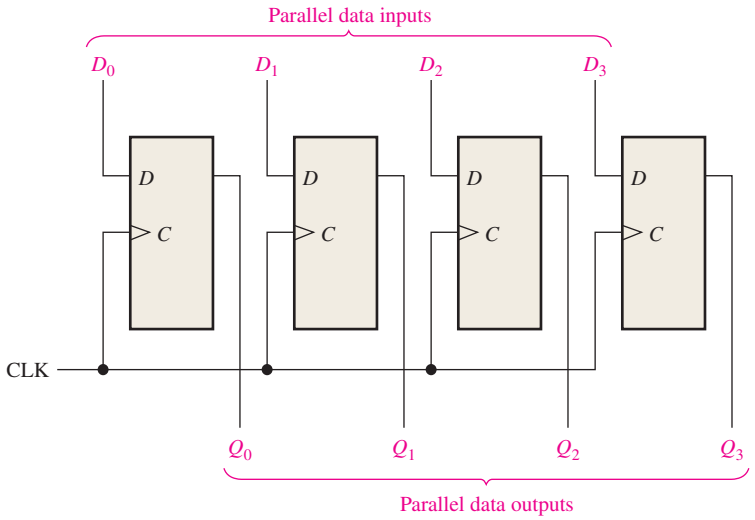
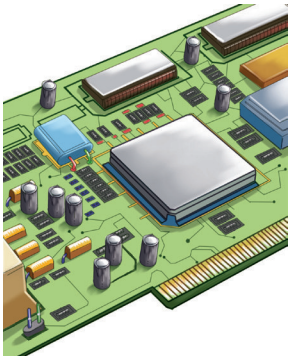


FIGURE 8–14 A parallel in/parallel out register.

IMPLEMENTATION: 4-BIT PARALLEL-ACCESS SHIFT REGISTER



Fixed-Function Device The 74HC195 can be used for parallel in/parallel out operation. Because it also has a serial input, it can be used for serial in/serial out and serial in/parallel out operations. It can be used for parallel in/serial out operation by using Q_3 as the output. A typical logic block symbol is shown in Figure 8–15.

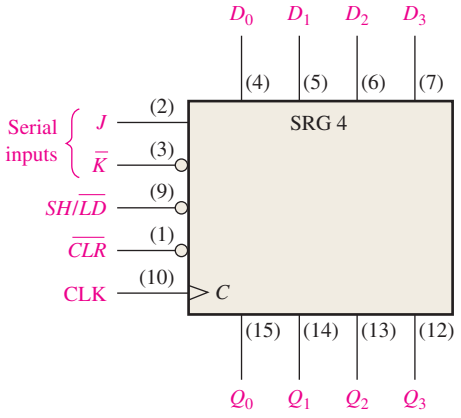


FIGURE 8–15 The 74HC195 4-bit parallel access shift register.

When the $SHIFT/\overline{LOAD}$ input ($SH/\overline{LD}$) is LOW, the data on the parallel inputs are entered synchronously on the positive transition of the clock. When ($SH/\overline{LD}$) is HIGH, stored data will shift right (Q_0 to Q_3) synchronously with the clock. Inputs J and $\overline{K}$ are the serial data inputs to the first stage of the register (Q_0); Q_3 can be used for serial output data. The active-LOW clear input is asynchronous.

The timing diagram in Figure 8–16 illustrates the operation of this register.

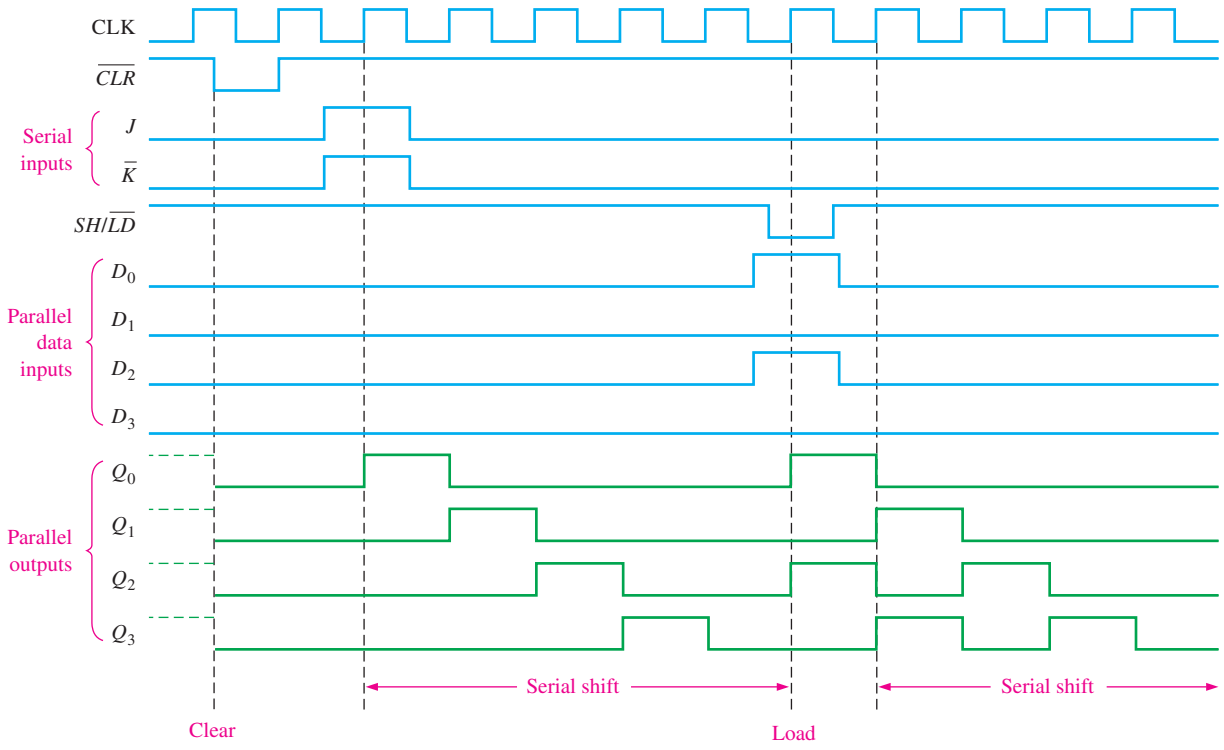


FIGURE 8–16 Sample timing diagram for a 74HC195 shift register.

Programmable Logic Device (PLD) The VHDL code for a 4-bit parallel in/parallel out shift register is as follows:

```
library ieee;
use ieee.std logic_1164.all;

entity ParInParOut is
   port (D0, D1, D2, D3, Clock: in std_logic;
         Q0, Q1, Q2, Q3: inout std_logic);
end entity ParInParOut;

architecture LogicOperation of ParInParOut is
   component dff1 is
      port (D, Clock: in std_logic;
            Q: inout std_logic);
end component dff1;

begin
   FF0: dff1 port map (D=>D0, Clock=>Clock, Q=>Q0);
   FF1: dff1 port map (D=>D1, Clock=>Clock, Q=>Q1);
   FF2: dff1 port map (D=>D2, Clock=>Clock, Q=>Q2);
   FF3: dff1 port map (D=>D3, Clock=>Clock, Q=>Q3);
end architecture LogicOperation;
```

SECTION 8–2 CHECKUP

1. Develop the logic diagram for the shift register in Figure 8–3, using J-K flip-flops to replace the D flip-flops.
2. How many clock pulses are required to enter a byte of data serially into an 8-bit shift register?
3. The bit sequence 1101 is serially entered (least-significant bit first) into a 4-bit parallel out shift register that is initially clear. What are the Q outputs after two clock pulses?
4. How can a serial in/parallel out register be used as a serial in/serial out register?
5. Explain the function of the $SHIFT/\overline{LOAD}$ input.
6. Is the parallel load operation in a 74HC165 shift register synchronous or asynchronous? What does this mean?
7. In Figure 8–14, $D_0 = 1, D_1 = 0, D_2 = 0$, and $D_3 = 1$. After three clock pulses, what are the data outputs?
8. For a 74HC195, $SH/\overline{LD} = 1, J = 1$, and $\overline{K} = 1$. What is Q_0 after one clock pulse?

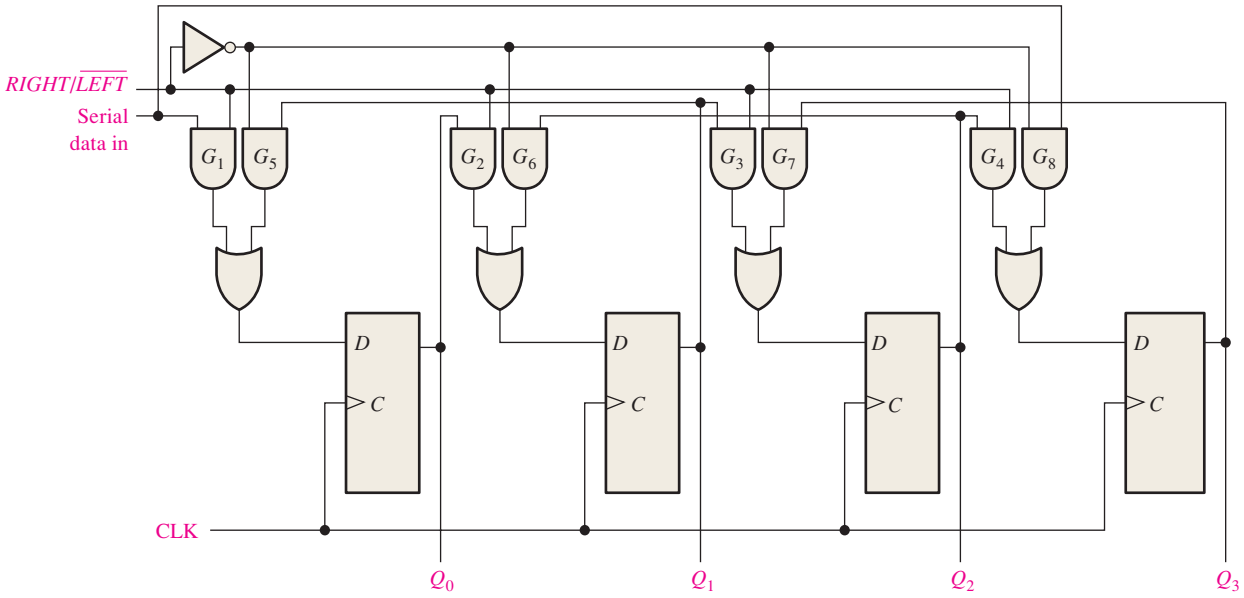
8–3 Bidirectional Shift Registers

A **bidirectional** shift register is one in which the data can be shifted either left or right. It can be implemented by using gating logic that enables the transfer of a data bit from one stage to the next stage to the right or to the left, depending on the level of a control line.

After completing this section, you should be able to

- Explain the operation of a bidirectional shift register
- Discuss the 74HC194 4-bit bidirectional universal shift register
- Develop and analyze timing diagrams for bidirectional shift registers

A 4-bit bidirectional shift register is shown in Figure 8–17. A HIGH on the $RIGHT/\overline{LEFT}$ control input allows data bits inside the register to be shifted to the right, and a LOW



MultiSim **FIGURE 8–17** Four-bit bidirectional shift register. Open file F08-17 to verify the operation.

enables data bits inside the register to be shifted to the left. An examination of the gating logic will make the operation apparent. When the *RIGHT*/$\overline{LEFT}$ control input is HIGH, gates G_1 through G_4 are enabled, and the state of the Q output of each flip-flop is passed through to the D input of the *following* flip-flop. When a clock pulse occurs, the data bits are shifted one place to the *right.* When the *RIGHT*/$\overline{LEFT}$ control input is LOW, gates G_5 through G_8 are enabled, and the Q output of each flip-flop is passed through to the D input of the *preceding* flip-flop. When a clock pulse occurs, the data bits are then shifted one place to the *left.*

EXAMPLE 8–4

Determine the state of the shift register of Figure 8–17 after each clock pulse for the given *RIGHT*/$\overline{LEFT}$ control input waveform in Figure 8–18(a). Assume that $Q_0 = 1$, $Q_1 = 1$, $Q_2 = 0$, and $Q_3 = 1$ and that the serial data-input line is LOW.

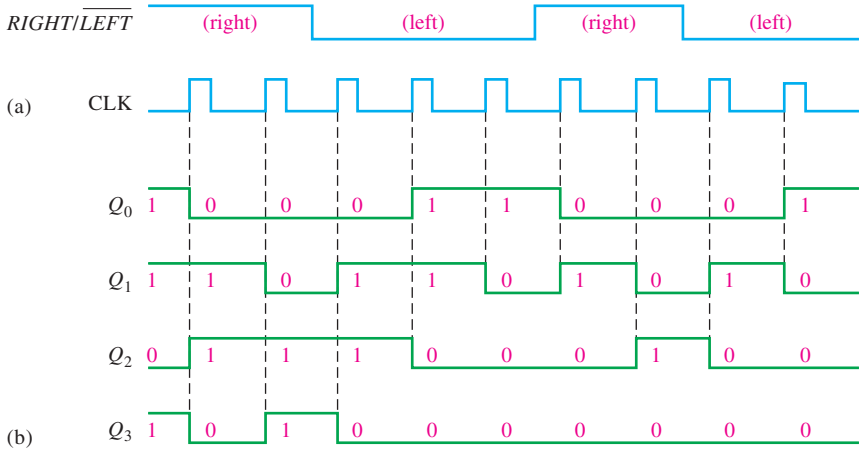


FIGURE 8–18

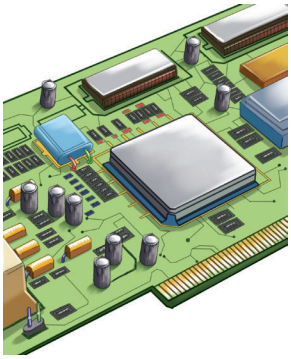
Solution

See Figure 8–18(b).

Related Problem

Invert the *RIGHT*/$\overline{LEFT}$ waveform, and determine the state of the shift register in Figure 8–17 after each clock pulse.

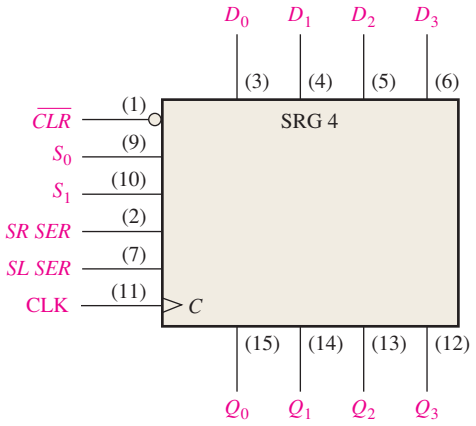
IMPLEMENTATION: 4-BIT BIDIRECTIONAL UNIVERSAL SHIFT REGISTER



Fixed-Function Device The 74HC194 is an example of a universal bidirectional shift register in integrated circuit form. A **universal shift register** has both serial and parallel input and output capability. A logic block symbol is shown in Figure 8–19, and a sample timing diagram is shown in Figure 8–20.

Parallel loading, which is synchronous with a positive transition of the clock, is accomplished by applying the four bits of data to the parallel inputs and a HIGH to the S_0 and S_1 inputs. Shift right is accomplished synchronously with the positive edge of the clock when S_0 is HIGH and S_1 is LOW. Serial data in this mode are entered at the shift-right serial input (*SR SER*). When S_0 is LOW and S_1 is HIGH, data bits shift left synchronously with the clock, and new data are entered at the shift-left serial input (*SL SER*). Input *SR SER* goes into the Q_0 stage, and *SL SER* goes into the Q_3 stage.

FIGURE 8–19 The 74HC194 4-bit bidirectional universal shift register.



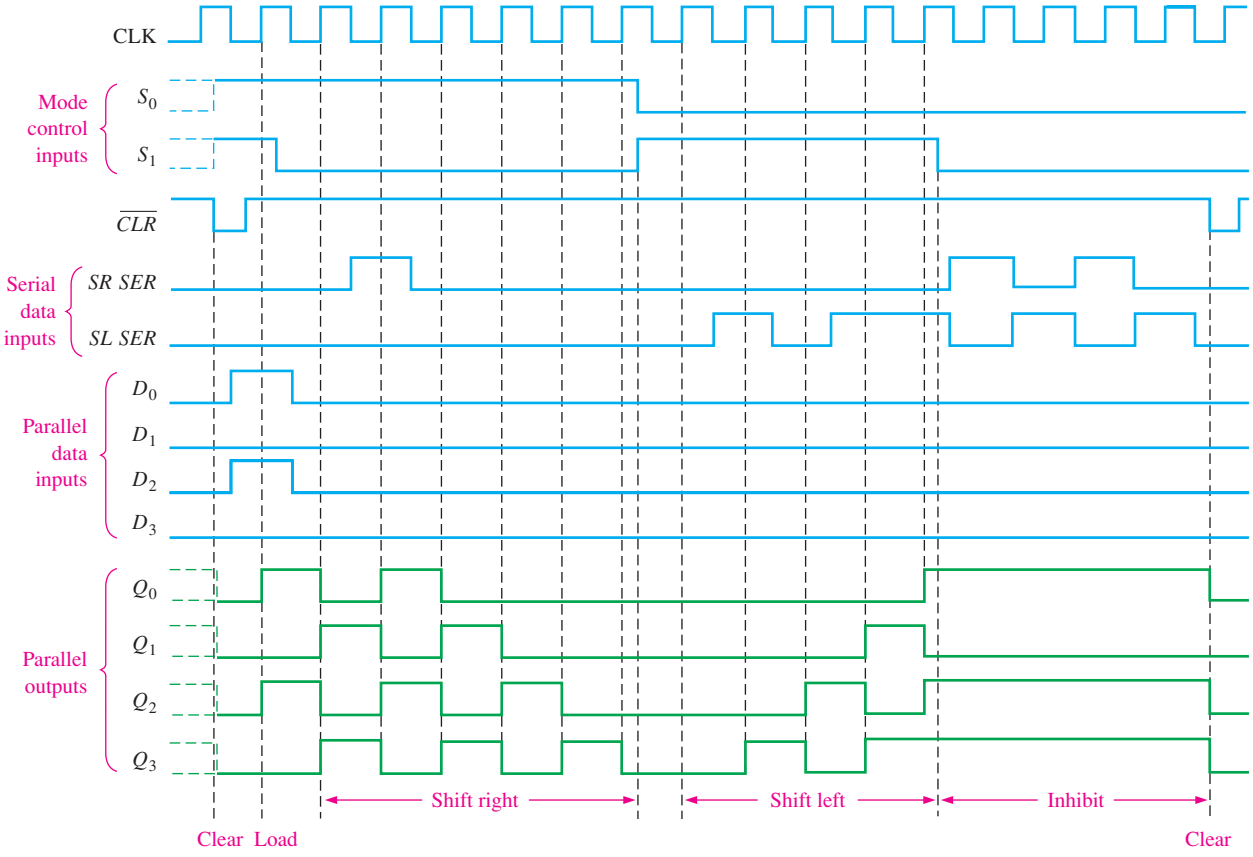


FIGURE 8–20 Sample timing diagram for a 74HC194 shift register.

Programmable Logic Device (PLD) The following code describes a 4-bit bidirectional shift register with a serial input:

```
library ieee;
use ieee.std_logic_1164.all;
entity FourBitBiDirSftReg is
port (R_L, DataIn, Clock: in std_logic;
   Q0, Q1, Q2, Q3: buffer std_logic);
end entity FourBitBiDirSftReg;
```

R_L: Right/left
DataIn: Serial input data
Clock: System clock
Q0-Q3: Register outputs

```
architecture LogicOperation of FourBitBiDirSftReg is
component dff1 is
   port(D,Clock: in std_logic; Q: out std_logic);   } D flip-flop component declaration
end component dff1;
signal D0, D1, D2, D3: std_logic;   Internal flip-flop inputs
begin
   DO <= (DataIn and R_L) or (not R_L and Q1);
   D1 <= (Q0 and R_L) or (not R_L and Q2);          } Describes the internal signals
   D2 <= (Q1 and R_L) or (not R_L and Q3);            with Boolean equations
   D3 <= (Q2 and R_L) or (not R_L and DataIn);
   FF0: dff1 port map(D => D0, Clock => Clock, Q => Q0);
   FF1: dff1 port map(D => D1, Clock => Clock, Q => Q1);   } Describes how the
   FF2: dff1 port map(D => D2, Clock => Clock, Q => Q2);     flip-flops are connected
   FF3: dff1 port map(D => D3, Clock => Clock, Q => Q3);
end architecture LogicOperation;
```

SECTION 8–3 CHECKUP

1. Assume that the 4-bit bidirectional shift register in Figure 8–17 has the following contents: $Q_0 = 1, Q_1 = 1, Q_2 = 0$, and $Q_3 = 0$. There is a 1 on the serial data-input line. If *RIGHT*/$\overline{LEFT}$ is HIGH for three clock pulses and LOW for two more clock pulses, what are the contents after the fifth clock pulse?

8–4 Shift Register Counters

A shift register counter is basically a shift register with the serial output connected back to the serial input to produce special sequences. These devices are often classified as counters because they exhibit a specified sequence of states. Two of the most common types of shift register counters, the Johnson counter and the ring counter, are introduced in this section.

After completing this section, you should be able to

- Discuss how a shift register counter differs from a basic shift register
- Explain the operation of a Johnson counter
- Specify a Johnson sequence for any number of bits
- Explain the operation of a ring counter and determine the sequence of any specific ring counter

The Johnson Counter

In a **Johnson counter** the complement of the output of the last flip-flop is connected back to the *D* input of the first flip-flop (it can be implemented with other types of flip-flops as well). If the counter starts at 0, this feedback arrangement produces a characteristic sequence of states, as shown in Table 8–3 for a 4-bit device and in Table 8–4 for a 5-bit device. Notice that the 4-bit sequence has a total of eight states, or bit patterns, and that the 5-bit sequence has a total of ten states. In general, a Johnson counter will produce a modulus of 2*n,* where *n* is the number of stages in the counter.

TABLE 8–3

Four-bit Johnson sequence.

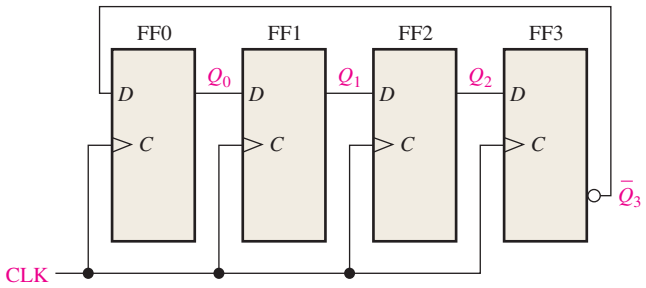
Clock Pulse	Q_0	Q_1	Q_2	Q_3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

TABLE 8–4

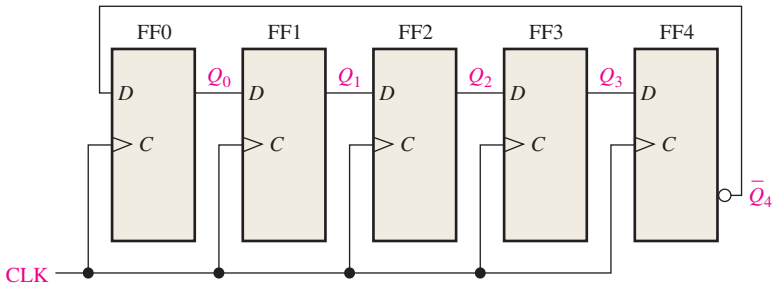
Five-bit Johnson sequence.

Clock Pulse	Q_0	Q_1	Q_2	Q_3	Q_4
0	0	0	0	0	0
1	1	0	0	0	0
2	1	1	0	0	0
3	1	1	1	0	0
4	1	1	1	1	0
5	1	1	1	1	1
6	0	1	1	1	1
7	0	0	1	1	1
8	0	0	0	1	1
9	0	0	0	0	1

The implementations of the 4-stage and 5-stage Johnson counters are shown in Figure 8–21. The implementation of a Johnson counter is very straightforward and is the same regardless of the number of stages. The Q output of each stage is connected to the D input of the next



(a) Four-bit Johnson counter



(b) Five-bit Johnson counter

FIGURE 8–21 Four-bit and 5-bit Johnson counters.

stage (assuming that *D* flip-flops are used). The single exception is that the $\overline{Q}$ output of the last stage is connected back to the *D* input of the first stage. As the sequences in Table 8–3 and 8–4 show, if the counter starts at 0, it will "fill up" with 1s from left to right, and then it will "fill up" with 0s again.

Diagrams of the timing operations of the 4-bit and 5-bit counters are shown in Figures 8–22 and 8–23, respectively.

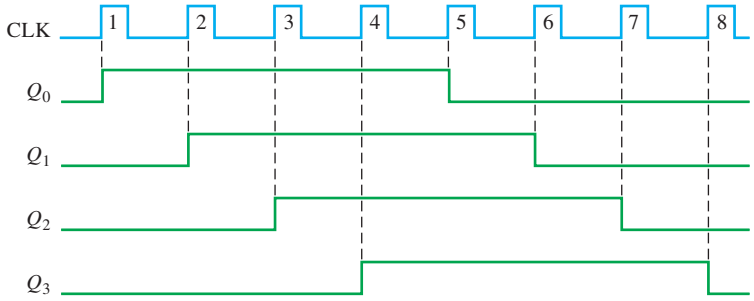


FIGURE 8–22 Timing sequence for a 4-bit Johnson counter.

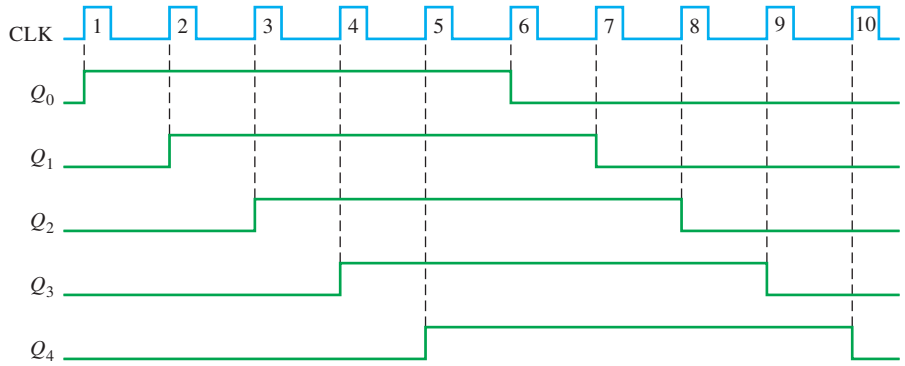


FIGURE 8–23 Timing sequence for a 5-bit Johnson counter.

The Ring Counter

A **ring counter** utilizes one flip-flop for each state in its sequence. It has the advantage that decoding gates are not required. In the case of a 10-bit ring counter, there is a unique output for each decimal digit.

A logic diagram for a 10-bit ring counter is shown in Figure 8–24. The sequence for this ring counter is given in Table 8–5. Initially, a 1 is preset into the first flip-flop, and the rest of the flip-flops are cleared. Notice that the interstage connections are the same as those for a

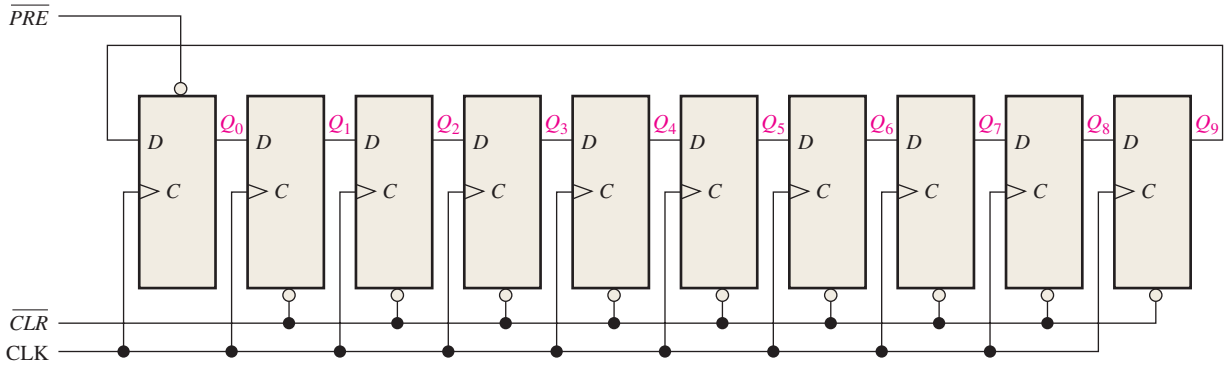


FIGURE 8–24 A 10-bit ring counter. Open file F08-24 to verify operation.

MultiSim

TABLE 8–5

Ten-bit ring counter sequence.

Clock Pulse	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9
0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	1

Johnson counter, except that Q rather than $\overline{Q}$ is fed back from the last stage. The ten outputs of the counter indicate directly the decimal count of the clock pulse. For instance, a 1 on Q_0 represents a zero, a 1 on Q_1 represents a one, a 1 on Q_2 represents a two, a 1 on Q_3 represents a three, and so on. You should verify for yourself that the 1 is always retained in the counter and simply shifted "around the ring," advancing one stage for each clock pulse.

Modified sequences can be achieved by having more than a single 1 in the counter, as illustrated in Example 8–5.

EXAMPLE 8–5

If a 10-bit ring counter similar to Figure 8–24 has the initial state 1010000000, determine the waveform for each of the Q outputs.

Solution

See Figure 8–25.

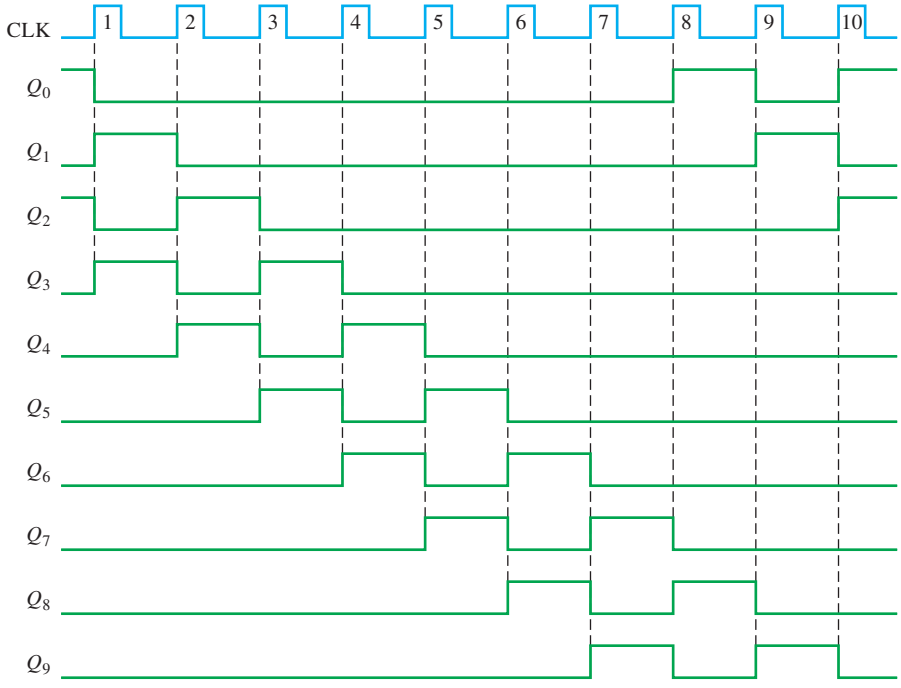


FIGURE 8–25

Related Problem

If a 10-bit ring counter has an initial state 0101001111, determine the waveform for each Q output.

SECTION 8–4 CHECKUP

1. How many states are there in an 8-bit Johnson counter sequence?
2. Write the sequence of states for a 3-bit Johnson counter starting with 000.

8–5 Shift Register Applications

Shift registers are found in many types of applications, a few of which are presented in this section.

After completing this section, you should be able to

- Use a shift register to generate a time delay
- Implement a specified ring counter sequence using a 74HC195 shift register
- Discuss how shift registers are used for serial-to-parallel conversion of data
- Define *UART*
- Explain the operation of a keyboard encoder and how registers are used in this application

Time Delay

A serial in/serial out shift register can be used to provide a time delay from input to output that is a function of both the number of stages (n) in the register and the clock frequency.

When a data pulse is applied to the serial input as shown in Figure 8–26, it enters the first stage on the triggering edge of the clock pulse. It is then shifted from stage to stage on each successive clock pulse until it appears on the serial output n clock periods later. This time-delay operation is illustrated in Figure 8–26, in which an 8-bit serial in/serial out shift register is used with a clock frequency of 1 MHz to achieve a time delay (t_d) of 8 μs (8 $\times$ 1 μs). This time can be adjusted up or down by changing the clock frequency. The time delay can also be increased by cascading shift registers and decreased by taking the outputs from successively lower stages in the register if the outputs are available, as illustrated in Example 8–6.

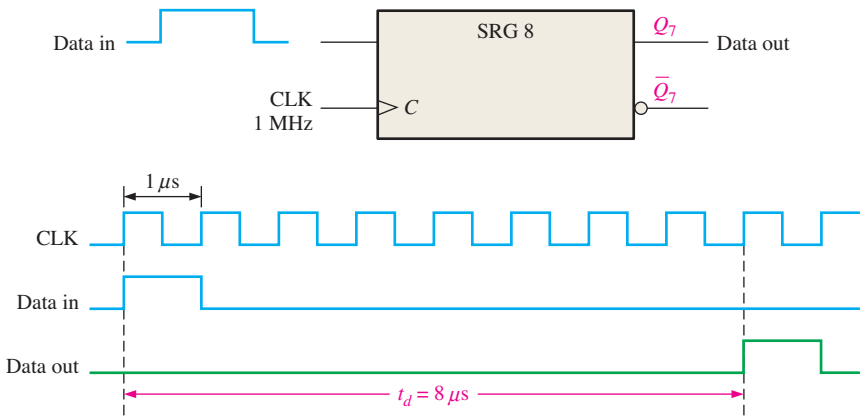


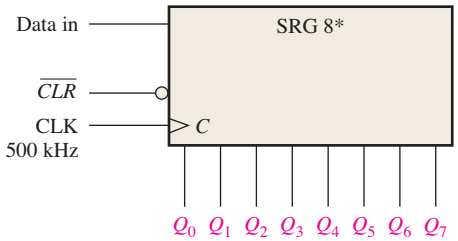
FIGURE 8–26 The shift register as a time-delay device.

InfoNote

Microprocessors have special instructions that can emulate a serial shift register. The accumulator register can shift data to the left or right. A right shift is equivalent to a divide-by-2 operation and a left shift is equivalent to a multiply-by-2 operation. Data in the accumulator can be shifted left or right with the rotate instructions; ROR is the rotate right instruction, and ROL is the rotate left instruction. Two other instructions treat the carry flag bit as an additional bit for the rotate operation. These are the RCR for rotate carry right and RCL for rotate carry left.

EXAMPLE 8–6

Determine the amount of time delay between the serial input and each output in Figure 8–27. Show a timing diagram to illustrate.



* Data shifts from Q_0 toward Q_7.

FIGURE 8–27

Solution

The clock period is 2 μs. Thus, the time delay can be increased or decreased in 2 μs increments from a minimum of 2 μs to a maximum of 16 μs, as illustrated in Figure 8–28.

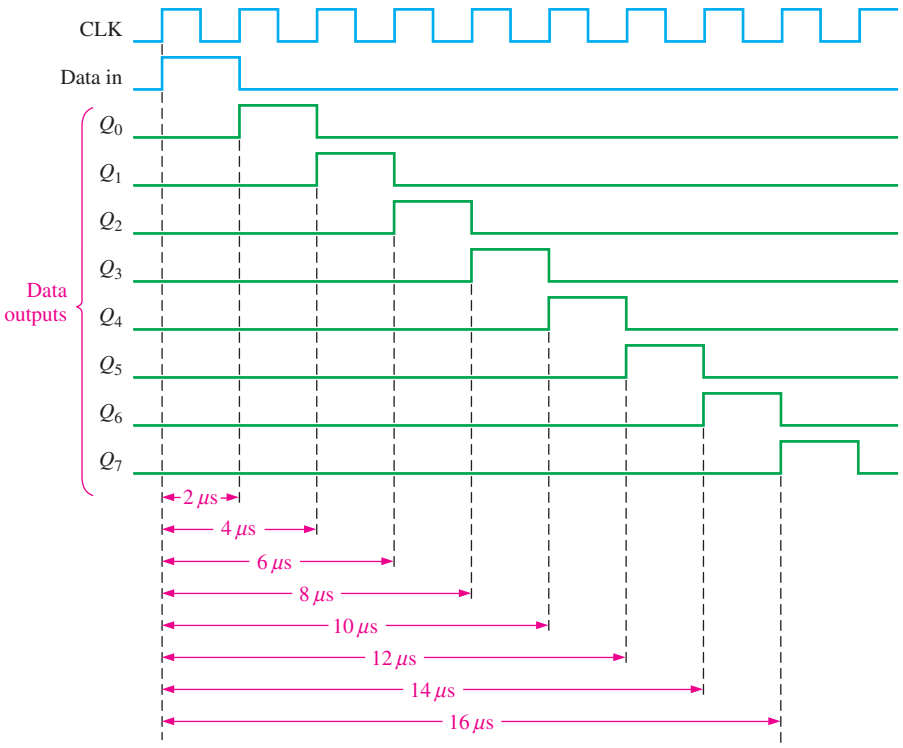
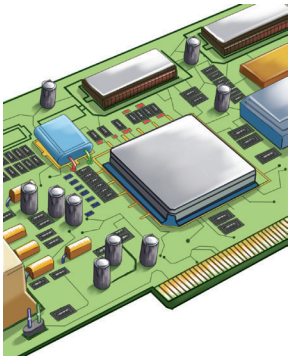


FIGURE 8–28 Timing diagram showing time delays for the register in Figure 8–27.

Related Problem

Determine the clock frequency required to obtain a time delay of 24 μs to the Q_7 output in Figure 8–27.

IMPLEMENTATION: A RING COUNTER



Fixed-Function Device If the output is connected back to the serial input, a shift register can be used as a ring counter. Figure 8–29 illustrates this application with a 74HC195 4-bit shift register.

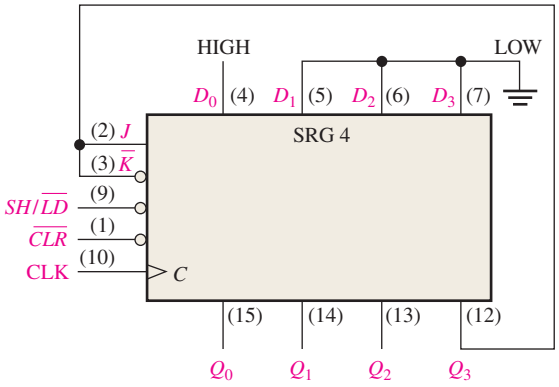


FIGURE 8–29 74HC195 connected as a ring counter.

Initially, a bit pattern of 1000 (or any other pattern) can be synchronously preset into the counter by applying the bit pattern to the parallel data inputs, taking the $SH/\overline{LD}$ input LOW, and applying a clock pulse. After this initialization, the 1 continues to circulate through the ring counter, as the timing diagram in Figure 8–30 shows.

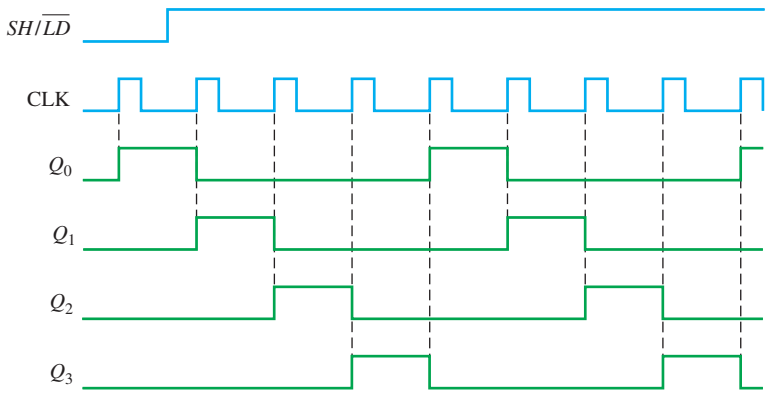


FIGURE 8–30 Timing diagram showing two complete cycles of the ring counter in Figure 8–29 when it is initially preset to 1000.

Programmable Logic Device (PLD) The VHDL code for a 4-bit ring counter using D flip-flops is as follows:

```
library ieee;
use ieee.std_logic_1164.all;

entity RingCtr is
   port (I, Clr, Clock: in std_logic;
         Q0, Q1, Q2, Q3: inout std_logic);
end entity RingCtr;

architecture LogicOperation of RingCtr is
```

I: Serial input bit to clock data into the shift register
Clr: Ring counter clear input
Clock: System clock
Q0-Q3: Ring counter output stages

```
component dff1 is
   port (D, Clock, Pre, Clr: in std_logic;
      Q: inout std_logic);
end component dff1;
begin
   FF0: dff1 port map(D=> Q3, Clock=>Clock, Q=>Q0, Pre=> not I, Clr=>'1');
   FF1: dff1 port map(D=> Q0, Clock=>Clock, Q=>Q1, Pre=>'1', Clr=>not Clr);
   FF2: dff1 port map(D=> Q1, Clock=>Clock, Q=>Q2, Pre=>'1', Clr=>not Clr);
   FF3: dff1 port map(D=> Q2, Clock=>Clock, Q=>Q3, Pre=>'1', Clr=>not Clr);
end architecture LogicOperation;
```

D flip-flop component used as storage for shift register

FF0-FF3 flip-flop instantiations show how flip-flops are connected and represent one flip-flop for each state in the ring counter sequence. FF0 Pre input acts as a serial input when I is high. FF1-FF3 Clr input clears flip-flop stages when Clr is low.

Serial-to-Parallel Data Converter

Serial data transmission from one digital system to another is commonly used to reduce the number of wires in the transmission line. For example, eight bits can be sent serially over one wire, but it takes eight wires to send the same data in parallel.

Serial data transmission is widely used by peripherals to pass data back and forth to a computer. For example, USB (universal serial bus) is used to connect keyboards printers, scanners, and more to the computer. All computers process data in parallel form, thus the requirement for serial-to-parallel conversion. A simplified serial-to-parallel data converter, in which two types of shift registers are used, is shown in Figure 8–31.

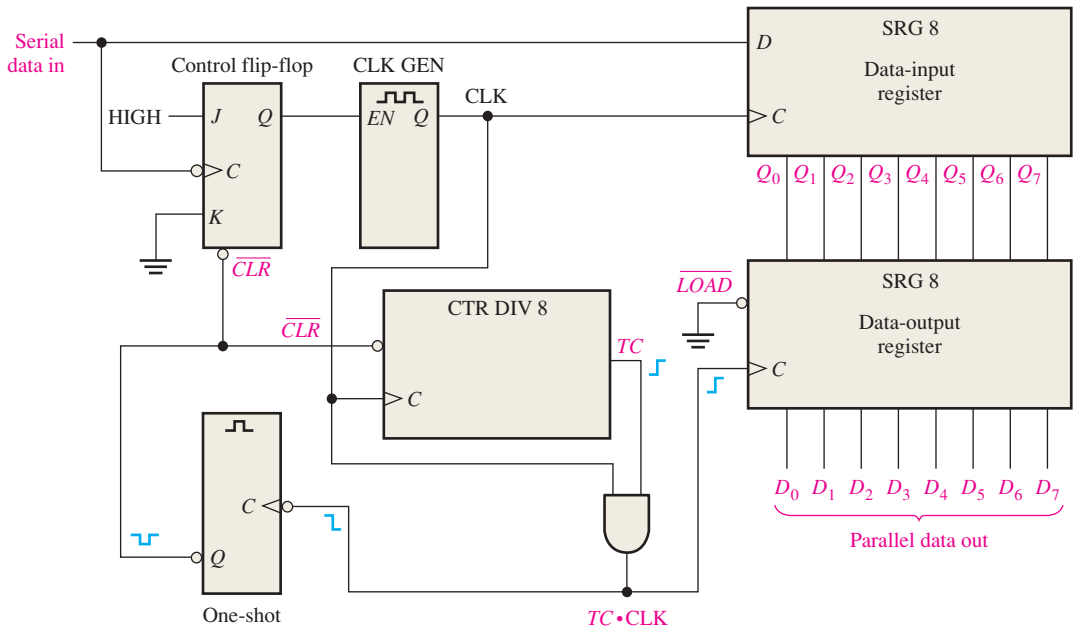


FIGURE 8–31 Simplified logic diagram of a serial-to-parallel converter.

To illustrate the operation of this serial-to-parallel converter, the serial data format shown in Figure 8–32 is used. It consists of eleven bits. The first bit (start bit) is always 0 and always begins with a HIGH-to-LOW transition. The next eight bits (D_7 through D_0) are the data bits (one of the bits can be parity), and the last one or two bits (stop bits) are always 1s. When no data are being sent, there is a continuous HIGH on the serial data line.

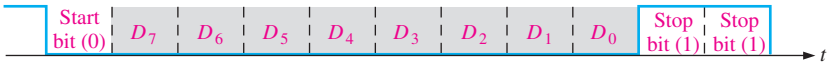


FIGURE 8–32 Serial data format.

The HIGH-to-LOW transition of the start bit sets the control flip-flop, which enables the clock generator. After a fixed delay time, the clock generator begins producing a pulse waveform, which is applied to the data-input register and to the divide-by-8 counter. The clock has a frequency precisely equal to that of the incoming serial data, and the first clock pulse after the start bit occurs during the first data bit.

The timing diagram in Figure 8–33 illustrates the following basic operation: The eight data bits (D_7 through D_0) are serially shifted into the data-input register. Shortly after the

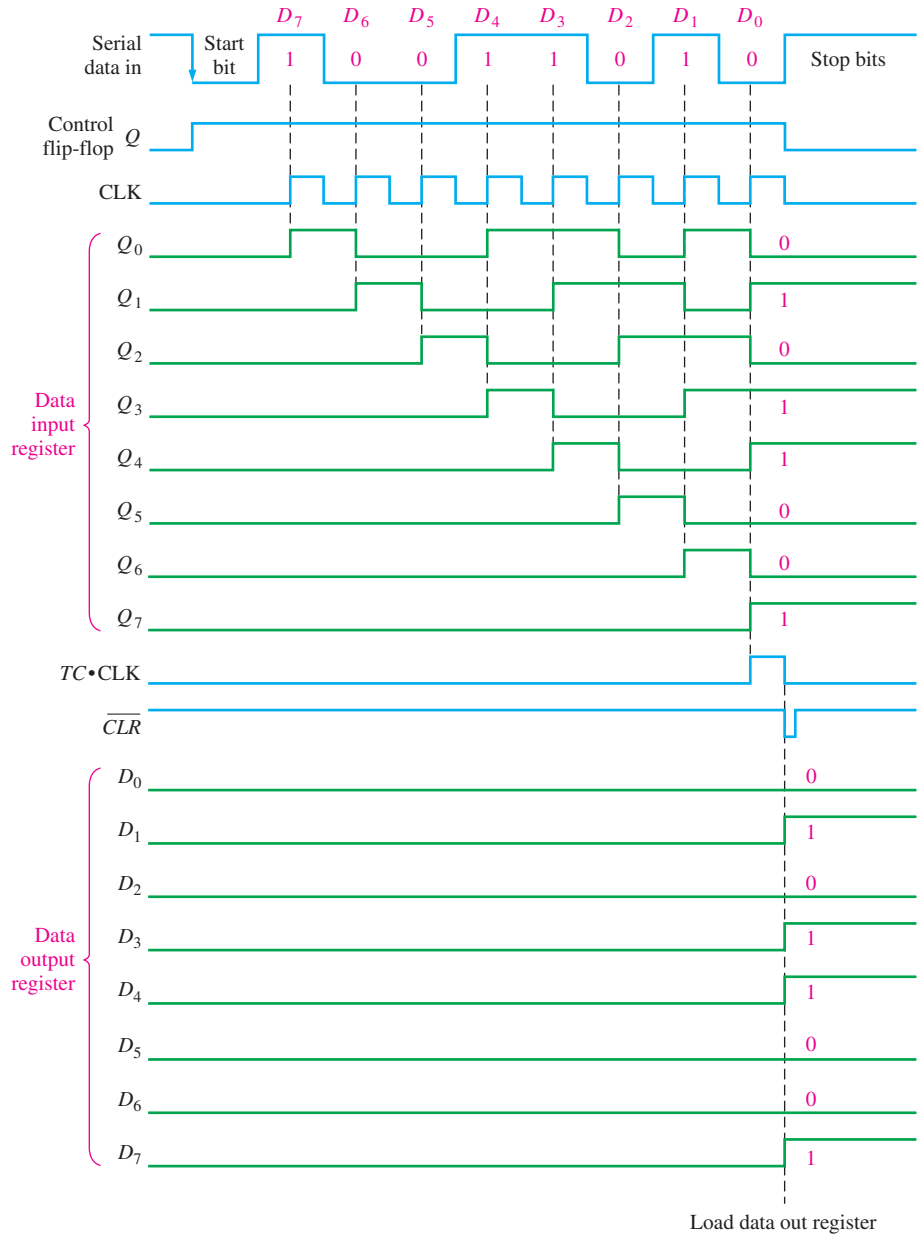


FIGURE 8–33 Timing diagram illustrating the operation of the serial-to-parallel data converter in Figure 8–31.

eighth clock pulse, the terminal count (*TC*) goes from LOW to HIGH, indicating the counter is at the last state. This rising edge is ANDed with the clock pulse, which is still HIGH, producing a rising edge at $TC \cdot \text{CLK}$. This parallel loads the eight data bits from the data-input shift register to the data-output register. A short time later, the clock pulse goes LOW and this HIGH-to-LOW transition triggers the one-shot, which produces a short-duration pulse to clear the counter and reset the control flip-flop and thus disable the clock generator. The system is now ready for the next group of eleven bits, and it waits for the next HIGH-to-LOW transition at the beginning of the start bit.

By reversing the process just stated, parallel-to-serial data conversion can be accomplished. Since the serial data format must be produced, start and stop bits must be added to the sequence.

Universal Asynchronous Receiver Transmitter (UART)

As mentioned, computers and microprocessor-based systems often send and receive data in a parallel format. Frequently, these systems must communicate with external devices that send and/or receive serial data. An interfacing device used to accomplish these conversions is the UART (Universal Asynchronous Receiver Transmitter). Figure 8–34 illustrates the UART in a general microprocessor-based system application.

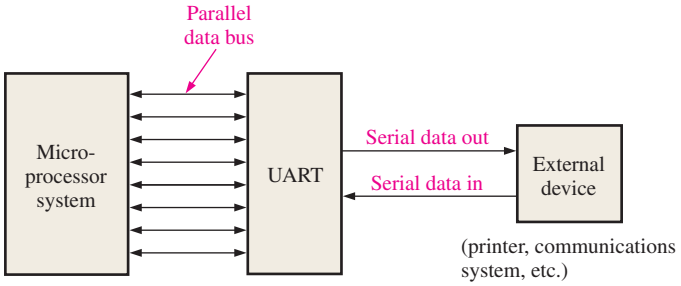


FIGURE 8–34 UART interface.

A UART includes both serial-to-parallel and parallel-to-serial conversion, as shown in the block diagram in Figure 8–35. The data bus is basically a set of parallel conductors along which data move between the UART and the microprocessor system. Buffers interface the data registers with the data bus.

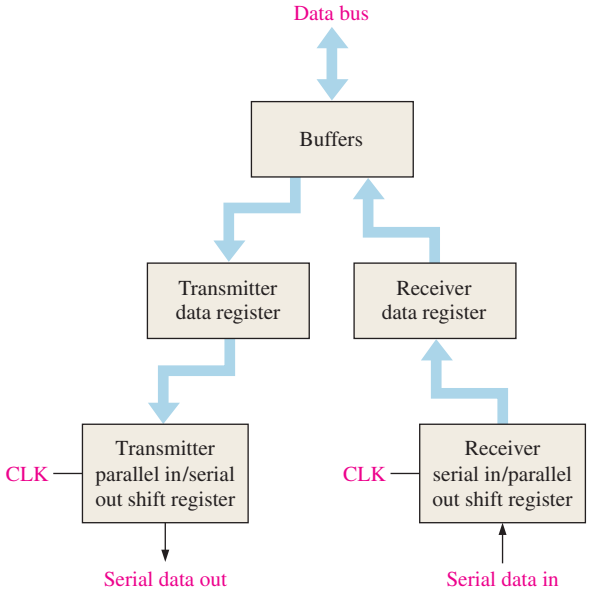


FIGURE 8–35 Basic UART block diagram.

The UART receives data in serial format, converts the data to parallel format, and places them on the data bus. The UART also accepts parallel data from the data bus, converts the data to serial format, and transmits them to an external device.

Keyboard Encoder

The keyboard encoder is a good example of the application of a shift register used as a ring counter in conjunction with other devices. Recall that a simplified computer keyboard encoder without data storage was presented in Chapter 6.

Figure 8–36 shows a simplified keyboard encoder for encoding a key closure in a 64-key matrix organized in eight rows and eight columns. Two parallel in/parallel out 4-bit shift

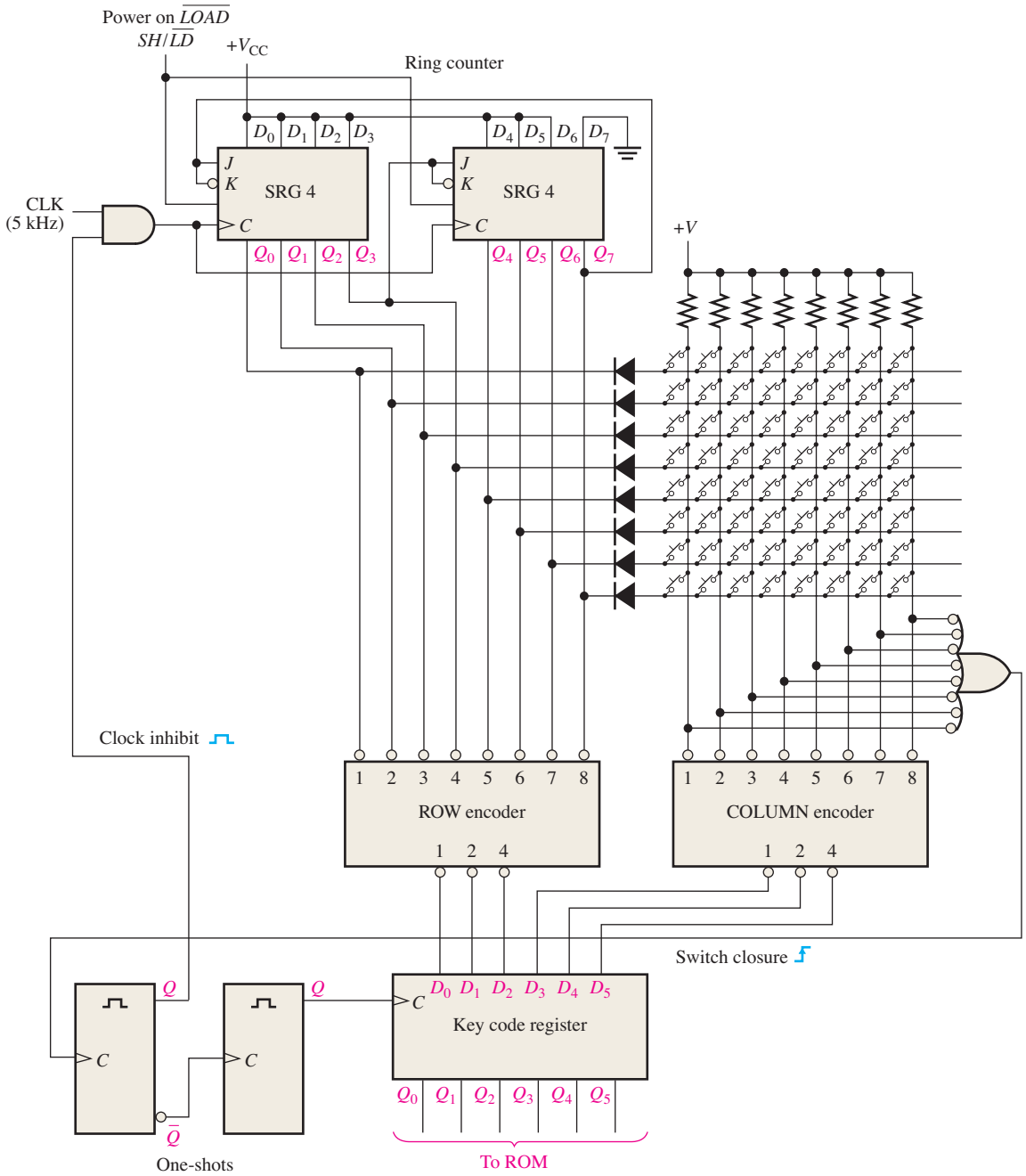


FIGURE 8–36 Simplified keyboard encoding circuit.

registers are connected as an 8-bit ring counter with a fixed bit pattern of seven 1s and one 0 preset into it when the power is turned on. Two priority encoders (introduced in Chapter 6) are used as eight-line-to-three-line encoders (9 input HIGH, 8 output unused) to encode the ROW and COLUMN lines of the keyboard matrix. A parallel in/parallel out register (key code) stores the ROW/COLUMN code from the priority encoders.

The basic operation of the keyboard encoder in Figure 8–36 is as follows: The ring counter "scans" the rows for a key closure as the clock signal shifts the 0 around the counter at a 5 kHz rate. The 0 (LOW) is sequentially applied to each ROW line, while all other ROW lines are HIGH. All the ROW lines are connected to the ROW encoder inputs, so the 3-bit output of the ROW encoder at any time is the binary representation of the ROW line that is LOW. When there is a key closure, one COLUMN line is connected to one ROW line. When the ROW line is taken LOW by the ring counter, that particular COLUMN line is also pulled LOW. The COLUMN encoder produces a binary output corresponding to the COLUMN in which the key is closed. The 3-bit ROW code plus the 3-bit COLUMN code uniquely identifies the key that is closed. This 6-bit code is applied to the inputs of the key code register. When a key is closed, the two one-shots produce a delayed clock pulse to parallel-load the 6-bit code into the key code register. This delay allows the contact bounce to die out. Also, the first one-shot output inhibits the ring counter to prevent it from scanning while the data are being loaded into the key code register.

The 6-bit code in the key code register is now applied to a ROM (read-only memory) to be converted to an appropriate alphanumeric code that identifies the keyboard character. ROMs are studied in Chapter 11.

SECTION 8–5 CHECKUP

1. In the keyboard encoder, how many times per second does the ring counter scan the keyboard?
2. What is the 6-bit ROW/COLUMN code (key code) for the top row and the left-most column in the keyboard encoder?
3. What is the purpose of the diodes in the keyboard encoder? What is the purpose of the resistors?

8–6 Logic Symbols with Dependency Notation

Two examples of ANSI/IEEE Standard 91-1984 symbols with dependency notation for shift registers are presented. Two specific IC shift registers are used as examples.

After completing this section, you should be able to

- Understand and interpret the logic symbols with dependency notation for the 74HC164 and the 74HC194 shift registers

The logic symbol for a 74HC164 8-bit serial in/parallel out shift register is shown in Figure 8–37. The common control inputs are shown on the notched block. The clear ($\overline{CLR}$) input is indicated by an *R* (for RESET) inside the block. Since there is no dependency prefix to link *R* with the clock (C1), the clear function is asynchronous. The right arrow symbol after C1 indicates data flow from Q_0 to Q_7. The *A* and *B* inputs are ANDed, as indicated by the embedded AND symbol, to provide the synchronous data input, 1*D*, to the first stage (Q_0). Note the dependency of *D* on *C*, as indicated by the 1 suffix on *C* and the 1 prefix on *D*.

Figure 8–38 is the logic symbol for the 74HC194 4-bit bidirectional universal shift register. Starting at the top left side of the control block, note that the $\overline{CLR}$ input is active-LOW and is asynchronous (no prefix link with *C*). Inputs S_0 and S_1 are mode inputs that

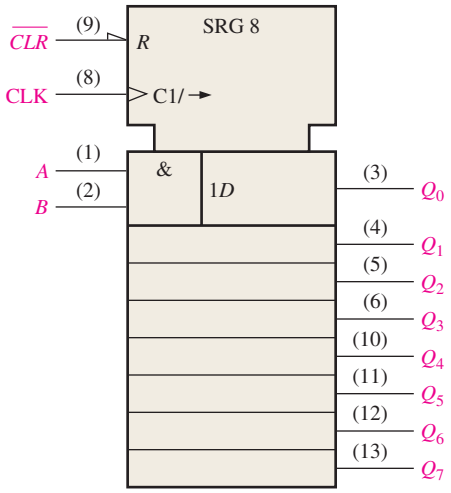


FIGURE 8–37 Logic symbol for the 74HC164.

determine the *shift-right, shift-left,* and *parallel load* modes of operation, as indicated by the $\frac{0}{3}$ dependency designation following the *M*. The $\frac{0}{3}$ represents the binary states of 0, 1, 2, and 3 on the S_0 and S_1 inputs. When one of these digits is used as a prefix for another input, a dependency is established. The $1\rightarrow/2\leftarrow$ symbol on the clock input indicates the following: $1\rightarrow$ indicates that a right shift (Q_0 toward Q_3) occurs when the mode inputs (S_0, S_1) are in the binary 1 state ($S_0 = 1$, $S_1 = 0$), $2\leftarrow$ indicates that a left shift (Q_3 toward Q_0) occurs when the mode inputs are in the binary 2 state ($S_0 = 0$, $S_1 = 1$). The shift-right serial input (*SR SER*) is both mode-dependent and clock-dependent, as indicated by 1, 4*D*. The parallel inputs (D_0, D_1, D_2, and D_3) are all mode-dependent (prefix 3 indicates parallel load mode) and clock-dependent, as indicated by 3, 4*D*. The shift-left serial input (*SL SER*) is both mode-dependent and clock-dependent, as indicated by 2, 4*D*.

The four modes for the 74HC194 are summarized as follows:

Do nothing:	$S_0 = 0, S_1 = 0$	(mode 0)
Shift right:	$S_0 = 1, S_1 = 0$	(mode 1, as in 1, 4*D*)
Shift left:	$S_0 = 0, S_1 = 1$	(mode 2, as in 2, 4*D*)
Parallel load:	$S_0 = 1, S_1 = 1$	(mode 3, as in 3, 4*D*)

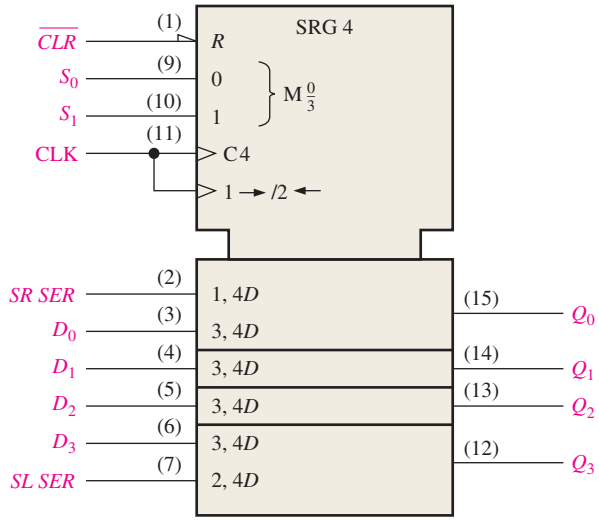
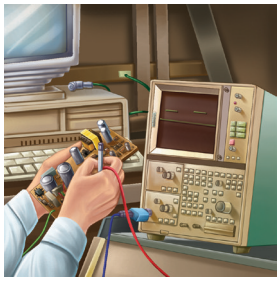


FIGURE 8–38 Logic symbol for the 74HC194.

SECTION 8–6 CHECKUP

1. In Figure 8–38, are there any inputs that are dependent on the mode inputs being in the 0 state?
2. Is the parallel load synchronous with the clock?

8–7 Troubleshooting



A traditional method of troubleshooting sequential logic and other more complex systems uses a procedure of "exercising" the circuit under test with a known input waveform (stimulus) and then observing the output for the correct bit pattern.

After completing this section, you should be able to

- Explain the procedure of "exercising" as a troubleshooting technique
- Discuss exercising of a serial-to-parallel converter

The serial-to-parallel data converter in Figure 8–31 is used to illustrate the "exercising" procedure. The main objective in exercising the circuit is to force all elements (flip-flops and gates) into all of their states to be certain that nothing is stuck in a given state as a result of a fault. The input test pattern, in this case, must be designed to force each flip-flop in the registers into both states, to clock the counter through all of its eight states, and to take the control flip-flop, the clock generator, the one-shot, and the AND gate through their paces.

The input test pattern that accomplishes this objective for the serial-to-parallel data converter is based on the serial data format in Figure 8–32. It consists of the pattern 10101010 in one serial group of data bits followed by 01010101 in the next group, as shown in Figure 8–39. These patterns are generated on a repetitive basis by a special test-pattern generator. The basic test setup is shown in Figure 8–40.

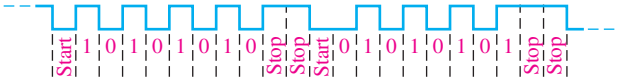


FIGURE 8–39 Sample test pattern.

After both patterns have been run through the circuit under test, all the flip-flops in the data-input register and in the data-output register have resided in both SET and RESET states, the counter has gone through its sequence (once for each bit pattern), and all the other devices have been exercised.

To check for proper operation, each of the parallel data outputs is observed for an alternating pattern of 1s and 0s as the input test patterns are repetitively shifted into the data-input register and then loaded into the data-output register. The proper timing diagram is shown in Figure 8–41. The outputs can be observed in pairs with a dual-trace oscilloscope, or all eight outputs can be observed simultaneously with a logic analyzer configured for timing analysis.

If one or more outputs of the data-output register are incorrect, then you must back up to the outputs of the data-input register. If these outputs are correct, then the problem is associated with the data-output register. Check the inputs to the data-output register directly on the pins of the IC for an open input line. Check that power and ground are correct (look for the absence of noise on the ground line). Verify that the load line is a solid LOW and that there are clock pulses on the clock input of the correct amplitude. Make

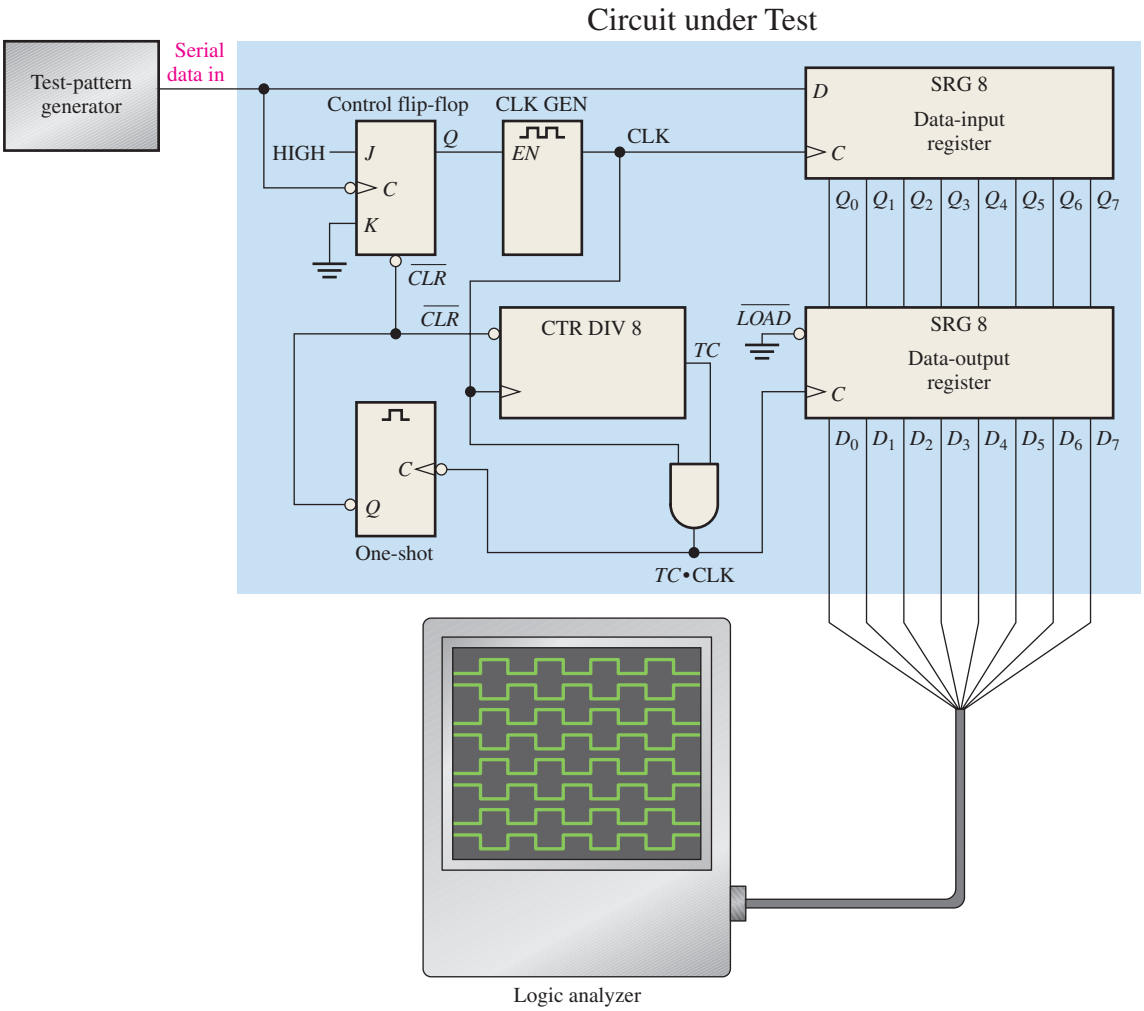


FIGURE 8–40 Basic test setup for the serial-to-parallel data converter of Figure 8–31.

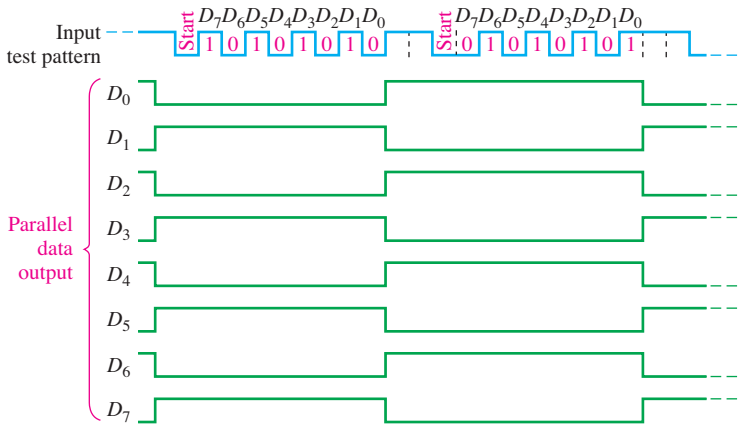


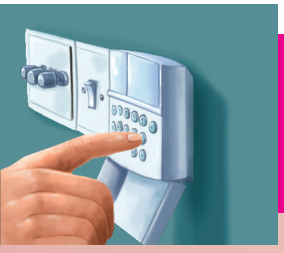
FIGURE 8–41 Proper outputs for the circuit under test in Figure 8–40. The input test pattern is shown.

sure that the connection to the logic analyzer did not short two output lines together. If all of these checks pass inspection, then it is likely that the output register is defective. If the data-input register outputs are also incorrect, the fault could be associated with the input register itself or with any of the other logic, and additional investigation is necessary to isolate the problem.

When measuring digital signals with an oscilloscope, you should always use dc coupling, rather than ac coupling. The reason that ac coupling is not best for viewing digital signals is that the 0 V level of the signal will appear at the *average* level of the signal, not at true ground or 0 V level. It is much easier to find a "floating" ground or incorrect logic level with dc coupling. If you suspect an open ground in a digital circuit, increase the sensitivity of the scope to the maximum possible. A good ground will never appear to have noise under this condition, but an open will likely show some noise, which appears as a random fluctuation in the 0 V level.

SECTION 8–7 CHECKUP

1. What is the purpose of providing a test input to a sequential logic circuit?
2. Generally, when an output waveform is found to be incorrect, what is the next step to be taken?



Applied Logic

Security System

A security system that provides coded access to a secured area is developed. Once a 4-digit security code is stored in the system, access is achieved by entering the correct code on a keypad. A block diagram for the security system is shown in Figure 8–42. The system consists of the security code logic, the code-selection logic, and the keypad. The keypad is a standard numeric keypad.

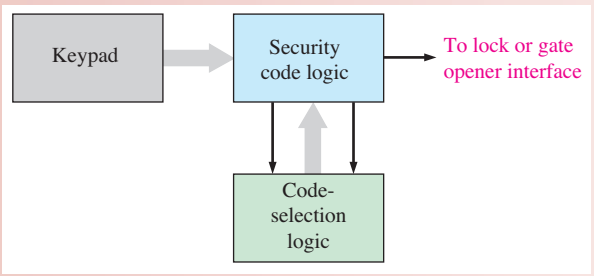


FIGURE 8–42 Block diagram of the security system.

Basic Operation

A 4-digit entry code is set into the memory with user-accessible DIP switches. Initially pressing the # key sets up the system for the first digit in the code. For entry, the code is entered one digit at a time on the keypad and converted to a BCD code for processing by the security code logic. If the entered code agrees with the stored code, the output activates the access mechanism and allows the door or gate, depending on the type of area that is secured, to be opened.

Exercise

1. Write the BCD code sequence produced by the code generator if the 4-digit access number 4739 is entered on the keypad.

The Security Code Logic

The security code logic compares the code entered on the keypad with the predetermined code from the code-selection logic. A logic diagram of the security code logic is shown in Figure 8–43.

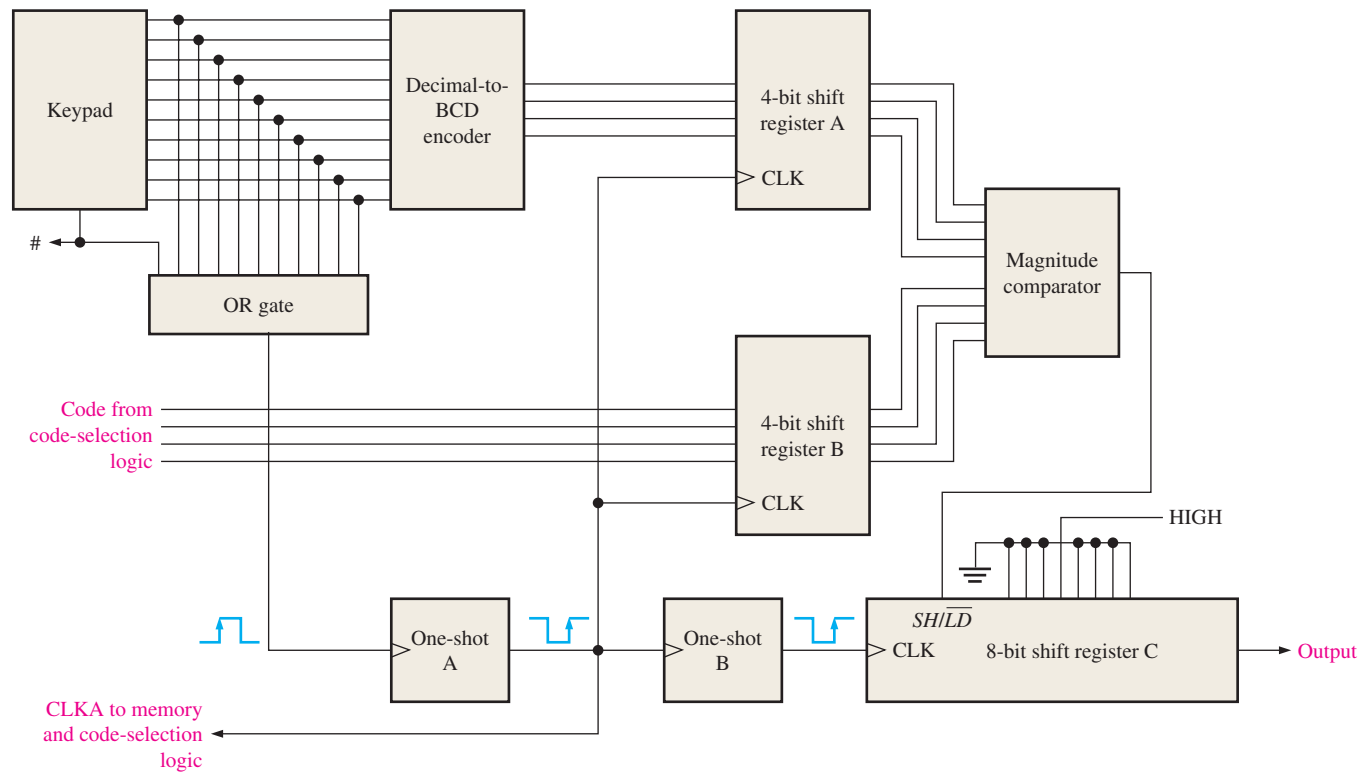


FIGURE 8–43 Block diagram of the security code logic with keypad.

In order to gain entry, first the # key on the keypad is pressed to trigger the one-shots, thus initializing the 8-bit register C with a preset pattern (00010000). Next the four digits of the code are entered in proper sequence on the keypad. As each digit is entered, it is converted to BCD by the decimal-to-BCD encoder, and a clock pulse is produced by one-shot A that shifts the 4-bit code into register A. The one-shot is triggered by a transition on the output of the OR gate when a key is pressed. At the same time, the corresponding digit from the code generator is shifted into register B. Also, one-shot B is triggered after one-shot A to provide a delayed clock pulse for register C to serially shift the preloaded pattern (00010000). The left-most three 0s are simply "fillers" and serve no purpose in the operation of the system. The outputs of registers A and B are applied to the comparator; if the codes are the same, the output of the comparator goes HIGH, placing shift register C in the SHIFT mode.

Each time an entered digit agrees with the preset digit, the 1 in shift register C is shifted right one position. On the fourth code agreement, the 1 appears on the output of the shift register and activates the mechanism to unlock the door or open the gate. If the code digits do not agree, the output of the comparator goes LOW, placing shift register C in the LOAD mode so the shift register is reinitialized to the preset pattern (00010000).

Exercise

2. What is the state of shift register C after two correct code digits are entered?
3. Explain the purpose of the OR gate.
4. If the digit 4 is entered on the keypad, what appears on the outputs of register A?

The Code-Selection Logic

A logic diagram of the code-selection logic is shown in Figure 8–44. This part of the system includes a set of DIP switches into which a 4-digit entry code is set. Initially pressing the # key sets up the system for the first digit in the code by causing a preset pattern to be loaded into the 4-bit shift register (0001). The four bits in the first code digit are selected by a HIGH on the Q_0 output of the shift register, enabling the four AND gates labeled A1–A4. As each digit of the code is entered on the keypad, the clock from the security code logic shifts the 1 in the shift register to sequentially enable each set of four AND gates. As a result, the BCD digits in the security code appear sequentially on the outputs.

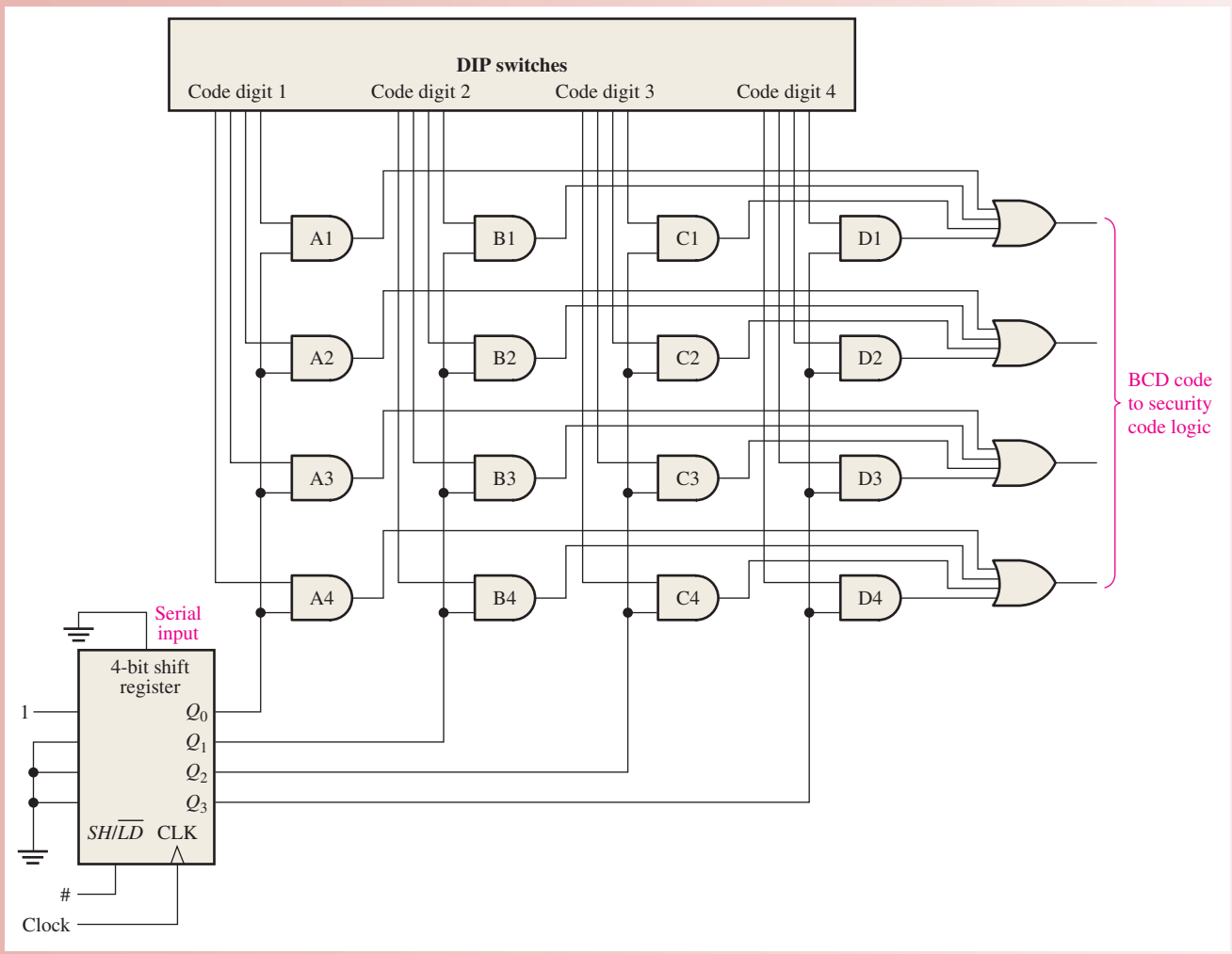


FIGURE 8–44 Logic diagram of the code-selection logic.

Security System with VHDL

The security system can be described using VHDL for implementation in a PLD. The three blocks of the system (keypad, security code logic, and code-selection logic) are combined in the program code to describe the complete system.

The security system block diagram is shown in Figure 8–45 as a programming model. Six program components perform the logical operations of the security system. Each component corresponds to a block or blocks in the figure. The security system program SecuritySystem contains the code that defines how the components interact.

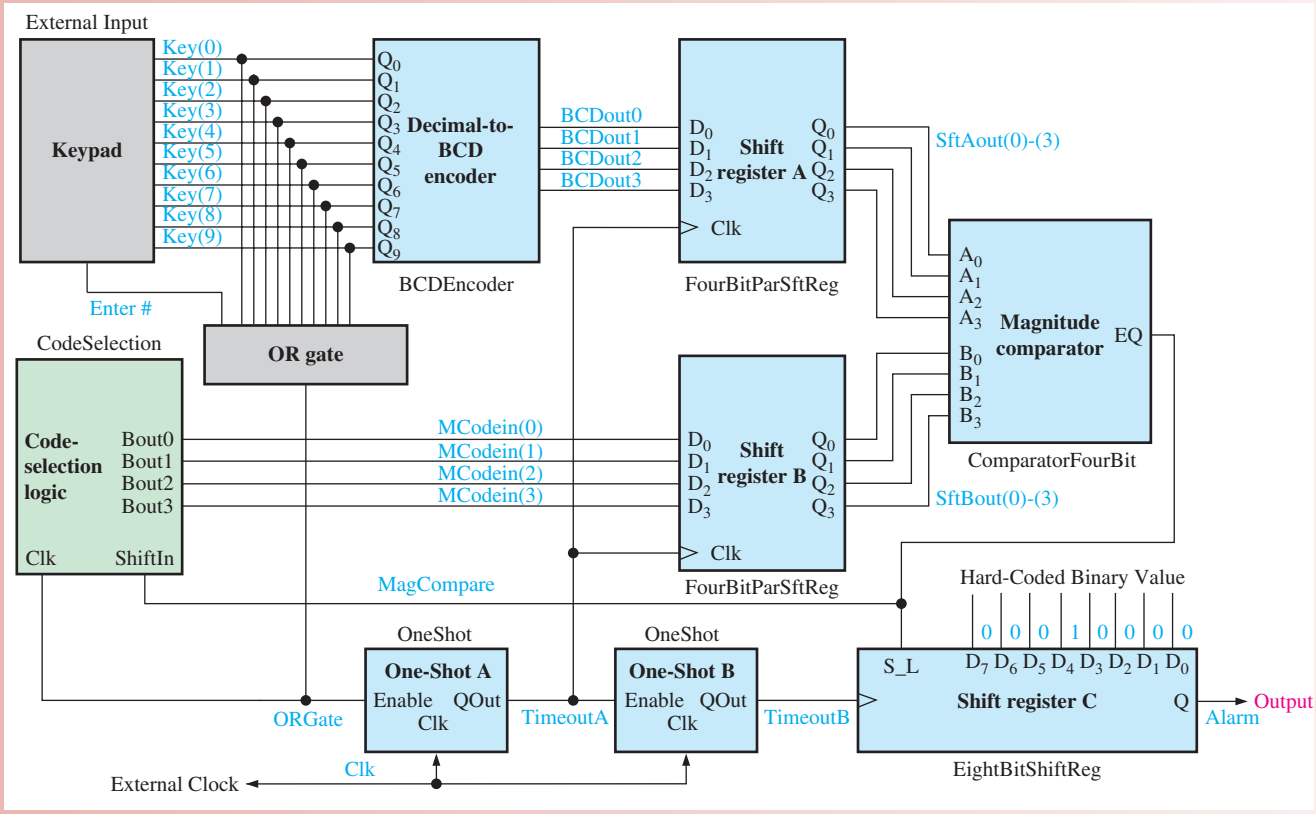


FIGURE 8–45 Security system block diagram as a programming model.

The security system includes a ten-bit input vector Key—one input bit for each decimal digit—and an input Enter, representing a typical numeric keypad. Once a key is pressed, the data stored in input array Key are sent to the decimal-to-BCD encoder (BCDEncoder). Its 4-bit output is then sent to the inputs of the 4-bit parallel in/parallel out shift register A (FourBitParSftReg). An external system clock applied to input Clk drives the overall security system. The Alarm output signal is set HIGH upon a successful arming operation.

Pressing the Enter key sends an initial HIGH clock signal to the code-selection logic block (CodeSelection), which loads an initial binary value of 1000 to shift register B. At this time, a binary 0000 is stored in shift register A, and the output of the magnitude comparator (ComparatorFourBit) is set LOW. The code-selection logic is now ready to present the first stored code value that is to be compared to the value of the first numeric keypad entry. At this time a LOW on the 8-bit parallel in/serial out shift register C (EightBitShiftReg) S_L input loads an initial value of 00010000.

When a numeric key is pressed, the output of the OR gate (ORGate) clocks the first stored value to the inputs of shift register B, and the output of the decimal-to-BCD encoder is sent to the inputs of shift register A. If the values in shift registers A and B match, the output of the magnitude comparator is set HIGH; and the code-selection logic is ready to clock in the next stored code value.

At the conclusion of four successful comparisons of stored code values against four correct keypad entries, the value 00010000 initially in shift register C will shift four places to the right, setting the Alarm output to a HIGH. An incorrect keypad entry will not match the stored code value in shift register B and the magnitude comparator will output a LOW. With the comparator output LOW, the code-selection logic will reset to the first stored code value; and the value 00010000 is reloaded into shift register C, starting the process over again.

To clock the keypad and the stored code values through the system, two one-shots (OneShot) are used. The one-shots allow data to stabilize before any action is taken. One-shot A receives an Enable signal from the keypad OR gate, which initiates the first timed process. The OR gate output is also sent to the code-selection logic, and the first code value from the code-selection logic is sent to the inputs of shift register A. When one-shot A times out, the selected keypad entry and the current code from the code-selection logic are stored in shift registers A and B for comparison by the magnitude comparator, and an Enable is sent to one-shot B. If the codes in shift registers A and B match, the value stored in shift register C shifts one place to the right after one-shot B times out.

The six components used in the security system program SecuritySystem are shown in Figure 8–46.

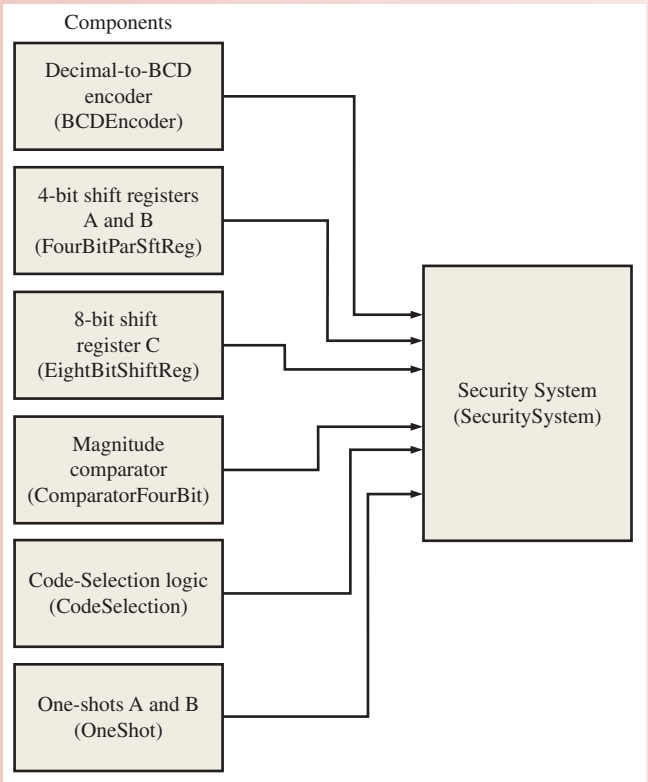


FIGURE 8–46 Security system components.

The VHDL program code for the security system is as follows:

```
library ieee;
use ieee.std_logic_1164.all;
```
Key : 10 - Key input
Enter : # - Key input
Clk : System clock
Alarm : Alarm output

```
entity SecuritySystem is
port (key: in std_logic_vector(0 to 9); Enter: in std_logic;
      Clk: in std_logic; Alarm: out std_logic);
end entity SecuritySystem;

architecture SecuritySystemBehavior of SecuritySystem is

component BCDEncoder is
port(D: in std_logic_vector(0 to 9);
     Q: out std_logic_vector(0 to 3));
end component BCDEncoder;
```
Component declaration for BCDEncoder

```
component FourBitParSftReg is
port(D: in std_logic_vector(0 to 3);
     Clk: in std_logic;
     Q: out std_logic_vector(0 to 3));
end component FourBitParSftReg;
```
Component declaration for FourBitParSftReg

```
component ComparatorFourBit is
port(A, B: in std_logic_vector(0 to 3);
     EQ: out std_logic);
end component ComparatorFourBit;
```
Component declaration for ComparatorFourBit

```
component OneShot is
port(Enable, Clk: in std_logic;
     QOut: buffer std_logic);
end component OneShot;
```
Component declaration for OneShot

```
component EightBitShiftReg is
port(S_L, Clk: in std_logic;
     D: in std_logic_vector(0 to 7);
     Q: buffer std_logic);
end component EightBitShiftReg;
```
Component declaration for EightBitShiftReg

```
component CodeSelection is
port(ShiftIn, Clk: in std_logic;
     Bout: out std_logic_vector(1 to 4));
end component CodeSelection;
```
Component declaration for CodeSelection

```
signal BCDout: std_logic_vector(0 to 3);
signal SftAout: std_logic_vector(0 to 3);
signal SftBout: std_logic_vector(0 to 3);
signal MCodein: std_logic_vector(0 to 3);
signal ORgate: std_logic;
signal MagCompare: std_logic;
signal TimeoutA, TimeoutB: std_logic;
```
BDCout: BCD encoder return
SftAout: Shift Register A return
SftBout: Shift Register B return
MCodein: Security Code value
ORgate: OR output from 10-keypad
MagCompare: Key entry to code compare
TimeoutA/B: One-shot timer variables

```
begin
ORgate <= (Key(0) or Key(1) or Key(2) or Key(3) or Key(4)
          or key(5) or Key(6) or Key(7) or Key(8) or Key(9));
```

Logic definition for ORGate

```
BCD: BCDEncoder
port map(D(0)=>Key(0),D(1)=>Key(1),D(2)=>Key(2),D(3)=>Key(3),
         D(4)=>Key(4),D(5)=>Key(5),D(6)=>Key(6),D(7)=>Key(7),D(8)=>Key(8),D(9)=>Key(9),
         Q(0)=>BCDout(0),Q(1)=>BCDout(1),Q(2)=>BCDout(2),Q(3)=>BCDout(3));
ShiftRegisterA: FourBitParSftReg
port map(D(0)=>BCDout(0),D(1)=>BCDout(1),D(2)=>BCDout(2),D(3)=>BCDout(3),
          Clk=>not TimeoutA,Q(0)=>SftAout(0),Q(1)=>SftAout(1),Q(2)=>SftAout(2),Q(3)=>SftAout(3));
ShiftRegisterB: FourBitParSftReg
port map(D(0)=>MCodein(0),D(1)=>MCodein(1),D(2)=>MCodein(2),D(3)=>MCodein(3),
        Clk=>not TimeoutA,Q(0)=>SftBout(0),Q(1)=>SftBout(1),Q(2)=>SftBout(2),Q(3)=>SftBout(3));
Magnitude Comparator: ComparatorFourBit port map(A=>SftAout,B=>SftBout,EQ=>MagCompare);
OSA:OneShot port map(Enable=>Enter or ORgate,Clk=>Clk,QOut=>TimeoutA);
OSB:OneShot port map(Enable=>not TimeoutA,Clk=>Clk,QOut=>TimeoutB);
ShiftRegisterC:EightBitShiftReg
port map(S_L=>MagCompare,Clk=> TimeoutB,D(0)=>'0',D(1)=>'0',
        D(2)=>'0',D(3)=>'1',D(4)=>'0',D(5)=>'0',D(6)=>'0',D(7)=>'0',Q=>Alarm);
CodeSelectionA: CodeSelection
port map(ShiftIn=>MagCompare,Clk=>Enter or ORGate,Bout=>MCodein);
end architecture SecuritySystemBehavior;
```

Component instantiations

Simulation

MultiSim

Open File AL08 in the Applied Logic folder on the website. Run the security code logic simulation using your Multisim software and observe the operation. A DIP switch is used to simulate the 10-digit keypad and switch J1 simulates the # key. Switches J2–J5 are used for test purposes to enter the code that is produced by the code selection logic in the complete system. Probe lights are used only for test purposes to indicate the states of registers A and B, the output of the comparator, and the output of register C.

Putting Your Knowledge to Work

Explain how the security code logic can be modified to accommodate a 5-digit code.

SUMMARY

- The basic types of data movement in shift registers are
  1. Serial in/shift right/serial out
  2. Serial in/shift left/serial out
  3. Parallel in/serial out
  4. Serial in/parallel out

5. Parallel in/parallel out
6. Rotate right
7. Rotate left

- Shift register counters are shift registers with feedback that exhibit special sequences. Examples are the Johnson counter and the ring counter.
- The Johnson counter has $2n$ states in its sequence, where n is the number of stages.
- The ring counter has n states in its sequence.

KEY TERMS

Key terms and other bold terms in the chapter are defined in the end-of-book glossary.

Bidirectional Having two directions. In a bidirectional shift register, the stored data can be shifted right or left.

Load To enter data into a shift register.

Register One or more flip-flops used to store and shift data.

Stage One storage element in a register.

TRUE/FALSE QUIZ

Answers are at the end of the chapter.

1. Shift registers consist of an arrangement of flip-flops.
2. A shift register cannot be used to store data.
3. A serial shift register accepts one bit at a time on a single line.
4. All shift registers are defined by specified sequences.
5. A shift register counter is a shift register with the serial output connected back to the serial input.
6. A shift register with four stages can store a maximum count of fifteen.
7. The Johnson counter is a special type of shift register.
8. The modulus of an 8-bit Johnson counter is eight.
9. A ring counter uses one flip-flop for each state in its sequence.
10. A shift register cannot be used as a time delay device.

SELF-TEST

Answers are at the end of the chapter.

1. A register's functions include
   (a) data storage **(b)** data movement
   (c) neither (a) not (b) **(d)** both (a) and (b)
2. To enter a byte of data serially into an 8-bit shift register, there must be
   (a) one clock pulse **(b)** two clock pulses
   (c) four clock pulses **(d)** eight clock pulses
3. To parallel load a byte of data into a shift register with a synchronous load, there must be
   (a) one clock pulse **(b)** one clock pulse for each 1 in the data
   (c) eight clock pulses **(d)** one clock pulse for each 0 in the data
4. The group of bits 10110101 is serially shifted (right-most bit first) into an 8-bit parallel output shift register with an initial state of 11100100. After two clock pulses, the register contains
   (a) 01011110 **(b)** 10110101
   (c) 01111001 **(d)** 00101101

5. With a 100 kHz clock frequency, eight bits can be serially entered into a shift register in
(a) 80 μs **(b)** 8 μs
(c) 80 ms **(d)** 10 μs

6. With a 1 MHz clock frequency, eight bits can be parallel entered into a shift register
(a) in 8 μs
(b) in the propagation delay time of eight flip-flops
(c) in 1 μs
(d) in the propagation delay time of one flip-flop

7. A modulus-8 Johnson counter requires
(a) eight flip-flops **(b)** four flip-flops
(c) five flip-flops **(d)** twelve flip-flops

8. A modulus-8 ring counter requires
(a) eight flip-flops **(b)** four flip-flops
(c) five flip-flops **(d)** twelve flip-flops

9. When an 8-bit serial in/serial out shift register is used for a 24 μs time delay, the clock frequency must be
(a) 41.67 kHz **(b)** 333 kHz
(c) 125 kHz **(d)** 8 MHz

10. The purpose of the ring counter in the keyboard encoding circuit of Figure 8–36 is
(a) to sequentially apply a HIGH to each row for detection of key closure
(b) to provide trigger pulses for the key code register
(c) to sequentially apply a LOW to each row for detection of key closure
(d) to sequentially reverse bias the diodes in each row

PROBLEMS

Answers to odd-numbered problems are at the end of the book.

Section 8–1 Shift Register Operations

1. What is a register?

2. What is the storage capacity of a register that can retain one byte of data?

3. What does the "shift capacity" of a register mean?

Section 8–2 Types of Shift Register Data I/Os

4. The sequence 1011 is applied to the input of a 4-bit serial shift register that is initially cleared. What is the state of the shift register after three clock pulses?

5. For the data input and clock in Figure 8–47, determine the states of each flip-flop in the shift register of Figure 8–3 and show the Q waveforms. Assume that the register contains all 1s initially.

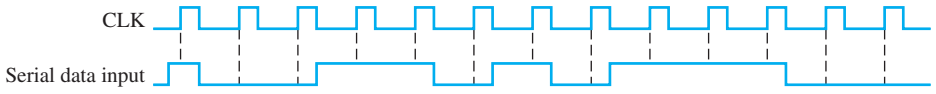


FIGURE 8–47

6. Solve Problem 5 for the waveforms in Figure 8–48.

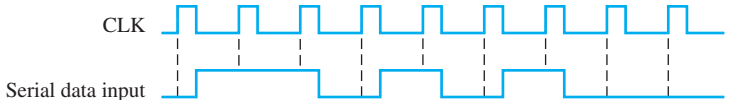


FIGURE 8–48

7. What is the state of the register in Figure 8–49 after each clock pulse if it starts in the 101001111000 state?

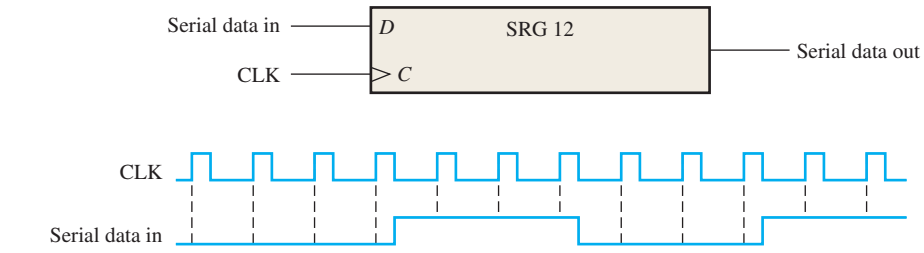


FIGURE 8–49

8. For the serial in/serial out shift register, determine the data-output waveform for the data-input and clock waveforms in Figure 8–50. Assume that the register is initially cleared.

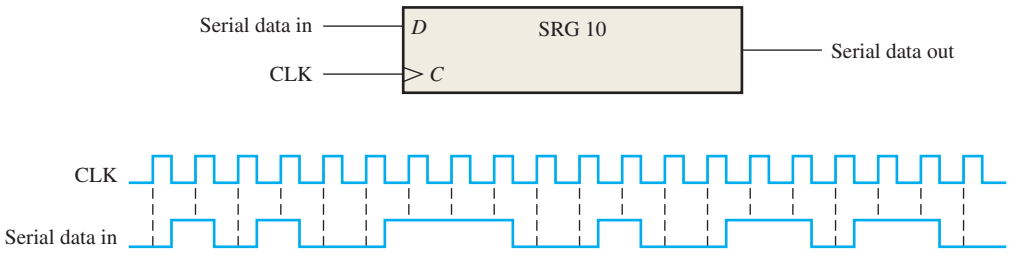


FIGURE 8–50

9. Solve Problem 8 for the waveforms in Figure 8–51.

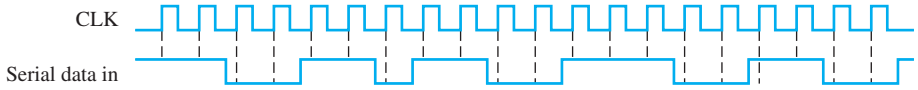


FIGURE 8–51

10. A leading-edge clocked serial in/serial out shift register has a data-output waveform as shown in Figure 8–52. What binary number is stored in the 8-bit register if the first data bit out (left-most) is the LSB?

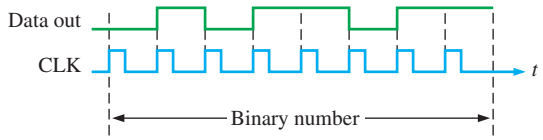


FIGURE 8–52

11. Show a complete timing diagram including the parallel outputs for the shift register in Figure 8–6. Use the waveforms in Figure 8–50 with the register initially clear.

12. Solve Problem 11 for the input waveforms in Figure 8–51.

13. Develop the Q_0 through Q_7 outputs for a 74HC164 shift register with the input waveforms shown in Figure 8–53.

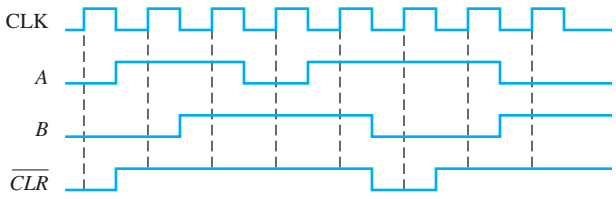


FIGURE 8–53

14. The shift register in Figure 8–54(a) has *SHIFT*/$\overline{LOAD}$ and CLK inputs as shown in part (b). The serial data input (*SER*) is a 0. The parallel data inputs are $D_0 = 1$, $D_1 = 0$, $D_2 = 1$, and $D_3 = 0$ as shown. Develop the data-output waveform in relation to the inputs.

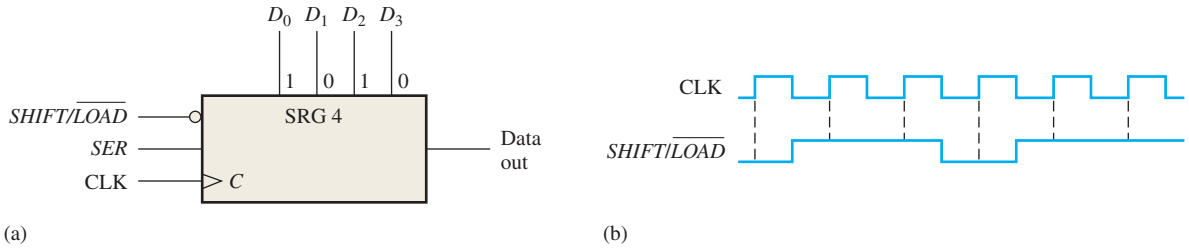


FIGURE 8–54

15. The waveforms in Figure 8–55 are applied to a 74HC165 shift register. The parallel inputs are all 0. Determine the Q_7 waveform.

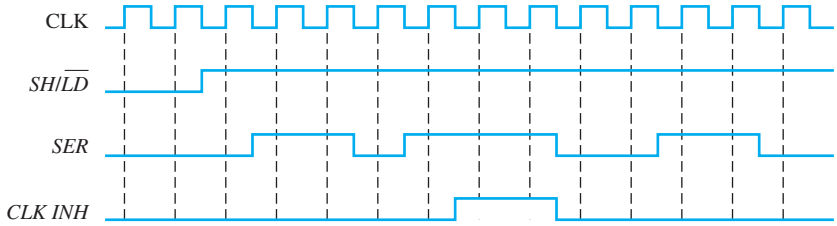


FIGURE 8–55

16. Solve Problem 15 if the parallel inputs are all 1.

17. Solve Problem 15 if the *SER* input is inverted.

18. Determine all the *Q* output waveforms for a 74HC195 4-bit shift register when the inputs are as shown in Figure 8–56.

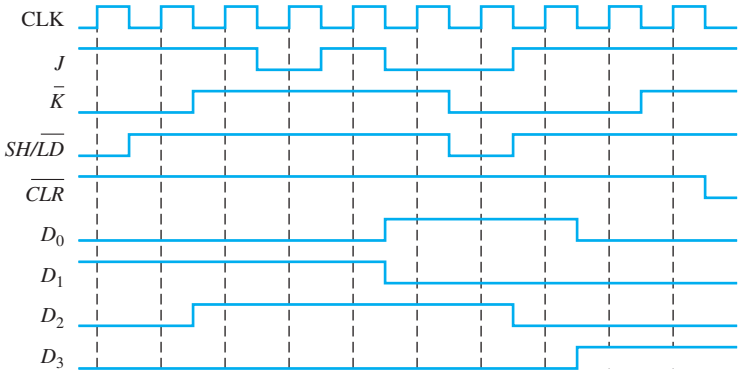


FIGURE 8–56

19. Solve Problem 18 if the *SH*/$\overline{LD}$ input is inverted and the register is initially clear.

20. Use two 74HC195 shift registers to form an 8-bit shift register. Show the required connections.

Section 8–3 Bidirectional Shift Registers

21. For the 8-bit bidirectional register in Figure 8–57, determine the state of the register after each clock pulse for the *RIGHT*/$\overline{LEFT}$ control waveform given. A HIGH on this input enables a shift to the right, and a LOW enables a shift to the left. Assume that the register is initially storing

the decimal number seventy-six in binary, with the right-most position being the LSB. There is a LOW on the data-input line.

FIGURE 8–57

22. Solve Problem 21 for the waveforms in Figure 8–58.

FIGURE 8–58

23. Use two 74HC194 4-bit bidirectional shift registers to create an 8-bit bidirectional shift register. Show the connections.

24. Determine the Q outputs of a 74HC194 with the inputs shown in Figure 8–59. Inputs D_0, D_1, D_2, and D_3 are all HIGH.

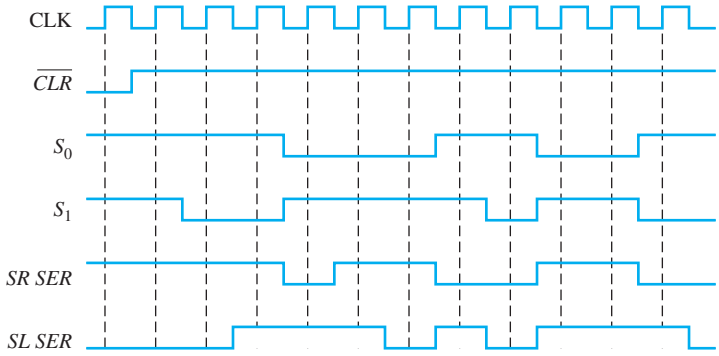


FIGURE 8–59

Section 8–4 Shift Register Counters

25. How many flip-flops are required to implement each of the following in a Johnson counter configuration:

(a) modulus-4
(b) modulus-8
(c) modulus-12
(d) modulus-18

26. Draw the logic diagram for a modulus-18 Johnson counter. Show the timing diagram and write the sequence in tabular form.

27. For the ring counter in Figure 8–60, show the waveforms for each flip-flop output with respect to the clock. Assume that FF0 is initially SET and that the rest are RESET. Show at least ten clock pulses.

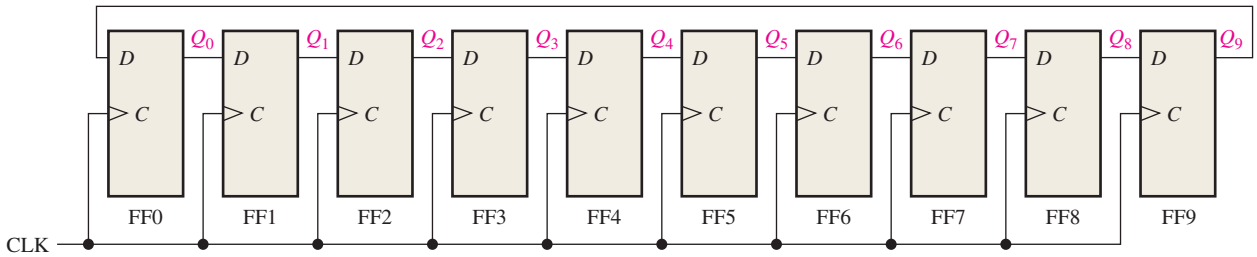


FIGURE 8–60

28. The waveform pattern in Figure 8–61 is required. Devise a ring counter, and indicate how it can be preset to produce this waveform on its Q_9 output. At CLK16 the pattern begins to repeat.

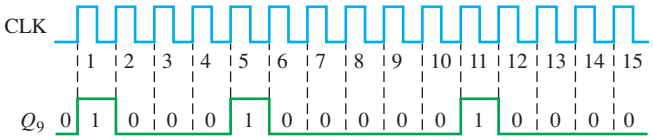


FIGURE 8–61

Section 8–5 Shift Register Applications

29. Use 74HC195 4-bit shift registers to implement a 16-bit ring counter. Show the connections.

30. What is the purpose of the power-on $\overline{LOAD}$ input in Figure 8–36?

31. What happens when two keys are pressed simultaneously in Figure 8–36?

Section 8–7 Troubleshooting

32. Based on the waveforms in Figure 8–62(a), determine the most likely problem with the register in part (b) of the figure.

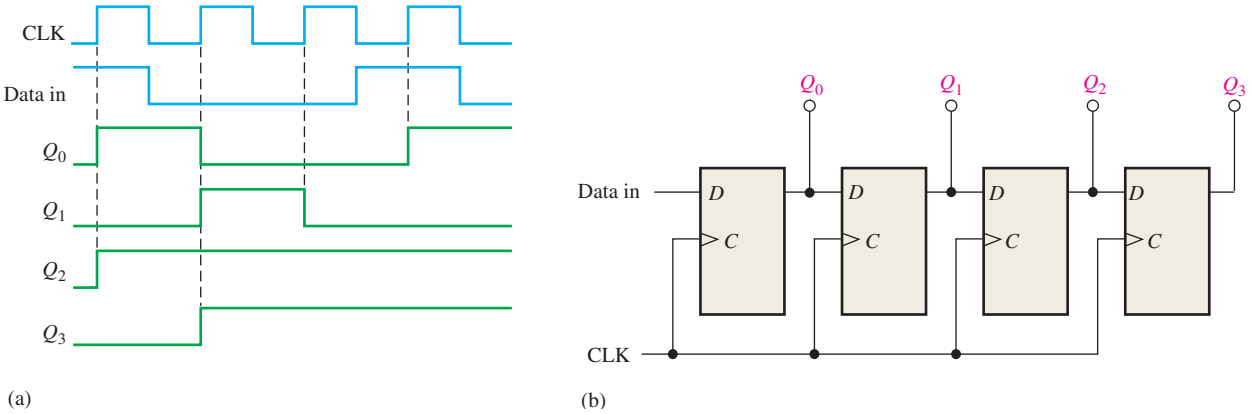


FIGURE 8–62

33. Refer to the parallel in/serial out shift register in Figure 8–10. The register is in the state where $Q_0Q_1Q_2Q_3 = 1001$, and $D_0D_1D_2D_3 = 1010$ is loaded in. When the $SHIFT/\overline{LOAD}$ input is taken HIGH, the data shown in Figure 8–63 are shifted out. Is this operation correct? If not, what is the most likely problem?

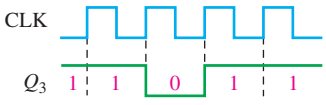


FIGURE 8–63

34. You have found that the bidirectional register in Figure 8–17 will shift data right but not left. What is the most likely fault?

35. For the keyboard encoder in Figure 8–36, list the possible faults for each of the following symptoms:

(a) The state of the key code register does not change for any key closure.
(b) The state of the key code register does not change when any key in the third row is closed. A proper code occurs for all other key closures.
(c) The state of the key code register does not change when any key in the first column is closed. A proper code occurs for all other key closures.
(d) When any key in the second column is closed, the left three bits of the key code ($Q_0Q_1Q_2$) are correct, but the right three bits are all 1s.

36. Develop a test procedure for exercising the keyboard encoder in Figure 8–36. Specify the procedure on a step-by-step basis, indicating the output code from the key code register that should be observed at each step in the test.

37. What symptoms are observed for the following failures in the serial-to-parallel converter in Figure 8–31:

(a) AND gate output stuck in HIGH state
(b) clock generator output stuck in LOW state
(c) third stage of data-input register stuck in SET state
(d) terminal count output of counter stuck in HIGH state

Applied Logic

38. What is the major purpose of the security code logic?

39. Assume the entry code is 1939. Determine the states of shift register A and shift register C after the second correct digit has been entered in Figure 8–43.

40. Assume the entry code is 7646 and the digits 7645 are entered. Determine the states of shift register A and shift register C after each of the digits is entered.

Special Design Problems

41. Specify the devices that can be used to implement the serial-to-parallel data converter in Figure 8–31. Develop the complete logic diagram, showing any modifications necessary to accommodate the specific devices used.

42. Modify the serial-to-parallel converter in Figure 8–31 to provide 16-bit conversion.

43. Design an 8-bit parallel-to-serial data converter that produces the data format in Figure 8–32. Show a logic diagram and specify the devices.

44. Design a power-on $\overline{LOAD}$ circuit for the keyboard encoder in Figure 8–36. This circuit must generate a short-duration LOW pulse when the power switch is turned on.

45. Implement the test-pattern generator used in Figure 8–40 to troubleshoot the serial-to-parallel converter.

46. Review the tablet-bottling system that was introduced in Chapter 1. Utilizing the knowledge gained in this chapter, implement registers A and B in that system using specific fixed-function IC devices.

Multisim Troubleshooting Practice

MultiSim

47. Open file P08-47. For the specified fault, predict the effect on the circuit. Then introduce the fault and verify whether your prediction is correct.

48. Open file P08-48. For the specified fault, predict the effect on the circuit. Then introduce the fault and verify whether your prediction is correct.

49. Open file P08-49. For the specified fault, predict the effect on the circuit. Then introduce the fault and verify whether your prediction is correct.

50. Open file P08-50. For the observed behavior indicated, predict the fault in the circuit. Then introduce the suspected fault and verify whether your prediction is correct.

51. Open file P08-51. For the observed behavior indicated, predict the fault in the circuit. Then introduce the suspected fault and verify whether your prediction is correct.

ANSWERS

SECTION CHECKUPS

Section 8–1 Shift Register Operations

1. The number of stages.

2. Storage and data movement are two functions of a shift register.

Section 8–2 Types of Shift Register Data I/Os

1. FF0: data input to J_0, $\overline{\text{data input}}$ to K_0; FF1: Q_0 to J_1, $\overline{Q}_0$ to K_1; FF2: Q_1 to J_2, $\overline{Q}_1$ to K_2; FF3: Q_2 to J_3, $\overline{Q}_2$ to K_3
2. Eight clock pulses
3. 0100 after 2 clock pulses
4. Take the serial output from the right-most flip-flop for serial out operation.
5. When $SHIFT/\overline{LOAD}$ is HIGH, the data are shifted right one bit per clock pulse. When $SHIFT/\overline{LOAD}$ is LOW, the data on the parallel inputs are loaded into the register.
6. The parallel load operation is asynchronous, so it is not dependent on the clock.
7. The data outputs are 1001.
8. $Q_0 = 1$ after one clock pulse

Section 8–3 Bidirectional Shift Registers

1. 1111 after the fifth clock pulse

Section 8–4 Shift Register Counters

1. Sixteen states are in an 8-bit Johnson counter sequence.
2. For a 3-bit Johnson counter: 000, 100, 110, 111, 011, 001, 000

Section 8–5 Shift Register Applications

1. 625 scans/second
2. $Q_5Q_4Q_3Q_2Q_1Q_0 = 011011$
3. The diodes provide unidirectional paths for pulling the ROWs LOW and preventing HIGHs on the ROW lines from being connected to the switch matrix. The resistors pull the COLUMN lines HIGH.

Section 8–6 Logic Symbols with Dependency Notation

1. No inputs are dependent on the mode inputs being in the 0 state.
2. Yes, the parallel load is synchronous with the clock as indicated by the 4*D* label.

Section 8–7 Troubleshooting

1. A test input is used to sequence the circuit through all of its states.
2. Check the input to that portion of the circuit. If the signal on that input is correct, the fault is isolated to the circuitry between the good input and the bad output.

RELATED PROBLEMS FOR EXAMPLES

8–1 See Figure 8–64.

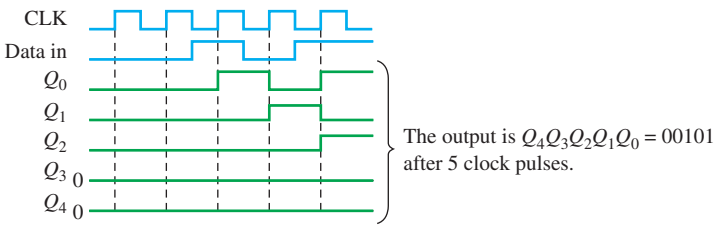


FIGURE 8–64

8–2 The state of the register after three additional clock pulses is 0000.

8–3 See Figure 8–65.

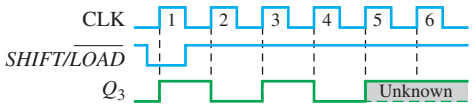


FIGURE 8–65

8–4 See Figure 8–66.

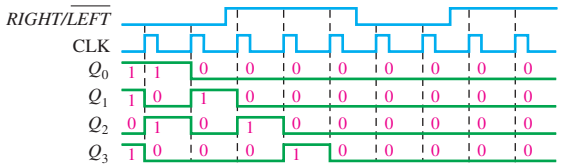


FIGURE 8–66

8–5 See Figure 8–67.

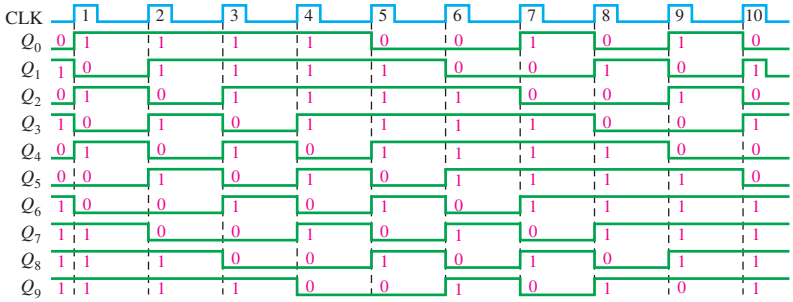


FIGURE 8–67

8–6 $f = 1/3\ \mu s = 333$ kHz

TRUE/FALSE QUIZ

1. T **2.** F **3.** T **4.** F **5.** T **6.** T **7.** T **8.** F **9.** T **10.** F

SELF-TEST

1. (d) **2.** (d) **3.** (a) **4.** (c) **5.** (a) **6.** (d) **7.** (b) **8.** (a) **9.** (b) **10.** (c)

CHAPTER 9

Counters

CHAPTER OUTLINE

CHAPTER OBJECTIVES

- Discuss the types of state machines
- Describe the difference between an asynchronous and a synchronous counter
- Analyze counter timing diagrams
- Analyze counter circuits
- Explain how propagation delays affect the operation of a counter
- Determine the modulus of a counter
- Modify the modulus of a counter
- Recognize the difference between a 4-bit binary counter and a decade counter
- Use an up/down counter to generate forward and reverse binary sequences
- Determine the sequence of a counter
- Use IC counters in various applications
- Design a counter that will have any specified sequence of states
- Use cascaded counters to achieve a higher modulus
- Use logic gates to decode any given state of a counter
- Eliminate glitches in counter decoding
- Explain how a digital clock operates
- Interpret counter logic symbols that use dependency notation
- Troubleshoot counters for various types of faults

KEY TERMS

Key terms are in order of appearance in the chapter.

- State machine
- Asynchronous
- Recycle
- Modulus
- Decade
- Synchronous
- Terminal count
- State diagram
- Cascade

VISIT THE WEBSITE

Study aids for this chapter are available at http://www.pearsonglobaleditions.com/floyd

INTRODUCTION

As you learned in Chapter 7, flip-flops can be connected together to perform counting operations. Such a group of flip-flops is a counter, which is a type of finite state machine. The number of flip-flops used and the way in which they are connected determine the number of states (called the modulus) and also the specific sequence of states that the counter goes through during each complete cycle.

Counters are classified into two broad categories according to the way they are clocked: asynchronous and synchronous. In asynchronous counters, commonly called *ripple counters,* the first flip-flop is clocked by the external clock pulse and then each successive flip-flop is clocked by the output of the preceding flip-flop. In synchronous counters, the clock input is connected to all of the flip-flops so that they are clocked simultaneously. Within each of these two categories, counters are classified primarily by the type of sequence, the number of states, or the number of flip-flops in the counter. VHDL codes for various types of counters are presented.

9–1 Finite State Machines

A **state machine** is a sequential circuit having a limited (finite) number of states occuring in a prescribed order. A counter is an example of a state machine; the number of states is called the *modulus*. Two basic types of state machines are the Moore and the Mealy. The **Moore state machine** is one where the outputs depend only on the internal present state. The **Mealy state machine** is one where the outputs depend on both the internal present state and on the inputs. Both types have a timing input (clock) that is not considered a controlling input. A design approach to counters is presented in this section.

After completing this section, you should be able to

- Describe a Moore state machine
- Describe a Mealy state machine
- Discuss examples of Moore and Mealy state machines

General Models of Finite State Machines

A Moore state machine consists of combinational logic that determines the sequence and memory (flip-flops), as shown in Figure 9–1(a). A Mealy state machine is shown in part (b).

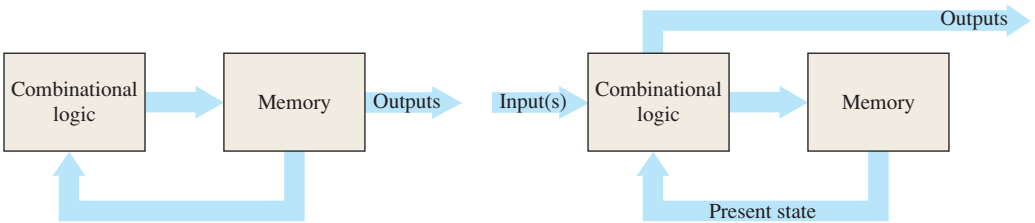
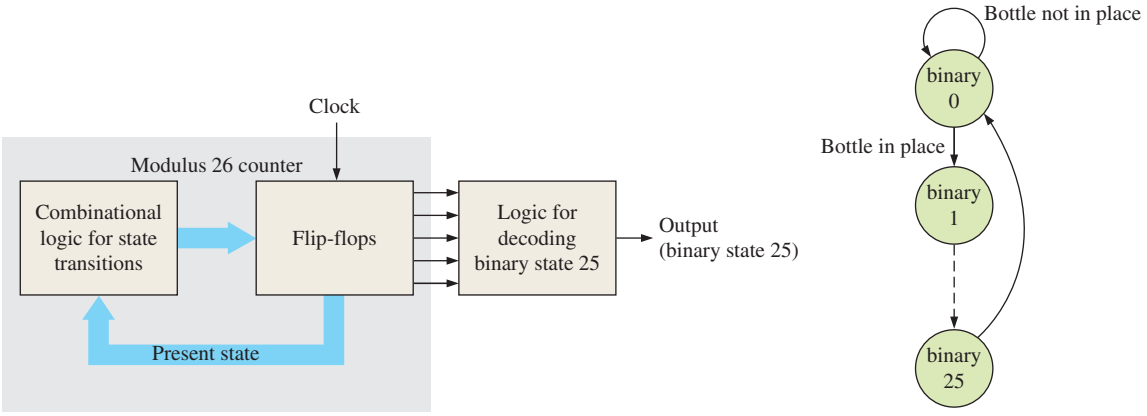


FIGURE 9–1 Two types of sequential logic.

In the Moore machine, the combinational logic is a gate array with outputs that determine the next state of the flip-flops in the memory. There may or may not be inputs to the combinational logic. There may also be output combinational logic, such as a decoder. If there is an input(s), it does not affect the outputs because they always correspond to and are dependent only on the present state of the memory. For the Mealy machine, the present state affects the outputs, just as in the Moore machine; but in addition, the inputs also affect the outputs. The outputs come directly from the combinational logic and not the memory.

Example of a Moore Machine

Figure 9–2(a) shows a Moore machine (modulus-26 binary counter with states 0 through 25) that is used to control the number of tablets (25) that go into each bottle in an assembly line. When the binary number in the memory (flip-flops) reaches binary twenty-five (11001), the counter recycles to 0 and the tablet flow and clock are cut off until the next bottle is in place. The combinational logic for the state transitions sets the modulus of the counter so that it sequences from binary state 0 to binary state 25, where 0 is the reset or rest state and the output combinational logic decodes binary state 25. There is no input in this case, other than the clock, so the next state is determined only by the present state, which makes this a Moore machine. One tablet is bottled for each clock pulse. Once a bottle is in place, the first tablet is inserted at binary state 1, the second at binary state 2, and the twenty-fifth tablet when the binary state is 25. Count 25 is decoded and used to stop the flow of tablets and the clock. The counter stays in the 0 state until the next bottle is in position (indicated by a 1). Then the clock resumes, the count goes to 1, and the cycle repeats, as illustrated by the state diagram in Figure 9–2(b).

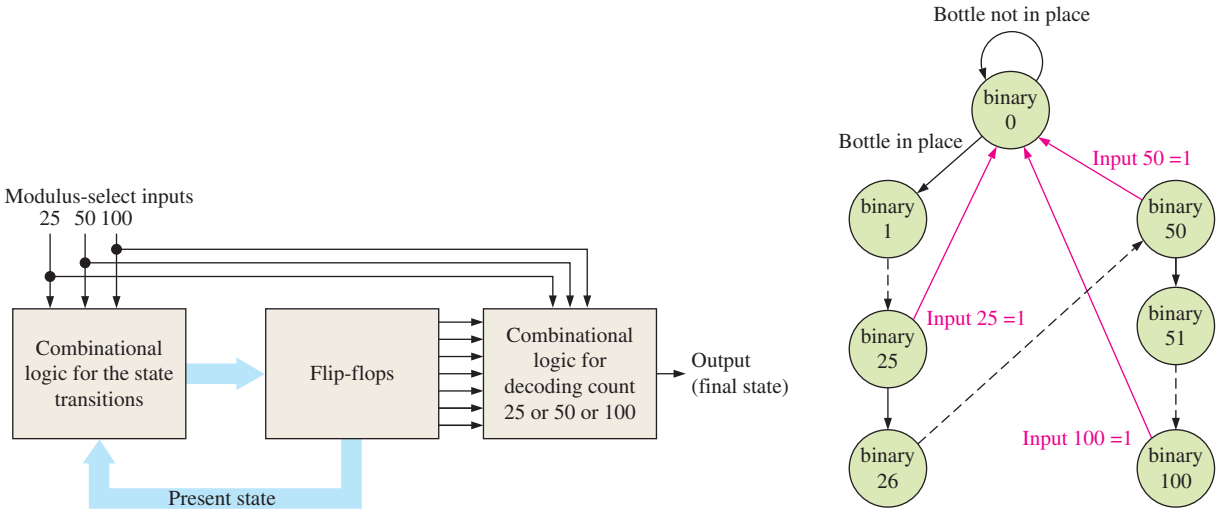


(a) Moore machine (b) State diagram

FIGURE 9–2 A fixed-modulus binary counter as an example of a Moore state machine. The dashed line in the state diagram means the states between binary 1 and 25 are not shown for simplicity.

Example of a Mealy Machine

Let's assume that the tablet-bottling system uses three different sizes of bottles: a 25-tablet bottle, a 50-tablet bottle, and a 100-tablet bottle. This operation requires a state machine with three different terminal counts: 25, 50, and 100. One approach is illustrated in Figure 9–3(a). The combinational logic sets the modulus of the counter depending on the modulus-select inputs. The output of the counter depends on both the present state and the modulus-select inputs, making this a Mealy machine. The state diagram is shown in part (b).



(a) Mealy machine (b) State diagram

FIGURE 9–3 A variable-modulus binary counter as an example of a Mealy state machine. The red arrows in the state diagram represent the recycle paths that depend on the input number. The black dashed lines mean the interim states are not shown for simplicity.

SECTION 9–1 CHECKUP

Answers are at the end of the chapter.

1. What characterizes a finite state machine?
2. Name the types of finite state machines.
3. Explain the difference between the two types of state machines.

9–2 Asynchronous Counters

The term **asynchronous** refers to events that do not have a fixed time relationship with each other and, generally, do not occur at the same time. An **asynchronous counter** is one in which the flip-flops (FF) within the counter do not change states at exactly the same time because they do not have a common clock pulse.

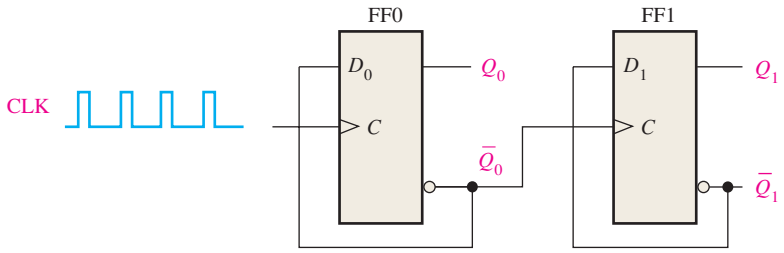
After completing this section, you should be able to

- Describe the operation of a 2-bit asynchronous binary counter
- Describe the operation of a 3-bit asynchronous binary counter
- Define *ripple* in relation to counters
- Describe the operation of an asynchronous decade counter
- Develop counter timing diagrams
- Discuss the implementation of a 4-bit asynchronous binary counter

A 2-Bit Asynchronous Binary Counter

The clock input of an asynchronous counter is always connected only to the LSB flip-flop.

Figure 9–4 shows a 2-bit counter connected for asynchronous operation. Notice that the clock (CLK) is applied to the clock input (C) of *only* the first flip-flop, FF0, which is always the least significant bit (LSB). The second flip-flop, FF1, is triggered by the $\overline{Q}_0$ output of FF0. FF0 changes state at the positive-going edge of each clock pulse, but FF1 changes only when triggered by a positive-going transition of the $\overline{Q}_0$ output of FF0. Because of the inherent propagation delay time through a flip-flop, a transition of the input clock pulse (CLK) and a transition of the $\overline{Q}_0$ output of FF0 can never occur at exactly the same time. Therefore, the two flip-flops are never simultaneously triggered, so the counter operation is asynchronous.



MultiSim

FIGURE 9–4 A 2-bit asynchronous binary counter. Open file F09-04 to verify operation. *A Multisim tutorial is available on the website.*

The Timing Diagram

Let's examine the basic operation of the asynchronous counter of Figure 9–4 by applying four clock pulses to FF0 and observing the Q output of each flip-flop. Figure 9–5 illustrates the changes in the state of the flip-flop outputs in response to the clock pulses. Both flip-flops are connected for toggle operation ($D = \overline{Q}$) and are assumed to be initially RESET (Q LOW).

Asynchronous counters are also known as ripple counters.

The positive-going edge of CLK1 (clock pulse 1) causes the Q_0 output of FF0 to go HIGH, as shown in Figure 9–5. At the same time the $\overline{Q}_0$ output goes LOW, but it has no effect on FF1 because a positive-going transition must occur to trigger the flip-flop. After the leading edge of CLK1, $Q_0 = 1$ and $Q_1 = 0$. The positive-going edge of CLK2 causes Q_0 to go LOW. Output $\overline{Q}_0$ goes HIGH and triggers FF1, causing Q_1 to go HIGH. After the leading edge of CLK2, $Q_0 = 0$ and $Q_1 = 1$. The positive-going edge of CLK3 causes Q_0 to go HIGH again. Output $\overline{Q}_0$ goes LOW and has no effect on FF1. Thus, after the leading edge of CLK3, $Q_0 = 1$ and $Q_1 = 1$. The positive-going edge of CLK4 causes Q_0 to go LOW, while $\overline{Q}_0$ goes HIGH and triggers FF1, causing Q_1 to go LOW. After the leading

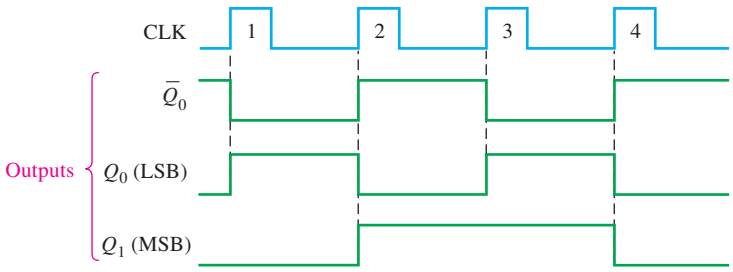


FIGURE 9–5 Timing diagram for the counter of Figure 9–4. As in previous chapters, output waveforms are shown in green.

edge of CLK4, $Q_0 = 0$ and $Q_1 = 0$. The counter has now recycled to its original state (both flip-flops are RESET).

In the timing diagram, the waveforms of the Q_0 and Q_1 outputs are shown relative to the clock pulses as illustrated in Figure 9–5. For simplicity, the transitions of Q_0, Q_1, and the clock pulses are shown as simultaneous even though this is an asynchronous counter. There is, of course, some small delay between the CLK and the Q_0 transition and between the $\overline{Q}_0$ transition and the Q_1 transition.

In digital logic, Q_0 is always the LSB unless otherwise specified.

Note in Figure 9–5 that the 2-bit counter exhibits four different states, as you would expect with two flip-flops ($2^2 = 4$). Also, notice that if Q_0 represents the least significant bit (LSB) and Q_1 represents the most significant bit (MSB), the sequence of counter states represents a sequence of binary numbers as listed in Table 9–1.

TABLE 9–1

Binary state sequence for the counter in Figure 9–4.

Clock Pulse	Q_1	Q_0
Initially	0	0
1	0	1
2	1	0
3	1	1
4 (recycles)	0	0

Since it goes through a binary sequence, the counter in Figure 9–4 is a binary counter. It actually counts the number of clock pulses up to three, and on the fourth pulse it recycles to its original state ($Q_0 = 0$, $Q_1 = 0$). The term **recycle** is commonly applied to counter operation; it refers to the transition of the counter from its final state back to its original state.

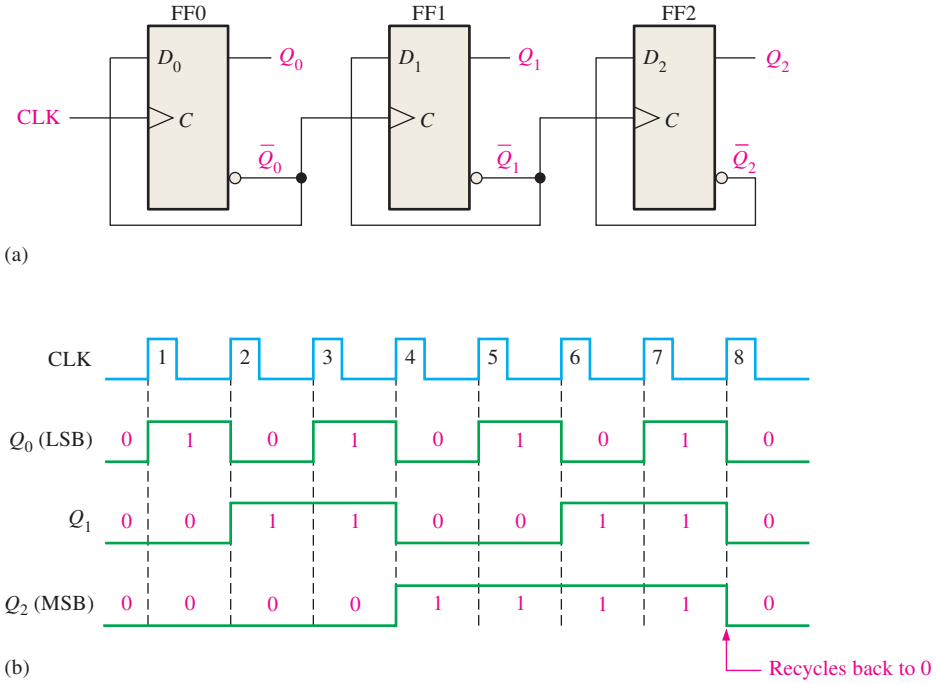
A 3-Bit Asynchronous Binary Counter

The state sequence for a 3-bit binary counter is listed in Table 9–2, and a 3-bit asynchronous binary counter is shown in Figure 9–6(a). The basic operation is the same as that of the 2-bit

TABLE 9–2

State sequence for a 3-bit binary counter.

Clock Pulse	Q_2	Q_1	Q_0
Initially	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8 (recycles)	0	0	0



MultiSim

FIGURE 9–6 Three-bit asynchronous binary counter and its timing diagram for one cycle. Open file F09-06 to verify operation.

counter except that the 3-bit counter has eight states, due to its three flip-flops. A timing diagram is shown in Figure 9–6(b) for eight clock pulses. Notice that the counter progresses through a binary count of zero through seven and then recycles to the zero state. This counter can be easily expanded for higher count, by connecting additional toggle flip-flops.

Propagation Delay

Asynchronous counters are commonly referred to as **ripple counters** for the following reason: The effect of the input clock pulse is first "felt" by FF0. This effect cannot get to FF1 immediately because of the propagation delay through FF0. Then there is the propagation delay through FF1 before FF2 can be triggered. Thus, the effect of an input clock pulse "ripples" through the counter, taking some time, due to propagation delays, to reach the last flip-flop.

To illustrate, notice that all three flip-flops in the counter of Figure 9–6 change state on the leading edge of CLK4. This ripple clocking effect is shown in Figure 9–7 for the first four clock pulses, with the propagation delays indicated. The LOW-to-HIGH transition of

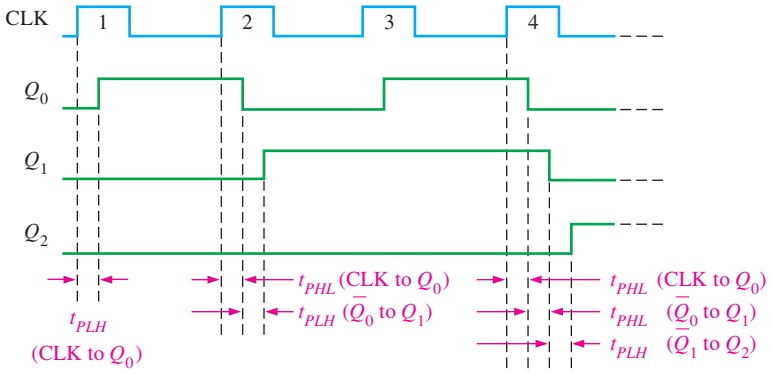


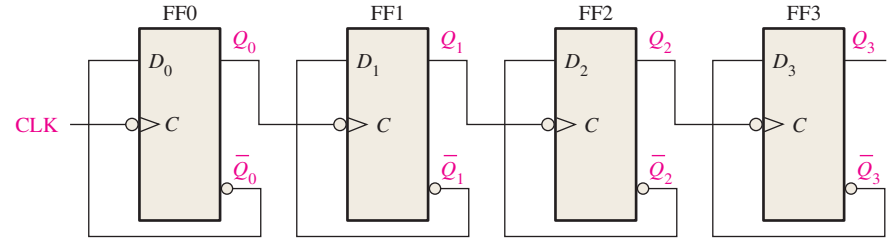
FIGURE 9–7 Propagation delays in a 3-bit asynchronous (ripple-clocked) binary counter.

Q_0 occurs one delay time (t_{PLH}) after the positive-going transition of the clock pulse. The LOW-to-HIGH transition of Q_1 occurs one delay time (t_{PLH}) after the positive-going transition of $\overline{Q}_0$. The LOW-to-HIGH transition of Q_2 occurs one delay time (t_{PLH}) after the positive-going transition of $\overline{Q}_1$. As you can see, FF2 is not triggered until two delay times after the positive-going edge of the clock pulse, CLK4. Thus, it takes three propagation delay times for the effect of the clock pulse, CLK4, to ripple through the counter and change Q_2 from LOW to HIGH.

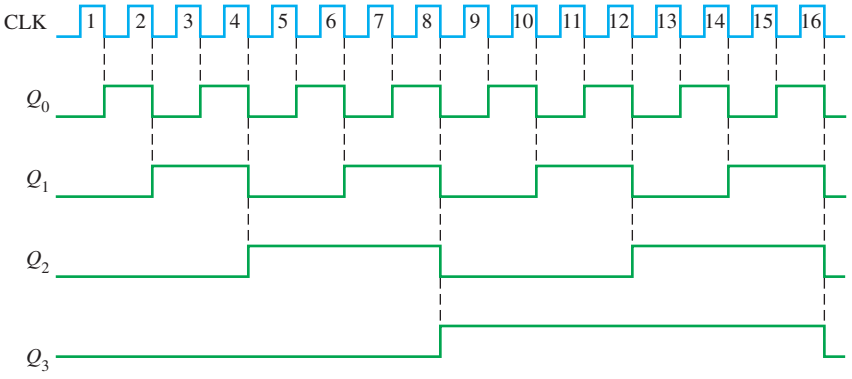
This cumulative delay of an asynchronous counter is a major disadvantage in many applications because it limits the rate at which the counter can be clocked and creates decoding problems. The maximum cumulative delay in a counter must be less than the period of the clock waveform.

EXAMPLE 9–1

A 4-bit asynchronous binary counter is shown in Figure 9–8(a). Each D flip-flop is negative edge-triggered and has a propagation delay for 10 nanoseconds (ns). Develop a timing diagram showing the Q output of each flip-flop, and determine the total propagation delay time from the triggering edge of a clock pulse until a corresponding change can occur in the state of Q_3. Also determine the maximum clock frequency at which the counter can be operated.



(a)



(b)

FIGURE 9–8 Four-bit asynchronous binary counter and its timing diagram. Open file F09-08 and verify the operation.

MultiSim

Solution

The timing diagram with delays omitted is as shown in Figure 9–8(b). For the total delay time, the effect of CLK8 or CLK16 must propagate through four flip-flops before Q_3 changes, so

$$t_{p(tot)} = 4 \times 10\text{ ns} = \mathbf{40\ ns}$$

The maximum clock frequency is

$$f_{\max} = \frac{1}{t_{p(tot)}} = \frac{1}{40 \text{ ns}} = \mathbf{25 \text{ MHz}}$$

The counter should be operated below this frequency to avoid problems due to the propagation delay.

Related Problem*

Show the timing diagram if all of the flip-flops in Figure 9–8(a) are positive edge-triggered.

*Answers are at the end of the chapter.

Asynchronous Decade Counters

A counter can have 2^n states, where n is the number of flip-flops.

The **modulus** of a counter is the number of unique states through which the counter will sequence. The maximum possible number of states (maximum modulus) of a counter is 2^n, where n is the number of flip-flops in the counter. Counters can be designed to have a number of states in their sequence that is less than the maximum of 2^n. This type of sequence is called a *truncated sequence.*

One common modulus for counters with truncated sequences is ten (called MOD10). Counters with ten states in their sequence are called **decade** counters. A **decade counter** with a count sequence of zero (0000) through nine (1001) is a BCD decade counter because its ten-state sequence produces the BCD code. This type of counter is useful in display applications in which BCD is required for conversion to a decimal readout.

To obtain a truncated sequence, it is necessary to force the counter to recycle before going through all of its possible states. For example, the BCD decade counter must recycle back to the 0000 state after the 1001 state. A decade counter requires four flip-flops (three flip-flops are insufficient because $2^3 = 8$).

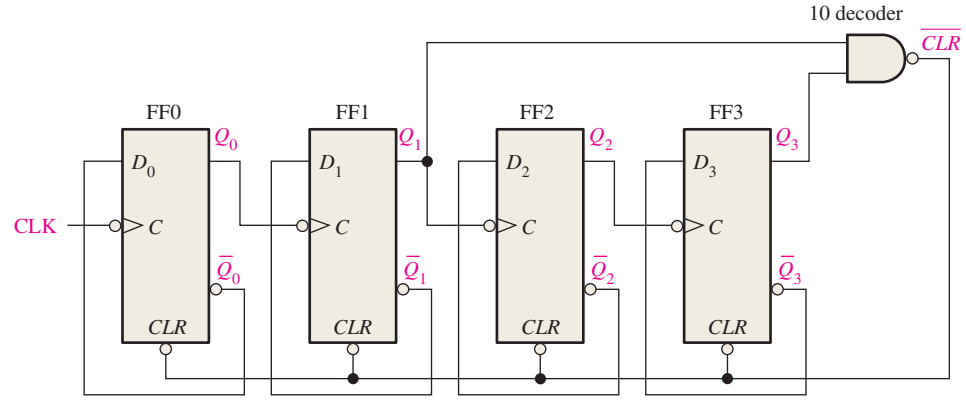
Let's use a 4-bit asynchronous counter such as the one in Example 9–1 and modify its sequence to illustrate the principle of truncated counters. One way to make the counter recycle after the count of nine (1001) is to decode count ten (1010) with a NAND gate and connect the output of the NAND gate to the clear ($\overline{CLR}$) inputs of the flip-flops, as shown in Figure 9–9(a).

Partial Decoding

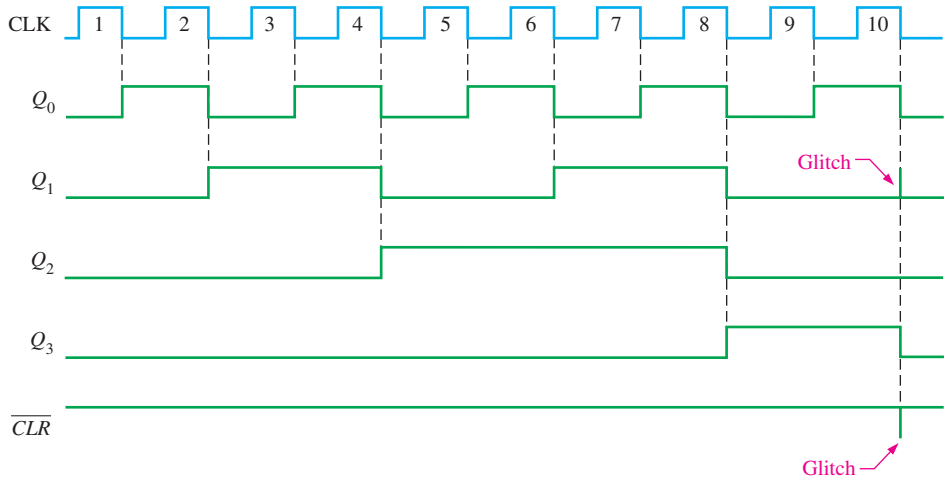
Notice in Figure 9–9(a) that only Q_1 and Q_3 are connected to the NAND gate inputs. This arrangement is an example of *partial decoding,* in which the two unique states ($Q_1 = 1$ and $Q_3 = 1$) are sufficient to decode the count of ten because none of the other states (zero through nine) have both Q_1 and Q_3 HIGH at the same time. When the counter goes into count ten (1010), the decoding gate output goes LOW and asynchronously resets all the flip-flops.

The resulting timing diagram is shown in Figure 9–9(b). Notice that there is a glitch on the Q_1 waveform. The reason for this glitch is that Q_1 must first go HIGH before the count of ten can be decoded. Not until several nanoseconds after the counter goes to the count of ten does the output of the decoding gate go LOW (both inputs are HIGH). Thus, the counter is in the 1010 state for a short time before it is reset to 0000, thus producing the glitch on Q_1 and the resulting glitch on the $\overline{CLR}$ line that resets the counter.

Other truncated sequences can be implemented in a similar way, as Example 9–2 shows.



(a)



(b)

FIGURE 9–9 An asynchronously clocked decade counter with asynchronous recycling.

EXAMPLE 9–2

Show how an asynchronous counter with J-K flip-flops can be implemented having a modulus of twelve with a straight binary sequence from 0000 through 1011.

Solution

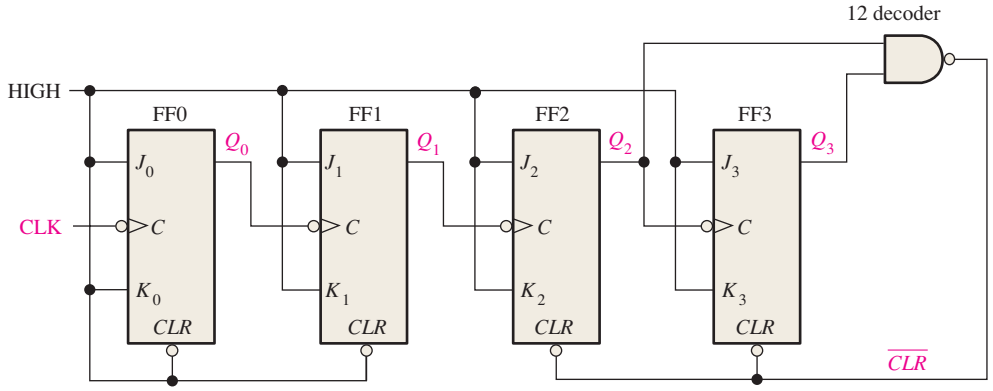
Since three flip-flops can produce a maximum of eight states, four flip-flops are required to produce any modulus greater than eight but less than or equal to sixteen.

When the counter gets to its last state, 1011, it must recycle back to 0000 rather than going to its normal next state of 1100, as illustrated in the following sequence chart:

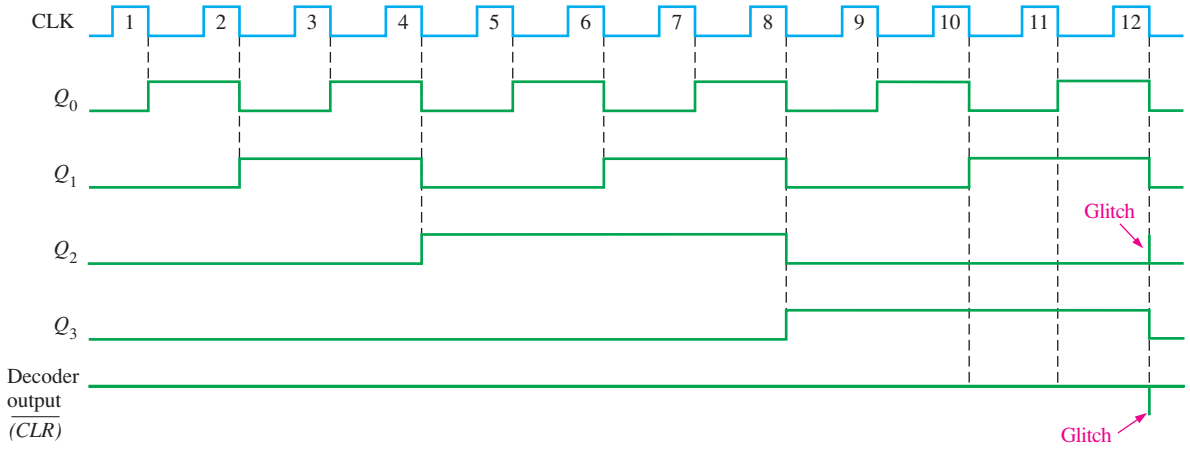
Q_3	Q_2	Q_1	Q_0	
0	0	0	0	←
.	.	.	.	
.	.	.	.	Recycles
.	.	.	.	
1	0	1	1	←
1	1	0	0	← Normal next state

Observe that Q_0 and Q_1 both go to 0 anyway, but Q_2 and Q_3 must be forced to 0 on the twelfth clock pulse. Figure 9–10(a) shows the modulus-12 counter. The NAND gate partially decodes count twelve (1100) and resets flip-flop 2 and flip-flop 3.

Thus, on the twelfth clock pulse, the counter is forced to recycle from count eleven to count zero, as shown in the timing diagram of Figure 9–10(b). (It is in count twelve for only a few nanoseconds before it is reset by the glitch on $\overline{CLR}$.)



(a)



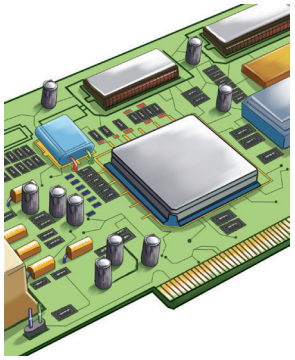
(b)

FIGURE 9–10 Asynchronously clocked modulus-12 counter with asynchronous recycling.

Related Problem

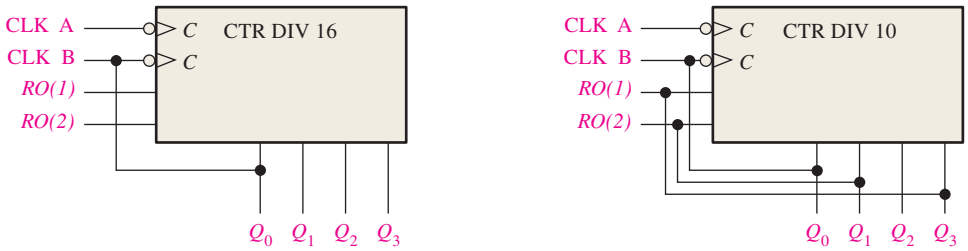
How can the counter in Figure 9–10(a) be modified to make it a modulus-13 counter?

IMPLEMENTATION: 4-BIT ASYNCHRONOUS BINARY COUNTER



Fixed-Function Device The 74HC93 is an example of a specific integrated circuit asynchronous counter. This device actually consists of a single flip-flop (CLK A) and a 3-bit asynchronous counter (CLK B). This arrangement is for flexibility. It can be used as a divide-by-2 device if only the single flip-flop is used, or it can be used as a modulus-8 counter if only the 3-bit counter portion is used. This device also provides gated reset inputs, *RO(1)* and *RO(2)*. When both of these inputs are HIGH, the counter is reset to the 0000 state $\overline{CLR}$.

Additionally, the 74HC93 can be used as a 4-bit modulus-16 counter (counts 0 through 15) by connecting the Q_0 output to the CLK B input as shown by the logic symbol in Figure 9–11(a). It can also be configured as a decade counter (counts 0 through 9) with asynchronous recycling by using the gated reset inputs for partial decoding of count ten, as shown by the logic symbol in Figure 9–11(b).



(a) 74HC93 connected as a modulus-16 counter

(b) 74HC93 connected as a decade counter

FIGURE 9–11 Two configurations of the 74HC93 asynchronous counter. (The qualifying label, CTR DIV *n*, indicates a counter with *n* states.)

Programmable Logic Device (PLD) The VHDL code for a generic 4-bit asynchronous binary counter using J-K flip flops with preset (PRN) and clear (CLRN) inputs is as follows:

```
library ieee;
use ieee.std_logic_1164.all;

entity AsyncFourBitBinCntr is
   port (Clock, Clr: in std_logic; Q0, Q1, Q2, Q3: inout std_logic);
end entity AsyncFourBitBinCntr;

architecture LogicOperation of AsyncFourBitBinCntr is
component jkff is
   port (J, K, Clk, PRN, CLRN: in std_logic; Q: out std_logic);
end component jkff;

begin
   FF0: jkff port map(J=>'1', K=>'1', Clk=>Clock,  CLRN=>Clr, PRN=>'1', Q=>Q0);
   FF1: jkff port map(J=>'1', K=>'1', Clk=>not Q0, CLRN=>Clr, PRN=>'1', Q=>Q1);
   FF2: jkff port map(J=>'1', K=>'1', Clk=>not Q1, CLRN=>Clr, PRN=>'1', Q=>Q2);
   FF3: jkff port map(J=>'1', K=>'1', Clk=>not Q2, CLRN=>Clr, PRN=>'1', Q=>Q3);
end architecture LogicOperation;
```

Inputs and outputs declared

J-K flip-flop component declaration

Instantiations define how each flip-flop is connected.

SECTION 9–2 CHECKUP

1. What does the term *asynchronous* mean in relation to counters?
2. How many states does a modulus-14 counter have? What is the minimum number of flip-flops required?

9–3 Synchronous Counters

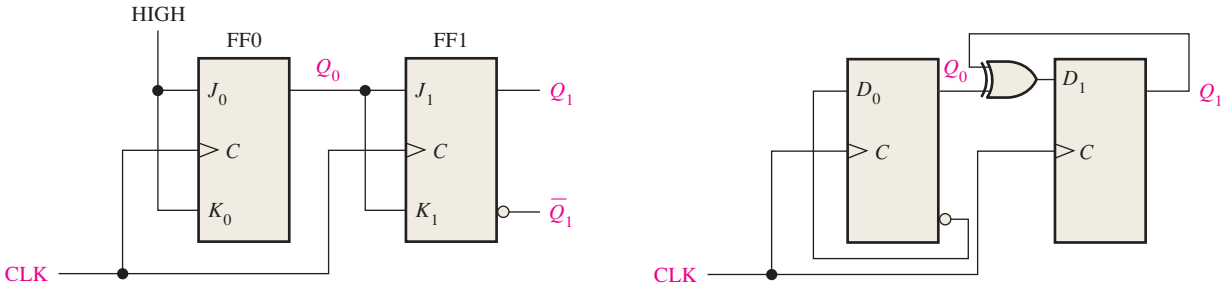
The term **synchronous** refers to events that have a fixed time relationship with each other. A **synchronous counter** is one in which all the flip-flops in the counter are clocked at the same time by a common clock pulse. J-K flip-flops are used to illustrate most synchronous counters. D flip-flops can also be used but generally require more logic because of having no direct toggle or no-change states.

After completing this section, you should be able to

- Describe the operation of a 2-bit synchronous binary counter
- Describe the operation of a 3-bit synchronous binary counter
- Describe the operation of a 4-bit synchronous binary counter
- Describe the operation of a synchronous decade counter
- Develop counter timing diagrams

A 2-Bit Synchronous Binary Counter

Figure 9–12 shows a 2-bit synchronous binary counter. Notice that an arrangement different from that for the asynchronous counter must be used for the J_1 and K_1 inputs of FF1 in order to achieve a binary sequence. A D flip-flop implementation is shown in part (b).



(a) J-K flip-flop (b) D flip-flop

FIGURE 9–12 2-bit synchronous binary counters.

The clock input goes to each flip-flop in a synchronous counter.

The operation of a J-K flip-flop synchronous counter is as follows: First, assume that the counter is initially in the binary 0 state; that is, both flip-flops are RESET. When the positive edge of the first clock pulse is applied, FF0 will toggle and Q_0 will therefore go HIGH. What happens to FF1 at the positive-going edge of CLK1? To find out, let's look at the input conditions of FF1. Inputs J_1 and K_1 are both LOW because Q_0, to which they are connected, has not yet gone HIGH. Remember, there is a propagation delay from the triggering edge of the clock pulse until the Q output actually makes a transition. So, $J = 0$ and $K = 0$ when the leading edge of the first clock pulse is applied. This is a no-change condition, and therefore FF1 does not change state. A timing detail of this portion of the counter operation is shown in Figure 9–13(a).

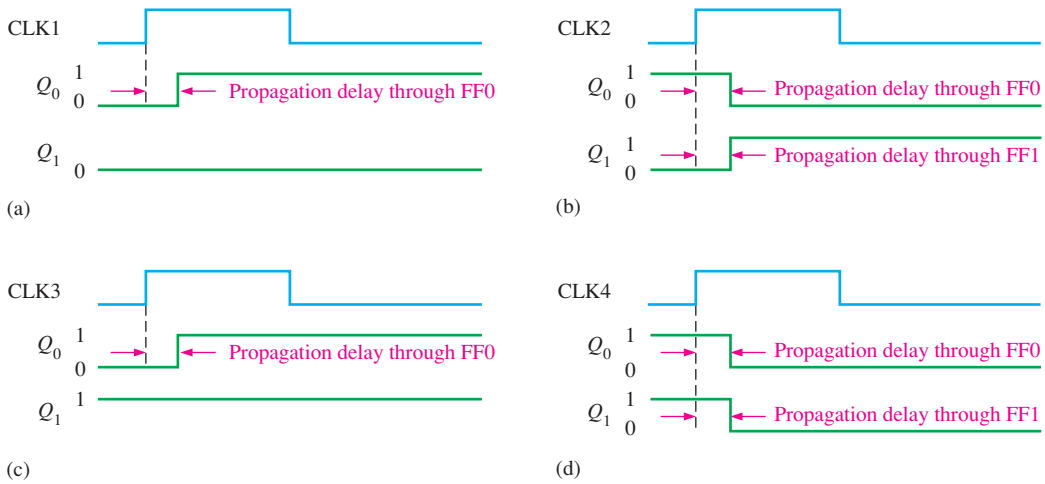


FIGURE 9–13 Timing details for the 2-bit synchronous counter operation (the propagation delays of both flip-flops are assumed to be equal).

After CLK1, $Q_0 = 1$ and $Q_1 = 0$ (which is the binary 1 state). When the leading edge of CLK2 occurs, FF0 will toggle and Q_0 will go LOW. Since FF1 has a HIGH ($Q_0 = 1$) on its J_1 and K_1 inputs at the triggering edge of this clock pulse, the flip-flop toggles and Q_1 goes HIGH. Thus, after CLK2, $Q_0 = 0$ and $Q_1 = 1$ (which is a binary 2 state). The timing detail for this condition is shown in Figure 9–13(b).

When the leading edge of CLK3 occurs, FF0 again toggles to the SET state ($Q_0 = 1$), and FF1 remains SET ($Q_1 = 1$) because its J_1 and K_1 inputs are both LOW ($Q_0 = 0$). After this triggering edge, $Q_0 = 1$ and $Q_1 = 1$ (which is a binary 3 state). The timing detail is shown in Figure 9–13(c).

Finally, at the leading edge of CLK4, Q_0 and Q_1 go LOW because they both have a toggle condition on their *J* and *K* inputs. The timing detail is shown in Figure 9–13(d). The counter has now recycled to its original state, binary 0. Examination of the D flip-flop counter in Figure 9–12(b) will show the timing diagram is the same as for the J-K flip-flop counter.

The complete timing diagram for the counters in Figure 9–12 is shown in Figure 9–14. Notice that all the waveform transitions appear coincident; that is, the propagation delays are not indicated. Although the delays are an important factor in the synchronous counter operation, in an overall timing diagram they are normally omitted for simplicity. Major waveform relationships resulting from the normal operation of a circuit can be conveyed completely without showing small delay and timing differences. However, in high-speed digital circuits, these small delays are an important consideration in design and troubleshooting.

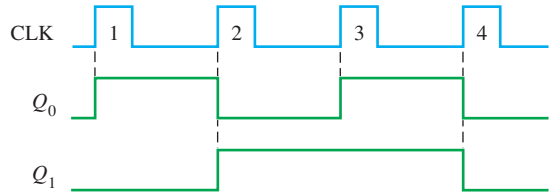
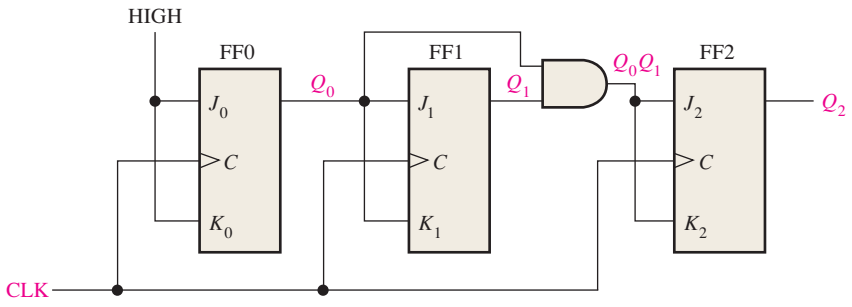


FIGURE 9–14 Timing diagram for the counters of Figure 9–12.

A 3-Bit Synchronous Binary Counter

A 3-bit synchronous binary counter is shown in Figure 9–15, and its timing diagram is shown in Figure 9–16. You can understand this counter operation by examining its sequence of states as shown in Table 9–3.



MultiSim

FIGURE 9–15 A 3-bit synchronous binary counter. Open file F09-15 to verify the operation.

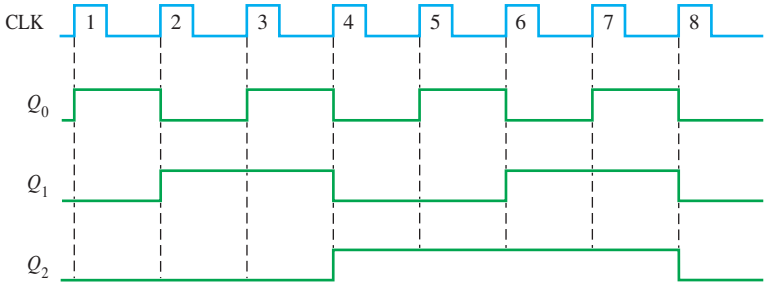


FIGURE 9–16 Timing diagram for the counter of Figure 9–15.

TABLE 9–3

State sequence for a 3-bit binary counter.

Clock Pulse	Q_2	Q_1	Q_0
Initially	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8 (recycles)	0	0	0

InfoNote

The TSC or *time stamp counter* in some microprocessors is used for performance monitoring, which enables a number of parameters important to the overall performance of a system to be determined exactly. By reading the TSC before and after the execution of a procedure, the precise time required for the procedure can be determined based on the processor cycle time. In this way, the TSC forms the basis for all time evaluations in connection with optimizing system operation. For example, it can be accurately determined which of two or more programming sequences is more efficient. This is a very useful tool for compiler developers and system programmers in producing the most effective code.

First, let's look at Q_0. Notice that Q_0 changes on each clock pulse as the counter progresses from its original state to its final state and then back to its original state. To produce this operation, FF0 must be held in the toggle mode by constant HIGHs on its J_0 and K_0 inputs. Notice that Q_1 goes to the opposite state following each time Q_0 is a 1. This change occurs at CLK2, CLK4, CLK6, and CLK8. The CLK8 pulse causes the counter to recycle. To produce this operation, Q_0 is connected to the J_1 and K_1 inputs of FF1. When Q_0 is a 1 and a clock pulse occurs, FF1 is in the toggle mode and therefore changes state. The other times, when Q_0 is a 0, FF1 is in the no-change mode and remains in its present state.

Next, let's see how FF2 is made to change at the proper times according to the binary sequence. Notice that both times Q_2 changes state, it is preceded by the unique condition in which both Q_0 and Q_1 are HIGH. This condition is detected by the AND gate and applied to the J_2 and K_2 inputs of FF2. Whenever both Q_0 and Q_1 are HIGH, the output of the AND gate makes the J_2 and K_2 inputs of FF2 HIGH, and FF2 toggles on the following clock pulse. At all other times, the J_2 and K_2 inputs of FF2 are held LOW by the AND gate output, and FF2 does not change state.

The analysis of the counter in Figure 9–15 is summarized in Table 9–4.

TABLE 9–4

Summary of the analysis of the counter in Figure 9–15.

	Outputs			*J-K* Inputs						At the Next Clock Pulse		
Clock Pulse	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0	FF2	FF1	FF0
Initially	0	0	0	0	0	0	0	1	1	NC*	NC	Toggle
1	0	0	1	0	0	1	1	1	1	NC	Toggle	Toggle
2	0	1	0	0	0	0	0	1	1	NC	NC	Toggle
3	0	1	1	1	1	1	1	1	1	Toggle	Toggle	Toggle
4	1	0	0	0	0	0	0	1	1	NC	NC	Toggle
5	1	0	1	0	0	1	1	1	1	NC	Toggle	Toggle
6	1	1	0	0	0	0	0	1	1	NC	NC	Toggle
7	1	1	1	1	1	1	1	1	1	Toggle	Toggle	Toggle
										Counter recycles back to 000.		

*NC indicates *No Change*.

A 4-Bit Synchronous Binary Counter

Figure 9–17(a) shows a 4-bit synchronous binary counter, and Figure 9–17(b) shows its timing diagram. This particular counter is implemented with negative edge-triggered flip-flops. The reasoning behind the *J* and *K* input control for the first three flip-flops is the same as previously discussed for the 3-bit counter. The fourth stage, FF3, changes only twice in the sequence. Notice that both of these transitions occur following the times that Q_0, Q_1, and Q_2 are all HIGH. This condition is decoded by AND gate G_2 so that when a

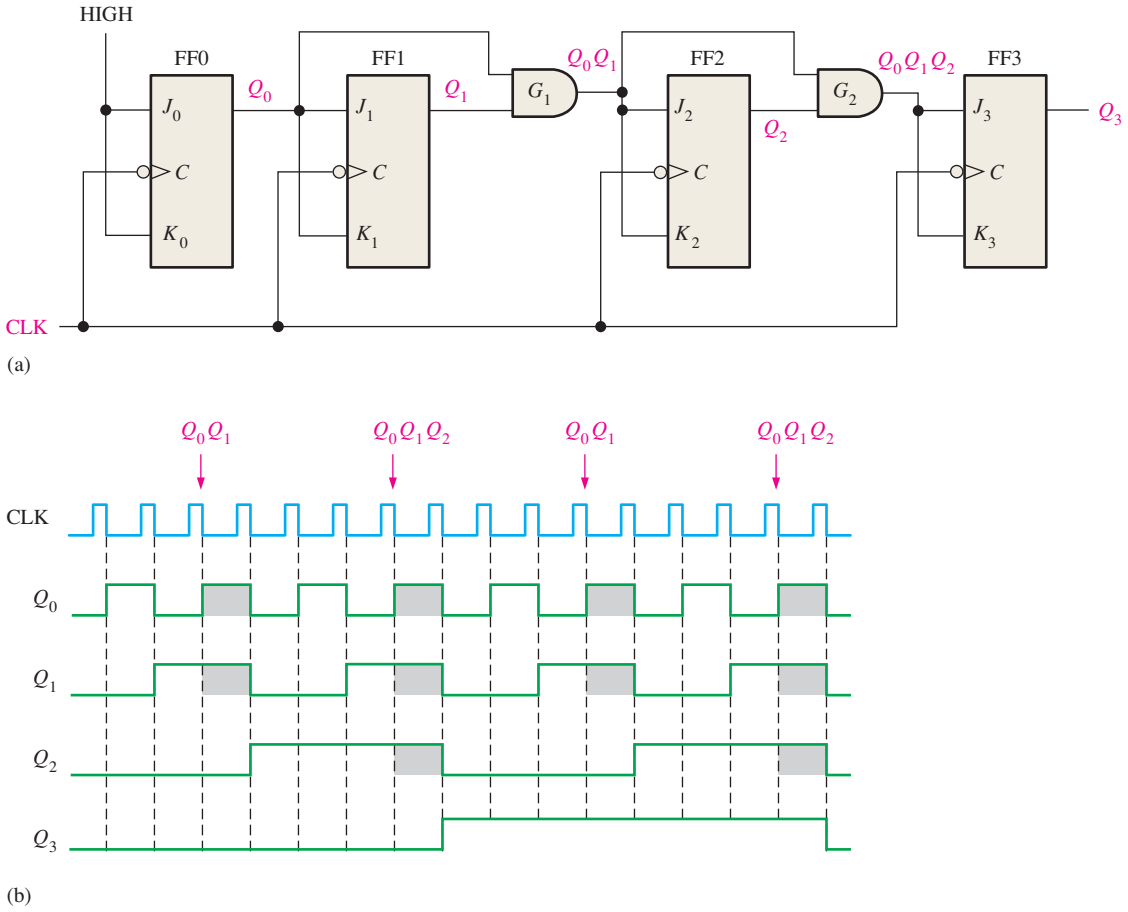


FIGURE 9–17 A 4-bit synchronous binary counter and timing diagram. Times where the AND gate outputs are HIGH are indicated by the shaded areas.

clock pulse occurs, FF3 will change state. For all other times the J_3 and K_3 inputs of FF3 are LOW, and it is in a no-change condition.

A 4-Bit Synchronous Decade Counter

A decade counter has ten states.

As you know, a BCD decade counter exhibits a truncated binary sequence and goes from 0000 through the 1001 state. Rather than going from the 1001 state to the 1010 state, it recycles to the 0000 state. A synchronous BCD decade counter is shown in Figure 9–18. The timing diagram for the decade counter is shown in Figure 9–19.

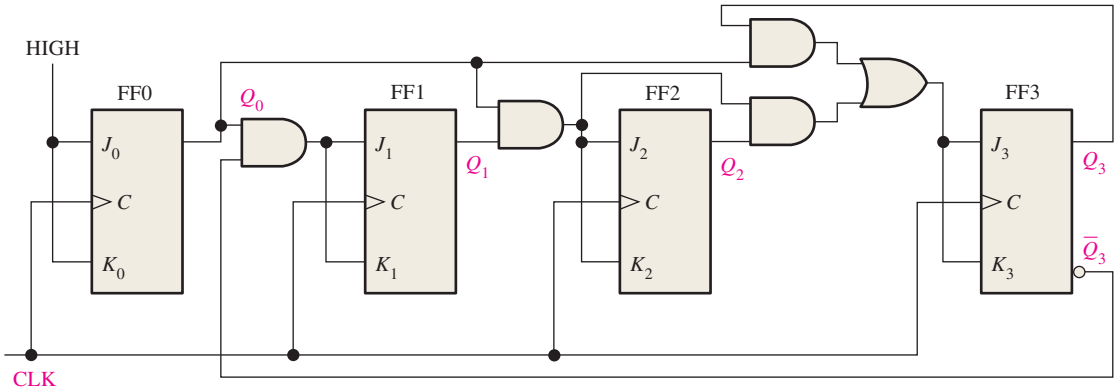


FIGURE 9–18 A synchronous BCD decade counter. Open file F09-18 to verify operation.

MultiSim

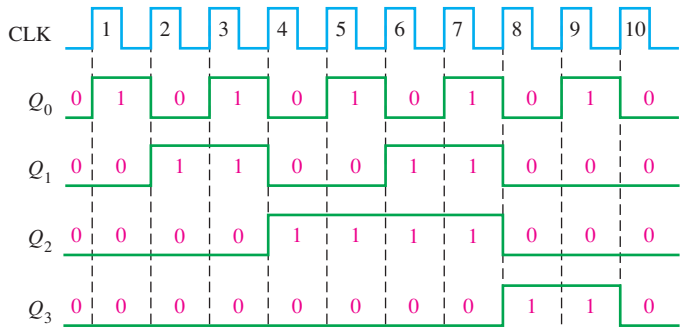


FIGURE 9–19 Timing diagram for the BCD decade counter (Q_0 is the LSB).

The counter operation is shown by the sequence of states in Table 9–5. First, notice that FF0 (Q_0) toggles on each clock pulse, so the logic equation for its J_0 and K_0 inputs is

$$J_0 = K_0 = 1$$

This equation is implemented by connecting J_0 and K_0 to a constant HIGH level.

TABLE 9–5

States of a BCD decade counter.

Clock Pulse	Q_3	Q_2	Q_1	Q_0
Initially	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
10 (recycles)	0	0	0	0

Next, notice in Table 9–5 that FF1 (Q_1) changes on the next clock pulse each time $Q_0 = 1$ and $Q_3 = 0$, so the logic equation for the J_1 and K_1 inputs is

$$J_1 = K_1 = Q_0\overline{Q}_3$$

This equation is implemented by ANDing Q_0 and $\overline{Q}_3$ and connecting the gate output to the J_1 and K_1 inputs of FF1.

Flip-flop 2 (Q_2) changes on the next clock pulse each time both $Q_0 = 1$ and $Q_1 = 1$. This requires an input logic equation as follows:

$$J_2 = K_2 = Q_0Q_1$$

This equation is implemented by ANDing Q_0 and Q_1 and connecting the gate output to the J_2 and K_2 inputs of FF2.

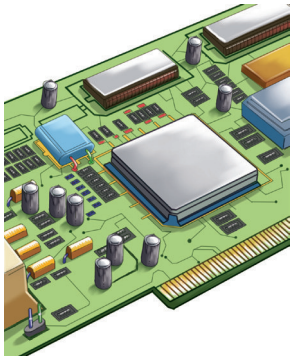
Finally, FF3 (Q_3) changes to the opposite state on the next clock pulse each time $Q_0 = 1$, $Q_1 = 1$, and $Q_2 = 1$ (state 7), or when $Q_0 = 1$ and $Q_3 = 1$ (state 9). The equation for this is as follows:

$$J_3 = K_3 = Q_0Q_1Q_2 + Q_0Q_3$$

This function is implemented with the AND/OR logic connected to the J_3 and K_3 inputs of FF3 as shown in the logic diagram in Figure 9–18. Notice that the differences between this

decade counter and the modulus-16 binary counter in Figure 9–17(a) are the $Q_0\overline{Q}_3$ AND gate, the Q_0Q_3 AND gate, and the OR gate; this arrangement detects the occurrence of the 1001 state and causes the counter to recycle properly on the next clock pulse.

IMPLEMENTATION: 4-BIT SYNCHRONOUS BINARY COUNTER



Fixed-Function Device The 74HC163 is an example of an integrated circuit 4-bit synchronous binary counter. A logic symbol is shown in Figure 9–20 with pin numbers in parentheses. This counter has several features in addition to the basic functions previously discussed for the general synchronous binary counter.

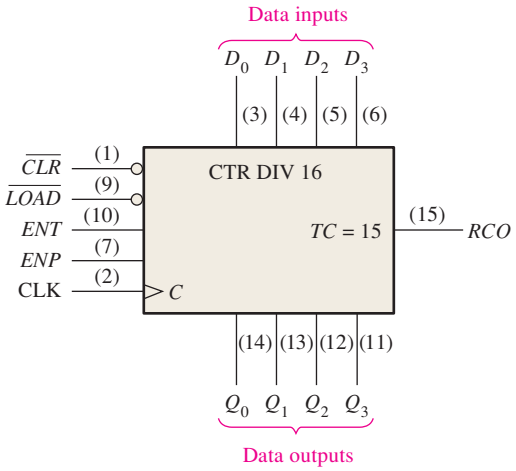


FIGURE 9–20 The 74HC163 4-bit synchronous binary counter. (The qualifying label CTR DIV 16 indicates a counter with sixteen states.)

First, the counter can be synchronously preset to any 4-bit binary number by applying the proper levels to the parallel data inputs. When a LOW is applied to the $\overline{LOAD}$ input, the counter will assume the state of the data inputs on the next clock pulse. Thus, the counter sequence can be started with any 4-bit binary number.

Also, there is an active-LOW clear input ($\overline{CLR}$), which synchronously resets all four flip-flops in the counter. There are two enable inputs, *ENP* and *ENT*. These inputs must both be HIGH for the counter to sequence through its binary states. When at least one input is LOW, the counter is disabled. The ripple clock output (*RCO*) goes HIGH when the counter reaches the last state in its sequence of fifteen, called the **terminal count** ($TC = 15$). This output, in conjunction with the enable inputs, allows these counters to be cascaded for higher count sequences.

Figure 9–21 shows a timing diagram of this counter being preset to twelve (1100) and then counting up to its terminal count, fifteen (1111). Input D_0 is the least significant input bit, and Q_0 is the least significant output bit.

Let's examine this timing diagram in detail. This will aid you in interpreting timing diagrams in this chapter or on manufacturers' data sheets. To begin, the LOW level pulse on the $\overline{CLR}$ input causes all the outputs (Q_0, Q_1, Q_2, and Q_3) to go LOW.

Next, the LOW level pulse on the $\overline{LOAD}$ input synchronously enters the data on the data inputs (D_0, D_1, D_2, and D_3) into the counter. These data appear on the Q outputs at the time of the first positive-going clock edge after $\overline{LOAD}$ goes LOW. This is the preset operation. In this particular example, Q_0 is LOW, Q_1 is LOW, Q_2 is HIGH, and Q_3 is HIGH. This, of course, is a binary 12 (Q_0 is the LSB).

The counter now advances through states 13, 14, and 15 on the next three positive-going clock edges. It then recycles to 0, 1, 2 on the following clock pulses. Notice that

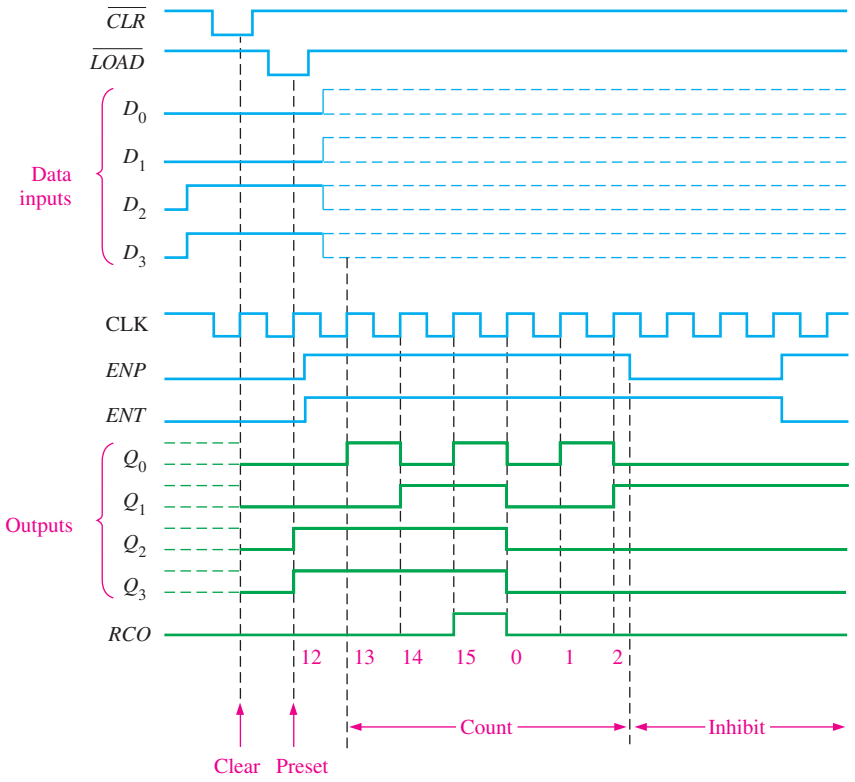


FIGURE 9–21 Timing example for a 74HC163.

both *ENP* and *ENT* inputs are HIGH during the state sequence. When *ENP* goes LOW, the counter is inhibited and remains in the binary 2 state.

Programmable Logic Device (PLD) The VHDL code for a 4-bit synchronous decade counter using J-K flip flops is as follows:

```
library ieee;
use ieee.std_logic_1164.all;

entity FourBitSynchDecadeCounter is
   port (Clk: in std_logic; Q0, Q1, Q2, Q3: inout std_logic);
end entity FourBitSynchDecadeCounter;

architecture LogicOperation of FourBitSynchDecadeCounter is

component jkff is
   port (J, K, Clk: in std_logic; Q: out std_logic);
end component jkff;

signal J1, J2, J3: std_logic;
begin
J1 <= Q0 and not Q3;
J2 <= Q1 and Q0;
J3 <= (Q2 and J2) or (Q0 and Q3);

FF0: jkff port map (J => '1', K => '1', Clk => Clk, Q => Q0);
FF1: jkff port map (J => J1, K => J1, Clk => Clk, Q => Q1);
FF2: jkff port map (J => J2, K => J2, Clk => Clk, Q => Q2);
FF3: jkff port map (J => J3, K => J3, Clk => Clk, Q => Q3);
end architecture LogicOperation;
```

Input and outputs declared

Component declaration for the J-K flip-flop

Boolean expressions for *J* input of each flip-flop (*J* = *K*)

Instantiations define connections for each flip-flop.

SECTION 9–3 CHECKUP

1. How does a synchronous counter differ from an asynchronous counter?
2. Explain the function of the preset feature of counters such as the 74HC163.
3. Describe the purpose of the *ENP* and *ENT* inputs and the *RCO* output for the 74HC163 counter.

9–4 Up/Down Synchronous Counters

An **up/down counter** is one that is capable of progressing in either direction through a certain sequence. An up/down counter, sometimes called a bidirectional counter, can have any specified sequence of states. A 3-bit binary counter that advances upward through its sequence (0, 1, 2, 3, 4, 5, 6, 7) and then can be reversed so that it goes through the sequence in the opposite direction (7, 6, 5, 4, 3, 2, 1, 0) is an illustration of up/down sequential operation.

After completing this section, you should be able to

- Explain the basic operation of an up/down counter
- Discuss the 74HC190 up/down decade counter

In general, most up/down counters can be reversed at any point in their sequence. For instance, the 3-bit binary counter can be made to go through the following sequence:

$$\overbrace{0, 1, 2, 3, 4, 5,}^{\text{UP}}\ \underbrace{4, 3, 2,}_{\text{DOWN}}\ \overbrace{3, 4, 5, 6, 7,}^{\text{UP}}\ \underbrace{6, 5,}_{\text{DOWN}}\ \text{etc.}$$

Table 9–6 shows the complete up/down sequence for a 3-bit binary counter. The arrows indicate the state-to-state movement of the counter for both its UP and its DOWN modes of operation. An examination of Q_0 for both the up and down sequences shows that FF0 toggles on each clock pulse. Thus, the J_0 and K_0 inputs of FF0 are

$$J_0 = K_0 = 1$$

TABLE 9–6

Up/Down sequence for a 3-bit binary counter.

Clock Pulse	Up	Q_2	Q_1	Q_0	Down
0	↻	0	0	0	↻
1	↻	0	0	1	↻
2	↻	0	1	0	↻
3	↻	0	1	1	↻
4	↻	1	0	0	↻
5	↻	1	0	1	↻
6	↻	1	1	0	↻
7	↻	1	1	1	↻

For the up sequence, Q_1 changes state on the next clock pulse when $Q_0 = 1$. For the down sequence, Q_1 changes on the next clock pulse when $Q_0 = 0$. Thus, the J_1 and K_1 inputs of FF1 must equal 1 under the conditions expressed by the following equation:

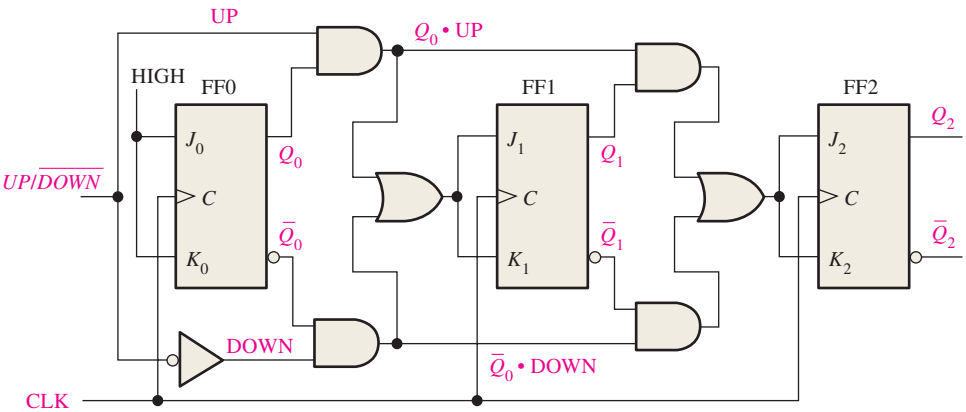
$$J_1 = K_1 = (Q_0 \cdot \text{UP}) + (\overline{Q}_0 \cdot \text{DOWN})$$

For the up sequence, Q_2 changes state on the next clock pulse when $Q_0 = Q_1 = 1$. For the down sequence, Q_2 changes on the next clock pulse when $Q_0 = Q_1 = 0$. Thus, the J_2 and K_2 inputs of FF2 must equal 1 under the conditions expressed by the following equation:

$$J_2 = K_2 = (Q_0 \cdot Q_1 \cdot \text{UP}) + (\overline{Q}_0 \cdot \overline{Q}_1 \cdot \text{DOWN})$$

Each of the conditions for the J and K inputs of each flip-flop produces a toggle at the appropriate point in the counter sequence.

Figure 9–22 shows a basic implementation of a 3-bit up/down binary counter using the logic equations just developed for the J and K inputs of each flip-flop. Notice that the $UP/\overline{DOWN}$ control input is HIGH for UP and LOW for DOWN.



MultiSim

FIGURE 9–22 A basic 3-bit up/down synchronous counter. Open file F09-22 to verify operation.

EXAMPLE 9–3

Show the timing diagram and determine the sequence of a 4-bit synchronous binary up/down counter if the clock and $UP/\overline{DOWN}$ control inputs have waveforms as shown in Figure 9–23(a). The counter starts in the all-0s state and is positive edge-triggered.

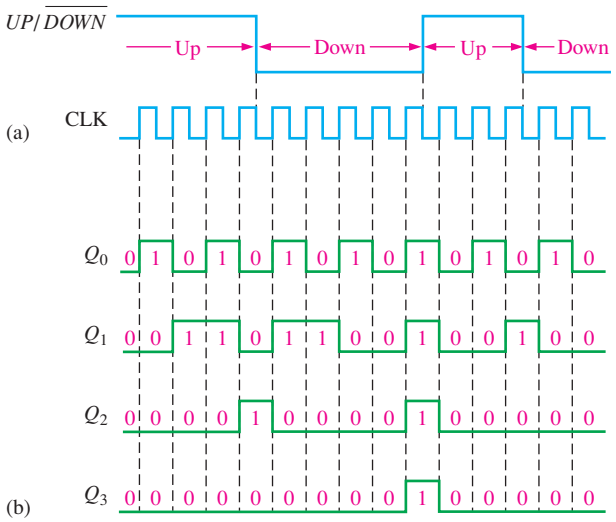


FIGURE 9–23

Solution

The timing diagram showing the Q outputs is shown in Figure 9–23(b). From these waveforms, the counter sequence is as shown in Table 9–7.

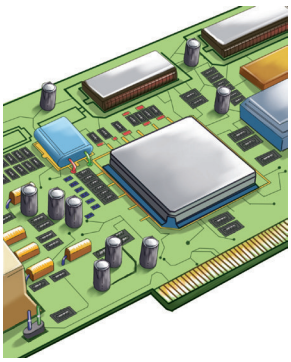
TABLE 9–7

Q_3	Q_2	Q_1	Q_0	
0	0	0	0	
0	0	0	1	
0	0	1	0	UP
0	0	1	1	
0	1	0	0	
0	0	1	1	
0	0	1	0	
0	0	0	1	DOWN
0	0	0	0	
1	1	1	1	
0	0	0	0	
0	0	0	1	UP
0	0	1	0	
0	0	0	1	DOWN
0	0	0	0	

Related Problem

Show the timing diagram if the $UP/\overline{DOWN}$ control waveform in Figure 9–23(a) is inverted.

IMPLEMENTATION: UP/DOWN DECADE COUNTER



Fixed-Function Device Figure 9–24 shows a logic diagram for the 74HC190, an example of an integrated circuit up/down synchronous decade counter. The direction of the count is determined by the level of the up/down input ($D/\overline{U}$). When this input is HIGH, the counter counts down; when it is LOW, the counter counts up. Also, this device can be preset to any desired BCD digit as determined by the states of the data inputs when the $\overline{LOAD}$ input is LOW.

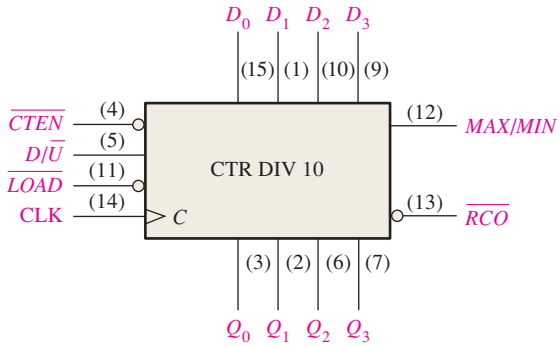


FIGURE 9–24 The 74HC190 up/down synchronous decade counter.

The *MAX/MIN* output produces a HIGH pulse when the terminal count nine (1001) is reached in the UP mode or when the terminal count zero (0000) is reached in the DOWN mode. The *MAX/MIN* output, the ripple clock output ($\overline{RCO}$), and the count enable input ($\overline{CTEN}$) are used when cascading counters. (Cascaded counters are discussed in Section 9–6.)

Figure 9–25 is a timing diagram that shows the 74HC190 counter preset to seven (0111) and then going through a count-up sequence followed by a count-down sequence. The *MAX/MIN* output is HIGH when the counter is in either the all-0s state (*MIN*) or the 1001 state (*MAX*).

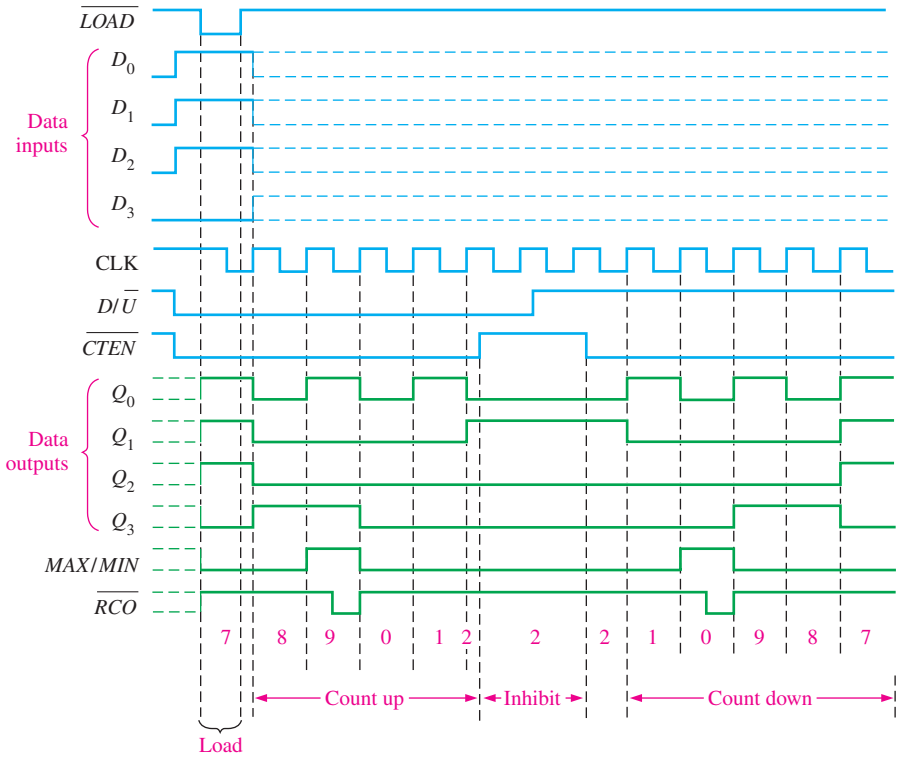


FIGURE 9–25 Timing example for a 74HC190.

Programmable Logic Device (PLD) A VHDL code for an up/down decade counter using J-K flip-flops is as follows:

```
library ieee;
use ieee.std_logic_1164.all;

entity UpDnDecadeCntr is
   port (UPDN, Clk: in std_logic; Q0, Q1, Q2, Q3: buffer std_logic);
end entity UpDnDecadeCntr;

architecture LogicOperation of UpDnDecadeCntr is

component jkff is
   port (J, K, Clk: in std_logic; Q: buffer std_logic);
end component jkff;
```

UPDN: Counter direction
Clk: System clock
Q0-Q3: Counter output

J-K flip flop component

```
function UpDown(A, B, C, D: in std_logic)
   return std_logic is
begin
   return((A and B) or (C and D));
end function UpDown;

signal J1Up, J1Dn, J1, J2, J3: std_logic;
begin
   J1Up <= UPDN and Q0; J1Dn <= not UPDN and not Q0;
   UpDn1: J1 <= UpDown(UPDN, Q0, not UPDN, not Q0);
   UpDn2: J2 <= UpDown(J1Up, Q1, J1Dn, not Q1);
   UpDn3: J3 <= UpDown(J1Up and Q1, Q2, J1Dn and not Q1, not Q2);

   FF0: jkff port map (J =>'1', K =>'1', Clk => Clk, Q => Q0);
   FF1: jkff port map (J => J1, K => J1, Clk => Clk, Q => Q1);
   FF2: jkff port map (J => J2, K => J2, Clk => Clk, Q => Q2);
   FF3: jkff port map (J => J3, K => J3, Clk => Clk, Q => Q3);
end architecture LogicOperation;
```

Function UpDown is a helper function performing the common logic between stages performed by the two AND gates applied to the OR gate supplying the J K inputs of the next stage. See Figure 9–22.

J1Up: Initial Up logic for FF1.
J1Dn: Initial Down logic for FF1.
J1-J3: Variable for combined UpDown applied to FF1-FF3.

Identifiers J1, J2, and J3 complete the up/down logic applied to the J and K inputs of flip-flop stages FF0-FF1. Using a function to perform operations common to multiple tasks simplifies the overall code design and implementation.

Flip-flop stages FF0-FF3 complete the Up/Down counter.

SECTION 9–4 CHECKUP

1. A 4-bit up/down binary counter is in the DOWN mode and in the 1010 state. On the next clock pulse, to what state does the counter go?
2. What is the terminal count of a 4-bit binary counter in the UP mode? In the DOWN mode? What is the next state after the terminal count in the DOWN mode?

9–5 Design of Synchronous Counters

In this section, you will learn the six steps to design a counter (state machine). As you learned in Section 9–1, sequential circuits can be classified into two types: (1) those in which the output or outputs depend only on the present internal state (Moore state machines) and (2) those in which the output or outputs depend on both the present state and the input or inputs (Mealy state machines). This section is recommended for those who want an introduction to counter design or to state machine design in general. It is not a prerequisite for any other material.

After completing this section, you should be able to

- Develop a state diagram for a given sequence
- Develop a next-state table for a specified counter sequence
- Create a flip-flop transition table
- Use the Karnaugh map method to derive the logic requirements for a synchronous counter
- Implement a counter to produce a specified sequence of states

Step 1: State Diagram

The first step in the design of a state machine (counter) is to create a state diagram. A **state diagram** shows the progression of states through which the counter advances when it is

clocked. As an example, Figure 9–26 is a state diagram for a basic 3-bit Gray code counter. This particular circuit has no inputs other than the clock and no outputs other than the outputs taken off each flip-flop in the counter. You may wish to review the coverage of the Gray code in Chapter 2 at this time.

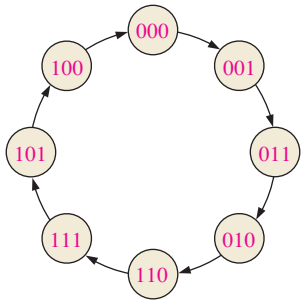


FIGURE 9–26 State diagram for a 3-bit Gray code counter.

Step 2: Next-State Table

Once the sequential circuit is defined by a state diagram, the second step is to derive a next-state table, which lists each state of the counter (present state) along with the corresponding next state. *The next state is the state that the counter goes to from its present state upon application of a clock pulse.* The next-state table is derived from the state diagram and is shown in Table 9–8 for the 3-bit Gray code counter. Q_0 is the least significant bit.

TABLE 9–8

Next-state table for 3-bit Gray code counter.

Present State			Next State		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	1	0	1	0
0	1	0	1	1	0
1	1	0	1	1	1
1	1	1	1	0	1
1	0	1	1	0	0
1	0	0	0	0	0

TABLE 9–9

Transition table for a J-K flip-flop.

Output Transitions			Flip-Flop Inputs	
Q_N		Q_{N+1}	J	K
0	⟶	0	0	X
0	⟶	1	1	X
1	⟶	0	X	1
1	⟶	1	X	0

Q_N: present state
Q_{N+1}: next state
X: "don't care"

Step 3: Flip-Flop Transition Table

Table 9–9 is a transition table for the J-K flip-flop. All possible output transitions are listed by showing the Q output of the flip-flop going from present states to next states. Q_N is the present state of the flip-flop (before a clock pulse) and Q_{N+1} is the next state (after a clock pulse). For each output transition, the J and K inputs that will cause the transition to occur are listed. An X indicates a "don't care" (the input can be either a 1 or a 0).

To design the counter, the transition table is applied to each of the flip-flops in the counter, based on the next-state table (Table 9–8). For example, for the present state 000,

Q_0 goes from a present state of 0 to a next state of 1. To make this happen, J_0 must be a 1 and you don't care what K_0 is ($J_0 = 1$, $K_0 = X$), as you can see in the transition table (Table 9–9). Next, Q_1 is 0 in the present state and remains a 0 in the next state. For this transition, $J_1 = 0$ and $K_1 = X$. Finally, Q_2 is 0 in the present state and remains a 0 in the next state. Therefore, $J_2 = 0$ and $K_2 = X$. This analysis is repeated for each present state in Table 9–8.

Step 4: Karnaugh Maps

Karnaugh maps can be used to determine the logic required for the *J* and *K* inputs of each flip-flop in the counter. There is a Karnaugh map for the *J* input and a Karnaugh map for the *K* input of each flip-flop. In this design procedure, each cell in a Karnaugh map represents one of the present states in the counter sequence listed in Table 9–8.

From the *J* and *K* states in the transition table (Table 9–9) a 1, 0, or X is entered into each present-state cell on the maps depending on the transition of the *Q* output for a particular flip-flop. To illustrate this procedure, two sample entries are shown for the J_0 and the K_0 inputs to the least significant flip-flop (Q_0) in Figure 9–27.

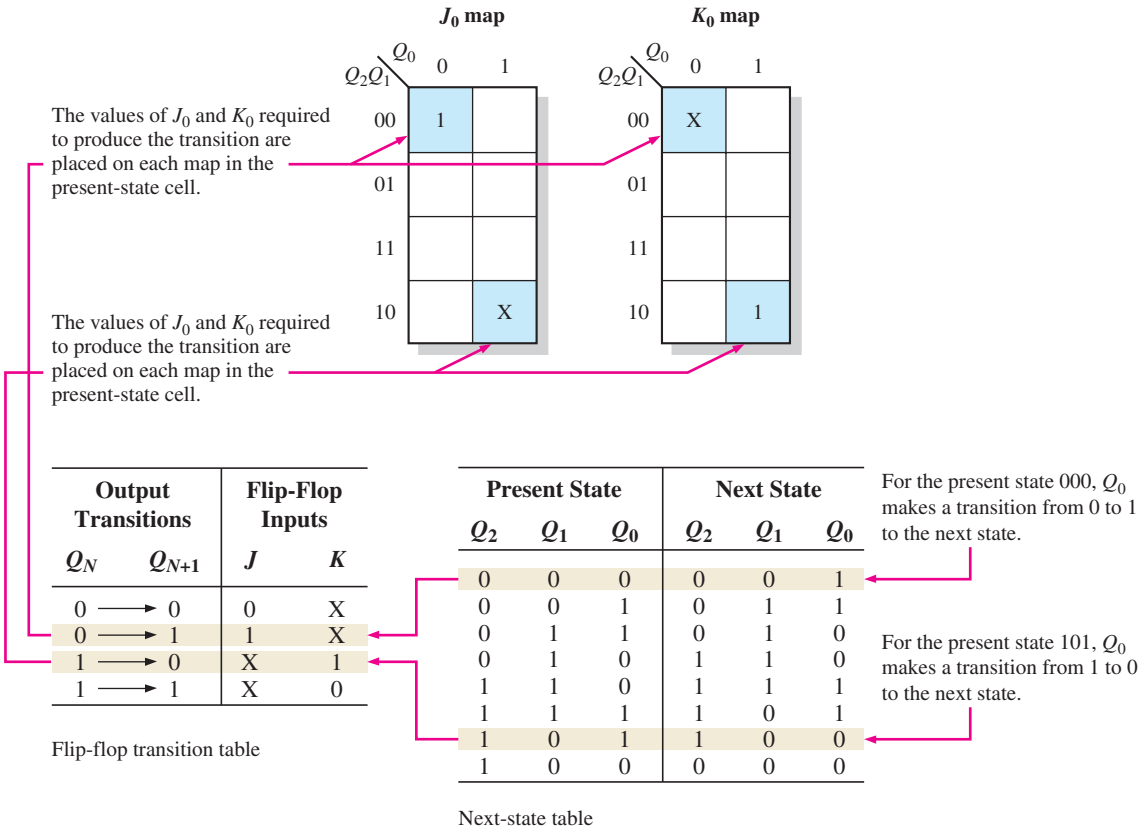


FIGURE 9–27 Examples of the mapping procedure for the counter sequence represented in Table 9–8 and Table 9–9.

The completed Karnaugh maps for all three flip-flops in the counter are shown in Figure 9–28. The cells are grouped as indicated and the corresponding Boolean expressions for each group are derived.

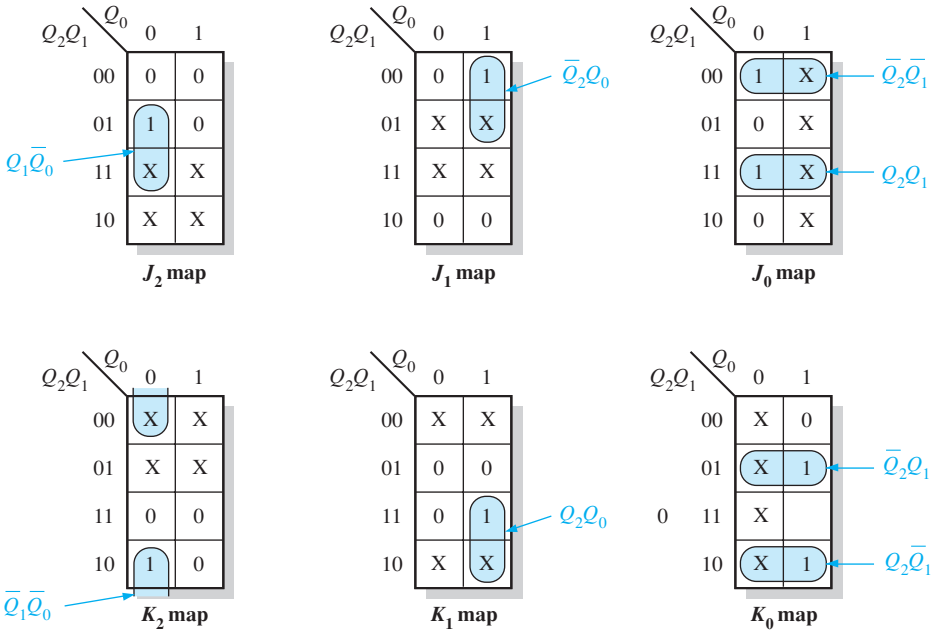


FIGURE 9–28 Karnaugh maps for present-state *J* and *K* inputs.

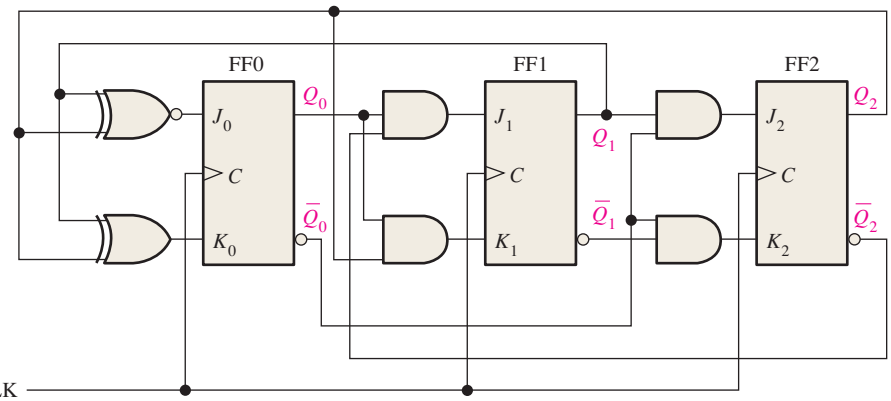
Step 5: Logic Expressions for Flip-Flop Inputs

From the Karnaugh maps of Figure 9–28 you obtain the following expressions for the *J* and *K* inputs of each flip-flop:

$$J_0 = Q_2Q_1 + \overline{Q}_2\overline{Q}_1 = \overline{Q_2 \oplus Q_1}$$
$$K_0 = Q_2\overline{Q}_1 + \overline{Q}_2Q_1 = Q_2 \oplus Q_1$$
$$J_1 = \overline{Q}_2Q_0$$
$$K_1 = Q_2Q_0$$
$$J_2 = Q_1\overline{Q}_0$$
$$K_2 = \overline{Q}_1\overline{Q}_0$$

Step 6: Counter Implementation

The final step is to implement the combinational logic from the expressions for the *J* and *K* inputs and connect the flip-flops to form the complete 3-bit Gray code counter as shown in Figure 9–29.



MultiSim

FIGURE 9–29 Three-bit Gray code counter. Open file F09-29 to verify operation.

A summary of steps used in the design of the 3-bit Gray code counter follows. In general, these steps can be applied to any state machine.

1. Specify the counter sequence and draw a state diagram.
2. Derive a next-state table from the state diagram.
3. Develop a transition table showing the flip-flop inputs required for each transition. The transition table is always the same for a given type of flip-flop.
4. Transfer the *J* and *K* states from the transition table to Karnaugh maps. There is a Karnaugh map for each input of each flip-flop.
5. Group the Karnaugh map cells to generate and derive the logic expression for each flip-flop input.
6. Implement the expressions with combinational logic, and combine with the flip-flops to create the counter.

This procedure is now applied to the design of other synchronous counters in Examples 9–4 and 9–5.

EXAMPLE 9–4

Design a counter with the irregular binary count sequence shown in the state diagram of Figure 9–30. Use D flip-flops.

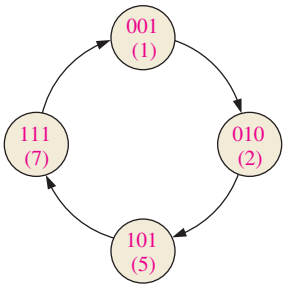


FIGURE 9–30

Solution

Step 1: The state diagram is as shown. Although there are only four states, a 3-bit counter is required to implement this sequence because the maximum binary count is seven. Since the required sequence does not include all the possible binary states, the invalid states (0, 3, 4, and 6) can be treated as "don't cares" in the design. However, if the counter should erroneously get into an invalid state, you must make sure that it goes back to a valid state.

Step 2: The next-state table is developed from the state diagram and is given in Table 9–10.

TABLE 9–10

Next-state table.

Present State			Next State		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	1	0	1	0
0	1	0	1	0	1
1	0	1	1	1	1
1	1	1	0	0	1

Step 3: The transition table for the D flip-flop is shown in Table 9–11.

TABLE 9–11

Transition table for a D flip-flop.

Output Transitions			Flip-Flop Input
Q_N		Q_{N+1}	D
0	⟶	0	0
0	⟶	1	1
1	⟶	0	0
1	⟶	1	1

Step 4: The D inputs are plotted on the present-state Karnaugh maps in Figure 9–31. Also "don't cares" can be placed in the cells corresponding to the invalid states of 000, 011, 100, and 110, as indicated by the red Xs.

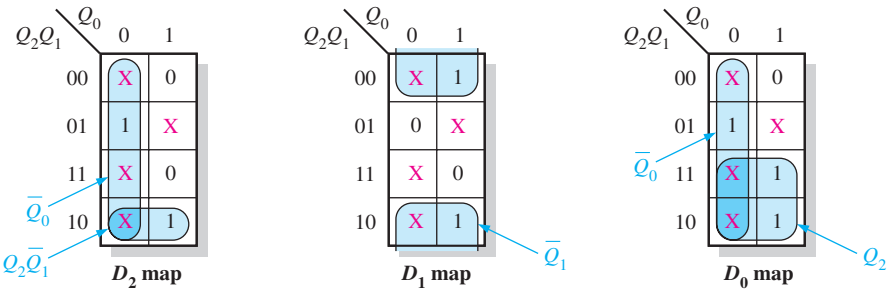


FIGURE 9–31

Step 5: Group the 1s, taking advantage of as many of the "don't care" states as possible for maximum simplification, as shown in Figure 9–31. The expression for each D input taken from the maps is as follows:

$$D_0 = \overline{Q}_0 + Q_2$$
$$D_1 = \overline{Q}_1$$
$$D_2 = \overline{Q}_0 + Q_2\overline{Q}_1$$

Step 6: The implementation of the counter is shown in Figure 9–32.

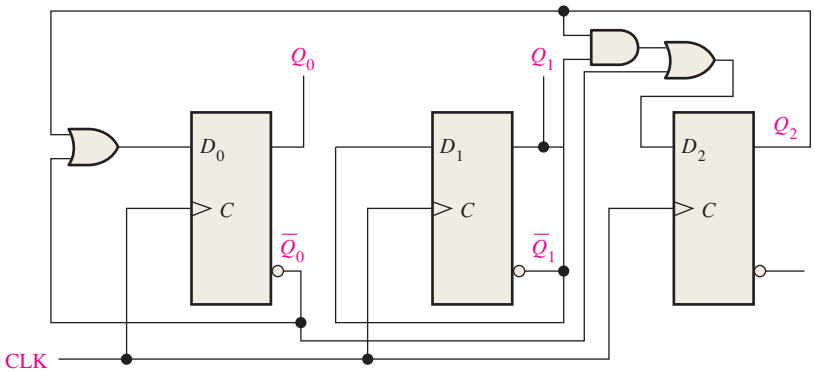


FIGURE 9–32

An analysis shows that if the counter, by accident, gets into one of the invalid states (0, 3, 4, 6), it will always return to a valid state according to the following sequences: $0 \rightarrow 3 \rightarrow 4 \rightarrow 7$, and $6 \rightarrow 1$.

Related Problem

Verify the analysis that proves the counter will always return (eventually) to a valid state from an invalid state.

EXAMPLE 9–5

Develop a synchronous 3-bit up/down counter with a Gray code sequence using J-K flip-flops. The counter should count up when an UP/$\overline{\text{DOWN}}$ control input is 1 and count down when the control input is 0.

Solution

Step 1: The state diagram is shown in Figure 9–33. The 1 or 0 beside each arrow indicates the state of the UP/$\overline{\text{DOWN}}$ control input, *Y*.

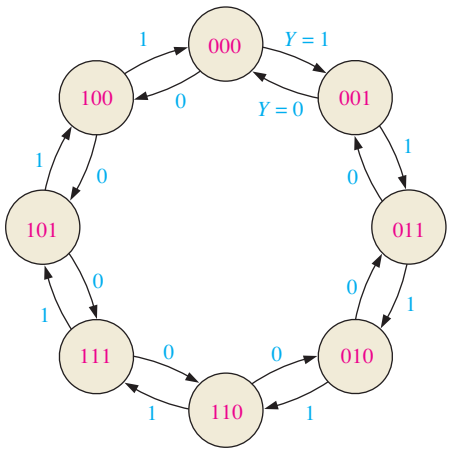


FIGURE 9–33 State diagram for a 3-bit up/down Gray code counter.

Step 2: The next-state table is derived from the state diagram and is shown in Table 9–12. Notice that for each present state there are two possible next states, depending on the UP/$\overline{\text{DOWN}}$ control variable, *Y*.

TABLE 9–12

Next-state table for 3-bit up/down Gray code counter.

Present State			Next State					
			$Y = 0$ (DOWN)			$Y = 1$ (UP)		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	1	1
0	1	1	0	0	1	0	1	0
0	1	0	0	1	1	1	1	0
1	1	0	0	1	0	1	1	1
1	1	1	1	1	0	1	0	1
1	0	1	1	1	1	1	0	0
1	0	0	1	0	1	0	0	0

Y = UP/$\overline{\text{DOWN}}$ control input.

Step 3: The transition table for the J-K flip-flops is repeated in Table 9–13.

TABLE 9–13

Transition table for a J-K flip-flop.

Output Transitions			Flip-Flop Inputs	
Q_N		Q_{N+1}	J	K
0	⟶	0	0	X
0	⟶	1	1	X
1	⟶	0	X	1
1	⟶	1	X	0

Step 4: The Karnaugh maps for the J and K inputs of the flip-flops are shown in Figure 9–34. The UP/$\overline{\text{DOWN}}$ control input, Y, is considered one of the state variables along with Q_0, Q_1, and Q_2. Using the next-state table, the information in the "Flip-Flop Inputs" column of Table 9–13 is transferred onto the maps as indicated for each present state of the counter.

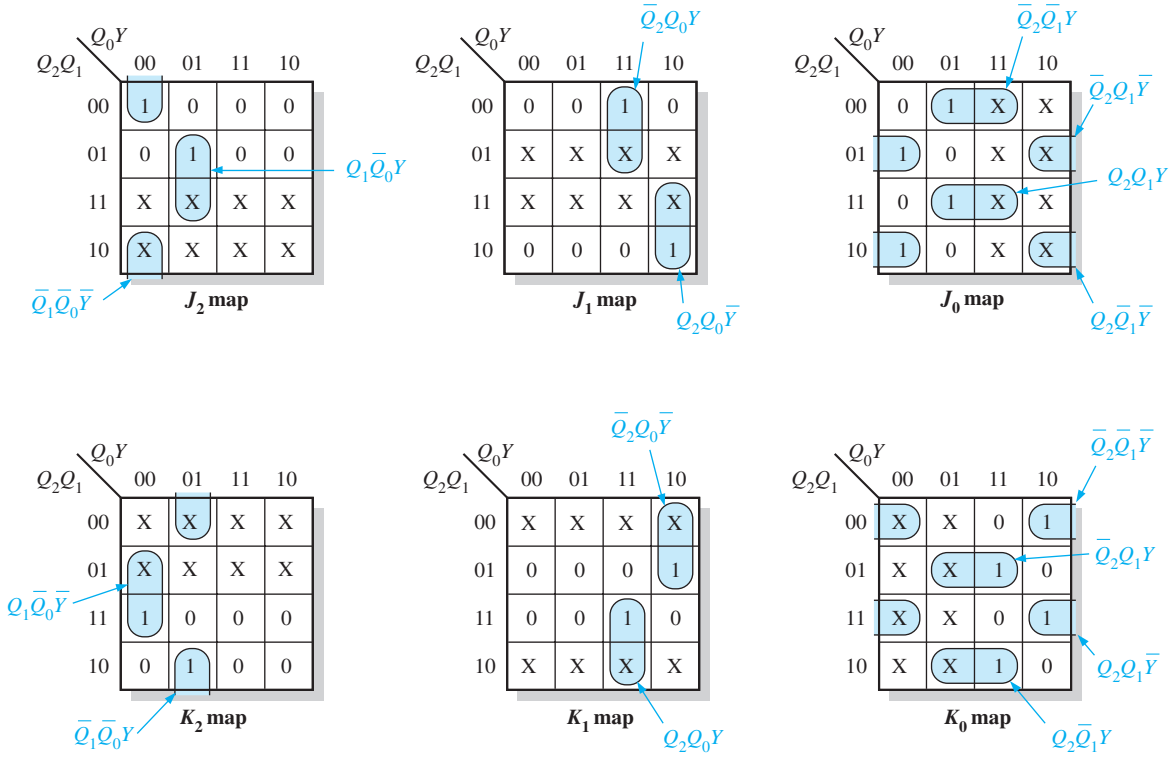


FIGURE 9–34 J and K maps for Table 9–12. The UP/$\overline{\text{DOWN}}$ control input, Y, is treated as a fourth variable.

Step 5: The 1s are combined in the largest possible groupings, with "don't cares" (Xs) used where possible. The groups are factored, and the expressions for the J and K inputs are as follows:

$$J_0 = Q_2Q_1Y + Q_2\overline{Q}_1\overline{Y} + \overline{Q}_2\overline{Q}_1Y + \overline{Q}_2Q_1\overline{Y} \qquad K_0 = \overline{Q}_2\overline{Q}_1\overline{Y} + \overline{Q}_2Q_1Y + Q_2\overline{Q}_1Y + Q_2Q_1\overline{Y}$$
$$J_1 = \overline{Q}_2Q_0Y + Q_2Q_0\overline{Y} \qquad K_1 = \overline{Q}_2Q_0\overline{Y} + Q_2Q_0Y$$
$$J_2 = Q_1\overline{Q}_0Y + \overline{Q}_1\overline{Q}_0\overline{Y} \qquad K_2 = Q_1\overline{Q}_0\overline{Y} + \overline{Q}_1\overline{Q}_0Y$$

Step 6: The J and K equations are implemented with combinational logic. This step is the Related Problem.

Related Problem

Specify the number of flip-flops, gates, and inverters that are required to implement the logic described in Step 5.

SECTION 9–5 CHECKUP

1. A flip-flop is presently in the RESET state and must go to the SET state on the next clock pulse. What must J and K be?
2. A flip-flop is presently in the SET state and must remain SET on the next clock pulse. What must J and K be?
3. A binary counter is in the $Q_3\overline{Q}_2Q_1\overline{Q}_0 = 1010$ state.
   (a) What is its next state?
   (b) What condition must exist on each flip-flop input to ensure that it goes to the proper next state on the clock pulse?

9–6 Cascaded Counters

Counters can be connected in cascade to achieve higher-modulus operation. In essence, **cascading** means that the last-stage output of one counter drives the input of the next counter.

After completing this section, you should be able to

- Determine the overall modulus of cascaded counters
- Analyze the timing diagram of a cascaded counter configuration
- Use cascaded counters as a frequency divider
- Use cascaded counters to achieve specified truncated sequences

Asynchronous Cascading

An example of two asynchronous counters connected in cascade is shown in Figure 9–35 for a 2-bit and a 3-bit ripple counter. The timing diagram is shown in Figure 9–36. Notice

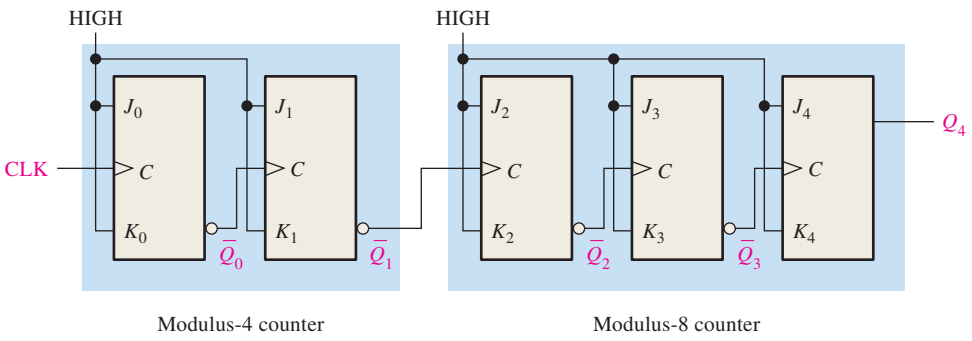


FIGURE 9–35 Two cascaded asynchronous counters (all J and K inputs are HIGH).

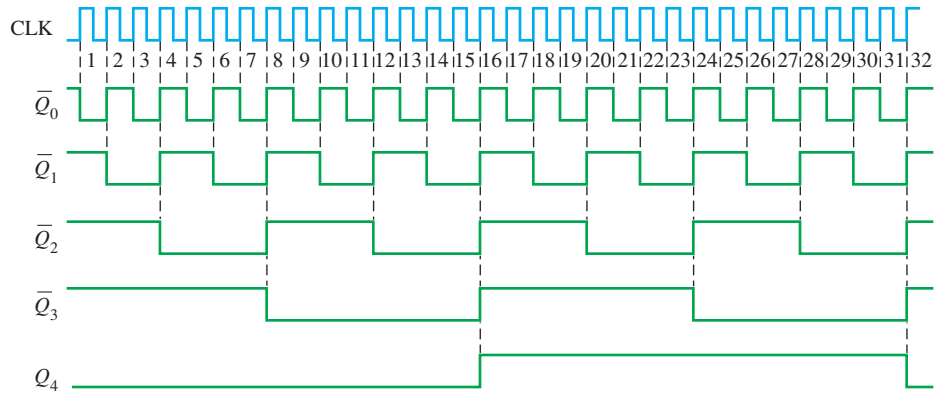


FIGURE 9–36 Timing diagram for the cascaded counter configuration of Figure 9–35.

The overall modulus of cascaded counters is equal to the product of the individual moduli.

that the final output of the modulus-8 counter, Q_4, occurs once for every 32 input clock pulses. The overall modulus of the two cascaded counters is $4 \times 8 = 32$; that is, they act as a divide-by-32 counter.

Synchronous Cascading

When operating synchronous counters in a cascaded configuration, it is necessary to use the count enable and the terminal count functions to achieve higher-modulus operation. On some devices the count enable is labeled simply *CTEN* (or some other designation such as *G*), and terminal count (*TC*) is analogous to ripple clock output (*RCO*) on some IC counters.

Figure 9–37 shows two decade counters connected in cascade. The terminal count (*TC*) output of counter 1 is connected to the count enable (*CTEN*) input of counter 2. Counter 2 is inhibited by the LOW on its *CTEN* input until counter 1 reaches its last, or terminal, state and its terminal count output goes HIGH. This HIGH now enables counter 2, so that when the first clock pulse after counter 1 reaches its terminal count (CLK10), counter 2 goes from its initial state to its second state. Upon completion of the entire second cycle of counter 1 (when counter 1 reaches terminal count the second time), counter 2 is again enabled and advances to its next state. This sequence continues. Since these are decade counters, counter 1 must go through ten complete cycles before counter 2 completes its first cycle. In other words, for every ten cycles of counter 1, counter 2 goes through one cycle. Thus, counter 2 will complete one cycle after one hundred clock pulses. The overall modulus of these two cascaded counters is $10 \times 10 = 100$.

InfoNote

The time stamp counter (TSC), mentioned in the last InfoNote, is a 64-bit counter. It is interesting to observe that if this counter (or any full-modulus 64-bit counter) is clocked at a frequency of 1 GHz, it will take 583 years for it to go through all of its states and reach its terminal count. In contrast, a 32-bit full-modulus counter will exhaust all of its states in approximately 4.3 seconds when clocked at 1 GHz. The difference is astounding.

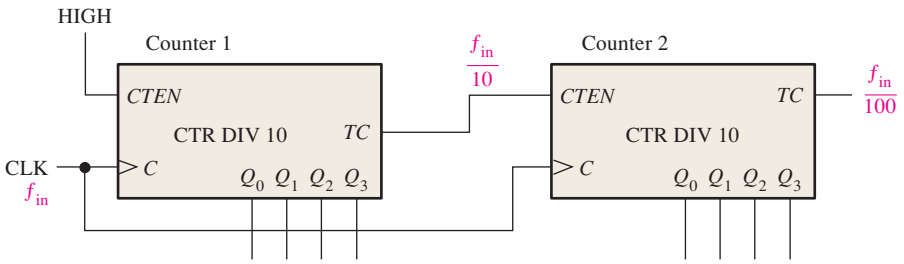


FIGURE 9–37 A modulus-100 counter using two cascaded decade counters.

When viewed as a frequency divider, the circuit of Figure 9–37 divides the input clock frequency by 100. Cascaded counters are often used to divide a high-frequency clock signal to obtain highly accurate pulse frequencies. Cascaded counter configurations used for such purposes are sometimes called *countdown chains*. For example, suppose that you have a basic clock frequency of 1 MHz and you wish to obtain 100 kHz, 10 kHz, and 1 kHz; a series of cascaded decade counters can be used. If the 1 MHz signal is divided by 10, the output is 100 kHz. Then if the 100 kHz signal is divided by 10, the output is 10 kHz. Another division by 10 produces the 1 kHz frequency. The general implementation of this countdown chain is shown in Figure 9–38.

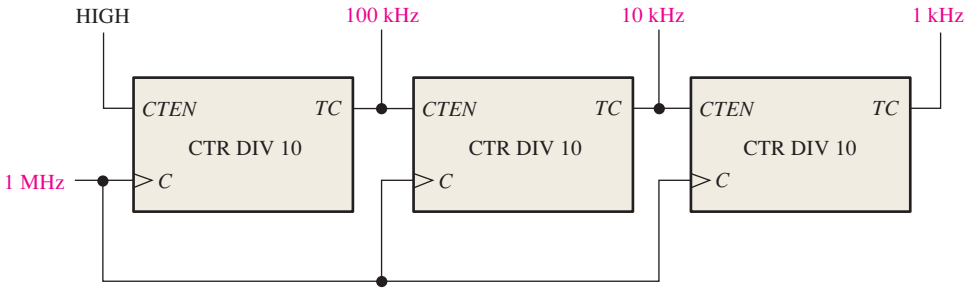


FIGURE 9–38 Three cascaded decade counters forming a divide-by-1000 frequency divider with intermediate divide-by-10 and divide-by-100 outputs.

EXAMPLE 9–6

Determine the overall modulus of the two cascaded counter configurations in Figure 9–39.

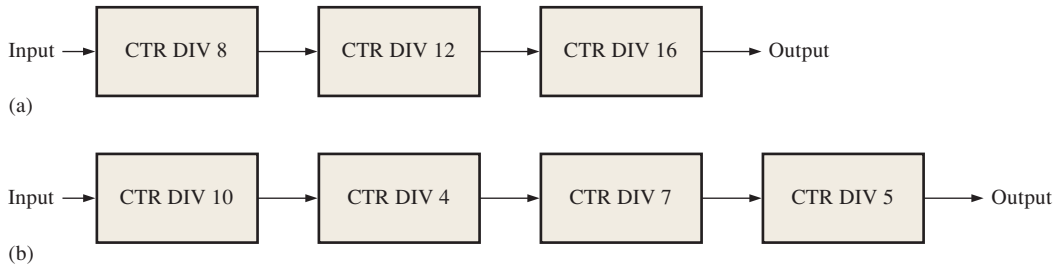


FIGURE 9–39

Solution

In Figure 9–39(a), the overall modulus for the 3-counter configuration is

$$8 \times 12 \times 16 = \mathbf{1536}$$

In Figure 9–39(b), the overall modulus for the 4-counter configuration is

$$10 \times 4 \times 7 \times 5 = \mathbf{1400}$$

Related Problem

How many cascaded decade counters are required to divide a clock frequency by 100,000?

EXAMPLE 9–7

Use 74HC190 up/down decade counters connected in the UP mode to obtain a 10 kHz waveform from a 1 MHz clock. Show the logic diagram.

Solution

To obtain 10 kHz from a 1 MHz clock requires a division factor of 100. Two 74HC190 counters must be cascaded as shown in Figure 9–40. The left counter produces a terminal count (*MAX/MIN*) pulse for every 10 clock pulses. The right counter produces a terminal count (*MAX/MIN*) pulse for every 100 clock pulses.

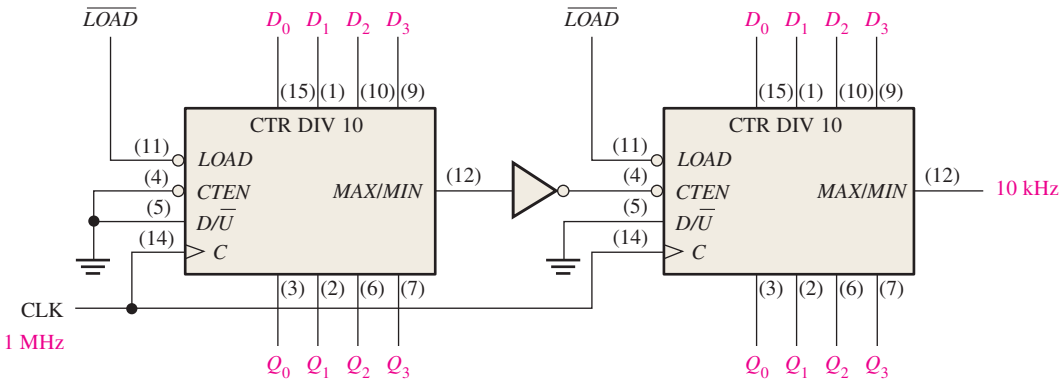


FIGURE 9–40 A divide-by-100 counter using two 74HC190 up/down decade counters connected for the up sequence.

Related Problem

Determine the frequency of the waveform at the Q_0 output of the second counter (the one on the right) in Figure 9–40.

Cascaded Counters with Truncated Sequences

The preceding discussion has shown how to achieve an overall modulus (divide-by-factor) that is the product of the individual moduli of all the cascaded counters. This can be considered *full-modulus cascading*.

Often an application requires an overall modulus that is less than that achieved by full-modulus cascading. That is, a truncated sequence must be implemented with cascaded counters. To illustrate this method, we will use the cascaded counter configuration in Figure 9–41. This particular circuit uses four 74HC161 4-bit synchronous binary counters. If these four counters (sixteen bits total) were cascaded in a full-modulus arrangement, the modulus would be

$$2^{16} = 65{,}536$$

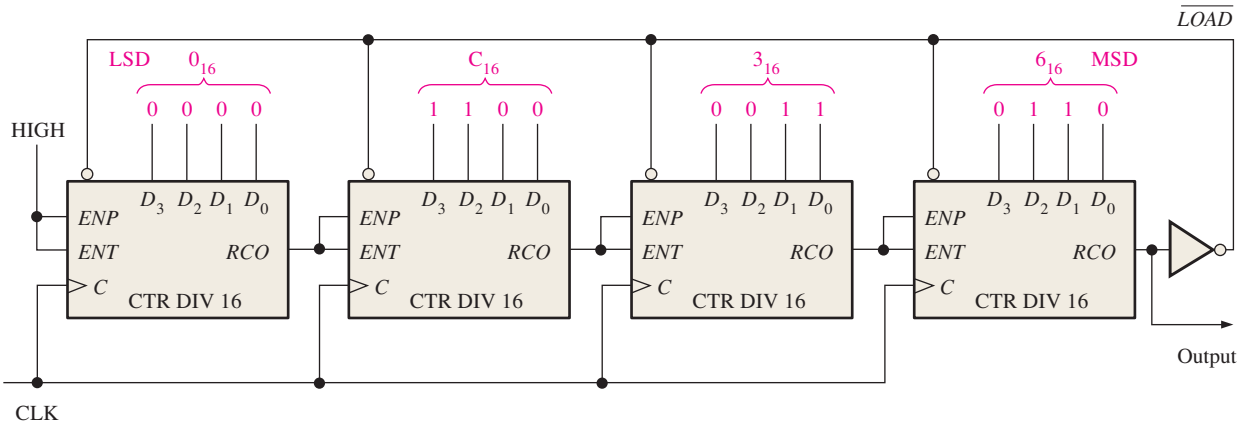


FIGURE 9–41 A divide-by-40,000 counter using 74HC161 4-bit binary counters. Note that each of the parallel data inputs is shown in binary order (the right-most bit D_0 is the LSB in each counter).

Let's assume that a certain application requires a divide-by-40,000 counter (modulus 40,000). The difference between 65,536 and 40,000 is 25,536, which is the number of states that must be *deleted* from the full-modulus sequence. The technique used in the circuit of Figure 9–41 is to preset the cascaded counter to 25,536 (63C0 in hexadecimal) each time it recycles, so that it will count from 25,536 up to 65,535 on each full cycle. Therefore, each full cycle of the counter consists of 40,000 states.

Notice in Figure 9–41 that the *RCO* output of the right-most counter is inverted and applied to the $\overline{LOAD}$ input of each 4-bit counter. Each time the count reaches its terminal value of 65,535, which is 1111111111111111_2, *RCO* goes HIGH and causes the number on the parallel data inputs ($63C0_{16}$) to be synchronously loaded into the counter with the clock pulse. Thus, there is one *RCO* pulse from the right-most 4-bit counter for every 40,000 clock pulses.

With this technique any modulus can be achieved by synchronous loading of the counter to the appropriate initial state on each cycle.

SECTION 9–6 CHECKUP

1. How many decade counters are necessary to implement a divide-by-1000 (modulus-1000) counter? A divide-by-10,000?
2. Show with general block diagrams how to achieve each of the following, using a flip-flop, a decade counter, and a 4-bit binary counter, or any combination of these:
   - **(a)** Divide-by-20 counter
   - **(b)** Divide-by-32 counter
   - **(c)** Divide-by-160 counter
   - **(d)** Divide-by-320 counter

9–7 Counter Decoding

In many applications, it is necessary that some or all of the counter states be decoded. The decoding of a counter involves using decoders or logic gates to determine when the counter is in a certain binary state in its sequence. For instance, the terminal count function previously discussed is a single decoded state (the last state) in the counter sequence.

After completing this section, you should be able to

- Implement the decoding logic for any given state in a counter sequence
- Explain why glitches occur in counter decoding logic
- Use the method of strobing to eliminate decoding glitches

Suppose that you wish to decode binary state 6 (110) of a 3-bit binary counter. When $Q_2 = 1$, $Q_1 = 1$, and $Q_0 = 0$, a HIGH appears on the output of the decoding gate, indicating that the counter is at state 6. This can be done as shown in Figure 9–42. This is called *active-HIGH decoding*. Replacing the AND gate with a NAND gate provides active-LOW decoding.

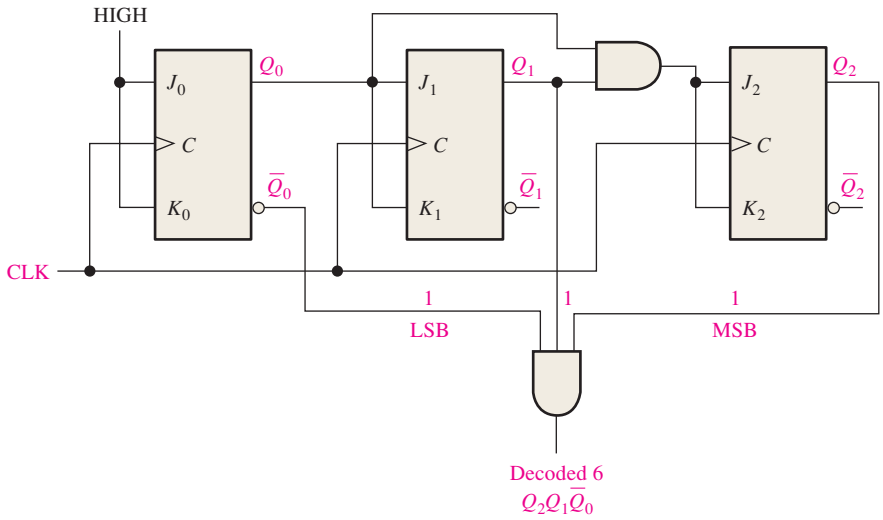


FIGURE 9–42 Decoding of state 6 (110). Open file F09-42 to verify operation.

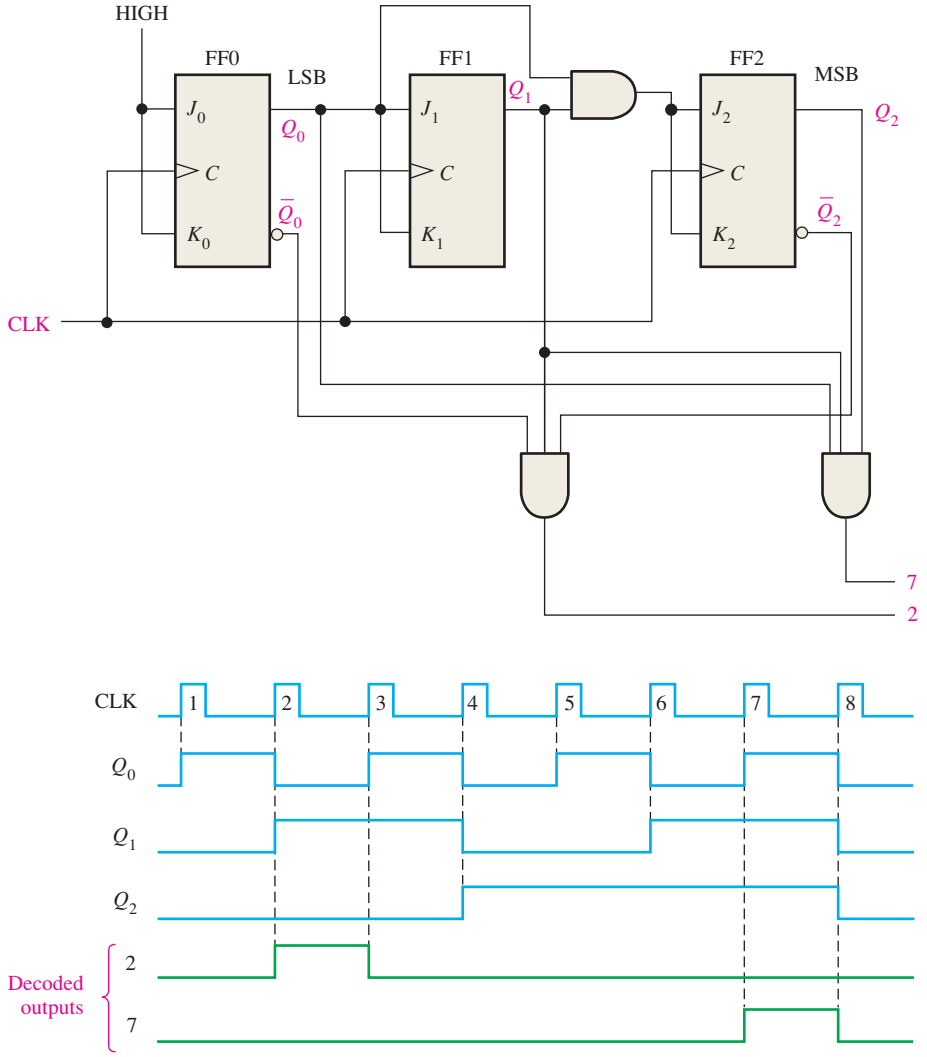
MultiSim

EXAMPLE 9–8

Implement the decoding of binary state 2 and binary state 7 of a 3-bit synchronous counter. Show the entire counter timing diagram and the output waveforms of the decoding gates. Binary 2 = $\overline{Q}_2Q_1\overline{Q}_0$ and binary 7 = $Q_2Q_1Q_0$.

Solution

See Figure 9–43. The 3-bit counter was originally discussed in Section 9–3 (Figure 9–15).



MultiSim

FIGURE 9–43 A 3-bit counter with active-HIGH decoding of count 2 and count 7. Open file F09-43 to verify operation.

Related Problem

Show the logic for decoding state 5 in the 3-bit counter.

Decoding Glitches

A glitch is an unwanted spike of voltage.

The problem of glitches produced by the decoding process was discussed in Chapter 6. As you have learned, the propagation delays due to the ripple effect in asynchronous counters create transitional states in which the counter outputs are changing at slightly different times. These transitional states produce undesired voltage spikes of short duration (glitches) on the outputs of a decoder connected to the counter. The glitch problem can also occur to some degree with synchronous counters because the propagation delays from the clock to the Q outputs of each flip-flop in a counter can vary slightly.

Figure 9–44 shows a basic asynchronous BCD decade counter connected to a BCD-to-decimal decoder. To see what happens in this case, let's look at a timing diagram in which the propagation delays are taken into account, as shown in Figure 9–45. Notice that these delays cause false states of short duration. The value of the false binary state at each critical transition is indicated on the diagram. The resulting glitches can be seen on the decoder outputs.

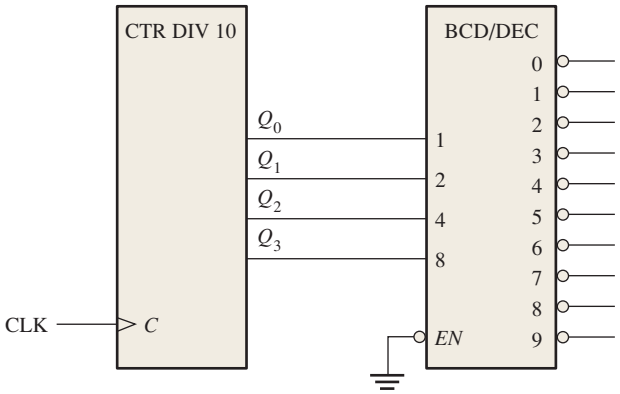


FIGURE 9–44 A basic decade (BCD) counter and decoder.

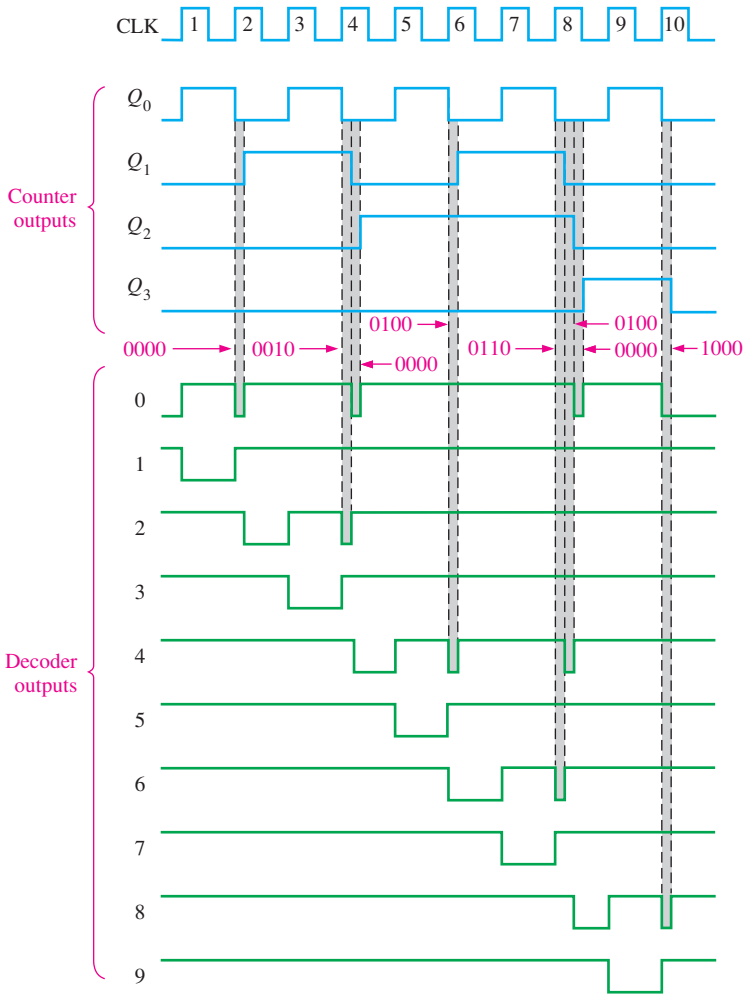


FIGURE 9–45 Outputs with glitches from the decoder in Figure 9–44. Glitch widths are exaggerated for illustration and are usually only a few nanoseconds wide.

One way to eliminate the glitches is to enable the decoded outputs at a time after the glitches have had time to disappear. This method is known as *strobing* and can be accomplished in the case of an active-HIGH clock by using the LOW level of the clock to enable the decoder, as shown in Figure 9–46. The resulting improved timing diagram is shown in Figure 9–47.

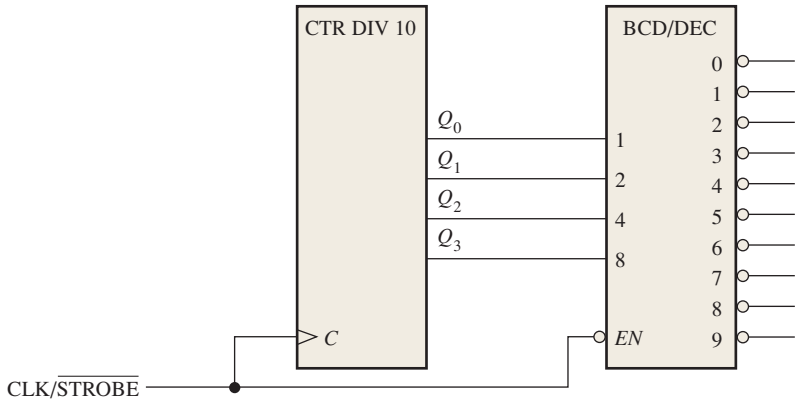


FIGURE 9–46 The basic decade counter and decoder with strobing to eliminate glitches.

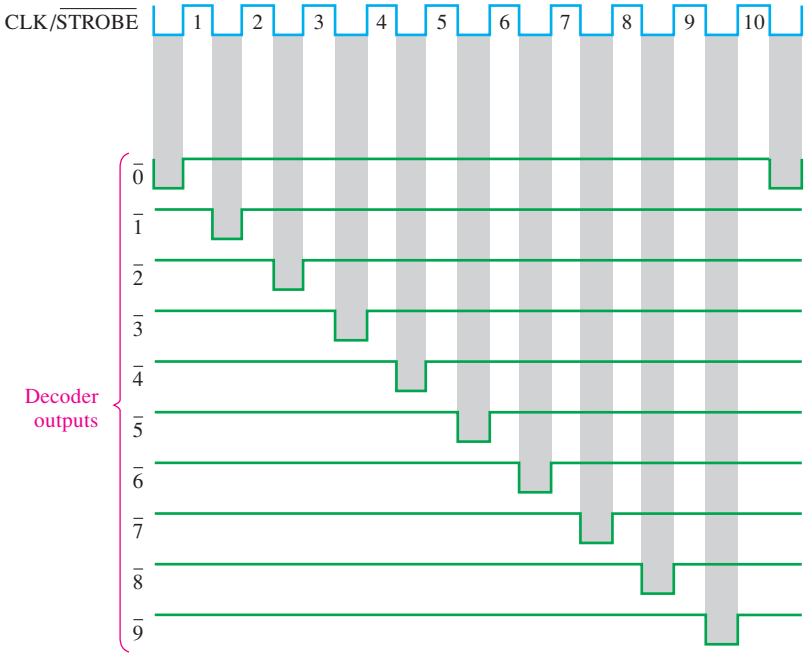


FIGURE 9–47 Strobed decoder outputs for the circuit of Figure 9–46.

SECTION 9–7 CHECKUP

1. What transitional states are possible when a 4-bit asynchronous binary counter changes from

(a) count 2 to count 3 **(b)** count 3 to count 4

(c) count 10_{10} to count 11_{10} **(d)** count 15 to count 0

9–8 Counter Applications

The digital counter is a useful and versatile device that is found in many applications. In this section, some representative counter applications are presented.

After completing this section, you should be able to

- Describe how counters are used in a basic digital clock system
- Explain how a divide-by-60 counter is implemented and how it is used in a digital clock

- Explain how the hours counter is implemented
- Discuss the application of a counter in an automobile parking control system
- Describe how a counter is used in the process of parallel-to-serial data conversion

A Digital Clock

A common example of a counter application is in timekeeping systems. Figure 9–48 is a simplified logic diagram of a digital clock that displays seconds, minutes, and hours. First, a 60 Hz sinusoidal ac voltage is converted to a 60 Hz pulse waveform and divided down to a 1 Hz pulse waveform by a divide-by-60 counter formed by a divide-by-10 counter followed by a divide-by-6 counter. Both the *seconds* and *minutes* counts are also produced by divide-by-60 counters, the details of which are shown in Figure 9–49. These counters count from 0 to 59 and then recycle to 0; synchronous decade counters are used in this particular implementation. Notice that the divide-by-6 portion is formed with a decade counter with a truncated sequence achieved by using the decoder count 6 to asynchronously clear the counter. The terminal count, 59, is also decoded to enable the next counter in the chain.

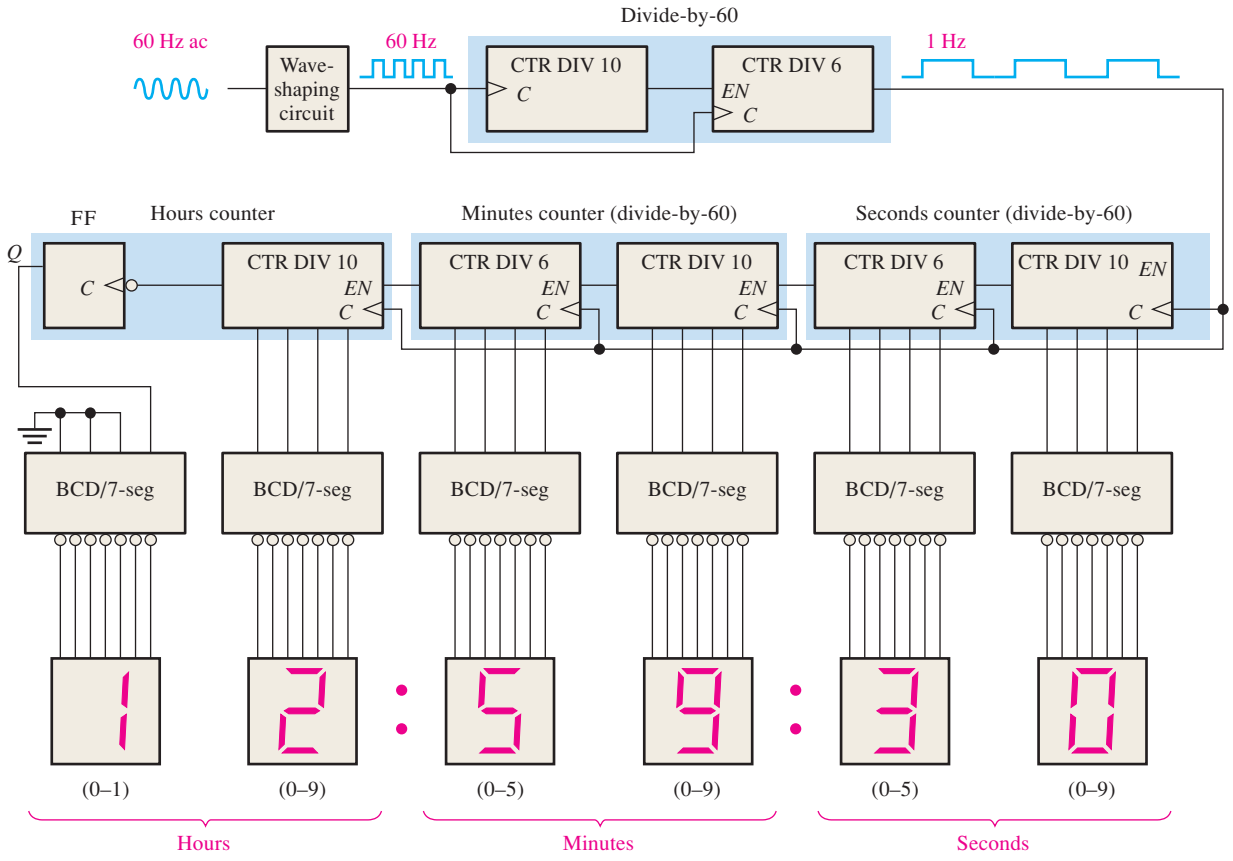


FIGURE 9–48 Simplified logic diagram for a 12-hour digital clock. Logic details using specific devices are shown in Figures 9–49 and 9–50.

The *hours* counter is implemented with a decade counter and a flip-flop as shown in Figure 9–50. Consider that initially both the decade counter and the flip-flop are RESET, and the decode-12 gate and decode-9 gate outputs are HIGH. The decade counter advances through all of its states from zero to nine, and on the clock pulse that recycles it from nine back to zero, the flip-flop goes to the SET state ($J = 1, K = 0$). This illuminates a 1 on the tens-of-hours display. The total count is now ten (the decade counter is in the zero state and the flip-flop is SET).

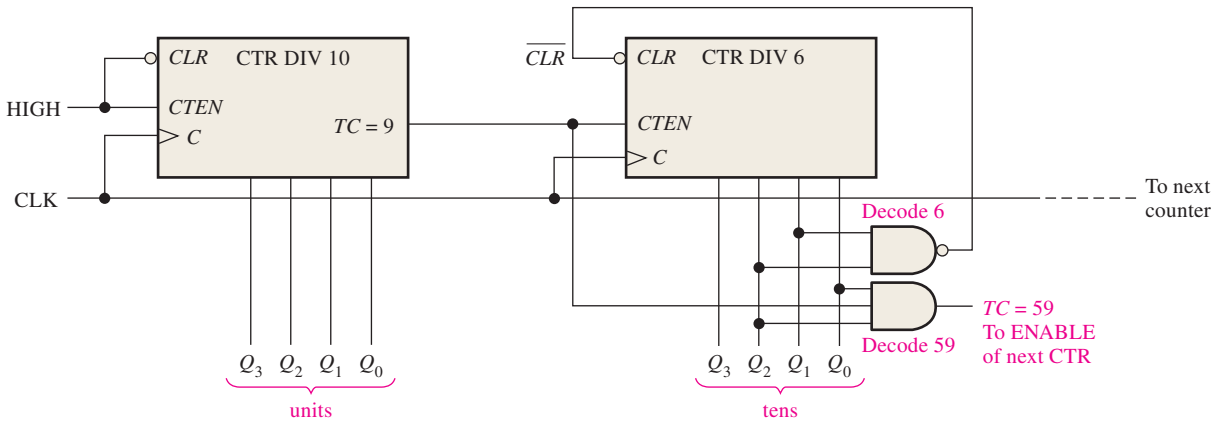


FIGURE 9–49 Logic diagram of typical divide-by-60 counter using synchronous decade counters. Note that the outputs are in binary order (the right-most bit is the LSB).

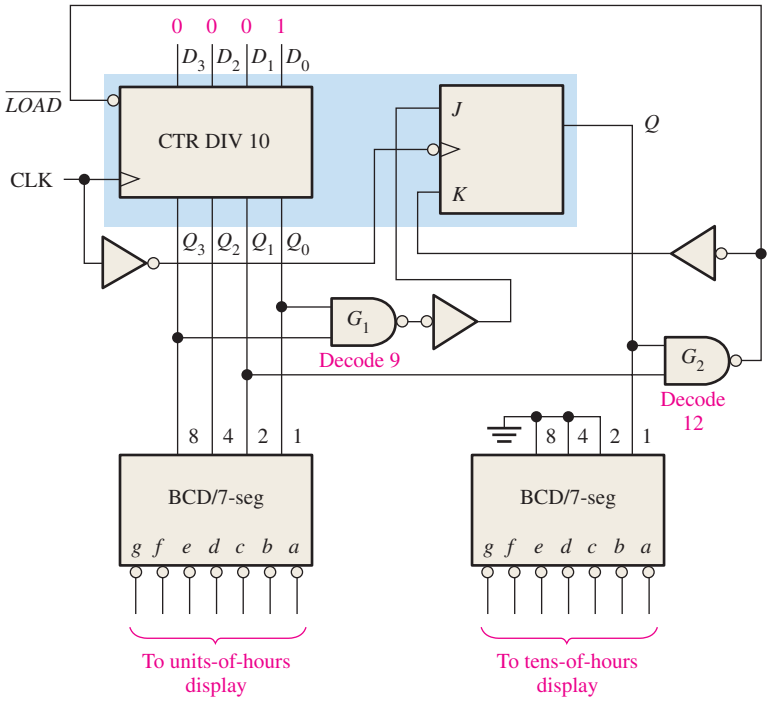


FIGURE 9–50 Logic diagram for hours counter and decoders. Note that on the counter inputs and outputs, the right-most bit is the LSB.

Next, the total count advances to eleven and then to twelve. In state 12 the Q_2 output of the decade counter is HIGH, the flip-flop is still SET, and thus the decode-12 gate output is LOW. This activates the $\overline{LOAD}$ input of the decade counter. On the next clock pulse, the decade counter is preset to 0001 from the data inputs, and the flip-flop is RESET ($J = 0$, $K = 1$). As you can see, this logic always causes the counter to recycle from twelve back to one rather than back to zero.

Automobile Parking Control

This counter example illustrates the use of an up/down counter to solve an everyday problem. The problem is to devise a means of monitoring available spaces in a one-hundred-space parking garage and provide for an indication of a full condition by illuminating a display sign and lowering a gate bar at the entrance.

A system that solves this problem consists of optoelectronic sensors at the entrance and exit of the garage, an up/down counter and associated circuitry, and an interface circuit that uses the counter output to turn the FULL sign on or off as required and lower or raise the gate bar at the entrance. A general block diagram of this system is shown in Figure 9–51.

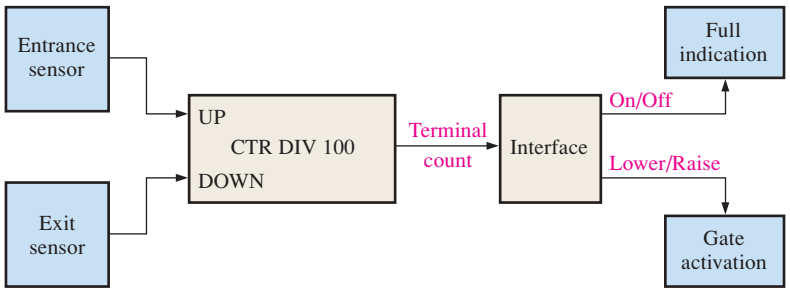


FIGURE 9–51 Functional block diagram for parking garage control.

A logic diagram of the up/down counter is shown in Figure 9–52. It consists of two cascaded up/down decade counters. The operation is described in the following paragraphs.

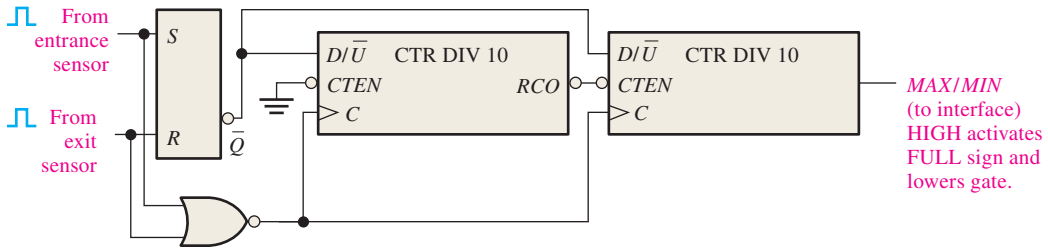


FIGURE 9–52 Logic diagram for modulus-100 up/down counter for automobile parking control.

The counter is initially preset to 0 using the parallel data inputs, which are not shown. Each automobile entering the garage breaks a light beam, activating a sensor that produces an electrical pulse. This positive pulse sets the S-R latch on its leading edge. The LOW on the $\overline{Q}$ output of the latch puts the counter in the UP mode. Also, the sensor pulse goes through the NOR gate and clocks the counter on the LOW-to-HIGH transition of its trailing edge. Each time an automobile enters the garage, the counter is advanced by one **(incremented)**. When the one-hundredth automobile enters, the counter goes to its last state (100_{10}). The *MAX/MIN* output goes HIGH and activates the interface circuit (no detail), which lights the FULL sign and lowers the gate bar to prevent further entry.

Incrementing a counter increases its count by one.

When an automobile exits, an optoelectronic sensor produces a positive pulse, which resets the S-R latch and puts the counter in the DOWN mode. The trailing edge of the clock decreases the count by one **(decremented)**. If the garage is full and an automobile leaves, the *MAX/MIN* output of the counter goes LOW, turning off the FULL sign and raising the gate.

Decrementing a counter decreases its count by one.

Parallel-to-Serial Data Conversion (Multiplexing)

A simplified example of data transmission using multiplexing and demultiplexing techniques was introduced in Chapter 6. Essentially, the parallel data bits on the multiplexer inputs are converted to serial data bits on the single transmission line. A group of bits appearing simultaneously on parallel lines is called *parallel data.* A group of bits appearing on a single line in a time sequence is called *serial data.*

Parallel-to-serial conversion is normally accomplished by the use of a counter to provide a binary sequence for the data-select inputs of a data selector/multiplexer, as illustrated in Figure 9–53. The *Q* outputs of the modulus-8 counter are connected to the data-select inputs of an 8-bit multiplexer.

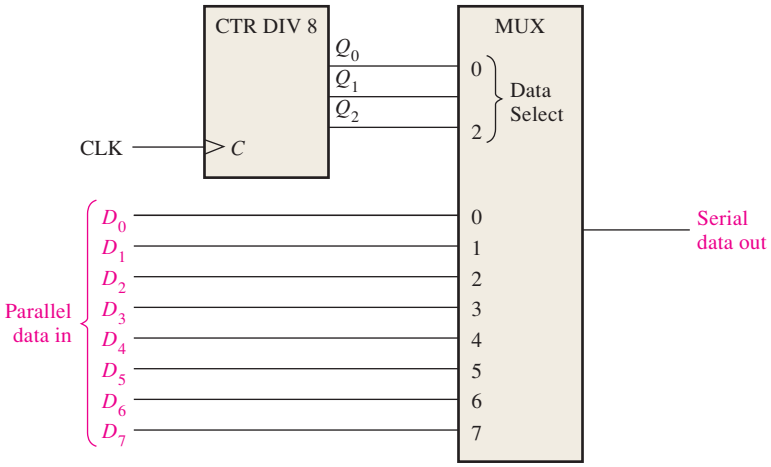


FIGURE 9–53 Parallel-to-serial data conversion logic.

Figure 9–54 is a timing diagram illustrating the operation of this circuit. The first byte (eight-bit group) of parallel data is applied to the multiplexer inputs. As the counter goes through a binary sequence from zero to seven, each bit, beginning with D_0, is sequentially

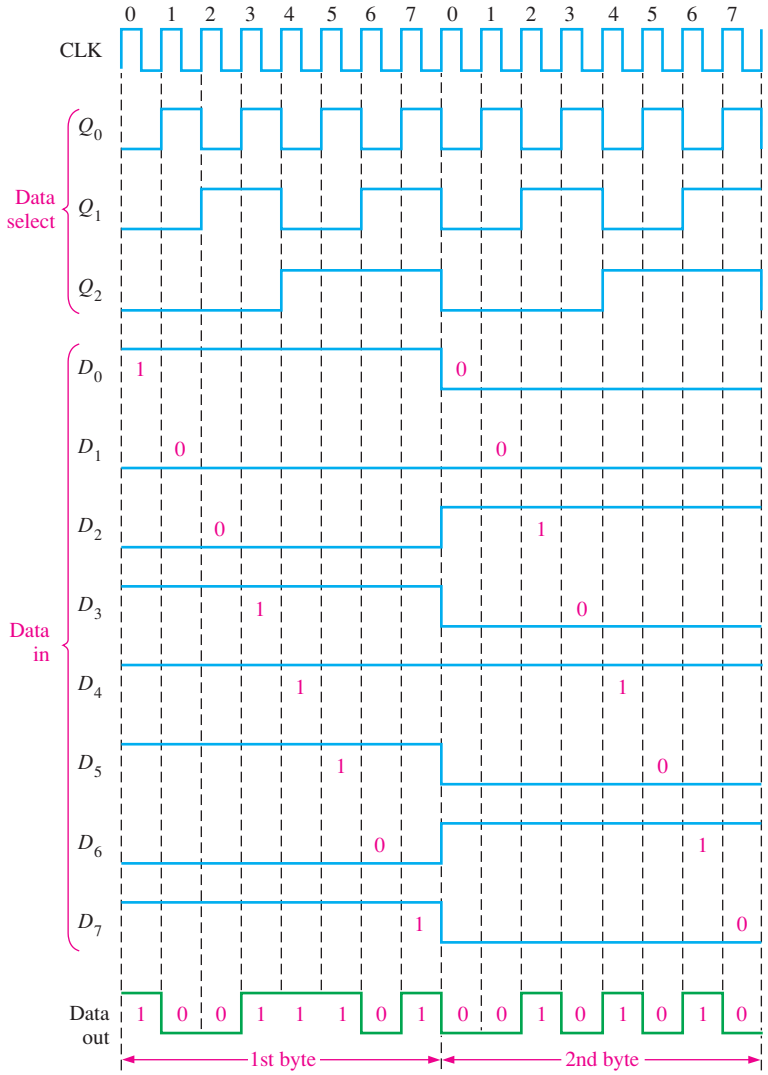


FIGURE 9–54 Example of parallel-to-serial conversion timing for the circuit in Figure 9–53.

InfoNote

Computers contain an internal counter that can be programmed for various frequencies and tone durations, thus producing "music." To select a particular tone, the programmed instruction selects a divisor that is sent to the counter. The divisor sets the counter up to divide the basic peripheral clock frequency to produce an audio tone. The duration of a tone can also be set by a programmed instruction; thus, a basic counter is used to produce melodies by controlling the frequency and duration of tones.

selected and passed through the multiplexer to the output line. After eight clock pulses the data byte has been converted to a serial format and sent out on the transmission line. When the counter recycles back to 0, the next byte is applied to the data inputs and is sequentially converted to serial form as the counter cycles through its eight states. This process continues repeatedly as each parallel byte is converted to a serial byte.

SECTION 9–8 CHECKUP

1. Explain the purpose of each NAND gate in Figure 9–50.
2. Identify the two recycle conditions for the hours counter in Figure 9–48, and explain the reason for each.

9–9 Logic Symbols with Dependency Notation

Up to this point, the logic symbols with dependency notation specified in ANSI/IEEE Standard 91-1984 have been introduced on a limited basis. In many cases, the symbols do not deviate greatly from the traditional symbols. A significant departure does occur, however, for some devices, including counters and other more complex devices. Although we will continue to use primarily the more traditional symbols throughout this book, a brief coverage of logic symbols with dependency notation is provided. A specific IC counter is used as an example.

After completing this section, you should be able to

- Interpret logic symbols that include dependency notation
- Identify the common block and the individual elements of a counter symbol
- Interpret the qualifying symbol
- Discuss control dependency
- Discuss mode dependency
- Discuss AND dependency

Dependency notation is fundamental to the ANSI/IEEE standard. Dependency notation is used in conjunction with the logic symbols to specify the relationships of inputs and outputs so that the logical operation of a given device can be determined entirely from its logic symbol without a prior knowledge of the details of its internal structure and without a detailed logic diagram for reference. This coverage of a specific logic symbol with dependency notation is intended to aid in the interpretation of other such symbols that you may encounter in the future.

The 74HC163 4-bit synchronous binary counter is used for illustration. For comparison, Figure 9–55 shows a traditional block symbol and the ANSI/IEEE symbol with dependency notation. Basic descriptions of the symbol and the dependency notation follow.

Common Control Block

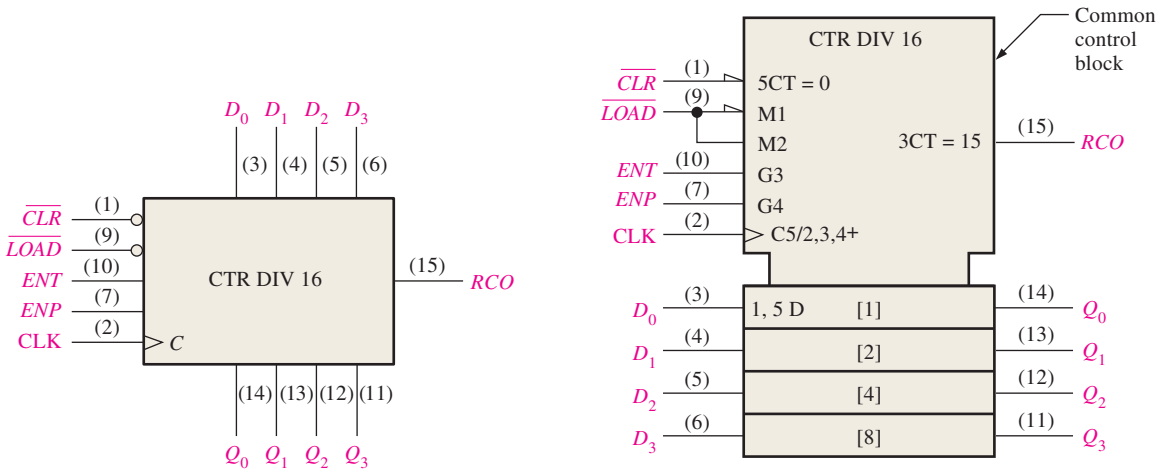
The upper block with notched corners in Figure 9–55(b) has inputs and an output that are considered common to all elements in the device and not unique to any one of the elements.

Individual Elements

The lower block in Figure 9–55(b), which is partitioned into four abutted sections, represents the four storage elements (D flip-flops) in the counter, with inputs D_0, D_1, D_2, and D_3 and outputs Q_0, Q_1, Q_2, and Q_3.

Qualifying Symbol

The label “CTR DIV 16” in Figure 9–55(b) identifies the device as a counter (CTR) with sixteen states (DIV 16).



(a) Traditional block symbol

(b) ANSI/IEEE Std. 91-1984 logic symbol

FIGURE 9–55 The 74HC163 4-bit synchronous counter.

Control Dependency (C)

As shown in Figure 9–55(b), the letter *C* denotes control dependency. Control inputs usually enable or disable the data inputs (*D*, *J*, *K*, *S*, and *R*) of a storage element. The *C* input is usually the clock input. In this case the digit 5 following *C* (C5/2,3,4+) indicates that the inputs labeled with a 5 prefix are dependent on the clock (synchronous with the clock). For example, 5CT = 0 on the $\overline{CLR}$ input indicates that the clear function is dependent on the clock; that is, it is a synchronous clear. When the $\overline{CLR}$ input is LOW (0), the counter is reset to zero ($CT = 0$) on the triggering edge of the clock pulse. Also, the 5 D label at the input of storage element [1] indicates that the data storage is dependent on (synchronous with) the clock. All labels in the [1] storage element apply to the [2], [4], and [8] elements below it since they are not labeled differently.

Mode Dependency (M)

As shown in Figure 9–55(b), the letter *M* denotes mode dependency. This label is used to indicate how the functions of various inputs or outputs depend on the mode in which the device is operating. In this case the device has two modes of operation. When the $\overline{LOAD}$ input is LOW (0), as indicated by the triangle input, the counter is in a preset mode (M1) in which the input data (D_0, D_1, D_2, and D_3) are synchronously loaded into the four flip-flops. The digit 1 following *M* in M1 and the 1 in the label 1, 5 D show a dependency relationship and indicate that input data are stored only when the device is in the preset mode (M1), in which $\overline{LOAD} = 0$. When the $\overline{LOAD}$ input is HIGH (1), the counter advances through its normal binary sequence, as indicated by M2 and the 2 in C5/2,3,4+.

AND Dependency (G)

As shown in Figure 9–55(b), the letter *G* denotes AND dependency, indicating that an input designated with *G* followed by a digit is ANDed with any other input or output having the same digit as a prefix in its label. In this particular example, the G3 at the *ENT* input and the 3CT = 15 at the *RCO* output are related, as indicated by the 3, and that relationship is an AND dependency, indicated by the *G*. This tells us that *ENT* must be HIGH (no triangle on the input) *and* the count must be fifteen ($CT = 15$) for the *RCO* output to be HIGH.

Also, the digits 2, 3, and 4 in the label C5/2,3,4+ indicate that the counter advances through its states when $\overline{LOAD} = 1$, as indicated by the mode dependency label M2, and when $ENT = 1$ and $ENP = 1$, as indicated by the AND dependency labels G3 and G4. The + indicates that the counter advances by one count when these conditions exist.

SECTION 9–9 CHECKUP

1. In dependency notation, what do the letters *C*, *M*, and *G* stand for?
2. By what letter is data storage denoted?

9–10 Troubleshooting

The troubleshooting of counters can be simple or quite involved, depending on the type of counter and the type of fault. This section will give you some insight into how to approach the troubleshooting of sequential circuits.

After completing this section, you should be able to

- Detect a faulty counter
- Isolate faults in maximum-modulus cascaded counters
- Isolate faults in cascaded counters with truncated sequences
- Determine faults in counters implemented with individual flip-flops

Counters

The symptom for a faulty counter is usually that it does not advance its count. If this is the case, then check power and ground on the chip. Look at these lines with a scope to make sure there is no noise present (a noisy ground may actually be open). Check that there are clock pulses and that they have the correct amplitude and rise time and that there is not extraneous noise on the line. (Sometimes clock pulses can be loaded down by other ICs, making it appear that the counter is faulty when it is not). If power, ground, and the clock pulses are okay, check all inputs (including enable, load, and clear inputs), to see that they are connected correctly and that the logic is correct. An open input can cause a counter to work correctly some of the time—inputs should never be left open, even if they are not used. (An unused input should be connected to an inactive level). If the counter is stuck in a state and the clock is present, determine what input should be present to advance the counter. This may point to a faulty input (including clear or load inputs), which can be caused by logic elsewhere in the circuit. If inputs are all checked okay, an output may be pulled LOW or HIGH by an external short or open (or another faulty IC), keeping the output from advancing.

Cascaded Counters with Maximum Modulus

A failure in one of the counters in a chain of cascaded counters can affect all the counters that follow it. For example, if a count enable input opens, it effectively acts as a HIGH (for TTL logic), and the counter is always enabled. This type of failure in one of the counters will cause that counter to run at the full clock rate and will also cause all the succeeding counters to run at higher than normal rates. This is illustrated in Figure 9–56 for a divide-by-1000 cascaded counter arrangement where an open enable (*CTEN*) input acts as a TTL HIGH and continuously enables the second counter. Other faults that can affect "downstream" counter stages are open or shorted clock inputs or terminal count outputs. In some of these situations, pulse activity can be observed, but it may be at the wrong frequency. Exact frequency or frequency ratio measurements must be made.

Cascaded Counters with Truncated Sequences

The count sequence of a cascaded counter with a truncated sequence, such as that in Figure 9–57, can be affected by other types of faults in addition to those mentioned for maximum-modulus cascaded counters. For example, a failure in one of the parallel data inputs, the $\overline{LOAD}$ input, or the inverter can alter the preset count and thus change the modulus of the counter.

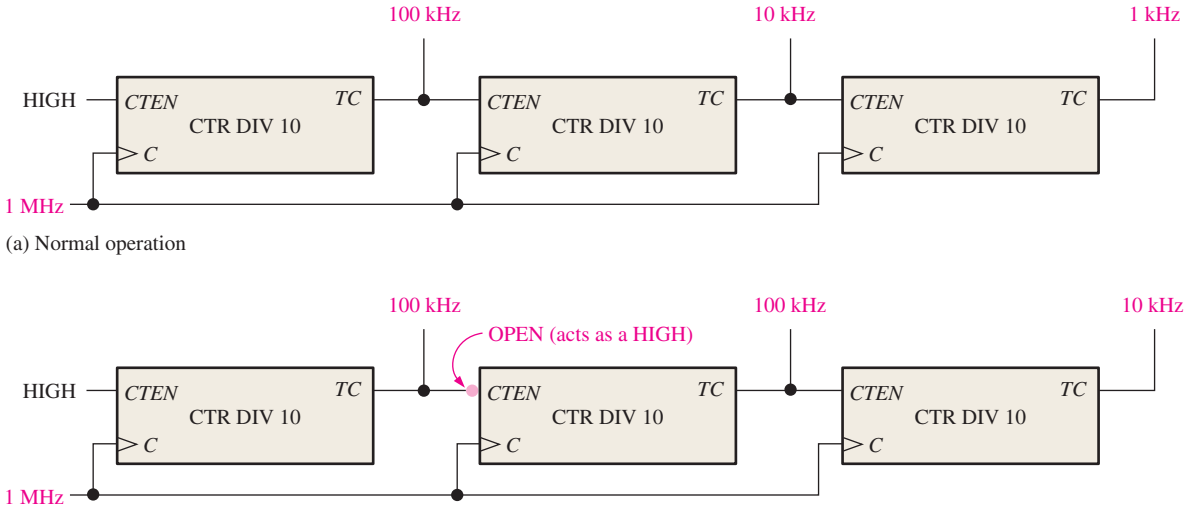


FIGURE 9–56 Example of a failure that affects following counters in a cascaded arrangement.

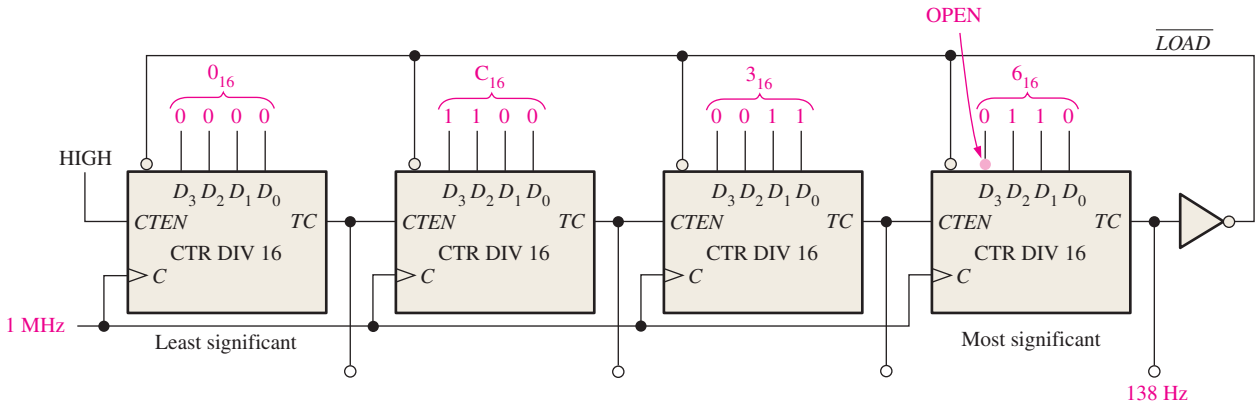


FIGURE 9–57 Example of a failure in a cascaded counter with a truncated sequence.

For example, suppose the D_3 input of the most significant counter in Figure 9–57 is open and acts as a HIGH. Instead of 6_{16} (0110) being preset into the counter, E_{16} (1110) is preset in. So, instead of beginning with $63C0_{16}$ ($25{,}536_{10}$) each time the counter recycles, the sequence will begin with $E3C0_{16}$ ($58{,}304_{10}$). This changes the modulus of the counter from 40,000 to 65,536 − 58,304 = 7232.

To check this counter, apply a known clock frequency, for example 1 MHz, and measure the output frequency at the final terminal count output. If the counter is operating properly, the output frequency is

$$f_{out} = \frac{f_{in}}{\text{modulus}} = \frac{1\text{ MHz}}{40{,}000} = 25\text{ Hz}$$

In this case, the specific failure described in the preceding paragraph will cause the output frequency to be

$$f_{out} = \frac{f_{in}}{\text{modulus}} = \frac{1\text{ MHz}}{7232} \cong 138\text{ Hz}$$

EXAMPLE 9–9

Frequency measurements are made on the truncated counter in Figure 9–58 as indicated. Determine if the counter is working properly, and if not, isolate the fault.

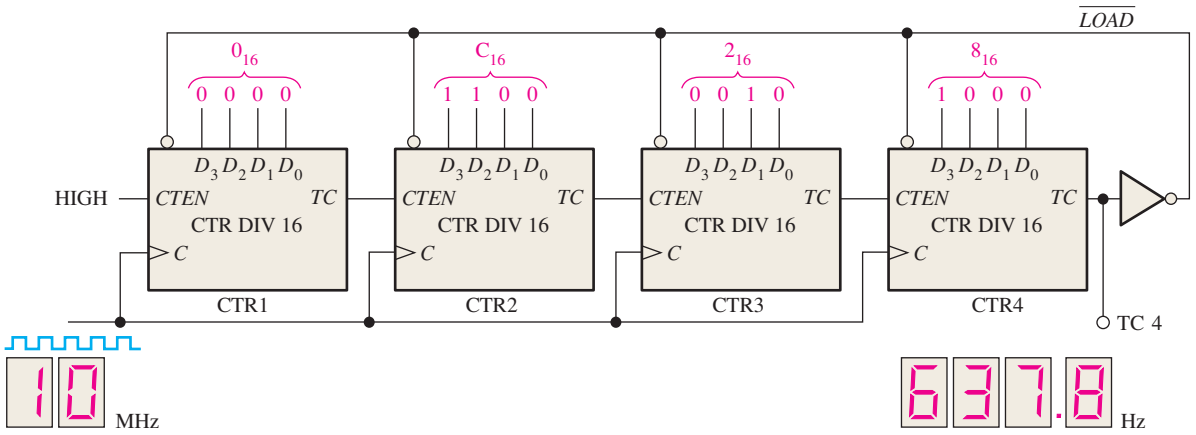


FIGURE 9–58

Solution

Check to see if the frequency measured at TC 4 is correct. If it is, the counter is working properly.

$$\begin{aligned}\text{truncated modulus} &= \text{full modulus} - \text{preset count}\\ &= 16^4 - 82C0_{16}\\ &= 65{,}536 - 33{,}472 = 32{,}064\end{aligned}$$

The correct frequency at TC 4 is

$$f_4 = \frac{10\text{ MHz}}{32{,}064} \cong 312\text{ Hz}$$

There is a problem. The measured frequency of 637.8 Hz does not agree with the correct calculated frequency of 312 Hz.

To find the faulty counter, determine the actual truncated modulus as follows:

$$\text{modulus} = \frac{f_{\text{in}}}{f_{\text{out}}} = \frac{10\text{ MHz}}{637.8\text{ Hz}} = 15{,}679$$

Because the truncated modulus should be 32,064, most likely the counter is being preset to the wrong count when it recycles. The actual preset count is determined as follows:

$$\begin{aligned}\text{truncated modulus} &= \text{full modulus} - \text{preset count}\\ \text{preset count} &= \text{full modulus} - \text{truncated modulus}\\ &= 65{,}536 - 15{,}679\\ &= 49{,}857\\ &= C2C0_{16}\end{aligned}$$

This shows that the counter is being preset to $C2C0_{16}$ instead of $82C0_{16}$ each time it recycles.

Counters 1, 2, and 3 are being preset properly but counter 4 is not. Since $C_{16} = 1100_2$, the D_2 input to counter 4 is HIGH when it should be LOW. This is most likely caused by an **open input.** Check for an external open caused by a bad solder connection, a broken conductor, or a bent pin on the IC. If none can be found, replace the IC and the counter should work properly.

Related Problem

Determine what the output frequency at TC 4 would be if the D_3 input of counter 3 were open.

Counters Implemented with Individual Flip-Flops

Counters implemented with individual flip-flop and gate ICs are sometimes more difficult to troubleshoot because there are many more inputs and outputs with external connections than there are in an IC counter. The sequence of a counter can be altered by a single open or short on an input or output, as Example 9–10 illustrates.

EXAMPLE 9–10

Suppose that you observe the output waveforms (green) that are indicated for the counter in Figure 9–59. Determine if there is a problem with the counter.

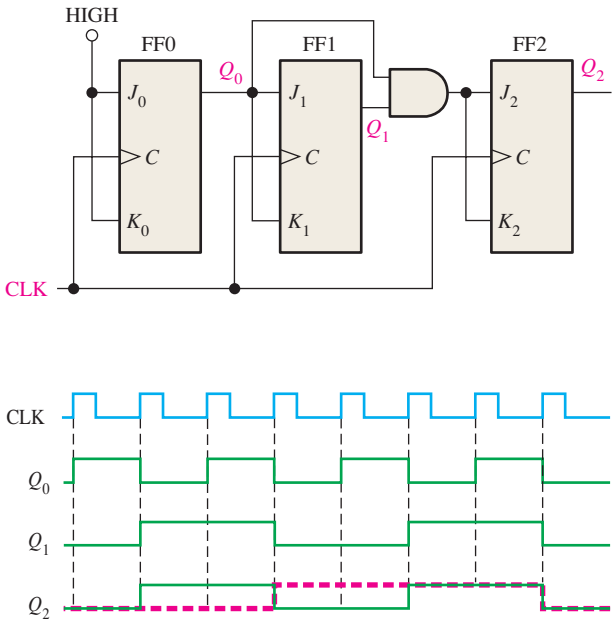


FIGURE 9–59

Solution

The Q_2 waveform is incorrect. The correct waveform is shown as a red dashed line. You can see that the Q_2 waveform looks exactly like the Q_1 waveform, so whatever is causing FF1 to toggle appears to also be controlling FF2.

Checking the *J* and *K* inputs to FF2, you find a waveform that looks like Q_0. This result indicates that Q_0 is somehow getting through the AND gate. The only way this can happen is if the Q_1 input to the AND gate is always HIGH. However, you have seen that Q_1 has a correct waveform. This observation leads to the conclusion that the lower input to the AND gate must be internally open and acting as a HIGH. Replace the AND gate and retest the circuit.

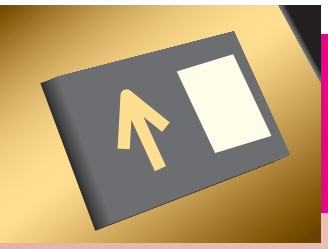
Related Problem

Describe the Q_2 output of the counter in Figure 9–59 if the Q_1 output of FF1 is open.

To observe the time relationship between two digital signals with a dual-trace analog oscilloscope, the proper way to trigger the scope is with the slower of the two signals. The reason for this is that the slower signal has fewer possible trigger points than the faster signal and there will be no ambiguity for starting the sweep. Vertical mode triggering uses a composite of both channels and should never be used for determining absolute time information. Since clock signals are usually the fastest signal in a digital system, they should not be used for triggering.

SECTION 9–10 CHECKUP

1. What failures can cause the counter in Figure 9–56 to have no pulse activity on any of the *TC* outputs?
2. What happens if the inverter in Figure 9–58 develops an open output?



Applied Logic

Elevator Controller: Part 1

This Applied Logic describes the operation and implementation of a service elevator controller for a seven-story building. The controller consists of logic that controls the elevator operation, a counter that determines the floor at which the elevator is located at any given time, and a floor number display. For simplicity, there is only one floor call and one floor request for each elevator cycle. A cycle occurs when the elevator is called to a given floor to pick up a passenger and the passenger is delivered to a requested floor. The elevator sequence for one cycle is shown in Figure 9–60.

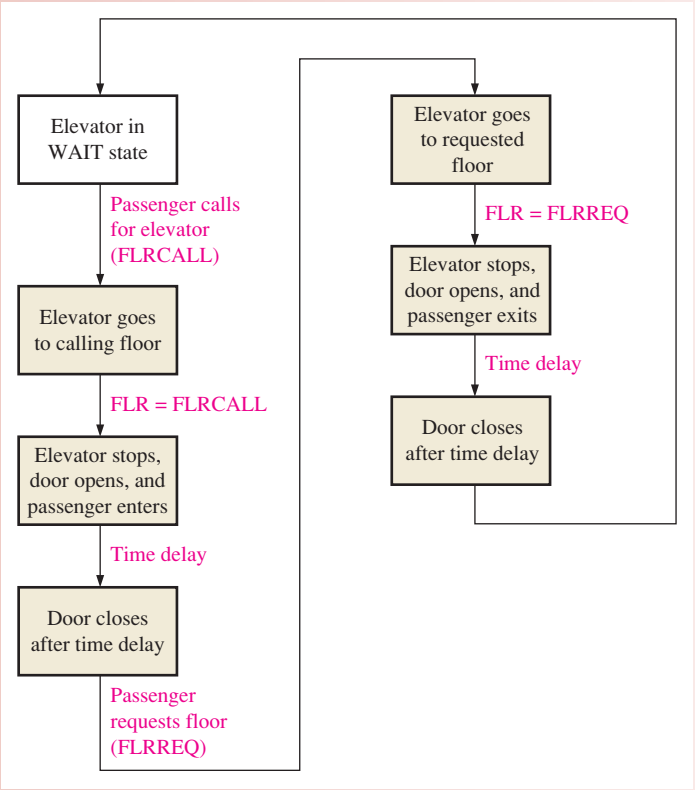


FIGURE 9–60 One cycle of the elevator operation.

The five states in the elevator control sequence are WAIT, DOWN, UP, STOP/OPEN, and CLOSE. In the WAIT state, the elevator is waiting on the last floor serviced for an external call button (FLRCALL) on any floor to be pressed. When there is a call for the elevator from any floor, the appropriate command (UP or DOWN) is issued. When the elevator arrives and stops at the calling floor, the door opens; the person enters and presses a number to request a destination floor. If the number of the requested floor is less than the number of the current floor, the elevator goes into the DOWN mode. If the number of the requested floor is greater than the number of the current floor, the elevator goes into the UP mode. The elevator goes to the STOP/OPEN mode at the requested floor to allow exit. After the door is open for a specified time, it closes and then goes back to the WAIT state until another floor call is received.

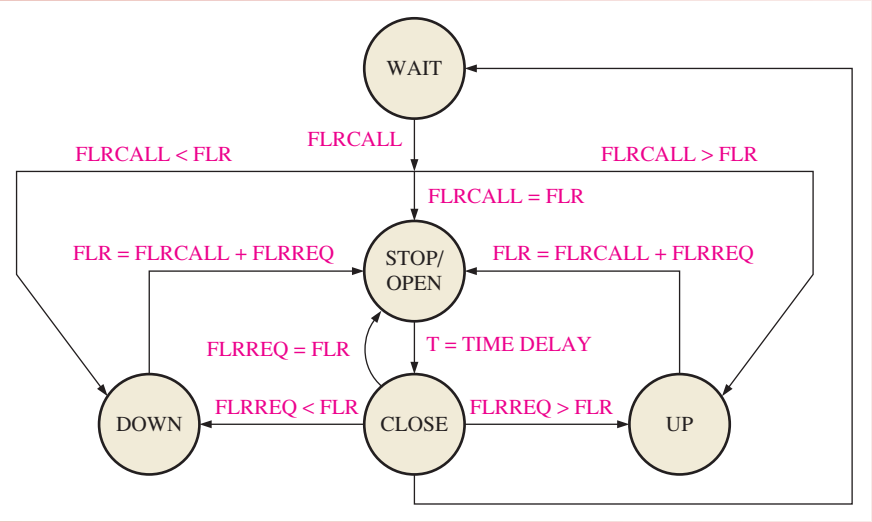


FIGURE 9–61 Elevator controller state diagram.

The following states are shown in the state diagram of Figure 9–61:

WAIT The system always begins in the WAIT state on the floor last serviced. When a floor call (FLRCALL) signal is received, the control logic determines if the number of the calling floor is greater than the current floor (FLRCALL > FLR), less than the current floor (FLRCALL < FLR), or equal to the current floor (FLRCALL = FLR) and puts the system in the UP mode, DOWN mode, or OPEN mode, respectively.

DOWN In this state, the elevator moves down toward the calling floor.

UP In this mode, the elevator moves up toward the calling floor.

STOP/OPEN This state occurs when the calling floor has been reached. When the number of the floor where the elevator is equals the number of the calling or requested floor, a signal is issued to stop the elevator and open the door.

CLOSE After a preset time (T) to allow entry or exit, the door closes.

The signals used by the elevator controller are defined as follows:

FLR Number of floor represented by a 3-bit binary code.

Floor sensor pulse A pulse issued at each floor to clock the floor counter to the next state.

FLRCALL Number of floor where a call for elevator service originates, represented by a 3-bit binary code.

Call pulse A pulse issued in conjunction with FLRCALL to clock the 3-bit code into a register.

FLRREQ Number of floor to which the passenger desires to go, represented by a 3-bit binary code.

Request pulse A pulse issued in conjunction with FLRREQ to clock the 3-bit code into a register.

UP A signal issued to the elevator motor control to cause the elevator to move from a lower floor to a higher floor.

DOWN A signal issued to the elevator motor control to cause the elevator to move from a higher floor to a lower floor.

STOP A signal issued to the elevator motor control to cause the elevator to stop.

OPEN A signal issued to door motor control to cause the door to open.

CLOSE A signal issued to the door motor control to cause the door to close.

Elevator Controller Block Diagram

Figure 9–62 shows the elevator controller block diagram, which consists of controller logic, a floor counter, and a floor number display. Assume that the elevator is on the first floor in

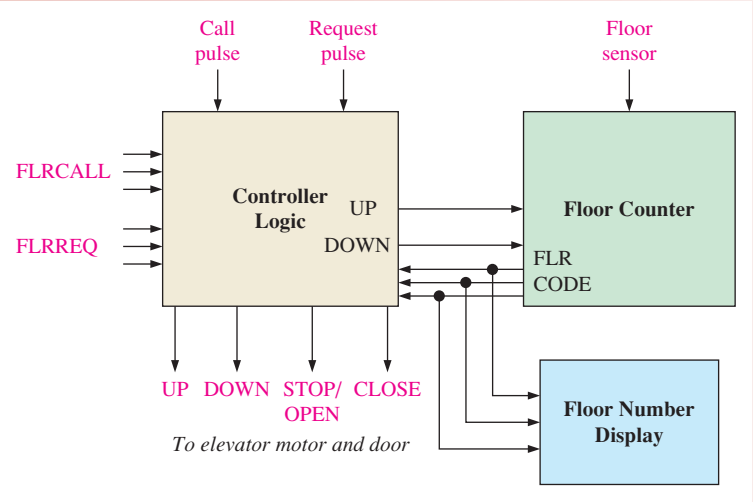


FIGURE 9–62 Elevator controller block diagram.

the WAIT state. The floor counter contains 001, which is the first floor code. Suppose the FLRCALL (101) comes in from the call button on the fifth floor. Since FLRCALL > FLR (101 > 001), the controller issues an UP command to the elevator motor. As the elevator moves up, the floor counter receives a floor sensor pulse as it reaches each floor which advances its state (001, 010, 011, 100, 101). When the fifth floor is reached and FLR = FLRCALL, the controller logic stops the elevator and opens its door. The process is repeated for a FLRREQ input.

The floor counter sequentially tracks the number of the floor and always contains the number of the current floor. It can count up or down and can reverse its state at any point under the direction of the state controller and the floor sensor input. A 3-bit counter is required since there are eight floors ($2^3 = 8$) including the basement, as shown in the floor counter state diagram in Figure 9–63.

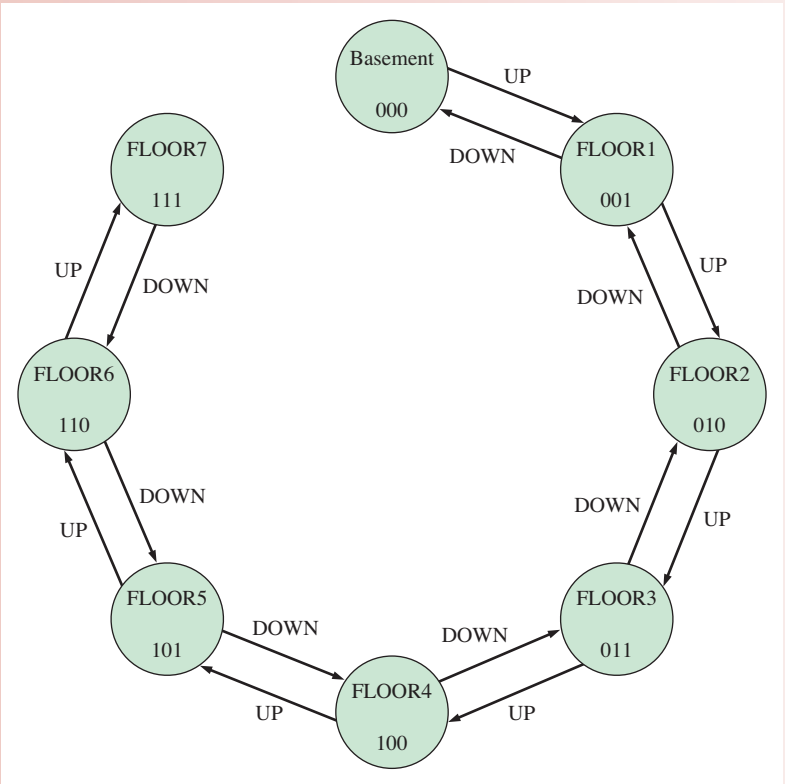


FIGURE 9–63 Floor counter state diagram.

Operation of Elevator Controller

The elevator controller logic diagram is shown in Figure 9–64. Elevator action is initiated by either a floor call (FLRCALL) or a floor request (FLRREQ). Keep in mind that FLRCALL is when a person calls the elevator to come to a particular floor. FLRREQ is when a passenger in the elevator requests to go to a specified floor. This simplified operation is based on a CALL/REQ sequence; that is, a call followed by a request followed by a call.

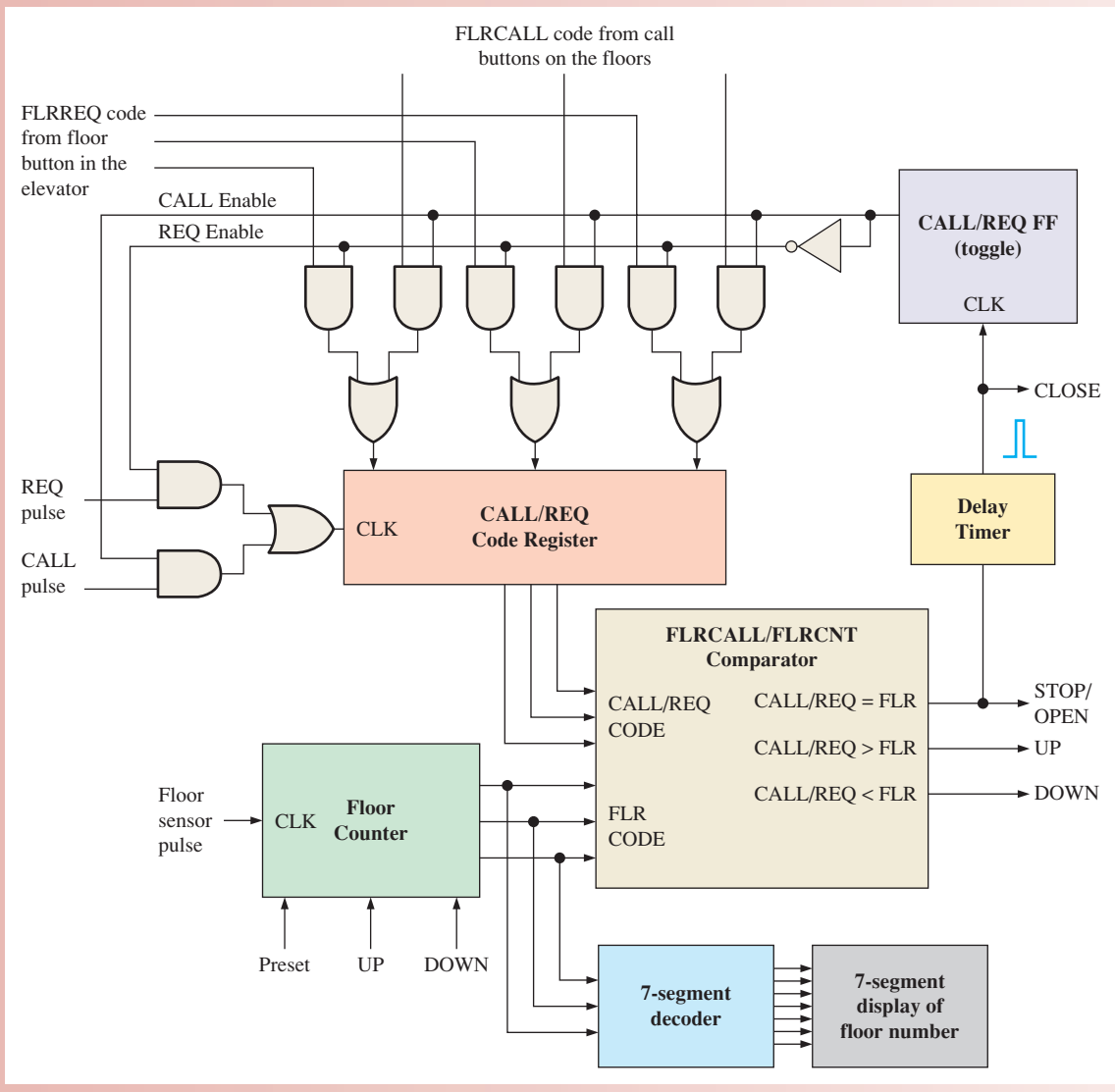


FIGURE 9–64 Elevator controller logic diagram.

As you know, FLRCALL and FLRREQ are 3-bit codes representing specific floors. When a person presses a call button on a given floor, the specific 3-bit code for that floor is placed on the inputs to the CALL/REQ code register and a CALL pulse is generated to enter the code into the register. The same process occurs when a request button is pressed inside the elevator. The code is input to the CALL/REQ code register, and a REQ pulse is generated to store the code in the register.

The elevator does not know the difference between a call and a request. The comparator determines if the destination floor number is greater than, less than, or equal to the current

floor where the elevator is located. As a result of this comparison, either an UP command, a DOWN command, or an OPEN command is issued to the elevator motor control. As the elevator moves toward the desired floor, the floor counter is either incremented at each floor as it goes up or decremented at each floor as it goes down. Once the elevator reaches the desired floor, a STOP/OPEN command is issued to the elevator motor control and to the door control. After a preset time, the delay timer issues a CLOSE signal to the elevator door control. As mentioned, this elevator design is limited to one floor call and one floor request per cycle.

Initialization The initial one-time setup requires that the elevator be placed at the basement level and the floor counter be preset to 000. After this, the counter will automatically move through the sequence of states determined by the elevator position.

Exercise

1. Explain the purpose of the floor counter.
2. Describe what happens during the WAIT mode.
3. How does the system know when the desired floor has been reached?
4. Discuss the limitations of the elevator design in Figure 9–64.

Implementation

The elevator controller can be implemented using fixed-function logic devices, a PLD programmed with a VHDL (or Verilog) code, or a programmed microcontroller or microprocessor. In the Chapter 10 Applied Logic, the VHDL program code for the elevator controller is presented. You will see how to program a PLD step by step.

Putting Your Knowledge to Work

What changes are required in the logic diagram of Figure 9–64 to upgrade the elevator controller for a ten-story building?

SUMMARY

- Asynchronous and synchronous counters differ only in the way in which they are clocked. The first stage of an asynchronous counter is driven by a clock pulse. Each succeeding stage is clocked by the output of the previous stage. In a synchronous counter, all stages are clocked by the same clock pulse. Synchronous counters can run at faster clock rates than asynchronous counters.
- The maximum modulus of a counter is the maximum number of possible states and is a function of the number of stages (flip-flops). Thus,

$$\text{Maximum modulus} = 2^n$$

  where n is the number of stages in the counter. The modulus of a counter is the *actual* number of states in its sequence and can be equal to or less than the maximum modulus.
- The overall modulus of cascaded counters is equal to the product of the moduli of the individual counters.

KEY TERMS

Key terms and other bold terms in the chapter are defined in the end-of-book glossary.

Asynchronous Not occurring at the same time.

Cascade To connect "end-to-end" as when several counters are connected from the terminal count output of one counter to the enable input of the next counter.

Decade Characterized by ten states or values.

Modulus The number of unique states through which a counter will sequence.

Recycle To undergo transition (as in a counter) from the final or terminal state back to the initial state.

State diagram A graphic depiction of a sequence of states or values.

State machine A logic system or circuit exhibiting a sequence of states conditioned by internal logic and external inputs; any sequential circuit exhibiting a specified sequence of states. Two types of state machine are Moore and Mealy.

Synchronous Occurring at the same time.

Terminal count The final state in a counter's sequence.

TRUE/FALSE QUIZ

Answers are at the end of the chapter.

1. A state machine is a sequential circuit having a limited number of states occurring in a prescribed order.
2. Synchronous counters cannot be realized using J-K flip-flops.
3. An asynchronous counter is also known as a ripple counter.
4. A decade counter has twelve states.
5. A counter with four stages has a maximum modulus of sixteen.
6. To achieve a maximum modulus of 32, sixteen stages are required.
7. If the present state is 1000, the next state of a 4-bit up/down counter in the DOWN mode is 0111.
8. Two cascaded decade counters divide the clock frequency by 10.
9. A counter with a truncated sequence has less than its maximum number of states.
10. To achieve a modulus of 100, ten decade counters are required.

SELF-TEST

Answers are at the end of the chapter.

1. A Moore state machine consists of combinational logic circuits that determine
   (a) sequences **(b)** memory
   (c) both (a) and (b) **(d)** neither (a) nor (b)
2. The output of a Mealy machine depends on its
   (a) inputs **(b)** next state
   (c) present state **(d)** answers (a) and (c)
3. The maximum cumulative delay of an asynchronous counter must be
   (a) more than the period of the clock waveform
   (b) less than the period of the clock waveform
   (c) equal to the period of the clock waveform
   (d) both (a) and (c)
4. A decade counter with a count of zero (0000) through nine (1001) is known as
   (a) an ASCII counter **(b)** a binary counter
   (c) A BCD counter **(d)** a decimal counter
5. The modulus of a counter is
   (a) the number of flip-flops
   (b) the actual number of states in its sequence
   (c) the number of times it recycles in a second
   (d) the maximum possible number of states
6. A 3-bit binary counter has a maximum modulus of
   (a) 3 **(b)** 6 **(c)** 8 **(d)** 16
7. A 5-bit binary counter has a maximum modulus of
   (a) 4 **(b)** 8 **(c)** 16 **(d)** 32
8. A modulus-12 counter must have
   (a) 12 flip-flops **(b)** 3 flip-flops
   (c) 4 flip-flops **(d)** synchronous clocking

9. Which one of the following is an example of a counter with a truncated modulus?
(a) Modulus 8 **(b)** Modulus 14
(c) Modulus 16 **(d)** Modulus 32

10. A 4-bit ripple counter consists of flip-flops that each have a propagation delay from clock to Q output of 12 ns. For the counter to recycle from 1111 to 0000, it takes a total of
(a) 12 ns **(b)** 24 ns
(c) 48 ns **(d)** 36 ns

11. A BCD counter is an example of
(a) a full-modulus counter **(b)** a decade counter
(c) a truncated-modulus counter **(d)** answers (b) and (c)

12. Which of the following is a valid state in an 8421 BCD counter?
(a) 1010 **(b)** 1011
(c) 1111 **(d)** 1000

13. Three cascaded modulus-10 counters have an overall modulus of
(a) 30 **(b)** 100
(c) 1000 **(d)** 10,000

14. A 10 MHz clock frequency is applied to a cascaded counter consisting of a modulus-5 counter, a modulus-8 counter, and two modulus-10 counters. The lowest output frequency possible is
(a) 10 kHz **(b)** 2.5 kHz
(c) 5 kHz **(d)** 25 kHz

15. A 4-bit binary up/down counter is in the binary state of zero. The next state in the DOWN mode is
(a) 0001 **(b)** 1111
(c) 1000 **(d)** 1110

16. The initial count of a modulus-13 binary counter is
(a) 0000 **(b)** 1111
(c) 1101 **(d)** 1100

PROBLEMS

Answers to odd-numbered problems are at the end of the book.

Section 9–1 Finite State Machines

1. Represent a decade counter with the terminal state decoded as a state machine. Identify the type and show the block diagram and the state diagram.

2. Identify the type of state machine for the traffic signal controller in Chapter 6. State the reason why it is the type you specified.

Section 9–2 Asynchronous Counters

3. For the ripple counter shown in Figure 9–65, show the complete timing diagram for eight clock pulses, showing the clock, Q_0, and Q_1 waveforms.

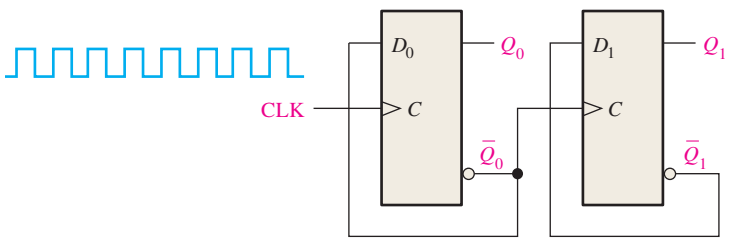


FIGURE 9–65

4. For the ripple counter in Figure 9–66, show the complete timing diagram for sixteen clock pulses. Show the clock, Q_0, Q_1, and Q_2 waveforms.

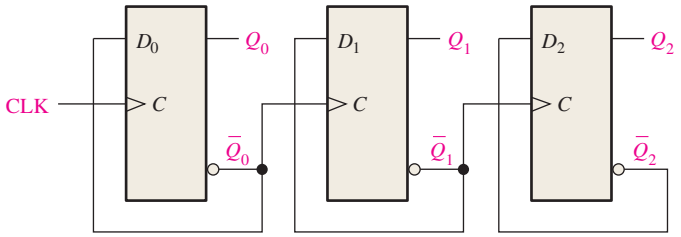


FIGURE 9–66

5. In the counter of Problem 4, assume that each flip-flop has a propagation delay from the triggering edge of the clock to a change in the Q output of 8 ns. Determine the worst-case (longest) delay time from a clock pulse to the arrival of the counter in a given state. Specify the state or states for which this worst-case delay occurs.

6. Show how to connect a 74HC93 4-bit asynchronous counter for each of the following moduli:

(a) 9 **(b)** 11 **(c)** 13 **(d)** 14 **(e)** 15

Section 9–3 Synchronous Counters

7. If the counter of Problem 5 were synchronous rather than asynchronous, what would be the longest delay time?

8. Show the complete timing diagram for the 5-stage synchronous binary counter in Figure 9–67. Verify that the waveforms of the Q outputs represent the proper binary number after each clock pulse.

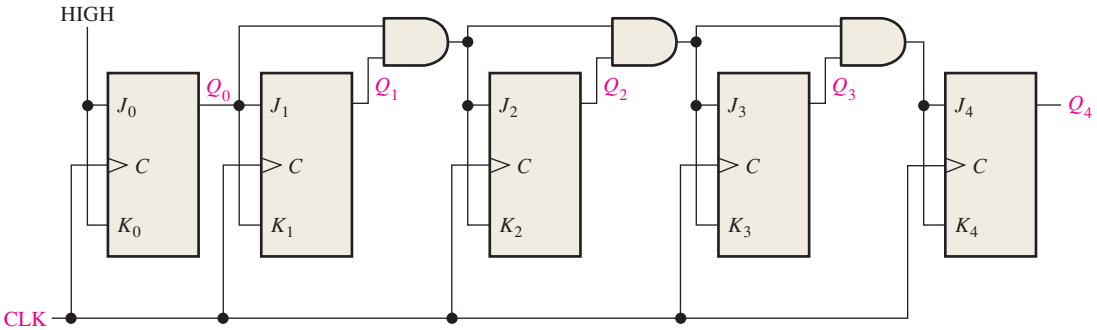


FIGURE 9–67

9. By analyzing the J and K inputs to each flip-flop prior to each clock pulse, prove that the decade counter in Figure 9–68 progresses through a BCD sequence. Explain how these conditions in each case cause the counter to go to the next proper state.

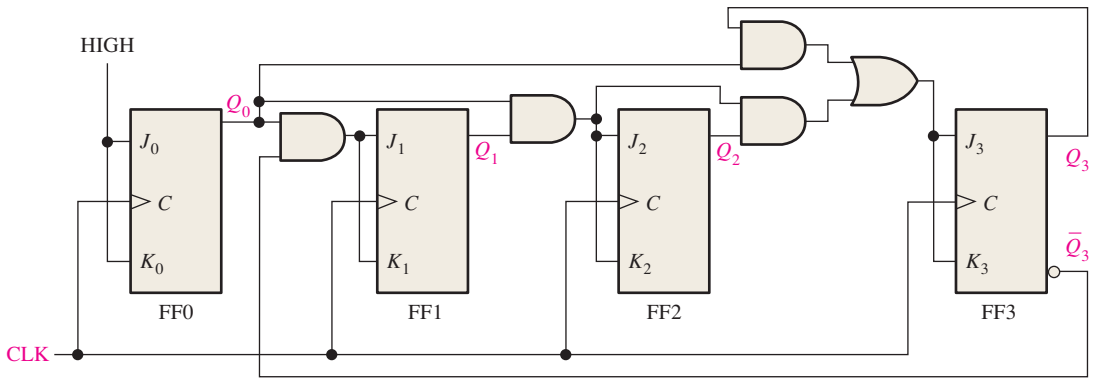


FIGURE 9–68

10. The waveforms in Figure 9–69 are applied to the count enable, clear, and clock inputs as indicated. Show the counter output waveforms in proper relation to these inputs. The clear input is asynchronous.

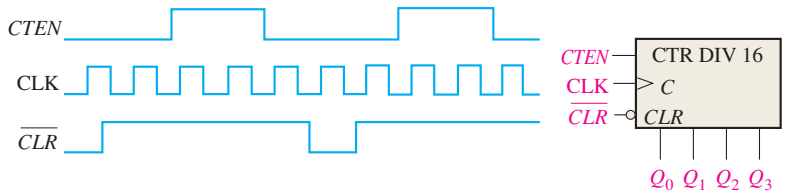


FIGURE 9–69

11. A BCD decade counter is shown in Figure 9–70. The waveforms are applied to the clock and clear inputs as indicated. Determine the waveforms for each of the counter outputs (Q_0, Q_1, Q_2, and Q_3). The clear is synchronous, and the counter is initially in the binary 1000 state.

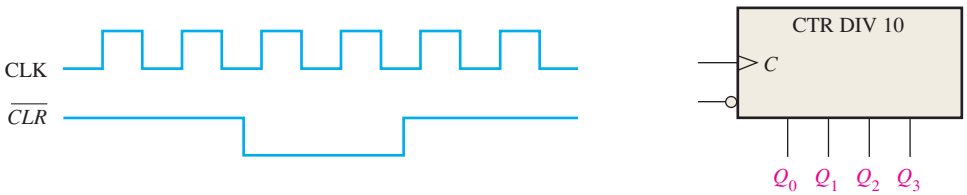


FIGURE 9–70

12. The waveforms in Figure 9–71 are applied to a 74HC163 binary counter. Determine the Q outputs and the *RCO*. The inputs are $D_0 = 1$, $D_1 = 1$, $D_2 = 0$, and $D_3 = 1$.

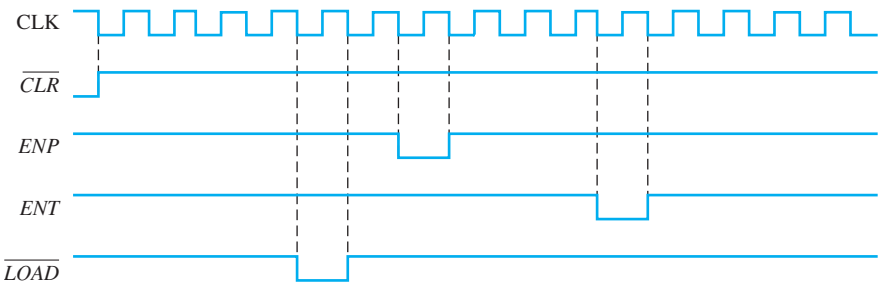


FIGURE 9–71

13. The waveforms in Figure 9–71 are applied to a 74HC161 counter. Determine the Q outputs and the *RCO*. The inputs are $D_0 = 1$, $D_1 = 0$, $D_2 = 0$, and $D_3 = 1$.

Section 9–4 Up/Down Synchronous Counters

14. Show a complete timing diagram for a 3-bit up/down counter that goes through the following sequence. Indicate when the counter is in the UP mode and when it is in the DOWN mode. Assume positive edge-triggering.

0, 1, 2, 3, 2, 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1, 0

15. Develop the Q output waveforms for a 74HC190 up/down counter with the input waveforms shown in Figure 9–72. A binary 0 is on the data inputs. Start with a count of 0000.

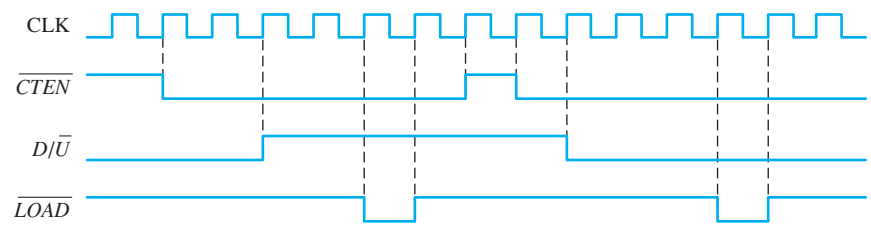


FIGURE 9–72

16. Repeat Problem 15 if the D/$\overline{U}$ input signal is inverted with the other inputs the same.

17. Repeat Problem 15 if the $\overline{CTEN}$ is inverted with the other inputs the same.

Section 9–5 Design of Synchronous Counters

18. Determine the sequence of the counter in Figure 9–73.

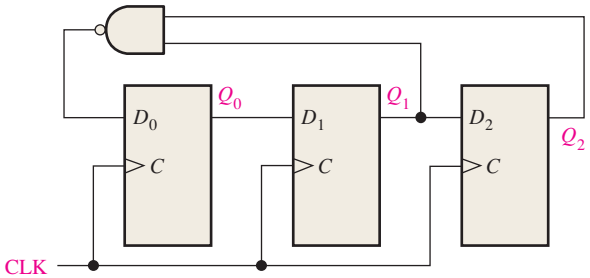


FIGURE 9–73

19. Determine the sequence of the counter in Figure 9–74. Begin with the counter cleared.

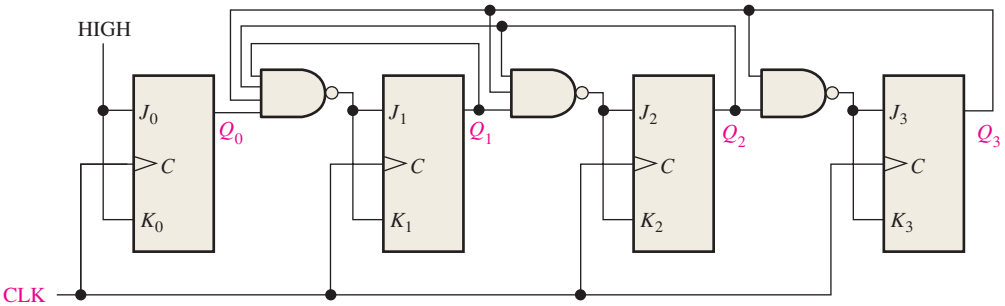


FIGURE 9–74

20. Design a counter to produce the following sequence. Use J-K flip-flops.

00, 10, 01, 11, 00, . . .

21. Design a counter to produce the following binary sequence. Use J-K flip-flops.

1, 4, 3, 5, 7, 6, 2, 1, . . .

22. Design a counter to produce the following binary sequence. Use J-K flip-flops.

0, 9, 1, 8, 2, 7, 3, 6, 4, 5, 0, . . .

23. Design a binary counter with the sequence shown in the state diagram of Figure 9–75.

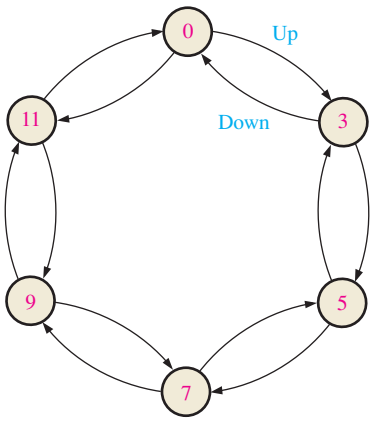


FIGURE 9–75

Section 9–6 Cascaded Counters

24. For each of the cascaded counter configurations in Figure 9–76, determine the frequency of the waveform at each point indicated by a circled number, and determine the overall modulus.

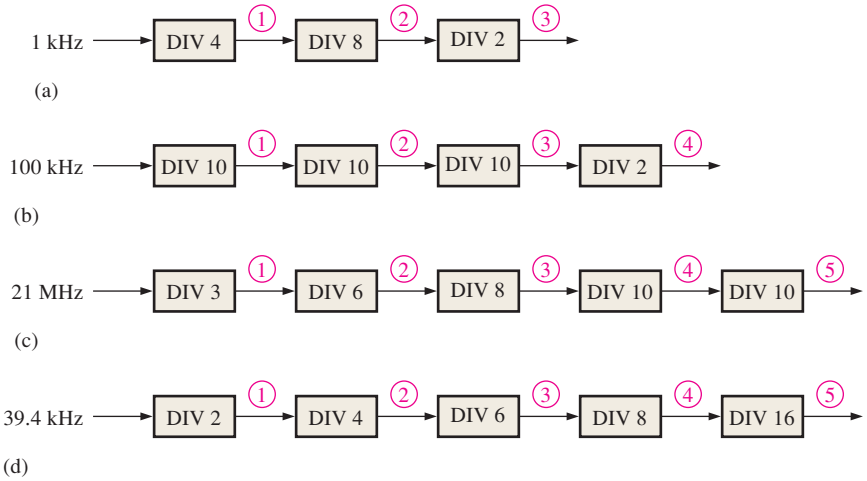


FIGURE 9–76

25. Expand the counter in Figure 9–38 to create a divide-by-10,000 counter and a divide-by-100,000 counter.

26. With general block diagrams, show how to obtain the following frequencies from a 10 MHz clock by using single flip-flops, modulus-5 counters, and decade counters:

(a) 5 MHz **(b)** 2.5 MHz **(c)** 2 MHz **(d)** 1 MHz **(e)** 500 kHz
(f) 250 kHz **(g)** 62.5 kHz **(h)** 40 kHz **(i)** 10 kHz **(j)** 1 kHz

Section 9–7 Counter Decoding

27. Given a BCD decade counter with only the Q outputs available, show what decoding logic is required to decode each of the following states and how it should be connected to the counter. A HIGH output indication is required for each decoded state. The MSB is to the left.

(a) 0001 **(b)** 0011 **(c)** 0101 **(d)** 0111 **(e)** 1000

28. For the 4-bit binary counter connected to the decoder in Figure 9–77, determine each of the decoder output waveforms in relation to the clock pulses.

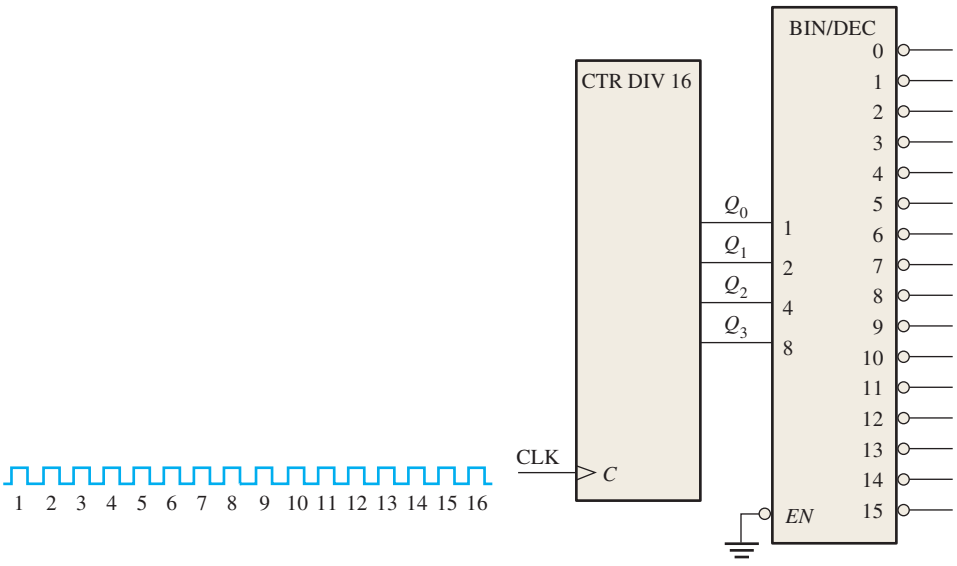


FIGURE 9–77

29. If the counter in Figure 9–77 is asynchronous, determine where the decoding glitches occur on the decoder output waveforms.

30. Modify the circuit in Figure 9–77 to eliminate decoding glitches.

31. Analyze the counter in Figure 9–42 for the occurrence of glitches on the decode gate output. If glitches occur, suggest a way to eliminate them.

32. Analyze the counter in Figure 9–43 for the occurrence of glitches on the outputs of the decoding gates. If glitches occur, make a design change that will eliminate them.

Section 9–8 Counter Applications

33. Assume that the digital clock of Figure 9–48 is initially reset to 12 o'clock. Determine the binary state of each counter after sixty-two 60 Hz pulses have occurred.

34. What is the output frequency of each counter in the digital clock circuit of Figure 9–48?

35. For the automobile parking control system in Figure 9–51, a pattern of entrance and exit sensor pulses during a given 24-hour period are shown in Figure 9–78. If there were 53 cars already in the garage at the beginning of the period, what is the state of the counter at the end of the 24 hours?

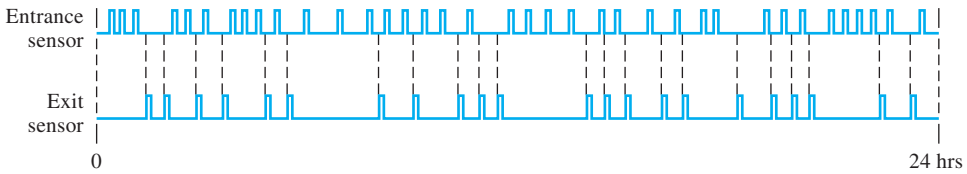


FIGURE 9–78

36. The binary number for decimal 57 appears on the parallel data inputs of the parallel-to-serial converter in Figure 9–53 (D_0 is the LSB). The counter initially contains all zeros and a 10 kHz clock is applied. Develop the timing diagram showing the clock, the counter outputs, and the serial data output.

Section 9–10 Troubleshooting

37. For the counter in Figure 9–4, show the timing diagram for the Q_0 and Q_1 waveforms for each of the following faults (assume Q_0 and Q_1 are initially LOW):

(a) clock input to FF0 shorted to ground
(b) Q_0 output open
(c) clock input to FF1 open
(d) D input to FF0 open
(e) D input to FF1 shorted to ground

38. Solve Problem 37 for the counter in Figure 9–12(b).

39. Isolate the fault in the counter in Figure 9–6 by analyzing the waveforms in Figure 9–79.

40. From the waveform diagram in Figure 9–80, determine the most likely fault in the counter of Figure 9–15.

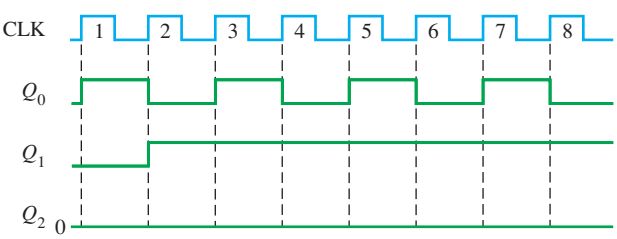


FIGURE 9–79

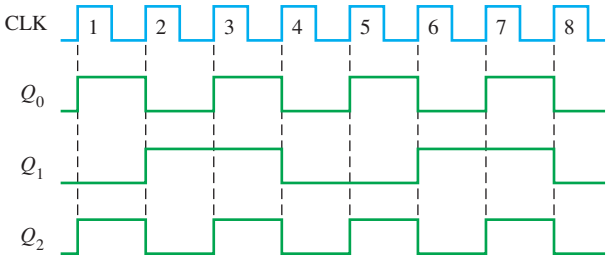


FIGURE 9–80

41. Solve Problem 40 if the Q_2 output has the waveform observed in Figure 9–81. Outputs Q_0 and Q_1 are the same as in Figure 9–80.

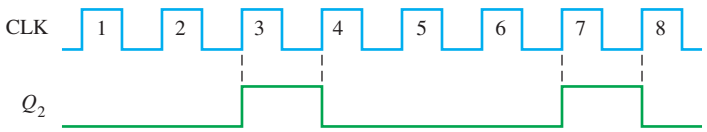


FIGURE 9–81

42. You apply a 5 MHz clock to the cascaded counter in Figure 9–41 and measure a frequency of 76.2939 Hz at the last *RCO* output. Is this correct, and if not, what is the most likely problem?

43. Develop a table for use in testing the counter in Figure 9–41 that will show the frequency at the final *RCO* output for all possible open failures of the parallel data inputs (D_0, D_1, D_2, and D_3) taken one at a time. Use 10 MHz as the test frequency for the clock.

44. The tens-of-hours 7-segment display in the digital clock system of Figure 9–48 continuously displays a 1. All the other digits work properly. What could be the problem?

45. What would be the visual indication of an open Q_1 output in the tens portion of the minutes counter in Figure 9–48? Also see Figure 9–49.

46. One day (perhaps a Monday) complaints begin flooding in from patrons of a parking garage that uses the control system depicted in Figures 9–51 and 9–52. The patrons say that they enter the garage because the gate is up and the FULL sign is off but that, once in, they can find no empty space. As the technician in charge of this facility, what do you think the problem is, and how will you troubleshoot and repair the system as quickly as possible?

Applied Logic

47. Propose a general design for generation of the 3-bit FLRCALL code and the Call pulse by the pressing of a single button.

48. Propose a general design for generation of the 3-bit FLRREQ code and the Request pulse by the pressing of one of seven buttons.

49. What changes are required to the logic diagram in Figure 9–64 to modify the elevator controller for a four-story building?

Special Design Problems

50. Design a modulus-1000 counter by using decade counters.

51. Modify the design of the counter in Figure 9–41 to achieve a modulus of 30,000.

52. Repeat Problem 51 for a modulus of 50,000.

53. Modify the digital clock in Figures 9–48, 9–49, and 9–50 so that it can be preset to any desired time.

54. Design an alarm circuit for the digital clock that can detect a predetermined time (hours and minutes only) and produce a signal to activate an audio alarm.

55. Modify the design of the circuit in Figure 9–52 for a 1000-space parking garage and a 3000-space parking garage.

56. Implement the parallel-to-serial data conversion logic in Figure 9–53 with specific fixed-function devices.

57. In Problem 19 it was found that the counter locks up and alternates between two states. It turns out that this operation is the result of a design flaw. Redesign the counter so that when it goes into the second of the lock-up states, it will recycle to the all-0s state on the next clock pulse.

Multisim Troubleshooting Practice

MultiSim

58. Open file P09-58. For the specified fault, predict the effect on the circuit. Then introduce the fault and verify whether your prediction is correct.

59. Open file P09-59. For the specified fault, predict the effect on the circuit. Then introduce the fault and verify whether your prediction is correct.

60. Open file P09-60. For the specified fault, predict the effect on the circuit. Then introduce the fault and verify whether your prediction is correct.

61. Open file P09-61. For the observed behavior indicated, predict the fault in the circuit. Then introduce the suspected fault and verify whether your prediction is correct.

62. Open file P09-62. For the observed behavior indicated, predict the fault in the circuit. Then introduce the suspected fault and verify whether your prediction is correct.

ANSWERS

SECTION CHECKUPS

Section 9–1 Checkup

1. A finite state machine is a sequential circuit having a finite number of states that occur in a specified order.

2. Moore state machine and Mealy state machine

3. The Moore state machine has an output(s) that is dependent on the present internal state only. The Mealy state machine has an output(s) that is dependent on both the present internal state and the value of the inputs.

Section 9–2 Asynchronous Counters

1. Asynchronous means that each flip-flop after the first one is enabled by the output of the preceding flip-flop.

2. A modulus-14 counter has fourteen states requiring four flip-flops.

Section 9–3 Synchronous Counters

1. All flip-flops in a synchronous counter are clocked simultaneously.

2. The counter can be preset (initialized) to any given state.

3. Counter is enabled when *ENP* and *ENT* are both HIGH; *RCO* goes HIGH when final state in sequence is reached.

Section 9–4 Up/Down Synchronous Counters

1. The counter goes to 1001.

2. UP: 1111: DOWN: 0000; the next state is 1111.

Section 9–5 Design of Synchronous Counters

1. $J = 1, K =$ X ("don't care")

2. $J = X$ ("don't care"), $K = 0$

3. **(a)** The next state is 1011.

(b) Q_3 (MSB): no-change or SET; Q_2: no-change or RESET; Q_1: no change or SET; Q_0 (LSB): SET or toggle

Section 9–6 Cascaded Counters

1. Three decade counters produce ÷ 1000; 4 decade counters produce ÷ 10,000.

2. **(a)** ÷ 20: flip-flop and DIV 10

(b) ÷ 32: flip-flop and DIV 16

(c) ÷ 160: DIV 16 and DIV 10

(d) ÷ 320: DIV 16 and DIV 10 and flip-flop

Section 9–7 Counter Decoding

1. **(a)** No transitional states because there is a single bit change

(b) 0000, 0001, 0010, 0101, 0110, 0111

(c) No transitional states because there is a single bit change

(d) 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110

Section 9–8 Counter Applications

1. Gate G_1 resets flip-flop on first clock pulse after count 9. Gate G_2 decodes count 12 to preset counter to 0001.
2. The hours decade counter advances through each state from zero to nine, and as it recycles from nine back to zero, the flip-flop is toggled to the SET state. This produces a ten (10) on the display. When the hours decade counter is in state 12, the decode NAND gate causes the counter to recycle to state 1 on the next clock pulse. The flip-flop resets. This results in a one (01) on the display.

Section 9–9 Logic Symbols with Dependency Notation

1. *C:* control, usually clock; *M:* mode; *G:* AND
2. *D* indicates data storage.

Section 9–10 Troubleshooting

1. No pulses on *TC* outputs: *CTEN* of first counter shorted to ground or to a LOW; clock input of first counter open; clock line shorted to ground or to a LOW; *TC* output of first counter shorted to ground or to a LOW.
2. With inverter output open, the counter does not recycle at the preset count but acts as a full-modulus counter.

RELATED PROBLEMS FOR EXAMPLES

9–1 See Figure 9–82.

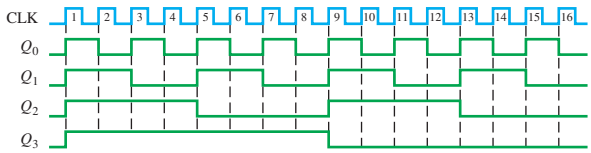


FIGURE 9–82

9–2 Connect Q_0 to the NAND gate as a third input (Q_2 and Q_3 are two of the inputs). Connect the $\overline{CLR}$ line to the $\overline{CLR}$ input of FF0 as well as FF2 and FF3.

9–3 See Figure 9–83.

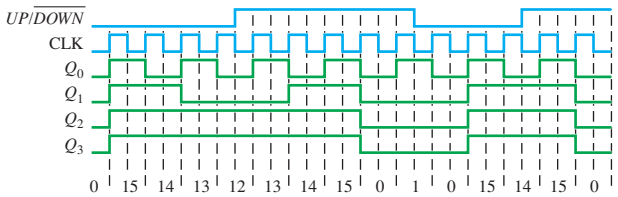


FIGURE 9–83

9–4 See Table 9–14.

TABLE 9–14

Present Invalid State			*D* Inputs			Next State			
Q_2	Q_1	Q_0	D_2	D_1	D_0	Q_2	Q_1	Q_0	
0	0	0	1	1	1	1	1	1	valid state
0	1	1	0	0	0	0	0	0	
1	0	0	1	1	1	1	1	1	
1	1	0	1	0	1	1	0	1	valid state

000 → 111
011 → 000 → 111
100 → 111
110 → 101

9–5 Three flip-flops, sixteen 3-input AND gates, two 4-input OR gates, four 2-input OR gates, and one inverter

9–6 Five decade counters are required. $10^5 = 100{,}000$

9–7 $f_{Q0} = 1\text{ MHz}/[(10)(2)] = 50\text{ kHz}$

9–8 See Figure 9–84.

9–9 $8AC0_{16}$ would be loaded. $16^4 - 8AC0_{16} = 65{,}536 - 32{,}520 = 30{,}016$
$f_{TC4} = 10\text{ MHz}/30{,}016 = 333.2\text{ Hz}$

9–10 See Figure 9–85.

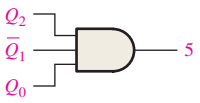


FIGURE 9–84

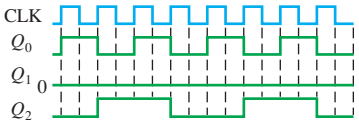


FIGURE 9–85

TRUE/FALSE QUIZ

1. T **2.** F **3.** T **4.** F **5.** T **6.** F **7.** T **8.** F **9.** T **10.** F

SELF-TEST

1. (c) **2.** (a) **3.** (b) **4.** (c) **5.** (b) **6.** (c) **7.** (d) **8.** (c)
9. (b) **10.** (c) **11.** (d) **12.** (d) **13.** (c) **14.** (b) **15.** (b) **16.** (a)

CHAPTER 11

Data Storage

CHAPTER OUTLINE

CHAPTER OBJECTIVES

- Define the basic memory characteristics
- Explain what a RAM is and how it works
- Explain the difference between static RAMs (SRAMs) and dynamic RAMs (DRAMs)
- Explain what a ROM is and how it works
- Describe the various types of PROMs
- Discuss the characteristics of a flash memory
- Describe the expansion of ROMs and RAMs to increase word length and word capacity
- Discuss special types of memories such as FIFO and LIFO
- Describe the basic organization of magnetic disks and magnetic tapes
- Describe the basic operation of magneto-optical disks and optical disks
- Describe the key elements in a memory hierarchy
- Describe several characteristics of cloud storage
- Describe basic methods for memory testing
- Develop flowcharts for memory testing

KEY TERMS

Key terms are in order of appearance in the chapter.

- Memory
- Byte
- Word
- Cell
- Address
- Capacity
- Write
- Read
- RAM
- ROM
- SRAM
- DRAM
- Bus
- PROM
- EPROM
- Flash memory
- FIFO
- LIFO
- Hard disk
- Blu-ray
- Memory hierarchy
- Cloud storage
- Server

VISIT THE WEBSITE

Study aids for this chapter are available at http://www.pearsonglobaleditions.com/floyd

INTRODUCTION

Chapter 8 covered shift registers, which are a type of storage device. The memory devices covered in this chapter are generally used for longer-term storage of larger amounts of data than registers can provide.

Computers and other types of systems require the permanent or semipermanent storage of large amounts of binary data. Microprocessor-based systems rely on storage devices for their operation because of the necessity for storing programs and for retaining data during processing.

In this chapter semiconductor memories and magnetic and optical storage media are covered. Also, memory hierarchy and cloud storage are discussed.

11–1 Semiconductor Memory Basics

Memory is the portion of a computer or other system that stores binary data. In a computer, memory is accessed millions of times per second, so the requirement for speed and accuracy is paramount. Very fast semiconductor memory is available today in modules with several GB (a gigabyte is one billion bytes) of capacity. These large-memory modules use exactly the same operating principles as smaller units, so we will use smaller ones for illustration in this chapter to simplify the concepts.

After completing this chapter, you should be able to

- Explain how a memory stores binary data
- Discuss the basic organization of a memory
- Describe the write operation
- Describe the read operation
- Describe the addressing operation
- Explain what RAMs and ROMs are

InfoNote

The general definition of *word* is a complete unit of information consisting of a unit of binary data. When applied to computer instructions, a word is more specifically defined as two bytes (16 bits). As an important part of assembly language used in computers, the DW (Define Word) directive means to define data in 16-bit units. This definition is independent of the particular microprocessor or the size of its data bus. Assembly language also allows definitions of bytes (8 bits) with the DB directive, double words (32 bits) with the DD directive, and quad-words (64 bits) with the QD directive.

Units of Binary Data: Bits, Bytes, Nibbles, and Words

As a rule, memories store data in units that have from one to eight bits. The smallest unit of binary data, as you know, is the **bit**. In many applications, data are handled in an 8-bit unit called a **byte** or in multiples of 8-bit units. The byte can be split into two 4-bit units that are called **nibbles**. Bytes can also be grouped into words. The term **word** can have two meanings in computer terminology. In memories, it is defined as a group of bits or bytes that acts as a single entity that can be stored in one memory location. In assembly language, a word is specifically defined as two bytes.

The Basic Memory Array

Each storage element in a memory can retain either a 1 or a 0 and is called a **cell**. Memories are made up of arrays of cells, as illustrated in Figure 11–1 using 64 cells as an example. Each block in the **memory array** represents one storage cell, and its location can be identified by specifying a row and a column.

The 64-cell array can be organized in several ways based on units of data. Figure 11–1(a) shows an 8 × 8 array, which can be viewed as either a 64-bit memory or an 8-byte memory. Part (b) shows a 16 × 4 array, which is a 16-nibble memory, and part (c) shows a 64 × 1

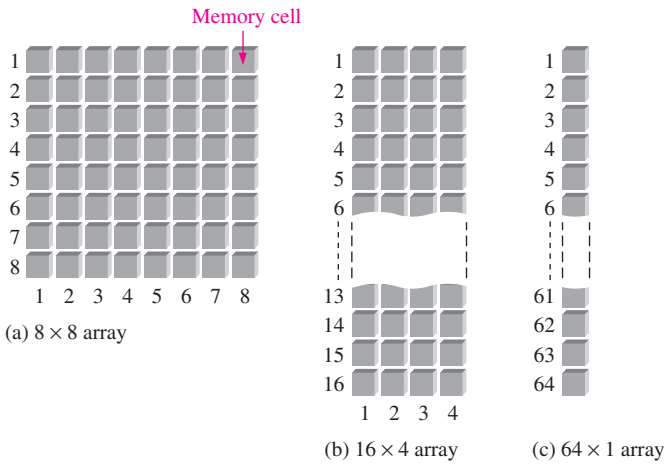
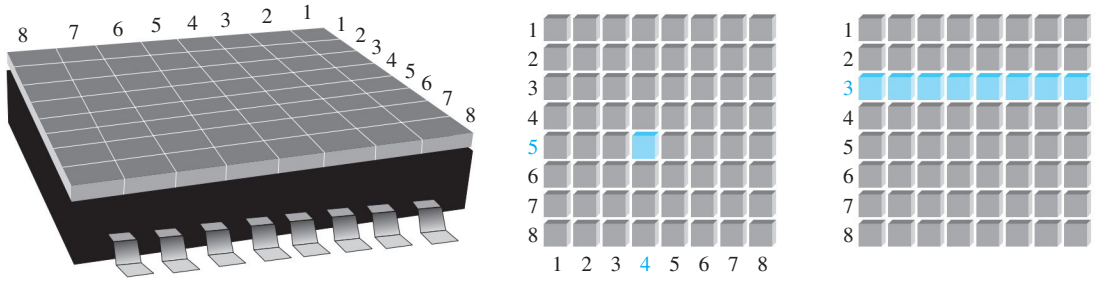


FIGURE 11–1 A 64-cell memory array organized in three different ways.

array, which is a 64-bit memory. A memory is identified by the number of words it can store times the word size. For example, a 16k × 8 memory can store 16,384 words of eight bits each. The inconsistency here is common in memory terminology. The actual number of words is always a power of 2, which, in this case, is $2^{14} = 16{,}384$. However, it is common practice to state the number to the nearest thousand, in this case, 16k.

Memory Address and Capacity

A representation of a small 8 × 8 memory chip is shown in Figure 11–2(a). The location of a unit of data in a memory array is called its **address**. For example, in part (b), the address of a bit in the 2-dimensional array is specified by the row and column as shown. In part (c), the address of a byte is specified only by the row. So, as you can see, the address depends on how the memory is organized into units of data. Personal computers have random-access memories organized in bytes. This means that the smallest group of bits that can be addressed is eight.



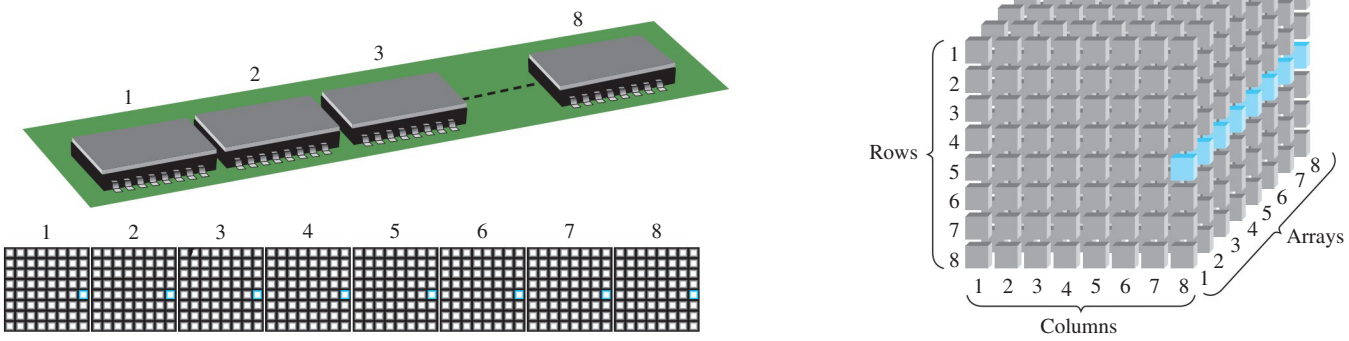
(a) Physical structure of 64-bit memory.

(b) The address of the blue bit is row 5, column 4.

(c) The address of the blue byte is row 3.

FIGURE 11–2 Examples of memory address in a 2-dimensional memory array.

Figure 11–3(a) illustrates the expansion of the 8 × 8 (64-bit) array to a 64-byte memory. The address of a byte in the array is specified by the row and column, as shown. In this case, the smallest group of bits that can be accessed is eight. This can be viewed as a 3-dimensional array, as shown in part (b).



(a) The 8 × 8 bit array expanded to a 64 × 8 bit array. This array forms a memory module.

(b) The address of the blue byte is row 5, column 8.

FIGURE 11–3 Example of memory address in an expanded (multiple) array.

The **capacity** of a memory is the total number of data units that can be stored. For example, in the bit-organized memory array in Figure 11–2(b), the capacity is 64 bits. In the byte-organized memory array in Figure 11–2(c), the capacity is 8 bytes, which is also

64 bits. In Figure 11–3, the capacity is 64 bytes. Computer memories typically have multiple gigabytes of internal memory. Computers usually transfer and store data as 64-bit words, in which case all eight bits of row five in each chip in Figure 11–3(a) would be accessed.

Memory Banks and Ranks

A **bank** is a section of memory within a single memory array (chip). A memory chip may have one or more banks. Memory banks can be used for storing frequently used information. Easier and faster access can be achieved by knowing the section of memory in which the data are stored. A **rank** is a group of chips that make up a memory module that stores data in units such as words or bytes. These terms are illustrated in Figure 11–4.

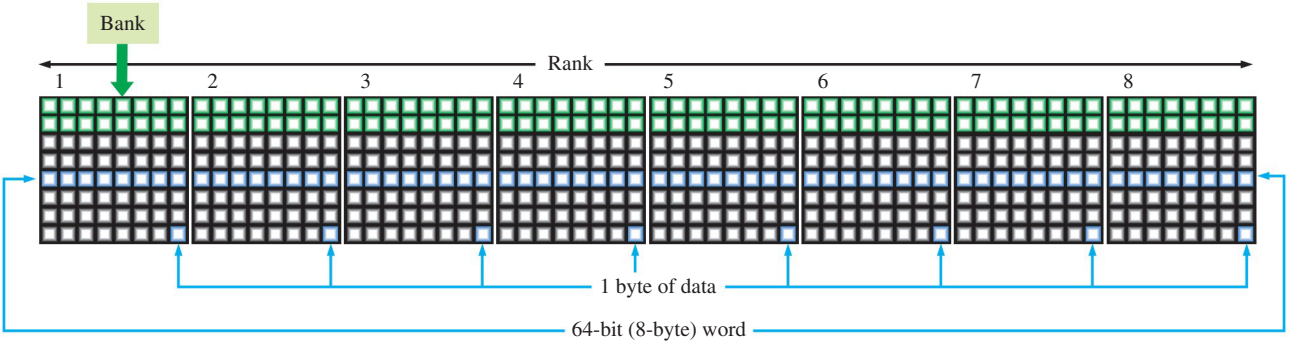


FIGURE 11–4 Simple illustration of memory bank and memory rank.

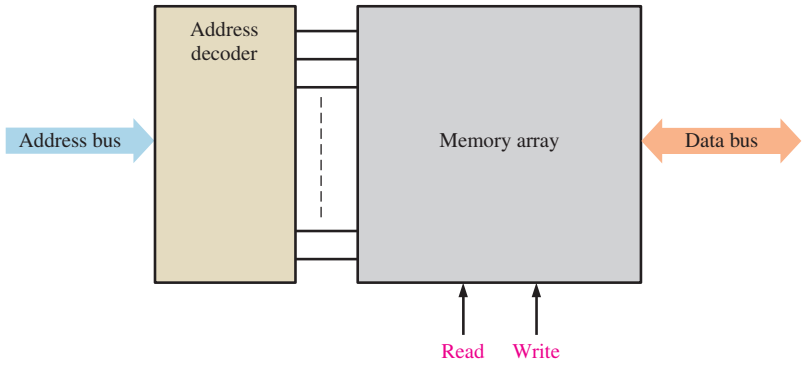
Basic Memory Operations

Addressing is the process of accessing a specified location in memory. Since a memory stores binary data, data must be put into the memory and data must be copied from the memory when needed. The **write** operation puts data into a specified address in the memory, and the **read** operation copies data out of a specified address in the memory. The addressing operation, which is part of both the write and the read operations, selects the specified memory address.

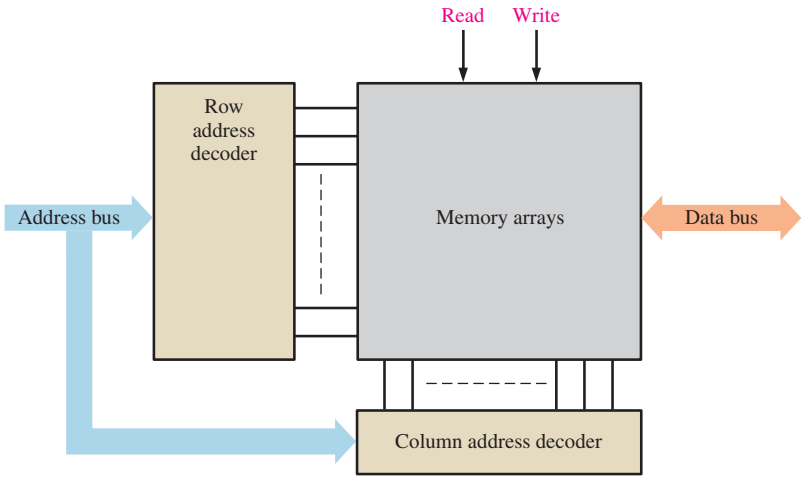
Data units go into the memory during a write operation and come out of the memory during a read operation on a set of lines called the *data bus.* As indicated in Figure 11–5, the data bus is bidirectional, which means that data can go in either direction (into the memory or out of the memory). In this case of byte-organized memories, the data bus has at least eight lines so that all eight bits in a selected address are transferred in parallel. For a write or a read operation, an address is selected by placing a binary code representing the desired address on a set of lines called the *address bus.* The address code is decoded internally, and the appropriate address is selected. In the case of the multiple-array memory in Figure 11–5(b) there are two decoders, one for the rows and one for the columns. The number of lines in the address bus depends on the capacity of the memory. For example, a 15-bit address code can select 32,768 locations (2^{15}) in the memory, a 16-bit address code can select 65,536 locations (2^{16}) in the memory, and so on. In personal computers a 32-bit address bus can select 4,294,967,296 locations (2^{32}), expressed as 4G.

The Write Operation

A simplified write operation is illustrated in Figure 11–6. To store a byte of data in the memory, a code held in the address register is placed on the address bus. Once the address code is on the bus, the address decoder decodes the address and selects the specified location in the memory. The memory then gets a write command, and the data byte held in the data register is placed on the data bus and stored in the selected memory address, thus completing the write operation. When a new data byte is written into a memory address, the current data byte stored at that address is overwritten (replaced with a new data byte).

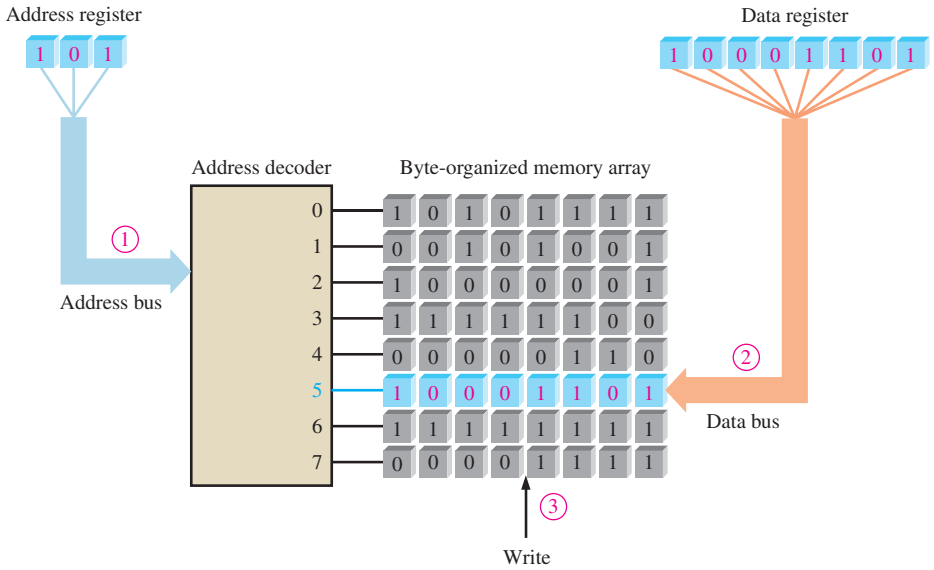


(a) Single-array memory



(b) Multiple-array memory

FIGURE 11–5 Block diagram of a single-array memory and a multiple-array memory showing address bus, address decoder(s), bidirectional data bus, and read/write inputs.

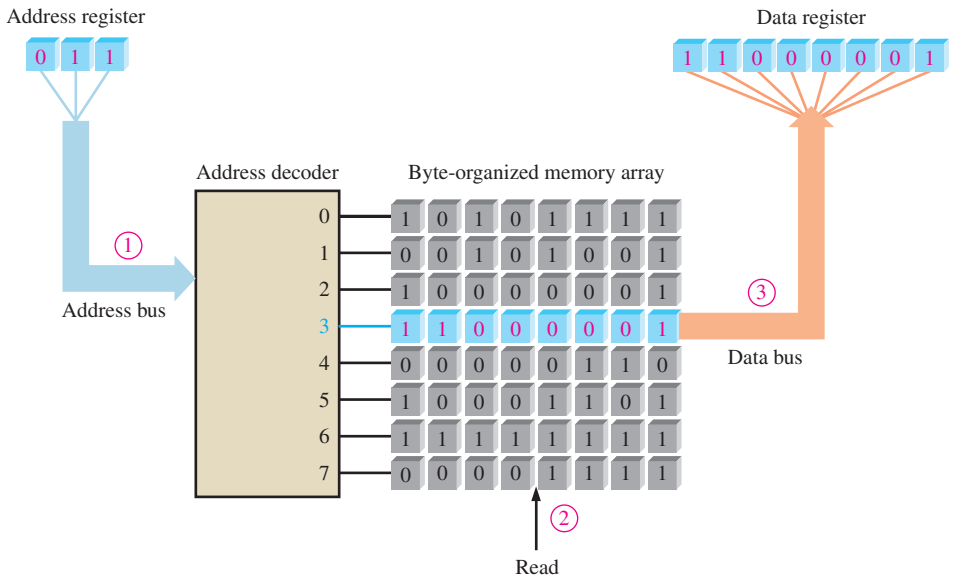


1. Address code 101 is placed on the address bus and address 5 is selected.
2. Data byte is placed on the data bus.
3. Write command causes the data byte to be stored in address 5, replacing previous data.

FIGURE 11–6 Illustration of the write operation.

The Read Operation

A simplified read operation is illustrated in Figure 11–7. Again, a code held in the address register is placed on the address bus. Once the address code is on the bus, the address decoder decodes the address and selects the specified location in the memory. The memory then gets a read command, and a "copy" of the data byte that is stored in the selected memory address is placed on the data bus and loaded into the data register, thus completing the read operation. When a data byte is read from a memory address, it also remains stored at that address. This is called *nondestructive read.*



1. Address code 011 is placed on the address bus and address 3 is selected.
2. Read command is applied.
3. The contents of address 3 is placed on the data bus and shifted into data register. The contents of address 3 is not erased by the read operation.

FIGURE 11–7 Illustration of the read operation.

RAMs and ROMs

The two major categories of semiconductor memories are the RAM and the ROM. **RAM** (random-access memory) is a type of memory in which all addresses are accessible in an equal amount of time and can be selected in any order for a read or write operation. All RAMs have both *read* and *write* capability. Because RAMs lose stored data when the power is turned off, they are **volatile** memories.

ROM (read-only memory) is a type of memory in which data are stored permanently or semipermanently. Data can be read from a ROM, but there is no write operation as in the RAM. The ROM, like the RAM, is a random-access memory but the term *RAM* traditionally means a random-access *read/write* memory. Several types of RAMs and ROMs will be covered in this chapter. Because ROMs retain stored data even if power is turned off, they are **nonvolatile** memories.

SECTION 11–1 CHECKUP

Answers are at the end of the chapter.

1. What is the smallest unit of data that can be stored in a memory?
2. What is the bit capacity of a memory that can store 256 bytes of data?

3. What is a write operation?
4. What is a read operation?
5. How is a given unit of data located in a memory?
6. Describe the difference between a RAM and a ROM.

11–2 The Random-Access Memory (RAM)

A RAM is a read/write memory in which data can be written into or read from any selected address in any sequence. When a data unit is written into a given address in the RAM, the data unit previously stored at that address is replaced by the new data unit. When a data unit is read from a given address in the RAM, the data unit remains stored and is not erased by the read operation. This nondestructive read operation can be viewed as copying the content of an address while leaving the content intact. A RAM is typically used for short-term data storage because it cannot retain stored data when power is turned off.

After completing this section, you should be able to

- Name the two categories of RAM
- Explain what a SRAM is
- Describe the SRAM storage cell
- Explain the difference between an asynchronous SRAM and a synchronous burst SRAM
- Explain the purpose of a cache memory
- Explain what a DRAM is
- Describe the DRAM storage cells
- Discuss the types of DRAM
- Compare the SRAM with the DRAM

The RAM Family

The two major categories of RAM are the *static RAM* (SRAM) and the *dynamic RAM* (DRAM). **SRAMs** generally use latches as storage elements and can therefore store data indefinitely *as long as dc power is applied.* **DRAMs** use capacitors as storage elements and cannot retain data very long without the capacitors being recharged by a process called **refreshing.** Both SRAMs and DRAMs will lose stored data when dc power is removed and, therefore, are classified as volatile memories.

Data can be read much faster from SRAMs than from DRAMs. However, DRAMs can store much more data than SRAMs for a given physical size and cost because the DRAM cell is much simpler and more cells can be crammed into a given chip area than in the SRAM.

The basic types of SRAM are the *asynchronous SRAM* and the *synchronous SRAM* with a burst feature. The basic types of DRAM are the *Fast Page Mode DRAM* (FPM DRAM), the *Extended Data Out DRAM* (EDO DRAM), the *Burst EDO DRAM* (BEDO DRAM), and the *synchronous DRAM* (SDRAM). These are shown in Figure 11–8.

Static RAMs (SRAMs)

Memory Cell

All SRAMs are characterized by latch memory cells. As long as dc power is applied to a **static memory** cell, it can retain a 1 or 0 state indefinitely. If power is removed, the stored data bit is lost.

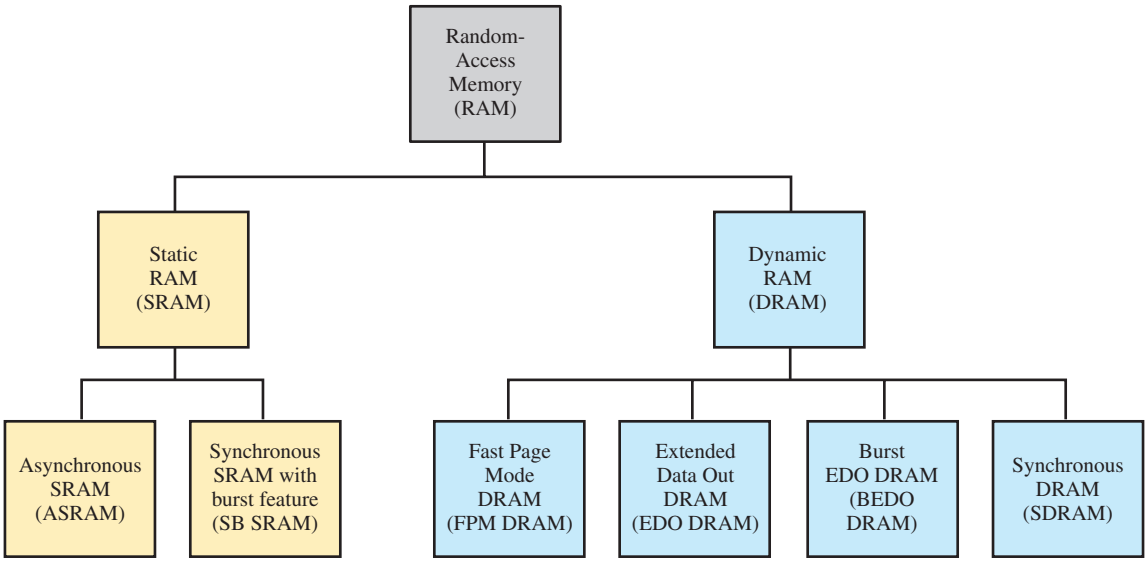


FIGURE 11–8 The RAM family.

Figure 11–9 shows a basic SRAM latch memory cell. The cell is selected by an active level on the Select line and a data bit (1 or 0) is written into the cell by placing it on the Data in line. A data bit is read by taking it off the Data out line.

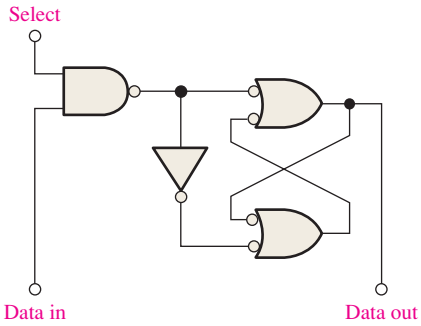


FIGURE 11–9 A typical SRAM latch memory cell.

Static Memory Cell Array

The memory cells in a SRAM are organized in rows and columns, as illustrated in Figure 11–10 for the case of an $n \times 4$ array. All the cells in a row share the same Row Select line. Each set of Data in and Data out lines go to each cell in a given column and are connected to a single data line that serves as both an input and output (Data I/O) through the data input and data output buffers.

To write a data unit, in this case a nibble (4 bits), into a given row of cells in the memory array, the Row Select line is taken to its active state and four data bits are placed on the Data I/O lines. The Write line is then taken to its active state, which causes each data bit to be stored in a selected cell in the associated column. To read a data unit, the Read line is taken to its active state, which causes the four data bits stored in the selected row to appear on the Data I/O lines.

Basic Asynchronous SRAM Organization

An asynchronous SRAM is one in which the operation is not synchronized with a system clock. To illustrate the general organization of a SRAM, a 32k $\times$ 8 bit memory is used. A logic symbol for this memory is shown in Figure 11–11.

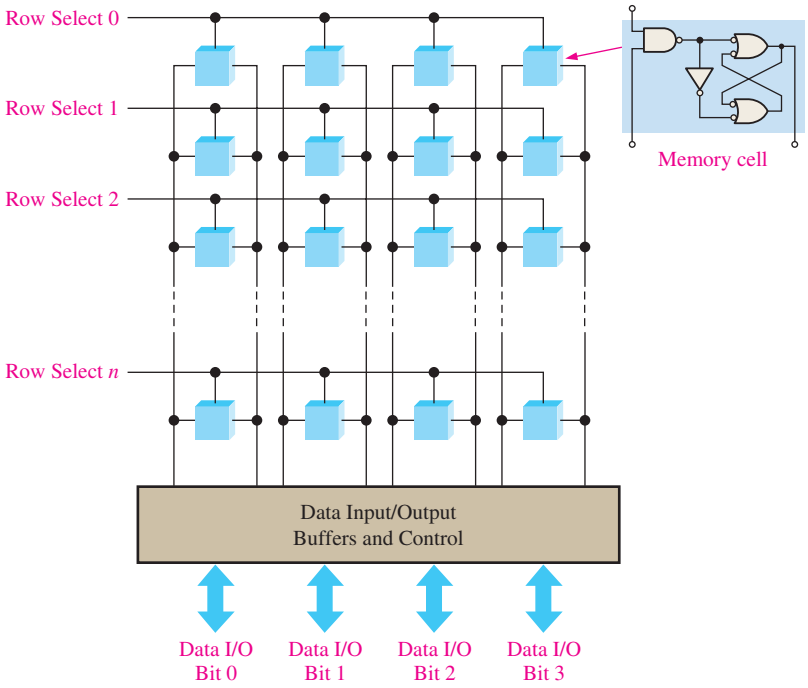


FIGURE 11–10 Basic SRAM array.

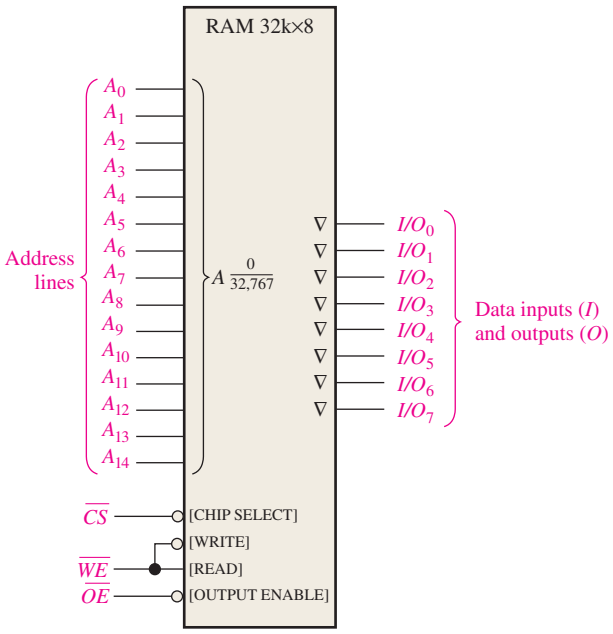


FIGURE 11–11 Logic diagram for an asynchronous 32k × 8 SRAM.

In the READ mode, the eight data bits that are stored in a selected address appear on the data output lines. In the WRITE mode, the eight data bits that are applied to the data input lines are stored at a selected address. The data input and data output lines (I/O_0 through I/O_7) share the same lines. During READ, they act as output lines (O_0 through O_7) and during WRITE they act as input lines (I_0 through I_7).

Tri-state Outputs and Buses

Tri-state buffers in a memory allow the data lines to act as either input or output lines and connect the memory to the data bus in a computer. These buffers have three output states:

HIGH (1), LOW (0), and HIGH-*Z* (open). Tri-state outputs are indicated on logic symbols by a small inverted triangle (∇), as shown in Figure 11–11, and are used for compatibility with bus structures such as those found in microprocessor-based systems.

Physically, a **bus** is one or more conductive paths that serve to interconnect two or more functional components of a system or several diverse systems. Electrically, a bus is a collection of specified voltage levels and/or current levels and signals that allow various devices to communicate and work properly together.

A microprocessor is connected to memories and input/output devices by certain bus structures. An address bus allows the microprocessor to address the memories, and a data bus provides for transfer of data between the microprocessor, the memories, and the input/output devices such as monitors, printers, keyboards, and modems. A control bus allows the microprocessor to control data transfers and timing for the various components.

Memory Array

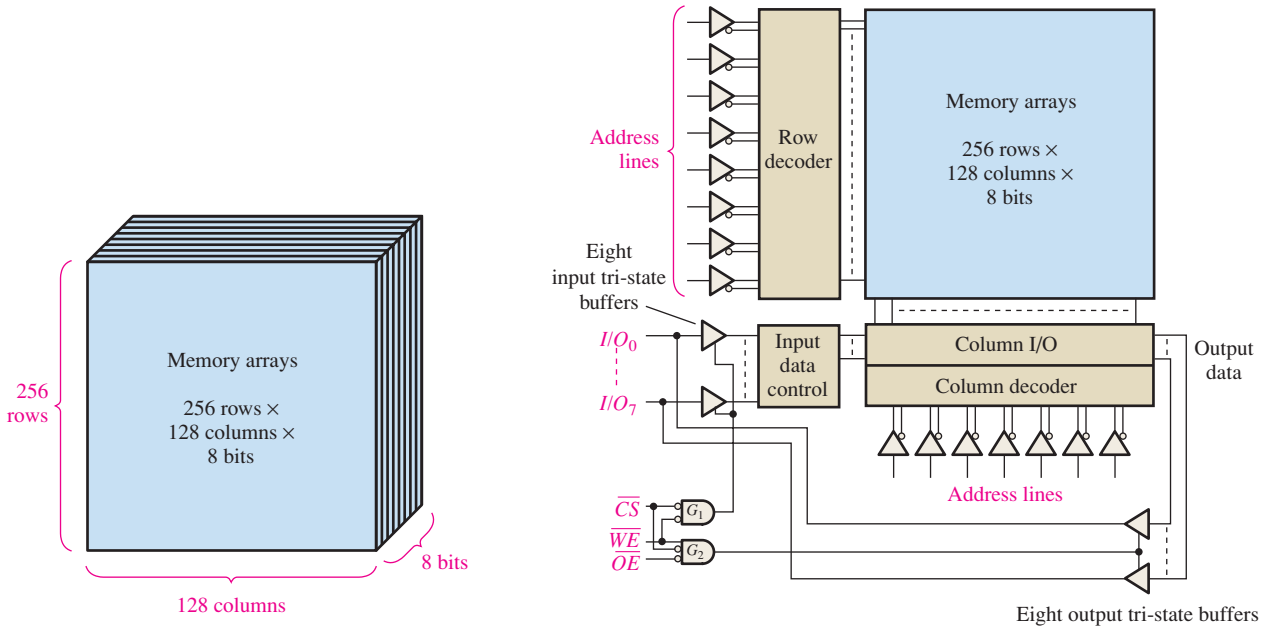
SRAM chips can be organized in single bits, nibbles (4 bits), bytes (8 bits), or multiple bytes (words with 16, 24, 32 bits, etc.).

Figure 11–12 shows the organization of a small 32k × 8 SRAM. The memory cell array is arranged in 256 rows and 128 columns, each with 8 bits, as shown in part (a). There are actually $2^{15} = 32{,}768$ addresses and each address contains 8 bits. The capacity of this example memory is 32,768 bytes (typically expressed as 32 kB). Although small by today's standards, this memory serves to introduce the basic concepts.

The SRAM in Figure 11–12(b) works as follows. First, the chip select, $\overline{CS}$, must be LOW for the memory to operate. (Other terms for chip select are *enable* or *chip enable.*) Eight of the fifteen address lines are decoded by the row decoder to select one of the 256 rows. Seven of the fifteen address lines are decoded by the column decoder to select one of the 128 8-bit columns.

Read

In the READ mode, the write enable input, $\overline{WE}$, is HIGH and the output enable, $\overline{OE}$, is LOW. The input tri-state buffers are disabled by gate G_1, and the column output tri-state



(a) Memory array configuration

(b) Memory block diagram

FIGURE 11–12 Basic organization of an asynchronous 32k × 8 SRAM.

buffers are enabled by gate G_2. Therefore, the eight data bits from the selected address are routed through the column I/O to the data lines (I/O_0 though I/O_7), which are acting as data output lines.

Write

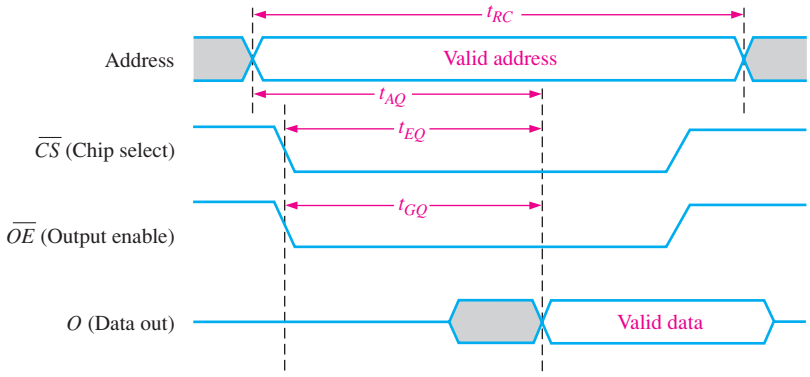
In the WRITE mode, $\overline{WE}$ is LOW and $\overline{OE}$ is HIGH. The input tri-state buffers are enabled by gate G_1, and the output tri-state buffers are disabled by gate G_2. Therefore, the eight input data bits on the data lines are routed through the input data control and the column I/O to the selected address and stored.

Read and Write Cycles

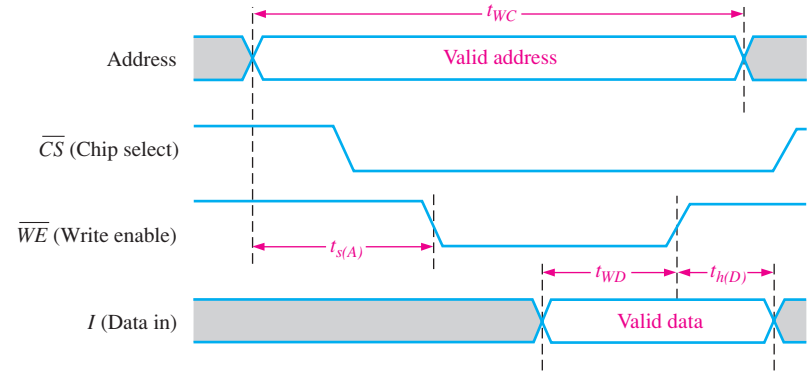
Figure 11–13 shows typical timing diagrams for a memory read cycle and a write cycle. For the read cycle shown in part (a), a valid address code is applied to the address lines for a specified time interval called the *read cycle time,* t_{RC}. Next, the chip select ($\overline{CS}$) and the output enable ($\overline{OE}$) inputs go LOW. One time interval after the $\overline{OE}$ input goes LOW, a valid data byte from the selected address appears on the data lines. This time interval is called the *output enable access time,* t_{GQ}. Two other access times for the read cycle are the *address access time,* t_{AQ}, measured from the beginning of a valid address to the appearance of valid data on the data lines and the *chip enable access time,* t_{EQ}, measured from the HIGH-to-LOW transition of $\overline{CS}$ to the appearance of valid data on the data lines.

During each read cycle, one unit of data, a byte in this case, is read from the memory.

For the write cycle shown in Figure 11–13(b), a valid address code is applied to the address lines for a specified time interval called the *write cycle time,* t_{WC}. Next, the chip



(a) Read cycle ($\overline{WE}$ HIGH)



(b) Write cycle ($\overline{WE}$ LOW)

FIGURE 11–13 Timing diagrams for typical read and write cycles for the SRAM in Figure 11–12.

select ($\overline{CS}$) and the write enable ($\overline{WE}$) inputs go LOW. The required time interval from the beginning of a valid address until the $\overline{WE}$ input goes LOW is called the *address setup time,* $t_{s(A)}$. The time that the $\overline{WE}$ input must be LOW is the write pulse width. The time that the input $\overline{WE}$ must remain LOW after valid data are applied to the data inputs is designated t_{WD}; the time that the valid input data must remain on the data lines after the $\overline{WE}$ input goes HIGH is the *data hold time,* $t_{h(D)}$.

During each write cycle, one unit of data is written into the memory.

Synchronous SRAM with Burst Feature

Unlike the asynchronous SRAM, a synchronous SRAM is synchronized with the system clock. For example, in a computer system, the synchronous SRAM operates with the same clock signal that operates the microprocessor so that the microprocessor and memory are synchronized for faster operation.

The fundamental concept of the synchronous feature of a SRAM can be shown with Figure 11–14, which is a simplified block diagram of a 32k × 8 memory for purposes of illustration. The synchronous SRAM is similar to the asynchronous SRAM in terms of the memory array, address decoder, and read/write and enable inputs. The basic difference is that the synchronous SRAM uses clocked registers to synchronize all inputs with the system clock. The address, the read/write input, the chip enable, and the input data are all latched into their respective registers on an active clock pulse edge. Once this information is latched, the memory operation is in sync with the clock.

For the purpose of simplification, a notation for multiple parallel lines or bus lines is introduced in Figure 11–14, as an alternative to drawing each line separately. A set of

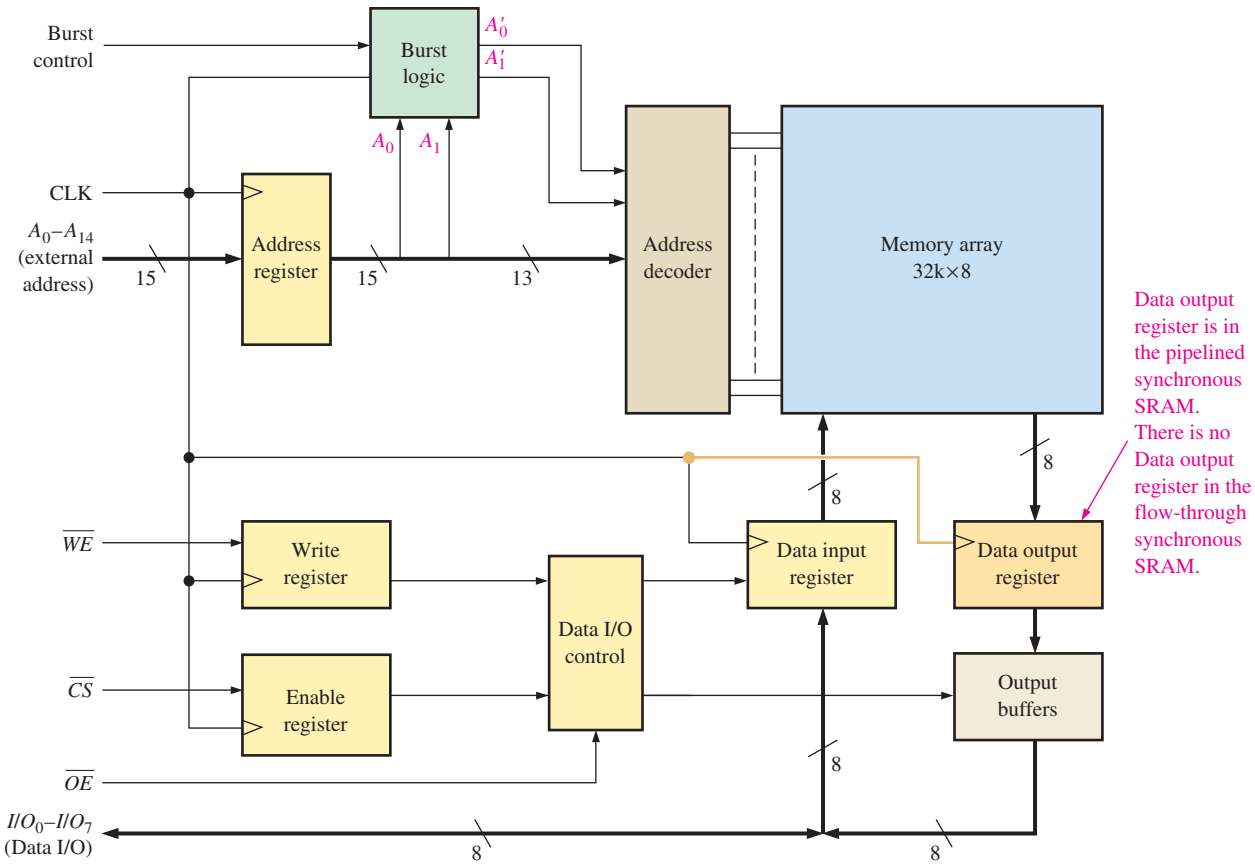
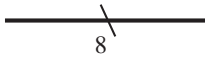


FIGURE 11–14 A basic block diagram of a synchronous SRAM with burst feature.

parallel lines can be indicated by a single heavy line with a slash and the number of separate lines in the set. For example, the following notation represents a set of 8 parallel lines:



The address bits A_0 through A_{14} are latched into the Address register on the positive edge of a clock pulse. On the same clock pulse, the state of the write enable ($\overline{WE}$) line and chip select ($\overline{CS}$) are latched into the Write register and the Enable register respectively. These are one-bit registers or simply flip-flops. Also, on the same clock pulse the input data are latched into the Data input register for a Write operation, and data in a selected memory address are latched into the Data output register for a Read operation, as determined by the Data I/O control based on inputs from the Write register, Enable register, and the Output enable ($\overline{OE}$).

Two basic types of synchronous SRAM are the *flow-through* and the *pipelined.* The flow-through synchronous SRAM does not have a Data output register, so the output data flow asynchronously to the data I/O lines through the output buffers. The **pipelined** synchronous SRAM has a Data output register, as shown in Figure 11–14, so the output data are synchronously placed on the data I/O lines.

The Burst Feature

As shown in Figure 11–14, synchronous SRAMs normally have an address burst feature, which allows the memory to read or write up to four sequential locations using a single address. When an external address is latched in the address register, the two lowest-order address bits, A_0 and A_1, are applied to the burst logic. This produces a sequence of four internal addresses by adding 00, 01, 10, and 11 to the two lowest-order address bits on successive clock pulses. The sequence always begins with the base address, which is the external address held in the address register.

The address burst logic in a typical synchronous SRAM consists of a binary counter and exclusive-OR gates, as shown in Figure 11–15. For 2-bit burst logic, the internal burst address sequence is formed by the base address bits A_2–A_{14} plus the two burst address bits A_1' and A_0'.

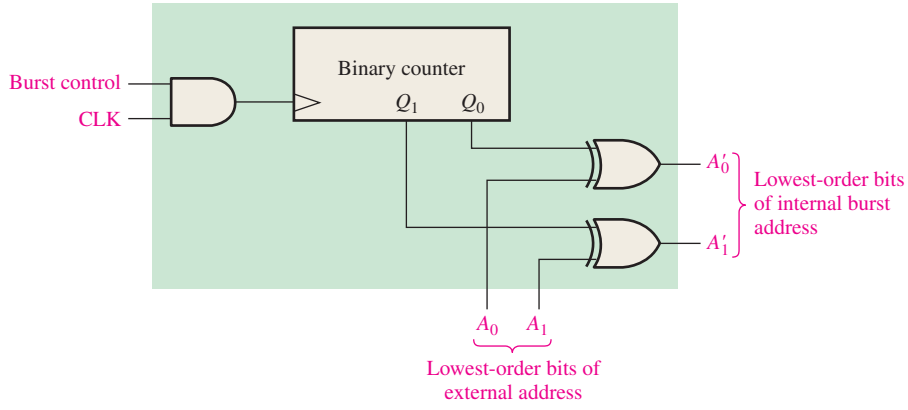


FIGURE 11–15 Address burst logic.

To begin the burst sequence, the counter is in its 00 state and the two lowest-order address bits are applied to the inputs of the XOR gates. Assuming that A_0 and A_1 are both 0, the internal address sequence in terms of its two lowest-order bits is 00, 01, 10, and 11.

Cache Memory

One of the major applications of SRAMs is in cache memories in computers. **Cache memory** is a relatively small, high-speed memory that stores the most recently used instructions or data from the larger but slower main memory. Cache memory can also use dynamic

RAM (DRAM), which is discussed next. Typically, SRAM is several times faster than DRAM. Overall, a cache memory gets stored information to the microprocessor much faster than if only high-capacity DRAM is used. Cache memory is basically a cost-effective method of improving system performance without having to resort to the expense of making all of the memory faster.

The concept of cache memory is based on the idea that computer programs tend to get instructions or data from one area of main memory before moving to another area. Basically, the cache controller "guesses" which area of the slow dynamic memory the CPU (central-processing unit) will need next and moves it to the cache memory so that it is ready when needed. If the cache controller guesses right, the data are immediately available to the microprocessor. If the cache controller guesses wrong, the CPU must go to the main memory and wait much longer for the correct instructions or data. Fortunately, the cache controller is right most of the time.

Cache Analogy

There are many analogies that can be used to describe a cache memory, but comparing it to a home refrigerator is perhaps the most effective. A home refrigerator can be thought of as a "cache" for certain food items while the supermarket is the main memory where all foods are kept. Each time you want something to eat or drink, you can go to the refrigerator (cache) first to see if the item you want is there. If it is, you save a lot of time. If it is not there, then you have to spend extra time to get it from the supermarket (main memory).

L1 and L2 Caches

A first-level cache (L1 cache) is usually integrated into the processor chip and has a very limited storage capacity. L1 cache is also known as *primary cache*. A second-level cache (L2 cache) may also be integrated into the processor or as a separate memory chip or set of chips external to the processor; it usually has a larger storage capacity than an L1 cache. L2 cache is also known as *secondary cache*. Some systems may have higher-level caches (L3, L4, etc.), but L1 and L2 are the most common. Also, some systems use a disk cache to enhance the performance of the hard disk because DRAM, although much slower than SRAM, is much faster than the hard disk drive. Figure 11–16 illustrates L1 and L2 cache memories in a computer system.

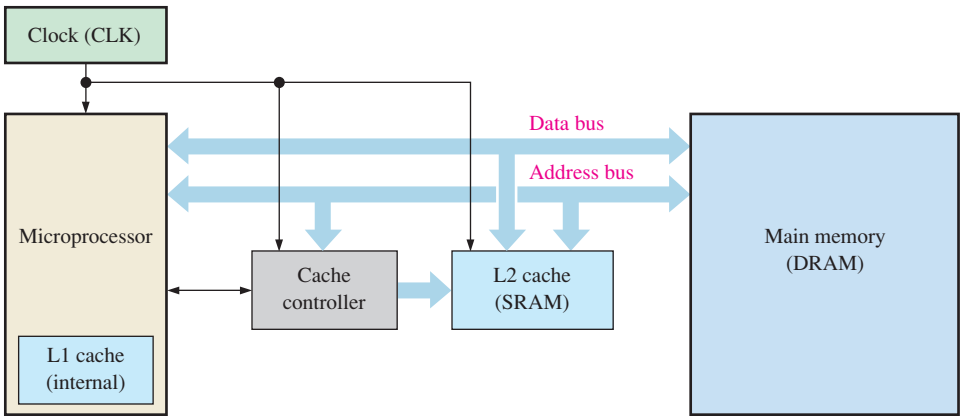


FIGURE 11–16 Block diagram showing L1 and L2 cache memories in a computer system.

Dynamic RAM (DRAM) Memory Cells

Dynamic memory cells store a data bit in a small capacitor rather than in a latch. The advantage of this type of cell is that it is very simple, thus allowing very large memory arrays to be constructed on a chip at a lower cost per bit. The disadvantage is that the

storage capacitor cannot hold its charge over an extended period of time and will lose the stored data bit unless its charge is refreshed periodically. To refresh requires additional memory circuitry and complicates the operation of the DRAM. Figure 11–17 shows a typical DRAM cell consisting of a single MOS transistor (MOSFET) and a capacitor.

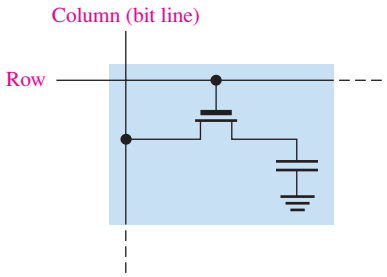


FIGURE 11–17 A MOS DRAM cell.

In this type of cell, the transistor acts as a switch. The basic simplified operation is illustrated in Figure 11–18 and is as follows. A LOW on the $R/\overline{W}$ line (WRITE mode) enables the tri-state input buffer and disables the output buffer. For a 1 to be written into the cell, the D_{IN} line must be HIGH, and the transistor must be turned on by a HIGH on the row line. The transistor acts as a closed switch connecting the capacitor to the bit line. This connection allows the capacitor to charge to a positive voltage, as shown in Figure 11–18(a). When a 0 is to be stored, a LOW is applied to the D_{IN} line. If the capacitor is storing a 0, it remains uncharged, or if it is storing a 1, it discharges as indicated in Figure 11–18(b). When the row line is taken back LOW, the transistor turns off and disconnects the capacitor from the bit line, thus "trapping" the charge (1 or 0) on the capacitor.

To read from the cell, the $R/\overline{W}$ (Read/$\overline{\text{Write}}$) line is HIGH, enabling the output buffer and disabling the input buffer. When the row line is taken HIGH, the transistor turns on and connects the capacitor to the bit line and thus to the output buffer (sense amplifier), so the data bit appears on the data-output line (D_{OUT}). This process is illustrated in Figure 11–18(c).

For refreshing the memory cell, the $R/\overline{W}$ line is HIGH, the row line is HIGH, and the refresh line is HIGH. The transistor turns on, connecting the capacitor to the bit line. The output buffer is enabled, and the stored data bit is applied to the input of the refresh buffer, which is enabled by the HIGH on the refresh input. This produces a voltage on the bit line corresponding to the stored bit, thus replenishing the capacitor. This is illustrated in Figure 11–18(d).

DRAM Organization

The major application of DRAMs is in the main memory of computers. The difference between DRAMs and SRAMs is the type of memory cell. As you have seen, the DRAM memory cell consists of one transistor and a capacitor and is much simpler than the SRAM cell. This allows much greater densities in DRAMs and results in greater bit capacities for a given chip area, although much slower access time.

Again, because charge stored in a capacitor will leak off, the DRAM cell requires a frequent refresh operation to preserve the stored data bit. This requirement results in more complex circuitry than in a SRAM. Several features common to most DRAMs are now discussed, using a generic 1M × 1 bit DRAM as an example.

Address Multiplexing

DRAMs use a technique called *address multiplexing* to reduce the number of address lines. Figure 11–19 shows the block diagram of a 1,048,576-bit (1 Mb) DRAM with a 1M × 1

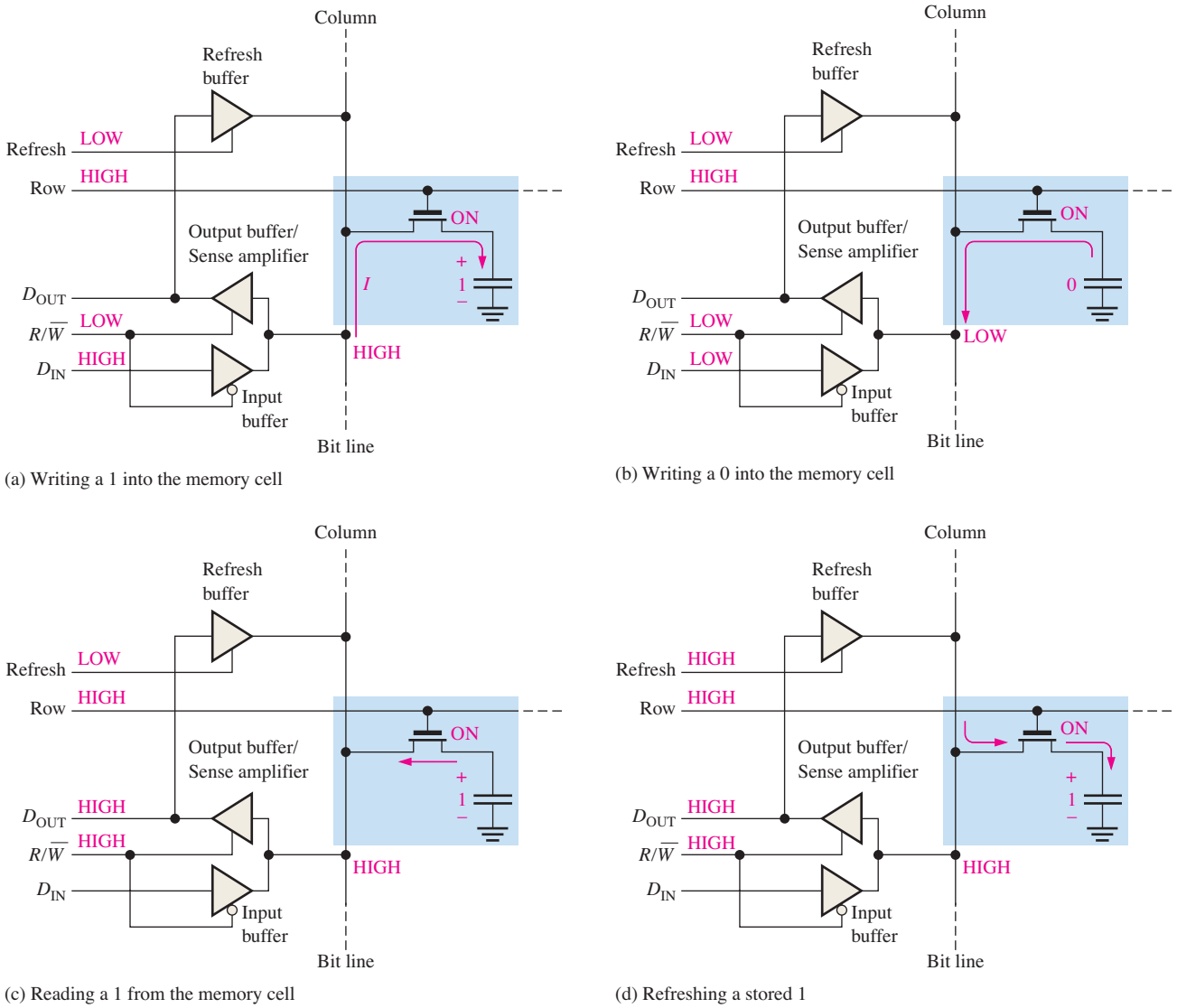


FIGURE 11–18 Basic operation of a DRAM cell.

organization. We will focus on the blue blocks to illustrate address multiplexing. The green blocks represent the refresh logic.

The ten address lines are time multiplexed at the beginning of a memory cycle by the row address select ($\overline{RAS}$) and the column address select ($\overline{CAS}$) into two separate 10-bit address fields. First, the 10-bit row address is latched into the row address register. Next, the 10-bit column address is latched into the column address register. The row address and the column address are decoded to select one of the 1,048,576 addresses ($2^{20} = 1{,}048{,}576$) in the memory array. The basic timing for the address multiplexing operation is shown in Figure 11–20.

Read and Write Cycles

At the beginning of each read or write memory cycle, $\overline{RAS}$ and $\overline{CAS}$ go active (LOW) to multiplex the row and column addresses into the registers, and decoders. For a read cycle, the $R/\overline{W}$ input is HIGH. For a write cycle, the $R/\overline{W}$ input is LOW. This is illustrated in Figure 11–21.

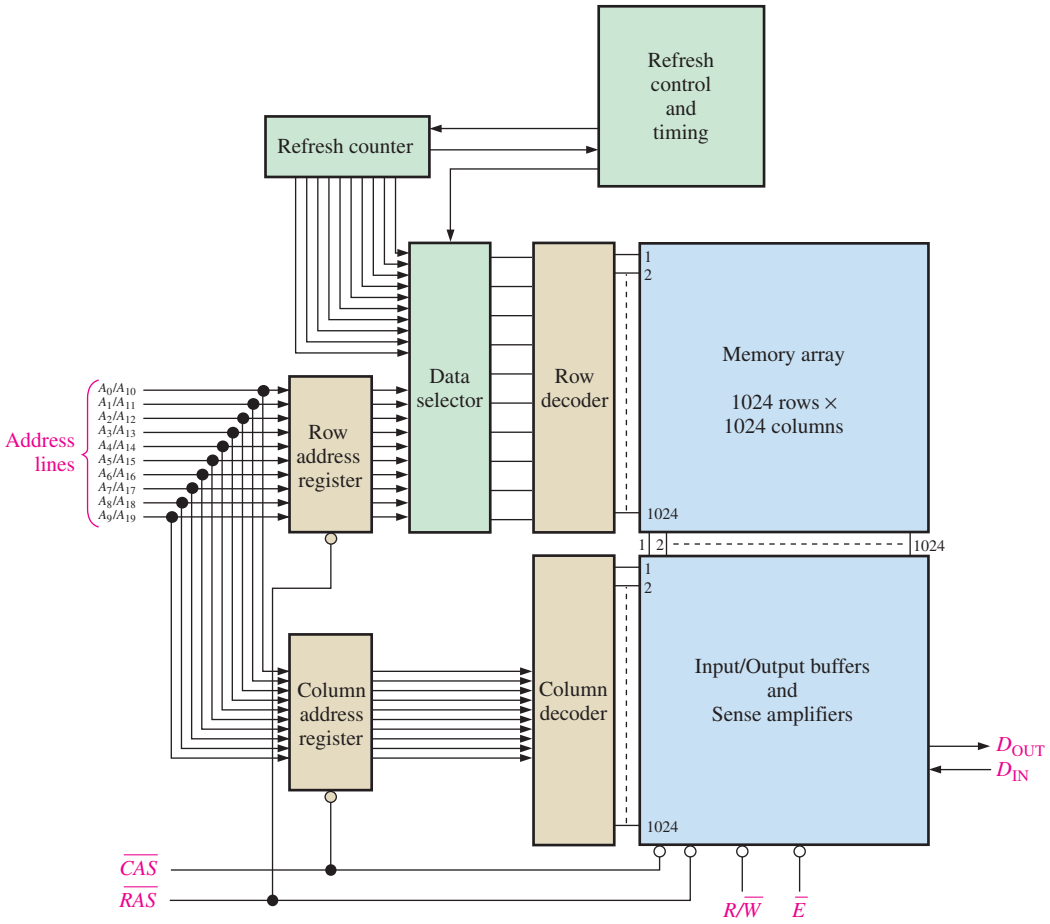


FIGURE 11–19 Simplified block diagram of a 1M × 1 DRAM.

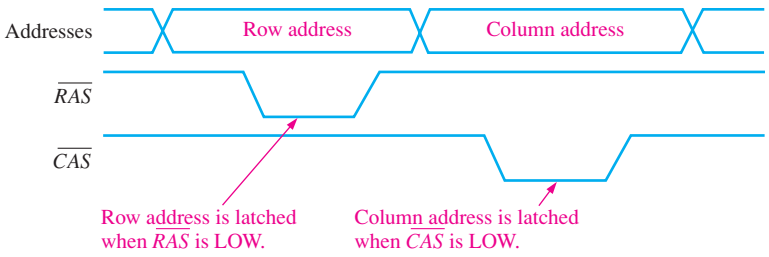


FIGURE 11–20 Basic timing for address multiplexing.

Fast Page Mode

In the normal read or write cycle described previously, the row address for a particular memory location is first loaded by an active-LOW $\overline{RAS}$ and then the column address for that location is loaded by an active-LOW $\overline{CAS}$. The next location is selected by another $\overline{RAS}$ followed by a $\overline{CAS}$, and so on.

A "page" is a section of memory available at a single row address and consists of all the columns in a row. Fast page mode allows fast successive read or write operations at each column address in a selected row. A row address is first loaded by $\overline{RAS}$ going LOW and remaining LOW while $\overline{CAS}$ is toggled between HIGH and LOW. A single row address is selected and remains selected while $\overline{RAS}$ is active. Each successive $\overline{CAS}$ selects another column in the selected row. So, after a fast page mode cycle, all of the addresses in the

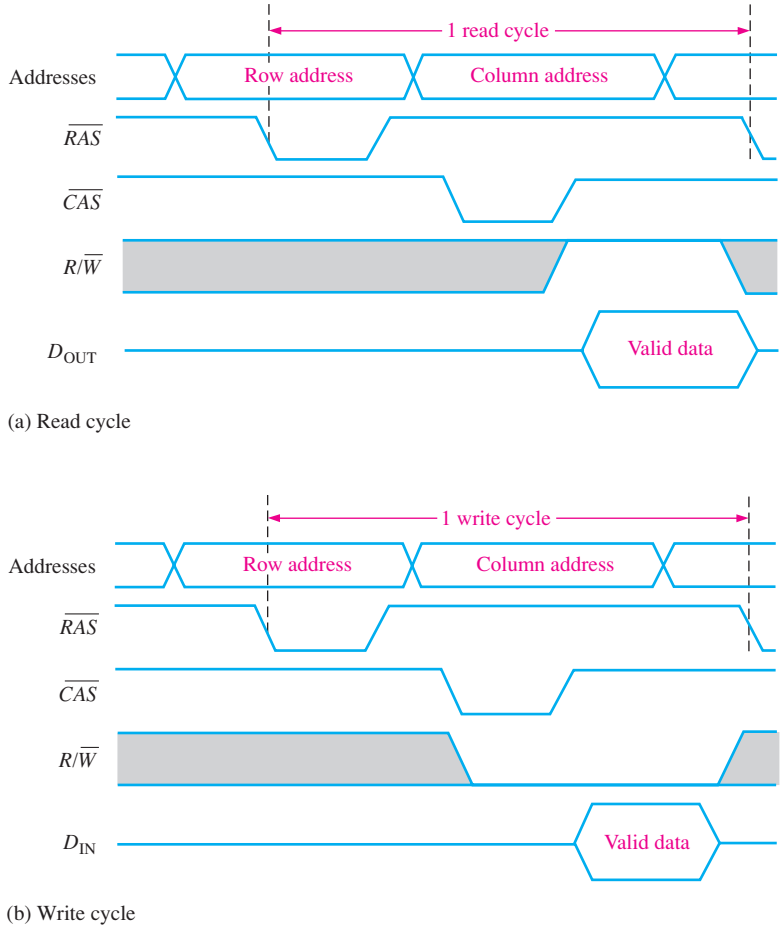


FIGURE 11–21 Timing diagrams for normal read and write cycles.

selected row have been read from or written into, depending on $R/\overline{W}$. For example, a fast page mode cycle for the DRAM in Figure 11–19 requires $\overline{CAS}$ to go active 1024 times for each row selected by $\overline{RAS}$.

Fast page mode operation for read is illustrated by the timing diagram in Figure 11–22. When $\overline{CAS}$ goes to its nonasserted state (HIGH), it disables the data outputs. Therefore, the transition of $\overline{CAS}$ to HIGH must occur only after valid data are latched by the external system.

Refresh Cycles

As you know, DRAMs are based on capacitor charge storage for each bit in the memory array. This charge degrades (leaks off) with time and temperature, so each bit must be periodically refreshed (recharged) to maintain the correct bit state. Typically, a DRAM must be refreshed every several milliseconds, although for some devices the refresh period can be much longer.

A read operation automatically refreshes all the addresses in the selected row. However, in typical applications, you cannot always predict how often there will be a read cycle, and so you cannot depend on a read cycle to occur frequently enough to prevent data loss. Therefore, special refresh cycles must be implemented in DRAM systems.

Burst refresh and *distributed refresh* are the two basic refresh modes for refresh operations. In burst refresh, all rows in the memory array are refreshed consecutively each refresh period. For a memory with a refresh period of 8 ms, a burst refresh of all rows occurs once every 8 ms. The normal read and write operations are suspended during a burst

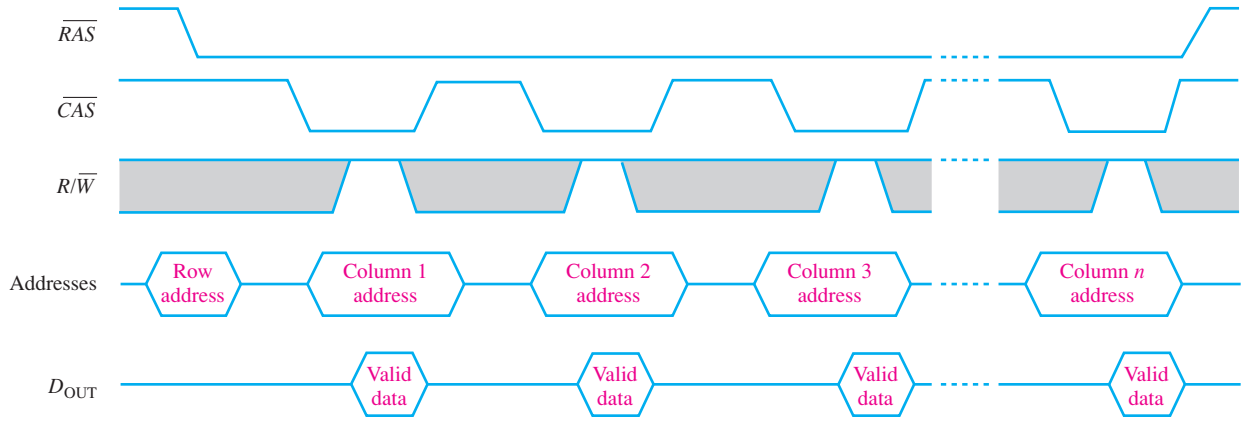


FIGURE 11–22 Fast page mode timing for a read operation.

refresh cycle. In distributed refresh, each row is refreshed at intervals interspersed between normal read or write cycles. For example, the memory in Figure 11–19 has 1024 rows. As an example, for an 8 ms refresh period, each row must be refreshed every 8 ms/1024 = 7.8 μs when distributed refresh is used.

The two types of refresh operations are $\overline{RAS}$ *only refresh* and $\overline{CAS}$ *before* $\overline{RAS}$ *refresh.* $\overline{RAS}$-only refresh consists of a $\overline{RAS}$ transition to the LOW (active) state, which latches the address of the row to be refreshed while $\overline{CAS}$ remains HIGH (inactive) throughout the cycle. An external counter is used to provide the row addresses for this type of operation.

The $\overline{CAS}$ before $\overline{RAS}$ refresh is initiated by $\overline{CAS}$ going LOW before $\overline{RAS}$ goes LOW. This sequence activates an internal refresh counter that generates the row address to be refreshed. This address is switched by the data selector into the row decoder.

Types of DRAMs

Now that you have learned the basic concept of a DRAM, let's briefly look at the major types. These are the *Fast Page Mode (FPM) DRAM,* the *Extended Data Out (EDO) DRAM,* the *Burst Extended Data Out (BEDO) DRAM,* and the *Synchronous (S) DRAM.*

FPM DRAM

Fast page mode operation was described earlier. Recall that a page in memory is all of the column addresses contained within one row address.

The idea of the **FPM DRAM** is based on the probability that the next several memory addresses to be accessed are in the same row (on the same page). Fortunately, this happens a large percentage of the time. FPM saves time over pure random accessing because in FPM the row address is specified only once for access to several successive column addresses whereas for pure random accessing, a row address is specified for each column address.

Recall that in a fast page mode read operation, the $\overline{CAS}$ signal has to wait until the valid data from a given address are accepted (latched) by the external system (CPU) before it can go to its nonasserted state. When $\overline{CAS}$ goes to its nonasserted state, the data outputs are disabled. This means that the next column address cannot occur until after the data from the current column address are transferred to the CPU. This limits the rate at which the columns within a page can be addressed.

EDO DRAM

The Extended Data Out DRAM, sometimes called *hyper page mode DRAM,* is similar to the FPM DRAM. The key difference is that the $\overline{CAS}$ signal in the **EDO DRAM** does not disable the output data when it goes to its nonasserted state because the valid data from the

current address can be held until $\overline{CAS}$ is asserted again. This means that the next column address can be accessed before the external system accepts the current valid data. The idea is to speed up the access time.

BEDO DRAM

The Burst Extended Data Out DRAM is an EDO DRAM with address burst capability. Recall from the discussion of the synchronous burst SRAM that the address burst feature allows up to four addresses to be internally generated from a single external address, which saves some access time. This same concept applies to the **BEDO DRAM.**

SDRAM

Faster DRAMs are needed to keep up with the ever-increasing speed of microprocessors. The Synchronous DRAM is one way to accomplish this. Like the synchronous SRAM discussed earlier, the operation of the **SDRAM** is synchronized with the system clock, which also runs the microprocessor in a computer system. The same basic ideas described in relation to the synchronous burst SRAM, also apply to the SDRAM.

This synchronized operation makes the SDRAM totally different from the other asynchronous DRAM types. With asynchronous memories, the microprocessor must wait for the DRAM to complete its internal operations. However, with synchronous operation, the DRAM latches addresses, data, and control information from the processor under control of the system clock. This allows the processor to handle other tasks while the memory read or write operations are in progress, rather than having to wait for the memory to do its thing as is the case in asynchronous systems.

DDR SDRAM

DDR stands for double data rate. A DDR SDRAM is clocked on both edges of a clock pulse, whereas a SDRAM is clocked on only one edge. Because of the double clocking, a DDR SDRAM is theoretically twice as fast as an SDRAM. Sometimes the SDRAM is referred to as an SDR SDRAM (single data rate SDRAM) for contrast with the DDR SDRAM.

SECTION 11–2 CHECKUP

1. List two types of SRAM.
2. What is a cache?
3. Explain how SRAMs and DRAMs differ.
4. Describe the refresh operation in a DRAM.
5. List four types of DRAM.

11–3 The Read-Only Memory (ROM)

A ROM contains permanently or semipermanently stored data, which can be read from the memory but either cannot be changed at all or cannot be changed without specialized equipment. A ROM stores data that are used repeatedly in system applications, such as tables, conversions, or programmed instructions for system initialization and operation. ROMs retain stored data when the power is off and are therefore nonvolatile memories.

After completing this section, you should be able to

- List the types of ROMs
- Describe a basic mask ROM storage cell
- Explain how data are read from a ROM
- Discuss internal organization of a typical ROM

The ROM Family

Figure 11–23 shows how semiconductor ROMs are categorized. The mask ROM is the type in which the data are permanently stored in the memory during the manufacturing process. The **PROM,** or programmable ROM, is the type in which the data are electrically stored by the user with the aid of specialized equipment. Both the mask ROM and the PROM can be of either MOS or bipolar technology. The **EPROM,** or erasable PROM, is strictly a MOS device. The **UV EPROM** is electrically programmable by the user, but the stored data must be erased by exposure to ultraviolet light over a period of several minutes. The electrically erasable PROM (**EEPROM** or E^2PROM) can be erased in a few milliseconds. The UV EPROM has been largely displaced by the EEPROM.

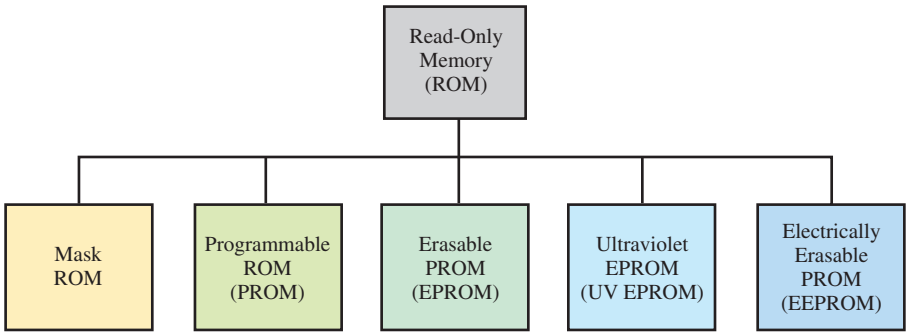


FIGURE 11–23 The ROM family.

The Mask ROM

The mask ROM is usually referred to simply as a ROM. It is permanently programmed during the manufacturing process to provide widely used standard functions, such as popular conversions, or to provide user-specified functions. Once the memory is programmed, it cannot be changed. Most IC ROMs utilize the presence or absence of a transistor connection at a row/column junction to represent a 1 or a 0.

Figure 11–24 shows MOS ROM cells. The presence of a connection from a row line to the gate of a transistor represents a 1 at that location because when the row line is taken HIGH, all transistors with a gate connection to that row line turn on and connect the HIGH (1) to the associated column lines. At row/column junctions where there are no gate connections, the column lines remain LOW (0) when the row is addressed.

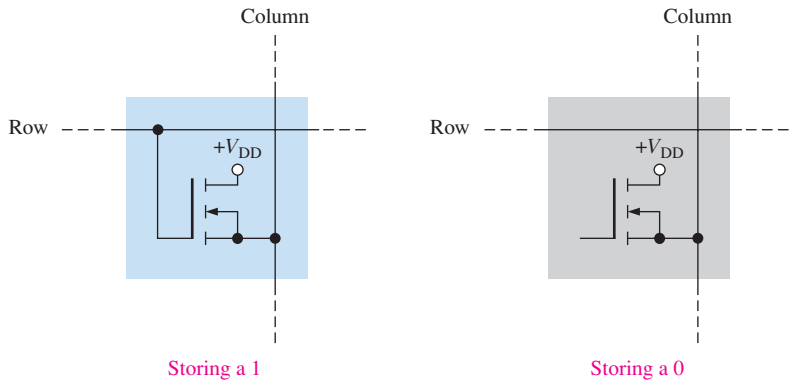


FIGURE 11–24 ROM cells.

To illustrate the ROM concept, Figure 11–25 shows a small, simplified ROM array. The blue squares represent stored 1s, and the gray squares represent stored 0s. The basic read operation is as follows. When a binary address code is applied to the address input lines, the

corresponding row line goes HIGH. This HIGH is connected to the column lines through the transistors at each junction (cell) where a 1 is stored. At each cell where a 0 is stored, the column line stays LOW because of the terminating resistor. The column lines form the data output. The eight data bits stored in the selected row appear on the output lines.

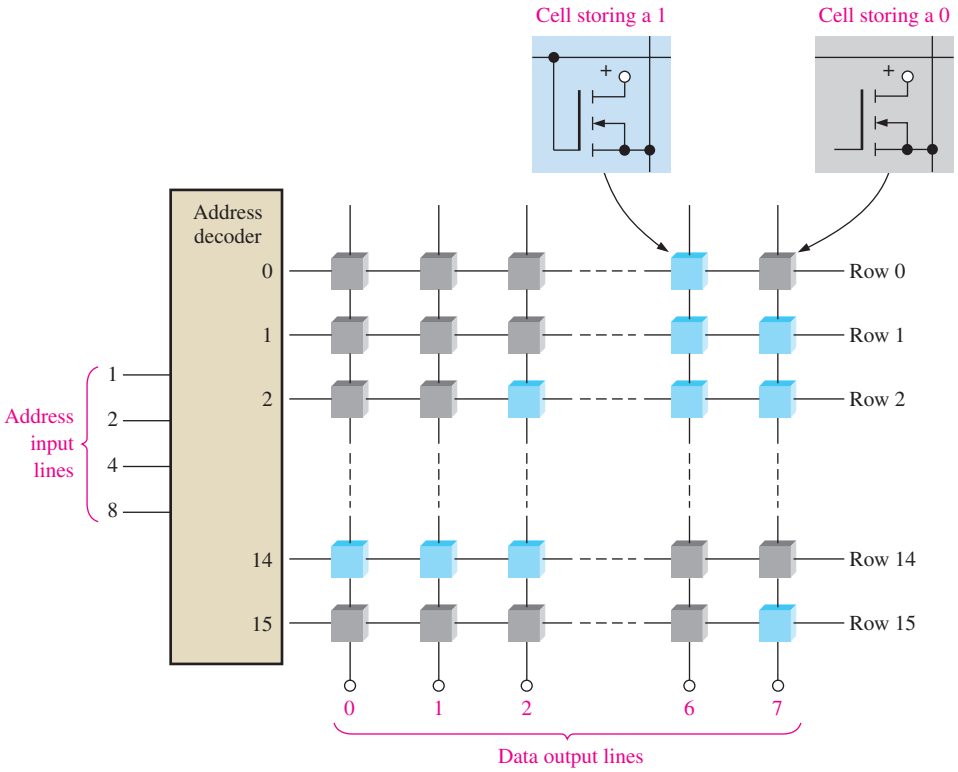


FIGURE 11–25 A representation of a 16 × 8-bit ROM array.

As you can see, the example ROM in Figure 11–25 is organized into 16 addresses, each of which stores 8 data bits. Thus, it is a 16 × 8 (16-by-8) ROM, and its total capacity is 128 bits or 16 bytes. ROMs can be used as look-up tables (LUTs) for code conversions and logic function generation.

EXAMPLE 11–1

Show a basic ROM, similar to the one in Figure 11–25, programmed for a 4-bit binary-to-Gray conversion.

Solution

Review Chapter 2 for the Gray code. Table 11–1 is developed for use in programming the ROM.

The resulting 16 × 4 ROM array is shown in Figure 11–26. You can see that a binary code on the address input lines produces the corresponding Gray code on the output lines (columns). For example, when the binary number 0110 is applied to the address input lines, address 6, which stores the Gray code 0101, is selected.

Related Problem*

Using Figure 11–26, determine the Gray code output when a binary code of 1011 is applied to the address input lines.

*Answers are at the end of the chapter.

TABLE 11–1

Binary				Gray			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

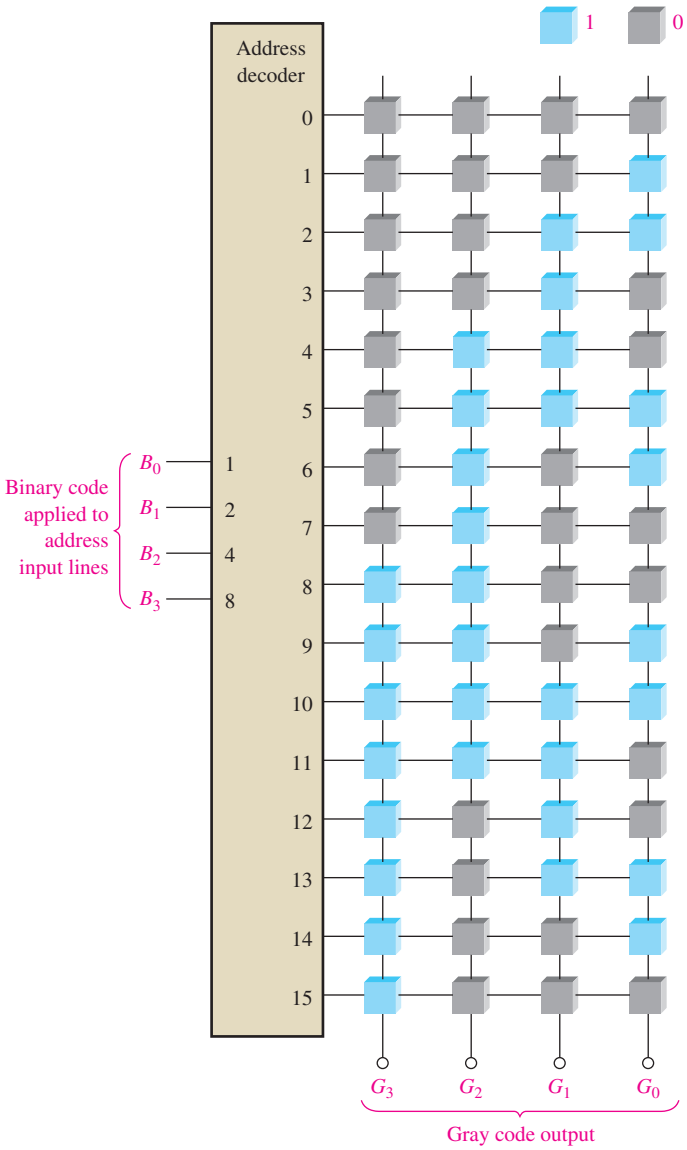


FIGURE 11–26 Representation of a ROM programmed as a binary-to-Gray code converter.

Internal ROM Organization

Most IC ROMs have a more complex internal organization than that in the basic simplified example just presented. To illustrate how an IC ROM is structured, let's use a 1024-bit device with a 256 × 4 organization. The logic symbol is shown in Figure 11–27. When any one of 256 binary codes (eight bits) is applied to the address lines, four data bits appear on the outputs if the chip select inputs are LOW. (256 addresses require eight address lines.)

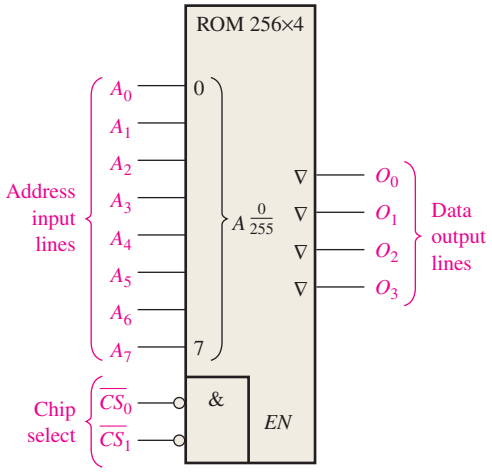


FIGURE 11–27 A 256 × 4 ROM logic symbol. The $A\frac{0}{255}$ designator means that the 8-bit address code selects addresses 0 through 255.

Although the 256 × 4 organization of this device implies that there are 256 rows and 4 columns in the memory array, this is not actually the case. The memory cell array is actually a 32 × 32 matrix (32 rows and 32 columns), as shown in the block diagram in Figure 11–28.

The ROM in Figure 11–28 works as follows. Five of the eight address lines (A_0 through A_4) are decoded by the row decoder (often called the Y decoder) to select one of the 32 rows. Three of the eight address lines (A_5 through A_7) are decoded by the column decoder (often called the X decoder) to select four of the 32 columns. Actually, the column decoder consists of four 1-of-8 decoders (data selectors), as shown in Figure 11–28.

The result of this structure is that when an 8-bit address code (A_0 through A_7) is applied, a 4-bit data word appears on the data outputs when the chip select lines ($\overline{CS}_0$ and $\overline{CS}_1$) are LOW to enable the output buffers. This type of internal organization (architecture) is typical of IC ROMs of various capacities.

InfoNote

ROM is used in a computer to store the BIOS (Basic Input/Output System). These are programs that are used to perform fundamental supervisory and support functions for the computer. For example, BIOS programs stored in the ROM control certain video monitor functions, provide for disk formatting, scan the keyboard for inputs, and control certain printer functions.

ROM Access Time

A typical timing diagram that illustrates ROM access time is shown in Figure 11–29. The **access time,** t_a, of a ROM is the time from the application of a valid address code on the input lines until the appearance of valid output data. Access time can also be measured from the activation of the chip select ($\overline{CS}$) input to the occurrence of valid output data when a valid address is already on the input lines.

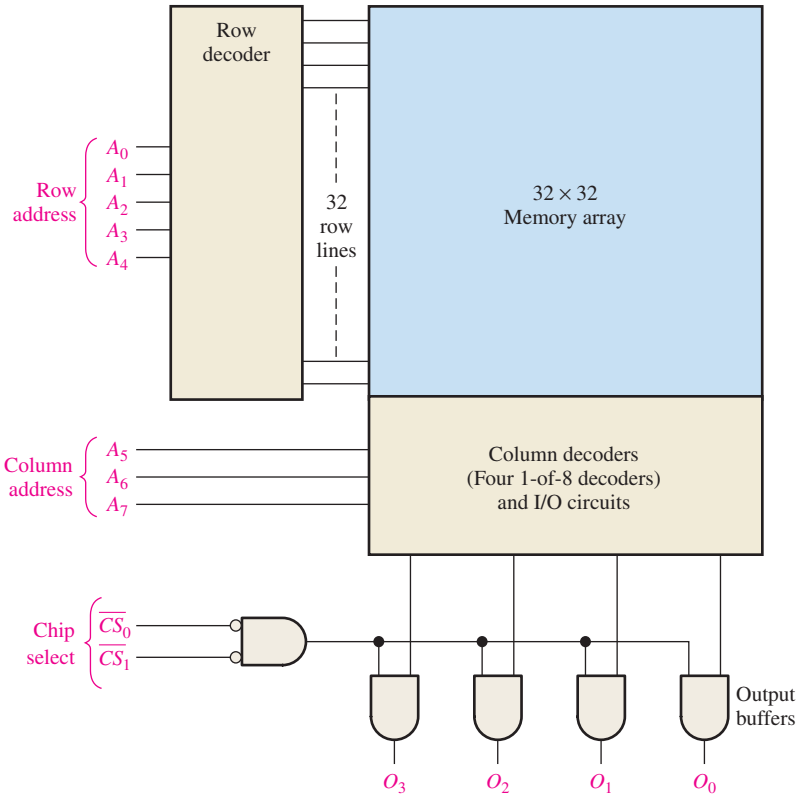


FIGURE 11–28 A 1024-bit ROM with a 256 × 4 organization based on a 32 × 32 array.

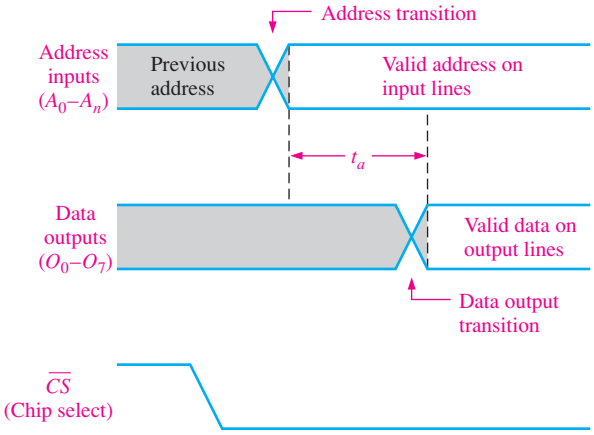


FIGURE 11–29 ROM access time (t_a) from address change to data output with chip select already active.

SECTION 11–3 CHECKUP

1. What is the bit storage capacity of a ROM with a 512 × 8 organization?
2. List the types of read-only memories.
3. How many address bits are required for a 2048-bit memory organized as a 256 × 8 memory?

11–4 Programmable ROMs

Programmable ROMs (PROMs) are basically the same as mask ROMs once they have been programmed. As you have learned, ROMs are a type of programmable logic device. The difference is that PROMs come from the manufacturer unprogrammed and are custom programmed in the field to meet the user's needs.

After completing this section, you should be able to

- Distinguish between a mask ROM and a PROM
- Describe a basic PROM memory cell
- Discuss EPROMs including UV EPROMs and EEPROMs
- Analyze an EPROM programming cycle

PROMs

A **PROM** uses some type of fusing process to store bits, in which a memory *link* is burned open or left intact to represent a 0 or a 1. The fusing process is irreversible; once a PROM is programmed, it cannot be changed.

Figure 11–30 illustrates a MOS PROM array with fusible links. The fusible links are manufactured into the PROM between the source of each cell's transistor and its column line. In the programming process, a sufficient current is injected through the fusible link to burn it open to create a stored 0. The link is left intact for a stored 1.

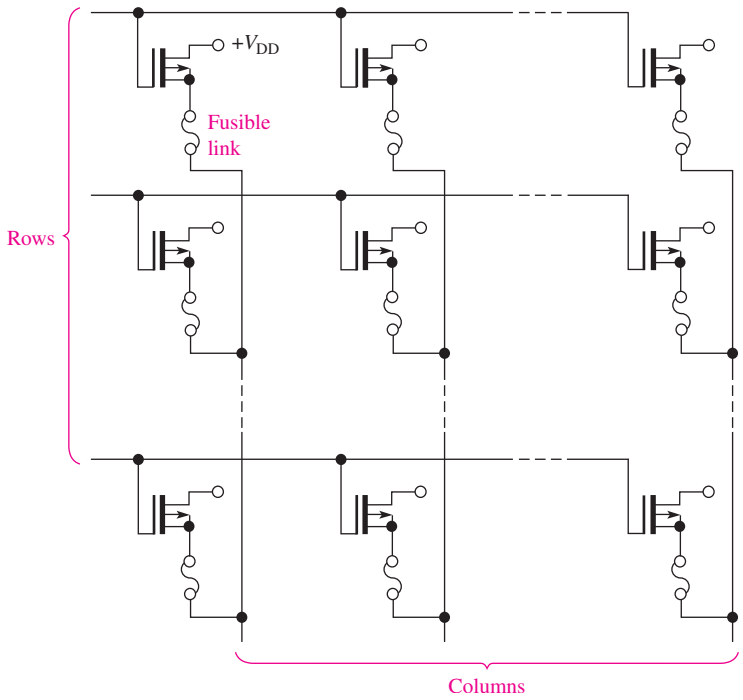


FIGURE 11–30 MOS PROM array with fusible links. (All drains are commonly connected to V_{DD}.)

Three basic fuse technologies used in PROMs are metal links, silicon links, and *pn* junctions. A brief description of each of these follows.

1. Metal links are made of a material such as nichrome. Each bit in the memory array is represented by a separate link. During programming, the link is either "blown" open

or left intact. This is done basically by first addressing a given cell and then forcing a sufficient amount of current through the link to cause it to open.

2. Silicon links are formed by narrow, notched strips of polycrystalline silicon. Programming of these fuses requires melting of the links by passing a sufficient amount of current through them. This amount of current causes a high temperature at the fuse location that oxidizes the silicon and forms an insulation around the now-open link.
3. Shorted junction, or avalanche-induced migration, technology consists basically of two *pn* junctions arranged back-to-back. During programming, one of the diode junctions is avalanched, and the resulting voltage and heat cause aluminum ions to migrate and short the junction. The remaining junction is then used as a forward-biased diode to represent a data bit.

EPROMs

An **EPROM** is an erasable PROM. Unlike an ordinary PROM, an EPROM can be reprogrammed if an existing program in the memory array is erased first.

An EPROM uses an NMOSFET array with an isolated-gate structure. The isolated transistor gate has no electrical connections and can store an electrical charge for indefinite periods of time. The data bits in this type of array are represented by the presence or absence of a stored gate charge. Erasure of a data bit is a process that removes the gate charge.

A typical EPROM is represented in Figure 11–31 by a logic diagram. Its operation is representative of that of other typical EPROMs of various sizes. As the logic symbol shows, this device has 2048 addresses ($2^{11} = 2048$), each with eight bits. Notice that the eight outputs are tri-state (∇).

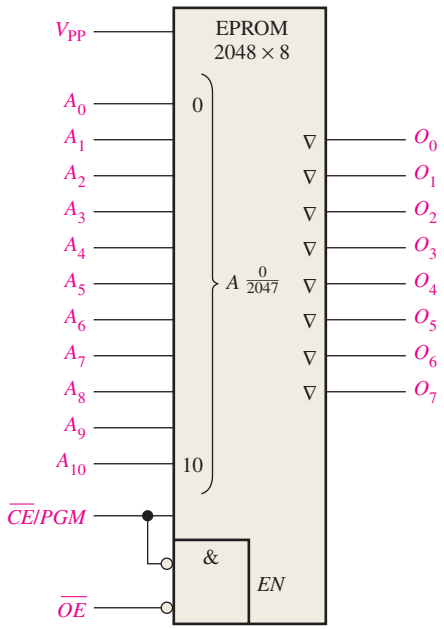


FIGURE 11–31 The logic symbol for a 2048 × 8 EPROM.

To read from the memory, the output enable input ($\overline{OE}$) must be LOW and the power-down/program ($\overline{CE}/PGM$) input LOW.

To program or write to the device, a high dc voltage is applied to V_{PP} and $\overline{OE}$ is HIGH. The eight data bits to be programmed into a given address are applied to the outputs (O_0

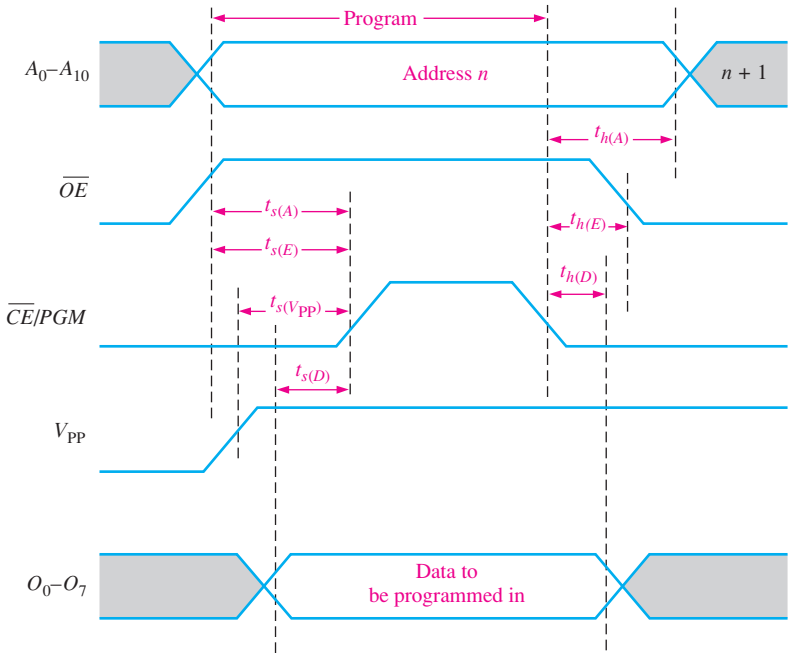


FIGURE 11–32 Timing diagram for a 2048 × 8 EPROM programming cycle, with critical setup times (t_s) and hold times (t_h) indicated.

through O_7), and the address is selected on inputs A_0 through A_{10}. Next, a HIGH level pulse is applied to the $\overline{CE}/PGM$ input. The addresses can be programmed in any order. A timing diagram for the programming is shown in Figure 11–32. These signals are normally produced by an EPROM programmer.

Two basic types of erasable PROMs are, the electrically erasable PROM (EEPROM) and the ultraviolet erasable PROM (UV EPROM). The UV EPROM is much less used than the EEPROM.

EEPROMs

An electrically erasable PROM can be both erased and programmed with electrical pulses. Since it can be both electrically written into and electrically erased, the EEPROM can be rapidly programmed and erased in-circuit for reprogramming. Two types of EEPROMs are the floating-gate MOS and the metal nitride-oxide silicon (MNOS). The application of a voltage on the control gate in the floating-gate structure permits the storage and removal of charge from the floating gate.

UV EPROMs

You can recognize the UV EPROM device by the UV transparent window on the package. The isolated gate in the **FET** of an ultraviolet EPROM is "floating" within an oxide insulating material. The programming process causes electrons to be removed from the floating gate. Erasure is done by exposure of the memory array chip to high-intensity ultraviolet radiation through the UV window on top of the package. The positive charge stored on the gate is neutralized after several minutes to an hour of exposure time.

SECTION 11–4 CHECKUP

1. How do PROMs differ from ROMs?
2. What represents a data bit in an EPROM?
3. What is the normal mode of operation for a PROM?

11–5 The Flash Memory

The ideal memory has high storage capacity, nonvolatility, in-system read and write capability, comparatively fast operation, and cost effectiveness. The traditional memory technologies such as ROM, PROM, EPROM, EEPROM, SRAM, and DRAM individually exhibit one or more of these characteristics. Flash memory has all of the desired characteristics.

After completing this section, you should be able to

- Discuss the basic characteristics of a flash memory
- Describe the basic operation of a flash memory cell
- Compare flash memories with other types of memories
- Discuss the USB flash drive

Flash memories are high-density read/write memories (high-density translates into large bit storage capacity) that are nonvolatile, which means that data can be stored indefinitely without power. High-density means that a large number of cells can be packed into a given surface area on a chip; that is, the higher the density, the more bits that can be stored on a given size chip. This high density is achieved in flash memories with a storage cell that consists of a single floating-gate MOS transistor. A data bit is stored as charge or the absence of charge on the floating gate depending if a 0 or a 1 is stored.

Flash Memory Cell

A single-transistor cell in a flash memory is represented in Figure 11–33. The stacked gate MOS transistor consists of a control gate and a floating gate in addition to the drain and source. The floating gate stores electrons (charge) as a result of a sufficient voltage applied to the control gate. A *0 is stored when there is more charge* and a *1 is stored when there is less or no charge.* The amount of charge present on the floating gate determines if the transistor will turn on and conduct current from the drain to the source when a control voltage is applied during a read operation.

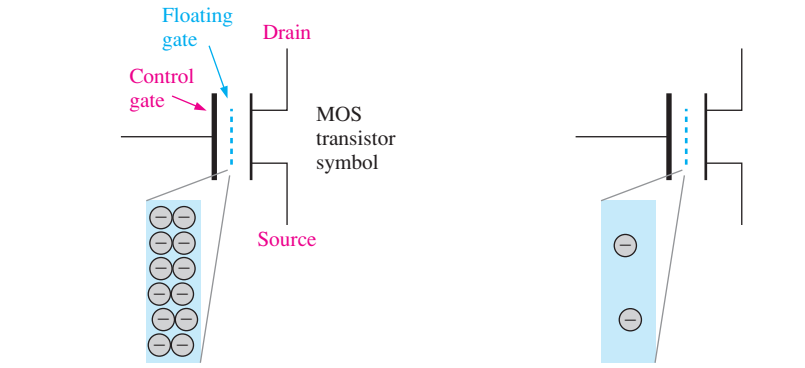


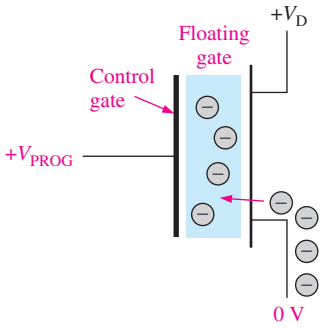
FIGURE 11–33 The storage cell in a flash memory.

Basic Flash Memory Operation

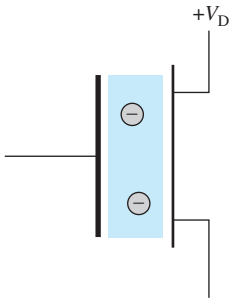
There are three major operations in a flash memory: the *programming* operation, the *read* operation, and the *erase* operation.

Programming

Initially, all cells are at the 1 state because charge was removed from each cell in a previous erase operation. The programming operation adds electrons (charge) to the floating gate of those cells that are to store a 0. No charge is added to those cells that are to store a 1. Application of a sufficient positive voltage to the control gate with respect to the source during programming attracts electrons to the floating gate, as indicated in Figure 11–34. Once programmed, a cell can retain the charge for up to 100 years without any external power.



To store a 0, a sufficient positive voltage is applied to the control gate with respect to the source to add charge to the floating gate during programming.



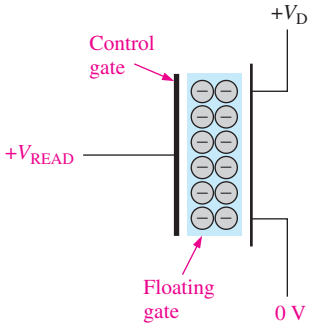
To store a 1, no charge is added and the cell is left in the erased condition.

FIGURE 11–34 Simplified illustration of storing a 0 or a 1 in a flash cell during the programming operation.

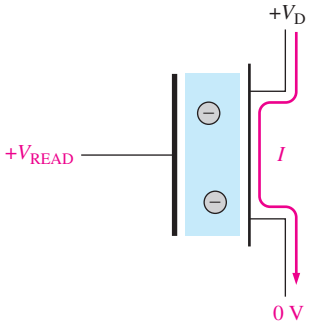
Read

During a read operation, a positive voltage is applied to the control gate. The amount of charge present on the floating gate of a cell determines whether or not the voltage applied to the control gate will turn on the transistor. If a 1 is stored, the control gate voltage is sufficient to turn the transistor on. If a 0 is stored, the transistor will not turn on because the control gate voltage is not sufficient to overcome the negative charge stored in the floating gate. Think of the charge on the floating gate as a voltage source that opposes the voltage applied to the control gate during a read operation. So the floating gate charge associated with a stored 0 prevents the control gate voltage from reaching the turn-on threshold, whereas the small or zero charge associated with a stored 1 allows the control gate voltage to exceed the turn-on threshold.

When the transistor turns on, there is current from the drain to the source of the cell transistor. The presence of this current is sensed to indicate a 1, and the absence of this current is sensed to indicate a 0. This basic idea is illustrated in Figure 11–35.



When a 0 is read, the transistor remains off because the charge on the floating gate prevents the read voltage from exceeding the turn-on threshold.

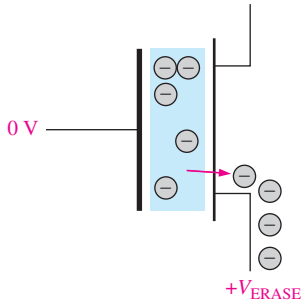


When a 1 is read, the transistor turns on because the absence of charge on the floating gate allows the read voltage to exceed the turn-on threshold.

FIGURE 11–35 The read operation of a flash cell in an array.

Erase

During an erase operation, charge is removed from all the memory cells. A sufficient positive voltage is applied to the transistor source with respect to the control gate. This is opposite in polarity to that used in programming. This voltage attracts electrons from the floating gate and depletes it of charge, as illustrated in Figure 11–36. A flash memory is always erased prior to being reprogrammed.



To erase a cell, a sufficient positive voltage is applied to the source with respect to the control gate to remove charge from the floating gate during the erase operation.

FIGURE 11–36 Simplified illustration of removing charge from a cell during erase.

Flash Memory Array

A simplified array of flash memory cells is shown in Figure 11–37. Only one row line is accessed at a time. When a cell in a given bit line turns on (stored 1) during a read operation, there is current through the bit line, which produces a voltage drop across the active load. This voltage drop is compared to a reference voltage with a comparator circuit and an output level indicating a 1 is produced. If a 0 is stored, then there is no current or little current in the bit line and an opposite level is produced on the comparator output.

The memory stick is a storage medium that uses flash memory technology in a physical configuration smaller than a stick of chewing gum. Memory sticks are typically available up to 64 GB capacities and as a kit with a PC card adaptor. Because of its compact design, it is ideal for use in small digital electronics products, such as laptop computers and digital cameras.

Comparison of Flash Memories with Other Memories

Let's compare flash memories with other types of memories with which you are already familiar.

Flash vs. ROM, EPROM, and EEPROM

Read-only memories are high-density, nonvolatile devices. However, once programmed the contents of a ROM can never be altered. Also, the initial programming is a time-consuming and costly process. The EEPROM has a more complex cell structure than either the ROM or UV EPROM and so the density is not as high, although it can be reprogrammed without being removed from the system. Because of its lower density, the cost/bit is higher than ROMs or EPROMs. Although the UV EPROM is a high-density, nonvolatile memory, it can be erased only by removing it from the system and using ultraviolet light. It can be reprogrammed only with specialized equipment.

A flash memory can be reprogrammed easily in the system because it is essentially a READ/WRITE device. The density of a flash memory compares with the ROM and EPROM because both have single-transistor cells. A flash memory (like a ROM, EPROM, or EEPROM) is nonvolatile, which allows data to be stored indefinitely with power off.

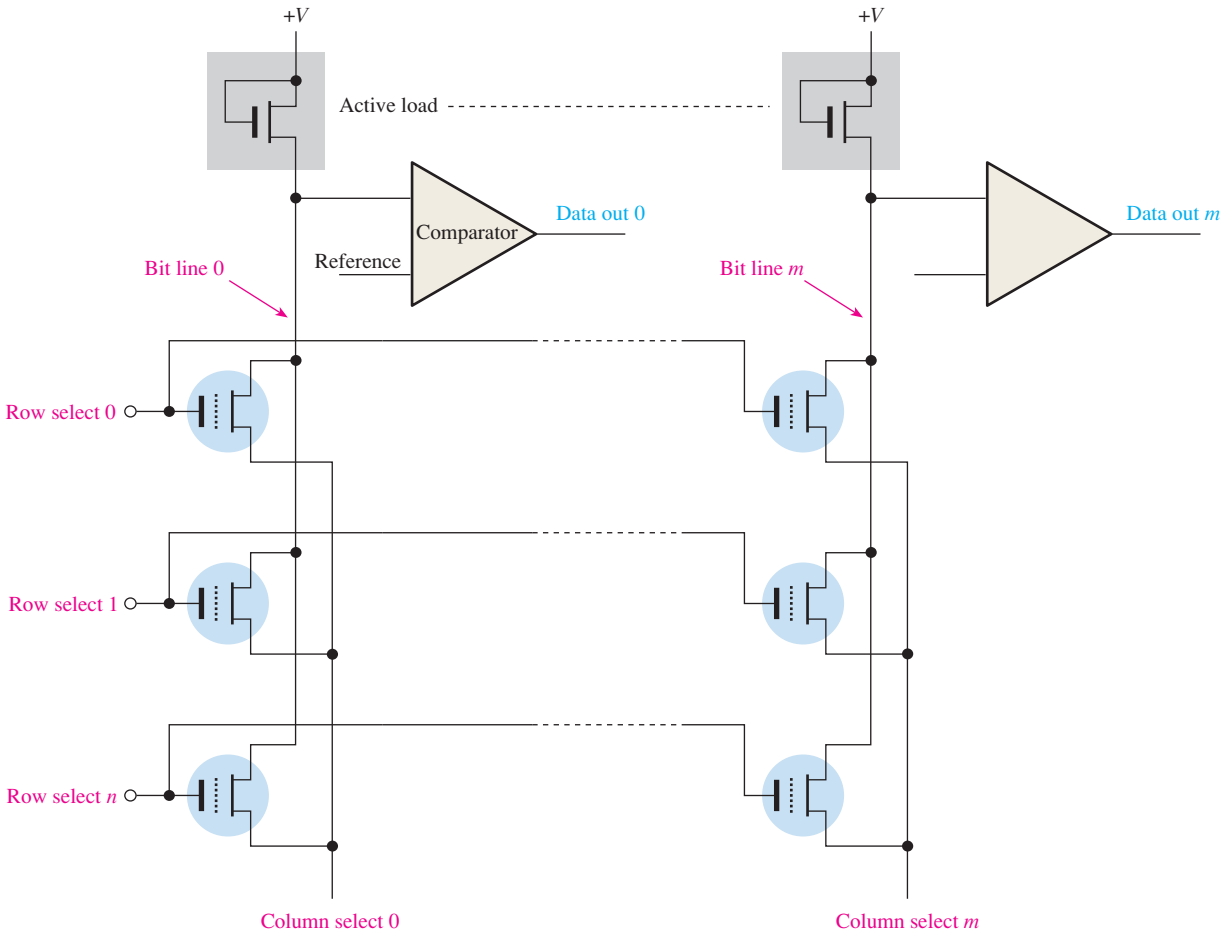


FIGURE 11–37 Basic flash memory array.

Flash vs. SRAM

As you have learned, static random-access memories are volatile READ/WRITE devices. A SRAM requires constant power to retain the stored data. In many applications, a battery backup is used to prevent data loss if the main power source is turned off. However, since battery failure is always a possibility, indefinite retention of the stored data in a SRAM cannot be guaranteed. Because the memory cell in a SRAM is basically a flip-flop consisting of several transistors, the density is relatively low.

A flash memory is also a READ/WRITE memory, but unlike the SRAM it is nonvolatile. Also, a flash memory has a much higher density than a SRAM.

Flash vs. DRAM

Dynamic random-access memories are volatile high-density READ/WRITE devices. DRAMs require not only constant power to retain data but also that the stored data must be refreshed frequently. In many applications, backup storage such as hard disk must be used with a DRAM.

Flash memories exhibit higher densities than DRAMs because a flash memory cell consists of one transistor and does not need refreshing, whereas a DRAM cell is one transistor plus a capacitor that has to be refreshed. Typically, a flash memory consumes much less power than an equivalent DRAM and can be used as a hard disk replacement in many applications.

Table 11–2 provides a comparison of the memory technologies.

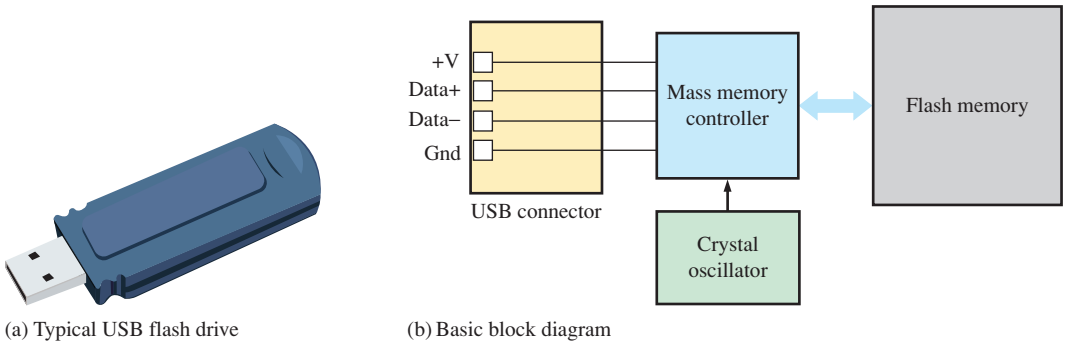
TABLE 11–2

Comparison of types of memories.

Memory Type	Nonvolatile	High-Density	One-Transistor Cell	In-System Writability
Flash	Yes	Yes	Yes	Yes
SRAM	No	No	No	Yes
DRAM	No	Yes	Yes	Yes
ROM	Yes	Yes	Yes	No
EEPROM	Yes	No	No	Yes
UV EPROM	Yes	Yes	Yes	No

USB Flash Drive

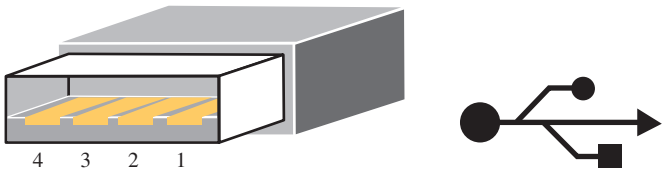
A USB flash drive consists of a flash memory connected to a standard USB connector housed in a small case about the size of a cigarette lighter. The USB connector can be plugged into a port on a personal computer and obtains power from the computer. These memories are usually rewritable and can have a storage capacity up to 512 GB (a number which is constantly increasing), with most ranging from 2 GB to 64 GB. A typical USB flash drive is shown in Figure 11–38(a), and a basic block diagram is shown in part (b).



(a) Typical USB flash drive (b) Basic block diagram

FIGURE 11–38 The USB flash drive.

The USB flash drive uses a standard USB A-type connector for connection to the computer, as shown in Figure 11–39(a). Peripherals such as printers use the USB B-type connector, which has a different shape and physical pin configuration. The USB icon is shown in part (b).



(a) Type A USB connector (b) USB icon

FIGURE 11–39 Connector and symbol.

SECTION 11–5 CHECKUP

1. What types of memories are nonvolatile?
2. What is a major advantage of a flash memory over a SRAM or DRAM?
3. List the three modes of operation of a flash memory.

11–6 Memory Expansion

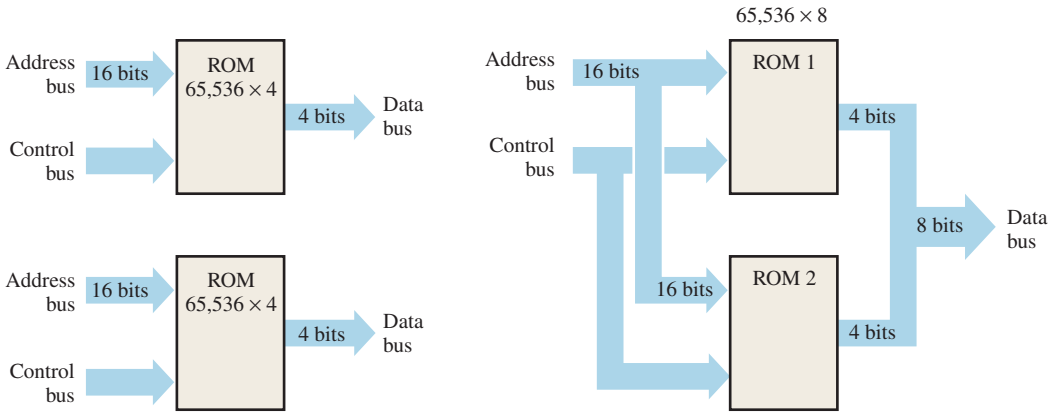
Available memory can be expanded to increase the word length (number of bits in each address) or the word capacity (number of different addresses) or both. Memory expansion is accomplished by adding an appropriate number of memory chips to the address, data, and control buses. SIMMs and DIMMs, which are types of memory expansion modules, are introduced.

After completing this section, you should be able to

- Define *word-length expansion*
- Show how to expand the word length of a memory
- Define *word-capacity expansion*
- Show how to expand the word capacity of a memory
- Discuss types of memory modules

Word-Length Expansion

To increase the **word length** of a memory, the number of bits in the data bus must be increased. For example, an 8-bit word length can be achieved by using two memories, each with 4-bit words as illustrated in Figure 11–40(a). As you can see in part (b), the 16-bit address bus is commonly connected to both memories so that the combination memory still has the same number of addresses ($2^{16} = 65{,}536$) as each individual memory. The 4-bit data buses from the two memories are combined to form an 8-bit data bus. Now when an address is selected, eight bits are produced on the data bus—four from each memory. Example 11–2 shows the details of 65,536 × 4 to 65,536 × 8 expansion.



(a) Two separate 65,536 × 4 ROMs

(b) One 65,536 × 8 ROM from two 65,536 × 4 ROMs

FIGURE 11–40 Expansion of two 65,536 × 4 ROMs into a 65,536 × 8 ROM to illustrate word-length expansion.

EXAMPLE 11–2

Expand the 65,536 × 4 ROM (64k × 4) in Figure 11–41 to form a 64k × 8 ROM. Note that "64k" is the accepted shorthand for 65,536. Why not "65k"? Maybe it's because 64 is also a power-of-two.

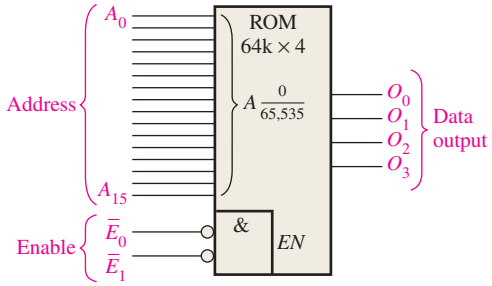


FIGURE 11–41 A 64k × 4 ROM.

Solution

Two 64k × 4 ROMs are connected as shown in Figure 11–42. Notice that a specific address is accessed in ROM 1 and ROM 2 at the same time. The four bits from a selected address in ROM 1 and the four bits from the corresponding address in ROM 2 go out in parallel to form an 8-bit word on the data bus. Also notice that a LOW on the enable line, $\overline{E}$, which forms a simple control bus, enables *both* memories.

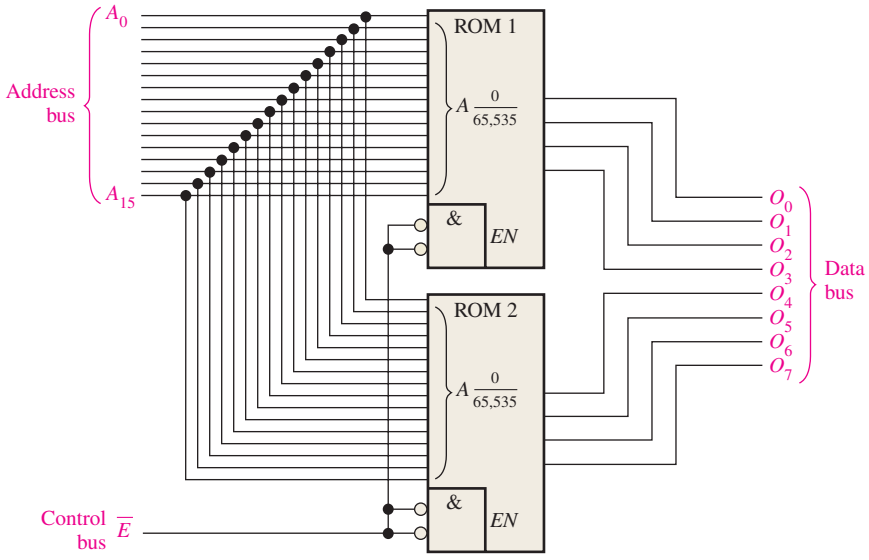


FIGURE 11–42

Related Problem

Describe how you would expand a 64k × 1 ROM to a 64k × 8 ROM.

EXAMPLE 11–3

Use the memories in Example 11–2 to form a 64k × 16 ROM.

Solution

In this case you need a memory that stores 65,536 16-bit words. Four 64k × 4 ROMs are required to do the job, as shown in Figure 11–43.

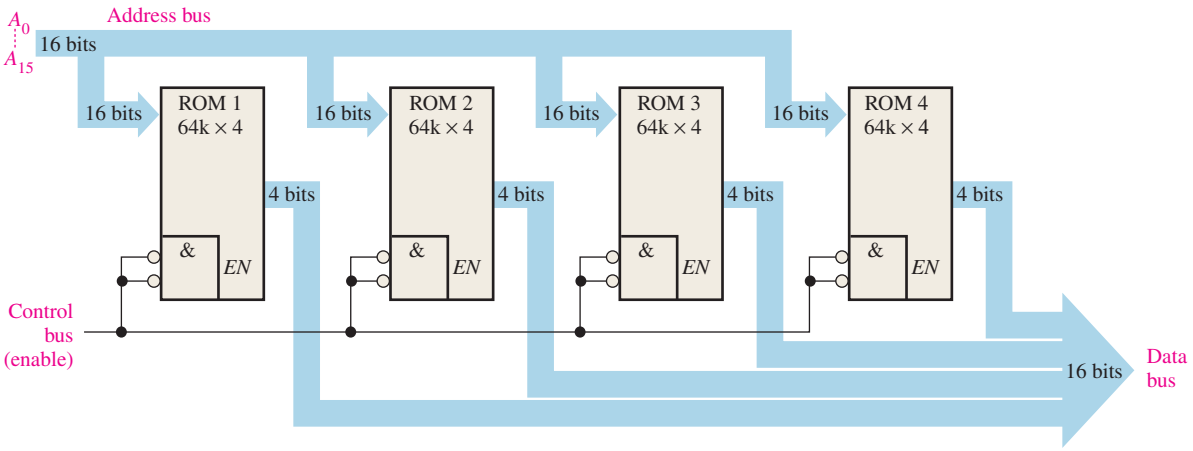


FIGURE 11–43

Related Problem

How many 64k × 1 ROMs would be required to implement the memory shown in Figure 11–43?

A ROM has only data outputs, but a RAM has both data inputs and data outputs. For word-length expansion in a RAM (SRAM or DRAM), the data inputs *and* data outputs form the data bus. Because the same lines are used for data input and data output, tri-state buffers are required. Most RAMs provide internal tri-state circuitry. Figure 11–44 illustrates RAM expansion to increase the data word length.

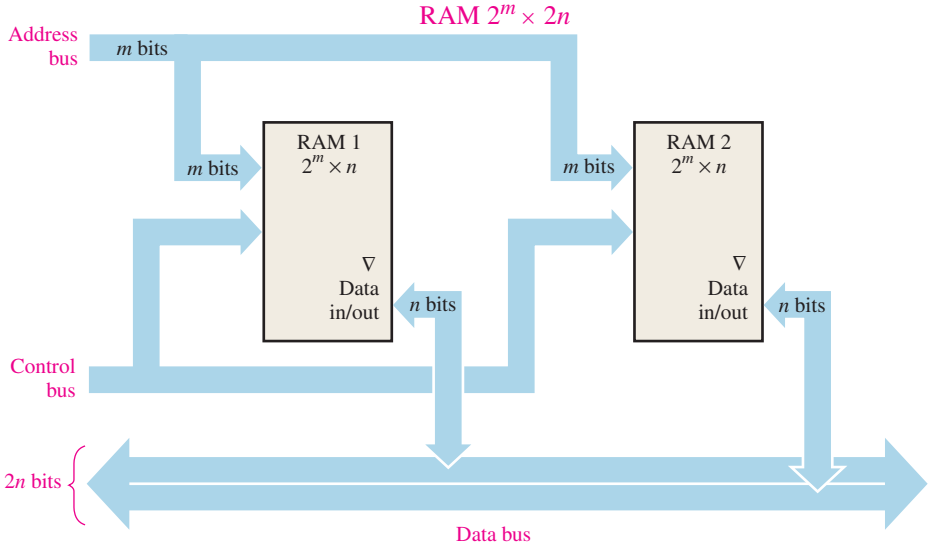


FIGURE 11–44 Illustration of word-length expansion with two $2^m \times n$ RAMs forming a $2^m \times 2n$ RAM.

EXAMPLE 11–4

Use 1M × 4 SRAMs to create a 1M × 8 SRAM.

Solution

Two 1M × 4 SRAMs are connected as shown in the simplified block diagram of Figure 11–45.

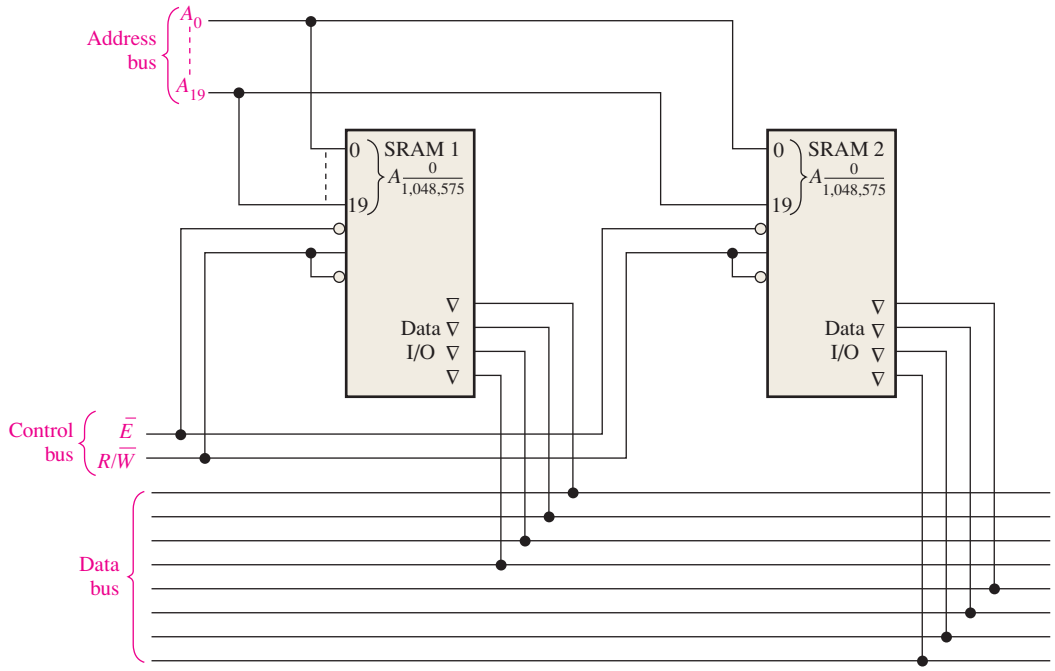


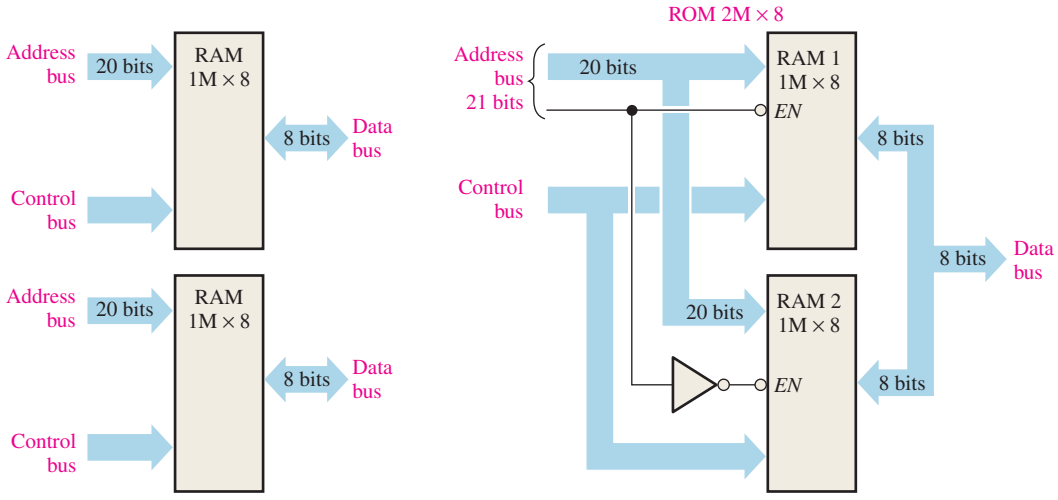
FIGURE 11–45

Related Problem

Use 1M × 8 SRAMs to create a 1M × 16 SRAM.

Word-Capacity Expansion

When memories are expanded to increase the **word capacity**, the *number of addresses is increased.* To achieve this increase, the number of address bits must be increased, as illustrated in Figure 11–46, (where two 1M × 8 RAMs are expanded to form a 2M × 8 memory).



(a) Individual memories each store 1,048,576 8-bit words

(b) Memories expanded to form a 2M × 8 RAM requiring a 21-bit address bus

FIGURE 11–46 Illustration of word-capacity expansion.

Each individual memory has 20 address bits to select its 1,048,576 addresses, as shown in part (a). The expanded memory has 2,097,152 addresses and therefore requires 21 address bits, as shown in part (b). The twenty-first address bit is used to enable the appropriate memory chip. The data bus for the expanded memory remains eight bits wide. Details of this expansion are illustrated in Example 11–5.

EXAMPLE 11–5

Use 512k × 4 RAMs to implement a 1M × 4 memory.

Solution

The expanded addressing is achieved by connecting the enable ($\overline{E}_0$) input to the twentieth address bit (A_{19}), as shown in Figure 11–47. Input $\overline{E}_1$ is used as an enable input common to both memories. When the twentieth address bit (A_{19}) is LOW, RAM 1 is selected (RAM 2 is disabled), and the nineteen lower-order address bits (A_0–A_{18}) access each of the addresses in RAM 1. When the twentieth address bit (A_{19}) is HIGH, RAM 2 is enabled by a LOW on the inverter output (RAM 1 is disabled), and the nineteen lower-order address bits (A_0 – A_{18}) access each of the RAM 2 addresses.

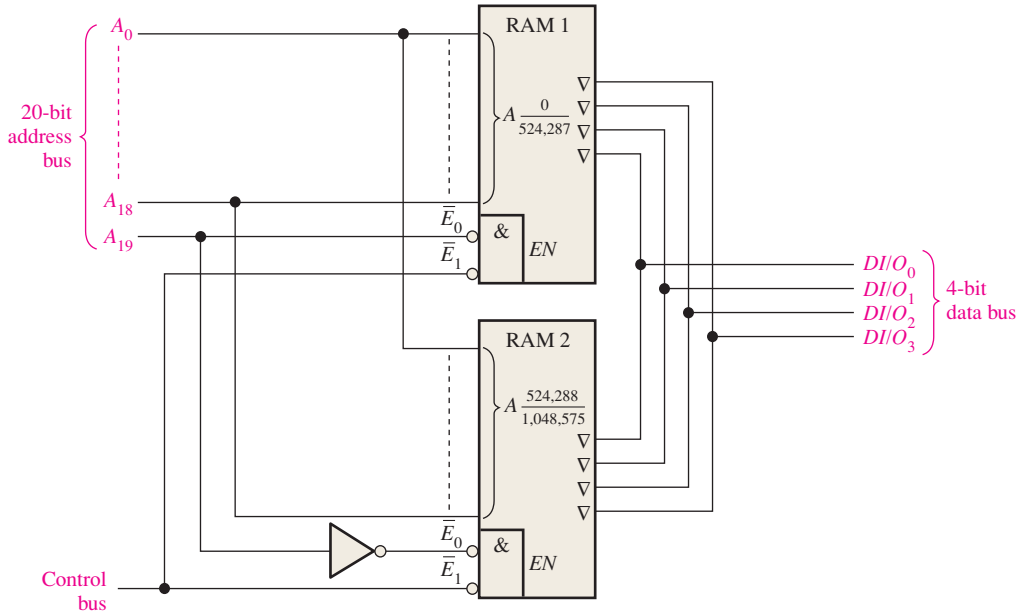


FIGURE 11–47

Related Problem

What are the ranges of addresses in RAM 1 and in RAM 2 in Figure 11–47?

Memory Modules

SDRAMs are available in modules consisting of multiple memory ICs arranged on a printed circuit board (PCB). The most common type of SDRAM memory module is called a **DIMM** (dual in-line memory module). Another version of the DIMM is the SODIMM (small-outline DIMM). A type of memory module, generally found in older equipment and essentially obsolete, is the **SIMM** (single in-line memory module). The SIMM has connection pins on one side of a PCB where the DIMM uses both sides of the board. DIMMs plug into a socket on the system mother board for memory expansion. A generic representation of a memory module is shown in Figure 11–48 with the system board connectors into which the modules are inserted.

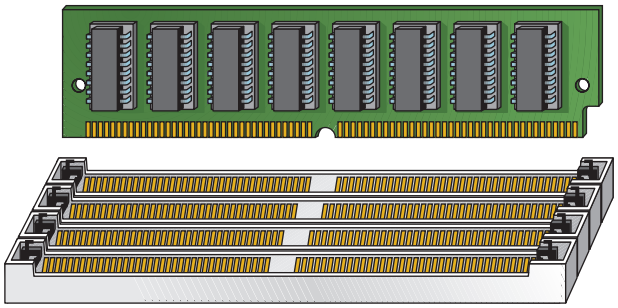


FIGURE 11–48 A memory module with connectors.

DIMMs generally contain DDR SDRAM memory chips. DDR means double data rate, so a DDR SDRAM transfers two blocks of data for each clock cycle rather than one like a standard SDRAM. Three basic types of modules are DDR, DDR2, and DDR3.

- DDR modules have 184 pins and require a 2.5 voltage source.
- DDR2 modules have 240 pins and require a 1.8 voltage source.
- DDR3 modules have 240 pins and require a 1.5 voltage source.

The DDR, DDR2, and DDR3 have transfer data rates of 1600 MB/s, 3200 MB/s, and 6400 MB/s respectively.

Memory components are extremely sensitive to static electricity. Use the following precautions when handling memory chips or modules such as DIMMs:

- Before handling, discharge your body's static charge by touching a grounded surface or wear a grounding wrist strap containing a high-value resistor if available. A convenient, reliable ground is the ac outlet ground.
- Do not remove components from their antistatic bags until you are ready to install them.
- Do not lay parts on the antistatic bags because only the inside is antistatic.
- When handling DIMMs, hold by the edges or the metal mounting bracket. Do not touch components on the boards or the edge connector pins.
- Never slide any part over any type of surface.
- Avoid plastic, vinyl, styrofoam, and nylon in the work area.

When installing DIMMs, follow these steps:

1. Line up the notches on the DIMM board with the notches in the memory socket.
2. Push firmly on the module until it is securely seated in the socket.
3. Generally, the latches on both sides of the socket will snap into place when the module is completely inserted. These latches also release the module, so it can be removed from the socket.

SECTION 11–6 CHECKUP

1. How many 16k × 1 RAMs are required to achieve a memory with a word capacity of 16k and a word length of eight bits?
2. To expand the 16k × 8 memory in question 1 to a 32k × 8 organization, how many more 16k × 1 RAMs are required?
3. What does DIMM stand for?

11–7 Special Types of Memories

In this section, the first in–first out (FIFO) memory, the last in–first out (LIFO) memory, the memory stack, and the charge-coupled device memory are covered.

After completing this section, you should be able to

- Describe a FIFO memory
- Describe a LIFO memory
- Discuss memory stacks
- Explain how to use a portion of RAM as a memory stack
- Describe a basic CCD memory

First In–First Out (FIFO) Memories

This type of memory is formed by an arrangement of shift registers. The term **FIFO** refers to the basic operation of this type of memory, in which the first data bit written into the memory is the first to be read out.

One important difference between a conventional shift register and a FIFO register is illustrated in Figure 11–49. In a conventional register, a data bit moves through the register only as new data bits are entered; in a FIFO register, a data bit immediately goes through the register to the right-most bit location that is empty.

Conventional Shift Register

Input	X	X	X	X	Output
0	0	*X*	*X*	*X*	→
1	1	0	*X*	*X*	→
1	1	1	0	*X*	→
0	0	1	1	1	→

X = unknown data bits.
In a conventional shift register, data stay to the left until "forced" through by additional data.

FIFO Shift Register

Input	—	—	—	—	Output
0	—	—	—	0	→
1	—	—	1	0	→
1	—	1	1	0	→
0	0	1	1	0	→

— = empty positions.
In a FIFO shift register, data "fall" through (go right).

FIGURE 11–49 Comparison of conventional and FIFO register operation.

Figure 11–50 is a block diagram of a FIFO serial memory. This particular memory has four serial 64-bit data registers and a 64-bit control register (marker register). When data are entered by a shift-in pulse, they move automatically under control of the marker register to the empty location closest to the output. Data cannot advance into occupied positions. However, when a data bit is shifted out by a shift-out pulse, the data bits remaining in the registers automatically move to the next position toward the output. In an asynchronous FIFO, data are shifted out independent of data entry, with the use of two separate clocks.

FIFO Applications

One important application area for the FIFO register is the case in which two systems of differing data rates must communicate. Data can be entered into a FIFO register at one rate and taken out at another rate. Figure 11–51 illustrates how a FIFO register might be used in these situations.

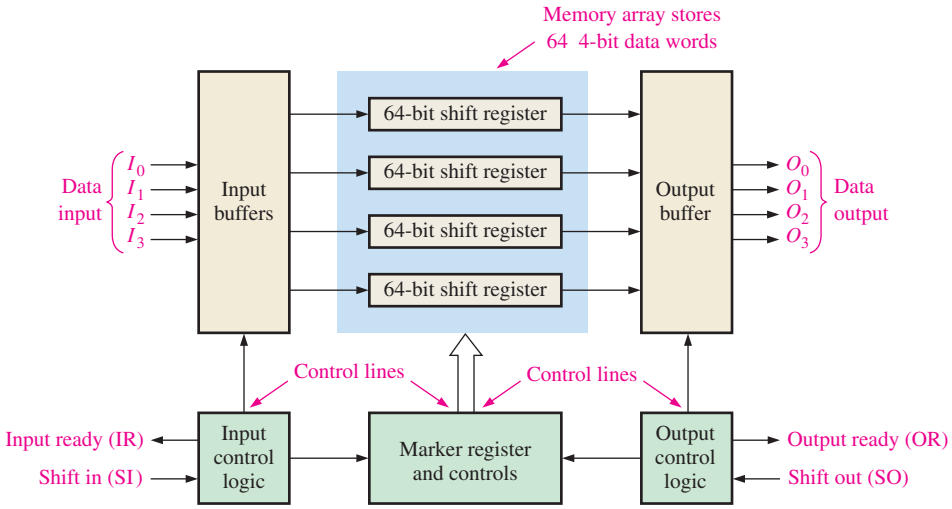


FIGURE 11–50 Block diagram of a typical FIFO serial memory.

(a) Irregular telemetry data can be stored and retransmitted at a constant rate.

(b) Data input at a slow keyboard rate can be stored and then transferred at a higher rate for processing.

(c) Data input at a constant rate can be stored and then output in bursts.

(d) Data in bursts can be stored and reformatted into a constant-rate output.

FIGURE 11–51 Examples of the FIFO register in data-rate buffering applications.

Last In–First Out (LIFO) Memories

The **LIFO** (last in–first out) memory is found in applications involving microprocessors and other computing systems. It allows data to be stored and then recalled in reverse order; that is, the last data byte to be stored is the first data byte to be retrieved.

Register Stacks

A LIFO memory is commonly referred to as a push-down stack. In some systems, it is implemented with a group of registers as shown in Figure 11–52. A stack can consist of any number of registers, but the register at the top is called the *top-of-stack.*

To illustrate the principle, a byte of data is loaded in parallel onto the top of the stack. Each successive byte pushes the previous one down into the next register. This process is illustrated in Figure 11–53. Notice that the new data byte is always loaded into the top register and the previously stored bytes are pushed deeper into the stack. The name *push-down stack* comes from this characteristic.

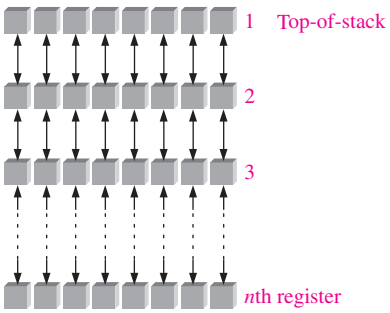


FIGURE 11–52 Register stack.

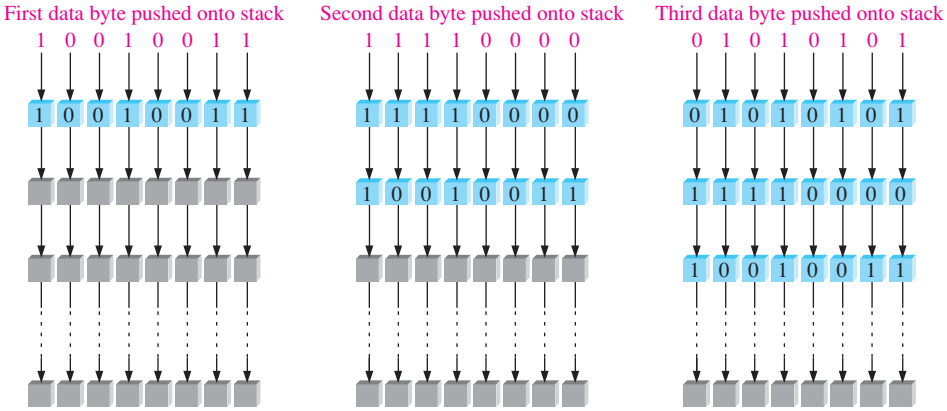


FIGURE 11–53 Simplified illustration of pushing data onto the stack.

Data bytes are retrieved in the reverse order. The last byte entered is always at the top of the stack, so when it is pulled from the stack, the other bytes pop up into the next higher locations. This process is illustrated in Figure 11–54.

RAM Stack

Another approach to LIFO memory used in microprocessor-based systems is the allocation of a section of RAM as the stack rather than the use of a dedicated set of registers. As you have seen, for a register stack the data move up or down from one location to the next. In

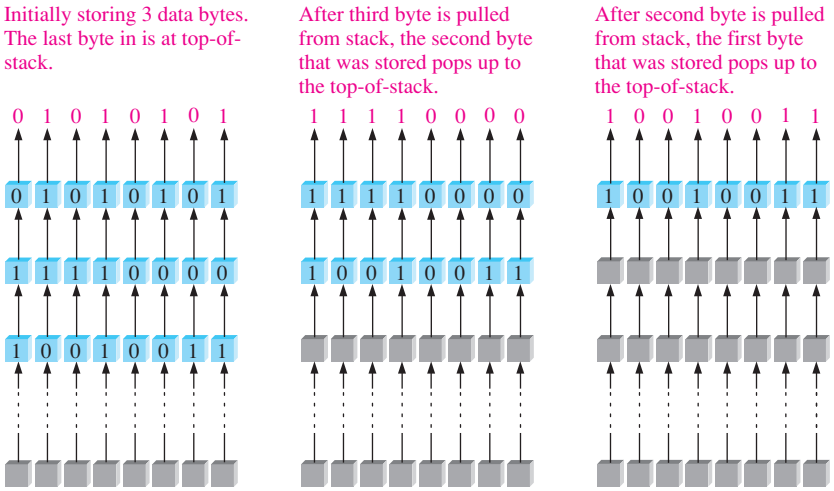


FIGURE 11–54 Simplified illustration of pulling data from the stack.

a RAM stack, the data do not move but the top-of-stack moves under control of a register called the stack pointer.

Consider a random-access memory that is byte organized—that is, one in which each address contains eight bits—as illustrated in Figure 11–55. The binary address 0000000000001111, for example, can be written as 000F in hexadecimal. A 16-bit address can have a *minimum* hexadecimal value of 0000_{16} and a *maximum* value of $FFFF_{16}$. With this notation, a 64 kB memory array can be represented as shown in Figure 11–55. The lowest memory address is 0000_{16} and the highest memory address is $FFFF_{16}$.

Now, consider a section of RAM set aside for use as a stack. A special separate register, the stack pointer, contains the address of the top of the stack, as illustrated in Figure 11–56. A 4-digit hexadecimal representation is used for the binary addresses. In the figure, the addresses are chosen for purposes of illustration.

Now let's see how data are pushed onto the stack. The stack pointer is initially at address $FFEE_{16}$, which is the top of the stack as shown in Figure 11–56(a). The stack pointer is then decremented (decreased) by two to $FFEC_{16}$. This moves the top of the stack to a lower memory address, as shown in Figure 11–56(b). Notice that the top of the stack is not stationary as in the fixed register stack but moves downward (to lower addresses) in the RAM as data words are stored. Figure 11–56(b) shows that two bytes (one data word) are then pushed onto the stack. After the data word is stored, the top of the stack is at $FFEC_{16}$.

Figure 11–57 illustrates the POP operation for the RAM stack. The last data word stored in the stack is read first. The stack pointer that is at FFEC is incremented (increased) by two to address $FFEE_{16}$ and a POP operation is performed as shown in part (b). Keep in mind that RAMs are nondestructive when read, so the data word still remains in the memory after a POP operation. A data word is destroyed only when a new word is written over it.

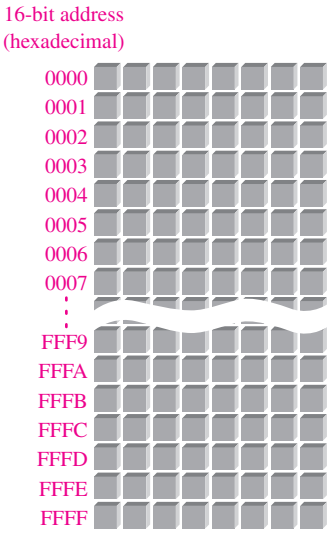
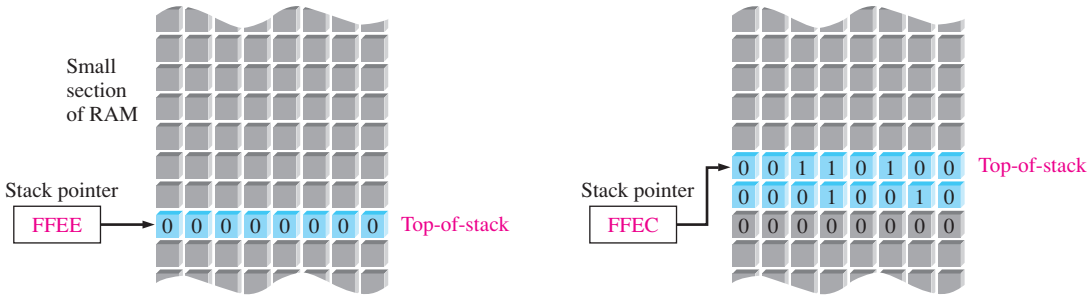


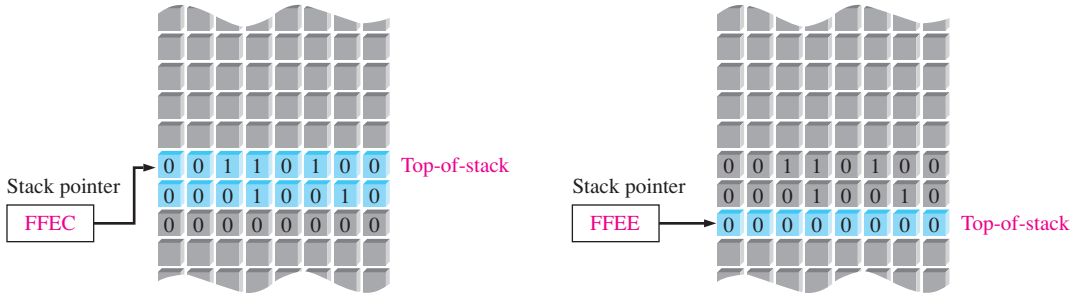
FIGURE 11–55 Representation of a 64 kB memory with the 16-bit addresses expressed in hexadecimal.



(a) The stack pointer is initially at FFEE before the data word 0001001000110100 (1234) is pushed onto the stack.

(b) The stack pointer is decremented by two and the data word 0001001000110100 is placed in the two locations prior to the original stack pointer location.

FIGURE 11–56 Illustration of the PUSH operation for a RAM stack.



(a) The stack pointer is at FFEC before the data word is copied (popped) from the stack.

(b) The stack pointer is incremented by two and the last data word stored is copied (popped) from the stack.

FIGURE 11–57 Illustration of the POP operation for the RAM stack.

A RAM stack can be of any depth, depending on the number of continuous memory addresses assigned for that purpose.

CCD Memories

The **CCD** (charge-coupled device) memory stores data as charges on capacitors and has the ability to convert optical images to electrical signals. Unlike the DRAM, however, the storage cell does not include a transistor. High density is the main advantage of CCDs, and these devices are widely used in digital imaging.

The CCD memory consists of long rows of semiconductor capacitors, called *channels.* Data are entered into a channel serially by depositing a small charge for a 0 and a large charge for a 1 on the capacitors. These charge packets are then shifted along the channel by clock signals as more data are entered.

As with the DRAM, the charges must be refreshed periodically. This process is done by shifting the charge packets serially through a refresh circuit. Figure 11–58 shows the basic concept of a CCD channel. Because data are shifted serially through the channels, the CCD memory has a relatively long access time. CCD arrays are used in many modern cameras to capture video images in the form of light-induced charge.

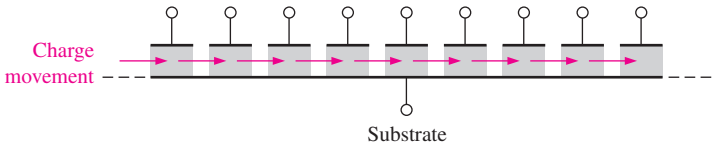


FIGURE 11–58 A CCD (charge-coupled device) channel.

SECTION 11–7 CHECKUP

1. What is a FIFO memory?
2. What is a LIFO memory?
3. Explain the PUSH operation in a memory stack.
4. Explain the POP operation in a memory stack.
5. What does the term *CCD* stand for?

11–8 Magnetic and Optical Storage

In this section, the basics of magnetic disks, magnetic tape, magneto-optical disks, and optical disks are introduced. These storage media are important, particularly in computer applications, where they are used for mass nonvolatile storage of data and programs.

After completing this section, you should be able to

- Describe a magnetic hard disk
- Discuss magnetic tape
- Discuss removable hard disks
- Explain the principle of magneto-optical disks
- Discuss the CD-ROM, CD-R, and CD-RW disks
- Describe the WORM
- Discuss the DVD-ROM

Magnetic Storage

Magnetic Hard Disks

Computers use hard disks as the internal mass storage media. **Hard disks** are rigid "platters" made of aluminum alloy or a mixture of glass and ceramic covered with a magnetic coating. Hard disk drives mainly come in three diameter sizes, 3.5 in., 2.5 in., and 1.8 in. Older formats of 8 in. and 5.25 in. are considered obsolete. A hard disk drive is hermetically sealed to keep the disks dust-free.

Typically, two or more disks are stacked on top of each other on a common shaft or spindle that turns the assembly at several thousand rpm. A separation between each disk allows for a magnetic read/write head that is mounted on the end of an actuator arm, as shown in Figure 11–59. There is a read/write head for both sides of each disk since data are recorded on both sides of the disk surface. The drive actuator arm synchronizes all the read/write heads to keep them in perfect alignment as they "fly" across the disk surface with a separation of only a fraction of a millimeter from the disk. A small dust particle could cause a head to "crash," causing damage to the disk surface.

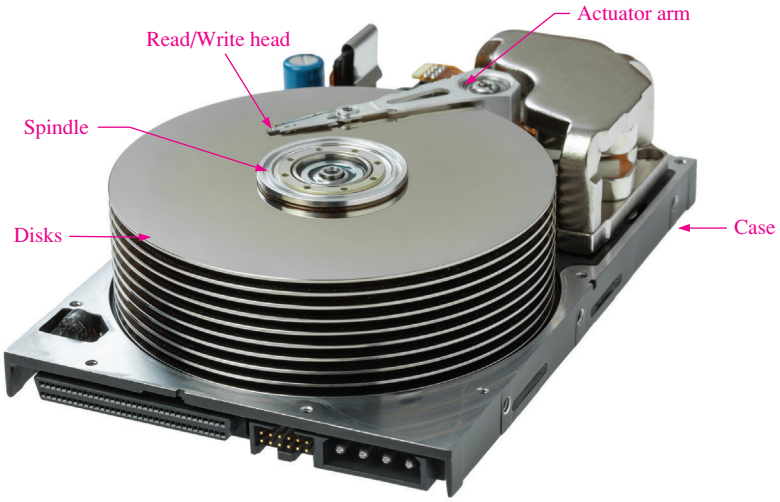


FIGURE 11–59 A hard disk drive. FrameAngel/Shutterstock

Basic Read/Write Head Principles

The hard drive is a random-access device because it can retrieve stored data anywhere on the disk in any order. A simplified diagram of the magnetic surface read/write operation is shown in Figure 11–60. The direction or polarization of the magnetic domains on the disk surface is controlled by the direction of the magnetic flux lines (magnetic field) produced

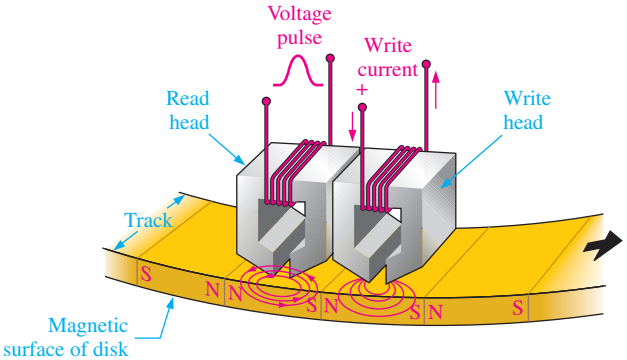


FIGURE 11–60 Simplified read/write head operation.

InfoNote

Data are stored on a hard drive in the form of files. Keeping track of the location of files is the job of the device driver that manages the hard drive (sometimes referred to as hard drive BIOS). The device driver and the computer's operating system can access two tables to keep track of files and file names. The first table is called the FAT (File Allocation Table). The FAT shows what is assigned to specific files and keeps a record of open sectors and bad sectors. The second table is the Root Directory which has file names, type of file, time and date of creation, starting cluster number, and other information about the file.

by the write head according to the direction of a current pulse in the winding. This magnetic flux magnetizes a small spot on the disk surface in the direction of the magnetic field. A magnetized spot of one polarity represents a binary 1, and one of the opposite polarity represents a binary 0. Once a spot on the disk surface is magnetized, it remains until written over with an opposite magnetic field.

When the magnetic surface passes a read head, the magnetized spots produce magnetic fields in the read head, which induce voltage pulses in the winding. The polarity of these pulses depends on the direction of the magnetized spot and indicates whether the stored bit is a 1 or a 0. The read and write heads are usually combined in a single unit.

Hard Disk Format

A hard disk is organized or formatted into tracks and sectors, as shown in Figure 11–61(a). Each track is divided into a number of sectors, and each track and sector has a physical address that is used by the operating system to locate a particular data record. Hard disks typically have from a few hundred to thousands of tracks and are available with storage capacities of up to 1 TB or more. As you can see in the figure, there is a constant number of tracks/sector, with outer sectors using more surface area than the inner sectors. The arrangement of tracks and sectors on a disk is known as the *format*.

A hard disk stack is illustrated in Figure 11–61(b). Hard disk drives differ in the number of disks in a stack, but there is always a minimum of two. All of the same corresponding tracks on each disk are collectively known as a cylinder, as indicated.

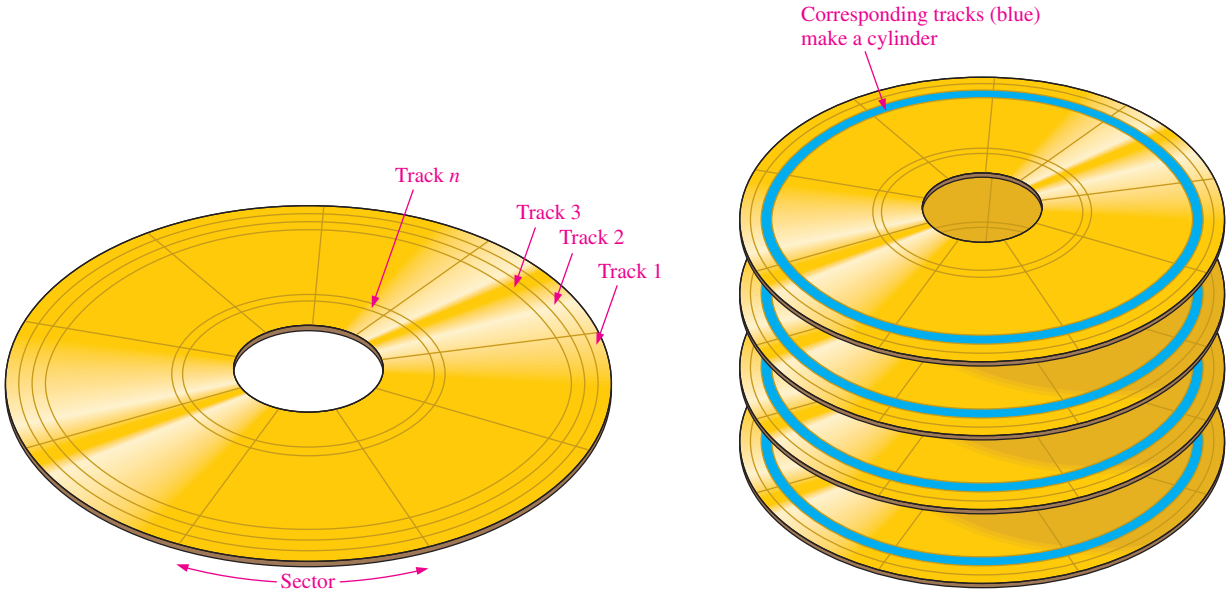


FIGURE 11–61 Hard disk organization and formatting.

Hard Disk Performance

Several basic parameters determine the performance of a given hard disk drive. A *seek* operation is the movement of the read/write head to the desired track. The **seek time** is the average time for this operation to be performed. Typically, hard disk drives have an average seek time of several milliseconds, depending on the particular drive.

The **latency period** is the time it takes for the desired sector to spin under the head once the head is positioned over the desired track. A worst case is when the desired sector is just past the head position and spinning away from it. The sector must rotate almost a full

revolution back to the head position. *Average latency period* assumes that the disk must make half of a revolution. Obviously, the latency period depends on the constant rotational speed of the disk. Disk rotation speeds are different for different disk drives but typically are from 4200 rpm to 15,000 rpm.

The sum of the average seek time and the average latency period is the *access time* for the disk drive.

Removable Hard Disk

A removable hard disk drive with a capacity of 1 TB is available. Keep in mind that the technology is changing so rapidly that there most likely will be further advancements at the time you are reading this.

Magnetic Tape

Tape is used for backup data from mass storage devices and typically is slower than disks because data on tape is accessed serially rather than randomly. There are several types that are available, including QIC, 8 mm, and DLT.

QIC is an abbreviation for quarter-inch cartridge and looks much like audio tape cassettes with two reels inside. Various QIC standards have from 28 to 108 tracks that can store from 80 MB to 1.6 GB. More recent innovations under the Travan standard have lengthened the tape and increased its width allowing storage capacities up to 10 GB. QIC tape drives use read/write heads that have a single write head with a read head on each side. This allows the tape drive to verify data just written when the tape is running in either direction. In the record mode, the tape moves past the read/write heads at approximately 100 inches/second, as indicated in Figure 11–62.

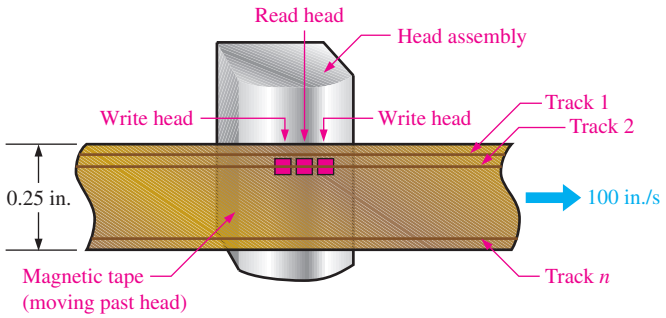


FIGURE 11–62 QIC tape.

8 mm tape was originally designed for the video industry but has been adopted by the computer industry as a reliable way to store large amounts of computer data.

DLT is an abbreviation for digital linear tape. DLT is a half-inch wide tape, which is 60% wider than 8 mm and, of course, twice as wide as standard QIC. Basically, DLT differs in the way the tape-drive mechanism works to minimize tape wear compared to other systems. DLT offers the highest storage capacity of all the tape formats with capacities ranging up to 800 GB.

Magneto-Optical Storage

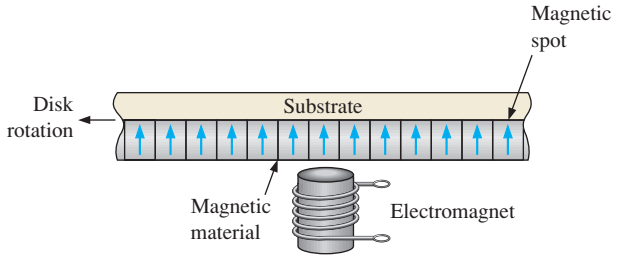
As the name implies, magneto-optical (MO) storage devices use a combination of magnetic and optical (laser) technologies. A **magneto-optical disk** is formatted into tracks and sectors similar to magnetic disks.

The basic difference between a purely magnetic disk and an MO disk is that the magnetic coating used on the MO disk requires heat to alter the magnetic polarization. Therefore, the MO is extremely stable at ambient temperature, making data unchangeable. To write a data bit, a high-power laser beam is focused on a tiny spot on the disk, and the

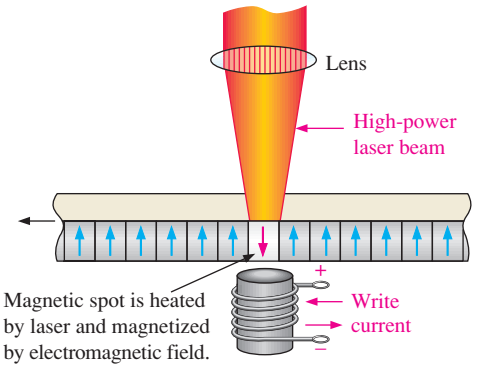
InfoNote

Tape is a viable alternative to disk due to its lower cost per bit. Though the density is lower than for a disk drive, the available surface on a tape is far greater. The highest-capacity tape media are generally on the same order as the largest available disk drive (about 1 TB—a terabyte is one trillion bytes.) Tape has historically offered enough advantage in cost over disk storage to make it a viable product, particularly for backup, where media removability is also important.

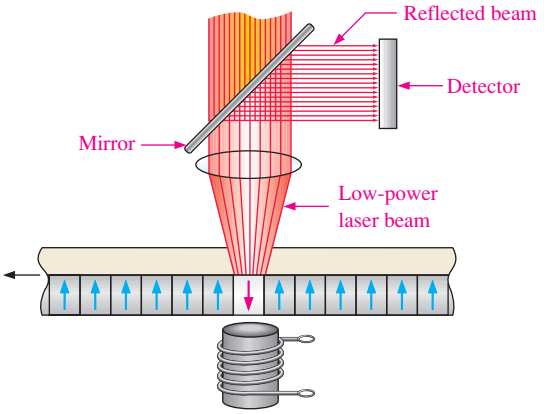
temperature of that tiny spot is raised above a temperature level called the Curie point (about 200°C). Once heated, the magnetic particles at that spot can easily have their direction (polarization) changed by a magnetic field generated by the write head. Information is read from the disk with a less-powerful laser than used for writing, making use of the Kerr effect where the polarity of the reflected laser light is altered depending on the orientation of the magnetic particles. Magnetic spots of one polarity represent 0s and magnetic spots of the opposite polarity represent 1s. Basic MO operation is shown in Figure 11–63, which represents a small cross-sectional area of a disk.



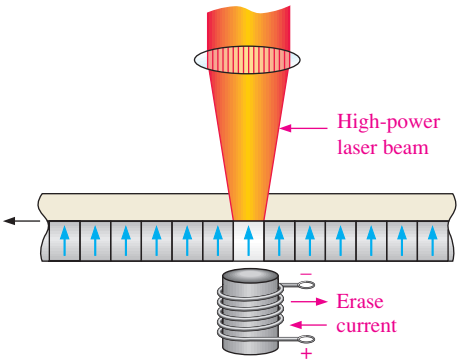
(a) Unrecorded disk



(b) Writing: A high-power laser beam heats the spot, causing the magnetic particles to align with the electromagnetic field.



(c) Reading: A low-power laser beam reflects off of the reversed-polarity magnetic particles and its polarization shifts. If the particles are not reversed, the polarization of the reflected beam is unchanged.



(d) Erasing: The electromagnetic field is reversed as the high-power laser beam heats the spot, causing the magnetic particles to be restored to the original polarity.

FIGURE 11–63 Basic principle of a magneto-optical disk.

Optical Storage

CD-ROM

The most common Compact Disk–Read-Only Memory is a 120 mm diameter disk with a sandwich of three coatings: a polycarbonate plastic on the bottom, a thin aluminum sheet for reflectivity, and a top coating of lacquer for protection. The **CD-ROM** disk is formatted in a single spiral track with sequential 2 kB sectors and has a capacity of 680 MB. Data are prerecorded at the factory in the form of minute indentations called *pits* and the flat area surrounding the pits called *lands*. The pits are stamped into the plastic layer and cannot be erased.

A CD player reads data from the spiral track with a low-power infrared laser, as illustrated in Figure 11–64. The data are in the form of pits and lands as shown. Laser light

reflected from a pit is 180° out-of-phase with the light reflected from the lands. As the disk rotates, the narrow laser beam strikes the series of pits and lands of varying lengths, and a photodiode detects the difference in the reflected light. The result is a series of 1s and 0s corresponding to the configuration of pits and lands along the track.

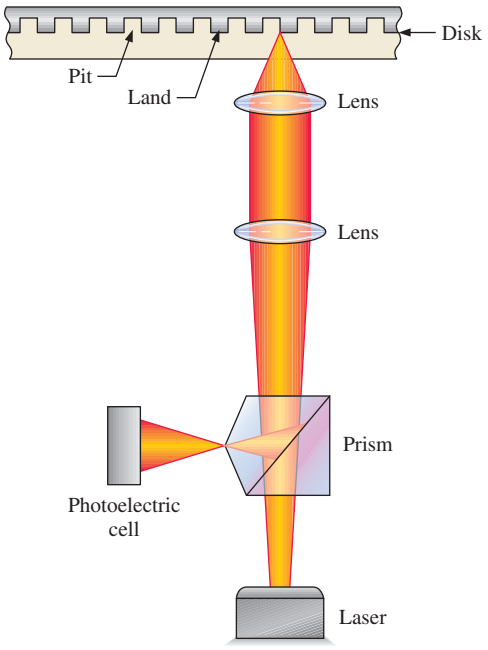


FIGURE 11–64 Basic operation of reading data from a CD-ROM.

WORM

Write Once/Read Many (**WORM**) is a type of optical storage that can be written onto one time after which the data cannot be erased but can be read many times. To write data, a low-power laser is used to burn microscopic pits on the disk surface. 1s and 0s are represented by the burned and nonburned areas.

CD-R

This is essentially a type of WORM. The difference is that the CD-Recordable allows multiple write sessions to different areas of the disk. The **CD-R** disk has a spiral track like the CD-ROM, but instead of mechanically pressing indentations on the disk to represent data, the CD-R uses a laser to burn microscopic spots into an organic dye surface. When heated beyond a critical temperature with a laser during read, the burned spots change color and reflect less light than the nonburned areas. Therefore, 1s and 0s are represented on a CD-R by burned and nonburned areas, whereas on a CD-ROM they are represented by pits and lands. Like the CD-ROM, the data cannot be erased once it is written.

CD-RW

The CD-Rewritable disk can be used to read and write data. Instead of the dye-based recording layer in the CD-R, the **CD-RW** commonly uses a crystalline compound with a special property. When it is heated to a certain temperature, it becomes crystalline when it cools; but if it is heated to a certain higher temperature, it melts and becomes amorphous when it cools. To write data, the focused laser beam heats the material to the melting temperature resulting in an amorphous state. The resulting amorphous areas reflect less light than the crystalline areas, allowing the read operation to detect 1s and 0s. The data can be erased or overwritten by heating the amorphous areas to a temperature above the crystallization

temperature but lower than the melting temperature that causes the amorphous material to revert to a crystalline state.

DVD-ROM

Originally DVD was an abbreviation for Digital Video Disk but eventually came to represent *Digital Versatile Disk.* Like the CD-ROM, **DVD-ROM** data are prestored on the disk. However, the pit size is smaller than for the CD-ROM, allowing more data to be stored on a track. The major difference between CD-ROM and DVD-ROM is that the CD is single-sided, while the DVD has data on both sides. Also, in addition to double-sided DVD disks, there are also multiple-layer disks that use semitransparent data layers placed over the main data layers, providing storage capacities of tens of gigabytes. To access all the layers, the laser beam requires refocusing going from one layer to the other.

Blu-Ray

The **Blu-ray** Disc (BD) is designed to eventually replace the DVD. The BD is the same size as DVDs and CDs. The name *Blu-ray* refers to the blue laser used to read the disc. DVDs use a red laser that has a longer wavelength. Information can be stored on a BD at a greater density and video definition than is possible with a DVD. The smaller Blu-ray laser beam can read recorded data in pits that are less than half the size of the pits on a DVD. A Blu-ray Disc can store about five times more data than a DVD. Typical storage capacities for conventional Blu-ray dual-layer discs are 50 GB, which is the industry standard for feature-length video. Triple layer and quadruple layer discs (BD-XL) can store 100 GB and 128 GB, respectively. Storage capacities up to 1 TB are currently under development.

SECTION 11–8 CHECKUP

1. List the major types of magnetic storage.
2. Generally, how is a magnetic disk organized?
3. How are data written on and read from a magneto-optical disk?
4. List the types of optical storage.

11–9 Memory Hierarchy

A memory system performs the data storage function in a computer. The memory system holds data temporarily during processing and also stores data and programs on a long-term basis. A computer has several types of memory, such as registers, cache, main, and hard disk. Other types of storage can also be used, such as magnetic tape, optical disk, and magnetic disk. Memory hierarchy as well as the system processor determines the processing speed of a computer.

After completing this section, you should be able to

- Discuss several types of memory
- Define memory hierarchy
- Describe key elements in a memory hierarchy

Three key characteristics of memory are cost, capacity, and access time. Memory cost is usually specified in cost per bit. The capacity of a memory is measured in the amount of data (bits or bytes) it can store. The access time is the time it takes to acquire a specified unit of data from the memory. The greater the capacity, the smaller the cost and the greater the access time. The smaller the access time, the greater the cost. The goal of using

a memory hierarchy is to obtain the shortest possible average access time while minimizing the cost.

The speed with which data can be processed depends both on the processor speed and on the time it takes to access stored data. **Memory hierarchy** is the arrangement of various memory elements within the computer architecture to maximize processing speed and minimize cost. Memory can be classified according to its "distance" from the processor in terms of the number of machine cycles or access time required to get data for processing. Distance is measured in time, not in physical location. Faster memory elements are considered closer to the processor compared to slower types of memory elements. Also, the cost per bit is much greater for the memory close to the processor than for the memory that is further from the processor. Figure 11–65 illustrates the arrangement of elements in a typical memory hierarchy.

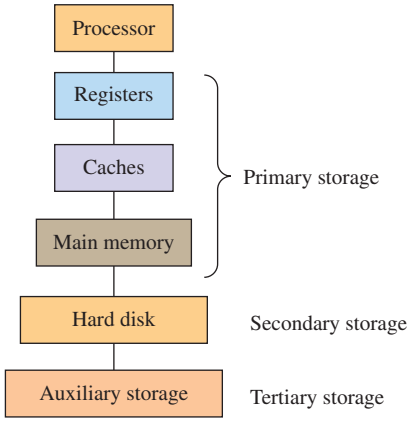


FIGURE 11–65 Typical memory hierarchy.

A primary distinction between the storage elements in Figure 11–65 is the time required for the processor to access data and programs. This access time is known as **memory latency**. The greater the latency, the further from the processor a storage element is considered to be. For example, typical register latency can be up to 1 or 2 ns, cache latency can be up to about 50 ns, main memory latency can be up to about 90 ns, and hard disk latency can be up to about 20 ms. Auxiliary memory latency can range up to several seconds.

Registers

Registers are memory elements that are located within the processor. They have a very small latency as well as a low capacity (number of bits that can be stored). One goal of programming is to keep as much frequently used data in the registers as possible. The number of registers in a processor can vary from the tens to hundreds.

Caches

The next level in the hierarchy is the memory cache, which provides temporary storage. The L1 cache is located in the processor, and the L2 cache is outside of the processor. A programming goal is to keep as much of a program as possible in the cache, especially the parts of a program that are most extensively used. There can be more than two caches in a memory system.

Main Memory

Main memory generally consists of two elements: RAM (random-access memory) and ROM (read-only memory). The RAM is a working memory that temporarily stores less

frequently used data and program instructions. The RAM is volatile, which means that the stored contents are lost when the power is turned off. The ROM is for permanent storage of frequently used programs and data; ROM is nonvolatile. Registers, caches, and main memory are considered primary storage.

Hard Disk

The hard disk has a very high latency and is used for mass storage of data and programs on a permanent basis. The hard disk is also used for virtual memory, space allocated for data when the primary memory fills up. In effect, virtual memory simulates primary memory with the disadvantage of high latency. Capacities range up to about 1 terabyte (TB).

$$1\text{ TB} = 1{,}000{,}000{,}000{,}000\text{ B} = 10^{12}\text{ B}$$

In addition to the internal hard disk, secondary storage can also include off-line storage. Off-line storage includes DVDs, CD-ROM, CD-RW, and USB flash drive. Off-line storage is removable storage.

Auxiliary Storage

Auxiliary storage, also called *tertiary storage,* includes magnetic tape libraries and optical jukeboxes. A **tape library** can store immense amounts of data (up to hundreds of petabytes). A petabyte (PB) is

$$1\text{ PB} = 1{,}000{,}000{,}000{,}000{,}000\text{ B} = 10^{15}\text{ B}$$

An **optical jukebox** is a robotic storage device that automatically loads and unloads optical disks. It may have as many as 2,000 slots for disks and can store hundreds of petabytes.

Relationship of Cost, Capacity, and Access Time

Figure 11–66 shows how capacity (the amount of data a memory can store) and cost per unit of storage varies as the distance from the processor, in terms of access time or latency, increases. The capacity increases and the cost decreases as access time increases.

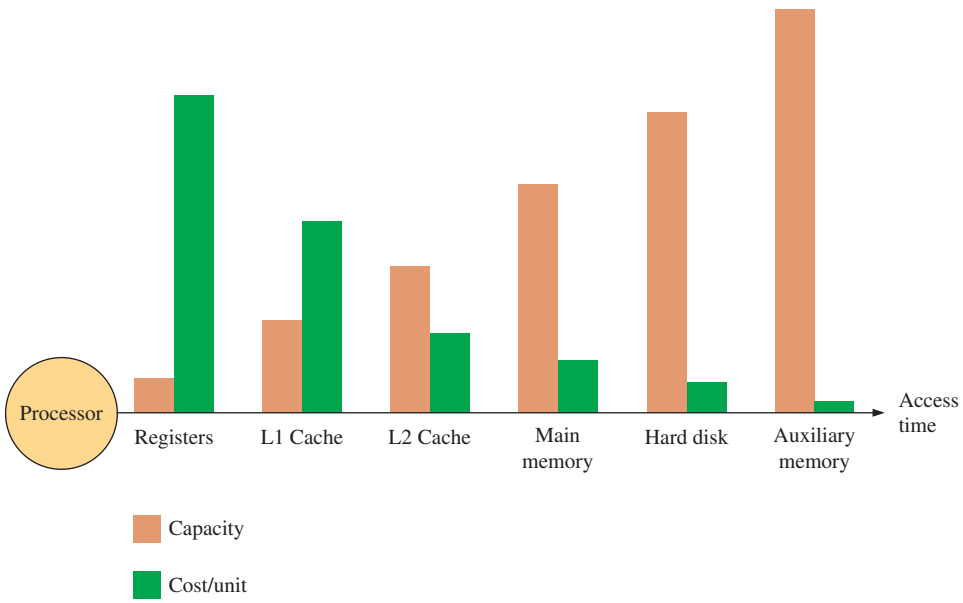


FIGURE 11–66 Changes in memory capacity and cost per unit of data as latency (access time) increases.

Memory Hierarchy Performance

In a computer system, the overall processing speed is usually limited by the memory, not the processor. Programming determines how well a particular memory hierarchy is utilized. The goal is to process data at the fastest rate possible. Two key factors in establishing maximum processor performance are locality and hit rate.

If a block of data is referenced, it will tend to be either referenced again soon or a nearby data block will be referenced soon. Frequent referencing of the same data block is known as *temporal locality*, and the program code should be arranged so that the piece of the data in the cache is reused frequently. Referencing an adjacent data block is known as *spatial locality*, and the program code should be arranged to use consecutive pieces of data on a frequent basis.

A **miss** is a failed attempt by the processor to read or write a block of data in a given level of memory (such as the cache). A miss causes the processor to have to go to a lower level of memory (such as main memory), which has a longer latency. The three types of misses are instruction read miss, data read miss, and data write miss. A successful attempt to read or write a block of data in a given level of memory is called a *hit*. Hits and misses are illustrated in Figure 11–67, where the processor is requesting data from the cache.

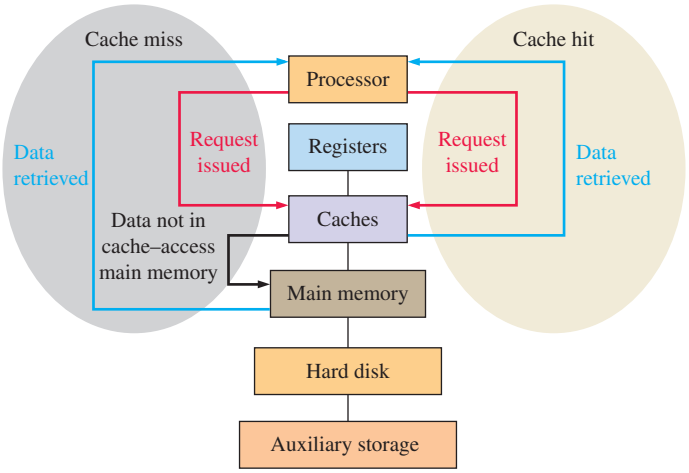


FIGURE 11–67 Illustration of a cache hit and a miss.

The **hit rate** is the percentage of memory accesses that find the requested data in the given level of memory. The *miss rate* is the percentage of memory accesses that fail to find the requested data in the given level of memory and is equal to 1– hit rate. The time required to access the requested information in a given level of memory is called the *hit time*. The higher the hit rate (hit to miss ratio), the more efficient the memory hierarchy is.

SECTION 11–9 CHECKUP

1. State the purpose of memory hierarchy.
2. What is access time?
3. How does memory capacity affect the cost per bit?
4. Does higher level memory generally have lower capacity than lower level memory?
5. What is a hit? A miss?
6. What determines the efficiency of the memory hierarchy?

11–10 Cloud Storage

Cloud storage is a system, usually maintained by a third party, for securely storing data in a remote location that can be conveniently accessed through the Internet. A file on a computer can be stored on secure remote servers and accessed by various user devices such as computers, smart phones, and tablets. Cloud storage eliminates the need for local backup storage such as external hard drives or CDs. When you use cloud storage, you are essentially storing your files or documents on Internet servers instead of or in addition to a computer. The term *cloud* may have originated from the use of a symbol that resembled a cloud on early network diagrams.

After completing this section, you should be able to

- Describe cloud storage
- Explain what a server is
- State the advantages of cloud storage
- Describe several properties of cloud storage

The Cloud Storage System

A **cloud storage** system consists of a remote network of servers (also called *nodes*) that are connected to a user device through the Internet, as shown in Figure 11–68. Some cloud storage systems accommodate only certain types of data such as e-mail or digital pictures, while others store all types of data and range in size from small operations with a few servers to very large operations that utilize hundreds of servers. A facility that houses cloud storage systems is called a **data center**. A typical storage cloud system can serve multiple users.

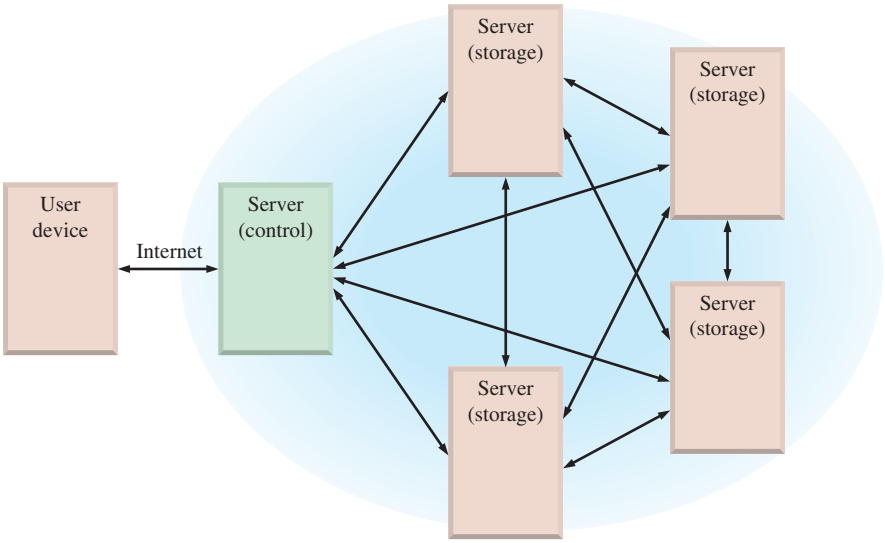


FIGURE 11–68 A typical cloud storage system architecture consists of a master control server and several storage servers that can be accessed by a user device over the Internet.

Servers typically operate within a client-server architecture, where the client is the user that is subscribing to the cloud storage. Theoretically, a **server** is any computerized process that shares a resource with one or more clients. More practically, a storage server is a computer and software with a large memory capacity that responds to requests across a network to provide file storage and access as well as services such as file sharing. The control server

(a) A typical rack of servers (b) A typical server room in a data center

FIGURE 11–69 Cloud servers. (a) Jojje/Shutterstock (b) Oleksiy Mark/Shutterstock

coordinates the activities within the storage cloud network among other servers and manages user access. A server rack and data center are shown in Figure 11–69.

At its simplest level, a cloud storage system needs just one storage server connected to the Internet. When copies of a file are sent by a client to the server over the Internet, the data are stored. When the client wishes to retrieve the data, the storage server (node) sends it back through a Web-based interface or allows the client to manipulate the file on the server itself.

Most cloud storage systems have many storage servers (hundreds in some cases) to provide both capacity and redundancy. A grouping of servers is sometimes called a *cluster*. Depending on the system architecture, a given system may have multiple clusters. A simple system with four storage servers illustrating file storage redundancy is shown in Figure 11–70. When a client sends data to the cloud, it is stored in multiple servers. This redundancy guarantees availability of data at any time to the client and makes the system highly reliable. Redundancy is necessary because a server requires periodic maintenance or may break down and need repairs. In addition to storage server redundancy, most cloud storage systems use power supply redundancy so that all servers are not operating from the same power source.

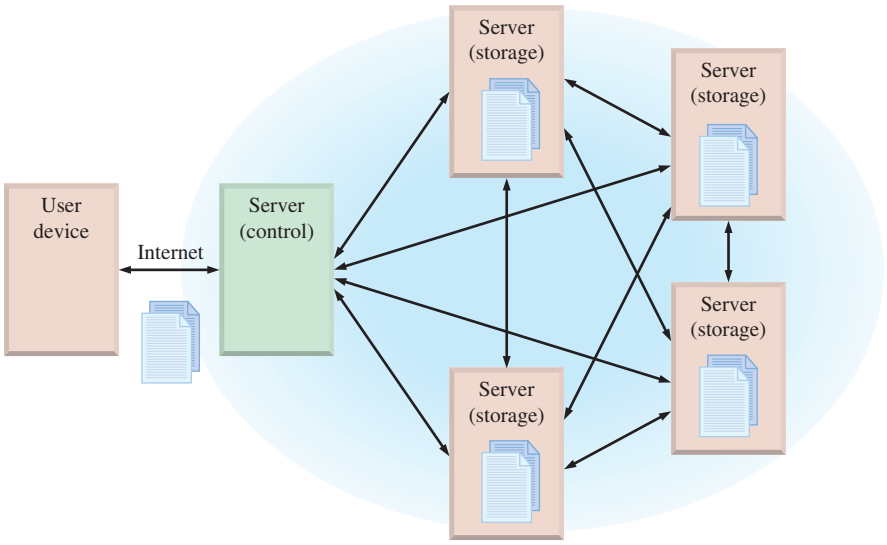


FIGURE 11–70 A simple cloud storage system with storage redundancy. In this case, the data are stored on four different servers.

In addition to reliability that provides assurance that a client's data are accurately stored and can be retrieved at any time, a second major factor for cloud storage is security that the data cannot be compromised. Generally, three methods are used to provide data security:

- *Encryption* or encoding, which prevents the data from being read or interpreted without proper decryption tools
- *Authentication*, which requires a name and password for access
- *Authorization*, which requires a list of only those people who can have access to the data

Cloud storage has certain advantages over traditional data storage in a computer. One advantage is that you can store and retrieve data from any physical location that has Internet access. A second advantage is that you don't have to use the same computer to store and retrieve data or carry a physical storage device for data backup around with you. Also, the user does not have to maintain the storage components. Another advantage of cloud storage is that other people can access your data (data sharing).

Architecture

The term *architecture* relates to how a cloud storage system is structured and organized. The primary purpose of cloud storage architecture is to deliver the service for data storage in a specific way. Architectures vary but generically most consist of a front end, a control, and a back end, as depicted in Figure 11–71.

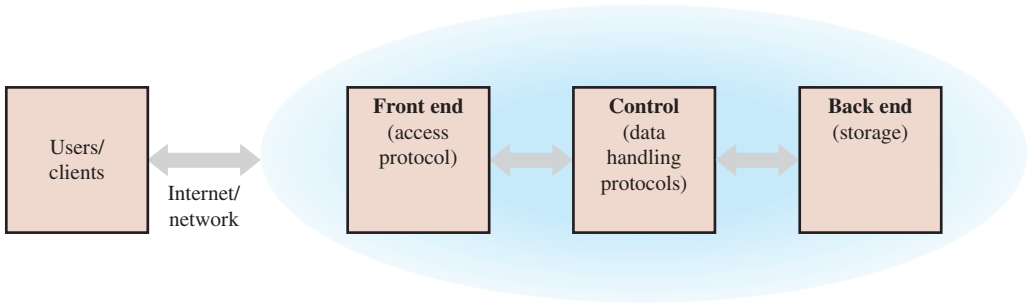


FIGURE 11–71 Generic architecture of a cloud storage system.

A cloud storage system uses various protocols within the architecture that determine how the data are accessed and handled. A **protocol** is a standardized set of software regulations, requirements, and procedures that control and regulate the transmission, processing, and exchange of data among devices. For example, common Internet protocols are HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), TCP/IP (Transfer Control Protocol/Internet Protocol), and SMTP (Simple Mail Transfer Protocol).

An API is an Application Programming Interface, which is essentially a protocol for access and utilization of a cloud storage system. There are many types of APIs. For example, a commonly used one is the REST API. REST stands for Representational State Transfer. An API is a software-to-software interface, not a user interface. With APIs, applications talk to each other "behind the scene" without user knowledge.

Cloud Storage Properties

The following cloud storage properties determine the performance of the system.

- *Latency.* The time between a request for data and the delivery of the data to the user is the **latency** of a system. Delay is due to the time for each component of the cloud storage system to respond to a request and to the time for data to be transferred to the user.

- *Bandwidth.* Bandwidth is a measure of the range of frequencies that can be simultaneously transferred to the cloud and is defined as a range of frequencies that can be handled by the system. Generally, the wider the bandwidth, the shorter the latency and vice versa.
- *Scalability.* The **scalability** property indicates the ability of a cloud storage system to handle increasing amounts of data in a smooth and easy manner; or it is the cloud's ability to improve movement of data through the system (throughput) when additional resources (typically hardware) are added. When the performance of a system improves proportionally to the storage capacity added, the system is said to be scalable. Scaling vertically (scale up) occurs when resources (hardware and memory) are added to a single server (node). Scaling horizontally (scale out) occurs when more servers (nodes) are added to a system.
- *Elasticity.* **Elasticity** is a cloud's ability to deal with variations in the amount of data (load) being transferred in and out of the storage system without service interrupts. There is a subtle difference between scalability and elasticity when describing a system's behavior. Essentially, *scalability* is a static parameter that indicates how much the system can be expanded, and *elasticity* is a dynamic parameter that refers to the implementation of scalability. For example, a storage system may be scalable from one to 100 servers. If the system is currently operating with 20 servers (nodes) and the data load doubles, its elasticity allows 20 more nodes to be added for a total of 40. Likewise, if the data load decreases by half, the elasticity allows 10 nodes to be removed. A server can be added or removed by powering it up or down in a proper manner without disrupting service to the user. Elasticity results in cost efficiency because only the number of servers required for the data load at any given time are consuming power.
- *Multitenancy.* The **multitenancy** property of a cloud storage system allows multiple users to share the same software applications and hardware and the same data storage mechanism but not to see each other's data.

SECTION 11–10 CHECKUP

1. What is a cloud storage system?
2. What is a server?
3. How does a user connect to a cloud storage system?
4. Name three advantages of a cloud system.

11–11 Troubleshooting

Because memories can contain large numbers of storage cells, testing each cell can be a lengthy and frustrating process. Fortunately, memory testing is usually an automated process performed with a programmable test instrument or with the aid of software for in-system testing. Most microprocessor-based systems provide automatic memory testing as part of their system software.

After completing this section, you should be able to

- Discuss the checksum method of testing ROMs
- Discuss the checkerboard pattern method of testing RAMs

ROM Testing

Since ROMs contain known data, they can be checked for the correctness of the stored data by reading each data word from the memory and comparing it with a data word that

is known to be correct. One way of doing this is illustrated in Figure 11–72. This process requires a reference ROM that contains the same data as the ROM to be tested. A special test instrument is programmed to read each address in both ROMs simultaneously and to compare the contents. A flowchart in Figure 11–73 illustrates the basic sequence.

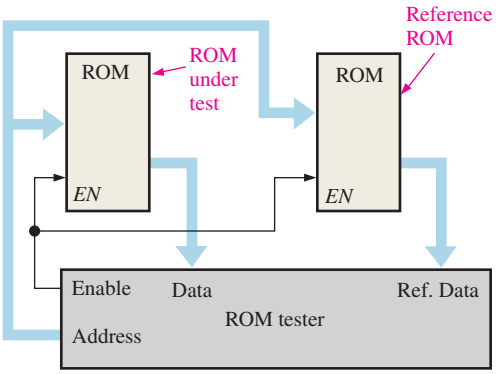
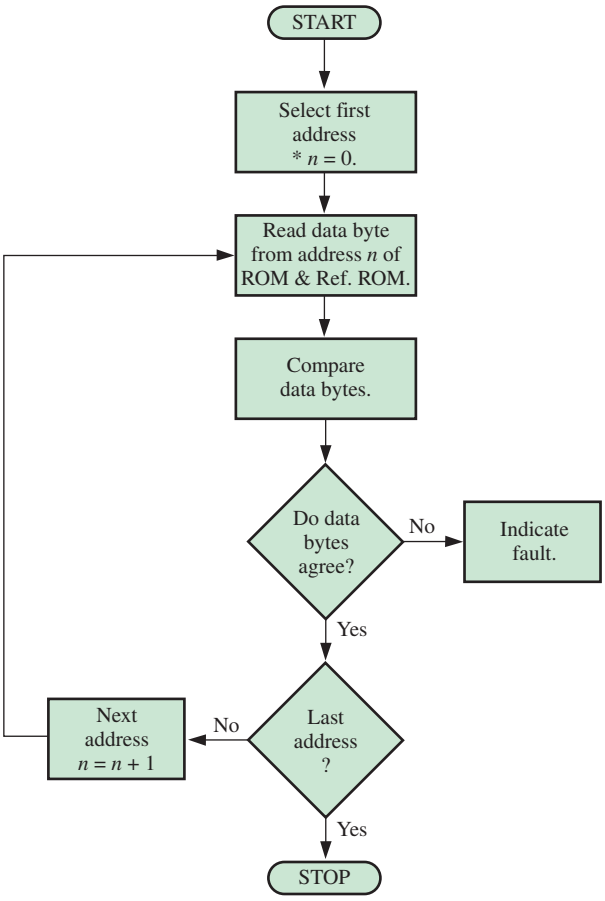


FIGURE 11–72 Block diagram for a complete contents check of a ROM.



* n is the address number.

FIGURE 11–73 Flowchart for a complete contents check of a ROM.

Checksum Method

Although the previous method checks each ROM address for correct data, it has the disadvantage of requiring a reference ROM for each different ROM to be tested. Also, a failure in the reference ROM can produce a fault indication.

In the checksum method a number, the sum of the contents of all the ROM addresses, is stored in a designated ROM address when the ROM is programmed. To test the ROM, the contents of all the addresses except the checksum are added, and the result is compared with the checksum stored in the ROM. If there is a difference, there is definitely a fault. If the checksums agree, the ROM is most likely good. However, there is a remote possibility that a combination of bad memory cells could cause the checksums to agree.

This process is illustrated in Figure 11–74 with a simple example. The checksum in this case is produced by taking the sum of each column of data bits and discarding the carries. This is actually an XOR sum of each column. The flowchart in Figure 11–75 illustrates the basic checksum test.

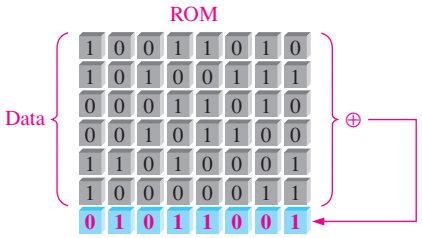


FIGURE 11–74 Simplified illustration of a programmed ROM with the checksum stored at a designated address.

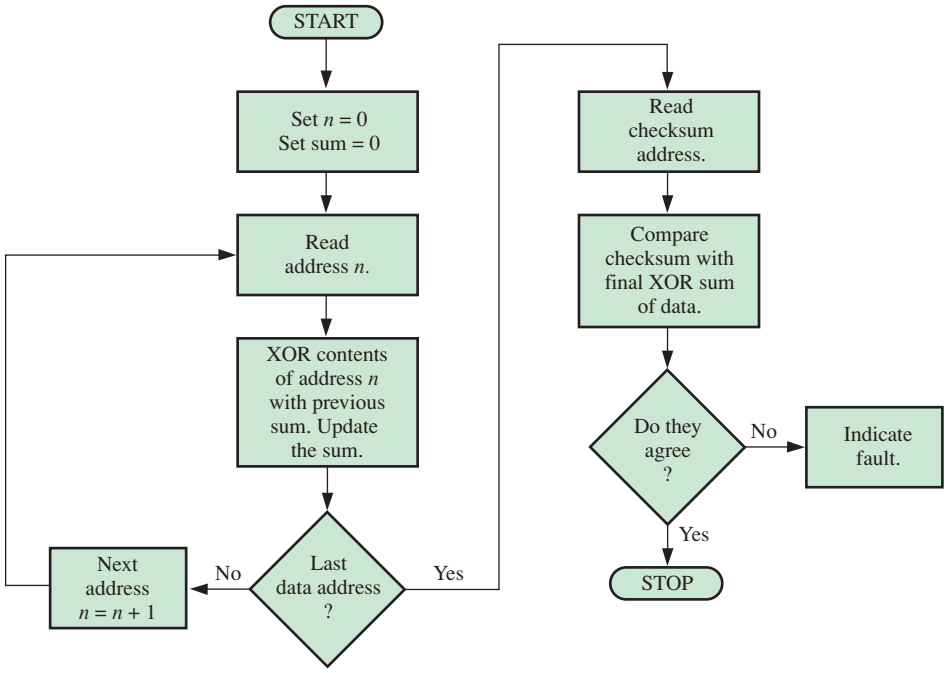


FIGURE 11–75 Flowchart for a basic checksum test.

The checksum test can be implemented with a special test instrument, or it can be incorporated as a test routine in the built-in (system) software or microprocessor-based systems. In that case, the ROM test routine is automatically run on system start-up.

RAM Testing

To test a RAM for its ability to store both 0s and 1s in each cell, first 0s are written into all the cells in each address and then read out and checked. Next, 1s are written into all the cells in each address and then read out and checked. This basic test will detect a cell that is stuck in either a 1 state or a 0 state.

Some memory faults cannot be detected with the all-0s–all-1s test. For example, if two adjacent memory cells are shorted, they will always be in the same state, both 0s or both 1s. Also, the all-0s–all-1s test is ineffective if there are internal noise problems such that the contents of one or more addresses are altered by a change in the contents of another address.

The Checkerboard Pattern Test

One way to more fully test a RAM is by using a checkerboard pattern of 1s and 0s, as illustrated in Figure 11–76. Notice that all adjacent cells have opposite bits. This pattern checks for a short between two adjacent cells; if there is a short, both cells will be in the same state.

After the RAM is checked with the pattern in Figure 11–76(a), the pattern is reversed, as shown in part (b). This reversal checks the ability of all cells to store both 1s and 0s.

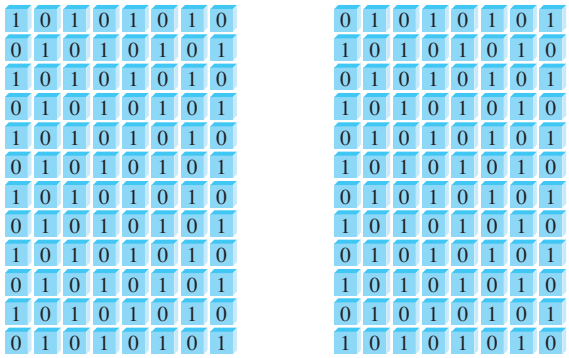


FIGURE 11–76 The RAM checkerboard test pattern.

A further test is to alternate the pattern one address at a time and check all the other addresses for the proper pattern. This test will catch a problem in which the contents of an address are dynamically altered when the contents of another address change.

A basic procedure for the checkerboard test is illustrated by the flowchart in Figure 11–77. The procedure can be implemented with the system software in microprocessor-based

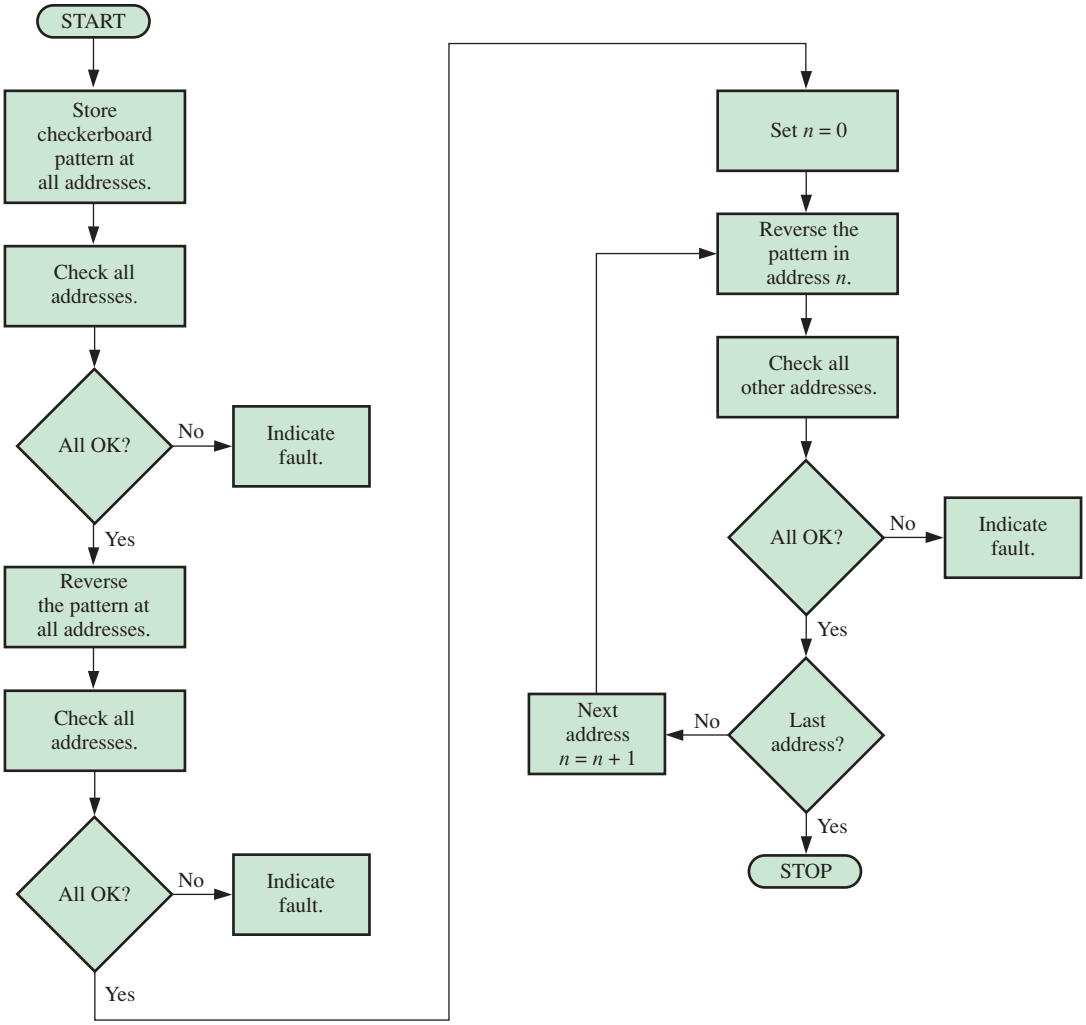


FIGURE 11–77 Flowchart for basic RAM checkerboard test.

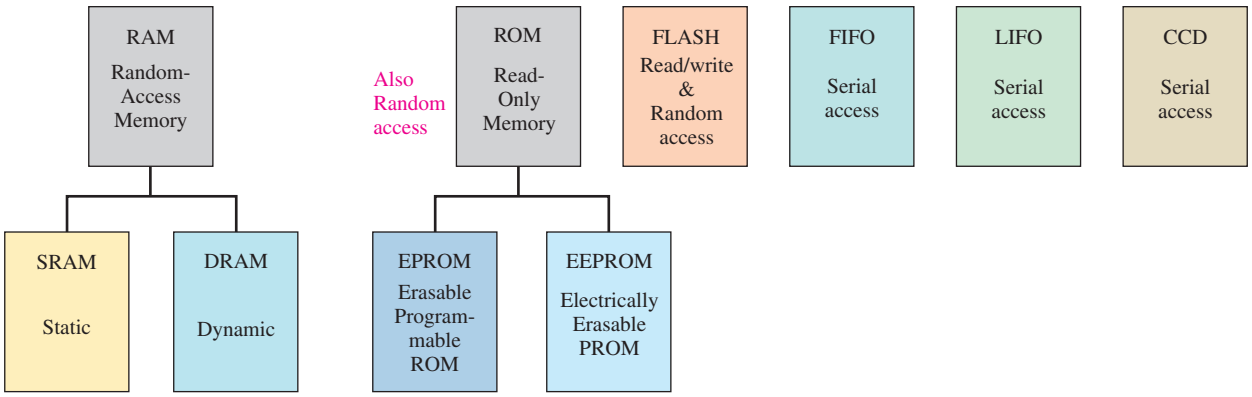
systems so that either the tests are automatic when the system is powered up or they can be initiated from the keyboard.

SECTION 11–11 CHECKUP

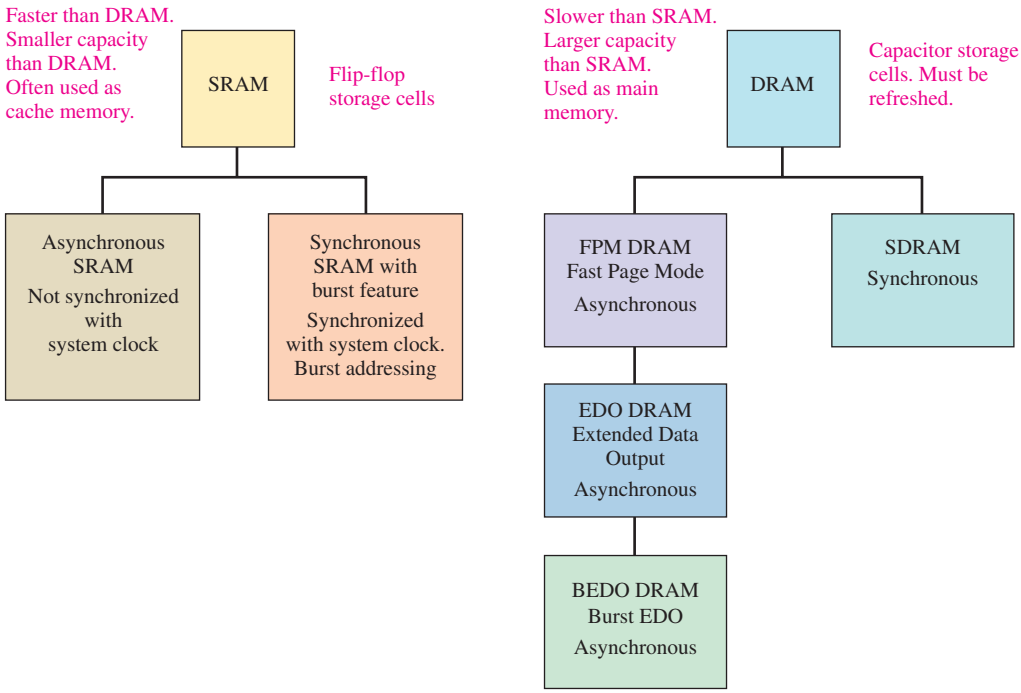
1. Describe the checksum method of ROM testing.
2. Why can the checksum method not be applied to RAM testing?
3. List the three basic faults that the checkerboard pattern test can detect in a RAM.

SUMMARY

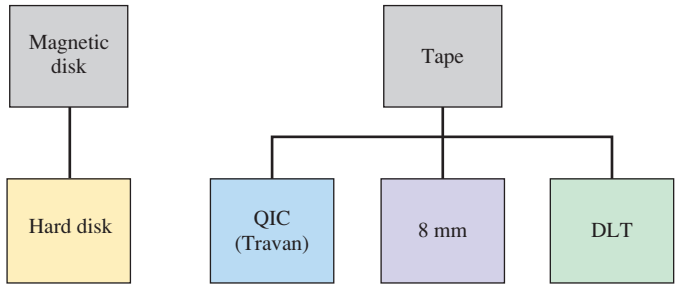
- Types of semiconductor memories:



- Types of SRAMs (Static RAMs) and DRAMs (Dynamic RAMs):



- Types of magnetic storage:



- Types of optical (laser) storage:

KEY TERMS

Key terms and other bold terms in the chapter are defined in the end-of-book glossary.

Address The location of a given storage cell or group of cells in a memory.

Blue-ray A disc storage technology that uses a blue laser to achieve more density and definition than a DVD.

Bus One or more interconnections that interface one or more devices based on a standardized specification.

Byte A group of eight bits.

Capacity The total number of data units (bits, nibbles, bytes, words) that a memory can store.

Cell A single storage element in a memory.

Cloud storage A network of servers that is connected to a user device through the Internet.

DRAM Dynamic random-access memory; a type of semiconductor memory that uses capacitors as the storage elements and is a volatile, read/write memory.

EPROM Erasable programmable read-only memory; a type of semiconductor memory device that typically uses ultraviolet light to erase data.

FIFO First in–first out memory.

Flash memory A nonvolatile read/write random-access semiconductor memory in which data are stored as charge on the floating gate of a certain type of FET.

Hard disk A magnetic storage device; typically, a stack of two or more rigid disks enclosed in a sealed housing.

LIFO Last in–first out memory; a memory stack.

Memory The portion of a computer or other system that stores binary data.

Memory hierarchy The arrangement of various memory elements within a computer architecture to achieve maximum performance.

PROM Programmable read-only memory; a type of semiconductor memory.

RAM Random-access memory; a volatile read/write semiconductor memory.

Read The process of retrieving data from a memory.

ROM Read-only memory; a nonvolatile random-access semiconductor memory.

Server Any computerized process that shares a resource with one or more clients. A computer and software with a large memory capacity that responds to requests across a network to provide file storage and access as well as services such as file sharing.

SRAM Static random-access memory; a type of volatile read/write semiconductor memory.

Word A group of bits or bytes that acts as a single entity that can be stored in one memory location; two bytes.

Write The process of storing data in a memory.

TRUE/FALSE QUIZ

Answers are at the end of the chapter.

1. A nibble consists of eight bits.
2. A memory cell can store a byte of data.
3. The location of a unit of data in a memory array is called its address.
4. A data bus is bidirectional in operation.
5. RAM is a *random address memory*.
6. Data stored in a static RAM is retained even after power is removed.
7. Cache is a type of memory used for intermediate or temporary storage of data.
8. Dynamic RAMs must be periodically refreshed to retain data.
9. ROM is a *read-only memory*.
10. A flash memory uses a flashing beam of light to store data.
11. Registers are at the top of a memory hierarchy.
12. Cloud storage is accessed through the Internet.

SELF-TEST

Answers are at the end of the chapter.

1. The bit capacity of a memory that has 512 addresses and can store 8 bits at each address is
   (a) 512 **(b)** 1024 **(c)** 2048 **(d)** 4096
2. A 16-bit word consists of
   (a) 3 bytes **(b)** 4 nibbles **(c)** 4 bytes **(d)** 3 bytes and 1 nibble
3. Data are stored in a random-access memory (RAM) during the
   (a) read operation **(b)** enable operation
   (c) write operation **(d)** addressing operation
4. Data that are stored at a given address in a random-access memory (RAM) are lost when
   (a) power goes off **(b)** the data are read from the address
   (c) new data are written at the address **(d)** answers (a) and (c)
5. A ROM is a
   (a) nonvolatile memory **(b)** volatile memory
   (c) read/write memory **(d)** byte-organized memory
6. A memory with 512 addresses has
   (a) 512 address lines **(b)** 12 address lines
   (c) 1 address line **(d)** 9 address lines
7. A byte-organized memory has
   (a) 1 data output line **(b)** 4 data output lines
   (c) 8 data output lines **(d)** 16 data output lines
8. The storage element of a DRAM is a
   (a) resistor **(b)** transistor **(c)** capacitor **(d)** diode
9. ADDRESS-BURST is a feature of
   (a) synchronous SRAM **(b)** asynchronous SRAM
   (c) fast page mode DRAM **(d)** synchronous DRAM

10. In a computer, the BIOS programs are stored in the
(a) ROM **(b)** RAM
(c) SRAM **(d)** DRAM

11. SRAM, DRAM, flash, and EEPROM are all
(a) magneto-optical storage devices **(b)** semiconductor storage devices
(c) magnetic storage devices **(d)** optical storage devices

12. Optical storage devices employ
(a) ultraviolet light **(b)** electromagnetic fields
(c) optical couplers **(d)** lasers

13. Memory latency is
(a) average down time **(b)** time to reference a block of data
(c) processor access time **(d)** the hit rate

14. A facility that houses a cloud storage system is called a
(a) server **(b)** data center
(c) computer center **(d)** cloud house

PROBLEMS

Answers to odd-numbered problems are at the end of the book.

Section 11–1 Semiconductor Memory Basics

1. How would you distinguish between the two memories in Figure 11–78?

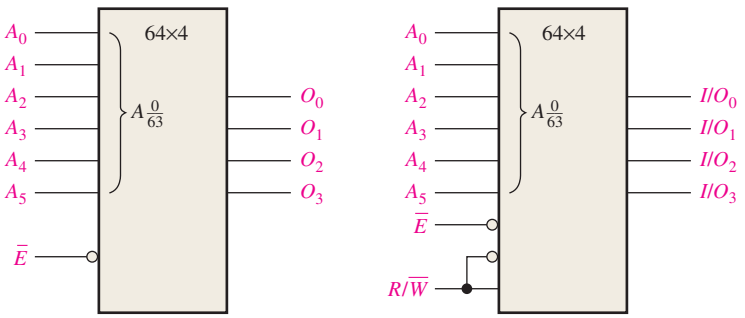


FIGURE 11–78

2. How are bits, bytes, nibbles, and words related?

3. Explain the basic memory operations.

4. What memory address (0 through 256) is represented by each of the following hexadecimal numbers?
(a) $0C_{16}$ **(b)** $5E_{16}$ **(c)** DF_{16}

Section 11–2 The Random-Access Memory (RAM)

5. A static memory array with four rows similar to the one in Figure 11–10 is initially storing all 0s. What is its content after the following conditions? Assume a 1 selects a row.

Row 0 = 1, Data in (Bit 0) = 1
Row 1 = 0, Data in (Bit 1) = 1
Row 2 = 1, Data in (Bit 2) = 0
Row 3 = 0, Data in (Bit 3) = 1

6. Draw a basic logic diagram for a 512 × 4-bit static RAM, showing all the inputs and outputs.

7. Assuming that a 64k × 8 SRAM has a structure similar to that of the SRAM in Figure 11–12. determine the number of rows and 8-bit columns in its memory cell array.
8. Redraw the block diagram in Figure 11–12 for a 64k × 8 memory.
9. What is cache memory?
10. What are the different types of RAM families available?

Section 11–3 The Read-Only Memory (ROM)

11. For the ROM array in Figure 11–79, determine the outputs for all possible input combinations, and summarize them in tabular form (Blue cell is a 1, gray cell is a 0).

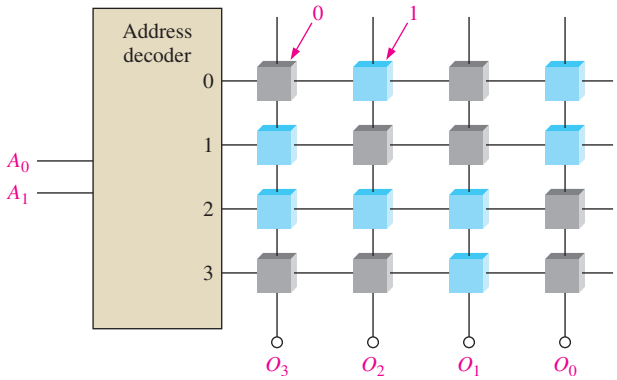


FIGURE 11–79

12. Determine the truth table for the ROM in Figure 11–80.

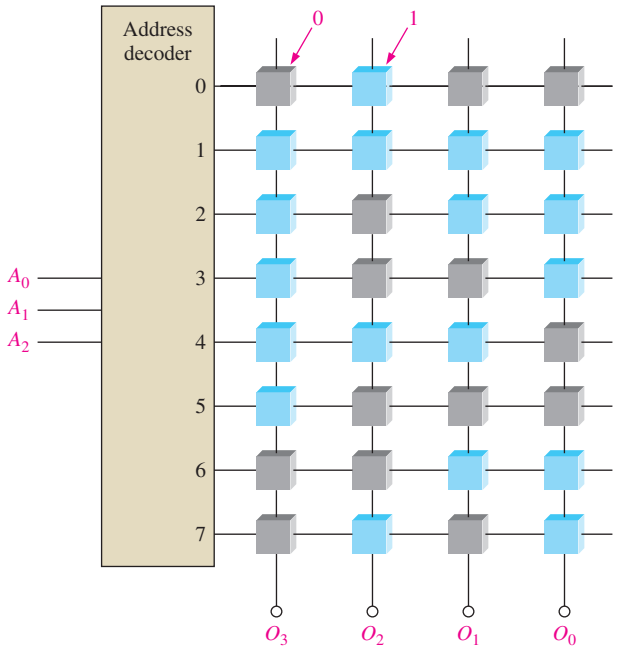


FIGURE 11–80

13. Using a procedure similar to that in Example 11–1, design a ROM for conversion of single-digit BCD to excess-3 code.
14. What is the total *bit* capacity of a ROM that has 14 address lines and 8 data outputs?

Section 11–4 Programmable ROMs

15. Assuming that the PROM matrix in Figure 11–81 is programmed by blowing a fuse link to create a 0, indicate the links to be blown to program an X^3 look-up table, where X is a number from 0 through 7.

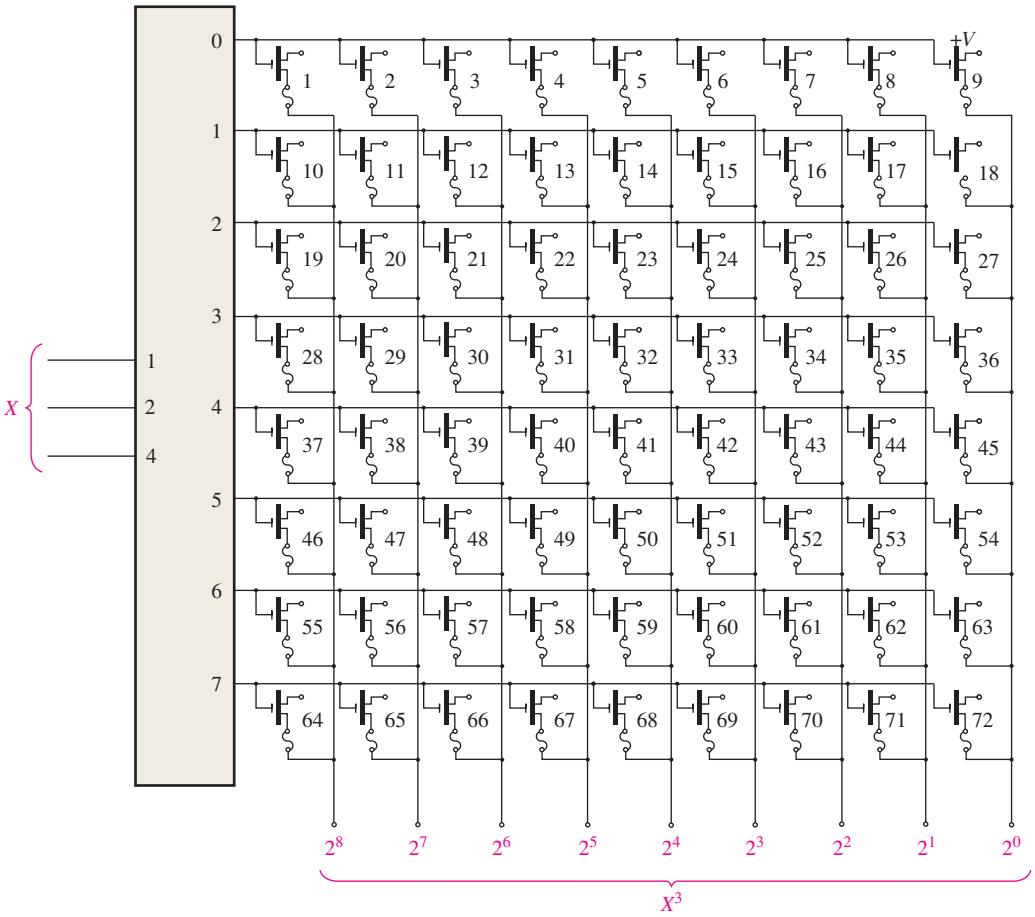


FIGURE 11–81

16. Determine the addresses that are programmed and the contents of each address after the programming sequence in Figure 11–82 has been applied to an EPROM like the one shown in Figure 11–31.

Section 11–6 Memory Expansion

17. Use 16k × 4 DRAMs to build a 64k × 8 DRAM. Show the logic diagram.

18. Using a block diagram, show how 64k × 1 dynamic RAMs can be expanded to build a 256k × 4 RAM.

19. What is the word length and the word capacity of the memory of Problem 17? Problem 18?

Section 11–7 Special Types of Memories

20. Complete the timing diagram in Figure 11–83 by showing the output waveforms that are initially all LOW for a FIFO serial memory like that shown in Figure 11–50.

21. Consider a 4096 × 8 RAM in which the last 64 addresses are used as a LIFO stack. If the first address in the RAM is 000_{16}, designate the 64 addresses used for the stack.

22. In the memory of Problem 21, sixteen bytes are pushed into the stack. At what address is the first byte in located? At what address is the last byte in located?

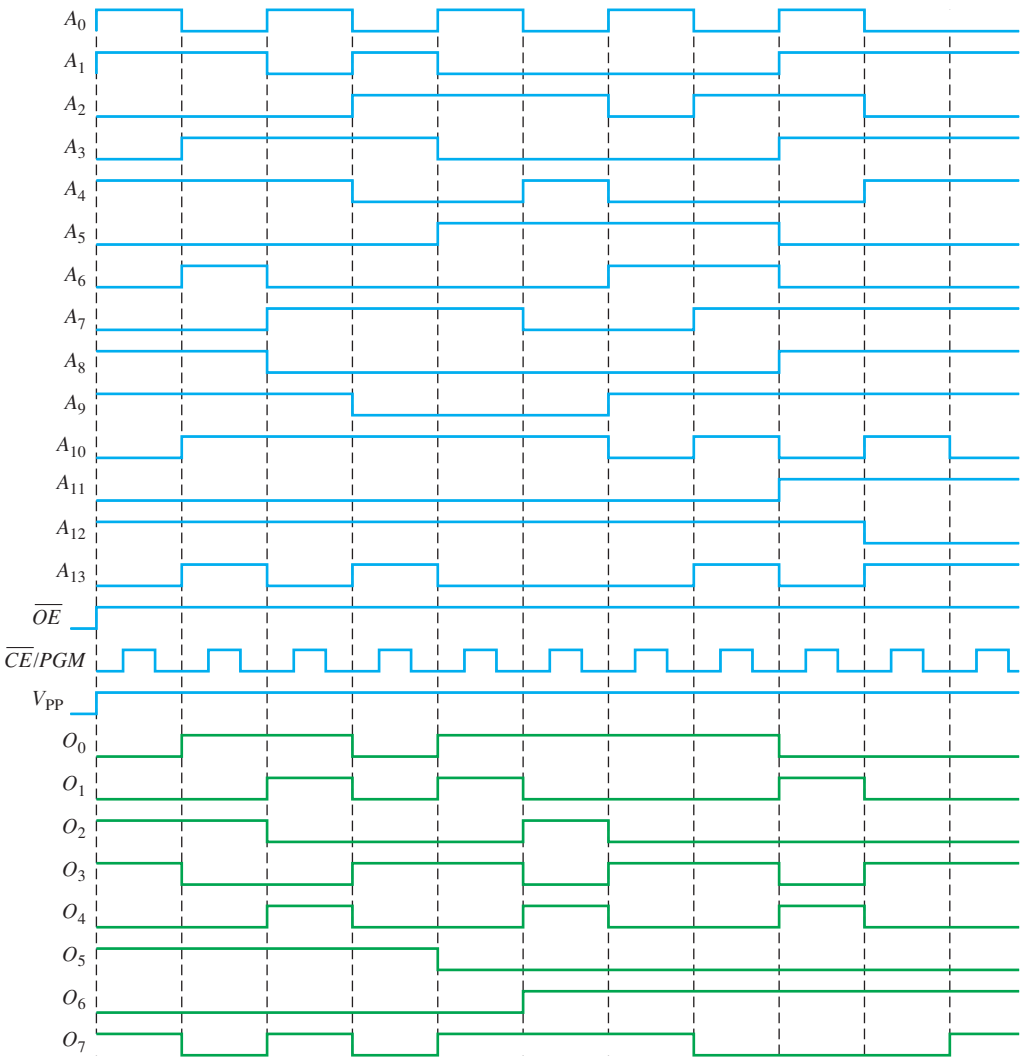


FIGURE 11–82

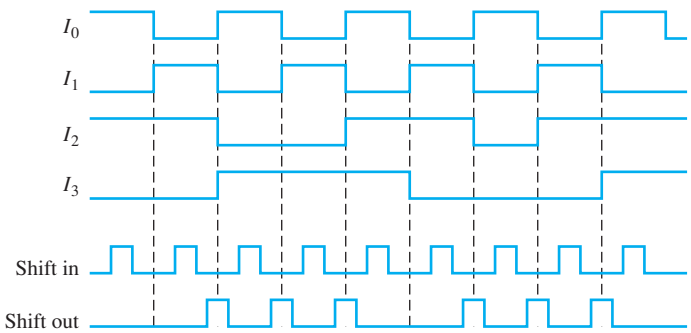


FIGURE 11–83

Section 11–8 Magnetic and Optical Storage

23. Describe the physical structure of a hard disk.

24. Explain the basic read/write principles involved in a hard/disk.

25. What are the parameters used to measure the performance of a hard disk?

26. What are the differences between a CD-R and a CD-RW?

27. What is the main difference between a CD and a DVD?

28. What is a Blu-ray disc?

Section 11–9 Memory Hierarchy

29. What does *memory hierarchy* mean?

30. What are the memory storage levels used in computers?

31. Describe hit rate.

32. If the miss rate in a certain memory is 0.2, what is the hit rate?

Section 11–10 Cloud Storage

33. Draw a diagram of a cloud storage system with six servers.

34. What does a server in a cloud storage system provide?

35. What is the architecture of a cloud storage system?

36. List five properties of a cloud storage system and briefly discuss each.

Section 11–11 Troubleshooting

37. Determine if the contents of the ROM in Figure 11–84 are correct.

38. A 128 × 8 ROM is implemented as shown in Figure 11–85. The decoder decodes the two most significant address bits to enable the ROMs one at a time, depending on the address selected.

(a) Express the lowest address and the highest address of each ROM as hexadecimal numbers.

(b) Assume that a single checksum is used for the entire memory and it is stored at the highest address. Develop a flowchart for testing the complete memory system.

(c) Assume that each ROM has a checksum stored at its highest address. Modify the flowchart developed in part (b) to accommodate this change.

(d) What is the disadvantage of using a single checksum for the entire memory rather than a checksum for each individual ROM?

ROM
1 0 1 1 1
1 1 1 1 0
1 1 0 1 1
1 0 1 1 0
1 1 1 0 1
1 1 1 0 0
0 0 0 0 1
Checksum 0 1 1 0 0

FIGURE 11–84

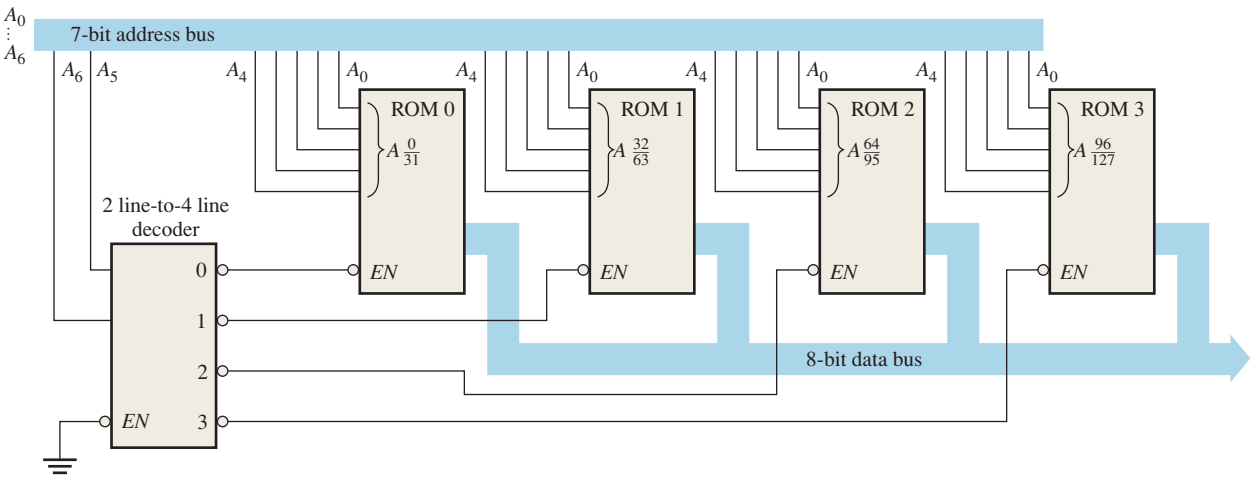


FIGURE 11–85

39. Suppose that a checksum test is run on the memory in Figure 11–85 and each individual ROM has a checksum at its highest address. What IC or ICs will you replace for each of the following error messages that appear on the system's video monitor?

(a) ADDRESSES 40–5F FAULTY **(b)** ADDRESSES 20–3F FAULTY

(c) ADDRESSES 00–7F FAULTY

ANSWERS

SECTION CHECKUPS

Section 11–1 Semiconductor Memory Basics

1. Bit is the smallest unit of data.

2. 256 bytes is 2048 bits.

3. A write operation stores data in memory.
4. A read operation takes a copy of data from memory.
5. A unit of data is located by its address.
6. A RAM is volatile and has read/write capability. A ROM is nonvolatile and has only read capability.

Section 11–2 The Random-Access Memory (RAM)

1. Asynchronous and synchronous with burst feature
2. A small fast memory between the CPU and main memory
3. SRAMs have latch storage cells that can retain data indefinitely while power is applied. DRAMs have capacitive storage cells that must be periodically refreshed.
4. The refresh operation prevents data from being lost because of capacitive discharge. A stored bit is restored periodically by recharging the capacitor to its nominal level.
5. FPM, EDO, BEDO, Synchronous

Section 11–3 The Read-Only Memory (ROM)

1. 512×8 equals 4096 bits.
2. Mask ROM, PROM, EPROM, UV EPROM, EEPROM
3. Eight bits of address are required for 256 byte locations ($2^8 = 256$).

Section 11–4 Programmable ROMs

1. PROMs are field-programmable; ROMs are not.
2. Presence or absence of stored charge
3. Read is the normal mode of operation for a PROM.

Section 11–5 The Flash Memory

1. Flash, ROM, EPROM, and EEPROM are nonvolatile.
2. Flash is nonvolatile; SRAM and DRAM are volatile.
3. Programming, read, erase

Section 11–6 Memory Expansion

1. Eight RAMs
2. Eight RAMs
3. DIMM: Dual in-line memory module

Section 11–7 Special Types of Memories

1. In a FIFO memory the *first* bit (or word) *in* is the *first* bit (or word) *out.*
2. In a LIFO memory the *last* bit (or word) *in* is the *first* bit (or word) *out.* A stack is a LIFO.
3. The PUSH operation or instruction adds data to the memory stack.
4. The POP operation or instruction removes data from the memory stack.
5. CCD is a charge-coupled device.

Section 11–8 Magnetic and Optical Storage

1. Magnetic storage: hard disk, tape, and magneto-optical disk
2. A magnetic disk is organized in tracks and sectors.
3. A magneto-optical disk uses a laser beam and an electromagnet.
4. Optical storage: CD-ROM, CD-R, CD-RW, DVD-ROM, WORM, Blu-ray Disc (BD)

Section 11–9 Memory Hierarchy

1. The purpose of memory hierarchy is to obtain the fastest access time at the lowest cost.
2. Access time is the time it takes a processor to retrieve (read) or write a block of data stored in the memory.

3. Generally, the higher the capacity the lower the cost per bit.
4. Yes
5. A hit is when the processor finds the requested data at the first place it looks. A miss is when the processor fails to find the requested data and has to go to another level of memory to find it.
6. The hit rate

Section 11–10 Cloud Storage

1. A cloud storage system is a remote network of servers connected to a user device through the Internet.
2. A server is any computerized process that shares a resource with one or more clients. Practically, a storage server is a computer and software with a large memory capacity that responds to requests across a network to provide file storage and access as well as services such as file sharing.
3. A user connects via Internet access.
4. Data storage and retrieval from any physical location with Internet access, any computer can be used and a local physical backup storage device is not necessary, and other users can be permitted to access your data.

Section 11–11 Troubleshooting

1. The contents of the ROM are added and compared with a prestored checksum.
2. Checksum cannot be used because the contents of a RAM are not fixed.
3. (1) a short between adjacent cells; (2) an inability of some cells to store both 1s and 0s; (3) dynamic altering of the contents of one address when the contents of another address change.

RELATED PROBLEMS FOR EXAMPLES

11–1 $G_3G_2G_1G_0 = 1110$

11–2 Connect eight 64k × 1 ROMs in parallel to form a 64k × 8 ROM.

11–3 Sixteen 64k × 1 ROMs

11–4 See Figure 11–86.

11–5 ROM 1: 0 to 524,287; ROM 2: 524,288 to 1,048,575

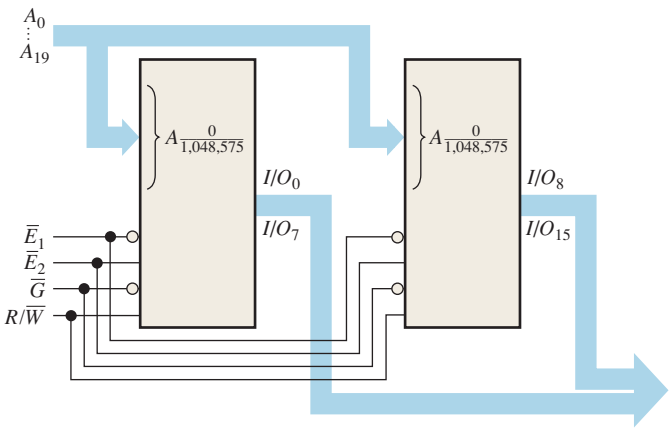


FIGURE 11–86

TRUE/FALSE QUIZ

1. F **2.** F **3.** T **4.** T **5.** F **6.** F
7. T **8.** T **9.** T **10.** F **11.** T **12.** T

SELF-TEST

1. (d) **2.** (b) **3.** (c) **4.** (d) **5.** (a) **6.** (d) **7.** (c)
8. (c) **9.** (a) **10.** (a) **11.** (b) **12.** (d) **13.** (c) **14.** (b)

CHAPTER 12

Signal Conversion and Processing

CHAPTER OUTLINE

CHAPTER OBJECTIVES

- Explain how analog signals are converted to digital form
- Discuss the purpose of filtering
- Describe the sampling process
- State the purpose of analog-to-digital conversion
- Explain how several types of ADCs operate
- State the purpose of digital-to-analog conversion
- Explain how DACs operate
- List the essential elements in a digital signal processing system
- Explain the basic concepts of a digital signal processor (DSP)
- Describe the basic architecture of a DSP
- Name some of the functions that a DSP performs

KEY TERMS

Key terms are in order of appearance in the chapter.

- Sampling
- Nyquist frequency
- Aliasing
- Analog-to-digital converter (ADC)
- Quantization
- Digital-to-analog converter (DAC)
- Digital signal processor (DSP)
- DSP core
- MIPS
- MFLOPS
- MMACS
- Pipeline
- Fetch
- Decode
- Execute

VISIT THE WEBSITE

Study aids for this chapter are available at http://www.pearsonglobaleditions.com/floyd

INTRODUCTION

This chapter provides an introduction to interfacing digital and analog systems using methods of analog-to-digital and digital-to-analog conversions.

Digital signal processing is a technology that is widely used in many applications, such as automotive, consumer, graphics/imaging, industrial, instrumentation, medical, military, telecommunications, and voice/speech applications. Digital signal processing incorporates mathematics, software programming, and processing hardware to manipulate analog signals.

12–1 Analog-to-Digital Conversion

In order to process signals using digital techniques, the incoming analog signal must be converted into digital form.

After completing this section, you should be able to

- Explain the basic process of converting an analog signal to digital
- Describe the purpose of the sample-and-hold function
- Define the Nyquist frequency
- Define the reason for *aliasing* and discuss how it is eliminated
- Describe the purpose of an ADC

Sampling and Filtering

An anti-aliasing filter and a sample-and-hold circuit are two functions typically found in a digital signal processing system. The sample-and-hold function does two operations, the first of which is sampling. **Sampling** is the process of taking a sufficient number of discrete values at points on a waveform that will define the shape of the waveform. The more samples you take, the more accurately you can define a waveform. Sampling converts an analog signal into a series of impulses, each representing the amplitude of the signal at a given instant in time. Figure 12–1 illustrates the process of sampling.

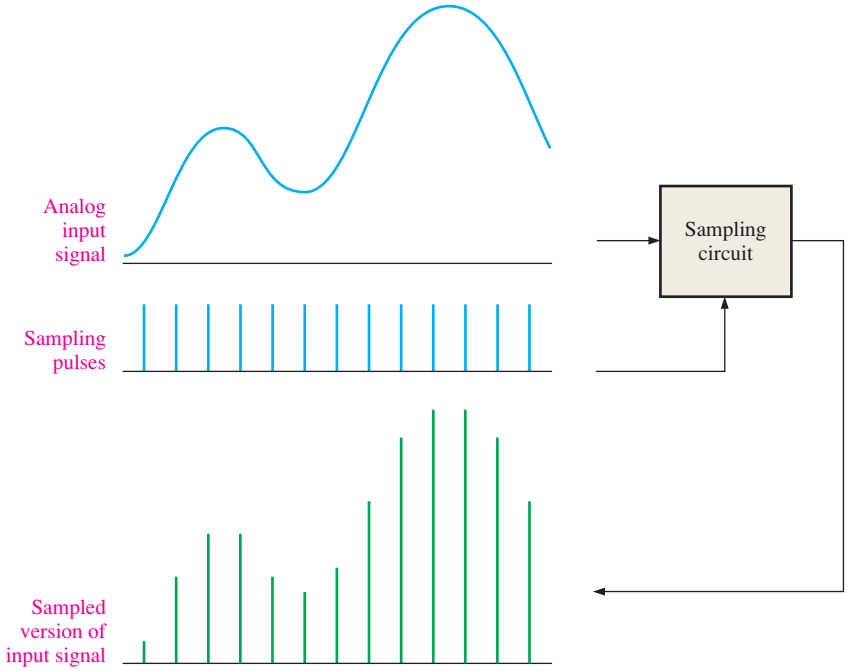


FIGURE 12–1 Illustration of the sampling process.

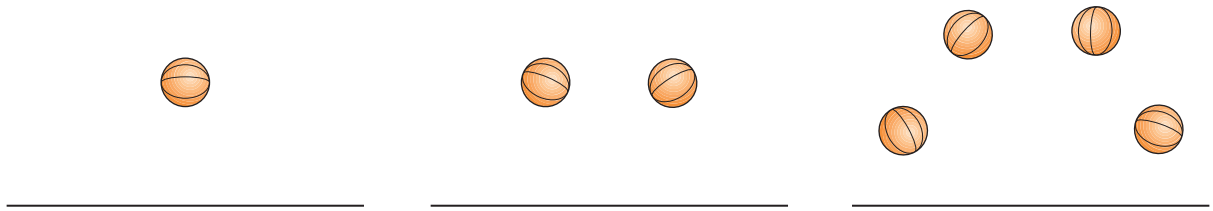
When an analog signal is to be sampled, there are certain criteria that must be met in order to accurately represent the original signal. All analog signals (except a pure sine wave) contain a spectrum of component frequencies. For a pure sine wave, these frequencies appear in multiples called *harmonics*. The harmonics of an analog signal are sine waves of different frequencies and amplitudes. When the harmonics of a given periodic waveform are added, the result is the original signal. Before a signal can be sampled, it must be passed through a low-pass filter (anti-aliasing filter) to eliminate harmonic frequencies above a certain value as determined by the Nyquist frequency.

The Sampling Theorem

Notice in Figure 12–1 that there are two input waveforms. One is the analog signal and the other is the sampling pulse waveform. The sampling theorem states that, in order to represent an analog signal, the sampling frequency, f_{sample}, must be at least twice the highest frequency component $f_{a(max)}$ of the analog signal. Another way to say this is that the highest analog frequency can be no greater than one-half the sampling frequency. The frequency $f_{a(max)}$ is known as the **Nyquist frequency** and is expressed in Equation 12–1. In practice, the sampling frequency should be more than twice the highest analog frequency.

$$f_{sample} > 2f_{a(max)} \qquad \text{Equation 12–1}$$

To intuitively understand the sampling theorem, a simple "bouncing-ball" analogy may be helpful. Although it is not a perfect representation of the sampling of electrical signals, it does serve to illustrate the basic idea. If a ball is photographed (sampled) at one instant during a single bounce, as illustrated in Figure 12–2(a), you cannot tell anything about the path of the ball except that it is off the floor. You can't tell whether it is going up or down or the distance of its bounce. If you take photos at two equally-spaced instants during one bounce, as shown in part (b), you can obtain only a minimum amount of information about its movement and nothing about the distance of the bounce. In this particular case, you know only that the ball has been in the air at the times the two photos were taken and that the maximum height of the bounce is at least equal to the height shown in each photo. If you take four photos, as shown in part (c), then the path that the ball follows during a bounce begins to emerge. The more photos (samples) that you take, the more accurately you can determine the path of the ball as it bounces.



(a) One sample of a ball during a single bounce

(b) Two samples of a ball during a single bounce. This is the absolute minimum required to tell anything about its movement, but generally insufficient to describe its path.

(c) Four samples of a ball during a single bounce form a rough picture of the path of the ball.

FIGURE 12–2 Bouncing ball analogy of sampling theory.

The Need for Filtering

Low-pass filtering is necessary to remove all frequency components (harmonics) of the analog signal that exceed the Nyquist frequency. If there are any frequency components in the analog signal that exceed the Nyquist frequency, an unwanted condition known as **aliasing** will occur. An alias is a signal produced when the sampling frequency is not at least twice the signal frequency. An alias signal has a frequency that is less than the highest frequency in the analog signal being sampled and therefore falls within the spectrum or frequency band of the input analog signal causing distortion. Such a signal is actually "posing" as part of the analog signal when it really isn't, thus the term *alias*.

Another way to view aliasing is by considering that the sampling pulses produce a spectrum of harmonic frequencies above and below the sample frequency, as shown in Figure 12–3. If the analog signal contains frequencies above the Nyquist frequency, these frequencies overlap into the spectrum of the sample waveform as shown and interference occurs. The lower frequency components of the sampling waveform become mixed in with the frequency spectra of the analog waveform, resulting in an aliasing error.

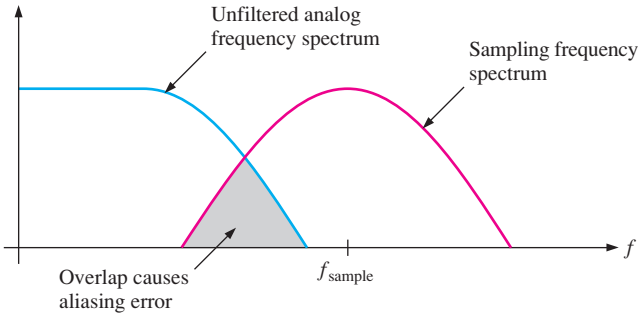


FIGURE 12–3 A basic illustration of the condition $f_{sample} < 2f_{a(max)}$.

A low-pass anti-aliasing filter must be used to limit the frequency spectrum of the analog signal for a given sample frequency. To avoid an aliasing error, the filter must at least eliminate all analog frequencies above the minimum frequency in the sampling spectrum, as illustrated in Figure 12–4. Aliasing can also be avoided by sufficiently increasing the sampling frequency. However, the maximum sampling frequency is usually limited by the performance of the analog-to-digital converter (ADC) that follows it.

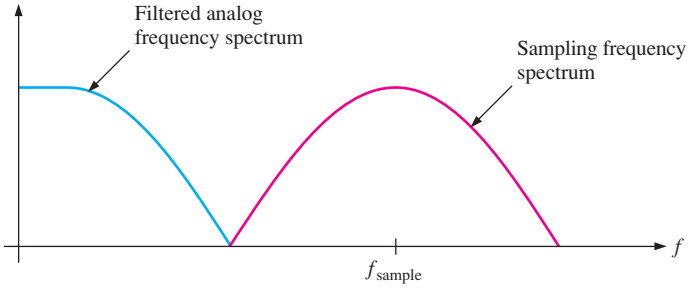


FIGURE 12–4 After low-pass filtering, the frequency spectra of the analog and the sampling signals do not overlap, thus eliminating aliasing error.

An Application

An example of the application of sampling is in digital audio equipment. The sampling rates used are 32 kHz, 44.1 kHz, or 48 kHz (the number of samples per second). The 48 kHz rate is the most common, but the 44.1 kHz rate is used for audio CDs and prerecorded tapes. According to the Nyquist rate, the sampling frequency must be at least twice the audio signal. Therefore, the CD sampling rate of 44.1 kHz captures frequencies up to about 22 kHz, which exceeds the 20 kHz specification that is common for most audio equipment.

Many applications do not require a wide frequency range to obtain reproduced sound that is acceptable. For example, human speech contains some frequencies near 10 kHz and, therefore, requires a sampling rate of at least 20 kHz. However, if only frequencies up to 4 kHz (ideally requiring an 8 kHz minimum sampling rate) are reproduced, voice is very understandable. On the other hand, if a sound signal is not sampled at a high enough rate, the effect of aliasing will become noticeable with background noise and distortion.

Holding the Sampled Value

The holding operation is the second part of the sample-and-hold function. After filtering and sampling, the sampled level must be held constant until the next sample occurs. This is necessary for the ADC to have time to process the sampled value. This sample-and-hold operation results in a "stairstep" waveform that approximates the analog input waveform, as shown in Figure 12–5.

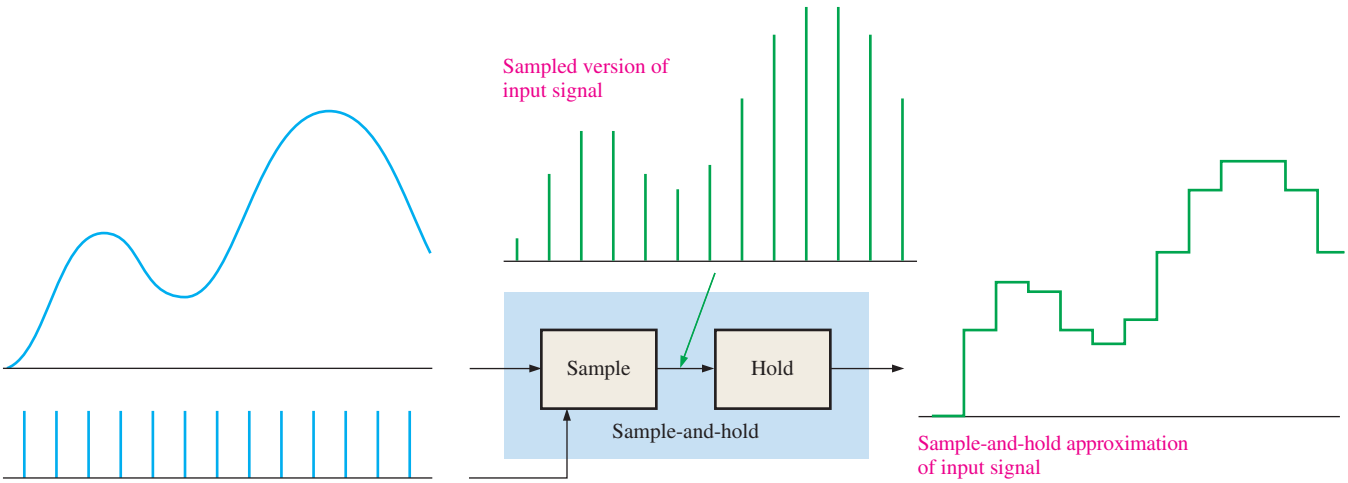


FIGURE 12–5 Illustration of a sample-and-hold operation.

Analog-to-Digital Conversion

Analog-to-digital conversion is the process of converting the output of the sample-and-hold circuit to a series of binary codes that represent the amplitude of the analog input at each of the sample times. The sample-and-hold process keeps the amplitude of the analog input signal constant between sample pulses; therefore, the analog-to-digital conversion can be done using a constant value rather than having the analog signal change during a conversion interval, which is the time between sample pulses. Figure 12–6 illustrates the basic function of an **analog-to-digital converter (ADC)**, which is a circuit that performs analog-to-digital conversion. The sample intervals are indicated by dashed lines.

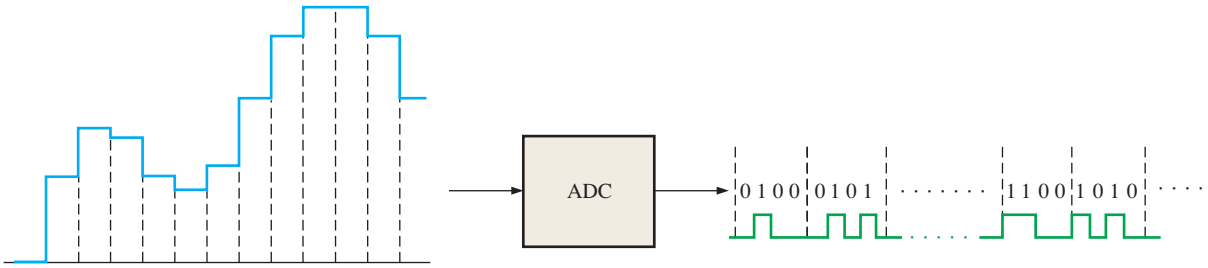


FIGURE 12–6 Basic function of an analog-to-digital converter (ADC) (The binary codes and number of bits are arbitrarily chosen for illustration only). The ADC output waveform that represents the binary codes is also shown.

Quantization

The process of converting an analog value to a code is called **quantization**. During the quantization process, the ADC converts each sampled value of the analog signal to a binary code. The more bits that are used to represent a sampled value, the more accurate is the representation.

To illustrate, let's quantize a reproduction of the analog waveform into four levels (0–3). Two bits are required for four levels. As shown in Figure 12–7, each quantization level is represented by a 2-bit code on the vertical axis, and each sample interval is numbered along the horizontal axis. The sampled data is held for the entire sample period. This data is quantized to the next lower level, as shown in Table 12–1 (for example, compare samples 3 and 4, which are assigned different levels).

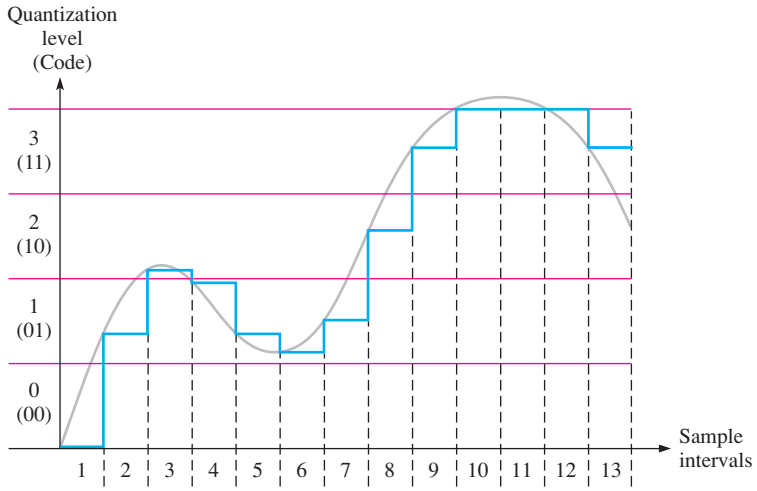


FIGURE 12–7 Sample-and-hold output waveform with four quantization levels. The original analog waveform is shown in light gray for reference.

TABLE 12–1

Two-bit quantization for the waveform in Figure 12–7.

Sample Interval	Quantization Level	Code
1	0	00
2	1	01
3	2	10
4	1	01
5	1	01
6	1	01
7	1	01
8	2	10
9	3	11
10	3	11
11	3	11
12	3	11
13	3	11

If the resulting 2-bit digital codes are used to reconstruct the original waveform, you would get the waveform shown in Figure 12–8. This operation is done by **digital-to-analog converters (DACs)**, which are circuits that perform digital-to-analog conversions. As you can see, quite a bit of accuracy is lost using only two bits to represent the sampled values.

Now, let's see how more bits will improve the accuracy. Figure 12–9 shows the same waveform with sixteen quantization levels (4 bits). The 4-bit quantization process is summarized in Table 12–2.

If the resulting 4-bit digital codes are used to reconstruct the original waveform, you would get the waveform shown in Figure 12–10. As you can see, the result is much more like the original waveform than for the case of four quantization levels in Figure 12–8. This shows that greater accuracy is achieved with more quantization bits. Typical integrated circuit ADCs use from 12 to 24 bits, and the sample-and-hold function is sometimes contained on the ADC chip. Several types of ADCs are introduced in the next section.

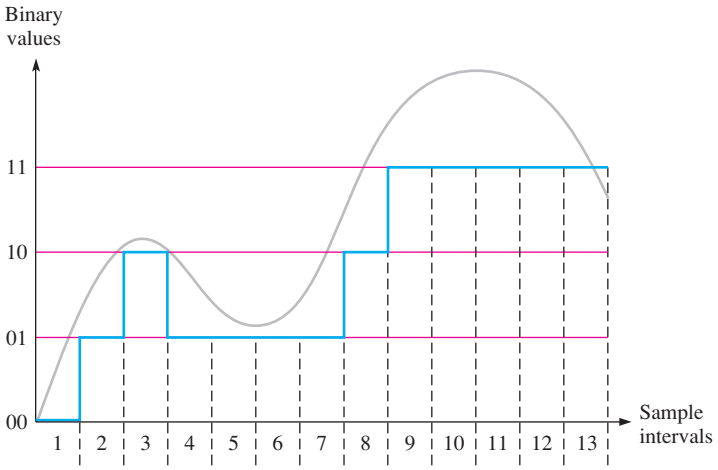


FIGURE 12–8 The reconstructed waveform in Figure 12–7 using four quantization levels (2 bits). The original analog waveform is shown in light gray for reference.

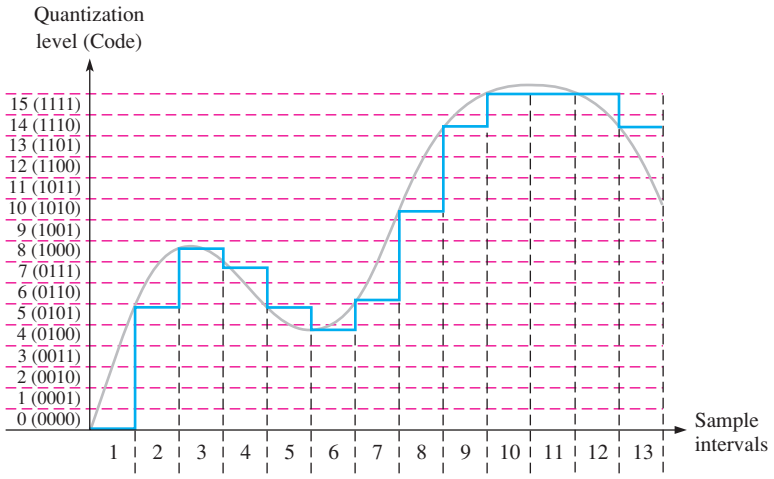


FIGURE 12–9 Sample-and-hold output waveform with sixteen quantization levels. The original analog waveform is shown in light gray for reference.

TABLE 12–2

Four-bit quantization for the waveform in Figure 12–9.

Sample Interval	Quantization Level	Code
1	0	0000
2	5	0101
3	8	1000
4	7	0111
5	5	0101
6	4	0100
7	6	0110
8	10	1010
9	14	1110
10	15	1111
11	15	1111
12	15	1111
13	14	1110

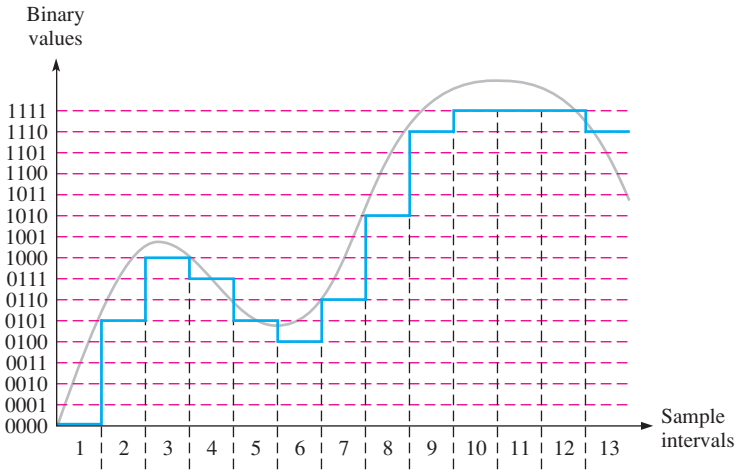


FIGURE 12–10 The reconstructed waveform in Figure 12–9 using sixteen quantization levels (4 bits). The original analog waveform is shown in light gray for reference.

SECTION 12–1 CHECKUP

Answers are at the end of the chapter.

1. What does sampling mean?
2. Why must you hold a sampled value?
3. If the highest frequency component in an analog signal is 20 kHz, what is the minimum sample frequency?
4. What does quantization mean?
5. What determines the accuracy of the quantization process?

12–2 Methods of Analog-to-Digital Conversion

As you have seen, analog-to-digital conversion is the process by which an analog quantity is converted to digital form. It is necessary when measured quantities must be in digital form for processing or for display or storage. Some common types of analog-to-digital converters (ADCs) are now examined. Two important ADC parameters are *resolution,* which is the number of bits, and *throughput,* which is the sampling rate an ADC can handle in units of samples per second (sps).

After completing this section, you should be able to

- Explain what an operational amplifier is
- Show how the op-amp can be used as an inverting amplifier or a comparator
- Explain how a flash ADC works
- Discuss dual-slope ADCs
- Describe the operation of a successive-approximation ADC
- Describe a delta-sigma ADC
- Discuss testing ADCs for a missing code, incorrect code and offset

A Quick Look at an Operational Amplifier

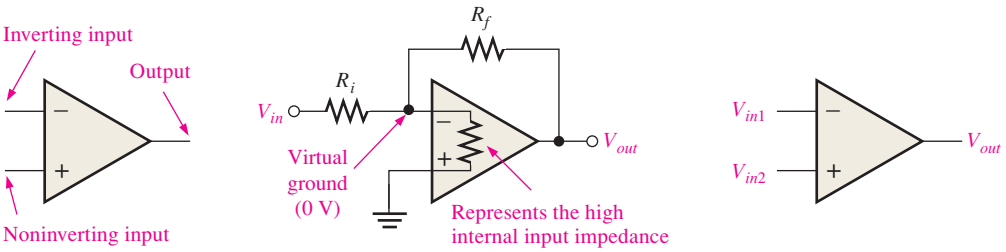
Before getting into analog-to-digital converters (ADCs), let's look briefly at an element that is common to most types of ADCs and digital-to-analog converters (DACs). This element is the operational amplifier, or op-amp for short. This is an abbreviated coverage of the op-amp.

An **op-amp** is a linear amplifier that has two inputs (inverting and noninverting) and one output. It has a very high voltage gain and a very high input impedance, as well as a very low output impedance. The op-amp symbol is shown in Figure 12–11(a). When used as an inverting amplifier, the op-amp is configured as shown in part (b). The feedback resistor, R_f, and the input resistor, R_i, control the voltage gain according to the formula in Equation 12–2, where V_{out}/V_{in} is the closed-loop voltage gain (closed loop refers to the feedback from output to input provided by R_f). The negative sign indicates inversion.

$$\frac{V_{out}}{V_{in}} = -\frac{R_f}{R_i} \qquad \text{Equation 12–2}$$

In the inverting amplifier configuration, the inverting input of the op-amp is approximately at ground potential (0 V) because feedback and the extremely high open-loop gain make the differential voltage between the two inputs extremely small. Since the noninverting input is grounded, the inverting input is at approximately 0 V, which is called *virtual ground.*

When the op-amp is used as a comparator, as shown in Figure 12–11(c), two voltages are applied to the inputs. When these input voltages differ by a very small amount, the op-amp is driven into one of its two saturated output states, either HIGH or LOW, depending on which input voltage is greater.



(a) Op-amp symbol

(b) Op-amp as an inverting amplifier with gain of R_f/R_i

(c) Op-amp as a comparator

FIGURE 12–11 The operational amplifier (op-amp).

Flash (Simultaneous) Analog-to-Digital Converter

The flash method utilizes special high-speed comparators that compare reference voltages with the analog input voltage. When the input voltage exceeds the reference voltage for a given comparator, a HIGH is generated. Figure 12–12 shows a 3-bit converter that uses seven comparator circuits; a comparator is not needed for the all-0s condition. A 4-bit converter of this type requires fifteen comparators. In general, $2^n - 1$ comparators are required for conversion to an n-bit binary code. The number of bits used in an ADC is its **resolution**. The large number of comparators necessary for a reasonable-sized binary number is one of the disadvantages of the **flash ADC**. Its chief advantage is that it provides a fast conversion time because of a high *throughput*, measured in samples per second (sps).

The reference voltage for each comparator is set by the resistive voltage-divider circuit. The output of each comparator is connected to an input of the priority encoder. The encoder is enabled by a pulse on the *EN* input, and a 3-bit code representing the value of the input appears on the encoder's outputs. The binary code is determined by the highest-order input having a HIGH level.

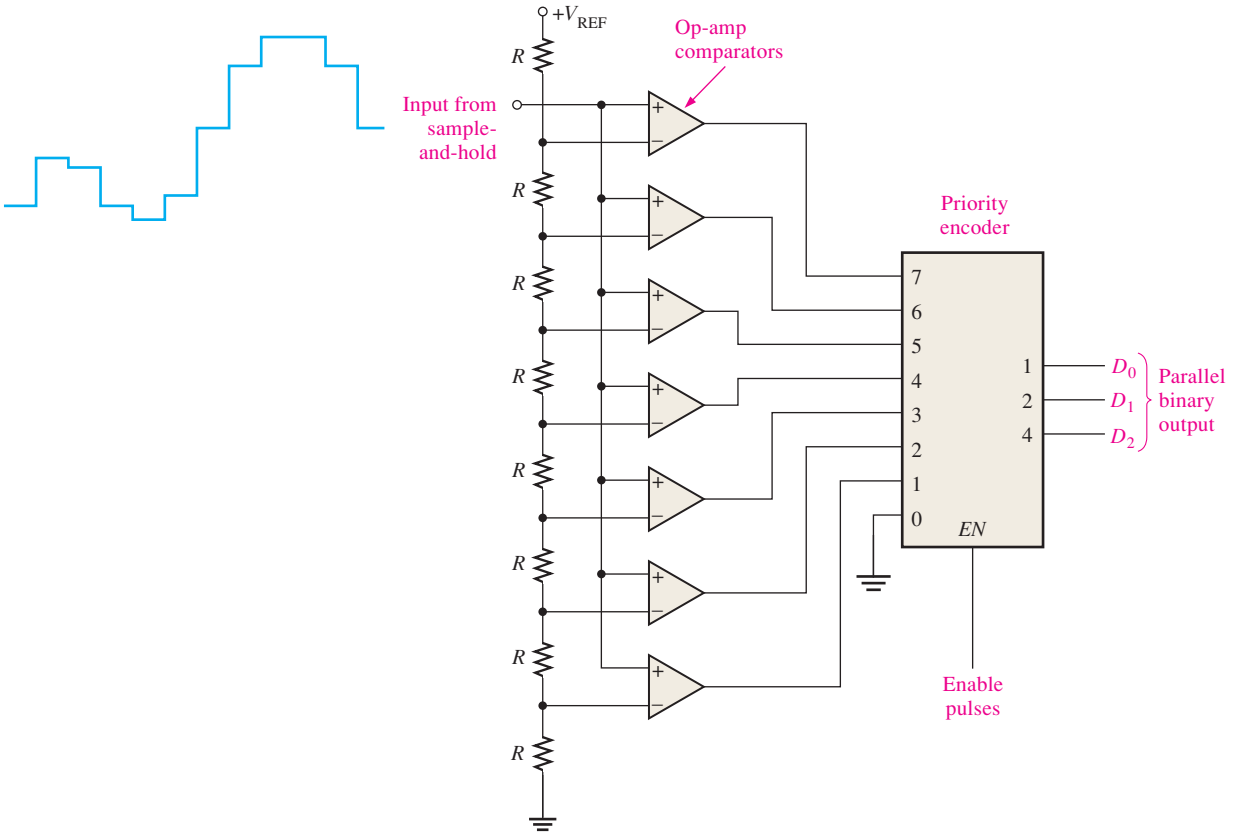


FIGURE 12–12 A 3-bit flash ADC.

The frequency of the enable pulses and the number of bits in the binary code determine the accuracy with which the sequence of binary codes represents the input of the ADC. The signal is sampled each time the enable pulse is active.

EXAMPLE 12–1

Determine the binary code output of the 3-bit flash ADC in Figure 12–12 for the input signal in Figure 12–13 and the encoder enable pulses shown. For this example, $V_{REF} = +8$ V.

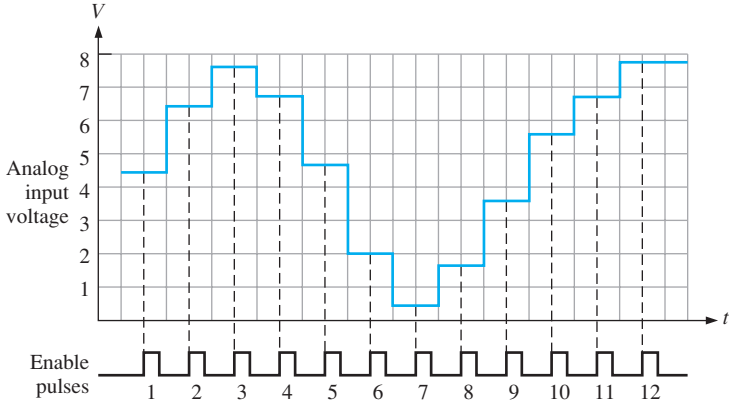


FIGURE 12–13 Sampling of values on a waveform for conversion to binary code.

Solution

The resulting digital output sequence is listed as follows and shown in the waveform diagram of Figure 12–14 in relation to the enable pulses:

100, 110, 111, 110, 100, 010, 000, 001, 011, 101, 110, 111

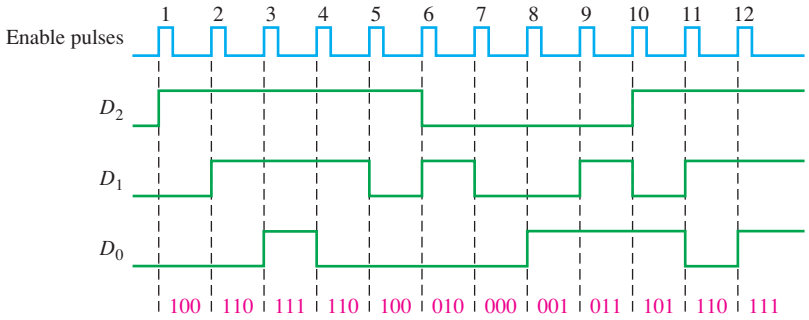


FIGURE 12–14 Resulting digital outputs for sample-and-hold values. Output D_0 is the LSB of the 3-bit binary code.

Related Problem*

If the enable pulse frequency in Figure 12–13 were halved, determine the binary numbers represented by the resulting digital output sequence for 6 pulses. Is any information lost?

*Answers are at the end of the chapter.

Dual-Slope Analog-to-Digital Converter

A dual-slope ADC is common in digital voltmeters and other types of measurement instruments. A ramp generator (integrator) is used to produce the dual-slope characteristic. A block diagram of a dual-slope ADC is shown in Figure 12–15.

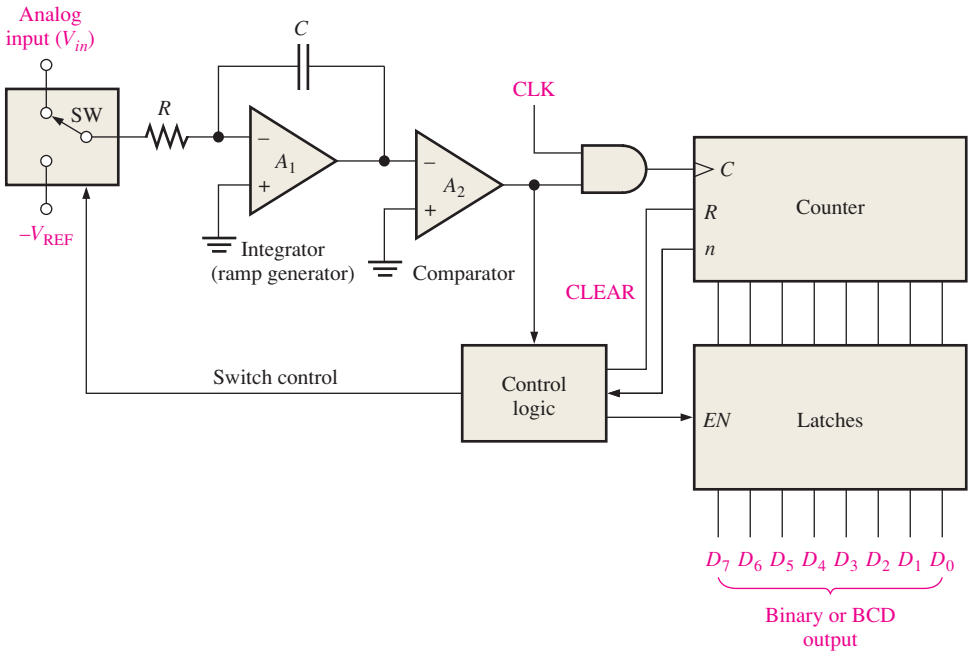
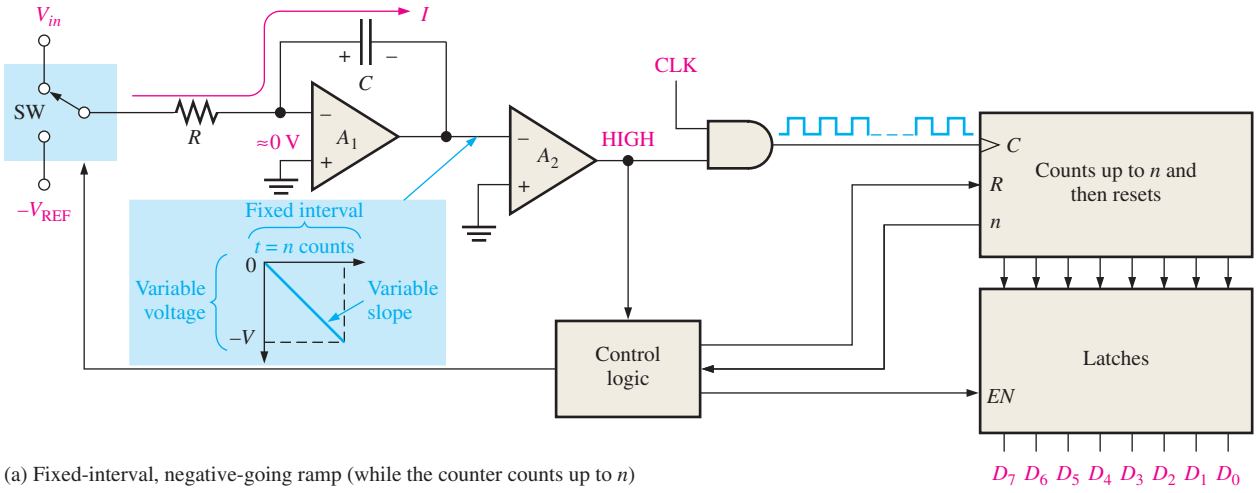
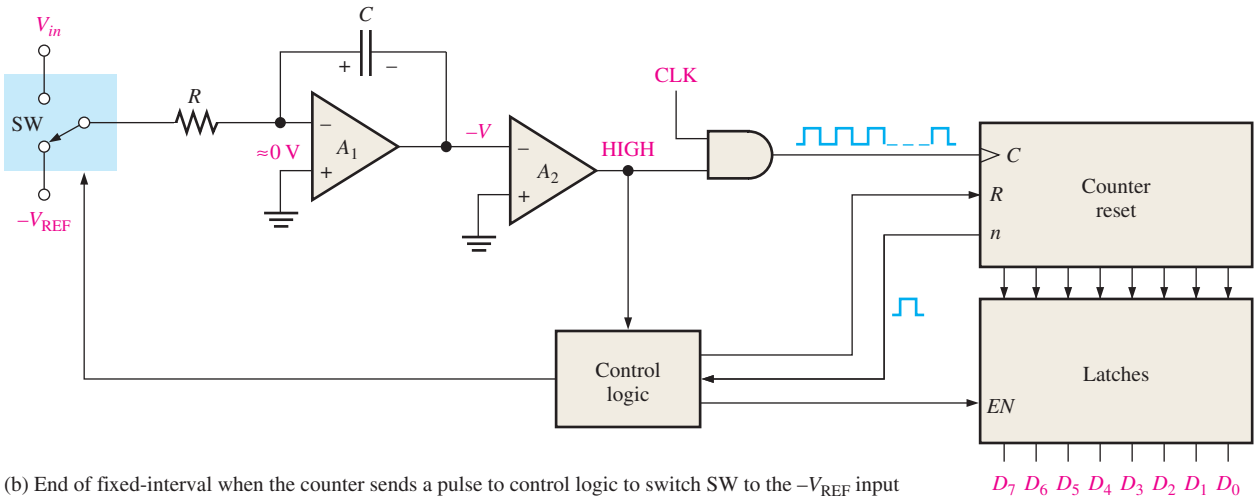


FIGURE 12–15 Basic dual-slope ADC.

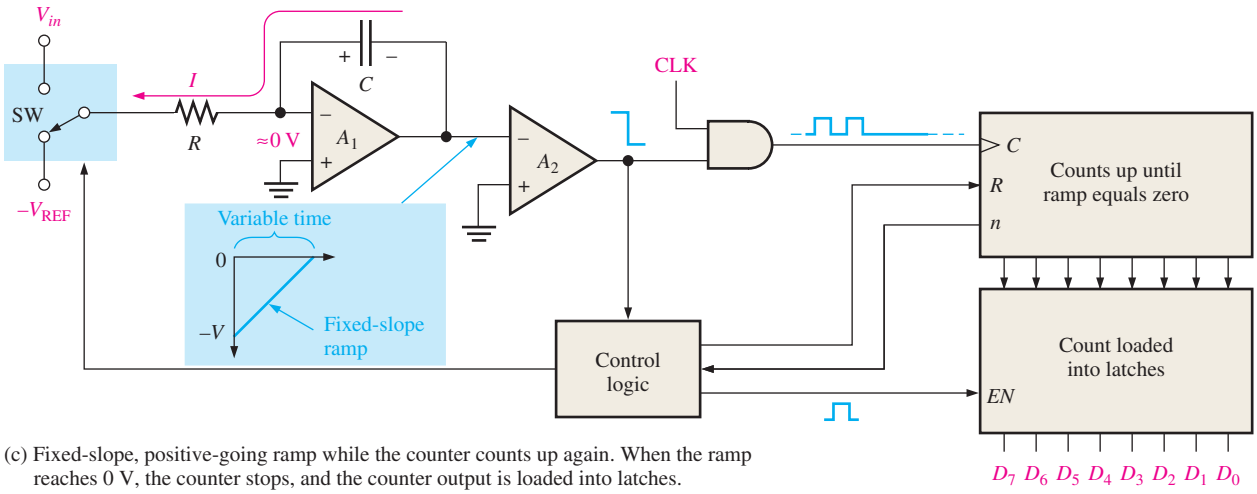
Figure 12–16 illustrates dual-slope conversion. Start by assuming that the counter is reset and the output of the integrator is zero. Now assume that a positive input voltage is applied to the input through the switch (SW) as selected by the control logic. Since the



(a) Fixed-interval, negative-going ramp (while the counter counts up to n)



(b) End of fixed-interval when the counter sends a pulse to control logic to switch SW to the $-V_{REF}$ input



(c) Fixed-slope, positive-going ramp while the counter counts up again. When the ramp reaches 0 V, the counter stops, and the counter output is loaded into latches.

FIGURE 12–16 Illustration of dual-slope conversion.

inverting input of A_1 is at virtual ground, and assuming that V_{in} is constant for a period of time, there will be constant current through the input resistor R and therefore through the capacitor C. Capacitor C will charge linearly because the current is constant, and as a result, there will be a negative-going linear voltage ramp on the output of A_1, as illustrated in Figure 12–16(a).

When the counter reaches a specified count (n), it will be reset (R), and the control logic will switch the negative reference voltage ($-V_{REF}$) to the input of A_1, as shown in Figure 12–16(b). At this point the capacitor is charged to a negative voltage ($-V$) proportional to the input analog voltage.

Now the capacitor discharges linearly because of the constant current from the $-V_{REF}$, as shown in Figure 12–16(c). This linear discharge produces a positive-going ramp on the A_1 output, starting at $-V$ and having a constant slope that is independent of the charge voltage. As the capacitor discharges, the counter advances from its RESET state. The time it takes the capacitor to discharge to zero depends on the initial voltage $-V$ (proportional to V_{in}) because the discharge rate (slope) is constant. When the integrator (A_1) output voltage reaches zero, the comparator (A_2) switches to the LOW state and disables the clock to the counter. The binary count is latched, thus completing one conversion cycle. The binary count is proportional to V_{in} because the time it takes the capacitor to discharge depends only on $-V$, and the counter records this interval of time.

Successive-Approximation Analog-to-Digital Converter

One of the most widely used methods of analog-to-digital conversion is successive-approximation. It has a much faster conversion time than the dual-slope conversion, but it is slower than the flash method. It also has a fixed conversion time that is the same for any value of the analog input.

Figure 12–17 shows a basic block diagram of a 4-bit successive approximation ADC. It consists of a DAC (DACs are covered in Section 12–3), a successive-approximation register (SAR), and a comparator. The basic operation is as follows: The input bits of the DAC are enabled (made equal to a 1) one at a time, starting with the most significant bit (MSB). As each bit is enabled, the comparator produces an output that indicates whether the input signal voltage is greater or less than the output of the DAC. If the DAC output is greater than the input signal, the comparator's output is LOW, causing the bit in the register to reset. If the output is less than the input signal, the 1 bit is retained in the register. The system does this with the MSB first, then the next most

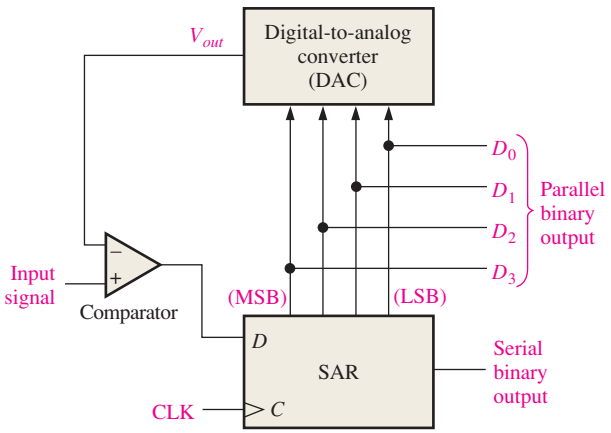


FIGURE 12–17 Successive-approximation ADC.

significant bit, then the next, and so on. After all the bits of the DAC have been tried, the conversion cycle is complete.

In order to better understand the operation of the successive-approximation ADC, let's take a specific example of a 4-bit conversion. Figure 12–18 illustrates the step-by-step conversion of a constant input voltage (5.1 V in this case). Let's assume that the DAC has the following output characteristics: $V_{out} = 8$ V for the 2^3 bit (MSB), $V_{out} = 4$ V for the 2^2 bit, $V_{out} = 2$ V for the 2^1 bit, and $V_{out} = 1$ V for the 2^0 bit (LSB).

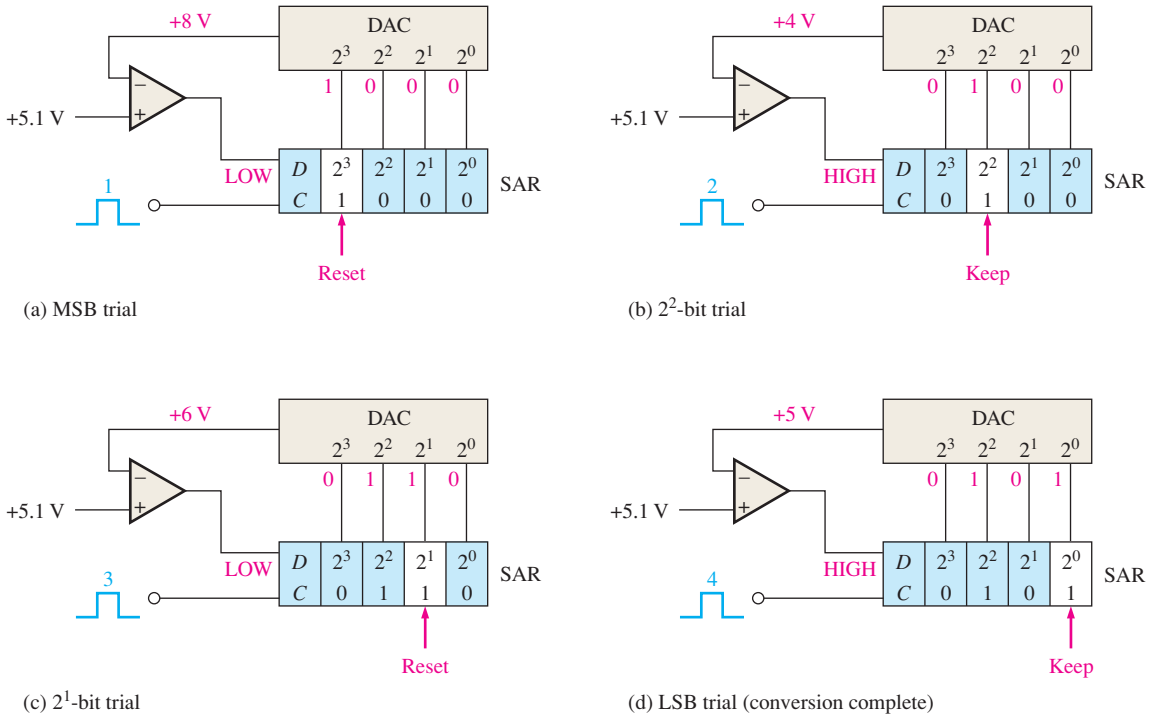


FIGURE 12–18 Illustration of the successive-approximation conversion process.

Figure 12–18(a) shows the first step in the conversion cycle with the MSB = 1. The output of the DAC is 8 V. Since this is greater than the input of 5.1 V, the output of the comparator is LOW, causing the MSB in the SAR to be reset to a 0.

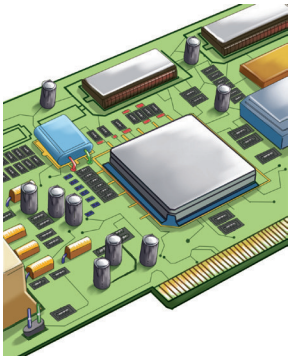
Figure 12–18(b) shows the second step in the conversion cycle with the 2^2 bit equal to a 1. The output of the DAC is 4 V. Since this is less than the input of 5.1 V, the output of the comparator switches to a HIGH, causing this bit to be retained in the SAR.

Figure 12–18(c) shows the third step in the conversion cycle with the 2^1 bit equal to a 1. The output of the DAC is 6 V because there is a 1 on the 2^2 bit input and on the 2^1 bit input; 4 V + 2 V = 6 V. Since this is greater than the input of 5.1 V, the output of the comparator switches to a LOW, causing this bit to be reset to a 0.

Figure 12–18(d) shows the fourth and final step in the conversion cycle with the 2^0 bit equal to a 1. The output of the DAC is 5 V because there is a 1 on the 2^2 bit input and on the 2^0 bit input; 4 V + 1 V = 5 V.

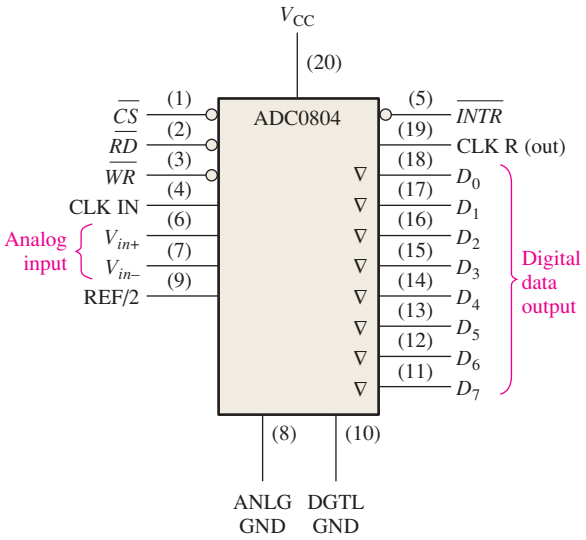
The four bits have all been tried, thus completing the conversion cycle. At this point the binary code in the register is 0101, which is approximately the binary value of the input of 5.1 V. Additional bits will produce an even more accurate result. Another conversion cycle now begins, and the basic process is repeated. The SAR is cleared at the beginning of each cycle.

IMPLEMENTATION: ANALOG-TO-DIGITAL CONVERTER



The ADC0804 is an example of a successive-approximation ADC. A block diagram is shown in Figure 12–19. This device operates from a +5 V supply and has a resolution of eight bits with a conversion time of 100 μs. Also, it has an on-chip clock generator. Optionally, an external clock can be used. The data outputs are tri-state, so they can be interfaced with a microprocessor bus system.

FIGURE 12–19 The ADC0804 analog-to-digital converter.



The basic operation of the device is as follows: The ADC0804 contains the equivalent of a 256-resistor DAC network. The successive-approximation logic sequences the network to match the analog differential input voltage ($V_{in+} - V_{in-}$) with an output from the resistive network. The MSB is tested first. After eight comparisons (sixty-four clock periods), an 8-bit binary code is transferred to output latches, and the interrupt ($\overline{INTR}$) output goes LOW. The device can be operated in a free-running mode by connecting the $\overline{INTR}$ output to the write ($\overline{WR}$) input and holding the conversion start ($\overline{CS}$) LOW. To ensure startup under all conditions, a LOW $\overline{WR}$ input is required during the power-up cycle. Taking $\overline{CS}$ low anytime after that will interrupt the conversion process.

When the $\overline{WR}$ input goes LOW, the internal successive-approximation register (SAR) and the 8-bit shift register are reset. As long as both $\overline{CS}$ and $\overline{WR}$ remain LOW, the ADC remains in a RESET state. Conversion starts one to eight clock periods after $\overline{CS}$ or $\overline{WR}$ makes a LOW-to-HIGH transition.

When a LOW is at both the $\overline{CS}$ and $\overline{RD}$ inputs, the tri-state output latch is enabled and the output code is applied to the D_0–D_7 lines. When either the $\overline{CS}$ or the $\overline{RD}$ input returns to a HIGH, the D_0–D_7 outputs are disabled.

Sigma-Delta Analog-to-Digital Converter

Sigma-delta is a widely used method of analog-to-digital conversion, particularly in telecommunications using audio signals. The method is based on **delta modulation** where the difference between two successive samples (increase or decrease) is quantized; other ADC methods were based on the absolute value of a sample. Delta modulation is a 1-bit quantization method.

The output of a delta modulator is a single-bit data stream where the relative number of 1s and 0s indicates the level or amplitude of the input signal. The number of 1s over a given number of clock cycles establishes the signal amplitude during that interval. A maximum number of 1s corresponds to the maximum positive input voltage. A number of 1s equal to one-half the

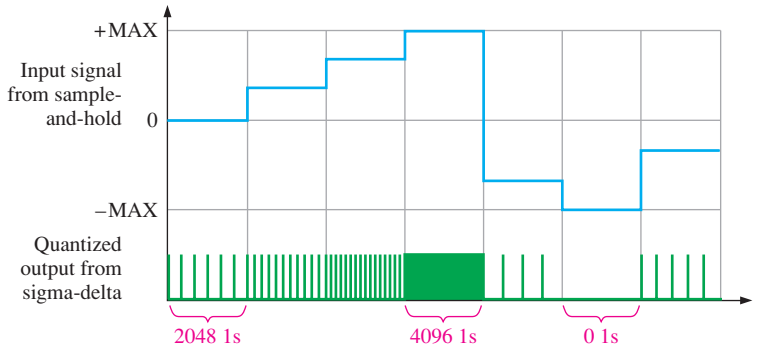


FIGURE 12–20 A simplified illustration of sigma-delta analog-to-digital conversion.

maximum corresponds to an input voltage of zero. No 1s (all 0s) corresponds to the maximum negative input voltage. This is illustrated in a simplified way in Figure 12–20. For example, assume that 4096 1s occur during the interval when the input signal is a positive maximum. Since zero is the midpoint of the dynamic range of the input signal, 2048 1s occur during the interval when the input signal is zero. There are no 1s during the interval when the input signal is a negative maximum. For signal levels in between, the number of 1s is proportional to the level.

The Sigma-Delta ADC Functional Block Diagram

The basic block diagram in Figure 12–21 accomplishes the conversion illustrated in Figure 12–20. The analog input signal and the analog signal from the converted quantized bit stream from the DAC in the feedback loop are applied to the summation (Σ) point. The difference (Δ) signal out of the Σ is integrated, and the 1-bit ADC increases or decreases the number of 1s depending on the difference signal. This action attempts to keep the quantized signal that is fed back equal to the incoming analog signal. The 1-bit quantizer is essentially a comparator followed by a latch.

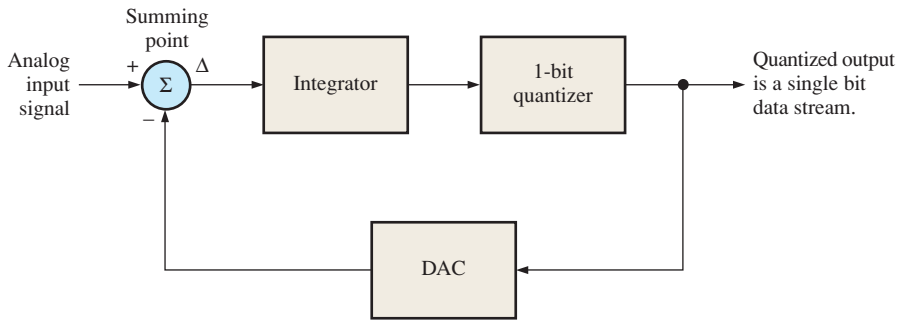


FIGURE 12–21 Partial functional block diagram of a sigma-delta ADC.

To complete the sigma-delta conversion process using one particular approach, the single bit data stream is converted to a series of binary codes, as shown in Figure 12–22. The counter counts the 1s in the quantized data stream for successive intervals. The code in the counter then represents the amplitude of the analog input signal for each interval. These codes are shifted out into the latch for temporary storage. What comes out of the latch is a series of *n*-bit codes, which completely represent the analog signal.

Testing Analog-to-Digital Converters

One method for testing ADCs is shown in Figure 12–23. A DAC is used as part of the test setup to convert the ADC output back to analog form for comparison with the test input.

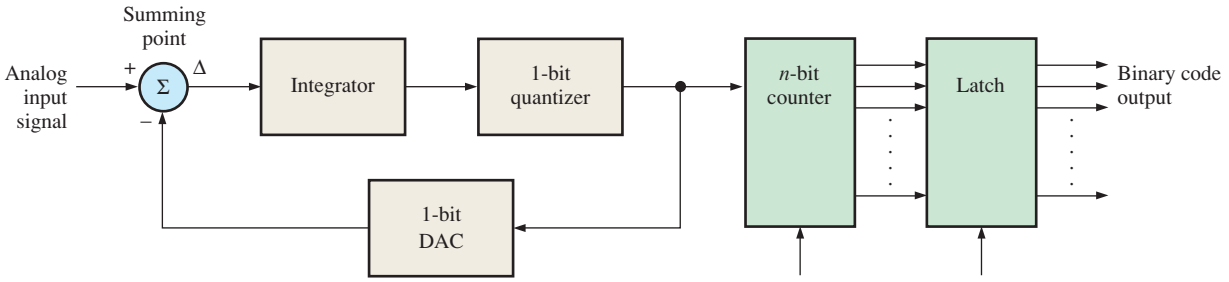


FIGURE 12–22 One type of sigma-delta ADC.

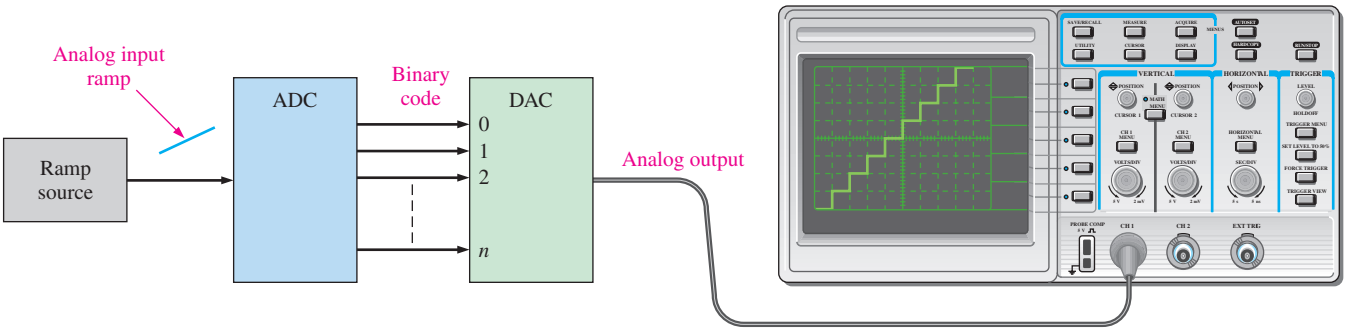


FIGURE 12–23 A method for testing ADCs.

A test input in the form of a linear ramp is applied to the input of the ADC. The resulting binary output sequence is then applied to the DAC test unit and converted to a stairstep ramp. The input and output ramps are compared for any deviation.

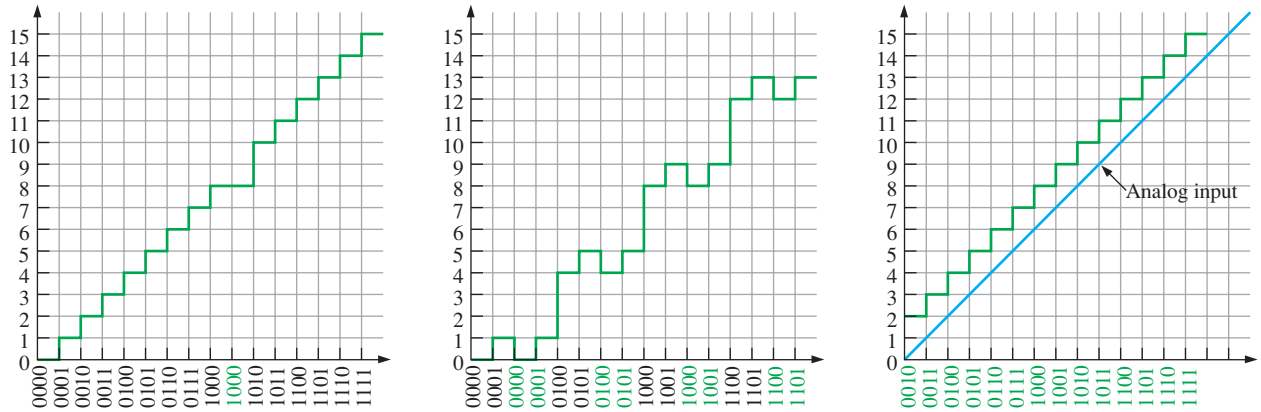
Analog-to-Digital Conversion Errors

Again, a 4-bit conversion is used to illustrate the principles. Let's assume that the test input is an ideal linear ramp.

Missing Code

The stairstep output in Figure 12–24(a) indicates that the binary code 1001 does not appear on the output of the ADC. Notice that the 1000 value stays for two intervals and then the output jumps to the 1010 value.

In a flash ADC, for example, a failure of one of the op-amp comparators can cause a missing-code error.



(a) Missing code (green) (b) Incorrect codes (green) (c) Offset

FIGURE 12–24 Illustrations of analog-to-digital conversion errors.

Incorrect Code

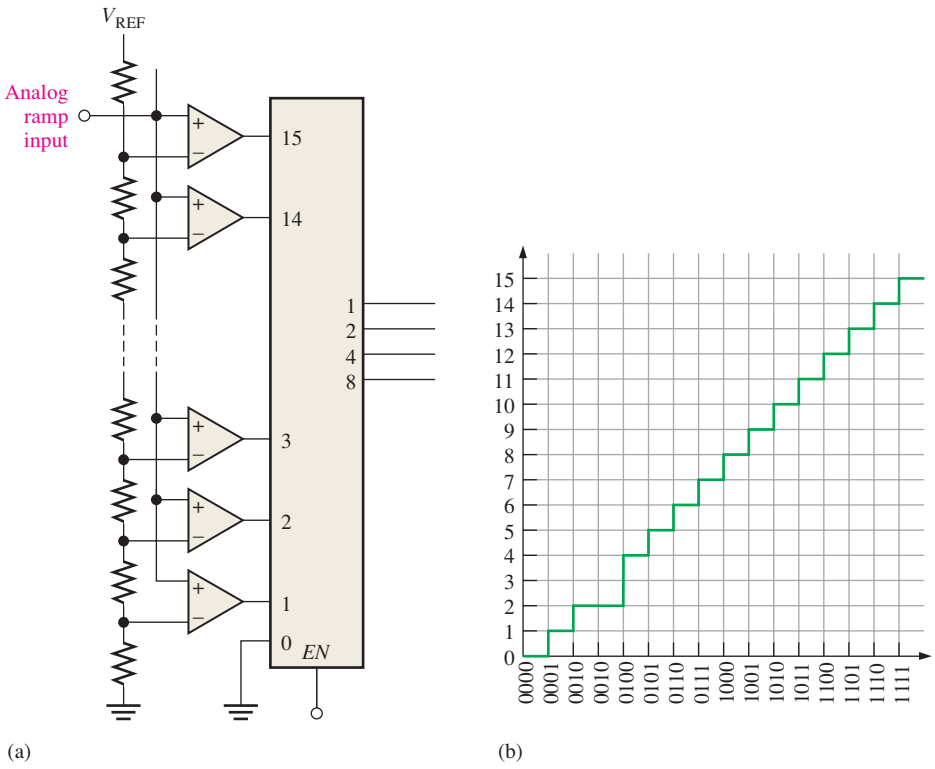
The stairstep output in Figure 12–24(b) indicates that several of the binary code words coming out of the ADC are incorrect. Analysis indicates that the 2^1-bit line is stuck in the LOW (0) state in this particular case.

Offset

Offset conditions are shown in 12–24(c). In this situation the ADC interprets the analog input voltage as greater than its actual value.

EXAMPLE 12–2

A 4-bit flash ADC is shown in Figure 12–25(a). It is tested with a setup like the one in Figure 12–23. The resulting reconstructed analog output is shown in Figure 12–25(b). Identify the problem and the most probable fault.



(a) (b)

FIGURE 12–25

Solution

The binary code 0011 is missing from the ADC output, as indicated by the missing step. Most likely, the output of comparator 3 is stuck in its inactive state (LOW).

Related Problem

Reconstruct the analog output in a test setup like in Figure 12–23 if the ADC in Figure 12–25(a) has comparator 8 stuck in the HIGH output state.

SECTION 12–2 CHECKUP

1. What is the fastest method of analog-to-digital conversion?
2. Which analog-to-digital conversion method produces a single-bit data stream?
3. Does the successive-approximation converter have a fixed conversion time?
4. Name two types of output errors in an ADC.

12–3 Methods of Digital-to-Analog Conversion

Digital-to-analog conversion is an important part of a digital processing system. Once the digital data has been processed, it is converted back to analog form. In this section, we will examine the theory of operation of two basic types of digital-to-analog converters (DACs) and learn about their performance characteristics.

After completing this section, you should be able to

- Explain the operation of a binary-weighted-input DAC
- Explain the operation of an *R*/2*R* ladder DAC
- Discuss resolution, accuracy, linearity, monotonicity, and settling time in a DAC
- Discuss the testing of DACs for nonmonotonicity, differential nonlinearity, low or high gain, and offset error

Binary-Weighted-Input Digital-to-Analog Converter

One method of digital-to-analog conversion uses a resistor network with resistance values that represent the binary weights of the input bits of the digital code. Figure 12–26 shows a 4-bit DAC of this type. Each of the input resistors will either have current or have no current, depending on the input voltage level. If the input voltage is zero (binary 0), the current is also zero. If the input voltage is HIGH (binary 1), the amount of current depends on the input resistor value and is different for each input resistor, as indicated in the figure.

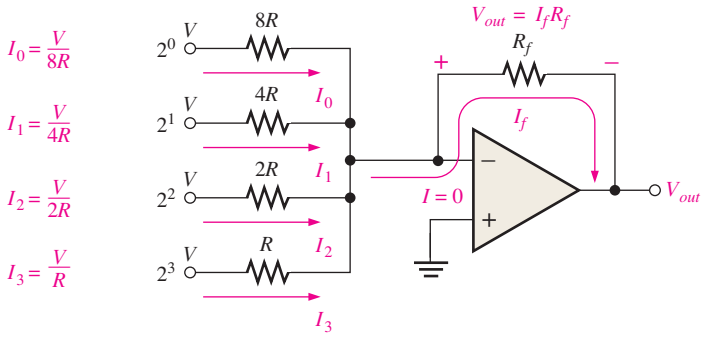


FIGURE 12–26 A 4-bit DAC with binary-weighted inputs.

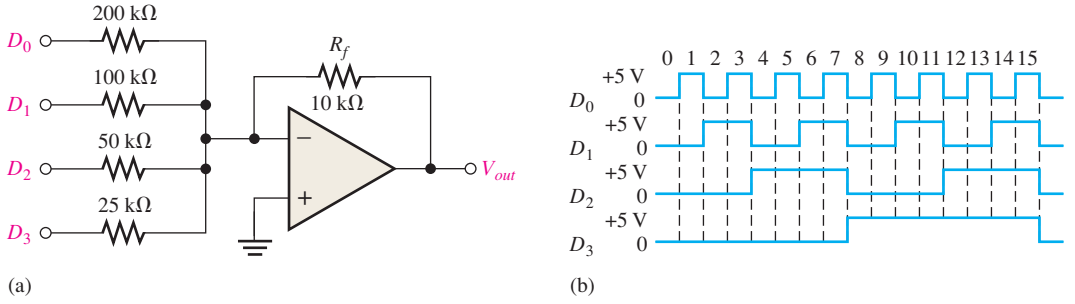
Since there is practically no current into the op-amp inverting (−) input, all of the input currents sum together and go through R_f. Since the inverting input is at 0 V (virtual ground), the drop across R_f is equal to the output voltage, so $V_{out} = I_f R_f$.

The values of the input resistors are chosen to be inversely proportional to the binary weights of the corresponding input bits. The lowest-value resistor (R) corresponds to the highest binary-weighted input (2^3). The other resistors are multiples of R (that is, $2R$, $4R$, and $8R$) and correspond to the binary weights 2^2, 2^1, and 2^0, respectively. The input currents are also proportional to the binary weights. Thus, the output voltage is proportional to the sum of the binary weights because the sum of the input currents is through R_f.

Disadvantages of this type of DAC are the number of different resistor values and the fact that the voltage levels must be exactly the same for all inputs. For example, an 8-bit converter requires eight resistors, ranging from some value of R to $128R$ in binary-weighted steps. This range of resistors requires tolerances of one part in 255 (less than 0.5%) to accurately convert the input, making this type of DAC very difficult to mass-produce.

EXAMPLE 12–3

Determine the output of the DAC in Figure 12–27(a) if the waveforms representing a sequence of 4-bit numbers in Figure 12–27(b) are applied to the inputs. Input D_0 is the least significant bit (LSB).



(a) (b)

FIGURE 12–27

Solution

First, determine the current for each of the weighted inputs. Since the inverting (−) input of the op-amp is at 0 V (virtual ground) and a binary 1 corresponds to +5 V, the current through any of the input resistors is 5 V divided by the resistance value.

$$I_0 = \frac{5\text{ V}}{200\text{ k}\Omega} = 0.025\text{ mA}$$

$$I_1 = \frac{5\text{ V}}{100\text{ k}\Omega} = 0.05\text{ mA}$$

$$I_2 = \frac{5\text{ V}}{50\text{ k}\Omega} = 0.1\text{ mA}$$

$$I_3 = \frac{5\text{ V}}{25\text{ k}\Omega} = 0.2\text{ mA}$$

Almost no current goes into the inverting op-amp input because of its extremely high impedance. Therefore, assume that all of the current goes through the feedback resistor R_f. Since one end of R_f is at 0 V (virtual ground), the drop across R_f equals the output voltage, which is negative with respect to virtual ground.

$$V_{out(D0)} = (10\text{ k}\Omega)(-0.025\text{ mA}) = -0.25\text{ V}$$
$$V_{out(D1)} = (10\text{ k}\Omega)(-0.05\text{ mA}) = -0.5\text{ V}$$
$$V_{out(D2)} = (10\text{ k}\Omega)(-0.1\text{ mA}) = -1\text{ V}$$
$$V_{out(D3)} = (10\text{ k}\Omega)(-0.2\text{ mA}) = -2\text{ V}$$

From Figure 12–27(b), the first binary input code is 0000, which produces an output voltage of 0 V. The next input code is 0001, which produces an output voltage of −0.25 V. The next code is 0010, which produces an output voltage of −0.5 V. The next code is 0011, which produces an output voltage of −0.25 V + −0.5 V = −0.75 V. Each successive binary code increases the output voltage by −0.25 V, so for this particular straight binary sequence on the inputs, the output is a stairstep waveform going from 0 V to −3.75 V in −0.25 V steps. This is shown in Figure 12–28.

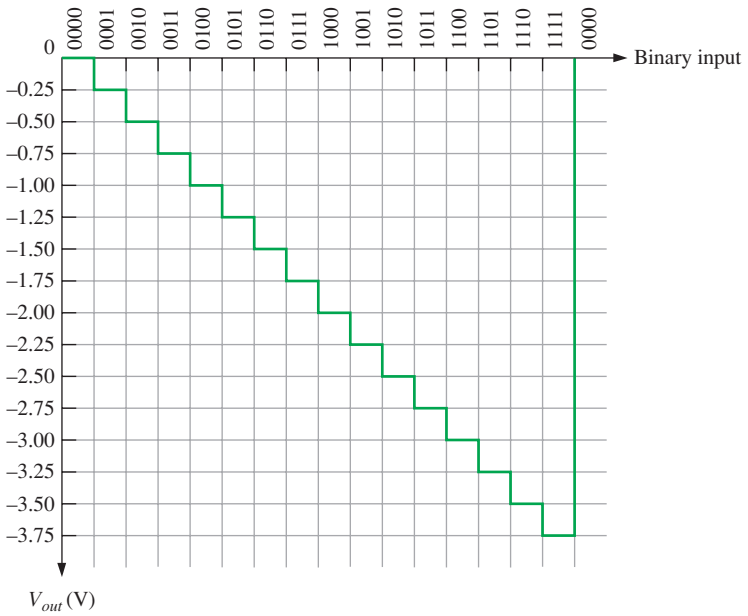


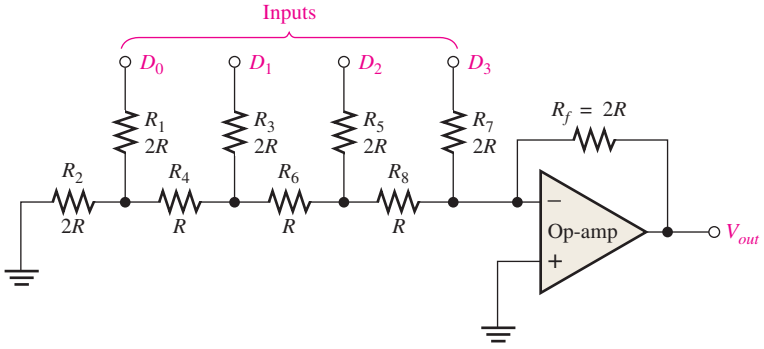
FIGURE 12–28 Output of the DAC in Figure 12–27.

Related Problem

Reverse the input waveforms to the DAC in Figure 12–27 (D_3 to D_0, D_2 to D_1, D_1 to D_2, D_0 to D_3) and determine the output.

The *R*/2*R* Ladder Digital-to-Analog Converter

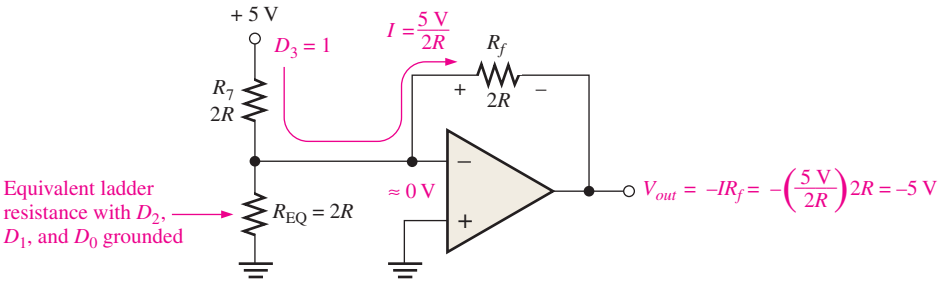
Another method of digital-to-analog conversion is the *R*/2*R* ladder, as shown in Figure 12–29 for four bits. It overcomes one of the problems in the binary-weighted-input DAC in that it requires only two resistor values.



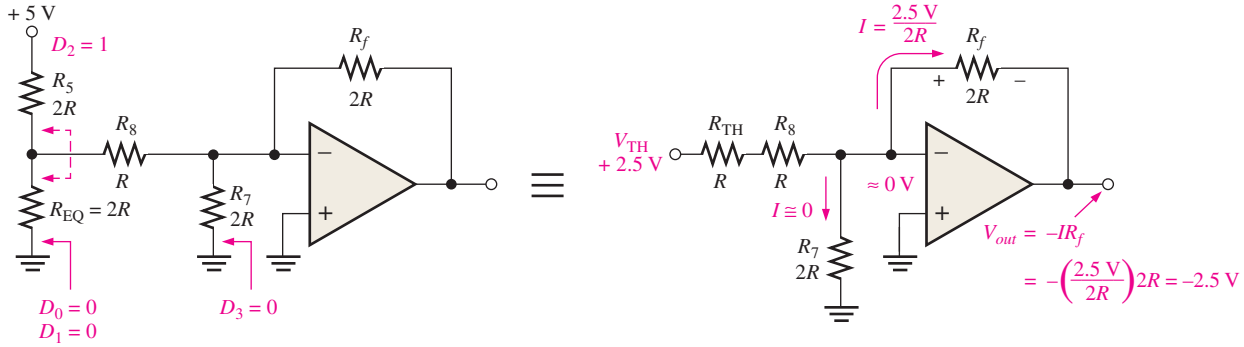
fg12_02900

FIGURE 12–29 An *R*/2*R* ladder DAC.

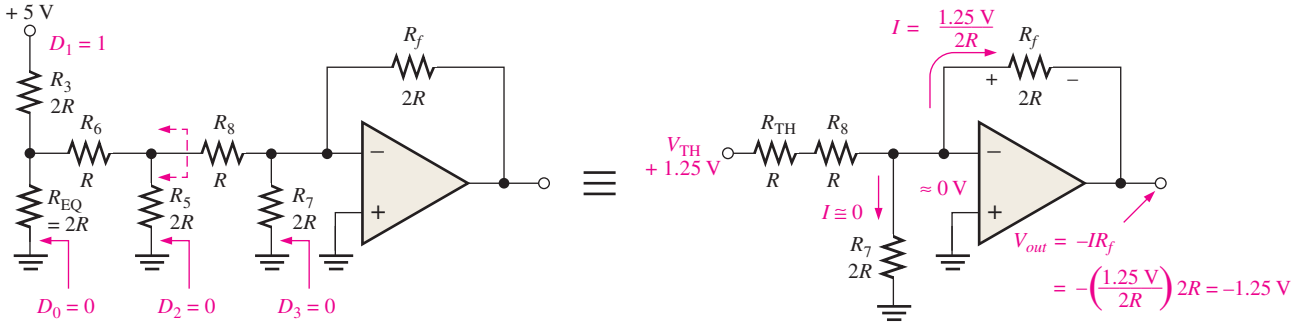
Start by assuming that the D_3 input is HIGH (+5 V) and the others are LOW (ground, 0 V). This condition represents the binary number 1000. A circuit analysis will show that this reduces to the equivalent form shown in Figure 12–30(a). Essentially no current goes



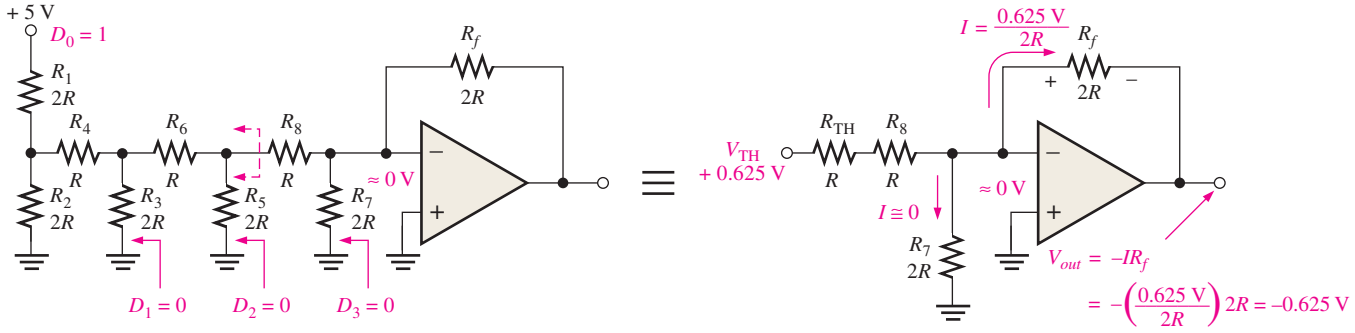
(a) Equivalent circuit for $D_3 = 1$, $D_2 = 0$, $D_1 = 0$, $D_0 = 0$



(b) Equivalent circuit for $D_3 = 0$, $D_2 = 1$, $D_1 = 0$, $D_0 = 0$



(c) Equivalent circuit for $D_3 = 0$, $D_2 = 0$, $D_1 = 1$, $D_0 = 0$



(d) Equivalent circuit for $D_3 = 0$, $D_2 = 0$, $D_1 = 0$, $D_0 = 1$

FIGURE 12–30 Analysis of the *R*/2*R* ladder DAC.

through the $2R$ equivalent resistance because the inverting input is at virtual ground. Thus, all of the current ($I = 5\text{ V}/2R$) through R_7 also goes through R_f, and the output voltage is -5 V. The operational amplifier keeps the inverting ($-$) input near zero volts ($\approx$0 V) because of negative feedback. Therefore, all current goes through R_f rather than into the inverting input.

Figure 12–30(b) shows the equivalent circuit when the D_2 input is at +5 V and the others are at ground. This condition represents 0100. If we thevenize* looking from R_8, we get 2.5 V in series with R, as shown. This results in a current through R_f of $I = 2.5\text{ V}/2R$, which gives an output voltage of −2.5 V. Keep in mind that there is no current into the op-amp inverting input and that there is no current through the equivalent resistance to ground because it has 0 V across it, due to the virtual ground.

Figure 12–30(c) shows the equivalent circuit when the D_1 input is at +5 V and the others are at ground. This condition represents 0010. Again thevenizing looking from R_8, you get 1.25 V in series with R as shown. This results in a current through R_f of $I = 1.25\text{ V}/2R$, which gives an output voltage of −1.25 V.

In part (d) of Figure 12–30, the equivalent circuit representing the case where D_0 is at +5 V and the other inputs are at ground is shown. This condition represents 0001. Thevenizing from R_8 gives an equivalent of 0.625 V in series with R as shown. The resulting current through R_f is $I = 0.625\text{ V}/2R$, which gives an output voltage of −0.625 V.

Notice that each successively lower-weighted input produces an output voltage that is halved, so that the output voltage is proportional to the binary weight of the input bits.

Performance Characteristics of Digital-to-Analog Converters

The performance characteristics of a DAC include resolution, accuracy, linearity, monotonicity, and settling time, each of which is discussed in the following list:

- *Resolution.* The resolution of a DAC is the reciprocal of the number of discrete steps in the output. This, of course, is dependent on the number of input bits. For example, a 4-bit DAC has a resolution of one part in $2^4 - 1$ (one part in fifteen). Expressed as a percentage, this is $(1/15)100 = 6.67\%$. The total number of discrete steps equals $2^n - 1$, where n is the number of bits. Resolution can also be expressed as the number of bits that are converted.
- *Accuracy.* Accuracy is derived from a comparison of the actual output of a DAC with the expected output. It is expressed as a percentage of a full-scale, or maximum, output voltage. For example, if a converter has a full-scale output of 10 V and the accuracy is ±0.1%, then the maximum error for any output voltage is $(10\text{ V})(0.001) = 10\text{ mV}$. Ideally, the accuracy should be no worse than ±1/2 of a least significant bit. For an 8-bit converter, the least significant bit is 0.39% of full scale. The accuracy should be approximately ±0.2%.
- *Linearity.* A linear error is a deviation from the ideal straight-line output of a DAC. A special case is an offset error, which is the amount of output voltage when the input bits are all zeros.
- *Monotonicity.* A DAC is **monotonic** if it does not take any reverse steps when it is sequenced over its entire range of input bits.
- *Settling time.* Settling time is normally defined as the time it takes a DAC to settle within ±1/2 LSB of its final value when a change occurs in the input code.

EXAMPLE 12–4

Determine the resolution, expressed as a percentage, of the following:

(a) an 8-bit DAC

(b) a 12-bit DAC

*Thevenin's theorem states that any circuit can be reduced to an equivalent voltage source in series with an equivalent resistance.

Solution

(a) For the 8-bit converter,

$$\frac{1}{2^8 - 1} \times 100 = \frac{1}{255} \times 100 = \mathbf{0.392\%}$$

(b) For the 12-bit converter,

$$\frac{1}{2^{12} - 1} \times 100 = \frac{1}{4095} \times 100 = \mathbf{0.0244\%}$$

Related Problem

Calculate the resolution for a 16-bit DAC.

Testing Digital-to-Analog Converters

The concept of DAC testing is illustrated in Figure 12–31. In this basic method, a sequence of binary codes is applied to the inputs, and the resulting output is observed. The binary code sequence extends over the full range of values from 0 to $2^n - 1$ in ascending order, where n is the number of bits.

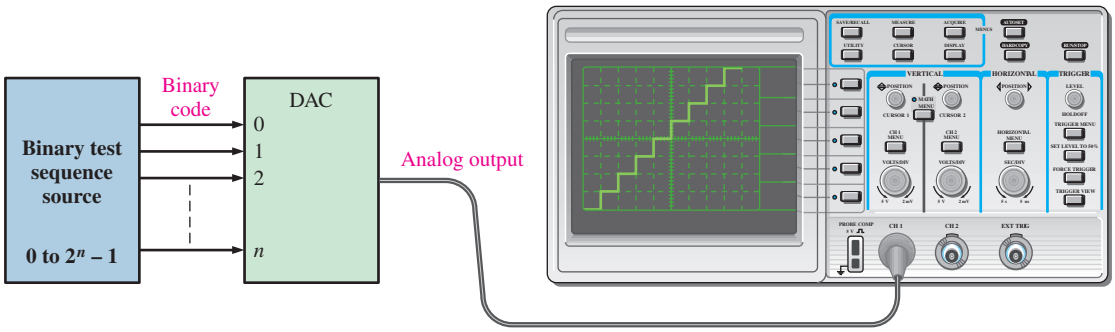


FIGURE 12–31 Basic test setup for a DAC.

The ideal output is a straight-line stairstep as indicated. As the number of bits in the binary code is increased, the resolution is improved. That is, the number of discrete steps increases, and the output approaches a straight-line linear ramp.

Digital-to-Analog Conversion Errors

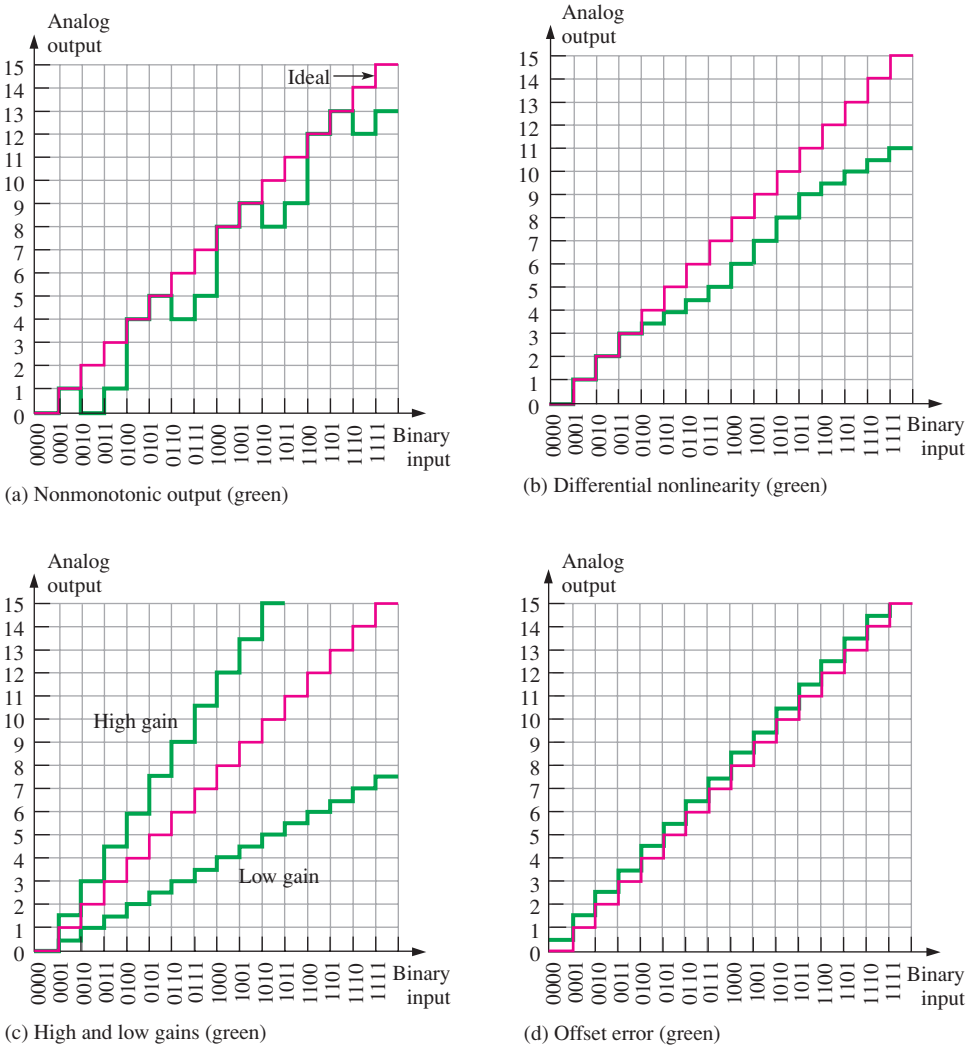
Several digital-to-analog conversion errors to be checked for are shown in Figure 12–32, which uses a 4-bit conversion for illustration purposes. A 4-bit conversion produces fifteen discrete steps. Each graph in the figure includes an ideal stairstep ramp for comparison with the faulty outputs.

Nonmonotonicity

The step reversals in Figure 12–32(a) indicate nonmonotonic performance, which is a form of nonlinearity. In this particular case, the error occurs because the 2^1 bit in the binary code is interpreted as a constant 0. That is, a short is causing the bit input line to be stuck LOW.

Differential Nonlinearity

Figure 12–32(b) illustrates differential nonlinearity in which the step amplitude is less than it should be for certain input codes. This particular output could be caused by the 2^2 bit having an insufficient weight, perhaps because of a faulty input resistor. We could also see steps with amplitudes greater than normal if a particular binary weight were greater than it should be.



(a) Nonmonotonic output (green)

(b) Differential nonlinearity (green)

(c) High and low gains (green)

(d) Offset error (green)

FIGURE 12–32 Illustrations of several digital-to-analog conversion errors.

Low or High Gain

Output errors caused by low or high gain are illustrated in Figure 12–32(c). In the case of low gain, all of the step amplitudes are less than ideal. In the case of high gain, all of the step amplitudes are greater than ideal. This situation may be caused by a faulty feedback resistor in the op-amp circuit.

Offset Error

An offset error is illustrated in Figure 12–32(d). Notice that when the binary input is 0000, the output voltage is nonzero and that this amount of offset is the same for all steps in the conversion. A faulty op-amp may be the culprit in this situation.

EXAMPLE 12–5

The DAC output in Figure 12–33 is observed when a straight 4-bit binary sequence is applied to the inputs. Identify the type of error, and suggest an approach to isolate the fault.

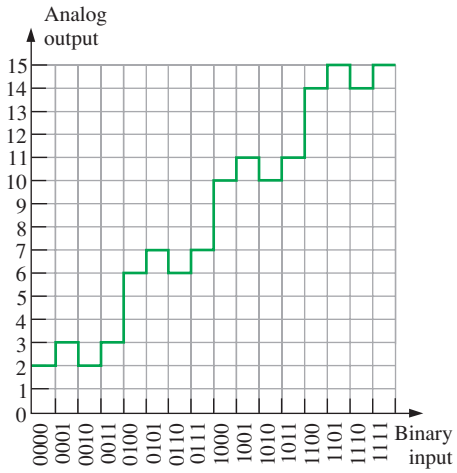


FIGURE 12–33

Solution

The DAC in this case is nonmonotonic. Analysis of the output reveals that the device is converting the following sequence, rather than the actual binary sequence applied to the inputs.

0010, 0011, 0010, 0011, 0110, 0111, 0110, 0111, 1010, 1011, 1010, 1011, 1110, 1111,1110, 1111

Apparently, the 2^1 bit is stuck in the HIGH (1) state. To find the problem, first monitor the bit input pin to the device. If it is changing states, the fault is internal to the DAC and it should be replaced. If the external pin is not changing states and is always HIGH, check for an external short to $+V$ that may be caused by a solder bridge somewhere on the circuit board.

Related Problem

Determine the output of a DAC when a straight 4-bit binary sequence is applied to the inputs and the 2^0 bit is stuck HIGH.

The Reconstruction Filter

The output of the DAC is a "stairstep" approximation of the original analog signal after it has been processed by the **digital signal processor (DSP)**, which is a special type of microprosessor that processes data in real time. The purpose of the low-pass reconstruction filter (sometimes called a postfilter) is to smooth out the DAC output by eliminating the higher frequency content that results from the fast transitions of the "stairsteps," as roughly illustrated in Figure 12–34.

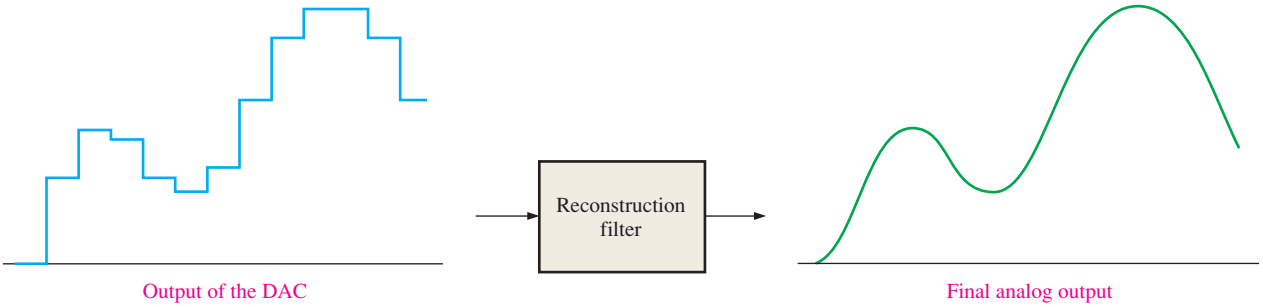


FIGURE 12–34 The reconstruction filter smooths the output of the DAC.

SECTION 12–3 CHECKUP

1. What is the disadvantage of the DAC with binary-weighted inputs?
2. What is the resolution of a 4-bit DAC?
3. How do you detect nonmonotonic behavior in a DAC?
4. What effect does low gain have on a DAC output?

12–4 Digital Signal Processing

Digital signal processing converts signals that naturally occur in analog form, such as sound, video, and information from sensors, to digital form and uses digital techniques to enhance and modify analog signal data for various applications.

After completing this section, you should be able to

- Discuss digital signal processing
- Draw a basic block diagram of a digital signal processing system

A digital signal processing system first translates a continuously varying analog signal into a series of discrete levels. This series of levels follows the variations of the analog signal and resembles a staircase, as illustrated for the case of a sine wave in Figure 12–35. The process of changing the original analog signal to a "stairstep" approximation is accomplished by a sample-and-hold circuit.

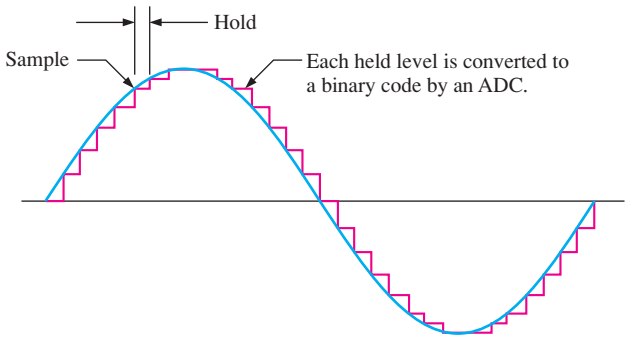


FIGURE 12–35 An original analog signal (sine wave) and its "stairstep" approximation.

Next, the "stairstep" approximation is quantized into binary codes that represent each discrete step on the "stairsteps" by a process called analog-to-digital (A/D) conversion. The circuit that performs A/D conversion is an analog-to-digital converter (ADC).

Once the analog signal has been converted to a binary coded form, it is applied to a DSP (digital signal processor). The DSP can perform various operations on the incoming data, such as removing unwanted interference, increasing the amplitude of some signal frequencies and reducing others, encoding the data for secure transmissions, and detecting and correcting errors in transmitted codes. DSPs make possible, among many other things, the cleanup of sound recordings, the removal of echos from communications lines, the enhancement of images from CT scans for better medical diagnosis, and the scrambling of cellular phone conversations for privacy.

After a DSP processes a signal, the signal can be converted back to an enhanced version of the original analog signal. This is accomplished by a digital-to-analog converter (DAC). Figure 12–36 shows a basic block diagram of a typical digital signal processing system.

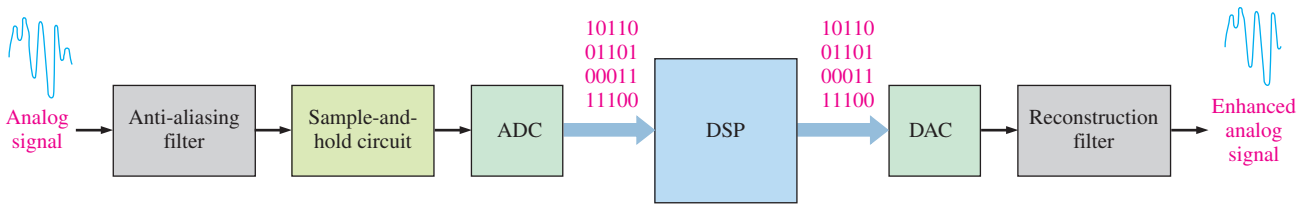


FIGURE 12–36 Basic block diagram of a typical digital signal processing system.

DSPs are actually a specialized type of microprocessor but are different from general-purpose microprocessors in a couple of significant ways. Typically, microprocessors are designed for general-purpose functions and operate with large software packages. DSPs are used for special-purpose applications; they are very fast number crunchers that must work in real time by processing information as it happens using specialized algorithms (programs). The analog-to-digital converter (ADC) in a system must take samples of the incoming analog data often enough to catch all the relevant fluctuations in the signal amplitude, and the DSP must keep pace with the sampling rate of the ADC by doing its calculations as fast as the sampled data are received. Once the digital data are processed by the DSP, they go to the digital-to-analog converter (DAC) and reconstruction filter for conversion back to analog form.

SECTION 12–4 CHECKUP

1. What does DSP stand for?
2. What does ADC stand for?
3. What does DAC stand for?
4. An analog signal is changed to a binary coded form by what circuit?
5. A binary coded signal is changed to analog form by what circuit?

12–5 The Digital Signal Processor (DSP)

Essentially, a digital signal processor (DSP) is a special type of microprocessor that processes data in real time. Its applications focus on the processing of digital data that represents analog signals. A DSP, like a microprocessor, has a central processing unit (CPU) and memory units in addition to many interfacing functions. Every time you use your cellular telephone, you are using a DSP, and this is only one example of its many applications.

After completing this chapter, you should be able to

- Explain the basic concepts of a DSP
- List some of the applications of DSPs
- Describe the basic functions of a DSP in a cell phone
- Discuss the TMS320C6000 series DSP

The digital signal processor (DSP) is the heart of a digital signal processing system. It takes its input from an ADC and produces an output that goes to a DAC, as shown in Figure 12–37. As you have learned, the ADC changes an analog waveform into data in the form of a series of binary codes that are then applied to the DSP for processing. After being processed by the DSP, the data go to a DAC for conversion back to analog form.

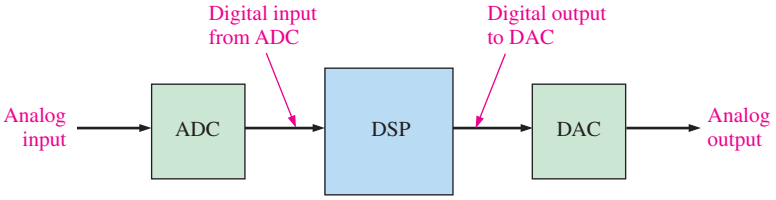


FIGURE 12–37 The DSP has a digital input and produces a digital output.

DSP Programming

DSPs are typically programmed in either assembly language or in C. Because programs written in assembly language can usually execute faster and because speed is critical in most DSP applications, assembly language is used much more in DSPs than in general-purpose microprocessors. Also, DSP programs are usually much shorter than traditional microprocessor programs because of their very specialized applications where much redundancy is used. In general, the instruction sets for DSPs tend to be smaller than for microprocessors.

DSP Applications

The DSP, unlike the general-purpose microprocessor, must typically process data in *real time;* that is, as it happens. Many applications in which DSPs are used cannot tolerate any noticeable delays, requiring the DSP to be extremely fast. In addition to cell phones, digital signal processors (DSPs) are used in multimedia computers, video recorders, CD players, hard disk drives, digital radio modems, and other applications to improve the signal quality. DSPs are also used in television applications. For example, television converters use DSP to provide compatibility with various television standards.

An important application of DSPs is in signal compression and decompression. In CD systems, for example, the music on the CD is in a compressed form so that it doesn't use as much storage space. It must be decompressed in order to be reproduced. Also signal compression is used in cell phones to allow a greater number of calls to be handled simultaneously in a local cell.

Telecommunications

The field of telecommunications involves transferring all types of information from one location to another, including telephone conversations, television signals, and digital data. Among other functions, the DSP facilitates multiplexing many signals onto one transmission channel because information in digital form is relatively easy to multiplex and demultiplex.

At the transmitting end of a telecommunications system, DSPs are used to compress digitized voice signals for conservation of bandwidth. Compression is the process of reducing the data rate. Generally, a voice signal is converted to digital form at 8000 samples per second (sps), based on a Nyquist frequency of 4 kHz. If 8 bits are used to encode each sample, the data rate is 64 kbps. In general, reducing (compressing) the data rate from 64 kbps to 32 kbps results in no loss of sound quality. When the data are compressed to 8 kbps, the sound quality is reduced noticeably. When compressed to the minimum of 2 kbps, the sound is greatly distorted but still usable for some applications where only word recognition and not quality is important. At the receiving end of a telecommunications system, the DSP decompresses the data to restore the signal to its original form.

Echoes, a problem in many long distance telephone connections, occur when a portion of a voice signal is returned with a delay. For shorter distances, this delay is barely noticeable; but as the distance between the transmitter and the receiver increases, so does the delay time of the echo. DSPs are used to effectively cancel the annoying echo, which results in a clear, undisturbed voice signal.

InfoNote

Sound cards used in computers use an ADC to convert sound from a microphone, audio CD player, or other source into a digital signal. The ADC sends the digital signal to a digital signal processor (DSP). Based on instructions from a ROM, one function of the DSP is to compress the digital signal so it uses less storage space. The DSP then sends the compressed data to the computer's processor which, in turn, sends the data to a hard drive or CD ROM for storage. To play a recorded sound, the stored data is retrieved by the processor and sent to the DSP where it is decompressed and sent to a DAC. The output of the DAC, which is a reproduction of the original sound signal, is applied to the speakers.

Music Processing

The DSP is used in the music industry to provide filtering, signal addition and subtraction, and signal editing in music preparation and recording. Also, another application of the DSP is to add artificial echo and reverberation, which are usually minimized by the acoustics of a sound studio, in order to simulate ideal listening environments from concert halls to small rooms.

Speech Generation and Recognition

DSPs are used in speech generation and recognition to enhance the quality of man/machine communication. The most common method used to produce computer-generated speech is digital recording. In digital recording, the human voice is digitized and stored, usually in a compressed form. During playback the stored voice data are uncompressed and converted back into the original analog form. Approximately an hour of speech can be stored using about 3 MB of memory.

Speech recognition is much more difficult to accomplish than speech generation. The DSP is used to isolate and analyze each word in the incoming voice signal. Certain parameters are identified in each word and compared with previous examples of the spoken word to create the closest match. Most systems are limited to a few hundred words at best. Also, significant pauses between words are usually required and the system must be "trained" for a given individual's voice. Speech recognition is an area of tremendous research effort and will eventually be applied in many commercial applications.

Radar

In *ra*dio *d*etection *a*nd *r*anging (radar) applications, DSPs provide more accurate determination of distance using data compression techniques, decrease noise using filtering techniques, thereby increasing the range, and optimize the ability of the radar system to identify specific types of targets. DSPs are also used in similar ways in sonar systems.

Image Processing

The DSP is used in image-processing applications such as the computed tomography (CT) and magnetic resonance imaging (MRI), which are widely used in the medical field for looking inside the human body. In CT, X-rays are passed through a section of the body from many directions. The resulting signals are converted to digital form and stored. This stored information is used to produce calculated images that appear to be slices through the human body that show great detail and permit better diagnosis.

Instead of X-rays, MRI uses magnetic fields in conjunction with radio waves to probe inside the human body. MRI produces images, just as CT, and provides excellent discrimination between different types of tissue as well as information such as blood flow through arteries. MRI depends entirely on digital signal processing methods.

In applications such as video telephones, digital television, and other media that provide moving pictures, the DSP uses image compression to reduce the number of bits needed, making these systems commercially feasible.

Filtering

DSPs are commonly used to implement digital filters for the purposes of separating signals that have been combined with other signals or with interference and noise and for restoring signals that are distorted. Although analog filters are quite adequate for some applications, the digital filter is generally much superior in terms of the performance that can be achieved. One drawback to digital filters is that the execute time required produces a delay from the time the analog signal is applied until the time the output appears. Analog filters present no delay problems because as soon as the input occurs, the response appears on the output. Analog filters are also less expensive than digital filters. Regardless of this, the overall performance of the digital filter is far superior in many applications.

The DSP in a Cellular Telephone

The digital cellular telephone is an example of how a DSP can be used. Figure 12–38 shows a simplified block diagram of a digital cell phone. The voice **codec** (codec is the abbreviation for coder/decoder) contains, among other functions, the ADC and DAC necessary to convert between the analog voice signal and a digital voice format. Sigma-delta conversion is typically used in most cell phone applications. For transmission, the voice signal from the microphone is converted to digital form by the ADC in the codec and then it goes to the DSP for processing. From the DSP, the digital signal goes to the rf (radio frequency) section where it is modulated and changed to the radio frequency for transmission. An incoming rf signal containing voice data is picked up by the antenna, demodulated, and changed to a digital signal. It is then applied to the DSP for processing, after which the digital signal goes to the codec for conversion back to the original voice signal by the DAC. It is then amplified and applied to the speaker.

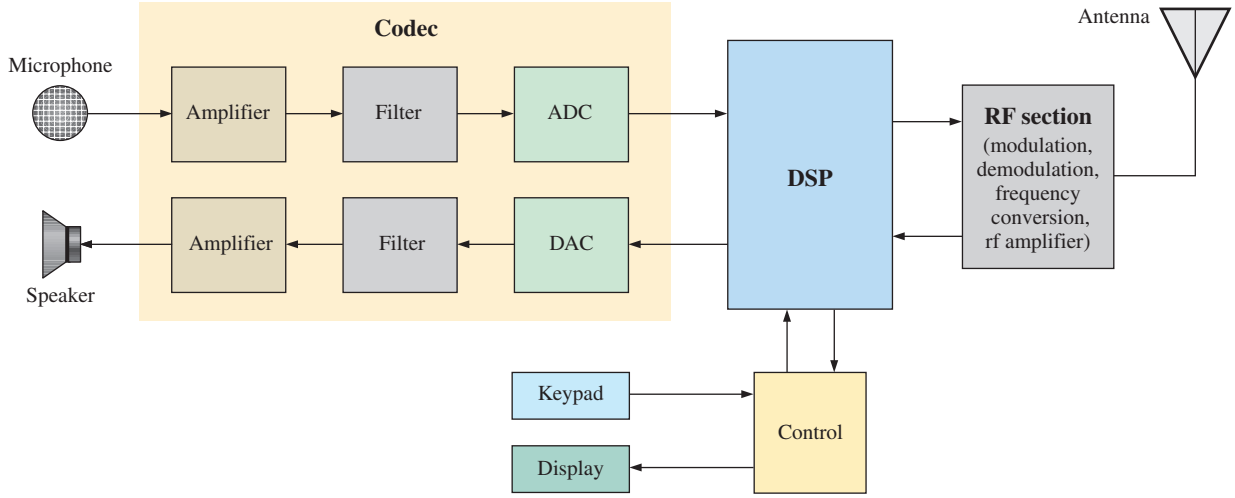


FIGURE 12–38 Simplified block diagram of a digital cellular phone.

Functions Performed by the DSP

In a cellular phone application, the DSP performs many functions to improve and facilitate the reception and transmission of a voice signal. Some of these DSP functions are as follows:

- *Speech compression.* The rate of the digital voice signal is reduced significantly for transmission in order to meet the bandwidth requirements.
- *Speech decompression.* The rate of the received digital voice signal is returned to its original rate in order to properly reproduce the analog voice signal.
- *Protocol handling.* The cell phone communicates with the nearest base in order to establish the location of the cell phone, allocates time and frequency slots, and arranges handover to another base station as the phone moves into another cell.
- *Error detection and correction.* During transmission, error detection and correction codes are generated and, during reception, detect and correct errors induced in the rf channel by noise or interference.
- *Encryption.* Converts the digital voice signal to a form for secure transmission and converts it back to original form during reception.

Basic DSP Architecture

As mentioned before, a DSP is basically a specialized microprocessor optimized for speed in order to process data in real time. Many DSPs are based on what is known as the *Harvard architecture,* which consists of a central processing unit (CPU) and two memories, one for data and the other for the program, as shown by the block diagram in Figure 12–39.

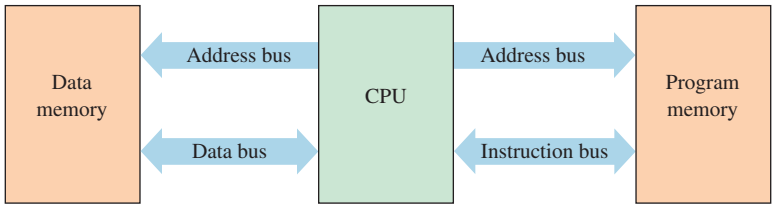


FIGURE 12–39 Many DSPs use the Harvard architecture (two memories).

A Specific DSP

DSPs are manufactured by several companies including Texas Instruments, Motorola, and Analog Devices. DSPs are available for both fixed-point and floating-point processing. Recall from Chapter 2 that these two methods differ in the way numbers are stored and manipulated. All floating-point DSPs can also handle numbers in fixed-point format. Fixed-point DSPs are less expensive than the floating-point versions and, generally, can operate faster. The details of DSP architecture can vary significantly, even within the same family. Let's look briefly at one particular DSP series as an example of how a DSP is generally organized.

Examples of DSPs available in the TMS320C6000 series include the TMS320C62xx, the TMS320C64xx, and the TMS320C67xx, which are part of Texas Instrument's TMS320 family of devices. A general block diagram for these devices is shown in Figure 12–40.

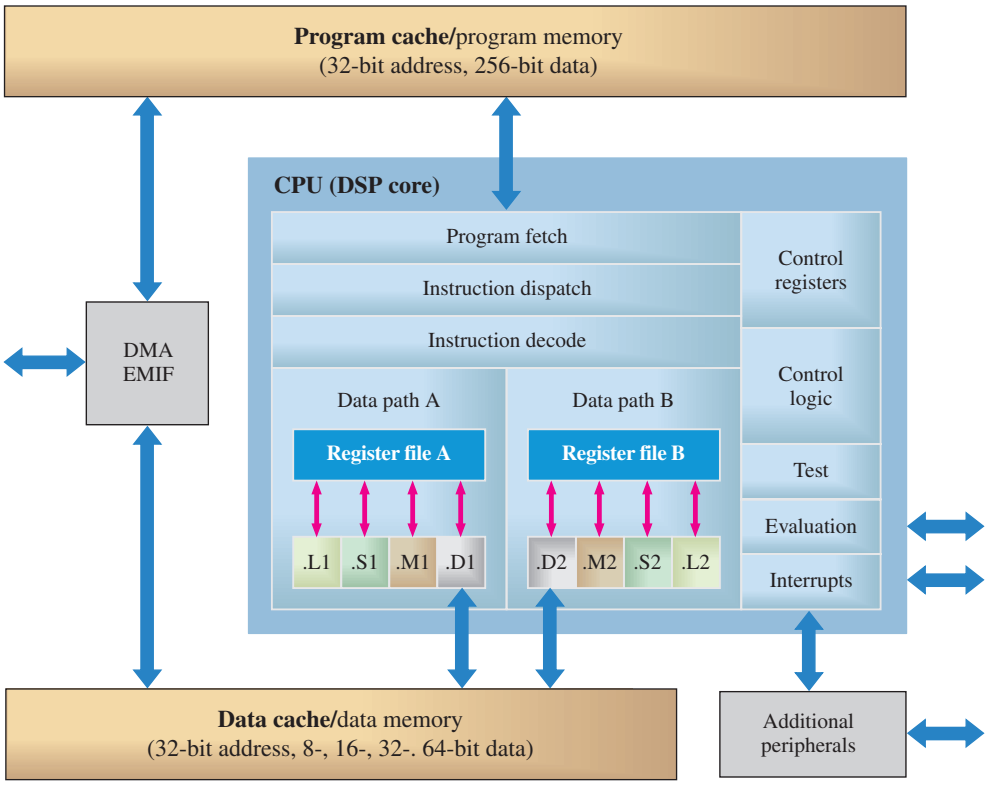


FIGURE 12–40 General block diagram of the TMS320C6000 series DSP.

The DSPs have a central processing unit (CPU), also known as the **DSP core,** that contains 64 general-purpose 32-bit registers in the C64xx and 32 general-purpose 32-bit registers in the C62xx and the C67xx. The C67xx can handle floating-point operations, whereas the C62xx and C64xx are fixed-point devices.

Each DSP has eight functional units that contain two 16-bit multipliers and six arithmetic logic units (ALUs). The performance of the three DSPs in the C6000 series in terms of

TABLE 12–3

TMS320C6000 series DSP data processing performance.

DSP	Type	Application	Processing Speed	Multiply/ Accumulate Speed
C62xx	Fixed-point	General-purpose	1200–2400 MIPS	300–600 MMACS
C64xx	Fixed-point	Special-purpose	3200–4800 MIPS	1600–2400 MMACS
C67xx	Floating-point	General-purpose	600–1000 MFLOPS	200–333 MMACS

MIPS (Million Instructions Per Second), **MFLOPS** (Million Floating-point Operations Per Second), and **MMACS** (Million Multiply/Accumulates per Second) is shown in Table 12–3.

Data Paths in the CPU

In the CPU, the program fetch, instruction dispatch, and instruction decode sections can provide eight 32-bit instructions to the functional units during every clock cycle. The CPU is split into two data paths, and instruction processing occurs in both data paths A and B. Each data path contains half of the general-purpose registers (16 in the C62xx and C67xx or 32 in the C64xx) and four functional units. The control register and logic are used to configure and control the various processor operations.

Functional Units

Each data path has four functional units. The M units (labeled .M1 and .M2 in Figure 12–40) are dedicated multipliers. The L units (labeled .L1 and .L2) perform arithmetic, logic, and miscellaneous operations. The S units (labeled .S1 and .S2) perform compare, shift, and miscellaneous arithmetic operations. The D units (labeled .D1 and .D2) perform load, store, and miscellaneous operations.

Pipeline

A **pipeline** allows multiple instructions to be processed simultaneously. A pipeline operation consists of three stages through which all instructions flow: *fetch, decode, execute.* Eight instructions at a time are first fetched from the program memory; they are then decoded, and finally they are executed.

During **fetch,** the eight instructions (called a packet) are taken from memory in four *phases,* as shown in Figure 12–41.

- *Program address generate (PG).* The program address is generated by the CPU.
- *Program address send (PS).* The program address is sent to the memory.
- *Program access ready wait (PW).* A memory read operation occurs.
- *Program fetch packet receive (PR).* The CPU receives the packet of instructions.

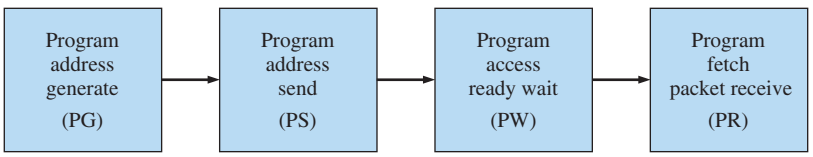


FIGURE 12–41 The four fetch phases of the pipeline operation.

Two phases make up the instruction **decode** stage of pipeline operation, as shown in Figure 12–42. The instruction dispatch (DP) phase is where the instruction packets are split into execute packets and assigned to the appropriate functional units. The instruction decode (DC) phase is where the instructions are decoded.

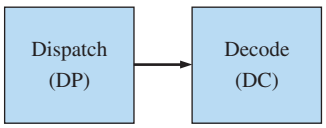


FIGURE 12–42 The two decode phases of the pipeline operation.

The **execute** stage of the pipeline operation is where the instructions from the decode stage are carried out. The execute stage has a maximum of five phases (E1 through E5), as shown in Figure 12–43. All instructions do not use all five phases. The number of phases used during execution depends on the type of instruction. Part of the execution of an instruction requires getting data from the data memory.

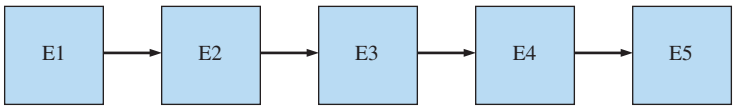


FIGURE 12–43 The five execute phases of pipeline operation.

Internal DSP Memory and Interfaces

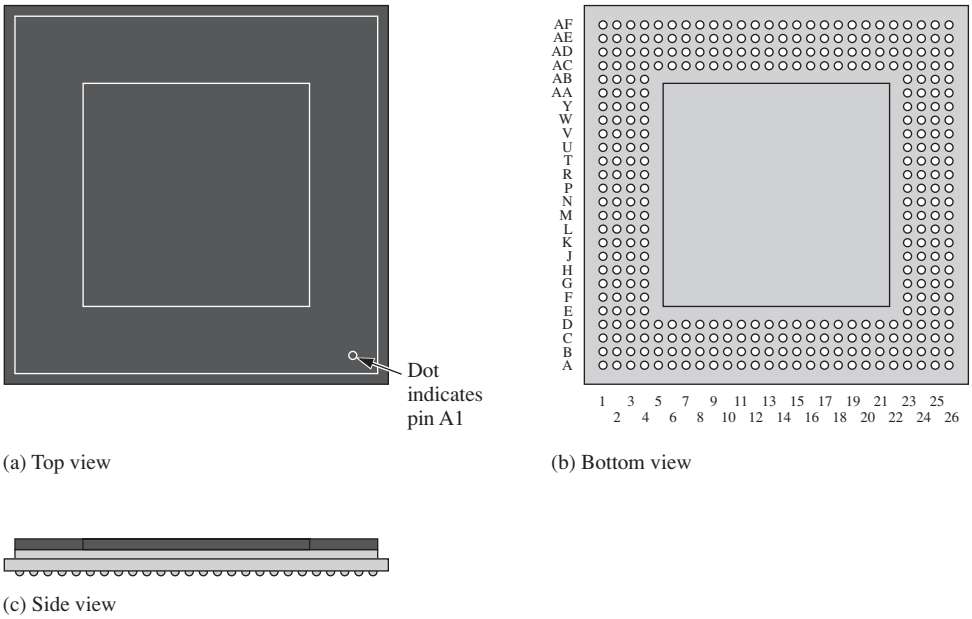
As you can see in Figure 12–40, there are two internal memories, one for data and one for program. The program memory is organized in 256 bit packets (eight 32-bit instructions) and there are 64 kB of capacity. The data memory also has a capacity of 64 kB and can be accessed in 8-, 16-, 32-, or 64-bit word lengths, depending on the specific device in the series. Both internal memories are accessed with a 32-bit address. The DMA (Direct Memory Access) is used to transfer data without going through the CPU. The EMIF (External Memory Interface) is used to support external memories when required in an application. Additional interface is provided for serial I/O ports and other external devices.

Timers

There are two general-purpose timers in the DSP that can be used for timed events, counting, pulse generation, CPU interrupts, and more.

Packaging

The TMS 3206000 series processors are available in 352-pin ball grid array (BGA) packages, as shown in Figure 12–44, and are implemented with CMOS technology.



(a) Top view

(b) Bottom view

(c) Side view

FIGURE 12–44 A 352-pin BGA package.

SECTION 12–5 CHECKUP

1. What is meant by the Harvard architecture?
2. What is a DSP core?
3. Name two categories of DSPs according to the type of numbers handled.
4. What are the two types of internal memory?
5. Define (a) MIPS (b) MFLOPS (c) MMACS.
6. Basically, what does pipelining accomplish?
7. Name the three stages of pipeline operation.
8. What happens during the fetch phase?

SUMMARY

- Sampling converts an analog signal into a series of impulses, each representing the signal amplitude at a given instant in time.
- The sampling theorem states that the sampling frequency must be at least twice the highest sampled frequency (Nyquist frequency).
- Analog-to-digital conversion changes an analog signal into a series of digital codes.
- Four types of analog-to-digital converters (ADCs) are flash (simultaneous), dual-slope, successive-approximation, and sigma-delta.
- Digital-to-analog conversion changes a series of digital codes that represent an analog signal back into the analog signal.
- Two types of digital-to-analog converters (DACs) are binary-weighted input and $R/2R$ ladder.
- Digital signal processing is the digital processing of analog signals, usually in real-time, for the purpose of modifying or enhancing the signal in some way.
- In general, a digital signal processing system consists of an anti-aliasing filter, a sample-and-hold circuit, an analog-to-digital converter, a DSP (digital signal processor), a digital-to-analog converter, and a reconstruction filter.
- A DSP is a specialized microprocessor optimized for speed in order to process data as it occurs (real-time).
- Most DSPs are based on the Harvard architecture, which means that there is a data memory and a program memory.
- A pipeline operation consists of fetch, decode, and execute stages.

KEY TERMS

Key terms and other bold terms in the chapter are defined in the end-of-book glossary.

Aliasing The effect created when a signal is sampled at less than twice the signal frequency. Aliasing creates unwanted frequencies that interfere with the signal frequency when the signal is recovered.

Analog-to-digital converter (ADC) A circuit used to convert an analog signal to digital form.

Decode A stage of the DSP pipeline operation in which instructions are assigned to functional units and are decoded.

Digital signal processor (DSP) A special type of microprocessor that processes data in real time.

Digital-to-analog converter (DAC) A circuit used to convert the digital representation of an analog signal back to the analog signal.

DSP core The central processing unit of a DSP.

Execute A stage of the DSP pipeline operation in which the decoded instructions are carried out.

Fetch A stage of the DSP pipeline operation in which an instruction is obtained from the program memory.

MFLOPS Million floating-point operations per second.

MIPS Million instructions per second.

MMACS Million multiply/accumulates per second.

Nyquist frequency The highest signal frequency that can be sampled at a specified sampling frequency; a frequency equal to or less than half the sampling frequency.

Pipeline Part of the DSP architecture that allows multiple instructions to be processed simultaneously.

Quantization The process whereby a binary code is assigned to each sampled value during analog-to-digital conversion.

Sampling The process of taking a sufficient number of discrete values at points on a waveform that will define the shape of the waveform.

TRUE/FALSE QUIZ

Answers are at the end of the chapter.

1. An analog signal can be converted to a digital signal using sampling.
2. An ADC is an analog data component.
3. Aliasing is a desired factor in sampling.
4. A higher sampling rate is more accurate than a lower sampling rate for a given analog signal.
5. MIPS stands for *memory instructions per second*.
6. Successful approximation is an analog-to-digital conversion method.
7. Delta modulation is based on the difference of two successive samples.
8. Two types of DAC are the binary-weighted input and the *R*/2*R* ladder.
9. The process of converting an analog value to a code is called *quantization*.
10. A flash ADC differs from a simultaneous ADC.

SELF-TEST

Answers are at the end of the chapter.

1. Which of following is not a type of ADC?
   (a) Flash ADC **(b)** Dual slope ADC
   (c) Recessive approximation ADC **(d)** sigma-delta ADC
2. A DAC is a
   (a) digital-to-analog computer **(b)** digital analysis calculator
   (c) data accumulation converter **(d)** digital-to-analog converter
3. Aliasing results in
   (a) oversampling
   (b) undersampling
   (c) guard-band formation
   (d) perfect sampling
4. According to the sampling theorem, the sampling frequency should be
   (a) less than half the highest signal frequency
   (b) greater than twice the highest signal frequency
   (c) less than half the lowest signal frequency
   (d) greater than the lowest signal frequency

5. An op-amp is a linear amplifier which has
(a) one input and one output **(b)** one input and two outputs
(c) two inputs and one output **(d)** two inputs and two outputs

6. The quantization process
(a) converts the sample-and-hold output to binary code
(b) converts a sample impulse to a level
(c) converts a sequence of binary codes to a reconstructed analog signal
(d) filters out unwanted frequencies before sampling takes place

7. Generally, an analog signal can be reconstructed more accurately with
(a) more quantization levels **(b)** fewer quantization levels
(c) a higher sampling frequency **(d)** a lower sampling frequency
(e) either answer (a) or (c)

8. The throughput of a flash ADC is measured in
(a) displacement per second **(b)** distance per second
(c) samples per minute **(d)** samples per second

9. A digital voltmeter uses a
(a) flash ADC **(b)** successive approximation ADC
(c) sigma-delta ADC **(d)** dual-slope ADC

10. The most common ADC seen in telecommunications based on audio signals is
(a) flash ADC
(b) successive approximation ADC
(c) sigma-delta ADC
(d) dual-slope ADC

11. In a binary weighted DAC, the lowest-value resistor corresponds to
(a) the highest binary weighted input
(b) the lowest binary weighted input
(c) the first input
(d) the last input

12. DSPs are typically programmed in
(a) assembly level languages
(b) the C programming language
(c) neither (a) nor (b)
(d) both (a) and (b)

13. A digital signal processing system usually operates in
(a) real time **(b)** imaginary time
(c) compressed time **(d)** computer time

14. The term *Harvard architecture* means
(a) a CPU and a main memory
(b) a CPU and two data memories
(c) a CPU, a program memory, and a data memory
(d) a CPU and two register files

15. The minimum number of general-purpose registers in the TMS320C6000 series DSPs is
(a) 32 **(b)** 64
(c) 16 **(d)** 8

16. The two internal memories in the TMS320C6000 series each have a capacity of
(a) 1 MB **(b)** 512 kB
(c) 64 kB **(d)** 32 kB

17. In the TMS320C6000 series pipeline operation, the number of instructions processed simultaneously is
(a) eight **(b)** four
(c) two **(d)** one

18. The stage of the pipeline operation in which instructions are retrieved from the memory is called
(a) execute **(b)** accumulate
(c) decode **(d)** fetch

PROBLEMS

Answers to odd-numbered problems are at the end of the book.

Section 12–1 Analog-to-Digital Conversion

1. The waveform shown in Figure 12–45 is applied to a sampling circuit and is sampled every 3 ms. Show the output of the sampling circuit. Assume a one-to-one voltage correspondence between the input and output.

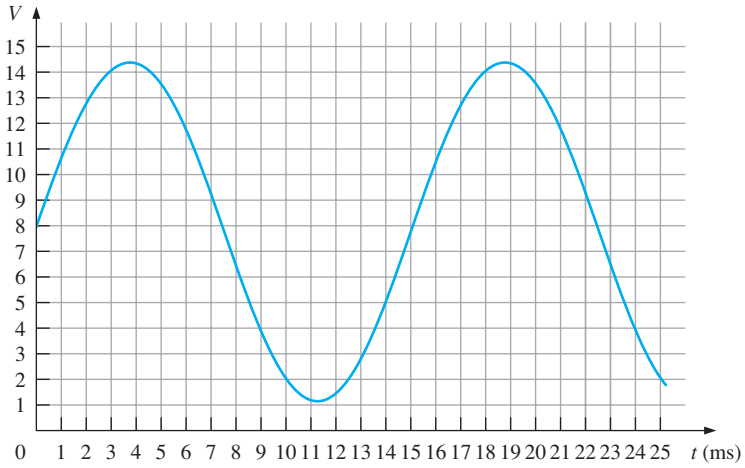


FIGURE 12–45

2. The output of the sampling circuit in Problem 1 is applied to a hold circuit. Show the output of the hold circuit.

3. If the output of the hold circuit in Problem 2 is quantized using two bits, what is the resulting sequence of binary codes?

4. Repeat Problem 3 using 4-bit quantization.

5. **(a)** Reconstruct the analog signal from the 2-bit quantization in Problem 3.

(b) Reconstruct the analog signal from the 4-bit quantization in Problem 4.

6. Graph the analog function represented by the following sequence of binary numbers: 1111, 1110, 1101, 1100, 1010, 1001, 1000, 0111, 0110, 0101, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1100, 1100, 1011, 1010, 1001.

Section 12–2 Methods of Analog-to-Digital Conversion

7. The input voltage to a certain op-amp inverting amplifier is 5 mV, and the output is 1 V. What is the closed-loop voltage gain?

8. To achieve a closed-loop voltage gain of 220 with an inverting amplifier, what value of feedback resistor do you use if $R_i = 2\ \text{k}\Omega$?

9. What is the gain of an inverting amplifier that uses a 33 kΩ feedback resistor if the input resistor is 1 kΩ?

10. How many comparators are required to form a 4-bit flash converter?

11. Determine the binary output code of a 3-bit flash ADC for the analog input signal in Figure 12–46.

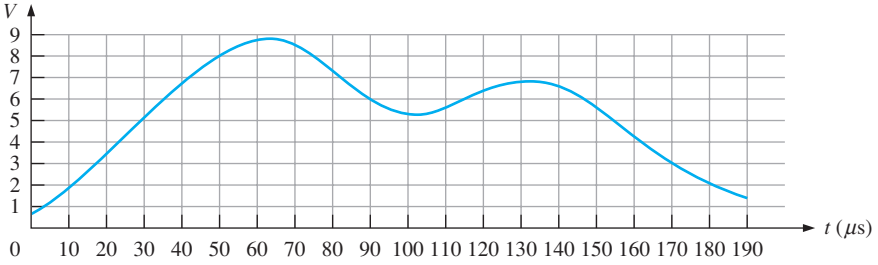


FIGURE 12–46

12. Repeat Problem 11 for the analog waveform in Figure 12–47.

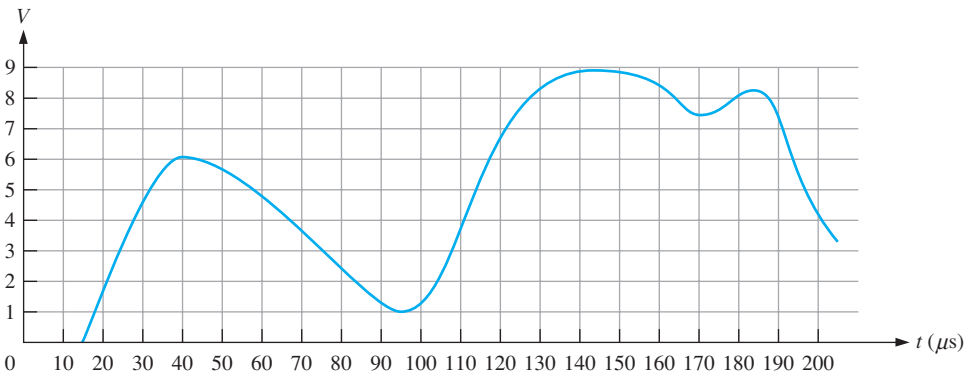


FIGURE 12–47

13. For a certain 2-bit successive-approximation ADC, the maximum ladder output is +8 V. If a constant +6 V is applied to the analog input, determine the sequence of binary states for the SAR.

14. Repeat Problem 13 for a 4-bit successive-approximation ADC.

15. An ADC produces the following sequence of binary numbers when an analog signal is applied to its input: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 0110, 0101, 0100, 0011, 0010, 0001, 0000.

(a) Reconstruct the input digitally.

(b) If the ADC failed so that the code 0111 were missing, what would the reconstructed output look like?

Section 12–3 Methods of Digital-to-Analog Conversion

16. In the 4-bit DAC in Figure 12–26, the lowest-weighted resistor has a value of 20 kΩ. What should the values of the other input resistors be?

17. Determine the output of the DAC in Figure 12–48(a) if the sequence of 4-bit numbers in part (b) is applied to the inputs. The data inputs have a low value of 0 V and a high value of +5 V.

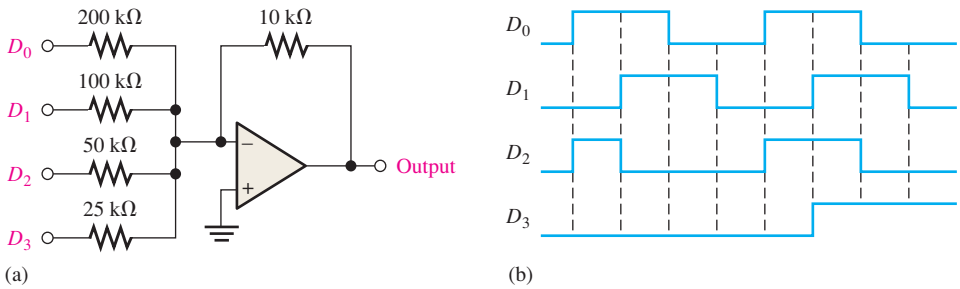


FIGURE 12–48

18. Repeat Problem 17 for the inputs in Figure 12–49.

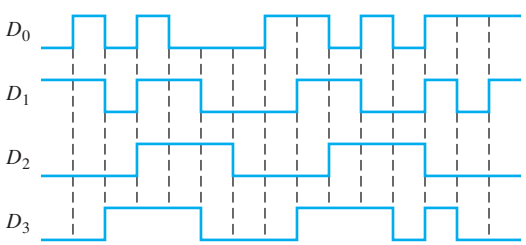


FIGURE 12–49

19. Determine the resolution expressed as a percentage, for each of the following DACs:

(a) 2-bit **(b)** 5-bit **(c)** 12-bit

20. Develop a circuit for generating an 8-bit binary test sequence for the test setup in Figure 12–31.

21. A 4-bit DAC has failed in such a way that the MSB is stuck in the 0 state. Draw the analog output when a straight binary sequence is applied to the inputs.

22. A straight binary sequence is applied to a 4-bit DAC, and the output in Figure 12–50 is observed. What is the problem?

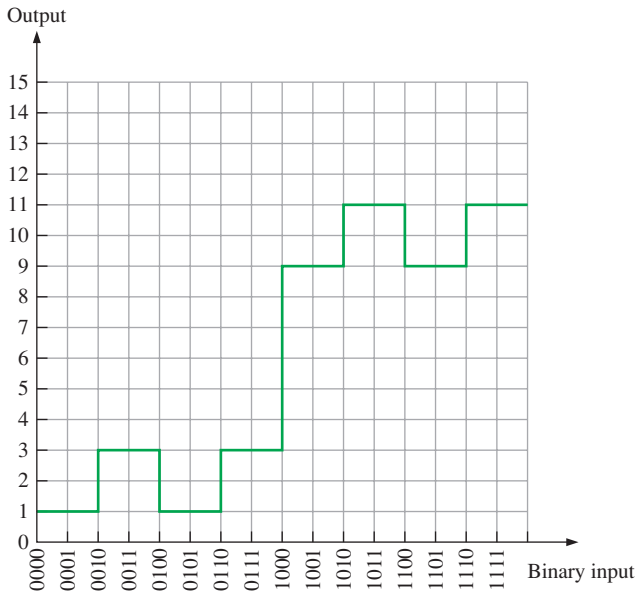


FIGURE 12–50

Section 12–4 Digital Signal Processing

23. How can an analog signal be converted into a stair-step approximation?

24. Fill in the appropriate functional names for the digital signal processing system block diagram in Figure 12–51.

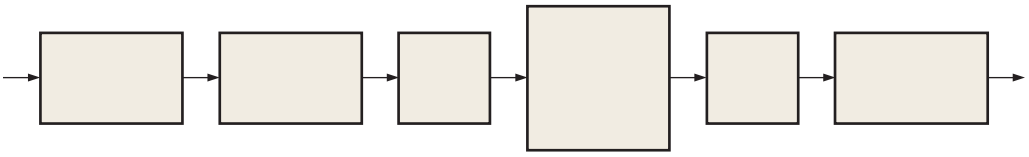


FIGURE 12–51

25. Explain the purpose of analog-to-digital conversion.

Section 12–5 The Digital Signal Processor (DSP)

26. A TMS320C62xx DSP has 32-bit instructions and is operating at 1800 MIPS. How many bytes per second is the DSP processing?

27. If the clock rate of a TMS320C64xx DSP is 600 MHz, how many instructions can it provide to the CPU functional units in one second?

28. How many floating-point operations can a DSP do in one second if it is specified at 2000 MFLOPS?

29. List and describe the four phases of the fetch operation in a TMS320C6000 series DSP.

30. List and describe the two phases of the decode operation in a TMS320C6000 series DSP.

ANSWERS

SECTION CHECKUPS

Section 12–1 Analog-to-Digital Conversion

1. Sampling is the process of converting an analog signal into a series of impulses, each representing the amplitude of the analog signal.

2. A sampled value is held to allow time to convert the value to a binary code.

3. The minimum sampling frequency is 40 kHz.

4. Quantization is the process of converting a sampled level to a binary code.

5. The number of bits determine quantization accuracy.

Section 12–2 Methods of Analog-to-Digital Conversion

1. The simultaneous (flash) method is fastest.

2. The sigma-delta method produces a single-bit data stream.

3. Yes, successive approximation has a fixed conversion time.

4. Missing code, incorrect code, and offset are types of ADC output errors.

Section 12–3 Methods of Digital-to-Analog Conversion

1. In a binary-weighted DAC, each resistor has a different value.

2. $(1/(2^4 - 1))100\% = 6.67\%$

3. A step reversal indicates nonmonotonic behavior in a DAC.

4. Step amplitudes in a DAC are less than ideal with low gain.

Section 12–4 Digital Signal Processing

1. DSP stands for digital signal processor.

2. ADC stands for analog-to-digital converter.

3. DAC stands for digital-to-analog converter.

4. The ADC changes an analog signal to binary coded form.

5. The DAC changes a binary coded signal to analog form.

Section 12–5 The Digital Signal Processor (DSP)

1. Harvard architecture means that there is a CPU and two memories, one for data and one for programs.

2. The DSP core is the CPU.

3. DSPs can be fixed-point or floating-point.

4. Internal memory types are data and program.

5. **(a)** MIPS—million instructions per second

(b) MFLOPS—million floating-point operations per second

(c) MMACS—million multiply/accumulates per second

6. Pipelining provides for the processing of multiple instructions simultaneously.

7. The stages of pipeline operation are fetch, decode, and execute.

8. During fetch, instructions are retrieved from the program memory.

RELATED PROBLEMS FOR EXAMPLES

12–1 100, 111, 100, 000, 011, 110. Yes, information is lost.

12–2 See Figure 12–52.

12–3 See Figure 12–53.

12–4 $(1/(2^{16} - 1))100\% = 0.00153\%$

12–5 See Figure 12–54.

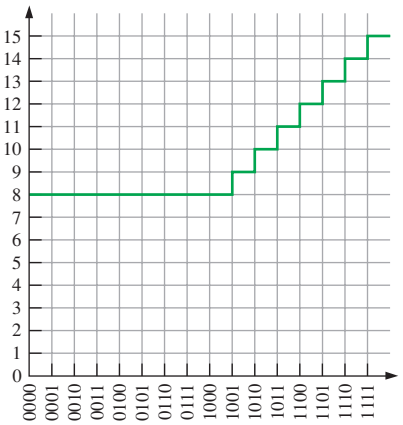


FIGURE 12–52

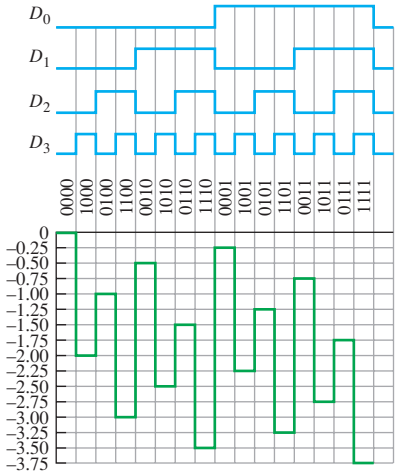


FIGURE 12–53

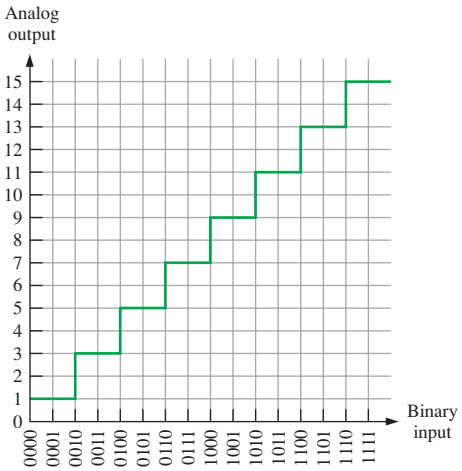


FIGURE 12–54

TRUE/FALSE QUIZ

1. T **2.** F **3.** F **4.** T **5.** F **6.** F **7.** T **8.** T **9.** T **10.** F

SELF-TEST

1. (c) **2.** (d) **3.** (b) **4.** (b) **5.** (c) **6.** (a)
7. (e) **8.** (d) **9.** (d) **10.** (c) **11.** (a) **12.** (d)
13. (a) **14.** (c) **15.** (a) **16.** (c) **17.** (a) **18.** (d)