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).

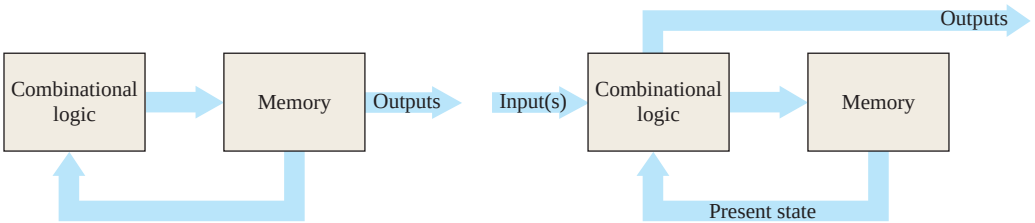
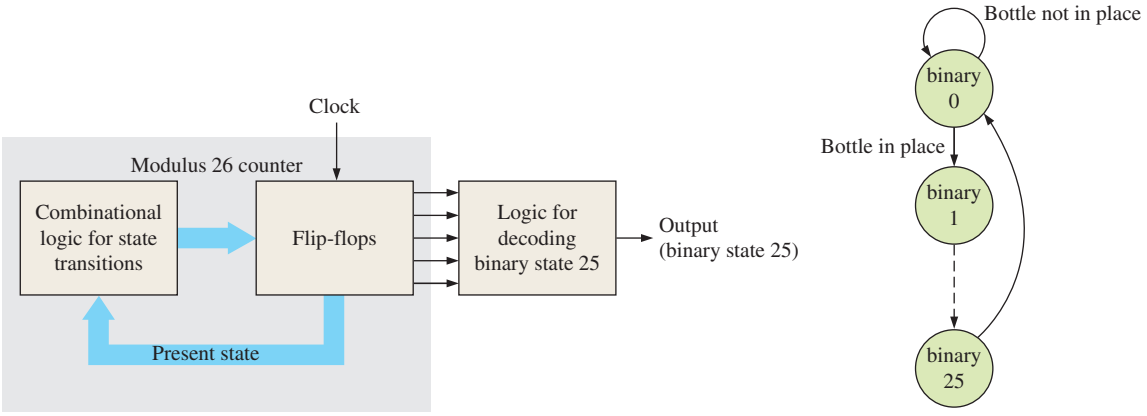


FIGURE 9–1 Two types of sequential logic.

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).

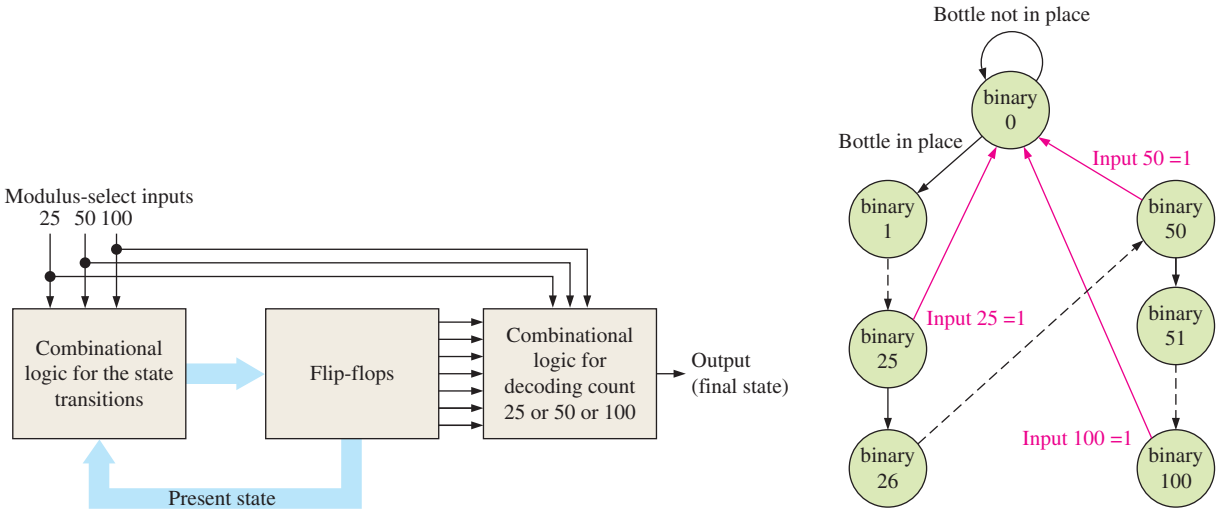


(a) Moore machine (b) State diagram

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.

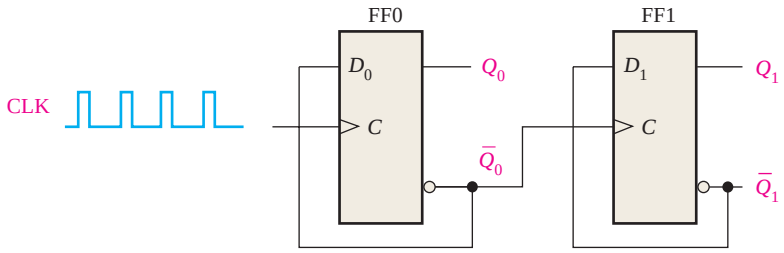
After completing this section, you should be able to

- 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

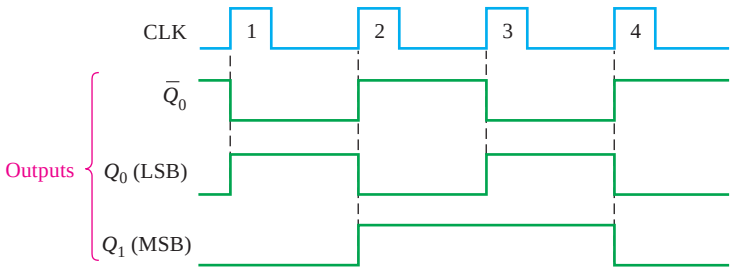


FIGURE 9–5 Timing diagram for the counter of Figure 9–4. As in previous chapters, output waveforms are shown in green.

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.

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.

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

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.

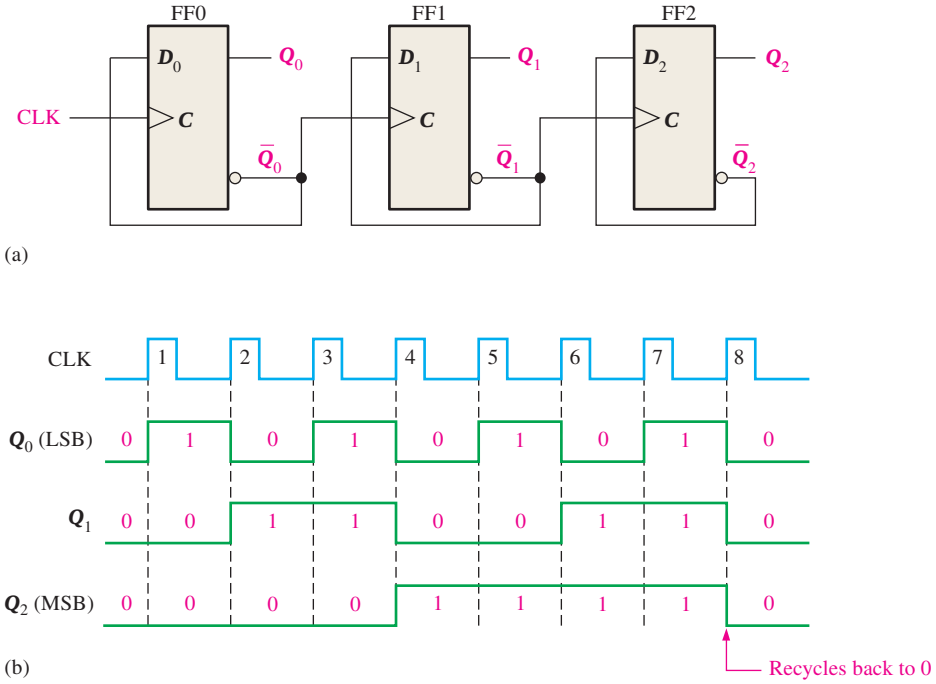
A 3-Bit Asynchronous Binary Counter

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

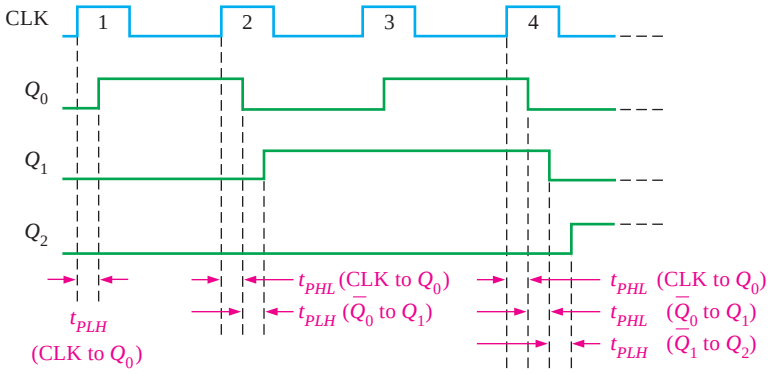


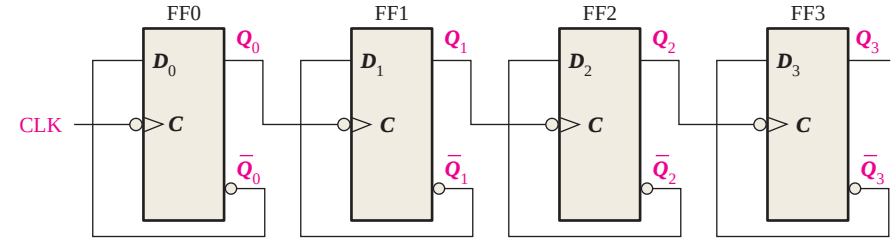
FIGURE 9–7 Propagation delays in a 3-bit asynchronous (ripple-clocked) binary counter.

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.

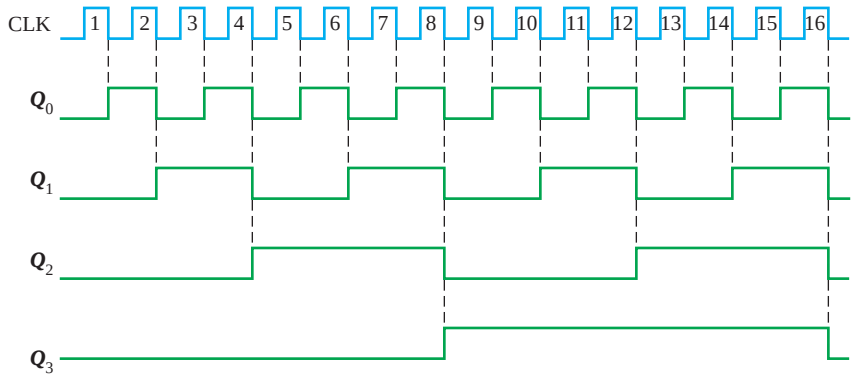
This cumulative delay of an asynchronous counter is a major disadvantage in many applications because it limits the rate at which the counter can be clocked and creates decoding problems. The maximum cumulative delay in a counter must be less than the period of the clock waveform.

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 $2^3 = 8$).

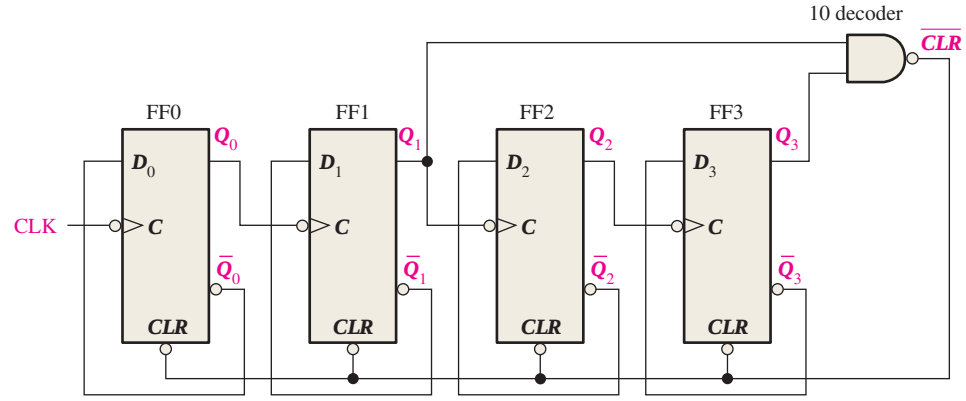
Let's use a 4-bit asynchronous counter such as the one in Example 9–1 and modify its sequence to illustrate the principle of truncated counters. One way to make the counter recycle after the count of nine (1001) is to decode count ten (1010) with a NAND gate and connect the output of the NAND gate to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

Partial Decoding

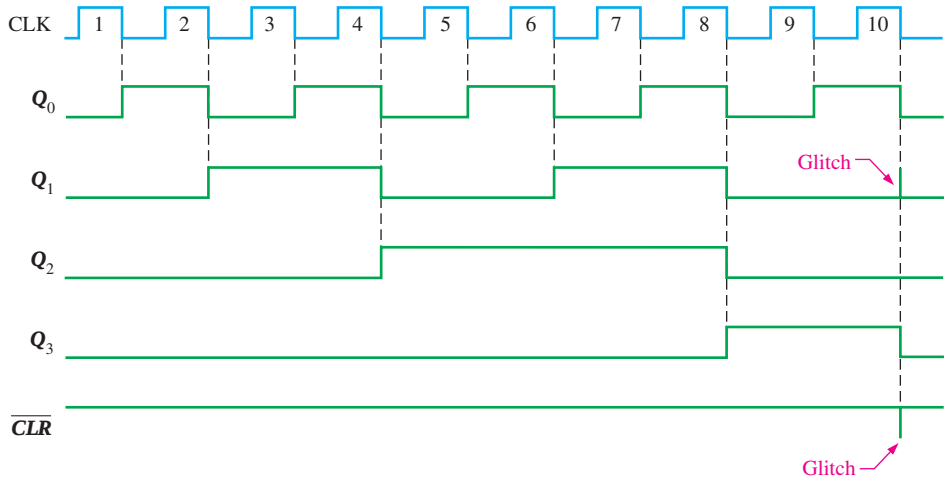
Notice in Figure 9–9(a) that only Q_1 and Q_3 are connected to the NAND gate inputs. This arrangement is an example of *partial decoding,* in which the two unique states ($Q_1 = 1$ and $Q_3 = 1$) are sufficient to decode the count of ten because none of the other states (zero through nine) have both 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

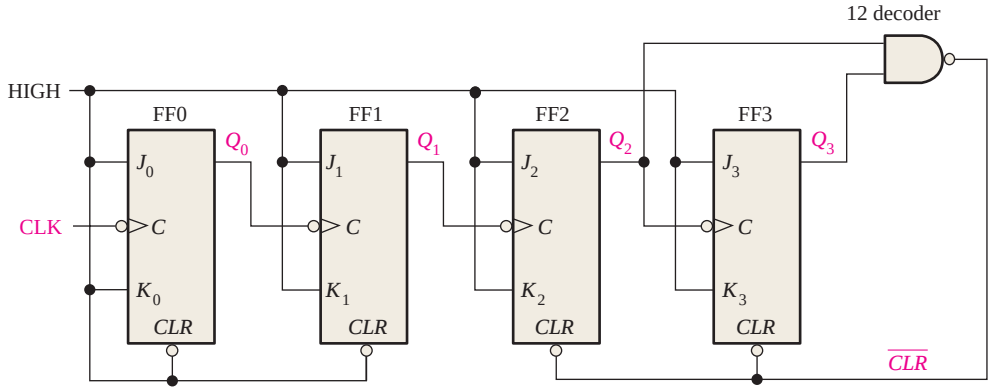
Since three flip-flops can produce a maximum of eight states, four flip-flops are required to produce any modulus greater than eight but less than or equal to sixteen.

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:

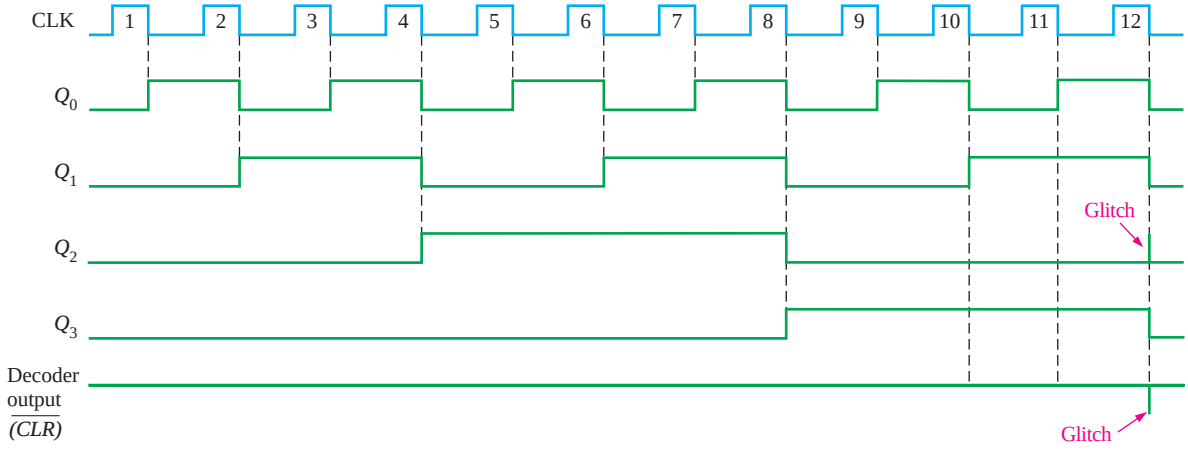
Q_3	Q_2	Q_1	Q_0	
0	0	0	0	←
.	.	.	.	
.	.	.	.	Recycles
.	.	.	.	
1	0	1	1	←
1	1	0	0	← Normal next state

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)



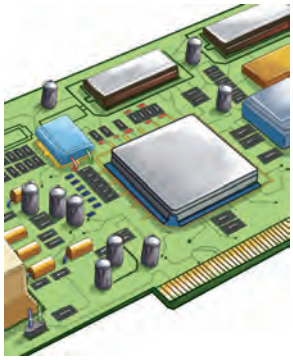
(b)

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

Related Problem

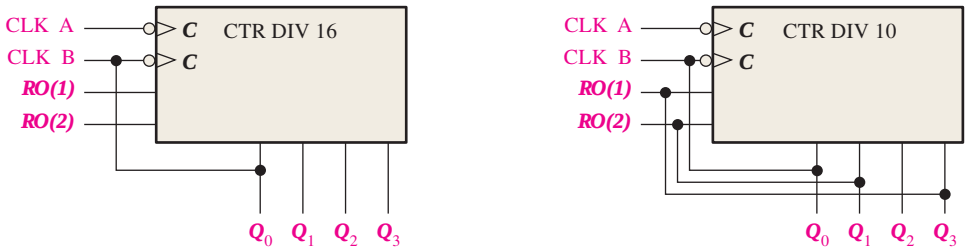
How can the counter in Figure 9–10(a) be modified to make it a modulus-13 counter?

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

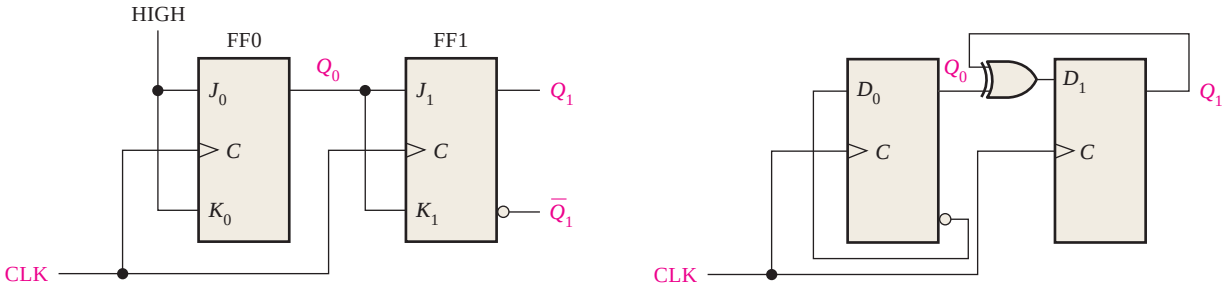
The term **synchronous** refers to events that have a fixed time relationship with each other. A **synchronous counter** is one in which all the flip-flops in the counter are clocked at the same time by a common clock pulse. J-K flip-flops are used to illustrate most synchronous counters. D flip-flops 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).

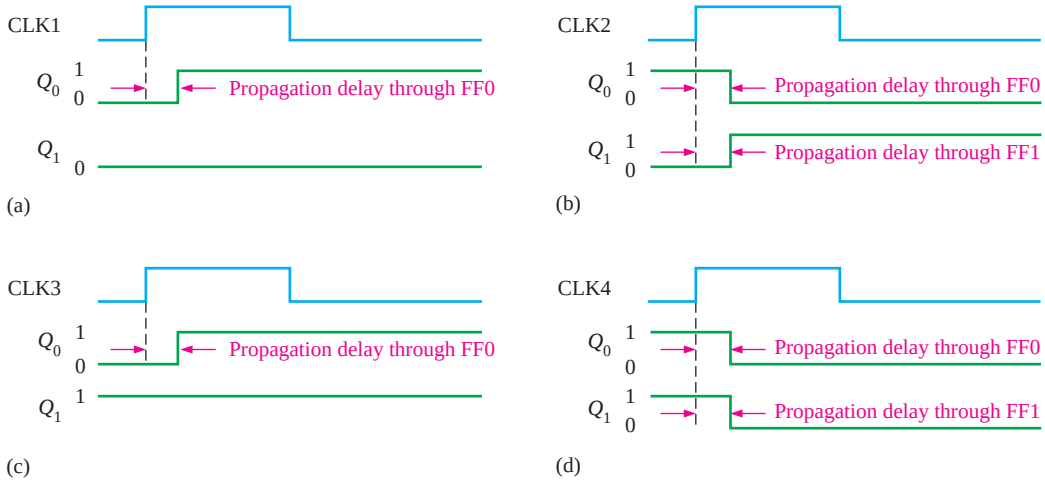


FIGURE 9–13 Timing details for the 2-bit synchronous counter operation (the propagation delays of both flip-flops are assumed to be equal).

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.

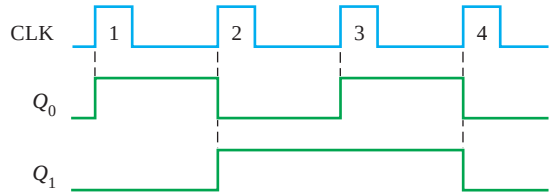
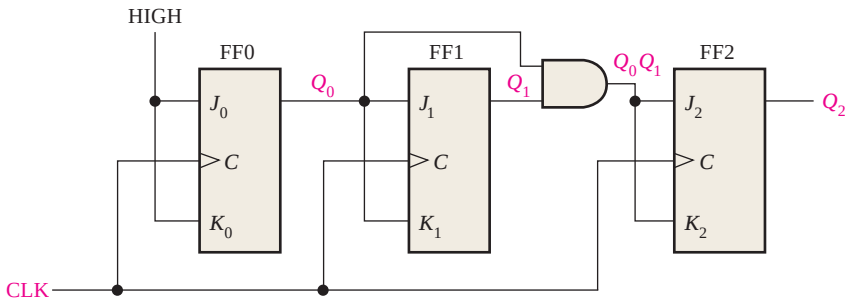


FIGURE 9–14 Timing diagram for the counters of Figure 9–12.

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.

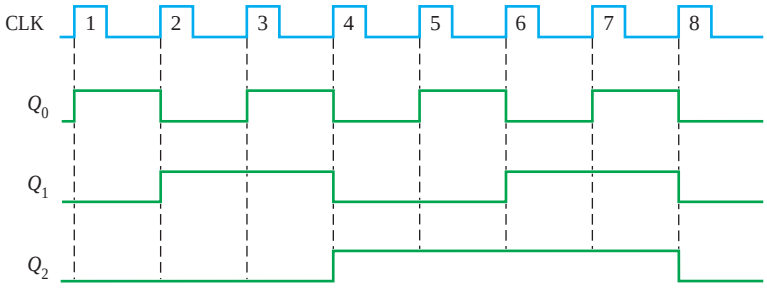


FIGURE 9–16 Timing diagram for the counter of Figure 9–15.

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

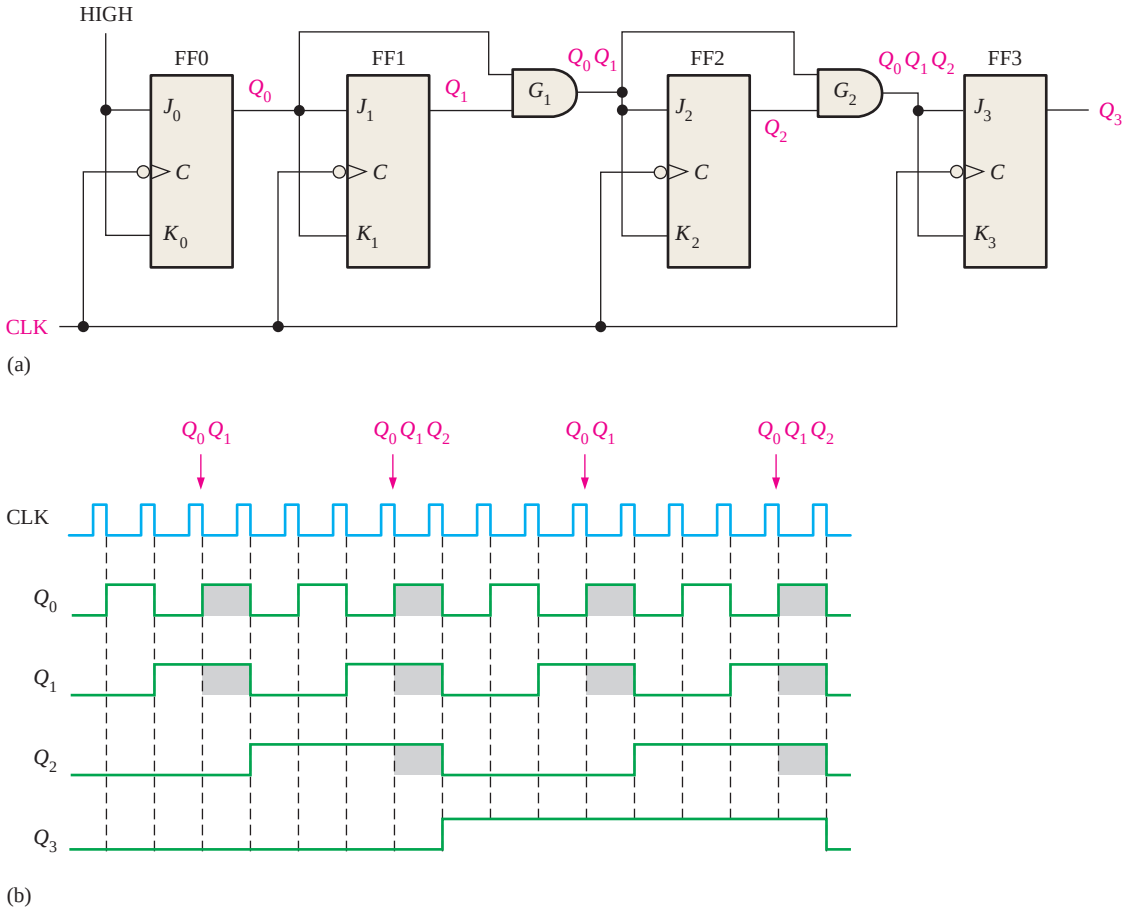


FIGURE 9–17 A 4-bit synchronous binary counter and timing diagram. Times where the AND gate outputs are HIGH are indicated by the shaded areas.

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.

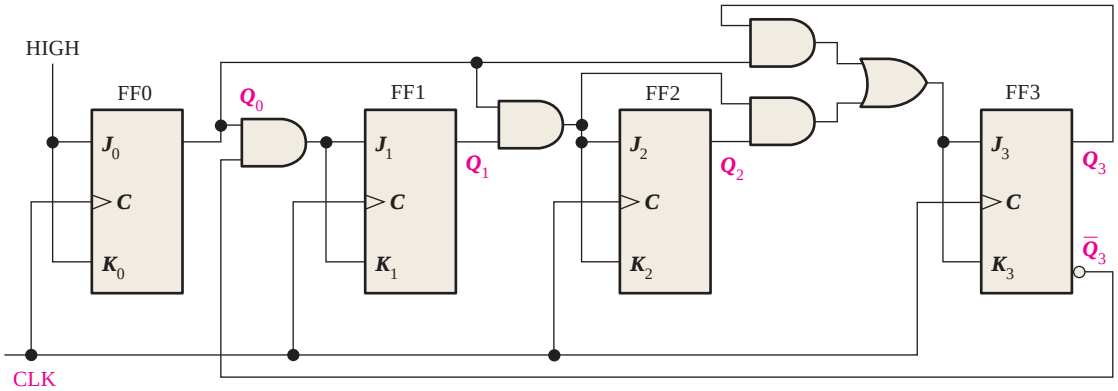


FIGURE 9–18 A synchronous BCD decade counter. Open file F09-18 to verify operation.

MultiSim

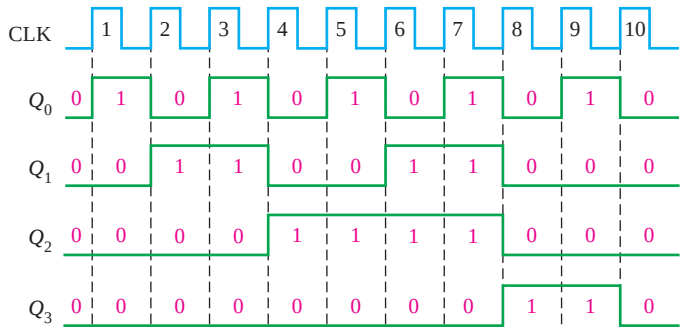


FIGURE 9–19 Timing diagram for the BCD decade counter (Q_0 is the LSB).

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.

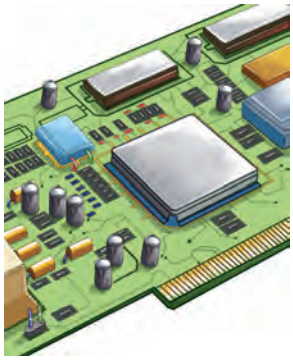
Finally, FF3 (Q_3) changes to the opposite state on the next clock pulse each time $Q_0 = 1$, $Q_1 = 1$, and $Q_2 = 1$ (state 7), or when $Q_0 = 1$ and $Q_3 = 1$ (state 9). The equation for this is as follows:

$$J_3 = K_3 = Q_0Q_1Q_2 + Q_0Q_3$$

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.

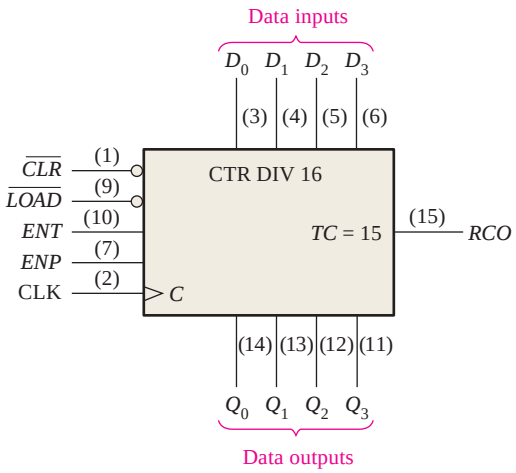


FIGURE 9–20 The 74HC163 4-bit synchronous binary counter. (The qualifying label CTR DIV 16 indicates a counter with sixteen states.)

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

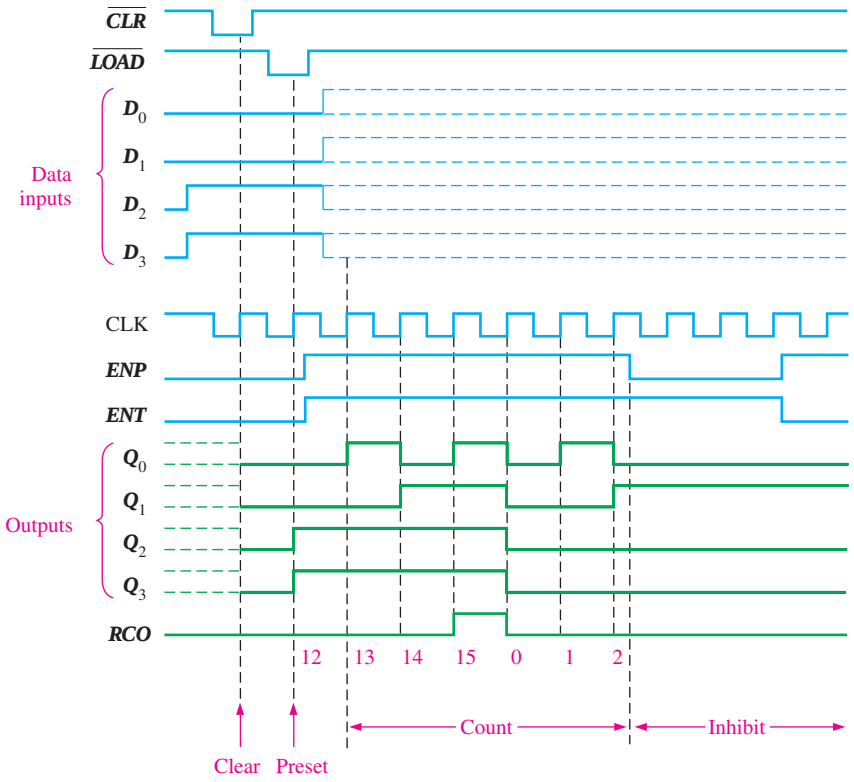


FIGURE 9–21 Timing example for a 74HC163.

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:

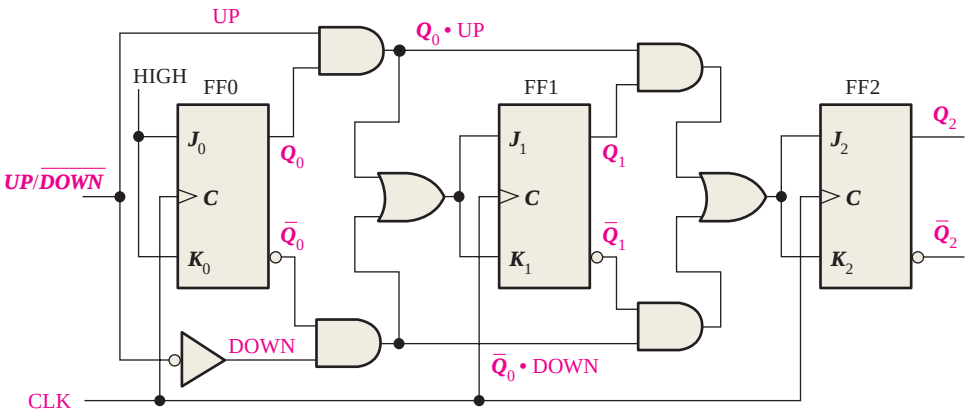
$$J_1 = K_1 = (Q_0 \cdot \text{UP}) + (\overline{Q}_0 \cdot \text{DOWN})$$

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.

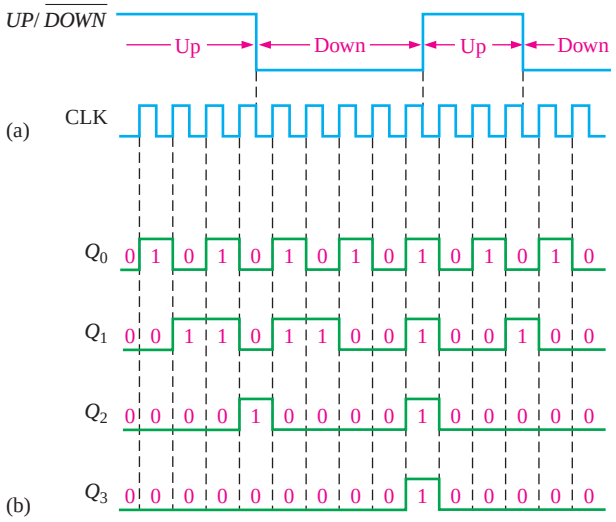


FIGURE 9–23

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.

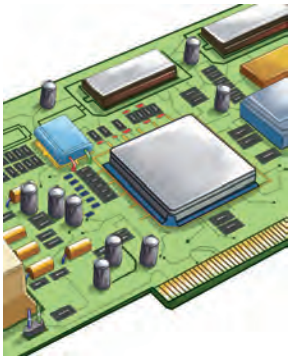
TABLE 9–7

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.

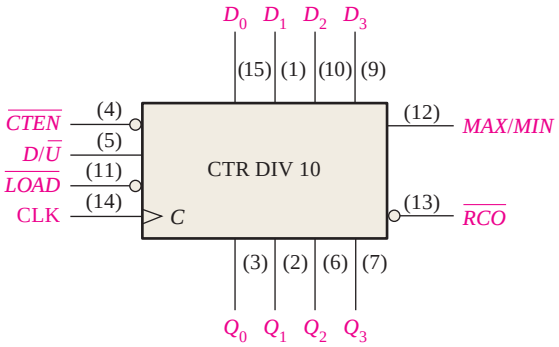


FIGURE 9–24 The 74HC190 up/down synchronous decade counter.

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*).

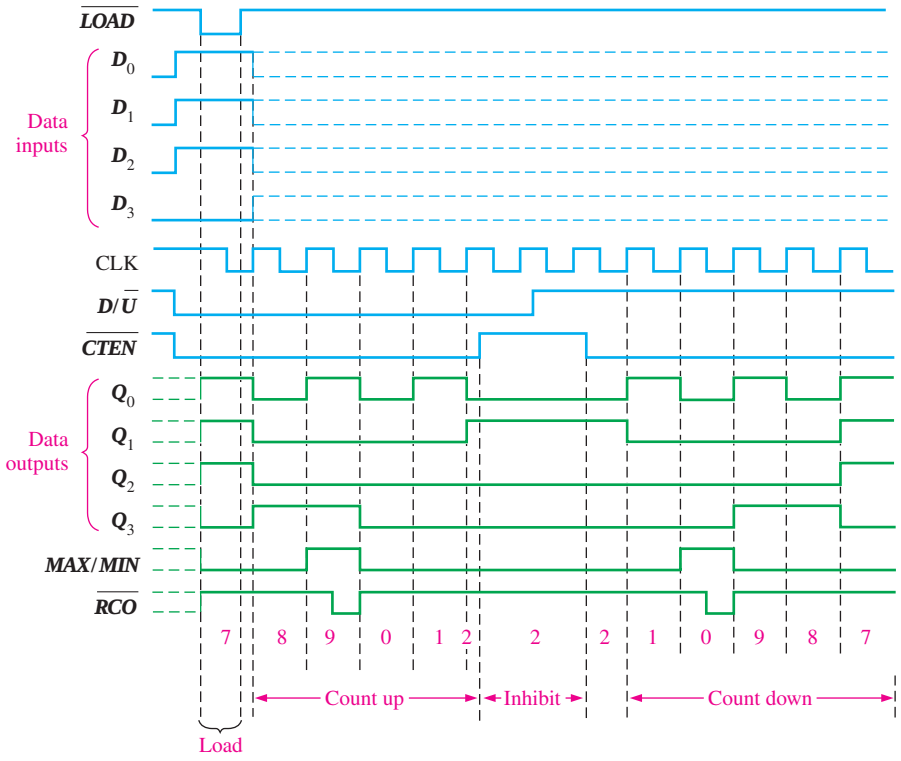


FIGURE 9–25 Timing example for a 74HC190.

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.

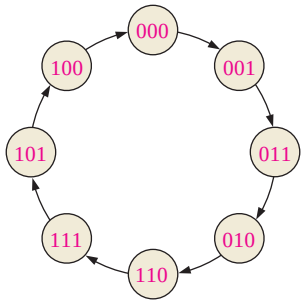


FIGURE 9–26 State diagram for a 3-bit Gray code counter.

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.

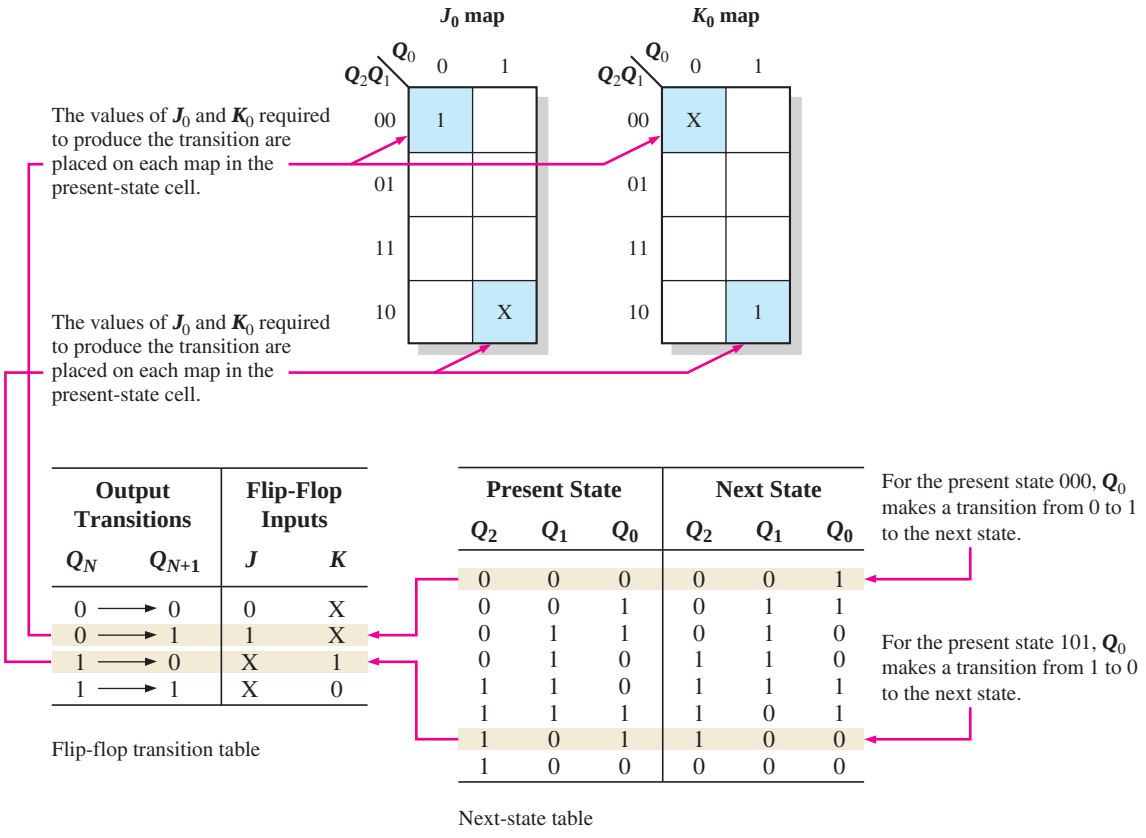


FIGURE 9–27 Examples of the mapping procedure for the counter sequence represented in Table 9–8 and Table 9–9.

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.

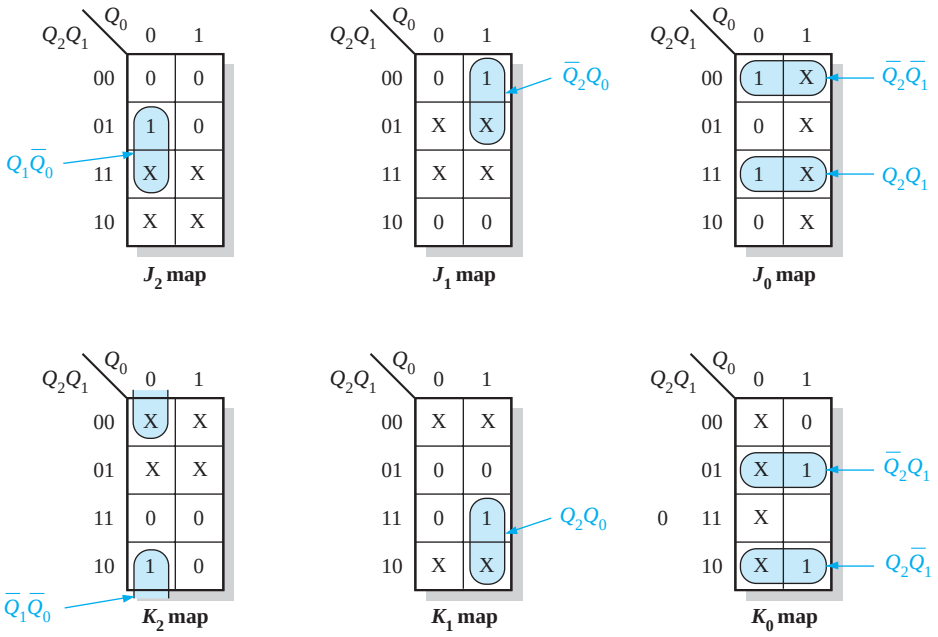


FIGURE 9–28 Karnaugh maps for present-state *J* and *K* inputs.

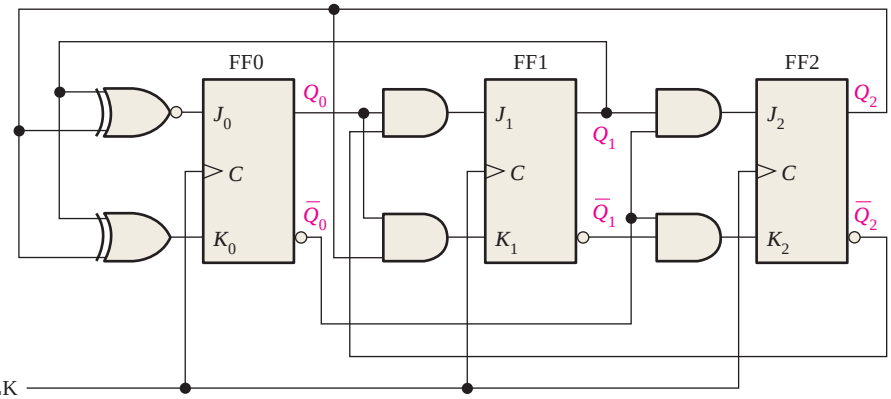
Step 5: Logic Expressions for Flip-Flop Inputs

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.

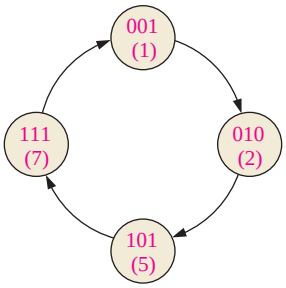


FIGURE 9–30

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.

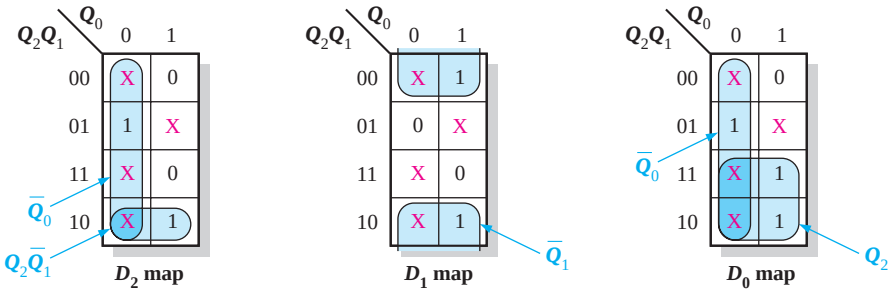


FIGURE 9–31

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.

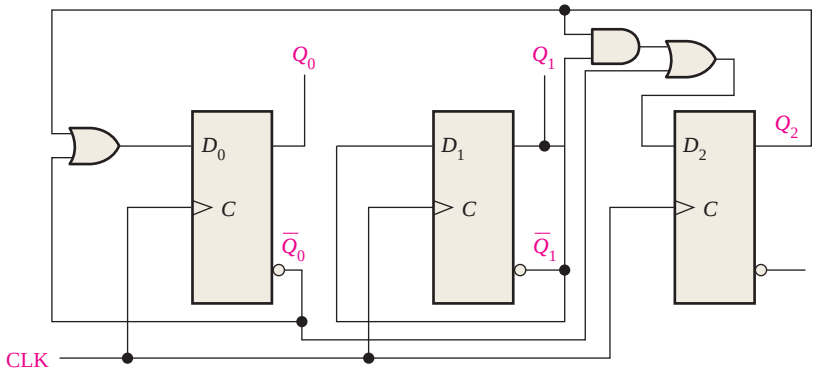


FIGURE 9–32

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*.

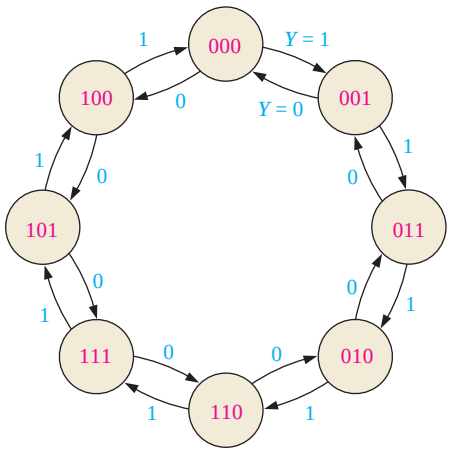


FIGURE 9–33 State diagram for a 3-bit up/down Gray code counter.

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.

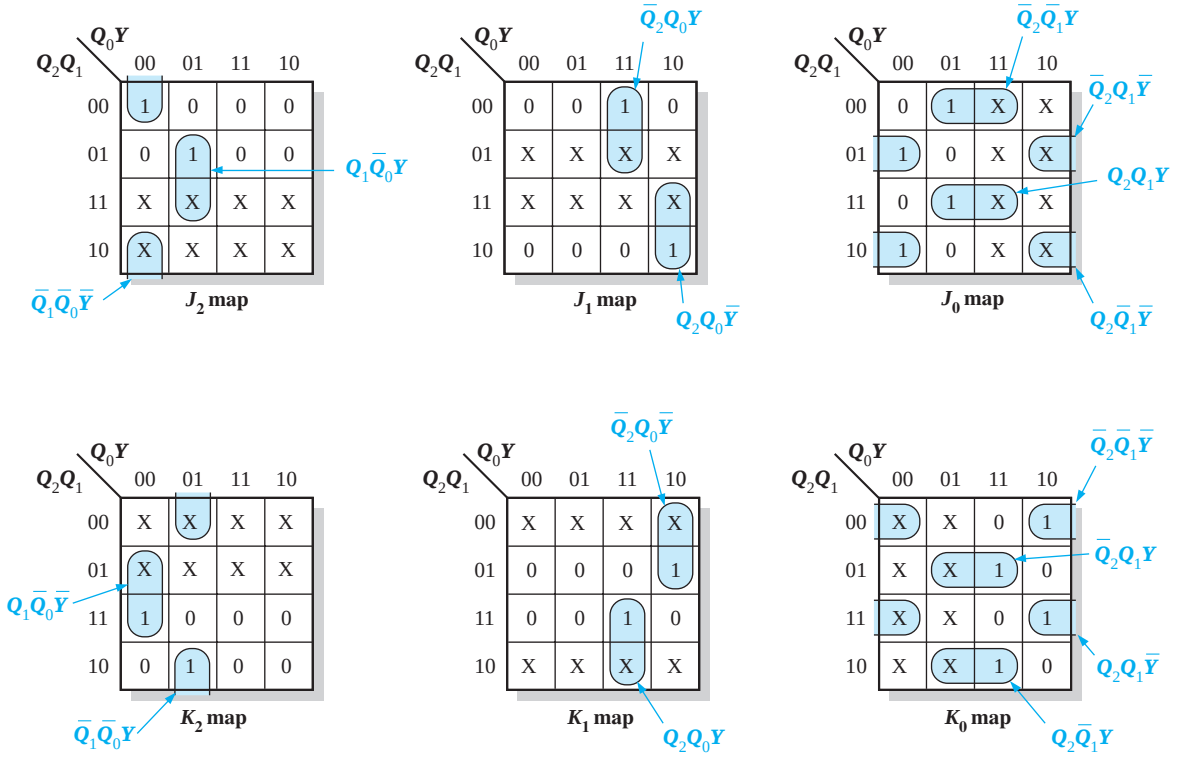


FIGURE 9–34 J and K maps for Table 9–12. The UP/$\overline{\text{DOWN}}$ control input, Y, is treated as a fourth variable.

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

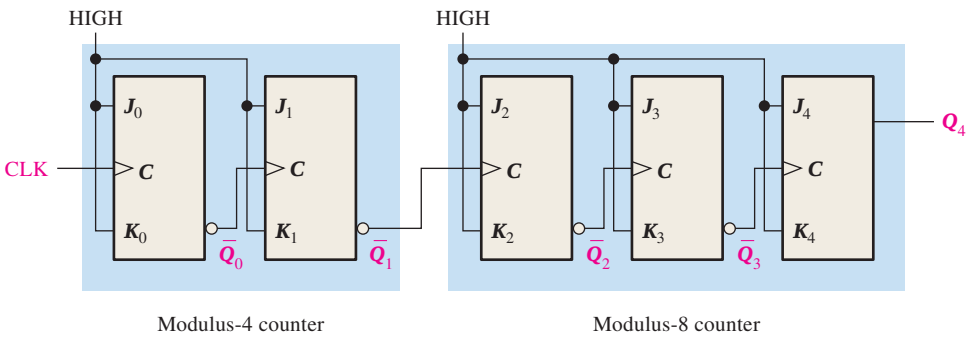


FIGURE 9–35 Two cascaded asynchronous counters (all J and K inputs are HIGH).

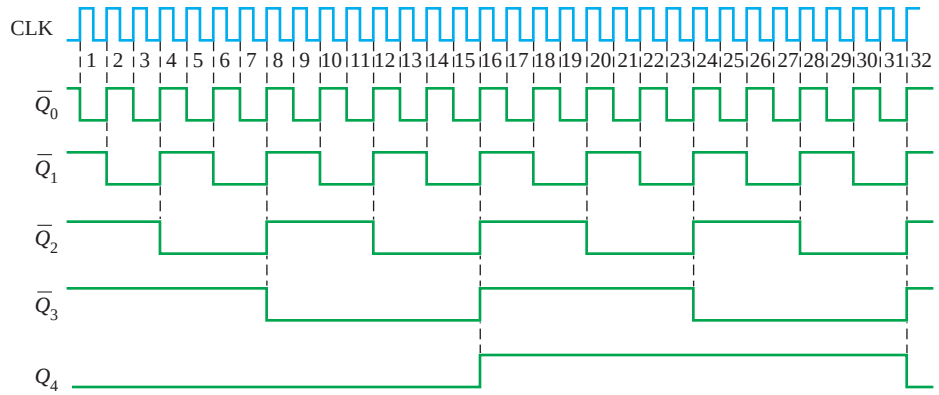


FIGURE 9–36 Timing diagram for the cascaded counter configuration of Figure 9–35.

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.

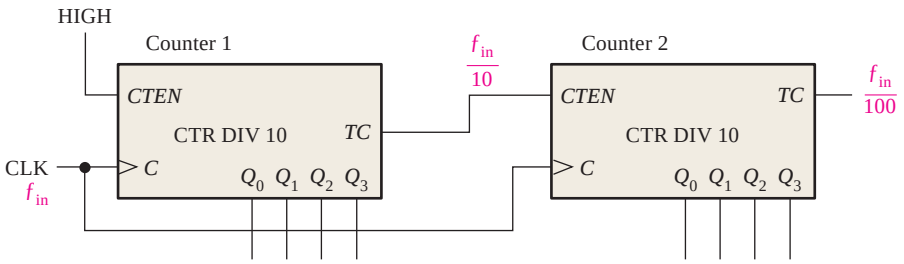


FIGURE 9–37 A modulus-100 counter using two cascaded decade counters.

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.

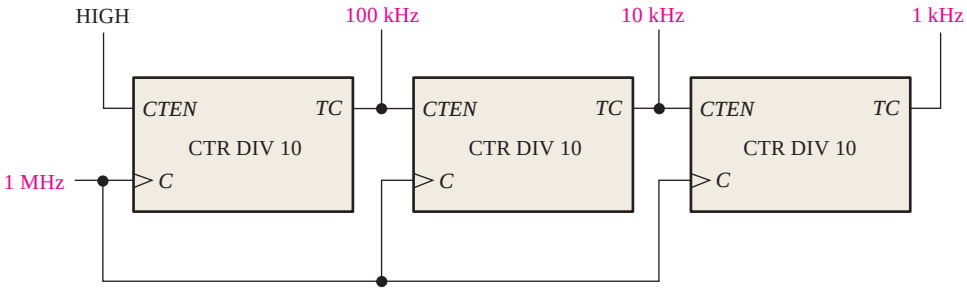


FIGURE 9–38 Three cascaded decade counters forming a divide-by-1000 frequency divider with intermediate divide-by-10 and divide-by-100 outputs.

EXAMPLE 9–6

Determine the overall modulus of the two cascaded counter configurations in Figure 9–39.

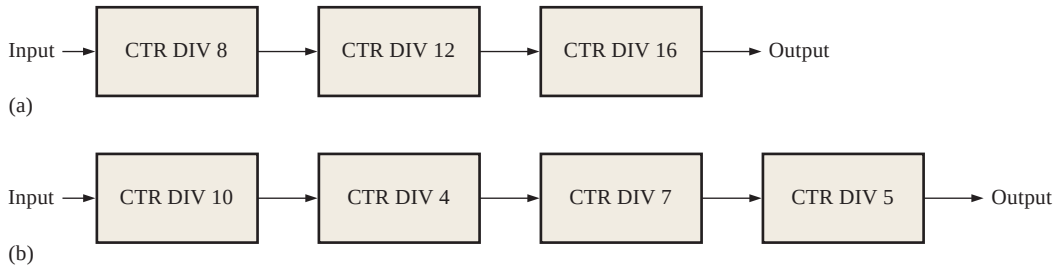


FIGURE 9–39

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.

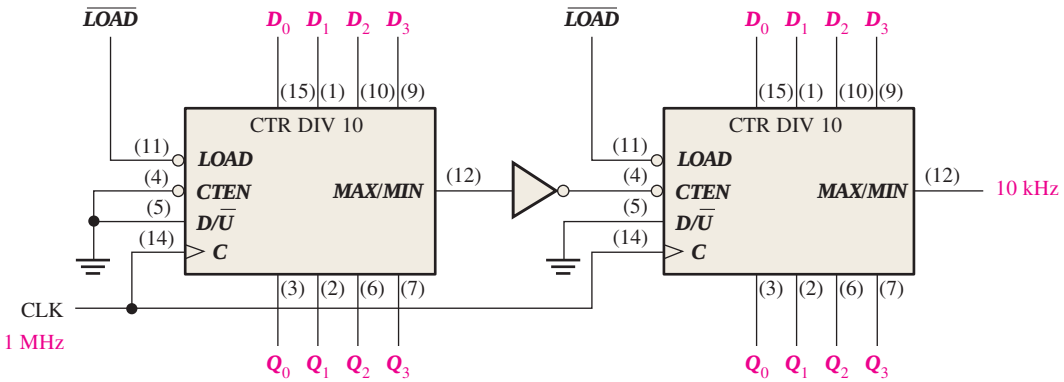


FIGURE 9–40 A divide-by-100 counter using two 74HC190 up/down decade counters connected for the up sequence.

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

$$2^{16} = 65{,}536$$

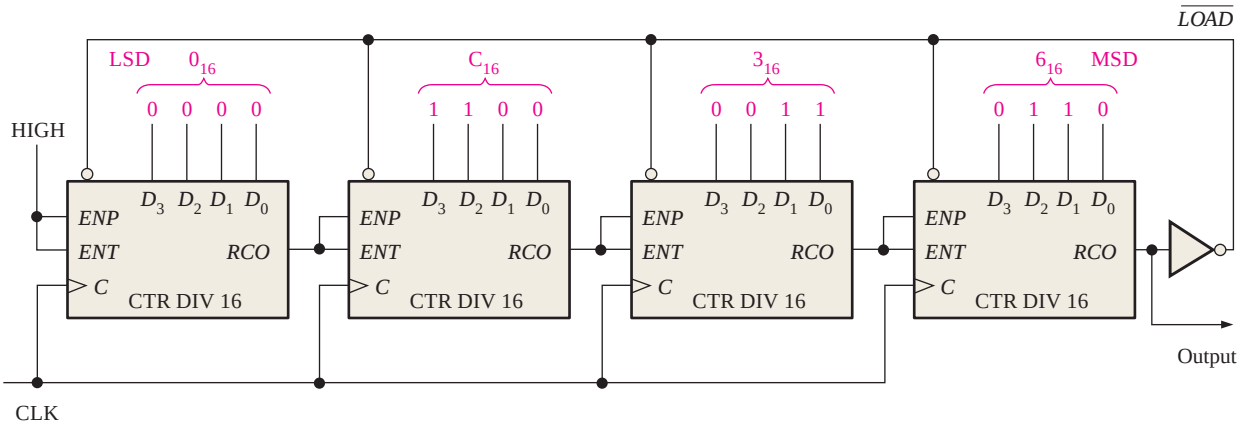


FIGURE 9–41 A divide-by-40,000 counter using 74HC161 4-bit binary counters. Note that each of the parallel data inputs is shown in binary order (the right-most bit D_0 is the LSB in each counter).

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.

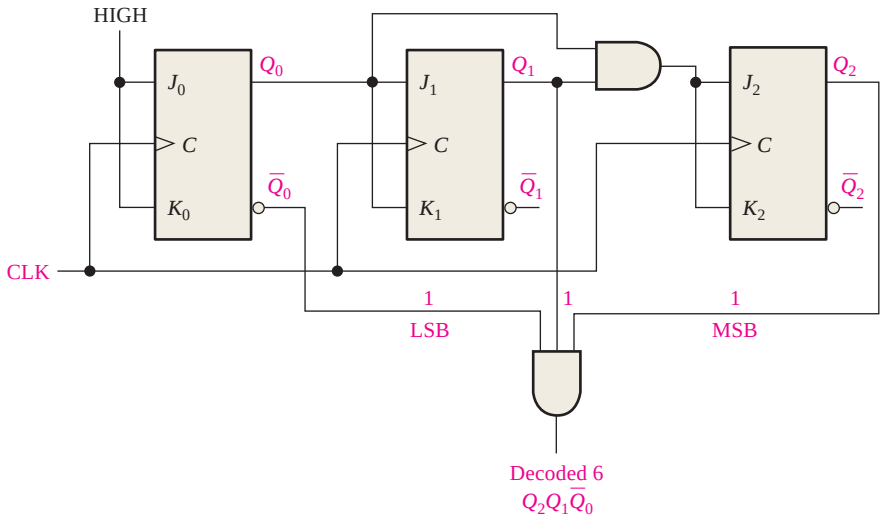


FIGURE 9–42 Decoding of state 6 (110). Open file F09-42 to verify operation.

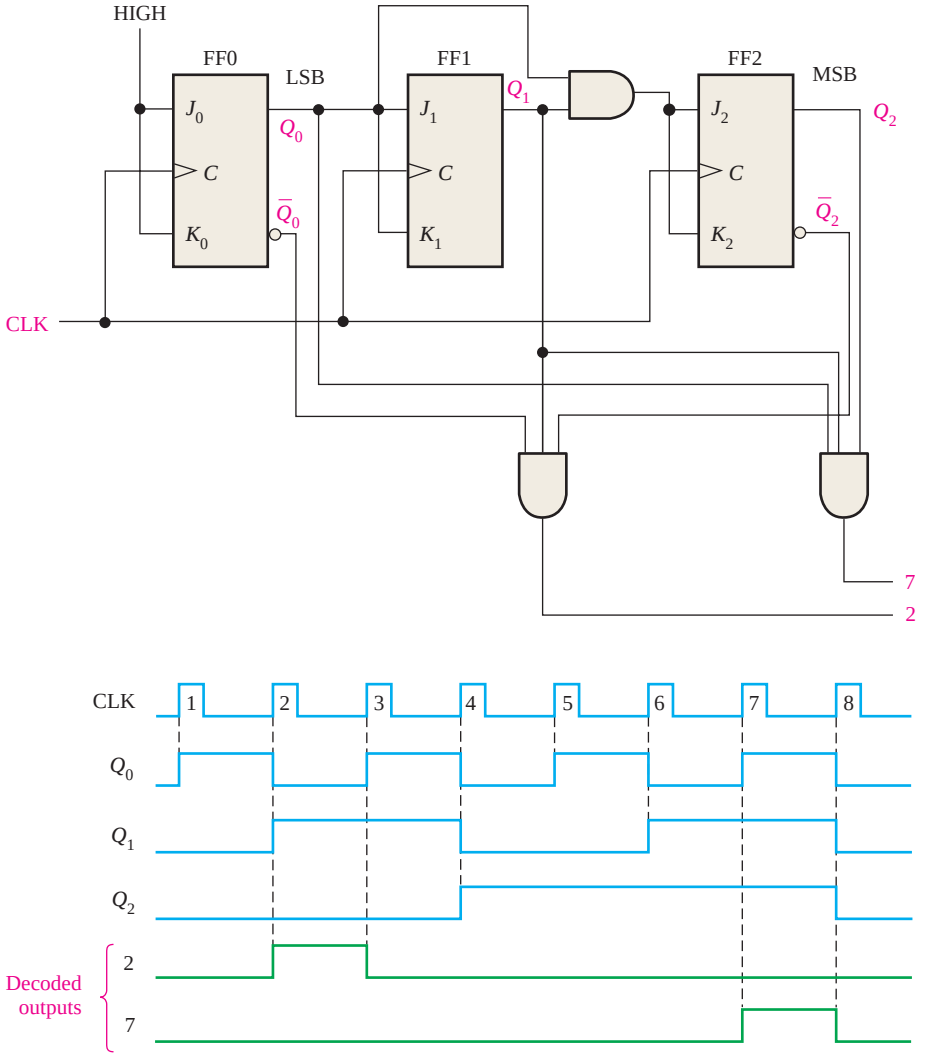
MultiSim

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.

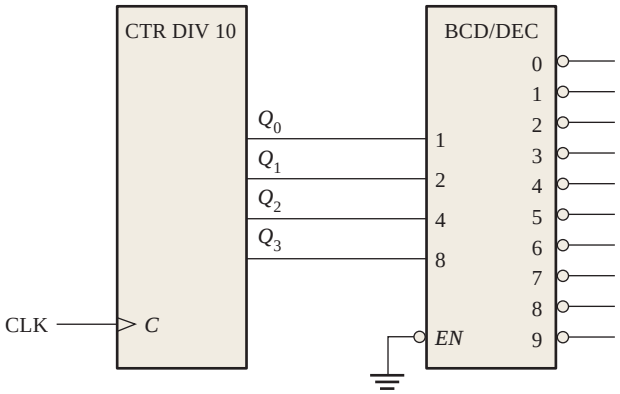


FIGURE 9–44 A basic decade (BCD) counter and decoder.

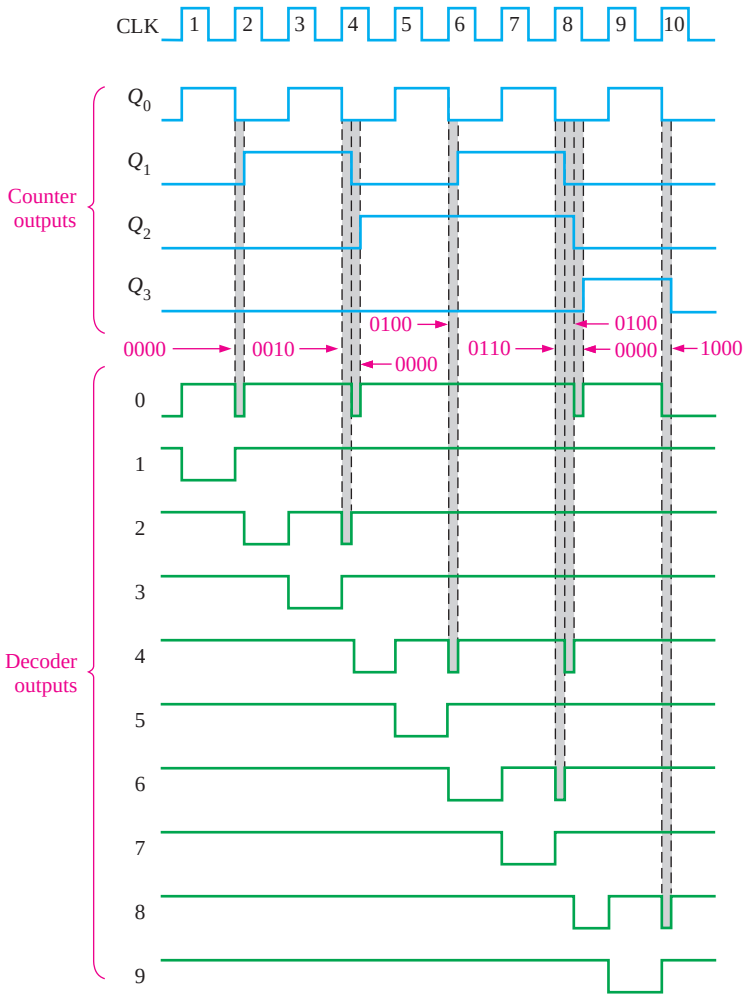


FIGURE 9–45 Outputs with glitches from the decoder in Figure 9–44. Glitch widths are exaggerated for illustration and are usually only a few nanoseconds wide.

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.

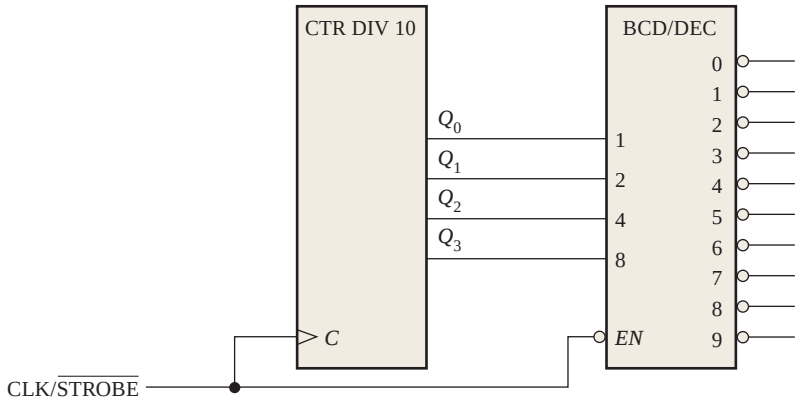


FIGURE 9–46 The basic decade counter and decoder with strobing to eliminate glitches.

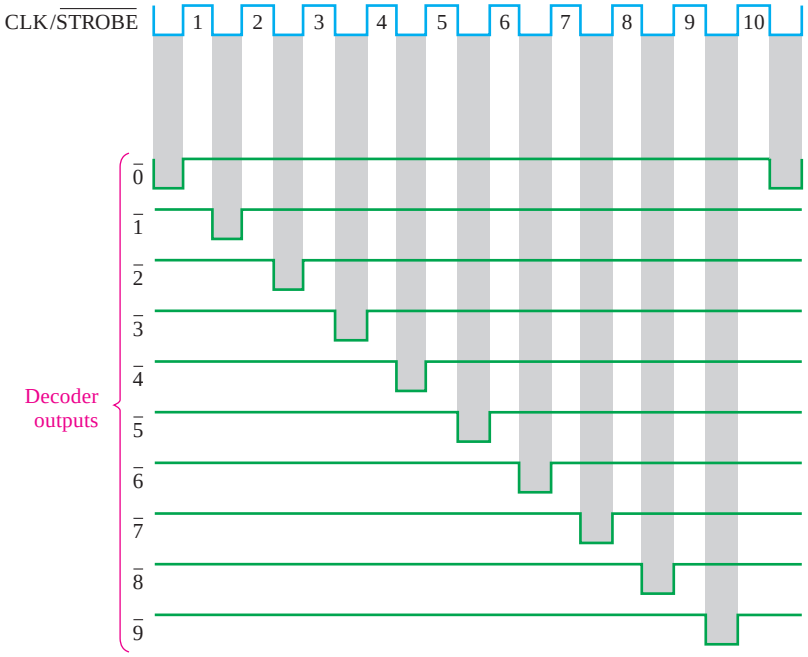


FIGURE 9–47 Strobed decoder outputs for the circuit of Figure 9–46.

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.

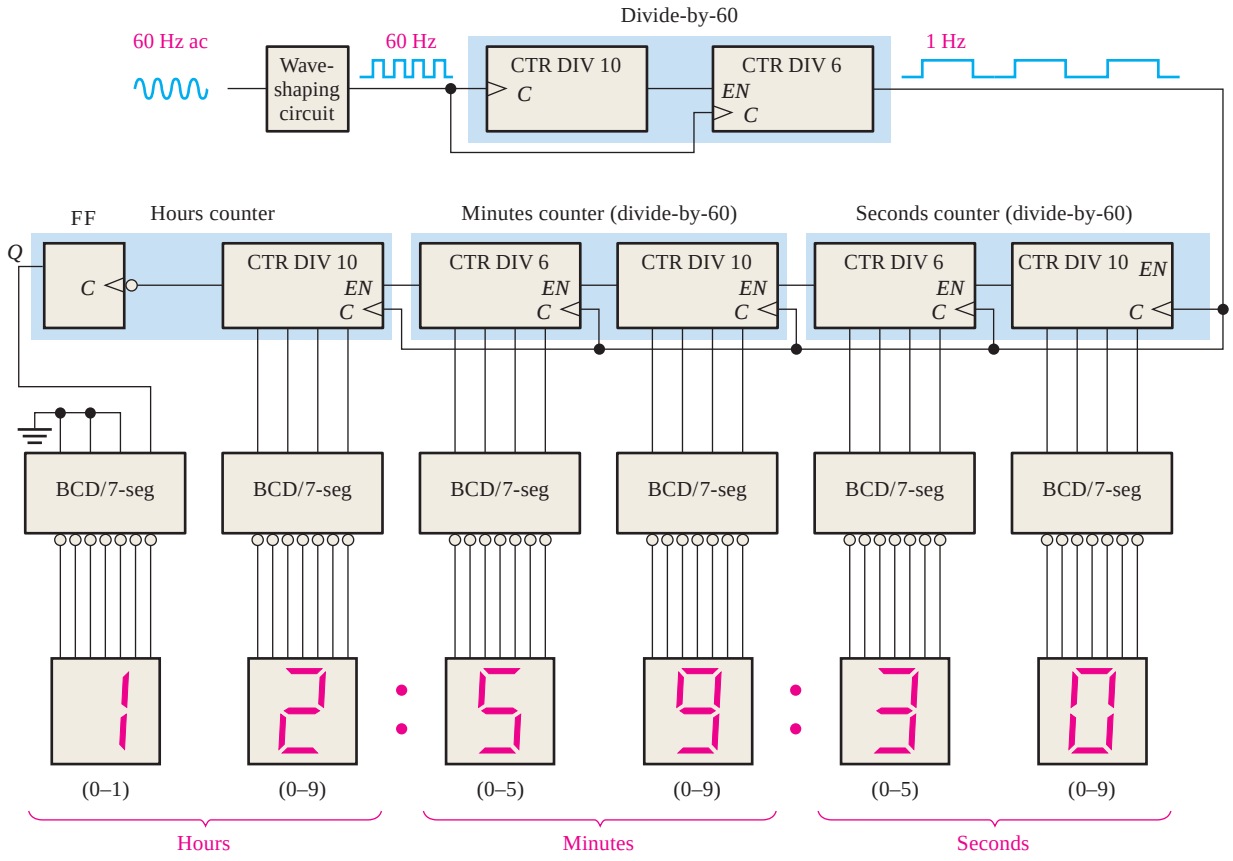


FIGURE 9–48 Simplified logic diagram for a 12-hour digital clock. Logic details using specific devices are shown in Figures 9–49 and 9–50.

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).

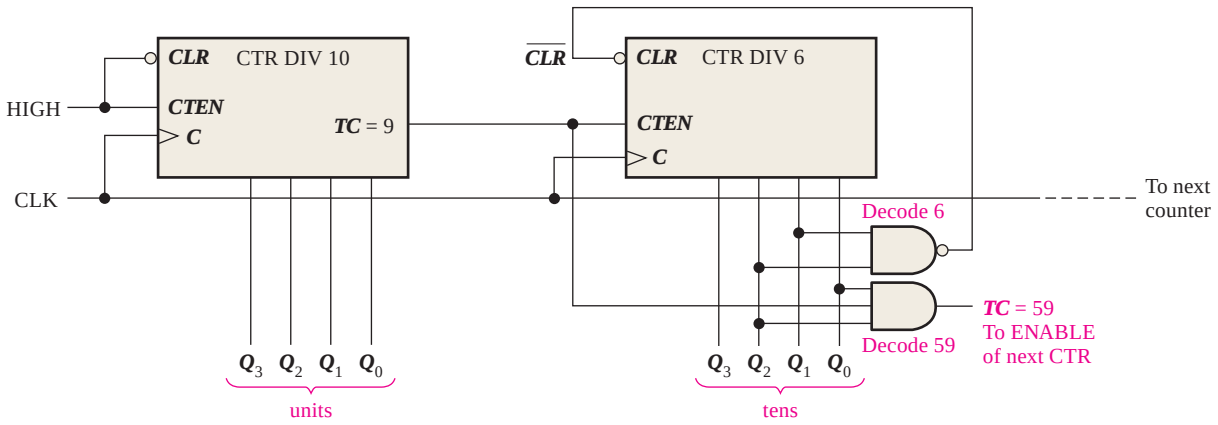


FIGURE 9–49 Logic diagram of typical divide-by-60 counter using synchronous decade counters. Note that the outputs are in binary order (the right-most bit is the LSB).

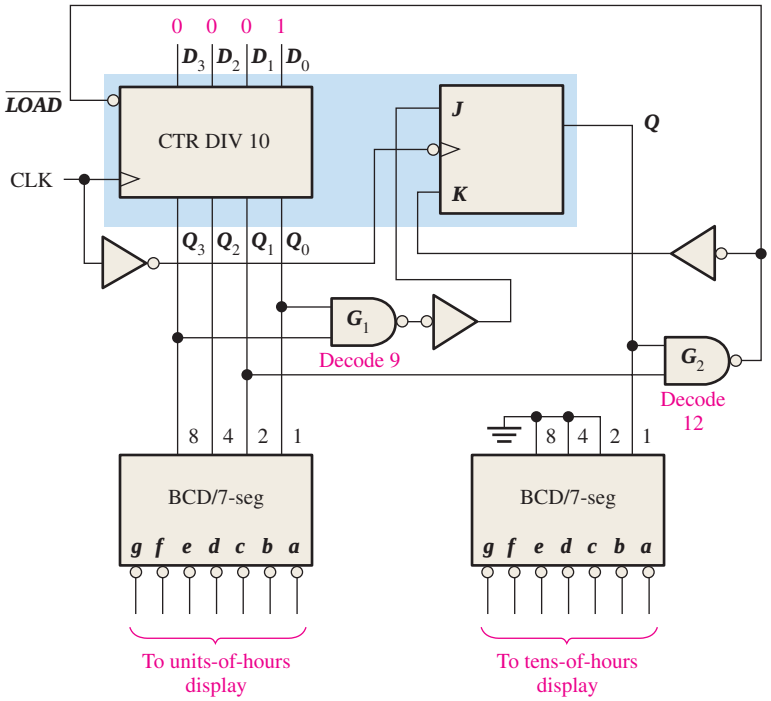


FIGURE 9–50 Logic diagram for hours counter and decoders. Note that on the counter inputs and outputs, the right-most bit is the LSB.

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.

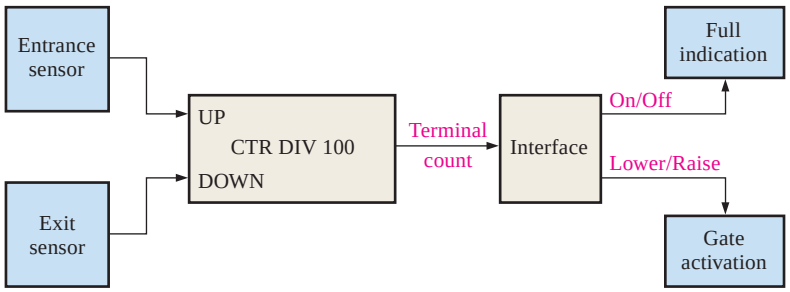


FIGURE 9–51 Functional block diagram for parking garage control.

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.

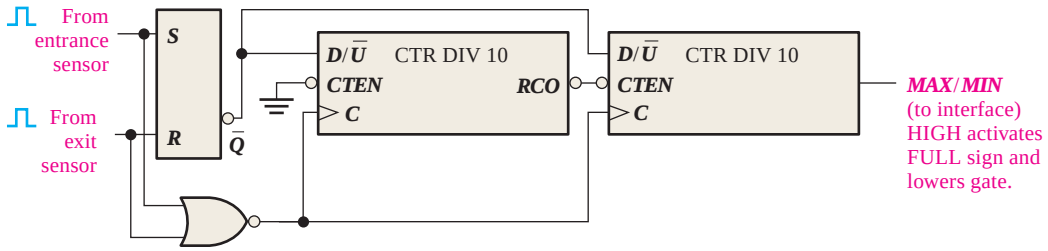


FIGURE 9–52 Logic diagram for modulus-100 up/down counter for automobile parking control.

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.

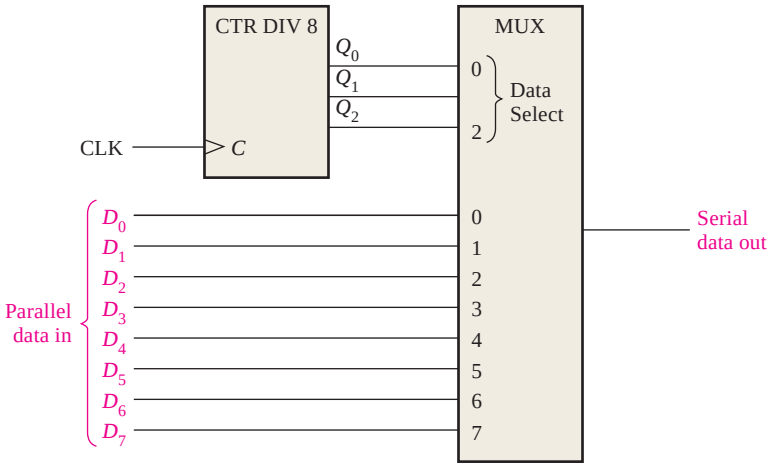


FIGURE 9–53 Parallel-to-serial data conversion logic.

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

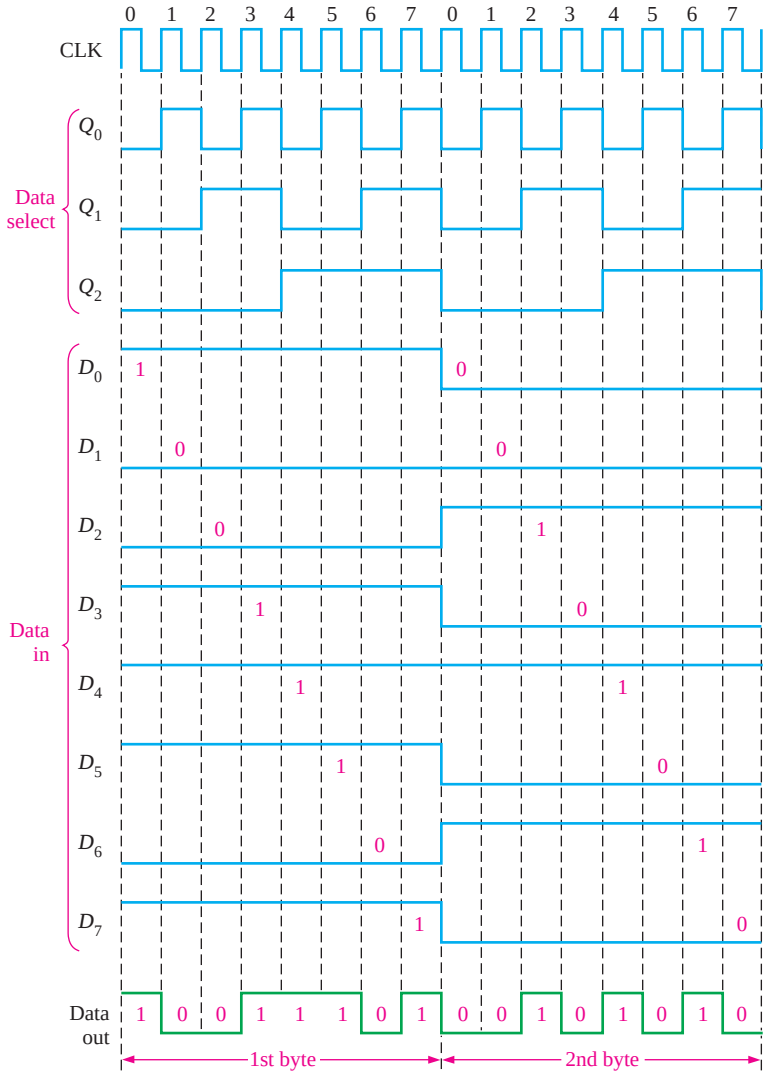


FIGURE 9–54 Example of parallel-to-serial conversion timing for the circuit in Figure 9–53.

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

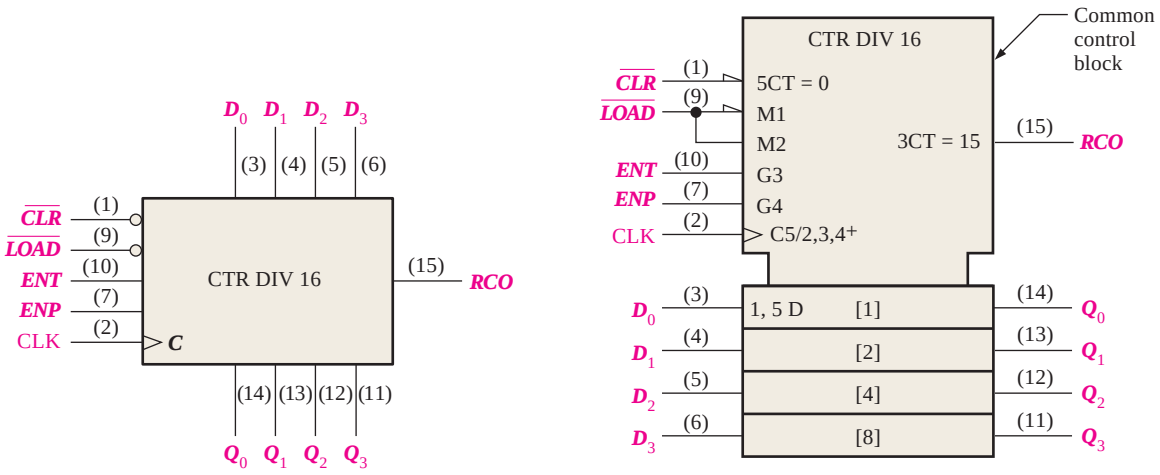
The upper block with notched corners in Figure 9–55(b) has inputs and an output that are considered common to all elements in the device and not unique to any one of the elements.

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?