7–1 Latches

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

After completing this section, you should be able to

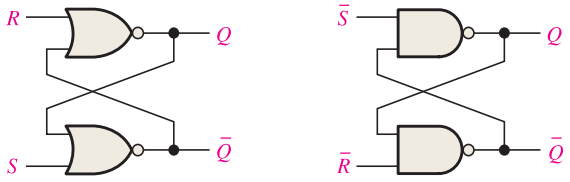
- Explain the operation of a basic S-R latch
- Explain the operation of a gated S-R latch
- Explain the operation of a gated D latch
- Implement an S-R or D latch with logic gates
- Describe the 74HC279A and 74HC75 quad latches

InfoNote

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

The S-R (SET-RESET) Latch

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



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

MultiSim

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

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

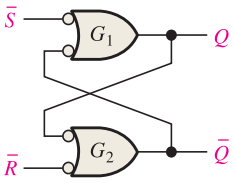


FIGURE 7–2 Negative-OR equivalent of the NAND gate $\overline{S}$-$\overline{R}$ latch in Figure 7–1(b).

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

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

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

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

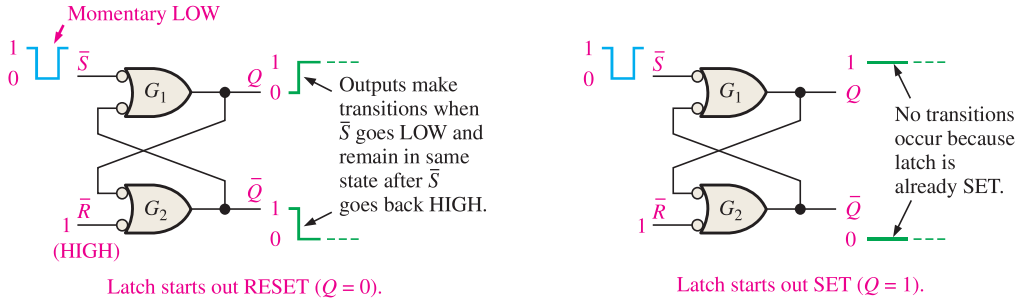
When Q is HIGH, $\overline{Q}$ is LOW, and when Q is LOW, $\overline{Q}$ is HIGH.

SET means that the Q output is HIGH.

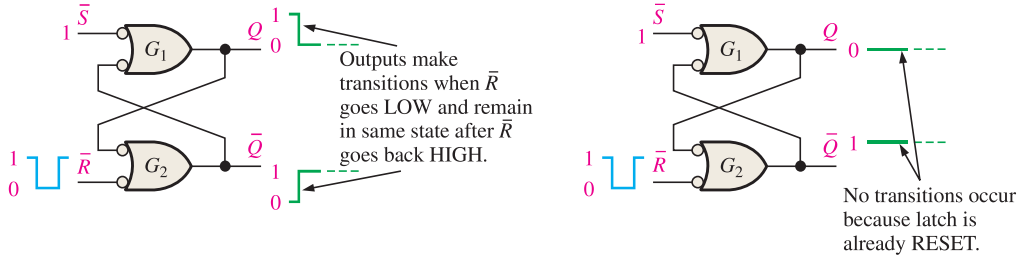
An invalid condition in the operation of an active-LOW input $\overline{S}$-$\overline{R}$ latch occurs when LOWs are applied to both $\overline{S}$ and $\overline{R}$ at the same time. As long as the LOW levels are simultaneously held on the inputs, both the Q and $\overline{Q}$ outputs are forced HIGH,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

RESET means that the Q output is LOW.

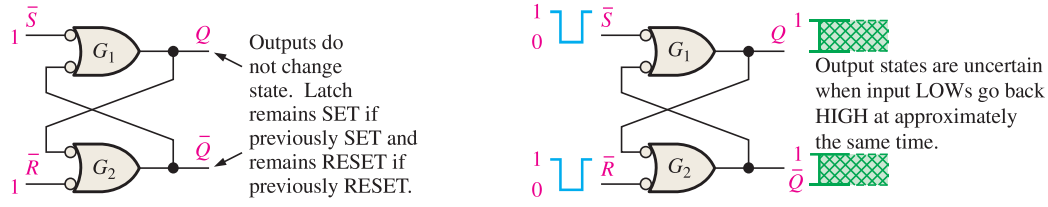
Figure 7–3 illustrates the active-LOW input $\overline{S}$-$\overline{R}$ latch operation for each of the four possible combinations of levels on the inputs. (The first three combinations are valid, but the last is not.) Table 7–1 summarizes the logic operation in truth table form. Operation of the active-HIGH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



(a) Two possibilities for the SET operation



(b) Two possibilities for the RESET operation



(c) No-change condition

(d) Invalid condition

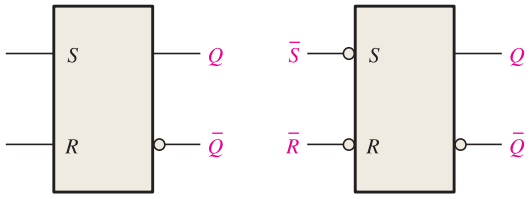
FIGURE 7–3 The three modes of basic $\overline{S}$-$\overline{R}$ latch operation (SET, RESET, no-change) and the invalid condition.

TABLE 7–1

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

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

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



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

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

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

EXAMPLE 7–1

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

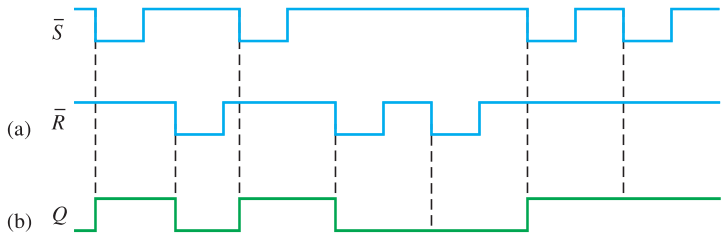


FIGURE 7–5

Solution

See Figure 7–5(b).

Related Problem*

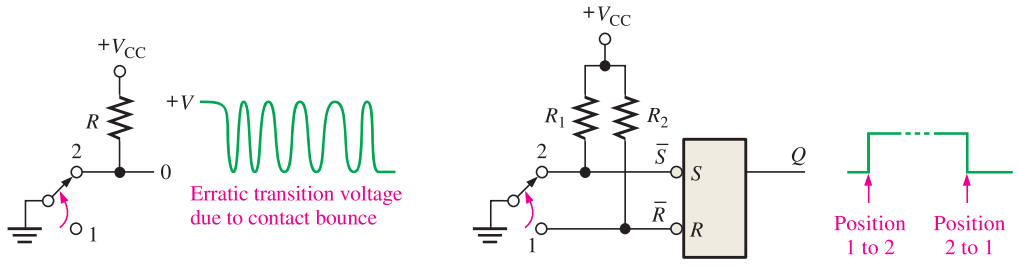
Determine the Q output of an active-HIGH input S-R latch if the waveforms in Figure 7–5(a) are inverted and applied to the inputs.

*Answers are at the end of the chapter.

An Application

The Latch as a Contact-Bounce Eliminator

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



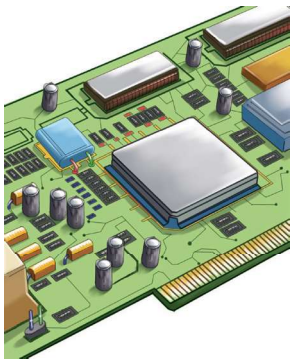
(a) Switch contact bounce

(b) Contact-bounce eliminator circuit

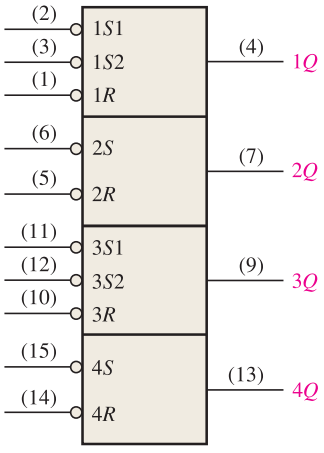
FIGURE 7–6 The $\overline{S}$-$\overline{R}$ latch used to eliminate switch contact bounce.

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

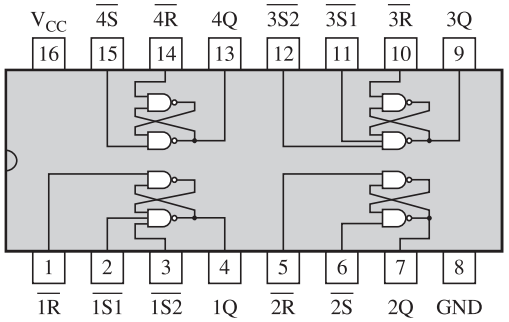
IMPLEMENTATION: $\overline{S}$-$\overline{R}$ LATCH



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



(a) Logic diagram



(b) Pin diagram

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

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

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

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

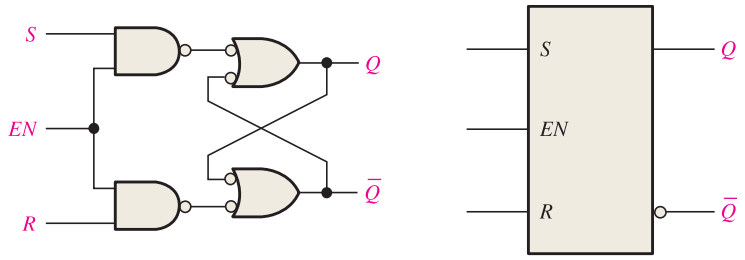
SNot: SET complement
RNot: RESET complement
Q: Latch output
QNot: Latch output complement

Boolean expressions define the outputs

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

The Gated S-R Latch

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



(a) Logic diagram (b) Logic symbol

FIGURE 7–8 A gated S-R latch.

EXAMPLE 7–2

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

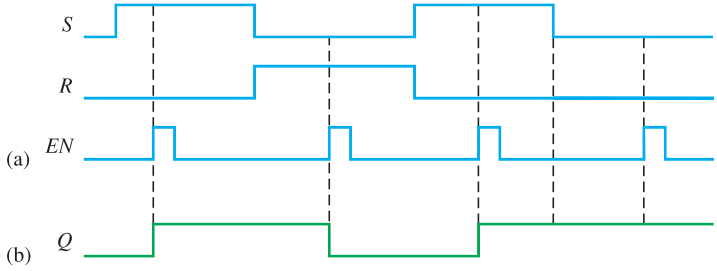


FIGURE 7–9

Solution

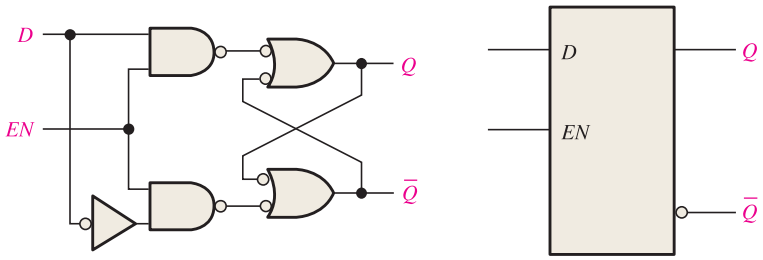
The Q waveform is shown in Figure 7–9(b). When S is HIGH and R is LOW, a HIGH on the EN input sets the latch. When S is LOW and R is HIGH, a HIGH on the EN input resets the latch. When both S and R are LOW, the Q output does not change from its present state.

Related Problem

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

The Gated D Latch

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



(a) Logic diagram (b) Logic symbol

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

MultiSim

EXAMPLE 7–3

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

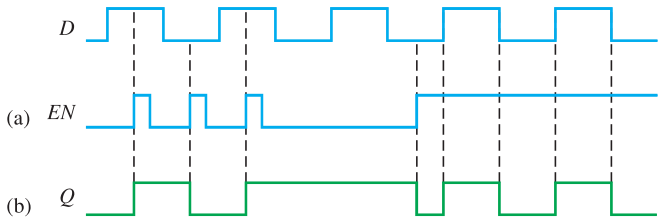


FIGURE 7–11

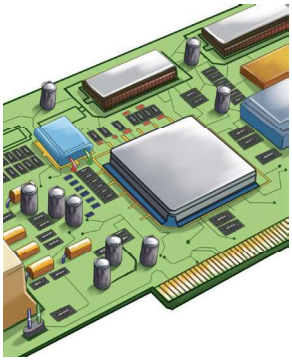
Solution

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

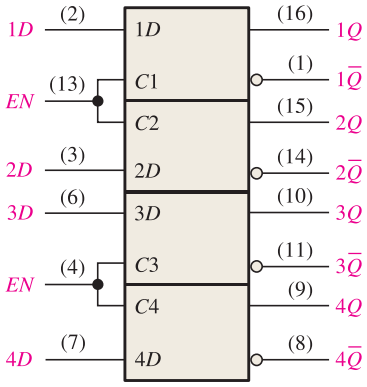
Related Problem

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

IMPLEMENTATION: GATED D LATCH



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



(a) Logic symbol

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

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

(b) Truth table (each latch)

FIGURE 7–12 The 74HC75 quad D latch.

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

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

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

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

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

} Boolean expressions define the outputs

SECTION 7–1 CHECKUP

Answers are at the end of the chapter.

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

7–2 Flip-Flops

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

After completing this section, you should be able to

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

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

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

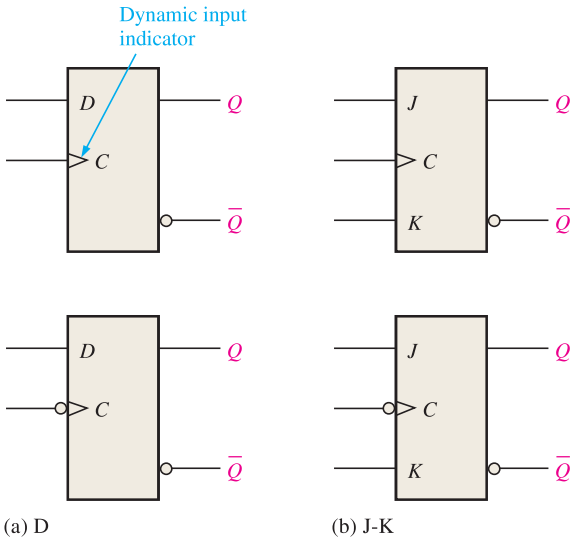


FIGURE 7–13 Edge-triggered flip-flop logic symbols (top: positive edge-triggered; bottom: negative edge-triggered).

The D Flip-Flop

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

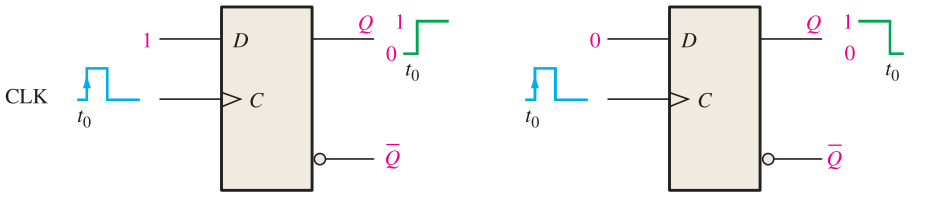
D flip-flop but *D* as variable.

InfoNote

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

is SET. When D is LOW, the Q output goes LOW on the triggering edge of the clock pulse, and the flip-flop is RESET.

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



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

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

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

TABLE 7–2

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

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

↑ = clock transition LOW to HIGH

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

EXAMPLE 7–4

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

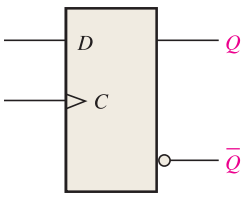


FIGURE 7–15

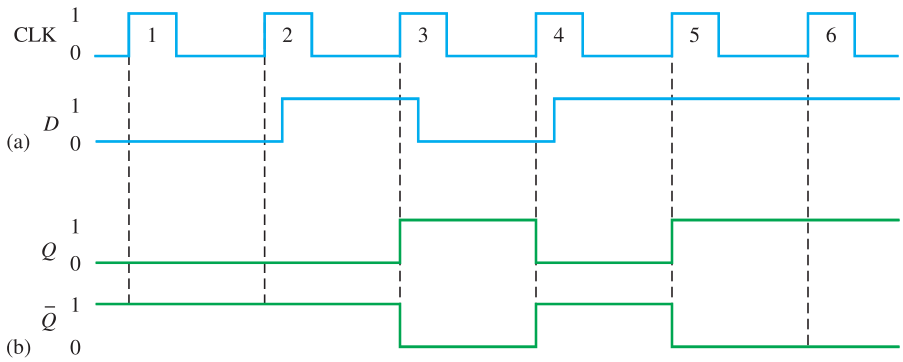


FIGURE 7–16

Solution

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

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

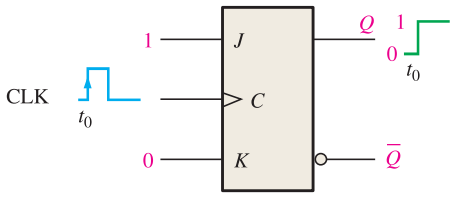
Related Problem

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

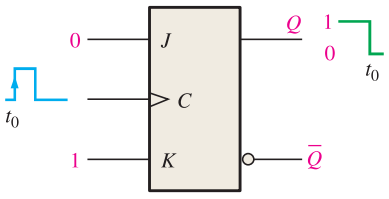
The J-K Flip-Flop

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

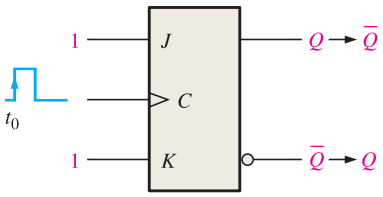
This basic operation of a positive edge-triggered flip-flop is illustrated in Figure 7–17, and Table 7–3 is the truth table for this type of flip-flop. Remember, *the flip-flop cannot change state except on the triggering edge of a clock pulse.* The J and K inputs can be changed at any time when the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



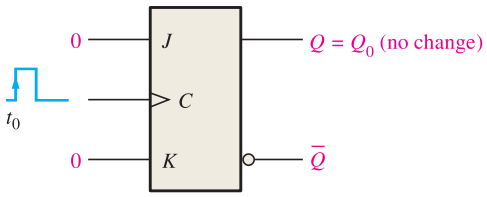
(a) $J = 1$, $K = 0$ flip-flop SETS on positive clock edge. (If already SET, it remains SET.)



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



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



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

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

TABLE 7–3

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

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

↑ = clock transition LOW to HIGH

Q_0 = output level prior to clock transition

EXAMPLE 7–5

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

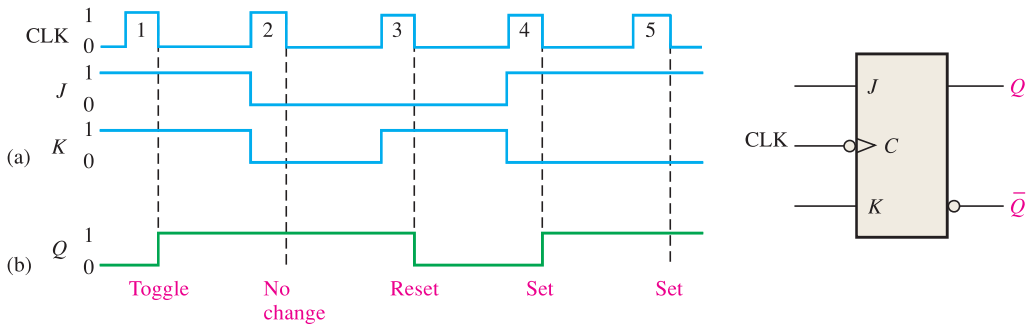


FIGURE 7–18

Solution

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

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

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

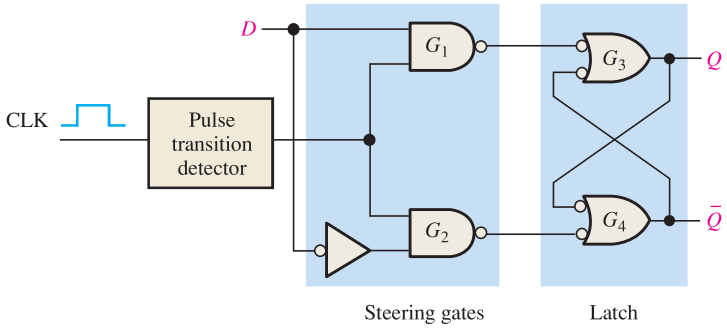
Related Problem

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

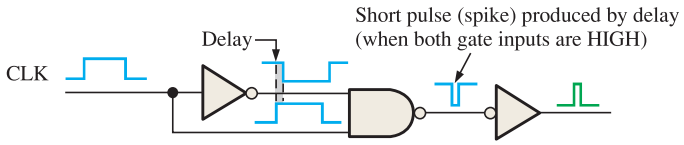
Edge-Triggered Operation

D Flip-Flop

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



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



(b) A type of pulse transition detector

FIGURE 7–19 Edge triggering.

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

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

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

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

InfoNote

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

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

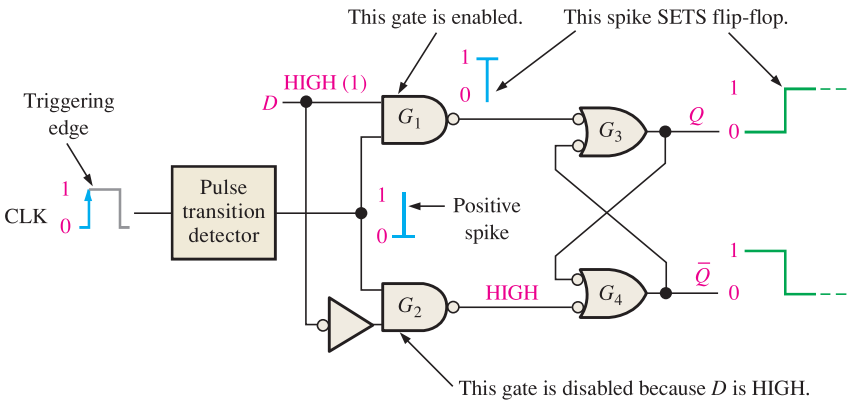


FIGURE 7–20 Flip-flop making a transition from the RESET state to the SET state on the positive-going edge of the clock pulse.

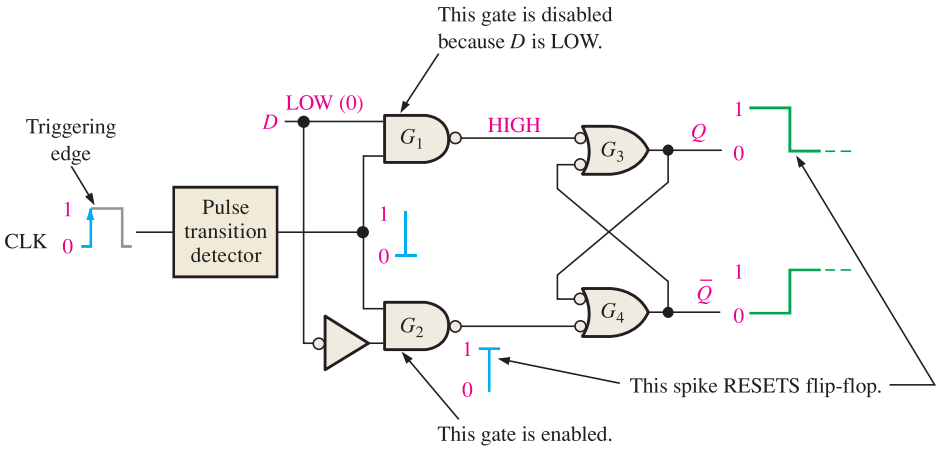


FIGURE 7–21 Flip-flop making a transition from the SET state to the RESET state on the positive-going edge of the clock pulse.

EXAMPLE 7–6

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

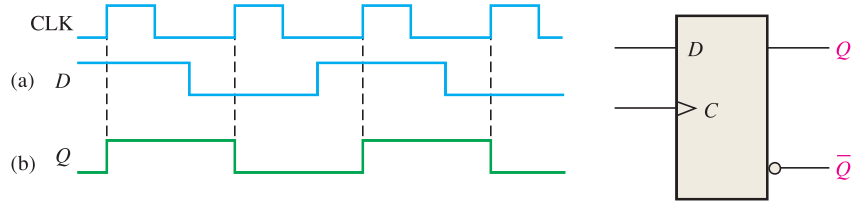


FIGURE 7–22

Solution

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

Related Problem

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

J-K Flip-Flop

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

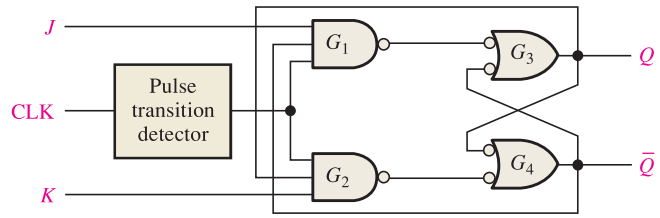


FIGURE 7–23 A simplified logic diagram for a positive edge-triggered J-K flip-flop.

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

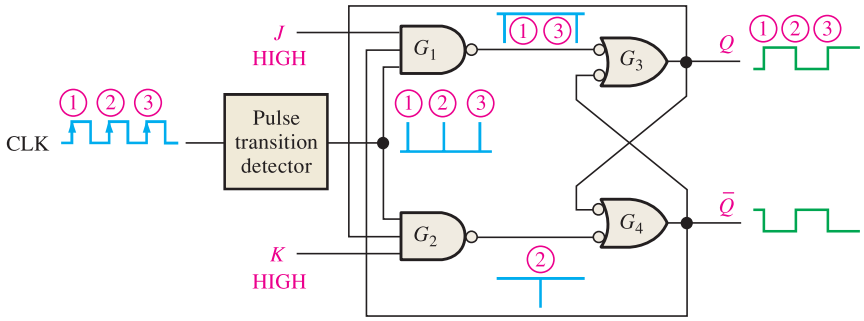


FIGURE 7–24 Transitions illustrating flip-flop operation.

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

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

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

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

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

Asynchronous Preset and Clear Inputs

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

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

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

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

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

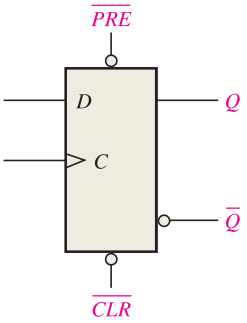


FIGURE 7–25 Logic symbol for a D flip-flop with active-LOW preset and clear inputs.

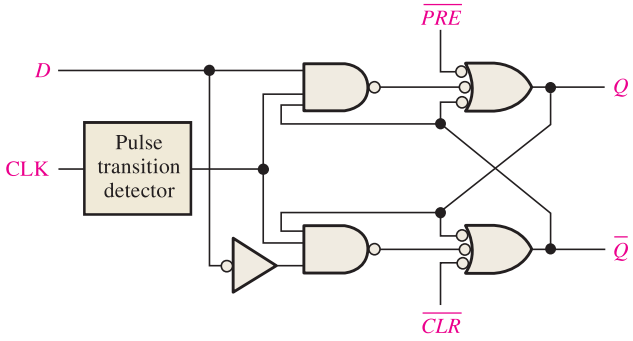
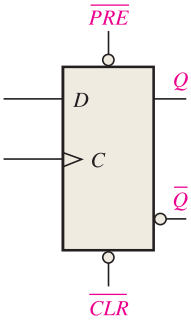
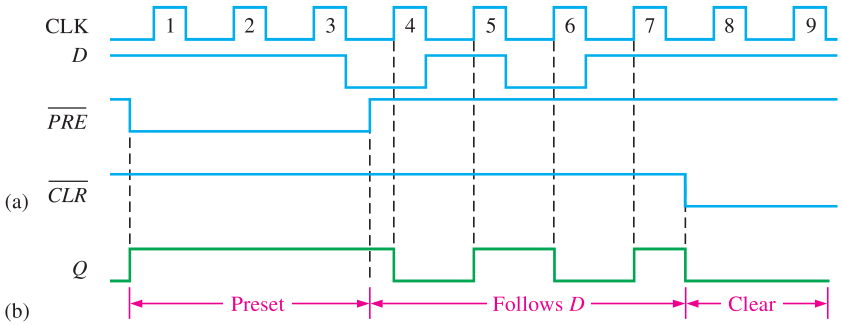


FIGURE 7–26 Logic diagram for a basic D flip-flop with active-LOW preset and clear inputs.

EXAMPLE 7–7

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





MultiSim

FIGURE 7–27 Open file F07-27 to verify the operation.

Solution

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

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

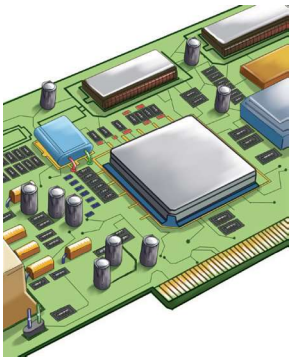
Related Problem

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

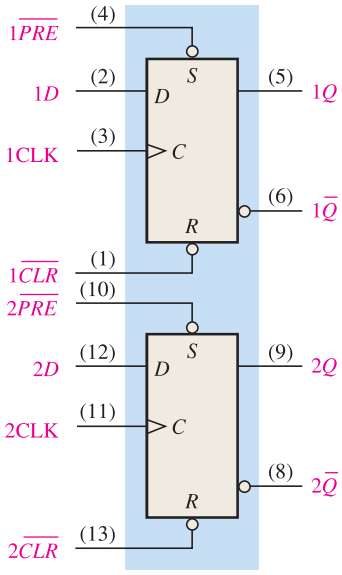
Let's look at two specific edge-triggered flip-flops. They are representative of the various types of flip-flops available in fixed-function IC form and, like most other devices, are available in CMOS and in bipolar (TTL) logic families.

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

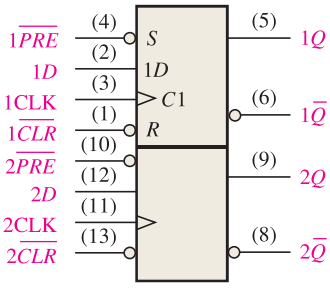
IMPLEMENTATION: D FLIP-FLOP



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



(a) Individual logic symbols



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

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

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

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

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

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

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

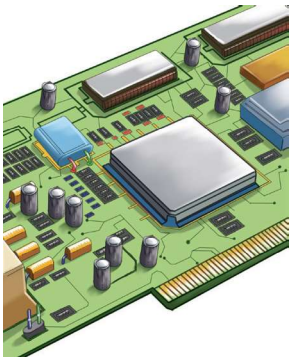
} Check for Preset and Clear conditions

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

Q is set HIGH when Pre input is LOW.

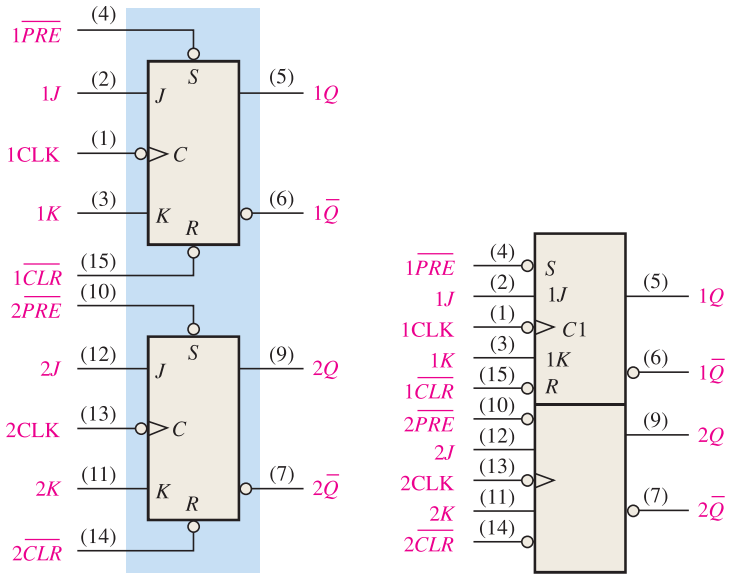
Q is set LOW when Clr input is LOW.

IMPLEMENTATION: J-K FLIP-FLOP



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

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



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

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

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

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

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

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

Inputs and outputs declared

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

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

EXAMPLE 7–8

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

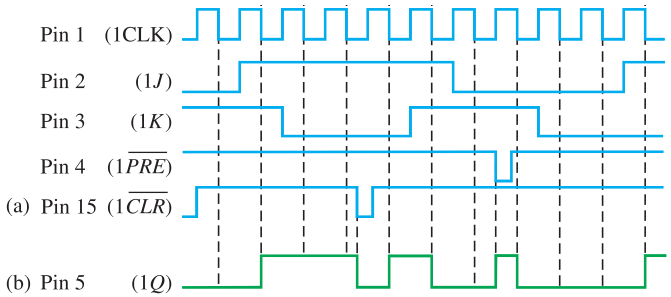


FIGURE 7–30

Solution

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

Related Problem

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

SECTION 7–2 CHECKUP

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

7–3 Flip-Flop Operating Characteristics

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

After completing this section, you should be able to

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

Propagation Delay Times

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

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

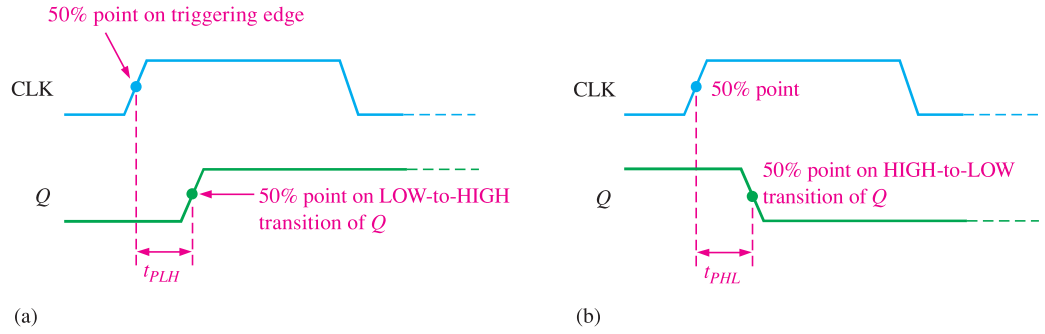


FIGURE 7–31 Propagation delays, clock to output.

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

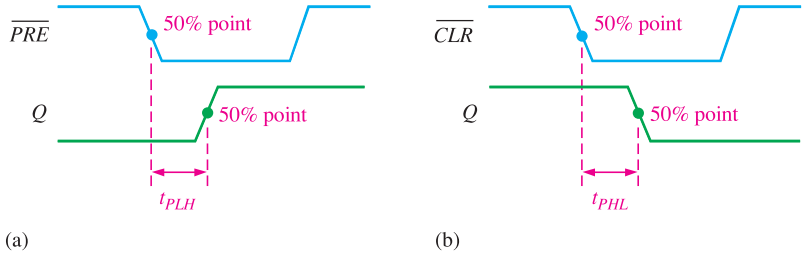


FIGURE 7–32 Propagation delays, preset input to output and clear input to output.

Set-up Time

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

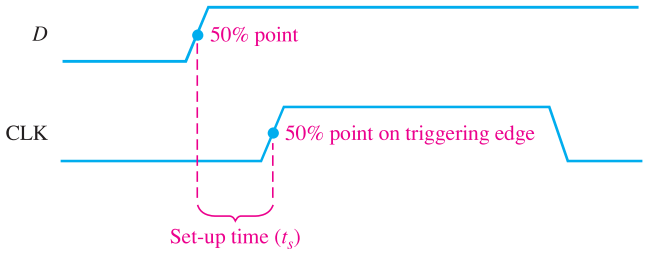


FIGURE 7–33 Set-up time (t_s). The logic level must be present on the *D* input for a time equal to or greater than t_s before the triggering edge of the clock pulse for reliable data entry.

Hold Time

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

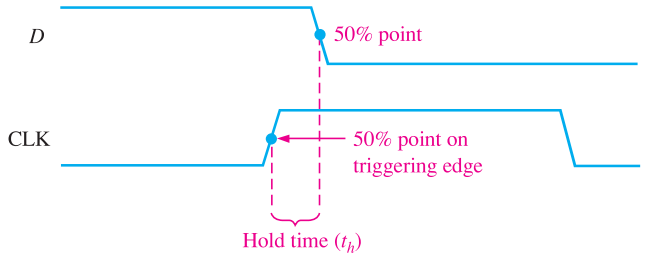


FIGURE 7–34 Hold time (t_h). The logic level must remain on the *D* input for a time equal to or greater than t_h after the triggering edge of the clock pulse for reliable data entry.

Maximum Clock Frequency

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

Pulse Widths

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

Power Dissipation

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

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

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

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

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

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

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

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

Comparison of Specific Flip-Flops

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

TABLE 7–4

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

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

SECTION 7–3 CHECKUP

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

7–4 Flip-Flop Applications

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

After completing this section, you should be able to

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

Parallel Data Storage

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

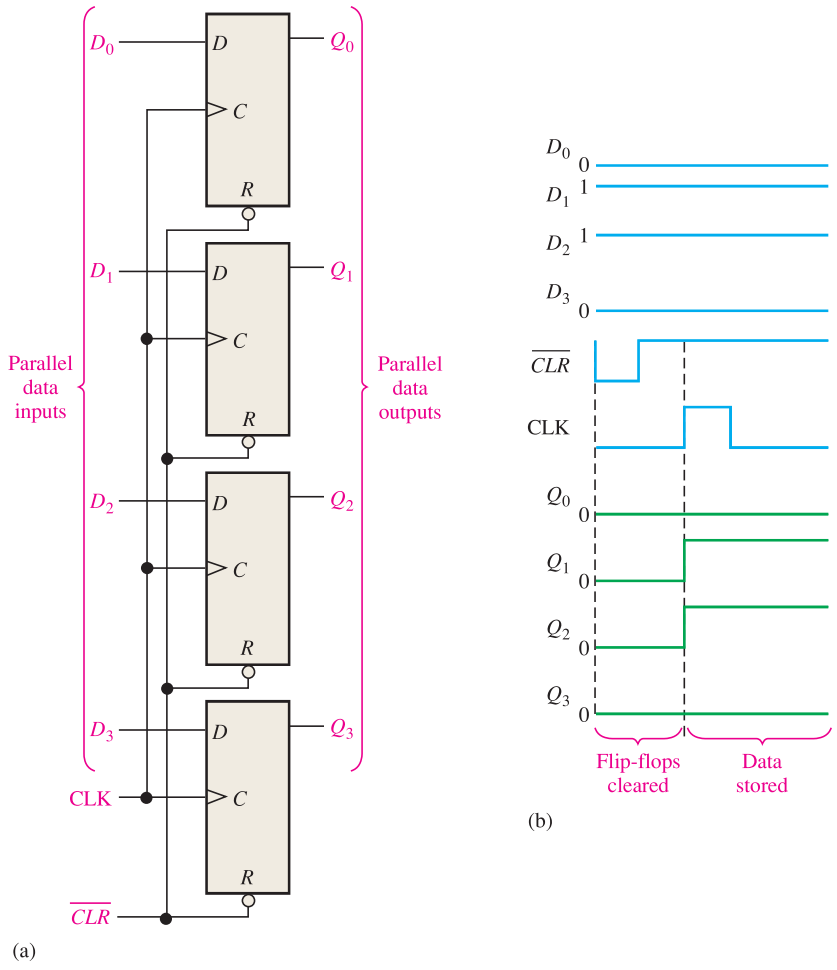


FIGURE 7–35 Example of flip-flops used in a basic register for parallel data storage.

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

Frequency Division

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

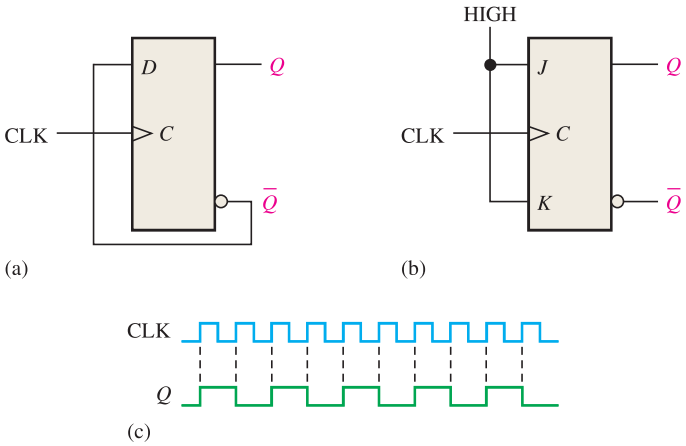


FIGURE 7–36 The D flip-flop and J-K flip-flop as a divide-by-2 device. Q is one-half the frequency of CLK. Open file F07-36 and verify the operation.

MultiSim

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

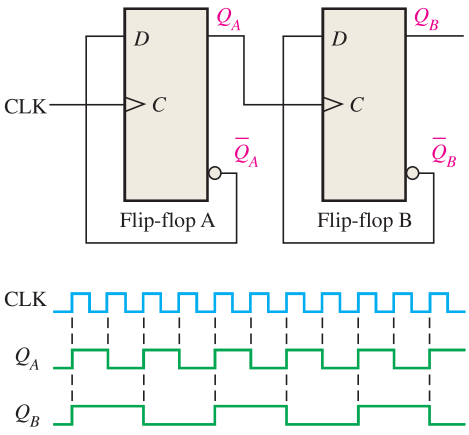


FIGURE 7–37 Example of two D flip-flops used to divide the clock frequency by 4. Q_A is one-half and Q_B is one-fourth the frequency of CLK. Open file F07-37 and verify the operation.

MultiSim

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

EXAMPLE 7–9

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

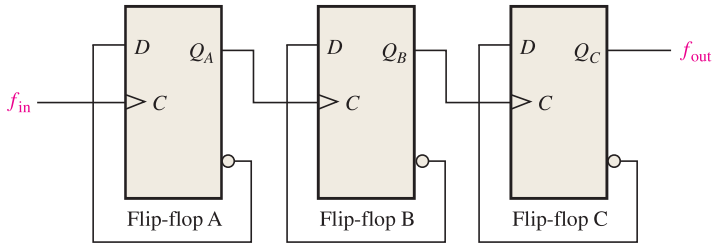


FIGURE 7–38

Solution

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

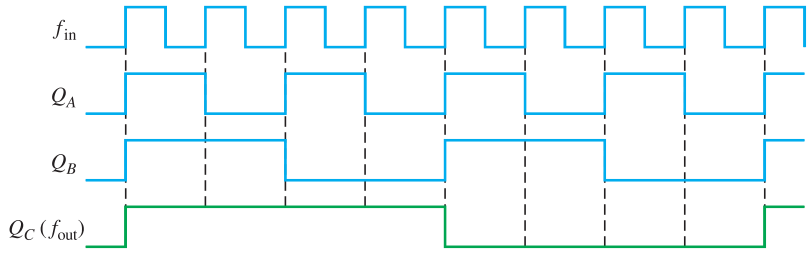


FIGURE 7–39

Related Problem

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

Counting

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

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

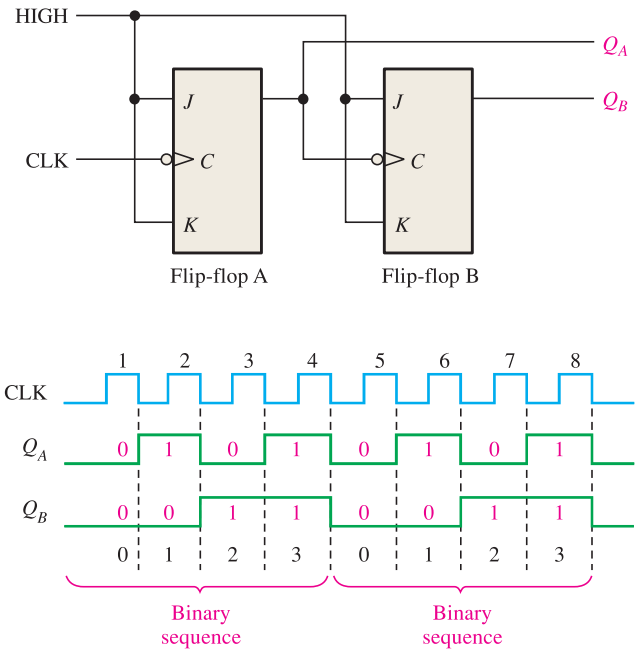


FIGURE 7–40 J-K flip-flops used to generate a binary count sequence (00, 01, 10, 11). Two repetitions are shown.

EXAMPLE 7–10

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

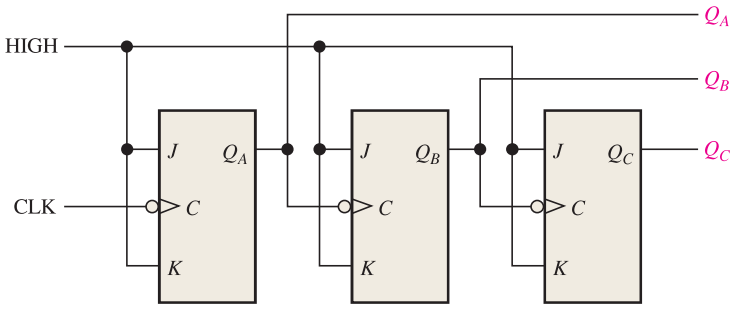


FIGURE 7–41

Solution

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

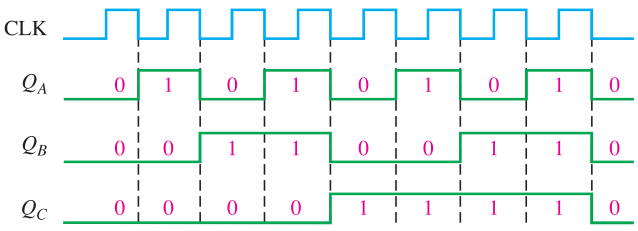


FIGURE 7–42

Related Problem

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

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).

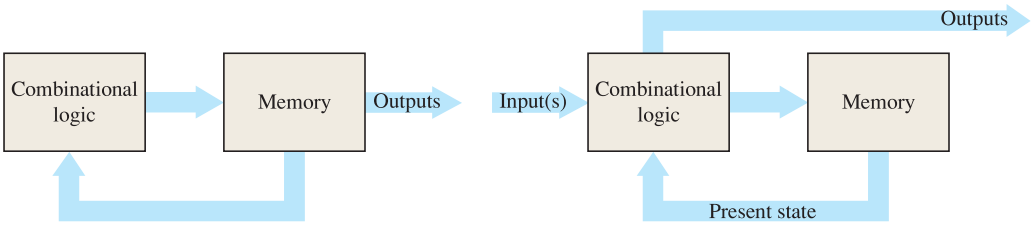
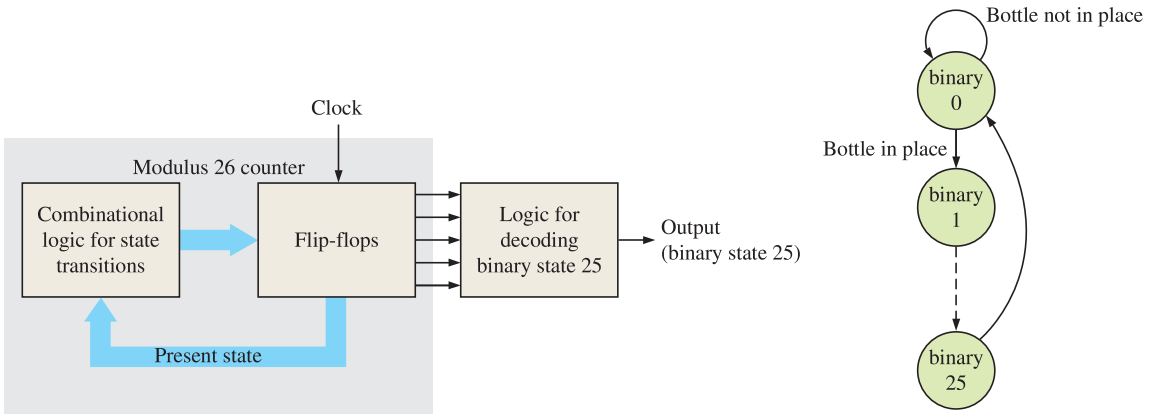


FIGURE 9–1 Two types of sequential logic.

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).

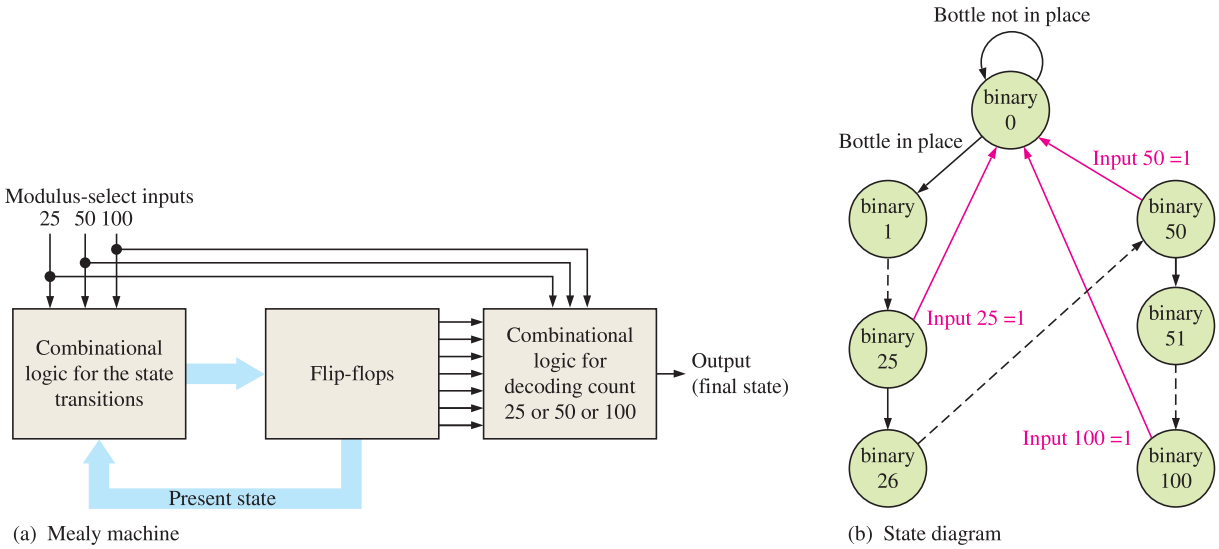


(a) Moore machine (b) State diagram

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.

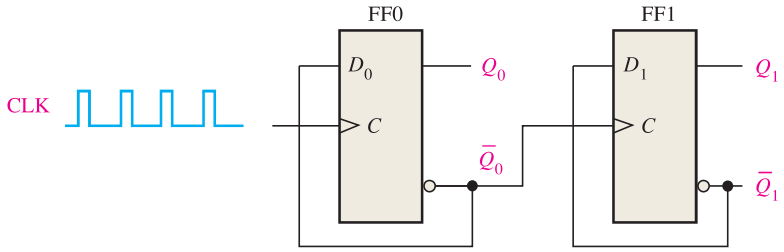
After completing this section, you should be able to

- 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

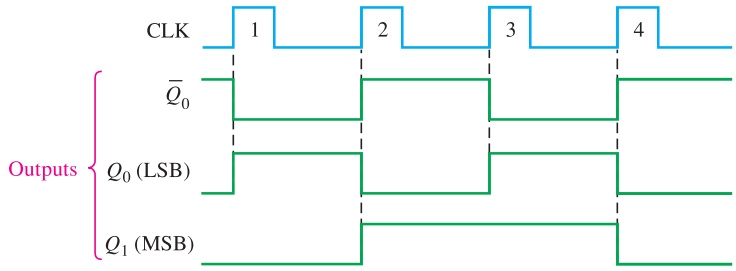


FIGURE 9–5 Timing diagram for the counter of Figure 9–4. As in previous chapters, output waveforms are shown in green.

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.

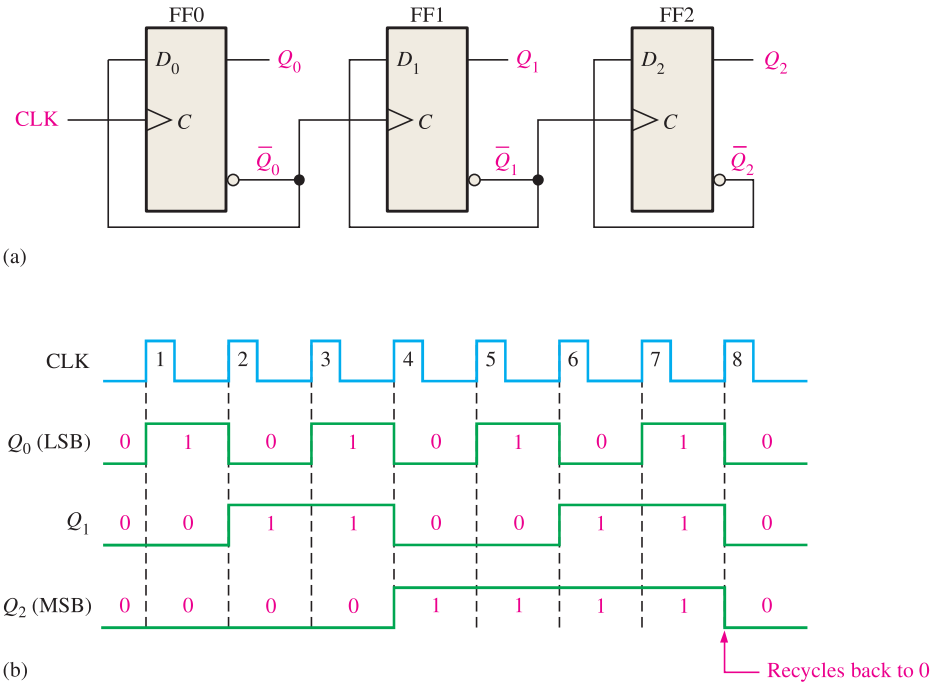
A 3-Bit Asynchronous Binary Counter

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

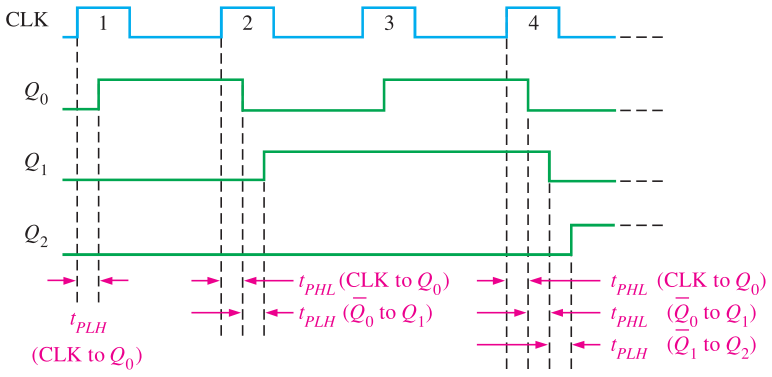


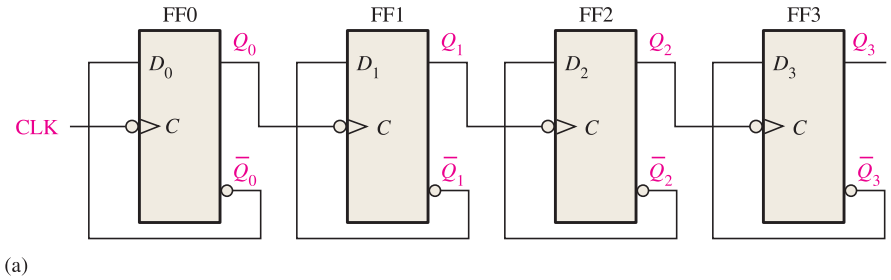
FIGURE 9–7 Propagation delays in a 3-bit asynchronous (ripple-clocked) binary counter.

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.

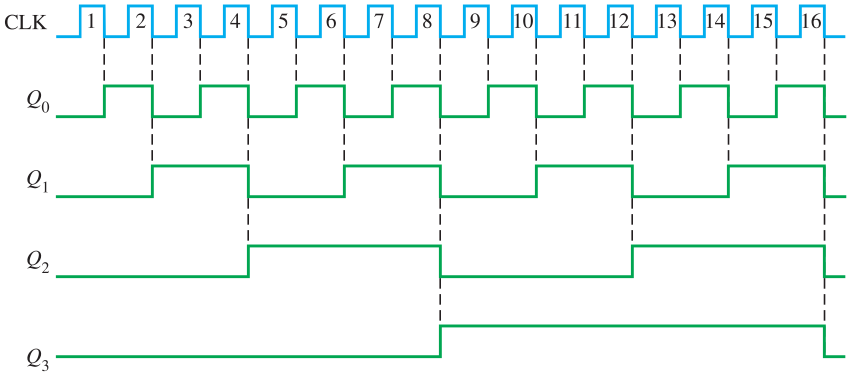
This cumulative delay of an asynchronous counter is a major disadvantage in many applications because it limits the rate at which the counter can be clocked and creates decoding problems. The maximum cumulative delay in a counter must be less than the period of the clock waveform.

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.

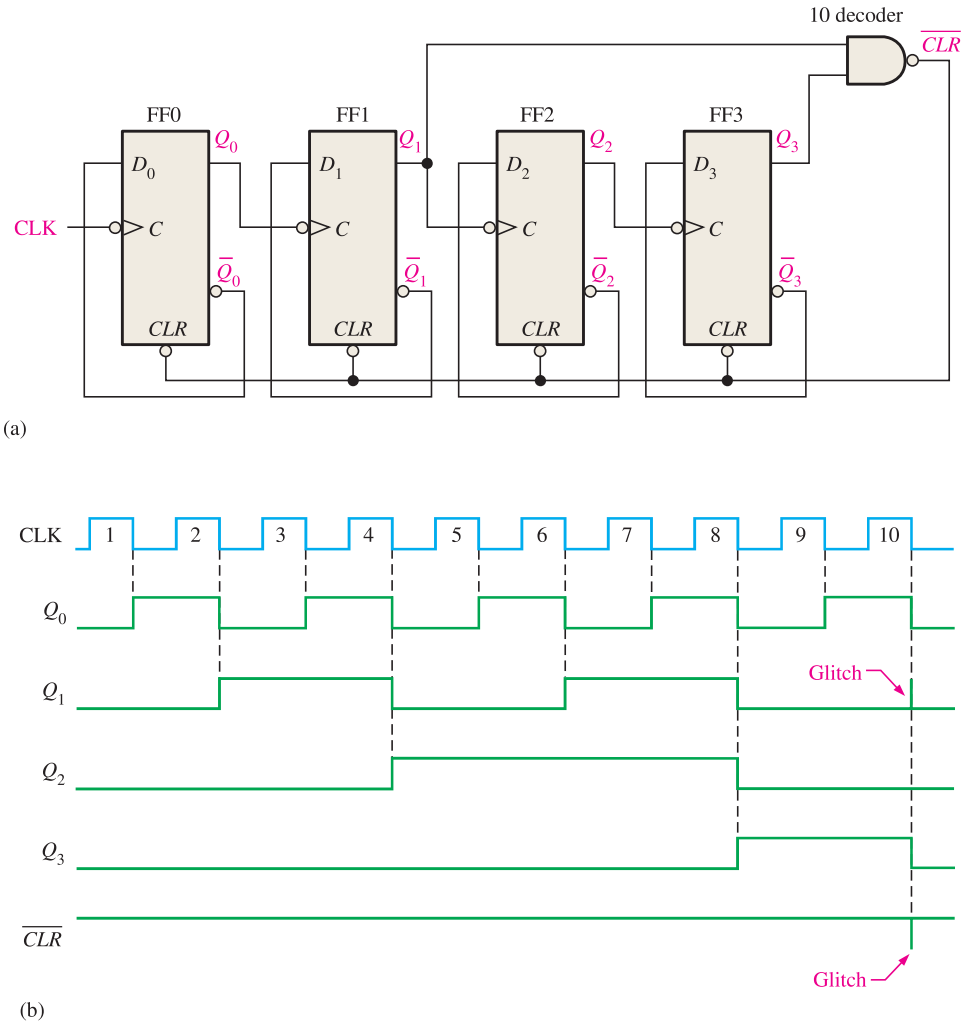


FIGURE 9–9 An asynchronously clocked decade counter with asynchronous recycling.

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 $\overline{CLR}$.)

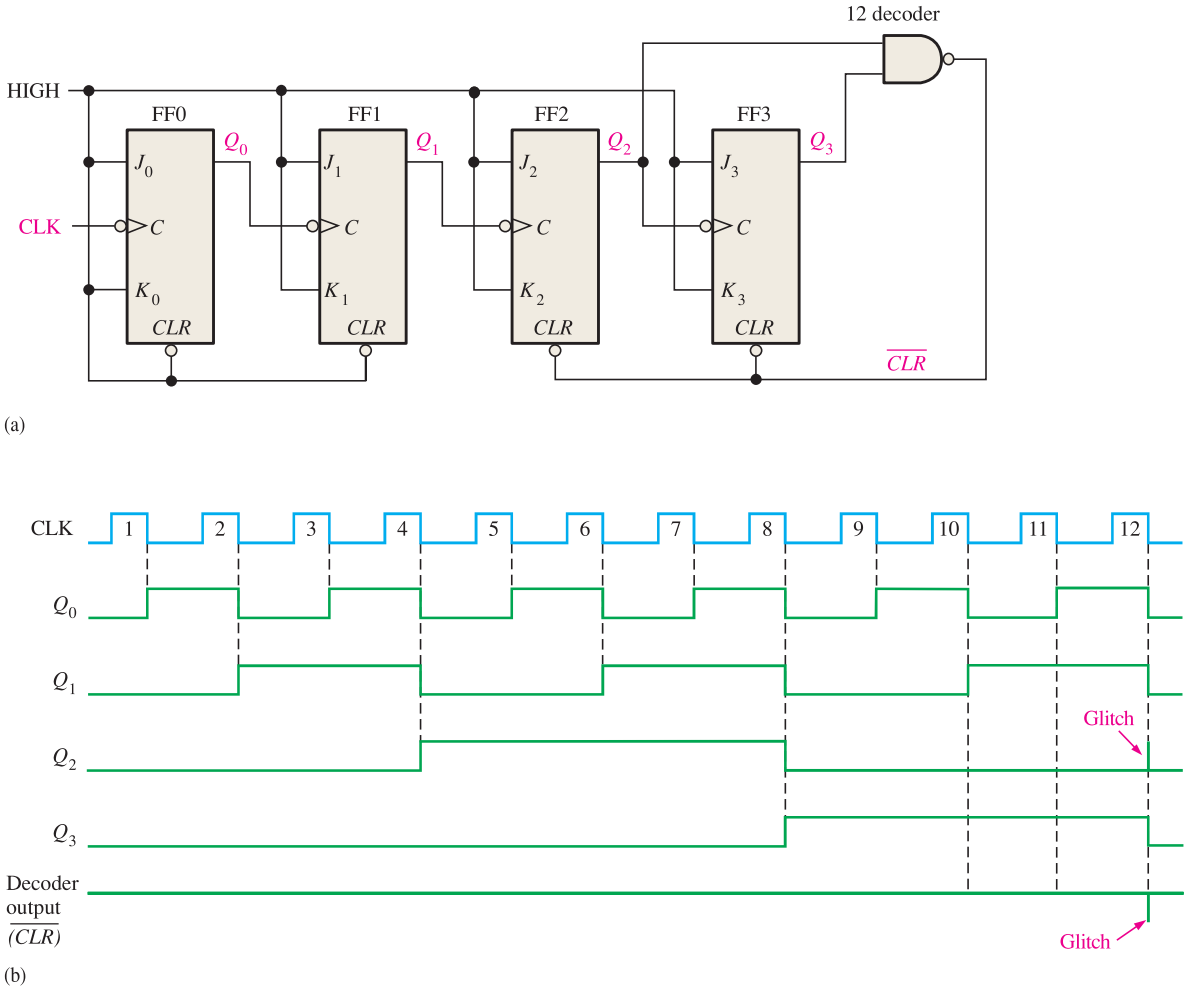
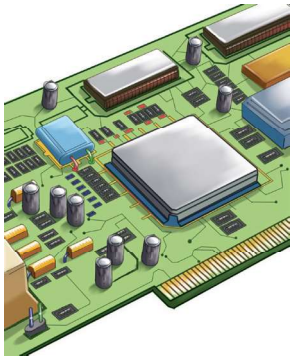


FIGURE 9–10 Asynchronously clocked modulus-12 counter with asynchronous recycling.

Related Problem

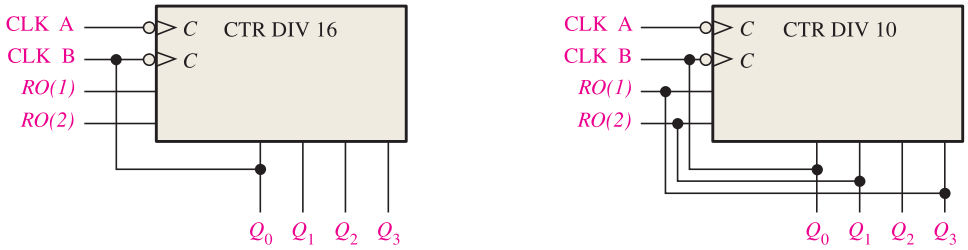
How can the counter in Figure 9–10(a) be modified to make it a modulus-13 counter?

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

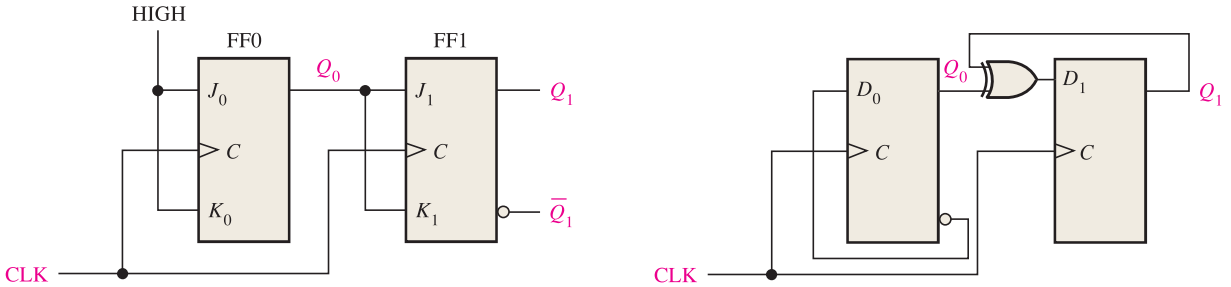
The term **synchronous** refers to events that have a fixed time relationship with each other. A **synchronous counter** is one in which all the flip-flops in the counter are clocked at the same time by a common clock pulse. J-K flip-flops are used to illustrate most synchronous counters. D flip-flops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

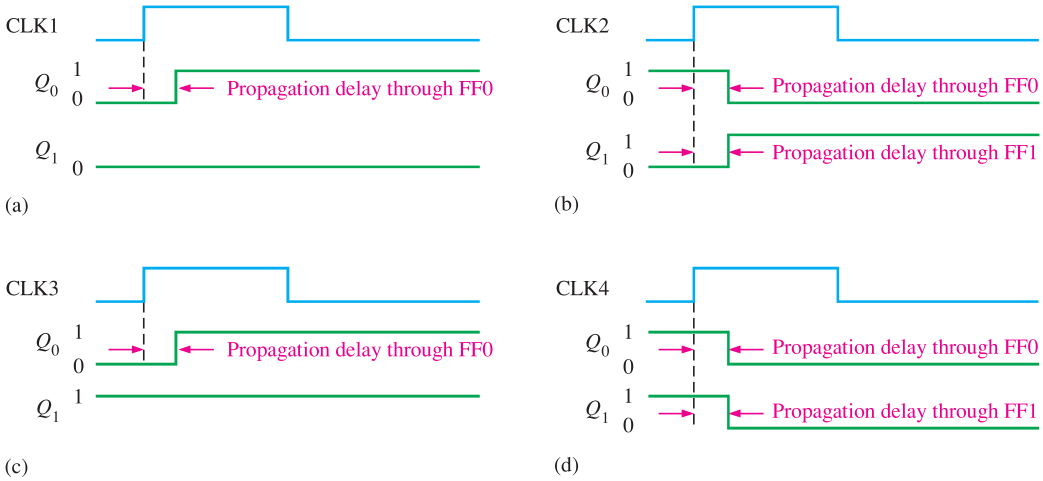


FIGURE 9–13 Timing details for the 2-bit synchronous counter operation (the propagation delays of both flip-flops are assumed to be equal).

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.

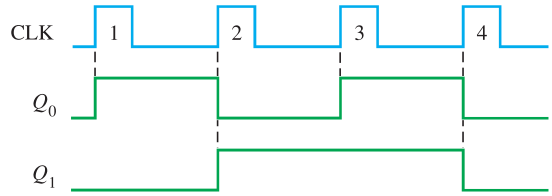
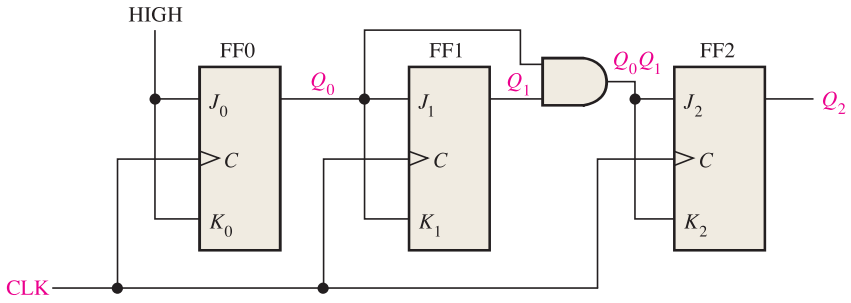


FIGURE 9–14 Timing diagram for the counters of Figure 9–12.

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.

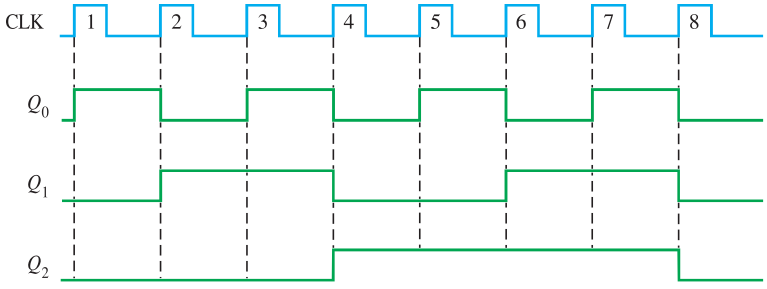


FIGURE 9–16 Timing diagram for the counter of Figure 9–15.

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

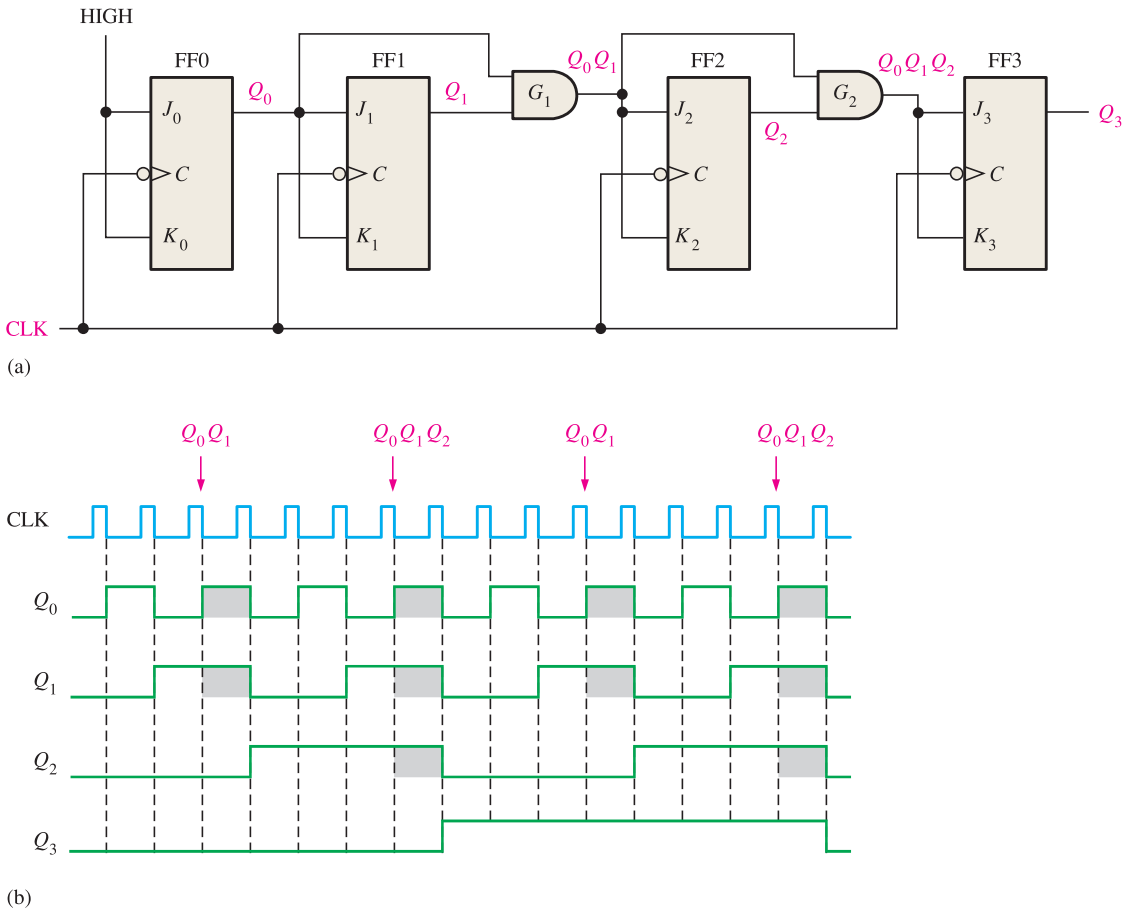


FIGURE 9–17 A 4-bit synchronous binary counter and timing diagram. Times where the AND gate outputs are HIGH are indicated by the shaded areas.

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.

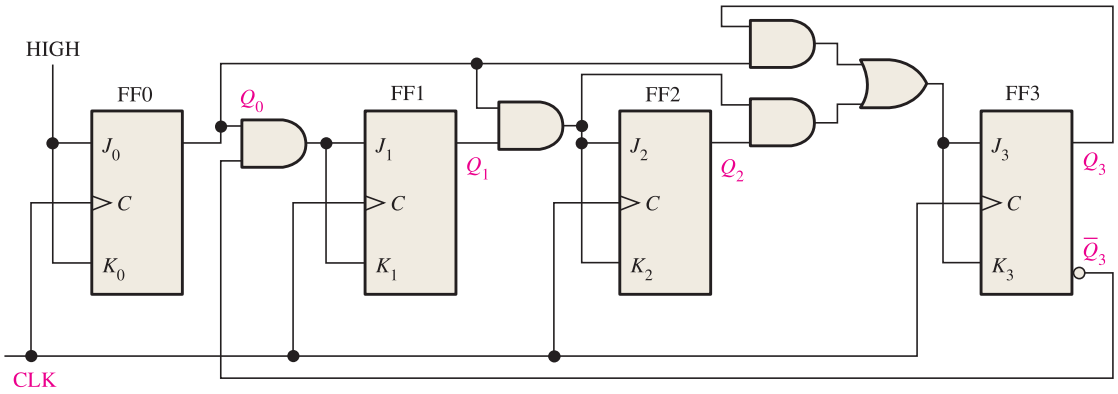


FIGURE 9–18 A synchronous BCD decade counter. Open file F09-18 to verify operation.

MultiSim

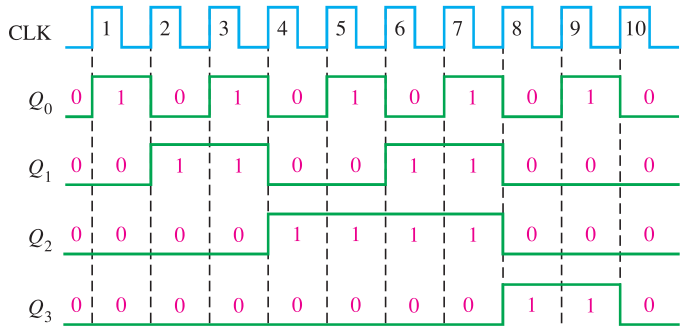


FIGURE 9–19 Timing diagram for the BCD decade counter (Q_0 is the LSB).

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.

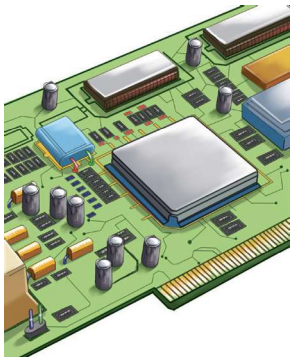
Finally, FF3 (Q_3) changes to the opposite state on the next clock pulse each time $Q_0 = 1$, $Q_1 = 1$, and $Q_2 = 1$ (state 7), or when $Q_0 = 1$ and $Q_3 = 1$ (state 9). The equation for this is as follows:

$$J_3 = K_3 = Q_0Q_1Q_2 + Q_0Q_3$$

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.

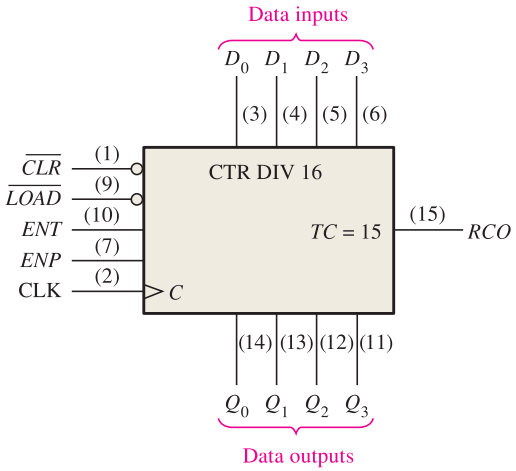


FIGURE 9–20 The 74HC163 4-bit synchronous binary counter. (The qualifying label CTR DIV 16 indicates a counter with sixteen states.)

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

MOI UNIVERSITY
DEPARTMENT OF MATHEMATICS, PHYSICS & COMPUTING
COM 320 CAT II
12TH MAY 2021

i. Using a diagram and a truth table, explain the operations of the following logic devices.
   a. S-R Latch (3 Marks)
   b. D flip flop (3 Marks)
   c. J-K flip flop (4 Marks)

ii. Explain the following flip flop operating characteristics.
   a. Set up time (2 Marks)
   b. Hold time (2 Marks)
   c. Maximum clock frequency. (2 Marks)

iii. Using examples, distinguish between synchronous and asynchronous counters. (4 Marks)