CHAPTER 6

Sequential Logic

6.1 Introduction

The digital circuits considered thus far have been combinational, i.e., the outputs at any instant of time are entirely dependent upon the inputs present at that time. Although every digital system is likely to have combinational circuits, most systems encountered in practice also include memory elements, which require that the system be described in terms of *sequential logic.*

A block diagram of a sequential circuit is shown in Fig. 6-1. It consists of a combinational circuit to which memory elements are connected to form a feedback path. The memory elements are devices capable of storing binary information within them. The binary information stored in the memory elements at any given time defines the *state* of the sequential circuit. The sequential circuit receives binary information from external inputs. These inputs, together with the present state of the memory elements, determine the binary value al the output terminals. They also determine the condition for changing the state in the memory elements. The block diagram demonstrates that the external outputs in a sequential circuit are a function not only of external inputs but also of the present state of the memory elements. The next state of the memory elements is also a function of external inputs and the present state. Thus, a sequential circuit is specified by a time sequence of inputs, outputs, and internal states.

There are two main types of sequential circuits. Their classification depends on the timing of their signals. A *synchronous* sequential circuit is a system whose behavior can be defined from the knowledge of its signals at discrete instants of time. The behavior of an *asynchronous* sequential circuit depends upon the order in which its input signals change and can be affected at any instant of time. The memory elements commonly used in asynchronous sequential circuits are time-delay devices. The memory capability of a time-delay device is due to the fact that it

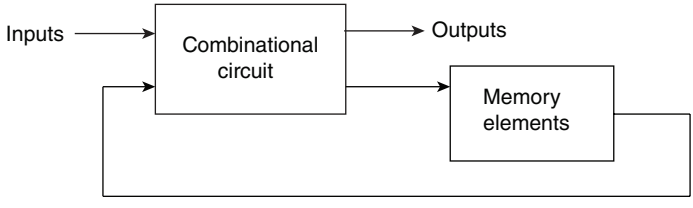


Figure 6.1 Block diagram of a sequential circuit

takes a finite time for the signal to propagate through the device. In practice, the internal propagation delay of logic gates is of sufficient duration to produce the needed delay, so that physical time-delay units may be unnecessary. In gate-type asynchronous systems, the memory elements of Fig. 6-1 consist of logic gates whose propagation delays constitute the required memory. Thus, an asynchronous sequential circuit may be regarded as a combinational circuit with feedback. Because of the feedback among logic gates, an asynchronous sequential circuit may, at times, become unstable. The instability problem imposes many difficulties on the designer. Hence they are not as commonly used as synchronous systems.

A synchronous sequential logic system, by definition, must employ signals that affect the memory elements only at discrete instants of time. One way of achieving this goal is to use pulses of limited duration throughout the system so that one pulse amplitude represents logic-1 and another pulse amplitude (or the absence of a pulse) represents logic-0. The difficulty with a system of pulses is that any two pulses arriving from separate independent sources to the inputs of the same gate will exhibit unpredictable delays, will separate the pulses slightly, and will result in unreliable operation.

Practical synchronous sequential logic systems use fixed amplitudes such as voltage levels for the binary signals. Synchronization is achieved by a timing device called a *master-clock generator* which generates a periodic train of *clock pulses.* The clock pulses are distributed throughout the system in such a way that memory elements are affected only with the arrival of the synchronization pulse. In practice, the clock pulses are applied into AND gates together with the signals that specify the required change in memory elements. The AND gate outputs can transmit signals only at instants which coincide with the arrival of clock pulses. Synchronous sequential circuits that use clock pulses in the inputs of memory elements are called *clocked sequential circuits.* Clocked sequential circuits are the type encountered most frequently. They do not manifest instability problems and their timing is easily broken down into independent discrete steps, each of which is considered separately. The sequential circuits discussed in this book are exclusively of the clocked type.

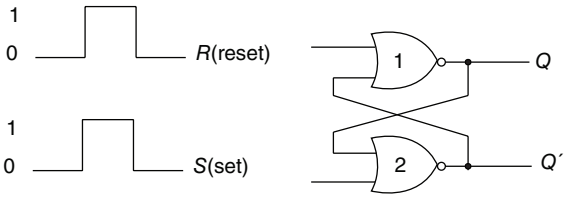
The memory elements used in clocked sequential circuits are called *flip-flops.* These circuits are binary cells capable of storing one bit of information. A flip-flop circuit has two outputs, one for the normal value and one for the complement value of the bit stored in it. Binary information can enter a flip-flop in a variety of ways, a fact which gives rise to different types of flip-flops. In the next section we examine the various types of flip-flops and define their logical properties.

6.2 Flip-Flops

A flip-Hop circuit can maintain a binary state indefinitely (as long as power is delivered to the circuit) until directed by an input signal to switch states. The major differences among various types of flip-flops are in the number of inputs they possess and m the manner in which the inputs affect the binary state. The most common types of flip-flops are discussed below,

6.2.1 Basic Flip-Flop Circuit

It was mentioned in Sections 4-7 and 4-8 that a flip-flop circuit can be constructed from two NAND gates or two NOR gates. These constructions are shown in the logic diagrams of Figs. 6-2 and 6-3. Each circuit forms a basic flip-flop upon which other more complicated types can be built. The cross-coupled connection from the output of one gate to the input of the other gate constitutes a feedback path. For this reason, the circuits are classified as asynchronous sequential



(a) Logic diragram

S	*R*	*Q*	*Q´*	
1	0	1	0	
0	0	1	0	(after *S* = 1, *R* = 0)
0	1	0	1	
0	0	0	1	(after *S* = 0, *R* = 1)
1	1	0	0	

(b) Truth table

Figure 6.2 Basic flip–flop circuit with NOR gates

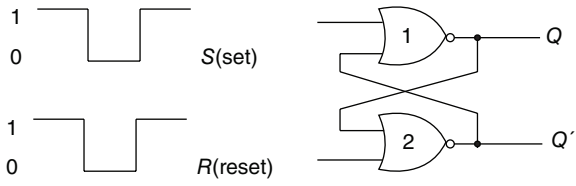
circuits. Each flip-flop has two outputs, Q and Q', and two inputs, *set* and *reset.* This type of flip-flop is sometimes called a *direct-coupled RS* flip-flop or *SR latch.* The *R* and *S* are the first letters of the two input names.

To analyze the operation of the circuit of Fig, 6-2, we must remember that the output of a NOR gate is 0 if any input is 1, and that the output is 1 only when all inputs are 0. As a starting point, assume that the set input is 1 and the reset input is 0. Since gate 2 has an input of 1, its output Q' must be 0, which puts both inputs of gate 1 at 0, so that output Q is 1. When the set input is returned to 0, the outputs remain the same, because output Q remains a 1, leaving one input of gate 2 at 1. That causes output Q' to stay at 0, which leaves both inputs of gate number 1 at 0, so that output Q is a 1. In the same manner it is possible to show that a 1 in the reset input changes output Q to 0 and Q' to 1. When the reset input returns to 0, the outputs do not change.

When a 1 is applied to both the set and the reset inputs, both Q and Q' outputs go to 0. This condition violates the fact that outputs Q and Q' are the complements of each other. In normal operation this condition must be avoided by making sure that l's are not applied to both inputs simultaneously.

A flip-flop has two useful states. When $Q = 1$ and $Q' = 0$, it is in the *set state* (or I-state). When $Q = 0$ and $Q' = 1$, it is in the *clear state* (or 0-state). The outputs Q and Q' are complements of each other and are referred to as the normal and complement outputs, respectively. The binary state of the flip-flop is taken to be the value of the normal output.

Under normal operation, both inputs remain at 0 unless the state of the flip-flop has to be changed. The application of a momentary 1 to the set input causes the flip-flop to go to the set state. The set input must go back to 0 before a 1 is applied to the reset input. A momentary 1 applied to the reset input causes the flip-flop to go the clear state. When both inputs are initially 0, a 1 applied to the set input while the flip-flop is in the set state or a 1 applied to the reset input while the flip-flop is in the clear state leaves the outputs unchanged. When a 1 is applied to both



(a) Logic diagram

S	*R*	*Q*	*Q´*	
1	0	0	1	
1	1	0	1	(after *S* = 1, *R* = 0)
0	1	1	0	
1	1	1	0	(after *S* = 0, *R* = 1)
0	0	1	1	

(b) Truth table

Figure 6.3 Basic flip – flop circuit with NAND gates

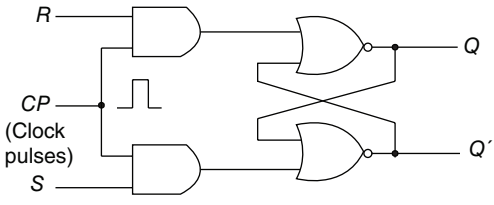
the set and the reset inputs, both outputs go to 0. This state is undefined and is usually avoided. If both inputs now go to 0, the state of the flip-flop is indeterminate and depends on which input remains a 1 longer before the transition to 0.

The NAND basic flip-flop circuit of Fig, 6-3 operates with both inputs normally at 1 unless the state of the flip-flop has to be changed. The application of a momentary 0 to the set input causes output Q to go to 1 and Q' to go to 0, thus putting the flip-flop into the set state. After the set input returns to 1, a momentary 0 to the reset input causes a transition to the clear state. When both inputs go to 0, both outputs go to 1—a condition avoided in normal flip-flop operation.

6.2.2 Clocked *RS* Flip-Flop

The basic flip-flop as it stands is an asynchronous sequential circuit. By adding gates to the inputs of the basic circuit, the flip-flop can be made to respond to input levels during the occurrence of a clock pulse. The clocked *RS* flip-flop shown in Fig. 6-4(a) consists of a basic NOR flip-flop and two AND gates. The outputs of the two AND gates remain at 0 as long as the clock pulse (abbreviated *CP*) is 0, regardless of the *S* and *R* input values. When the clock pulse goes to 1, information from the *S* and *R* inputs is allowed to reach the basic flip-flop. The set state is reached with $S = 1$, $R = 0$. and $CP = 1$. To change to the clear state, the inputs must be $S = 0$, $R = 1$, and $CP = 1$. With both $S = 1$ and $R = 1$, the occurrence of a clock pulse causes both outputs to momentarily go to 0. When the pulse is removed, the state of the flip-flop is indeterminate, i.e., either state may result, depending on whether the set or the reset input of the basic flip-flop remains a 1 longer before the transition to 0 at the end of the pulse.

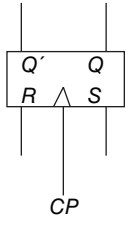
The graphic symbol for the clocked *RS* flip-flop is shown in Fig, 6-4(b). It has three inputs: *S, R,* and *CP.* The *CP* input is not written within the box because it is recognized from the marked



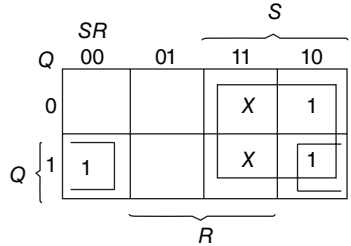
(a) Logic diagram

Q	S	R	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	indeterminate
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	indeterminate

(c) Characteristic table



(b) Graphic symbol



(d) Characteristic equation

Figure 6.4 Clocked RS flip-flop

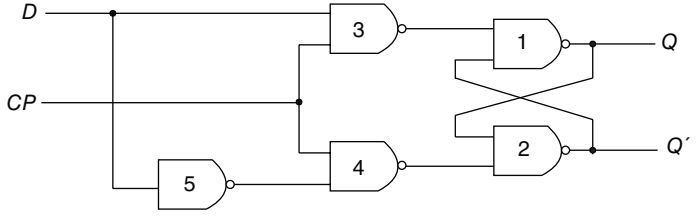
small triangle. The triangle is a symbol for a *dynamic indicator* and denotes the fact that the flip-flop responds to an input clock *transition* from a low-level (binary 0) to a high-level (binary 1) signal. The outputs of the flip-flop are marked with Q and Q' *within* the box. The flip-flop can be assigned a different variable name even though Q, is written inside the box. In that case the letter chosen for the flip-flop variable is marked *outside* the box along the output line. The state of the flip-flop is determined from the value of its normal output Q. If one wishes to obtain the complement of the normal output, it is not necessary to insert an inverter, because the complemented value is available directly from output Q'.

The characteristic table for the flip-flop is shown in Fig. 6-4(c). This table summarizes the operation of the flip-flop in a tabular form. Q is the binary state of the flip-flop at a given time (referred to as *present state*), the S and R columns give the possible values of the inputs, and $Q(t + 1)$ is the state of the flip-flop after the occurrence of a clock pulse (referred to as *next state*).

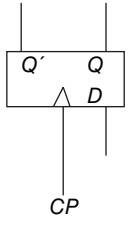
The characteristic equation of the flip-flop is derived in the map of Fig. 6-4(d). This equation specifies the value of the next state as a function of the present state and the inputs. The characteristic equation is an algebraic expression for the binary information of the characteristic table. The two indeterminate states are marked by X's in the map, since they may result in either a 1 or a 0. However, the relation $SR = 0$ must be included as pan of the characteristic equation to specify that both S and R cannot equal 1 simultaneously.

6.2.3 *D* Flip-Flop

The *D* flip-flop shown in Fig. 6-5 is a modification of the clocked *RS* flip-flop. NAND gates 1 and 2 form a basic flip-flop and gates 3 and 4 modify it into a clocked *RS* flip-flop. The *D* input goes directly to the *S* input, and its complement, through gate 5, is applied to the *R* input. As long as the clock pulse input is at 0, gates 3 and 4 have a 1 in their outputs, regardless of the value of the other inputs. This conforms to the requirement that the two inputs of a basic NAND flip-flop



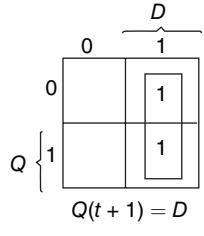
(a) Logic diagram with NAND gates



(b) Graphic symbol

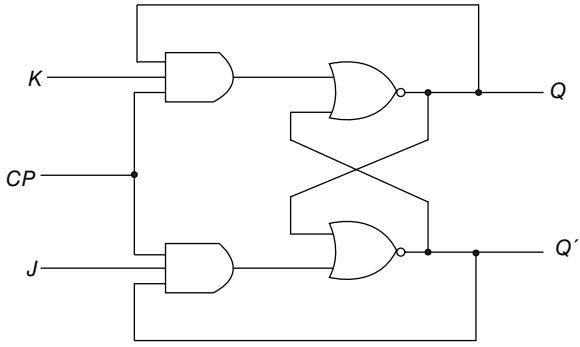
Q	D	Q(t + 1)
0	0	0
0	1	1
1	0	0
1	1	1

(c) Characteristic table

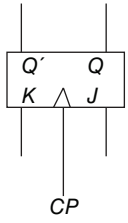


(d) Characteristic equation

Figure 6.5 Clocked *D* flip-flop



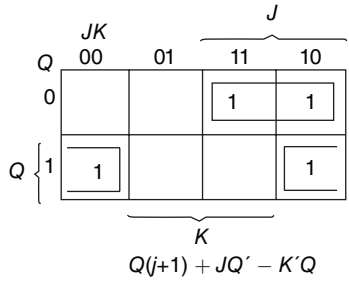
(a) Logic diagram



(b) Graphic symbol

Q	J	K	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

(c) Characteristic table



$Q(j+1) + JQ' - K'Q$

(d) Characteristic equation

Figure 6.6 Clocked *JK* flip–flop

(Fig. 6-3) remain initially at the 1 level. The *D* input is sampled during the occurrence of a clock pulse, if it is 1, the output of gate 3 goes to 0, switching the flip-flop to the set state (unless it was already set). If it is 0, the output of gate 4 goes to 0, switching the flip-flop to the clear state.

The *D* flip-flop receives the designation from its ability to transfer "data" into a flip-flop. It is basically an *RS* flip-flop with an inverter in the *R* input. The added inverter reduces the number of inputs from two to one. This type of flip-flop is sometimes called a *gated D-latch.* The *CP* input is often given the variable designation *G* (*for gate*) to indicate that this input enables the gated latch to make possible the data entry into the flip-flop.

The symbol for a clocked *D* flip-flop is shown in Fig. 6-5(b). The characteristic table is listed in part (c) and the characteristic equation is derived in part (d), The characteristic equation shows that the next state of the flip-flop is the same as the *D* input and is independent of the value of the present state.

6.2.4 *JK* Flip-Flop

A *JK* flip-flop is a refinement of the *RS* flip-flop in that the indeterminate state of the *RS* type is defined in the *JK* type. Inputs *J* and *K* behave like inputs *S* and *R* to set and clear the flip-flop (note that in a *JK* flip-flop, the letter *J* is for *set* and the letter *K* is for *clear*). When inputs are

applied to both J and K simultaneously, the flip-flop switches to its complement state, that is, if $Q = 1$, it switches to $Q = 0$, and vice versa.

A clocked JK flip-flop is shown in Fig. 6-6(a). Output Q is ANDed with K and CP inputs so that the flip-flop is cleared during a clock pulse only if Q was previously 1. Similarly, output Q' is ANDed with J and CP inputs so that the flip-flop is set with a clock pulse only if Q' was previously I.

As shown in the characteristic table in Fig. 6-6(c), the JK flip-flop behaves like an RS flip-flop, except when both J and K are equal to 1. When both J and K are 1, the clock pulse is transmitted through one AND gate only—the one whose input is connected to the flip-flop output which is presently equal to 1. Thus, if $Q = 1$, the output of the upper AND gate becomes 1 upon application of a clock pulse, and the flip-flop is cleared. If $Q' = 1$, the output of the lower AND gate becomes a 1 and the flip-flop is set. In either case, the output state of the flip-flop is complemented.

The inputs in the graphic symbol for the JK flip-flop must be marked with a J (under Q) and K (under (Q'). The characteristic equation is given in Fig. 6-4(d) and is derived from the map of the characteristic table.

Note that because of the feedback connection in the JK flip-flop, a CP signal which remains a 1 (while $J = K = 1$) after the outputs have been complemented once will cause repeated and continuous transitions of the outputs. To avoid this undesirable operation, the clock pulses must have a time duration which is shorter than the propagation delay through the flip-flop. This is a restrictive requirement, since the operation of the circuit depends on the width of the pulses. For this reason, JK flip-flops are never constructed as shown in Fig, 6-6(a). The restriction on the pulse width can be eliminated with a master-slave or edge-triggered construction, as discussed in the next section. The same reasoning applies to the T flip-flop presented below.

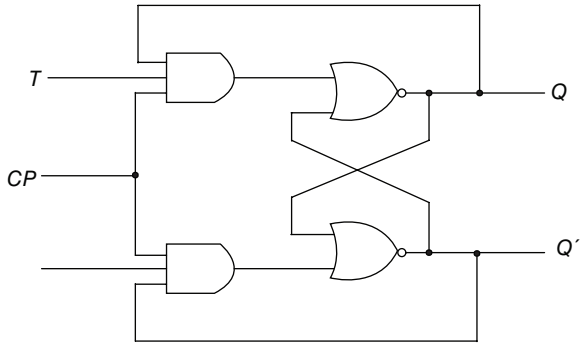
The T flip-flop is a single-input version of the JK flip-flop. As shown in Fig. 6-7(a), the T flip-flop is obtained from a JK type if both inputs are tied together. The designation T comes from the ability of the flip-flop to "toggle," or change state. Regardless of the present state of the flip-flop, it assumes the complement state when the clock pulse occurs while input T is logic-1. The symbol, characteristic table, and characteristic equation of the T flip-flop are shown in Fig. 6-7, parts (b), (c), and (d), respectively.

The flip-flops introduced in this section are the most common types available commercially. The analysis and design procedures developed in this chapter are applicable for any clocked flip-flop once its characteristic table is defined.

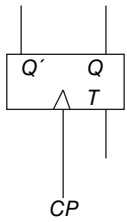
6.3 Triggering of Flip-flops

The state of a flip-flop is switched by a momentary change in the input signal. This momentary change is called a *trigger* and the transition it causes is said to trigger the flip-flop. Asynchronous flip-flops, such as the basic circuits of Figs. 6-2 and 6-3, require an input trigger defined by a change of signal *level.* This level must be returned to its initial value (0 in the NOR and 1 in the NAND flip-flop) before a second trigger is applied. Clocked flip-flops are triggered *by pulses.* A pulse starts from an initial value of 0, goes momentarily to 1, and after a short time, returns to its initial 0 value. The time interval from the application of the pulse until the output transition occurs is a critical factor that needs further investigation.

As seen from the block diagram of Fig. 6-1, a sequential circuit has a feedback path between the combinational circuit and the memory elements. This path can produce instability if



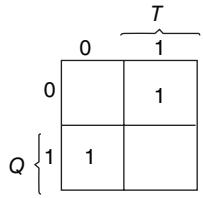
(a) Logic diagram



(b) Graphic symbol

Q	T	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

(c) Characteristic table



$Q(t-1) = TQ' - T'Q$

(d) Characteristic equation

Figure 6.7 Clocked *T* flip-flop

the outputs of memory elements (flip-flops) are changing while the outputs of the combinational circuit that go to flip-flop inputs are being sampled by the clock pulse. This timing problem can be prevented if the outputs of flip-flops do not start changing until the pulse input has returned to 0. To ensure such an operation, a flip-flop must have a signal propagation delay from input to output in excess of the pulse duration. This delay is usually very difficult to control if the designer depends entirely on the propagation delay of logic gates. One way of ensuring the proper delay is to include within the flip-flop circuit a physical delay unit having a delay equal to or greater than the pulse duration. A better way to solve the feedback timing problem is to make the flip-flop sensitive to the pulse *transition* rather than the pulse duration.

A clock pulse may be either positive or negative. A positive clock source remains at 0 during the interval between pulses and goes to 1 during the occurrence of a pulse. The pulse goes through two signal transitions; from 0 to 1 and the return from 1 to 0. As shown in Fig. 6-8, the positive transition is defined as the *positive edge* and the negative transition as the *negative edge.* This definition applies also to negative pulses.

The clocked flip-flops introduced in Section 6-2 are triggered during the positive edge of the pulse, and the state transition starts as soon as the pulse reaches the logic-1 level. The new state of the flip-flop may appear at the output terminals while the input pulse is still 1. If the other inputs of the flip-flop change while the clock is still 1, the flip-flop will start responding to these new values and a new output state may occur. When this happens, the output of one flip-flop cannot be applied to the inputs of another flip-flop when both are triggered by the same clock

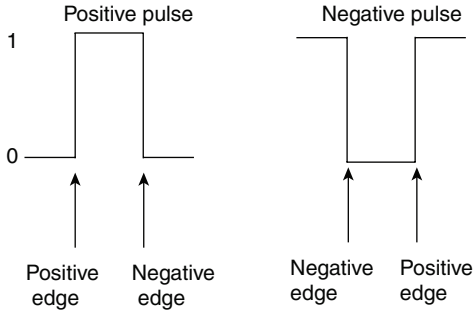


Figure 6.8 Definition of clock pulse transition

pulse. However, if we can make the flip-flop respond to the positive (or negative) edge transition *only,* instead of the entire pulse duration, then the multiple-transition problem can be eliminated.

One way to make the flip-flop respond only to a pulse transition is to use capacitive coupling. In this configuration, an *RC* (resistor-capacitor) circuit is inserted in the clock input of the flip-flop. This circuit generates a spike in response to a momentary change of input signal. A positive edge emerges from such a circuit with a positive spike, and a negative edge emerges with a negative spike. Edge triggering is achieved by designing the flip-flop to neglect one spike and trigger on the occurrence of the other spike. Another way to achieve edge triggering is to use a master-slave or edge-triggered flip-flop as discussed below.

6.3.1 Master-Slave Flip-Flop

A master-slave flip-flop is constructed from two separate flip-flops. One circuit serves as a master and the other as a slave, and the overall circuit is referred to as a *master-stave flip-flop.* The logic diagram of an *RS* master-slave flip-flop is shown in Fig. 6-9. It consists of a master flip-flop, a slave flip-flop, and an inverter. When clock pulse *CP* is 0, the output of the inverter is 1. Since the

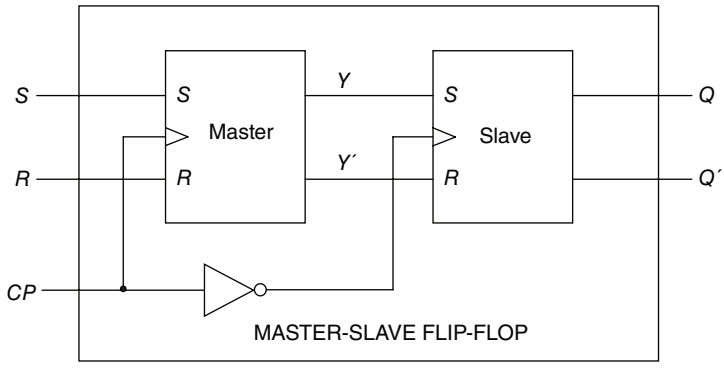


Figure 6.9 Logic diagram of master-slave flip-flop

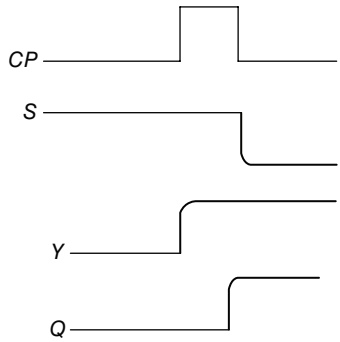


Figure 6.10 Timing relationships in a master-slave flip-flop

clock input of the slave is 1, the flip-flop is enabled and output Q is equal to Y, while Q' is equal to Y'. The master flip-flop is disabled because $CP = 0$. When the pulse becomes 1, the information then at the external R and S inputs is transmitted to the master flip-flop. The slave flip-flop, however, is isolated as long as the pulse is at its 1 level, because the output of the inverter is 0. When the pulse returns to 0, the master flip-flop is isolated, which prevents the external inputs from affecting it. The slave flip-flop then goes to the same state as the master flip-flop.

The timing relationships shown in Fig. 6-10 illustrate the sequence of events that occur in a master-slave flip-flop. Assume that the flip-flop is in the clear state prior to the occurrence of a pulse, so that $Y = 0$ and $Q = 0$. The input conditions are $S = 1$, $R = 0$, and the next clock pulse should change the flip-flop to the set state with $Q = 1$. During the pulse transition from 0 to 1, the master flip-flop is set and changes Y to 1. The slave flip-flop is not affected because its CP input is 0. Since the master flip-flop is an internal circuit, its change of state is not noticeable in the outputs Q and Q'. When the pulse returns to 0, the information from the master is allowed to pass through to the slave, making the external output $Q = 1$. Note that the external S input can be changed at the same time that the pulse goes through its negative edge transition. This is because, once the CP reaches 0, the master is disabled and its R and S inputs have no influence until the next clock pulse occurs. Thus, in a master-slave flip-flop, it is possible to switch the output of the flip-flop and its input information with the same clock pulse. It must be realized that the S input could come from the output of another master-slave flip-flop that was switched with the same clock pulse.

The behavior of the master-slave flip-flop just described dictates that the state changes in all flip-flops coincide with the negative edge transition of the pulse. However, some IC master-slave flip-flops change output states in the positive edge transition of clock pulses. This happens in flip-flops that have an additional inverter between the CP terminal and the input of the master. Such flip-flops are triggered with negative pulses (see Fig. 6-8), so that the negative edge of the pulse affects the master and the positive edge affects the slave and the output terminals.

The master-slave combination can be constructed for any type of flip-flop by adding a clocked RS flip-flop with an inverted clock to form the slave. An example of a master-slave JK flip-flop constructed with NAND gates is shown in Fig. 6-11. It consists of two flip-flops; gates 1 through 4 form the master flip-flop, and gates 5 through 8 form the slave flip-flop. The infor-

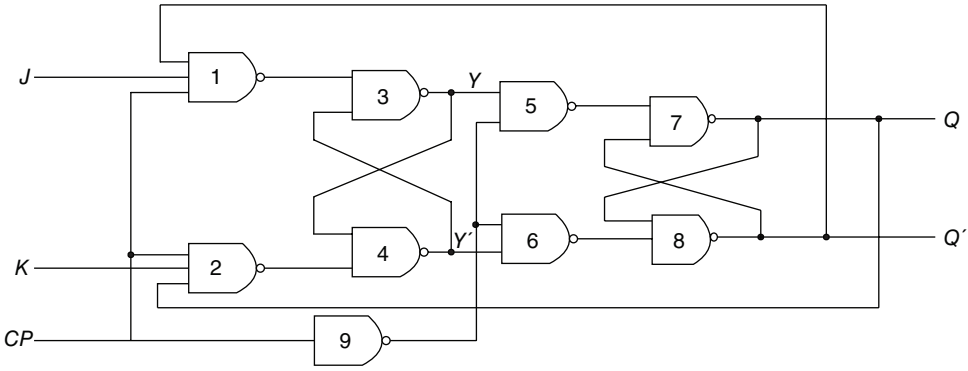


Figure 6.11 Clocked master-slave *JK* flip-flop

mation present at the *J* and *K* inputs is transmitted to the master flip-flop on the positive edge of a clock pulse and is held there until the negative edge of the clock pulse occurs, after which it is allowed to pass through to the slave flip-flop. The clock input is normally 0, which keeps the outputs of gates 1 and 2 at the 1 level. This prevents the *J* and *K* inputs from affecting the master flip-flop. The slave flip-flop is a clocked *RS* type, with the master flip-flop supplying the inputs and the clock input being inverted by gate 9. When the clock is 0, the output of gate 9 is 1, so that output *Q* is equal to *Y*. and *Q′* is equal to *Y′*. When the positive edge of a clock pulse occurs, the master flip-flop is affected and may switch states. The slave flip-flop is isolated as long as the clock is at the 1 level, because the output of gate 9 provides a 1 to both inputs of the NAND basic flip-flop of gates 7 and 8. When the clock input returns to 0, the master flip-flop is isolated from the *J* and *K* inputs and the slave flip-flop goes to the same state as the master flip-flop.

Now consider a digital system containing many master-slave flip-flops, with the outputs of some flip-flops going to the inputs of other flip-flops. Assume that clock pulse inputs to all flip-flops are synchronized (occur at the same time). At the beginning of each clock pulse, some of the master elements change state, but all flip-flop outputs remain at their previous values. After the clock pulse returns to 0. some of the outputs change state, but none of these new states have an effect on any of the master elements until the next clock pulse. Thus the states of flip-flops in the system can be changed simultaneously during the same clock pulse, even though outputs of flip-flops are connected to inputs of flip-flops. This is possible because the new state appears at the output terminals only after the clock pulse has returned to 0. Therefore, the binary content of one flip-flop can be transferred to a second flip-flop and the content of the second transferred to the first, and both transfers can occur during the same clock pulse.

6.3.2 Edge-Triggered Flip-Flop

Another type of flip-flop that synchronizes the state changes during a clock pulse transition is the *edge-triggered* flip-flop. In this type of flip-flop, output transitions occur at a specific level of the clock pulse. When the pulse input level exceeds this threshold level, the inputs are locked out and the flip-flop is therefore unresponsive to further changes in inputs until the clock pulse returns to 0 and another pulse occurs. Some edge-triggered flip-flops cause a transition on the positive edge of the pulse, and others cause a transition on the negative edge of the pulse.

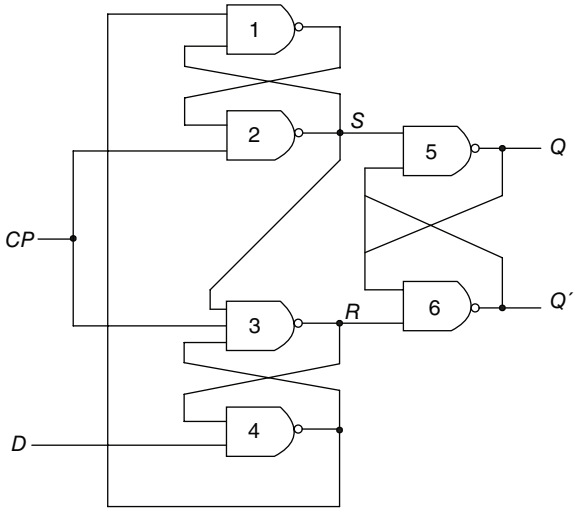
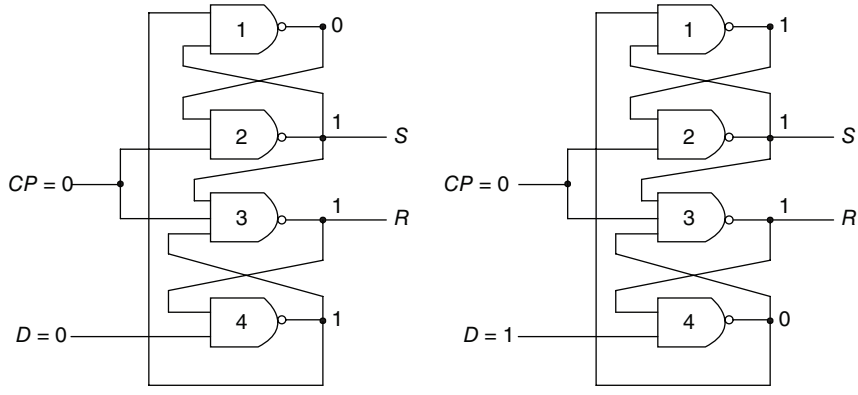


Figure 6.12 *D*-type positive-edge-triggered flip-flop

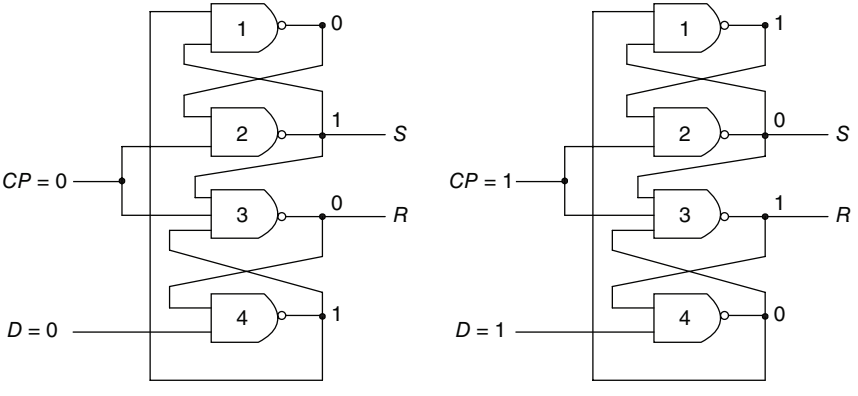
The logic diagram of a *D*-type positive-edge-triggered flip-flop is shown in Fig. 6-12. It consists of three basic flip-flops of the type shown in Fig, 6-3. NAND gates 1 and 2 make up one basic flip-flop and gates 3 and 4 another. The third basic flip-flop comprising gates 5 and 6 provides the outputs to the circuit. Inputs 5 and *R* of the third basic flip-flop must be maintained at logic-1 for the outputs to remain in their steady-state values, When $S = 0$ and $R = 1$, the output goes to the set state with $Q = 1$. When $S = 1$ and $R = 0$, the output goes to the clear state with $Q = 0$. Inputs *S* and *R* are determined from the states of the other two basic flip-flops. These two basic flip-flops respond to the external inputs *D* (data) and *CP* (clock pulse).

The operation of the circuit is explained in Fig. 6-13, where gates 1-4 are redrawn to show all possible transitions. Outputs *S* and *R* from gates 2 and 3 go to gates 5 and 6, as shown in Fig. 6-12, to provide the actual outputs of the flip-flop. Figure 6-13(a) shows the binary values at the outputs of the four gates when $CP = 0$. Input *D* may be equal to 0 or 1. In either case, a *CP* of 0 causes the outputs of gates 2 and 3 to go to I, thus making $S = R = 1$, which is the condition for a steady-state output. When $D = 0$, gate 4 has a 1 output, which causes the output of gate 1 to go to 0. When $D = 1$, gate 4 goes to 0, which causes the output of gate 1 to go to 1. These are the two possible conditions when the *CP* terminal, being 0, disables any changes at the outputs of the flip-flop, no matter what the value of *D* happens to be.

There is a definite time, called the *setup* time, in which the *D* input must be maintained at a constant value prior to the application of the pulse. The setup time is equal to the propagation delay through gates 4 and 1 since a change in *D* causes a change in the outputs of these two gates. Assume now that *D* does not change during the setup time and that input *CP* becomes 1. This situation is depicted in Fig. 6-13(b). If $D = 0$ when *CP* becomes I. then *S* remains 1 but *R* changes to 0. This causes the output of the flip-flop *Q* to go to 0 (in Fig, 6-12). If now, while $CP = 1$, there is a change in the *D* input, the output of gate 4 will remain at 1 (even if *D* goes to 1), since one of the gate inputs comes from *R* which is maintained at 0. Only when *CP* returns to 0 can the output of gate 4 change; but then both *R* and *S* become 1, disabling any changes in the output



(a) With $CP = 0$



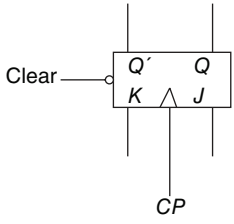
(b) With $CP = 1$

Figure 6.13 Operation of the *D*-type edge-triggered flip-flop

of the flip-flop. However, there is a definite time, called the *hold time,* that the D input must not change after the application of the positive-going transition of the pulse. The hold time is equal to the propagation delay of gate 3, since it must be ensured that R becomes 0 in order to maintain the output of gate 4 at 1, regardless of the value of D.

If $D = 1$ when $CP = 1$, then S changes to 0 but R remains at 1, which causes the output of the flip-flop Q to go to 1. A change in D while $CP = 1$ does not alter S and R because gate t is maintained at 1 by the 0 signal from S. When CP goes to zero, both S and R go to 1 to prevent the output from undergoing any changes.

In summary, when the input clock pulse makes a positive-going transition, the value of D is transferred to Q. Changes in D when CP is maintained at a steady 1 value do not affect Q. Moreover, a negative pulse transition does not affect the output, nor does it when $CP = 0$. Hence, the edge-triggered flip-flop eliminates any feedback problems in sequential circuits just as a master-slave flip-flop does. The setup time and hold time must be taken into consideration when using this type of flip-flop.



Function table

Inputs				Outputs	
Clear	Clock	*J*	*K*	*Q*	*Q*´
0	*X*	*X*	*X*	0	1
1	↓	0	0	No change	
1	↓	0	1	0	1
1	↓	1	0	1	0
1	↓	1	1	Toggle	

Figure 6.14 *JK* flip-flop with direct clear

When using different types of flip-flops in the same sequential circuit, one must ensure that all flip-flop outputs make their transitions at the same time, i.e., during either the negative edge or the positive edge of the pulse. Those flip-flops that behave opposite from the adopted polarity transition can be changed easily by the addition of inverters in their clock inputs. An alternate procedure is to provide both positive and negative pulses (by means of an inverter), and then apply the positive pulses to flip-flops that trigger during the negative edge and negative pulses to flip-flops that trigger during the positive edge, or vice versa.

6.3.3 Direct Inputs

Flip-flops available in IC packages sometimes provide special inputs for setting or clearing the flip-flop asynchronously. These inputs are usually called *direct preset* and *direct clear.* They affect the flip-flop on a positive (or negative) value of the input signal without the need for a clock pulse. These inputs are useful for bringing all flip-flops to an initial state prior to their clocked operation. For example, after power is turned on in a digital system, the states or its flip-flops are indeterminate. A *clear* switch clears all the flip-flops to an initial cleared state and a *start* switch begins the system's clocked operation. The clear switch must clear all flip-flops asynchronously without the need for a pulse.

The graphic symbol of a master-slave flip-flop with direct clear is shown in Fig. 6-14. The clock or *CP* input has a circle under the small triangle to indicate that the outputs change during the negative transition of the pulse. (The absence of the small circle would indicate a positive-edge-triggered flip-flop.) The direct clear input also has a small circle to indicate that, normally, this input must be maintained at 1. If the clear input is maintained at 0, the flip-flop remains cleared, regardless of the other inputs or the clock pulse. The function table specifies the circuit operation. The *X*'s are don't-care conditions which indicate that a 0 in the direct clear input disables all other inputs. Only when the clear input is 1 would a negative transition of the clock have an effect on the outputs. The outputs do not change if $J = K = 0$. The flip-flop toggles or complements when $J = K = 1$. Some flip-flops may also have a direct preset input which sets the output Q to 1 (and Q' to 0) asynchronously.

When direct asynchronous inputs are available in a master-slave flip-flop, they must connect to both the master and the stave in order to override the other inputs and the clock. A direct clear in the *JK* master-slave flip-flop of Fig. 6-10 is connected to the inputs of gales 1, 4, and 8. A direct clear in the *D* edge-triggered flip-flop of Fig. 6-12 is connected to the inputs of gates 2 and 6.

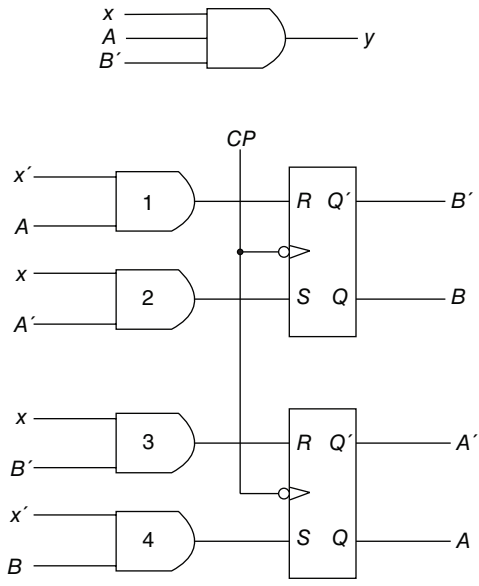


Figure 6.15 Example of a clocked sequential circuit

6.4 Analysis of Clocked Sequential Circuits

The behavior of a sequential circuit is determined from the inputs, the outputs, and the states of its flip-flops. Both the outputs and the next state are a function of the inputs and the present state. The analysis of sequential circuits consists of obtaining a table or a diagram for the time sequence of inputs, outputs, and internal states. It is also possible to write Boolean expressions that describe the behavior of sequential circuits. However, these expressions must include the necessary time sequence either directly or indirectly.

A logic diagram is recognized as the circuit of a sequential circuit if it includes flip-flops. The flip-flops may be of any type and the logic diagram may or may not include combinational gates. In this section, we first introduce a specific example of a clocked sequential circuit and then present various methods for describing the behavior of sequential circuits. The specific example will be used throughout the discussion to illustrate the various methods.

6.4.1 An Example of a Sequential Circuit

An example of a clocked sequential circuit is shown in Fig. 6-15. It has one input variable *x,* one output variable *y,* and two clocked *RS* flip-flops labeled *A* and *B.* The cross-connections from outputs of flip-flops to inputs of gates are not shown by line drawings so as to facilitate the tracing of the circuit. Instead, the connections are recognized from the letter symbol marked in each input. For example, the input marked x' in gate 1 designates an input from the complement of *x.* The second input marked *A* designates a connection to the normal output of flip-flop *A.*

We shall assume negative edge triggering in both flip-flops and in the source that produces the external input *x*. Therefore, the signals for a given present, state are available during the time from the termination of a clock pulse to the termination of the next clock pulse, at which time the circuit goes to the next state.

Table 6-1 State table for circuit of Fig. 6-15

	Next state		Output	
Present State	$x = 0$	$x = 1$	$x = 0$	$x = 1$
AB	AB	AB	y	y
00	00	01	0	0
01	11	01	0	0
10	10	00	0	1
11	10	11	0	0

6.4.2 State Table

The time sequence of inputs, outputs, and flip-flop states may be enumerated in a *state table.*[*] The state table for the circuit of Fig. 6-15 is shown in Table 6-1. It consists of three sections labeled *present state, next state,* and *output.* The *present state* designates the states of flip-flops before the occurrence of a clock pulse. The *next state* shows the states of flip-flops after the application of a clock pulse, and the *output* section lists the values of the output variables during the present state. Both the next state and output sections have two columns, one for $x = 0$ and the other for $x = 1$.

The derivation of the state table starts from an assumed initial state. The initial state of most practical sequential circuits is defined to be the state with 0's in all flip-flops. Some sequential circuits have a different initial state and some have none at all. In either case, the analysis can always start from any arbitrary state. In this example, we start deriving the state table from the initial state 00.

When the present state is 00, $A = 0$ and $B = 0$. From the logic diagram, we see that with both flip-flops cleared and $x = 0$, none of the AND gates produce a logic-1 signal. Therefore, the next state remains unchanged. With $AB = 00$ and $x = 1$, gate 2 produces a logic-1 signal at the S input of flip-flop B and gate 3 produces a logic-1 signal at the R input of flip-flop A. When a clock pulse triggers the flip-flops, A is cleared and B is set, making the next state 01. This information is listed in the first row of the state table.

In a similar manner, we can derive the next state starting from the other three possible present states. In general, the next state is a function of the inputs, the present state, and the type of flip-flop used. With RS flip-flops, for example, we must remember that a 1 in input S sets the flip-flop and a 1 in input R clears the flip-flop, regardless of its previous state. A 0 in both the S' and R inputs leaves the flip-flop unchanged, whereas a 1 in both the S and R inputs shows a bad design and an indeterminate state table.

The entries for the output section are easier to derive. In this example, output y is equal to 1 only when $x = 1$, $A = 1$, and $B = 0$. Therefore, the output columns are marked with 0's, except when the present state is 10 and input $x = 1$, for which y is marked with a 1.

The state table of any sequential circuit is obtained by the same procedure used in the example. In general, a sequential circuit with m flip-flops and n input variables will have 2^m rows, one for each state. The next state and output sections each will have 2^n columns, one for each input combination.

[*]Switching circuit theory books call this table a *transition table.* They reserve the name state table for a table with internal states represented by arbitrary symbols.

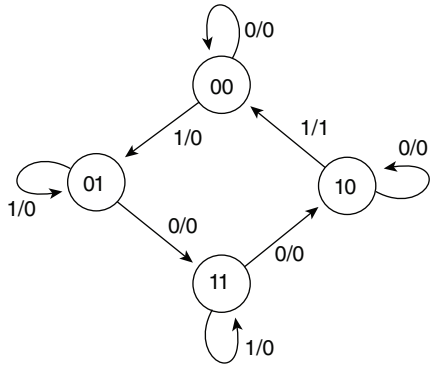


Figure 6.16 State diagram for the circuit of Fig. 6.16

The external outputs of a sequential circuit may come from logic gates or from memory elements. The output section in the state table is necessary only if there are outputs from logic gates. Any external output taken directly from a flip-flop is already listed in the present state column of the state table. Therefore, the output section of the state table can be excluded if there are no external outputs from logic gates.

6.4.3 State Diagram

The information available in a state table may be represented graphically in a *state diagram.* In this diagram, a state is represented by a circle, and the transition between states is indicated by directed lines connecting the circles. The state diagram of the sequential circuit of Fig. 6-15 is shown in Fig. 6-16. The binary number inside each circle identifies the state the circle represents. The directed lines are labeled with two binary numbers separated by a /. The input value that causes the state transition is labeled first; the number after the symbol / gives the value of the output during the present state. For example, the directed line from state 00 to 01 is labeled 1/0, meaning that the sequential circuit is in a present state 00 while $x = 1$ and $x = 0$, and that on the termination of the next clock pulse, the circuit goes to next state 01. A directed line connecting a circle with itself indicates that no change of state occurs. The state diagram provides the same information as the state table and is obtained directly from Table 6-1.

There is no difference between a state table and a state diagram except in the manner of representation. The state table is easier to derive from a given logic diagram and the state diagram follows directly from a state table. The state diagram gives a pictorial view of state transitions and is in a form suitable for human interpretation of the circuit operation. The state diagram is often used as the initial design specification of a sequential circuit.

6.4.4 State Equations

A *state equation* (also known as an *application equation*) is an algebraic expression that specifies the conditions for a flip-flop state transition. The left side of the equation denotes the next state of a flip-flop and the right side, a Boolean function that specifies the present state conditions

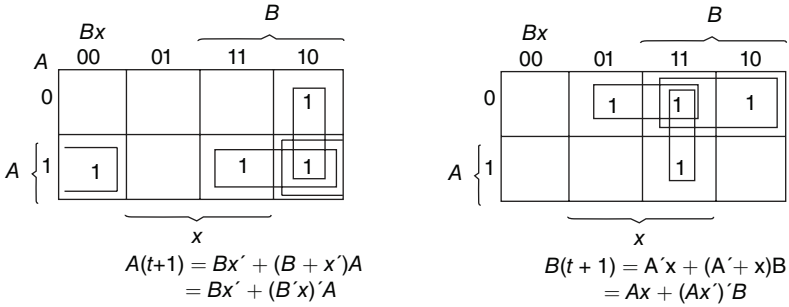


Figure 6.17 State equation for flip-flops *A* and *B*

that make the next state equal to 1. A state equation is similar in form to a flip-flop characteristic equation, except that it specifies the next state conditions in terms of external input variables and other flip-flop values. The state equation is derived directly from a state table. For example, the state equation for flip-flop *A* is derived from inspection of Table 6-1. From the next state columns, we note that flip-flop *A* goes to the 1 state four times: when $x = 0$ and $AB = 01$ or 10 or 11, or when $x = 1$ and $AB = 11$. This can be expressed algebraically in a state equation as follows:

$$A(t + 1) = (A'B + AB' + AB)\,x' + ABx$$

The right-hand side of the state equation is a Boolean function for a *present state.* When this function is equal to 1, the occurrence of a clock pulse causes flip-flop *A* to have a next state of 1. When the function is equal to 0, the clock pulse causes *A* to have a next state of 0. The left side of the equation identifies the flip-flop by its letter symbol, followed by the time function designation $(t + 1)$, to emphasize that this value is to be reached by the flip-flop *one* pulse sequence later.

The state equation is a Boolean function with time included. It is applicable only in clock sequential circuits, since $A\,(t + 1)$ is defined to change value with the occurrence of a clock pulse at discrete instants of time.

The state equation for flip-flop *A* is simplified by means of a map as shown in Fig. 6-17(a). With some algebraic manipulation, the function can be expressed in the following form:

$$A\,(t + 1) = B'x + (B'x)'A$$

If we let $Bx' = S$ and $B'\,x = R$. we obtain the relationship:

$$A\,(t + 1) = S + R'A$$

which is the characteristic equation of an *RS* flip-flop [Fig. 6-4(d)]. This relationship between the state equation and the flip-flop characteristic equation can be justified from inspection of the logic diagram of Fig. 6-15. In it we see that the *S* input of flip-flop *A* is equal to the Boolean function Bx' and the *R* input is equal to $B'x$. Substituting these functions into the flip-flop characteristic equation results in its state equation for this sequential circuit.

The state equation for a flip-flop in a sequential circuit may be derived from a state table or from a logic diagram. The derivation from the state table consists of obtaining the Boolean function specifying the conditions that make the next state of the flip-flop a 1. The derivation from a

logic diagram consists of obtaining the functions of the flip-flop inputs and substituting them into the flip-flop characteristic equation.

The derivation of the state equation for flip-flop *B* from the state table is shown in the map of Fig. 6-17(b). The 1's marked in the map are the present state and input combinations that cause the flip-flop to go to a next state of 1. These conditions are obtained directly from Table 6-1. The simplified form obtained in the map is manipulated algebraically, and the state equation obtained is:

$$B(t + 1) = A'x + (Ax')' B$$

The state equation can be derived directly from the logic diagram. From Fig. 6-15, we see that the signal for input *S* of flip-flop *B* is generated by the function $A' x$ and the signal for input *R* by the function Ax'. Substituting $S = A' x$ and $R = Ax'$ into an *RS* flip-flop characteristic equation given by:

$$B (t + 1) = S + R'B$$

we obtain the state equation derived above.

The state equations of all flip-flops, together with the output functions, fully specify a sequential circuit. They represent, algebraically, the same information a state table represents in tabular form and a state diagram represents in graphical form.

6.4.5 Flip-flop Input Functions

The logic diagram of a sequential circuit consists of memory elements and gates. The type of flip-flops and their characteristic table specify the logical properties of the memory elements. The interconnections among the gates form a combinational circuit and may be specified algebraically with Boolean functions. Thus, knowledge of the type of flip-flops and a list of the Boolean functions of the combinational circuit provide all the information needed to draw the logic diagram of a sequential circuit. The part of the combinational circuit that generates external outputs is described algebraically by the *circuit output functions.* The part of the circuit that generates the inputs to flip-flops are described algebraically by a set of Boolean functions called *flip-flop input functions* or sometimes *input equations.*

We shall adopt the convention of using two letters to designate a flip-flop input variable: the first to designate the name of the input and the second the name of the flip-flop. As an example, consider the following flip-flop input functions:

$$JA = BC' x + B'Cx'$$
$$KA = B + y$$

JA and *KA* designate two Boolean variables. The first letter in each denotes the *J* and *K* input, respectively, of a *JK* flip-flop. The second letter *A* is the symbol name of the flip-flop. The right side of each equation is a Boolean function for the corresponding flip-flop input variable. The implementation of the two input functions is shown in the logic diagram of Fig. 6-18. The *JK* flip-flop has an output symbol *A* and two inputs labeled *J* and *K.* The combinational circuit drawn in the diagram is the implementation of the algebraic expression given by the input functions. The outputs of the combinational circuit are denoted by *JA* and *KA* in the input functions and go to the *J* and *K* inputs, respectively, of flip-flop *A*.

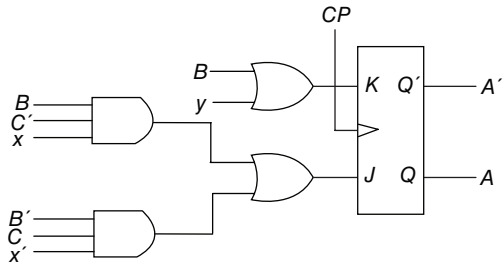


Figure 6.18 Implementation of the flip-flop input functions $JA = BC'x + B'Cx'$ and $KA = B + y$

From this example, we see that a flip-flop input function is an algebraic expression for a combinational circuit. The two-letter designation is a variable name for an *output* of the combinational circuit. This output is always connected to the *input* (designated by the first letter) of a flip-flop (designated by the second letter).

The sequential circuit of Fig. 6-15 has one input *x,* one output *y* and two *RS* flip-flops denoted by *A* and *B.* The logic diagram can be expressed algebraically with four flip-flop input functions and one circuit output function as follows:

$$SA = Bx' \qquad RA = B'x$$
$$SB = A'x \qquad RB = Ax'$$
$$y = AB'x$$

This set of Boolean functions fully specifies the logic diagram, Variables *SA* and *RA* specify an *RS* flip-flop labeled *A;* variables *SB* and *RB* specify a second *RS* flip-flop labeled *B.* Variable *y* denotes the output. The Boolean expressions for the variables specify the combinational circuit part of the sequential circuit.

The flip-flop input functions constitute a convenient algebraic form for specifying a logic diagram of a sequential circuit. They imply the type of flip-flop from the first letter of the input variable and they fully specify the combinational circuit that drives the flip-flop. Time is not included explicitly in these equations but is implied from the clock pulse operation. It is sometimes convenient to specify a sequential circuit algebraically with circuit output functions and flip-flop input functions instead of drawing the logic diagram.

6.5 State Reduction and Assignment†

The analysis of sequential circuits starts from a circuit diagram and culminates in a state table or diagram. The design of a sequential circuit starts from a set of specifications and culminates in a logic diagram. Design procedures are presented starting from Section 6-7. This section discusses certain properties of sequential circuits that may be used to reduce the number of gates and flip-flops during the design.

†This section may be omitted without loss of continuity.

6.5.1 State Reduction

Any design process must consider the problem of minimizing the cost of the final circuit. The two most obvious cost reductions are reductions in the number of flip-flops and the number of gates. Because these two items seem the most obvious, they have been extensively studied and investigated. In fact, a large portion of the subject of switching theory is concerned with finding algorithms for minimizing the number of flip-flops and gates in sequential circuits.

The reduction of the number of flip-flops in a sequential circuit is referred to as the *state reduction* problem. State reduction algorithms are concerned with procedures for reducing the number of states in a state table while keeping the external input-output requirements unchanged. Since m flip-flops produce 2^m states, a reduction in the number of states may (or may not) result in a reduction in the number of flip-flops. An unpredictable effect in reducing the number of flip-flops is that sometimes the equivalent circuit (with less flip-flops) may require more combinational gates.

We shall illustrate the need for state reduction with an example. We start with a sequential circuit whose specification is given in the state diagram of Fig. 6-19. In this example, only the input-output sequences are important; the internal states are used merely to provide the required sequences. For this reason, the states marked inside the circles are denoted by letter symbols instead of by their binary values. This is in contrast to a binary counter, where the binary value sequence of the states themselves are taken as the outputs.

There are an infinite number of input sequences that may be applied to the circuit; each results in a unique output sequence. As an example, consider the input sequence 01010110100 starting from the initial state *a.* Each input of 0 or 1 produces an output of 0 or 1 and causes the circuit to go to the next state. From the state diagram, we obtain the output and state sequence for the given input sequence as follows: With the circuit in initial state *a*, an input of 0 produces an output of 0 and the circuit remains in state *a.* With present state *a,* and input of 1, the output

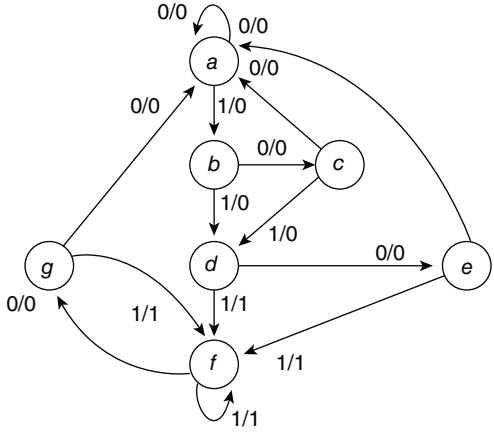


Figure 6-19 State diagram

Table 6-2 State table

	Next state		Output	
Present state	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

is 0 and the next state is *b.* With present state *b* and input of 0, the output is 0 and next state is *c.* Continuing this process, we find the complete sequence to be as follows:

state	*a*	*a*	*b*	*c*	*d*	*e*	*f*	*f*	*g*	*f*	*g*	*a*
input	0	1	0	1	0	1	1	0	1	0	0	
output	0	0	0	0	0	1	1	0	1	0	0	

In each column, we have the present state, input value, and output value. The next state is written on top of the next column. It is important to realize that in this circuit the states themselves are of secondary importance because we are interested only in output sequences caused by input sequences.

Now let us assume that we have found a sequential circuit whose state diagram has less than seven states and we wish to compare it with the circuit whose state diagram is given by Fig. 6-19. If identical input sequences are applied to the two circuits and identical outputs occur for all input sequences, then the two circuits are said to be equivalent (as far as the input-output is concerned) and one may be replaced by the other. The problem of state reduction is to find ways of reducing the number of states in a sequential circuit without altering the input-output relationships.

We shall now proceed to reduce the number of states for this example. First, we need the state table; it is more convenient to apply procedures for state reduction here than in state diagrams. The state table of the circuit is listed in Table 6-2 and is obtained directly from the state diagram of Fig. 6-19.

An algorithm for the state reduction of a completely specified state table is given here without proof: "Two states are said to be equivalent if, for each member of the set of inputs, they give exactly the same output and send the circuit either to the same state or to an equivalent state. When two states are equivalent, one of them can be removed without altering the input-output relationships."

We shall apply this algorithm to Table 6-2. Going through the state table, we look for two present states that go to the same next state and have the same output for both input combinations. States *g* and *e* are two such states; they both go to states *a* and *f* and have outputs of 0 and 1 for $x = 0$ and $x = 1$, respectively. Therefore, states *g* and *e* are equivalent; one can be removed.

Table 6-3 Reducing the state table

	Next state		Output	
Present state	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	*a*	*b*	0	0
b	*c*	*d*	0	0
c	*a*	*d*	0	0
d	*e*	~~*f*~~ *d*	0	1
e	*a*	~~*f*~~ *d*	0	1
~~*f*~~	~~*g*~~ *e*	*f*	0	1
~~*g*~~	*a*	*f*	0	1

The procedure of removing a state and replacing it by its equivalent is demonstrated in Table 6-3. The row with present state *g* is crossed out and state *g* is replaced by state *e* each time it occurs in the next state columns.

Present state *f* now has next states *e* and *f* and outputs 0 and 1 for $x = 0$ and $x = 1$, respectively. The same next states and outputs appear in the row with present state *d.* Therefore, states *f* and *d* are equivalent; state *f* can be removed and replaced by *d.* The final reduced table is shown in Table 6-4. The state diagram for the reduced table consists of only five states and is shown in Fig. 6-20. This state diagram satisfies the original input-output specifications and will produce the required output sequence for any given input sequence. The following list derived from the state diagram of Fig. 6-20 is for the input sequence used previously. We note that the same output sequence results although the state sequence is different:

state	a	a	b	c	d	e	d	d	e	d	e	a
input	0	1	0	1	0	1	1	0	1	0	0	
output	0	0	0	0	0	1	1	0	1	0	0	

In fact, this sequence is exactly the same as that obtained for Fig. 6-19, if we replace *e* by *g* and *d* by *f*.

It is worth noting that the reduction in the number of states of a sequential circuit is possible if one is interested only in external input-output relationships. When external outputs are taken

Table 6-4 Reduced state table

	Next state		Output	
Present state	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	*a*	*b*	0	0
b	*c*	*d*	0	0
c	*a*	*d*	0	0
d	*e*	*d*	0	1
e	*a*	*d*	0	1

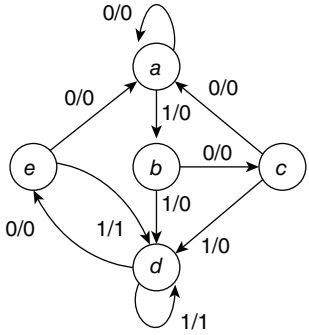


Figure 6-20 Reduced state diagram

directly from flip-flops, the outputs must be independent of the number of states before state reduction algorithms are applied.

The sequential circuit of this example was reduced from seven to five states. In either case, the representation of the states with physical components requires that we use three flip-flops, because m flip-flops can represent up to 2^m distinct states. With three flip-flops, we can formulate up to eight binary states denoted by binary numbers 000 through 111, with each bit designating the state of one flip-flop. If the state table of Table 6-2 is used, we must assign binary values to seven states; the remaining state is unused. If the state table of Table 6-4 is used, only five states need binary assignment, and we are left with three unused states, Unused states are treated as don't-care conditions during the design of the circuit. Since don't-care conditions usually help in obtaining a simpler Boolean function, it is more likely that the circuit with five states will require fewer combinational gates than the one with seven states. In any case, the reduction from seven to five states does not reduce the number of flip-flops. In general, reducing the number of states in a state table is likely to result in a circuit with less equipment. However, the fact that a state table has been reduced to fewer states does not guarantee a saving in the number of flip-flops or the number of gates.

6.5.2 State Assignment

The cost of the combinational circuit part of a sequential circuit can be reduced by using the known simplification methods for combinational circuits. However, there is another factor, known as the *state assignment* problem, that comes into play in minimizing the combinational gates. State assignment procedures are concerned with methods for assigning binary values to states in such a way as to reduce the cost of the combinational circuit that drives the flip-flops. This is particularly helpful when a sequential circuit is viewed from its external input-output terminals. Such a circuit may follow a sequence of internal states, but the binary values of the individual states may be of no consequence as long as the circuit produces the required sequence of outputs for any given sequence of inputs. This does not apply to circuits whose external outputs are taken directly from flip-flops with binary sequences fully specified.

The binary state assignment alternatives available can be demonstrated in conjunction with the sequential circuit specified in Table 6-4. Remember that, in this example, the binary values

Table 6-5 Three possible binary state assignments

State	Assignment I	Assignment 2	Assignment 3
a	001	000	000
b	010	010	100
c	011	011	010
d	100	101	101
e	101	111	011

of the states are immaterial as long as their sequence maintains the proper input-output relationships. For this reason, any binary number assignment is satisfactory as long as each state is assigned a unique number. Three examples of possible binary assignments are shown in Table 6-5 for the five states of the reduced table. Assignment 1 is a straight binary assignment for the sequence of states from *a* through *e.* The other two assignments are chosen arbitrarily. In fact, there are 140 different distinct assignments for this circuit (11).

Table 6-6 is the reduced state table with binary assignment 1 substituted for the letter symbols of the five states.‡ It is obvious that a different binary assignment will result in a state table with different binary values for the states, while the input-output relationships remain the same. The binary form of the state table is used to derive the combinational circuit part of the sequential circuit. The complexity of the combinational circuit obtained depends on the binary state assignment chosen. The design of the sequential circuit presented in this section is completed in Example 6-1 of Section 6-7.

Various procedures have been suggested that lead to a particular binary assignment from the many available. The most common criterion is that the chosen assignment should result in a simple combinational circuit for the flip-flop inputs. However, to date, there are no state assignment procedures that guarantee a minimal-cost combinational circuit. State assignment is one of the challenging problems of switching theory. The interested reader will find a rich and growing literature on this topic. Techniques for dealing with the state assignment problem are beyond the scope of this book.

‡A state table with binary assignment is sometimes called a *transition table.*

Table 6-6 Reduced state table with binary assignment 1

	Next state		Output	
Present state	$x = 0$	$x = 1$	$x = 0$	$x = 1$
001	001	010	0	0
010	011	100	0	0
011	001	100	0	0
100	101	100	0	1
101	001	100	0	1

6.6 Flip-flop Excitation Tables

The characteristic tables for the various flip-flops were presented in Section 6-2. A characteristic table defines the logical property of the flip-flop and completely characterizes its operation. Integrated-circuit flip-flops are sometimes defined by a characteristic table tabulated somewhat differently. This second form of the characteristic tables for *RS, JK, D,* and *T* flip-flops is shown in Table 6-7. They represent the same information as the characteristic tables of Figs. 6-4(c) through 6-7(c).

Table 6-7 defines the state of each flip-flop as a function of its inputs and previous state. $Q(t)$ refers to the present state and $Q(t + 1)$ to the next state after the occurrence of a clock pulse. The characteristic table for the *RS* flip-flop shows that the next state is equal to the present state when inputs *S* and *R* are both 0. When the *R* input is equal to 1, the next clock pulse clears the flip-flop. When the *S* input is equal to 1, the next clock pulse sets the flip-flop. The question mark for the next state when *S* and *R* are both equal to 1 simultaneously designates an indeterminate next state.

The table for the *JK* flip-flop is the same as that for the *RS* when *J* and *K* are replaced by *S* and *R,* respectively, except for the indeterminate case. When both *A* and *K* are equal to 1, the next state is equal to the complement of the present state, i.e., $Q(t + 1) = Q'(t)$. The next state of the *D* flip-flop is completely dependent on the input *D* and independent of the present state. The next state of the *T* flip-flop is the same as the present state if $T = 0$ and complemented if $T = 1$.

The characteristic table is useful for analysis and for defining the operation of the flip-flop. It specifies the next state when the inputs and present state are known. During the design process we usually know the transition from present state to next state and wish to find the flip-flop input conditions that will cause the required transition. For this reason, we need a table that lists the required inputs for a given change of state. Such a list is called an *excitation table.*

Table 6-8 presents the excitation tables for the four flip-flops. Each table consists of two columns, $Q(t)$ and $Q(t + 1)$, and a column for each input to show how the required transition is achieved. There are four possible transitions from present state to next state. The required input conditions for each of the four transitions are derived from the information available in the

Table 6-7 Flip-flop characteristic tables

S	*R*	$Q(t+1)$
0	0	$Q(t)$
0	1	0
1	0	1
1	1	?

(a) *RS*

J	*K*	$Q(t+1)$
0	0	$Q(t)$
0	1	0
1	0	1
1	1	$Q'(t)$

(b) *JK*

D	$Q(t+1)$
0	0
1	1

(c) *D*

T	$Q(t+1)$
0	$Q(t)$
1	$Q'(t)$

(*d*) *T*

Table 6-8 Flip-flop excitation tables

$Q(t)$	$Q(t+1)$	S	R
0	0	0	*X*
0	1	1	0
1	0	0	1
1	1	*X*	0

(a) *RS*

$Q(t)$	$Q(t+1)$	*J*	*K*
0	0	0	*X*
0	1	1	*X*
1	0	*X*	1
1	1	*X*	0

(b) *JK*

$Q(t)$	$Q(t+1)$	*D*
0	0	0
0	1	1
1	0	0
1	1	1

(c) *D*

$Q(t)$	$Q(t+1)$	T
0	0	0
0	1	1
1	0	1
1	1	0

(d) *T*

characteristic table. The symbol *X* in the tables represents a don't-care condition, i.e., it does not matter whether the input is 1 or 0.

6.6.1 *RS* Flip-flop

The excitation table for the *RS* flip-flop is shown in Table 6-8(a). The first row shows the flip-flop in the 0-state at time *t*. It is desired to leave it in the 0-state after the occurrence of the pulse. From the characteristic table, we find that if *S* and *R* are both 0, the flip-flop will not change state. Therefore, both *S* and *R* inputs should be 0. However, it really doesn't matter if *R* is made a 1 when the pulse occurs, since it results in leaving the flip-flop in the 0-state. Thus, *R* can be 1 or 0 and the flip-flop will remain in the 0-state at $t + 1$, Therefore, the entry under *R* is marked by the don't-care condition *X*.

If the flip-flop is in the 0-state and it is desired to have it go to the l-state, then from the characteristic table, we find that the only way to make $Q(t + 1)$ equal to 1 is to make $S = 1$ and $R = 0$. If the flip-flop is to have a transition from the 1-state to the 0-state, we must have $S = 0$ and $R = 1$.

The last condition that may occur is for the flip-flop to be in the 1-state and remain in the 1-state. Certainly *R* must be 0; we do not want to clear the flip-flop. However, *S* may be either a 0 or a 1. If it is 0, the flip-flop does not change and remains in the 1-state; if it is 1, it sets the flip-flop to the 1-state as desired. Therefore, *S* is listed as a don't-care condition.

6.6.2 *JK* Flip-Flop

The excitation table for the *JK* flip-flop is shown in Table 6-8(b). When both present state and next state are 0, the *J* input must remain at 0 and the *K* input can be either 0 or 1. Similarly, when both present state and next state are 1, the *K* input must remain at 0 while the *J* input can be 0 or 1. If the flip-flop is to have a transition from the 0-state to the 1-state. *J* must be equal to 1 since the *J* input sets the flip-flop. However, input *K* may be either 0 or a 1. If $K = 0$, the $J = 1$ condition

sets the flip-flop as required; if $K = 1$ and $J = 1$, the flip-flop is complemented and goes from the 0-state to the 1-state as required. Therefore the K input is marked with a don't-care condition for the 0-to-1 transition. For a transition from the 1-state to the 0-state, we must have $K = 1$, since the K input clears the flip-flop. However, the J input may be either 0 or 1, since $J = 0$ has no effect, and $J = 1$ together with $K = 1$ complements the flip-flop with a resultant transition from the 1-state to the 0-state.

The excitation table for the *JK* flip-flop illustrates the advantage of using this type when designing sequential circuits. The fact that it has so many don't-care conditions indicates that the combinational circuits for the input functions are likely to be simpler because don't-care terms usually simplify a function.

6.6.3 *D* Flip-Flop

The excitation table for the *D* flip-flop is shown in Table 6-8(c). From the characteristic table, Table 6-7(c), we note that the next state is always equal to the *D* input and independent of the present state. Therefore, *D* must be 0 if $Q(t + 1)$ has to be 0, and 1 if $Q(t + 1)$ has to be 1, regardless of the value of $Q(t)$.

6.6.4 *T* Flip-Flop

The excitation table for the *T* flip-flop is shown in Table 6-8(d). From the characteristic table, Table 6-7(d), we find that when input $T = 1$, the state of the flip-flop is complemented; when $T = 0$, the state of the flip-flop remains unchanged. Therefore, when the state of the flip-flop must remain the same, the requirement is that $T = 0$. When the state of the flip-flop has to be complemented, *T* must equal 1.

6.6.5 Other Flip-Flops

The design procedure to be described in this chapter can be used with any flip-flop. It is necessary that the flip-flop characteristic table, from which it is possible to develop a new excitation table, be known. The excitation table is then used to determine the flip-flop input functions, as explained in the next section.

6.7 Design Procedure

The design of a clocked sequential circuit starts from a set of specifications and culminates in a logic diagram or a list of Boolean functions from which the logic diagram can be obtained. In contrast to a combinational circuit, which is fully specified by a truth table, a sequential circuit requires a state table for its specification. The first step in the design of sequential circuits is to obtain a state table or an equivalent representation, such as a state diagram or state equations.

A synchronous sequential circuit is made up of flip-flops and combinational gates. The design of the circuit consists of choosing the flip-flops and then finding a combinational gate structure which, together with the flip-flops, produces a circuit that fulfills the stated specifications. The number of flip-flops is determined from the number of states needed in the circuit. The combinational circuit is derived from the state table by methods presented in this chapter. In fact, once the type and number of flip-flops are determined, the design process involves a

transformation from the sequential circuit problem into a combinational circuit problem. In this way the techniques of combinational circuit design can be applied.

This section presents a procedure for the design of sequential circuits. Although intended to serve as a guide for the beginner, this procedure can be shortened with experience. The procedure is first summarized by a list of consecutive recommended steps as follows:

1. The word description of the circuit behavior is stated. This may be accompanied by a state diagram, a timing diagram, or other pertinent information.
2. From the given information about the circuit, obtain the state table.
3. The number of states may be reduced by state reduction methods if the sequential circuit can be characterized by input-output relationships independent of the number of states.
4. Assign binary values to each state if the state table obtained in step 2 or 3 contains letter symbols.
5. Determine the number of flip-flops needed and assign a letter symbol to each.
6. Choose the type of flip-flop to be used.
7. From the state table, derive the circuit excitation and output tables.
8. Using the map or any other simplification method, derive the circuit output functions and the flip-flop input functions.
9. Draw the logic diagram.

The word specification of the circuit behavior usually assumes that the reader is familiar with digital logic terminology. It is necessary that the designer use intuition and experience to arrive at the correct interpretation of the circuit specifications, because word descriptions may be incomplete and inexact. However, once such a specification has been set down and the state table obtained, it is possible to make use of the formal procedure to design the circuit.

The reduction of the number of states and the assignment of binary values to the states were discussed in Section 6-5. The examples that follow assume that the number of states and the binary assignment for the states are known. As a consequence, steps 3 and 4 of the design will not be considered in subsequent discussions.

It has already been mentioned that the number of flip-flops is determined from the number of states. A circuit may have unused binary states if the total number of states is less than 2^m. The unused states are taken as don't-care conditions during the design of the combinational circuit part of the circuit.

The type of flip-flop to be used may be included in the design specifications or may depend on what is available to the designer. Many digital systems are constructed entirely with *JK* flip-flops because they are the most versatile available. When many types of flip-flops are available, it is advisable to use the *RS* or *D* flip-flop for applications requiring transfer of data (such as shift registers), the *T* type for applications involving complementation (such as binary counters), and the *JK* type for general applications.

The external output information is specified in the output section of the state table. From it we can derive the circuit output functions. The excitation table for the circuit is similar to that of the individual flip-flops, except that the input conditions are dictated by the information available in the present state and next state columns of the state table. The method of obtaining the excitation table and the simplified flip-flop input functions is best illustrated by an example.

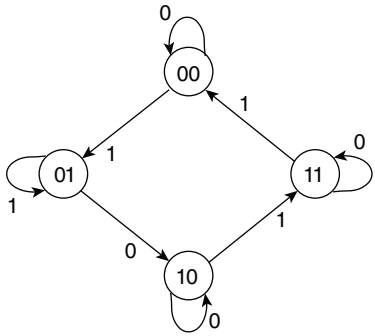


Figure 6-21 State diagram

We wish to design the clocked sequential circuit whose state diagram is given in Fig. 6-21. The type of flip-flop to be used is *JK*.

The state diagram consists of four states with binary values already assigned. Since the directed lines are marked with a single binary digit without a /, we conclude that there is one input variable and no output variables. (The state of the flip-flops may be considered the outputs of the circuit.) The two flip-flops needed to represent the four states are designated *A* and *B*. The input variable is designated *x*.

The state table for this circuit, derived from the state diagram, is shown in Table 6-9. Note that there is no output section for this circuit. We shall now show the procedure for obtaining the excitation table and the combinational gate structure.

The derivation of the excitation table is facilitated if we arrange the state table in a different form. This form is shown in Table 6-10, where the present state and input variables are arranged in the form of a truth table. The next state value for each present state and input conditions is copied from Table 6-9. The excitation table of a circuit is a list of flip-flop input conditions that will cause the required state transitions and is a function of the type of flip-flop used. Since this example specified *JK* flip-flops, we need columns for the *J* and *K* inputs of flip-flops *A* (denoted by *JA* and *KA*) and *B* (denoted by *JB* and *KB*).

The excitation table for the *JK* flip-flop was derived in Table 6-8(b). This table is now used to derive the excitation table of the circuit. For example, in the first row of Table 6-10 we have a transition for flip-flop *A* from 0 in the present state to 0 in the next state. In Table 6-8(b) we find that a transition of states from 0 to 0 requires that input $J = 0$ and input $K = X$. So 0 and *X* are

Table 6-9 State table

Present state		Next state			
		$x = 0$		$x = 1$	
A	*B*	*A*	*B*	*A*	*B*
0	0	0	0	0	1
0	1	1	0	0	1
1	0	1	0	1	1
1	1	1	1	0	0

Table 6-10 Excitation table

Inputs of combinational circuit			Next state		Outputs of combinational circuit			
Present state		Input			Flip-flop inputs			
A	*B*	*X*	*A*	*B*	*JA*	*KA*	*JB*	*KB*
0	0	0	0	0	0	*X*	0	*X*
0	0	1	0	1	0	*X*	1	*X*
0	1	0	1	0	1	*X*	*X*	1
0	1	1	0	1	0	*X*	*X*	0
1	0	0	1	0	*X*	0	0	*X*
1	0	1	1	1	*X*	0	1	*X*
1	1	0	1	1	*X*	0	*X*	0
I	1	1	0	0	*X*	1	*X*	1

copied in the first row under *JA* and *KA,* respectively. Since the first row also shows a transition for flip-flop *B* from 0 in the present state to 0 in the next state, 0 and *X* are copied in the first row under *JB* and *KB.* The second row of Table 6-10 shows a transition for flip-flop *B* from 0 in the present state to 1 in the next state. From Table 6-8(b) we find that a transition from 0 to 1 requires that input $J = 1$ and input $K = X$. So 1 and *X* are copied in the second row under *JB* and *KB,* respectively. This process is continued for each row of the table and for each flip-flop, with the input conditions as specified in Table 6-8(b) being copied into the proper row of the particular flip-flop being considered.

Let us now pause and consider the information available in an excitation table such as Table 6-10. We know chat a sequential circuit consists of a number of flip-flops and a combinational circuit. Figure 6-22 shows the two *JK* flip-flops needed for the circuit and a box to represent the combinational circuit. From the block diagram, it is clear that the outputs of the combinational circuit go to flip-flop inputs and external outputs (if specified). The inputs to the combinational circuit are the external inputs and the present state values of the flip-flops. Moreover, the Boolean functions that specify a combinational circuit are derived from a truth table that shows the input-output relations of the circuit. The truth table that describes the combinational circuit is available in the excitation table. The combinational circuit *inputs* are specified under the present state and input columns, and the combinational circuit *outputs* are specified under the flip-flop input columns. Thus, an excitation table transforms a state diagram to the truth table needed for the design of the combinational circuit part of the sequential circuit.

The simplified Boolean functions for the combinational circuit can now be derived. The inputs are the variables *A*, *B,* and *x*; the outputs are the variables *JA, KA, JB,* and *KB.* The information from the truth table is transferred into the maps of Fig. 6-23, where the four simplified flip-flop input functions are derived:

$$JA = Bx' \qquad KA = Bx$$
$$JB = x \qquad KB = A \odot x$$

The logic diagram is drawn in Fig. 6-24 and consists of two flip-flops, two AND gates, one equivalence gate, and one inverter.

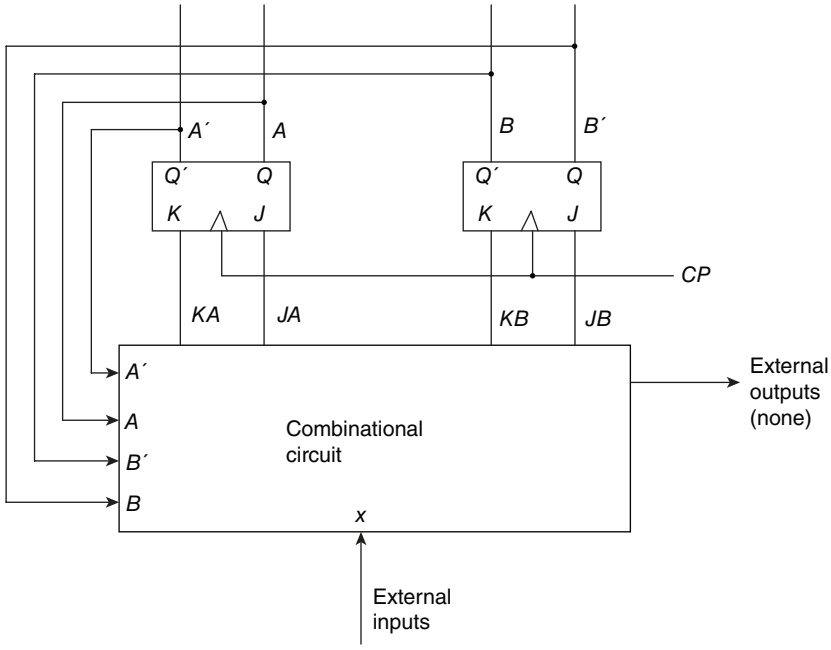


Figure 6-22 Block digram of sequential circuit

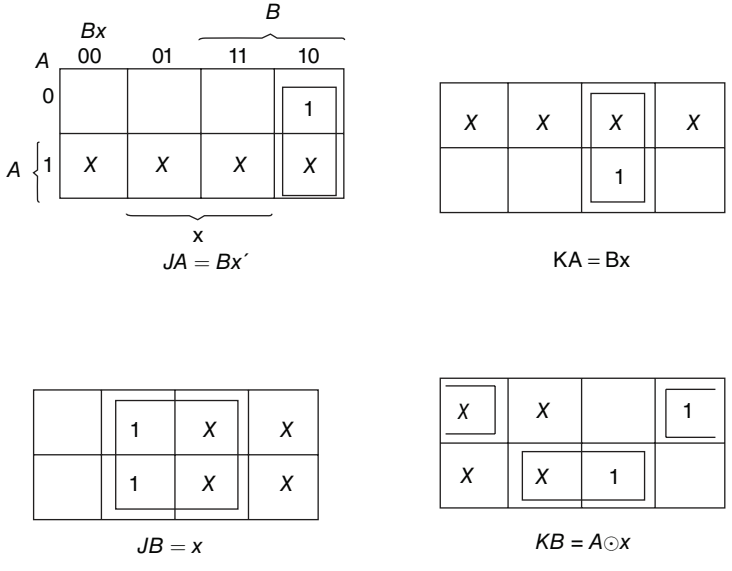


Figure 6-23 Maps for combinational circuit

With some experience, it is possible to reduce the amount of work involved in the design of the combinational circuit. For example, it is possible to obtain the information for the maps of Fig. 6-23 directly from Table 6-9, without having to derive Table 6-10. This is done by systematically going through each present state and input combination in Table 6-9 and comparing it with the binary values of the corresponding next state. The required input conditions as specified by the flip-flop excitation in Table 6-8 is then determined. Instead of inserting the 0, 1, or *X* thus obtained into the excitation table, it can be written down directly into the appropriate square of the appropriate map.

The excitation table of a sequential circuit with *m* flip-flops, *k* inputs per flip-flop, and *n* external inputs consists of $m + n$ columns for the present state and input variables and up to 2^{m+n} rows listed in some convenient binary count. The next state section has *m* columns, one for each flip-flop. The flip-flop input values are listed in *mk* columns, one for each input of each flip-flop. If the circuit contains *j* outputs, the table must include *j* columns. The truth table of the combinational circuit is taken from the excitation table by considering the $m + n$ present state and input columns as *inputs* and the $mk + j$ flip-flop input values and external outputs as *outputs.*

6.7.1 Design with Unused States

A circuit with *m* flip-flops would have 2^m states. There are occasions when a sequential circuit may use less than this maximum number of states. States that are not used in specifying the sequential circuit are not listed in the state table. When simplifying the input functions to flip-flops, the unused states can be treated as don't-care conditions.

EXAMPLE 6-1: Complete the design of the sequential circuit presented in Section 6-5. Use the reduced state table with assignment 1 as given in Table 6-6. The circuit is to employ *RS* flip-flops.

The state table of Table 6-6 is redrawn in Table 6-11 in the form convenient for obtaining the excitation table. The flip-flop input conditions are derived from the present state and next state columns of the state table. Since *RS* flip-flops are used, we need to refer to Table 6-8(a) for the excitation conditions of this type of flip-flop. The three flip-flops are given variable names *A, B,* and C. The input variable is *x* and the output variable is *y.* The excitation table of the circuit provides all the information needed for the design.

There are three unused states in this circuit: binary states 000, 110, and 111. When an input of 0 or 1 is included with these unused states, we obtain six don't-care minterms: 0, 1, 12, 13, 14, and 15. These six binary combinations are not listed in the table under present state and input and are treated as don't-care terms.

The combinational circuit part of the sequential circuit is simplified in the maps of Fig. 6-25. There are seven maps in the diagram. Six maps are for simplifying the input functions for the three *RS* flip-flops. The seventh map is for simplifying the output *y.* Each map has six *X*'s in the squares of the don't-care minterms 0, 1, 2, 13, 14, and 15. The other don't-care terms in the maps come from the *X*'s in the flip-flop input columns of the table. The simplified functions are listed under each map. The logic diagram obtained from these Boolean functions is drawn in Fig. 6-26.

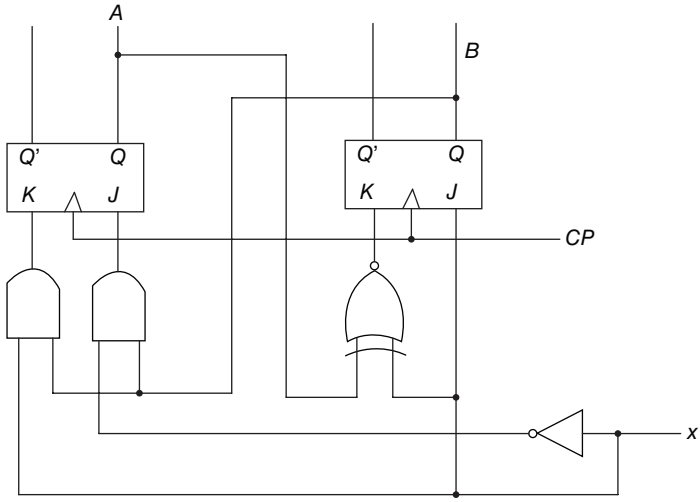


Figure 6-24 Logic diagram of sequential circuit

One factor neglected up to this point in the design is the initial state of a sequential circuit. When power is first turned on in a digital system, one does not know in what state the flip-flops will settle. It is customary to provide a *master-reset* input whose purpose is to initialize the states of all flip-flops in the system. Typically, the master reset is a signal applied to all flip-flops asynchronously before the clocked operations start. In most cases flip-flops are cleared to 0 by the master-reset signal, but some may be set to 1. For example, the circuit of Fig. 6-26 may initially be reset to a state $ABC = 001$, since state 000 is not a valid state for this circuit.

But what if a circuit is not reset to an initial valid state? Or worse, what if, because of a noise signal or any other unforeseen reason, the circuit finds itself in one of its invalid states?

Table 6-11 Excitation table for Example 6-1

Present state			Input	Next state			Flip-flop inputs						Output
A	*B*	*C*	*x*	*A*	*B*	*C*	*SA*	*RA*	*SB*	*RB*	*SC*	*RC*	*y*
0	0	1	0	0	0	1	0	*X*	0	*X*	*X*	0	0
0	0	1	1	0	1	0	0	*X*	1	0	0	1	0
0	1	0	0	0	1	1	0	*X*	*X*	0	1	0	0
0	1	0	1	1	0	0	1	0	0	1	0	*X*	0
0	1	1	0	0	0	1	0	X	0	1	*X*	0	0
0	1	1	1	1	0	0	1	0	0	1	0	1	0
1	0	0	0	1	0	1	*X*	0	0	*X*	1	0	0
1	0	0	1	1	0	0	*X*	0	0	*X*	0	*X*	1
1	0	1	0	0	0	1	0	1	0	*X*	*X*	0	0
1	0	1	1	1	0	0	*X*	0	0	*X*	0	1	1

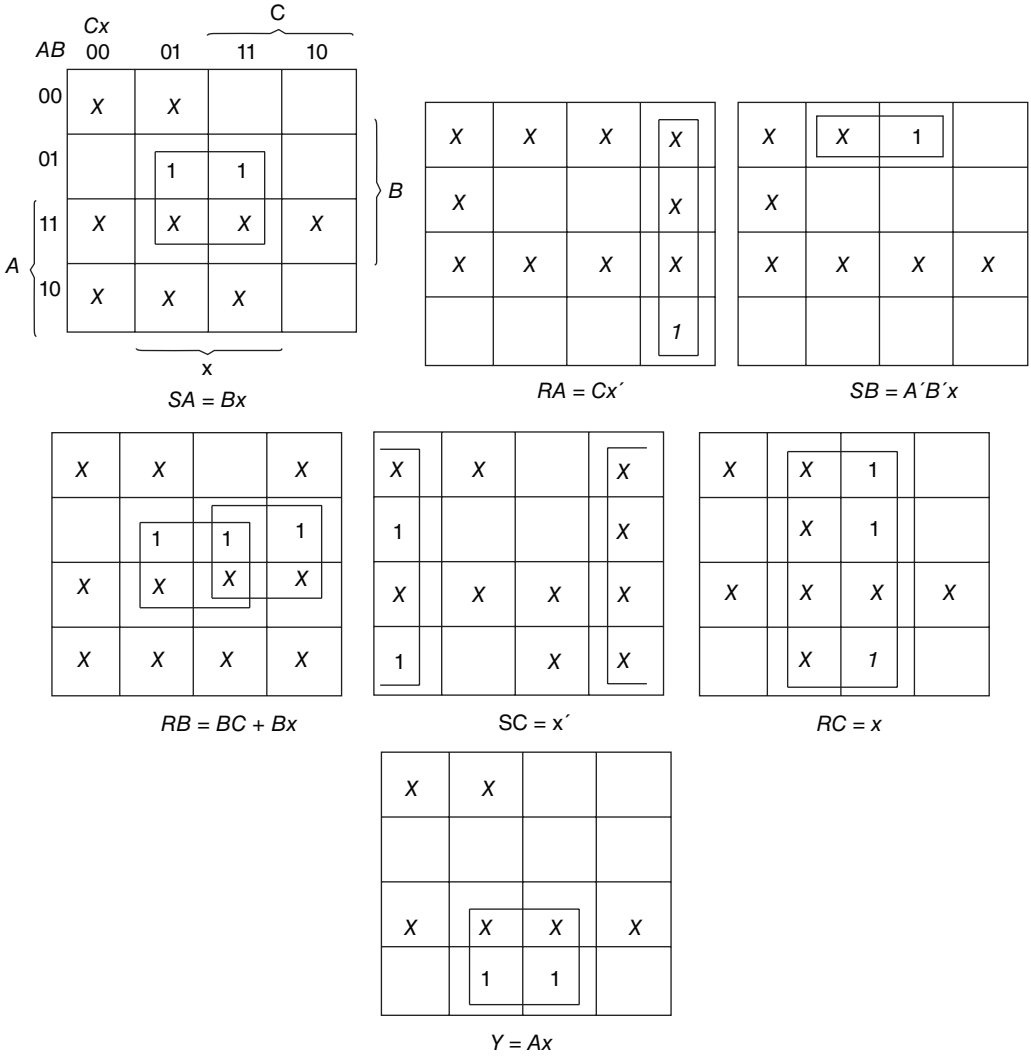


Figure 6-25 Maps for simplifying the sequential circuit of Example 6-1

In that case it is necessary to ensure that the circuit eventually goes into one of the valid states so it can resume normal operation. Otherwise, if the sequential circuit circulates among invalid states, there will be no way to bring it back to its intended sequence of state transitions. Although one can assume that this undesirable condition is not supposed to occur, a careful designer must ensure that this situation never occurs.

It was stated previously that unused states in a sequential circuit can be treated as don't-care conditions. Once the circuit is designed, the m flip-flops in the system can be in any one of 2^m possible states. If some of these states were taken as don't-care conditions, the circuit must be investigated to determine the effect of these unused states. The next state from invalid states can

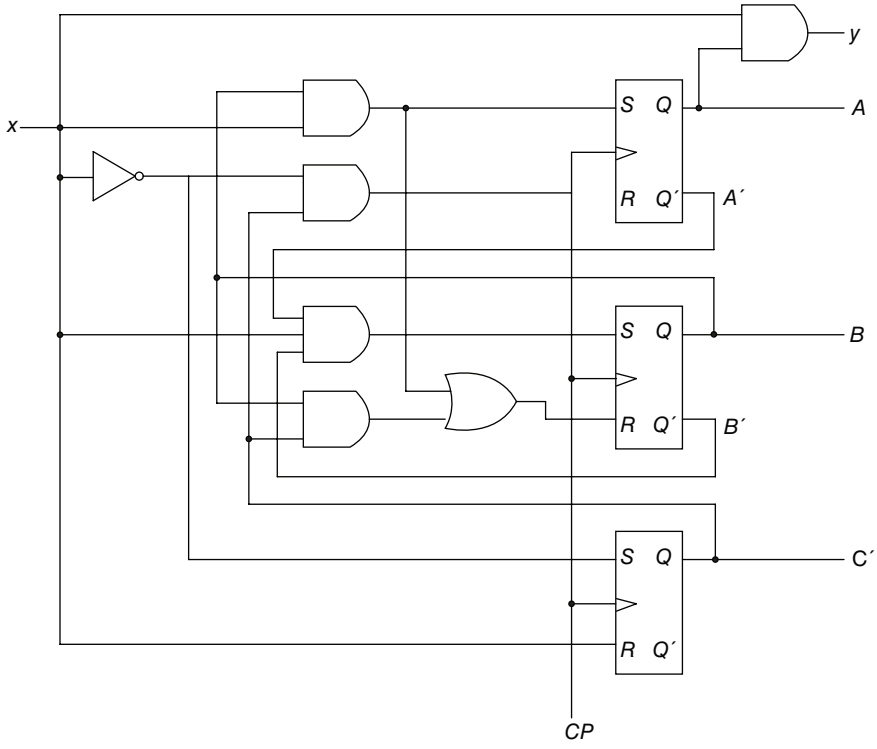


Figure 6-26 Logic diagram for Example 6-1

be determined from the analysis of the circuit. In any case, it is always wise to analyze a circuit obtained from a design to ensure that no mistakes were made during the design process.

EXAMPLE 6-2: Analyze the sequential circuit obtained in Example 6-1 and determine the effect of the unused states.

The unused states are 000, 110, and 111. The analysis of the circuit is done by the method outlined in Section 6-4. The maps of Fig. 6-25 may also help in the analysis. What is needed here is to start with the circuit diagram of Fig. 6-26 and derive the state table or diagram. If the derived state table is identical to Table 6-6 (or the state-table part of Table 6-11), then we know that the design is correct. In addition, we must determine the next states from the unused states 000. 110, and 111.

The maps of Fig. 6-25 can help in finding the next state from each of the unused states. Take, for instance, the unused state 000. If the circuit, for some reason, happens to be in the present state 000, an input $x = 0$ will transfer the circuit to some next state and an input $x = 1$ will transfer it to another (or the same) next state. We first investigate minterm $ABCx = 0000$. From the maps, we see that this minterm is not included in any function except for *SC,* i.e., the set input of flip-flop *C*. Therefore, flip-flops *A* and *B* will not change but flip-flop *C* will

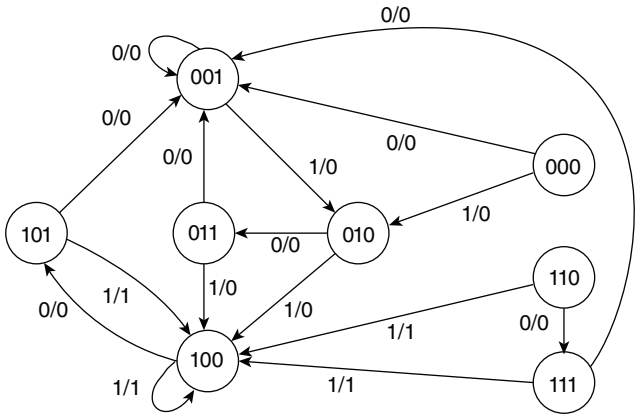


Figure 6-27 State diagram for the circuit of Fig. 6-26

be set to 1. Since the present state is $ABC = 000$, the next state will be $ABC = 001$. The maps also show that minterm $ABCx = 0001$ is included in the functions for *SB* and *RC*. Therefore, *B* will be set and *C* will be cleared. Starting with $ABC = 000$ and setting *B,* we obtain the next state $ABC = 010$ (*C* is already cleared). Investigation of the map for outputs *y* shows that *y* will be 0 for these two minterms.

The result of the analysis procedure is shown in the state diagram of Fig. 6-27. The circuit operates as intended, as long as it stays within the states 001, 010, 011, 010, and 101. If it ever finds itself in one of the invalid states 000, 110, or 111, it goes to one of the valid states within one or two clock pulses. Thus the circuit is self-starting and self-correcting, since it eventually goes to a valid state from which it continues to operate as required.

An undesirable situation would have occurred if the next state of 110 for $x = 1$ happened to be 111 and the next state of 111 for $x = 0$ or 1 happened to be 110. Then, if the circuit starts from 110 or 111, it will circulate and stay between these two states forever. Unused states that cause such undesirable behavior should be avoided; if they are found to exist, the circuit should be redesigned. This can be done most easily by specifying a valid next state for any unused state that is found to circulate among invalid states.

6.8 Design of Counters

A sequential circuit that goes through a prescribed sequence of states upon the application of input pulses is called a *counter.* The input pulses, called *count pulses,* may be clock pulses, or they may originate from an external source and may occur at prescribed intervals of time or at random. In a counter, the sequence of states may follow a binary count or any other sequence of states. Counters are found in almost all equipment containing digital logic. They are used for counting the number of occurrences of an event and are useful for generating timing sequences to control operations in a digital system.

Of the various sequences a counter may follow, the straight binary sequence is the simplest and most straightforward. A counter that follows the binary sequence is called a *binary counter,*

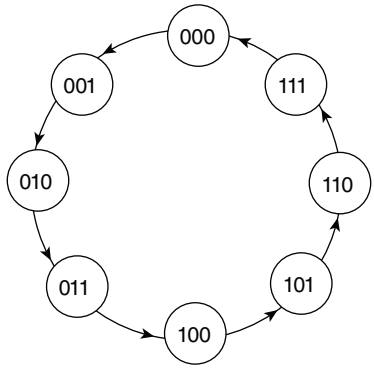


Figure 6-28 State diagram of a 3-bit binary counter

An *n*-bit binary counter consists of *n* flip-flops and can count in binary from 0 to $2^n - 1$. *As* an example, the state diagram of a 3-bit counter is shown in Fig. 6-28. As seen from the binary states indicated inside the circles, the flip-flop outputs repeat the binary count sequence with a return to 000 after 111. The directed lines between circles are not marked with input-output values as in other state diagrams. Remember that state transitions in clocked sequential circuits occur during a clock pulse; the flip-flops remain in their present states if no pulse occurs. For this reason, the clock pulse variable *CP* does not appear explicitly as an input variable in a state diagram or state table. From this point of view, the state diagram of a counter does not have to show input-output values along the directed lines. The only input to the circuit is the count pulse, and the outputs are directly specified by the present states of the flip-flops. The next state of a counter depends entirely on its present state, and the state transition occurs every time the pulse occurs. Because of this property, a counter is completely specified by a list of the *count sequence,* i.e., the sequence of binary states that it undergoes.

The count sequence of a 3-bit binary counter is given in Table 6-12. The next number in the sequence represents the next state reached by the circuit upon the application of a count pulse.

Table 6-12 Excitation table for a 3-bit binary counter

Count sequence			Flip-flop inputs		
A_2	A_1	A_0	TA_2	TA_1	TA_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	1	1	1
1	0	0	0	0	I
1	0	1	0	1	I
1	1	0	0	0	1
1	1	1	1	1	1

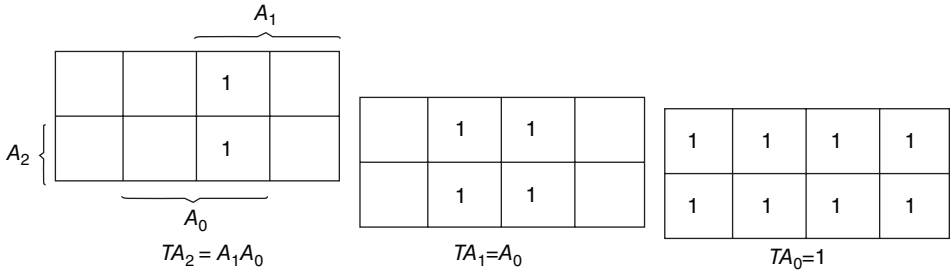


Figure 6-29 Maps for a 3-bit binary counter

The count sequence repeats after it reaches the last value, so that state 000 is the next state after 111. The count sequence gives all the information needed to design the circuit. It is not necessary to list the next states in a separate column because they can be read from the next number in the sequence. The design of counters follows the same procedure as that outlined in Section 6-7, except that the excitation table can be obtained directly from the count sequence.

Table 6-12 is the excitation table for the 3-bit binary counter. The three flip-flops are given variable designations A_2, A_1, and A_0. Binary counters are most efficiently constructed with T flip-flops (or JK flip-flop with J and K lied together). The flip-flop excitation for the T inputs is derived from the excitation table of the T flip-flop and from inspection of the state transition from a given count (present state) to the next below it (next state). As an illustration, consider the flip-flop input entries for row 001. The present state here is 001 and the next state is 010, which is the next count in the sequence. Comparing these two counts, we note that A_2 goes from 0 to 0; so TA_2 is marked with a 0 because flip-flop A_2 must remain unchanged when a clock pulse occurs. A_1 goes from 0 to 1; so TA_1 is marked with a 1 because this flip-flop must be complemented in the next clock pulse. Similarly, A_0 goes from 1 to 0, indicating that it must be complemented; so TA_0 is marked with a 1. The last row with present state 111 is compared with the first count 000 which is its next state. Going from all 1's to all 0's requires that all three flip-flops be complemented.

The flip-flop input functions from the excitation tables are simplified in the maps of Fig. 6-29. The Boolean functions listed under each map specify the combinational-circuit part of the counter. Including these functions with the three flip-flops, we obtain the logic diagram of the counter as shown in Fig. 6-30.

A counter with n flip-flops may have a binary sequence of less than 2^n numbers. A BCD counter counts the binary sequence from 0000 to 1001 and returns to 0000 to repeat the

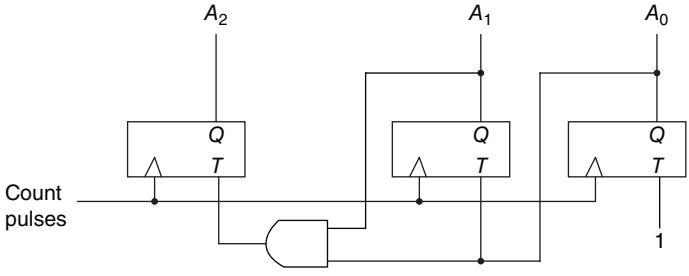


Figure 6-30 Logic diagram of 3-bit binary counter

sequence. Other counters may follow an arbitrary sequence which may not be the straight binary sequence. In any case, the design procedure is the same. The count sequence is listed and the excitation table is obtained by comparing a present count with the next count listed below it. A tabulated count sequence always assumes a repeated count, so that the next state of the last entry is the first count listed.

EXAMPLE 6-3: Design a counter that has a repeated sequence of six states as listed in Table 6-13.

In this sequence, flip-flops *B* and *C* repeat the binary count 00, 01, 10, while flip-flop A alternates between 0 and 1 every three counts. The count sequence for *A, B, C* is not straight binary and two states, 011 and 111, are not used. The choice of *JK* flip-flops results in the excitation table of Table 6-13. Inputs *KB* and *KC* have only 1's and *X*'s in their columns, so these inputs are always 1. The other flip-flop input functions can be simplified using minterms 3 and 7 as don't-care conditions. The simplified functions are:

$$JA = B \qquad KA = B$$
$$JB = C \qquad KB = 1$$
$$JC = B' \qquad KC = 1$$

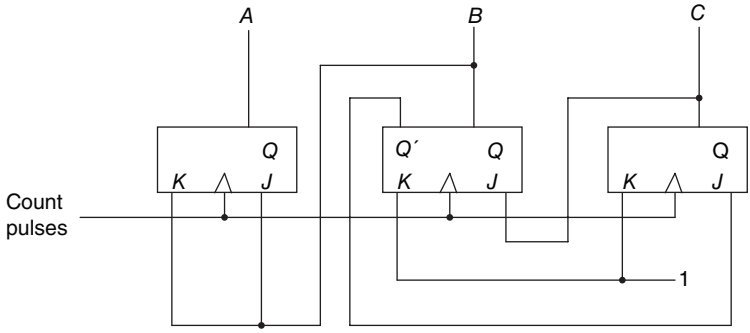
The logic diagram of the counter is shown in Fig, 6-31(a). Since there are two unused states, we analyze the circuit to determine their effect. The state diagram so obtained is drawn in Fig. 6-31(b). If the circuit ever goes to an invalid state, the next count pulse transfers it to one of the valid states, and it continues to count correctly. Thus the counter is self-starting. A self-starting counter is one that can start from any state but eventually reaches the normal count sequence.

6.9 Design with State Equations

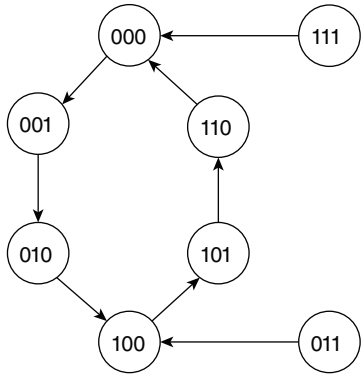
A sequential circuit can be designed by means of state equations rather than an excitation table. As shown in Section 6-4, a state equation is an algebraic expression that gives the conditions for the next state as a function of the present state and input variables. The state equations of a sequential circuit express in algebraic form the same information which is expressed in tabular form in a state table.

Table 6-13 Excitation table for Example 6-3

Count sequence			Flip-flop inputs					
A	*B*	*C*	*JA*	*KA*	*JB*	*KB*	*JC*	*KC*
0	0	0	0	*X*	0	*X*	1	*X*
0	0	1	0	*X*	1	*X*	*X*	1
0	1	0	1	*X*	*X*	1	0	*X*
1	0	0	*X*	0	0	*X*	1	*X*
1	0	1	*X*	0	1	*X*	*X*	1
1	1	0	*X*	1	*X*	1	0	*X*



(a) Logic diagram of counter



(b) State diagram of counter

Figure 6-31 Solution to Example 6-3

The state equation method is convenient when the circuit is already specified in this form or when the state equations are easily derived from the state table. This is the preferred method when *D* flip-flops are used. The method may sometimes be convenient to use with *JK* flip-flops. The application of this procedure to circuits with *RS* or *T* flip-flops is possible but involves a considerable amount of algebraic manipulation. Here we will show the application of this method to sequential circuits employing *D* or *JK* flip-flops. The starting point in each case is the flip-flop characteristic equation derived in Section 6-2.

6.9.1 Sequential Circuits with *D* Flip-flops

The characteristic equation of the *D* flip-flop is derived in Fig, 6-5(d):

$$Q(t + 1) = D$$

This equation states that the next state of the flip-flop is equal to the present value of its *D* input and is independent of the value of the present state. This means that the entries for the next state in the state table are exactly the same as the *D* inputs. Therefore, it is not necessary to derive the

flip-flop input conditions for the excitation table because this information is already available in the next state columns.

Take, for example, the excitation table of Table 6-10. The next state column for *A* has four 1's, and so does the column for the next state of *B.* To design this circuit with *D* flip-flops, we write the state equations and equate them to the corresponding *D* inputs:

$$A(t + 1) = DA(A, B, x) = \sum(2,4,5,6)$$
$$B(t + 1) = DB(A, B, x) = \sum(1,3,5.6)$$

where *DA* and *DB* are the flip-flop input functions for *D* flip-flops *A* and *B,* respectively, and each function is expressed as the sum of four minterms. The simplified functions can be obtained by means of two three-variable maps. The simplified flip-flop input functions are:

$$DA = AB' + Bx'$$
$$DB = A'x + B'x + A\,B\,x'$$

If there are unused states in the sequential circuit, they must be considered, together with the inputs, as don't-care combinations. The don't-care minterms thus obtained can be used to simplify the state equations of the *D* flip-flop input functions.

EXAMPLE 6-4: Design a sequential circuit with four flip-flops, *A*, *B*, C, and *D.* The next states of *B*, C, and *D* are equal to the present states of *A*, *B*, and C. respectively. The next state of *A* is equal to the exclusive-OR, of the present states of C and *D.*

From the statement of the problem, it is convenient to first write the state equations for the circuit:

$$A(t + 1) = C \oplus D$$
$$B(t + 1) = A$$
$$C(t + 1) = B$$
$$D(t + 1) = C$$

This circuit specifies a *feedback shift register.* In a feedback shift register, each flip-flop transfers or shifts its content to the next flip-flop when a clock pulse occurs, but the next state of the first flip-flop (*A* in this case) is some function of the present state of other flip-flops. Since the state equations are very simple, the most convenient flip-flop to use is the *D* type.

The flip-flop input functions for this circuit are taken directly from the state equations, with the next state variable replaced by the flip-flop input variable:

$$DA = C \oplus D$$
$$DB = A$$
$$DC = B$$
$$DD = C$$

The circuit can be constructed with four *D* flip-flops and one exclusive-OR gate.

6.9.2 State Equations with *JK* Flip-flops§

The characteristic equation for the *JK* flip-flop is derived in Fig. 6-6(d):

$$Q(t + 1) = (J)Q' + (K')Q$$

Input variables *J* and *K* are enclosed in parentheses so as not to confuse the AND terms of the characteristic equation with the two-letter convention which has been used to represent the flip-flop input variables.

The sequential circuit can be derived directly from the state equations without having to draw the excitation table. This is done by means of a matching process between the state equation for each flip-flop and the general characteristic equation of the *JK* flip-flop. The matching process consists of manipulating each state equation until it is in the form of the characteristic equation. Once this is done, the functions for inputs *J* and *K* can be extracted and simplified. This must be done for each state equation listed, and its flip-flop variable name *A, B,* C, etc., must replace the letter *Q* in the characteristic equation.

A given state equation for $Q(t + 1)$ may be already expressed as a function of Q and Q'. More often, either Q or Q' or both would be absent in the Boolean expression. It is then necessary to manipulate the expression algebraically until both Q and Q' are included in the expression. The following example demonstrates all the possibilities that may be encountered.

EXAMPLE 6-5: Design a sequential circuit with *JK* flip-flops to satisfy the following state equations:

$$A(t + 1) = A'B'CD + A'B'C + ACD + AC'D'$$
$$B(t + 1) = A'C + CD' + A'BC$$
$$C(t + 1) = B$$
$$D(t + 1) = D'$$

The input functions for flip-flop *A* are derived by this method by arranging the state equation and matching it with the characteristic equation as follows:

$$\begin{aligned} A(t + 1) &= (B'CD + B'C)A' + (CD + C'D')A \\ &= (J)A' + (K')A \end{aligned}$$

From the equality of the two equations, we deduce the input functions for flip-flop *A* to be:

$$J = B'CD + B'C = B'C$$
$$K = (CD + C'D')' = CD' + CD$$

The state equation for flip-flop *B* can be arranged as follows:

$$B(t + 1) = (A'C + CD') + (A'C')B$$

§This part may be omitted without loss of continuity.

However, this form is not suitable for matching with the characteristic equation because the variable B' is missing. If the first quantity in parentheses is ANDed with $(B' + B)$, the equation remains the same but with the variable B' included. Thus:

$$\begin{aligned} B(t+1) &= (A'C + CD')(B' + B) + (A'C')B \\ &= (A'C + CD')B' + (A'C + CD' + A'C')B \\ &= (J)B' + (K')B \end{aligned}$$

From the equality of the two equations, we deduce the input functions for flip-flop *B:*

$$\begin{aligned} J &= A'C + CD' \\ K &= (A'C + CD' + A'C')' = AC' + AD \end{aligned}$$

The state equation for flip-flop *C* can be manipulated as follows:

$$\begin{aligned} C(t+1') = B &= B(C' + \mathrm{C}) = BC' + BC \\ &= (J)C' + (K')C \end{aligned}$$

The input functions for flip-flop *C* are:

$$\begin{aligned} J &= B \\ K &= B\text{'} \end{aligned}$$

Finally, the state equation for flip-flop *D* may be manipulated for the purpose of matching as follows:

$$\begin{aligned} D(t+1) = D' &= 1.D' + O.D \\ &= (J)D' + (K')D \end{aligned}$$

which gives the input function:

$$J = K = 1$$

The derived input functions can be accumulated and listed together. The two-letter convention to designate the flip-flop input variable, not used in the above derivation, is used below:

$JA = B'C$	$KA = CD' + C'D$
$JB = A'C + CD'$	$KB = AC' + AD$
$JC = B$	$KC = B'$
$JD = 1$	$KD = 1$

The design procedure introduced here is an alternative method for determining the flip-flop input functions of a sequential circuit when *JK* flip-flops are employed. To use this procedure when a state diagram or state table is initially specified, it is necessary that the state equations be derived by the procedure outlined in Section 6-4. The state-equation method for finding flip-flop input functions can be extended to cover unused states which are considered as don't-care conditions. The don't-care minterms are written in the form of a state equation and manipulated

until they are in the form of the characteristic equation for the particular flip-flop considered. The *J* and *K* functions in the don't-care state equation are then taken as don't-care minterms when simplifying the input functions for a particular flip-flop.

REFERENCES

1. Marcus, M. P., *Switching Circuits for Engineers,* 3rd ed. Englewood Cliffs, NJ.; Prentice-Hall, 1975.
2. McCluskey, E. J., *Introduction to the Theory of Switching Circuits.* New York: McGraw-Hill Book Co., 1965.
3. Miller, R. E., *Switching Theory,* two volumes. New York: John Wiley and Sons, 1965.
4. Krieger, M., *Basic Switching Circuit Theory.* New York: The Macmillan Co., 1967.
5. Hill. F. J., and G. R. Peterson, *Introduction to Switching Theory and Logical Design.* New York: John Wiley and Sons, 1974,
6. Givone, D. D., *Introduction to Switching Circuit Theory.* New York: McGraw-Hill Book Co., 1970.
7. Kohavi, Z., *Switching and Finite Automata Theory.* New York: McGraw-Hill Book Co., 1970.
8. Phister M., *The Logical Design of Digital Computers.* New York: John Wiley and Sons, 1958.
9. Paull, M. C. and S. H. Unger, "Minimizing the Number of States in Incompletely Specified Sequential Switching Functions." *IRE Trans on Electronic Computers,* Vol. EC-8, No. 3 (September 1959), 356-66.
10. Hartmanis, J., "On the State Assignment Problem for Sequential Machines I." *IRE Trans, on Electronic Computers,* Vol. EC-10, No. 2 (June 1961), 157-65.
11. McCluskey, E. J., and S. H. Unger, "A Note on the Number of Internal Assignments for Sequential Circuits." *IRE Trans. on Electronic Computer,* Vol. EC-8, No. 4 (December 1959), 439-40.

PROBLEMS

6-1. Show the logic diagram of a clocked *RS* flip-flop with four NAND gates.

6-2. Show the logic diagram of a clocked *D* flip-flop with AND and NOR gates.

6-3. Show that the clocked *D* flip-flop of Fig. 6-5(a) can be reduced by one gate.

6-4. Consider a *JK′* flip-flop, i.e., a *JK* flip-flop with an inverter between external input *K′* and internal input *K.*

(a) Obtain the flip-flop characteristic table.

(b) Obtain the characteristic equation.

(c) Show that tying the two external inputs together forms a *D* flip-flop.

6-5. A set-dominate flip-flop has a set and a reset input. It differs from a conventional *RS* flip-flop in that an attempt to simultaneously set and reset results in setting the flip-flop.

(a) Obtain the characteristic table and characteristic equation for the set-dominate flip-flop.

(b) Obtain a logic diagram for an asynchronous set-dominate flip-flop.

6-6. Obtain the logic diagram of a master-slave *JK* flip-flop with AND and NOR gates. Include a provision for setting and clearing the flip-flop asynchronously (without a clock).

6-7. This problem investigates the operation of the master-slave *JK* flip-flop through the binary transition in the internal gales of Fig. 6-11. Evaluate the binary values (0 or 1) in the outputs of the nine gates when the inputs to the circuit go through the following sequence:

(a) $CP = 0$, $Y = 0$, $Q = 0$, and $J = K = 1$.

(b) After *CP* goes to 1 (*Y* should go to 1; *Q* remains at 0).

(c) After *CP* goes to 0 and immediately after that *J* goes to 0(*Q* should go to 1; *Y* is unaffected).

(d) After *CP* goes to 1 again (*Y* should go to 0).

(e) After *CP* goes back to 0 and immediately after that *K* goes to 0 (*Q* should go to 0).

(f) All succeeding pulses have no effect as long as *J* and *K* remain at 0.

6-8. Connect an asynchronous clear terminal to the inputs of gates 2 and 6 of the flip-flop in Fig. 6-12.

(a) Show that when the clear input is 0, the flip-flop is cleared, and remains cleared, regardless of the values of *CP* and *D* inputs.

(b) Show that when the clear input is 1, it has no effect on the normal clocked operations.

6-9. The full-adder of Fig. P6-10 receives two external inputs *x* and *y;* the third input *z* comes from the output of a *D* flip-flop. The carry output is transferred to the flip-flop every clock pulse. The external *S* output gives the sum of *x, y,* and *z.* Obtain the state table and state diagram of the sequential circuit.

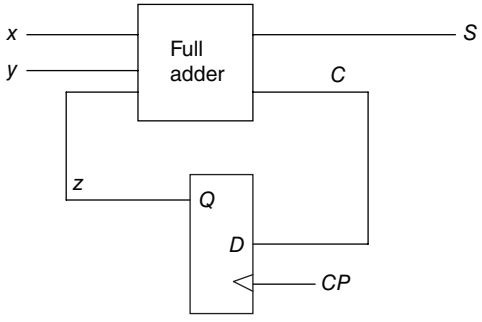


Figure P6-10

6-10. Derive the state table and state diagram of the sequential circuit of Fig. P6-11. What is the function of the circuit?

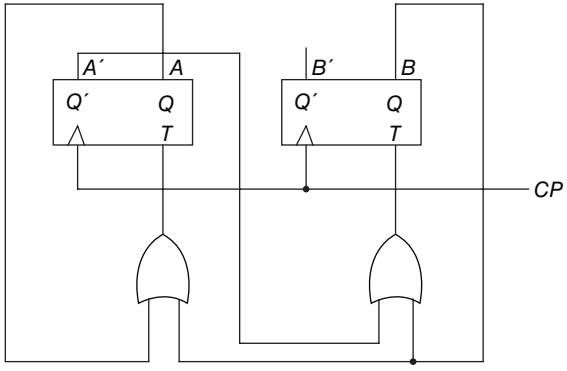


Figure P6-11

6-11. A sequential circuit has four flip-flops A, B, C, D and an input x. It is described by the following state equations:

$$A(t+1) = (CD' + C'D)x + (CD + C'D')x'$$
$$B(t+1) = A$$
$$C(t+1) = B$$
$$D(t+1) = C$$

(a) Obtain the sequence of states when $x = 1$, starting from state $ABCD = 0001$.

(b) Obtain the sequence of states when $x = 0$, starting from state $ABCD = 0000$.

6-12. A sequential circuit has two flip-flops (A and B), two inputs (x and y) and an output (z). The flip-flop input functions and the circuit output function are as follows:

$$JA = xB + y'B' \qquad KA = xy'B'$$
$$JB = xA' \qquad KB = xy' + A$$
$$z = xyA + x'y'B$$

Obtain the logic diagram, state table, state diagram, and state equations.

6-13. Reduce the number of states in the following state table and tabulate the reduced state table.

Present state	Next state		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	f	b	0	0
b	d	c	0	0
c	f	e	0	0
d	g	a	1	0
e	d	c	0	0
f	f	b	1	1
g	g	h	0	1
h	g	a	1	0

6-14. Starting from state a of the state table in problem 6-14, find the output sequence generated with an input sequence 01110010011.

6-15. Repeal problem 6-15 using the reduced table of problem 6-14. Show that the same output sequence is obtained.

6-16. Substitute binary assignment 2 of Table 6-5 to the states in Table 6-4 and obtain the binary state table. Repeal with binary assignment 3.

6-17. Obtain the excitation table of the JK' flip-flop described in problem 6-4.

6-18. Obtain the excitation table of the set-dominate flip-flop described in problem 6-5.

6-19. A sequential circuit has one input and one output. The state diagram is shown in Fig. P6-20. Design the sequential circuit with (a) T flip-flops, (b) RS flip-flops, and (c) JK flip-flops.

6-20. Design the circuit of a 4-bit register that converts the binary number stored in the register to its 2's complement value when input $x = 1$. The flip-flops of the register are of the RST type. This flip-flop has three inputs: two inputs have RS capabilities and one has a T capability. The RS inputs are used to transfer the 4-bit number when an input $y = 1$. Use the T input for the conversion.

6-21. Repeat Example 6-1 with binary assignment 3 of Table 6-5. Use JK flip-flops.

6-22. Design a BCD counter with *JK* flip-flops.

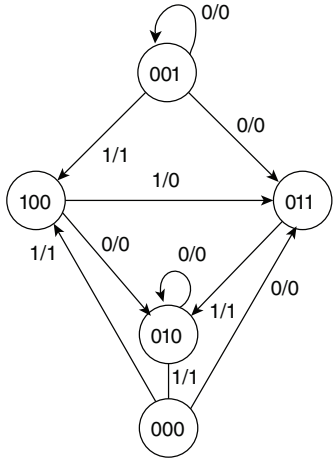


Figure P6-20

6-23. Design a counter that counts the decimal digits according to the 2, 4, 2, 1 code (Table 1-2). Use *T* flip-flops.

6-24. Verify the circuit obtained in Example 6-5 by using the excitation table method.

6-25. Design the sequential circuit described by the following state equations. Use *JK* flip-flops.

$$A(t+1) = xAB + yA'C + xy$$
$$B(t+1) = xAC + y'BC'$$
$$C(t+1) = x'B + yAB'$$

6-26. (a) Derive the state equations for the sequential circuit specified by Table 6-6, Section 6-5. List the don't-care terms. (b) Derive the flip-flop input functions from the state equations (and don't-care terms) using the method outlined in Example 6-5. Use *JK* flip-flops.

6-27. Differentiate between sequential circuit and combinational circuit.

6-28. What is the problem found in RS flip-flop? Explain how it is solved in JK flip flop.

6-29. What is the necessity of mater-slave flip-flop? Explain working of D master-slave flip-flip. Realize with all NOR gates.

6-30. Covert t flip-flip to D flip-flop.

6-31. With help of JK flip-flop design a counter which counts following binary sequence 2, 3, 1, 7, 4, 0 and repeat.

6-32. With help of RS flip-flop design a counter which counts following binary sequence 1, 3, 5, 7, 9 and repeat.

6-33. With help of T flip-flop design a counter which counts following binary sequence 0, 2, 4, 6, 8 and repeat.

6-34. With help of D flip-flop design a counter which counts following binary sequence 1, 2, 6, 4, 3 and repeat.

6-35. What is lock out in a counter? How it can be avoided?

6-36. Bring out the differences between edge triggered and level triggered flip-flip.

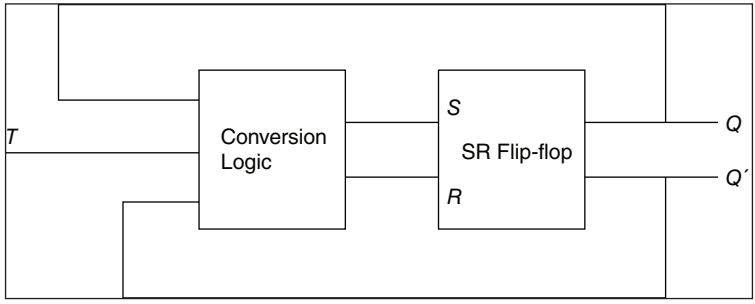
6-37. Design a counter with JK flip-flip which counts binary sequence of 7, 4, 2, 1, 5….. . Is there a chance of lockout? Explain why. How can you improve that? Explain with your design.

6-38. Explain up-down counter which counts decimal digits according to 84-2-1 with JK flip-flop. Use one control bit for up and down counter.

SOLVED EXAMPLE

With the help of SR flip-flip, realize T flip-flop.

Step 1 – (Logic Diagram)



Step 2

Truth Table of required (*T*) flip-flop

FF input	Present state	Next State
T	Q_n	Q_{n+1}
0	0	0
1	0	1
1	1	0
0	1	1

Excitation table of given (*SR*) flip-flop

Present state	Next State	Flip-flop inputs	
Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

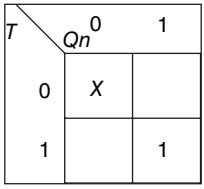
Step 3

Truth table of the required flip-flip (combining truth table and excitation table)

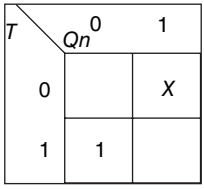
FF input	Present state	Next State	Flip-flop inputs	
T	Q_n	Q_{n+1}	S	R
0	0	0	0	X
1	0	1	1	0
1	1	0	0	1
0	1	1	X	0

Step 4

Finding the expression of given flip in terms of required flip-flop



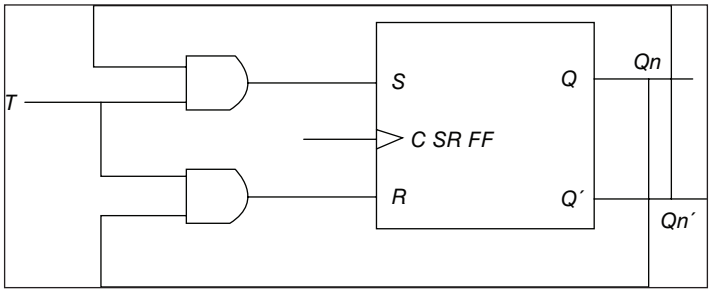
$R = TQ$



$S = TQ'$

Step 5

Logic Diagram of the converted Flip-flop.



CHAPTER 7

Registers, Counters, and the Memory Unit

7.1 Introduction

A clocked sequential circuit consists of a group of flip-flops and combinational gates connected to form a feedback path. The flip-flops are essential because, in their absence, the circuit reduces to a purely combinational circuit (provided there is no feedback path). A circuit with only flip-flops is considered a sequential circuit even in the absence of combinational gates.

An MSI circuit that contains storage cells within it is, by definition, a sequential circuit. MSI circuits that include flip-flops or other storage cells are usually classified by the function they perform rather than by the name "sequential circuit." These MSI circuits are classified in one of three categories: registers, counters, or random-access memory. This chapter presents various registers and counters available in IC form and explains their operation. The organization of the random-access memory is also presented.

A *register* is a group of binary storage cells suitable for holding binary information. A group of flip-flops constitutes a register, since each flip-flop is a binary cell capable of storing one bit of information. An *n*-bit register has a group of *n* flip-flops and is capable of storing any binary information containing *n* bits. In addition to the flip-flops, a register may have combinational gates that perform certain data-processing tasks. In its broadest definition, a register consists of a group of flip-flops and gates that effect their transition. The flip-flops hold binary information and the gates control when and how new information is transferred into the register.

Counters were introduced in Section 6-8. A counter is essentially a register that goes through a predetermined sequence of states upon the application of input pulses. The gates in a counter are connected in such a way as to produce a prescribed sequence of binary states in the register. Although counters are a special type of register, it is common to differentiate them by giving them a special name.

A memory unit is a collection of storage cells together with associated circuits needed to transfer information in and out of storage. A random-access memory (RAM) differs from a read-only memory (ROM) in that a RAM can transfer the stored information out (read) and is also capable of receiving new information in for storage (write). A more appropriate name for such a memory would be *read-write memory.*

Registers, counters, and memories are extensively used in the design of digital systems in general and digital computers in particular. Registers can also be used to facilitate the design of sequential circuits. Counters are useful for generating timing variables to sequence and control

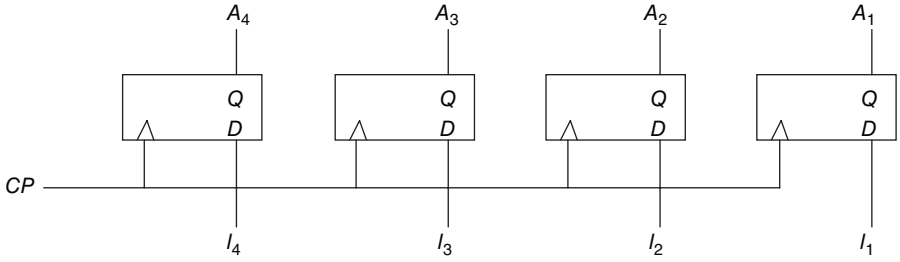


Figure 7-1 4-bit register

the operations in a digital system. Memories are essential for storage of programs and data in a digital computer. Knowledge of the operation of these components is indispensable for the understanding of the organization and design of digital systems.

7.2 Registers

Various types of registers are available in MSI circuits. The simplest possible register is one that consists of only flip-flops without any external gates. Figure 7-1 shows such a register constructed with four *D*-type flip-flops and a common clock pulse input. The clock pulse input, *CP,* enables all flip-flops so that the information presently available at the four inputs can be transferred into the 4-bit register. The four outputs can be sampled to obtain the information presently stored in the register.

The way that the flip-flops in a register are triggered is of primary importance. If the flip-flops are constructed with gated *D*-type latches as in Fig, 6-5, then information present at a data (*D*) input is transferred to the *Q* output when the enable (*CP*) is 1, and the *Q* output follows the input data as long as the *CP* signal remains 1. When *CP* goes to 0, the information that was present at the data input just before the transition is retained at the *Q* output. In other words, the flip-flops are sensitive to the pulse duration, and the register is enabled for as long as *CP* = 1. A register that responds to the pulse duration is commonly called a *gated latch,* and the *CP* input is frequently labeled with the variable *G* (instead of *CP*). Latches are suitable for use as temporary storage of binary information that is to be transferred to an external destination. They should not be used in the design of sequential circuits that have feedback connections.

As explained in Section 6-3, a flip-flop can be used in the design of clocked sequential circuits provided it is sensitive to the pulse transition rather than the pulse duration. This means that the flip-flops in the register must be of the edge-triggered or master-slave type. Normally, it is not possible to distinguish from a logic diagram whether a flip-flop is a gated latch, edge-triggered, or master-slave, because the graphic symbols for all three are the same. The distinction must be made from the name given to the unit. A group of flip-flops sensitive to pulse duration is usually called a *latch,* whereas a group of flip-flops sensitive to pulse transition is called a *register.*[*] A register can always replace a latch, but the converse should be done with caution to make sure that outputs from a latch never go to other flip-flop inputs that are triggered with the same common clock pulse. In subsequent discussions, we will always assume that any group of flip-flops

[*]For example, IC type 7475 is a 4-bit latch, whereas type 74175 is a 4-bit register.

drawn constitutes a *register* and that all flip-flops are of the edge-triggered or master-slave type. If the register is sensitive to the pulse duration, it will be referred to as a *latch.*

7.2.1 Register with Parallel Load

The transfer of new information into a register is referred to as *loading* the register. If all the bits of the register are loaded simultaneously with a single clock pulse, we say that the loading is done in parallel. A pulse applied to the *CP* input of the register of Fig. 7-1 will load all four inputs in parallel. In this configuration, the clock pulse must be inhibited from the *CP* terminal if the content of the register must be left unchanged. In other words, the *CP* input acts as an enable signal which controls the loading of new information into the register. When *CP* goes to 1, the input information is loaded into the register. If *CP* remains at 0, the content of the register is not changed. Note that the change of state in the outputs occurs at the positive edge of the pulse. If a flip-flop changes state at the negative edge, there will be a small circle under the triangle symbol in the *CP* input of the flip-flop.

Most digital systems have a master-clock generator that supplies a continuous train of clock pulses. All clock pulses are applied to all flip-flops and registers in the system. The master-clock generator acts like a pump that supplies a constant beat to all parts of the system. A separate control signal then decides what specific clock pulses will have an effect on a particular register. In such a system, the clock pulses must be ANDed with the control signal, and the output of the AND gate is then applied to the *CP* terminal of the register shown in Fig. 7-1. When the control signal is 0, the output of the AND gate is 0, and the stored information in the register remains unchanged. Only when the control signal is a 1 does the clock pulse pass through the AND gate and into the *CP* terminal for new information to be loaded into the register. Such a control variable is called a *load* control input.

Inserting an AND gate in the path of clock pulses means that logic is performed with clock pulses. The insertion of logic gates produces propagation delays between the master-clock generator and the clock inputs of flip-flops. To fully synchronize the system, we must ensure that all clock pulses arrive at the same time to all inputs of alt flip-flops so that they can all change simultaneously. Performing logic with clock pulses inserts variable delays and may throw the system out of synchronism. For this reason, it is advisable (but not necessary, as long as the delays are taken into consideration) to apply clock pulses directly to all flip-flops and control the operation of the register with other inputs, such as the *R* and *S* inputs of an *RS* flip-flop.

A 4-bit register with a load control input using *RS* flip-flops is shown in Fig. 7-2. The *CP* input of the register receives continuous synchronized pulses which are applied to ail flip-flops. The inverter in the *CP* path causes all flip-flops to be triggered by the negative edge of the incoming pulses. The purpose of the inverter is to reduce the loading of the master-clock generator. This is because the *CP* input is connected to only one gate (the inverter) instead of the four-gate inputs that would have been required if the connections were made directly into the flip-flop clock inputs (marked with small triangles).

The *clear* input goes to a special terminal in each, flip-flop through a noninverting buffer gate. When this terminal goes to 0, the flip-flop is cleared asynchronously. The clear input is useful for clearing the register to all 0's prior to its clocked operation. The clear input must be maintained at 1 during normal clocked operations (see Fig. 6-14).

The *load* input goes through a buffer gate (to reduce loading) and through a series of AND gates to the *R* and *S* inputs of each flip-flop. Although clock pulses are continuously present, it is

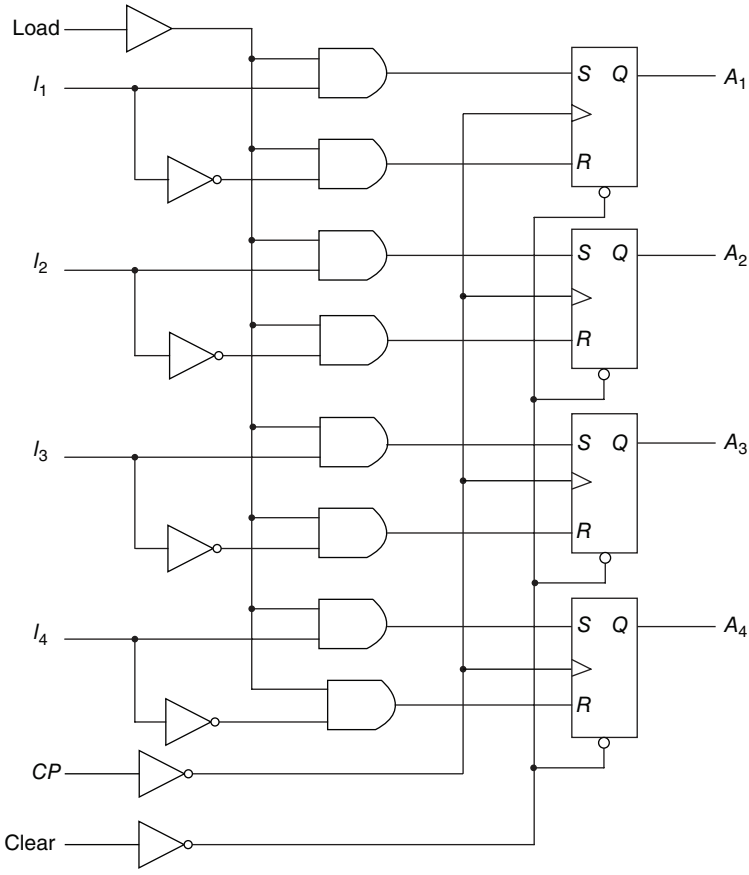


Figure 7-2 4-bit register with parallel load

the load input that controls the operation of the register. The two AND gates and the inverter associated with each input *I* determine the values of *R* and *S.* If the load input is 0, both *R* and *S* are 0, and no change of state occurs with any clock pulse. Thus, the load input is a control variable which can prevent any information change in the register as long as its input is 0. When the load control goes to 1, inputs I_1, through I_4 specify what binary information is loaded into the register on the next clock pulse. For each *I* that is equal to 1, the corresponding flip-flop inputs are $S = 1$, $R = 0$. For each *I* that is equal to 0, the corresponding flip-flop inputs are $S = 0$, $R = 1$, Thus, the input value is transferred into the register provided the load input is 1, the clear input is 1, and a clock pulse goes from 1 to 0. This type of transfer is called a *parallel-load* transfer because all bits of the register are loaded simultaneously. If the buffer gate associated with the load input is changed to an inverter gate, then the register is loaded when the load input is 0 and inhibited when the load input is 1.

A register with parallel load can be constructed with *D* flip-flops as shown in Fig. 7-3. The clock and clear inputs are the same as before. When the load input is 1, the *I* inputs are transferred into the register on the next clock pulse. When the load input is 0, the circuit inputs are inhibited and the *D* flip-flops are reloaded with their present value, thus maintaining the con-

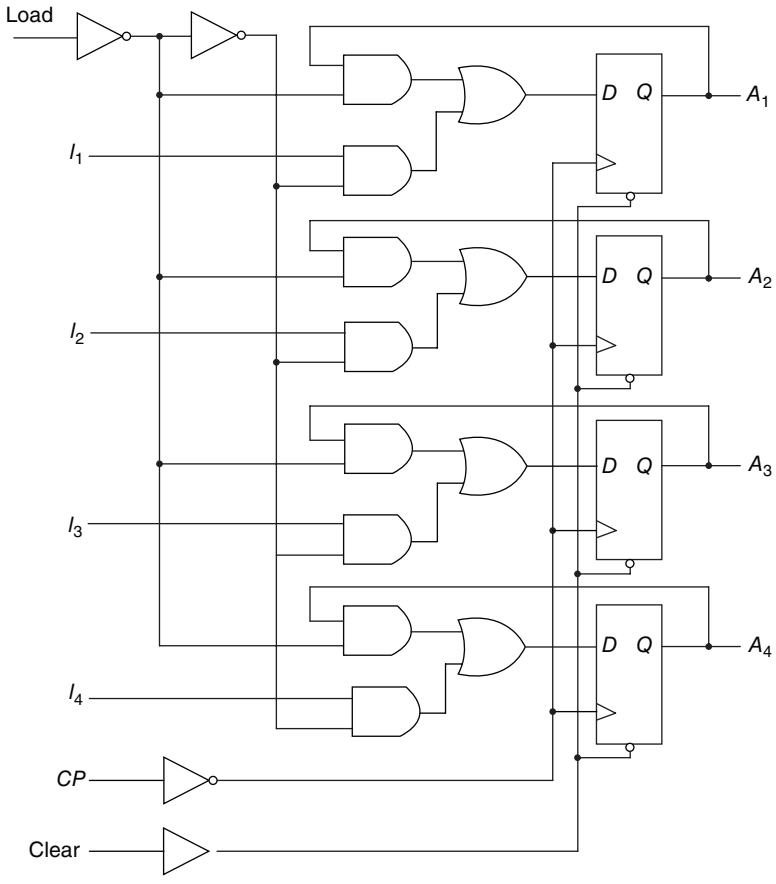


Figure 7-3 Register with parallel load using *D* flip-flops

tent of the register. The feedback connection in each Flip-flop is necessary when *D* type is used because a *D* flip-flop does not have a "no-change" input condition. With each clock pulse, the *D* input determines the next state of the output. To leave the output unchanged, it is necessary to make the *D* input equal to the present *Q* output in each flip-flop.

7.2.2 Sequential Logic Implementation

We saw in Chapter 6 that a clocked sequential circuit consists of a group of flip-flops and combinational gates. Since registers are readily available as MSI circuits, it becomes convenient at times to employ a register as part of the sequential circuit. A block diagram of a sequential circuit that uses a register is shown in Fig. 7-4. The present state of the register and the external inputs determine the next state of the register and the values of external outputs. Part of the combinational circuit determines the next state and the other part generates the outputs. The next state value from the combinational circuit is loaded into the register with a clock pulse. If the register has a load input, it must be set to I; otherwise, if the register has no load input (as in Fig. 7-1), the next state value will be transferred automatically every clock pulse.

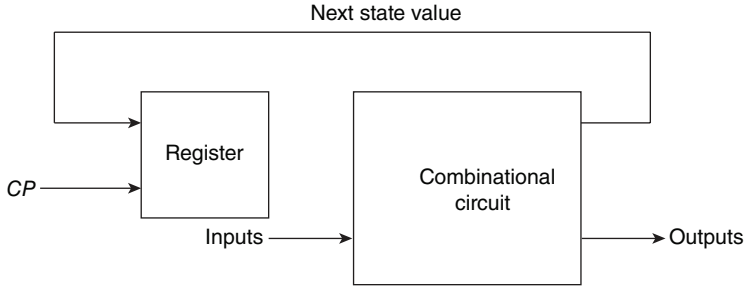


Figure 7-4 Block diagram of a sequential circuit

The combinational circuit part of a sequential circuit can be implemented by any of the methods discussed in Chapter 5. It can be constructed with SSI gates, with ROM, or with a programmable logic array (PLA). By using a register, it is possible to reduce the design of a sequential circuit to that of a combinational circuit connected to a register.

EXAMPLE 7-1: Design the sequential circuit whose state table is listed in Fig, 7-5(a).

The state table specifies two flip-flops A_1 and A_2, one input x, and one output y. The next state and output information is obtained directly from the table:

$$A_1(t+1) = \sum(4,6)$$
$$A_2(t+1) = \sum(1, 2, 5, 6)$$
$$y(A_1, A_2, x) = \sum(3, 7)$$

The minterm values are for variables A_1, A_2, and x, which are the present state and input variables. The functions for the next state and output can be simplified by means of maps to give:

$$A_1(t+1) = A_1x'$$
$$A_2(t+1) = A_2 \oplus x$$
$$y = A_2x$$

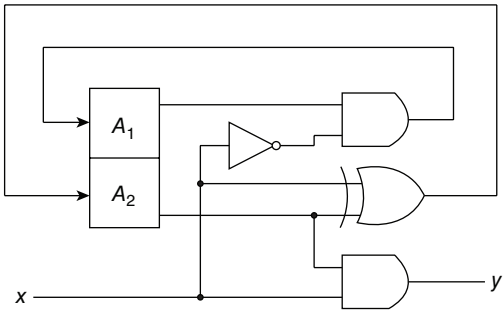
The logic diagram is shown in Fig. 7-5(b).

EXAMPLE 7-2: Repeat Example 7-1, but now use a ROM and a register.

The ROM can be used to implement the combinational circuit and the register will provide the flip-flops. The number of inputs to the ROM is equal to the number of flip-flops plus the number of external inputs. The number of outputs of the ROM is equal to the number of flip-flops plus the number of external outputs. In this case we have three inputs and three outputs for the ROM; so its size must be 8×3. The implementation is shown in Fig. 7-6. The ROM truth table is identical to the state table with "present state" and "inputs" specifying the address of ROM and "next state" and "outputs" specifying the ROM outputs. The next state values must be connected from the ROM outputs to the register inputs.

Present state		Input	Next state		Output
A_1	A_2	x	A_1	A_2	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	0	1
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1

(a) State Table



(b) Logic Diagram

Figure 7-5 Example of sequential-circuit implementation

7.3 Shift Registers

A register capable of shifting its binary information either to the right or to the left is called a *shift register.* The logical configuration of a shift register consists of a chain of flip-flops connected in cascade, with the output of one flip-flop connected to the input of the next flip-flop. All flip-flops receive a common clock pulse which causes the shift from one stage to the next.

The simplest possible shift register is one that uses only flip-flops, as shown in Fig. 7-7. The Q output of a given flip-flop is connected to the D input of the flip-flop at its right. Each clock pulse shifts the contents of the register one bit position to the right. The *serial input* determines

ROM truth table

Address			Outputs		
1	2	3	1	2	3
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	0	1
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1

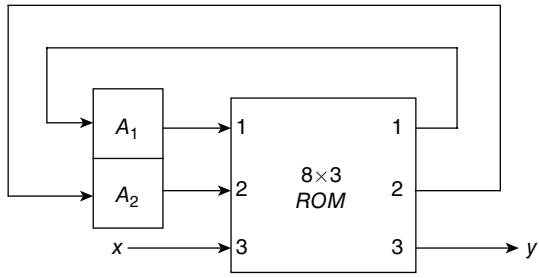


Figure 7-6 Sequential circuit using a register and a ROM

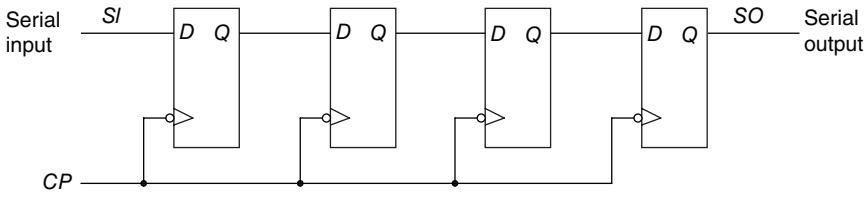


Figure 7-7 Shift register

what goes into the leftmost flip-flop during the shift. The *serial output* is taken from the output of the rightmost flip-flop prior to the application of a pulse. Although this register shifts its contents to the right, if we turn the page upside down, we find that the register shifts its contents to the left. Thus a unidirectional shift register can function either as a shift-right or as a shift-left register.

The register in Fig. 7-7 shifts its contents with every clock pulse during the negative edge of the pulse transition. (This is indicated by the small circle associated with the clock input in all flip-flops.) If we want to control the shift so that it occurs only with certain pulses but not with others, we must control the *CP* input of the register. It will be shown later that the shift operations can be controlled through the *D* inputs of the flip-flops rather than through the *CP* input. If, however, the shift register in Fig. 7-7 is used, the shift can easily be controlled by means of an external AND gate as shown below.

7.3.1 Serial Transfer

A digital system is said to operate in a serial mode when information is transferred and manipulated one bit at a time. The content of one register is transferred to another by shifting the bits from one register to the other. The information is transferred one bit at a time by shifting the bits out of the source register into the destination register.

The serial transfer of information from register *A* to register *B* is done with shift registers, as shown in the block diagram of Fig. 7-8(a). The serial output (*SO*) of register *A* goes to the serial input (*SI*) of register *B.* To prevent the loss of information stored in the source register, the *A* register is made to circulate its information by connecting the serial output to its serial input

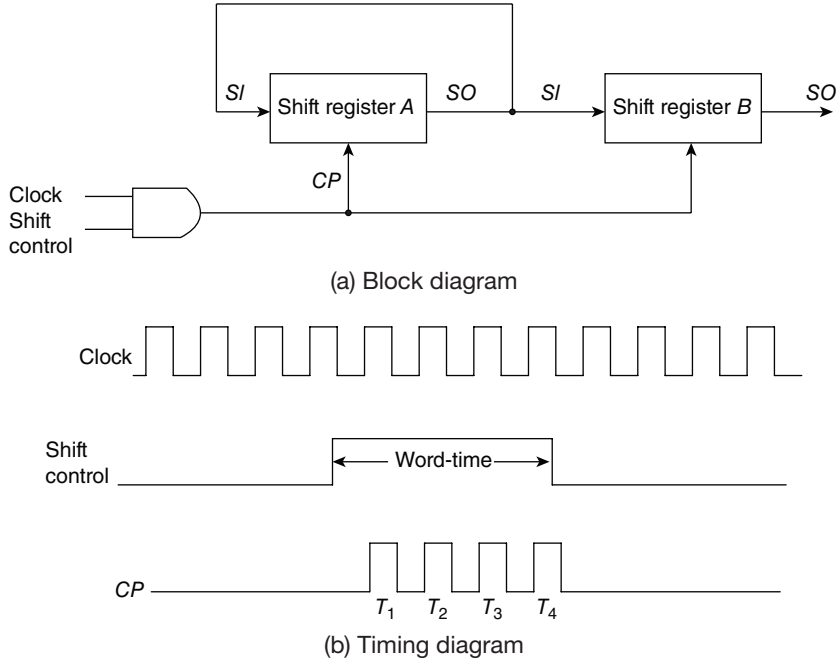


Figure 7-8 Serial transfer from register *A* to register *B*

Table 7-1 Serial transfer example

Timing pulse	Shift register A				Shift register B				Serial output of B
Initial value	1	0	1	1	0	0	1	0	0
After T_1	1	1	0	1	1	0	0	1	1
After T_2	1	1	1	0	1	1	0	0	0
After T_1	0	1	1	1	0	1	1	0	0
After T_4	1	0	1	1	1	0	1	1	1

terminal. The initial content of register *B* is shifted out through its serial output and is lost unless it is transferred to a third shift register. The shift-control input determines when and by how many times the registers are shifted. This is done by the AND gate that allows clock pulses to pass into the *CP* terminals only when the shift-control is 1.

Suppose the shift registers have four bits each. The control unit that supervises the transfer must be designed in such a way that it enables the shift registers, through the shift-control signal, for a fixed time duration equal to four clock pulses. This is shown in the timing diagram of Fig. 7-8(b). The shift-control signal is synchronized with the clock and changes value just after the negative edge of a clock pulse. The next four clock pulses find the shift-control signal in the I state, so the output of the AND gate connected to the *CP* terminals produces the four pulses T_1, T_2, T_3, and T_4. The fourth pulse changes the shift control to 0 and the shift registers are disabled.

Assume that the binary content of *A* before the shift is 1011 and that of *B,* 0010. The serial transfer from *A* to *B* will occur in four steps as shown in Table 7-1. After the first pulse T_1, the rightmost bit of *A* is shifted into the leftmost bit of *B* and, at the same time, this bit is circulated into the leftmost position of *A.* The other bits of *A* and *B* are shifted once to the right. The previous serial output from *B* is lost and its value changes from 0 to 1. The next three pulses perform identical operations, shifting the bits of *A* into S, one at a time. After the fourth shift, the shift control goes to 0 and both registers *A* and *B* have the value 1011. Thus, the content of *A* is transferred into *B* while the content of *A* remains unchanged.

The difference between serial and parallel modes of operation should be apparent from this example. In the parallel mode, information is available from all bits of a register and all bits can be transferred simultaneously during one clock pulse. In the serial mode, the registers have a single serial input and a single serial output. The information is transferred one bit at a time while the registers are shifted in the same direction.

Computers may operate in a serial mode, a parallel mode, or in a combination of both. Serial operations are slower because of the time it takes to transfer information in and out of shift registers. Serial computers, however, require less hardware to perform operations because one common circuit can be used over and over again to manipulate the bits coming out of shift registers in a sequential manner. The time interval between clock pulses is called the *bit time,* and the time required to shift the entire contents of a shift register is called the *word time.* These timing sequences are generated by the control section of the system. In a parallel computer, control signals are enabled during one clock pulse interval. Transfers into registers are in parallel, and they occur upon application of a single clock pulse. In a serial computer, control signals must be maintained for a period equal to one word time. The pulse applied every bit time transfers the

result of the operation, one at a time, into a shift register. Most computers operate in a parallel mode because this is a faster mode of operation.

7.3.2 Bidirectional Shift Register with Parallel Load

Shift registers can be used for convening serial data to parallel data, and vice versa. If we have access to all the flip-flop outputs of a shift register, then information entered serially by shifting can be taken out in parallel from the outputs of the flip-flops. If a parallel load capability is added to a shift register, then data entered in parallel can be taken out in serial fashion by shifting the data stored in the register.

Some shift registers provide the necessary input and output terminals for parallel transfer. They may also have both shift-right and shift-left capabilities. The most general shift register has all the capabilities listed below. Others may have only some of these functions, with at least one shift operation.

1. A *clear* control to clear the register to 0.
2. A *CP* input for clock pulses to synchronize all operations.
3. A *shift-right* control to enable the shift-right operation and the serial input and output lines associated with the shift-right.
4. A *shift-left* control to enable the shift-left operation and the *serial input* and *output* lines associated with the shift-left.
5. A *parallel-bad* control to enable a parallel transfer and the *n* input lines associated with the parallel transfer.
6. *n* parallel output lines.
7. A control state that leaves the information in the register unchanged even though clock pulses are continuously applied.

A register capable of shifting both right and left is called a *bidirectional shift register.* One that can shift in only one direction is called a *unidirectional shift register.* If the register has both shift and parallel-load capabilities, it is called a *shift register with parallel load.*

The diagram of a shift register that has all the capabilities listed above is shown in Fig. 7-9.[†] It consists of four *D* flip-flops, although *RS* flip-flops could be used provided an inverter is inserted between the *S* and *R* terminals. The four multiplexers (MUX) are part of the register and are drawn here in block diagram form. (See Fig. 5-16 for the logic diagram of the multiplexer.) The four multiplexers have two common selection variables, s_1 and s_0. Input 0 in each MUX is selected when $s_1s_0 = 00$, input 1 is selected when $s_1s_0 = 01$, and similarly for the other two inputs to the multiplexers.

The s_1 and s_0 inputs control the mode of operation of the register as specified in the function entries of Table 7-2. When $s_1s_0 = 00$. the present value of the register is applied to the *D* inputs of the flip-flops. This condition forms a path from the output of each flip-flop into the input of the same flip-flop. The next clock pulse transfers into each flip-flop the binary value it held previously, and no change of state occurs. When $s_1s_0 = 01$, terminals 1 of the multiplexer inputs have a path to the *D* inputs of the flip-flops. This causes a shift-right operation, with the serial input

[†]This is similar to IC type 74194.

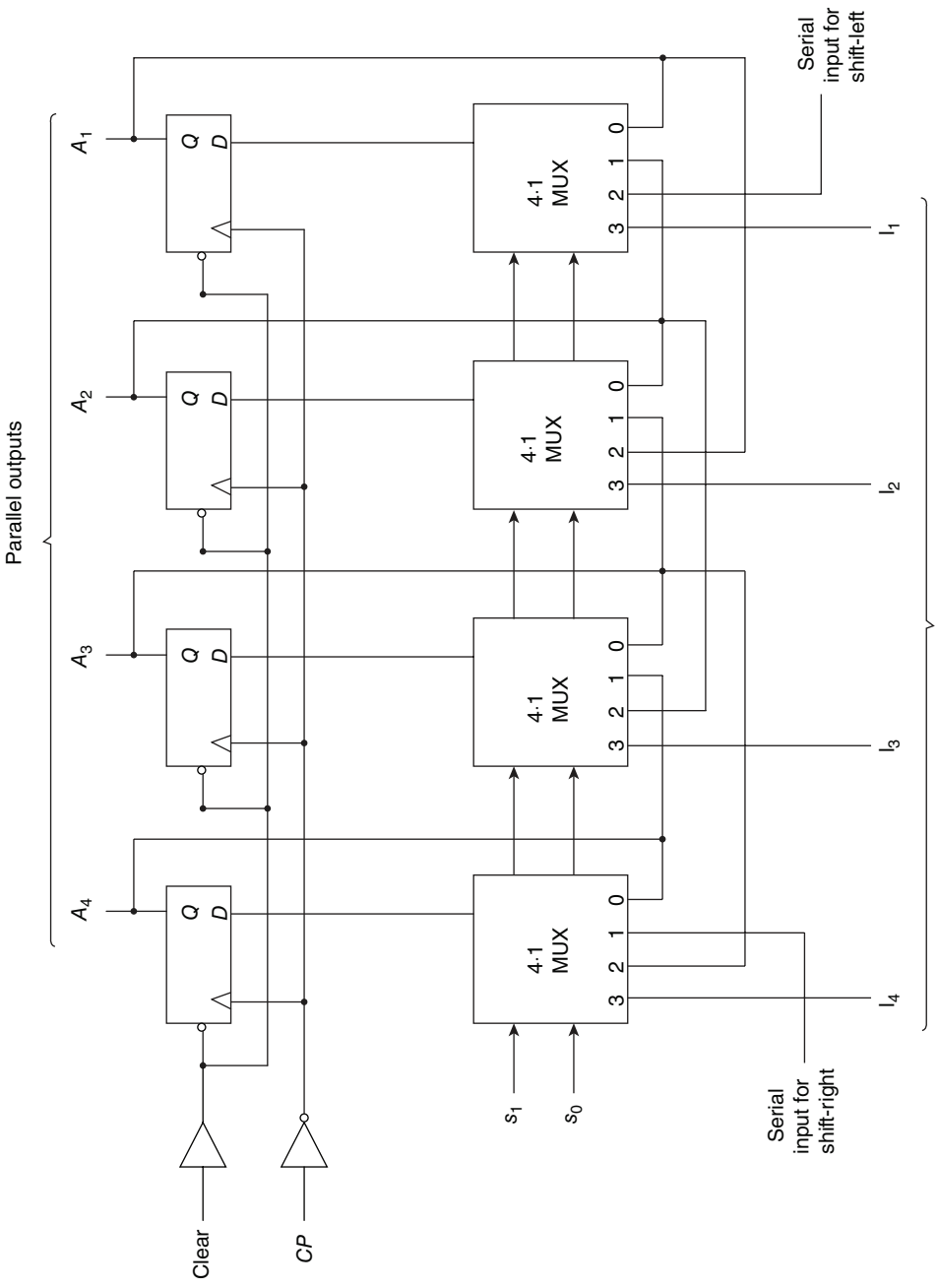


Figure 7-9 4-bit bidirectional shift register with parallel load

Table 7-2 Function table for the register of Fig. 7-9

Mode control		Register operation
s_1	s_0	
0	0	No change
0	1	Shirt right
1	0	Shift left
1	1	Parallel load

transferred into flip-flop A_4. When s_1s_0= 10, a shift-left operation results, with the other serial input going into flip-flop A_1. Finally, when s_1s_0 = 11, the binary information on the parallel input lines is transferred into the register simultaneously during the next clock pulse.

A bidirectional shift register with parallel load is a general-purpose register capable of performing three operations: shift left, shift right, and parallel load. Not all shift registers available in MSI circuits have all these capabilities. The particular application dictates the choice of one MSI shift register over another.

7.3.3 Serial Addition

Operations in digital computers are mostly done in parallel because this is a faster mode of operation. Serial operations are slower but require less equipment. To demonstrate the serial mode of operation, we present here the design of a serial adder. The parallel counterpart was discussed in Section 5-2.

The two binary numbers to be added serially are stored in two shift registers. Bits are added one pair at a time, sequentially, through a single full-adder (FA) circuit, as shown in Fig. 7-10. The carry out of the full-adder is transferred to a *D* flip-flop. The output of this flip-flop is then used as an input carry for the next pair of significant bits. The two shift registers are shifted to the right for one word-time period. The sum bits from the *S* output of the full-adder could be transferred into a third shift register. By shifting the sum into *A* white the bits of *A* are shifted out, it is possible to use one register for storing both the augend and the sum bits. The serial input (SI) of register *B* is able to receive a new binary number while the addend bits are shifted out during the addition.

The operation of the serial adder is as follows. Initially, the *A* register holds the augend, the *B* register holds the addend, and the carry flip-flop is cleared to 0. The serial outputs (*SO*) of *A* and *B* provide a pair of significant bits for the full-adder at *x* and *y.* Output *Q* of the flip-flop gives the input carry at *z*. The shift-right control enables both registers and the carry flip-flop; so at the next clock pulse, both registers are shifted once to the right, the sum bit from *S* enters the leftmost flip-flop of *A,* and the output carry is transferred into flip-flop *Q.* The shift-right control enables the registers for a number of clock pulses equal to the number of bits in the registers. For each succeeding clock pulse, a new sum bit is transferred to *A,* a new carry is transferred to *Q,* and both registers are shifted once to the right. This process continues until the shift-right control is disabled. Thus, the addition is accomplished by passing each pair of bits together with the previous carry through a single full-adder circuit and transferring the sum, one bit at a time, into register *A.*

If a new number has to be added to the contents of register *A,* this number must be first transferred serially into register *B.* Repeating the process once more will add the second number to the previous number in *A.*

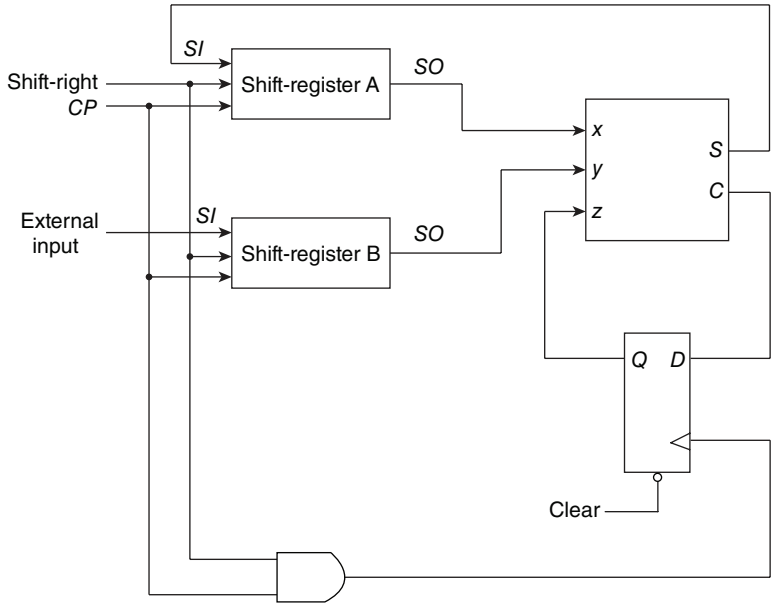


Figure 7-10 Serial adder

Comparing the serial adder with the parallel adder described in Section 5-2, we note the following differences. The parallel adder must use registers with parallel-load capability, whereas the serial adder uses shift registers. The number of full-adder circuits in the parallel adder is equal to the number of bits in the binary numbers, whereas the serial adder requires only one full-adder circuit and a carry flip-flop. Excluding the registers, the parallel adder is a purely combinational circuit, whereas the serial adder is a sequential circuit. The sequential circuit in the serial adder consists of a full-adder circuit and a flip-flop that stores the output carry. This is typical in serial operations because the result of a bit-time operation may depend not only on the present inputs but also on previous inputs.

To show that bit-time operations in serial computers may require a sequential circuit, we will redesign the serial adder by considering it a sequential circuit.

EXAMPLE 7-3: Design a serial adder using a sequential-logic procedure.

First, we must stipulate that two shift registers are available to store the binary numbers to be added serially. The serial outputs from the registers are designated by variables x and y. The sequential circuit to be designed will not include the shift registers; they will be inserted later to show the complete unit. The sequential circuit proper has two inputs, x and y, that provide a pair of significant bits, an output S that generates the sum bit, and flip-flop Q for storing the carry. The present state of Q provides the present value of the carry. The clock pulse that shifts the registers enables flip-flop Q to load the next carry. This carry is then used with the next pair of bits in x and y. The state table that specifies the sequential circuit is given in Table 7-3.

Table 7-3 Excitation table for serial adder

Present state	Inputs		Next state	Output	Flip-flop inputs	
Q	*x*	*y*	*Q*	*S*	*JQ*	*KQ*
0	0	0	0	0	0	*X*
0	0	1	0	1	0	*X*
0	1	0	0	1	0	*X*
0	1	1	1	0	1	*X*
1	0	0	0	1	*X*	1
1	0	1	1	0	*X*	0
1	1	0	1	0	*X*	0
1	1	1	1	1	*X*	0

The present state of *Q* is the present value of the carry. The present carry in *Q* is added together with inputs *x* and *y* to produce the sum bit in output *S*. The next state of *Q* is equivalent to the output carry. Note that the state table entries are identical to the entries in a full-adder truth table, except that the input carry is now the present state of *Q* and the output carry is now the next state of *Q*.

If we use a *D* flip-flop for *Q,* we obtain the same circuit as in Fig. 7-10 because the input requirements of the *D* input are the same as the next state values. If we use a *JK* flip-flop for *Q,* we obtain the input excitation requirements listed m Table 7-3. The three Boolean functions of interest are the flip-flop input functions for *JQ* and *KQ* and output *S*. These functions are specified in the excitation table and can be simplified by means of maps:

$$JQ = xy$$
$$KQ = x'y' = (x + y)'$$
$$S = x \oplus y \oplus Q$$

As shown in Fig. 7-11, the circuit consists of three gates and a *JK* flip-flop. The two shift registers are also included in the diagram to show the complete serial adder. Note that output *S* is a function not only of *x* and *y* but also of the present state of *Q,* The next state of *Q* is a function of the present values of *x* and *y* that come out of the serial outputs of the shift registers.

7.4 RIPPLE COUNTERS

MSI counters come in two categories: ripple counters and synchronous counters. In a ripple counter, the flip-flop output transition serves as a source for triggering other flip-flops. In other words, the *CP* inputs of all flip-flops (except the first) are triggered not by the incoming pulses but rather by the transition that occurs in other flip-flops. In a synchronous counter, the input pulses are applied to all *CP* inputs of all flip-flops. The change of state of a particular flip-flop is dependent on the present state of other flip-flops. Synchronous MSI counters are discussed in the next section. Here we present some common MSI ripple counters and explain their operation.

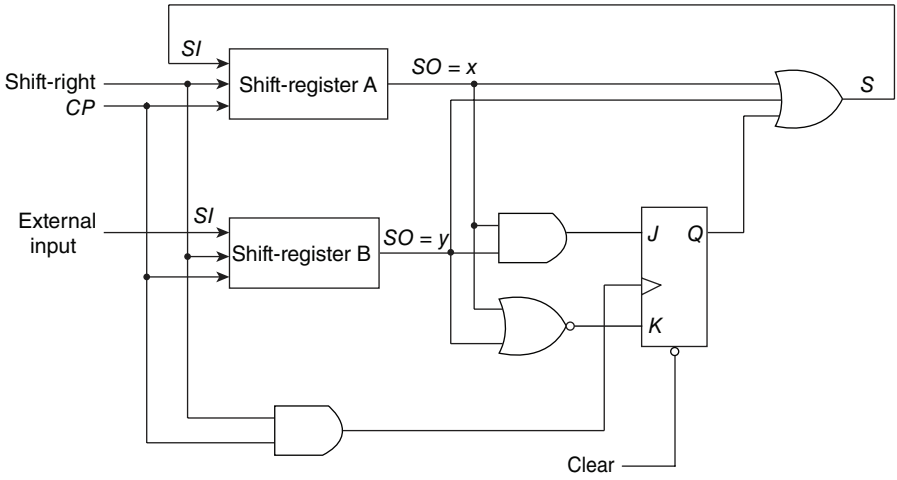


Figure 7-11 Second form of a serial adder

7.4.1 Binary Ripple Counter

A binary ripple counter consists of a series connection of complementing flip-flops (*T* or *JK* type), with the output of each flip-flop connected to the *CP* input of the next higher-order flip-flop. The flip-flop holding the least significant bit receives the incoming count pulses. The diagram of a 4-bit binary ripple counter is shown in Fig. 7-12. All *J* and *K* inputs are equal to 1. The small circle in the *CP* input indicates that the flip-flop complements during a negative-going transition or when the output to which it is connected goes from 1 to 0. To understand the operation of the binary counter, refer to its count sequence given in Table 7-4. It is obvious that the lowest-order bit A_1 must be complemented with each count pulse. Every time A_1 goes from 1 to 0, it complements A_2. Every time A_2 goes from 1 to 0, it complements A_3, and so on. For example, take the transition from count 0111 to 1000. The arrows in the table emphasize the transitions in this case, A_1 is complemented with the count pulse. Since A_1 goes from 1 to 0, it triggers A_2 and complements it. As a result, A_2 goes from 1 to 0, which in turn complements A_3, A_3 now goes from 1 to 0, which complements A_4. The output transition of A_4, if connected to a next stage, will not trigger the next flip-flop since it goes from 0 to 1. The flip-flops change one at a time in rapid succession, and the signal propagates through the counter in a *ripple* fashion. Ripple counters are sometimes called *asynchronous counters.*

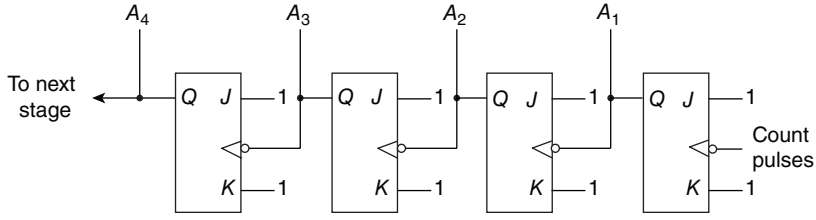


Figure 7-12 4-bit binary ripple counter

Table 7-4 Count sequence for a binary ripple counter

Count sequence				Conditions for complementing flip-flops	
A_4	A_3	A_2	A_1		
0	0	0	0	Complement A_1	
0	0	0	1	Complement A_1	A_1 will go from 1 to 0 and complement A_2
0	0	1	0	Complement A_1	
0	0	1	1	Complement A_1	A_1 will go from 1 to 0 and complement A_2;
					A_2 will go from 1 to 0 and complement A_3
0	1	0	0	Complement A_1	
0	1	0	1	Complement A_1	A_1 will go from 1 to 0 and complement A_2
0	1	1	0	Complement A_1	
0	1	1	1	Complement A_1	A_1 will go from 1 to 0 and complement A_2;
					A_2 will go from 1 to 0 and complement A_3;
					A_3 will go from 1 to 0 and complement A_4
1	0	0	0	and so on…	

A binary counter with a reverse count is called a binary *down-counter*. In a down-counter, the binary count is decremented by 1 with every input count pulse. The count of a 4-bit down-counter starts from binary 15 and continues to binary counts 14, 13, 12, ... 0 and then back to 15. The circuit of Fig. 7-12 will function as a binary down-counter if the outputs are taken from the complement terminals *Q'* of all flip-flops. If only the normal outputs of flip-flops are available, the circuit must be modified slightly as described below.

A list of the count sequence of a count-down binary counter shows that the lowest-order bit must be complemented with every count pulse. Any other bit in the sequence is complemented if its previous lower-order bit goes from 0 to 1. Therefore, the diagram of a binary down-counter looks the same as in Fig. 7-12, provided all flip-flops trigger on the positive edge of the pulse. (The small circles in the *CP* inputs must be absent.) If negative-edge-triggered flip-flops are used, then the *CP* input of each flip-flop must be connected to the *Q'* output of the previous flip-flop. Then when *Q* goes from 0 to 1. *Q'* will go from 1 to 0 and complement the next flip-flop as required.

7.4.2 BCD Ripple Counter

A decimal counter follows a sequence of ten states and returns to 0 after the count of 9. Such a counter must have at least four flip-flops to represent each decimal digit, since a decimal digit is represented by a binary code with at least four bits. The sequence of states in a decimal counter is dictated by the binary code used to represent a decimal digit. If BCD is used, the sequence of states is as shown in the state diagram of Fig. 7-13. This is similar to a binary counter, except that the state after 1001 (code for decimal digit 9) is 0000 (code for decimal digit 0).

The design of a decimal ripple counter or of any ripple counter not following the binary sequence is not a straightforward procedure. The formal tools of logic design can serve only as a guide. A satisfactory end product requires the ingenuity and imagination of the designer.

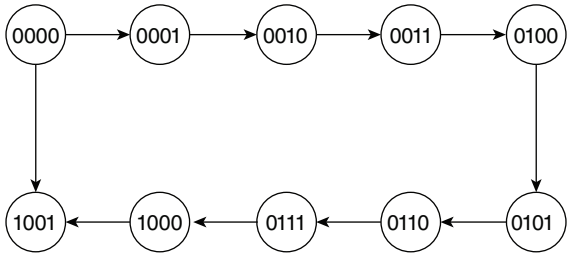


Figure 7-13 State diagram of a decimal BCD counter

The logic diagram of a BCD ripple counter is shown in Fig, 7-14.‡ The four outputs are designated by the letter symbol Q with a numeric subscript equal to the binary weight of the corresponding bit in the BCD code. The flip-flops trigger on the negative edge, i.e., when the *CP* signal goes from 1 to 0. Note that the output of Q_1, is applied to the *CP* inputs of both Q_2 and Q_8 and the output of Q_2 is applied to the *CP* input of Q_4. The *J* and *K* inputs are connected either to a permanent 1 signal or to outputs of flip-flops, as shown in the diagram.

A ripple counter is an asynchronous sequential circuit and cannot be described by Boolean equations developed for describing clocked sequential circuits. Signals that affect the flip-flop transition depend on the order in which they change from 1 to 0. The operation of the counter can be explained by a list of conditions for flip-flop transitions. These conditions are derived from the logic diagram and from knowledge of how a *JK* flip-flop operates. Remember that when the *CP* input goes from 1 to 0, the flip-flop is set if $J = 1$, is cleared if $K = 1$, is complemented if $J = K = 1$. and is left unchanged if $J = K = 0$. The following are the conditions for each flip-flop state transition:

1. Q_1 is complemented on the negative edge of every count pulse.
2. Q_2 is complemented if $Q_8 = 0$ and Q_1 goes from 1 to 0. Q_2 is cleared if $Q_8 = 1$ and Q_1 goes from 1 to 0.

‡This circuit is similar to IC type 7490.

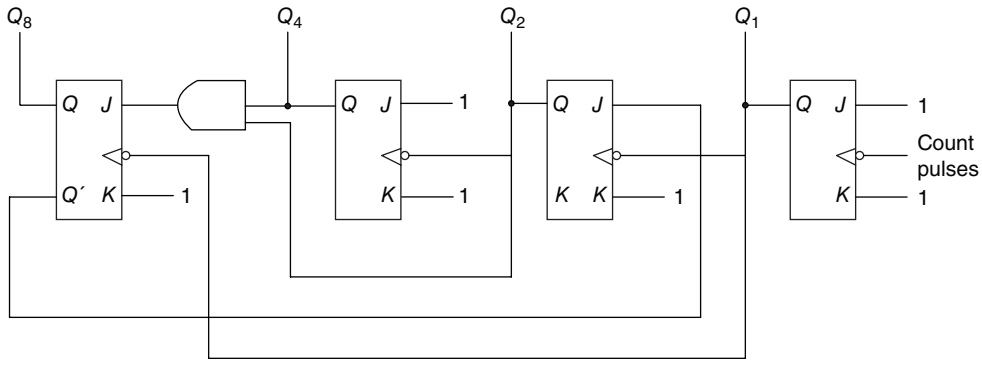


Figure 7-14 Logic diagram of a BCD ripple counter

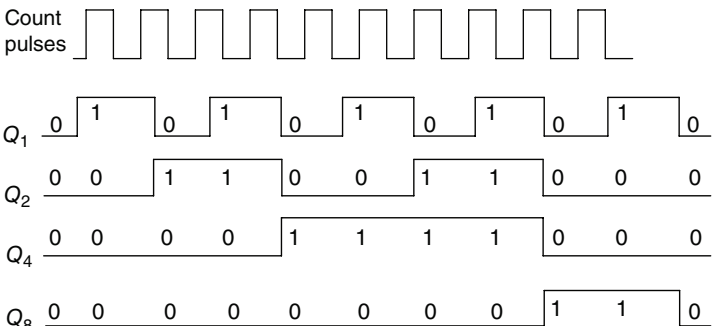


Figure 7-15 Timing diagram for the decimal counter of Fig. 7-14

3. Q_4 is complemented when Q_2 goes from 1 to 0.
4. Q_8 is complemented when $Q_4Q_2 = 11$ and Q_1 goes from 1 to 0. Q_8 is cleared if either Q_4 or Q_2 is 0 and Q_1 goes from 1 to 0.

To verify that these conditions result in the sequence required by a BCD ripple counter, it is necessary to verify that the flip-flop transitions indeed follow a sequence of states as specified by the state diagram of Fig. 7-13. Another way to verify the operation of the counter is to derive the timing diagram for each flip-flop from the conditions listed above. This diagram is shown in Fig. 7-15 with the binary states listed after each clock pulse. Q_1 changes state after each clock pulse. Q_2 complements every time Q_1 goes from 1 to 0 as long as $Q_8 = 0$. When Q_8 becomes 1, Q_2 remains cleared at 0. Q_4 complements every time Q_2 goes from 1 to 0. Q_8 remains cleared as long as Q_2 or Q_1 is 0. When both Q_2 and Q_4 become 1's, Q_8 complements when Q_1 goes from 1 to 0. Q_8 is cleared on the next transition of Q_1.

The BCD counter of Fig. 7-14 is a *decade* counter, since it counts from 0 to 9. To count in decimal from 0 to 99, we need a two-decade counter. To count from 0 to 999, we need a three-decade counter. Multiple-decade counters can be constructed by connecting BCD counters in cascade, one for each decade. A three-decade counter is shown in Fig. 7-16. The inputs to the second and third decades come from Q_8 of the previous decade. When Q_8 in one decade goes from 1 to 0, it triggers the count for the next higher-order decade while its own decade goes from 9 to 0. For instance, the count after 399 will be 400.

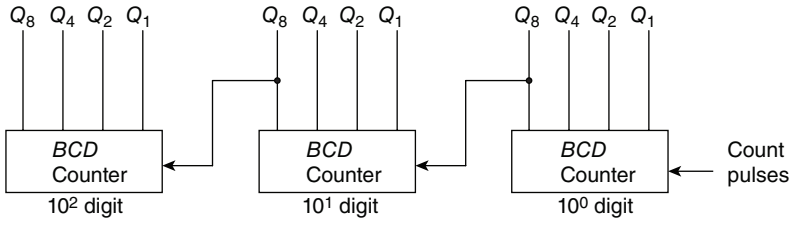


Figure 7-16 Block diagram of a 3-decade decimal BCD counter

7.5 Synchronous-counters

Synchronous counters are distinguished from ripple counters in that clock pulses are applied to the *CP* inputs of *all* flip-flops. The common pulse triggers all the flip-flops simultaneously, rather than one at a lime in succession as in a ripple counter. The decision whether a flip-flop is to be complemented or not is determined from the values of the *J* and *K* inputs at the time of the pulse. If $J = K = 0$, the flip-flop remains unchanged. If $J = K = 1$, the flip-flop complements.

A design procedure for any type of synchronous counter was presented in Section 6-8. The design of a 3-bit binary counter was carried out in detail and is illustrated in Fig. 6-30. In this section, we present some typical MSI synchronous counters and explain their operation. It must be realized that there is no need to design a counter if it is already available commercially in IC form.

7.5.1 Binary Counter

The design of synchronous binary counters is so simple that there is no need to go through a rigorous sequential-logic design process. In a synchronous binary counter, the flip-flop in the lowest-order position is complemented with every pulse.

This means that its *J* and *K* inputs must be maintained at logic-1. A flip-flop in any other position is complemented with a pulse provided all the bits in the lower-order positions are equal to 1, because the lower-order bits (when all l's) will change to 0's on the next count pulse. The binary count dictates that the next higher-order bit be complemented. For example, if the present state of a 4-bit counter is $A_4A_3A_2A_1 = 0011$, the next count will be 0100. A_1 is always complemented. A_1 is complemented because the present state of $A_1 = 1$. A_3 is complemented because the present state of $A_2A_1 = 11$. But A_4 is not complemented because the present state of $A_3A_2A_1 = 011$, which does not give an all-1's condition.

Synchronous binary counters have a regular pattern and can easily be constructed with complementing flip-flops and gates. The regular pattern can be clearly seen from the 4-bit counter depicted in Fig. 7-17. The *CP* terminals of all flip-flops are connected to a common clock-pulse source. The first stage A_1 has its *J* and *K* equal to 1 if the counter is enabled. The other *J* and *K* inputs are equal to 1 if all previous low-order bits are equal to 1 and the count is enabled. The chain of AND gates generates the required logic for the *J* and *K* inputs in each stage. The counter can be extended to any number of stages, with each stage having an additional flip-flop and an AND gate that gives an output of 1 if all previous flip-flop outputs are 1's.

Note that the flip-flops trigger on the negative edge of the pulse. This is not essential here as it was with the ripple counter. The counter could also be triggered on the positive edge of the pulse.

7.5.2 Binary Up-Down Counter

In a synchronous count-down binary counter, the flip-flop in the lowest-order position is complemented with every pulse. A flip-flop in any other position is complemented with a pulse provided all the lower-order bits are equal to 0. For example, if the present state of a 4-bit count-down binary counter is $A_4A_3A_2A_1 = 1100$, the next count will be 1011, A_1 is always complemented. A_2 is complemented because the present state of $A_1 = 0$. A_3 is complemented because the present state of $A_2A_1 = 00$. But A_4 is not complemented because the present state of $A_3A_2A_1 = 100$, which is not an all-0's condition.

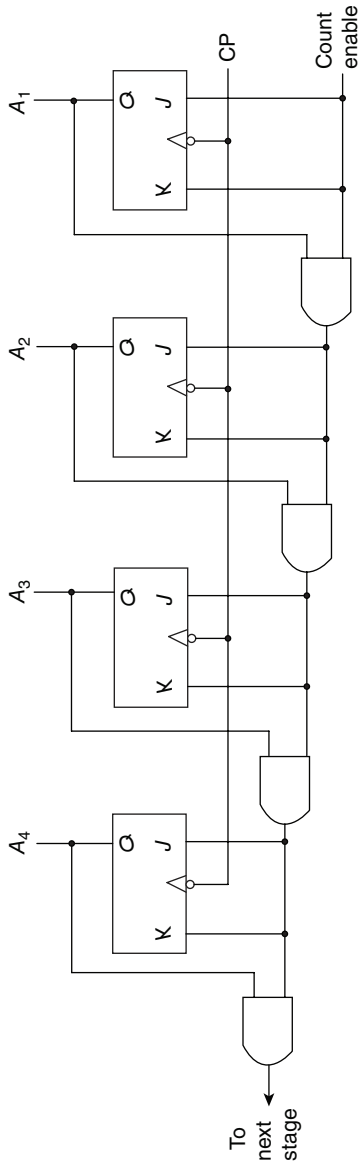


Figure 7-17 4-bit synchronous binary counter

A count-down binary counter can be constructed as shown in Fig. 7-17, except that the inputs to the AND gates must come from the complement outputs Q' and not from the normal outputs Q of the previous flip-flops. The two operations can be combined in one circuit. A binary counter capable of counting either up or down is shown in Fig. 7-18. The T flip-flops employed in this circuit may be considered as JK flip-flops with the J and K terminals tied together. When the *up* input control is 1, the circuit counts up, since the T inputs are determined from the previous values of the normal outputs in Q. When the *down* input control is 1, the circuit counts down, since the complement outputs Q' determine the states of the T inputs. When both the *up* and *down* signals are 0's, the register does not change state but remains in the same count.

7.5.3 BCD Counter

A BCD counter counts in binary-coded decimal from 0000 to 1001 and back to 0000. Because of the return to 0 after a count of 9, a BCD counter does not have a regular pattern as in a straight binary count. To derive the circuit of a BCD synchronous counter, it is necessary to go through a design procedure as discussed in Section 6-8.

The count sequence of a BCD counter is given in Table 7-5. The excitation for the T flip-flops is obtained from the count sequence. An output y is also shown in the table. This output is equal to 1 when the counter present state is 1001. In this way, y can enable the count of the next-higher-order decade while the same pulse switches the present decade from 1001 to 0000.

The flip-flop input functions from the excitation table can be simplified by means of maps. The unused states for minterms 10 to 15 are taken as don't-care terms. The simplified functions are listed below:

$$
\begin{aligned}
TQ_1 &= 1 \\
TQ_2 &= Q_8Q_1 \\
TQ_4 &= Q_2Q_1 \\
TQ_8 &= Q_8Q_1 + Q_4Q_2Q_1 \\
y &= Q_8Q_1
\end{aligned}
$$

The circuit can he easily drawn with four T flip-flops, five AND gates, and one OR gate.

Synchronous BCD counters can be cascaded to form a counter for decimal numbers of any length. The cascading is done as in Fig. 7-16, except that output y must be connected to the count input of the next-higher-order decade.

7.5.4 Binary Counter with Parallel Load

Counters employed in digital systems quite often require a parallel-load capability for transferring an initial binary number prior to the count operation. Figure 7-19 shows the logic diagram of a register that has a parallel-load capability and can also operate as a counter.[§] The input load control, when equal to 1, disables the count sequence and causes a transfer of data from inputs I_1 through I_4 into flip-flops A_1 through A_4, respectively. If the load input is 0 and the count input control is 1, the circuit operates as a counter. The clock pulses then cause the state of the flip-flops to change according to the binary count sequence. If both control inputs are 0, clock pulses do not change the state of the register.

[§]This is similar but not identical to IC type 74161.

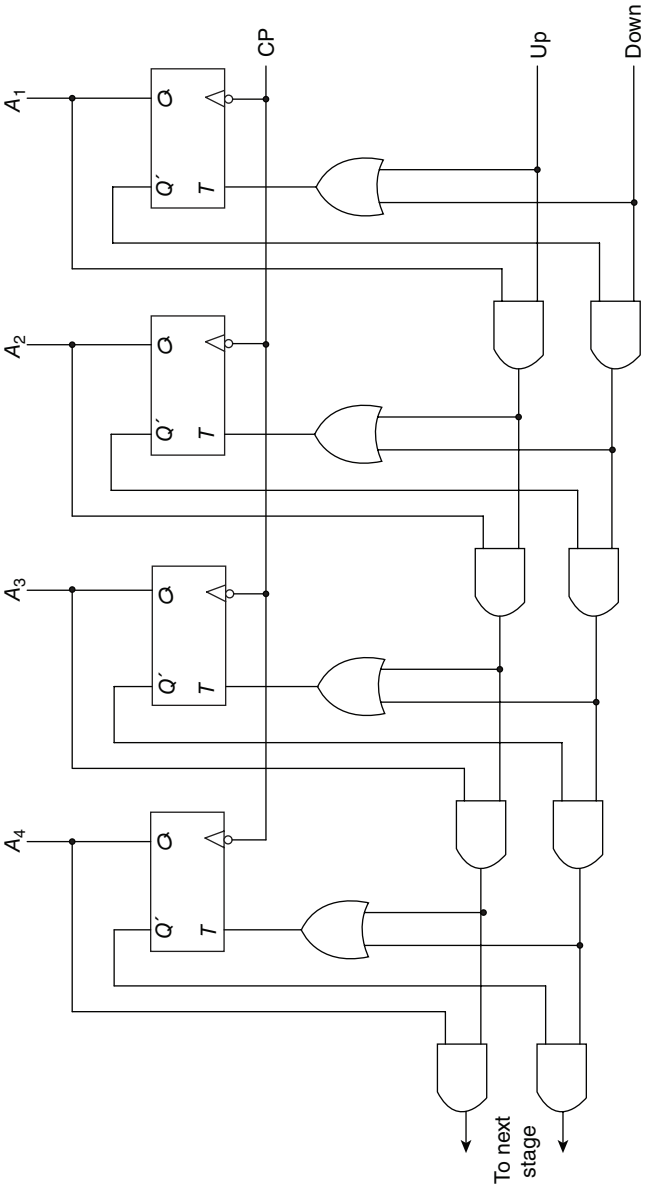


Figure 7-18 4-bit up-down binary counter

Table 7-5 Excitation table for a BCD counter

Count sequence				Flip-flop inputs				Output carry
Q_8	Q_4	Q_2	Q_1	TQ_8	TQ_4	TQ_2	TQ_1	y
0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	1	1	0
0	0	1	0	0	0	0	1	0
0	0	1	1	0	1	1	1	0
0	1	0	0	0	0	0	1	0
0	1	0	1	0	0	1	1	0
0	1	1	0	0	0	0	1	0
0	1	1	1	1	1	1	1	0
1	0	0	0	0	0	0	1	0
1	0	0	1	1	0	0	1	1

The carry-out terminal becomes a 1 if all flip-flops are equal to 1 while the count input is enabled. This is the condition for complementing the flip-flop holding the next-higher-order bit. This output is useful for expanding the counter to more than four bits. The speed of the counter is increased if this carry is generated directly from the outputs of all four flip-flops instead of going through a chain of AND gates. Similarly, each flip-flop is associated with an AND gate that receives all previous flip-flop outputs directly to determine when the flip-flop should he complemented.

The operation of the counter is summarized in Table 7-6. The four control inputs: clear, *CP,* load, and count determine the next output state. The clear input is asynchronous and, when equal to 0, causes the counter to be cleared to all 0's, regardless of the presence of clock pulse's or other inputs. This is indicated in the table by the *X* entries, which symbolize don't-care conditions for the other inputs, so their value can be either 0 or 1. The clear input must go to the 1 state for the clocked operations listed in the next three entries in the table. With the load and count inputs both at 0, the outputs do not change, whether a pulse is applied in the *CP* terminal or not. A load input of 1 causes a transfer from inputs $I_1 - I_4$ into the register during the positive edge of an input pulse. The input information is loaded into the register regardless of the value of the count input, because the count input is inhibited when the load input is 1. If the load input is maintained at 0, the count input controls the operation of the counter. The outputs change to the next binary count on the positive-edge transition of every clock pulse, but no change of state occurs if the count input is 0.

The 4-bit counter shown in Fig. 7-19 can be enclosed in one IC package. Two ICs are necessary for the construction of an 8-bit counter; four ICs for a 16-bit counter; and so on. The carry output of one IC must be connected to the count input of the IC holding the four next-higher-order bits of the counter.

Counters with parallel-load capability having a specified number of bits are very useful in the design of digital systems. Later we will refer to them as registers with load and increment

Table 7-6 Function table for the counter of Fig. 7-19

Clear	*CP*	Load	Count	Function
0	*X*	*X*	*X*	Clear to 0
1	*X*	0	0	No change
1	↑	1	*X*	Load inputs
1	↑	0	1	Count next binary state

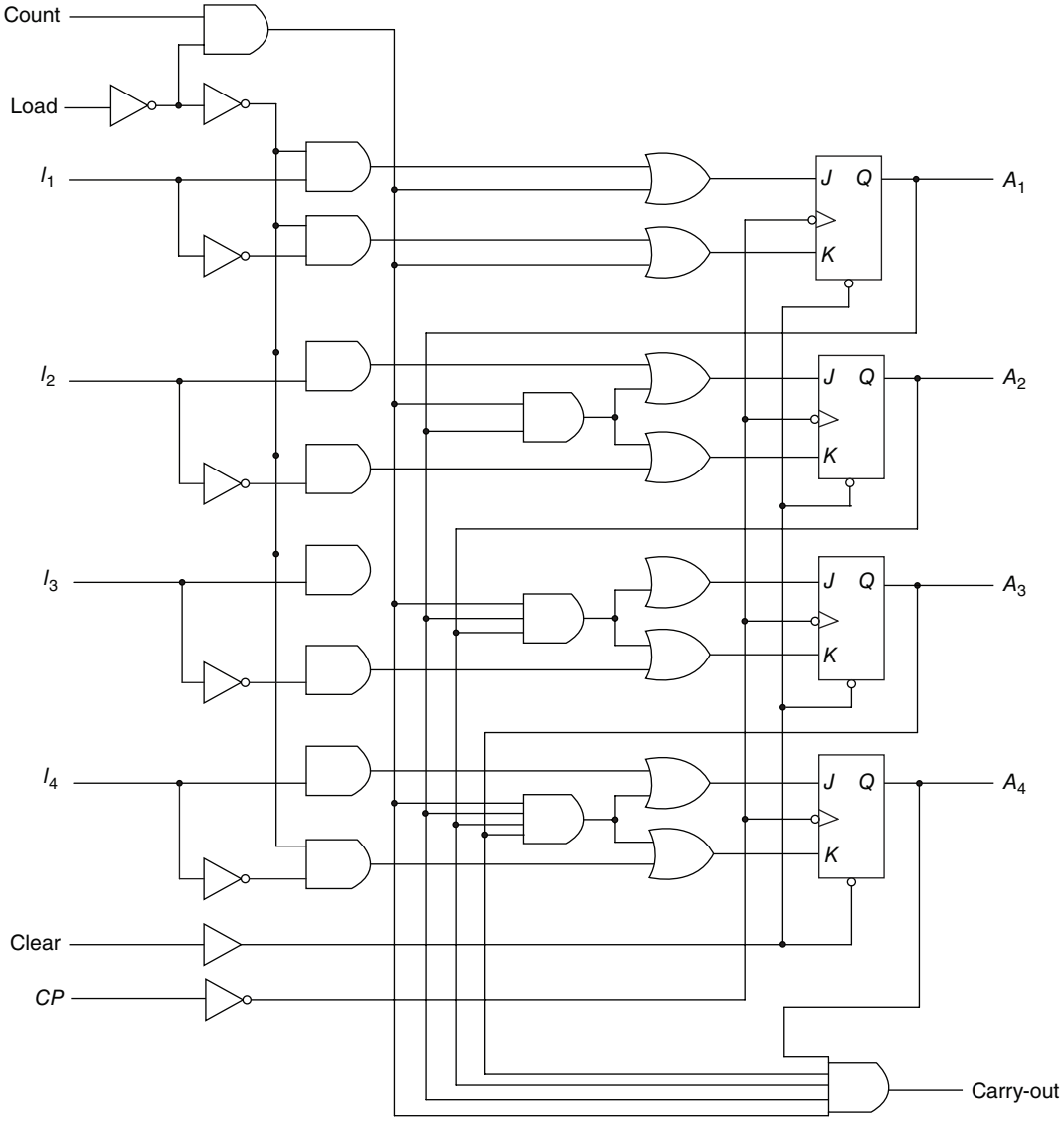


Figure 7-19 4-bit binary counter with parallel load

capabilities. The *increment* function is an operation that adds 1 to the present content of a register. By enabling the count control during one clock pulse period, the content of the register can be incremented by 1.

A counter with parallel load can be used to generate any desired number of count sequences. A modulo- N (abbreviated mod N) counter is a counter that goes through a repeated sequence of N counts. For example, a 4-bit binary counter is a mod-16 counter. A BCD counter is a mod-10 counter. In some applications, one may not be concerned with the particular N states that a mod-N counter uses. If this is the case, then a counter with parallel load can be used to construct any *mod-N* counter, with N being any value desired. This is shown in the following example.

EXAMPLE 7-4: Construct a mod-6 counter using the MSI circuit specified in Fig. 7-19.

Figure 7-20 shows four ways in which a counter with parallel load can be used to generate a sequence of six counts. In each case the count control is set to 1 to enable the count through the pulses in the *CP* input. We also use the facts that the load control inhibits the count and that the clear operation is independent of other control inputs.

The AND gate in Fig. 7-20(a) detects the occurrence of state 0101 in the output. When the counter is in this state, the load input is enabled and an all-0's input is loaded into the register. Thus, the counter goes through binary states 0, 1, 2, 3, 4, and 5 and then returns to 0. This produces a sequence of six counts.

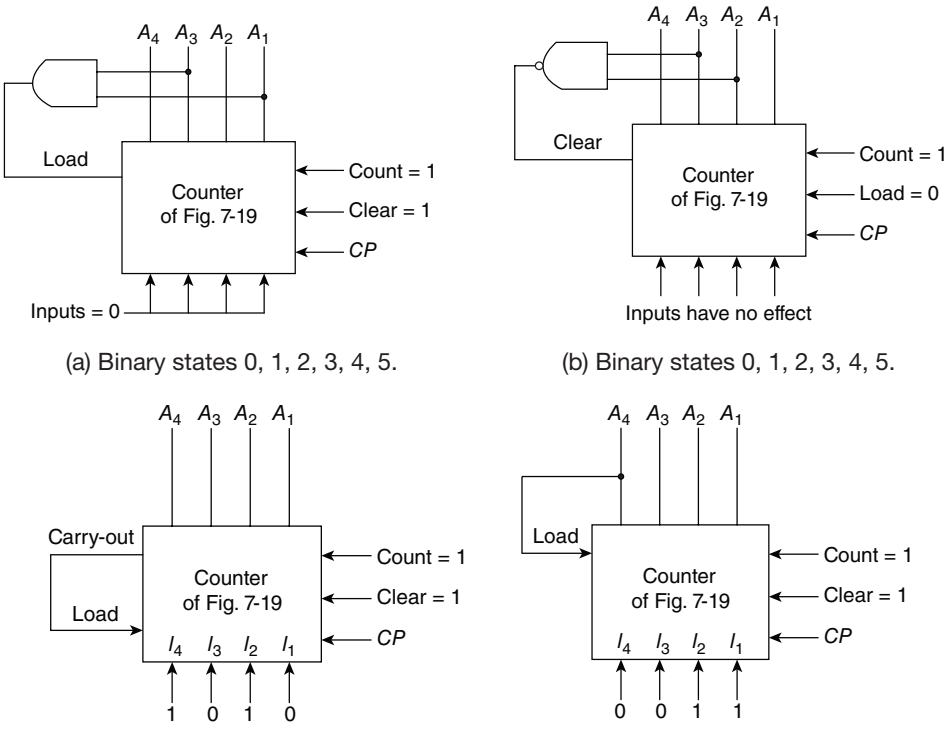
The clear input of the register is asynchronous, i.e., it does not depend on the clock. In Fig. 7-20(b), the NAND gate detects the count of 0110, but as soon as this count occurs, the register is cleared. The count 0110 has no chance of staying on for any appreciable time because the register goes immediately to 0. A momentary spike occurs in output A_2 as the count goes from 0101 to 0110 and immediately to 0000. This momentary spike may be undesirable and for this reason this configuration is not recommended. If the counter has a synchronous clear input, it would be possible to clear the counter with the clock after an occurrence of the 0101 count.

Instead of using the first six counts, we may want to choose the last six counts from 10 to 15. In this case it is possible to take advantage of the output carry to load a number in the register. In Fig. 7-20(c), the counter starts with count 1010 and continues to 1111. The output carry generated during the last state enables the load control, which then loads the input which is set at 1010.

It is also possible to choose any intermediate count of six states. The mod-6 counter of Fig. 7-20(d) goes through the count sequence 3, 4, 5, 6, 7, and 8. When the last count 1000 is reached, output A_4 goes to 1 and the load control is enabled. This loads into the register the value of 0011, and the binary count continues from this state.

7.6 Timing Sequences

The sequence of operations in a digital system are specified by a control unit. The control unit that supervises the operations in a digital system would normally consist of timing signals that determine the time sequence in which the operations are executed. The timing sequences in the control unit can be easily generated by means of counters or shift registers. This section demonstrates the use of these MSI functions in the generation of timing signals for a control unit.



(a) Binary states 0, 1, 2, 3, 4, 5.

(b) Binary states 0, 1, 2, 3, 4, 5.

(c) Binary states 10, 11, 12, 13, 14, 15.

(d) Binary states 3, 4, 5, 6, 7, 8.

Figure 7-20 Four ways to achieve a mod-6 counter using a counter with parallel load

7.6.1 Word-time Generation

First, we demonstrate a circuit that generates the required timing signal for serial mode of operation. Serial transfer of information was discussed in Section 7-3, with an example depicted in Fig. 7-8. The control unit in a serial computer must generate a *word-time* signal that stays on for a number of pulses equal to the number of bits in the shift registers. The word-time signal can be generated by means of a counter that counts the required number of pulses.

Assume that the word-time signal to be generated must stay on for a period of eight clock pulses. Figure 7-21(a) shows a counter circuit that accomplishes this task. Initially, the 3-bit counter is cleared to 0. A start signal will set flip-flop Q. The output of this flip-flop supplies the word-time control and also enables the counter. After the count of eight pulses, the flip-flop is reset and Q goes to 0. The timing diagram of Fig. 7-21(b) demonstrates the operation of the circuit. The start signal is synchronized with the clock and stays on for one clock pulse period. After Q is set to 1, the counter starts counting the clock pulses. When the counter reaches the count of 7 (binary 111), it sends a stop signal to the reset input of the flip-flop. The stop signal becomes a 1 after the negative-edge transition of pulse 7. The next clock pulse switches the counter to the 000 state and also clears Q. Now the counter is disabled and the word-time signal stays at 0. Note that the word-time control stays on for a period of eight pulses. Note also that the stop signal in this

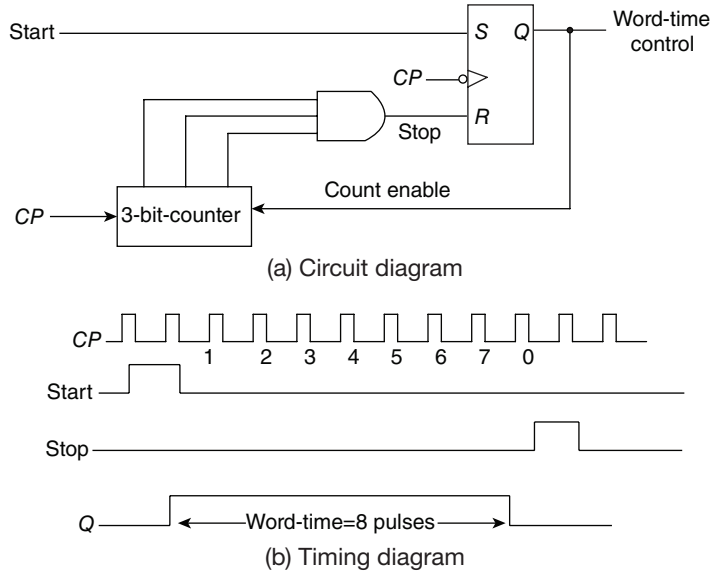


Figure 7-21 Generation of a word-time control for serial operations

circuit can be used to start another word-count control in another circuit just as the start signal is used in this circuit.

7.6.2 Timing Signals

In a parallel mode of operation, a single clock pulse can specify the time at which an operation should be executed. The control unit in a digital system that operates in the parallel mode must generate timing signals that stay on for only one clock pulse period, but these timing signals must be distinguished from each other.

Timing signals that control the sequence of operations in a digital system can be generated with a shift register or a counter with a decoder. A *ring counter* is a circular shift register with only one flip-flop being set at any particular lime; all others are cleared. The single bit is shifted from one flip-flop lo the other to produce the sequence of timing signals. Figure 7-22(a) shows a 4-bit shift register connected as a ring counter. The initial value of the register is 1000, which produces the variable T_0. The single bit is shifted right with every clock pulse and circulates back from T_3 to T_0. Each flip-flop is in the 1 state once every four clock pulses and produces one of the four timing signals shown in Fig. 7-22(c). Each output becomes a 1 after the negative-edge transition of a clock pulse and remains 1 during the next clock pulse.

The liming signals can be generated also by continuously enabling a 2-bit counter that goes through four distinct states. The decoder shown in Fig. 7-22(b) decodes the four states of the counter and generates the required sequence of timing signals.

The liming signals, when enabled by the clock pulses, will provide multiple-phase clock pulses. For example, if T_0 is ANDed with *CP,* the output of the AND gate will generate clock pulses at one-fourth the frequency of the master-clock pulses. Multiple-phase clock pulses can be used for controlling different registers with different time scales.

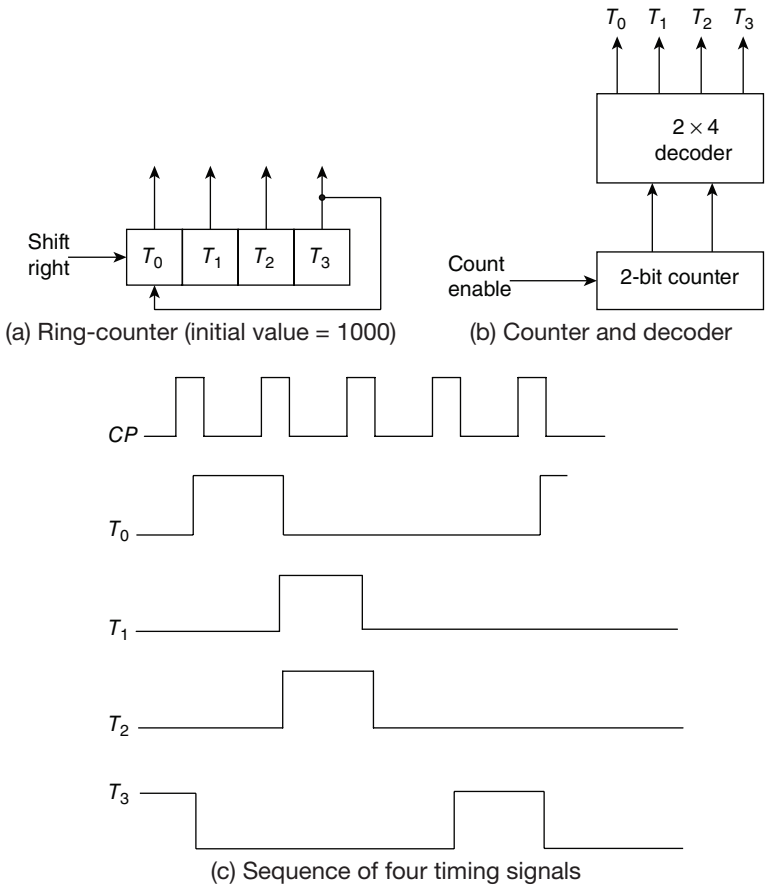
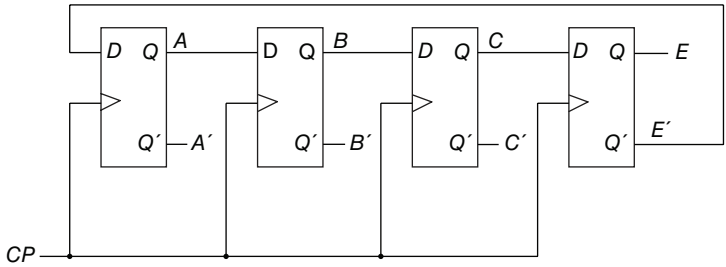


Figure 7-22 Generation of timing signals

To generate 2^n timing signals, we need either a shift register with 2^n flip-flops or an n-bit counter together with an n-*to*-2^n line decoder. For example, 16 timing signals can be generated with a 16-bit shift register connected as a ring counter or with a 4-bit counter and a 4-to-16 line decoder, in the first case, we need 16 flip-flops. In the second case, we need four flip-flops and 16 4-input AND gates for the decoder. It is also possible to generate the timing signals with a combination of a shift register and a decoder. In this way, the number of flip-flops is less than in a ring counter, and the decoder requires only 2-input gates. This combination is sometimes called a *Johnson counter.*

7.6.3 Johnson Counter

A k-bit ring counter circulates a single bit among the flip-flops to provide k distinguishable states. The number of states can be doubled if the shift register is connected as a *switch-tail* ring counter. A switch-tail ring counter is a circular shift register with the complement output of the last flip-flop connected to the input of the first flip-flop. Figure 7-23(a) shows such a shift register. The



(a) 4-stage switch-tail ring counter

Sequence number	Flip-flop outputs A	B	C	D	AND gate required for output
1	0	0	0	0	$A'E'$
2	1	0	0	0	AB'
3	1	1	0	0	BC'
4	1	1	1	0	CE'
5	1	1	1	1	AE
6	0	1	1	1	$A'B$
7	0	0	1	1	$B'C$
8	0	0	0	1	$C'E$

(b) Count sequence and required decoding

Figure 7-23 Construction of a Johnson counter

circular connection is made from the complement output of the rightmost flip-flop to the input of the leftmost flip-flop. The register shifts us contents once to the right with every clock pulse, and at the same time, the complement value of the *E* flip-flop is transferred into the *A* flip-flop. Starting from a cleared state, the switch-tail ring counter goes through a sequence of eight states as listed in Fig. 7-23(b). In general, a *k*-bit switch-tail ring counter will go through a sequence of 2*k* states. Starting from all 0's, each shift operation inserts 1's from the left until the register is filled with all 1's. In the following sequences, 0's are inserted from the left until the register is again filled with all 0's.

A Johnson counter is a *k*-bit switch-tail ring counter with 2*k* decoding gates to provide outputs for 2*k* timing signals. The decoding gates are not shown in Fig. 7-23 but are specified in the last column of the table. The eight AND gates listed in the table, when connected to the circuit, will complete the construction of the Johnson counter. Since each gate is enabled during one particular state sequence, the outputs of the gates generate eight timing sequences in succession.

The decoding of a *k*-bit switch-tail ring counter to obtain 2*k* timing sequences follows a regular pattern. The all-0's state is decoded by taking the complement of the two extreme flip-flop outputs. The all-l's state is decoded by taking the normal outputs of the two extreme flip-flops. All other states are decoded from an adjacent 1, 0 or 0, 1 pattern in the sequence. For example, sequence 7 has an adjacent 0, 1 pattern in flip-flops *B* and *C*. The decoded output is then obtained by taking the complement of *B* and the normal output of *C*, or *B'C*.

One disadvantage of the circuit in Fig. 7-23(a) is that, if it finds itself in an unused state, it will persist in moving from one invalid state to another and never find its way to a valid state. This difficulty can be corrected by modifying the circuit to avoid this undesirable condition. One

correcting procedure is to disconnect the output from flip-flop *B* that goes to the *D* input of flip-flop C, and instead enable the input of flip-flop *C* by the function:¶

$$DC = (A + C)B$$

where *DC* is the flip-flop input function for the *D* input of flip-flop *C*.

Johnson counters can be constructed for any number of timing sequences. The number of flip-flops needed is one-half the number of timing signals. The number of decoding gates is equal to the number of timing signals and only 2-input gates are employed.

7.7 The Memory Unit

The registers in a digital computer may be classified as either operational or storage type. An *operational* register is capable of storing binary information in its flip-flops and, in addition, has combinational gates capable of data-processing tasks. A *storage* register is used solely for temporary storage of binary information. This information cannot be altered when transferred in and out of the register. A *memory unit* is a collection of storage registers together with the associated circuits needed to transfer information in and out of the registers. The storage registers in a memory unit are called *memory registers.*

The bulk of the registers in a digital computer are memory registers, to which information is transferred for storage and from which information is available when needed for processing. Comparatively few operational registers are found in the processor unit. When data processing takes place, the information from selected registers in the memory unit is first transferred to the operational registers in the processor unit. Intermediate and final results obtained in the operational registers are transferred back to selected memory registers. Similarly, binary information received from input devices is first stored in memory registers; information transferred to output devices is taken from registers in the memory unit.

The component that forms the binary cells of registers in a memory unit must have certain basic properties, the most important of which are: (1). It must have a reliable two-state property for binary representation. (2) It must be small in size. (3) The cost per bit of storage should be as low as possible. (4) The time of access to a memory register should be reasonably fast. Examples of memory unit components are magnetic cores, semiconductor ICs, and magnetic surfaces on tapes, drums, or disks.

A memory unit stores binary information in groups called *words,* each word being stored in a memory register. A word in memory is an entity of *n* bits that moves in and out of storage as a unit. A memory word may represent an operand, an instruction, a group of alphanumeric characters, or any binary-coded information. The communication between a memory unit and its environment is achieved through two control signals and two external registers. The control signals specify the direction of transfer required, that is, whether a word is to be stored in a memory register or whether a word previously stored is to be transferred out of a memory register. One external register specifies the particular memory register chosen out of the thousands available; the other specifies the particular bit configuration of the word in question. The control signals and the registers are shown in the block diagram of Fig. 7-24.

The memory *address register* specifies the memory word selected. Each word in memory is assigned a number identification starting from 0 up to the maximum number of words available.

¶This is the way it is done in IC type 4022.

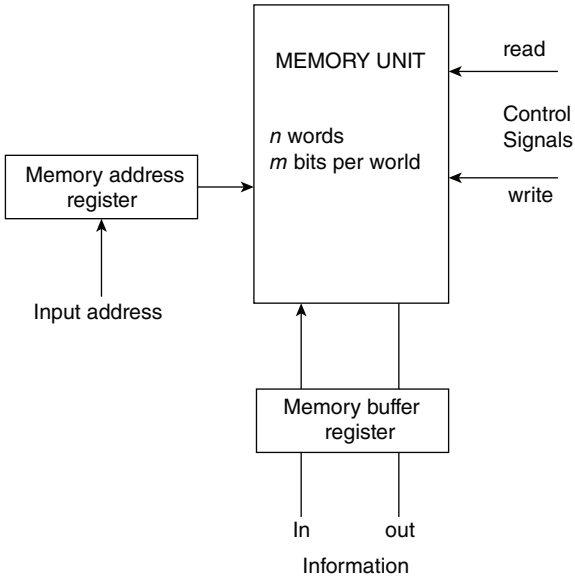


Figure 7-24 Block diagram of a memory unit showing communication with environment

To communicate with a specific memory word its location number, or *address,* is transferred to the address register. The internal circuits of the memory unit accept this address from the register and open the paths needed to select the word called. An address register with n bits can specify up to 2^n memory words. Computer memory units can range from 1024 words, requiring an address register of 10 bits, to $1{,}048{,}576 = 2^{20}$ words, requiring a 20-bit address register.

The two control signals applied to the memory unit are called *read* and *write.* A write signal specifies a transfer-in function; a read signal specifies a transfer-out function. Each is referenced from the memory unit. Upon accepting one of the control signals, the internal control circuits inside the memory unit provide the desired function. Certain types of storage units, because of their component characteristics, destroy the information stored in a cell when the bit in that cell is read out. Such a unit is said to be a destructive read-out memory, as opposed to a nondestructive memory where the information remains in the cell after it is read out. In either case, the old information is always destroyed when new information is written. The sequence of internal control in a destructive read-out memory must provide control signals that will cause the word to be restored into its binary cells if the application calls for a nondestructive function.

The information transfer to and from registers in memory and the external environment is communicated through one common register called the memory *buffer register* (other names are *information register* and *storage register*). When the memory unit receives a *write* control signal, the internal control interprets the contents of the buffer register to be the bit configuration of the word to be stored in a memory register. With a *read* control signal, the internal control sends the word from a memory register into the buffer register. In each case the contents of the address register specify the particular memory register referenced for writing or reading.

Let us summarize the information transfer characteristics of a memory unit by an example. Consider a memory unit of 1024 words with eight bits per word. To specify 1024 words, we need