|CSE 308: Digital System Design|Homework 2|

Reading

Mano's Chapter 5, 6, and 7. Skim and pay greater attention to sections 5-7, 6-4, 6-7, 6-8, 7-7, and 7-8. Attached herewith are the scanned copies of Chapters.

minterms are associated with A' and the second half with A. By circling the minterms of the function and applying the rules for finding values for the multiplexer inputs, we obtain the implementation shown. ■

Let us now compare the multiplexer method with the decoder method for implementing combinational circuits. The decoder method requires an OR gate for each output function, but only one decoder is needed to generate all minterms. The multiplexer method uses smaller-size units but requires one multiplexer for each output function. It would seem reasonable to assume that combinational circuits with a small number of outputs should be implemented with multiplexers. Combinational circuits with many output functions would probably use fewer ICs with the decoder method.

Although multiplexers and decoders may be used in the implementation of combinational circuits, it must be realized that decoders are mostly used for decoding binary information and multiplexers are mostly used to form a selected path between multiple sources and a single destination.

5-7 READ-ONLY MEMORY (ROM)

We saw in Section 5-5 that a decoder generates the 2^n minterms of the n input variables. By inserting OR gates to sum the minterms of Boolean functions, we were able to generate any desired combinational circuit. A read-only memory (ROM) is a device that includes both the decoder and the OR gates within a single IC package. The connections between the outputs of the decoder and the inputs of the OR gates can be specified for each particular configuration. The ROM is used to implement complex combinational circuits within one IC package or as permanent storage for binary information.

A ROM is essentially a memory (or storage) device in which permanent binary information is stored. The binary information must be specified by the designer and is then embedded in the unit to form the required interconnection pattern. ROMs come with special internal electronic fuses that can be "programmed" for a specific configuration. Once the pattern is established, it stays within the unit even when power is turned off and on again.

A block diagram of a ROM is shown in Fig. 5-21. It consists of n input lines and m output lines. Each bit combination of the input variables is called an *address*. Each bit combination that comes out of the output lines is called a *word*. The number of bits per word is equal to the number of output lines, m. An address is essentially a binary number that denotes one of the minterms of n variables. The number of distinct addresses possible with n input variables is 2^n. An output word can be selected by a unique address, and since there are 2^n distinct addresses in a ROM, there are 2^n distinct words that are said to be stored in the unit. The word available on the output lines at any given time depends on the address value applied to the input lines. A ROM is charac-

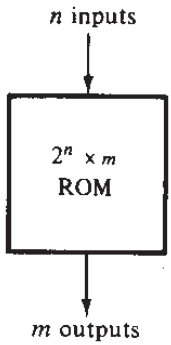


FIGURE 5-21
ROM block diagram

terized by the number of words 2^n and the number of bits per word m. This terminology is used because of the similarity between the read-only memory and the random-access memory, which is presented in Section 7-7.

Consider a 32 × 8 ROM. The unit consists of 32 words of 8 bits each. This means that there are eight output lines and that there are 32 distinct words stored in the unit, each of which may be applied to the output lines. The particular word selected that is presently available on the output lines is determined from the five input lines. There are only five inputs in a 32 × 8 ROM because $2^5 = 32$, and with five variables, we can specify 32 addresses or minterms. For each address input, there is a unique selected word. Thus, if the input address is 00000, word number 0 is selected and it appears on the output lines. If the input address is 11111, word number 31 is selected and applied to the output lines. In between, there are 30 other addresses that can select the other 30 words.

The number of addressed words in a ROM is determined from the fact that n input lines are needed to specify 2^n words. A ROM is sometimes specified by the total number of bits it contains, which is $2^n \times m$. For example, a 2048-bit ROM may be organized as 512 words of 4 bits each. This means that the unit has four output lines and nine input lines to specify $2^9 = 512$ words. The total number of bits stored in the unit is $512 \times 4 = 2048$.

Internally, the ROM is a combinational circuit with AND gates connected as a decoder and a number of OR gates equal to the number of outputs in the unit. Figure 5-22 shows the internal logic construction of a 32 × 4 ROM. The five input variables are decoded into 32 lines by means of 32 AND gates and 5 inverters. Each output of the decoder represents one of the minterms of a function of five variables. Each one of the 32 addresses selects one and only one output from the decoder. The address is a 5-bit number applied to the inputs, and the selected minterm out of the decoder is the one marked with the equivalent decimal number. The 32 outputs of the decoder are connected through fuses to each OR gate. Only four of these fuses are shown in the diagram, but actually each OR gate has 32 inputs and each input goes through a fuse that can be blown as desired.

The ROM is a two-level implementation in sum of minterms form. It does not have to be an AND-OR implementation, but it can be any other possible two-level minterm

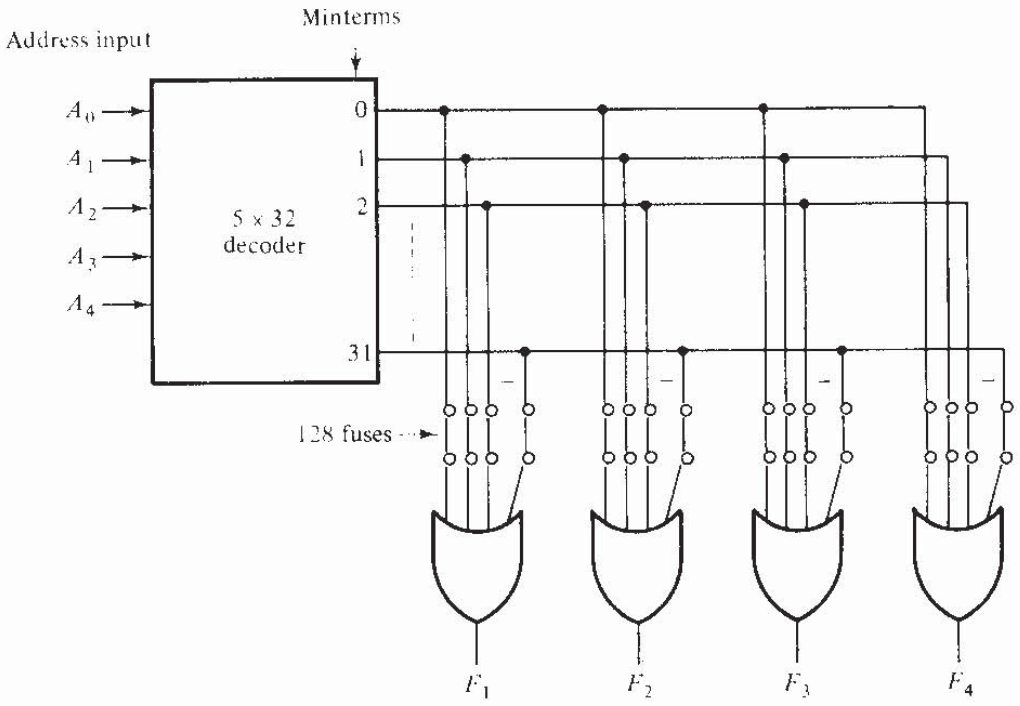


FIGURE 5-22
Logic construction of a 32 × 4 ROM

implementation. The second level is usually a wired-logic connection (see Section 3-7) to facilitate the blowing of fuses.

ROMs have many important applications in the design of digital computer systems. Their use for implementing complex combinational circuits is just one of these applications. Other uses of ROMs are presented in other parts of the book in conjunction with their particular applications.

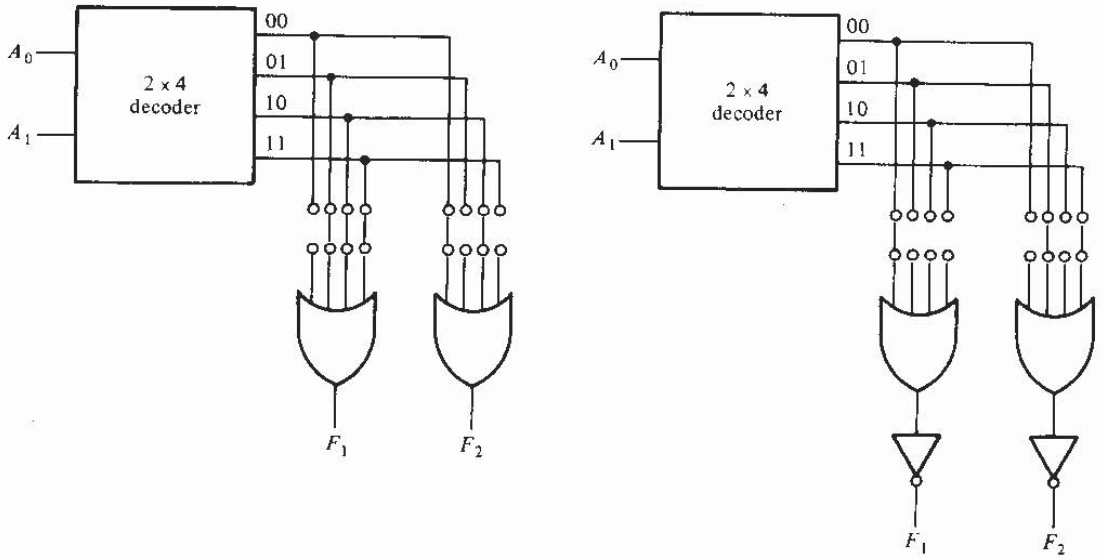
Combinational Logic Implementation

From the logic diagram of the ROM, it is clear that each output provides the sum of all the minterms of the n input variables. Remember that any Boolean function can be expressed in sum of minterms form. By breaking the links of those minterms not included in the function, each ROM output can be made to represent the Boolean function of one of the output variables in the combinational circuit. For an n-input, m-output combinational circuit, we need a $2^n \times m$ ROM. The blowing of the fuses is referred to as *programming* the ROM. The designer need only specify a ROM program table that gives the information for the required paths in the ROM. The actual programming is a hardware procedure that follows the specifications listed in the program table.

Let us clarify the process with a specific example. The truth table in Fig. 5-23(a) specifies a combinational circuit with two inputs and two outputs. The Boolean functions can be expressed in sum of minterms:

A_1	A_0	F_1	F_2
0	0	0	1
0	1	1	0
1	0	1	1
1	1	1	0

(a) Truth table



(b) ROM with AND-OR gates

(c) ROM with AND-OR-INVERT gates

FIGURE 5-23
Combinational-circuit implementation with a 4 × 2 ROM

$$F_1(A_1, A_0) = \Sigma(1, 2, 3)$$
$$F_2(A_1, A_0) = \Sigma(0, 2)$$

When a combinational circuit is implemented by means of a ROM, the functions must be expressed in sum of minterms or, better yet, by a truth table. If the output functions are simplified, we find that the circuit needs only one OR gate and an inverter. Obviously, this is too simple a combinational circuit to be implemented with a ROM. The advantage of a ROM is in complex combinational circuits. This example merely demonstrates the procedure and should not be considered in a practical situation.

The ROM that implements the combinational circuit must have two inputs and two outputs; so its size must be 4 × 2. Figure 5-23(b) shows the internal construction of such a ROM. It is now necessary to determine which of the eight available fuses must be blown and which should be left intact. This can be easily done from the output functions listed in the truth table. Those minterms that specify an output of 0 should not have a path to the output through the OR gate. Thus, for this particular case, the truth table shows three 0's, and their corresponding fuses to the OR gates must be blown. It

is obvious that we must assume here that an open input to an OR gate behaves as a 0 input.

Some ROM units come with an inverter after each of the OR gates and, as a consequence, they are specified as having initially all 0's at their outputs. The programming procedure in such ROMs requires that we open the paths of the minterms (or addresses) that specify an output of 1 in the truth table. The output of the OR gate will then generate the complement of the function, but the inverter placed after the OR gate complements the function once more to provide the normal output. This is shown in the ROM of Fig. 5-23(c).

The previous example demonstrates the general procedure for implementing any combinational circuit with a ROM. From the number of inputs and outputs in the combinational circuit, we first determine the size of ROM required. Then we must obtain the programming truth table of the ROM; no other manipulation or simplification is required. The 0's (or 1's) in the output functions of the truth table directly specify those fuses that must be blown to provide the required combinational circuit in sum of minterms form.

In practice, when one designs a circuit by means of a ROM, it is not necessary to show the internal gate connections of fuses inside the unit, as was done in Fig. 5-23. This was shown there for demonstration purposes only. All the designer has to do is specify the particular ROM (or its designation number) and provide the ROM truth table, as in Fig. 5-23(a). The truth table gives all the information for programming the ROM. No internal logic diagram is needed to accompany the truth table.

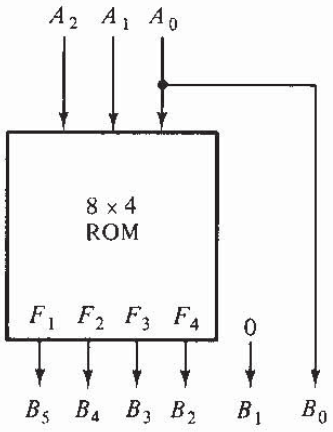
Example 5-3

Design a combinational circuit using a ROM. The circuit accepts a 3-bit number and generates an output binary number equal to the square of the input number.

The first step is to derive the truth table for the combinational circuit. In most cases, this is all that is needed. In some cases, we can fit a smaller truth table for the ROM by using certain properties in the truth table of the combinational circuit. Table 5-5 is the

TABLE 5-5
Truth Table for Circuit of Example 5-3

Inputs			Outputs						
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	Decimal
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49



(a) Block diagram

A_2	A_1	A_0	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	0	1
1	1	1	1	1	0	0

(b) ROM truth table

FIGURE 5-24
ROM implementation of Example 5-3

truth table for the combinational circuit. Three inputs and six outputs are needed to accommodate all possible numbers. We note that output B_0 is always equal to input A_0; so there is no need to generate B_0 with a ROM since it is equal to an input variable. Moreover, output B_1 is always 0, so this output is always known. We actually need to generate only four outputs with the ROM; the other two are easily obtained. The minimum-size ROM needed must have three inputs and four outputs. Three inputs specify eight words, so the ROM size must be 8 × 4. The ROM implementation is shown in Fig. 5-24. The three inputs specify eight words of four bits each. The other two outputs of the combinational circuit are equal to 0 and A_0. The truth table in Fig. 5-24 specifies all the information needed for programming the ROM, and the block diagram shows the required connections. ■

Types of ROMs

The required paths in a ROM may be programmed in two different ways. The first is called *mask programming* and is done by the manufacturer during the last fabrication process of the unit. The procedure for fabricating a ROM requires that the customer fill out the truth table the ROM is to satisfy. The truth table may be submitted on a special form provided by the manufacturer. More often, it is submitted in a computer input medium in the format specified on the data sheet of the particular ROM. The manufacturer makes the corresponding mask for the paths to produce the 1's and 0's according to the customer's truth table. This procedure is costly because the vendor charges the customer a special fee for custom masking a ROM. For this reason, mask programming is economical only if large quantities of the same ROM configuration are to be manufactured.

For small quantities, it is more economical to use a second type of ROM called a *programmable read-only memory,* or PROM. When ordered, PROM units contain all 0's (or all 1's) in every bit of the stored words. The fuses in the PROM are blown by application of current pulses through the output terminals. A blown fuse defines one binary state and an unbroken link represents the other state. This allows the user to program the unit in the laboratory to achieve the desired relationship between input addresses and stored words. Special units called *PROM programmers* are available commercially to facilitate this procedure. In any case, all procedures for programming ROMs are *hardware* procedures even though the word *programming* is used.

The hardware procedure for programming ROMs or PROMs is irreversible and, once programmed, the fixed pattern is permanent and cannot be altered. Once a bit pattern has been established, the unit must be discarded if the bit pattern is to be changed. A third type of unit available is called *erasable PROM,* or EPROM. EPROMs can be restructured to the initial value (all 0's or all 1's) even though they have been changed previously. When an EPROM is placed under a special ultraviolet light for a given period of time, the shortwave radiation discharges the internal gates that serve as contacts. After erasure, the ROM returns to its initial state and can be reprogrammed. Certain ROMs can be erased with electrical signals instead of ultraviolet light, and these are called *electrically erasable PROMs,* or EEPROMs.

The function of a ROM can be interpreted in two different ways. The first interpretation is of a unit that implements any combinational circuit. From this point of view, each output terminal is considered separately as the output of a Boolean function expressed in sum of minterms. The second interpretation considers the ROM to be a storage unit having a fixed pattern of bit strings called *words*. From this point of view, the inputs specify an *address* to a specific stored word, which is then applied to the outputs. For example, the ROM of Fig. 5-24 has three address lines, which specify eight stored words as given by the truth table. Each word is four bits long. This is the reason why the unit is given the name *read-only memory*. *Memory* is commonly used to designate a storage unit. *Read* is commonly used to signify that the contents of a word specified by an address in a storage unit is placed at the output terminals. Thus, a ROM is a memory unit with a fixed word pattern that can be read out upon application of a given address. The bit pattern in the ROM is permanent and cannot be changed during normal operation.

ROMs are widely used to implement complex combinational circuits directly from their truth tables. They are useful for converting from one binary code to another (such as ASCII to EBCDIC and vice versa), for arithmetic functions such as multipliers, for display of characters in a cathode-ray tube, and in many other applications requiring a large number of inputs and outputs. They are also employed in the design of control units of digital systems. As such, they are used to store fixed bit patterns that represent the sequence of control variables needed to enable the various operations in the system. A control unit that utilizes a ROM to store binary control information is called a *microprogrammed control unit*.

Synchronous 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 at 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

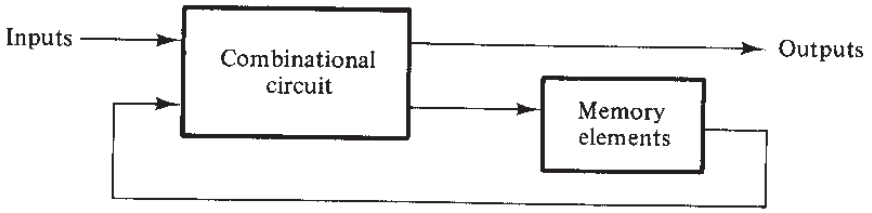


FIGURE 6-1
Block diagram of a sequential circuit

used in asynchronous sequential circuits are time-delay devices. The memory capability of a time-delay device is due to the finite time it takes 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. Asynchronous sequential circuits are presented in Chapter 9.

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 that 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 divided into independent discrete steps, each of which is considered separately. The sequential circuits discussed in this chapter 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 that 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-flop 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 in the manner in which the inputs affect the binary state. The most common types of flip-flops are discussed in what follows.

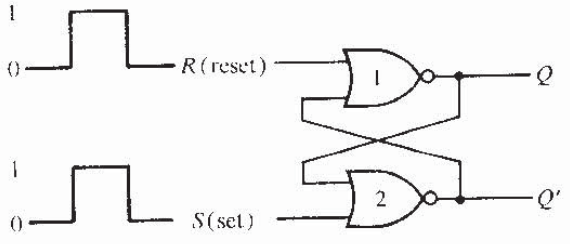
Basic Flip-Flop Circuit

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 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 1'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

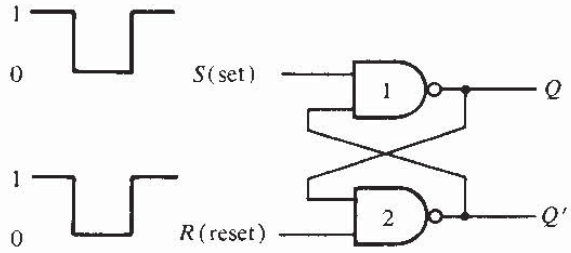


(a) Logic diagram

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



(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

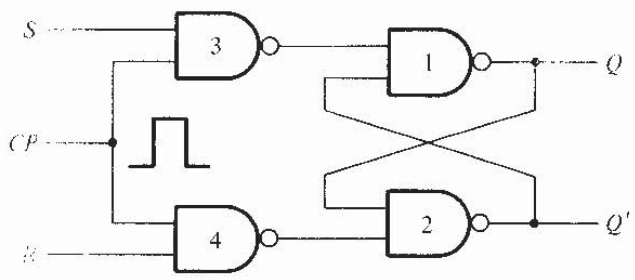
1-state). When $Q = 0$ and $Q' = 1$, it is in the *clear state* (or 0-state). The outputs Q and Q' are complements of each other and are referred to as the normal and complement outputs, respectively. The binary state of the flip-flop is taken to be the value of the normal output.

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 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.

RS Flip-flop

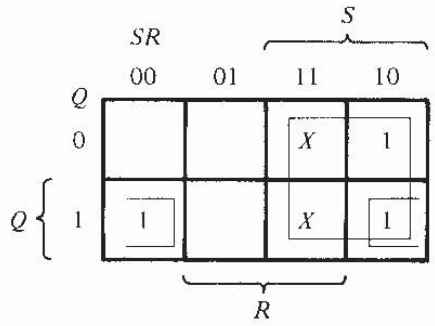
The operation of the basic flip-flop can be modified by providing an additional control input that determines when the state of the circuit is to be changed. An *RS* flip-flop with a clock pulse (*CP*) input is shown in Fig. 6-4(a). It consists of a basic flip-flop circuit and two additional NAND gates. The pulse input acts as an enable signal for the other two inputs. The outputs of NAND gates 3 and 4 stay at the logic 1 level as long as the *CP* input remains at 0. This is the quiescent condition for the basic flip-flop. When the pulse input goes to 1, information from the S or R input is allowed to reach the output. The set state is reached with $S = 1$, $R = 0$, and $CP = 1$. This causes the output of gate 3 to go to 0, the output of gate 4 to remain at 1, and the output of the flip-flop at Q to go to 1. To change to the reset state, the inputs must be $S = 0$, $R = 1$, and $CP = 1$.



(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

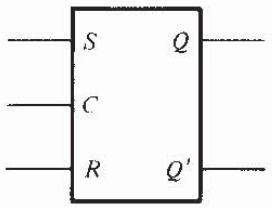
(b) Characteristic table



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

$SR = 0$

(c) Characteristic equation



(d) Graphic symbol

FIGURE 6-4
RS flip-flop

In either case, when CP returns to 0, the circuit remains in its previous state. When $CP = 1$ and both the S and R inputs are equal to 0, the state of the circuit does not change.

An indeterminate condition occurs when $CP = 1$ and both S and R are equal to 1. This condition places 0's in the outputs of gates 3 and 4 and 1's in both outputs Q and Q'. When the CP input goes back to 0 (while S and R are maintained at 1), it is not possible to determine the next state, as it depends on whether the output of gate 3 or gate 4 goes to 1 first. This indeterminate condition makes the circuit of Fig. 6-4(a) difficult to manage and it is seldom used in practice. Nevertheless, it is an important circuit because all other flip-flops are constructed from it.

The characteristic table of the flip-flop is shown in Fig. 6-4(b). This table shows the operation of the flip-flop in tabular form. Q is an abbreviation of $Q(t)$ and stands for the binary state of the flip-flop before the application of a clock pulse, referred to as the *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 application of a single pulse, referred to as the *next state*. Note that the CP input is not included in the characteristic table. The table must

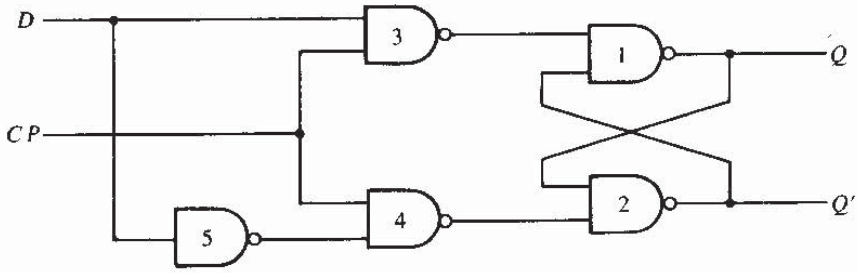
be interpreted as follows: Given the present state Q and the inputs S and R, the application of a single pulse in the CP input causes the flip-flop to go to the next state, $Q(t + 1)$.

The characteristic equation of the flip-flop is derived in the map of Fig. 6-4(c). 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 with dont't-care X's in the map, since they may result in either 1 or 0. However, the relation $SR = 0$ must be included as part of the characteristic equation to specify that both S and R cannot equal to 1 simultaneously.

The graphic symbol of the RS flip-flop is shown in Fig. 6-4(d). It consists of a rectangular-shape block with inputs S, R, and C. The outputs are Q and Q', where Q' is the complement of Q (except in the indeterminate state).

D Flip-Flop

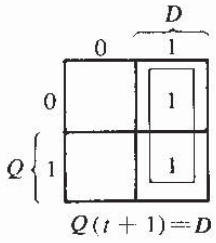
One way to eliminate the undesirable condition of the indeterminate state in the RS flip-flop is to ensure that inputs S and R are never equal to 1 at the same time. This is done in the D flip-flop shown in Fig. 6-5(a). The D flip-flop has only two inputs: D and CP. The D input goes directly to the S input and its complement is applied to the R input. As long as the pulse input is at 0, the outputs of gates 3 and 4 are at the 1 level and the circuit cannot change state regardless of the value of D. The D input is sampled when $CP = 1$. If D is 1, the Q output goes to 1, placing the circuit in the set state. If D is 0, output Q goes to 0 and the circuit switches to the clear state.



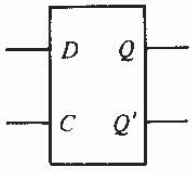
(a) Logic diagram

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

(b) Characteristic table



(c) Characteristic equation



(d) Graphic symbol

FIGURE 6-5
D flip-flop

The *D* flip-flop receives the designation from its ability to hold *data* into its internal storage. This type of flip-flop is sometimes called a *gated D-latch*. The *CP* input is often given the designation *G* (for *gate*) to indicate that this input enables the gated latch to make possible data entry into the circuit. The binary information present at the data input of the *D* flip-flop is transferred to the *Q* output when the *CP* input is enabled. The output follows the data input as long as the pulse remains in its 1 state. When the pulse goes to 0, the binary information that was present at the data input at the time the pulse transition occurred is retained at the *Q* output until the pulse input is enabled again.

The characteristic table for the D flip-flop is shown in Fig. 6-5(b). It shows that the next state of the flip-flop is independent of the present state since $Q(t + 1)$ is equal to input *D* whether *Q* is equal to 0 or 1. This means that an input pulse will transfer the value of input *D* into the output of the flip-flop independent of the value of the output before the pulse was applied. The characteristic equation shows clearly that $Q(t + 1)$ is equal to *D*.

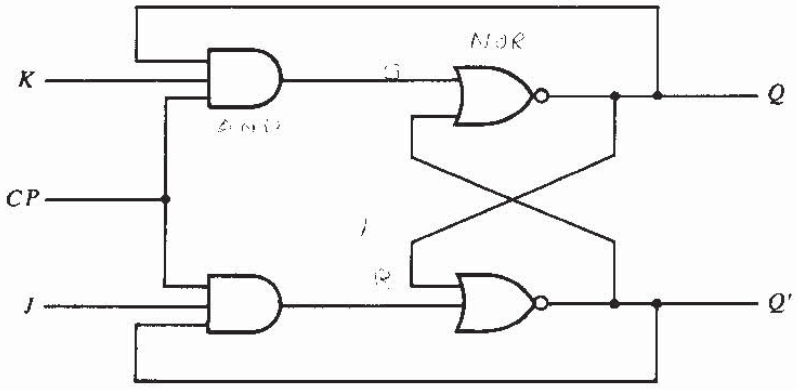
The graphic symbol for the level sensitive *D* flip-flop is shown in Fig. 6-5(d). The graphic symbol for a transition-sensitive *D* flip-flop is shown later in Fig. 6-14.

JK and T Flip-Flops

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, respectively. The input marked *J* is for *set* and the input marked *K* is for *reset*. When both inputs *J* and *K* are equal to 1, the flip-flop switches to its complement state, that is, if $Q = 1$, it switches to $Q = 0$, and vice versa.

A *JK* flip-flop constructed with two cross-coupled NOR gates and two AND gates 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 flop-flop is set with a clock pulse only when Q' was previously 1. When both *J* and *K* are 1, the input pulse is transmitted through one AND gate only: the one whose input is connected to the flip-flop output that is presently equal to 1. Thus, if $Q = 1$, the output of the upper AND gate becomes 1 upon application of the clock pulse, and the flip-flop is cleared. If $Q' = 1$, the output of the lower AND gate becomes 1 and the flip-flop is set. In either case, the output state of the flip-flop is complemented. The behavior of the *JK* flip-flop is demonstrated in the characteristic table of Fig. 6-6(b).

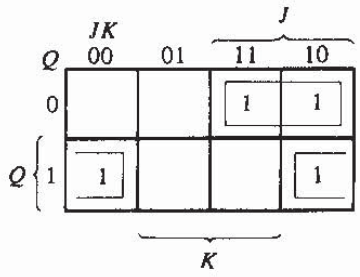
It is very important to realize that because of the feedback connection in the *JK* flip-flop, a *CP* pulse that remains in the 1 state while both *J* and *K* are equal to 1 will cause the output to complement again and repeat complementing until the pulse goes back to 0. To avoid this undesirable operation, the clock pulse must have a time duration that is shorter than the propagation delay time of the flip-flop. This is a restrictive requirement, since the operation of the circuit depends on the width of the pulse. 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.



(a) Logic diagram

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

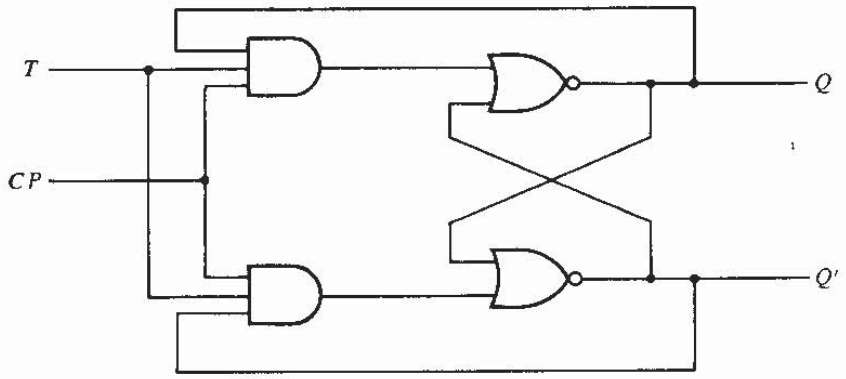
(b) Characteristic table



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

(c) Characteristic equation

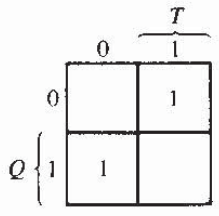
FIGURE 6-6
JK flip-flop



(a) Logic diagram

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

(b) Characteristic table



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

(c) Characteristic equation

FIGURE 6-7
T flip-flop

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 the JK flip-flop when both inputs are tied together. The designation T comes from the ability of the flip-flop to "toggle," or complement, its state. Regardless of the present state, the flip-flop complements its output when the clock pulse occurs while input T is 1. The characteristic table and characteristic equation show that when $T = 0$, $Q(t + 1) = Q$, that is, the next state is the same as the present state and no change occurs. When $T = 1$, then $Q(t + 1) = Q'$, and the state of the flip-flop is complemented.

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 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

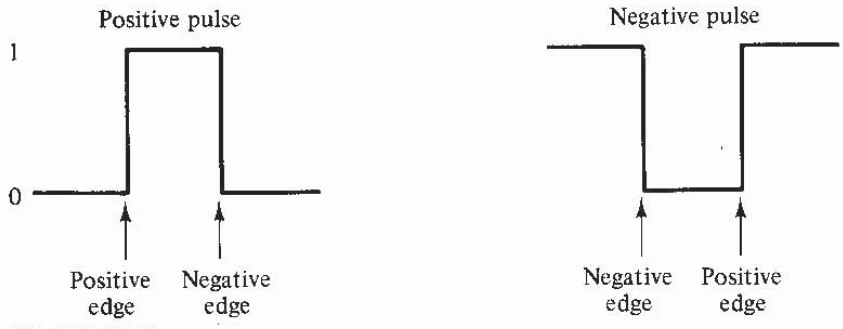


FIGURE 6-8
Definition of clock-pulse transition

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 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 in what follows.

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–slave 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 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

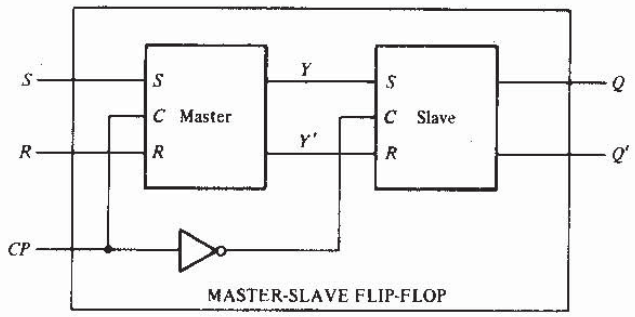


FIGURE 6-9
Logic diagram of a master–slave flip-flop

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 transi-

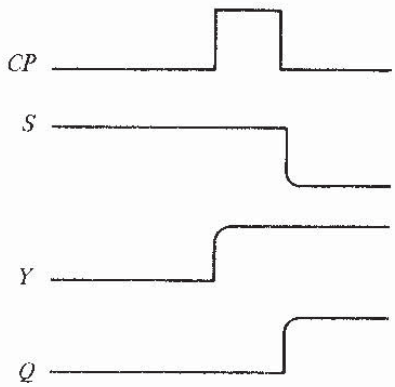


FIGURE 6-10
Timing relationships in a master–slave flip-flop

tion 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 information 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

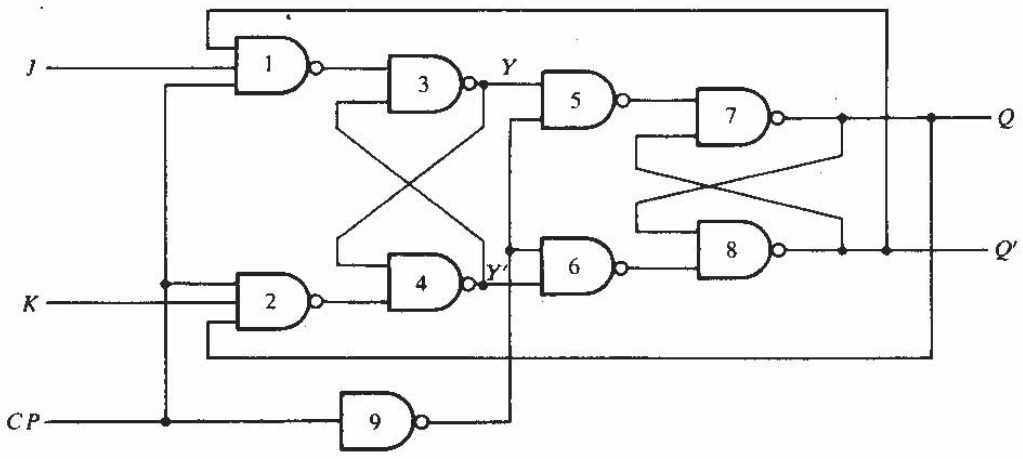


FIGURE 6-11
Clocked master–slave *JK* flip-flop

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.

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.

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 S 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

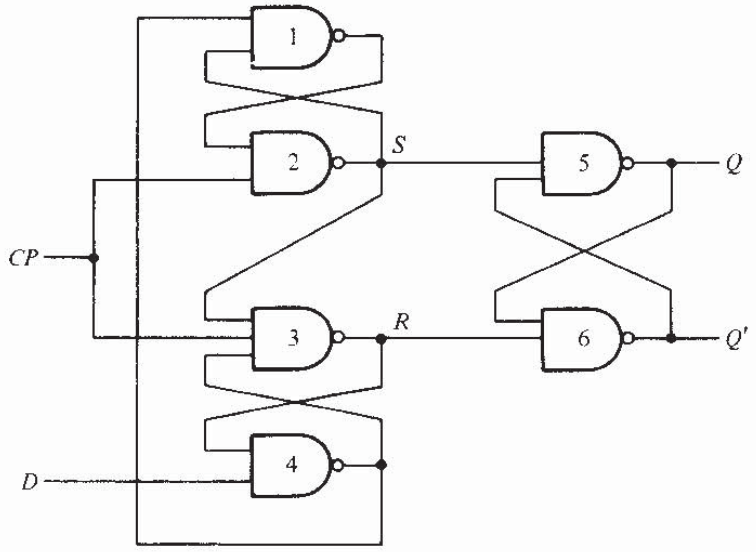
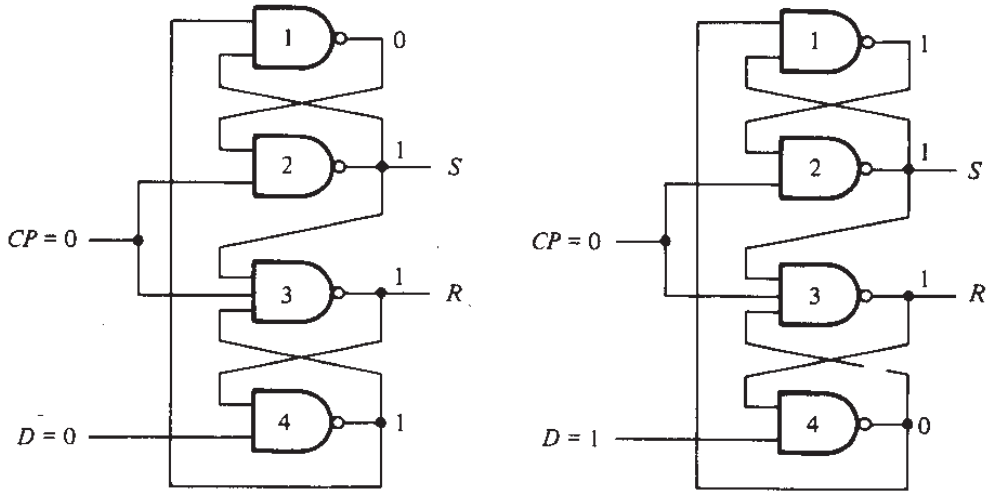


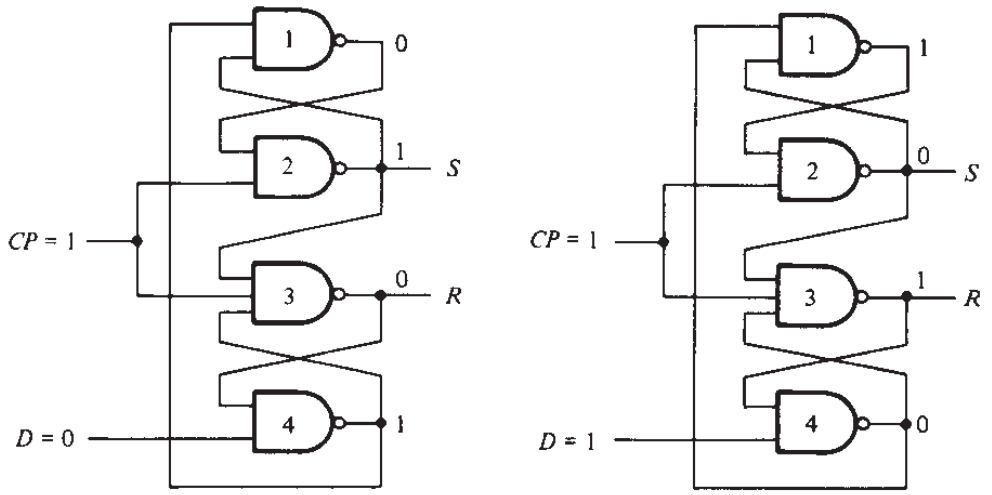
FIGURE 6-12
D-type positive-edge-triggered flip-flop

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 1, thus making $S = R = 1$, which is the condition for a steady-state output. When



(a) With $CP = 0$



(b) With $CP = 1$

FIGURE 6-13
Operation of the D-type edged-triggered flip-flop

$D = 0$, 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 1, 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 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.

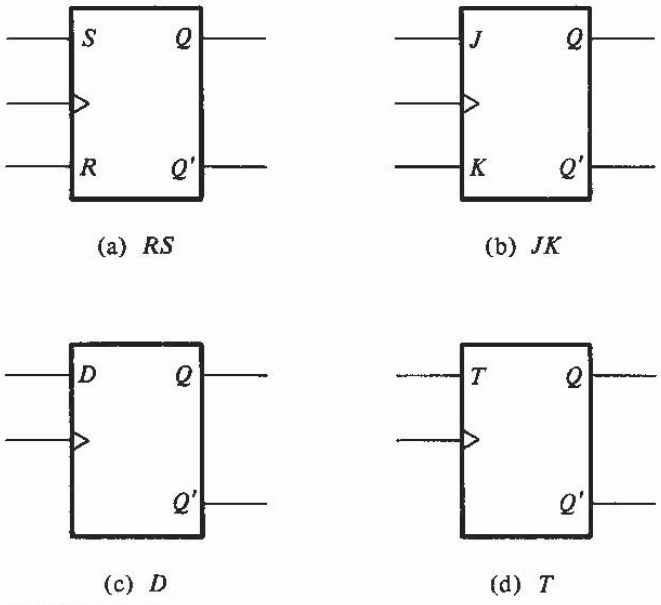
If $D = 1$ when $CP = 1$, then S changes to 0, but R remains at 1, which causes the output of the flip-flop Q to go to 1. A change in D while $CP = 1$ does not alter S and R, because gate 1 is maintained at 1 by the 0 signal from S. When CP goes to zero, both S and R go to 1 to prevent the output from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

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.

Graphic Symbols

The graphic symbols for four flip-flops are shown in Fig. 6-14. The input letter symbols in each diagram designate the type of flip-flop such as RS, JK, D, and T. The clock-pulse input is recognized in the diagram from the arrowhead-shape symbol. This is a symbol of a *dynamic indicator* and denotes that the flip-flop responds to a positive-edge transition of the clock. The presence of a small circle outside the block along the dy-



(a) *RS* (b) *JK*

(c) *D* (d) *T*

FIGURE 6-14
Graphic symbols for flip-flops

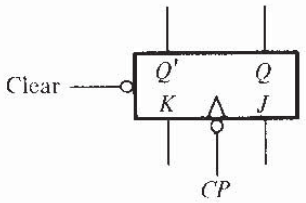
namic indicator designates a negative-edge transition for triggering the flip-flop. The letter symbol C is used for the clock input when the flip-flop responds to a pulse level rather than a pulse transition. This was shown in Fig. 6-5(d) for the level-sensitive D flip-flop.

The outputs of the flip-flop are marked with the letter symbol Q and Q' within the block. The flip-flop may be assigned a different variable name even though Q is written inside the block. In that case, the letter symbol for the flip-flop output is marked outside the block 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 output, it is not necessary to use an inverter because the complement value is available directly from Q'.

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 of 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 negative-edge-triggered *JK* flip-flop with direct clear is shown in Fig. 6-15. The clock-pulse input *CP* has a small circle under the dynamic



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 6-15
JK flip-flop with direct clear

symbol to indicate that the outputs change in response to a negative transition of the clock. 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.

6-4 ANALYSIS OF CLOCKED SEQUENTIAL CIRCUITS

The behavior of a sequential circuit is determined from the inputs, the outputs, and the state of its flip-flops. The outputs and the next state are both a function of the inputs and the present state. The analysis of a sequential circuit 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 the sequential circuit. However, these expressions must include the necessary time sequence, either directly or indirectly.

A logic diagram is recognized as a clocked 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 circuit gates. In this section, we first introduce a specific example of a clocked sequential circuit with D flip-flops and use it to present the basic methods for describing the behavior of sequential circuits. Additional examples are used throughout the discussion to illustrate other procedures.

Sequential-Circuit Example

An example of a clocked sequential circuit is shown in Fig. 6-16. The circuit consists of two D flip-flops A and B, an input x, and an output y. Since the D inputs determine

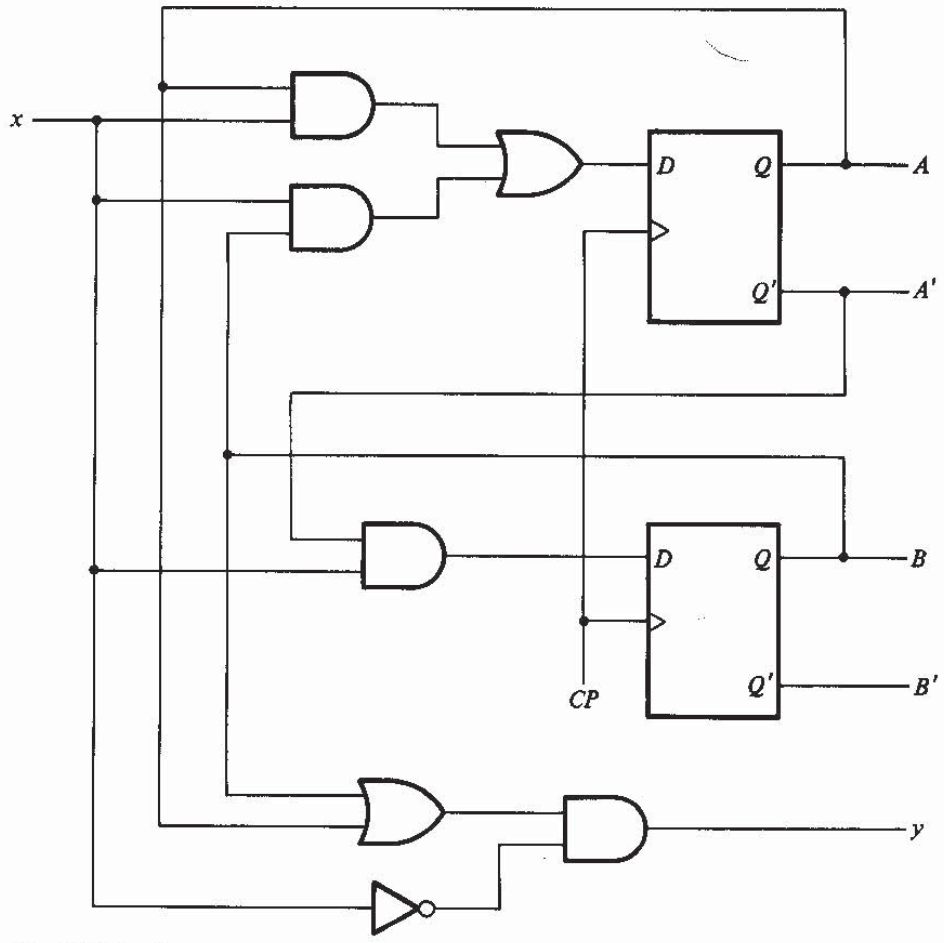


FIGURE 6-16
Example of a sequential circuit

the flip-flops' next state, it is possible to write a set of next-state equations for the circuit:

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

A state equation is an algebraic expression that specifies the condition for a flip-flop state transition. The left side of the equation denotes the next state of the flip-flop and the right side of the equation is a Boolean expression that specifies the present state and input conditions that make the next state equal to 1. Since all the variables in the Boolean expressions are a function of the present state, we can omit the designation (t) after each variable for convenience. The previous equations can be expressed in more compact form as follows:

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

The Boolean expressions for the next state can be derived directly from the gates that form the combinational-circuit part of the sequential circuit. The outputs of the combinational circuit are applied to the D inputs of the flip-flops. The D input values determine the next state.

Similarly, the present-state value of the output can be expressed algebraically as follows:

$$y(t) = [A(t) + B(t)]x'(t)$$

Removing the symbol (t) for the present state, we obtain the output Boolean function:

$$y = (A + B)x'$$

State Table

The time sequence of inputs, outputs, and flip-flop states can be enumerated in a *state table*. The state table for the circuit of Fig. 6-16 is shown in Table 6-1. The table consists of four sections labeled *present state*, *input*, *next state*, and *output*. The present-state section shows the states of flip-flops A and B at any given time t. The input section gives a value of x for each possible present state. The next-state section shows the states of the flip-flops one clock period later at time $t + 1$. The output section gives the value of y for each present state.

The derivation of a state table consists of first listing all possible binary combinations of present state and inputs. In this case, we have eight binary combinations from 000 to 111. The next-state values are then determined from the logic diagram or from the state equations. The next state of flip-flop A must satisfy the state equation

$$A(t + 1) = Ax + Bx$$

The next-state section in the state table under column A has three 1's where the present

TABLE 6-1
State Table for the Circuit of Fig. 6-16

Present State		Input	Next State		Output
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

TABLE 6-2
Second Form of the State Table

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

state and input value satisfy the conditions that the present state of A and input x are both equal to 1 or the present state of B and input x are both equal to 1. Similarly, the next state of flip-flop B is derived from the state equation

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

It is equal to 1 when the present state of A is 0 and input x is equal to 1. The output column is derived from the output equation

$$y = Ax' + Bx$$

The state table of any sequential circuit with D-type flip-flops is obtained by the same procedure outlined in the previous example. In general, a sequential circuit with m flip-flops and n inputs needs 2^{m+n} rows in the state table. The binary numbers from 0 through $2^{m+n} - 1$ are listed under the present-state and input columns. The next-state section has m columns, one for each flip-flop. The binary values for the next state are derived directly from the state equations. The output section has as many columns as there are output variables. Its binary value is derived from the circuit or from the Boolean function in the same manner as in a truth table. Note that the examples in this chapter use only one input and one output variable, but, in general, a sequential circuit may have two or more inputs or outputs.

It is sometimes convenient to express the state table in a slightly different form. In the other configuration, the state table has only three sections: present state, next state, and output. The input conditions are enumerated under the next-state and output sections. The state table of Table 6-1 is repeated in Table 6-2 using the second form. For each present state, there are two possible next states and outputs, depending on the value of the input. We will use both forms of the state table. One form may be preferable over the other, depending on the application.

State Diagram

The information available in a state table can be represented graphically in a state diagram. In this type of 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

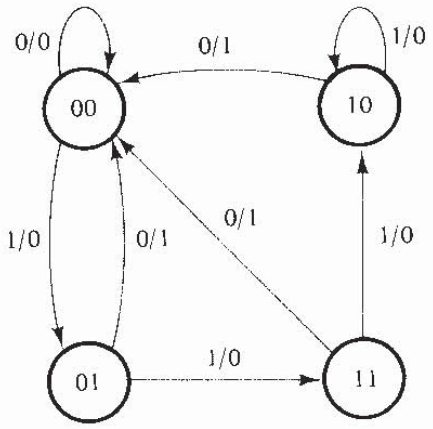


FIGURE 6-17
State diagram of the circuit of Fig. 6-16

the sequential circuit of Fig. 6-16 is shown in Fig. 6-17. The state diagram provides the same information as the state table and is obtained directly from Table 6-2. The binary number inside each circle identifies the state of the flip-flops. The directed lines are labeled with two binary numbers separated by a slash. The input value during the present state is labeled first and the number after the slash gives the output during the present state. For example, the directed line from state 00 to 01 is labeled 1/0, meaning that when the sequential circuit is in the present state 00 and the input is 1, the output is 0. After a clock transition, the circuit goes to the next state, 01. The same clock transition may change the input value. If the input changes to 0, then the output becomes 1, but if the input remains at 1, the output stays at 0. This information is obtained from the state diagram along the two directed lines emanating from the circle representing state 01. A directed line connecting a circle with itself indicates that no change of state occurs.

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 the state table. The state diagram gives a pictorial view of state transitions and is the form suitable for human interpretation of the circuit operation. For example, the state diagram of Fig. 6-17 clearly shows that, starting from state 00, the output is 0 as long as the input stays at 1. The first 0 input after a string of 1's gives an output of 1 and transfers the circuit back to the initial state 00.

Flip-Flop Input Functions

The logic diagram of a sequential circuit consists of flip-flops and gates. 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 function; 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.

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. 6-16 has one input x, one output y, and two D flip-flops A and B. The logic diagram can be expressed algebraically with two flip-flop input functions and one output-circuit function:

$$DA = Ax + Bx$$

$$DB = A'x$$

$$y = (A + B)x'$$

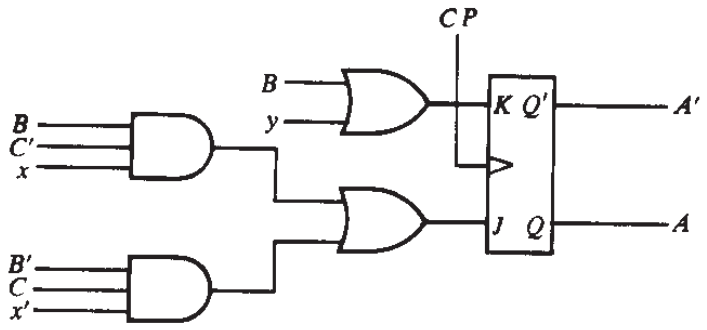


FIGURE 6-18
Implementation of the flip-flop input functions $JA = BC'x + B'Cx'$ and $KA = B + y$

This set of Boolean functions provides all the necessary information for drawing the logic diagram of the sequential circuit. The symbol *DA* specifies a *D* flip-flop labeled *A*. *DB* specifies a second *D* flip-flop labeled *B*. The Boolean expressions associated with these two variables and the expression for output *y* 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.

Characteristic Tables

The analysis of a sequential circuit with flip-flops other than the *D* type is complicated because the relationship between the inputs of the flip-flop and the next state is not straightforward. This relationship is best described by means of a characteristic table rather than a state equation. The characteristic tables of four flip-flops were presented in Section 6-2. When analyzing sequential circuits, it is more convenient to present the characteristic table in a somewhat different form. The modified form of the characteristic tables of four types of flip-flops are shown in Table 6-3. They define the next state as a function of the inputs and present state. $Q(t)$ refers to the present state prior to the application of a pulse. $Q(t + 1)$ is the next state one clock period later. Note that the clock-pulse input is not listed in the characteristic table, but is implied to occur between time t and $t + 1$.

The characteristic table for the *JK* flip-flop shows that the next state is equal to the

TABLE 6-3
Flip-Flop Characteristic Tables

JK Flip-Flop

J	K	$Q(t + 1)$	
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	$Q'(t)$	Complement

RS Flip-Flop

S	R	$Q(t + 1)$	
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	?	Unpredictable

D Flip-Flop

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

T Flip-Flop

T	$Q(t + 1)$	
0	$Q(t)$	No change
1	$Q'(t)$	Complement

present state when inputs J and K are both equal to 0. This can be expressed as $Q(t + 1) = Q(t)$, indicating that the clock pulse produces no change of state. When $K = 1$ and $J = 0$, the clock pulse resets the flip-flop and $Q(t + 1) = 0$. With $J = 1$ and $K = 0$, the flip-flop sets and $Q(t + 1) = 1$. When both J and K are equal to 1, the next state changes to the complement of the present state, which can be expressed as $Q(t + 1) = Q'(t)$.

The RS flip-flop is similar to the JK when S is replaced by J and R by K except for the indeterminate case. The question mark for the next state when S and R are both equal to 1 indicates an unpredictable next state.

The next state of a D flip-flop is dependent only on the D input and independent of the present state, which can be expressed as $Q(t + 1) = D$. This means that the next-state value can be obtained directly from the binary logic value of the D input. Note that the D flip-flop does not have a "no-change" condition. This condition can be accomplished either by disabling the clock pulses or by leaving the clock pulses and connecting the output back into the D input when the state of the flip-flop must remain the same.

The T flip-flop is obtained from a JK flip-flop when inputs J and K are tied together. The characteristic table has only two conditions. When $T = 0$ ($J = K = 0$), a clock pulse does not change the state. When $T = 1$ ($J = K = 1$), a clock pulse complements the state of the flip-flop.

Analysis with *JK* and Other Flip-Flops

It was shown previously that the next-state values of a sequential circuit with D flip-flops can be derived directly from the next-state equations. When other types of flip-flops are used, it is necessary to refer to the characteristic table. The next-state values of a sequential circuit that uses any other type of flip-flop such as JK, RS, or T can be derived by following a two-step procedure:

1. Obtain the binary values of each flip-flop input function in terms of the present-state and input variables.
2. Use the corresponding flip-flop characteristic table to determine the next state.

To illustrate this procedure, consider the sequential circuit with two JK flip-flops A and B and one input x, as shown in Fig. 6-19. The circuit has no outputs and, therefore, the state table does not need an output column. (The outputs of the flip-flops may be considered as the outputs in this case.) The circuit can be specified by the following flip-flop input functions:

$$JA = B \qquad JB = x'$$

$$KA = Bx' \qquad KB = A'x + Ax' = A \oplus x$$

The state table of the sequential circuit is shown in Table 6-4. First, we derive the binary values listed under the columns labeled *flip-flop inputs*. These columns are not part of the state table, but they are needed for the purpose of evaluating the next state

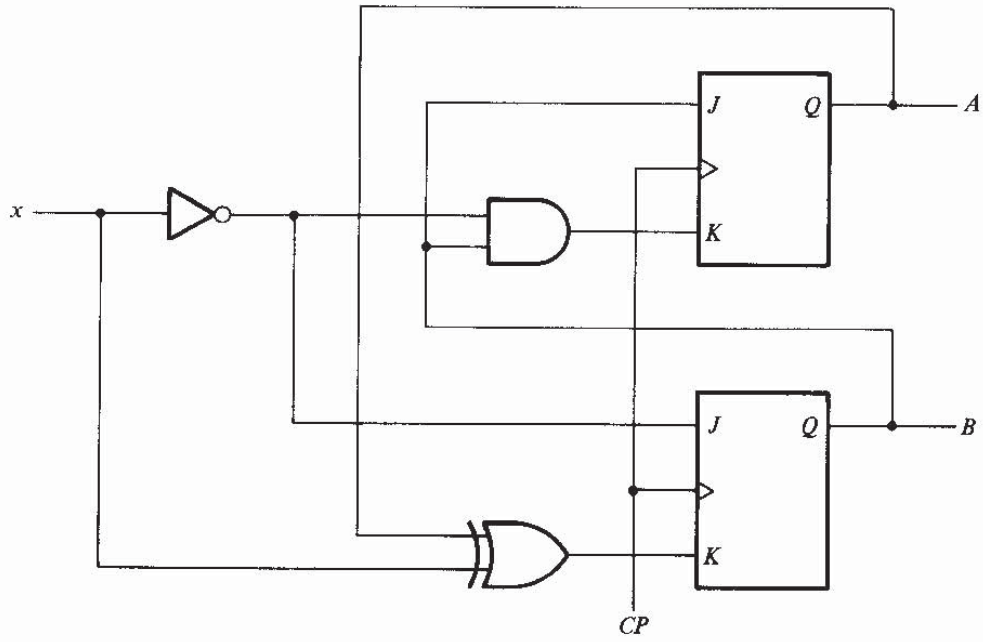


FIGURE 6-19
Sequential circuit with JK flip-flops

as specified in step 1 of the procedure. These binary values are obtained directly from the four input flip-flop functions in a manner similar to that for obtaining a truth table from an algebraic expression. The next state of each flip-flop is evaluated from the corresponding J and K inputs and the characteristic table of the JK flip-flop listed in Table 6-3. There are four cases to consider. When $J = 1$ and $K = 0$, the next state is 1. When $J = 0$ and $K = 1$, the next state is 0. When $J = K = 0$, there is no change of state and the next-state value is the same as the present state. When $J = K = 1$, the

TABLE 6-4
State Table for Sequential Circuit with *JK* flip-Flops

Present state		Input	Next state		Flip-flop inputs			
A	*B*	*x*	*A*	*B*	*JA*	*KA*	*JB*	*KB*
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1	0
0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0

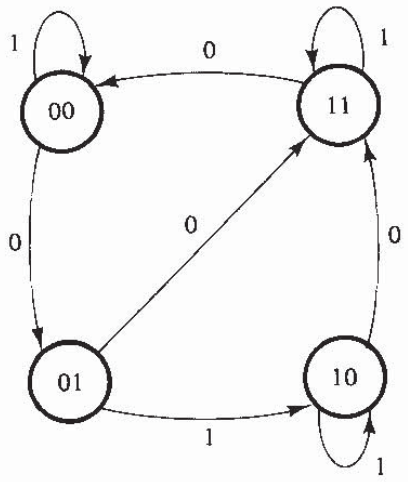


FIGURE 6-20
State diagram of the circuit of Fig. 6-19

next-state bit is the complement of the present-state bit. Examples of the last two cases occur in the table when the present state AB is 10 and input x is 0. JA and KA are both equal to 0 and the present state of A is 1. Therefore, the next state of A remains the same and is equal to 1. In the same row of the table, JB and KB are both equal to 1. Since the present state of B is 0, the next state of B is complemented and changes to 1.

The state diagram of the sequential circuit is shown in Fig. 6-20. Note that since the circuit has no outputs, the directed lines out of the circles are marked with one binary number only to designate the value of input x.

Mealy and Moore Models

The most general model of a sequential circuit has inputs, outputs, and internal states. It is customary to distinguish between two models of sequential circuits: the Mealy model and the Moore model. In the Mealy model, the outputs are functions of both the present state and inputs. In the Moore model, the outputs are a function of the present state only. An example of a Mealy model is shown in Fig. 6-16. Output y is a function of both input x and the present state of A and B. The corresponding state diagram shown in Fig. 6-17 has both the input and output values included along the directed lines between the circles. An example of a Moore model is shown in Fig. 6-19. Here the outputs are taken from the flip-flops and are a function of the present state only. The corresponding state diagram in Fig. 6-20 has only the inputs marked along the directed lines. The outputs are the flip-flop states marked inside the circles. The outputs of a Moore model can be a combination of flip-flop variables such as $A \oplus B$. This output is a function of the present state only even though it requires an additional exclusive-OR gate to generate it.

The state table of a Mealy model sequential circuit must include an output section that is a function of both the present state and inputs. When the outputs are taken directly from the flip-flops, the state table can exclude the output section because the out-

puts are already listed in the present-state columns of the state table. In a general Moore model sequential circuit, there may be an output section, but it will be a function of the present state only.

In a Moore model, the outputs of the sequential circuit are synchronized with the clock because they depend on only flip-flop outputs that are synchronized with the clock. In a Mealy model, the outputs may change if the inputs change during the clock-pulse period. Moreover, the outputs may have momentary false values because of the delay encountered from the time that the inputs change and the time that the flip-flop outputs change. In order to synchronize a Mealy type circuit, the inputs of the sequential circuit must be synchronized with the clock and the outputs must be sampled only during the clock-pulse transition.

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.

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. 6-21. 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;

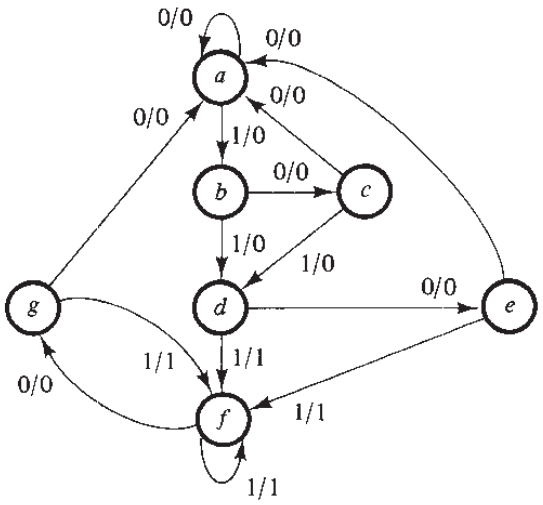


FIGURE 6-21
State diagram

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 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. 6-21. 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 6-5 and is obtained directly from the state diagram of Fig. 6-21.

TABLE 6-5
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

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 6-5. 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. The procedure of removing a state and replacing it by its equivalent is demonstrated in Table 6-6. 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

TABLE 6-6
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

TABLE 6-7
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

final reduced table is shown in Table 6-7. The state diagram for the reduced table consists of only five states and is shown in Fig. 6-22. 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. 6-22 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. 6-21, if we replace g by e and f by d.

The checking of each pair of states for possible equivalence can be done systematically by means of a procedure that employs an implication table. The implication table consists of squares, one for every suspected pair of possible equivalent states. By judicious use of the table, it is possible to determine all pairs of equivalent states in a state table. The use of the implication table for reducing the number of states in a state table is demonstrated in Section 9-5.

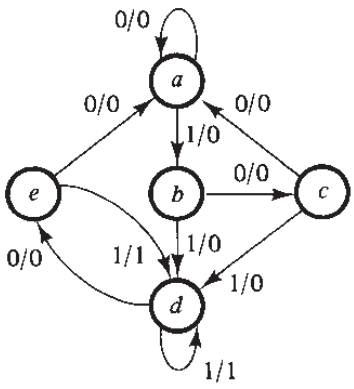


FIGURE 6-22
Reduced state diagram

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 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 6-5 is used, we must assign binary values to seven states; the remaining state is unused. If the state table of Table 6-7 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.

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.

TABLE 6-8
Three Possible Binary State Assignments

State	Assignment 1	Assignment 2	Assignment 3
a	001	000	000
b	010	010	100
c	011	011	010
d	100	101	101
e	101	111	011

TABLE 6-9
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

The binary state-assignment alternatives available can be demonstrated in conjunction with the sequential circuit specified in Table 6-7. Remember that, in this example, the binary values 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 6-8 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.

Table 6-9 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, whereas 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.

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.

6-6 FLIP-FLOP EXCITATION TABLES

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 6-10
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

(b) T

Table 6-10 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 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.

RS Flip-Flop

The excitation table for the RS flip-flop is shown in Table 6-10(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, Table 6-3, 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 1-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.

JK Flip-Flop

The excitation table for the *JK* flip-flop is shown in Table 6.10(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.

D Flip-Flop

The excitation table for the D flip-flop is shown in Table 6-10(c). From the characteristic table, Table 6-3, 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 $Q(t)$.

T Flip-Flop

The excitation table for the T flip-flop is shown in Table 6-10(d). From the characteristic table, Table 6-3, 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.

Other Flip-Flops

The design procedure to be described in the next section 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.

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 that, 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:

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 m flip-flops can represent up to 2^m distinct 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 *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.

We wish to design the clocked sequential circuit whose state diagram is given in Fig. 6-23. 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 slash, 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 6-11. 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 6-12, where the present state and input

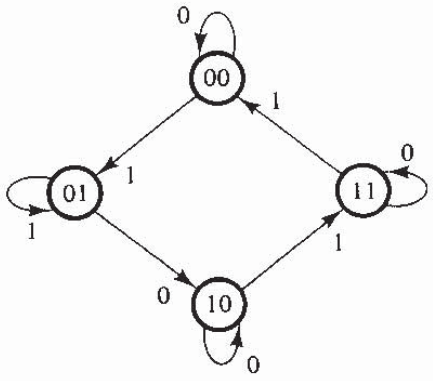


FIGURE 6-23
State diagram for design example

TABLE 6-11
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

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 6-11. 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 6-10(b). This table is now used to derive the excitation table of the circuit. For example, in the first row of Table 6-12, we have a transition for flip-flop *A* from 0 in the present state to 0 in the next state. In Table 6-10(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 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 6-12 shows a transition for flip-flop *B* from 0 in the present

TABLE 6-12
Excitation Table

Inputs of Combinational Circuit					Outputs of Combinational Circuit			
Present State		Input	Next State		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
1	1	1	0	0	X	1	X	1

state to 1 in the next state. From Table 6.10(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 6-10(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 6-12. We know that a sequential circuit consists of a number of flip-flops and a combinational circuit. Figure 6-24 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

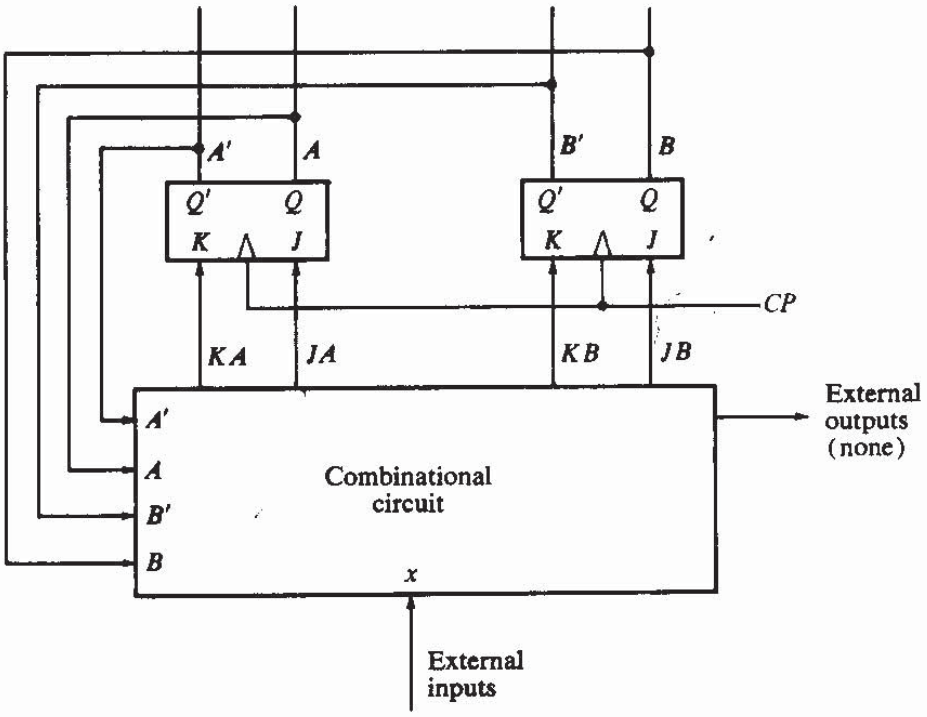


FIGURE 6-24
Block diagram of sequential circuit

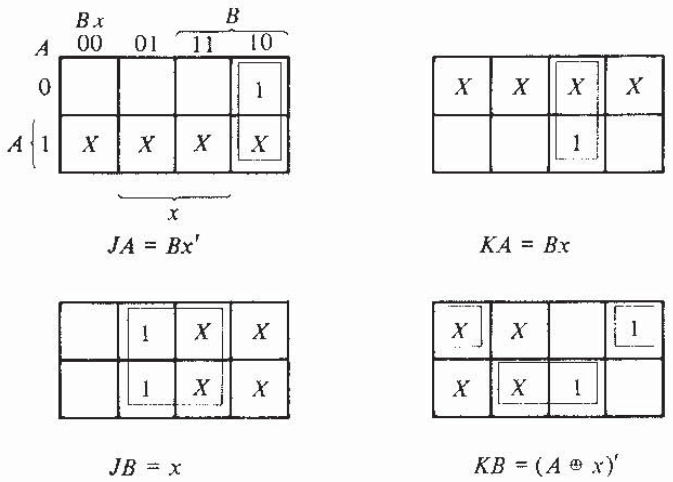


FIGURE 6-25
Maps for combinational circuit

KB. The information from the truth table is transferred into the maps of Fig. 6-25, where the four simplified flip-flop input functions are derived:

$$JA = Bx' \qquad KA = Bx$$

$$JB = x \qquad KB = (A \oplus x)'$$

The logic diagram is drawn in Fig. 6-26 and consists of two flip-flops, two AND gates, one exclusive-NOR gate, and one inverter.

The excitation table of a sequential circuit with m flip-flops, k inputs per flip-flop,

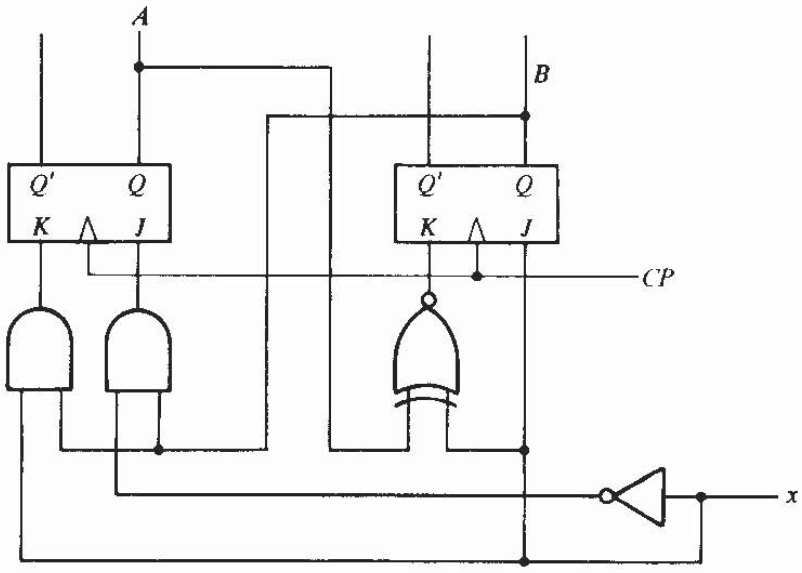


FIGURE 6-26
Logic diagram of sequential circuit

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*.

Design with D Flip-Flops

The time it takes to design a sequential circuit that uses D flip-flops can be shortened if we utilize the fact that the next state of the flip-flop is equal to its D input prior to the application of a clock pulse. This is shown in the excitation table of the D flip-flop listed in Table 6-10(c). The excitation table clearly shows that $D = Q(t + 1)$, which means that the next-state values in the state table specify the D input conditions directly, so there is no need for an excitation table as required with other types of flip-flops.

The design procedure with D flip-flops will be demonstrated by means of an example. We wish to design a clocked sequential circuit that operates according to the state table shown in Table 6-13. This table is the same as the state table part of Table 6-12 except for an additional column that includes an output y. For this case, it is not necessary to include the excitation table for flip-flop inputs DA and DB since $DA = A(t + 1)$ and $DB = B(t + 1)$. The flip-flop input functions can be obtained directly from the next-state columns of A and B and expressed in sum of minterms as follows:

$$DA(A, B, x) = \Sigma\,(2, 4, 5, 6)$$
$$DB(A, B, x) = \Sigma\,(1, 3, 5, 6)$$
$$y(A, B, x) = \Sigma\,(1, 5)$$

TABLE 6-13
State Table for Design with *D* Flip-Flops

Present State		Input	Next State		Output
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	0
1	1	1	0	0	0

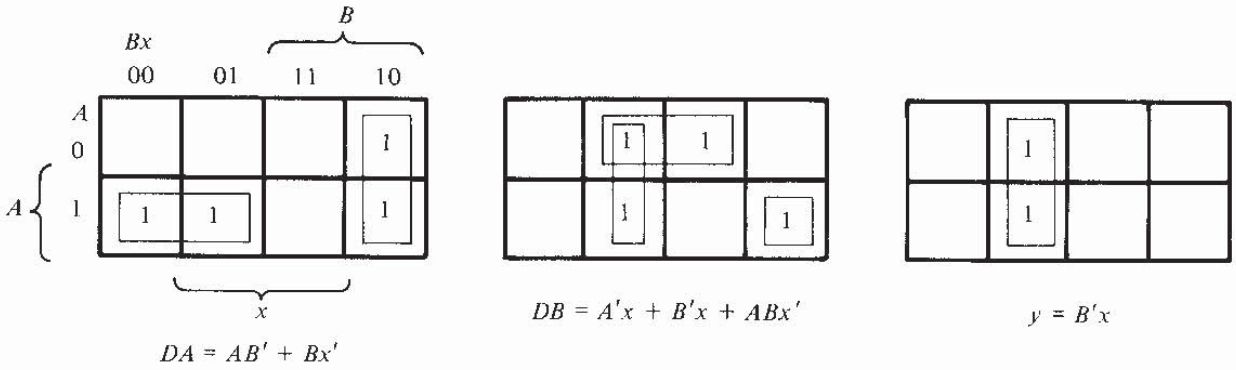


FIGURE 6-27
Maps for input functions and output *y*

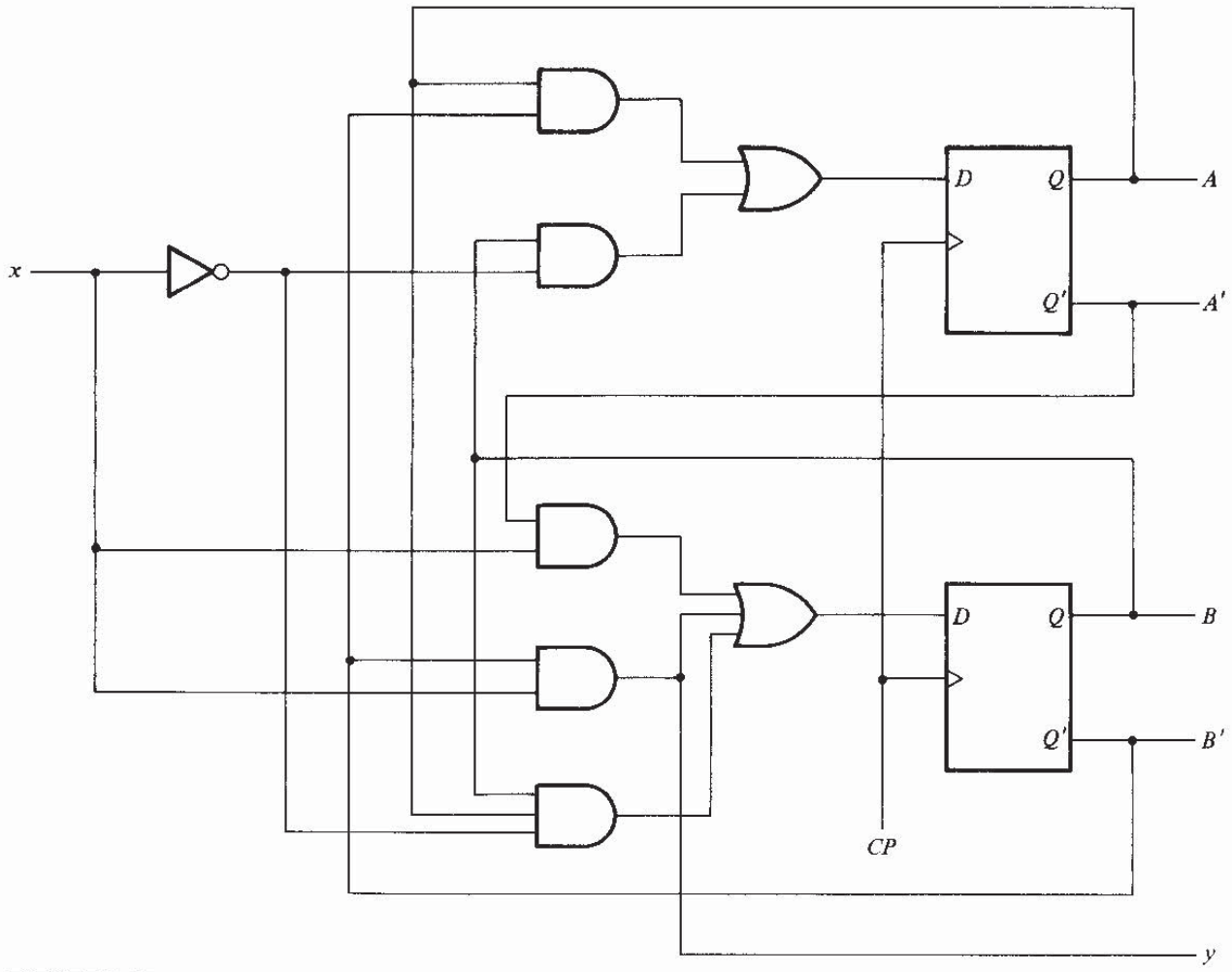


FIGURE 6-28
Logic diagram of a sequential circuit with *D* flip-flops

where A and B are the present-state values of flip-flops A and B, x is the input, and DA and DB are the input functions. The minterms for output y are obtained from the output column in the state table.

The Boolean functions are simplified by means of the maps plotted in Fig. 6-27. The simplified functions are

$$DA = AB' + Bx'$$
$$DB = A'x + B'x + ABx'$$
$$y = B'x$$

The logic diagram of the sequential circuit is shown in Fig. 6-28.

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.

Consider the state table shown in Table 6-14. There are five states listed in the table: 001, 010, 011, 100, and 101. The other three states, 000, 110, and 111, are not used. When an input of 0 or 1 is included with these unused states, we obtain six 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 conditions.

The state table is extended into an excitation table with *RS* flip-flops. The flip-flop input conditions are derived from the present-state and next-state values of the state table. Since *RS* flip-flops are used, we need to refer to Table 6-10(a) for the excitation

TABLE 6-14
State Table with Unused States

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

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 of the sequential circuit.

The combinational-circuit part of the sequential circuit is simplified in the maps of Fig. 6-29. 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.

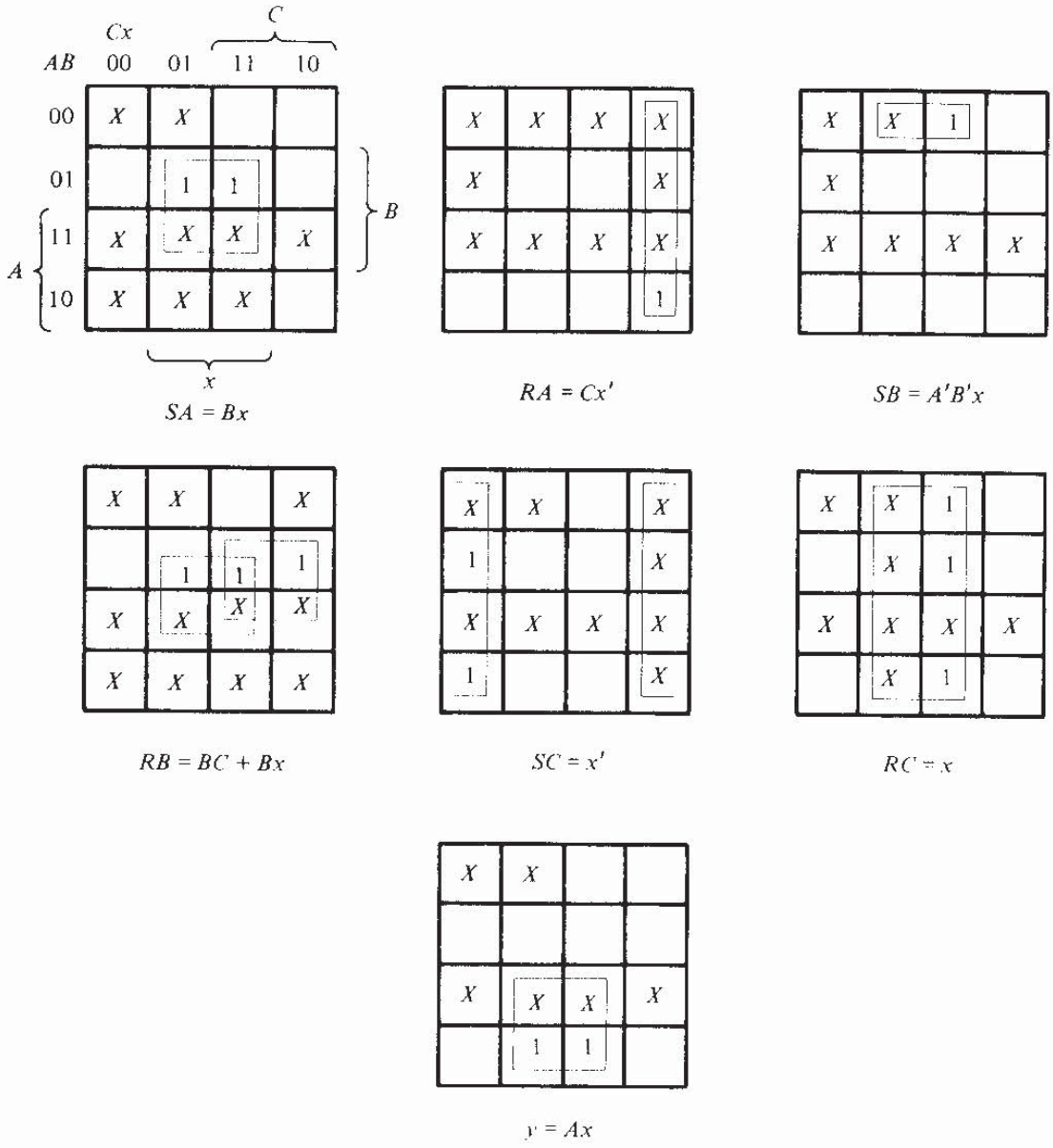


FIGURE 6-29
Maps for simplifying the sequential circuit

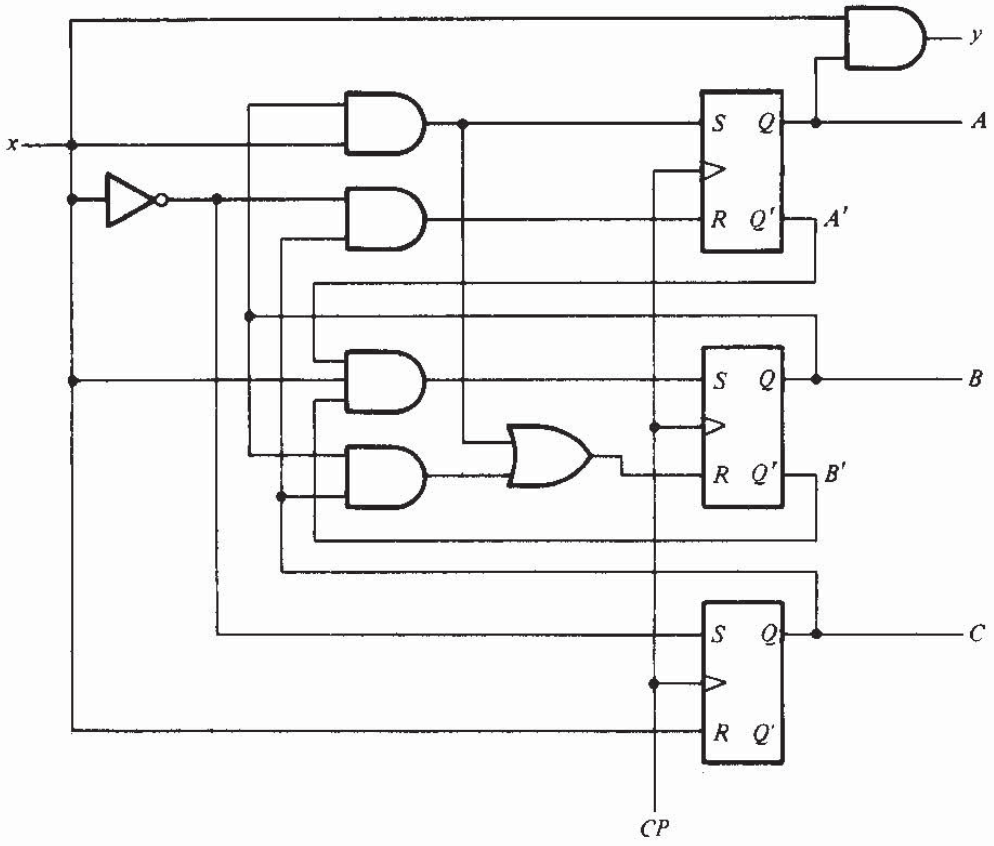


FIGURE 6-30
Logic diagram with *RS* flip-flops

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 shown in Fig. 6-30.

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. 6.30 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? 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 in-

tended 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 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.

Analysis of Previously Designed Circuit

We wish to analyze the sequential circuit of Fig. 6-30 to determine whether it operates according to the original state table and also determine the effect of the unused states on the circuit operation. The unused states are 000, 110, and 111. The analysis of the circuit can be done by the method outlined in Section 6-4. The maps of Fig. 6-29 may also help in the analysis. What is needed here is to start with the circuit diagram of Fig. 6-30 and derive the state table or diagram. If the derived state table is identical to the state-table part of Table 6-14, 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. 6-29 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 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 output 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. 6-31. The circuit operates as intended, as long as it stays within the states 001, 010, 011, 100, 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-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

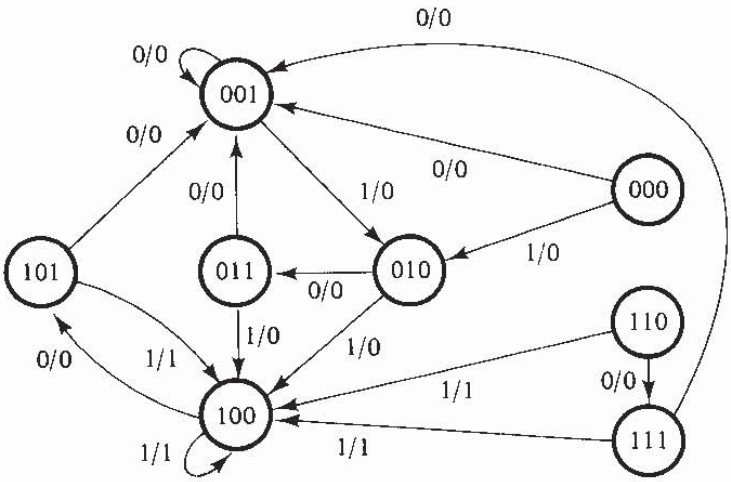


FIGURE 6-31
State diagram for the circuit of Fig. 6-30

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*. 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. 6-32. 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-

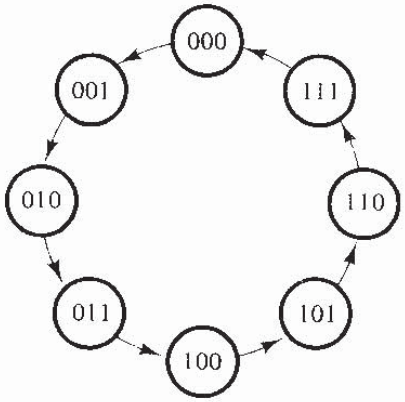


FIGURE 6-32
State diagram of a 3-bit binary counter

flops. The next state of a counter depends entirely on its present state, and the state transition occurs every time the pulse occurs.

Table 6-15 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-flops with J and K tied 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 of the present state to the 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

TABLE 6-15
Excitation Table for 3-Bit Counter

Present State			Next State			Flip-Flop Inputs		
A_2	A_1	A_0	A_2	A_1	A_0	TA_2	TA_1	TA_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

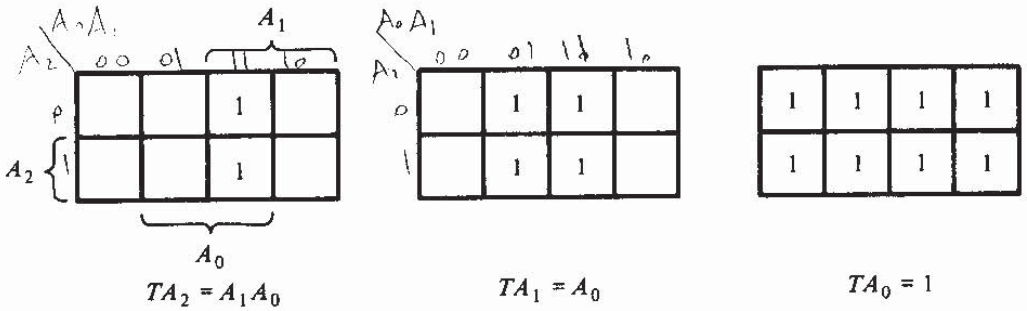


FIGURE 6-33
Maps for a 3-bit binary counter

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. 6-33. 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. 6-34.

Counter with Nonbinary Sequence

A counter with n flip-flops may have a binary sequence of less than 2^n states. A BCD counter counts the binary states from 0000 to 1001 and returns to 0000 to repeat the sequence. Other counters may follow an arbitrary sequence that may not be the straight binary sequence. In any case, the design procedure is the same. The state table is obtained from the count sequence and the counter is designed using sequential-circuit design techniques. As an example, consider the counter specified in Table 6-16. The count has a repeated sequence of six states, with flip-flops B and C repeating the binary count 00, 01, 10, while flip-flop A alternates between 0 and 1 every three counts. The count sequence is not straight binary and two states, 011 and 111, are not included in the count. The choice of JK flip-flops results in the flip-flop input conditions listed in

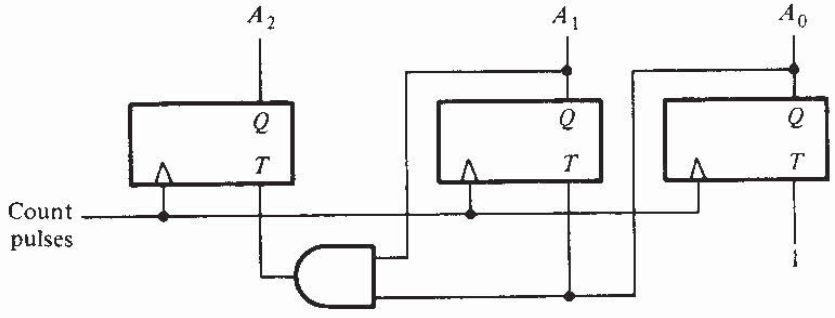


FIGURE 6-34
Logic diagram of a 3-bit binary counter

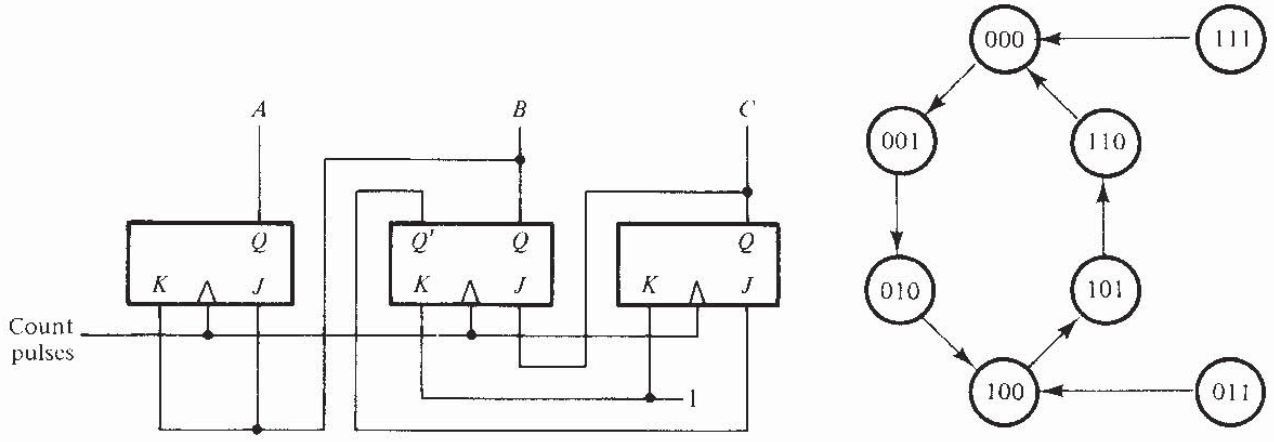
TABLE 6-16
Excitation Table for Counter

Present State			Next State			Flip-Flop Inputs					
A	*B*	*C*	*A*	*B*	*C*	*JA*	*KA*	*JB*	*KB*	*JC*	*KC*
0	0	0	0	0	1	0	*X*	0	*X*	1	*X*
0	0	1	0	1	0	0	*X*	1	*X*	*X*	1
0	1	0	1	0	0	1	*X*	*X*	1	0	*X*
1	0	0	1	0	1	*X*	0	0	*X*	1	*X*
1	0	1	1	1	0	*X*	0	1	*X*	*X*	1
1	1	0	0	0	0	*X*	1	*X*	1	0	*X*

the table. Inputs *KB* and *KC* have only 1's and *X*'s in their columns, so these inputs are always equal to 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. 6-35(a). Since there are two unused states, we analyze the circuit to determine their effect. If the circuit happens to be in state 011 because of an error signal, the circuit goes to state 100 after the application



(a) Logic diagram of counter

(b) State diagram of counter

FIGURE 6-35
Logic and state diagrams

of a clock pulse. This is obtained by noting that while the circuit is in present state 011, the outputs of the flip-flops are $A = 0$, $B = 1$, and $C = 1$. From the flip-flop input functions, we obtain $JA = KA = 1$, $JB = KB = 1$, $JC = 0$, and $KC = 1$. Therefore, flip-flop A is complemented and goes to 1. Flip-flop B is also complemented and goes to 0. Flip-flop C is reset to 0 because $KC = 1$. This results in next state 100. In a similar manner, we can evaluate the next state from present state 111 to be 000.

The state diagram including the unused states is shown in Fig. 6-35(b). If the circuit ever goes to one of the unused states because of an error, the next count pulse transfers it to one of the valid states and the circuit continues to count correctly. Thus, the counter is self-correcting. A self-correcting counter is one that if it happens to be in one of the unused states, it eventually reaches the normal count sequence after one or more clock pulses.

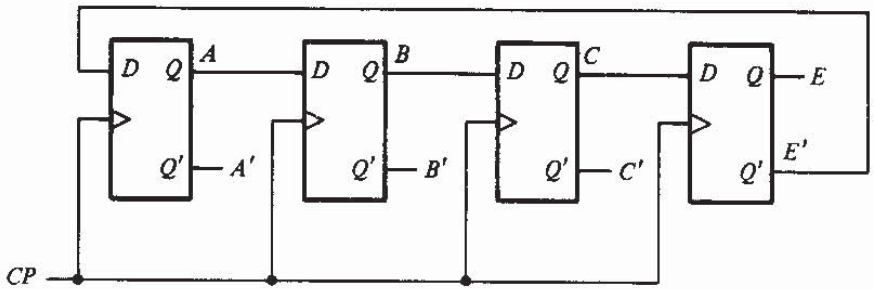
REFERENCES

1. MANO, M. M., *Computer Engineering: Hardware Design*. Englewood Cliffs, NJ: Prentice-Hall, 1988.
2. KOHAVI, Z., *Switching and Automata Theory*, 2nd Ed. New York: McGraw-Hill, 1978.
3. HILL, F. J., and G. R. PETERSON, *Introduction to Switching Theory and Logical Design*, 3rd Ed. New York: John Wiley, 1981.
4. ROTH, C. H., *Fundamentals of Logic Design*, 3rd Ed. New York: West, 1985.
5. SHIVA, S. G., *Introduction to Logic Design*. Glenview, IL: Scott, Foresman, 1988.
6. MCCLUSKEY, E. J., *Logic Design Principles*. Englewood Cliffs, NJ: Prentice-Hall, 1986.
7. BREEDING, K. J., *Digital Design Fundamentals*. Englewood Cliffs, NJ: Prentice-Hall, 1989.
8. ERCEGOVAC, M. D., and T. Lang, *Digital Systems and Hardware/Firmware Algorithms*. New York: John Wiley, 1985.
9. MANGE, D., *Analysis and Synthesis of Logic Systems*. Norwood, MA: Artech House, 1986.
10. DIETMEYER, D. L., *Logic Design of Digital Systems*. Boston: Allyn and Bacon, 1988.

PROBLEMS

6-1 Construct a D flip-flop that has the same characteristics as the one shown in Fig. 6-5, but instead of using NAND gates, use NOR and AND gates. (Remember that a one-input NOR gate is equivalent to an inverter.)

6-2 Construct a D flip-flop that has the same characteristics as the one shown in Fig. 6-5, but instead of using NAND gates, use NOR gates.



(a) Four-stage switch-tail ring counter

Sequence number	Flip-flop outputs A	B	C	E	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

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 RANDOM-ACCESS MEMORY (RAM)

A memory unit is a collection of storage cells together with associated circuits needed to transfer information in and out of the device. Memory cells can be accessed for information transfer to or from any desired random location and hence the name *random-access memory*, abbreviated RAM.

A memory unit stores binary information in groups of bits called *words*. A word in

memory is an entity of bits that move in and out of storage as a unit. A memory word is a group of 1's and 0's and may represent a number, an instruction, one or more alphanumeric characters, or any other binary-coded information. A group of eight bits is called a *byte*. Most computer memories use words that are multiples of 8 bits in length. Thus, a 16-bit word contains two bytes, and a 32-bit word is made up of four bytes. The capacity of a memory unit is usually stated as the total number of bytes that it can store.

The communication between a memory and its environment is achieved through data input and output lines, address selection lines, and control lines that specify the direction of transfer. A block diagram of the memory unit is shown in Fig. 7-24. The n data input lines provide the information to be stored in memory and the n data output lines supply the information coming out of memory. The k address lines specify the particular word chosen among the many available. The two control inputs specify the direction of transfer desired: The write input causes binary data to be transferred into the memory, and the read input causes binary data to be transferred out of memory.

The memory unit is specified by the number of words it contains and the number of bits in each word. The address lines select one particular word. Each word in memory is assigned an identification number, called an address, starting from 0 and continuing with 1, 2, 3, up to $2^k - 1$, where k is the number of address lines. The selection of a specific word inside the memory is done by applying the k-bit binary address to the address lines. A decoder inside the memory accepts this address and opens the paths needed to select the word specified. Computer memories may range from 1024 words, requiring an address of 10 bits, to 2^{32} words, requiring 32 address bits. It is customary to refer to the number of words (or bytes) in a memory with one of the letters K (kilo), M (mega), or G (giga). K is equal to 2^{10}, M is equal to 2^{20}, and G is equal to 2^{30}. Thus, $64K = 2^{16}$, $2M = 2^{21}$, and $4G = 2^{32}$.

Consider, for example, the memory unit with a capacity of 1K words of 16 bits each. Since $1K = 1024 = 2^{10}$ and 16 bits constitute two bytes, we can say that the memory can accommodate $2048 = 2K$ bytes. Figure 7-25 shows the possible content of the first three and the last three words of this memory. Each word contains 16 bits,

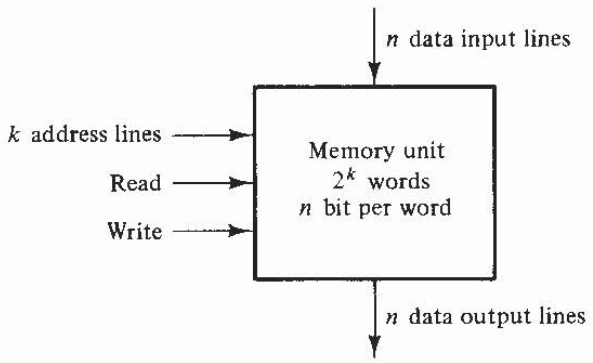


FIGURE 7-24
Block diagram of a memory unit

Memory address		Memory content
Binary	decimal	
0000000000	0	1011010101011101
0000000001	1	1010101110001001
0000000010	2	0000110101000110
⋮	⋮	⋮
1111111101	1021	1001110100010100
1111111110	1022	0000110100011110
1111111111	1023	1101111000100101

FIGURE 7-25
Content of a 1024 × 16 memory

which can be divided into two bytes. The words are recognized by their decimal address from 0 to 1023. The equivalent binary address consists of 10 bits. The first address is specified with ten 0's, and the last address is specified with ten 1's. This is because 1023 in binary is equal to 1111111111. A word in memory is selected by its binary address. When a word is read or written, the memory operates on all 16 bits as a single unit.

The 1K × 16 memory of Fig. 7-25 has 10 bits in the address and 16 bits in each word. As another example, a 64K × 10 memory will have 16 bits in the address (since $64K = 2^{16}$) and each word will consist of 10 bits. The number of address bits needed in a memory is dependent on the total number of words that can be stored in the memory and is independent of the number of bits in each word. The number of bits in the address is determined from the relationship $2^k = m$, where m is the total number of words, and k is the number of address bits.

Write and Read Operations

The two operations that a random-access memory can perform are the write and read operations. The write signal specifies a transfer-in operation and the read signal specifies a transfer-out operation. On accepting one of these control signals, the internal circuits inside the memory provide the desired function. The steps that must be taken for the purpose of transferring a new word to be stored into memory are as follows:

1. Transfer the binary address of the desired word to the address lines.
2. Transfer the data bits that must be stored in memory to the data input lines.
3. Activate the *write* input.

TABLE 7-7
Control Inputs to Memory Chip

Memory Enable	Read/Write	Memory Operation
0	*X*	None
1	0	Write to selected word
1	1	Read from selected word

The memory unit will then take the bits from the input data lines and store them in the word specified by the address lines.

The steps that must be taken for the purpose of transferring a stored word out of memory are as follows:

1. Transfer the binary address of the desired word to the address lines.
2. Activate the *read* input.

The memory unit will then take the bits from the word that has been selected by the address and apply them to the output data lines. The content of the selected word does not change after reading.

Commercial memory components available in integrated-circuit chips sometimes provide the two control inputs for reading and writing in a somewhat different configuration. Instead of having separate read and write inputs to control the two operations, some integrated circuits provide two other control inputs: one input selects the unit and the other determines the operation. The memory operations that result from these control inputs are specified in Table 7-7.

The memory enable (sometimes called the chip select) is used to enable the particular memory chip in a multichip implementation of a large memory. When the memory enable is inactive, the memory chip is not selected and no operation is performed. When the memory enable input is active, the read/write input determines the operation to be performed.

Types of Memories

The mode of access of a memory system is determined by the type of components used. In a random-access memory, the word locations may be thought of as being separated in space, with each word occupying one particular location. In a sequential-access memory, the information stored in some medium is not immediately accessible, but is available only at certain intervals of time. A magnetic-tape unit is of this type. Each memory location passes the read and write heads in turn, but information is read out only when the requested word has been reached. The *access time* of a memory is the time required to select a word and either read or write it. In a random-access memory, the access time is always the same regardless of the particular location of the word. In a sequential-access memory, the time it takes to access a word depends on the position of the word with respect to the reading-head position and therefore, the access time is variable.

Integrated-circuit RAM units are available in two possible operating modes, *static* and *dynamic*. The static RAM consists essentially of internal flip-flops that store the binary information. The stored information remains valid as long as power is applied to the unit. The dynamic RAM stores the binary information in the form of electric charges that are applied to capacitors. The capacitors are provided inside the chip by MOS transistors. The stored charge on the capacitors tends to discharge with time and the capacitors must be periodically recharged by *refreshing* the dynamic memory. Refreshing is done by cycling through the words every few milliseconds to restore the decaying charge. Dynamic RAM offers reduced power consumption and larger storage capacity in a single memory chip, but static RAM is easier to use and has shorter read and write cycles.

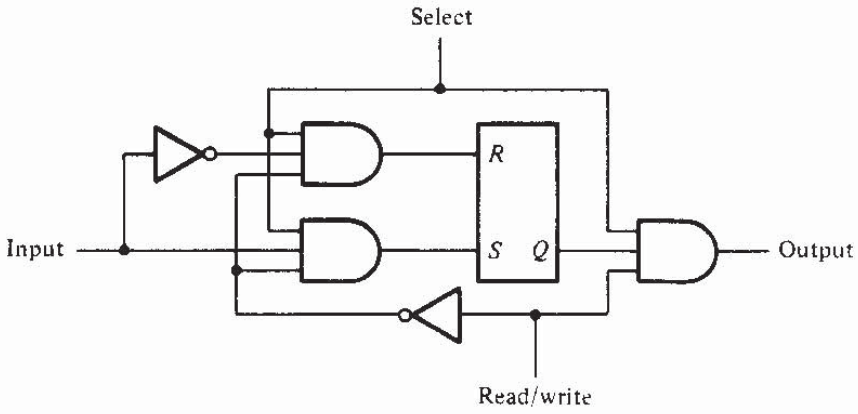
Memory units that lose the stored information when power is turned off are said to be *volatile*. Integrated-circuit RAMs, both static and dynamic, are of this category since the binary cells need external power to maintain the stored information. In contrast, a nonvolatile memory, such as magnetic disk, retains its stored information after removal of power. This is because the data stored on magnetic components is manifested by the direction of magnetization, which is retained after power is turned off. Another nonvolatile memory is the read-only memory (ROM) discussed in Section 5-7. A nonvolatile property is desirable in digital computers to store programs that are needed while the computer is in operation. Programs and data that cannot be altered are stored in ROM. Other large programs are maintained on magnetic disks. When power is turned on, the computer can use the programs from ROM. The other programs residing on disks can be transferred into the computer RAM as needed. Before turning the power off, the user transfers the binary information from the computer RAM into a disk if this information must be retained.

7-8 MEMORY DECODING

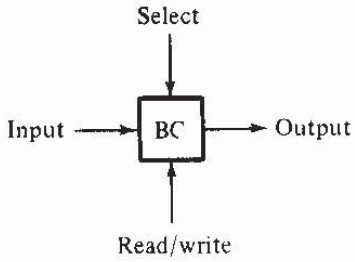
In addition to the storage components in a memory unit, there is a need for decoding circuits to select the memory word specified by the input address. In this section, we present the internal construction of a random-access memory and demonstrate the operation of the decoder. To be able to include the entire memory in one diagram, the memory unit presented here has a small capacity of 12 bits arranged in 4 words of 3 bits each. In addition to internal decoders, a memory unit may also need external decoders. This happens when integrated-circuit RAM chips are connected in a multichip memory configuration. The use of an external decoder to provide a large capacity memory will be demonstrated by means of an example.

Internal Construction

The internal construction of a random-access memory of m words with n bits per word consists of $m \times n$ binary storage cells and associated decoding circuits for selecting individual words. The binary storage cell is the basic building block of a memory unit.



(a) Logic diagram



(b) Block diagram

FIGURE 7-26
Memory cell

The equivalent logic of a binary cell that stores one bit of information is shown in Fig. 7-26. Although the cell is shown to include gates and a flip-flop, internally, it is constructed with two transistors having multiple inputs. A binary storage cell must be very small in order to be able to pack as many cells as possible in the area available in the integrated-circuit chip. The binary cell stores one bit in its internal flip-flop. It has three inputs and one output. The select input enables the cell for reading or writing and the read/write input determines the cell operation when it is selected. A 1 in the read/write input provides the read operation by forming a path from the flip-flop to the output terminal. A 0 in the read/write input provides the write operation by forming a path from the input terminal to the flip-flop. Note that the flip-flop operates without a clock and is similar to an *SR* latch (see Fig. 6-2).

The logical construction of a small RAM is shown in Fig. 7-27. It consists of 4 words of 3 bits each and has a total of 12 binary cells. Each block labeled *BC* represents the binary cell with its three inputs and one output, as specified in Fig. 7-26(b). A memory with four words needs two address lines. The two address inputs go through a 2×4 decoder to select one of the four words. The decoder is enabled with the memory-enable input. When the memory enable is 0, all outputs of the decoder are 0

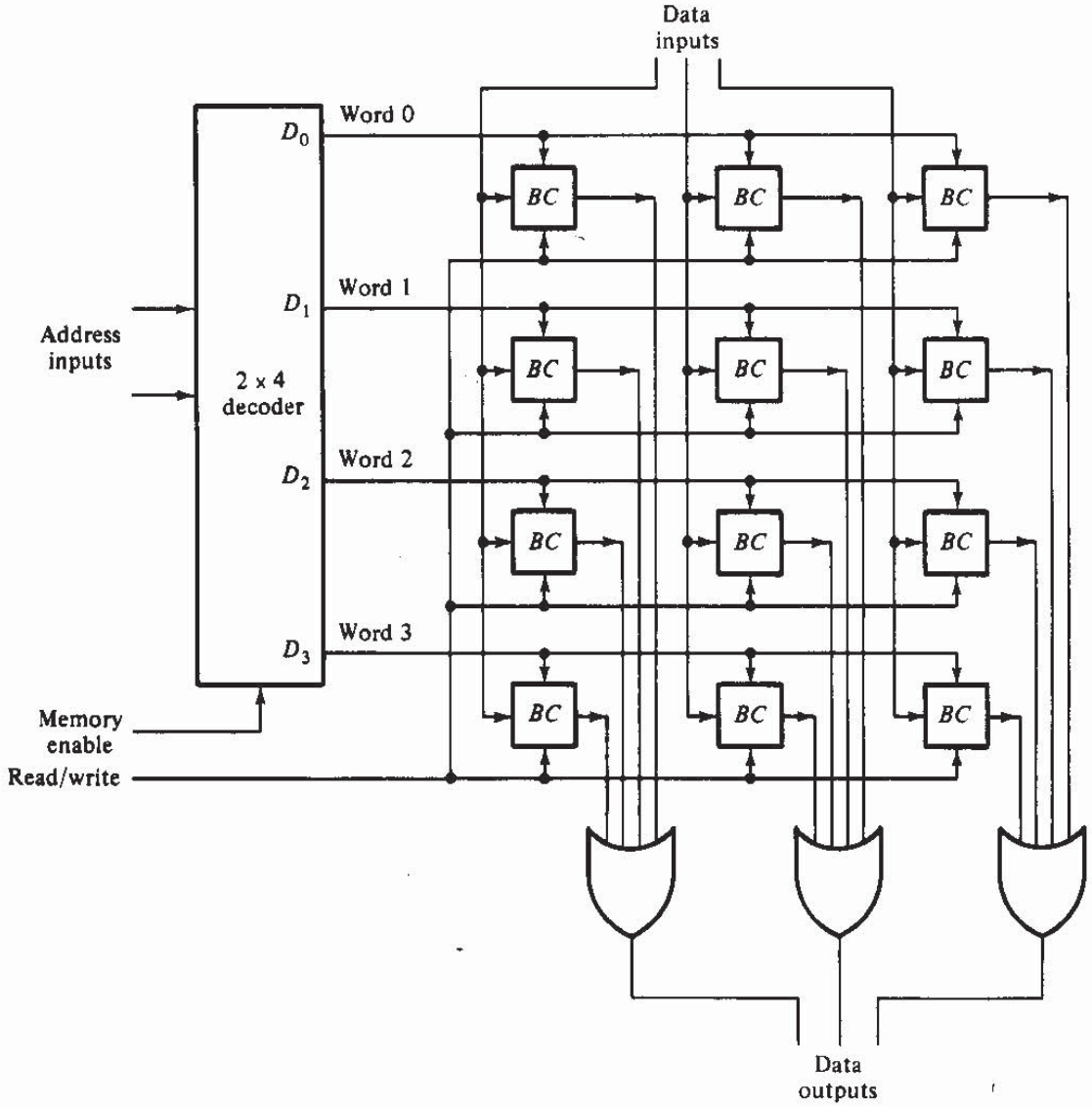


FIGURE 7-27
Logical construction of a 4 × 3 RAM

and none of the memory words are selected. With the memory enable at 1, one of the four words is selected, dictated by the value in the two address lines. Once a word has been selected, the read/write input determines the operation. During the read operation, the four bits of the selected word go through OR gates to the output terminals. During the write operation, the data available in the input lines are transferred into the four binary cells of the selected word. The binary cells that are not selected are disabled and their previous binary values remain unchanged. When the memory-enable input that goes into the decoder is equal to 0, none of the words are selected and the contents of all cells remain unchanged regardless of the value of the read/write input.

Commercial random-access memories may have a capacity of thousands of words and each word may range from 1 to 64 bits. The logical construction of a large capacity memory would be a direct extension of the configuration shown here. A memory with 2^k words of n bits per word requires k address lines that go into a $k \times 2^k$ decoder. Each one of the decoder outputs selects one word of n bits for reading or writing.

Array of RAM Chips

Integrated-circuit RAM chips are available in a variety of sizes. If the memory unit needed for an application is larger than the capacity of one chip, it is necessary to combine a number of chips in an array to form the required memory size. The capacity of the memory depends on two parameters: the number of words and the number of bits per word. An increase in the number of words requires that we increase the address length. Every bit added to the length of the address doubles the number of words in memory. The increase in the number of bits per word requires that we increase the length of the data input and output lines, but the address length remains the same.

To demonstrate with an example, let us first introduce a typical RAM chip, as shown in Fig. 7-28. The capacity of the RAM is 1024 words of 8 bits each. It requires a 10-bit address and 8 input and output lines. These are shown in the block diagram by a single line and a number indicating the total number of inputs or outputs. The chip-select (*CS*) input selects the particular RAM chip and the read/write (RW) input specifies the read or write operation when the chip is selected.

Suppose that we want to increase the number of words in the memory by using two or more RAM chips. Since every bit added to the address doubles the binary number that can be formed, it is natural to increase the number of words in factors of 2. For example, two RAM chips will double the number of words and add one bit to the composite address. Four RAM chips multiply the number of words by 4 and add two bits to the composite address.

Consider the possiblity of constructing a 4K × 8 RAM with four 1K × 8 RAM chips. This is shown in Fig. 7-29. The 8 input data lines go to all the chips. The outputs must be ORed together to form the common 8 output data lines. (The OR gates are not shown in the diagram.) The 4K word memory requires a 12-bit address. The 10 least significant bits of the address are applied to the address inputs of all four chips.

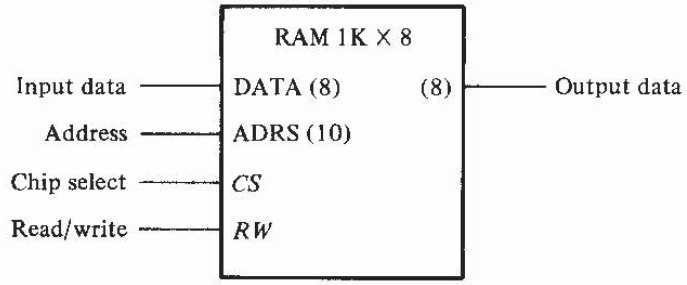


FIGURE 7-28
Block diagram of a 1K × 8 RAM chip.

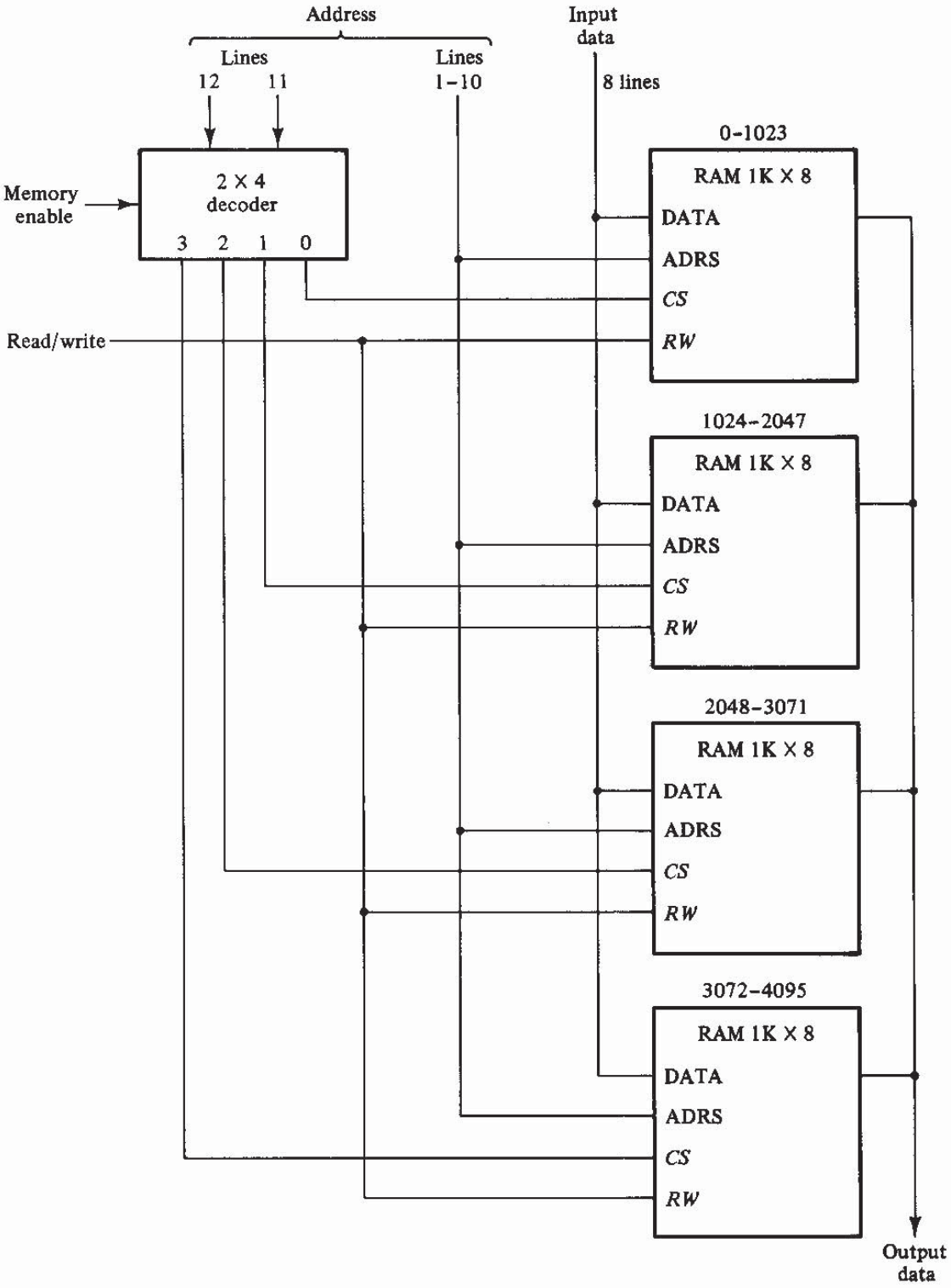


FIGURE 7-29
Block diagram of a 4K × 8 RAM.

The other two most significant bits are applied to a 2 × 4 decoder. The four outputs of the decoder are applied to the *CS* inputs of each chip. The memory is disabled when the memory-enable input of the decoder is equal to 0. This causes all four outputs of the decoder to be in the 0 state and none of the chips are selected. When the decoder is enabled, address bits 12 and 11 determine the particular chip that is selected. If bits 12 and 11 are equal to 00, the first RAM chip is selected. The remaining ten address bits select a word within the chip in the range from 0 to 1023. The next 1024 words are selected from the second RAM chip with a 12-bit address that starts with 01 and follows by the ten bits from the common address lines. The address range for each chip is listed in decimal over its block diagram in Fig. 7-29.

It is also possible to combine two chips to form a composite memory containing the same number of words but with twice as many bits in each word. Figure 7-30 shows the interconnection of two 1K × 8 chips to form a 1K × 16 memory. The 16 input and output data lines are split between the two chips. Both receive the same 10-bit address and the common *CS* and *RW* control inputs.

The two techniques just described may be combined to assemble an array of identical chips into a large-capacity memory. The composite memory will have a number of bits per word that is a multiple of that for one chip. The total number of words will increase in factors of 2 times the word capacity of one chip. An external decoder is needed to

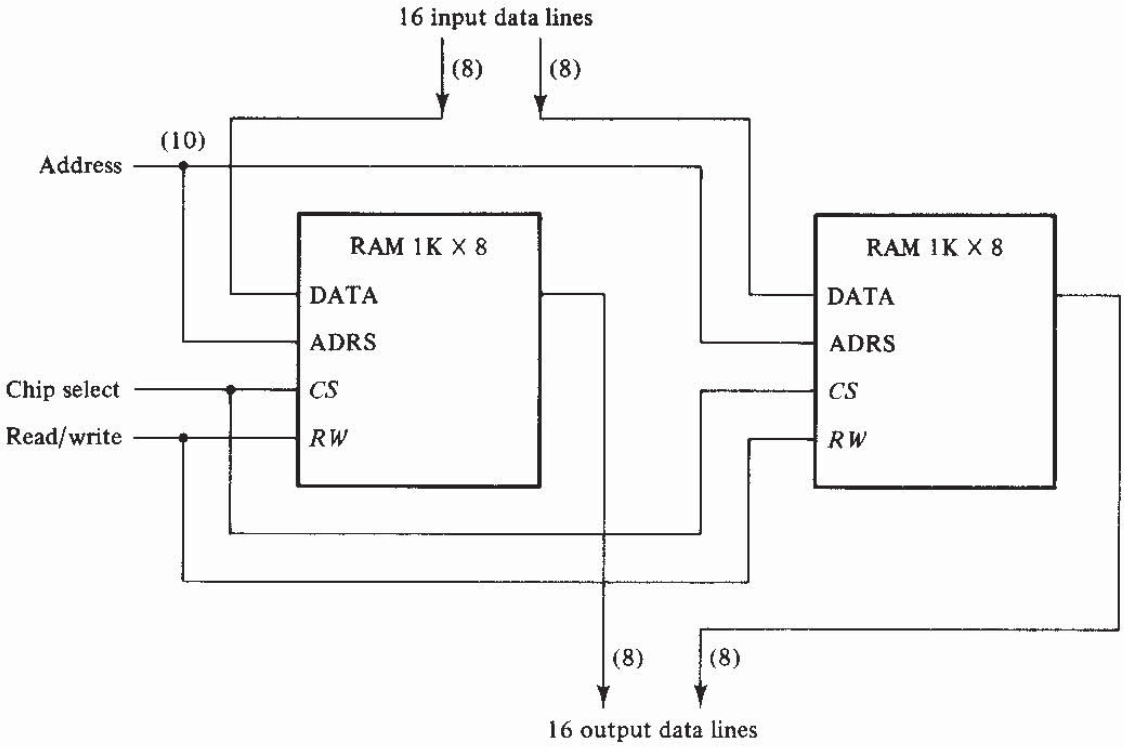


FIGURE 7-30
Block diagram of a 1K × 16 RAM.

select the individual chips from the additional address bits of the composite memory.

To reduce the number of pins in the package, many RAM integrated circuits provide common terminals for the input data and output data. The common terminals are said to be *bidirectional,* which means that for the read operation, they act as outputs, and for the write operation, they act as inputs.

7-9 ERROR-CORRECTING CODE

The complexity level of a memory array may cause occasional errors in storing and retrieving the binary information. The reliability of a memory unit may be improved by employing error-detecting and correcting codes. The most common error-detection scheme is the parity bit. (See Section 4-9.) A parity bit is generated and stored along with the data word in memory. The parity of the word is checked after reading it from memory. The data word is accepted if the parity sense is correct. If the parity checked results in an inversion, an error is detected, but it cannot be corrected.

An error-correcting code generates multiple check bits that are stored with the data word in memory. Each check bit is a parity over a group of bits in the data word. When the word is read from memory, the associated parity bits are also read from memory and compared with a new set of check bits generated from the read data. If the check bits compare, it signifies that no errror has occurred. If the check bits do not compare with the stored parity, they generate a unique pattern, called a *syndrome,* that can be used to identify the bit in error. A single error occurs when a bit changes in value from 1 to 0 or from 0 to 1 during the write or read operation. If the specific bit in error is identified, then the error can be corrected by complementing the erroneous bit.

Hamming Code

One of the most common error-correcting codes used in random-access memories was devised by R. W. Hamming. In the Hamming code, k parity bits are added to an n-bit data word, forming a new word of $n + k$ bits. The bit positions are numbered in sequence from 1 to $n + k$. Those positions numbered as a power of 2 are reserved for the parity bits. The remaining bits are the data bits. The code can be used with words of any length. Before giving the general characteristics of the code, we will illustrate its operation with a data word of eight bits.

Consider, for example, the 8-bit data word 11000100. We include four parity bits with the 8-bit word and arrange the 12 bits as follows:

Bit position:	1	2	3	4	5	6	7	8	9	10	11	12
	P_1	P_2	1	P_4	1	0	0	P_8	0	1	0	0

The four parity bits, P_1, P_2, P_4, and P_8, are in positions 1, 2, 4, and 8, respectively. The eight bits of the data word are in the remaining positions. Each parity bit is calculated as follows: