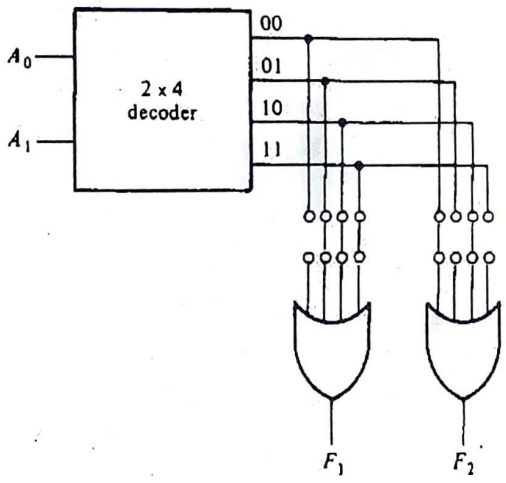
Combinational Logic Implementation

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.

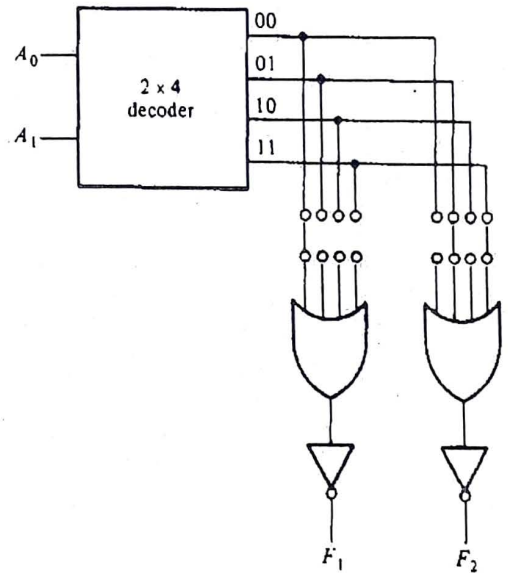
Let us clarify the process with a specific example. The truth table in Fig. 5-23(a) specifies a combinational circuit with two inputs and two outputs. The Boolean functions can be expressed in sum of minterms:

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.

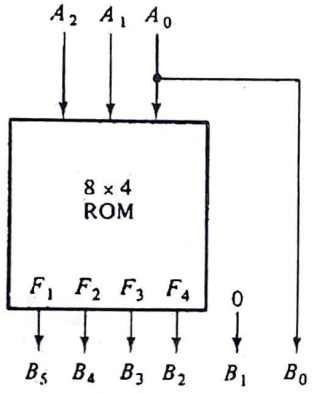
Example 5-3

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*.

Volatile Memory:

Any type of memory that requires the application of electrical power in order to store information. If the electrical power is removed, all information stored in the memory will be lost. Many semiconductor memories are volatile, while all magnetic memories are nonvolatile, which means that they can store information without electrical power. i.e. RAM.

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

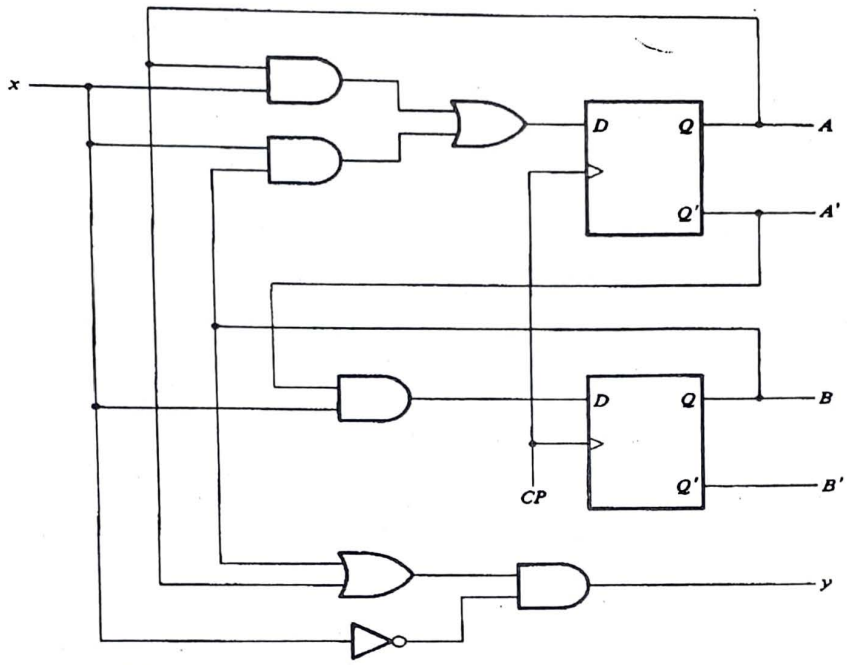


FIGURE 6-16
Example of a sequential circuit

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

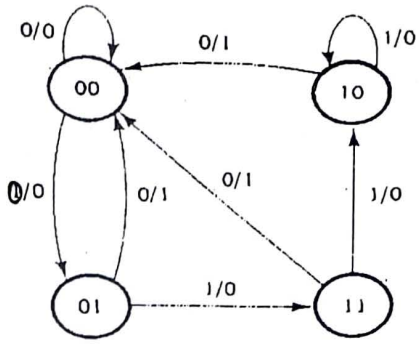


FIGURE 6-17
State diagram of the circuit of Fig. 6-16

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.

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.

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.

Ref. Book: Digital Logic and Computer Design - by Morris Mano.