Chapter 6

Synchronous Sequential Machines

A block diagram of a sequential circuit was shown in the introduction of Chapter 5. That chapter concentrated on one part of such a circuit: the memory devices, or flip-flops. Those circuits in which state transitions are controlled, or synchronized, by a clock are said to be *clocked,* or *synchronous,* sequential circuits. Other sequential circuits exist, called *asynchronous* circuits, in which state transitions are not synchronized by a clock. These are less common, although they do have important applications. We will postpone the discussion of such circuits to Chapter 7.[1]

A number of tools are used to describe the behavior of sequential logic circuits and to analyze and design them. We will introduce and develop such tools in this chapter. Included are formal procedures for the design of synchronous machines. Finally, we will concentrate on one class of such circuits and their design: circuits called *counters.*

1 BASIC CONCEPTS

The generic description of a problem requiring the design of a synchronous sequential logic circuit can be given as follows.

> *Design a digital circuit whose outputs are to take on specific values after a specific sequence of inputs has taken place.*

Such a problem statement is very broad. What is clear is that

- There are to be certain sequences of inputs to the circuit.

[1]With or without adjectives to qualify it, the term *machine* is often used to designate a sequential circuit, as in the title of this chapter. Because such circuits can have only a finite number of states, they have also been called *finite-state machines.* Since finiteness is all that is possible in the physical world, this adjective is often dropped and the circuits in question are simply called *state machines.* "Machine" normally has the connotation of something physical. However, in the present usage, the term refers to an abstract entity described by mathematical, graphical, or tabular means, as we will describe in this chapter.

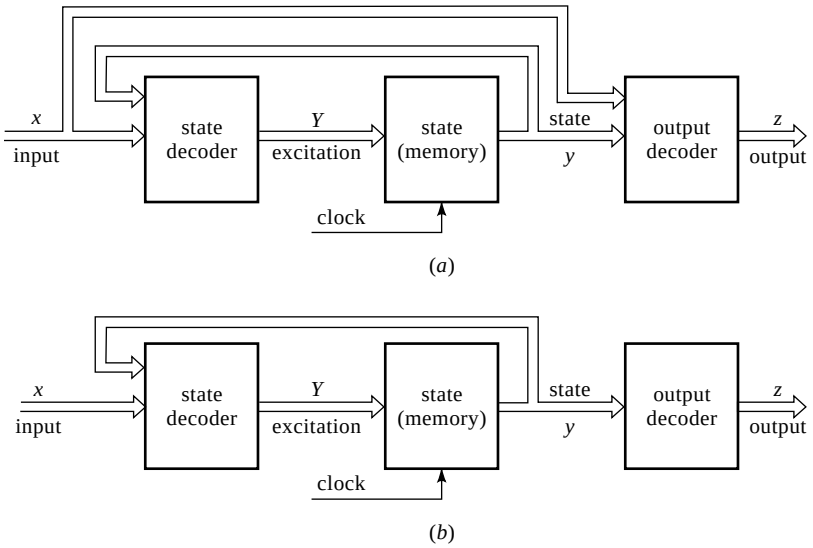


Figure 1 Models of Mealy and Moore machines.

- Different sequences of past inputs will constitute different states or conditions of the circuit.
- Specific outputs result only following a specified input sequence.

The general description does not specify whether the output will depend just on the past inputs or on the latest input as well.

These two possibilities lead to different structures, first explored in the mid-1950s by two people named Mealy and Moore. As mentioned in Chapter 5, in *Mealy machines* the outputs depend both on the present state (resulting from past inputs) and on the current input. The outputs in *Moore machines* depend only on the present state (resulting from past inputs). Attention will be devoted to both types.

Block diagrams of both Mealy and Moore models of sequential machines were given in Chapter 5, Figure 1. Somewhat refined versions are shown in Figure 1 here. The open arrowheads imply multiple variables. For example, input x stands for the set of variables $\{x_1, x_2, \ldots, x_n\}$. The combinational part of the circuit is broken down into two separate parts: the *state decoder* and the *output decoder.* The state decoders in both models accept as inputs both primary (external) inputs and the present state. In the Moore machine, however, the output decoder accepts only the present state to yield outputs. In the simplest case, there is no output decoder at all; the states themselves are the outputs. In a given machine, there may be some outputs that are Moore-type outputs and others that aren't. Such a machine must be classified as a *Mealy machine,* since at least *some* of its outputs depend not only on the state, but also on the inputs. Thus, the Mealy machine is the more general (and more common) type.

The behavior of synchronous sequential logic circuits can be described in a number of ways. At any given clock pulse, the state of the circuit is the *present state.* Signals present at the input terminals at that time are the *inputs.* This

Short ____
Even ____

combination of present state and input results in two things: a transition to the *next state* and an *output*. At the next clock pulse, the process is repeated, except that the present state is now what the next state was at the preceding clock pulse. It is possible to conceive of this process as never ending; that is, the "next state" is never a state that had been previously encountered. In this case, the machine would be infinite—a peculiar machine indeed. Barring such an unlikely event, somewhere along the line the next state will be a previously encountered state. After this, the machine will retrace its steps over and over; no new states will be encountered.

Several means are available for illustrating the following sequence:

present state/input → clock pulse → next state/output

One of these means is graphical/diagrammatic; another is tabular. A third approach utilizes a chart not unlike a flow chart describing an algorithm. All are treated in this chapter.

State Diagram

The graphical/diagrammatic tool for describing sequential circuit behavior is known as a *linear graph*.[2] For each state of the circuit, there is a corresponding node in the graph. (The circle representing the node is made large enough that the symbol for the state can be written inside.) With the machine in any one state (node in the graph), at the occurrence of a clock pulse, there will be a state transition to the next state and there will be an output, both in accordance with the problem statement. For a single-input machine, two lines emanate from each node, one each for a 0 and for a 1 input. For two input variables, four lines emanate from each node, one for each input combination: 00, 01, and so on. (How many lines will emanate from each node if the number of input variables is *n*?) Along each line we write the input value and the corresponding output separated by a slash. The resulting graph is called a *state diagram*.

For some state machines it is known from the statement of the problem just how many distinct states the machine has. However, in general, the number of possible states is initially unknown. To establish the state diagram, we arbitrarily choose an initial state and label it, say, A. (State names can be anything convenient.) A state is identified by unambiguously specifying how it is reached. To say, for example, that "state S is reached when the input is 1" is inadequate,[3] because the statement does not *unambiguously* identify it: Does this 1 follow another 1? Is it the first 1 after a string of 0's? Does this 1 follow a string of two or more preceding 1's? One unambiguous specification would be: "State S is reached by the second of two input 1's after one or more 0's."

Because it is difficult to describe, in the abstract, both the state diagram and the tabular tool to be discussed next, we will continue this discussion in conjunction with an example.

---

[2]A linear graph is a set of *nodes*, or *vertices* (drawn as circles), interconnected by a set of directed lines (or arcs), that is, lines that have an orientation indicated by an arrowhead.

[3]Except for a trivial case that will be described shortly.

___ Short

___ Even

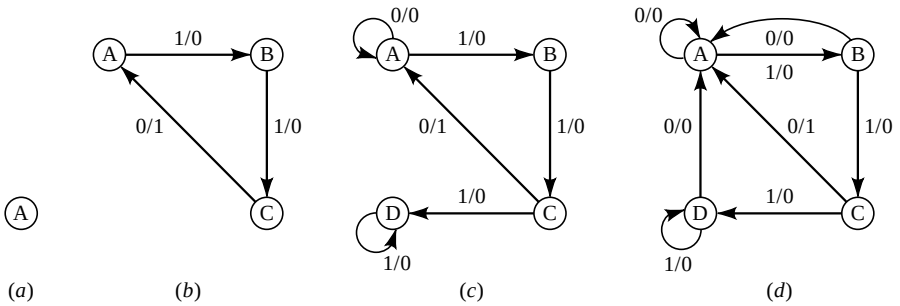


Figure 2 State diagram of sequence detector.

EXAMPLE 1

A synchronous sequential machine with a single input line x and a single output line z is to be designed. The specifications are as follows: the output is to be $z = 1$ if and only if the specific input sequence ...0110 occurs at consecutive clock pulses; otherwise $z = 0$. (Unless otherwise stated, the most recent bit of a sequence is the one on the right in all cases.) Suppose, for example, that an input sequence is ...0110110. The first 4 bits identified constitute an *acceptable* (that is, *output-producing*) input sequence. But the fourth bit starts another acceptable sequence that overlaps with the first. Hence, the output sequence will be ...0001001. (Confirm.) Such a machine is called a *sequence detector.*

Let's identify our initial state A as the state reached by an input $x = 0$ regardless of the preceding sequence of inputs. It doesn't matter if the preceding input bit is 1 or 0 for a 0 input bit to start an acceptable sequence. We start the state diagram by drawing a node labeled A. Following this, there are two possible strategies:

- We can explore the consequences (next state/output) resulting from each possible input starting in this state, and continue in this fashion with all the next states encountered along the way.
- We can assume an acceptable sequence and pursue the consequences (string of next states and outputs), adding states as needed. Then we return to each state encountered along the way to fill in the consequences of inputs that are not part of an acceptable sequence.

The second method is carried out in Figure 2. Starting from state A (reached by a 0 input), we assume an input sequence 110 to complete an acceptable sequence. The result is shown in Figure 2*b*. (Confirm that the two additional states shown must be introduced along the way. Also confirm each of the steps that follow.)

Starting at each state in Figure 2*b*, only one of the two possible inputs has been used so far. Now we fill in the other possibilities. From state A, an input of 0 leads back to state A/output 0. From state B, an input of 0 also leads back to A/output 0. But what is the next state if there is an input of 1 while the machine is in state C (which was reached by an input sequence 011)? It can't be to any of the three states reached so far (confirm this), so it must be to a new state D/output 0. The state diagram so far is shown in Figure 2*c*. Finally, from this new state, an input of 0 leads back to A/output 0 and an input of 1 leads back to D/output 0. The final diagram is shown in Figure 2*d*.

Short ___
Even ___

Study the last diagram. A is the state reached by the first bit in an acceptable sequence. The sequence is aborted if the next input is also 0. Now it is this last 0 that starts an acceptable sequence. Any number of additional inputs of 0 lead to the same result: the latest 0 becomes the first bit of an acceptable sequence. The last state, D, is an acceptable-sequence spoiler; it is reached by an input of 1 following a sequence ...011. ■

Notice in Figure 2 that all but one of the arcs in the graph are labeled with an output of 0. A lot of clutter could be avoided if we adopt the convention that only outputs of 1 will be shown explicitly. When the output associated with a particular arc is 0 (or 00, 000, etc. for more output variables), henceforth it will not be shown explicitly on the state diagram.

In constructing a state diagram, there are generally two major decision points:

1. Choosing the initial state
2. When in a particular state, deciding whether the transition resulting from a particular input is to an existing state or to a new state not yet identified

In some (not all) sequential machines there is a specific *reset* state; the machine must be in this state at the starting time. In such cases, the initial state is predetermined. When there is no reset state, the initial state is chosen arbitrarily, as in the preceding example. Although the problem statement might guide the choice of initial state, different designers might choose different initial states.[4] No problem. Assuming there are no mistakes, two state diagrams constructed with different initial states will be *isomorphic;* that is, they will become identical by an appropriate interchange of state names.[5]

As for the second decision point, a transition to a new state rather than to an existing state will result in more states in the state diagram. Eventually, a circuit must be implemented. Generally speaking, more states mean more flip-flops, though not proportionately more. A circuit with n flip-flops, for example, will have 2^n states. Conversely, then, eight (2^3) states will require three flip-flops. But even as few as five states will still require three flip-flops. Thus, if a state diagram has five states already, increasing the number of states to as many as eight will not increase the number of flip-flops needed. Thus, introducing more states in a state diagram may simply mean introducing redundancies; these might be removable later.

Although in the preceding example we constructed a state diagram that describes all the state transitions of the desired machine, we still did not complete a design. Before tackling that task, we turn to the tabular tool for describing the behavior of a sequential machine.[6]

[4]They may also give the states different names, the placement of the nodes might be different, and the curvatures of the lines joining the nodes might be different. All of these are trivial matters.

[5]This assumes that no extra states are introduced, as will be described shortly.

[6]Suppose we reconsider the statement "State S is reached when the input is 1." The state diagram will then consist of just two states: S is the state reached by an input 1 and, say, T is reached by an input 0. Any other 0 inputs while in state T will return the machine to state T. A 1 input will send the machine back to state S. Any other 1 inputs will keep the machine in state S. Construct a state diagram for yourself. This is a trivial "machine."

Short
Even

PS	NS,z $x = 0$	$x = 1$
A	A,0	B,0
B	A,0	C,0
C	A,1	D,0
D	A,0	D,0

Figure 3 State table of sequence detector in Example 1 obtained from its state diagram.

State Table

A table can be described by its row headings (or names) and its column headings. Its entries occur at the intersections of the rows and columns. To describe the operation of a synchronous machine, it is customary to choose the row headings as the present states and the column headings as the inputs.

Since two outcomes (next state, output) result from an input to the circuit when the circuit is in a particular state, it is conceivable to construct two separate tables. The entries in one of these would be the circuit outputs—hence it is called the *output table.* The entries in the other table would be the next states. Since the table is intended to show transitions from a present state to a next state, it might be tempting to call this table a state transition table. But in the preceding chapter, state transition table was the name given to the table that specifies the next state resulting from inputs to a flip-flop for each present state of the flip-flop. So we use a different name; it is called simply a *state table.* In the present usage, the circuit is not limited to a single flip-flop but encompasses an entire machine.

Remember from Figure 1*b* that the output of a Moore machine depends only on the present state. For such a machine, there will be only one output combination for each input combination; hence, a separate output table makes more sense. For a Mealy machine, on the other hand, we will combine the state and output tables into a single table in which the entries are both the next states and the resulting outputs, separated by a comma.[7] This will be illustrated for Example 1, whose state diagram was obtained in Figure 2. (Of course, once this table is available, it is always possible to separate the next-state part and the output part into two separate tables if there is a reason for doing so.)

Constructing a State Table from a State Diagram

Once a state diagram is available, the corresponding state/output table is easily constructed. The state diagram in Figure 2 has four states. Hence, the corresponding state/output table will have four rows. Starting in any state, the output and the next state can be obtained from the diagram and entered in the table. The result is given in Figure 3.

Exercise 1 Notice in Figure 2 that state A is reached from each state (including A) by an input 0 but with different outputs: 0 from states A, B, D and 1 from state C. Instead, let's suppose that state A in the state diagram of Figure 2 is

Short ___
Even ___

[7]It is also sometimes called a *flow table.*

identified as the state reached by an input 0 resulting in an output 1. Now start from Figure 2*a* and assume that a 0 input while in state A or state B leads to a new state E rather than back to A. Complete the resulting state diagram. (You don't need to take the following advice, but you won't get tangled up in crossing lines if you put A, B, and C in a row, with D under C and E under B.) Then, from the diagram, construct a state table.

Answer[8]

Examine states A and E in your table from Exercise 1. Both are reached by an input bit 0. Hence, starting from either state, the next states and outputs must be the same for each input bit. In this case, these two states can't be distinguished from each other. This is the basis for a definition:

> *Two states are said to be* indistinguishable *if, for each input combination, the resulting outputs and next states are the same.*

Actually, the next states need not be the same—only indistinguishable, as just defined.

On this basis, states A and E are indistinguishable. If all E next states in the table are replaced by A, and row E is eliminated, the table obtained in Exercise 1 will become the same table obtained in Figure 3.

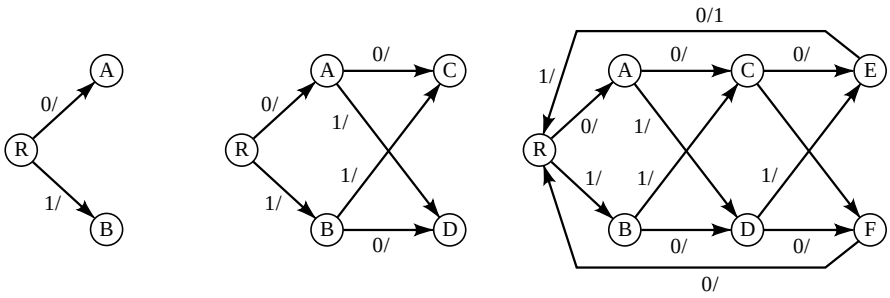
There are formal procedures for extending the concept just defined. We will pursue this generalization in the following section and discuss ways of reducing the number of rows of a state table. Consequently, in constructing a state diagram in the design of a sequential circuit, there is no great need to worry about introducing redundant states; such states can be removed subsequently. On the other hand, there is no point in needlessly extending a state table, since effort will be needed later to reduce it. When in doubt while constructing a state table, by all means introduce a new state. However, restrain yourself if you are certain that the relevant conditions have already been identified by an existing state.

EXAMPLE 2

(*Note:* One of the topics treated in section 4 of Chapter 1 is the Hamming code. Review it if you need a refresher.) To an n-bit message, an additional k bits are added, making the parity of the resulting $(n + k)$-bit string either odd or even—our choice. In this example, let $n = 3$ and $k = 1$; let's choose odd parity. Suppose that a 4-bit string is to be received; the first 3 bits constitute the message, and the fourth bit is always 0 (equivalent to a blank). If the number of 1's in the 3-bit message is odd, the parity bit is to remain 0. If the number of 1's is even, a 1 bit is to be generated

[8]Now, only one arc of the graph with input 1 enters A (from C/output 1). Four arcs enter E (from A, B, D, and E, all with 0 in, 0 out. D is a spoiler state, entered by a string of three 1's starting at A; any further 1's while in state D will keep the state in D. An input 1 while in state E is the first 1 after one or more 0's; in that respect, an input 1 while in state E should lead to the same state as a 1 while in state A, namely, state B. ◆

Short
Even



(*a*) (*b*) (*c*)

Figure 4 State diagram of parity-bit generator.

	NS,*z*	
PS	$x = 0$	$x = 1$
R	A,0	B,0
A	C,0	D,0
B	D,0	C,0
C	E,0	F,0
D	F,0	E,0
E	R,1	R,×
F	R,0	R,×

Figure 5 State table for parity-bit generator.

and inserted in the fourth position to make the parity of the 4-bit string odd. In either case, after the fourth bit, transition is to be to a reset state, in which the machine is ready to receive the next message sequence. Such a machine is a *parity-bit generator.* Our objective is to create a state table.

The first step is to create a state diagram. The machine is to be in the reset state (say R) when the first bit arrives. The partial state diagram after the first bit is shown in Figure 4*a*. There cannot be a 1 output until the parity bit arrives and completes a 4-bit string whose parity is even. At each input, the parity of the input bits up to that point is either even or odd, so transition is to one of two possible next states: an even-parity state or an odd-parity state.

The partial diagram after the second message bit is shown in Figure 4*b*. Why is it necessary for the next state at the second input bit to be a new state rather than one of the existing odd-parity or even-parity states, A or B? (Give it some thought before you look at the footnote.[9]) Confirm the details of the complete diagram shown in Figure 4*c*; for example, show all possible input sequences by which states E and F are reached. An output of 1 occurs only upon returning to reset from state E following one of the sequences 0000, 0110, and 1100.

The next step is to construct the flow table from the state diagram. It is shown in Figure 5. The fourth input bit is never 1; so what will the output be for

[9]If A can be reached both after a single 0 and after a string 00, for example, then the count of the number of input bits is lost. Hence, we can't tell when the third bit has arrived in order to decide whether or not to generate a 1-bit, nor can we tell when the fourth bit has arrived so as to go back to the reset state.

Short ____
Even ____

PS	NS, z	
	$x = 0$	$x = 1$
A	A,0	B,0
B	A,0	C,0
C	A,1	D,0
D	A,0	D,0

(*a*)

y_1y_2	$(y_1y_2)^+$	
	$x = 0$	$x = 1$
A→00	00	01
B→01	00	10
D→11	00	11
C→10	00	11

(*b*)

y_1y_2	y_1^+	
	$x = 0$	$x = 1$
00	0	0
01	0	1
11	0	1
10	0	1

(*c*)

y_1y_2	y_2^+	
	$x = 0$	$x = 1$
00	0	1
01	0	0
11	0	1
10	0	1

(*d*)

Figure 6 State and transition tables for sequence detector.

$x = 1$ from states E and F? Confirm all the details of this table by reference to the state diagram. ■

2 STATE ASSIGNMENTS

The preceding section described the initial stage in the design of a Mealy-model synchronous sequential machine. From a word description of the specifications of the problem, it consists of constructing a state/output table (possibly after constructing a state diagram). During this process, it is possible that redundant states are introduced; consequently, the table might have more states than necessary to perform the desired task. Procedures for eliminating redundant states would be useful, leading to a reduced table with fewer states that is equivalent, in some sense, to the original table. Fortunately, such procedures do exist, and we will discuss them later. For the present, assume that a reduced table is available.

The state of a sequential machine at a given time is the condition in which past inputs have left it. This information is stored in the flip-flops; the state is, thus, described collectively by the outputs of the flip-flops. The next step in the design, then, is to identify the states in the table with specific flip-flop outputs. That is the subject of concern in this section. We will develop the subject by reference to the examples in the preceding section.

EXAMPLE 3

The state table derived for the sequence detector in Example 1 was given in Figure 3 and is repeated in Figure 6*a*. The minimum number of state variables needed for a circuit implementation of this table is $\lceil \log_2 4 \rceil = 2$, where $\lceil k \rceil$ denotes the *ceiling* of k, the smallest integer not less than k. Let us designate the state variables as y_1 and y_2. There are four possible combinations of values of these two variables. How should these four combinations be assigned individually to each of the four states? Before considering a general answer to that question, let us arbitrarily make the assignment shown in Figure 6*b*. (Let's temporarily neglect the output z and concentrate on the states.) The result is a table that, for each present combination of state-variable values and each input value, specifies the next combination of state-variable values. This is a *state-transition table.*

Note that there are two different orders: the order of listing the states in the state table (alphabetical) and the order of the combinations of state-variable values. If the assignment is made so that both orders are maintained, there is no prob-

Transition Required			
y	to y^+	J	K
0	0	0	×
0	1	1	×
1	0	×	1
1	1	×	0

(*a*)

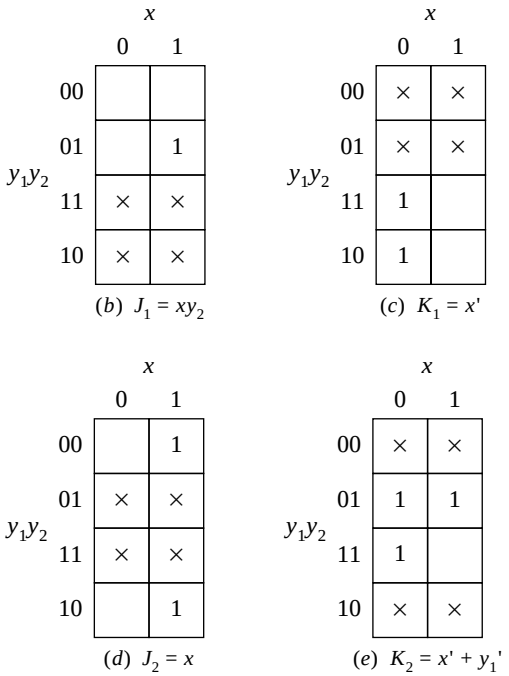


Figure 7 Excitation maps for the sequence detector.

lem. For any other assignment, either one or the other order can be maintained, but not both. It is much more convenient to maintain the order of the value combinations than the order of the state names, since the former is directly transferable to a logical map. This is what has been done in Figure 6*b*; states C and D are out of alphabetical order, but the value combinations are in logic map order.

For simplicity, the transition table in Figure 6*b* is separated into two tables in Figures 6*c* and 6*d*, one for each state variable. If we were to implement the design with *D* flip-flops, each of these tables would represent the excitation map for one of the flip-flops; for a *D* flip-flop, the present input (excitation) is the same as the next state. Just for pedagogical reasons let's implement it with *JK* flip-flops instead.[10]

The first requirement is to determine logic maps for excitations *J* and *K* for each flip-flop. For this we use the excitation requirements for *JK* flip-flops given in Chapter 5, Figure 17, and repeated here in Figure 7*a*. For each flip-flop, this table gives the required values of *J* and *K* for each transition from a present-state value to a next-state value. In each case, *J* and *K* are to be obtained as the output of a combinational circuit whose inputs are the circuit input x and present states y_1 and y_2. This requires combining the transitions from present to next state in Figures 6*c* and 6*d* with the transition requirements table in Figure 7*a*. Thus, from state $y_1y_2 = 11$ and $x = 1$, transition is to $y_1^+ y_2^+ = 11$ from Figures 6*c* and 6*d*. That is, for both y_1 and y_2, transition is from 1 to 1 for $x = 1$. But from Figure 7*a*, the requirements for a 1 to 1 transition is

[10]Only one combinational logic circuit is needed for the excitation when implemention is with *D* flip-flops. With *JK* flop-flops, two circuits are needed, one each for *J* and *K*. It is, of course, possible that the circuit required with a *D* flip-flop is more complex; nevertheless, with present-day implementations with PLDs, it is usually preferable to use *D* flip-flops. Furthermore, *JK* flip-flops require more chip area in ASIC technology.

Short ____
Even ____

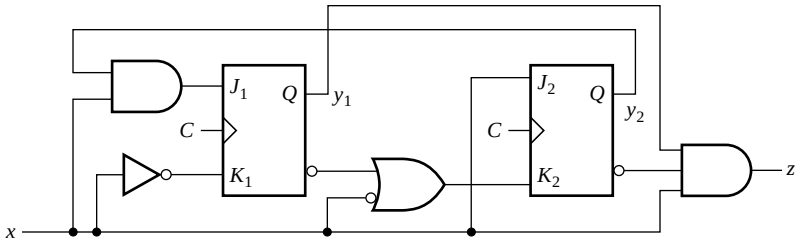


Figure 8 Implementation of the sequence detector.

$J = \times$ and $K = 0$. These constitute the content of one square in the logic maps for each J and K. The completed logic maps are shown in Figures 7*b* to 7*e*. (Confirm each of them and confirm the expressions for J and K given under the maps.)

To complete the implementation, an expression for the output function must be obtained. From the flow table in Figure 6*a* the output is 1 for exactly one state (state C) and one input ($x = 0$). Since C has the assignment $y_1y_2 = 10$, the expression for the output function is

$$z = x'y_1y_2'$$

The complete implementation is shown in Figure 8. (Confirm it all.) ■

Analysis

Once a sequential circuit has been designed and a logic diagram constructed, how can we tell that errors have not been made and that the circuit outputs actually satisfy the original specifications? As discussed in relation to combinational circuits in Chapter 3, a process called *verification* is carried out. This involves making measurements (of voltage, say) at appropriate points in any circuit (not just logic circuits) to verify that the actual values are what they are supposed to be theoretically.

Again, as discussed in Chapter 3, there is no point in physically implementing the circuit before verification. Once a paper (or software-generated) sequential circuit has been obtained, one can *analyze* the circuit at various points to verify that the logic values at these points are indeed the values required by the design specs. Compared with the process of design, logic-circuit analysis is rather trivial. One starts at any point in the circuit (gate or circuit outputs, or MUX, flip-flop, or register inputs) and determines logic expressions for these variables. This is repeated until expressions for all outputs are obtained. Then one inserts all possible input values into these expressions. The values obtained are then compared with what they are supposed to be.

As an example, look back at the sequence-detector design in Figure 8. Carry out an analysis of the circuit and obtain expressions for the J's and K's and the output z. (Don't peek at Figure 9 until you have completed it.) The lines on the time axis in Figure 9 represent the rising edge of the clock signal. Using these expressions and the transition table of JK flip-flops, and assuming the input sequence shown on the first line, the values of the other variables are determined column by column. Note that, when $x = 0$, the values of the J's and K's do not depend on the states of the flip-flops; hence, the next states in the

___ Short
___ Even

		Time										
x		0	1	1	0	1	1	0	0	1	1	1
J_1	xy_2	0	0	1	0	0	1	0	0			
K_1	x'	1	0	0	1	0	0	1	1			
J_2	x	0	1	1	0	1	1	0	0			
K_2	$(xy_1)'$	1	1	1	1	1	1	1	1			
y_1^+		0	0	1	0	0	1	0	0			
y_2^+		0	1	0	0	1	0	0	0			
z	$x'y_1y_2'$	0	0	0	1	0	0	1	0			

Figure 9 Timing table for the implementation in Figure 8.

first column are based only on $x = 0$. When $x = 1$, the values of J_1 and K_2 do depend on the states (not the next states in the same column but the states in the previous column). Verify the remaining columns and complete the last three columns in Figure 9. Verify that the outputs satisfy the specifications.

Exercise 2 Using the information in Figure 9, choose an appropriate scale and draw a timing diagram that includes the circuit input, the flip-flop inputs, the next states, and the circuit output. ◆

Rules of Thumb for Assigning States

A number of loose ends in the preceding development remain to be explored. The first is the simple observation that, when a transition table is obtained after a state assignment is made, as in Figure 6*b*, it is not essential to rewrite it in the form of the individual state variable transition tables, as was done in Figures 6*c* and 6*d*, before constructing the excitation maps. Instead, for each input value, concentrate on the column corresponding to one of the state variables, say y_1, mentally blocking the others from your perception, and construct the maps directly from the general transition table. Practice doing that for Figure 6*b*.

A more important consideration is the following. Given a state table having k states, the number of state variables needed to implement it is $n = \lceil \log_2 k \rceil$. An immediate decision is needed as to which of the 2^n combinations of state-variable values should be assigned to each of the k states. For a nontrivial number of states, many different possibilities exist for this assignment.

Exercise 3 The state table for the sequence detector in Example 1 was given in Figure 6*a*. The implementation of the circuit using the assignment in Figure 6*b* was given in Figure 8. Instead, use the following assignment and find an implementation for the circuit: A: 00, B: 01, C: 11, D: 10. Compare the number of gates with the number in Figure 8.
Answer[11]

Short ___
Even ___

[11]One more gate than the number in Figure 8. ◆

1. Two present states should be assigned adjacent codes if they have the same next state for:
   a. Each input combination
   b. Different input combinations, if the next state can also be given adjacent assignments
   c. Some input combinations, but not necessarily all
2. For all inputs, codes assigned to the next states for each present state should be adjacent.
3. Assignments should simplify the output function.

Figure 10 State assignment rules.

In general, the choice of assignment will influence the implementation. Different assignments lead to different maps of flip-flop excitation and output and, hence, to different expressions for excitation functions and output functions. Unfortunately, there is no general theory on assignments—and so no algorithm—that will result in simplicity of implementation. Experience is the only guide to making a state assignment.

General models of sequential circuits were given in Figure 1. Although the actual circuit in Figure 8 is quite simple, it illustrates the Mealy model well. The nonsequential part of the circuit consists of two classes: the combinational circuit that implements the excitations and the one that implements the output, as expected. Once state reduction has been carried out (until you learn how, you'll have to subcontract it out), the extent of the memory (the number of flip-flops) is fixed. Economy of implementation, then, is a matter of reducing the number of IC packages (and gates) in either the state decoder, the output decoder, or both.

Recall that the number of prime implicants and the number of literals in a prime implicant can be reduced when there are many adjacent minterms. Hence, it comes down to this: How do we choose state assignments so as to achieve a large number of adjacencies? Not much in the way of generalities can be deduced from an examination of Example 3. However, on the basis of a great deal of experience, some heuristic "rules" have been formulated as guides in making a state assignment for the case of a single input. Figure 10 lists a number of such rules, in priority order.

For a given state table, it is unlikely that all the adjacencies specified by these rules can be achieved. When there is a conflict, the higher-priority rules take precedence. Even if the rules can be fully implemented, they do not guarantee an optimal assignment. That is, the rules do not constitute an optimal algorithm.

Furthermore, they do not necessarily lead to a unique assignment; it may be possible for the required adjacencies to be achieved by different assignments. Even so, the rules will reduce the number of alternatives that must be checked. Finally, even for the same assignment, the number of logic gates using *JK* flip-flops might be different from the number using *D* or other flip-flops. Notwithstanding all that, using these rules as a guide is reasonable.

EXAMPLE 4

An application of the assignment rules in the implementation of a sequential machine is illustrated in the state table shown in Figure 11*a*. Since there are seven states,

___ Short
___ Even

PS	NS,z $x = 0$	$x = 1$
A	B,1	B,0
B	C,0	D,1
C	E,1	F,0
D	F,0	E,1
E	G,0	A,0
F	A,0	G,0
G	B,0	B,0

(*a*)

Rule	Adjacencies
1a	AG
1b	CD if EF
	EF if AG
2	CD, EF, AG
3	AC, BD

(*b*)

y_2y_3 \ y_1	0	1
00	A	E
01	G	F
11		B
10	C	D

(*c*)

Figure 11 Example state table, adjacencies, and assignment map.

the circuit will need $\lceil \log_2 7 \rceil = 3$ flip-flops. The adjacencies of states called for by the adjacency rules are shown in Figure 11*b*. (Don't fail to confirm these.) It is now a problem of determining a state assignment so that as many as possible of these adjacencies is achieved. To help in this process, an assignment map can be created, as shown in Figure 11*c*. This is a map whose coordinates are the three state variables. Each square in the map corresponds to a combination of state variable values.

The placement of the states in the map is initiated by deciding on the state that is to have the assignment 000. If there is a reset state in the problem, it is reasonable to give it this assignment; if not, the choice is arbitrary. Suppose the combination 000 is assigned to state A; then G must have an adjacent assignment. But each cell in the map has *three* others adjacent to it; which one is chosen for G depends on which other adjacencies are needed. In the present example, G is not required to be adjacent to any other state, so the placement is very flexible. One possibility is to assign 001 to G. The remaining assignments are made using the same approach, resulting in the assignment map in Figure 11*c*.

The next step in the design process is to construct transition and output tables. This is done using the assignment map and the given state table. The result is shown in Figure 12*a*. Let us again assume that *JK* flip-flops are to be used in the implementation; refer to Figure 7*a* for the transition requirements for this flip-flop. From Figures 7*a* and 12*a*, we construct the *J* and *K* excitation maps for each flip-flop. The result for the first flip-flop is shown in Figure 12*b*. The others are obtained similarly. Expressions for the excitation and output functions are given in Figure 12*c* ■

Exercise 4 Construct the output map and the excitation maps for the other two flip-flops in Example 4 and confirm the expressions given in Figure 12*c*. ◆

Exercise 5 In Example 4, suppose that the implementation is to use *D* flip-flops. Determine the maps for the *D* excitations and, from these, the state decoder. Compare the hardware with the case that uses *JK* flip-flops. ◆

EXAMPLE 5

Short ___
Even ___

The objective of this example is to find a good assignment scheme for the state table in Figure 13*a*.

		$y_1^+y_2^+y_3^+$		z	
	$y_1y_2y_3$	$x = 0$	$x = 1$	$x = 0$	$x = 1$
A	000	111	111	1	0
G	001	111	111	0	0
	011	—	—	—	—
C	010	100	101	1	0
E	100	001	000	0	0
F	101	000	001	0	0
B	111	010	110	0	1
D	110	101	100	0	1

(*a*)

y_2y_3 \ xy_1	00	01	11	10
00	1	×	×	1
01	1	×	×	1
11	×	×	×	×
10	1	×	×	1

$J_1 = 1$

y_2y_3 \ xy_1	00	01	11	10
00	×	1	1	×
01	×	1	1	×
11	×	1		×
10	×			×

$K_1 = y_2' + x'y_3$

(*b*)

$$J_2 = y_1'$$
$$K_2 = y_3'$$
$$J_3' = (x' + y_1')(x + y_1 + y_2')$$
$$K_3 = (x + y_1)(x' + y_3')$$
$$z = x'y_1'y_3' + xy_1y_2$$

(*c*)

Figure 12 Transition table and excitation maps.

PS	NS,z $x = 0$	$x = 1$
A	B,0	B,1
B	F,0	D,1
C	E,1	G,1
D	A,0	C,0
E	D,1	G,0
F	F,0	A,0
G	C,1	B,0

(*a*)

Rule	Adjacencies
1c	AG, BF, CE
2	AC, AF, BC
	DF, DG, EG
3	AB, AC, BC
	CE, CG, EG

(*b*)

y_2y_3 \ y_1	0	1
00	A	C
01	G	E
11	D	B
10	F	

y_2y_3 \ y_1	0	1
00	A	G
01	F	D
11	B	
10	C	E

(*c*)

Figure 13 State table, required adjacencies, and assignment maps for Example 5.

The first step is to determine the adjacencies using the adjacency rules; they are listed in Figure 13*b* (verify, please). With seven states, the number of state variables needed is three. Each cell in a three-variable map is adjacent to three other cells, so each state can be adjacent, at most, to three other states. In a full three-variable map, a total of 12 adjacencies are thus available. (Confirm this.) There are only seven states in this example; it turns out that the maximum number of adjacencies present is nine. From the list of required adjacencies, it is clear that A is required to be adjacent to four other states: G, C, F, and B. Since the lowest-priority adjacency is AB,

___ Short
___ Even

1. *State table:* Given the specifications of a problem in natural language, construct a state table satisfying the specifications, perhaps by first constructing a state diagram.
2. *Equivalent reduced table:* Use appropriate procedures to determine equivalent states and to remove redundant states, thus generating an equivalent reduced table. (Procedures will be considered in the following section.)
3. *State assignment:* Choose a state assignment.
4. *Transition and output tables:* Use the assignment to construct these.
5. *Excitation maps:* Choose a flip-flop type; using the transition table and the excitation requirements for the chosen flip-flop, construct excitation maps.
6. *Excitation functions:* Derive expressions for these from the maps.
7. *Output functions:* Derive expressions for these from the output table.
8. *Implementation:* Implement the state decoder from the excitation functions and the output decoder from the output functions.

Figure 14 Design procedure for Mealy machines.

that one should be the first to be abandoned. Similarly, the adjacency CG should be abandoned, since C (and G) cannot be made adjacent to four other states.

Whenever there is a choice, try to achieve those adjacencies at one priority level that are also required at a lower level. Aside from AB and CG (which we abandoned), the remaining adjacencies in rule 3 are also required by higher-order rules.

The number of achievable adjacencies in this example turns out to be nine, equal to the maximum possible. Figure 13*c* shows two assignment maps; each achieves all nine required adjacencies. Using *JK* flip-flops, the first one can be implemented with five AND and four OR gates. The second one needs one more OR gate. ■

Exercise 6 Carry out implementations for the two assignments given in Example 5 and confirm the stated results. Convert to all NAND gates. ◆

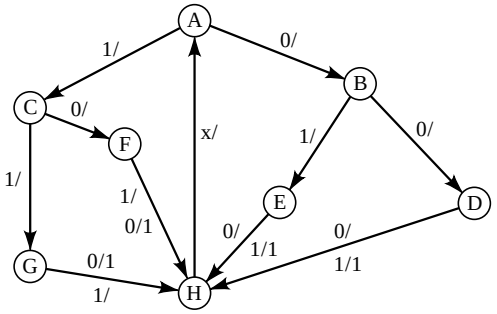
3 GENERAL DESIGN PROCEDURE

Each of the elements of a procedure for the design of synchronous sequential machines has been discussed in preceding sections of this chapter. We are now ready to consolidate these elements into a general design procedure. We will illustrate this general procedure by applying it to some specific examples.

Mealy Machine

The Mealy and Moore circuits are models that were shown in Figure 1. When a sequential circuit design problem is specified in terms of the outputs desired for specific sequences of inputs, no model is generally specified. For a given design requirement, it is conceivable to carry out the design based on either model. That means two different designs (state tables) can be obtained. It also means that one of the designs can be obtained from the other. In this book we will deal with both Mealy and Moore machines. The general design procedure for Mealy machines is given in Figure 14.

Short ___
Even ___



(*a*)

	NS,*z*	
PS	$x = 0$	$x = 1$
A	B,0	C,0
B	D,0	E,0
C	F,0	G,0
D	H,0	H,1
E	H,0	H,1
F	H,1	H,0
G	H,1	H,0
H	A,0	A,0

(*b*)

Figure 15 State diagram and state table of change-of-level detector.

EXAMPLE 6

A sequential circuit is to be designed having a single input line x and a single output line z. Starting in a reset state, the circuit receives input sequences consisting of 3-bit binary words. Each input word follows the preceding word with a delay of one clock period. The circuit must be in the reset state at the beginning of each word. The output is to be $z = 1$ upon receipt of the third bit of a word if the total number of level changes (from 0 to 1 or from 1 to 0) is odd (101, for example, has two level changes while 001 has only one). Such a circuit is called a *change-of-level detector*.

As the first step, we'll obtain the state diagram. Starting from the reset state, no matter what the third bit is, the circuit is to wait one clock period and return to the reset state at the fourth clock pulse. That means the third bit of the input word sends the circuit to a *waiting* state. Figure 15*a* shows the state diagram. The reset state is A and the waiting state is H. (To confirm this diagram, cover the waiting state and all the lines coming into it and out of it; then describe each of the states reached after 2 bits in terms of the number of level changes. Confirm the output resulting from each input that sends the circuit to the waiting state.) The diagram and the statement of the problem make it clear that each state is unique and there are no redundant states; no two states are equivalent to each other.

The next step is the state table; this is easily constructed from the state diagram and is shown in Figure 15*b*. (Confirm this, please.) Since each state is unique, the state table cannot be further reduced. The number of flip-flops needed is $\lceil \log_2 8 \rceil = 3$. The rules of thumb for adjacency, shown in Figure 16*a*, lead to the adjacency map in Figure 16*b*.

For variety, this time let's use D flip-flops, for which the present excitation is the same as the next state. Hence, the entries in the transition table also specify the D excitations, so logic maps for the D's can be constructed directly from the transition tables. Likewise for the output. We will supply the results here, but we expect you to confirm all this in an exercise. The following expressions result from the maps.

$$D_1 = y_2 + xy_1' + y_1'y_3 \qquad D_2 = y_2'y_3$$
$$D_3 = y_1'y_2'y_3' + xy_2'y_3 \qquad z = x'y_1 + xy_1y_2$$

___ Short
___ Even

Rule	Adjacencies
1a	DE, DF, DG
	EF, EG, FG
2	FG, BC, DE
3	DE, FG

Adjacencies not achieved: DG, EF.

(*a*)

y_2y_3 \ y_1	0	1
00	A	H
01	B	C
11	G	E
10	F	D

(*b*)

	$y_1^+y_2^+y_3^+$		z	
$y_1y_2y_3$	$x = 0$	$x = 1$	$x = 0$	$x = 1$
A → 000	001	101	0	0
B → 001	110	111	0	0
G → 011	100	100	1	0
F → 010	100	100	1	0
H → 100	000	000	0	0
C → 101	010	011	0	0
E → 111	100	100	0	1
D → 110	100	100	0	1

(*c*)

Figure 16 Adjacency rules, assignment map, and transition table for the change-of-level detector.

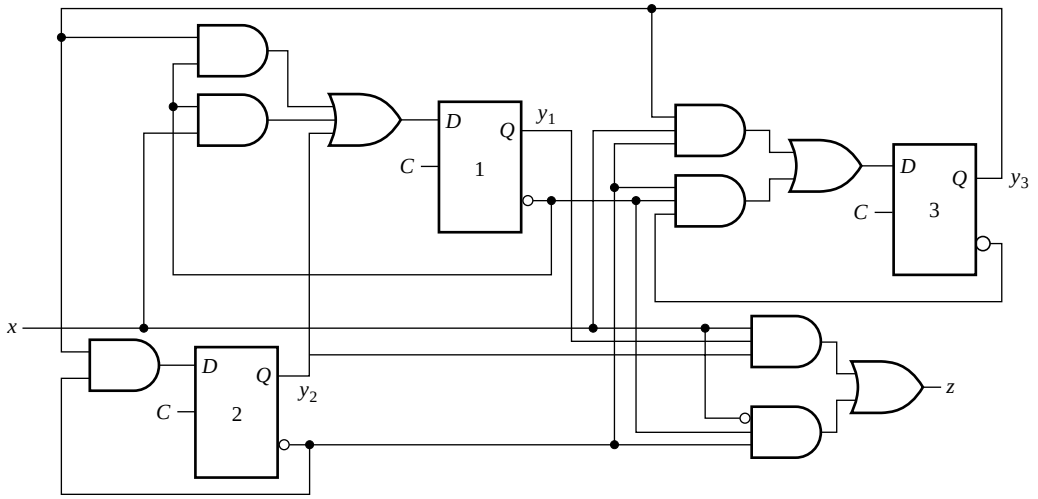


Figure 17 Implementation of the change-of-level detector.

The resulting circuit is shown in Figure 17. ■

Exercise 7 Using the transition table in Figure 16*c*, construct excitation and output maps. From these, confirm the preceding expressions for the *D* excitations and the outputs. Verify the implementation in Figure 12. ◆

EXAMPLE 7

A synchronous sequential machine is to have a single input line and a single output line. The circuit is to receive messages of 5-bit words coded in 2-out-of-5 code. (See Chapter 1 for a description of codes.) The purpose of the circuit is to detect an error in any of the words. Thus, the output is to become 1 whenever a 5-bit word does not represent a valid code word. At the end of each word the machine is to return to the reset state.

Short ____
Even ____

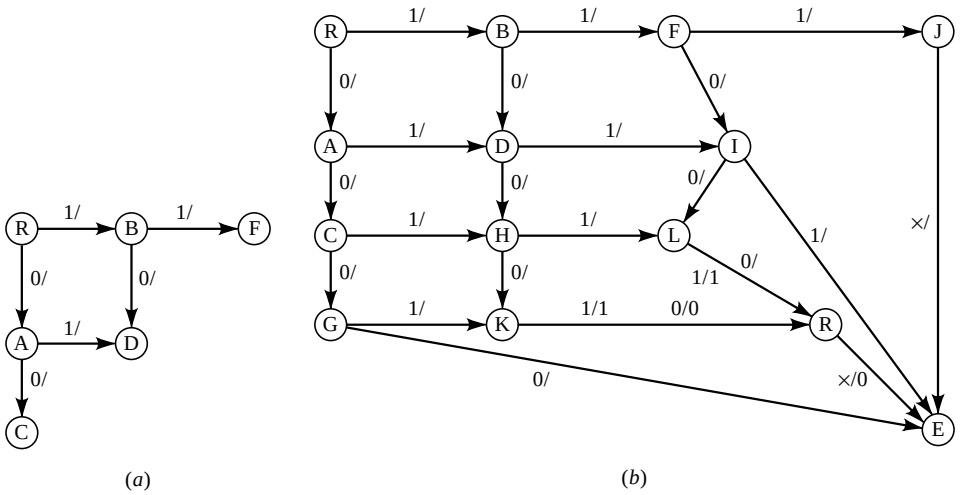


Figure 18 State diagram of error detector in 2-out-of-5 code words.

State Diagram Let the reset state be labeled R. One possibility for the structure of the state diagram is a tree, starting at state R. In such a diagram, when the machine is in any state, each of the two possible input values leads to a new state. Thus, the state diagram will have $2^5 = 32$ states. That not all of these are independent states is seen by examining the information that is needed upon receipt of the *k*th bit:

- How many bits have been received so far?
- How many of these are 1 bits?

Starting with the reset state, after the receipt of the second bit of a word, there are three possibilities: the number of 1 bits received is 0, 1, or 2. The partial state diagram is shown in Figure 18*a*. The three possibilities identify just three states after receipt of the second bit. A tree structure would require four states at this point. Upon receipt of the third bit of the word, there are four possibilities for the number of 1 bits received to that point—0, 1, 2, and 3—and thus four new states, labeled G, H, I, J in Figure 18*b*. Note that no matter what the fourth bit is while at present state J, the received word will never be in 2-out-of-5 code. Hence, from state J, the next state will be an error state (for which the letter E is reserved), but the output will not become 1 until the arrival of the fifth bit, whether a 1 or a 0.

The completed state diagram is shown in Figure 18*b*; to avoid clutter in the diagram, with lines running back to R from each of the states reached after 4 bits, a second copy of R is provided near these latter states. The two copies of R constitute the same state.

State Table The next step is to construct the state table from the state diagram. Do this and confirm the table given in Figure 19*a*.

Assignment Map The number of flip-flops needed is $\lceil \log_2 13 \rceil = 4$. Apply the rules of thumb for state adjacencies and confirm the list given in Figure 19. An assignment map that achieves all but one of the adjacencies is given in Figure 19*b*. It is impossible to achieve all three of the adjacencies required by rule 1. Since two of them are

___ Short
___ Even

PS	NS,*z* $x = 0$	$x = 1$
R	A,0	B,0
A	C,0	D,0
B	D,0	F,0
C	G,0	H,0
D	H,0	I,0
E	R,1	R,1
F	I,0	J,0
G	E,0	K,0
H	K,0	L,0
I	L,0	E,0
J	E,0	E,0
K	R,1	R,0
L	R,0	R,1

(*a*)

Rule	Adjacencies
1a	EK, EL, KL
1c	GJ, IJ
2	AB, CD, DF, GH, HI, IJ, EK, EL, KL
3	EK, EL

y_3y_4 \ y_1y_2	00	01	11	10
00	R	C	A	I
01	E	L		J
11	K	F		G
10		D	B	H

(*b*)

$y_1y_2y_3y_4$	$y_1'y_2'y_3'y_4'$ $x = 0$	$x = 1$
R → 0000	1100	1110
E → 0001	0000	0000
K → 0011	0000	0000
0010	xxxx	xxxx
C → 0100	1011	1010
L → 0101	0000	0000
F → 0111	1000	1001
D → 0110	1010	1000
I → 1000	0101	0001
J → 1001	0001	0001
G → 1011	0001	0011
H → 1010	0011	0101
A → 1100	0100	0110
1101	xxxx	xxxx
1111	xxxx	xxxx
B → 1110	0110	0111

(*c*)

Figure 19 State table, assignment map, and transition table for example.

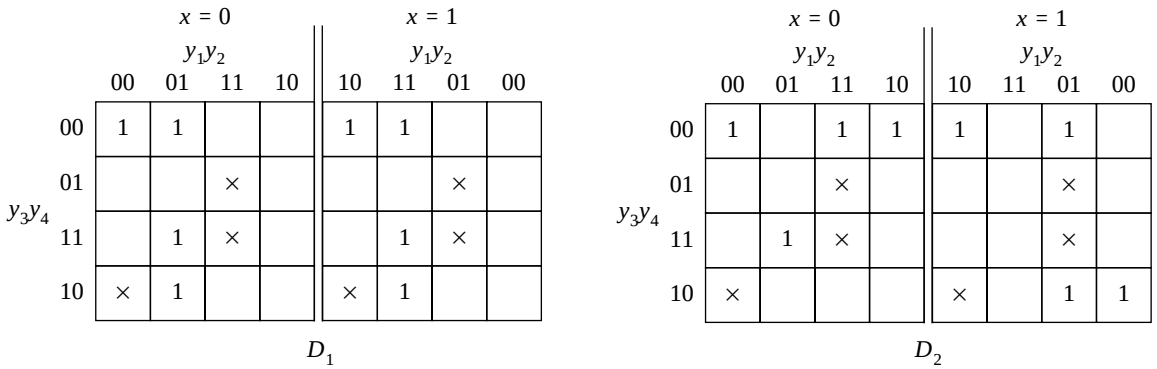


Figure 20 Excitation maps for error detector.

Short ____
Even ____

required by lower-priority rules, it is these two that are actually achieved. (Confirm everything as you go along.)

Transition Table The transition table resulting from the adjacency map is shown in Figure 19*c*. Since four state variables imply 16 possible states, and there are actually only 13, three of the state variable combinations do not correspond to states of the machine. Hence, we don't care what the next states resulting from such nonexistent present states will be.

Flip-Flop Type, Excitation and Output Functions Assume the use of *D* flip-flops in the implementation. The excitation and output maps require five variables. The excitation maps are shown in Figure 20. Confirm each of these maps and the resulting excitation and output expressions that follow. (The output map is simple enough that it is not shown.)

$$D_1 = y_1'y_4' + y_1'y_2y_3$$
$$D_2 = y_1y_2 + y_1'y_2'y_4' + x'y_1y_3'y_4' + xy_1y_3y_4'$$
$$D_3 = y_1y_2y_3 + x'y_1y_2'y_4' + xy_1y_2 + xy_1'y_3'y_4' + xy_1y_3y_4$$
$$D_4 = y_1y_2' + xy_1y_3 + xy_2y_3y_4 + xy_1'y_2y_3'y_4'$$
$$z = x'y_1'y_2'y_4 + xy_1'y_3'y_4$$ ■

Moore Machine

As shown in the model in Figure 1*b*, in a Moore machine the outputs do not depend directly on the inputs. Hence, in the state table of a Moore machine there is a single output column for each present state, independent of the input. An example will illustrate some of the features.

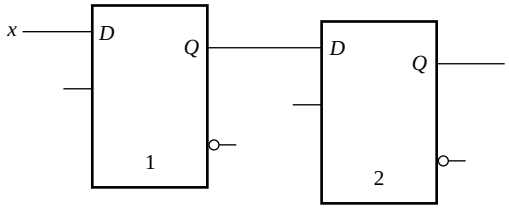
EXAMPLE 8

A synchronous state machine is to be designed to serve as an odd-parity checker. The inputs to be checked for odd parity arrive on an input line *x*, but the parity is checked only while the signal on another input line *y* (a synchronizing input) is 1. An output *z*, depending only on the state, is to become 1 when the parity fails to be odd. A possible sequence of inputs and output are

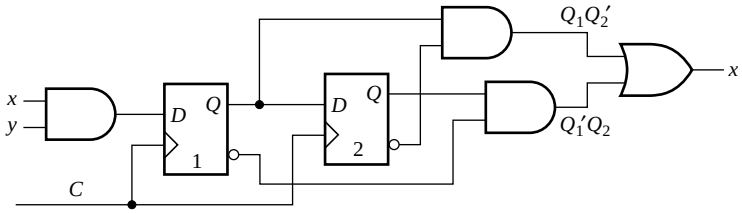
y: 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0
x: × × × 1 0 1 0 1 1 1 0 1 0 0 × ×
z: - - - 0 0 0 0 1 1 1 1 1 0 - - -

Since the output is to depend only on the state, parity is determined from a memory of just the last two input *x* bits. This suggests two flip-flops, let's say *D* flip-flops. A register consisting of two *D* flip-flops, labeled 1 and 2, is shown in Figure 21*a*.[12] Note that, at any given clock tick, Q_1 will have whatever value *x* had on the preceding clock tick. Similarly, Q_2 will have whatever value Q_1 had at the preceding

[12]Registers were introduced in Chapter 5; review if necessary.



(*a*)



(*b*)

Figure 21 Moore circuit realizing Example 8.

clock tick. Using this information, construct three-variable logic maps (x, Q_1, and Q_2) for Q_1^+ and Q_2^+ and confirm that $Q_1^+ = x$ and $Q_2^+ = Q_1$. Since the output depends only on the state, a logic map for the output will be just a two-variable map (Q_1 and Q_2). Draw this map and determine an expression for the output.
Answer[13]

The requirement that the process will proceed only when $y = 1$ can be met by an AND gate with inputs x and y. The input to the first flip-flop will remain 0 when $y = 0$, but it will be x when $y = 1$. The full circuit, including the output decoder, is shown in Figure 21*b*. (Please verify.) ■

What was done in this example seems almost like cheating: no state diagram was drawn, no state tables created, no assignments made, and so on. We have seen that two flip-flops will do the trick and that, hence, there is a maximum of four states. As an exercise, construct a state table using the assignments (00, 01, 10, and 11) as state names and construct a separate column for the output, independent of the input. Using this table, confirm the logic maps you previously constructed.

4 STATE EQUIVALENCE AND MACHINE MINIMIZATION

Exercise 1 in section 1 (and also Example 7) demonstrated that more than one state diagram (or state table) can be constructed to satisfy the description of a design problem. If one such table has nine states and another has four, the first requires twice as many flip-flops in its implementation as the second. But no

Short ____
Even ____

[13] $z = Q_1Q_2' + Q_1'Q_2 = Q_1 \oplus Q_2$

more information needs to be stored in the first machine than in the second. It must be that whatever task is performed by some of the nine states is performed also by other states, rendering some of the nine states superfluous.

It would be of great benefit to detect these superfluous states and remove them, thus leaving a reduced state table. We have already mentioned the possible reduction in the complexity of the circuit. Once a reduced machine is implemented, what is possibly of even more value is that the reduced complexity makes verification (the experimental determination of faults in a machine) considerably simpler. The purpose of this section is to develop procedures for reducing a given state table to one that carries out the same function with fewer states.

Distinguishability and Equivalence

A finite-state machine operates by receiving a sequence of input symbols, making transitions of state, and emitting output symbols. The input sequence becomes transformed into the output sequence. Look back at the state table in Figure 11 and suppose the circuit is in state B when an input 1 occurs. A transition is made to state D. We describe this by saying that state D *succeeds* state B under an input 1, or that D is the *1-successor* of B. Now suppose that the longer input string 011 arrives when the machine is initially in state A. The final state reached will be E. (Trace out the sequence of states encountered and verify.) So E might be called the 011-successor of A. In general,

> *If input sequence X is applied to a finite-state machine that is in state S_i and the machine makes a final transition to S_j, then S_j is the X-successor of S_i.*

If the same input sequence *X* is applied to a machine twice, once when the machine is in state S_i and once when it is in S_j, the two output sequences produced may or may not be the same. If they are the same, we will not be able to distinguish the two initial states by means of that particular input sequence. Now suppose that the output sequence is always the same, starting from each of the two states, no matter what input sequence is applied. Clearly, the two states could never be distinguished from each other. This leads to the following definition:

> *Two states S_i and S_j in a finite-state machine are* equivalent *if the same output sequence is produced in response to an input sequence, starting in either state, and this is true for every finite input sequence.*

If the potential equivalence of two states were to be checked using this definition, a whole career would be needed to check all possible input sequences. Clearly, a shorter test is needed. Suppose that, when the machine is started in two different states, the output sequences produced by the same input sequence are *not* the same. The two states can then be distinguished. We make the following statement:

> *Two states S_i and S_j of a machine are* distinguishable *if and only if there exists at least one finite input sequence that produces different output sequences starting first from state S_i and then from S_j. If the distinguishing sequence has length k, then the two states are k-distinguishable.*

___ Short
___ Even

PS	NS,z $x = 0$	$x = 1$
A	B,0	D,1
B	C,1	D,1
C	C,0	A,0
D	A,0	C,1

(*a*)

PS	NS,z $x = 0$	$x = 1$
A	B,0	D,1
B	C,1	D,1
C	E,0	A,1
D	A,0	C,1
E	B,0	D,1

(*b*)

Figure 22 Example machines.

To illustrate, consider states A and D in the state table given in Figure 22*a* as initial states. The outputs are the same for either input of length 1 (0 or 1). Hence, states A and D are not 1-distinguishable. Now take an input 00, of length 2. Starting from A, the output is 01, but starting from D, it is 00. Hence, states A and D are 2-distinguishable. This leads to the following definition:

Two states that are not k-distinguishable are k-equivalent.

The previous definition for equivalence can now be expressed as follows:

Two states of a machine are equivalent if they are k-equivalent for all k.

In the preceding discussion we have concentrated on output sequences in response to an input sequence. No attention has been paid to the next states that result along the way. In Figure 22*a*, for example, we found states A and D to be 1-equivalent; then we tested an input string of length 2. Suppose, instead, that we consider the next states after the first input. The next states are not the same starting from states A and D. Furthermore, it is evident that, using those next states as present states, the same output does not result for each input. Hence, the original states A and D are distinguishable.

Let's pursue this line, using Figure 22*b*, which is almost the same table as the one in Figure 22*a*. Now states A, D, and E are all 1-equivalent. Furthermore, the next states from A and E are the same for each input. Hence, after the first input bit, the transition from each of states A and E will be to the same next state; the outputs thereafter will be exactly the same. Hence, A and E are equivalent.

Machine Minimization

The preceding discussion gives a clue as to how to find the states of a machine that are equivalent to each other. Starting from an input sequence of length 1, we group those states that are 1-equivalent. These states are distinguishable from the others. Next we examine the next states to decide on their distinguishability, and so on. The details of the process are best described with an example.

EXAMPLE 9

Short ___
Even ___

A state table is given in Figure 23*a*. The objective is to find all groups of equivalent states and to reduce the table to one having a minimal number of states.

PS	NS,z	
	$x = 0$	$x = 1$
A	D,1	G,1
B	C,0	D,1
C	E,0	F,1
D	F,1	B,1
E	B,0	F,1
F	D,1	C,1
G	A,0	D,1

(*a*)

$P_1 = \{ADF; BCEG\}$
$P_2 = \{ADF; BCE; G\}$
$P_3 = \{A; DF; BCE; G\}$
$P_4 = P_3$

(*b*)

PS	NS,z	
	$x = 0$	$x = 1$
A → S_1	S_3,1	S_4,1
BCE → S_2	S_2,0	S_3,1
DF → S_3	S_3,1	S_2,1
G → S_4	S_1,0	S_3,1

(*c*)

Figure 23 Partitioning and machine minimization.

In accordance with the plan, we start by identifying those states that are distinguishable with an input sequence of length 1. We find that the group of states A, D, F have the same output for $x = 0$; they also have the same output for $x = 1$. Hence, they are not distinguishable with an input sequence of length 1. Similarly, confirm that the states B, C, E, G are indistinguishable with an input sequence of length 1. But these two groups of states are distinguishable from each other. Thus, the totality of all the states in the table can be *partitioned* into two blocks of states, written as follows: $P_1 = \{ADF; BCEG\}$. Within each block, the states are indistinguishable with an input sequence of length 1, but those in one block are distinguishable from those in the other.

Next we examine the successor states from all states in each block, one at a time. If, for each input symbol, the next states from all states in a block are not in the same block but fall in two distinct blocks, then the two sub-blocks are distinguishable. Hence, the original block must be subdivided. Thus, for $x = 1$, the next states from the block BCEG are DFFD; these are all in the block ADF. However, for $x = 0$, the next states from the block BCEG are CEBA; all except the next state from state G are in the same block. Hence, block BCEG must be subdivided into two blocks, BCE and G. The resulting partition is $P_2 = \{ADF; BCE; G\}$, as shown in Figure 23*b*; it is a *refined* version of P_1.

The next states from any one block in partition P_2 were found to be in the same block; hence, these next states will have the same outputs for each input bit. (That's because the outputs were the same, even for the larger blocks in partition P_1.) These next states in each block are, then, 1-equivalent. Hence, their predecessor states are 2-equivalent.

The process is now repeated with partition P_2. Again we take each block one at a time and, for each input, examine their next states to see if they fall in the same new block. (Clearly, blocks containing a single state need not be examined.) For each input, the next states from the block BCE fall in the same block. This is also true for $x = 0$ for block ADF; however, for $x = 1$, the next states for block ADF are GBC. These next states are not in the same block; hence, a further refinement of P_2 is needed, as shown in Figure 23*b*. The states in each new block are 3-equivalent.

The process must be repeated on the multistate blocks in partition P_3. Go through the process and confirm that no further refinements of the partition are

Short
Even

needed. The states within each block cannot be distinguished; hence, they are equivalent. This final partition is called the *equivalence partition.* Each of the four blocks in the equivalence partition constitutes a state of the machine to which each of the original states in that block is equivalent. If these states are labeled S_i, a reduced state table can be constructed, as shown in Figure 23*c*. ■

The description of this process is far lengthier than the actual effort involved in carrying it out.[14] Note, in this example, that the reduced machine needs just two flip-flops in its implementation, whereas the original table required three.

The subject of this section constitutes the second step in the general design procedure described in the preceding section. As noted earlier, when you initially construct a state table to satisfy the specifications of a design problem, it isn't necessary to spend a lengthy amount of time to ensure that there are no redundant states. Any redundant states introduced earlier can always be removed in the machine minimization step.[15]

5 MACHINES WITH FINITE MEMORY SPANS[16]

What distinguishes sequential circuits from combinational circuits is memory. The information stored in memory, together with an input sequence, determines an output sequence. But how much past data is it necessary for the machine to remember? Is it necessary for the machine to remember past *inputs only,* or can its future behavior be determined from the present input and a memory of past *outputs*? Or is a memory of both inputs *and* outputs neccessary?

We will examine three classes of machines in this section. In the three respective cases, the present output is determined by the present input plus

1. A limited number of the immediately preceding inputs
2. A limited number of the immediately preceding outputs
3. A limited number of the immediately preceding inputs *and outputs*

The machines are all said to have *finite memory spans.*

Not all finite-state machines have this characteristic; a machine with a waiting state, for example, does not. It cannot produce an output while in this state, no matter how long it stays there, even if it receives an acceptable input sequence. Each of the three classes of finite-memory machines will be discussed and implemented in certain specific structures.

[14]The process is algorithmic in nature. That means software can be produced to carry it out. Here we will concentrate on the principles.

[15]The treatment here has been limited to completely specified machines. Incompletely specified machines require the introduction of several additional concepts that require extensive development. These concepts apply also to asynchronous machines, and they will be treated in Chapter 7. If you skip Chapter 7, you will also be skipping coverage of incompletely specified synchronous machines.

[16]This section can be omitted without penalty in terms of preparation for material that follows.

Short ___
Even ___

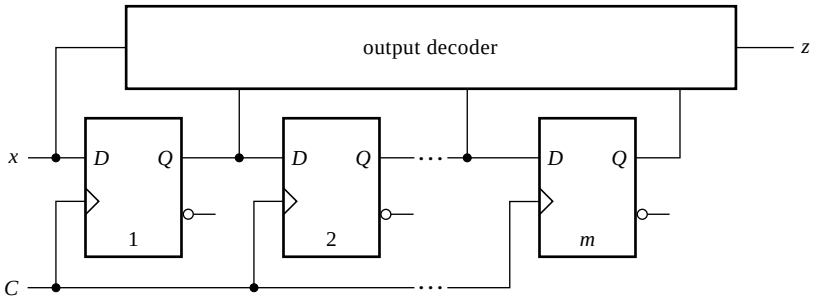


Figure 24 Canonic implementation of a finite-input-memory machine.

Machines with Finite Input Memory

One possible formal definition of the machines to be discussed in this section is the following.

> *A finite-state machine M is said to have* finite input memory *of memory span (or order) m if the present state of M can be determined uniquely from the preceding m input symbols but no fewer than m.*

It is clear from the definition that a circuit implementing the memory is an m-flip-flop shift register in which the last m inputs are stored. The present state consists of the outputs of the m flip-flops in the register. A *canonic* implementation consists of an m-flip-flop shift register and a combinational output decoder, as shown in Figure 24.

The shift register in Figure 24 is a serial-to-parallel converter. The input information arrives sequentially and is stored in the shift register. When the last bit of an input sequence of appropriate length arrives, both that input and the previously stored information are applied to the combinational logic at the same time, in parallel; the logic of the decoder then produces the desired output.

Since the state of a finite-input-memory machine of span m after an m-bit input sequence is known, the output will become known when the next bit arrives. Hence, such a machine can also be defined as one whose output is determined by the present input and the preceding input sequence of m bits.

The design procedure for a machine with finite memory span can be simplified if the specifications of the design problem allow us to recognize its nature; only the output decoder needs to be designed.

EXAMPLE 10

A synchronous sequential circuit with a single input line x and a single output line z is to be designed so as to produce an output $z = 1$ whenever an input symbol completes a sequence of 4 identical input bits; the output is to be 0 otherwise.

Just prior to receipt of each input symbol, the machine must remember only the preceding 3 bits. It then produces a 1 or a 0 on the basis of those 3 bits and the present input bit. This, then, is a machine having finite input memory of span 3. Labeling

___ Short
___ Even

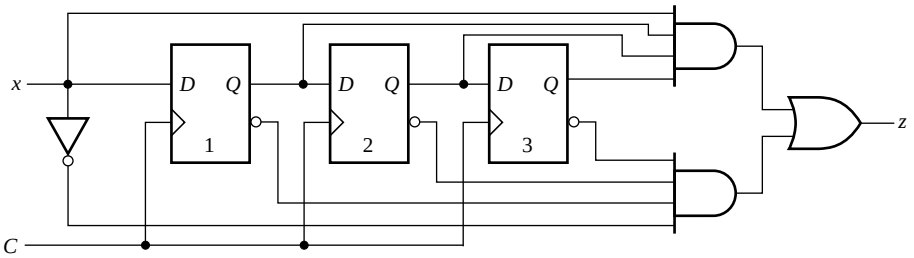


Figure 25 Canonic implementation of Example 10.

the states Q with subscripts 1, 2, and 3, from left to right, the minterms are 0000 and 1111. Hence, the output is easily written as

$$z = xQ_1Q_2Q_3 + x'Q_1'Q_2'Q_3'$$

The canonic implementation of the circuit is shown in Figure 25. ■

Although the preceding example included just a single input line, the concept and the canonic implementation apply to any number of input lines. A separate shift register is needed for each input line.

In addition to the main inputs, some machines have one or more control inputs in order to change the instructions for the generation of an output. It is the present values of those inputs that do the controlling; past control inputs need not be stored. To illustrate, in the preceding example there might be a control input x_c that, under the previously given conditions of the problem, permits the output to become 1 only if $x_c = 1$. For $x_c = 0$ the output could be specified as something else in terms of the main input x and its past values. In the implementation of the machine, the shift register in Figure 25 would not change; only the output decoder logic would be modified.

Machines with Finite Output Memory

In the second class of machines being considered, it is the preceding *outputs* that are to be remembered rather than the inputs. The definition of this class of machines is as follows:

> *A sequential machine M is said to have* finite output memory *of memory span (or order) m if the present output of M can be determined from the present input and the immediately preceding m (but no fewer) output symbols.*

Again the definition makes it clear that the memory can be implemented with a shift register (left shift) of m flip-flops in which the preceding m *outputs* are stored. Then, when the next input symbol arrives, the output is determined by both this input and the previously stored outputs. The canonical implementation is shown in Figure 26. The input to the shift register is the most recent output symbol. It is also clear that, if there is more than one output line, the canonic implementation will require more than one shift register.

Short ___
Even ___

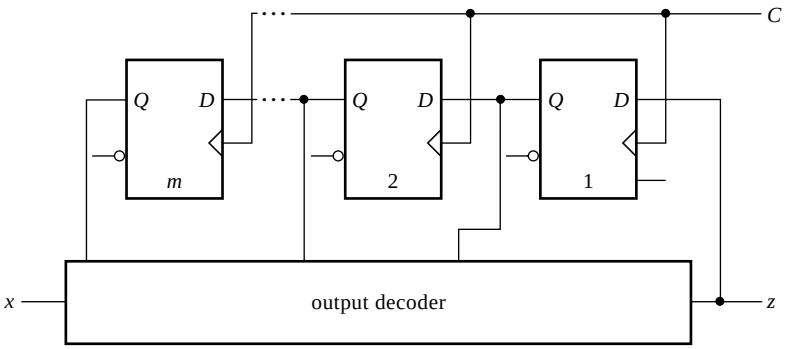


Figure 26 Canonic form of finite-output-memory machine.

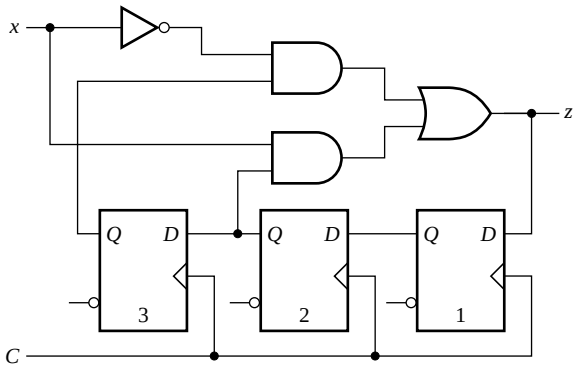


Figure 27 Finite-output-memory machine implementing Example 11.

EXAMPLE 11

A single-input, single-output sequential machine is to be designed. For an input $x = 1$, the output is to equal what the output was two clock periods earlier; and for $x = 0$, the output should equal what the output was three clock periods earlier.

The most that the machine must remember is three preceding output symbols. Hence, the desired machine has finite output memory of span 3. An expression for the output is easily written. If we label the states Q_1 to Q_3, then when $x = 1$, the output should be Q_2; and when $x = 0$ (which means $x' = 1$), the output should be Q_3. Hence,

$$z = xQ_2 + x'Q_3$$

(Verify the values of z when $x = 1$ and when $x = 0$.) The implementation is shown in Figure 27. ■

Exercise 8 From the problem statement in Example 11, draw a logic map for the output, with the input and the three states as map variables. From the map, write a minimal sum-of-products expression and confirm the expression in the

___ Short
___ Even

example. (This implementation does not depend on recognition of the nature of the circuit as finite-output memory.) ◆

Finite-Memory Machines

The third class of finite-state machines under discussion depends not just on a fixed number of past inputs or on a fixed number of past outputs, but on both. The formal definition follows:

> *A sequential machine M is a* finite-memory machine *of span m if the present state can be determined uniquely from the preceding m_i (but no fewer) input symbols and the preceding m_o (but no fewer) output symbols; the span is* $m = \max\{m_i, m_o\}$.

You might conjecture that a canonic implementation of this machine would merge the two canonic implementations in Figures 24 and 26 by including two shift registers, one to store the m_i past inputs and one to store the m_o past outputs. While such an implementation is possible in some cases, it turns out that it is not universally possible.

Consider a finite-input-memory machine. From the definition just given, this machine is also a finite-memory machine. That is, if the present state is determined by the first m_i input symbols only, knowing also the first m_o output symbols will not detract from this. This finite-memory machine, however, is not a finite-output-memory machine; it is a finite-memory machine by virtue of having finite input memory. A similar argument can be made that a machine with finite output memory is, by virtue of this fact, a finite-memory machine, although not a finite-input-memory machine.

The converse is not true. That is, it is possible for a machine to be a finite-memory machine without having either finite input memory or finite output memory. To establish the validity of this claim requires the introduction of several additional concepts and algorithms that would take us too far afield. We will therefore abandon further consideration of the subject here but will provide some problems at the end of the chapter so you can explore it to some extent.

6 SYNCHRONOUS COUNTERS

We now turn to a class of sequential machines that perform a particular type of operation. A *counter* is a sequential machine that, starting at a particular state, cycles through a fixed sequence of states and then returns to its initial state; thereafter, it repeats this process. The number of distinct states in the counter is called its *modulo number*.

In some cases, the useful information from the counter may be simply the state it happens to be in. In this case, there is no output decoder circuit and no other output lines but its flip-flop outputs. In other cases, an output other than the state may be required. In synchronous counters, the signal that excites the counter is very often the clock. At other times, other inputs (called *control inputs*) are also provided. (It is also possible for counters to be asynchronous; such counters will be considered in Chapter 7.)

Counters can be used for a number of purposes. A common purpose is to extend the time scale—that is, to introduce delay in the inevitable march of the clock. This is done by producing an output (or control) signal for each k periods of the clock; this

Short ___
Even ___

signal, rather than the clock, then controls the timing of a subsequent operation. Another purpose of a counter is to produce sequential words in some specific code. Of course, just plain counting is an important purpose—for example, counting how many times some process has been carried out. It would be very useful if

- The counter were cleared (set to 0) when first turned on or after it has gone through its count, or
- The counter were set at some specific value

This is done by external CLEAR and RESET inputs.

Single-Mode Counters

A counter is said to be *single mode* if the only external input is the clock and the only outputs are the states—the flip-flop outputs. The counter is described by specifying

- Its modulo number
- The code assigned to the states

The number of flip-flops needed in a counter is implicit in its modulo number. Thus, a modulo-k counter will require $\lceil \log_2 k \rceil$ flip-flops. Modulo-6 or modulo-8 counters, counting in binary or Gray code, will have three flip-flops; in common terminology we call them "3-bit counters."

If the code in which the counting takes place is specified, there is no point in giving the states arbitrary names (such as letters of the alphabet) and then later making a state assignment. Rather, each successive state is given an assignment on the basis of the code being used. It is convenient to assign the starting state of the counter the code word 00...0, unless there is some reason not to do so. (Return to Chapter 1 to review the subject of codes.) As a matter of fact, the state assignment problem, so important in the machines considered so far, disappears in a counter. The codes representing the states are specified beforehand.

The number of code words is fixed by the modulo number. Several codes are shown in Figure 28 for modulo numbers 6 and 8. Starting at code word 00...0, the counter is to cycle through each code word and return to 00...0 after the last word in each code.

Unit-Distance Counters

After the modulo number of a counter has been specified, the next question that comes up is, What code should be used in making the successive state assignments? Perhaps the simplest answer is: binary. The disadvantage of this code is that more than one bit value changes in a clock period. Thus, in going from 001 to 010, both the second and the third bits (counting left to right) must change value. This means that more than one flip-flop output must change simultaneously. If it should happen that one change occurs even slightly before the other(s), there may be a momentary transition to the wrong state—or even to an invalid state, one that is not among the states of the counter. For this reason, a code whose *distance* is 1 is preferable.[17]

[17]The distance between two code words is the number of bits that must be complemented in one word to transform it to the other word. A code in which the distance between each pair of consecutive code words is k is a *distance-k* code. See Chapter 1 for a discussion of codes.

Short
Even

Binary	Unit Distance	Creeping	One-hot
000	000	000	00000
001	001	100	10000
010	011	110	01000
011	010	111	00100
100	110	011	00010
101	100	001	00001

(*a*)

Binary	Gray	Creeping	One-hot
000	000	0000	0000000
001	001	1000	1000000
010	011	1100	0100000
011	010	1110	0010000
100	110	1111	0001000
101	111	0111	0000100
110	101	0011	0000010
111	100	0001	0000001

(*b*)

Figure 28 Codes for use in modulo-6 and modulo-8 counters. (*a*) Modulo-6. (*b*) Modulo-8.

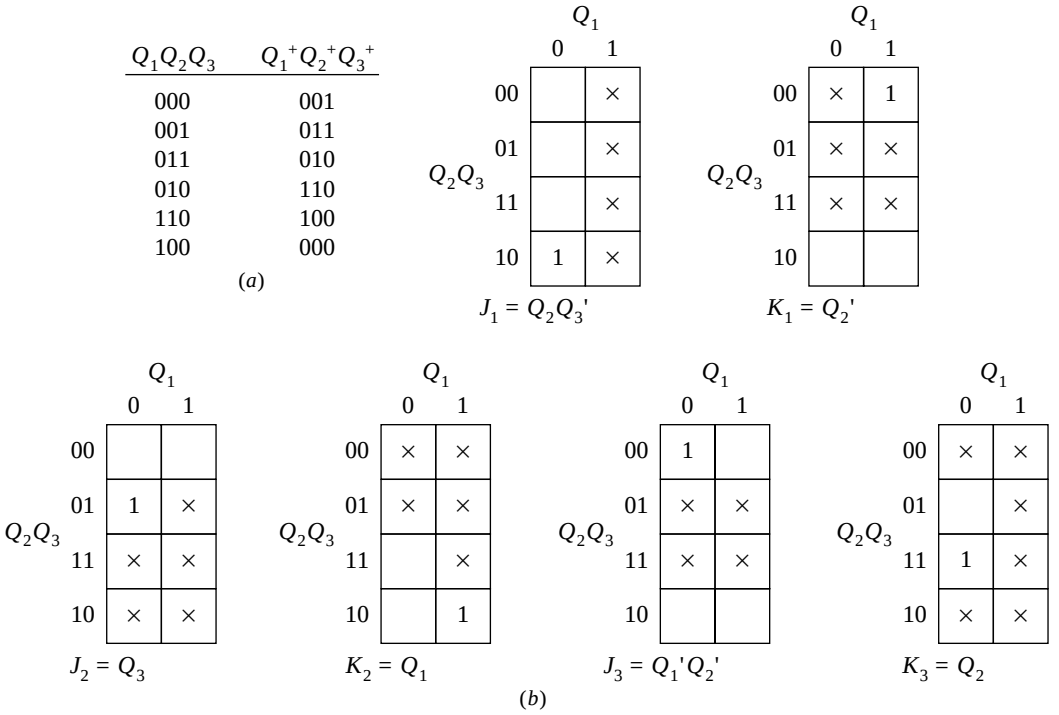


Figure 29 Three-bit, modulo-6 counter design.

The codes in the second and third columns in Figure 28*a* are unit-distance codes. Suppose the code in the second column is selected. Since there are no inputs (besides the clock) and no outputs, there will be no input/output information in the state diagram. Drawing the state diagram will be left to you (do it now). Since there is no output decoder, the only other hardware in the circuit besides the three flip-flops is the state decoder. Since the states are identified by their assignment in accordance with the code, the resulting state table is the transition table shown in Figure 29*a*. Assuming *JK* flip-flops, we use the excitation requirements tables in Chapter 5, Figure 17 to construct the logic maps for the *J* and *K* excitations. This is done line by line in the excitation table for each column. (These tables appear on the inside

Short ____
Even ____

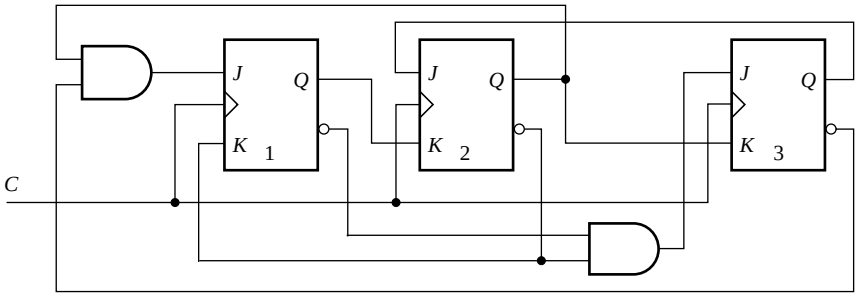


Figure 30 *JK* flip-flop implementation of modulo-6 counter.

cover of the book. You could make yourself a copy and have it handy so you can consult it without having to hunt in the book each time you want to use the tables.)

From the excitation requirements in Figure 17, Chapter 5, J = 1 only for the transition from 0 to 1. In the transition table of this example (for flip-flop 1), this occurs only when $Q_1Q_2Q_3 = 010$. Hence, a 1 is entered in the corresponding cell in the logic map for J_1 in Figure 29*b*. Also, J_1 is a don't-care for a pres-ent state $Q_1 = 1$, independent of the other states. In addition, all J's and K's are don't-cares for the present states that never occur (110, 111). All this confirms the logic map for J_1 in Figure 29*b*.

Exercise 9 Use the same approach to construct the logic maps for the other J's and K's. Confirm your results using Figure 29*b*. ◆

Exercise 10 From the maps in Figure 29*b*, construct the combinational hardware of the state decoder. Confirm your circuit with the implementation in Figure 30. ◆

Ring Counters

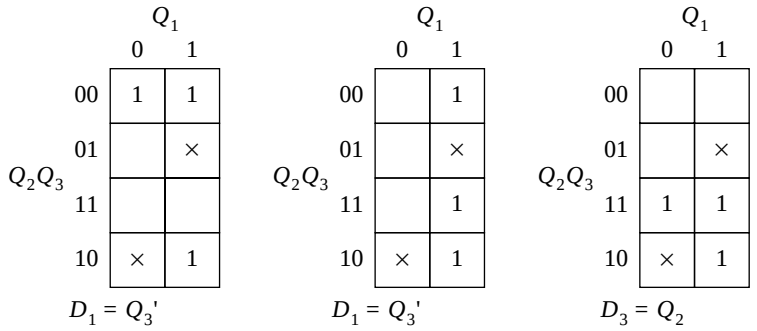
Another unit-distance code in Figure 28*a* is the creeping code. A counter designed to count in this code is called a *ring counter.* The state table, which is also the transition table (since the states are identified by their assignments), is shown in Figure 31*a*. This time, let's assume that D flip-flops are to be used. Then the excitations are the next states.

Exercise 11 Draw logic maps for the D inputs from the transition table. Confirm using Figure 31*b*, but only *after* drawing your own! ◆

It is clear that the output of one flip-flop is the excitation to the next one, except that the output of the last flip-flop (the tail) is complemented before becoming the input to the first flip-flop. (For this reason it is called a *twisted-tail counter* or some colorful variation of this.) Hence, this counter is nothing but a serial shift register with its complemented output fed into its input. For practice, draw the implementation circuit.

$Q_1Q_2Q_3$	$Q_1^+Q_2^+Q_3^+$
000	100
100	110
110	111
111	011
011	001
001	000

(*a*)



(*b*)

Figure 31 Transition table and excitation maps for twisted-tail ring counter.

Hang-up States

Ring counters have a problem that is not evident from Figure 31. What would happen if, for some reason, the counter enters one of the unused states (101 or 010; see Figure 28*a*)? This could happen when the power is first turned on, for example, or as a result of noise in the circuit. Nothing significant would happen if the next state were not an unused state. In this case, the count would resume on the next clock pulse. But, on the other hand, if each next state for subsequent clock pulses is an unused state, the counter will *hang up.* A state in which a counter hangs up so that the count cannot proceed is called (surprise!) a *hang-up state.*

Well, what is the situation in Figure 31? Because of the excitation equations, if the present state happens to be the unused state $Q_1Q_2Q_3 = 010$, then the next state will be $Q_1^+Q_2^+Q_3^+ = D_1D_2D_3 = Q_3'Q_1Q_2 = 0'01 = 101$; this is the other unused state. (By going through a similar process, show that the next state after that will be 010 again.) Once the circuit enters one of the two unused states, it hangs up and cycles between them; it will never return to the counting sequence. Hence, this counter would be defective and worthless.

Since the problem is caused by the excitation equations, resulting in a sequencing between the unused states, it can be solved by disrupting the excitation equations. Not all the equations need be modified. If we would like to retain the use of a shift register, we should concentrate on the excitation equation of only the first flip-flop. Considering the map for D_1, the hang-up problem arose because we used as a 1 the don't-care in the 010 position to form a 2-cube. Instead, let's reassign the value and take it as a 0. Then, the expression for D_1 becomes

$$D_1 = Q_2'Q_3' + Q_1Q_3'$$

For the unused present state $Q_1Q_2Q_3 = 010$, the next state of Q_1 will be $Q_1^+ = D_1 = 0$, and so the next state will be 001. We have escaped the hang-up state!

Exercise 12 Suppose the present state is the other unused state, 101. Using the new expression for D_1 and the old ones for D_2 and D_3, determine the next state and discuss whether the hang-up state is escaped. ◆

Short ____
Even ____

The ring counter design incorporating the preceding change is shown in Figure 32. It is a *self-correcting* design in that the hang-up states have been

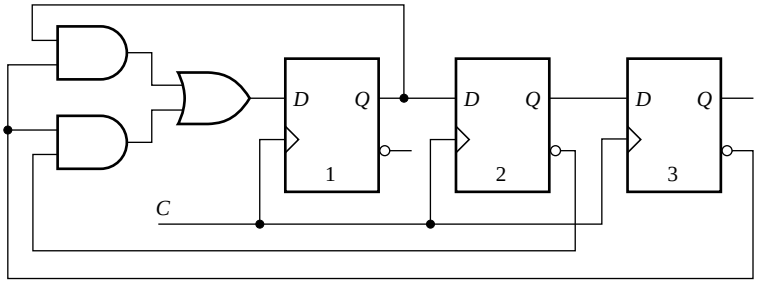


Figure 32 Self-correcting modulo-6 ring counter.

avoided. The worst-case cost of this design is a two-clock-pulse delay in the count if the counter inadvertently enters one of the two hang-up states. Although individual flip-flops have been shown, an MSI shift register is appropriate for the implementation.

Multimode Counters

A counter is called a *multimode* counter if it has, in addition to the clock, external inputs and, possibly, external outputs besides the state outputs. It is "multimode" because the counting sequence might depend not only on the clock but also on some other control signals. Likewise, special output lines besides the flip-flop outputs may be provided.

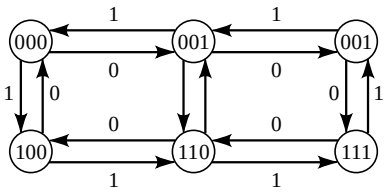
Such a counter might be used, for example, in a system in which a number of consecutive operations are to be performed. Only when one operation is completed is the next one to start. So a control signal indicating the end of a particular operation will increase the count, thus causing a transition to the state in which the counter should remain while the next operation is performed. Completion of this next operation again generates a signal that increases the count. When the last operation in the system is completed, the state should revert to the initial, or reset, state. Along the way, while a specific operation is being performed and the machine is in some particular state, the occurrence of some circumstance before the operation is completed may require returning to an *earlier* count rather than advancing. The number of operations to be performed will determine the counter's modulo number.

Modulo-6 Up-Down Counter

We will now consider an example of a multimode counter. Besides the clock, a synchronous counter is to have one input line, x. The count is to increase by one when $x = 0$ and to decrease by one when $x = 1$. Assume that six operations are to be controlled, so the modulo number is 6. Suppose, also, that the creeping code is to be used. In this example, although there is an input besides the clock, there are no other outputs besides the flip-flop outputs.

Note from the creeping code in Figure 28*a* that, from any count (state), the count advances following $x = 0$ and regresses following $x = 1$. Draw the state (transition) diagram before peeking at Figure 33*a*; then confirm it. The transi-

___ Short
___ Even

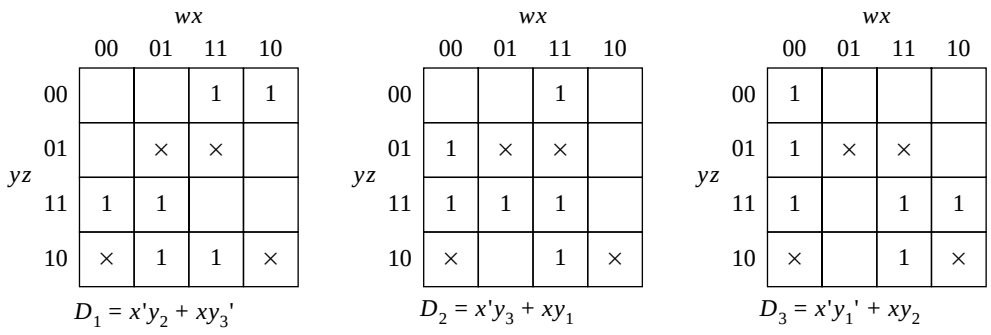


(*a*)

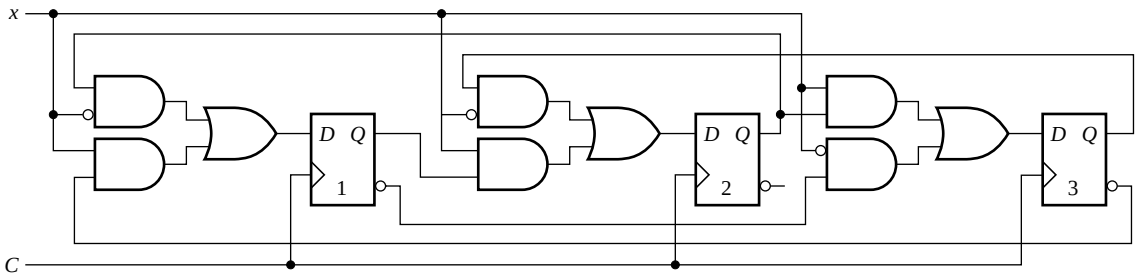
	$y_1^+y_2^+y_3^+$	
$y_1y_2y_3$	$x = 0$	$x = 1$
000	001	100
100	000	110
110	100	111
111	110	011
011	111	001
001	011	000

(*b*)

Figure 33 State table and transition table for a modulo-6 up-down counter.



(*a*)



(*b*)

Figure 34 Design of modulo-6 up-down counter.

tion table is easily constructed from the transition diagram; do it yourself before confirming it in Figure 33*b*.

Assuming the use of *D* flip-flops, the excitations are the next states. The next step is to construct the logic maps for the excitations from the transition table; do this before you peek at Figure 34*a*. The final step is the implementation. Carry this out and then confirm it using Figure 34*b*.

7 ALGORITHMIC STATE MACHINES

Short ____
Even ____

Two tools were used early in this chapter for describing a sequential machine: state diagrams and state tables. Both of these are useful. In this section we will describe another tool of great utility. Review Example 1 (a single-input, single-output circuit) and

Figure 2, where state diagrams were introduced. Starting at any state, the arrival of an input sends the machine to one of the possible next states, depending on the input. This wording is reminiscent of the condition block in a flow chart representing an algorithm. Indeed, a diagram resembling a program flow chart can be created that contains the same information conveyed by either the state diagram or the state table.

Basic Principles

Review the statement of, and state diagram for, Example 1 early in the chapter. With the machine in any specific state, the arrival of an input bit requires a decision concerning the output to be emitted and the state to which the machine is to be directed. In the state diagram of Figure 2, the condition is shown as a directed line leaving each circle representing a state. A flow chart must include analogues of

- Circles representing states
- Directed lines leading to the next states with specified outputs

Remember that a sequential circuit is referred to as a *state machine* (short for finite-state machine). Since the problem statement for the design of a state machine is like an algorithm, the machine is also called an *algorithmic state machine,* or ASM. A flow chart, called an *ASM chart,* can be constructed that describes the operation of an algorithmic state machine. A flow chart describing *any* algorithm includes a condition box where a decision has to be made; this is the familiar diamond shape shown in Figure 35*a* or the variation of it in Figure 35*b*. It is called the *decision box*. The lines leading out from a decision box are the *exit paths*.

In an ASM flow chart, the circle enclosing a state in a state diagram is replaced by a rectangle called the *state box,* as shown in Figure 32*c*. Instead of writing the state name inside, as in the state diagram, both the state name and its binary code are written above the rectangle. (In a Moore machine, outputs are associated with each state; hence, in that case, the appropriate output is written inside the state box.) Note that the two boxes described so far exist in all ASM charts.

In Mealy machines the output depends on both the input and current state. In this case another box is used in the chart, called a *conditional output box*. To distinguish it from a state box, an oval shape is used, as shown in Figure 32*d*.

In a state machine, a tick of the clock initiates action, whether a new input is present or not. Starting at each state, decisions must be made as to the state transitions and the output. A basic unit in an ASM chart can be thought of as consisting of a single state box and all other decision and conditional boxes whose exit path leads to another state. Such a unit is called an ASM *block*. So an ASM chart is simply an interconnection of such ASM blocks. Each state to which a transition is made is the beginning point of another block. For clarity, it often helps to encircle the individual blocks (using dashed lines), but doing so adds nothing to the chart. The important entities are the state, condition, and conditional output boxes; whether or not these combinations of boxes are encircled to delineate an ASM block is secondary.

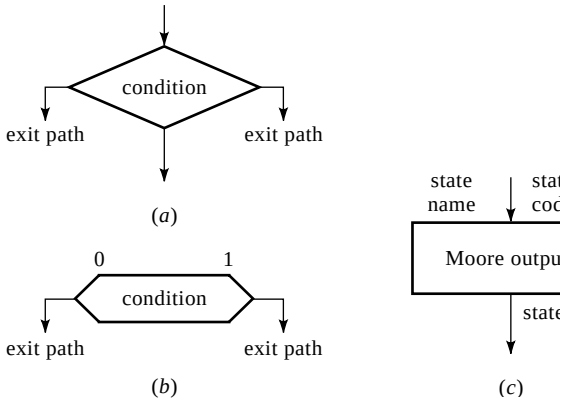


Figure 35 Boxes in ASM charts. (*a*), (*b*) Decision box. (*c*) State box. (*d*) Conditional output box.

EXAMPLE 12

An ASM chart is to be constructed for an automatic garage-door opener. Let x be the input signal resulting from a sensor actuated by a physical switch; the output is z. The value of x toggles when the switch is activated. The output signal z controls the mechanism that opens and closes the garage door; z also toggles. A low-frequency clock synchronizes the system. Assuming the garage door is closed, when the switch is activated the input becomes 1 and the machine changes state at the next clock edge. The next time the switch is activated, the input signal goes to 0. In response to this input the output goes to 0 at the next clock edge. We seek an ASM chart to describe this simple system.

The machine has two states: open and closed. An ASM block starting from the closed state is shown in Figure 36*a*. It does not indicate how this state is reached. After the "open" input signal is received, the output 1 is emitted. That means the garage door should open, so the next state is the "open" state. The next time the switch is actuated, the input toggles ($x = 0$), and so does the output ($z = 0$). The "closed" state is reached when $x = 0$ while in the open state. Figure 36*b* shows the completed chart. ■

The utility of ASM charts is not evident from this simple example. To get on with something more substantial, let's return to Example 1 at the very beginning of this chapter, describing the sequence detector.

EXAMPLE 13

In Example 1, the output z is to become 1 only after receipt of the input sequence ...0110, independent of the sequence that precedes it. (Look over that example before going on.) There are two possibilities: Either the most recent 0 bit in the preceding sequence forms the first bit of an acceptable sequence (the *overlap-*

Short ___
Even ___

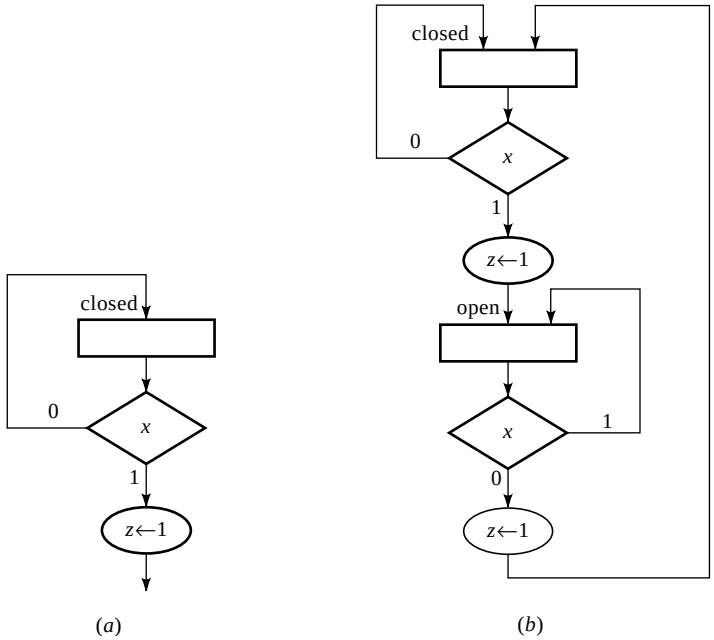


Figure 36 ASM chart for a garage-door opener.

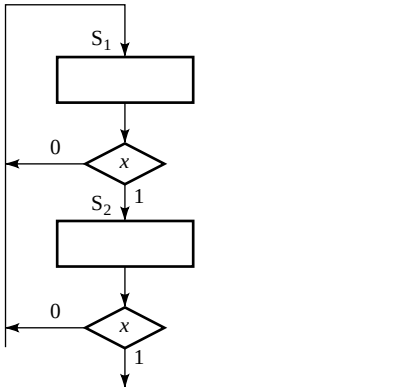
ping possibility), or the machine returns to a reset state and we start over. Obviously the design will be different in the two cases. Example 1 dealt with the overlapping case.

Let state S_1 be the state arrived at by an $x = 0$ input. The first two blocks are shown in the partial ASM chart in Figure 37*a*. If the input is 0 while the machine is in either of the first two states, transition is back to the initial state, as shown. The completed ASM chart is shown in Figure 37*b*. While in state S_3, receipt of an input $x = 1$ will spoil the acceptable sequence and will send the machine to a state where it will stay for each consecutive $x = 1$. It escapes from this state to the initial state with an input $x = 0$. However, if $x = 0$ while the machine is in state S_3, it will emit an output of 1 and will also return to S_1. ■

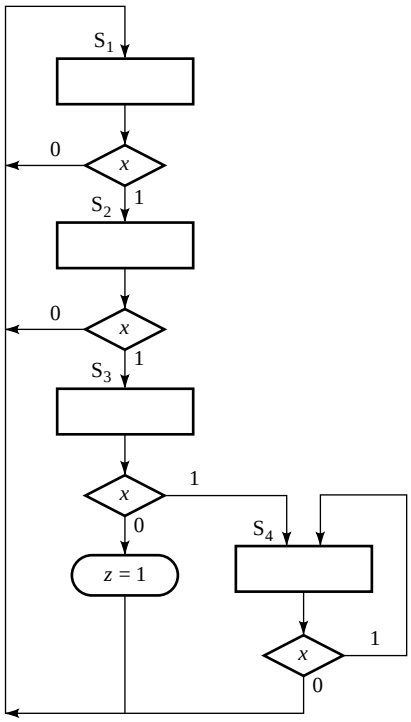
EXAMPLE 14

ASM charts are particularly useful when there are a large number of inputs (state diagrams are unreasonable to use with three or more inputs). Consider the design of a traffic light controller for an intersection that is normally busy during the day and not very busy at night. There is a preferred direction to maintain the light green, but during the day the intersection is busy enough that the car detection sensors are not used. During the day the traffic light moves through a fixed sequence of green, yellow, and red for the same duration in both directions. During the night the signal for the preferred direction stays green unless a car is detected in the nonpreferred direction. The preferred direction is north/south.

___ Short
___ Even



(*a*)



(*b*)

Figure 37 ASM chart for Example 13.

The traffic light controller has four inputs: one sensor input to detect the presence of a car on the nonpreferred route, one time-of-day input to signal day or night, and two timing inputs to determine the minimum duration of a green signal in a given direction and the duration of the yellow signal. The controller has six outputs, one for each color signal (green, yellow, and red) in each direction (north/south and east/west). The controller can be thought of as running two algorithms depending on the time of day, so the time-of-day input determines which algorithm is executed. The only difference between operation during the day and night is that the car presence sensor is used during the night to decide whether or not to maintain the light green

Short ____
Even ____

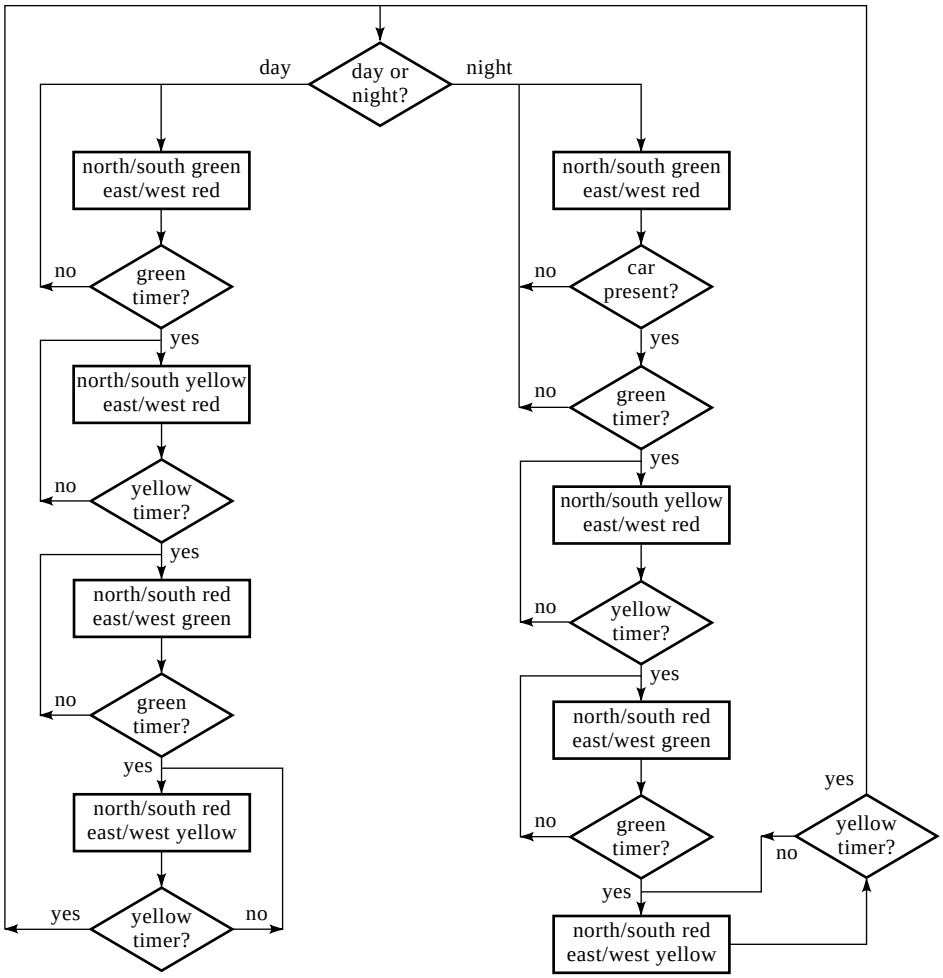


Figure 38 ASM chart for Example 14.

in the preferred direction. The ASM chart describing the system is shown in Figure 38. Before you look it over, try creating one yourself and confirm your version by comparison. ■

8 ASYNCHRONOUS INPUTS

Recall from Chapter 5 that circuit implementations of state machines must satisfy the setup (t_{su}) and hold (t_h) time requirements of flip-flops. Excitation signals must be stable for a time period t_{su} before the clock transition and must be held for a time period t_h after the clock transition. Signals generated within the state machine are not a concern, since the delay of a flip-flop is typically longer than the hold time, and the clock-cycle time can be increased to provide sufficient margin to ensure that the setup time is honored. If a faster cycle time is required, then the next state and output decoders must be redesigned to de-

___ Short
___ Even

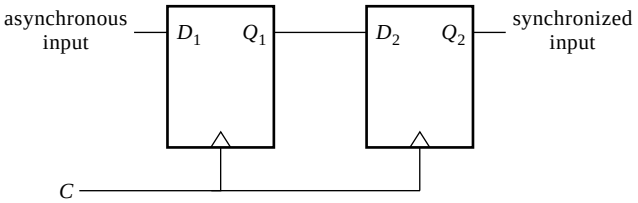


Figure 39 Synchronizing asynchronous inputs to a state machine.

crease the delay.

Signals produced outside the state machine may or may not be synchronized by the same clock. They may be signals from a sensor, switch, keyboard, or communications channel. These signals switch at arbitrary times, so there is no guarantee that they will honor the setup and hold time requirements of the flip-flops. When these flip-flop requirements are violated, the state transition is unpredictable. The flip-flop can enter an undefined state (a voltage level between low and high) for a short period of time before switching (unpredictably) to a defined state (0 or 1). The undefined state is called a *metastable state.* A typical flip-flop remains in the metastable state for a short period of time, but the stable state it reaches after metastability cannot be predicted. This can cause a state machine to make an erroneous state transition.

To decrease the probability that an asynchronous input will cause an erroneous state transition, adding an extra flip-flop, as shown in Figure 39, can synchronize the signals. In this circuit the first flip-flop can enter metastability, but it will very likely reach a stable state before the next clock event. Thus, the probability that the output of the second flip-flop has a valid state is very high (much higher than if only one flip-flop were used to synchronize the signal).

Asynchronous Communication (Handshaking)

It is sometimes necessary for two synchronous sequential circuits with different clocks to communicate data to one another. The two systems could be two computers, a computer and an input or output device, or even a CPU and memory. This communication is asynchronous, so a simple protocol is required to ensure that the data is transmitted properly. The protocol used for asynchronous communication is commonly referred to as *handshaking.* Each of two independently clocked machines M_1 and M_2 that communicate with one another uses a sequence of signals to request data from, or send data to, the other machine.

Thus, M_1 and M_2 have control signals and data signals passing between them. The machine that initiates the communication (say, M_1) sends a request signal to the other machine, M_2; it also sends a control signal indicating the desire to read or write data to machine M_2. For a read request, M_2 responds with an acknowledge signal when the data is ready for M_1. For a write request, M_1 must make the data available before sending the request to M_2. M_1 holds that data on the data connection until M_2 responds with an acknowledge signal. Communication from M_2 to M_1 works in a similar manner. The connections required to implement this

Short ___
Even ___

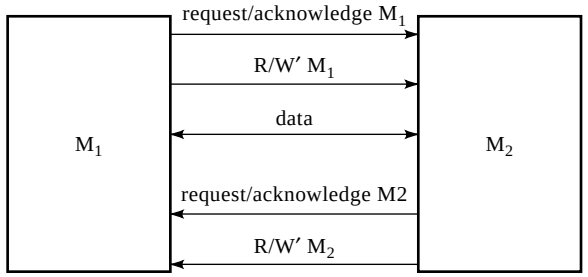


Figure 40 Connections required for asynchronous communication between two independently clocked state machines.

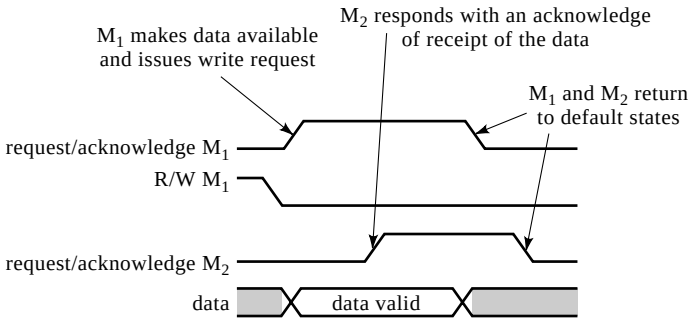


Figure 41 The timing diagram for a write operation from M_1 to M_2.

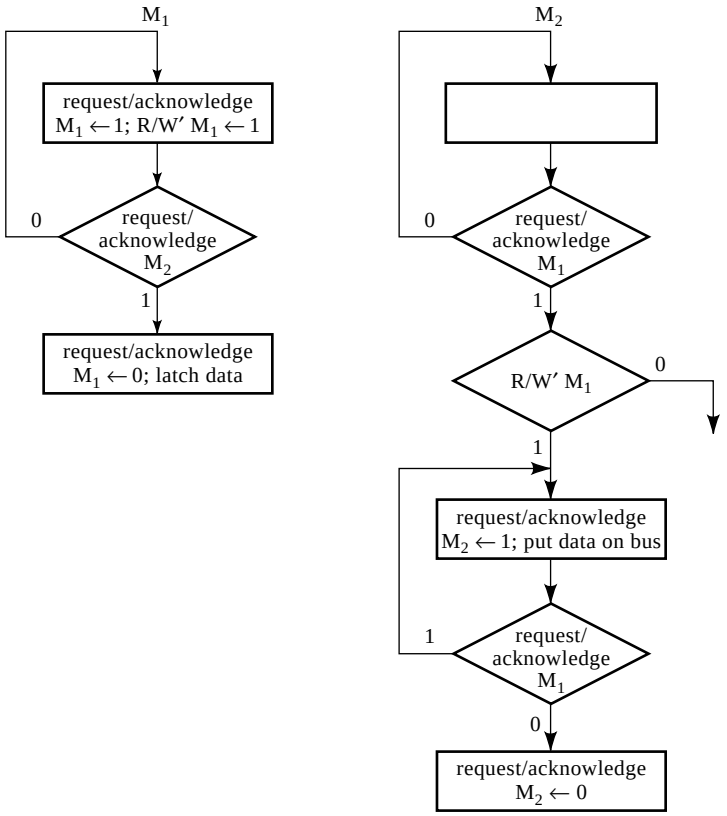


Figure 42 ASM charts for an asynchronous read request from machine M_1 to M_2.

Short
Even

protocol are shown in Figure 40. The timing diagram for M_1 to write (or send) data to M_2 is shown in Figure 41.

Exercise 13 Draw the timing diagram for M_1 to read (or receive) data from M_2. ◆

The sequence of control signals required for communication between machines M_1 and M_2 is produced by a sequence of state transitions within the machines themselves. ASM charts for a read request from M_1 to M_2 are shown in Figure 42.

CHAPTER SUMMARY AND REVIEW

This chapter introduced the design of sequential circuits of the synchronous variety. To describe how transitions from one state to another take place, we developed the tools of state diagrams, state tables, and algorithmic state machines. We introduced the class of sequential circuits called counters. Topics included the following:

- Mealy model and Moore model of a sequential machine
- State diagrams, translation of a performance specification into a state diagram, verifying the state diagram
- State table, construction of a state table from a state diagram or directly from a problem specification
- Difference between a reset state and an initial state
- Assignment of binary values to states
- Analysis of sequential circuits
- State transition and output tables
- Choice of flip-flop type in design
- Flip-flop excitation maps
- Implementation of sequential circuits from excitation maps
- Distinguishability and equivalence of states
- Minimization of machines
- Machines with finite memory spans:
  - Finite-input-memory machines
  - Finite-output-memory machines
  - Finite-memory machines
- Single-mode synchronous counters:
  - Unit-distance counters
  - Ring counters
- Hang-up states
- Multimode counters
- Algorithmic state machines
- ASM charts
- Condition box
- State box
- Conditional output

Short ___
Even ___

- ASM block
- Synchronizing machines with asynchronous inputs
- Asynchronous communications—handshaking

PROBLEMS

Many of the problems that follow require the design of a sequential machine. The design process involves several steps. As you study the chapter, you may wish to tackle early sections of each problem before you have studied everything necessary to complete the design. You can then return to each problem as you learn each successive step. Save the early parts of the solution as you go along.

1 A synchronous sequential circuit having a single input x and a single output z is to be designed. The output z is to become 1 upon completion of the input sequence 0101, whether it forms part of an overlapping string (such as 00010101) or not.

- **a.** Construct a state diagram and a state table.
- **b.** Construct an appropriate state assignment map.
- **c.** Assume the use of *JK* flip-flops and construct a transition table.
- **d.** Construct excitation and output maps.
- **e.** Draw the diagram of the resulting circuit.
- **f.** Repeat parts *c*, *d*, and *e* but with *D* flip-flops. Compare the complexity of the circuits.

2 The output z of a single-input, single-output synchronous sequential circuit is to become 1 whenever the input sequence is either 1101 or 1001. (Overlapping sequences are to yield multiple outputs.) Carry out the six parts of the design specified in Problem 1.

3 Use the same conditions as Problem 2 except that the circuit is to return to a reset state upon emitting a 1 output. (What does that do to overlapping input sequences?)

4 Carry out all design parts specified in Problem 1 for each of the following specifications.

- **a.** The output of a single-input, single-output machine is to be $z = 1$ if the present input x is the XOR of its preceding two values.
- **b.** The output is $z = 0$ when consecutive input bits of 0 are of even length and consecutive input bits of 1 are of odd length. The output is to be $z = 1$ whenever there is a discrepancy in this pattern.
- **c.** The output becomes $z = 1$ whenever the input bit is the logical product of its previous two values.

5 The input sequence of a single-input, single-output sequential machine is made up of consecutive 4-bit words. Each word is an entity; words are not formed by overlapping strings. The output is to be 1 whenever the number of 1 bits in a word is odd. Carry out all parts of the design specified in Problem 1.

6 Repeat Problem 5 except that, besides having an odd number of 1 bits, the output becomes 1 only if the 4-bit word starts with a 1. Carry out all parts of the design specified in Problem 1.

7 Construct a state table for the parity-bit generator described in Example 2 directly from the problem description, without reference to the state diagram. Compare with Figure 4.

8 A parity-bit generator is to receive 4-bit coded messages followed by a blank space (a 0). A parity bit of 1 is to be generated and inserted into the blank space if and only if the parity (the number of 1's) of the preceding 4 bits is odd.

- **a.** Construct a state diagram and a state table for this parity-bit generator.
- **b.** Test your diagram to verify that, starting from the reset state, it gives the correct output for various 4-bit messages.
- **c.** Carry out the remaining parts of the design specified in Problem 1.

___ Short
___ Even

9 A sequential machine has been found to have three states, A, B, and C; there are, then, just two state variables. Specify three different assignments of 2-bit code words to the three states, such that any other assignment amounts to either interchanging the two state variables, inverting either variable, or both.

10 After assigning the combination 000 to state A in Example 4, the adjacency AG can be satisfied by assigning G to one of two other squares in the assignment map besides 001. Choose another square to satisfy this adjacency; then use the adjacency rules to obtain an assignment different from the one in Example 4.

- **a.** Complete the implementation, using *JK* flip-flops, and compare the complexity of the state and output decoders with that obtained in Example 4.
- **b.** Find an implementation using *D* flip-flops; again compare the complexity.

11 In Example 6 start from the transition table in Figure 16*c* and assume that implementation is to be carried out with *JK* flip-flops. Construct logic maps for the *J* and *K* excitations, determine the excitation functions, and draw the resulting sequential circuit diagram. Compare the complexity with that of the implementation using *D* flip-flops in Figure 17.

12 In Example 6 (the change-of-level detector), sets of adjacencies called for by the rules of thumb are {DE, DF, EF} and {DG, EG, DE}. Only two of the adjacencies in each set can be achieved. In Example 6, the unachieved adjacencies were chosen to be DG and EF. Suppose, instead, that the unachieved adjacencies are taken to be DF and EG, all others being achieved.

- **a.** Construct the resulting assignment map.
- **b.** Again assume implementation is with *D* flip-flops; construct the transition and output tables.
- **c.** Construct the excitation maps for the *D* flip-flops.
- **d.** Using the preceding results, draw the diagram of the resulting circuit.
- **e.** Compare the amount of hardware with the circuit in Example 6 (Figure 17).

13 A single-output sequential machine has a data input x and two control inputs c_2 and c_1. The output is to equal the input but delayed by one, two, three, or four clock pulses, as determined by the control-input code c_2c_1 = 00, 01, 10, 11, respectively. Write an expression for the output function and design the circuit, explaining each step.

14 A 4-bit serial-in, parallel-out right-shift shift register with asynchronous preset has its initial state preset at $y_3y_2y_1y_0 = 1101$, where y_0 is the state of the flip-flop at the input of the register. There is no external input except the clock, the register's excitation coming from the state decoder only. The desired output sequence is 110111001000; it repeats after these 12 bits. Design the combinational logic (state decoder) as a minimal circuit. An appropriate diagram is shown in Figure P14.

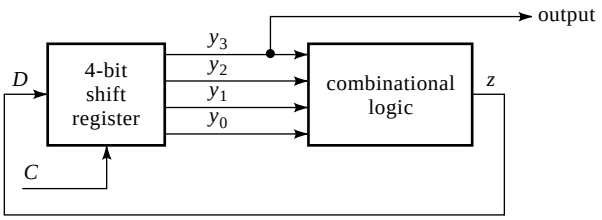


Figure P14

15 The same structure as the circuit in Figure P14 is to be used, but this time the excitation of the shift register is to be

$$D = y_1y_3 + y_2y_4 + y_3y_4$$

Assume that the initial state of the shift register has been set at 0101. Find the output sequence from the shift register.

Short ____
Even ____

16 A combinational logic circuit is to be designed with a single output, D. This output is to become the input to a D flip-flop.

- **a.** Suppose that this circuit has two inputs, J and K. Design the circuit so that, together with the D flip-flop, it constitutes a JK flip-flop.
- **b.** Suppose, instead, that the combinational logic circuit has a single input labeled T. Design the circuit so that, together with the D flip-flop, it constitutes a toggle flip-flop.

17 A synchronous sequential circuit has two input lines, x_1 and x_2, and an output line, z. The data line is x_1 and x_2 is a reset line. Whenever $x_2 = 1$, the circuit is reset. When x_2 becomes 0, the first 4 bits on the message line constitute a message word. The output is to become 1 if the message received is 1010. At the end of the fourth bit of any word received when $x_2 = 0$, the circuit is to enter a waiting state, where it remains until it is reset and where the output is 0 for any input bits after the fourth bit.

- **a.** Construct a state diagram and a state table.
- **b.** Carry out the rest of the design and implement it using JK flip-flops.
- **c.** Construct a timing diagram showing the clock and the inputs and outputs.

18 Repeat Problem 17 if the message is to be 1100, using

- **a.** D flip-flops
- **b.** JK flip-flops

19 Repeat Problem 18 if the word length is 5 bits and the message is to be 11011.

20 Design a sequential machine that is to have all of the following characteristics:

No 2 consecutive output bits can be 1.
The output bit cannot be 1 upon receipt of the second of 2 consecutive input 0 bits.
The output will be 1 if either of 2 consecutive input bits is 1, unless it conflicts with the first requirement.

A possible input/output sequence, for example, is the following:

x: 0010101111010000
z: -010101010101000

21 Design a single-input, single-output synchronous sequential circuit that is to have the following characteristics.

No 3 consecutive output bits can be 1.
The output will be 1 if, of 3 consecutive input bits, just two are 1, unless it conflicts with the first requirement.

A possible input/output sequence, for example, is the following:

x: 000011010111001010110110
z: - - 000110011010000101101l0

22 Design a sequential machine for which the output becomes 1 if and only if just 1 of the following 3 bits is a 1: the present input bit and the last 2 output bits. A possible input/output sequence, for example, is the following:

x: - - 001011011100101001001
z: 110101000100001101101l0

___ Short
___ Even

23 A sequential machine having the following properties is to be designed. It has two inputs: x and c, a control input. When $c = 0$, the machine returns to a reset state A, independent of x. With the machine in reset state A, whenever $c = 1$, the values of x at the next three clock pulses will constitute a binary word. An output of $z = 1$ is to occur at the third bit if and only if all 3 bits are the same: 000 or 111. Otherwise the output is to be 0. The state entered by the machine at the third bit of an input word is a waiting state. The machine does not leave this state until $c = 0$ occurs, at which time it is reset.

- **a.** Construct a state diagram and state table for this machine.
- **b.** If possible, reduce this table to one having the fewest states.
- **c.** Carry out the necessary steps to arrive at excitation equations, assuming the machine is to be implemented with D flip-flops. Then draw the corresponding circuit diagram.
- **d.** Repeat part *c* for the case where the machine is to be implemented with JK flip-flops.

24 **a.** Given the state table in Figure P24, assume the following input sequence with the machine initially in state A:

10011101011001

Determine the resulting output sequence.

- **b.** Obtain a minimal reduced state table equivalent to the given one, letting new state A be the block in which the old state A appears.
- **c.** Again starting in state A of the reduced table, assume the same input sequence and find the resulting output sequence. Compare the result with the one in part *b*. Are you surprised?

	NS,z	
PS	$x = 0$	$x = 1$
A	B,0	G,0
B	B,1	H,1
C	F,1	D,0
D	B,0	H,0
E	F,1	D,0
F	F,0	C,1
G	E,0	A,0
H	E,0	A,0

Figure P24

25 **a.** Design a single-input, single-output synchronous sequential circuit that produces an output of 1 whenever there is an odd number of 1's in the latest three input symbols.

- **b.** Draw several timing diagrams, assuming different combinations of inputs.

26 Design a single-input, single-output synchronous sequential circuit that generates an output of 1 whenever the latest 4 input symbols correspond to a binary number that is

- **a.** A multiple of 3
- **b.** A multiple of 5
- **c.** A multiple of either 3 or 5

27 Data appearing on a line synchronized with a clock should never have three or more consecutive 0's or four or more consecutive 1's.

- **a.** Design a sequential circuit that will detect such sequences and generate an output of 1 whenever they occur.
- **b.** Construct appropriate timing diagrams for different combinations of inputs.

Short ____
Even ____

28 The output of a synchronous sequential circuit is to be the same as the input but delayed by three or four clock periods under the control of a second input, c. The delay is to be three clock periods when $c = 0$ and four clock periods when $c = 1$.

- **a.** Design the circuit.
- **b.** Construct timing diagrams for the two cases.

29 A synchronous sequential circuit has two input lines, w and x, and a single output line, z. Let $W = w_2w_1w_0$ and $X = x_2x_1x_0$ be 3-bit sequences on the input lines representing binary numbers, the most recent bits being w_2 and x_2. The output is to be 1 whenever $W \geq X$.

- **a.** Design the circuit.
- **b.** Draw timing diagrams for several input word combinations.

30 A synchronous sequential circuit has two data inputs a and b, a control input c, and a single output z. The output is 0 except that $z = 1$ under either of two conditions:

$c = 0$ and a and b had identical values two clock periods earlier
$c = 1$ and $a = b'$ three clock periods earlier

- **a.** Design the circuit.
- **b.** Construct a timing diagram, showing the clock, input, and output signals for the two values of c.

31 A single-input, single-output synchronous sequential circuit is to generate an output of 1 whenever x has the same value it had three clock periods earlier; otherwise the output is to be 0.

- **a.** Design the circuit.
- **b.** Construct timing diagrams showing the clock, input, and output signals for the two cases.

32 A single-input, single-output synchronous sequential circuit is to generate an output of 1 whenever any of the following input sequences occurs: 011, 1001, 11011. The output is to be 0 otherwise.

- **a.** Design the circuit.
- **b.** Draw timing diagrams for the possible input sequences, showing the clock, input, and output signals.

33 A single-input, single-output machine is to have outputs as follows:

$$z(t) = z(t-1)z(t-2) \quad \text{when } x(t) = 0$$
$$z(t) = z(t-3) \quad \text{when } x(t) = 1$$

If this machine has a finite memory span, specify its class and obtain a canonic implementation.

34 Modify Problem 33 by introducing a control input, c. The output specified in Problem 33 is to be obtained when $c = 1$. When $c = 0$, on the other hand, the output is to be 1 whenever the last 2 input bits are identical. Obtain a canonical implementation.

35 For each of the following machines, determine if it is a finite-memory machine. If it is, determine its type (finite-input-memory, finite-output-memory, or neither) and its order.

- **a.** *Serial parity generator:* The machine receives data bit-serially on its input and indicates on its output if the total number of 1's received so far is even or odd.
- **b.** *Serial adder:* The machine has two inputs and receives two binary numbers bit-serially on these input lines, least significant bit first. When bits of weight 2^i (for some i) are being received, the output is to be the sum bit of the same weight.

Short
Even

c. *Serial multiplier by a constant:* The machine receives a binary number bit-serially on its input, least significant bit first, and multiplies it by a fixed constant k (which need not be a power of 2). When an input bit of weight 2^i is being received, the output must be the product bit of the same weight.

d. *Divisible-by-k indicator:* The machine receives a binary number bit-serially on its input, least significant bit first, and indicates its divisibility by a fixed constant k that is not a power of 2. The output is 1 if and only if the binary number received up to that clock period (including the current input) is divisible by k.

e. Repeat part *d* assuming that k is a power of 2.

f. Repeat part *d* assuming that the binary number is received *most significant bit* first.

g. Repeat part *e* assuming that the binary number is received *most significant bit* first.

36 Sequential machines M_1 and M_2 are finite-input-memory of order m_1 and m_2, respectively. A new machine M is obtained by cascading M_1 with M_2. That is, the output of M_1 is the input of M_2. The input and output of M are, respectively, the input of M_1 and the output of M_2. Determine if M is a finite-input-memory machine and, if it is, determine its order.

37 M_1 and M_2 are finite-input-memory machines of order m_1 and m_2, respectively. A new machine M is obtained as follows. The inputs of M_1 and M_2 are tied together and constitute the input to M. The outputs of M_1 and M_2 are brought out separately and together form the output of M. Determine if M is a finite-input-memory machine and, if it is, determine its order.

38 Sequential machines M_1 and M_2 are finite-output-memory machines of order m_1 and m_2, respectively. A new machine M is constructed as in Problem 37. Determine if M is a finite-output-memory machine and, if it is, determine its order.

39 Odd-length counters do not have unit-distance codes, but almost do, having only one transition (usually the pivotal one in the middle of the count) in which more than 1 bit must change.

a. Design a self-correcting modulo-m ring counter, where $m = 5$.

b. Repeat for $m = 7$.

c. Repeat for $m = 9$.

40 The *creeping code* is a 5-bit code for decimal digits generated as follows. The code for digit 0 is 00000. The code for any digit d_i is obtained from the code for the preceding digit d_{i-1} by first setting the msb of d_i equal to the complement of the lsb of d_{i-1}, and then setting the lower 4 bits of d_i equal to the upper 4 bits of d_{i-1}, in the same order. (See the section in Chapter 1 on codes.)

a. Using *D* flip-flops, design a synchronous modulo-10 counter that counts in creeping code; draw the circuit.

b. Modify the design so that the circuit has an output $z = 1$ whenever the count is either 4 or 7.

41 A multimode counter has one pulse input line x that is synchronized with the clock and two level output lines f and g that respond to the rising edge of the clock. Level changes on the input line are separated by at least four clock periods. The operation of the counter is to be as follows:

f becomes 1 at each clock pulse.
g becomes 1 two clock pulses later.
f goes to 0 at the next clock pulse after g becomes 1.
g goes to 0 at the next clock pulse after f goes to 0.

a. Construct a timing diagram showing the clock waveform and the waveforms of x, f, and g.

b. Design the counter using a distance-1 code and draw the circuit.

c. Design the counter using the creeping code, making sure that it is self-correcting.

Short ____
Even ____

42 The schematic diagram of a universal left/right shift register is shown in Chapter 5, Figure 24; this one is a 4-bit register.

- **a.** Draw a transition diagram (a state diagram whose states are already assigned codes) for a 3-bit universal register. Starting in any state, either a 0 or a 1 can be shifted either to the left or to the right. Hence, there will be four arcs leaving each node, indicated by 0L, 0R, 1L, and 1R.
- **b.** Notice how a standard ring counter sequence of length 4 and a twisted-tail ring counter sequence of length 6 are generated from this transition diagram.
- **c.** Design a decoder circuit for a 3-bit universal register so that the combined circuit with a schematic similar to Figure 24 in Chapter 5 is a modulo-m counter, starting with 000. Take

  - $m = 4$ (two possibilities)
  - $m = 5$ (four possibilities)
  - $m = 6$ (six possibilities)
  - $m = 7$ (two possibilities)
  - $m = 8$ (four possibilities)

- **d.** Draw a transition diagram for a 4-bit universal register and notice how a standard ring counter sequence of length 5 and a twisted-tail ring counter sequence of length 8 are generated from this.
- **e.** Design a decoder circuit for a 4-bit universal register so that the entire circuit in Figure 24 in Chapter 5 is a modulo-m counter starting with 0000. Take $m = 8$; how many possible sequences of length 8 are there?
- **f.** Repeat *e* for $m = 9$.

43 Suppose that the counter implemented by *JK* flip-flops in Figure 30 is to be implemented by *T* flip-flops, obtained from *JK* flip-flops by setting $J = K$.

- **a.** Find the excitation maps (maps of T) of the three flip-flops.
- **b.** From these, determine the state decoder.
- **c.** Compare the hardware requirements with those using the *JK* flip-flops.

44 A state machine, with two inputs A and B and a single output C, is to be designed. The output is to become 1 only if the number of input 1's since the machine was reset is an exact multiple of 4. It doesn't matter on which input line a 1 occurs.

- **a.** Construct a state diagram. (As you go about this task, think of the following things. Is 0 a multiple of 4? With the machine in a particular state, what difference in next state and output would there be for inputs $AB = 01$ or 10? To what state would the machine go if, having already received three 1's, the next input is $AB = 11$? In such an event, what would the output be?)
- **b.** How many different assignments are possible? Select an appropriate assignment.
- **c.** Assume the use of *D* flip-flops and construct excitation maps.
- **d.** Write expressions for the excitations and the output.
- **e.** Draw a circuit implementing these expressions.
- **f.** If you want, try another assignment and repeat parts *c*, *d*, and *e*. Compare the two realizations. Was this fun?

45 For each table in Figure P45, the overall objective is to construct a minimal reduced table equivalent to the original one.

- **a.** Partition the states so that all states in a partition are 1-equivalent.
- **b.** Refine the partitions so that all states in the new partitions are 2-equivalent.

___ Short
___ Even

c. Continue refining the partitions until an equivalence partition is obtained.
d. Construct a reduced state table with each final partition as a state.
e. Compare the number of flip-flops needed in implementing both the original tables and the reduced tables.
f. Implement each reduced table using *JK* flip-flops.
g. Repeat *f* using *D* flip-flops.
h. Using *D* flip-flops, implement the table in Figure P45*b* before reduction. Compare the number of flip-flops and circuit complexity for the two implementations.

	NS,*z*	
PS	$x = 0$	$x = 1$
A	A,0	C,0
B	D,1	A,0
C	F,0	F,0
D	E,1	B,0
E	G,1	G,0
F	C,0	C,0
G	B,1	H,0
H	H,0	C,0

(*a*)

	NS,*z*			
PS	$x_1x_2 = 00$	$x_1x_2 = 01$	$x_1x_2 = 11$	$x_1x_2 = 10$
A	B,0	G,1	C,1	D,0
B	A,1	E,0	D,1	G,1
C	H,0	G,1	A,1	C,0
D	H,0	G,1	C,1	D,0
E	C,1	H,0	D,1	C,1
F	D,1	H,0	C,1	G,1
G	H,1	G,1	A,1	F,0
H	D,1	E,0	A,1	G,1

(*b*)

Figure P45

46 A state machine has a single input *N* and a single output *D*. Four-bit messages arrive at the input. The purpose of the circuit is to detect when a 4-bit message is not a BCD word. That is, $D = 1$ whenever the 4-bit word is not a decimal number in BCD code. Assume that the circuit returns to its initial (reset) state at the end of each 4-bit word.

a. Construct a state diagram and a state table. (Confirm that your diagram produces the correct outputs.)
b. By partitioning, reduce the table to a minimum.
c. Choose two 4-bit words, one that is and one that isn't a decimal number in BCD code. Draw timing diagrams for these two cases.

47 Modify Problem 46 as follows. The 4-bit words are not consecutive; when the last bit of a word is received, the machine enters a waiting state. While it is in this state, the signal that another 4-bit word is coming is the appearance of 3 consecutive 1 bits. Upon receipt of the third 1 bit, the machine enters the reset state, ready for the next 4-bit message.

a. Construct a new state diagram and a new state table.
b. Reduce the state table by partitioning the states.

48 **a.** Design a synchronous BCD counter. (It might be called a modulo-10 counter.) The only input is the clock. Draw a timing diagram that includes the clock and all flip-flop output waveforms.
b. Modify the design for a counter that is to be just one decade of a decimal BCD counter. That is, each decade is to represent a decimal digit in the 10^k position of a decimal number.

Short ____
Even ____

49 A certain binary signal consists of a periodic sequence of pulses having the same width as the clock pulse, synchronized with the clock. For a certain application, it is expected that the number of bits in a string of 0's is odd and the number of 1's is even. A state machine is to be

designed to detect errors from this configuration. That is, $z = 1$ whenever an even string of 0's or an odd string of 1's is detected.

- **a.** Construct a state diagram and a state table. (Think about how the machine will know that a string of like bits has ended.)
- **b.** If your table is not minimal, reduce it.
- **c.** Make an "optimal" state assignment and construct a transition table.
- **d.** Assume the use of *D* flip-flops and write expressions for the excitations.
- **e.** Draw a circuit diagram implementing these expressions.

50 A synchronous sequential circuit has two input lines, x_1 and x_2, and two output lines, z_1 and z_2. At each clock tick, the combination x_1x_2 constitutes a 2-bit binary number. If the present value of the input number is less than its immediately preceding value, then the outputs are $z_1z_2 = 10$. If the present value is greater than the preceding value, then $z_1z_2 = 01$. If it is the same, $z_1z_2 = 00$.

- **a.** Design the circuit.
- **b.** Draw timing diagrams for the three cases.

51 Five-bit words arriving on a line are expected to be messages in 2-out-of-5 code. However, there may be errors. A synchronous machine is to be designed whose output is 1 only when the fifth bit is received and the completed word is not a valid word in 2-out-of-5 code. The 5-bit words are consecutive; as soon as one 5-bit word is completed, the circuit should be ready to receive the first bit of the next word.

- **a.** Construct a state diagram. (*Hint:* To how many distinct states can the circuit make a transition for each incoming bit after the first?)
- **b.** Construct a state table.
- **c.** Make an appropriate state assignment and construct a transition table.
- **d.** Assuming the use of *D* flip-flops, obtain expressions for the excitation and output functions.
- **e.** Construct timing diagrams for the clock, the input bits, and the resulting output.

52
- **a.** A synchronous sequential machine has one input line x and one output line z. The machine is intended to receive a binary number of unknown length on the input line, with the *least* significant bit first, and to indicate on z its divisibility by 5. That is, for any time t, $z(t) = 1$ if and only if the binary number $x(t) \ldots x(0)$ is divisible by 5. Construct a state table for such a machine and minimize the number of states.
- **b.** Generalize part *a*: If divisibility by a number p is to be detected, where p is a known constant, determine a tight upper bound in terms of p on the number of states needed in the machine.
- **c.** Repeat part *a* assuming that the *most* significant bit is received first.
- **d.** Repeat part *b* assuming that the *most* significant bit is received first.

53 Figure P53 shows a schematic diagram of a synchronous modulo-10 counter whose states are 0–9. The Q_3–Q_0 outputs represent the count. If CE ("count enable") is 1, then the counter increments to the next state at the next clock pulse. Otherwise it retains the current count. The output TC is 1 if and only if the count is 9.

The objective of this problem is to design a modulo-10^{16} counter by stringing together several modulo-10 counters. It is claimed that the CE inputs to the modulo-10 counters in the string are analogous to the carry inputs to the full adders in a multibit adder, and hence, the carry-lookahead principles can be used in designing the modulo-10^{16} counter.

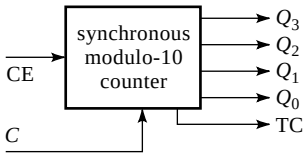


Figure P53

___ Short
___ Even

a. Derive the generate and propagate expressions for a modulo-10 counter.
b. Suppose 4-bit lookahead units of the type shown in Figure 8, Chapter 4, with inputs P_1–P_4, G_1–G_4, and C_1 and outputs C_2–C_5 are available. Using these and the modulo-10 counters (and no other logic) design a lookahead modulo-10^{16} counter.
c. Obtain a modulo-10^{16} counter by replacing the lookahead units in the answer to *b* by only AND gates.

54 **a.** Design a modulo-2 (2-bit) binary counter using *D* flip-flops. The counter goes through the sequence 00 01 10 11 00 ... . The machine is also to have an output line that emits a 1 at the count 11.
b. Draw an appropriate timing diagram.

55 A state machine is to have a single input line and a single output line. The output is to remain 0 until the last bit of either of the sequences ...0000 or ...1111 occurs, at which time the output becomes 1.

a. Construct a state diagram and a state table; then reduce the table to one having the fewest states.
b. Construct an ASM chart.
c. Assuming implementation with *D* flip-flops, construct the transition tables and, from these, construct a circuit implementation.

56 A certain state machine is to have the function of detecting when an incoming string of 7 bits is not the biquinary code for a decimal digit. The machine has two inputs: DATA and CONTROL. DATA consists of 7-bit words that are to represent the decimal digits in biquinary code. CONTROL is a signal that initiates an examination of DATA. When CONTROL = 0 for one or more clock ticks, the output remains 0. When CONTROL becomes 1 and stays 1, the machine is to examine the next 7 bits in DATA. Meanwhile the output remains 0; it becomes 1 only if the seventh bit completes a word that is *not* a decimal digit in biquinary code.

a. Construct a state table for this machine.
b. Assuming the use of *D* flip-flops, construct transition tables. Using these tables, design a circuit implementation.

57 A sequential comparator, with two input lines x and y and a single output line z, is to be designed. X ($x_n x_{n-1} x_{n-2}$) is a 3-bit word on line x and, similarly, Y ($y_n y_{n-1} y_{n-2}$) is a 3-bit word on line y. Taking X and Y as 3-bit binary numbers, the output is to be 1 only if $X \geq Y$.

a. Construct a state diagram and a state table for this machine.
b. Assume *D* flip-flops are to be used. Construct transition tables and, from these, a circuit implementation.
c. Repeat *b* using *JK* flip-flops.
d. Someone suggested implementing the circuit with two parallel-read shift registers and some combinational logic. Carry out this suggestion.

58 The objective of this problem is to design a Moore-model modulo-8 up-down counter. (Modulo-8 means that the machine counts from 0 to 7 in binary. "Up-down" means that when the count advances from 7 (111) it goes to 0 (000), and when it drops from 0 it goes to 7.) Besides the clock, the machine is to have a single input, x. When $x = 0$, the count will drop by 1 from its present value and, when $x = 1$, the count will increase by 1 from its present value, both occurring at the clock tick. Assume that *D* flip-flops are to be used and that there is no output decoder, the states being the outputs of the flip-flops taken as a binary number.

a. Draw a diagram showing the three flip-flops and the state decoder as a rectangle. (Can you identify the nature of this machine?)
b. Construct a transition table directly rather than using arbitrary names for the states and making a state assignment later; use the binary values of the count to identify the present and next states.

Short ____
Even ____

c. Construct logic maps for each next state.
d. Design the state decoder and complete the implementation.
e. Using arbitrary times of input changes relative to the clock, draw timing diagrams showing the clock pulses, the input, and the flip-flop outputs.

59 The objective is to design a modulo-8 up-down counter with a single input x and three output lines. The binary number represented by the outputs $z_2z_1z_0$ is the count. It is to increase by 1 when $x = 1$ and decrease by 1 when $x = 0$. Design the circuit.
60 The purpose is to design a 3-bit binary up counter with no other inputs but the clock. At each clock tick, the counter cycles through the sequence 000, 001, 011, 111, 101, 100, after which it repeats the sequence. The other two possible states are not to occur.

a. Using the state codes as state "names," construct a state table directly. Decide how to handle the entries corresponding to the rows of combinations that are not to occur.
b. Assuming the use of D flip-flops, the next state for each position in any row is the same as the required value of D. Construct logic maps for the required value of D in terms of the present states. From these, write an expression for each D.
c. Construct the circuit diagram to implement the counter.
d. Draw a timing diagram, showing waveforms for the clock and for the outputs of the three flip-flops.
e. Now assume toggle (T) flip-flops are to be used. From the excitation requirements for T flip-flops in Figure 17, Chapter 5, construct new logic maps for each T and construct a circuit diagram implementing the counter. Compare this with the implementation using D flip-flops. Show if there will be any changes in the timing diagrams.

61 **a.** Repeat Problem 60 if the counter sequence is to be the following: 000, 010, 001, 011, 101, 100, 000, ….
b. Repeat Problem 60 if the counter sequence is to be the following: 000, 011, 111, 101, 001.

62 The state table in Figure P62*a* is to be implemented with two Lemon flip-flops.

	NS,z	
PS	$x = 0$	$x = 1$
A	C,0	B,1
B	A,0	A,0
C	A,0	D,0
D	C,0	A,1

(a)

	NS,z	
PS	$x = 0$	$x = 1$
A	C,0	B,0
B	A,1	C,1
C	B,0	D,0
D	C,1	C,0

(b)

Figure P62

a. Using the results of Problem 18 in Chapter 5, specify all possible state assignments, justifying your response.
b. Choose one of the possible assignments and carry out a circuit implementation.
c. Repeat part *b* for a different assignment. Are there reasons for selecting one possible implementation over the other?
d. Repeat each part for the state table shown in Figure P62*b*.

63 A counter is to have a single 1-bit control input C. When $C = 0$, the 3-bit counter is to sequence through the binary code. When $C = 1$, it is to sequence through the Gray code.

Binary code: 000 001 010 011 100 101 110 111 → 000
Gray code: 000 001 011 010 110 111 101 100 → 000

The only outputs are the flip-flop outputs representing the states.

___ Short
___ Even

a. Construct a state diagram, labeling the states by their 3-bit codes. Show the transitions to the appropriate next states for each C.
b. Draw an ASM chart for the counter.
c. Suppose the present state is 000 when the control input takes on the following sequence.

C: 1 1 0 0 0 1 0 0 0 1 1

Construct a table with the input as column 1, the present-state code as column 2 and the next-state code as column 3.
d. Draw a circuit implementing the counter.

64 A synchronous sequential circuit has a single output z and two inputs, x and r. The output is a delayed version of the x input under the control of r. When $r = 1$, the output equals what the x input was three clock periods earlier. When $r = 0$, the output equals what the x input was two clock periods earlier. Design the circuit using appropriate flip-flops. Show and explain all intermediate steps.

65 A state machine has a single output and—besides the clock—three inputs: a data input x and the outputs of a modulo-4 counter c_1 and c_0. The output of the machine is to equal the data input but delayed by a number of clock periods determined by the count c_1c_0: the delay in output is one clock period at count 00, two clock periods at count 01, and so on.

a. Use whatever you need (state diagram, state table, logic maps, etc.) to arrive at an expression for the output. Explain your reasoning.
b. Find an implementation of the circuit.

66 A customer has placed an order from your engineering design shop for a single-input, single-output synchronous sequential machine. The output is to become 1 whenever, starting at some time, the number of input 1's exceeds the number of input 0's. An example is

x: 0 1 1 0 1 1 0 0 ...
z: 0 0 1 0 1 1 1 1 0 ...

Either (a) provide a statement as to why such a machine is impossible or (*b*) construct a state diagram of a machine that satisfies this requirement. In the latter case, construct a state table and implement it.

67 A synchronous sequential circuit is to have three inputs: A, B, and C. The single output z is to be 0 except for the following possible inputs:

$z = 1$ when $C = 0$ and A and B had identical values two clock periods earlier.
$z = 1$ when $C = 1$ and A and B had opposite values three clock periods earlier.

Obtain an implementation of the circuit and discuss its nature.

68 **a.** In Example 7, use the expressions for the flip-flop excitations and the output z on page 218 to construct a circuit realization.
b. Analyze the resulting circuit to verify the original expressions.
c. If these expressions cannot be verified by your circuit, repeat part *a* until verification is achieved.

69 Design a modulo-8 up-down counter. The count is to appear in BCD code on three output lines as $z_2z_1z_0$.

Short ___
Even ___