chapter

8

Synchronous Sequential Circuits

Chapter Objectives

In this chapter you will learn about:

- Design techniques for circuits that use flip-flops
- The concept of states and their implementation with flip-flops
- Synchronous control by using a clock signal
- Sequential behavior of digital circuits
- A complete procedure for designing synchronous sequential circuits
- VHDL specification of sequential circuits
- The concept of finite state machines

In preceding chapters we considered combinational logic circuits in which outputs are determined fully by the present values of inputs. We also discussed how simple storage elements can be implemented in the form of flip-flops. The output of a flip-flop depends on the state of the flip-flop rather than the value of its inputs at any given time; the inputs cause changes in the state.

In this chapter we deal with a general class of circuits in which the outputs depend on the past behavior of the circuit, as well as on the present values of inputs. They are called *sequential circuits*. In most cases a clock signal is used to control the operation of a sequential circuit; such a circuit is called a *synchronous sequential circuit*. The alternative, in which no clock signal is used, is called an *asynchronous sequential circuit*. Synchronous circuits are easier to design and are used in a vast majority of practical applications; they are the topic of this chapter. Asynchronous circuits will be discussed in Chapter 9.

Synchronous sequential circuits are realized using combinational logic and one or more flip-flops. The general structure of such a circuit is shown in Figure 8.1. The circuit has a set of primary inputs, W, and produces a set of outputs, Z. The values of the outputs of the flip-flops are referred to as the *state*, Q, of the circuit. Under control of the clock signal, the flip-flop outputs change their state as determined by the combinational logic that feeds the inputs of these flip-flops. Thus the circuit moves from one state to another. To ensure that only one transition from one state to another takes place during one clock cycle, the flip-flops have to be of the edge-triggered type. They can be triggered either by the positive (0 to 1 transition) or by the negative (1 to 0 transition) edge of the clock. We will use the term *active clock edge* to refer to the clock edge that causes the change in state.

The combinational logic that provides the input signals to the flip-flops derives its inputs from two sources: the primary inputs, W, and the present (current) outputs of the flip-flops, Q. Thus changes in state depend on both the present state and the values of the primary inputs.

Figure 8.1 indicates that the outputs of the sequential circuit are generated by another combinational circuit, such that the outputs are a function of the present state of the flip-flops and of the primary inputs. Although the outputs always depend on the present state, they do not necessarily have to depend directly on the primary inputs. Thus the connection shown in blue in the figure may or may not exist. To distinguish between these two possibilities, it is customary to say that sequential circuits whose outputs depend only on the state of the circuit are of *Moore* type, while those whose outputs depend on both the state and the primary inputs are of *Mealy* type. These names are in honor of Edward Moore and George Mealy, who investigated the behavior of such circuits in the 1950s.

Sequential circuits are also called *finite state machines (FSMs)*, which is a more formal name that is often found in technical literature. The name derives from the fact that the functional behavior of these circuits can be represented using a finite number of states. In this chapter we will often use the term *finite state machine*, or simply *machine*, when referring to sequential circuits.

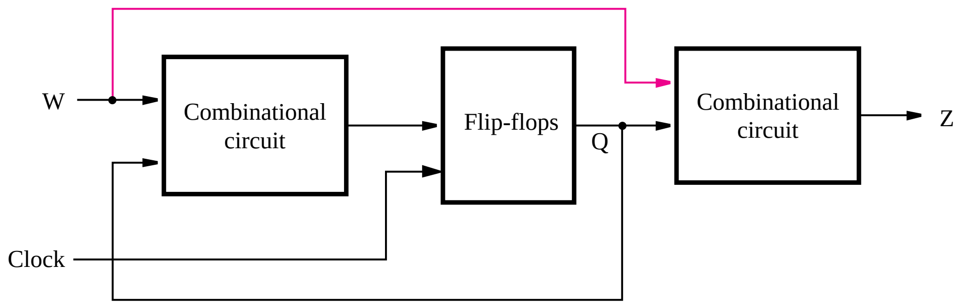


Figure 8.1 The general form of a sequential circuit.

8.1 Basic Design Steps

We will introduce the techniques for designing sequential circuits by means of a simple example. Suppose that we wish to design a circuit that meets the following specification:

1. The circuit has one input, w, and one output, z.
2. All changes in the circuit occur on the positive edge of a clock signal.
3. The output z is equal to 1 if during two immediately preceding clock cycles the input w was equal to 1. Otherwise, the value of z is equal to 0.

Thus, the circuit detects if two or more consecutive 1s occur on its input w. Circuits that detect the occurrence of a particular pattern on its input(s) are referred to as *sequence detectors*.

From this specification it is apparent that the output z cannot depend solely on the present value of w. To illustrate this, consider the sequence of values of the w and z signals during 11 clock cycles, as shown in Figure 8.2. The values of w are assumed arbitrarily; the values of z correspond to our specification. These sequences of input and output values indicate that for a given input value the output may be either 0 or 1. For example, $w = 0$ during clock cycles t_2 and t_5, but $z = 0$ during t_2 and $z = 1$ during t_5. Similarly, $w = 1$ during t_1 and t_8, but $z = 0$ during t_1 and $z = 1$ during t_8. This means that z is not determined only by the present value of w, so there must exist different states in the circuit that determine the value of z.

8.1.1 State Diagram

The first step in designing a finite state machine is to determine how many states are needed and which transitions are possible from one state to another. There is no set procedure for this task. The designer must think carefully about what the machine has to accomplish. A good way to begin is to select one particular state as a *starting* state; this is the state that the circuit should enter when power is first turned on or when a *reset* signal is applied. For our example let us assume that the starting state is called state A. As long as the input w is 0, the circuit need not do anything, and so each active clock edge should result in the circuit remaining in state A. When w becomes equal to 1, the machine should recognize this, and move to a different state, which we will call state B. This transition takes place on the next active clock edge after w has become equal to 1. In state B, as in state A, the circuit should keep the value of output z at 0, because it has not yet seen $w = 1$ for two consecutive clock cycles. When in state B, if w is 0 at the next active clock edge, the circuit should move back to state A. However, if $w = 1$ when in state B, the circuit should change to a third state, called C, and it should then generate an output $z = 1$. The circuit should remain in

Clock cycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0

Figure 8.2 Sequences of input and output signals.

state C as long as $w = 1$ and should continue to maintain $z = 1$. When w becomes 0, the machine should move back to state A. Since the preceding description handles all possible values of input w that the machine can encounter in its various states, we can conclude that three states are needed to implement the desired machine.

Now that we have determined in an informal way the possible transitions between states, we will describe a more formal procedure that can be used to design the corresponding sequential circuit. Behavior of a sequential circuit can be described in several different ways. The conceptually simplest method is to use a pictorial representation in the form of a *state diagram*, which is a graph that depicts states of the circuit as nodes (circles) and transitions between states as directed arcs. The state diagram in Figure 8.3 defines the behavior that corresponds to our specification. States A, B, and C appear as nodes in the diagram. Node A represents the starting state, and it is also the state that the circuit will reach *after* an input $w = 0$ is applied. In this state the output z should be 0, which is indicated as $A/z=0$ in the node. The circuit should remain in state A as long as $w = 0$, which is indicated by an arc with a label $w = 0$ that originates and terminates at this node. The first occurrence of $w = 1$ (following the condition $w = 0$) is recorded by moving from state A to state B. This transition is indicated on the graph by an arc originating at A and terminating at B. The label $w = 1$ on this arc denotes the input value that causes the transition. In state B the output remains at 0, which is indicated as $B/z=0$ in the node.

When the circuit is in state B, it will change to state C if w is still equal to 1 at the next active clock edge. In state C the output z becomes equal to 1. If w stays at 1 during subsequent clock cycles, the circuit will remain in state C maintaining $z = 1$. However, if w becomes 0 when the circuit is either in state B or in state C, the next active clock edge will cause a transition to state A to take place.

In the diagram we indicated that the *Reset* input is used to force the circuit into state A, which is possible regardless of what state the circuit happens to be in. We could treat

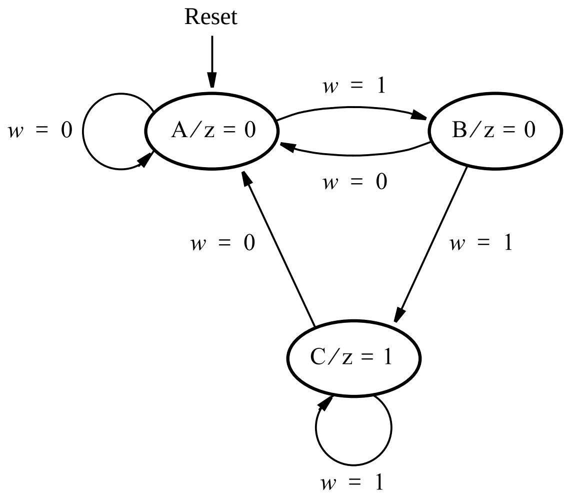


Figure 8.3 State diagram of a simple sequential circuit.

Reset as just another input to the circuit, and show a transition from each state to the starting state *A* under control of the input *Reset*. This would complicate the diagram unnecessarily. States in a finite state machine are implemented using flip-flops. Since flip-flops usually have reset capability, as discussed in Chapter 7, we can assume that the *Reset* input is used to clear all flip-flops to 0 by using this capability. We will indicate this as shown in Figure 8.3 to keep the diagrams as simple as possible.

8.1.2 State Table

Although the state diagram provides a description of the behavior of a sequential circuit that is easy to understand, to proceed with the implementation of the circuit, it is convenient to translate the information contained in the state diagram into a tabular form. Figure 8.4 shows the *state table* for our sequential circuit. The table indicates all transitions from each *present state* to the *next state* for different values of the input signal. Note that the output z is specified with respect to the present state, namely, the state that the circuit is in at present time. Note also that we did not include the *Reset* input; instead, we made an implicit assumption that the first state in the table is the starting state.

We now show the design steps that will produce the final circuit. To explain the basic design concepts, we first go through a traditional process of manually performing each design step. This is followed by a discussion of automated design techniques that use modern computer aided design (CAD) tools.

8.1.3 State Assignment

The state table in Figure 8.4 defines the three states in terms of letters A, B, and C. When implemented in a logic circuit, each state is represented by a particular valuation (combination of values) of *state variables*. Each state variable may be implemented in the form of a flip-flop. Since three states have to be realized, it is sufficient to use two state variables. Let these variables be y_1 and y_2.

Now we can adapt the general block diagram in Figure 8.1 to our example as shown in Figure 8.5, to indicate the structure of the circuit that implements the required finite state machine. Two flip-flops represent the state variables. In the figure we have not specified the type of flip-flops to be used; this issue is addressed in the next subsection. From the

Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	A	B	0
B	A	C	0
C	A	C	1

Figure 8.4 State table for the sequential circuit in Figure 8.3.

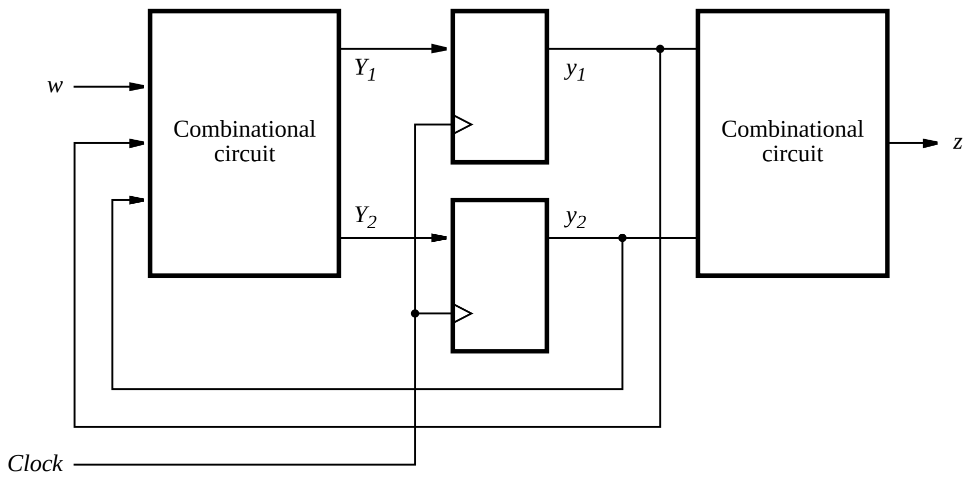


Figure 8.5 A general sequential circuit with input w, output z, and two state flip-flops.

specification in Figures 8.3 and 8.4, the output z is determined only by the present state of the circuit. Thus the block diagram in Figure 8.5 shows that z is a function of only y_1 and y_2; our design is of Moore type. We need to design a combinational circuit that uses y_1 and y_2 as input signals and generates a correct output signal z for all possible valuations of these inputs.

The signals y_1 and y_2 are also fed back to the combinational circuit that determines the next state of the FSM. This circuit also uses the primary input signal w. Its outputs are two signals, Y_1 and Y_2, which are used to set the state of the flip-flops. Each active edge of the clock will cause the flip-flops to change their state to the values of Y_1 and Y_2 at that time. Therefore, Y_1 and Y_2 are called the *next-state variables*, and y_1 and y_2 are called the *present-state variables*. We need to design a combinational circuit with inputs w, y_1, and y_2, such that for all valuations of these inputs the outputs Y_1 and Y_2 will cause the machine to move to the next state that satisfies our specification. The next step in the design process is to create a truth table that defines this circuit, as well as the circuit that generates z.

To produce the desired truth table, we assign a specific valuation of variables y_1 and y_2 to each state. One possible assignment is given in Figure 8.6, where the states A, B, and C are represented by $y_2y_1 = 00$, 01, and 10, respectively. The fourth valuation, $y_2y_1 = 11$, is not needed in this case.

The type of table given in Figure 8.6 is usually called a *state-assigned table*. This table can serve directly as a truth table for the output z with the inputs y_1 and y_2. Although for the next-state functions Y_1 and Y_2 the table does not have the appearance of a normal truth table, because there are two separate columns in the table for each value of w, it is obvious that the table includes all of the information that defines the next-state functions in terms of valuations of inputs w, y_1, and y_2.

	Present state	Next state		Output
		$w = 0$	$w = 1$	
	$y_2 y_1$	$Y_2 Y_1$	$Y_2 Y_1$	z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	dd	dd	d

Figure 8.6 State-assigned table for the sequential circuit in Figure 8.4.

8.1.4 Choice of Flip-Flops and Derivation of Next-State and Output Expressions

From the state-assigned table in Figure 8.6, we can derive the logic expressions for the next-state and output functions. But first we have to decide on the type of flip-flops that will be used in the circuit. The most straightforward choice is to use D-type flip-flops, because in this case the values of Y_1 and Y_2 are simply clocked into the flip-flops to become the new values of y_1 and y_2. In other words, if the inputs to the flip-flops are called D_1 and D_2, then these signals are the same as Y_1 and Y_2. Note that the diagram in Figure 8.5 corresponds exactly to this use of D-type flip-flops. For other types of flip-flops, such as JK type, the relationship between the next-state variable and inputs to a flip-flop is not as straightforward; we will consider this situation in section 8.7.

The required logic expressions can be derived as shown in Figure 8.7. We use Karnaugh maps to make it easy for the reader to verify the validity of the expressions. Recall that in Figure 8.6 we needed only three of the four possible binary valuations to represent the states. The fourth valuation, $y_2 y_1 = 11$, should never occur in the circuit because the circuit is constrained to move only within states A, B, and C; therefore, we may choose to treat this valuation as a don't-care condition. The resulting don't-care squares in the Karnaugh maps are denoted by d's. Using the don't cares to simplify the expressions, we obtain

$$Y_1 = w\bar{y}_1\bar{y}_2$$
$$Y_2 = w(y_1 + y_2)$$
$$z = y_2$$

If we do not use don't cares, then the resulting expressions are slightly more complex; they are shown in the gray-shaded area of Figure 8.7.

Since $D_1 = Y_1$ and $D_2 = Y_2$, the logic circuit that corresponds to the preceding expressions is implemented as shown in Figure 8.8. Observe that a clock signal is included, and the circuit is provided with an active-low reset capability. Connecting the clear input on the flip-flops to an external *Resetn* signal, as shown in the figure, provides a simple means

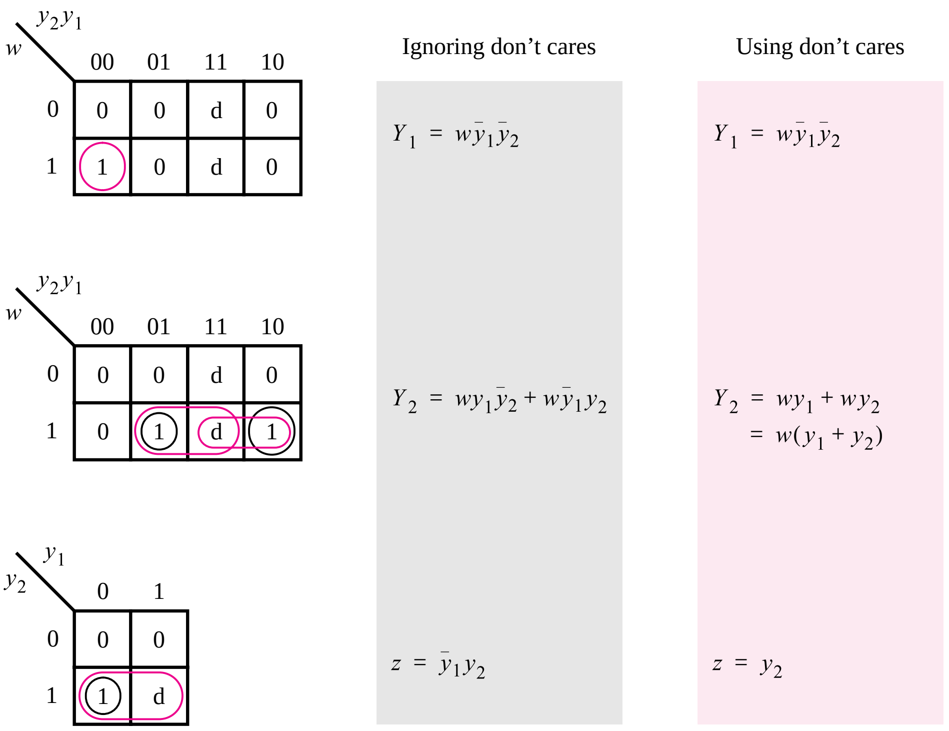

Ignoring don't cares
Using don't cares
$Y_1 = w\bar{y}_1\bar{y}_2$
$Y_1 = w\bar{y}_1\bar{y}_2$
$Y_2 = wy_1\bar{y}_2 + w\bar{y}_1y_2$
$Y_2 = wy_1 + wy_2 = w(y_1 + y_2)$
$z = \bar{y}_1y_2$
$z = y_2$

Figure 8.7 Derivation of logic expressions for the sequential circuit in Figure 8.6.

for forcing the circuit into a known state. If we apply the signal *Resetn* $= 0$ to the circuit, then both flip-flops will be cleared to 0, placing the FSM into the state $y_2y_1 = 00$.

8.1.5 Timing Diagram

To understand fully the operation of the circuit in Figure 8.8, let us consider its timing diagram presented in Figure 8.9. The diagram depicts the signal waveforms that correspond to the sequences of values in Figure 8.2.

Because we are using positive-edge-triggered flip-flops, all changes in the signals occur shortly after the positive edge of the clock. The amount of delay from the clock edge depends on the propagation delays through the flip-flops. Note that the input signal w is also shown to change slightly after the active edge of the clock. This is a good assumption because in a typical digital system an input such as w would be just an output of another circuit that is synchronized by the same clock. We discuss the synchronization of input signals with the clock signal in section 10.3.

A key point to observe is that even though w changes slightly after the active clock edge, and thus the value of w is equal to 1 (or 0) for almost the entire clock cycle, no change in the circuit will occur until the beginning of the next clock cycle when the positive edge

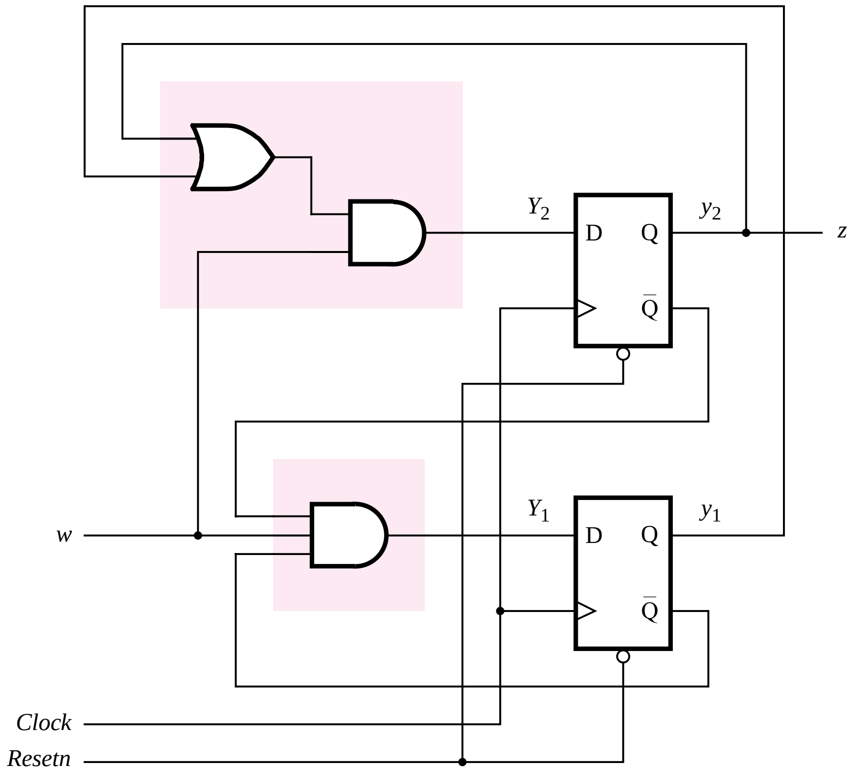


Figure 8.8 Final implementation of the sequential circuit in Figure 8.7.

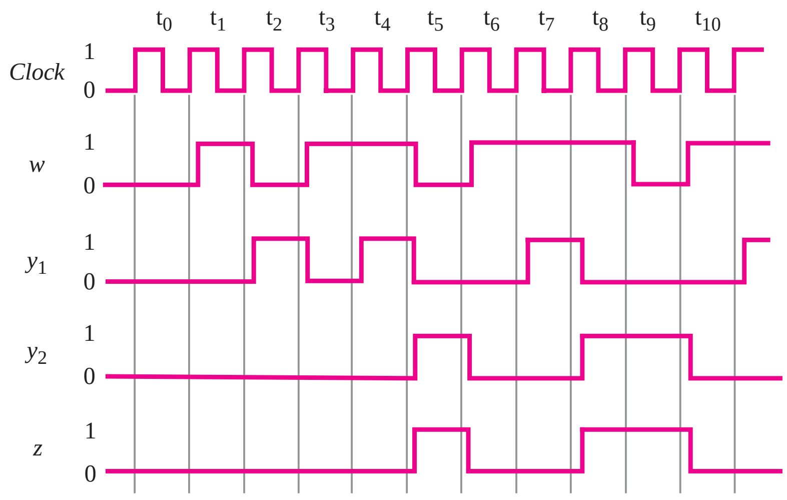


Figure 8.9 Timing diagram for the circuit in Figure 8.8.

causes the flip-flops to change their state. Thus the value of w must be equal to 1 for two clock cycles if the circuit is to reach state C and generate the output $z = 1$.

8.1.6 Summary of Design Steps

We can summarize the steps involved in designing a synchronous sequential circuit as follows:

1. Obtain the specification of the desired circuit.
2. Derive the states for the machine by first selecting a starting state. Then, given the specification of the circuit, consider all valuations of the inputs to the circuit and create new states as needed for the machine to respond to these inputs. To keep track of the states as they are visited, create a state diagram. When completed, the state diagram shows all states in the machine and gives the conditions under which the circuit moves from one state to another.
3. Create a state table from the state diagram. Alternatively, it may be convenient to directly create the state table in step 2, rather than first creating a state diagram.
4. In our sequential circuit example, there were only three states; hence it was a simple matter to create the state table that does not contain more states than necessary. However, in practice it is common to deal with circuits that have a large number of states. In such cases it is unlikely that the first attempt at deriving a state table will produce optimal results. Almost certainly we will have more states than is really necessary. This can be corrected by a procedure that minimizes the number of states. We will discuss the process of state minimization in section 8.6.
5. Decide on the number of state variables needed to represent all states and perform the state assignment. There are many different state assignments possible for a given sequential circuit. Some assignments may be better than others. In the preceding example we used what seemed to be a natural state assignment. We will return to this example in section 8.2 and show that a different assignment may lead to a simpler circuit.
6. Choose the type of flip-flops to be used in the circuit. Derive the next-state logic expressions to control the inputs to all flip-flops and then derive logic expressions for the outputs of the circuit. So far we have used only D-type flip-flops. We will consider other types of flip-flops in section 8.7.
7. Implement the circuit as indicated by the logic expressions.

Example 8.1 We have illustrated the design steps using a very simple sequential circuit. From the reader's point of view, a circuit that detects that an input signal was high for two consecutive clock pulses may not have much practical significance. We will now consider an example that is closely tied to practical application.

Section 7.14 introduced the concept of a bus and showed the connections that have to be made to allow the contents of a register to be transferred into another register. The

12 valuations of the state variables as don't cares, the next-state expressions are

$$\begin{aligned} Y_1 &= \overline{w}y_1 + y_4 \\ Y_2 &= wy_1 \\ Y_3 &= y_2 \\ Y_4 &= y_3 \end{aligned}$$

It is instructive to note that we can derive these expressions simply by inspecting the state diagram in Figure 8.11. Flip-flop y_1 should be set to 1 if the FSM is in state A and $w = 0$, or if the FSM is in state D; hence $Y_1 = \overline{w}y_1 + y_4$. Flip-flop y_2 should be set to 1 if the present state is A and $w = 1$; hence $Y_2 = wy_1$. Flip-flops y_3 and y_4 should be set to 1 if the FSM is presently in state B or C, respectively; hence $Y_3 = y_2$ and $Y_4 = y_3$.

The output expressions are just the outputs of the flip-flops, such that

$$\begin{aligned} R1_{out} = R2_{in} &= y_3 \\ R1_{in} = R3_{out} &= Done = y_4 \\ R2_{out} = R3_{in} &= y_2 \end{aligned}$$

These expressions are simpler than those derived in Example 8.2, but four flip-flops are needed, rather than two.

An important feature of the one-hot state assignment is that it often leads to simpler output expressions than do assignments with the minimal number of state variables. Simpler output expressions may lead to a faster circuit. For instance, if the outputs of the sequential circuit are just the outputs of the flip-flops, as is the case in our example, then these output signals are valid as soon as the flip-flops change their states. If more complex output expressions are involved, then the propagation delay through the gates that implement these expressions must be taken into account. We will consider this issue in section 8.8.2.

The examples considered to this point show that there are many ways to implement a given finite state machine as a sequential circuit. Each implementation is likely to have a different cost and different timing characteristics. In the next section we introduce another way of modeling FSMs that leads to even more possibilities.

8.3 MEALY STATE MODEL

Our introductory examples were sequential circuits in which each state had specific values of the output signals associated with it. As we explained at the beginning of the chapter, such finite state machines are said to be of Moore type. We will now explore the concept of Mealy-type machines in which the output values are generated based on both the state of the circuit and the present values of its inputs. This provides additional flexibility in the design of sequential circuits. We will introduce the Mealy-type machines, using a slightly altered version of a previous example.

The essence of the first sequential circuit in section 8.1 is to generate an output $z = 1$ whenever a second occurrence of the input $w = 1$ is detected in consecutive clock cycles. The specification requires that the output z be equal to 1 in the clock cycle that follows the detection of the second occurrence of $w = 1$. Suppose now that we eliminate this latter requirement and specify instead that the output z should be equal to 1 in the same clock cycle when the second occurrence of $w = 1$ is detected. Then a suitable input-output sequence may be as shown in Figure 8.22. To see how we can realize the behavior given in this table, we begin by selecting a starting state, A. As long as $w = 0$, the machine should remain in state A, producing an output $z = 0$. When $w = 1$, the machine has to move to a new state, B, to record the fact that an input of 1 has occurred. If w remains equal to 1 when the machine is in state B, which happens if $w = 1$ for at least two consecutive clock cycles, the machine should remain in state B and produce an output $z = 1$. As soon as w becomes 0, z should immediately become 0 and the machine should move back to state A at the next active edge of the clock. Thus the behavior specified in Figure 8.22 can be achieved with a two-state machine, which has a state diagram shown in Figure 8.23. Only two states are needed because we have allowed the output value to depend on the present value of the input as well as the present state of the machine. The diagram indicates that if the machine is in state A, it will remain in state A if $w = 0$ and the output will be 0. This is indicated by an arc with the label $w = 0/z = 0$. When w becomes 1, the output stays at 0 until the machine moves to state B at the next active clock edge. This is denoted by the arc from A to B with the label $w = 1/z = 0$. In state B the output will be 1 if $w = 1$, and the machine will remain in state B, as indicated by the label $w = 1/z = 1$ on the corresponding arc. However, if $w = 0$ in state B, then the output will be 0 and a transition to state A will take place at the next active clock edge. A key point to understand is that during the present clock cycle the output value corresponds to the label on the arc emanating from the present-state node.

We can implement the FSM in Figure 8.23, using the same design steps as in section 8.1. The state table is shown in Figure 8.24. The table shows that the output z depends on the present value of input w and not just on the present state. Figure 8.25 gives the

Clock cycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	1	0	0	1	1	0	0

Figure 8.22 Sequences of input and output signals.

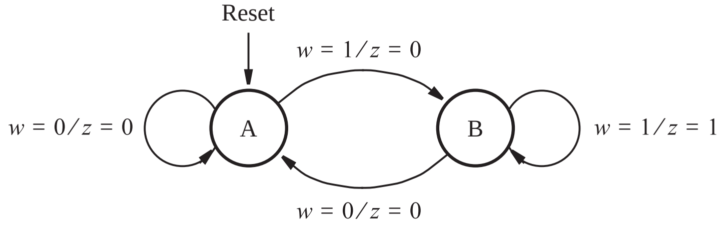


Figure 8.23 State diagram of an FSM that realizes the task in Figure 8.22.

Present state	Next state		Output z	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
A	A	B	0	0
B	A	B	0	1

Figure 8.24 State table for the FSM in Figure 8.23.

	Present state	Next state		Output	
		$w = 0$	$w = 1$	$w = 0$	$w = 1$
	y	Y	Y	z	z
A	0	0	1	0	0
B	1	0	1	0	1

Figure 8.25 State-assigned table for the FSM in Figure 8.24.

state-assigned table. Because there are only two states, it is sufficient to use a single state variable, y. Assuming that y is realized as a D-type flip-flop, the required next-state and output expressions are

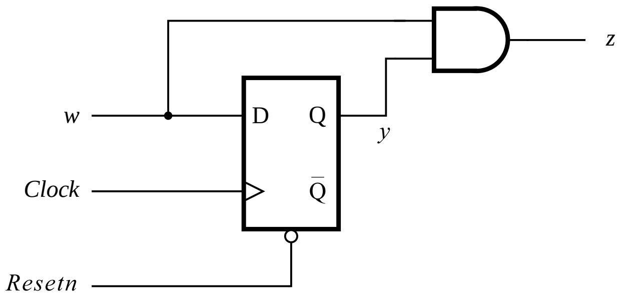
$$Y = D = w$$

$$z = wy$$

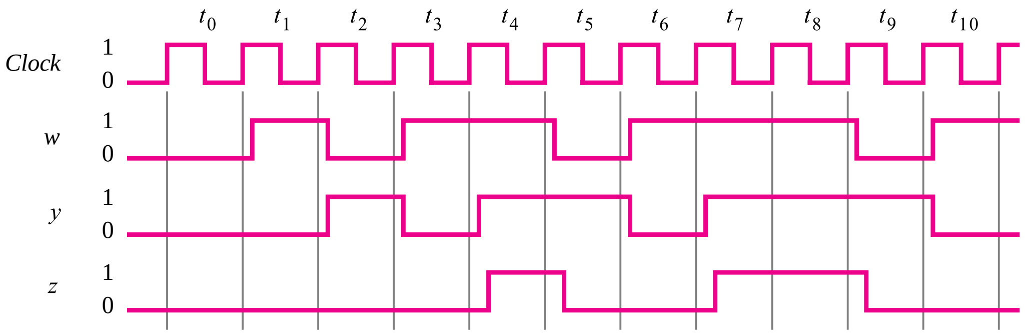
The resulting circuit is presented in Figure 8.26 along with a timing diagram. The timing diagram corresponds to the input-output sequences in Figure 8.22.

The greater flexibility of Mealy-type FSMs often leads to simpler circuit realizations. This certainly seems to be the case in our examples that produced the circuits in Figures 8.8, 8.17, and 8.26, assuming that the design requirement is only to detect two consecutive occurrences of input w being equal to 1. We should note, however, that the circuit in Figure 8.26 is not the same in terms of output behavior as the circuits in Figures 8.8 and 8.17. The difference is a shift of one clock cycle in the output signal in Figure 8.26*b*. If we wanted to produce exactly the same output behavior using the Mealy approach, we could modify the circuit in Figure 8.26*a* by adding another flip-flop as shown in Figure 8.27. This flip-flop merely delays the output signal, Z, by one clock cycle with respect to z, as indicated in the timing diagram. By making this change, we effectively turn the Mealy-type circuit into a Moore-type circuit with output Z. Note that the circuit in Figure 8.27 is essentially the same as the circuit in Figure 8.17.

Example 8.4 In Example 8.1 we considered the control circuit needed to swap the contents of two registers, implemented as a Moore-type finite state machine. The same task can be achieved using a



(a) Circuit



(b) Timing diagram

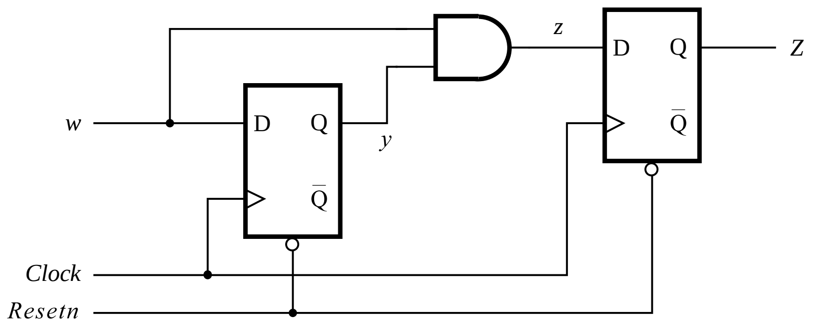
Figure 8.26 Implementation of FSM in Figure 8.25.

Mealy-type FSM, as indicated in Figure 8.28. State A still serves as the reset state. But as soon as w changes from 0 to 1, the output control signals $R2_{out}$ and $R3_{in}$ are asserted. They remain asserted until the beginning of the next clock cycle, when the circuit will leave state A and change to B. In state B the outputs $R1_{out}$ and $R2_{in}$ are asserted for both $w = 0$ and $w = 1$. Finally, in state C the swap is completed by asserting $R3_{out}$ and $R1_{in}$.

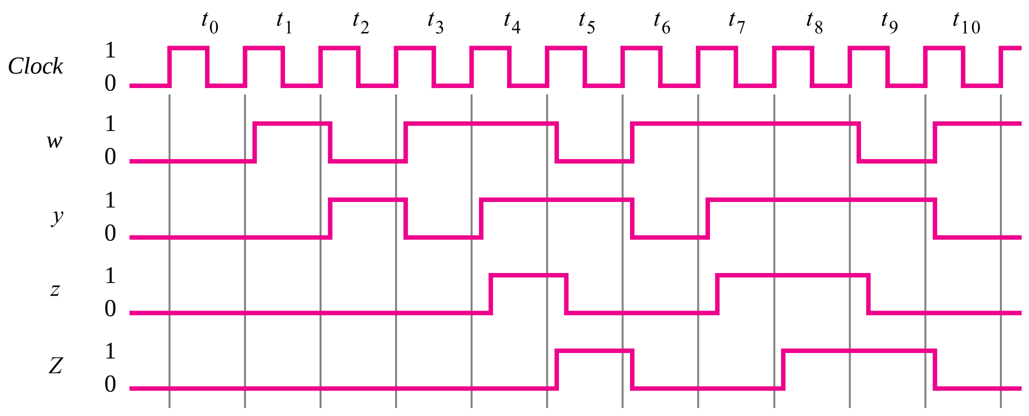
The Mealy-type realization of the control circuit requires three states. This does not necessarily imply a simpler circuit because two flip-flops are still needed to implement the state variables. The most important difference in comparison with the Moore-type realization is the timing of output signals. A circuit that implements the FSM in Figure 8.28 generates the output control signals one clock cycle sooner than the circuits derived in Examples 8.1 and 8.2.

Note also that using the FSM in Figure 8.28, the entire process of swapping the contents of $R1$ and $R2$ takes three clock cycles, starting and finishing in state A. Using the Moore-type FSM in Example 8.1, the swapping process involves four clock cycles before the circuit returns to state A.

Suppose that we wish to implement this FSM using one-hot encoding. Then three flip-flops are needed, and the states A, B, and C may be assigned the valuations $y_3y_2y_1 =$ 001, 010, and 100, respectively. Examining the state diagram in Figure 8.28, we can derive



(a) Circuit



(b) Timing diagram

Figure 8.27 Circuit that implements the specification in Figure 8.2.

the next-state equations by inspection. The input to flip-flop y_1 should have the value 1 if the FSM is in state A and $w = 0$ or if the FSM is in state C; hence $Y_1 = \overline{w}y_1 + y_3$. Flip-flop y_2 should be set to 1 if the FSM is in state A and $w = 1$; hence $Y_2 = wy_1$. Flip-flop y_3 should be set to 1 if the present state is B; hence $Y_3 = y_2$. The derivation of the output expressions, which we leave as an exercise for the reader, can also be done by inspection. The corresponding circuit is shown in Figure 7.58, in section 7.14, where it was derived using an ad hoc approach.

The preceding discussion deals with the basic principles involved in the design of sequential circuits. Although it is essential to understand these principles, the manual approach used in the examples is difficult and tedious when large circuits are involved. We will now show how CAD tools are used to greatly simplify the design task.

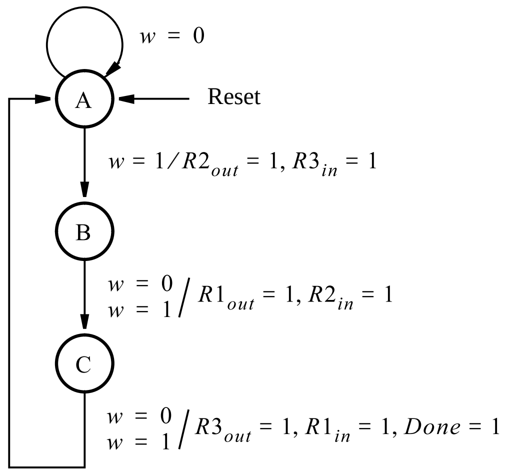

$w = 0$
Reset
A
$w = 1 / R2_{out} = 1, R3_{in} = 1$
B
$w = 0$, $w = 1$ / $R1_{out} = 1, R2_{in} = 1$
C
$w = 0$, $w = 1$ / $R3_{out} = 1, R1_{in} = 1, Done = 1$

Figure 8.28 State diagram for Example 8.4.

8.5.1 MEALY-TYPE FSM FOR SERIAL ADDER

Let $A = a_{n-1}a_{n-2} \cdots a_0$ and $B = b_{n-1}b_{n-2} \cdots b_0$ be two unsigned numbers that have to be added to produce $Sum = s_{n-1}s_{n-2} \cdots s_0$. Our task is to design a circuit that will perform serial addition, dealing with a pair of bits in one clock cycle. The process starts by adding bits a_0 and b_0. In the next clock cycle, bits a_1 and b_1 are added, including a possible carry from the bit-position 0, and so on. Figure 8.39 shows a block diagram of a possible implementation. It includes three shift registers that are used to hold A, B, and *Sum* as the computation proceeds. Assuming that the input shift registers have parallel-load capability, as depicted in Figure 7.19, the addition task begins by loading the values of A and B into these registers. Then in each clock cycle, a pair of bits is added by the adder FSM, and at the end of the cycle the resulting sum bit is shifted into the *Sum* register. We will use positive-edge-triggered flip-flops in which case all changes take place soon after the positive edge of the clock, depending on the propagation delays within the various flip-flops. At this time the contents of all three shift registers are shifted to the right; this shifts the existing sum bit into *Sum*, and it presents the next pair of input bits a_i and b_i to the adder FSM.

Now we are ready to design the required FSM. This cannot be a combinational circuit because different actions will have to be taken, depending on the value of the carry from the previous bit position. Hence two states are needed: let G and H denote the states where the carry-in values are 0 and 1, respectively. Figure 8.40 gives a suitable state diagram, defined as a Mealy model. The output value, s, depends on both the state and the present value of the inputs a and b. Each transition is labeled using the notation ab/s, which indicates the value of s for a given valuation ab. In state G the input valuation 00 will produce $s = 0$, and the FSM will remain in the same state. For input valuations 01 and 10, the output will be $s = 1$, and the FSM will remain in G. But for 11, $s = 0$ is generated, and the machine

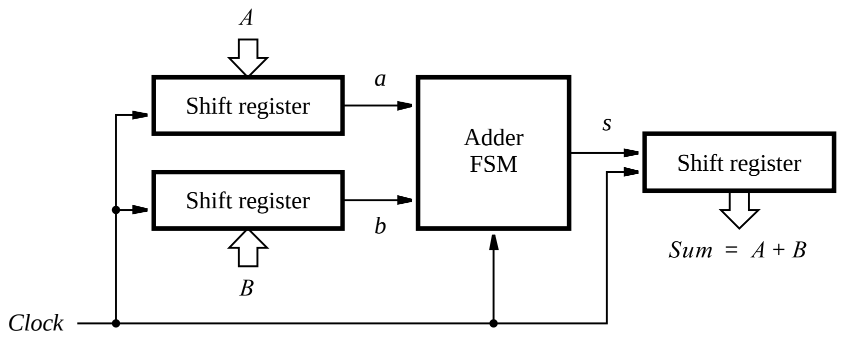


Figure 8.39 Block diagram for the serial adder.

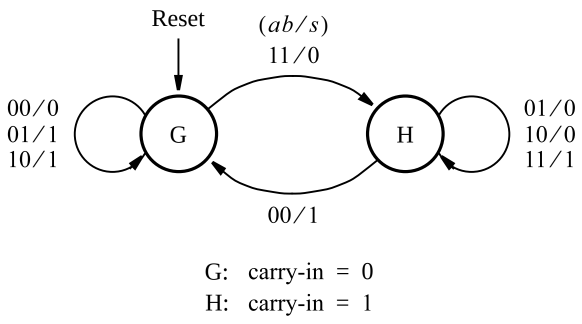


Figure 8.40 State diagram for the serial adder FSM.

moves to state H. In state H valuations 01 and 10 cause $s = 0$, while 11 causes $s = 1$. In all three of these cases, the machine remains in H. However, when the valuation 00 occurs, the output of 1 is produced and a change into state G takes place.

The corresponding state table is presented in Figure 8.41. A single flip-flop is needed to represent the two states. The state assignment can be done as indicated in Figure 8.42. This assignment leads to the following next-state and output equations

$$Y = ab + ay + by$$
$$s = a \oplus b \oplus y$$

Present state	Next state				Output s			
	ab = 00	01	10	11	00	01	10	11
G	G	G	G	H	0	1	1	0
H	G	H	H	H	1	0	0	1

Figure 8.41 State table for the serial adder FSM.

Present state	Next state				Output			
	ab = 00	01	10	11	00	01	10	11
y	Y				s			
0	0	0	0	1	0	1	1	0
1	0	1	1	1	1	0	0	1

Figure 8.42 State-assigned table for Figure 8.41.

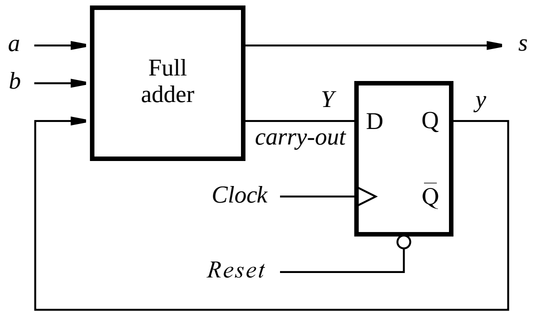


Figure 8.43 Circuit for the adder FSM in Figure 8.39.

Comparing these expressions with those for the full-adder in section 5.2, it is obvious that y is the carry-in, Y is the carry-out, and s is the sum of the full-adder. Therefore, the adder FSM box in Figure 8.39 consists of the circuit shown in Figure 8.43. The flip-flop can be cleared by the *Reset* signal at the start of the addition operation.

The serial adder is a simple circuit that can be used to add numbers of any length. The structure in Figure 8.39 is limited in length only by the size of the shift registers.

8.7 Design of a Counter Using the Sequential Circuit Approach

In this section we discuss the design of a counter circuit using the general approach for designing sequential circuits. From Chapter 7 we already know that counters can be realized as cascaded stages of flip-flops and some gating logic, where each stage divides the number of incoming pulses by two. To keep our example simple, we choose a counter of small size but also show how the design can be extended to larger sizes. The specification for the counter is

- The counting sequence is 0, 1, 2, . . . , 6, 7, 0, 1, . . .
- There exists an input signal w. The value of this signal is considered during each clock cycle. If $w = 0$, the present count remains the same; if $w = 1$, the count is incremented.

The counter can be designed as a synchronous sequential circuit using the design techniques introduced in the previous sections.

8.7.1 State Diagram and State Table for a Modulo-8 Counter

Figure 8.60 gives a state diagram for the desired counter. There is a state associated with each count. In the diagram state A corresponds to count 0, state B to count 1, and so on. We show the transitions between the states needed to implement the counting sequence. Note that the output signals are specified as depending only on the state of the counter at a given time, which is the Moore model of sequential circuits.

The state diagram may be represented in the state-table form as shown in Figure 8.61.

8.7.2 State Assignment

Three state variables are needed to represent the eight states. Let these variables, denoting the present state, be called y_2, y_1, and y_0. Let Y_2, Y_1, and Y_0 denote the corresponding next-state functions. The most convenient (and simplest) state assignment is to encode

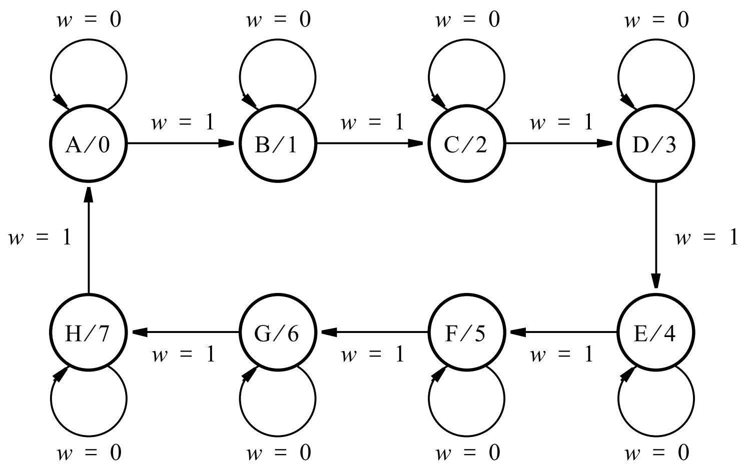


Figure 8.60 State diagram for the counter.

Present state	Next state		Output
	$w=0$	$w=1$	
A	A	B	0
B	B	C	1
C	C	D	2
D	D	E	3
E	E	F	4
F	F	G	5
G	G	H	6
H	H	A	7

Figure 8.61 State table for the counter.

each state with the binary number that the counter should give as output in that state. Then the required output signals will be the same as the signals that represent the state variables. This leads to the state-assigned table in Figure 8.62.

The final step in the design is to choose the type of flip-flops and derive the expressions that control the flip-flop inputs. The most straightforward choice is to use D-type flip-flops. We pursue this approach first. Then we show the alternative of using JK-type flip-flops. In either case the flip-flops must be edge triggered to ensure that only one transition takes place during a single clock cycle.

	Present state	Next state		Count
		$w = 0$	$w = 1$	
	$y_2y_1y_0$	$Y_2Y_1Y_0$	$Y_2Y_1Y_0$	$z_2z_1z_0$
A	000	000	001	000
B	001	001	010	001
C	010	010	011	010
D	011	011	100	011
E	100	100	101	100
F	101	101	110	101
G	110	110	111	110
H	111	111	000	111

Figure 8.62 State-assigned table for the counter.

8.7.3 Implementation Using D-Type Flip-Flops

When using D-type flip-flops to realize the finite state machine, each next-state function, Y_i, is connected to the D input of the flip-flop that implements the state variable y_i. The next-state functions are derived from the information in Figure 8.62. Using Karnaugh maps in Figure 8.63, we obtain the following implementation

$$\begin{aligned} D_0 &= Y_0 = \overline{w}y_0 + w\overline{y}_0 \\ D_1 &= Y_1 = \overline{w}y_1 + y_1\overline{y}_0 + wy_0\overline{y}_1 \\ D_2 &= Y_2 = \overline{w}y_2 + \overline{y}_0y_2 + \overline{y}_1y_2 + wy_0y_1\overline{y}_2 \end{aligned}$$

The resulting circuit is given in Figure 8.64. It is not obvious how to extend this circuit to implement a larger counter, because no clear pattern is discernible in the expressions for D_0, D_1, and D_2. However, we can rewrite these expressions as follows

$$\begin{aligned} D_0 &= \overline{w}y_0 + w\overline{y}_0 \\ &= w \oplus y_0 \\ D_1 &= \overline{w}y_1 + y_1\overline{y}_0 + wy_0\overline{y}_1 \\ &= (\overline{w} + \overline{y}_0)y_1 + wy_0\overline{y}_1 \\ &= \overline{wy_0}y_1 + wy_0\overline{y}_1 \\ &= wy_0 \oplus y_1 \end{aligned}$$

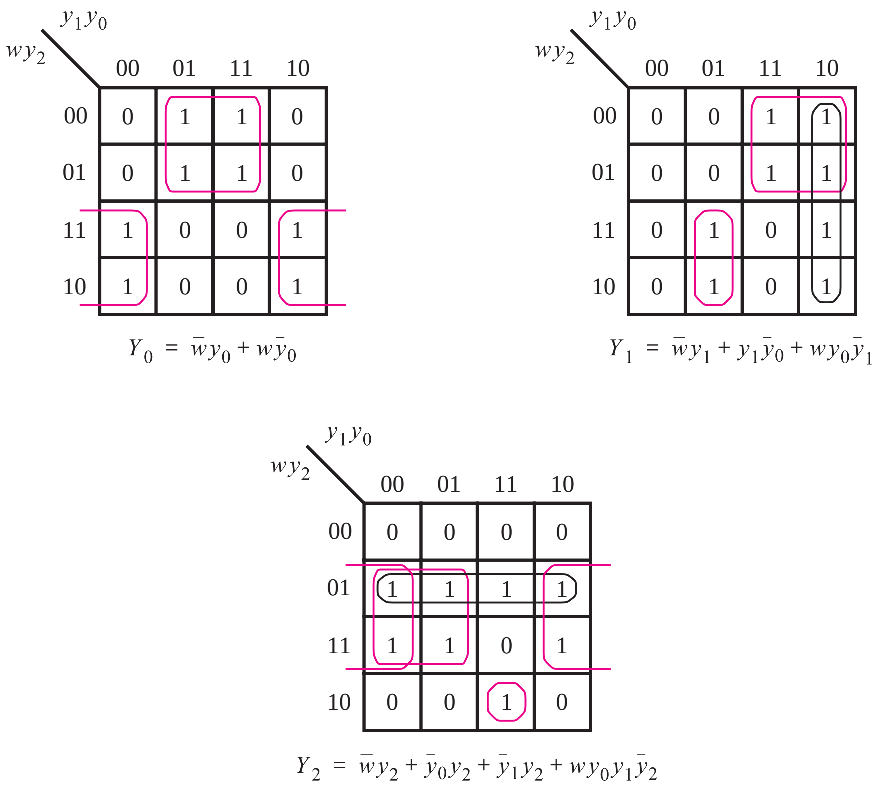

y_1y_0
wy_2
$Y_0 = \overline{w}y_0 + w\overline{y}_0$
$Y_1 = \overline{w}y_1 + y_1\overline{y}_0 + wy_0\overline{y}_1$
$Y_2 = \overline{w}y_2 + \overline{y}_0y_2 + \overline{y}_1y_2 + wy_0y_1\overline{y}_2$

Figure 8.63 Karnaugh maps for D flip-flops for the counter.

$$\begin{aligned} D_2 &= \overline{w}y_2 + \overline{y}_0y_2 + \overline{y}_1y_2 + wy_0y_1\overline{y}_2 \\ &= (\overline{w} + \overline{y}_0 + \overline{y}_1)y_2 + wy_0y_1\overline{y}_2 \\ &= \overline{wy_0y_1}y_2 + wy_0y_1\overline{y}_2 \\ &= wy_0y_1 \oplus y_2 \end{aligned}$$

Then an obvious pattern emerges, which leads to the circuit in Figure 7.24.

8.7.4 Implementation Using JK-Type Flip-Flops

JK-type flip-flops provide an attractive alternative. Using these flip-flops to implement the sequential circuit specified in Figure 8.62 requires derivation of J and K inputs for each flip-flop. The following control is needed:

- If a flip-flop in state 0 is to remain in state 0, then $J = 0$ and $K = d$ (where d means that K can be equal to either 0 or 1).

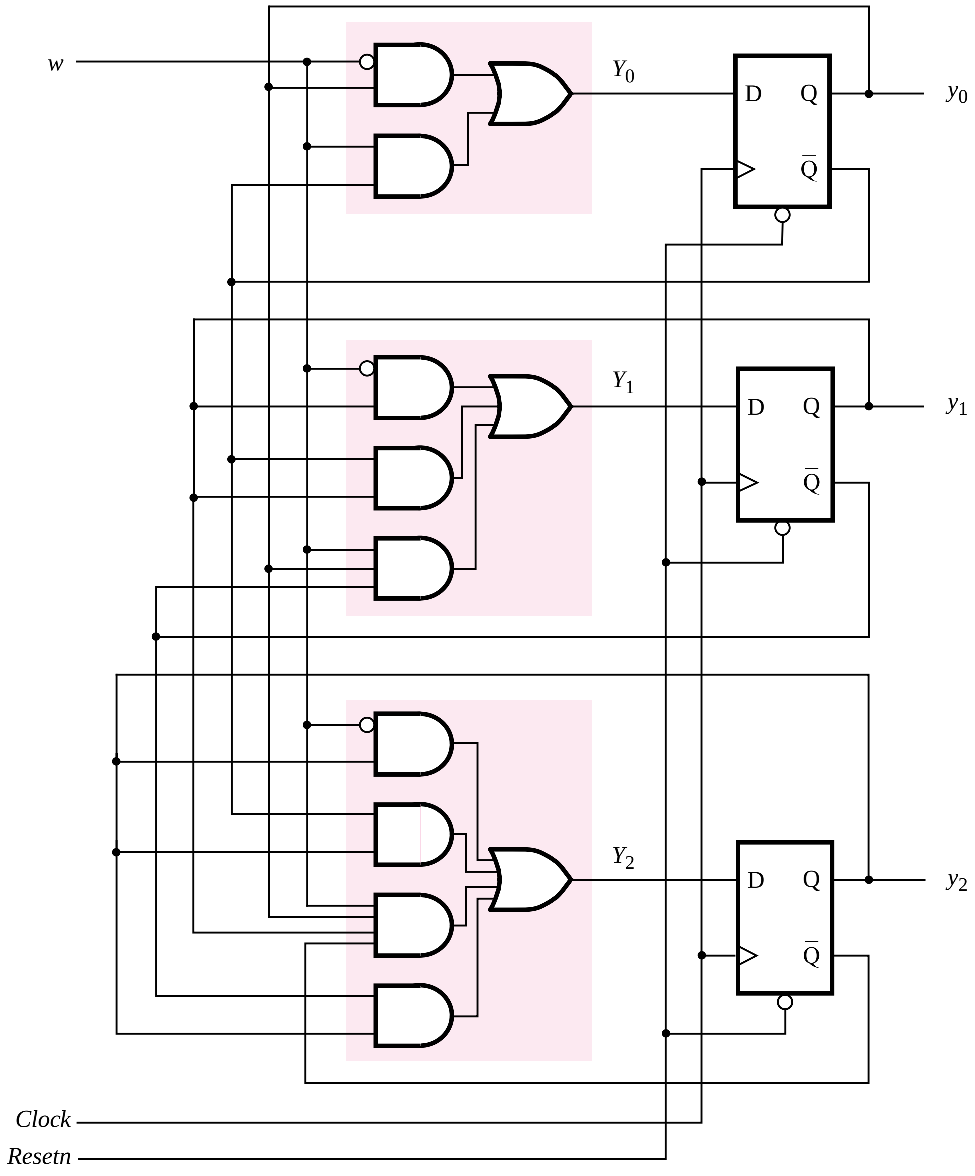


Figure 8.64 Circuit diagram for the counter implemented with D flip-flops.

- If a flip-flop in state 0 is to change to state 1, then $J = 1$ and $K = d$.
- If a flip-flop in state 1 is to remain in state 1, then $J = d$ and $K = 0$.
- If a flip-flop in state 1 is to change to state 0, then $J = d$ and $K = 1$.

Following these guidelines, we can create a truth table that specifies the required values of the J and K inputs for the three flip-flops in our design. Figure 8.65 shows a modified version of the state-assigned table in Figure 8.62, with the J and K input functions included. To see how this table is derived, consider the first row in which the present state is $y_2y_1y_0 = 000$. If $w = 0$, then the next state is also $Y_2Y_1Y_0 = 000$. Thus the present value of each flip-flop is 0, and it should remain 0. This implies the control $J = 0$ and $K = d$ for all three flip-flops. Continuing with the first row, if $w = 1$, the next state will be $Y_2Y_1Y_0 = 001$. Thus flip-flops y_2 and y_1 still remain at 0 and have the control $J = 0$ and $K = d$. However, flip-flop y_0 must change from 0 to 1, which is accomplished with $J = 1$ and $K = d$. The rest of the table is derived in the same manner by considering each present state $y_2y_1y_0$ and providing the necessary control signals to reach the new state $Y_2Y_1Y_0$.

A state-assigned table is essentially the state table in which each state is encoded using the state variables. When D flip-flops are used to implement an FSM, the next-state entries in the state-assigned table correspond directly to the signals that must be applied to the D inputs. This is not the case if some other type of flip-flops is used. A table that gives the state information in the form of the flip-flop inputs that must be "excited" to cause the transitions to the next states is usually called an *excitation table*. The excitation table in Figure 8.65 indicates how JK flip-flops can be used. In many books the term excitation table is used even when D flip-flops are involved, in which case it is synonymous with the state-assigned table.

Once the table in Figure 8.65 has been derived, it provides a truth table with inputs y_2, y_1, y_0, and w, and outputs J_2, K_2, J_1, K_1, J_0, and K_0. We can then derive expressions for

	Present state	Flip-flop inputs								Count
		$w = 0$				$w = 1$				
	$y_2y_1y_0$	$Y_2Y_1Y_0$	J_2K_2	J_1K_1	J_0K_0	$Y_2Y_1Y_0$	J_2K_2	J_1K_1	J_0K_0	$z_2z_1z_0$
A	000	000	0d	0d	0d	001	0d	0d	1d	000
B	001	001	0d	0d	d0	010	0d	1d	d1	001
C	010	010	0d	d0	0d	011	0d	d0	1d	010
D	011	011	0d	d0	d0	100	1d	d1	d1	011
E	100	100	d0	0d	0d	101	d0	0d	1d	100
F	101	101	d0	0d	d0	110	d0	1d	d1	101
G	110	110	d0	d0	0d	111	d0	d0	1d	110
H	111	111	d0	d0	d0	000	d1	d1	d1	111

Figure 8.65 Excitation table for the counter with JK flip-flops.

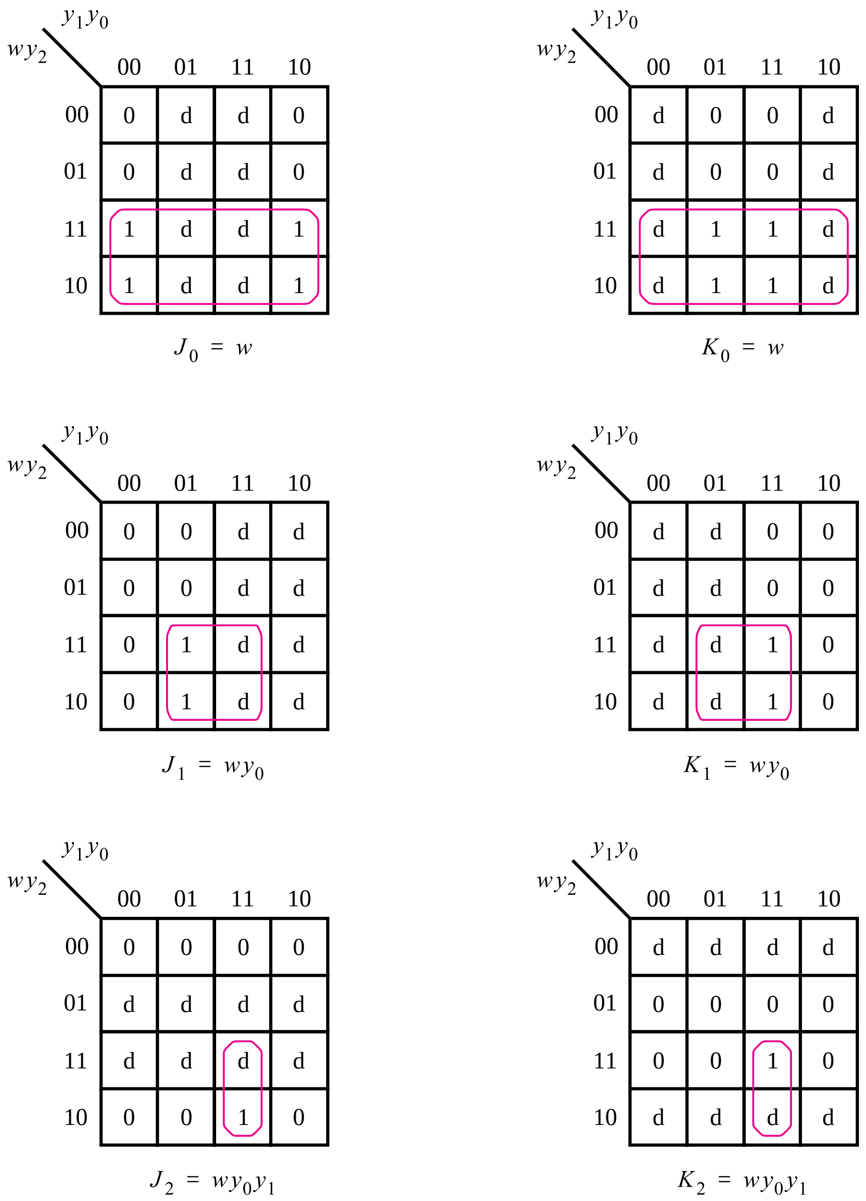


Figure 8.66 Karnaugh maps for JK flip-flops in the counter.

these outputs as shown in Figure 8.66. The resulting expressions are

$$J_0 = K_0 = w$$
$$J_1 = K_1 = wy_0$$
$$J_2 = K_2 = wy_0y_1$$

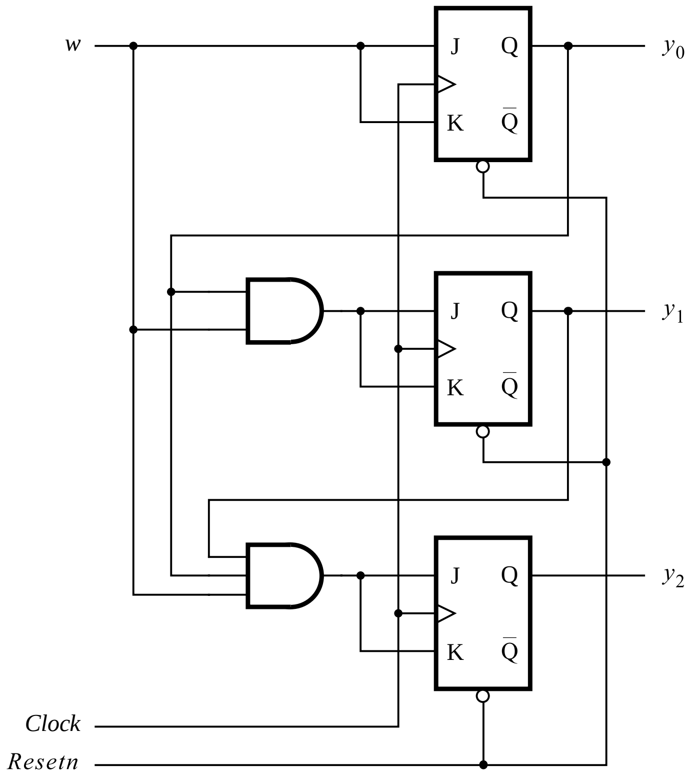


Figure 8.67 Circuit diagram using JK flip-flops.

This leads to the circuit shown in Figure 8.67. It is apparent that this design can be extended easily to larger counters. The pattern $J_n = K_n = wy_0y_1 \cdots y_{n-1}$ defines the circuit for each stage in the counter. Note that the size of the AND gate that implements the product term $y_0y_1 \cdots y_{n-1}$ grows with successive stages. A circuit with a more regular structure can be obtained by factoring out the previously needed terms as we progress through the stages of the counter. This gives

$$\begin{aligned} J_2 = K_2 &= (wy_0)y_1 &&= J_1y_1 \\ J_n = K_n &= (wy_0 \cdots y_{n-2})y_{n-1} &&= J_{n-1}y_{n-1} \end{aligned}$$

Using the factored form, the counter circuit can be realized as indicated in Figure 8.68. In this circuit all stages (except the first) look the same. Note that this circuit has the same structure as the circuit in Figure 7.23 because connecting the J and K inputs of a flip-flop together turns the flip-flop into a T flip-flop.

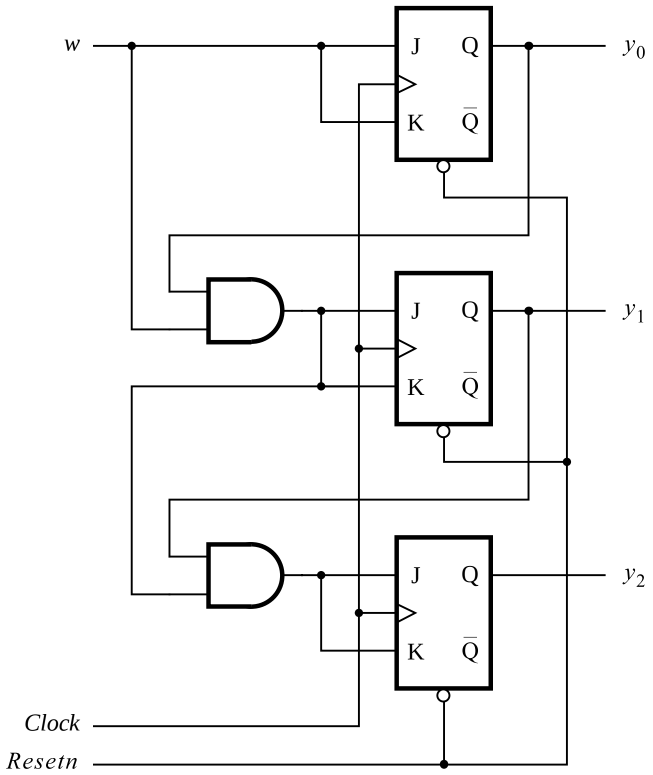


Figure 8.68 Factored-form implementation of the counter.

8.7.5 Example—A Different Counter

Having considered the design of an ordinary counter, we will now apply this knowledge to design a slightly different counterlike circuit. Suppose that we wish to derive a three-bit counter that counts the pulses on an input line, w. But instead of displaying the count as 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, . . . , this counter must display the count in the sequence 0, 4, 2, 6, 1, 5, 3, 7, 0, 4, and so on. The count is to be represented directly by the flip-flop values themselves, without using any extra gates. Namely, Count = $Q_2Q_1Q_0$.

Since we wish to count the pulses on the input line w, it makes sense to use w as the clock input to the flip-flops. Thus the counter circuit should always be enabled, and it should change its state whenever the next pulse on the w line appears. The desired counter can be designed in a straightforward manner using the FSM approach. Figures 8.69 and 8.70 give the required state table and a suitable state assignment. Using D flip-flops, we obtain the next-state equations

$$D_2 = Y_2 = \overline{y}_2$$
$$D_1 = Y_1 = y_1 \oplus y_2$$

Present state	Next state	Output $z_2z_1z_0$
A	B	0 0 0
B	C	1 0 0
C	D	0 1 0
D	E	1 1 0
E	F	0 0 1
F	G	1 0 1
G	H	0 1 1
H	A	1 1 1

Figure 8.69 State table for counterlike example.

Present state $y_2y_1y_0$	Next state $Y_2Y_1Y_0$	Output $z_2z_1z_0$
0 0 0	1 0 0	0 0 0
1 0 0	0 1 0	1 0 0
0 1 0	1 1 0	0 1 0
1 1 0	0 0 1	1 1 0
0 0 1	1 0 1	0 0 1
1 0 1	0 1 1	1 0 1
0 1 1	1 1 1	0 1 1
1 1 1	0 0 0	1 1 1

Figure 8.70 State-assigned table for Figure 8.69.

$$\begin{aligned} D_0 = Y_0 &= y_0\overline{y}_1 + y_0\overline{y}_2 + \overline{y}_0 y_1 y_2 \\ &= y_0(\overline{y}_1 + \overline{y}_2) + \overline{y}_0 y_1 y_2 \\ &= y_0 \oplus y_1 y_2 \end{aligned}$$

This leads to the circuit in Figure 8.71.

The reader should compare this circuit with the normal up-counter in Figure 7.24. Take the first three stages of that counter, set the *Enable* input to 1, and let $Clock = w$. Then the two circuits are essentially the same with one small difference in the order of bits in the count. In Figure 7.24 the top flip-flop corresponds to the least-significant bit of the count, whereas in Figure 8.71 the top flip-flop corresponds to the most-significant bit of the count. This is not just a coincidence. In Figure 8.70 the required count is defined as $Count = y_2y_1y_0$. However, if the bit patterns that define the states are viewed in the reverse order and interpreted as binary numbers, such that $Count = y_0y_1y_2$, then the states

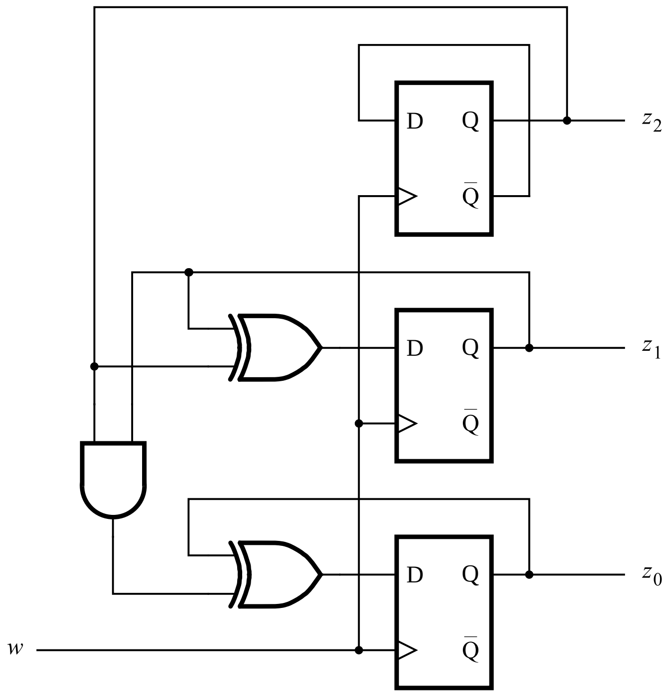


Figure 8.71 Circuit for Figure 8.70.

$A, B, C, \ldots, H$ have the values $0, 1, 2, \ldots, 7$. These values are the same as the values that are associated with the normal three-bit up-counter.