chapter

9

ASYNCHRONOUS SEQUENTIAL CIRCUITS

CHAPTER OBJECTIVES

In this chapter you will learn about:

- Sequential circuits that are not synchronized by a clock
- Analysis of asynchronous sequential circuits
- Synthesis of asynchronous sequential circuits
- The concept of stable and unstable states
- Hazards that cause incorrect behavior of a circuit
- Timing issues in digital circuits

In the previous chapter we covered the design of synchronous sequential circuits in which the state variables are represented by flip-flops that are controlled by a clock. The clock is a periodic signal that consists of pulses. Changes in state can occur on the positive or negative edge of each clock pulse. Since they are controlled by pulses, synchronous sequential circuits are said to operate in *pulse mode*. In this chapter we present sequential circuits that do not operate in pulse mode and do not use flip-flops to represent state variables. These circuits are called *asynchronous sequential circuits*.

In an asynchronous sequential circuit, changes in state are not triggered by clock pulses. Instead, changes in state are dependent on whether each of the inputs to the circuit has the logic level 0 or 1 at any given time. To achieve reliable operation, the inputs to the circuit must change in a specific manner. In this introductory discussion we will concentrate on the simplest case in which a constraint is imposed that the inputs must change one at a time. Moreover, there must be sufficient time between the changes in input signals to allow the circuit to reach a *stable state*, which is achieved when all internal signals stop changing. A circuit that adheres to these constraints is said to operate in the *fundamental mode*.

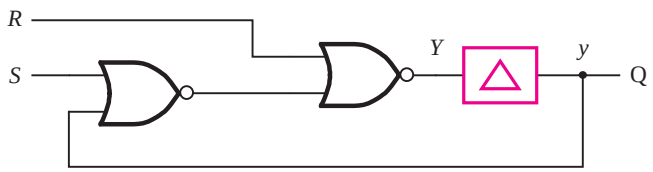
Asynchronous circuits are much more difficult to design than synchronous circuits. Specialized techniques, which are beyond the scope of this book, have been developed for dealing with large asynchronous circuits. Our main reason for the discussion in this chapter is the fact that the asynchronous circuits, even in their simplest form, provide an excellent vehicle for gaining a deeper understanding of the operation of digital circuits in general. In particular, they illustrate the timing issues caused by propagation delays in logic circuits.

The design approaches presented in this chapter are classical techniques that are suitable only for very small circuits. They are easy to understand and they demonstrate the problems that arise from timing constraints. In synchronous circuits these problems are avoided by using a clock as a synchronizing mechanism.

9.1 ASYNCHRONOUS BEHAVIOR

To introduce asynchronous sequential circuits, we will reconsider the basic latch circuit in Figure 7.4. This Set-Reset (SR) latch is redrawn in Figure 9.1*a*. The feedback loop gives rise to the sequential nature of the circuit. It is an asynchronous circuit because changes in the value of the output, Q, occur without having to wait for a synchronizing clock pulse. In response to a change in either the S (Set) or R (Reset) input, the value of Q will change after a short propagation time through the NOR gates. In Figure 9.1*a* the combined propagation delay through the two NOR gates is represented by the box labeled Δ. Then, the NOR gate symbols represent ideal gates with zero delay. Using the notation in Chapter 8, Q corresponds to the *present state* of the circuit, represented by the *present-state variable*, y. The value of y is fed back through the circuit to generate the value of the *next-state variable*, Y, which represents the *next state* of the circuit. After the Δ time delay, y takes the value of Y. Observe that we have drawn the circuit in a style that conforms to the general model for sequential circuits presented in Figure 8.90.

By analyzing the SR latch, we can derive a state-assigned table, as illustrated in Figure 9.1*b*. When the present state is $y = 0$ and the inputs are $S = R = 0$, the circuit produces $Y = 0$. Since $y = Y$, the state of the circuit will not change. We say that the circuit is *stable* under these input conditions. Now assume that R changes to 1 while S remains at 0. The



(a) Circuit with modeled gate delay

Present state y	Next state			
	$SR = 00$	01	10	11
	Y	Y	Y	Y
0	(0)	(0)	1	(0)
1	(1)	0	(1)	0

(b) State-assigned table

Figure 9.1 Analysis of the SR latch.

circuit still generates $Y = 0$ and remains stable. Assume next that S changes to 1 and R remains at 1. The value of Y is unchanged, and the circuit is stable. Then let R change to 0 while S remains at 1. This input valuation, $SR = 10$, causes the circuit to generate $Y = 1$. Since $y \neq Y$, the circuit is not stable. After the Δ time delay, the circuit changes to the new present state $y = 1$. Once this new state is reached, the value of Y remains equal to 1 as long as $SR = 10$. Hence the circuit is again stable. The analysis for the present state $y = 1$ can be completed using similar reasoning.

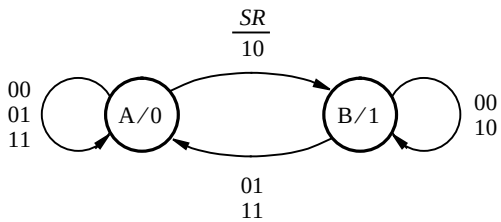
The concept of stable states is very important in the context of asynchronous sequential circuits. For a given valuation of inputs, if a circuit reaches a particular state and remains in this state, then the state is said to be stable. To clearly indicate the conditions under which the circuit is stable, it is customary to encircle the stable states in the table, as illustrated in Figure 9.1*b*.

From the state-assigned table, we can derive the state table in Figure 9.2*a*. The state names A and B represent the present states $y = 0$ and $y = 1$, respectively. Since the output Q depends only on the present state, the circuit is a Moore-type FSM. The state diagram that represents the behavior of this FSM is shown in Figure 9.2*b*.

The preceding analysis shows that the behavior of an asynchronous sequential circuit can be represented as an FSM in a similar way as the synchronous sequential circuits in Chapter 8. Consider now performing the opposite task. That is, given the state table in Figure 9.2*a*, we can synthesize an asynchronous circuit as follows: After performing the

Present state	Next state				Output Q
	$SR = 00$	01	10	11	
A	(A)	(A)	B	(A)	0
B	(B)	A	(B)	A	1

(a) State table



(b) State diagram

Figure 9.2 FSM model for the SR latch.

state assignment, we have the state-assigned table in Figure 9.1*b*. This table represents a truth table for Y, with the inputs y, S, and R. Deriving a minimal product-of-sums expression yields

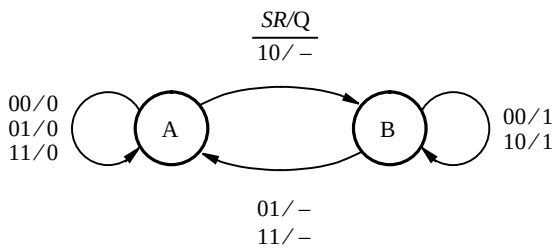
$$Y = \overline{R} \cdot (S + y)$$

If we were deriving a synchronous sequential circuit using the methods in Chapter 8, then Y would be connected to the D input of a flip-flop and a clock signal would be used to control the time when the changes in state take place. But since we are synthesizing an asynchronous circuit, we do not insert a flip-flop in the feedback path. Instead, we create a circuit that realizes the preceding expression using the necessary logic gates, and we feed back the output signal as the present-state input y. Implementation using NOR gates results in the circuit in Figure 9.1*a*. This simple example suggests that asynchronous circuits and synchronous circuits can be synthesized using similar techniques. However, we will see shortly that for more complex asynchronous circuits, the design task is considerably more difficult.

To further explore the nature of asynchronous circuits, it is interesting to consider how the behavior of the SR latch can be represented in the form of a Mealy model. As depicted in Figure 9.3, the outputs produced when the circuit is in a stable state are the same as in the Moore model, namely 0 in state A and 1 in state B. Consider now what happens

Present state	Next state				Output, Q			
	$SR = 00$	01	10	11	00	01	10	11
A	(A)	(A)	B	(A)	0	0	–	0
B	(B)	A	(B)	A	1	–	1	–

(a) State table



(b) State diagram

Figure 9.3 Mealy representation of the SR latch.

when the state of the circuit changes. Suppose that the present state is A and that the input valuation SR changes from 00 to 10. As the state table specifies, the next state of the FSM is B. When the circuit reaches state B, the output Q will be 1. But in the Mealy model, the output is supposed to be affected immediately by a change in the input signals. Thus while still in state A, the change in SR to 10 should result in Q = 1. We could have written a 1 in the corresponding entry in the top row of the state table, but we have chosen to leave this entry unspecified instead. The reason is that since Q will change to 1 as soon as the circuit reaches state B, there is little to be gained in trying to make Q go to 1 a little sooner. Leaving the entry unspecified allows us to assign either 0 or 1 to it, which may make the circuit that implements the state table somewhat simpler. A similar reasoning leads to the conclusion that the two output entries where a change from B to A takes place can also be left unspecified.

Using the state assignment $y = 0$ for A and $y = 1$ for B, the state-assigned table represents a truth table for both Y and Q. The minimal expression for Y is the same as for the Moore model. To derive an expression for Q, we need to set the unspecified entries to 0 or 1. Assigning a 0 to the unspecified entry in the first row and 1 to the two unspecified entries in the second row produces Q = y and results in the circuit in Figure 9.1a.

Terminology

In the preceding discussion we used the same terminology as in the previous chapter on synchronous sequential circuits. However, when dealing with asynchronous sequential

circuits, it is customary to use two different terms. Instead of a "state table," it is more common to speak of a *flow table*, which indicates how the changes in state flow as a result of the changes in the input signals. Instead of a "state-assigned table," it is usual to refer to a *transition table* or an *excitation table*. We will use the terms *flow table* and *excitation table* in this chapter. A flow table will define the state changes and outputs that must be generated. An excitation table will depict the transitions in terms of the state variables. The term excitation table derives from the fact that a change from a stable state is performed by "exciting" the next-state variables to start changing towards a new state.

9.2 ANALYSIS OF ASYNCHRONOUS CIRCUITS

To gain familiarity with asynchronous circuits, it is useful to analyze a few examples. We will keep in mind the general model in Figure 8.90, assuming that the delays in the feedback paths are a representation of the propagation delays in the circuit. Then each gate symbol will represent an ideal gate with zero delay.

Example 9.1 **GATED D LATCH** In Chapters 7 and 8, we used the gated D latch as a key component in circuits that are controlled by a synchronizing clock. It is instructive to analyze this latch as an asynchronous circuit, where the clock is just one of the inputs. It is reasonable to assume that the signals on the D and clock inputs do not change at the same time, thus meeting the basic requirement of asynchronous circuits.

Figure 9.4*a* shows the gated D latch drawn in the style of the model of Figure 8.90. This circuit was introduced in Figure 7.8 and discussed in section 7.3. The next-state expression for this circuit is

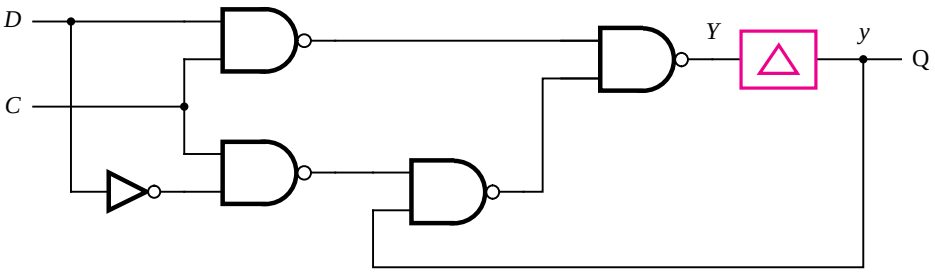
$$\begin{aligned} Y &= (C \uparrow D) \uparrow ((C \uparrow \overline{D}) \uparrow y) \\ &= CD + \overline{C}y + Dy \end{aligned}$$

The term Dy in this expression is redundant and could be deleted without changing the logic function of Y. Hence the minimal expression is

$$Y = CD + \overline{C}y$$

The reason that the circuit implements the redundant term Dy is that this term solves a race condition known as a *hazard*; we will discuss hazards in detail in section 9.6.

Evaluating the expression for Y for all valuations of C, D, and y leads to the excitation table in Figure 9.4*b*. Note that the circuit changes its state only when $C = 1$ and D is different from the present state, y. In all other cases the circuit is stable. Using the symbols A and B to represent the states $y = 0$ and $y = 1$, we obtain the flow table and the state diagram shown in parts (*c*) and (*d*).



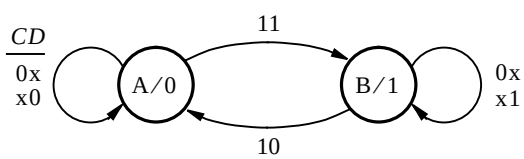
(a) Circuit

Present state y	Next state $CD = 00$	01	10	11	Q
	Y	Y	Y	Y	
0	(0)	(0)	(0)	1	0
1	(1)	(1)	0	(1)	1

(b) Excitation table

Present state	Next state $CD = 00$	01	10	11	Q
A	(A)	(A)	(A)	B	0
B	(B)	(B)	A	(B)	1

(c) Flow table



(d) State diagram

Figure 9.4 The gated D latch.

Example 9.2 **MASTER-SLAVE D FLIP-FLOP** In Example 9.1 we analyzed the gated D latch as an asynchronous circuit. Actually, all practical circuits are asynchronous. However, if the circuit's behavior is tightly controlled by a clock signal, then simpler operating assumptions can be used, as we did in Chapter 8. Recall that in a synchronous sequential circuit all signals change values in synchronization with the clock signal. Now we will analyze another synchronous circuit as if it were an asynchronous circuit.

Two gated D latches are used to implement the master-slave D flip-flop, as illustrated in Figure 7.10. This circuit is reproduced in Figure 9.5. We can analyze the circuit by treating it as a series connection of two gated D latches. Using the results from Example 9.1, the simplified next-state expressions can be written as

$$Y_m = CD + \overline{C}y_m$$
$$Y_s = \overline{C}y_m + Cy_s$$

where the subscripts m and s refer to the master and slave stages of the flip-flop. These expressions lead to the excitation table in Figure 9.6*a*. Labeling the four states as $S1$ through $S4$, we derive the flow table in Figure 9.6*b*. A state-diagram form of this information is given in Figure 9.7.

Let us consider the behavior of this FSM in more detail. The state $S1$, where $y_m y_s = 00$, is stable for all input valuations except $CD = 11$. When $C = 1$, the value of D is stored in the master stage; hence $CD = 11$ causes the flip-flop to change to S3, where $y_m = 1$ and $y_s = 0$. If the D input now changes back to 0, while the clock remains at 1, the flip-flop moves back to the state $S1$. The transitions between $S1$ and $S3$ indicate that if $C = 1$, the output of the master stage, $Q_m = y_m$, tracks the changes in the D input signal without affecting the slave stage. From $S3$ the circuit changes to $S4$ when the clock goes to 0. In $S4$ both master and slave stages are set to 1 because the information from the master stage is transferred to the slave stage on the negative edge of the clock. Now the flip-flop remains in $S4$ until the clock goes to 1 and the D input changes to 0, which causes a change to $S2$. In $S2$ the master stage is cleared to 0, but the slave stage remains at 1. Again the flip-flop may change between $S2$ and $S4$ because the master stage will track the changes in the D input signal while $C = 1$. From $S2$ the circuit changes to $S1$ when the clock goes low.

In Figures 9.6 and 9.7, we indicated that the flip-flop has only one output Q, which one sees when the circuit is viewed as a negative-edge-triggered flip-flop. From the observer's

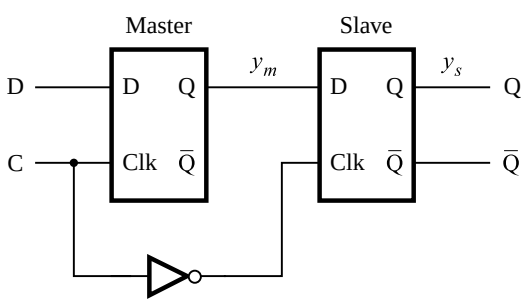


Figure 9.5 Circuit for the master-slave D flip-flop.

Present state $y_m y_s$	Next state $CD = 00$	01	10	11	Output Q
	$Y_m Y_s$				
00	(00)	(00)	(00)	10	0
01	00	00	(01)	11	1
10	11	11	00	(10)	0
11	(11)	(11)	01	(11)	1

(a) Excitation table

Present state	Next state $CD = 00$	01	10	11	Output Q
S1	(S1)	(S1)	(S1)	S3	0
S2	S1	S1	(S2)	S4	1
S3	S4	S4	S1	(S3)	0
S4	(S4)	(S4)	S2	(S4)	1

(b) Flow table

Present state	Next state $CD = 00$	01	10	11	Output Q
S1	(S1)	(S1)	(S1)	S3	0
S2	S1	–	(S2)	S4	1
S3	–	S4	S1	(S3)	0
S4	(S4)	(S4)	S2	(S4)	1

(c) Flow table with unspecified entries

Figure 9.6 Excitation and flow tables for Example 9.2.

point of view, the flip-flop has only two states, 0 and 1. But internally, the flip-flop consists of the master and slave parts, which gives rise to the four states described above.

We should also examine the basic assumption that the inputs must change one at a time. If the circuit is stable in state $S2$, for which $CD = 10$, it is impossible to go from this state to $S1$ under the influence of the input valuation $CD = 01$ because this simultaneous change

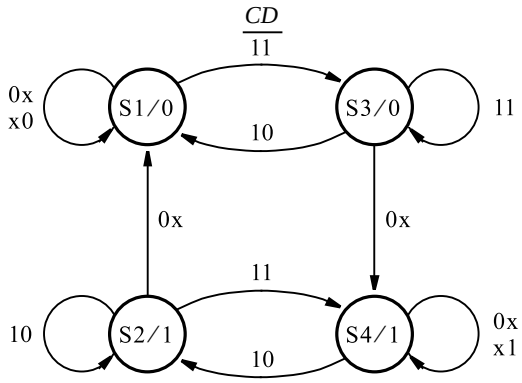


Figure 9.7 State diagram for the master-slave D flip-flop.

in both inputs cannot occur. Thus in the second row of the flow table, instead of showing $S2$ changing to $S1$ under $CD = 01$, this entry can be labeled as unspecified. The change from $S2$ to $S1$ can be caused only by CD changing from 10 to 00. Similarly, if the circuit is in state $S3$, where $CD = 11$, it cannot change to $S4$ by having $CD = 00$. This entry can also be left unspecified in the table. The resulting flow table is shown in Figure 9.6*c*.

If we reverse the analysis procedure and, using the state assignment in Figure 9.6*a*, synthesize logic expressions for Y_m and Y_s, we get

$$Y_m = CD + \overline{C}y_m + y_mD$$
$$Y_s = \overline{C}y_m + Cy_s + y_my_s$$

The terms y_mD and y_my_s in these expressions are redundant. As mentioned earlier, they are included in the circuit to avoid race conditions, which are discussed in section 9.6.

Example 9.3 Consider the circuit in Figure 9.8. It is represented by the following expressions

$$Y_1 = y_1\overline{y}_2 + w_1\overline{y}_2 + \overline{w}_1\overline{w}_2y_1$$
$$Y_2 = y_1y_2 + w_1y_2 + w_2 + \overline{w}_1\overline{w}_2y_1$$
$$z = \overline{y}_1y_2$$

The corresponding excitation and flow tables are given in Figure 9.9.

Some transitions in the flow table will not occur in practice because of the assumption that both w_1 and w_2 cannot change simultaneously. In state A the circuit is stable under the valuation $w_2w_1 = 00$. Its inputs cannot change to 11 without passing through the valuations 01 or 10, in which case the new state would be B or C, respectively. Thus the transition from A under $w_2w_1 = 11$ can be left unspecified. Similarly, if the circuit is stable in state B, in which case $w_2w_1 = 01$, it is impossible to force a change to state D by changing the inputs to $w_2w_1 = 10$. This entry should also be unspecified. If the circuit is stable in state C under $w_2w_1 = 11$, it is not possible to go to A by changing the inputs directly to $w_2w_1 = 00$.

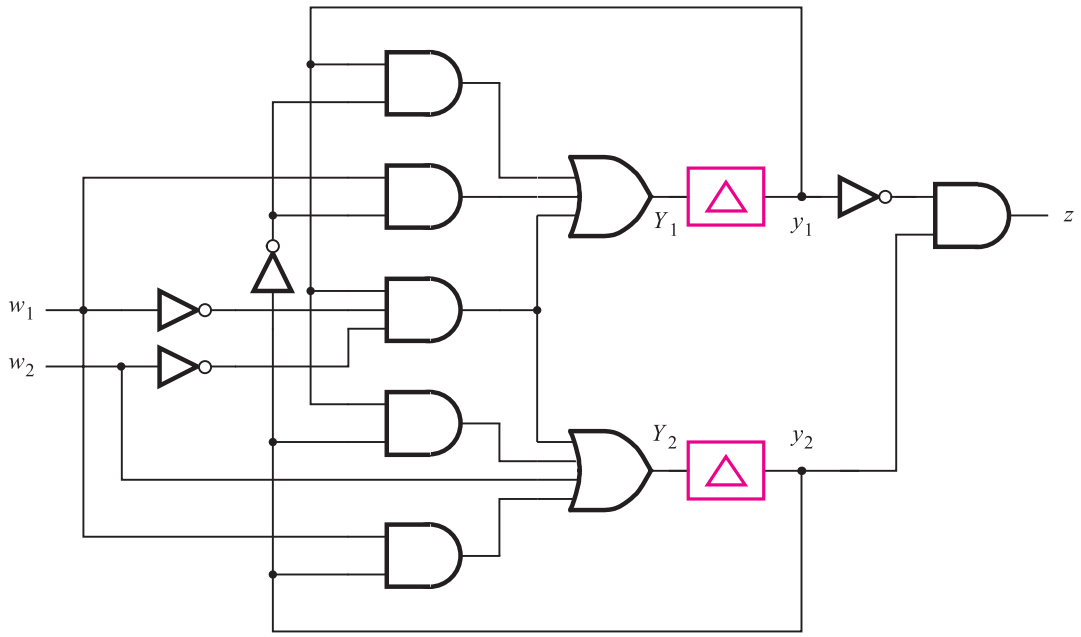


Figure 9.8 Circuit for Example 9.3.

However, the transition to A is possible by changing the inputs one at a time because the circuit remains stable in C for both $w_2w_1 = 01$ and $w_2w_1 = 10$.

A different situation arises if the circuit is stable in state D under $w_2w_1 = 00$. It may seem that the entry under $w_2w_1 = 11$ should be unspecified because this input change cannot be made from the stable state D. But suppose that the circuit is stable in state B under $w_2w_1 = 01$. Now let the inputs change to $w_2w_1 = 11$. This causes a change to state D. The circuit indeed changes to D, but it is not stable in this state for this input condition. As soon as it arrives into state D, the circuit proceeds to change to state C as required by $w_2w_1 = 11$. It is then stable in state C as long as both inputs remain at 1. The conclusion is that the entry that specifies the change from D to C under $w_2w_1 = 11$ is meaningful and should not be omitted. The transition from the stable state B to the stable state C, which passes through state D, illustrates that it is not imperative that all transitions be directly from one stable state to another. A state through which a circuit passes en route from one stable state to another is called an *unstable state*. Transitions that involve passing through an unstable state are not harmful as long as the unstable state does not generate an undesirable output signal. For example, if a transition is between two stable states for which the output signal should be 0, it would be unacceptable to pass through an unstable state that causes the output to be 1. Even though the circuit changes through the unstable state very quickly, the short glitch in the output signal is likely to be troublesome. This is not a problem in our example. When the circuit is stable in B, the output is $z = 0$. When the inputs change to $w_2w_1 = 11$, the transition to state D maintains the output at 0. It is only when the circuit finally changes into state C that z will change to 1. Therefore, the change from $z = 0$ to $z = 1$ occurs only once during the course of these transitions.

Present state y_2y_1	Next state $w_2w_1 = 00$	01	10	11	Output z
	Y_2Y_1	Y_2Y_1	Y_2Y_1	Y_2Y_1	
00	(00)	01	10	11	0
01	11	(01)	11	11	0
10	00	(10)	(10)	(10)	1
11	(11)	10	10	10	0

(a) Excitation table

Present state	Next state $w_2w_1 = 00$	01	10	11	Output z
A	(A)	B	C	D	0
B	D	(B)	D	D	0
C	A	(C)	(C)	(C)	1
D	(D)	C	C	C	0

(b) Flow table

Figure 9.9 Excitation and flow tables for the circuit in Figure 9.8.

Present state	Next state $w_2w_1 = 00$	01	10	11	Output z
A	(A)	B	C	–	0
B	D	(B)	–	D	0
C	A	(C)	(C)	(C)	1
D	(D)	C	C	C	0

Figure 9.10 Modified flow table for Example 9.3.

A modified flow table, showing the unspecified transitions, is presented in Figure 9.10. The table indicates the behavior of the circuit in Figure 9.8 in terms of state transitions. If we don't know what the circuit is supposed to do, it may be difficult to discover the practical application for a given circuit. Fortunately, in practice the purpose of the circuit is known, and the analysis is done by the designer to ascertain that the circuit performs as desired. In

our example it is apparent that the circuit generates the output $z = 1$ in state C, which it reaches as a result of some input patterns that are detected using the other three states. The state diagram derived from Figure 9.10 is shown in Figure 9.11.

This diagram actually implements a control mechanism for a simple vending machine that accepts two types of coins, say, dimes and nickels, and dispenses merchandise such as candy. If w_1 represents a nickel and w_2 represents a dime, then a total of 10 cents must be deposited to get the FSM into state C where the candy is released. The coin mechanism accepts only one coin at a time, which means that $w_2w_1 = 11$ can never occur. Therefore, the transition discussed above, from B to C, through the unstable state D would not occur. Observe that both states B and D indicate that 5 cents has been deposited. State B indicates that a nickel is presently being sensed by the coin receptor, while D indicates that 5 cents has been deposited and the coin receptor is presently empty. In state D it is possible to deposit either a nickel or a dime, both leading to state C. No distinction is made between the two types of coins in state D; hence the machine would not give change if 15 cents is deposited. From state A a dime leads directly to state C. Knowing that the condition $w_2w_1 = 11$ will not occur allows the flow table to be specified as shown in Figure 9.12. If

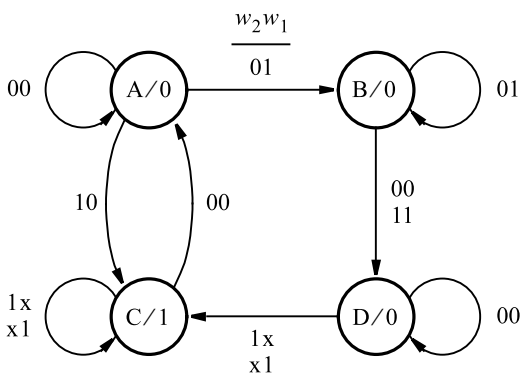


Figure 9.11 State diagram for Example 9.3.

Present state	Next state				Output z
	$w_2w_1 = 00$	01	10	11	
A	(A)	B	C	—	0
B	D	(B)	—	—	0
C	A	(C)	(C)	—	1
D	(D)	C	C	—	0

$w_2 \equiv$ dime $\quad w_1 \equiv$ nickel

Figure 9.12 Flow table for a simple vending machine.

we were to synthesize the sum-of-products logic expressions for Y_1 and Y_2, using the state assignment in Figure 9.9*a*, we would end up with the circuit in Figure 9.8.

Steps in the Analysis Process

We have demonstrated the analysis process using illustrative examples. The required steps can be stated as follows:

- A given circuit is interpreted in the form of the general model in Figure 8.90. That is, each feedback path is cut, and a delay element is inserted at the point where the cut is made. The input signal to the delay element represents a corresponding next-state variable, Y_i, while the output signal is the present-state variable, y_i. A cut can be made anywhere in a particular loop formed by the feedback connection, as long as there is only one cut per (state variable) loop. Thus the number of cuts that should be made is the smallest number that results in there being no feedback anywhere in the circuit except from the output of a delay element. This minimal number of cuts is sometimes referred to as the *cut set*. Note that the analysis based on a cut made at one point in a given loop may not produce the same flow table as an analysis on a cut made at some other point in this loop. But both flow tables would reflect the same functional behavior in terms of the applied inputs and generated outputs.
- Next-state and output expressions are derived from the circuit.
- The excitation table corresponding to the next-state and output expressions is derived.
- A flow table is obtained, associating some (arbitrary) names with the particular encoded states.
- A corresponding state diagram is derived from the flow table if desired.

9.3 SYNTHESIS OF ASYNCHRONOUS CIRCUITS

Synthesis of asynchronous sequential circuits follows the same basic steps used to synthesize the synchronous circuits, which were discussed in Chapter 8. There are some differences due to the asynchronous nature, which make the asynchronous circuits more difficult to design. We will explain the differences by investigating a few design examples. The basic steps are

- Devise a state diagram for an FSM that realizes the required functional behavior.
- Derive the flow table and reduce the number of states if possible.
- Perform the state assignment and derive the excitation table.
- Obtain the next-state and output expressions.
- Construct a circuit that implements these expressions.

When devising a state diagram, or perhaps the flow table directly, it is essential to ensure that when the circuit is in a stable state, the correct output signals are generated. Should it

be necessary to pass through an unstable state, this state must not produce an undesirable output signal.

Minimization of states is not straightforward. A minimization procedure is described in section 9.4.

State assignment is not done with the sole purpose of reducing the cost of the final circuit. In asynchronous circuits some state assignments may cause the circuit to be unreliable. We will explain this problem using the examples that follow.

Example 9.4

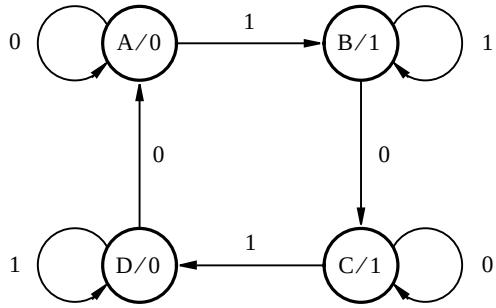
SERIAL PARITY GENERATOR Suppose that we want to design a circuit that has an input w and an output z, such that when pulses are applied to w, the output z is equal to 0 if the number of previously applied pulses is even and z is equal to 1 if the number of pulses is odd. Hence the circuit acts as a serial parity generator.

Let A be the state that indicates that an even number of pulses has been received. Using the Moore model, the output z will be equal to 0 when the circuit is in state A. As long as $w = 0$, the circuit should remain in A, which is specified by a transition arc that both originates and terminates in state A. Thus A is stable when $w = 0$. When the next pulse arrives, the input $w = 1$ should cause the FSM to move to a new state, say, B, which produces the output $z = 1$. When the FSM reaches B, it must remain stable in this state as long as $w = 1$. This is specified by a transition arc that originates and terminates in B. The next input change occurs when w goes to 0. In response the FSM must change to a state where $z = 1$ and which corresponds to the fact that a complete pulse has been observed, namely, that w has changed from 1 to 0. Let this state be C; it must be stable under the input condition $w = 0$. The arrival of the next pulse makes $w = 1$, and the FSM must change to a state, D, that indicates that an even number of pulses has been observed and that the last pulse is still present. The state D is stable under $w = 1$, and it causes the output to be $z = 0$. Finally, when w returns to 0 at the end of the pulse, the FSM returns to state A, which indicates an even number of pulses and w equal to 0 at the present time. The resulting state diagram is shown in Figure 9.13*a*.

A key point to understand is why it is necessary to have four states rather than just two, considering that we are merely trying to distinguish between the even and odd number of input pulses. States B and C cannot be combined into a single state even though they both indicate that an odd number of pulses has been observed. Suppose we had simply tried to use state B alone for this purpose. Then it would have been necessary to add an arc with a label 0 that originates and terminates in state B, which is fine. The problem is that without state C, there would have to be a transition from state B directly to D if the input is $w = 1$ to respond to the next change in the input when a new pulse arrives. It would be impossible to have B both stable under $w = 1$ and have a change to D effected for the same input condition. Similarly, we can show that the states A and D cannot be combined into a single state.

Figure 9.13*b* gives the flow table that corresponds directly to the state diagram. In many cases the designer can derive a flow table directly. We are using the state diagram mostly because it provides a simpler visual picture of the effect of the transitions in an FSM.

The next step is to assign values to the states in terms of the state variables. Since there are four states in our FSM, there have to be at least two state variables. Let these variables



(a) State diagram

Present State	Next state		Output z
	$w = 0$	$w = 1$	
A	(A)	B	0
B	C	(B)	1
C	(C)	D	1
D	A	(D)	0

(b) Flow table

Figure 9.13 Parity-generating asynchronous FSM.

be y_1 and y_2. As a first attempt at the state assignment, let the states A, B, C, and D be encoded as $y_2y_1 = 00$, 01, 10, and 11, respectively. This assignment leads to the excitation table in Figure 9.14*a*. Unfortunately, it has a major flaw. The circuit that implements this table is stable in state $D = 11$ under the input condition $w = 1$. But consider what happens next if the input changes to $w = 0$. According to the excitation table, the circuit should change to state $A = 00$ and remain stable in this state. The problem is that in going from $y_2y_1 = 11$ to $y_2y_1 = 00$ both state variables must change their values. This is unlikely to occur at exactly the same time. In an asynchronous circuit the values of the next-state variables are determined by networks of logic gates with varying propagation delays. Thus we should expect that one state variable will change slightly before the other, which could put the circuit into a state where it may react to the input in an undesirable way. Suppose that y_1 changes first. Then the circuit goes from $y_2y_1 = 11$ to $y_2y_1 = 10$. As soon as it reaches this state, C, it will attempt to remain there if $w = 0$, which is a wrong outcome. On the other hand, suppose that y_2 changes first. Then there will be a change from $y_2y_1 = 11$ to $y_2y_1 = 01$, which corresponds to state B. Since $w = 0$, the circuit will now try to change to $y_2y_1 = 10$. This again requires that both y_1 and y_2 change; assuming that y_1 changes first

Present state y_2y_1	Next state Y_2Y_1, $w = 0$	Next state Y_2Y_1, $w = 1$	Output z
00	(00)	01	0
01	10	(01)	1
10	(10)	11	1
11	00	(11)	0

(a) Poor state assignment

Present state y_2y_1	Next state Y_2Y_1, $w = 0$	Next state Y_2Y_1, $w = 1$	Output z
00	(00)	01	0
01	11	(01)	1
11	(11)	10	1
10	00	(10)	0

(b) Good state assignment

Figure 9.14 State assignment for Figure 9.13*b*.

in the transition from $y_2y_1 = 01$, the circuit will find itself in the state $y_2y_1 = 00$, which is the correct destination state, A. This discussion indicates that the required transition from D to A will be performed correctly if y_2 changes before y_1, but it will not work if y_1 changes before y_2. The result depends on the outcome of the "race" to change between the signals y_1 and y_2.

The uncertainty caused by multiple changes in the state variables in response to an input that should lead to a predictable change from one stable state to another has to be eliminated. The term *race condition* is used to refer to such unpredictable behavior. We will discuss this issue in detail in section 9.5.

Race conditions can be eliminated by treating the present-state variables as if they were inputs to the circuit, meaning that only one state variable is allowed to change at a time. For our example the assignment $A = 00$, $B = 01$, $C = 11$, and $D = 10$ achieves this objective. The resulting excitation table is presented in Figure 9.14*b*. The reader should verify that all transitions involve changing a single state variable.

From Figure 9.14*b* the next-state and output expressions are

$$Y_1 = w\bar{y}_2 + \bar{w}y_1 + y_1\bar{y}_2$$
$$Y_2 = wy_2 + \bar{w}y_1 + y_1y_2$$
$$z = y_1$$

The last product term in the expressions for Y_1 and Y_2 is included to deal with possible hazards, which are discussed in section 9.6. The corresponding circuit is shown in Figure 9.15.

It is interesting to consider how the serial parity generator could be implemented using a synchronous approach. All that is needed is a single flip-flop that changes its state with the arrival of each input pulse. The positive-edge-triggered *D* flip-flop in Figure 9.16

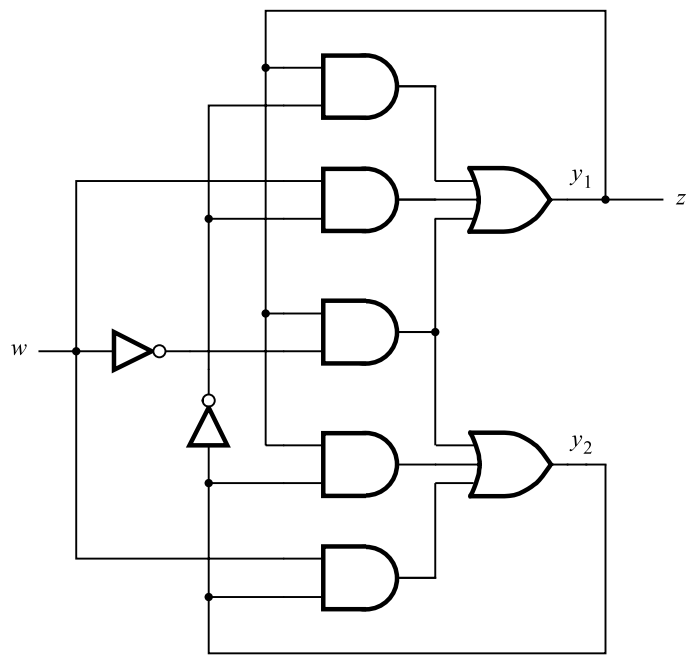


Figure 9.15 Circuit that implements the FSM in Figure 9.13*b*.

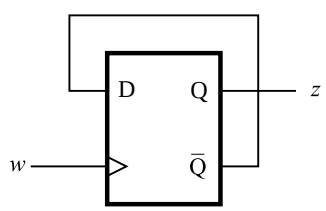


Figure 9.16 Synchronous solution for Example 9.4.

accomplishes the task, assuming that the flip-flop is initially set to Q = 0. The logic complexity of the flip-flop is exactly the same as the circuit in Figure 9.15. Indeed, if we use the preceding expressions for Y_1 and Y_2 and substitute C for w, D for $\bar{y}_2$, y_m for y_1, and y_s for y_2, we end up with the excitation expressions shown for the master-slave D flip-flop in Example 9.2. The circuit in Figure 9.15 is actually a negative-edge-triggered master-slave flip-flop, with the complement of its Q output (y_2) connected to its D input. The output z is connected to the output of the master stage of the flip-flop.

MODULO-4 COUNTER **Example 9.5**

Chapters 7 and 8 described how counters can be implemented using flip-flops. Now we will synthesize a counter as an asynchronous sequential circuit. Figure 9.17 depicts a state diagram for a modulo-4 up-counter, which counts the number of pulses on an input line, w. The circuit must be able to react to all changes in the input signal; thus it must take specific actions at both the positive and negative edges of each pulse. Therefore, eight states are needed to deal with the edges in four consecutive pulses.

The counter begins in state A and stays in this state as long as $w = 0$. When w changes to 1, a transition to state B is made and the circuit remains stable in this state as long as $w = 1$. When w goes back to 0, the circuit moves to state C and remains stable until w becomes 1 again, which causes a transition to state D, and so on. Using the Moore model, the states correspond to specific counts. There are two states for each particular count: the state that the FSM enters when w changes from 0 to 1 at the start of a pulse and the state that the FSM enters when w goes back to 0 at the end of the pulse. States B and C correspond to the count of 1, states D and E to 2, and states F and G to 3. States A and H represent the count of 0.

Figure 9.18 shows the flow and excitation tables for the counter. The state assignment is chosen such that all transitions between states require changing the value of only one

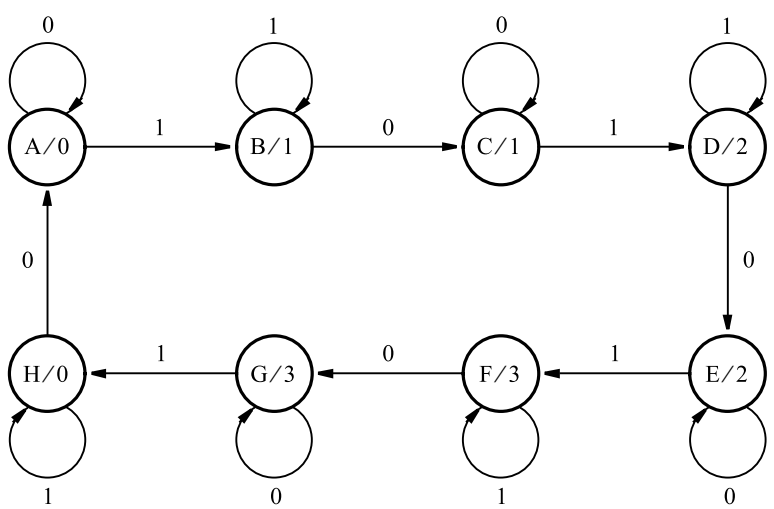


Figure 9.17 State diagram for a modulo-4 counter.

Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	(A)	B	0
B	C	(B)	1
C	(C)	D	1
D	E	(D)	2
E	(E)	F	2
F	G	(F)	3
G	(G)	H	3
H	A	(H)	0

(a) Flow table

Present state	Next state		Output
	$w = 0$	$w = 1$	
$y_3y_2y_1$	$Y_3Y_2Y_1$		z_2z_1
000	(000)	001	00
001	011	(001)	01
011	(011)	010	01
010	110	(010)	10
110	(110)	111	10
111	101	(111)	11
101	(101)	100	11
100	000	(100)	00

(b) Excitation table

Mod-8 output
$z_3z_2z_1$
000
001
010
011
100
101
110
111

(c) Output for counting the edges

Figure 9.18 Flow and excitation tables for a modulo-4 counter.

state variable to eliminate the possibility of race conditions. The output is encoded as a binary number, using variables z_2 and z_1. From the excitation table the next-state and output expressions are

$$\begin{aligned}
Y_1 &= \overline{w}y_1 + wy_2y_3 + w\overline{y}_2\overline{y}_3 + y_1y_2y_3 + y_1\overline{y}_2\overline{y}_3 \\
&= \overline{w}y_1 + (w + y_1)(y_2y_3 + \overline{y}_2\overline{y}_3) \\
Y_2 &= wy_2 + \overline{w}y_1\overline{y}_3 + \overline{y}_1y_2 + y_2\overline{y}_3 \\
Y_3 &= wy_3 + y_1y_3 + \overline{y}_1y_2\overline{w} + y_2y_3 \\
z_1 &= y_1 \\
z_2 &= y_1y_3 + \overline{y}_1y_2
\end{aligned}$$

These expressions define the circuit that implements the required modulo-4 pulse counter.

In the preceding derivation we designed a circuit that changes its state on every edge of the input signal w, requiring a total of eight states. Since the circuit is supposed to count the number of complete pulses, which contain a rising and a falling edge, the output count z_2z_1 changes its value only in every second state. This FSM behaves like a synchronous sequential circuit in which the output count changes only as a result of w changing from 0 to 1.

Suppose now that we want to count the number of times the signal w changes its value, that is, the number of its edges. The state transitions specified in Figures 9.17 and 9.18 define an FSM that can operate as a modulo-8 counter for this purpose. We only need to specify a distinct output in each state, which can be done as shown in Figure 9.18*c*. The values of $z_3z_2z_1$ indicate the counting sequence 0, 1, 2, . . . , 7, 0. Using this specification of the output and the state assignment in Figure 9.18*b*, the resulting output expressions are

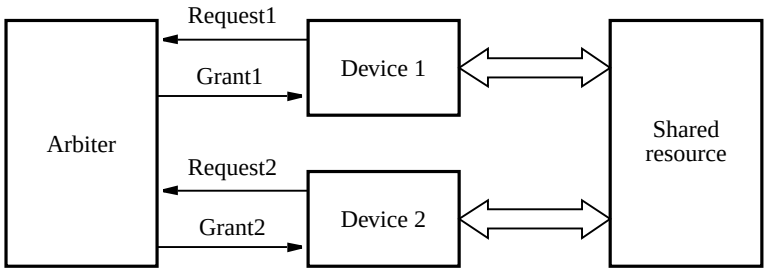
$$\begin{aligned}
z_1 &= y_1 \oplus y_2 \oplus y_3 \\
z_2 &= y_2 \oplus y_3 \\
z_3 &= y_3
\end{aligned}$$

Example 9.6

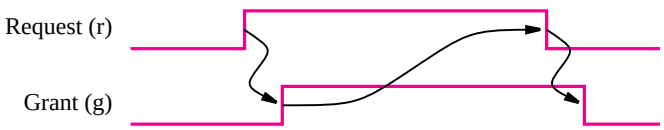
A SIMPLE ARBITER In computer systems it is often useful to have some resource shared by a number of different devices. Usually, the resource can be used by only one device at a time. When various devices need to use the resource, they have to request to do so. These requests are handled by an arbiter circuit. When there are two or more outstanding requests, the arbiter may use some priority scheme to choose one of them, as already discussed in section 8.8.

We will now consider an example of a simple arbiter implemented as an asynchronous sequential circuit. To keep the example small, suppose that two devices are competing for the shared resource, as indicated in Figure 9.19*a*. Each device communicates with the arbiter by means of two signals—*Request* and *Grant*. When a device needs to use the shared resource, it raises its Request signal to 1. Then it waits until the arbiter responds with the Grant signal.

Figure 9.19*b* illustrates a commonly used scheme for communication between two entities in the asynchronous environment, known as *handshake signaling*. Two signals are



(a) Arbitration structure



(b) Handshake signaling

Figure 9.19 Arbitration example.

used to provide the handshake. A device initiates the activity by raising a request, $r = 1$. When the shared resource is available, the arbiter responds by issuing a grant, $g = 1$. When the device receives the grant signal, it proceeds to use the requested shared resource. When it completes its use of the resource, it drops its request by setting $r = 0$. When the arbiter sees that $r = 0$, it deactivates the grant signal, making $g = 0$. The arrows in the figure indicate the cause-effect relationships in this signaling scheme; a change in one signal causes a change in the other signal. The time elapsed between the changes in the cause-effect signals depends on the specific implementation of the circuit. A key point is that there is no need for a synchronizing clock.

A state diagram for our simple arbiter is given in Figure 9.20. There are two inputs, the request signals r_1 and r_2, and two outputs, the grant signals g_1 and g_2. The diagram depicts the Moore model of the required FSM, where the arcs are labeled as r_2r_1 and the state outputs as g_2g_1. The quiescent state is A, where there are no requests. State B represents the situation in which Device 1 is given permission to use the resource, and state C denotes the same for Device 2. Thus B is stable if $r_2r_1 = 01$, and C is stable if $r_2r_1 = 10$. To conform to the rules of asynchronous circuit design, we will assume that the inputs r_1 and r_2 become activated one at a time. Hence, in state A it is impossible to have a change from $r_2r_1 = 00$ to $r_2r_1 = 11$. The situation where $r_2r_1 = 11$ occurs only when a second request is raised before the device that has the grant signal completes its use of the shared resource, which can happen in states B and C. If the FSM is stable in either state B or C, it will remain in this state if both r_1 and r_2 go to 1.

The flow table is given in Figure 9.21*a*, and the excitation table is presented in Figure 9.21*b*. It is impossible to choose a state assignment such that all changes between states

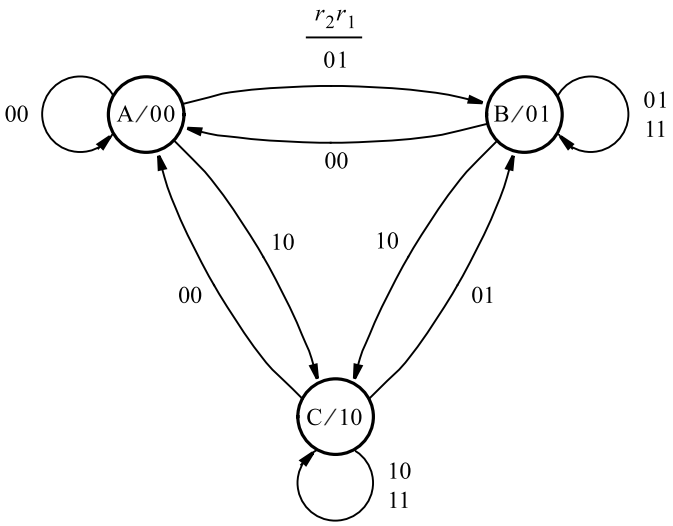


Figure 9.20 State diagram for the arbiter.

Present state	Next state $r_2r_1 = 00$	01	10	11	Output g_2g_1
A	(A)	B	C	–	00
B	A	(B)	C	(B)	01
C	A	B	(C)	(C)	10

(a) Flow table

	Present state y_2y_1	Next state $r_2r_1 = 00$	01	10	11	Output g_2g_1
		Y_2Y_1				
A	00	(00)	01	10	–	00
B	01	00	(01)	10	(01)	01
C	10	00	01	(10)	(10)	10
D	11	–	01	10	–	dd

(b) Excitation table

Figure 9.21 Implementation of the arbiter.

A, B, and C involve a change in a single state variable only. In the chosen assignment the transitions to or from state A are handled properly, but the transitions between states B and C involve changes in the values of both state variables y_1 and y_2. Suppose that the circuit is stable in state B under input valuation $r_2r_1 = 11$. Now let the inputs change to $r_2r_1 = 10$. This should cause a change to state C, which means that the state variables must change from $y_2y_1 = 01$ to 10. If y_1 changes faster than y_2, then the circuit will find itself momentarily in state $y_2y_1 = 00$, which leads to the desired final state because from state A there is a specified transition to C under the input valuation 10. But if y_2 changes faster than y_1, the circuit will reach the state $y_2y_1 = 11$, which is not defined in the flow table. To make sure that even in this case the circuit will proceed to the required destination C, we can include the state $y_2y_1 = 11$, labeled D, in the excitation table and specify the required transition as shown in the figure. A similar situation arises when the circuit is stable in C under $r_2r_1 = 11$, and it has to change to B when r_2 changes from 1 to 0.

The output values for the extra state D are indicated as don't cares. Whenever a specific output is changing from 0 to 1 or from 1 to 0, exactly when this change takes place is not important if the correct value is produced when the circuit is in a stable state. The don't-care specification may lead to a simpler realization of the output functions. It is important to ensure that unspecified outputs will not result in a value that may cause erroneous behavior. From Figure 9.21*b* it is possible that during the short time when the circuit passes through the unstable state D the outputs become $g_2g_1 = 11$. This is harmless in our example because the device that has just finished using the shared resource will not try to use it again until its grant signal has returned to 0 to indicate the end of the handshake with the arbiter. Observe that if this condition occurs when changing from B to C, then g_1 remains 1 slightly longer and g_2 becomes 1 slightly earlier. Similarly, if the transition is from C to B, then the change in g_1 from 0 to 1 happens slightly earlier and g_2 changes to 0 slightly later. In both of these cases there is no glitch on either g_1 or g_2.

From the excitation table the following next-state and output expressions are derived

$$\begin{aligned} Y_1 &= \bar{r}_2 r_1 + r_1 \bar{y}_2 \\ Y_2 &= r_2 \bar{r}_1 + r_2 y_2 \\ g_1 &= y_1 \\ g_2 &= y_2 \end{aligned}$$

Rewriting the first two expressions as

$$\begin{aligned} Y_1 &= r_1(\bar{r}_2 + \bar{y}_2) \\ &= r_1 \overline{r_2 y_2} \\ Y_2 &= r_2(\bar{r}_1 + y_2) \end{aligned}$$

produces the circuit in Figure 9.22. Observe that this circuit responds very quickly to the changes in the input signals. This behavior is in sharp contrast to the arbiter discussed in section 8.8 in which the synchronizing clock determines the minimum response time.

The difficulty with the race condition that arises in state changes between B and C can be resolved in another way. We can simply prevent the circuit from reaching an unspecified state. Figure 9.23*a* shows a modified flow table in which transitions between states B and

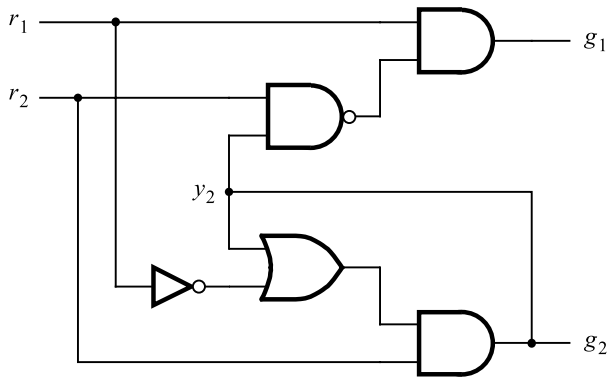


Figure 9.22 The arbiter circuit.

Present state	Next state				Output g_2g_1
	$r_2r_1 = 00$	01	10	11	
A	(A)	B	C	–	00
B	A	(B)	A	(B)	01
C	A	A	(C)	(C)	10

(a) Modified flow table

Present state y_2y_1	Next state				Output g_2g_1
	$r_2r_1 = 00$	01	10	11	
	Y_2Y_1				
00	(00)	01	10	–	00
01	00	(01)	00	(01)	01
10	00	00	(10)	(10)	10

(b) Modified excitation table

Figure 9.23 An alternative for avoiding a critical race in Figure 9.21*a*.

C are made via state A. If the circuit is stable in B and the input valuation changes from $r_2r_1 = 11$ to 10, a change to A will occur first. As soon as the circuit reaches A, which is not stable for the input valuation 10, it will proceed to the stable state C. The detour through the unstable state A is acceptable because in this state the output is $g_2g_1 = 00$, which is consistent with the desired operation of the arbiter. The change from C to B is handled using the same approach. From the modified excitation table in Figure 9.23*b*, the following next-state expressions are derived

$$Y_1 = r_1\bar{y}_2$$
$$Y_2 = \bar{r}_1 r_2 \bar{y}_1 + r_2 y_2$$

These expressions give rise to a circuit different from the one in Figure 9.22. However, both circuits implement the functionality required in the arbiter.

Next we will attempt to design the same arbiter using the Mealy model specification. From Figure 9.20 it is apparent that the states B and C are fundamentally different because for the input $r_2r_1 = 11$ they must produce two different outputs. But state A is unique only to the extent that it generates the output $g_2g_1 = 00$ whenever $r_2r_1 = 00$. This condition could be specified in both B and C if the Mealy model is used. Figure 9.24 gives a suitable state diagram. The flow and excitation tables are presented in Figure 9.25, which lead to the following expressions

$$Y = r_2\bar{r}_1 + \bar{r}_1 y + r_2 y$$
$$g_1 = r_1\bar{y}$$
$$g_2 = r_2 y$$

Despite needing a single state variable, this circuit requires more gates for implementation than does the Moore version in Figure 9.22.

An important notion in the above examples is that it is necessary to pay careful attention to the state assignment, to avoid races in changing of the values of the state variables. Section 9.5 deals with this issue in more detail.

We made the basic assumption that the request inputs to the arbiter FSM change their values one at a time, which allows the circuit to reach a stable state before the next change takes place. If the devices are totally independent, they can raise their requests at any time. Suppose that each device raises a request every few seconds. Since the arbiter circuit needs only a few nanoseconds to change from one stable state to another, it is quite unlikely that both devices will raise their requests so close to each other that the arbiter circuit will produce erroneous outputs. However, while the probability of an error caused by the simultaneous

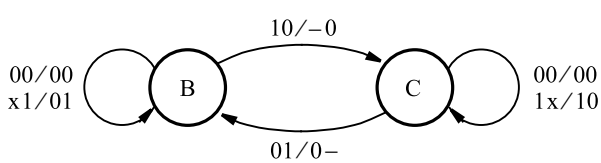


Figure 9.24 Mealy model for the arbiter FSM.

Present state	Next state				Output g_2g_1			
	$r_2r_1 = 00$	01	10	11	00	01	10	11
B	(B)	(B)	C	(B)	00	01	−0	01
C	(C)	B	(C)	(C)	00	0−	10	10

(a) Flow diagram

Present state y	Next state				Output			
	$r_2r_1 = 00$	01	10	11	00	01	10	11
	Y				g_2g_1			
0	(0)	(0)	1	(0)	00	01	$d0$	01
1	(1)	0	(1)	(1)	00	$0d$	10	10

(b) Excitation table

Figure 9.25 Mealy model implementation of the arbiter FSM.

arrival of requests is extremely low, it is not zero. If this small possibility of an error cannot be tolerated, then it is possible to feed the request signals through a special circuit called the *mutual exclusion* (ME) element. This circuit has two inputs and two outputs. If both inputs are 0, then both outputs are 0. If only one input is 1, then the corresponding output is 1. If both inputs are 1, the circuit makes one output go to 1 and keeps the other at 0. Using the ME element would change the design of the arbiter slightly; because the valuation $r_2r_1 = 11$ would never occur, all entries in the corresponding column in Figure 9.21 would be don't cares. The ME element and the issue of simultaneous changes in input signals are discussed in detail in reference [6]. Finally, we should note that a similar problem arises in synchronous circuits in which one or more inputs are generated by a circuit that is not controlled by a common clock. We will deal with this issue in section 10.3.3 in Chapter 10.

9.4 STATE REDUCTION

In Chapter 8 we saw that reducing the number of states needed to realize the functionality of a given FSM usually leads to fewer state variables, which means that fewer flip-flops are required in the corresponding synchronous sequential circuit. In asynchronous sequential circuits it is also useful to try to reduce the number of states because this usually results in simpler implementations.