False and w, r are "don't cares." With t = F, the path from s_1 exit to s_1 entrance is now active (Figure 7.15). The ASM dwells in s_1 until t goes True in period 11, and during this period the path from s_1 to s_0 is open (Figure 7.15). Clock edge 11 moves the ASM from state s_1 to state s_0.

On the waveforms, the values of "don't care" variables are marked with an x and the values of "care" variables are marked with a small circle (○). For example, input w is associated with s_0. When the state machine is in s_0, w is active (we care, so we use ○). When the state machine is in other states, w is inactive (we don't care, so we use x).

EXERCISE 7.12

Starting from state s_2 in Figure 7.15, what is the sequence of states when the input minterm sequence is $w'r't$, $w'r't$, $wr't'$, $wr't'$, $wr't'$, $w'r't'$, $w'r't$, $w'r't'$, . . . ?

Answer: $s_2s_0s_0s_2s_2s_2s_2s_0s_0$ . . . ■

7.5 Synthesis: From ASM Chart to Circuit

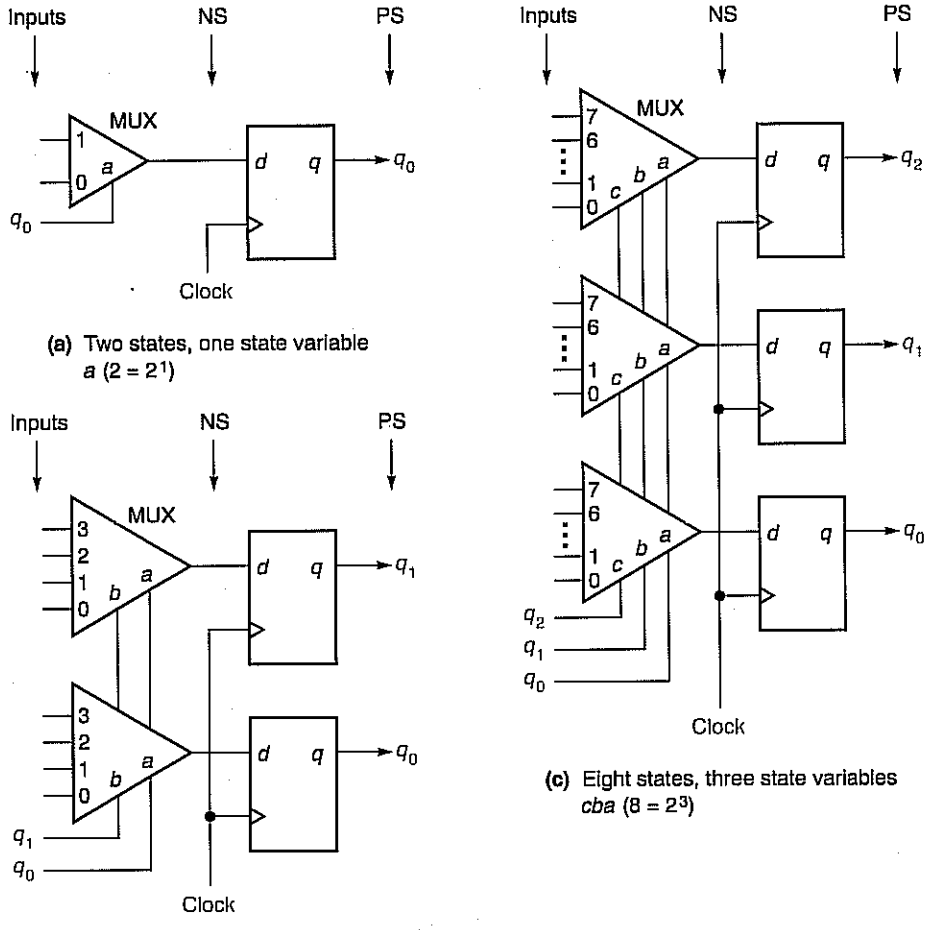
We now return to our main topic of ASM-based design to discuss the synthesis of the state machine. Recall that synthesis begins with selection of the algorithm being implemented, continues with design of the data path, and proceeds to the design of the state machine controlling the data path. Once the state machine design step is reached, the designer will have developed an intimate understanding of the data path and of the signals needed to implement the algorithm. We have included at the end of Chapter 8 (Section 8.4) an example of this process to guide the neophyte designer. Here we assume the data path is defined or that the ASM method is being used to design a state machine that does not control a data path, such as a counter.

The logic circuit synthesis method to be presented shortly is direct in the sense that we provide what is called the standard solution (Figure 7.17). Our intuition tells us that this solution is valid for any number of states. To us the multiplexer solution is understood in an instant. Nevertheless, sixty-four-to-one multiplexers, for example, are not necessarily a practical solution for a state machine with 64 states. There are a number of approaches to solutions, such as attempts at state reduction. Our preferred solution for large state machines is to have a read-only memory (ROM) (see Chapter 9) replace the multiplexers, as discussed in Section 10.5.3, and to use the ROM in combination with linked state machines, as discussed in Section 7.8.2. These alternate solutions of complex problems are straightforward applications of basic concepts we now present.

The multiplexer solution clearly shows the next state for any present state.

For large state machines ROM data words can represent the next state and present outputs.

FIGURE 7.17
Standard ASM circuit



(a) Two states, one state variable *a* ($2 = 2^1$)

(b) Four states, two state variables *ba* ($4 = 2^2$)

(c) Eight states, three state variables *cba* ($8 = 2^3$)

Logic circuit design requires equations. In turn, equations create a need for a truth table. So our method starts by extracting a truth table from the ASM chart. Logic equations are derived from the truth table after adding D flip-flop excitation columns to the truth table. Then the equations are recast into multiplexer equation format and directly applied to the standard multiplexer circuit solution. There is no one-and-only right answer to state machine design because there are a multitude of possible circuit formats. Nevertheless, we offer this one very practical method that always results in one very practical answer. The importance of the method lies in the ideas behind it. To illustrate the method, let us consider a specific example.

We design only the controller of the memory module in Figure 7.18. The memory module ASM chart is shown in Figure 7.15. Any memory module consists of a controller (sequential state machine and output circuits) and a data path, as shown in Figure 7.18. The data path is controlled by the state machine. (Data path design is covered in later chapters).

7.5.1 State Machine Synthesis

STEP 1 ***Create the ASM chart.*** The ASM chart (Figure 7.15) is repeated here for convenience.

STEP 2 ***Extract the state machine truth table from the ASM chart*** (Figure 7.15). The design of a state machine centers on calculating the next-

FIGURE 7.18
Memory module block diagram

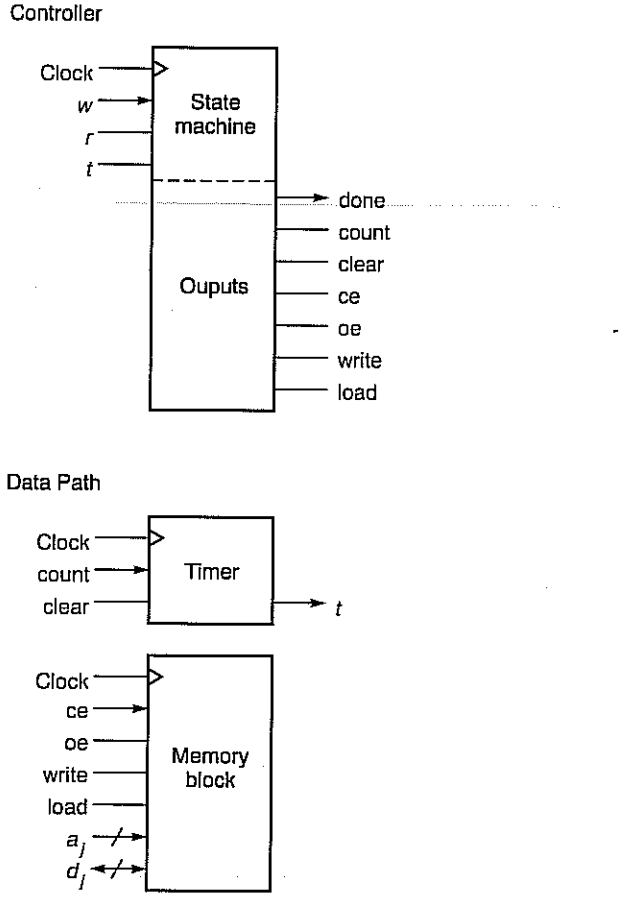
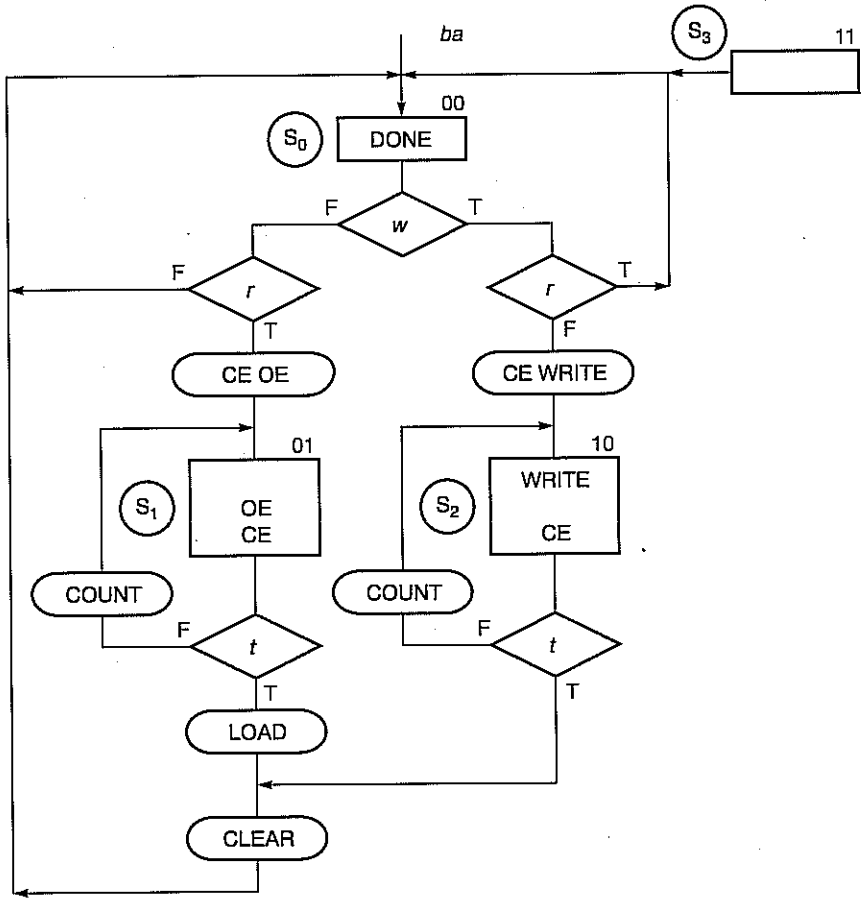


FIGURE 7.15
Memory module ASM chart (repeated)



state number from the present-state number and the present-state input variables. The chart has four states and three input variables, w, r, and t. Therefore there are four present states, which we list in the Present State column as s_0, s_1, s_2, and s_3 (see Table 7.1). Three columns (w, r, and t) are needed for the three input variables; one column is needed for the next-state number. Here is how to analyze the ASM chart to fill the columns.

State s_0 Variables w, r are in the s_0 ASM block. Two variables have four combinations of values, resulting in four truth table rows, as shown in Table 7.1. Variable t is not associated with s_0, so we mark the t column with four dashes, signifying "don't care."

The ASM chart tells us the next state for any path. In Figure 7.10, starting from the s_0 rectangle the $w'r'$ path returns to s_0, the $w'r$ path

goes to s_1, the wr' path goes to s_2, and the wr path returns to s_0. Figures 7.19a and 7.19b illustrate two of these paths.

State s_1 Variable t is in the s_1 ASM block. One variable has two combinations of values, resulting in two truth table rows, as shown in Table 7.1. Variables w and r are not associated with s_1, so we mark the w and r columns with four dashes. Starting from the s_1 rectangle, the t' path returns to s_1 and the t path goes to s_0. Figures 7.19c and 7.19d illustrate these paths.

State s_2 Variable t is in the s_2 ASM block. One variable has two combinations of values, resulting in two truth table rows, as shown in Table 7.1. Variables w and r are not associated with s_2, so we mark the w and r columns with four dashes. Starting from the s_2 rectangle, the t' path returns to s_2 and the t path goes to s_0.

State s_3 There are no variables assigned to this ASM block. Starting from the s_3 rectangle the path goes to s_0.

STEP 3 ***Add D flip-flop excitation columns*** (one column for each flip-flop). Two D flip-flops constitute the state register. They hold the present-state number. The number at the d inputs will be stored in the D flip-flops at the next clock edge. When s_0 is to be the next state, the number at the d inputs should be 00, which in voltage format is LL

Flip-flop defining equations bridge truth tables to circuits.

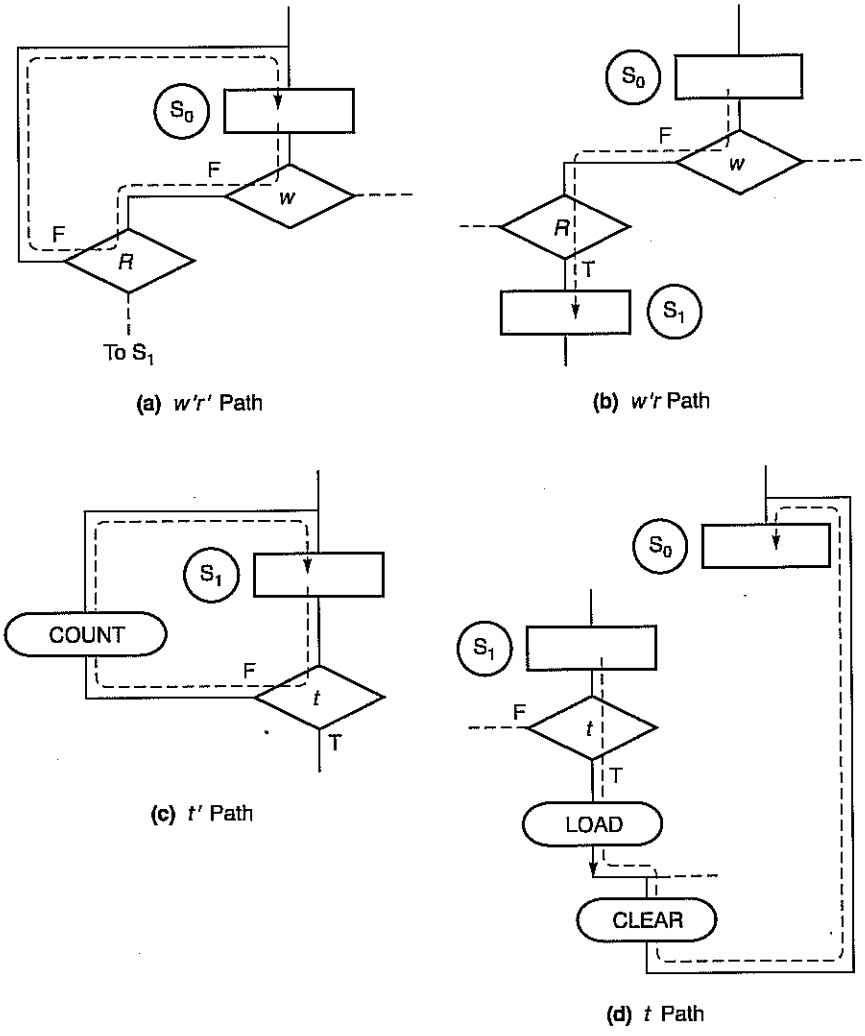
Truth table rows represent paths in ASM charts.

TABLE 7.1 Memory Module Controller's Truth Table

PS[1]	PI[2] w	r	t	NS[3]
s_0	F	F	–	s_0
	F	T	–	s_1
	T	F	–	s_2
	T	T	–	s_0
s_1	–	–	F	s_1
	–	–	T	s_0
s_2	–	–	F	s_2
	–	–	T	s_0
s_3	–	–	–	s_0

[1] Present state.
[2] Present inputs.
[3] Next state.

FIGURE 7.19
ASM chart paths to next state



because we use the active high assignment T := H. The simplicity of the D flip-flop next-state equation $q^+ = d$ allows us to enter numbers in the d columns that are equal to the next-state number. We enter the binary number corresponding to the next state (Table 7.2); we enter 00 for s_0, 01 for s_1, and so forth to define the D flip-flop functions d_1 and d_0.

STEP 4 ***Derive logic equations.*** We write D flip-flop excitation input equations using information from the truth table in Table 7.2. To make a state

TABLE 7.2 Memory Module Controller's Truth Table (continued)

PS	PI			NS		
	w	r	t	s_j	d_1	d_0
s_0	F	F	–	s_0	0	0
	F	T	–	s_1	0	1
	T	F	–	s_2	1	0
	T	T	–	s_0	0	0
s_1	–	–	F	s_1	0	1
	–	–	T	s_0	0	0
s_2	–	–	F	s_2	1	0
	–	–	T	s_0	0	0
s_3	–	–	–	s_0	0	0

assignment, we replace the state identifiers s_j with minterms m_j of variables $q_1 q_0$.

$$d_1 = s_0 wr' + s_2 t' = m_0 wr' + m_2 t'$$

$$d_0 = s_0 w'r + s_1 t' = m_0 w'r + m_1 t'$$

The next-state number delivered to the D flip-flop inputs is $d_1 d_0$. The d_1 and d_0 equations are implemented by a combinational-logic network. The state variables q_1, q_0 and the input variables w, r, t are inputs to the combinational-logic network that we call the *next-state-number generator*.

Anticipating the use of multiplexers, we cast the equations into the form of multiplexer equations. First we recognize that all multiplexer inputs must be connected to some source. We therefore include all missing states by adding zeros in the form $s_j \times 0$. Adding zeros does not invalidate the equations. Next we replace state numbers with minterms. This is permissible because our state assignments make minterms equal the states ($m_j = s_j$).

$$d_1 = s_0 wr' + s_2 t'$$

$$d_1 = s_0 wr' + s_1 \times 0 + s_2 t' + s_3 \times 0$$

$$d_1 = wr' m_0 + 0 \times m_1 + t' m_2 + 0 \times m_3$$

$$d_0 = s_0 w'r + s_1 t'$$

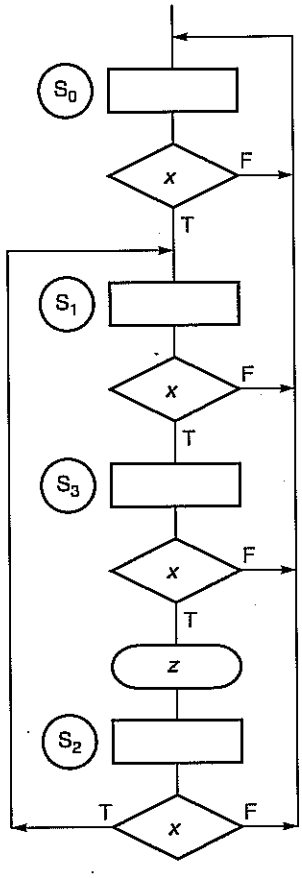
$$d_0 = s_0 w'r + s_1 t' + s_2 \times 0 + s_3 \times 0$$

$$d_0 = w'r m_0 + t' m_1 + 0 \times m_2 + 0 \times m_3$$

STEP 5 ***Use the standard state machine multiplexer circuit to implement the logic equations.*** The minterm coefficients are the multiplexer inputs. This means we can write down the minterm coefficients as inputs to the multiplexers in the standard circuit with two flip-flops (see Figure 7.17b). The result is the circuit in Figure 7.20, which implements the memory module ASM chart state machine. This completes the state machine design. Next we design the output circuits.

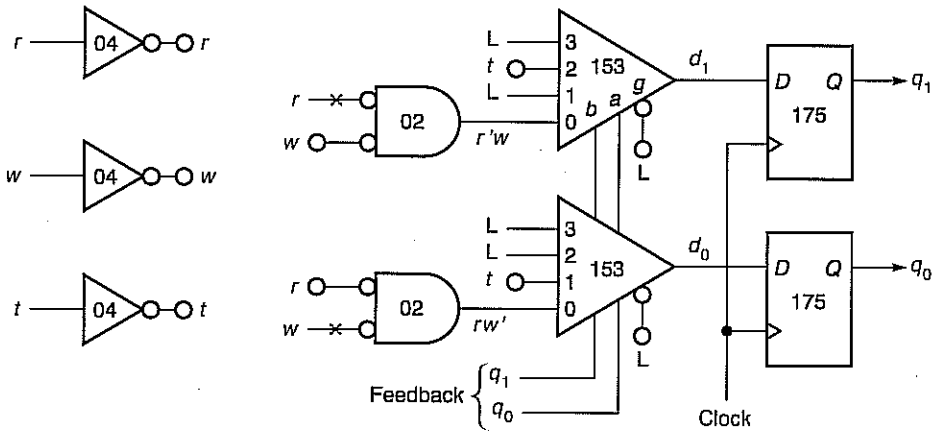
EXERCISE 7.13 Find the circuit for the ASM chart.

Answer: Figure 7.2 on page 339.



■

FIGURE 7.20
Memory module state machine



7.5.2 Output Circuit Synthesis

The memory module state machine we have designed reports the present state PS. Unconditional outputs depend only on the PS, whereas conditional outputs depend on the PS *and* the PI (present inputs). We start with a copy of the memory module truth table PS and PI columns from Table 7.2.

STEP 1 ***Add Output Columns to the Truth Table.*** For each unconditional and conditional output we add one column to the truth table (Table 7.3). There is no need to note the type of output.

Unconditional outputs are present outputs asserted in a present state.

Unconditional outputs are found inside ASM chart state rectangles. An output listed in state s_j is asserted when the ASM is in s_j. Since the assertion is not dependent on the input variables, for each output listed in state s_j we enter a T on each row corresponding to state s_j. For example, DONE is active when the ASM is in state s_0. There are four s_0 rows in the truth table because two input variables are active in s_0. We make four T entries in the DONE column, one per s_0 row. We repeat this process for all other unconditional outputs in all states.

Conditional outputs are asserted in a present state if the associated input minterm is true.

Conditional outputs are found inside ovals. Ovals are part of each ASM block's binary tree. An output listed in state s_j is asserted when the ASM is in s_j *and* the path to the oval is active. Since the assertion is dependent on the input variables, for each output listed in the state s_j binary tree we enter a T on each row corresponding to the active path in state s_j. For example, the ASM chart specifies that CLEAR is active when the ASM is in state s_1 *and* input t is True. We make one T entry in the CLEAR column in the s_1 row where t = T. We repeat this

process for all other conditional outputs in all states. Table 7.3 is constructed by this method. Observe that an output, such as OE, is conditional in s_0 and unconditional in s_1.

STEP 2 ***Derive Output Logic Equations.*** In the usual manner we obtain equations for the outputs from the truth table. There is one equation per column and one term per T entry. Notice how we write down the equations. The format anticipates using multiplexers as well as minimizing the potential for errors. This time we do not add states with zero coefficients to the equations, because we now know the missing states in effect specify L sources to corresponding multiplexer inputs.

$$\begin{aligned}
\text{DONE} &= s_0 &&= m_0\\
\text{CE} &= s_0(rw' + r'w) + s_1 + s_2 &&= m_0(rw' + r'w) + m_1 + m_2\\
\text{OE} &= s_0(rw') + s_1 &&= m_0(rw') + m_1\\
\text{WRITE} &= s_0(r'w) + s_2 &&= m_0(r'w) + m_2\\
\text{COUNT} &= s_1t' + s_2t' &&= m_1t' + m_2t'\\
\text{CLEAR} &= s_1t + s_2t &&= m_1t + m_2t\\
\text{LOAD} &= s_1t &&= m_1t
\end{aligned}$$

STEP 5 ***Use the standard output multiplexer circuit to implement the logic equations*** (Figure 7.21). The minterm coefficients are the multiplexer

TABLE 7.3 Truth Table for Memory Module Outputs

PS	PI			PO						
s_j	w	r	t	DONE	CE	OE	WRITE	COUNT	CLEAR	LOAD
s_0	F	F	–	T	.	.	.	.	.	.
	F	T	–	T	T	T	.	.	.	.
	T	F	–	T	T	.	T	.	.	.
	T	T	–	T	.	.	.	.	.	.
s_1	–	–	F	.	T	T	.	T	.	.
	–	–	T	.	T	T	.	.	T	T
s_2	–	–	F	.	T	.	T	T	.	.
	–	–	T	.	T	.	T	.	T	.
s_3	–	–	–	.	.	.	.	.	.	.

FIGURE 7.21
Memory module standard output circuit

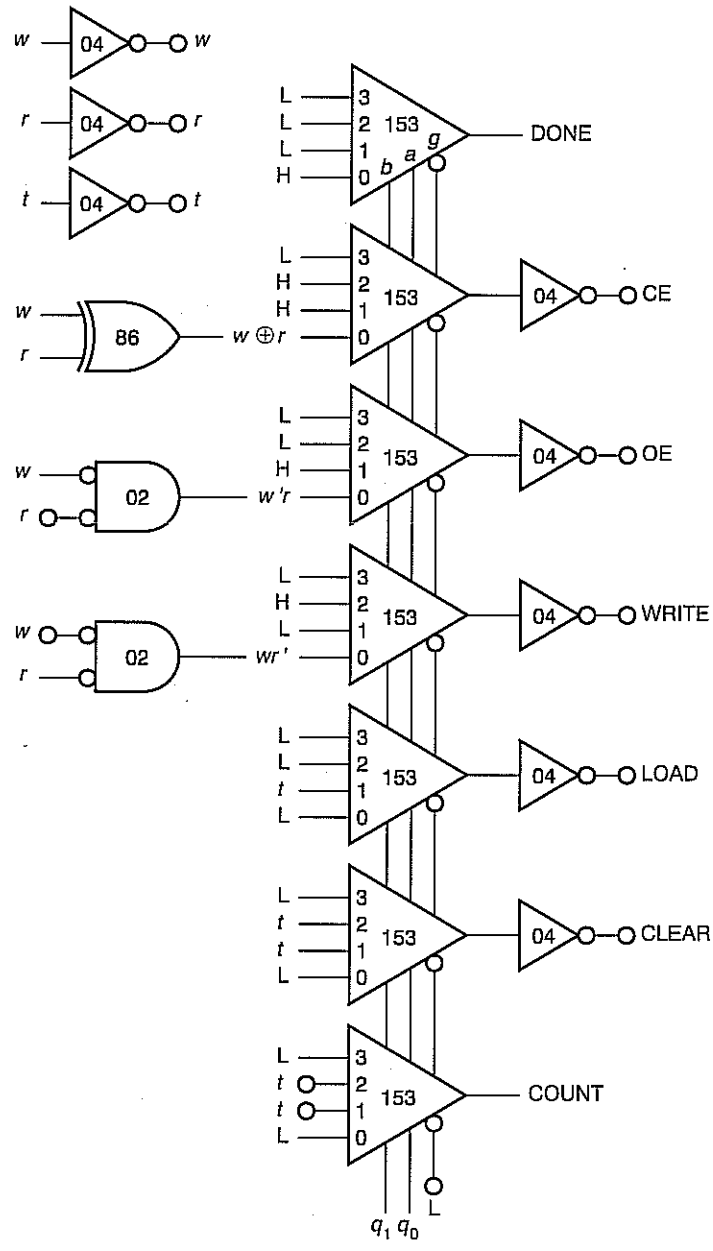
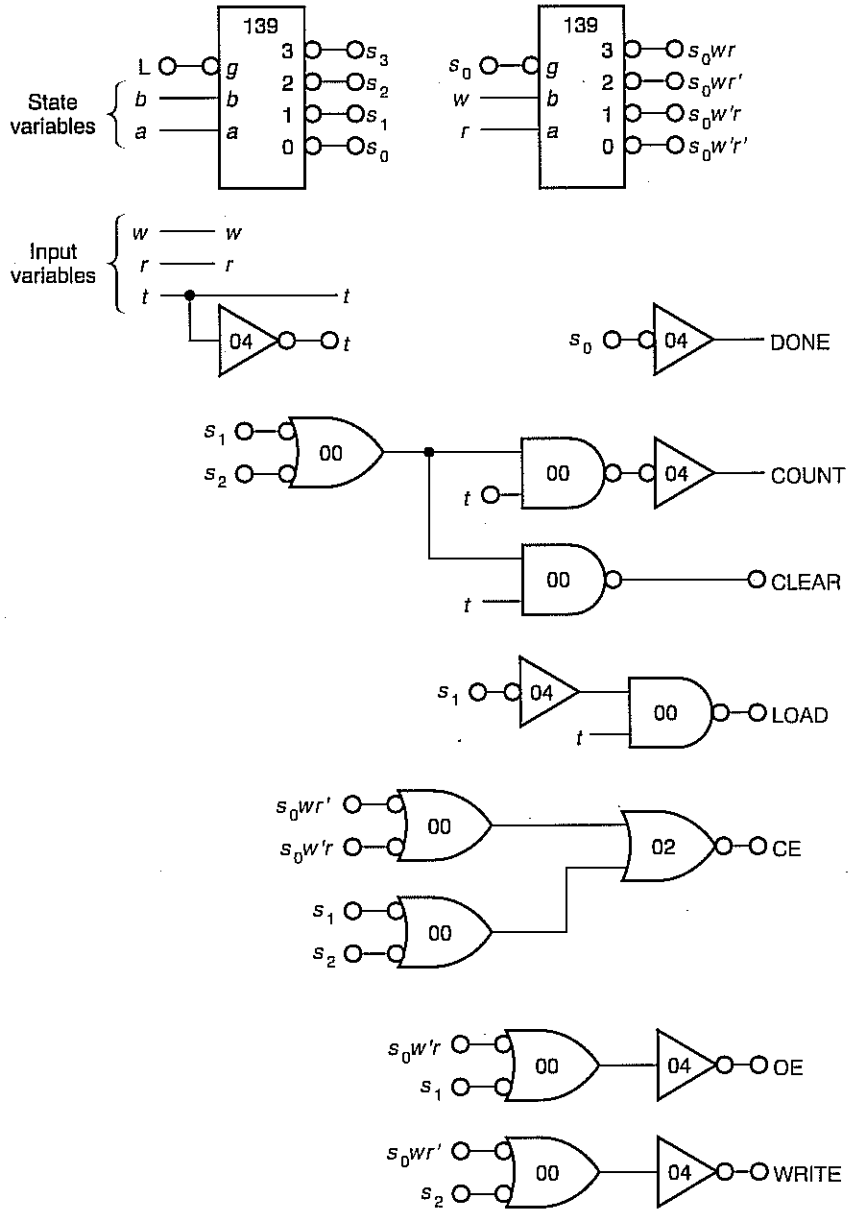


FIGURE 7.22
Memory module traditional output circuits



inputs. This means we can write down the minterm coefficients as inputs to the multiplexers in the standard output circuit. The result is the circuit in Figure 7.21, which implements the memory module ASM chart outputs. Compare this circuit to a more traditional circuit (Figure 7.22). In practice we do not know how to

predict which circuit form uses fewer gates and combinational building blocks.

7.6 Analysis: From Circuit to ASM Chart

If we did not know the ASM chart for the circuits in Figure 7.21 and Figure 7.23, the analysis questions would be as follows. Given the circuits, what is the ASM chart (that is, what is the next state for each present state)? Second, what are the active outputs in each present state? The analysis starts by extracting flip-flop input equations from the state machine circuit for the purpose of using them to create the present-state/next-state truth table. Then ASM blocks are drawn from information in the truth table. The assembly of ASM blocks results in the ASM chart. The second part of the analysis starts by extracting output equations from the output circuits. The equations determine where outputs are inserted into the ASM chart.

STEP 1 ***Derive the d_1, d_0 flip-flop input equations.*** The multiplexer outputs are the flip-flop d_1, d_0 inputs (Figure 7.23). Minterm notation simplifies the equations and provides a one-to-one correspondence to state numbers s_j (i.e., the state assignment). The zero coefficients correspond to the L inputs.

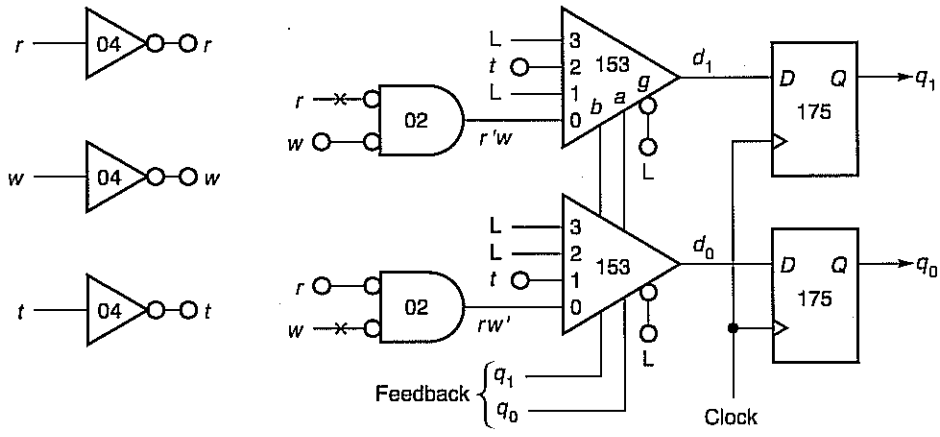
$$\begin{aligned} d_1 &= wr'm_0 + 0 \times m_1 + t'm_2 + 0 \times m_3 \\ &= wr's_0 + 0 \times s_1 + t's_2 + 0 \times s_3 \\ d_0 &= w'rm_0 + t'm_1 + 0 \times m_2 + 0 \times m_3 \\ &= w'rs_0 + t's_1 + 0 \times s_2 + 0 \times s_3 \end{aligned}$$

STEP 2 ***Extract a truth table from the d_j equations.*** The given state machine circuit (Figure 7.23) has a D flip-flop state register. This two-bit register represents two variables, q_1 and q_0. The variables q_1 and q_0 represent four possible states s_j numbered, in binary, 00, 01, 10, and 11.

We build a truth table in Table 7.4 by assigning q_1q_0 the four state numbers 0, 1, 2, and 3, in sequence, in the Present State PS column. Then we add columns for d_1 and d_0 and enter the minterm coefficients. Next we add PI columns, one for each input variable used in the coefficients. Adding rows as needed, we fill in the PI columns with all input variable 0, 1 combinations for variables active in a state. Then we add "don't care" dashes for (inactive) variables not associated with a state.

Minterms' coefficients are next-state conditions.

FIGURE 7.23
State machine



We calculate the next-state numbers NS by evaluating the terms in the d_1, d_0 columns using the input variable values in each row. Then we use the assignments $m_j = s_j$ to convert next-state numbers to state identifiers s_j. *Note:* each row of the truth table represents a path in the ASM chart that ends at the next state.

STEP 3 ***Draw an ASM block for each state.***

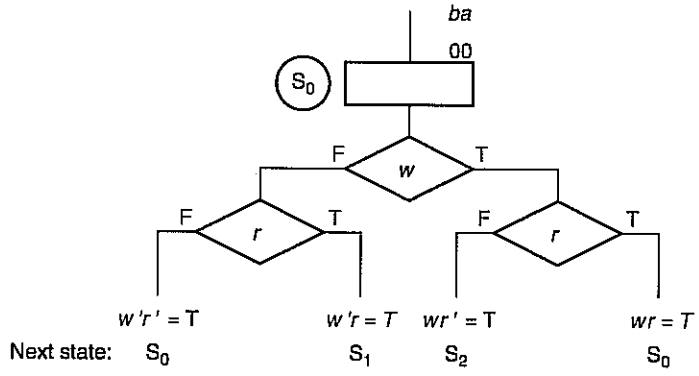
State 0 The two variables w and r are active in state zero. This means the two input variables w, r are in the state-zero (s_0) ASM block. Input variables w and r form a binary tree of branch diamonds at the state-

Truth tables show the input minterms associated with a present state.

TABLE 7.4 Next-State Truth Table

PS			PI			NS		
s_j	d_1	d_0	w	r	t	q_1^+	q_0^+	NS
s_0	wr'	$w'r$	0	0	–	0	0	s_0
			0	1	–	0	1	s_1
			1	0	–	1	0	s_2
			1	1	–	0	0	s_0
s_1	0	t'	–	–	0	0	1	s_1
			–	–	1	0	0	s_0
s_2	t'	0	–	–	0	1	0	s_2
			–	–	1	0	0	s_0
s_3	0	0	–	–	–	0	0	s_0

FIGURE 7.24
ASM block for s_0



zero output line. The w, r binary tree has four exits to next states (Figure 7.24).

State 1 One variable t is active in state one. Therefore only input variable t is in the state-one (s_1) ASM block. The t binary tree has two exits to next states (Figure 7.25).

State 2 One variable t is active in state two. Therefore only input variable t is in the state-two (s_2) ASM block. The t binary tree has two exits to next states (Figure 7.26).

State 3 No variables are active in state three. Therefore the state rectangle constitutes the state-three (s_3) ASM block (Figure 7.27).

STEP 4 ***Form the ASM chart from the ASM blocks.*** We interconnect the blocks in Figures 7.24 to 7.27 so that the states shown at the end of each exit path are the next states listed in the truth table. The result is the ASM chart without outputs in Figure 7.28.

FIGURE 7.25
ASM block for s_1

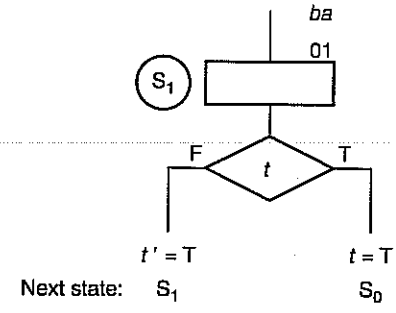


FIGURE 7.26
ASM block for s_2

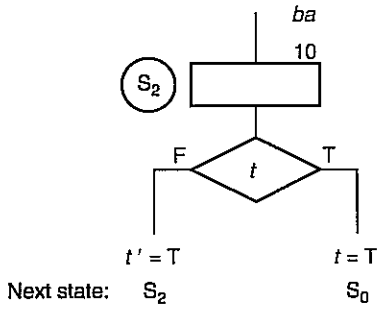


FIGURE 7.27
ASM block for s_3

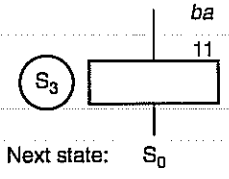
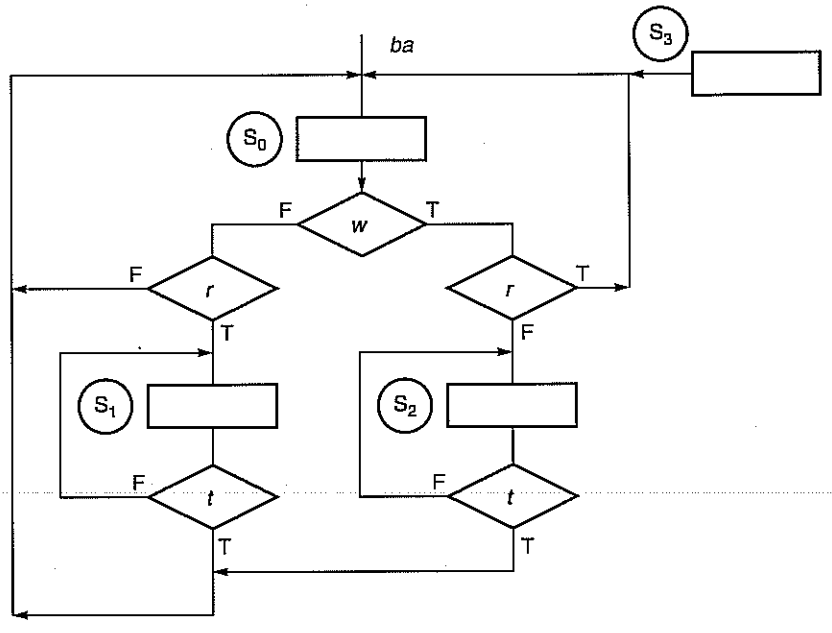


FIGURE 7.28
ASM chart with no outputs

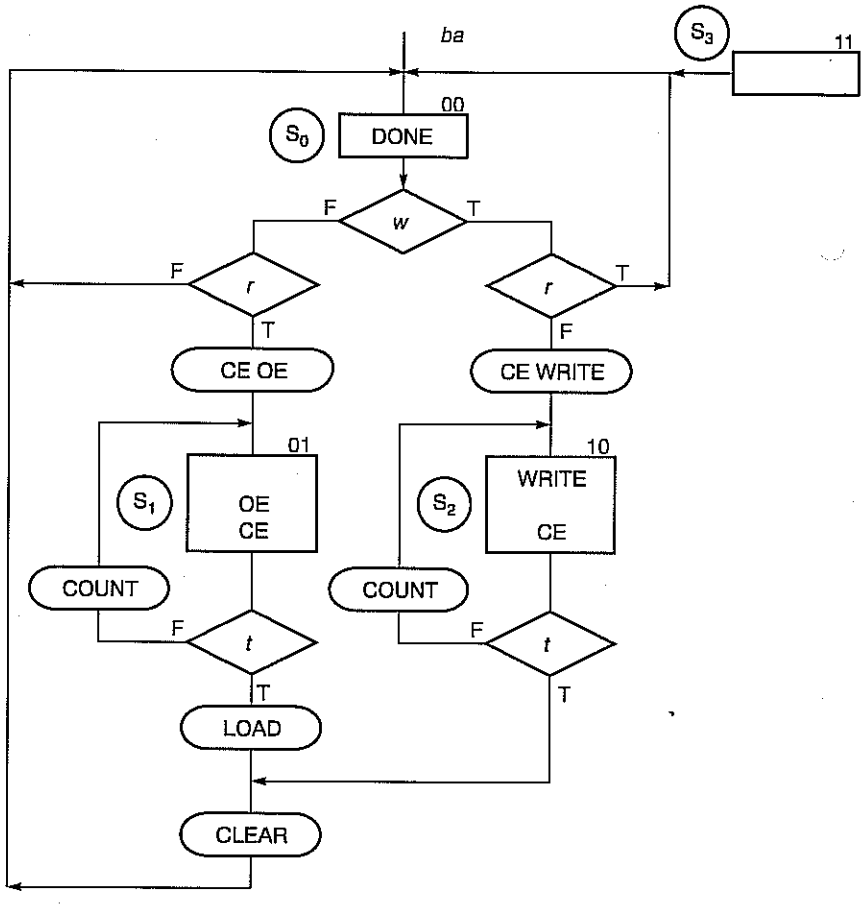


STEP 5 ***Extract equations from the output circuits.*** Equations are derived from the output circuits (see Figure 7.21 or Figure 7.22) in the usual manner.

$$\begin{aligned}
\text{DONE} &= s_0 \\
\text{COUNT} &= t's_1 + t's_2 \\
\text{CLEAR} &= ts_1 + ts_2 \\
\text{LOAD} &= ts_1 \\
\text{CE} &= (w'r + wr')s_0 + s_1 + s_2 \\
\text{OE} &= (w'r)s_0 + s_1 \\
\text{WRITE} &= (wr')s_0 + s_2
\end{aligned}$$

An output equation term with 1 as the state coefficient is an unconditional output.

FIGURE 7.29
ASM chart with outputs



An output equation term with an expression as the state coefficient is a conditional output.

Equation terms with a coefficient of one are unconditional. Unconditional outputs are entered into state rectangles (Figure 7.29) according to the equations. Equation terms with variables as coefficients are conditional. Conditional outputs are entered into ovals placed at the end of appropriate binary tree paths (Figure 7.29) defined by the equations.

7.7 Asynchronous and Synchronous Inputs

The timing diagrams in Figure 7.30 show the effect on outputs AF and AT when input variable X is synchronous or asynchronous.

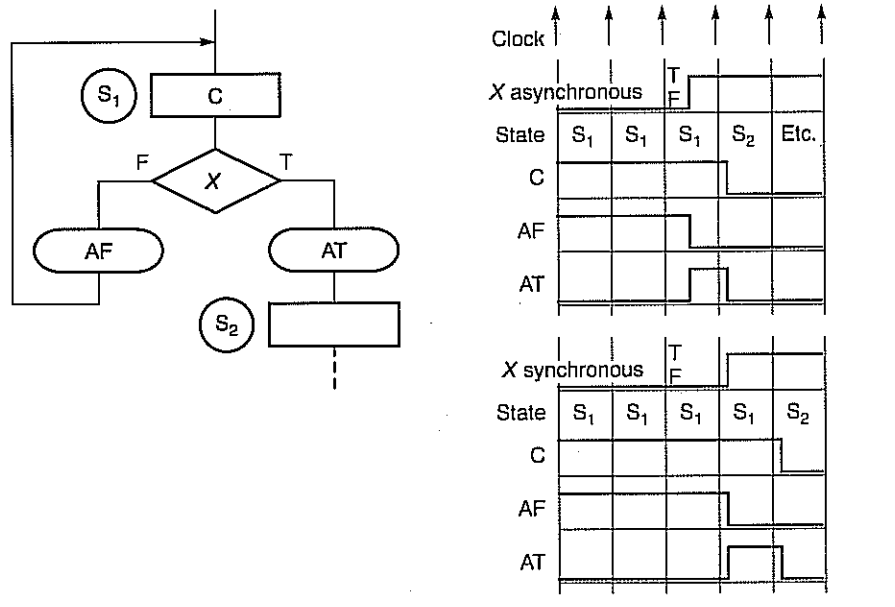
Asynchronous X Logical errors may occur if X changes state during a required setup or hold time. Also, the AT waveform may end up being too narrow (causing its own errors) when X goes high, as shown. The AF waveform goes low during the clock period. AF being low may also be "too narrow" before the next clock edge, in the sense of violating a setup time requirement.

Synchronous X Now the AT waveform duration is always high a full clock period and AF goes low at the beginning of the clock period.

> Synchronize asynchronous sources before connecting them to synchronous machine inputs

A suggested synchronizer circuit is discussed in Section 6.4.4.

FIGURE 7.30
Synchronous and asynchronous inputs



Let us consider the ASM fragment and timing diagram in Figure 7.30 when X is synchronous. If we assume the logic machine is in state s_1, then:

Unconditional output C is active
and
Conditional output AF is active if input variable X is False. The state machine moves to state s_1 on the next clock edge. In this case the next state is the same state. The system dwells in present state s_1.

or

Unconditional output C is active
and
Conditional output AT is active if input variable X is True. The state machine moves to state s_2 on the next clock edge.

The next state is loaded at each active clock edge. When X is False, the next state is s_1, which may give the impression that nothing happens. However, something always happens; the active clock edge moves the machine from s_1 to s_1 or from s_1 to s_2.

7.8 Practical ASM Topics

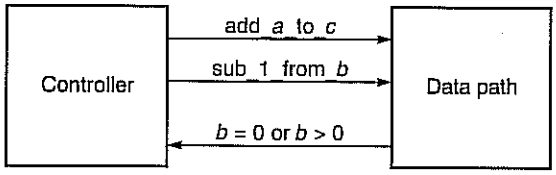
The ASM executes all events in any ASM block at the same time; the ASM executes events in parallel. This advantage leads to complications with loops, which are discussed first. Then the concept of the linked ASM is introduced.

7.8.1 Control Loops

An ASM dwelling in one state is one form of control loop (similar to a programming language loop). This simple form also illustrates one trap a designer can fall into. The problem arises from the fact that equations such as $c = c + a$ are executed by the clock edge at the *end* of clock periods. The possible negative consequences are not necessarily clear until a timing diagram is drawn. Once understood, the negative consequences are avoided by judicious choice of the initial how-many-times-to-loop value or the type of loop.

The ASM outputs "add_a_to_c" and "sub_1_from_b" are inputs to the data path controlled by the state machine (Figure 7.31). The data path that executes the ASM outputs and evaluates $c = c + a$ and

FIGURE 7.31
ASM controller and data path



$b = b - 1$ uses the same clock as the ASM state machine. Therefore the controller and the controlled are synchronous. The actions corresponding to ASM outputs "add_a_to_c" and "sub_1_from_b" are data path actions in the *next state* resulting from the present-state inputs to the data path.

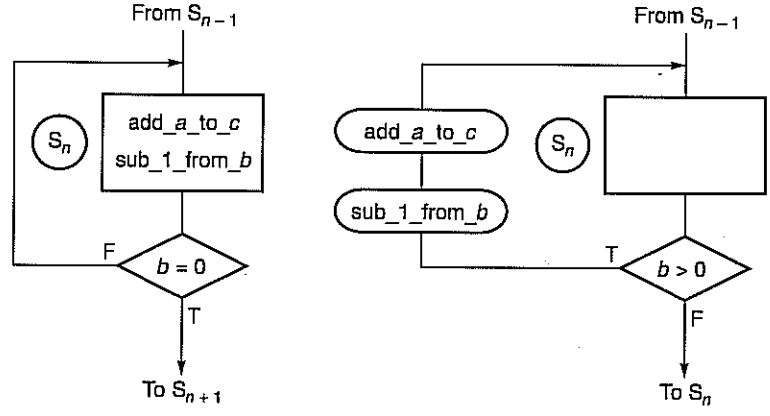
The begin-until loop uses unconditional outputs. The begin-until loop in Figure 7.32a always spends one or more clock cycle(s) in its state, because the exit condition is evaluated at the end of a loop cycle. Unlike the begin-until loop, the while-do-repeat loop uses conditional outputs. The while-do-repeat loop in Figure 7.32b also spends one or more clock cycles in its state, because the exit condition is still evaluated at the end of a loop cycle.

The sequence of events for the begin-until loop when $b = 0$ is as follows (Figure 7.33). At the end of clock cycle$_2$ the state machine moves from state s_{n-1} to s_n to enter the loop. Variable values do not change during cycle$_3$ because the equations are not executed until the end of cycle$_3$.

Unconditional output "equations" are executed by the active clock edge at the end of a state time.

At the end of cycle$_3$, b is decremented and a is added to c. Since $b = 0$ during cycle$_3$, the state machine moves from s_n to s_{n+1} in cycle$_4$. In begin-until loop ASM fragments, unconditional equations execute

FIGURE 7.32
Loops in ASM charts



(a) Begin-Until Loop ASM (b) While-Do-Repeat Loop ASM

FIGURE 7.33
Begin-until loop timing ($b = 0$)

Clock cycle		↑ 1	↑ 2	↑ 3*	↑ 4	↑ 5	↑ . . .
State		s_{n-2}	s_{n-1}	s_n	s_{n+1}	. . .	
Loop cycle		—	—	1	—	—	
Values:	a	5	5	5	5	5	. . .
	b	0	0	0	−1	−1	. . .
	c	0	0	0	5	5	. . .

* loop entered

$b + 1$ times. The case when $b = 3$ is illustrated in Figure 7.34. This is typical, and comparable to what can happen in a computer program.

The value of b determines the next state in Figure 7.34. At the end of cycle_3, cycle_4, and cycle_5 the next-state decision is to go to s_n, because b is greater than zero. At the end of cycle_6 the next-state decision is to go to s_{n+1}, because $b = 0$. The machine dwells in s_n for four cycles. This is why $c = 4a = 20$ in cycle_7.

The sequence of events for the while-do-repeat loop when $b = 0$ is as follows (Figure 7.35). At the end of clock cycle_2 the state machine moves from state s_{n-1} to s_n to enter the loop. Variable values do not change during cycle_3, because the equations are not executed until the end of cycle_3. At the end of cycle_3 the equations are not executed, because $b = 0$. And since $b = 0$ during cycle_3, the state machine moves from s_n to s_{n+1} in cycle_4.

Conditional output "equations" are also executed by the active clock edge at the end of a state time.

In while-do-repeat loop ASM fragments, conditional equations execute b times. The case when $b = 3$ is illustrated in Figure 7.36.

Observe that conditional outputs may be added to begin-until loops, and unconditional outputs may be added to while-do-repeat loops.

FIGURE 7.34
Begin-until loop timing ($b = 3$)

Clock cycle		↑ 1	↑ 2	↑ 3*	↑ 4	↑ 5	↑ 6	↑ 7	↑ 8	↑ 9	↑ 10 ↑
State		s_{n-2}	s_{n-1}	s_n	s_n	s_n	s_n	s_{n+1}	. . .		
Loop cycle		—	—	1	2	3	4				
Values:	a	5	5	5	5	5	5	5	. . .		
	b	3	3	3	2	1	0	−1	. . .		
	c	0	0	0	5	10	15	20	. . .		

* loop entered

FIGURE 7.35
While-do-repeat loop timing ($b = 0$)

Clock cycle		↑ 1	↑ 2	↑ 3*	↑ 4	↑ 5	↑ . . .
State		s_{n-2}	s_{n-1}	s_n	s_{n+1}	. . .	
Loop cycle		—	—	1	—	—	
Values:	a	5	5	5	5	5	. . .
	b	0	0	0	0	0	. . .
	c	0	0	0	0	0	. . .

* loop entered

7.8.2 Linked ASMs

Linked ASMs reduce the number of states required to implement the system, thereby simplifying the hardware circuit. Any digital function that is used repeatedly by an algorithm may be implemented as a stand-alone ASM linked to the mainline ASM implementing the algorithm. Linking allows repeated use of a function's circuit. This repeated use represents states not added to the mainline ASM. (Readers with computer programming experience will recognize that the linked ASM is analogous to the subroutine.) Links are established when an output from ASM_j is an input to ASM_k, and vice versa. A mainline ASM linked to ASMs that are in turn linked to other ASMs and so on is an example of top-down design at its best. Perhaps several examples can make the point.

A link is one ASM's output connected to another ASM's input.

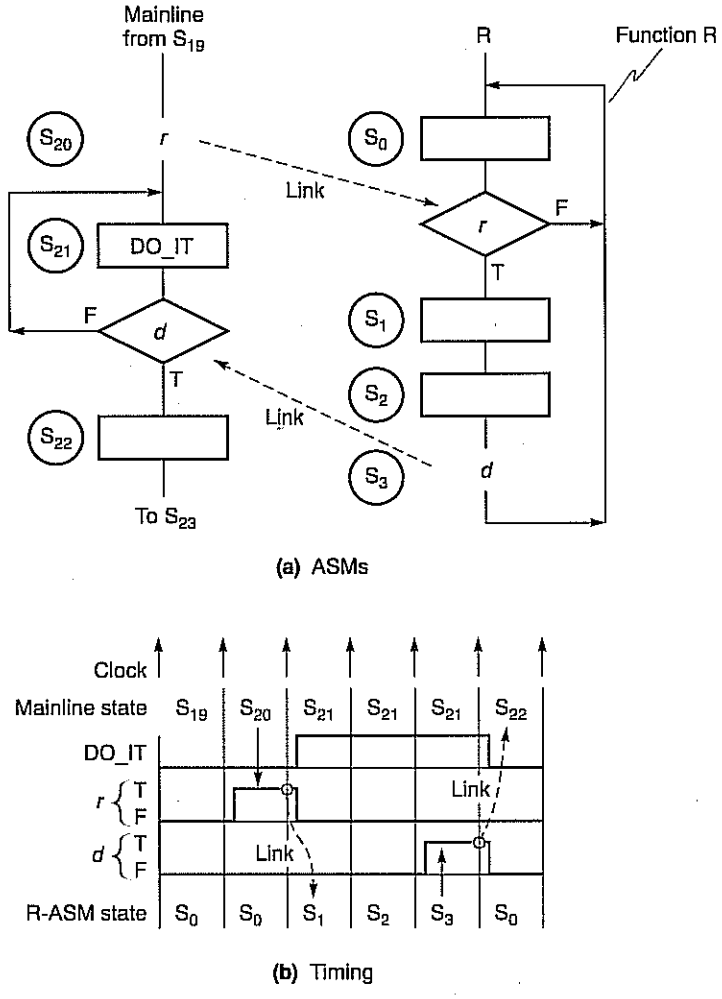
Let us suppose that the fragment of the mainline ASM in Figure 7.37 is linked to an ASM implementing function R, where R is whatever function we please. There may be many links to the R-ASM from various mainline states. Further, let's suppose function R is at rest until called by the mainline ASM.

FIGURE 7.36
While-do-repeat loop timing ($b = 3$)

Clock cycle		↑ 1	↑ 2	↑ 3*	↑ 4	↑ 5	↑ 6	↑ 7	↑ 8	↑ 9	↑ 10 ↑
State		s_{n-2}	s_{n-1}	s_n	s_n	s_n	s_n	s_{n+1}	. . .		
Loop cycle		—	—	1	2	3	4				
Values:	a	5	5	5	5	5	5	5	. . .		
	b	3	3	3	2	1	0	0	. . .		
	c	0	0	0	5	10	15	15	. . .		

* loop entered

FIGURE 7.37
Linked ASM



Links are established by outputs r from mainline states to input r in the R-ASM (Figure 7.37). Mainline output r is input r to the R-ASM. With input r asserted the R-ASM starts and executes its task. Meanwhile, the mainline dwells in waiting state s_{21} until input d is asserted (Figure 7.37). Mainline input d is output d from the R-ASM. For example, when the mainline needs to output the DO_IT command (from mainline state s_{21} in Figure 7.37) for a specific time, R-ASM provides the time delay function. This situation is shown in Figure 7.37a, where states s_1, s_2, and s_3 define a three-clock-period delay. The wise designer draws a timing diagram to verify correct performance (Figure 7.37b).

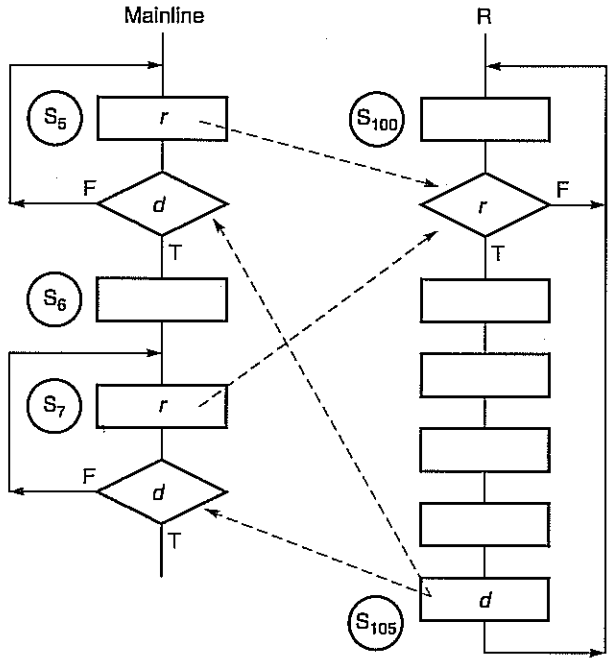


FIGURE 7.38
Linked ASM—Multiple calls

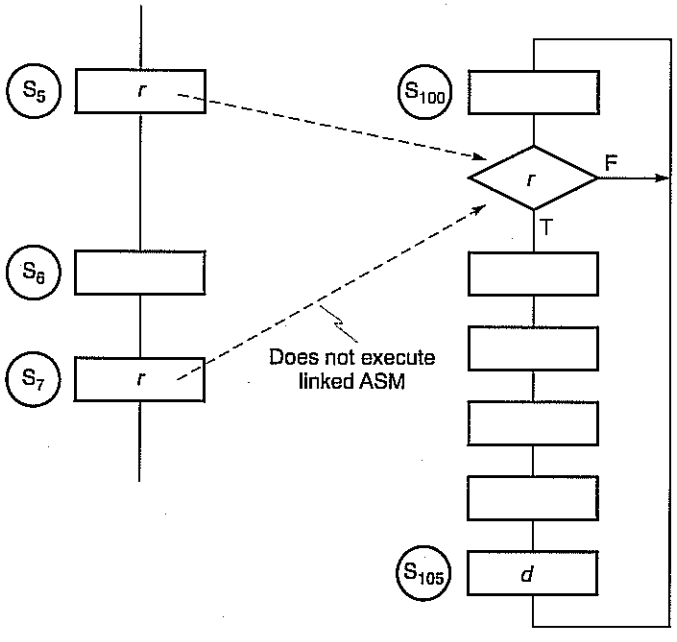


FIGURE 7.39
Linked ASM—Multiple calls—No feedback

Multiple calls are readily implemented (Figure 7.38). Observe how the waiting state, s_5 or s_7, emits output r while waiting. The R-ASM does not care about r once the R-ASM has moved from its resting state, s_{100}. Also note that output d must be in s_{105} and not in s_{100}.

Linked ASMs usually have fewer states than one ASM performing the same algorithm.

The linked ASM can execute in parallel with the mainline when so desired. The mainline calls the link ASM to start it executing. Then the mainline continues on with its task as the link ASM executes in parallel. In this case the branch back to the same state (when d is False) is not used and the d feedback link is omitted (Figure 7.39). When the d feedback link is omitted, execution of R depends on how many mainline states intervene between calls (Figure 7.39). R is not executed when the intervening number of mainline states is less than the number of executing states in R (Figure 7.39). A programmer would say subsequent calls to R are not executed unless the current call has completed. Caution is needed with this high-risk practice.

EXERCISE 7.14 Find the parallel sequences of states for the linked ASM in Figure P7.8 on page 407 (s_0 to s_0). Start from s_0 and s_{10}.

Answer:

ASM_0	s_0	s_1	s_1	s_1	s_1	s_2	s_3	s_0
ASM_1	s_{10}	s_{10}	s_{11}	s_{12}	s_{13}	s_{10}	s_{10}	s_{10}

■

EXERCISE 7.15 Find the parallel sequences of states for the linked ASM in Figure P7.9 on page 408 when R is False (s_0 to s_0). Start from s_0 and s_{10}.

Answer:

ASM_0	s_0	s_1	s_2	s_3	s_0
ASM_1	s_{10}	s_{10}	s_{10}	s_{10}	s_{10}

■

EXERCISE 7.16 Find the parallel sequences of states for the linked ASM in Figure P7.9 on page 408 when R is True (s_0 to s_0 to s_0). Start from s_0 and s_{10}.

Answer:

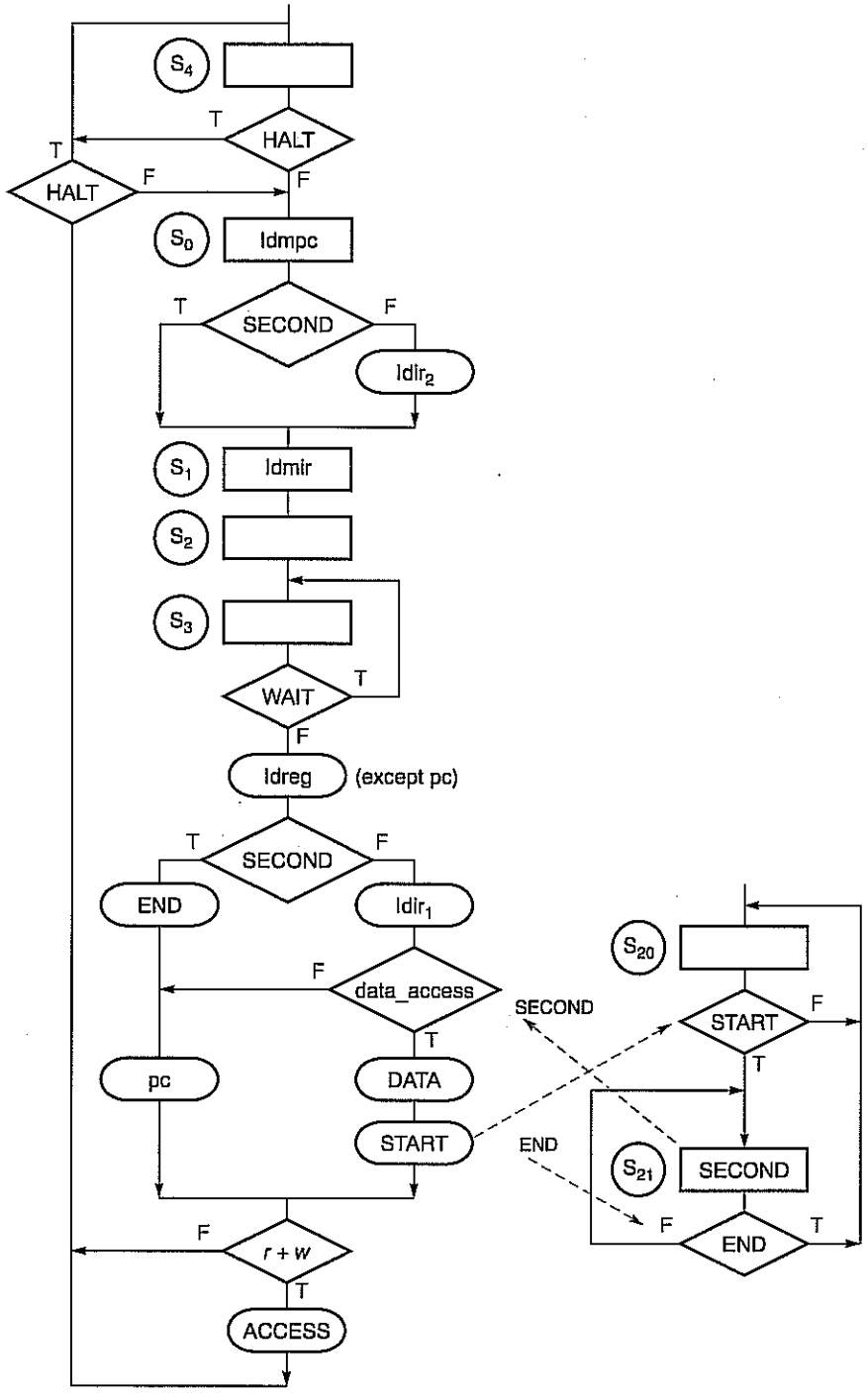
ASM_0	s_0	s_1	s_2	s_3	s_0	s_1	s_2	s_3	s_0
ASM_1	s_{10}	s_{10}	s_{10}	s_{10}	s_{11}	s_{11}	s_{11}	s_{11}	s_{10}

■

7.9 Design Example: Computer Controller

At this point we are in a position to design any state machine we please. However, the straightforward process we have just learned could burden us unduly unless we gain further insight, by studying more complex examples such as the ASM representing a simple pipelined computer controller shown in Figure 7.40. (The details of the names and the control process are left to a computer design course.)

FIGURE 7.40
ASM for a simple pipelined computer controller



State machine truth table The conventional procedure would be to make a (large and complex) state machine truth table with one column for every input variable. This is not necessary if we make two observations:

1. s_1 is the state following s_0 regardless of the value of the input Second: in s_0, Second is a "don't care." The next state is always s_1.
2. The input tree from the WAIT diamond to the $r + w$ diamond has only one entrance and one exit, and goes nowhere else. The same is true of the $r + w$ diamond.

In effect the only inputs in s_3 that influence the state machine's next-state decisions are HALT and WAIT, which is why the truth table in Table 7.5 is simple. (Try building it using every input variable.)

Insight: look for simplifying equivalent paths.

A very different consideration is the weight given to inputs by implication when they are listed in the truth table. In Table 7.5 WAIT has been given greater weight than HALT by virtue of its position, because in s_3 HALT is below WAIT in the tree. To appreciate this fine point, make a table with HALT in the first position.

Insight: select input weight carefully.

TABLE 7.5 Computer Controller Next-State Truth Table

PS	PI		NS
s_j	WAIT	HALT	s_j^+
s_0	–	–	s_1
s_1	–	–	s_2
s_2	–	–	s_3
s_3	F	F	s_0
	F	T	s_4
	T	–	s_3
s_4	–	F	s_0
	–	T	s_4

TABLE 7.6 Computer Controller Output Truth Table

Let w = WAIT, s = SECOND, d = data_access, $r = r + w$

PS	PI				OUTPUTS									
s_j	w	s	d	r	ldmpc	ldir_2	ldmir	ldreg	ldir_1	data	st	end	pc	access
s_0	–	F	–	–	T	T	.	.	.	.	.	.	.	.
	–	T	–	–	T	.	.	.	.	.	.	.	.	.
s_1	–	–	–	–	.	.	T	.	.	.	.	.	.	.
s_2	–	–	–	–	.	.	.	.	.	.	.	.	.	.
s_3	T	–	–	–	.	.	.	.	.	.	.	.	.	.
	F	F	F	–	.	.	.	T	T	.	.	.	T	.
	F	F	T	–	.	.	.	T	T	T	T	.	.	.
	F	T	–	–	.	.	.	T	.	.	.	T	T	.
	–	–	–	T	.	.	.	.	.	.	.	.	.	T
s_4	–	–	–	–	.	.	.	.	.	.	.	.	.	.

Output truth table Here, too, input weight selection is important. Tree hierarchy determines the weight. In s_3 the hierarchy is WAIT, SECOND, data_access, and $r + w$.

Insight (repeated): select input weight carefully.

Input variables in independent trees are "don't cares" in foreign trees. Variable $r + w$ is foreign to the tree starting with WAIT in the input diamond. The variables WAIT, SECOND, and data_access are foreign to the $r + w$ tree. See s_3 in Table 7.6.

Insight: foreign variables are "don't cares."

SUMMARY

State variable: A quantity capable of assuming the value 0 or 1.

State: The condition of the sequential digital circuit, as specified by a set of state variables q_j, reflecting the history of the state machine.

Branch: Associated with each state are none, one, two, or more input variables. The present state of these input variables in some combination selects the next state. The next state is determined without conditions when there are no input variables.

Outputs: States may specify unconditional or conditional outputs. Unconditional outputs assigned to a state are active when the system is in that state. Conditional outputs assigned to a state are active when the system is in that state *and* when input conditions are as specified (see *Branch* above).

Asynchronous machine: The present state is the state of the q_j at the time before inputs change. The next state is the state of the q_j after the circuit has reacted to input changes.

Synchronous machine: The present state is the state of the q_j at the time before clock edge occurs. The next state is the state of the q_j after clock edge occurs and after the circuit has reacted to the clock edge. The inputs and the present state determine which state is next; the clock edge determines when the state machine goes to the next state.

There is always a next state: The next state is loaded at each active clock edge. When the next state is the same as the present state, the impression that nothing happens may be created. However, something always happens; the active clock edge moves the machine from the present state to the next state.

Digital Design Method

Step 1. Create the ASM chart from the algorithm. Use the three symbols for state, branch, and conditional output to build ASM blocks, which are then connected together to make up the chart. Do not be surprised to find that this step takes up most of the design time for any reasonably complex digital design. You will find yourself constantly reworking the ASM chart. This process is a source of ideas and insight.

Step 2. Make a state machine truth table. Draw up a truth table relating input variables and the present state to the present (conditional and unconditional) outputs and the next state. This information is taken from the ASM chart.

Step 3. Add columns for flip-flop inputs and outputs to the truth table.

Step 4. Derive logic equations. Draw up the Boolean equations for flip-flop excitation inputs, the unconditional outputs, and the conditional outputs. Recast the equations into multiplexer format.

Step 5. Synthesize the hardware from the equations.

Our standard circuit for the state machine and output circuits uses a multiplexer format (see Figures 7.17 and 7.21).

REFERENCES

Breeding, K. J. 1989. *Digital Design Fundamentals*. Englewood Cliffs, N.J.: Prentice-Hall.

Clare, C. R. 1971. *Designing Logic Systems Using State Machines*. New York: McGraw-Hill.

Roth, C. H. 1992. *Fundamentals of Logic Design*, 4th. ed. St. Paul, Minn.: West.

PROJECT 09 Demonstration Report

Name _____ SID# _____ Section _____
(last) *(initials)*

Approved by _____ Date _____

Grade on report _____

System Clock Design

1. Design the single-pulse/periodic clock circuit.

_____ truth table from ASM chart with D flip-flops
_____ equations for d_j inputs to flip-flops and outputs
_____ state machine circuit
_____ output circuits
_____ debounced switch input circuits

2. Build the circuit.

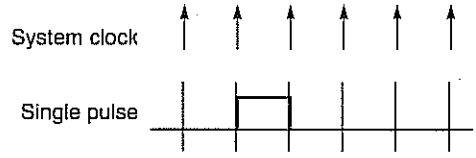
_____ circuit documents

3. Use the truth table signal generator to test the circuit.

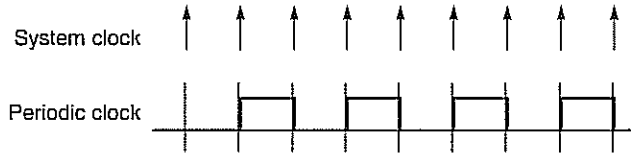
_____ input/output timing diagram

PROJECT 09 System Clock Design

Background: A single pulse circuit emits a signal that is true for one complete clock period for each press of a push button. The signal is synchronous with the clock edges.



On the other hand, a periodic clock with 50% duty cycle emits a sequence of complete clock periods, where each is two-system-clock-periods in duration. By 50% duty cycle we mean that the signal is at level H 50% of the time and at level L the other 50% of the time.



(four complete clock periods)

So-called system clocks usually have a mode switch for selecting single-pulse or periodic mode of operation.

The important design task is to emit only *complete* events when the mode switch is transferred from single to periodic and vice versa.

Project Tasks

1. Design a single-pulse/periodic clock circuit with debounced switch inputs using the ASM chart in Figure 7.10. Let q_1 and q_0 be the ASM state variables.

2. Build the circuit. Use 74LS00 gates to debounce switches. Use 74LS00, 74LS153, and 74LS175. Use an LED for the Ready report. Save it on your breadboard for future projects.

3. Test the circuit. Connect signal generator output clk to the clock input, and let $x = q_3$, $y = q_2 + q_1'$. Plot q_1, q_2, q_3, y, p, and Ready.

PROBLEMS

Note: In all problems requiring timing diagrams show "real" waveforms with finite propagation delays after clock edges.

7.1 The equations for a two-flip-flop circuit using a 74LS74 are: $d_0 = q_1'$, $d_1 = q_0$. Make the PS, NS table. Derive the state diagram, and draw the circuit. Start with all q_j low (state zero), and plot the q_j outputs on a timing diagram.

7.2 The equations for a two-flip-flop circuit using a 74LS109 are: $j_0 = q_1'$, $k_0 = q_0$, $j_1 = q_0$, $k_1 = q_1$. Make the PS, NS table. Derive the state diagram, and draw the circuit. Start with all q_j low (state zero), and plot the q_j outputs on a timing diagram.

7.3 The equations for a two-flip-flop circuit using a 74LS74 are: $d_0 = q_1'$, $d_1 = q_1'q_0 + q_1q_0'$. Make the PS, NS table. Derive the state diagram, and draw the circuit. Start with all q_j low (state zero), and plot the q_j outputs on a timing diagram.

7.4 The equations for a three-flip-flop circuit using a 74LS74 are: $d_0 = q_2$, $d_1 = q_2$ xor q_0, $d_2 = q_2$ xor q_1. Make the PS, NS table. Derive the state diagram, and draw the circuit. Start with all q_j high (state 7), and plot the q_j outputs on a timing diagram.

7.5 The equations for a three-flip-flop circuit using a 74LS74 are: $d_0 = q_2$, $d_1 = q_2$ xor q_0, $d_2 = q_1$. Make the PS, NS table. Derive the state diagram, and draw the circuit. Start with all q_j high (state 7), and plot the q_j outputs on a timing diagram.

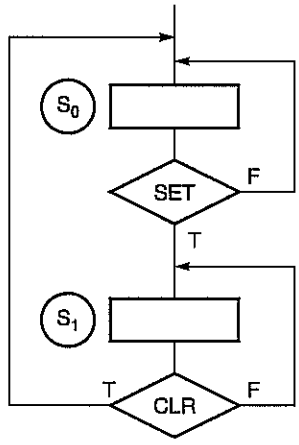
7.6 The equations for a two-flip-flop circuit using a 74LS109 are: $j_0 = 1$, $k_0 = 1$, $j_1 = q_0$, $k_1 = q_0$. Make the PS, NS table. Derive the state diagram, and draw the circuit. Start with all q_j low (state zero), and plot the q_j outputs on a timing diagram.

7.7 The equations for a two-flip-flop circuit using a 74LS109 are: $j_0 = q_0'$, $k_0 = q_0$, $j_1 = q_1$ xor q_0, $k_1 = q_1$ xor q_0'. Make the PS, NS table. Derive the state diagram, and draw the circuit. Start with all q_j low (state zero), and plot the q_j outputs on a timing diagram.

7.8 The equations for a two-flip-flop circuit using a 74LS109 are: $j_0 = 1$, $k_0 = 1$, $j_1 = q_0'$, $k_1 = q_0'$. Make the PS, NS table. Derive the state diagram, and draw the circuit. Start with state 2, and plot the q_j outputs on a timing diagram.

7.9 The equations for a two-flip-flop circuit using a 74LS109 are: $j_0 = q_0'$, $k_0 = q_0$, $j_1 = q_1$ xor q_0', $k_1 = q_1$ xor q_0. Make the PS, NS table.

FIGURE P7.1



Derive the state diagram, and draw the circuit. Start with all q_j low (state zero), and plot the q_j outputs on a timing diagram.

7.10 Design the ASM specified by the ASM chart in Figure P7.1.

(a) Use D flip-flops and multiplexers.
(b) Use JK flip-flops, gates, and decoders.

7.11 Design the ASM specified by the ASM chart in Figure P7.2.

(a) Use D flip-flops and multiplexers.
(b) Use JK flip-flops, gates, and decoders.

FIGURE P7.2

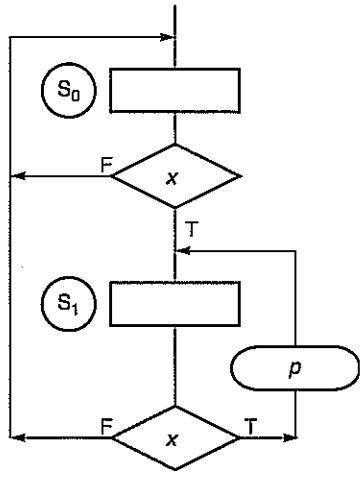


FIGURE P7.3

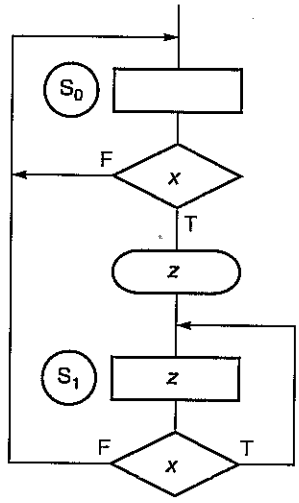
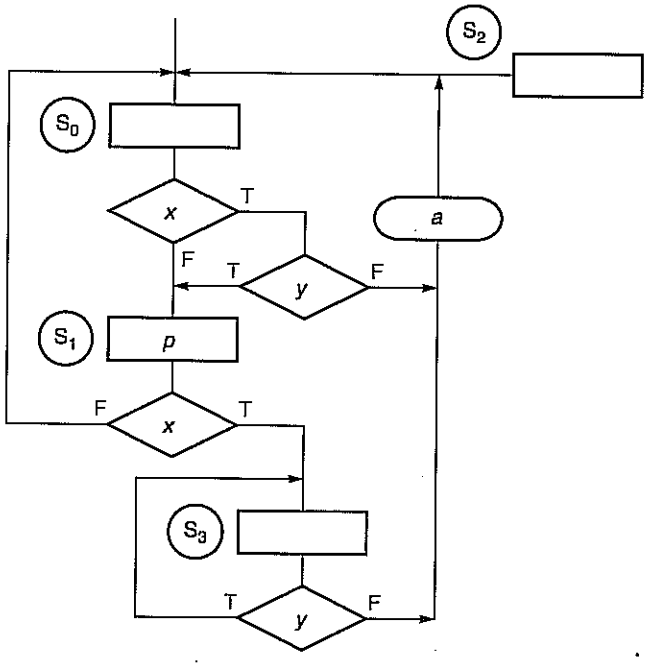


FIGURE P7.4



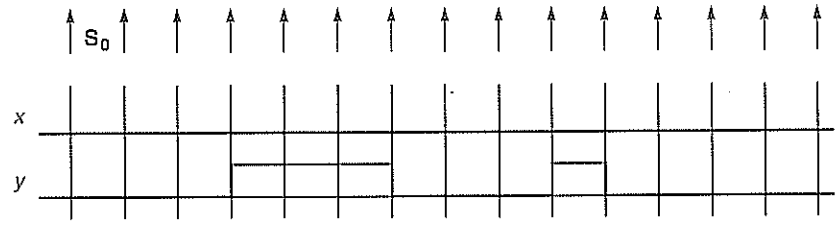
7.12 Design the ASM specified by the ASM chart in Figure P7.3.

(a) Use D flip-flops and multiplexers.
(b) Use JK flip-flops, gates, and decoders.

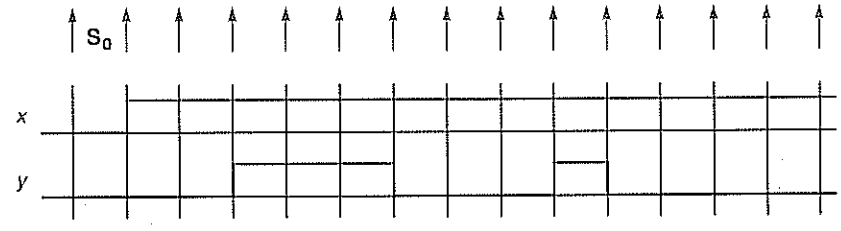
7.13 Design the ASM specified by the ASM chart in Figure P7.4.

(a) Use D flip-flops and multiplexers.
(b) Use JK flip-flops, gates, and decoders.

7.14 Given the x, y waveforms, plot p and a for the ASM chart in Figure P7.4. Show the sequence of states.



(a)

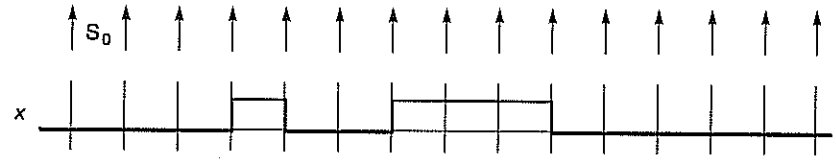


(b)

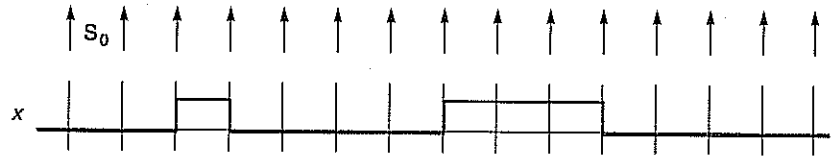
7.15 Design the ASM specified by the ASM chart in Figure P7.5.

(a) Use D flip-flops and multiplexers.
(b) Use JK flip-flops, gates, and decoders.
(c) Use D flip-flops and one-hot format.

7.16 Given the x waveforms, plot $p_1p_2r_1r_1$ for the ASM chart in Figure P7.5. Show the sequence of states.



(a)



(b)

7.17 Design an ASM that generates the sequence of states 0132013 etc. Use q_1q_0 as state variables. Draw the ASM chart.

(a) Use D flip-flops and multiplexers.
(b) Use JK flip-flops, gates, and decoders.

7.18 Design an ASM that generates the sequence of states 243675124 etc. Use $q_2q_1q_0$ as state variables. Draw the ASM chart.

(a) Use D flip-flops and multiplexers.
(b) Use JK flip-flops, gates, and decoders.

7.19 The sequence of states is 320132 etc. when input u is True. When u is False the sequence of states is 1023102 etc. Draw the ASM chart. Design the circuit.

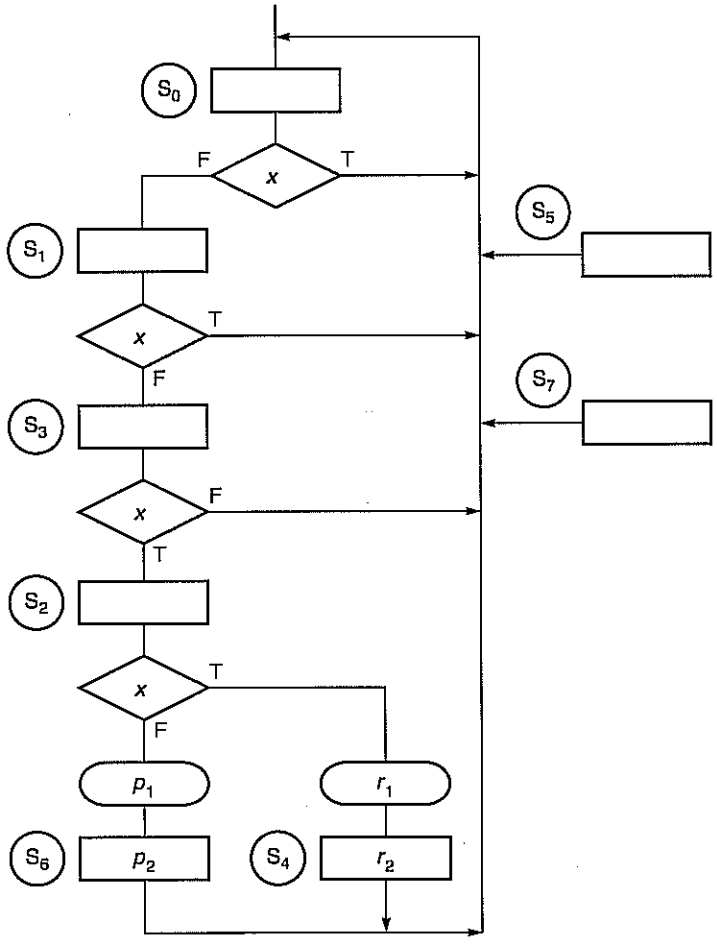
(a) Use D flip-flops and multiplexers.
(b) Use JK flip-flops, gates, and decoders.

7.20 Convert the timing diagram (0 = L, 1 = H) to an ASM chart. Design the circuit. q is a conditional output.

Input x	0	1	1	1	0	0	1	0	1	1	1	0	1	0
Output q	0	0	1	0	1	0	1	0	0	1	0	1	1	0

Use D flip-flops in a one-hot format.

FIGURE P7.5

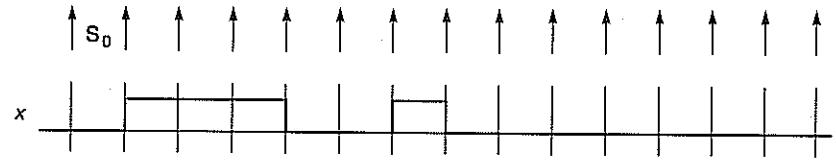


7.21 Design the ASM specified by the ASM chart in Figure P7.6.

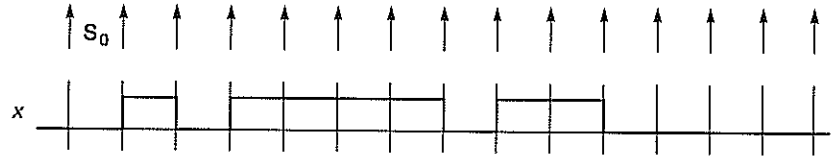
(a) Use D flip-flops and multiplexers.
(b) Use JK flip-flops, gates, and decoders.

7.22 Design the ASM specified by the ASM chart in Figure P7.7. Use D flip-flops in a one-hot format.

7.23 Given the x waveform, enter the sequence of states for the state machine in Example 7.5.



7.24 Given the x waveform, enter the sequence of states for the state machine in Example 7.6, and plot the g output waveform.



7.25 In Figure 7.32a, replace the exit condition $b = 0$ with $b = 1$, and repeat the analysis of Figure 7.34.

7.26 In Figure 7.32b, replace the exit condition $b > 0$ with $b = 0$, and repeat the analysis of Figure 7.36.

7.27 Design the circuit specified by the linked ASM chart in Figure P7.8.

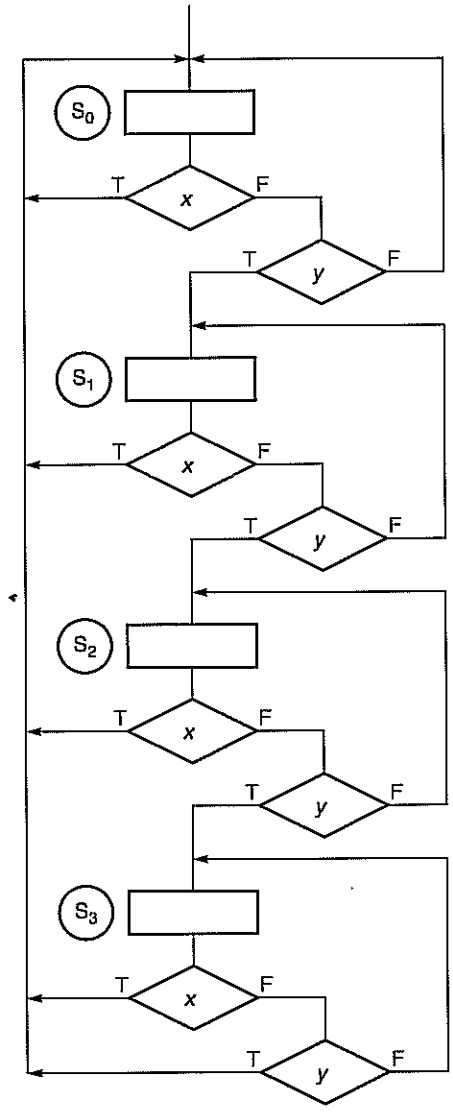
7.28 Design the circuit specified by the linked ASM chart in Figure P7.9.

7.29 Refer to the ASM chart linked to a 163 chip in Figure P7.10. Design the state machine specified by the ASM chart. Make a diagram showing the sequence of states, and waveforms for count, end, w, and load_6. Show the count number in each clock period.

7.30 Modify the ASM in Figure P7.10. Replace load_6 with load_n, where n = 0 to F hex. How would you set n? Draw three timing diagrams for three different n. Let n = 7, E, and F.

7.31 Refer to the ASM chart linked to a 163 chip in Figure P7.11. Design the state machine specified by the ASM chart. Make a diagram showing the sequence of states, and waveforms for count, end, load_n, and w. Show the count number in each clock period.

FIGURE P7.6



7.32 Refer to the ASM chart linked to two 163 chips in Figure P7.12. Design the state machine specified by the ASM chart. Make a diagram showing the sequence of states, and waveforms for count, =FF, =4F, load_EF, and *w*. Show the count number in each clock period.

7.33 Refer to Figure 7.40. Design the state machine.

7.34 Refer to Figure 7.40. Design the output circuits.

FIGURE P7.7

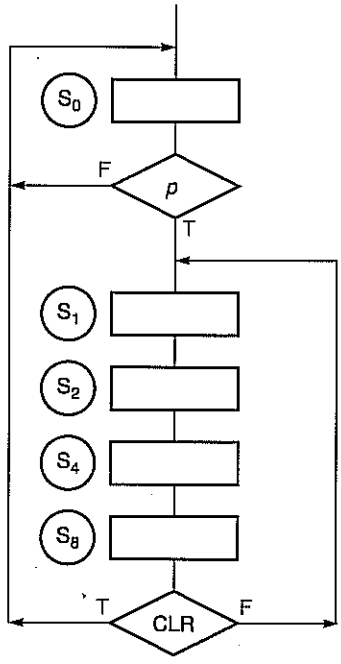


FIGURE P7.8

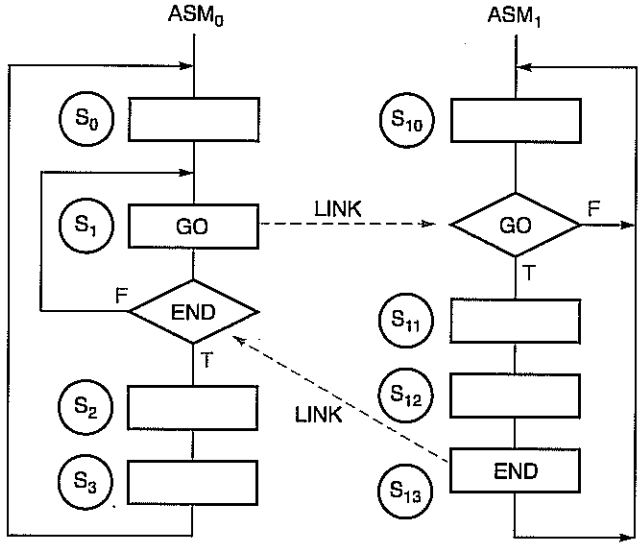


FIGURE P7.9

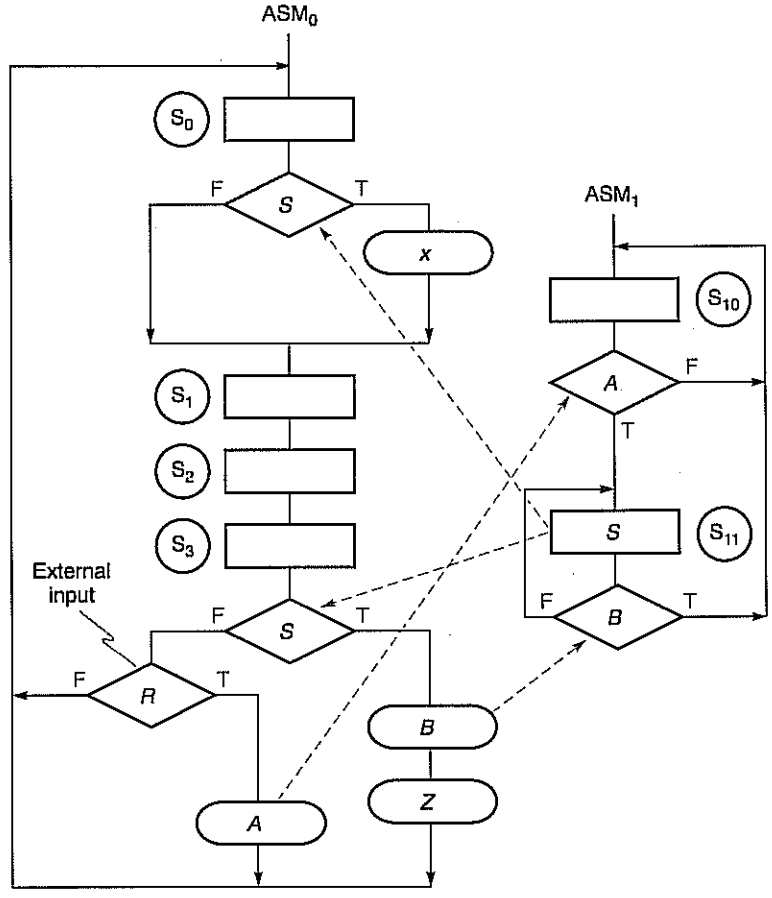


FIGURE P7.10

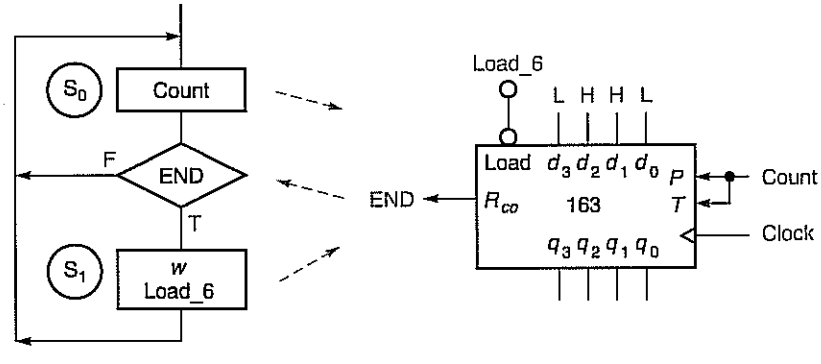


FIGURE P7.11

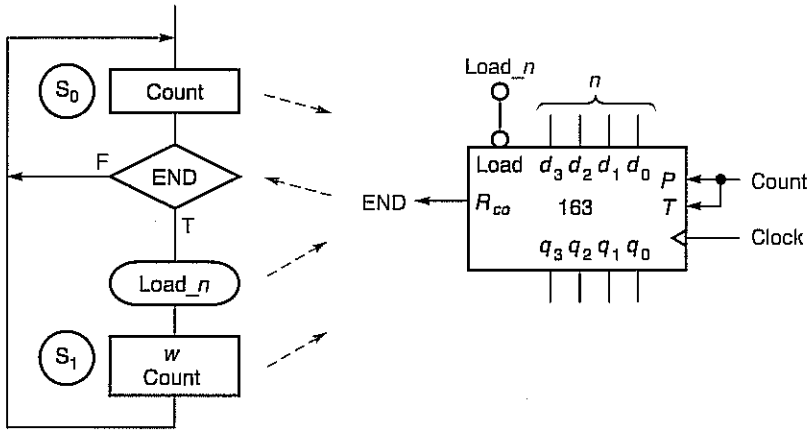
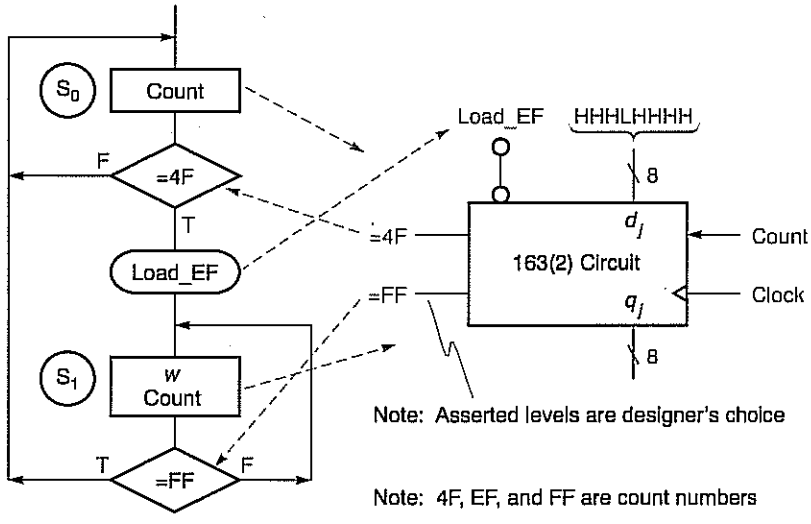


FIGURE P7.12



8 SEQUENTIAL BUILDING BLOCKS