CHAPTER 11

Sequential Circuits

TABLE OF CONTENTS

Chapter 10 discusses combinational devices, which are commonly used in computer design. As useful as these devices are, however, it is impossible to build even the smallest computers as an interconnection of combinational circuits. In all the combinational devices mentioned, the output depends only on the input. When the input changes, it is only a matter of a few gate delays before that change is reflected in the output.

The distinguishing characteristic of a sequential circuit is its state.

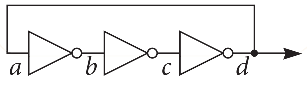
The *state* of the circuit is the characteristic that distinguishes a sequential circuit from a combinational circuit. A sequential circuit can remember what state it is in. It has memory, in other words. The output of a sequential circuit depends not only on the input, but also on its state.

This chapter shows how to construct the basic sequential elements and how to connect them to form useful blocks at successively higher levels of abstraction. It concludes with a description of devices that are connected to build a Pep/9 computer in the following chapter.

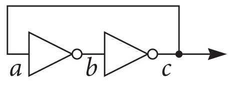
11.1 Latches and Clocked Flip-Flops

Sequential devices are constructed from the same gates described in Chapter 10, but with a different type of connection called *feedback*. In combinational circuits and the Boolean expressions that describe them, the output of each gate goes to the input of a previously unconnected gate. A feedback connection, however, forms a loop or cycle where the output of one or more gates "feeds back" to the input of a previous gate of the circuit.

FIGURE 11.1
Simple feedback circuits.



(a) An unstable circuit.



(b) A stable circuit.

FIGURE 11.1 shows two simple circuits with feedback connections. Part (a) is a string of three inverters with the output of the last inverter fed back to the input of the first inverter. To analyze the behavior of this circuit, suppose that point *d* has a value of 1. Because *d* is connected to *a* with the feedback loop, point *a* must also have a value of 1. One gate delay later, point *b* must have a value of 0. (Although we previously ignored the delay through an inverter for simplicity, we must take it into account with this circuit.) One more gate delay later, point *c* will have a value of 1, and after a third gate delay, *d* will have a value of 0.

The problem now is that we started our analysis assuming that point *d* has a value of 1. It will now change to 0, and three gate delays later it will become 1 again. The circuit will oscillate, with the values at each point in the circuit switching back and forth between 1 and 0 every few gate delays. A state that remains constant only for a duration of a few gate delays is called an *unstable state*.

Unstable states

In Figure 11.1(b), if you assume point *c* has a value of 1, then point *a* will be 1, *b* will be 0, and *c* will be 1, which is consistent with the first assumption. Such a state is stable. The points at all parts of the circuit will retain their values indefinitely.

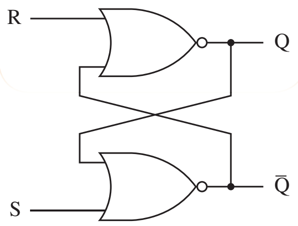
Stable states

Another possible stable state is for points *c* and *a* to have a value of 0 and *b* to have a value of 1. If you construct this circuit, which state will it have? Will point *c* be 0 or 1? Like all electrical devices, gates require a source of electrical power and must be turned on to operate. If you construct the circuit of Figure 11.1(b) and turn it on, its state will be established at random. About half the time when you turn on the circuit, point *c* will be 0, and about half the time it will be 1. The circuit will remain indefinitely in the state that is established when the power is turned on.

The SR Latch

To be useful, a sequential device needs a mechanism for setting its state. Such a device is the *SR latch* of FIGURE 11.2. Its two inputs are S and R, and its two outputs are Q and $\bar{Q}$ (pronounced *Q bar*). The feedback connections are from Q to the input of the bottom NOR and from $\bar{Q}$ to the input of the top NOR.

FIGURE 11.2
The SR latch.



To see the possibility of a stable state, suppose S and R are both 0, and Q is also 0. The two inputs to the bottom gate are both 0, which makes $\bar{Q}$ 1. The two inputs to the top NOR are 0 (from R) and 1 (from $\bar{Q}$). Thus, the output of the top NOR is 0, which is consistent with our first assumption about Q. So the stable state is Q $\bar{Q}$ = 01 when SR = 00.

Starting with this stable state, consider what happens if you change input S to 1. FIGURE 11.3 summarizes the sequence of events. T_g stands for a time interval of one gate delay, typically 2 ns.

At time 0, S changes to 1. That makes the two inputs to the bottom gate 1 (from S) and 0 (from Q). One gate delay later, the effect of that change propagates to $\bar{Q}$, which becomes 0. Now the two inputs to the top gate are 0 (from R) and 0 (from $\bar{Q}$). After another gate delay, the output of the top gate

FIGURE 11.3
Changing S to 1 in the SR latch.

Time	S	R	Q	$\bar{Q}$	Stability
Initial	0	0	0	1	Stable
0	1	0	0	1	Unstable
T_g	1	0	0	0	Unstable
$2T_g$	1	0	1	0	Stable

FIGURE 11.4
Changing S back to 0 in the SR latch.

Time	S	R	Q	$\bar{Q}$	Stability
Initial	1	0	1	0	Stable
0	0	0	1	0	Stable

becomes 1. Now the input to the bottom gate is 1 (from S) and 1 (from Q). That makes the output of the bottom gate 0.

The output of the bottom gate was already 0, however, so it does not change. Because a trace through the feedback connections shows consistent values, this last state is stable. The two intermediate states in Figure 11.3 are unstable because they last for only a few gate delays.

What happens if you change S back to 0? FIGURE 11.4 shows the sequence of events. The two inputs to the bottom gate are 0 (from S) and 1 (from Q). That makes the output of the bottom gate 0. Because it was already 0, no other changes propagate through the circuit, and the state is stable.

Figures 11.3 and 11.4 show that $\bar{Q}$ is always the complement of Q when the SR latch is in a stable state. The bar is another common notation for the complement and is equivalent to the prime notation used in Chapter 10.

Output depends on input and state.

Compare the first state of Figure 11.3 with the last state of Figure 11.4. In both cases the inputs are SR = 00, but in the first case the output is Q = 0 and in the second Q = 1. The output depends not only on the input, but also on the state of the latch.

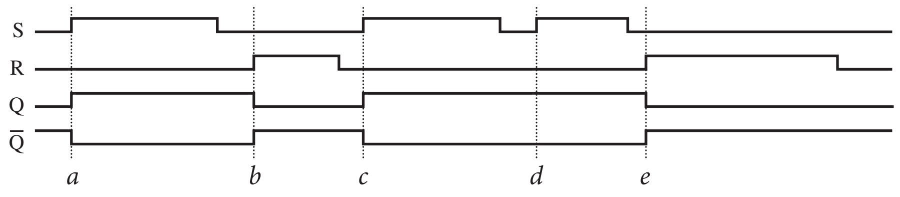
The effect of changing S to 1 and then back to 0 was to set the state to Q = 1. If the latch begins in the state Q = 1 with SR = 00, a similar analysis shows that changing R to 1 and then back to 0 will reset the state to Q = 0. *S* stands for *set*, and *R* stands for *reset*.

An SR latch is analogous to a light switch on a wall. Changing S to 1 and back to 0 is like flipping the switch up to turn the light on. Changing R to 1 and back to 0 is like flipping the switch down. If the switch is already up and you try flipping it up, nothing changes. The switch stays up. Similarly, if Q is already 1 and you change S to 1 and back to 0, the state does not change. Q stays 1.

The normal input condition for the SR latch

The normal input condition for the SR latch is SR = 00. To set or reset the latch, you change S or R to 1 and then back to 0. Normally, S and R are not 1 simultaneously. If S and R are both 1, Q and $\bar{Q}$ will both be 0, and $\bar{Q}$ will not be the complement of Q. Furthermore, if you change SR = 11 to SR = 00 simultaneously, the state of the latch is unpredictable. About half the time it will return to $Q\,\bar{Q}$ = 01 and half the time to $Q\,\bar{Q}$ = 10. In practice, SR = 11 should not occur.

FIGURE 11.5
A timing diagram for the SR latch.



A *timing diagram* is a graphical representation of the behavior of a sequential circuit as the signals evolve in time. FIGURE 11.5 is a timing diagram that shows the change in outputs Q and $\overline{Q}$ as inputs S and R change. The horizontal axis is time, and the vertical axis is the electrical voltage that represents 1 when it is high and 0 when it is low.

The timing diagram shows the initial state as Q = 0. When S goes to 1 at time *a*, it immediately sets Q to 1 and $\overline{Q}$ to 0. The diagram shows the transitions occurring simultaneously. As shown in our previous analysis, $\overline{Q}$ will change one gate delay after S changes, and Q will change one gate delay later than that. The timing diagram assumes that the time scale is too large to show a time interval as short as a gate delay.

When S goes back to 0, the state does not change. When R goes to 1 at time *b*, it resets Q to 0. When S goes to 1 again at time *c*, it sets Q to 1. At time *d*, the 0-to-1 transition of S does not change the state of the latch because Q is already 1. At all points of the diagram, $\overline{Q}$ is the inverse of Q.

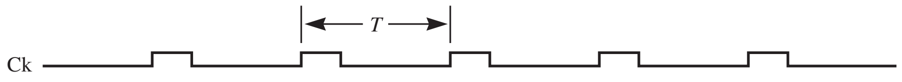
Figure 11.5 also shows the transitions as instantaneous. Nothing in nature occurs in zero time. If you magnify the time scale, the transitions will show a more gradual change with a finite slope at every point. You can think of the simultaneous and instantaneous transitions in the timing diagram as a high level of abstraction that hides the details of the gate delays and gradual slopes at a lower level of abstraction.

The Clocked SR Flip-Flop

A subsystem in a computer consists of many combinational and sequential devices. Each sequential device is like an SR latch, which is in one of two states. As the machine executes its von Neumann cycle, the states of all the sequential devices change with time. To control this large collection of devices in an orderly fashion, the machine maintains a clock and requires all the devices to change their states at the same time. The clock generates a sequence of pulses, as in FIGURE 11.6. *Ck* stands for *clock pulse*.

Every sequential device has a Ck input in addition to its other inputs. The device is designed to respond to its inputs only during a clock pulse. The

FIGURE 11.6
A sequence of clock pulses.



The clock period

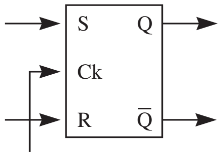
time between pulses, indicated by *T* in the figure, is the *period* of the clock. The shorter the period, the more frequently the devices will change their states and the faster the circuit will compute.

FIGURE 11.7 is an SR latch with a clock input, called a *flip-flop*. It consists of the same pair of NOR gates with feedback that were shown in Figure 11.2. But instead of the SR inputs going directly into the NOR gates, they go through two AND gates that act as an enable. Figure 11.7(a) is the block diagram, and Figure 11.7(b) is an implementation. Notice that the convention for the block diagram has S opposite from Q at the top of the block, and the implementation has S opposite $\overline{\text{Q}}$.

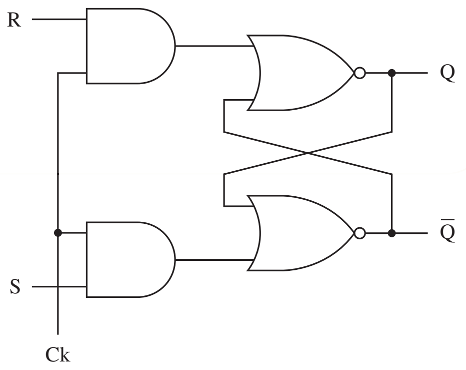
During the time that Ck is low, the inputs to the NOR gates will be 0 regardless of the values of S and R. When there are inputs of 0 into the NOR gates, it means that the latch will not change its state. During the time that Ck is high, the values of S and R pass through the enable gates unchanged. The device behaves as the SR latch of Figure 11.2. The AND gates shield the NOR gates from the effect of S and R except during the time that Ck is high. FIGURE 11.8 is a timing diagram that shows the behavior of the device. $\overline{\text{Q}}$ is always the complement of Q and is not shown in the figure.

When S goes to 1, that change does not affect Q because the clock is still low. At time *a* when the clock goes high, Ck allows SR = 10 to pass through the AND gates and set the latch to Q = 1. A little later when the clock goes

FIGURE 11.7AB
The clocked SR flip-flop.

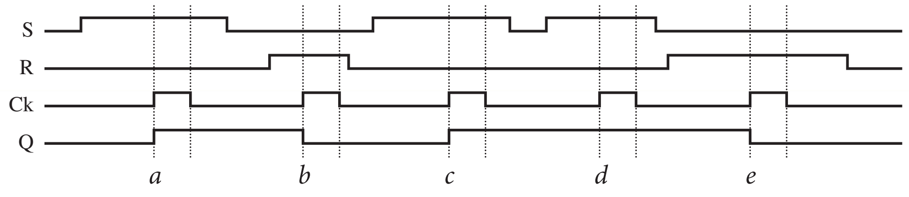


(a) Block diagram.



(b) Implementation.

FIGURE 11.8
A timing diagram of the clocked SR flip-flop.



low, Ck disables the SR input from the latch. When R goes to 1 before time *b*, it cannot affect the state of the latch. Only at time *b*, when Ck goes high again, can R reset the latch.

It would be physically possible to let S and R make several transitions during a single time interval when the clock is high, but that does not happen in practice. The idea is to design the circuit to set up the SR input for the desired transition and wait for the next clock pulse. When the clock pulse arrives, the state may change according to the values of S and R. After the clock goes low, the circuit prepares the SR values for the next pulse.

Digitizing the time axis

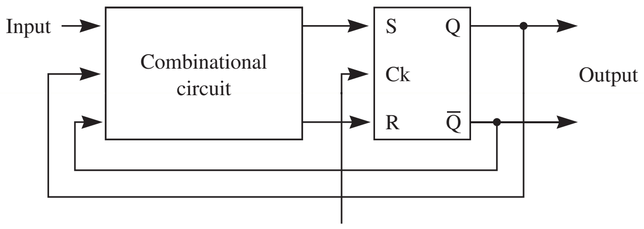
The evenly spaced clock pulses force any change in state to occur only at evenly spaced time intervals. The S and R inputs of Figure 11.8 are identical to those of Figure 11.5, but the corresponding state changes in Q have been smoothed out by the clock. The effect of a clock is to make time (the horizontal axis of a timing diagram) digital in the same way that the electrical circuitry makes the voltage signal (vertical axis) digital. Just as a signal must be either high or low and never anything in between, a state change of a sequential device must occur at either one clock pulse or another—never between two pulses.

The Master–Slave SR Flip-Flop

The clocked flip-flop of Figure 11.7 is called *level sensitive* because the latch responds to Ck only when the clock is at a high level. Although the device is constrained to follow the clock as desired, it has a serious practical deficiency, illustrated in FIGURE 11.9.

The figure shows a possible interconnection of an SR device. It is common for the output of a sequential device to contain a feedback loop through some combinational circuit that eventually leads to the input of the same sequential device. The figure shows a three-input, two-output combinational circuit, two of whose inputs are feedback from the output of the SR sequential device. This feedback loop is in addition to the feedback of the NOR gates not shown in the figure within the SR block.

FIGURE 11.9
A possible interconnection of an SR device.



Consider what might happen if the SR flip-flop were level sensitive. Suppose SR is 10, $Q\bar{Q}$ is 01, and the clock is low. Because the clock disables SR from the NOR latch, S cannot set Q to 1. Now suppose the clock goes high, and after a few gate delays SR sets Q to 1.

Now imagine that the change in $Q\bar{Q}$, after propagating through the combinational circuit with the same external input, makes SR = 01. If Ck is still high, the clock will allow the value of SR to reset Q to 0 after a few more gate delays. Unfortunately, a value of 01 for $Q\bar{Q}$ will propagate through the combinational circuit again and change SR to 10.

Possibility of an unstable state with feedback

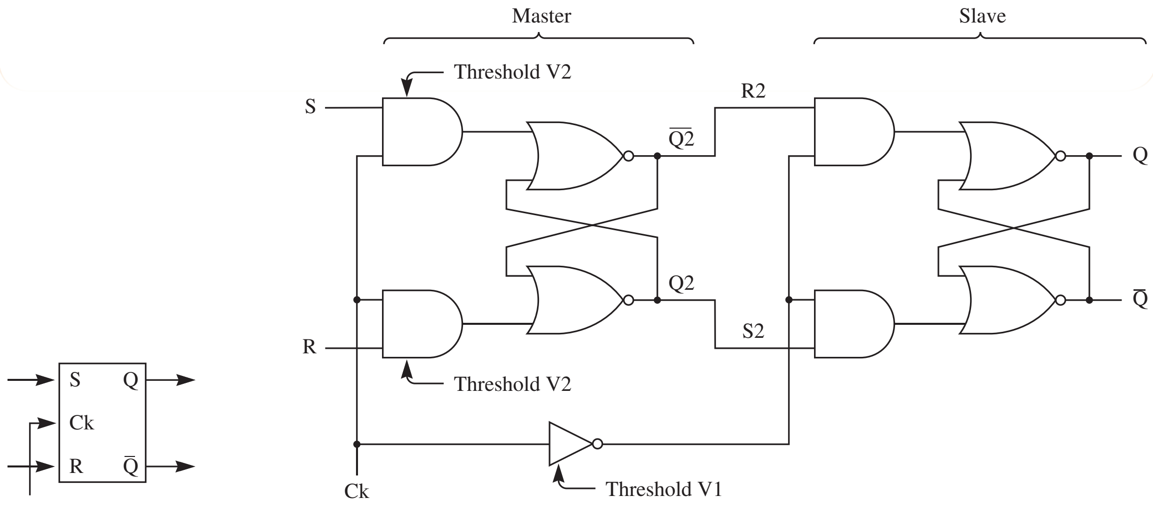
You should recognize this situation as unstable. Every few gate delays, the feedback connection from the sequential device forces the SR flip-flop to change its state as long as the clock is high. The state may change hundreds of times while the clock is high. When the clock eventually goes low at the end of its pulse, it would be impossible to predict exactly what state the flip-flop would be in.

Because feedback connections through combinational circuits are necessary for the construction of computer subsystems, we need a sequential device that is not only constrained to change its state during a clock pulse, but is also immune from further changes through the feedback connection. The device needs to be sensitive to its input for an extremely short period of time—so short that no matter how fast the feedback propagates through the combinational circuit and changes SR, that change cannot again affect the state of the flip-flop.

Two design solutions to the instability problem from feedback

Two techniques for designing such devices are edge-triggered and master–slave. *Edge-triggered* flip-flops are not designed to be sensitive to their inputs when the clock is at a high level, but rather when the clock is making a transition from a low level to a high level. The implementation of an edge-triggered flip-flop is more difficult to understand than the implementation of a master–slave flip-flop and will not be considered here even though it is

FIGURE 11.10
The master–slave SR flip-flop.



(**a**) Block diagram. (**b**) Implementation.

the more common of the two. It is enough to say that both types of flip-flops solve the same problem arising from feedback connections.

Operation of the master–slave circuit

FIGURE 11.10 shows an implementation of the *master–slave* SR flip-flop. Both the master and the slave are level-sensitive, clocked SR flip-flops. The $\overline{Q}$ output of the master ($\overline{Q2}$) connects to the R input of the slave (R2), and the Q output of the master (Q2) connects to the S input of the slave (S2). Ck connects to the enable of the master, and the complement of Ck connects to the enable of the slave. The block diagram of a master–slave flip-flop is identical to the block diagram of the level-sensitive flip-flop.

Because the master is an SR flip-flop, Q2 and $\overline{Q2}$ will always be the complement of each other. Because Q2 is connected to S2 of the slave and $\overline{Q2}$ is connected to R2 of the slave, when the slave is clocked, the slave will either be set or reset, depending on the state of the master. If the master is in state Q2 = 1, the master will set the slave to Q = 1 also. If the master is in state Q2 = 0, it will reset the slave to Q = 0. The reason for the master–slave terminology is that the slave obediently takes the state of the master when the slave is clocked.

The threshold of a gate

The *threshold* of a gate is the value of the input signal that causes the output to change. For the master–slave circuit to function properly, the inverter and the enable gates of the master must be designed with special

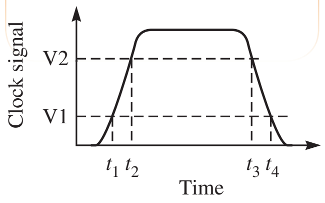


FIGURE 11.11 Timing detail of a single clock pulse.

threshold values. The threshold of the inverter, V1, must be less than the threshold of the master enable gates, V2.

FIGURE 11.11 shows V1 and V2 on a magnified timing diagram of Ck during one clock pulse. The clock does not make an instantaneous transition from a low to a high value. Instead, it gradually increases first to value V1 at time t_1, then to value V2 at time t_2 on the way to its high level. On the way down, it passes through value V2 at time t_3, and then V1 at time t_4.

Before the pulse begins its upward transition, the master is disabled from the input. Regardless of the value of SR, the master will remain in its established state. The inverter ensures that the slave input is connected to the master because the slave inputs are enabled. The slave must be in the same state as the master.

As the clock signal rises and falls, passing through times t_1, t_2, t_3, and t_4, the effect of the circuit is as follows:

- t_1: Isolate slave from master.
- t_2: Connect master to input.
- t_3: Isolate master from input.
- t_4: Connect slave to master.

At time t_1 the signal reaches the threshold of the inverter, which makes the output of the inverter change from 1 to 0. A 0 to the enable gates of the slave shields the slave from any further effects of S2 and R2. Whatever state the slave was in at time t_1, it will remain in that state as long as Ck is above the threshold value V1.

At time t_2 the signal reaches the threshold of the master enable gates, which makes the master sensitive to the SR inputs. If SR is 10, the input will set the master to Q2 = 1. If SR is 01, the input will reset the master to Q2 = 0. If SR is 00, the master will not change its state. However, if the master does change its state, its new state will not affect the slave because the slave was isolated from the master at time t_1.

Consider how this arrangement protects the flip-flop from the feedback connection of Figure 11.9. The feedback is based on the Q $\bar{\text{Q}}$ output of the slave, which does not change as a result of the input to the master. The fact that V1 is less than V2 ensures that the slave will be isolated from the master before the master becomes sensitive to the input. Even if the gate delay through the combinational circuit were zero, the feedback would not affect the state of the slave.

When Ck makes its high-to-low transition, the clock reaches value V2 at time t_3. V2 is the threshold of the enable gates for the master, so now the master becomes insensitive to the input. Because V2 is greater than V1, the slave is still isolated from the effect of the master.

At time t_4 the clock signal becomes less than the threshold of the inverter. The output of the inverter changes from 0 to 1, connecting the slave to the master. Whatever state the master was in is forced on the slave. The slave may change its state. If there is feedback from the output of the slave to the input of the master, it will not affect the master because the master was isolated from the input at time t_3.

The decompression chamber analogy

A rough analogy of the operation of a master–slave circuit is a decompression chamber in a spacecraft. Inside the craft, the astronauts do not need to wear spacesuits. To go for a space walk outside the craft, an astronaut dons a spacesuit and approaches the decompression chamber, which has two doors that are initially closed.

The astronaut opens the inner door, which connects the chamber with the craft, and steps inside the chamber. She closes the inner door, isolating the chamber from the craft, and opens the outer door, connecting the chamber with outer space. She exits the outer door, closing it behind her. At no time are both doors open at the same time. If they were, the craft would lose all its air to outer space.

Similarly, the master–slave circuit has two doors—one to isolate or connect the master to the input and one to isolate or connect the slave to the master. At no time are both doors open, with the master connected to the input and the slave connected to the master. If they were, a feedback loop might cause an unstable state in the circuit.

FIGURE 11.12 is a timing diagram of the behavior of a master–slave SR flip-flop. A change occurs in Q, the state of the slave latch, at time t_4, when the slave is connected to the master. That is during the high-to-low transition of Ck.

Characteristic tables

Because a flip-flop is not combinational, a truth table is not sufficient to characterize its behavior. Instead, flip-flops have *characteristic tables* that specify the state of the device after one clock pulse for a given input and initial state. FIGURE 11.13 is the characteristic table for the SR flip-flop.

S(t) and R(t) are the inputs at time t before a clock pulse. Q(t) is the state of the flip-flop before a clock pulse, and Q(t + 1) is the state after the pulse.

FIGURE 11.12
A timing diagram of the master–slave SR flip-flop.

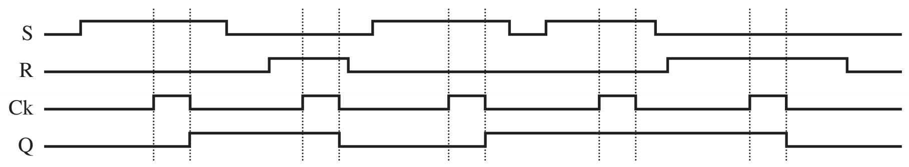


FIGURE 11.13
The characteristic table for the SR flip-flop.

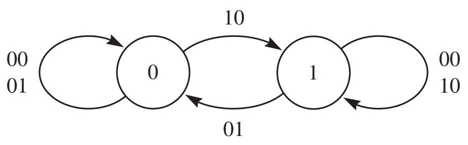
S(*t*)	R(*t*)	Q(*t*)	Q(*t* + 1)	Condition
0	0	0	0	No change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	–	Not defined
1	1	1	–	

The table shows that if SR is 00, the device does not change its state when clocked. If SR is 01, the device resets to Q = 0, and if it is 10, the device sets to Q = 1. It is impossible to predict what the state would be if SR = 11 when clocked.

The characteristic table is, in essence, a state transition table similar to Figure 7.11 for a finite-state machine. An SR flip-flop is a finite-state machine with two possible states, Q = 0 and Q = 1. As with any finite-state machine, its behavior can be characterized by a state transition diagram. FIGURE 11.14 is the state transition diagram for an SR flip-flop.

Circles denote the states of the machine with the value of Q inside the circle. Transitions are labeled with the values of SR that produce the given transition. For example, the transition from Q = 0 to Q = 1 is labeled with SR = 10.

FIGURE 11.14
The state transition diagram for an SR flip-flop.



The Basic Flip-Flops

Four flip-flops are common in computer design:

SR	Set/reset
JK	Set/reset/toggle
D	Data or delay
T	Toggle

The previous section shows how to construct the SR flip-flop. Its characteristic table defines its behavior. The other three flip-flops have their own characteristic tables that define their behaviors. Each one can be constructed from the SR flip-flop together with a few other gates. Like the SR flip-flop, the others all have Q and $\overline{\text{Q}}$ outputs. The JK flip-flop has two inputs labeled *J* and *K* in place of *S* and *R*. The D and T flip-flops each have only one input.

There is a systematic procedure to construct the other flip-flops from the SR flip-flop. The general circuit has the structure of Figure 11.9, where the Q and $\overline{\text{Q}}$ outputs of the SR flip-flop act as the Q and $\overline{\text{Q}}$ outputs of the device under construction. For the JK flip-flop, the input line of Figure 11.9 is actually two input lines, one for J and one for K. For the D flip-flop, the one input line is labeled *D*, and for the T flip-flop, it is labeled *T*. To design each flip-flop, you must determine the logic gates and their interconnection in the box labeled *Combinational circuit*.

As with any combinational circuit design, once you determine the required input and output in the form of a truth table, you can construct the minimum AND-OR circuit with the help of a Karnaugh map. So, the first step is to write down the required input and output of the box labeled *Combinational circuit* in Figure 11.9. A useful tool to determine the specification of the circuit is the *excitation table* of the SR flip-flop in FIGURE 11.15.

FIGURE 11.15
The excitation table for the SR flip-flop.

Q(*t*)	Q(*t* + 1)	S(*t*)	R(*t*)
0	0	0	×
0	1	1	0
1	0	0	1
1	1	×	0

Excitation tables versus characteristic tables

Contrast the excitation table with the characteristic table in Figure 11.13. The characteristic table tells you what the next state is, given the current input and the current state. But the excitation table tells you what the current input must be, given the desired transition. Here is how you build the excitation table from the characteristic table.

Constructing excitation tables

The first table entry is for the transition from Q = 0 to Q = 0. Two possible inputs permit this transition: SR = 00 and SR = 01. An SR value of 00 specifies the no-change condition. An SR value of 01 specifies the reset condition. Either of these conditions will cause the flip-flop to make the transition from Q = 0 to Q = 0. The × in the entry under R(t) is a don't-care value. As long as S is 0, you do not care what the value of R is. The transition will be from Q = 0 to Q = 0 regardless of the value of R.

The second entry in the table is for the transition from Q = 0 to Q = 1. The only way to force this transition is to input 10 for SR, the set condition. Similarly, the third entry is for the transition from Q = 1 to Q = 0, which can occur only with a value of 01 for SR.

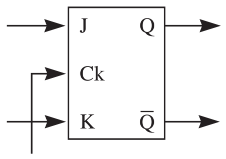
The last entry is for the transition from Q = 1 to Q = 1. The two possible input conditions that permit this transition are SR = 00, the no-change condition, and SR = 10, the set condition. Regardless of the value of S, if R is 0, the transition will occur. The × under S(t) indicates the don't-care condition for S.

The JK Flip-Flop

The JK flip-flop resolves the undefined transition in the SR flip-flop. The J input acts like S, setting the device, and the K input acts like R, resetting the device. But when JK = 11, the condition is called the *toggle condition*. To toggle means to switch from one state to the other. In the toggle condition, if the initial state is 0, the final state will be 1, and if the initial state is 1, the final state will be 0. FIGURE 11.16 is the block diagram and the characteristic table for the JK flip-flop.

For the JK flip-flop, the box labeled *Combinational circuit* in Figure 11.9 has three inputs and two outputs. The inputs are J, K, and Q, which comes from the feedback connection. FIGURE 11.17 is the same figure with the J and K inputs explicit. The dashed box defines a JK flip-flop with three inputs (J, K, and Ck) and two outputs (Q and $\overline{Q}$). The Q feedback is at a lower level of abstraction and is not visible to the user of the JK flip-flop. Figure 11.17 shows only one feedback line, Q, from the output of the SR flip-flop. The $\overline{Q}$ output from the SR flip-flop is also available as a possible input to the box labeled *Combinational circuit*. However, when you design a combinational circuit, you always assume that both the input signal and its complement are available. Figure 11.17 emphasizes that the combinational

FIGURE 11.16
The JK flip-flop.

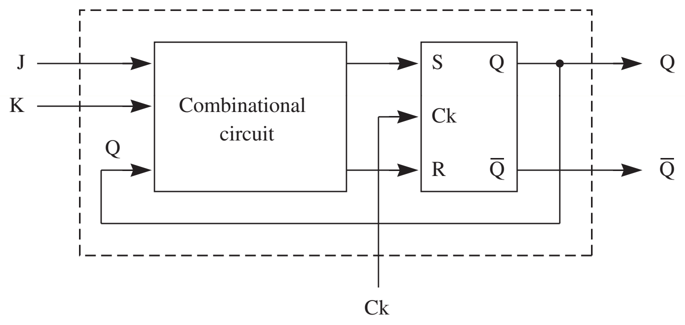


(a) Block diagram.

J(*t*)	K(*t*)	Q(*t*)	Q(*t* + 1)	Condition
0	0	0	0	No change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	1	Toggle
1	1	1	0	

(b) Characteristic table.

FIGURE 11.17
Constructing a JK flip-flop from an SR flip-flop.



circuit is a three-input, two-output circuit. The outputs are S and R, and each is a function of three inputs. That is, S = S(J, K, Q) and R = R(J, K, Q). So to construct a JK flip-flop from an SR flip-flop, you need to design two three-input combinational circuits, one for S and one for R.

Constructing design tables

First, write down the design table of FIGURE 11.18 with the help of the SR excitation table. The design table tells you the inputs that are necessary for the SR flip-flop, given the transitions that the JK flip-flop must make. The first three columns list all the possible input combinations of the

FIGURE 11.18
The design table to construct a JK flip-flop from an SR flip-flop.

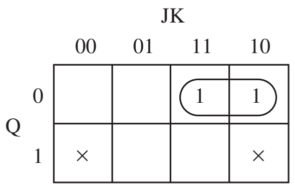
Q(t)	J(t)	K(t)	Q(t + 1)	S(t)	R(t)
0	0	0	0	0	×
0	0	1	0	0	×
0	1	1	1	1	0
0	1	0	1	1	0
1	0	0	1	×	0
1	0	1	0	0	1
1	1	1	0	0	1
1	1	0	1	×	0

three inputs to the combinational circuit. It is a good idea to order the JK values as they will appear in the Karnaugh maps. The fourth column is the value of Q after the clock pulse. Each value of Q(t + 1) comes from the characteristic table of the JK flip-flop. For example, in the third row, JK = 11, which is the toggle condition. So the initial state Q(t) = 0 toggles to Q(t + 1) = 1. The last two columns come from the excitation table for the SR flip-flop, given Q(t) and Q(t + 1). For example, the third row has transition Q(t) Q(t + 1) = 01. The excitation table shows that SR must be 10 for that transition.

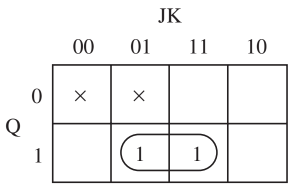
The next step is to write down the Karnaugh maps for the functions. The entries in FIGURE 11.19 (a) come from the column labeled S(t) in the design table, and the entries in (b) come from R(t).

By inspection of the Karnaugh maps, you can write the minimized AND-OR expression for S as S = J$\overline{\text{Q}}$, and for R as R = KQ. FIGURE 11.20 shows the complete design. You implement the JK flip-flop with an SR flip-flop and two two-input AND gates. You can see how the design works by considering all the possible values of JK. If JK = 00, then SR = 00 regardless of the state, and the state does not change. If JK = 11 and Q = 0, then SR = 10 and Q will change to 1. If Q = 1 initially, then SR = 01 and Q will change to 0. In both cases, the state toggles, as it should for JK = 11. You should convince yourself that the circuit works correctly for JK = 01 and JK = 10 as well.

FIGURE 11.19
The Karnaugh maps to construct a JK flip-flop from an SR flip-flop.

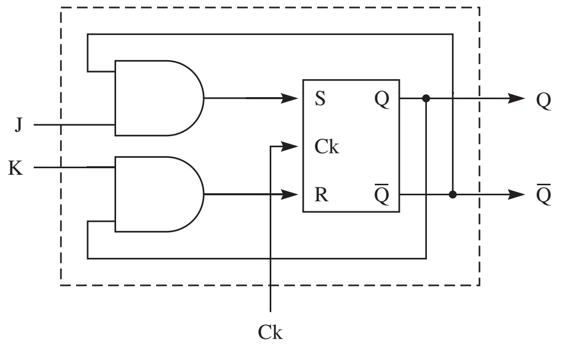


(a) Karnaugh map for S.



(b) Karnaugh map for R.

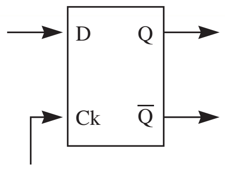
FIGURE 11.20
Implementation of the JK flip-flop.



The D Flip-Flop

The D flip-flop is a data flip-flop with only one input, D, besides the clock. FIGURE 11.21 (a) is its block diagram and (b) is its characteristic table. The table shows that Q(t + 1) is independent of Q(t). It depends only on the value of D at time t. The D flip-flop stores the data until the next clock pulse. Part (c) of the figure shows a timing diagram. This flip-flop is also called a

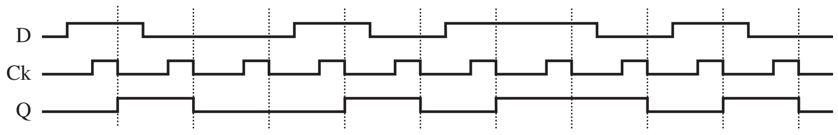
FIGURE 11.21
The D flip-flop.



(a) Block diagram.

D(t)	Q(t)	Q(t + 1)	Condition
0	0	0	Delay
0	1	0	
1	0	1	Delay
1	1	1	

(b) Characteristic table.



(c) A timing diagram.

delay flip-flop because on the timing diagram, the shape of Q is identical to that of D except for a time delay.

To construct a D flip-flop from an SR flip-flop, first construct the design table. Because there is only one input besides Q, there are only four rows in the table, as FIGURE 11.22 (a) shows. Parts (b) and (c) show the Karnaugh maps, which contain only four cells instead of eight. Minimization of the AND-OR circuits gives S = D and R = $\overline{\text{D}}$.

FIGURE 11.23 is the implementation of the D flip-flop. It requires only a single inverter in addition to the SR flip-flop. This implementation has no feedback connections from Q or $\overline{\text{Q}}$ as the JK implementation does, because the next state does not depend on the current state.

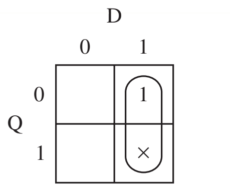
The T Flip-Flop

The T flip-flop is a toggle flip-flop. Like the D flip-flop, it has only one input, T, besides the clock. FIGURE 11.24 (a) is the block diagram and (b) is the characteristic table. The T input acts like a control line that specifies a

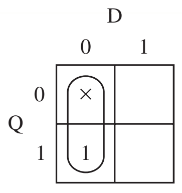
FIGURE 11.22
The design table and Karnaugh maps to construct a D flip-flop from an SR flip-flop.

Q(*t*)	D(*t*)	Q(*t* + 1)	S(*t*)	R(*t*)
0	0	0	0	×
0	1	1	1	0
1	1	1	×	0
1	0	0	0	1

(a) Design table.



(b) Karnaugh map for S.



(c) Karnaugh map for R.

FIGURE 11.23
Implementation of the D flip-flop.

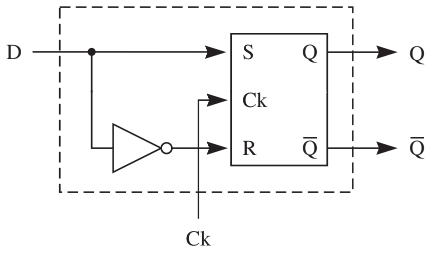
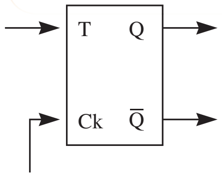


FIGURE 11.24
The T flip-flop.



(a) Block diagram.

T(t)	Q(t)	Q(t + 1)	Condition
0	0	0	No change
0	1	1	
1	0	1	Toggle
1	1	0	

(b) Characteristic table.

selective toggle. If T is 0, the flip-flop does not change its state, and if T is 1, the flip-flop toggles. Implementation of the T flip-flop is an exercise at the end of the chapter.

Excitation Tables

The preceding sections show how to construct the JK, D, and T flip-flops from the SR flip-flop and a few other gates. You can use the same systematic procedure to construct any flip-flop from any other flip-flop. It might seem pointless, for example, to construct a D flip-flop from a JK flip-flop, because you make a JK from an SR with two extra gates in the first place, and the D requires only an SR with an inverter. But the fact that you can construct any flip-flop from any other shows that all the flip-flops are equivalent in power. That is, any processing that you can do with any flip-flop you can do with any other with a few extra gates.

For example, assuming that you have a JK flip-flop, and you want to construct a T flip-flop, you would write the design table from the characteristic table for the T and excitation table for the JK. In general, to construct flip-flop A from flip-flop B, you need the characteristic table for A and the excitation table for B. FIGURE 11.25 shows the excitation tables for the JK, D, and T flip-flops.

You should verify the entries of each excitation table. For example, the first entry in the JK table is for the transition from Q = 0 to Q = 0. Using the same reasoning as with the SR flip-flop, the transition will occur if JK is 00 or 01—hence, the don't-care condition under K(t). The second table entry also has a don't-care condition. The transition from Q = 0 to Q = 1 can occur under two conditions: JK can be either 10, the set condition, or 11, the toggle condition. Both allow Q to change from 0 to 1.

FIGURE 11.25
Excitation tables for the JK, D, and T flip-flops.

Q(t)	Q(t + 1)	J(t)	K(t)
0	0	0	×
0	1	1	×
1	0	×	1
1	1	×	0

(a) The JK flip-flop.

Q(t)	Q(t + 1)	D(t)
0	0	0
0	1	1
1	0	0
1	1	1

(b) The D flip-flop.

Q(t)	Q(t + 1)	T(t)
0	0	0
0	1	1
1	0	1
1	1	0

(c) The T flip-flop.

11.2 Sequential Analysis and Design

A sequential circuit consists of an interconnection of gates and flip-flops. Conceptually, you can group all the gates together into a combinational circuit and all the flip-flops together in a group of *state registers*, as in FIGURE 11.26. This is a generalization of Figure 11.9, which contained only one state register and whose output required no additional gates from the combinational circuit.

The solid arrows in Figure 11.26 represent one or more connecting lines. The input and output lines are the external connections to the environment of the circuit. The lines from the combinational circuit to the state registers are the input lines to SR, JK, D, or T flip-flops. The feedback lines are from the flip-flops' Q and $\overline{Q}$ outputs to the combinational circuit. The figure assumes a common clock line (not shown) to each flip-flop in the group of state registers.

Between clock pulses, the combinational part of the circuit produces its output from the external input and the state of the circuit—that is, the states of the individual flip-flops. The amount of time it takes to produce the combinational output and the state register input depends on how many gate levels are in the circuit. The Ck period is adjusted to be long enough to allow the input to propagate through the combinational circuit to the output before the next clock pulse. All the state registers are edge-triggered or master–slave to protect against multiple propagations through the feedback loop.

As in Figure 11.14, you can describe the behavior of a general sequential circuit with a state transition diagram or its corresponding state transition table. The difference is that Figure 11.14 is for a single device with two possible states, whereas Figure 11.26 is for *n* flip-flops. Because each flip-flop has two possible states, the sequential circuit has a total of 2^n states.

The difference between analysis and design at the hardware level is the same as the difference at the software level. FIGURE 11.27 illustrates the

FIGURE 11.26
A general sequential circuit.

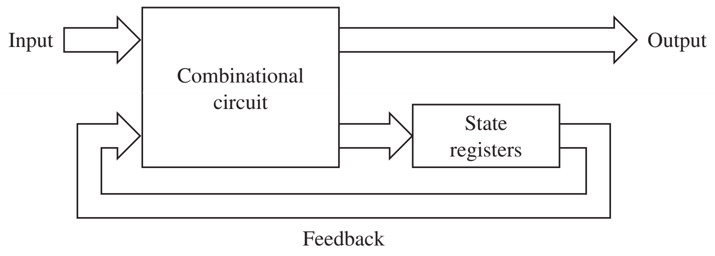
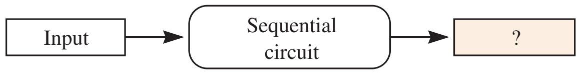
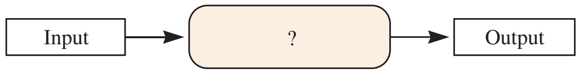


FIGURE 11.27
The difference between analysis and design.



(a) Analysis—The input and sequential circuit are given. The output is to be determined.



(b) Design—The input and desired output are given. The sequential circuit is to be determined.

difference. In analysis, the input and sequential circuit are given and the output is to be determined. In design, the input and desired output are given and the sequential circuit is to be determined.

The next section shows how to determine the output from a given sequential circuit and input stream. The general approach is to construct the analysis table from the circuit. From the analysis table, you can easily determine the state transition table, the state transition diagram, and the output stream for any given input stream.

A Sequential Analysis Problem

Suppose the circuit of FIGURE 11.28 is given. The state registers are the two T flip-flops labeled *FFA* and *FFB*. The combinational circuit is the group of two AND gates and one OR gate. The inputs are X1 and X2, and the single output is Y. The feedback loop consists of the line from Q of FFA, labeled *A*, and the two lines from Q and $\overline{\text{Q}}$ of FFB, labeled *B* and $\overline{B}$. The input to FFA is labeled *TA*, and the input to FFB is labeled *TB*.

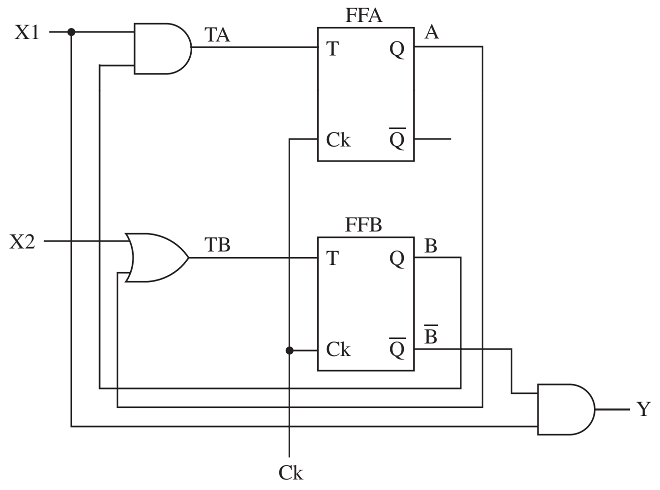
Because there are two flip-flops, there are four possible states,

AB = 00
AB = 01
AB = 10
AB = 11

where, for example, AB = 01 means that Q = 0 in FFA and Q = 1 in FFB. Because there are two inputs, there are four possible input combinations,

X1 X2 = 00
X1 X2 = 01
X1 X2 = 10
X1 X2 = 11

FIGURE 11.28
A circuit for analysis.



The sequential analysis problem

Here is the problem. You are given an initial state, AB, and an initial input, X1 X2. (a) What is the initial output? (b) What will be the next state after a clock pulse occurs? Because there are four states, and with each state there are four possible input combinations, you must answer these questions 16 times. The analysis table in **FIGURE 11.29** provides a systematic tool to determine the answers.

The first four columns are a list of all possible combinations of the initial state and initial input. From Figure 11.28, the Boolean expressions for Y(t), TA(t), and TB(t) are

$$\text{Y}(t) = \text{X1}(t) \cdot \overline{\text{B}}(t)$$
$$\text{TA}(t) = \text{X1}(t) \cdot \text{B}(t)$$
$$\text{TB}(t) = \text{X2}(t) + \text{A}(t)$$

So, you compute the column for Y(t) as the AND of the column for X1(t) and the complement of the column for B(t). You compute the column for TA(t) as the AND of the column for X1(t) and the column for B(t). You compute the column for TB(t) as the OR of the column for X2(t) and the column for A(t). You compute the last two columns from the characteristic table for the T flip-flop, the initial state of a flip-flop, and its initial input.

Example 11.1 Consider the column for B(t + 1). In the first row, the initial state of FFB is 0 from the column for B(t). The flip-flop input is 0 from the column for TB(t). From the characteristic table for the T flip-flop in

FIGURE 11.29
The analysis table for the circuit of Figure 11.28.

A(*t*)	B(*t*)	X1(*t*)	X2(*t*)	Y(*t*)	TA(*t*)	TB(*t*)	A(*t* + 1)	B(*t* + 1)
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	1
0	0	1	0	1	0	0	0	0
0	0	1	1	1	0	1	0	1
0	1	0	0	0	0	0	0	1
0	1	0	1	0	0	1	0	0
0	1	1	0	0	1	0	1	1
0	1	1	1	0	1	1	1	0
1	0	0	0	0	0	1	1	1
1	0	0	1	0	0	1	1	1
1	0	1	0	1	0	1	1	1
1	0	1	1	1	0	1	1	1
1	1	0	0	0	0	1	1	0
1	1	0	1	0	0	1	1	0
1	1	1	0	0	1	1	0	0
1	1	1	1	0	1	1	0	0

Figure 11.24(b), 0 is the no-change condition. So the state remains the same, and B($t + 1$) is 0. ■

Example 11.2 For the same column, consider the second row. The initial state of FFB is again 0 from the column for B(t). This time the flip-flop input is 1 from the column for TB(t). From the characteristic table for the T flip-flop, 1 is the toggle condition. So the state toggles, and B($t + 1$) is 1. ■

The state transition table in **FIGURE 11.30** is a simple rearrangement of selected columns from the analysis table. For a given initial state A(t) B(t) and a given input X1(t) X2(t), it lists the next state A($t + 1$) B($t + 1$) and the initial

FIGURE 11.30
The state transition table for the circuit of Figure 11.28.

A(*t*) B(*t*)	X1(*t*) X2(*t*) 00	01	10	11
00	00, 0	01, 0	00, 1	01, 1
01	01, 0	00, 0	11, 0	10, 0
10	11, 0	11, 0	11, 1	11, 1
11	10, 0	10, 0	00, 0	00, 0
	A(*t* + 1) B(*t* + 1), Y(*t*)			

output Y(*t*). States are listed as ordered pairs. Entries in the body of the table are the next state followed by the initial output, separated by a comma.

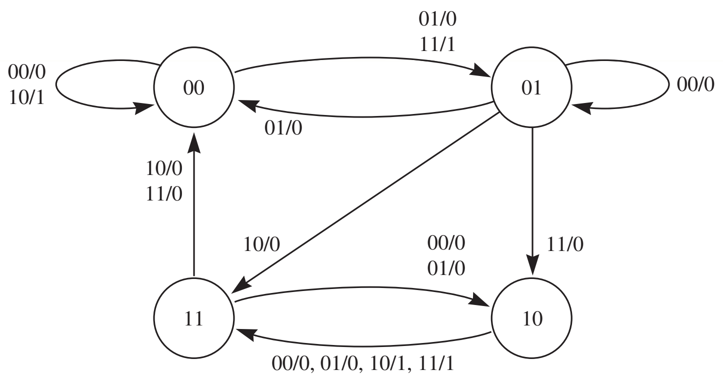
It is usually easier to visualize the behavior of the circuit from its state transition diagram rather than its state transition table. FIGURE 11.31 is the state transition diagram constructed from the state transition table. The standard convention is to label transitions as ordered pairs of the input followed by the initial output, separated by a slash.

Determining the output stream from the input stream

To determine the output from a given input stream, assume that you start in state AB = 11 and input the following values of X1 X2:

11, 11, 00, 10, 01

FIGURE 11.31
The state transition diagram for the circuit of Figure 11.28. Transitions are labeled X1(*t*) X2(*t*) / Y(*t*).



From the diagram, you will make transitions to the states

11, 00, 01, 01, 11, 10

and generate the following output

0, 1, 0, 0, 0

The analysis is similar for sequential circuits containing other flip-flops or even mixtures of different types of flip-flops. Using your knowledge of combinational circuits, you simply determine the input of each flip-flop for all possible combinations of inputs and states. Then, from the characteristic table of each flip-flop, you determine what the next state will be. In general, if there are *m* inputs and *n* flip-flops, you will need to analyze 2^m2^n transitions.

Preset and Clear

The output sequence for the previous problem assumes that the flip-flops are both in their Q = 1 state. A legitimate question is: How do the flip-flops get in their start states? In practice, most flip-flops are constructed with two additional inputs, called *preset* and *clear*. These inputs are *asynchronous*; that is, they do not depend on the clock pulse to function. FIGURE 11.32 shows the block diagram of an SR flip-flop with asynchronous preset and clear inputs.

In normal operation, both preset and clear lines are 0. To initialize the flip-flop to Q = 1, send preset to 1 and back to 0. To initialize the flip-flop to Q = 0, send clear to 1 and back to 0. You do not need to send a clock pulse while either input is high for it to function. Implementation of the asynchronous preset and clear is an exercise at the end of the chapter.

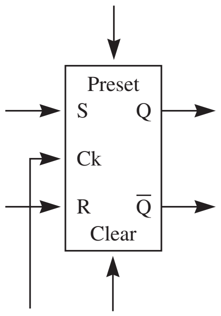
Sequential Design

With sequential design, the behavior of the circuit is given, frequently in the form of a state transition diagram, and the implementation of the circuit with a minimum number of gates is to be determined. Also given in the problem formulation is the type of flip-flop to be used for the state registers in the sequential circuit.

The design procedure has three steps. First, from the state transition diagram, tabulate the transitions of the circuit in a design table. For each combination of initial state and input, list the initial output and the next state. Then, from the excitation tables, list the necessary flip-flop input conditions to cause the transitions.

Second, transfer the entries for each flip-flop input to a Karnaugh map. It helps to prevent mistakes if you look ahead in your tabulation of the design table and list the entries in the same order as a Karnaugh map. Design the

FIGURE 11.32
The block diagram of an SR flip-flop with asynchronous preset and clear.



combinational part of the sequential circuit by minimizing the expressions from the Karnaugh map.

Third, draw the minimized combinational circuit. Each flip-flop input will come from a combinational circuit two gate levels deep at most. You should recognize this procedure as a generalization of the procedure to construct the JK, D, and T flip-flops from the SR flip-flop.

A Sequential Design Problem

This example illustrates the design procedure. The problem is to implement the state transition diagram of FIGURE 11.33 with SR flip-flops. As in Figure 11.31, the transitions are labeled with the input values, X1 X2, and the initial output, Y. The values in the state circles are the Q values of the first SR flip-flop, FFA, and the second SR flip-flop, FFB.

This machine has only three input combinations and four states, for a total of 12 transitions. The input combinations are 01, 11, and 10. Because the combination 00 is not expected to occur, you can treat it as a don't-care condition to help with the minimization.

To implement a finite-state machine with eight states, you need three flip-flops. To implement a machine with five to seven states also requires three flip-flops, but some of the states are not expected to occur and can be treated as don't-care states.

FIGURE 11.34 is the first step in the design process. The four columns on the left are all the possible combinations of initial state and input. The middle three columns are a simple tabulation of the initial output and next state from Figure 11.33.

FIGURE 11.33
The state transition diagram for a design problem.

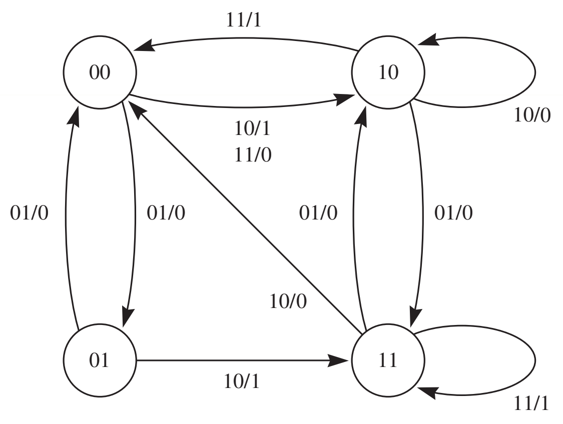


FIGURE 11.34
The design table for the state transition diagram of Figure 11.33.

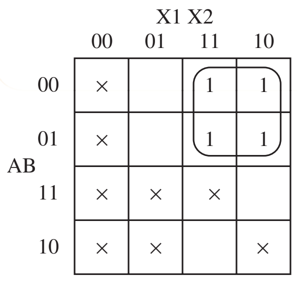
Initial State		Initial Input		Initial Output	Next State		Flip-Flop Input Conditions			
							FFA		FFB	
A(*t*)	B(*t*)	X1(*t*)	X2(*t*)	Y(*t*)	A(*t* + 1)	B(*t* + 1)	SA(*t*)	RA(*t*)	SB(*t*)	RB(*t*)
0	0	0	1	0	0	1	0	×	1	0
0	1	0	1	0	0	0	0	×	0	1
1	1	0	1	0	1	0	×	0	0	1
1	0	0	1	0	1	1	×	0	1	0
0	0	1	1	0	1	0	1	0	0	×
0	1	1	1	0	1	1	1	0	×	0
1	1	1	1	1	1	1	×	0	×	0
1	0	1	1	1	0	0	0	1	0	×
0	0	1	0	1	1	0	1	0	0	×
0	1	1	0	1	1	1	1	0	×	0
1	1	1	0	0	0	0	0	1	0	1
1	0	1	0	0	1	0	×	0	0	×

The four columns on the right come from the fact that there are two SR flip-flops, each with two inputs. SA is the S input of FFA, RA is the R input of FFA, and so on. The flip-flop input conditions that produce the given transition come from the excitation table for the SR flip-flop, Figure 11.15.

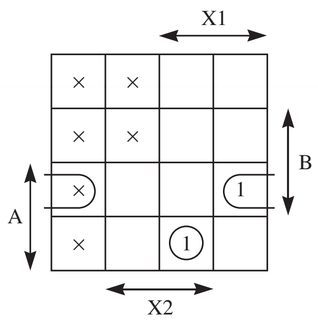
For example, consider the first line for the transition from AB = 00 to AB = 01. The transition for FFA is from Q = 0 to Q = 0. The excitation table shows that transition is caused by a 0 for S and a don't-care condition for R. The transition for FFB is from Q = 0 to Q = 1. The excitation table shows that transition is caused by 10 for SR.

The next step is to consider that each flip-flop input is a function of four variables—the initial state, AB, and the input, X1 X2. To design the combinational circuit requires a four-variable Karnaugh map for each flip-flop input. FIGURE 11.35 shows the Karnaugh maps from Figure 11.34. The map for input SA, Figure 11.35(a), shows the row values for the state

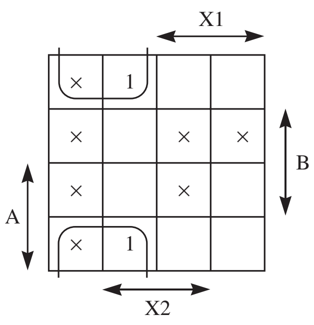
FIGURE 11.35
The Karnaugh maps for Figure 11.34.



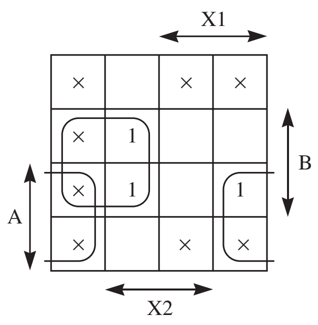
(a) SA = $\bar{A}$ X1



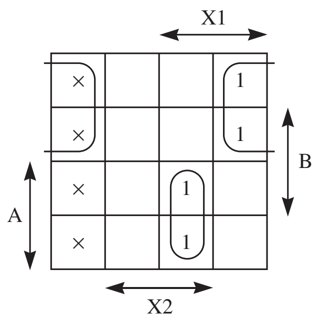
(b) RA = A B $\overline{X2}$ + A $\bar{B}$ X1 X2



(c) SB = $\bar{B}$ $\overline{X1}$



(d) RB = B $\overline{X1}$ + A $\overline{X2}$



(e) Y = $\bar{A}$ $\overline{X2}$ + A X1 X2

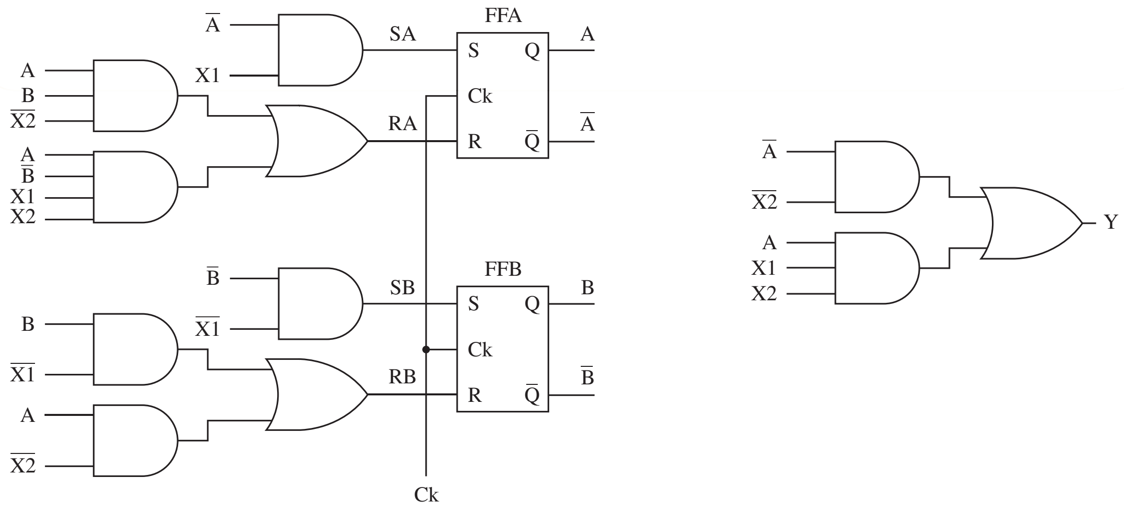
AB and the column values for the input X1 X2. Note that the combination X1 X2 = 00; the first column in the Karnaugh maps is a don't-care condition.

The output, Y, is also a function of the initial state and input and requires a Karnaugh map for minimization. Shown below each Karnaugh map is the corresponding minimized expression.

FIGURE 11.36 is the resulting sequential circuit. Rather than show the feedback connections explicitly, the diagram is in abbreviated form.

After completing the design, you might notice in Figure 11.35(c) and (d) that cell 8 of the Karnaugh map is covered and so appears to have the value 1. How can that be when Figure 11.35(a) and (b) do not have it covered and so it appears to have the value 0? How can it have the value 0 and 1 at the same time? Karnaugh maps do not show what happens in practice to the don't-care conditions. The specification of the circuit in Figure 11.33 assumes that the external input X1 X2 comes from some unknown source and that the combination X1 X2 = 00 will never occur in practice. Therefore, the combination represented by cell 8—namely, A B X1 X2 = 1000—will

FIGURE 11.36
The minimized sequential network for Figure 11.33.



never occur. You can choose to have your circuit behave any way you want for that combination, because it will never occur.

The same consideration arises if you need to design a machine with the number of states not equal to a power of 2. Say you want to design a five-state machine. Two flip-flops are not enough, because they provide only four states. With three flip-flops, you have eight states possible, but only five will occur in practice. The other three can be don't-care conditions in the design. After you design such a circuit, the unused states will be inaccessible from the used states, but not necessarily vice versa. That is, there may be transitions from the unused to the used states, but such transitions will be irrelevant because they will never occur. They are analogous to dead code in a program.

The design procedure illustrated in this example has two flip-flops and two inputs, for a total of four variables in the Karnaugh map. Four-variable maps would also be required for a design with three flip-flops and one input, or one flip-flop and three inputs. The procedure for one flip-flop and two inputs, or two flip-flops and one input, would be identical except that three-variable Karnaugh maps would suffice for the minimization.

Some sequential circuits require minimization of a combinational circuit with more than four variables. Karnaugh maps cannot conveniently handle a problem of that size. Systematic procedures exist to deal with these larger problems, of which the most common is the Quine-McCluskey algorithm, but those are beyond the scope of this text.

11.3 Computer Subsystems

Computers are designed as a set of interconnected subsystems. Each subsystem is a black box with a well-specified interface. Sometimes the subsystem consists of an individual integrated circuit, in which case the interface is specified by the operating characteristics of the wire pins of the physical package. At a lower level of abstraction, the subsystem could be part of several subsystems within an integrated circuit. Or, at a higher level, the subsystem could be a printed circuit board made up of several integrated circuits.

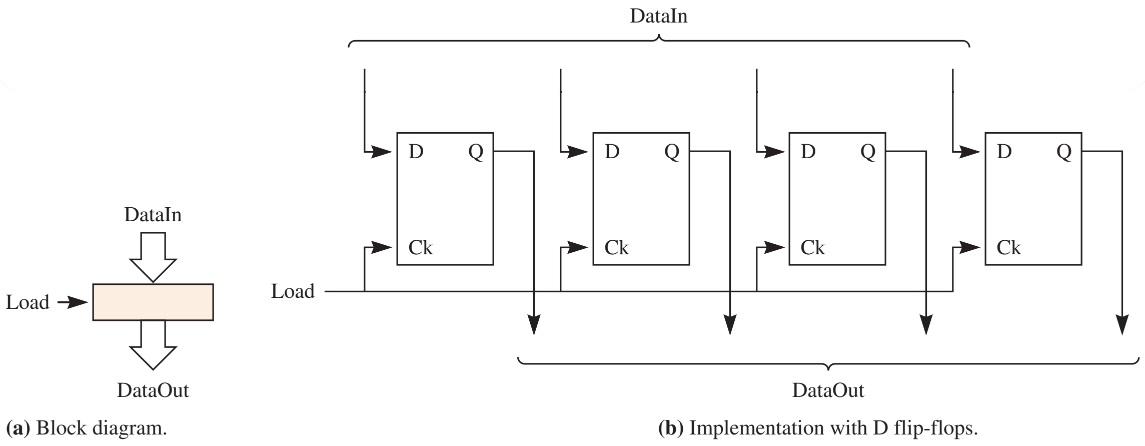
However the subsystem is physically implemented, a bus connects the subsystems together. The bus could be a single wire from the output of a gate of one subsystem to the input of a gate of another subsystem. Or the bus could be a group of wires containing both data and control signals, like the bus that connects main memory to the CPU of Pep/9 shown in Figure 4.1.

Registers

A basic building block of the Level ISA3 machine is the register. You are familiar with the 16-bit registers in the Pep/9 CPU. The instruction set includes instructions to manipulate the register contents. FIGURE 11.37 shows a block diagram and implementation of a 4-bit register.

The register has four data inputs, four data outputs, and a control input that is labeled *Load*. The block diagram shows the four data lines as a single

FIGURE 11.37
A four-bit register.



wide arrow. The implementation shows the data lines individually. The register is simply an array of D flip-flops. Each data input line connects to the D input of a flip-flop. Each data output line connects to the Q output of a flip-flop. The load input connects to the clock inputs of all the flip-flops. From this point on, figures that contain sequential circuits are shaded to distinguish them from combinational circuits.

The register operates by presenting the values you wish to load on the data input lines. You then clock them into the register by sending the load signal to 1 and back to 0. All four values are loaded simultaneously into the register. If each flip-flop is a master–slave device, the output appears when the slave is connected to the master at time t_4 of Figure 11.11.

Each D flip-flop is a bit in the register. An 8-bit register would have 8 flip-flops, and a 16-bit register would have 16 flip-flops. The number of flip-flops would not affect the speed of the load operation because the load into each flip-flop occurs simultaneously. The block diagram of Figure 11.37(a) is the same regardless of the number of bits in the register.

Buses

Suppose you have two subsystems, A and B, that need to send data back and forth. The simplest way to connect them is to have two unidirectional buses—one for sending data from A to B and one from B to A. The first bus would be a group of wires, each one from the output of a gate in A to the input of a gate in B. Each wire from the second would be from the output of a gate in B to the input of a gate in A.

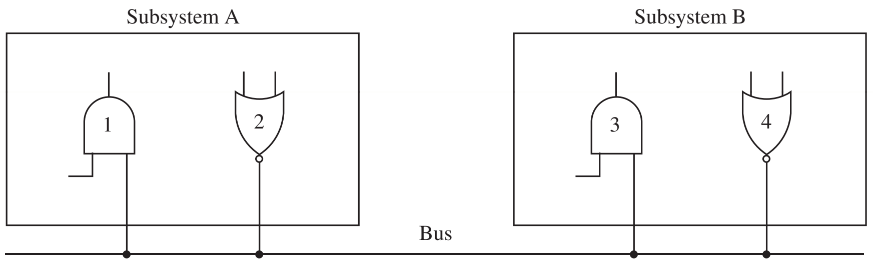
Unidirectional buses

One problem with this arrangement is the sheer number of wires for a wide bus. If you want to send 64 bits at a time, you need two unidirectional buses, for a total of 128 wires. You can cut that number in half by using a bidirectional bus. The tradeoff is speed. With two unidirectional buses, you can send information from A to B and B to A at the same time, an impossibility with a bidirectional bus. You also must pay the price of a small setup time if you need to change the direction of the data flow on the bus before sending the data.

Bidirectional buses

FIGURE 11.38 shows the problem that must be solved to implement a bidirectional bus. An AND gate represents the master clock-enable gate of a master–slave flip-flop that is part of a register. A NOR gate represents the slave Q output gate of the flip-flop of another register. To send data from A to B, the output of gate 2 must be connected to the input of gate 3. Going from B to A, the output of gate 4 must be connected to the input of gate 1. The problem is with gates 2 and 4. You can always connect the output of one gate to the input of another, but you cannot connect the output of two gates together. Suppose gate 2 wants to send a 1 to gate 3, but the output of gate

FIGURE 11.38
The bidirectional bus problem.



4 happens to be Ø at the time. Their outputs are in conflict, so which one predominates?

The answer depends on the technology underlying the fabrication of the gates. With some logic gate families, you can actually connect the outputs of several gates together and if one or more gates output a 1, the common bus will transfer a 1. This kind of gate is said to have the *wired-OR property* because the signal on the bus acts like the output of an OR gate. With other families, connecting the outputs of two gates together clobbers the circuit, causing unpredictable havoc. Even with wired-OR gates, the bidirectional bus problem still exists. For example, if gate 2 wants to send a Ø to gate 3, but the output of gate 4 happens to be 1 at the time, gate 3 would erroneously detect a 1.

The wired-OR property

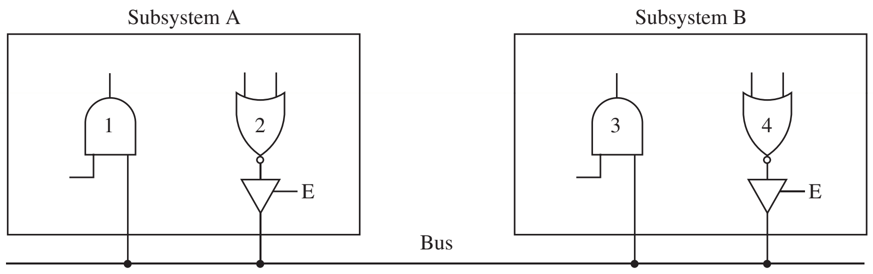
To make the bidirectional bus work properly, you need a way to temporarily disconnect gate 4 from the bus when gate 2 is putting data on the bus, and vice versa. A tri-state buffer can do precisely that. It has one data input, one enable control input, and one output. FIGURE 11.39 shows

Tri-state buffers

FIGURE 11.39
Truth table for the tri-state buffer.

E	*a*	*x*
0	0	Disconnected
0	1	Disconnected
1	0	0
1	1	1

FIGURE 11.40
Solution to the bidirectional bus problem with the tri-state buffer.



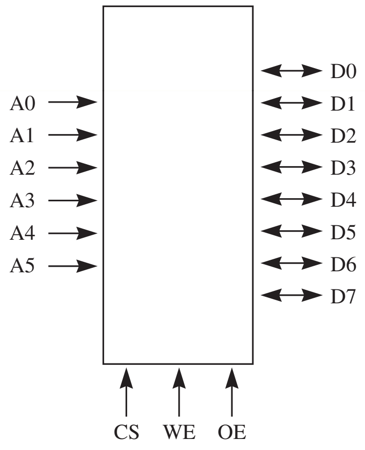
the truth table for the tri-state buffer where E is the enable control line, *a* is the input, and *x* is the output. When the device is enabled, the input passes through to the output unmodified. When it is disabled, the output is in effect disconnected from the circuit. Electrically, the output is in a high-impedance state. The device is called a *tri-state buffer* because the output can be in one of three states—0, 1, or disconnected.

FIGURE 11.40 shows how the tri-state buffer solves the bidirectional bus problem. There is a tri-state buffer between the output of every gate and the bus. To send data from A to B, you enable the tri-state buffers in A and disable the tri-state buffers in B, and vice versa. In order for this scheme to function properly, the subsystems must cooperate and never let their tri-state buffers be enabled simultaneously.

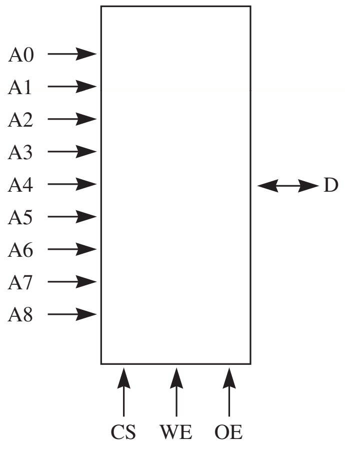
Memory Subsystems

Memory subsystems are constructed from several integrated circuit memory chips. FIGURE 11.41 shows two memory chips, each one of which stores 512 bits. Memory chips have a set of address lines labeled starting with *A0*, a set of data lines labeled starting with *D0*, and a set of control lines labeled *CS*, *WE*, and *OE*. The data lines have two arrowheads to indicate that they should be connected to a bidirectional bus. Part (a) shows the bits organized as a set of 64 eight-bit words. There are six address lines, because 2^6 is 64. Each possible combination of input values accesses a separate eight-bit word. Part (b) shows the same number of bits organized as a set of 512 one-bit words. There are 9 address lines because 2^9 is 512. In general, a memory chip with 2^n words has n address lines. The number of address and data lines of the chips in Figure 11.41 is unrealistically tiny to keep the examples simple. Memory chips are manufactured nowadays with billions of bits.

FIGURE 11.41
Two integrated circuit memory chips that store 512 bits.



(a) 64 × 8 memory chip.



(b) 512 × 1 memory chip.

The control lines serve the following purpose:

Memory chip control lines

- CS (chip select) to enable or select the memory chip
- WE (write enable) to write or store a memory word to the chip
- OE (output enable) to enable the output buffer to read a word from the chip

Writing to a memory chip

Reading from a memory chip

In practice, control lines are normally asserted low.

To store a word to a chip, set the address lines to the address where the word is to be stored, set the data lines to the value that you want to store, select the chip by setting CS to 1, and execute the write by setting WE to 1. To read a word from a chip, set the address lines to the address from which you want to read, select the chip by setting CS to 1, and enable the output by setting OE to 1, after which the data you want to read will appear on the data lines. In practice, the control lines for most memory chips are asserted low. That is, they are normally maintained at a high voltage, representing 1, and are activated by setting them to a low voltage, representing Ø. This text assumes that memory chip control lines are asserted high to keep the examples simple.

FIGURE 11.42 shows the implementation of a 4 × 2 memory chip with two address lines and two data lines. It stores four two-bit words. Each bit is a D flip-flop. The sequential devices are shaded to distinguish them from the combinational devices. The address lines drive a 2 × 4 decoder, one of whose

FIGURE 11.42
A 4 × 2 memory chip.

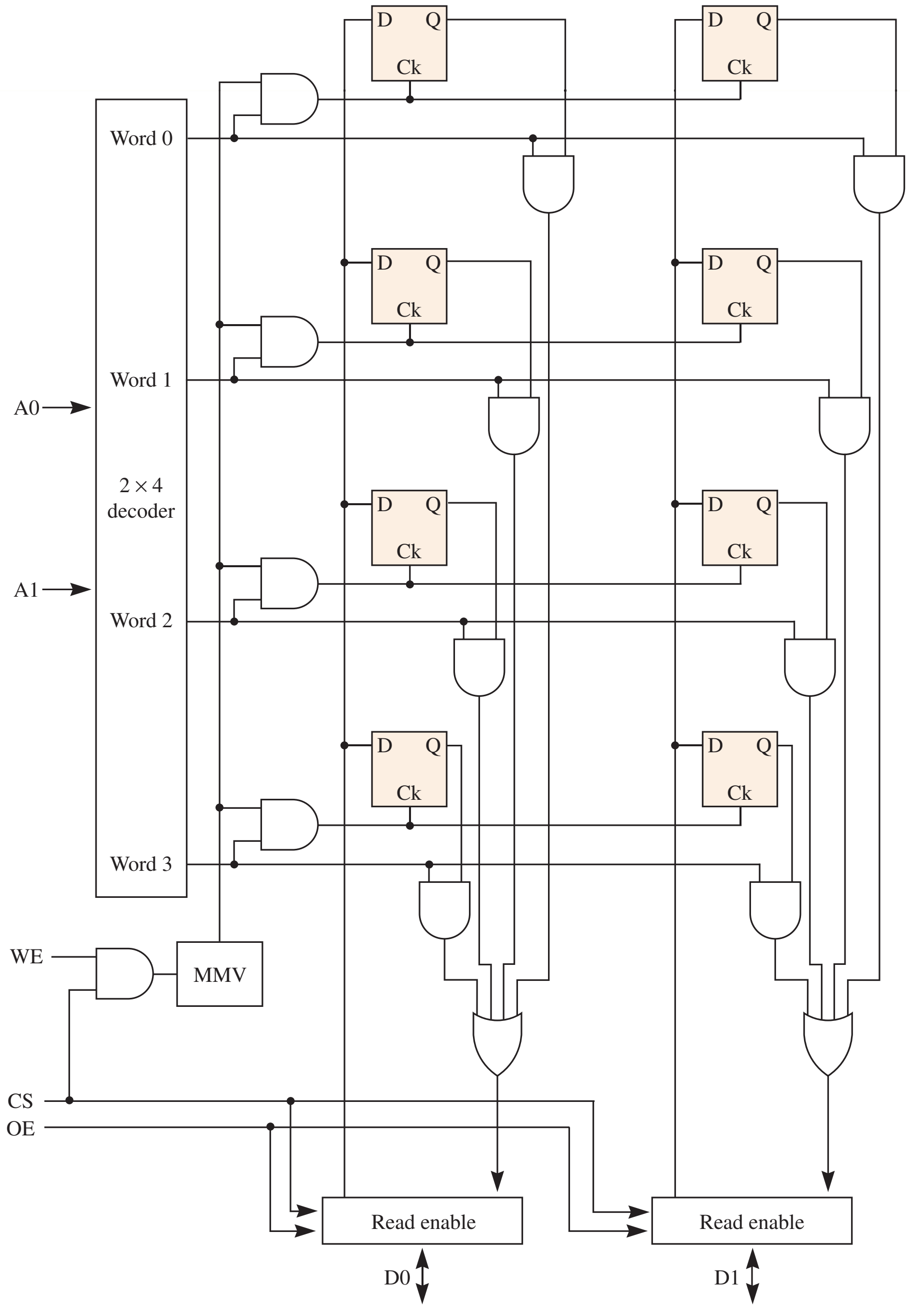
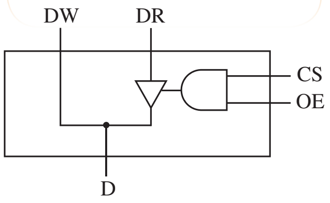


FIGURE 11.43 Implementation of the box labeled *Read enable* in Figure 11.42.



outputs is 1 and the other three 0. The decoder output line that is 1 selects the row of D flip-flops that make up the word accessed by the chip.

The box labeled *Read enable* provides the interface to a bidirectional bus. FIGURE 11.43 shows its implementation. DR is the data read line coming from the OR gate in Figure 11.42, DW is the data write line going to the D inputs of the flip-flops, and D is the data interface to the bidirectional bus.

FIGURE 11.44 is a truth table of the circuit. The chip is normally in one of three modes.

- CS = 0: The chip is not selected.
- CS = 1, WE = 1, OE = 0: The chip is selected for write.
- CS = 1, WE = 0, OE = 1: The chip is selected for read.

It is not permitted for WE and OE to both be 1 at the same time. The truth table and its implementation show that when CS is 0, DR is disconnected from the bidirectional bus regardless of any of the other control lines. When CS is 1 and OE is 0, DR is also disconnected. This is the write mode, in which case data is being fed from the bidirectional bus to DW. When CS is 1 and OE is 1, the tri-state buffer is enabled and data is being fed from DR to the bidirectional bus.

The memory read operation

To see how a *memory read works*, consider the following scenario, where A1 A0 = 10, CS = 1, WE = 0, and OE = 1. The values for A1 A0 make the line labeled Word 2 out of the decoder 1, and the other word lines 0. The *Word 2* line enables the AND gates connected to the Q outputs of the D flip-flops in row 2, and disables the AND gates connected to the flip-flop outputs of all the other rows. Consequently, data from the second row flows through the two OR gates into the Read enable box and onto the bidirectional bus.

The monostable multivibrator

A memory write works in conjunction with the box labeled *MMV*, which stands for *monostable multivibrator*, in Figure 11.42. Assuming that the D flip-flops are of the master–slave variety, to do a store requires a Ck pulse to go from low to high then high to low, as Figure 11.11 shows. A monostable multivibrator is a device that provides such a pulse. FIGURE 11.45 shows the timing diagram of a monostable multivibrator with an initial delay. When the input line goes high, it triggers a delay circuit. After a predetermined time interval, the delay circuit triggers the monostable multivibrator, which emits a clock pulse with a predetermined width. Monostable multivibrators are also known as *one-shot devices* because when they are activated they emit a single "one shot" pulse.

The memory write operation

To see how a memory write works, consider the following scenario, where A1 A0 = 10, CS = 1, WE = 1, and OE = 0. Assuming that the address lines, data lines, and control lines are all set simultaneously, the memory circuit must wait for the address signals to propagate through the decoder before clocking the data into the flip-flops. The initial delay in the monostable multivibrator is engineered to allow sufficient time for the outputs of the decoder to be set before clocking in the data. The Read enable circuit puts the data from the bidirectional bus on the input of all the flip-flops. However, when the monostable multivibrator emits the clock pulse, three of the four AND gates to which it is connected will disable the pulse from reaching their rows. It will only reach the row of Word 2, so those are the only flip-flops that will store the data.

FIGURE 11.44
Truth table for the Read enable box.

CS	OE	Operation
0	×	Disconnected
1	0	Disconnected
1	1	Connect DR to D

Several types of memory chips are available on the market. The circuit model in Figure 11.42 most closely resembles what is known as *static memory*, or *SRAM*. In practice, a master–slave D flip-flop is not the basis of bit storage, as it requires more transistors than are necessary. Many static RAM devices use a circuit that is a modification of Figure 11.1(b), a stable circuit consisting of a pair of inverters with feedback. It takes only two additional transistors to implement a mechanism for setting the state. The advantage of static RAM is speed. The disadvantage is its physical size on the chip, because several transistors are required for each bit cell.

SRAM

To overcome the size disadvantage of static memory, *dynamic memory*, or *DRAM*, uses only one transistor and one capacitor per bit cell. You store data by storing electrical charge on the capacitor. Because of the small size of the bit cells, DRAM chips have much higher storage capacities than SRAM chips. The problem with DRAM is that the charge slowly leaks off the capacitors within a few milliseconds of being fully charged. Before too much charge leaks off, the memory subsystem must read the data from the cell and charge the capacitor back up if necessary. As you would expect, the refresh operation takes time, and DRAM memory is slower than SRAM.

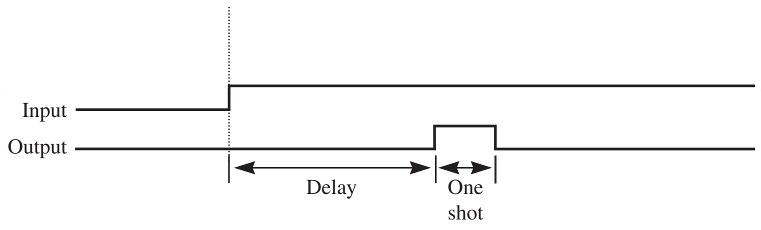
DRAM

In contrast to read/write memory, *read-only memory*, or *ROM*, is designed for situations where the data to be stored never changes. The data for each bit cell can be set at the factory when the chip is manufactured, in which case the user supplies the manufacturer with the bit pattern to store. Alternatively, a *programmable ROM*, or *PROM*, chip allows the user to

ROM

PROM

FIGURE 11.45
Timing diagram of a monostable multivibrator with initial delay.



program the bit pattern. The process is accomplished by selectively blowing out a set of fuses embedded in the chip and is irreversible. To overcome the irreversibility disadvantage, an *erasable PROM*, or *EPROM*, has the capability to erase the entire chip by exposing the circuit to ultraviolet radiation. EPROM chips, are packaged underneath a transparent window so the circuit can be exposed to the radiation. To erase an EPROM chip, you must remove it from the computer to expose it and then you must reprogram the entire chip. The *electrically erasable PROM*, or *EEPROM*, allows you to erase an individual cell with the right combination of electrical signals so that the device does not need to be removed to be reprogrammed. The circuitry to program a cell uses different voltage levels from those to read data during normal operation of the chip, and is therefore more complex to design.

EPROM

EEPROM

SRAM and DRAM are volatile. That is, when you power off the circuit you lose the data. ROM devices are nonvolatile because they retain their data without a source of external power. *Flash memory* is popular in consumer handheld devices. It is a type of EEPROM and has the advantage of retaining its data when the device is turned off. With flash memory, you can read an individual cell, but you can only write an entire block of cells. Before writing the block, it must be completely erased. A flash card consists of an array of flash chips. A solid-state hard drive is the same thing, but with circuitry at the interface to make it appear to be a hard drive. It is not really a hard drive and has no moving parts. Compared to hard drives of the same size, flash drives are faster but hold much less data. An indication of how micro hard drive and flash memory technologies compete in the market is that manufacturers offer hard drives in a package whose interface makes them appear to be flash memory. You can plug them into your digital camera in place of the memory card. So now we have flash memory pretending to be a hard drive and a hard drive pretending to be a flash memory card, a testament to the practical power of abstraction.

Flash memory

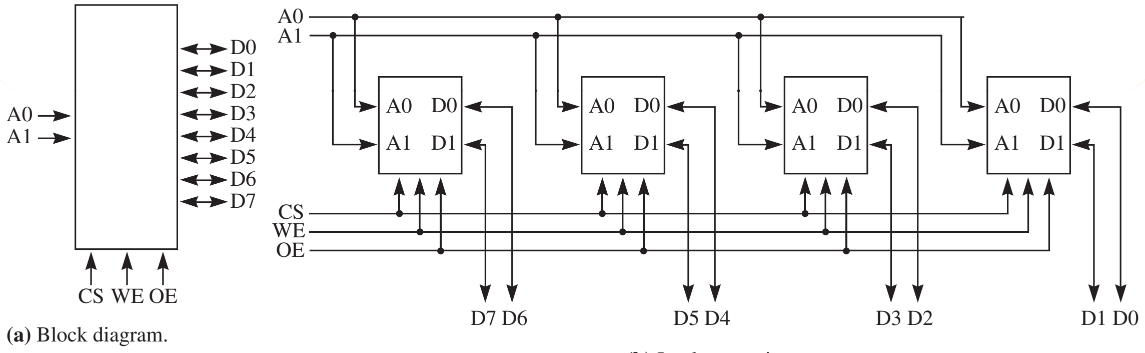
Address Decoding

A single memory chip sometimes does not have the capacity to provide main memory storage for an entire computer. You must combine several chips into a memory subsystem to provide adequate capacity. Most computers are byte addressable, as is Pep/9. A chip like the one in Figure 11.41(a) would be convenient for such a machine because the word size of the chip matches the cell size that the CPU addresses.

Suppose, however, that you have a set of 4×2 chips like the one in Figure 11.42 and you want to use it in Pep/9. Because the word size of the chip is 2 and the size of an addressable cell for the CPU is 8, you must group four 4×2 chips to construct a 4×8 memory module. FIGURE 11.46 shows the interconnections. You can see that the input and output lines of the

FIGURE 11.46

Constructing a 4 × 8 memory module from four 4 × 2 memory chips.



(b) Implementation.

module are identical to the input and output lines of what would be a 4 × 8 chip. The bits of each byte in memory are distributed over four chips. The bits of the byte at address A1 A0 = 01 are stored in the second row (Word 1) of all four chips.

Similarly, it would take eight of the chips in Figure 11.41(b) to construct a 512 × 8 memory module. For high reliability, you could use 11 of the chips for each eight-bit cell with the three extra chips used for single-error correction, as described in Section 9.4. With such an ECC system, the bits of each byte would be spread out over all 11 chips.

These examples show how to combine several $n \times m$ chips to make an $n \times k$ module where k is greater than m. In general, k must be a multiple of m. You simply hook up k/m chips, with all their address and control lines in common, and assign the data lines from each chip to the lines of the module.

A different problem in constructing memory subsystems is when you have several $n \times m$ chips, m is equal to the size of the addressable cell for the CPU, and you want an $l \times m$ module where l is greater than n. In other words, if you have a set of chips whose word size is equal to the size of the addressable cell of the CPU, how do you connect them to add memory to your computer? The key is to use the chip select line CS so that for all address requests from the CPU, no more than one chip is selected. The technique for connecting a memory chip to an address bus is called *address decoding*. There are two variations—full address decoding and partial address decoding.

Address decoding

FIGURE 11.47 shows the memory map for a CPU with eight address lines capable of storing 2^8, or 256, bytes. The scenario is unrealistically small to keep the example simple. You have four chips that you need to wire into the address space of the CPU—a 64-byte RAM to install at address 0, a

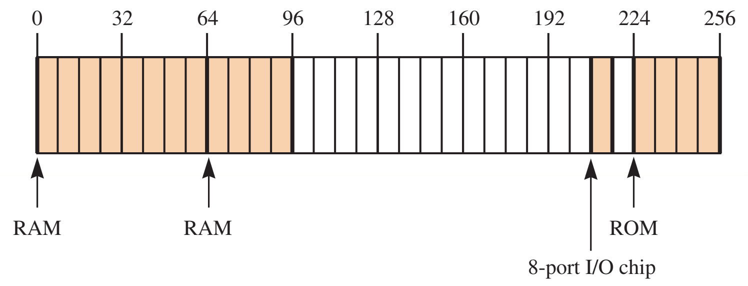


FIGURE 11.47
The memory map of a 256-byte memory with eight address lines.

32-byte RAM to install at address 64, an eight-port memory-mapped I/O chip to install at address 208, and a 32-byte ROM to install at address 224.

Memory-mapped input/output

The eight-port memory-mapped I/O chip corresponds to the input and output devices at `charIn` and `charOut` in Pep/9. In practice, each I/O device communicates through memory with a control register and a data register. For example, in Figure 11.47 the keyboard's control register might be at address 000 in the chip and its data register at address 001. If the chip is wired into the memory map at address 208, then the CPU will see the keyboard's control register at address 208 and its data register at address 209. It might `LDBA` the control register from 208 to detect that a key has been pressed, and if so, then `LDBA` from 209 to get the ASCII character from the keyboard buffer. For memory-mapped I/O to function, the system requires circuitry to detect when loads and stores are done to any addresses to which I/O devices are mapped. Detection of such events activates the circuitry necessary to control the I/O devices.

You determine how to connect the chips to the address bus with the help of the table in FIGURE 11.48. For each chip, write down in binary the minimum address (that is, the address of the starting byte of the chip) and the maximum address (that is, the address of the last byte of the chip). Comparing these two bit patterns, you determine the general form of the address range for which each chip is responsible. For example, the general address of the eight-port I/O chip is 1101 0xxx, which means that it is

FIGURE 11.48
A table for address decoding the memory map of Figure 11.47.

Device	64 × 8 RAM	32 × 8 RAM	8-port I/O	32 × 8 ROM
Minimum address	0000 0000	0100 0000	1101 0000	1110 0000
Maximum address	0011 1111	0101 1111	1101 0111	1111 1111
General address	00xx xxxx	010x xxxx	1101 0xxx	111x xxxx

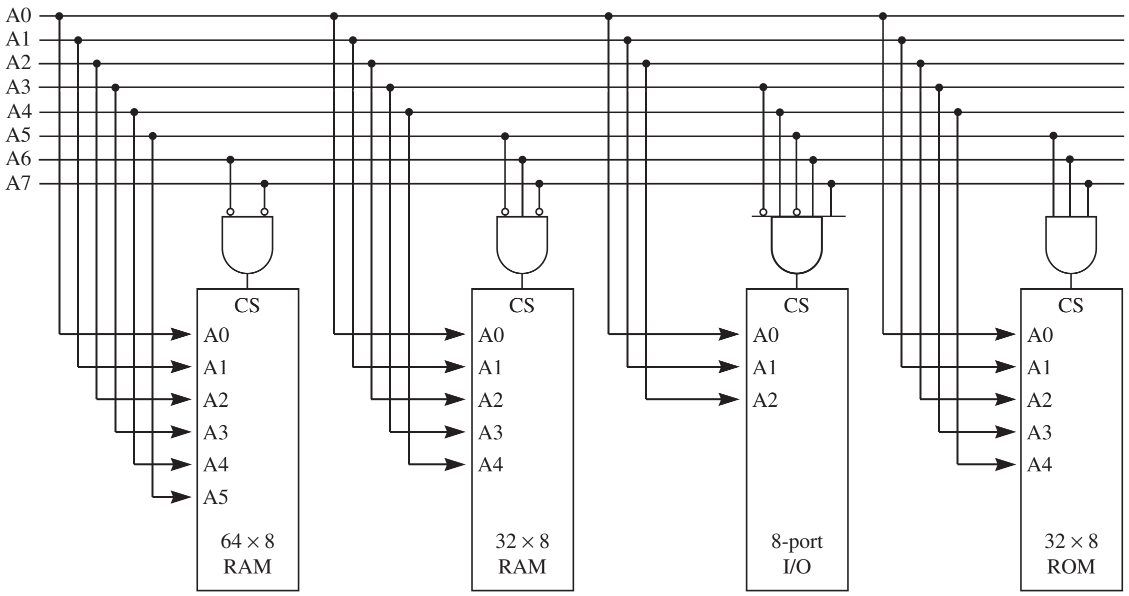
responsible for the range of addresses from 1101 0000 to 1101 0111. Each letter x can be 0 or 1. Consequently, the eight-port I/O chip must be selected when, and only when, the first five digits are 11010.

FIGURE 11.49 shows the chips wired to the address bus with full address decoding. The three address lines of the eight-port I/O chip connect to the three least significant address bus lines. The most significant five address lines feed into the chip select through a pair of inverters and an AND gate. (The inverters are abbreviated in the figure and are shown as the inverted inputs to the AND gate.) You can see from the circuit that the chip select line of the eight-port I/O chip will be 1 if and only if the first five bits of the address on the bus are 11010.

To keep the figure uncluttered, it does not show the data lines of the chips. The data lines of the RAM and ROM chips all connect to an eight-bit bidirectional data bus. Also not shown are the control inputs, WE and OE, which are connected to the common WE and OE lines of the memory module.

Partial address decoding is possible if you know that your memory subsystem will never be expanded by adding more memory. A typical situation would be a small computer-controlled appliance that is not user-upgradeable. The idea is to reduce the number of gates in the decoding circuits to the bare minimum needed for the system to access the devices.

FIGURE 11.49
Full address decoding for the memory map of Figure 11.47.



The minimization technique is to write the general addresses of the chips, one per row, and inspect the columns. For the chips in Figure 11.48, the general addresses are

0$\underline{0}$xx xxxx, 64 × 8 RAM
$\underline{01}$0x xxxx, 32 × 8 RAM
$\underline{110}$1 0xxx, 8-port I/O chip
$\underline{111}$x xxxx, 32 × 8 ROM

Consider the first chip, and inspect the columns of the general addresses to see how you can uniquely determine the first chip with the smallest amount of information. Note that the second column, corresponding to address line A6, is 0 for the first chip and 1 for all the other chips. Therefore, you can select the first chip if A6 is 0 regardless of the value of A7.

Now consider the second chip. With full address decoding, you must test three address lines—A7, A6, and A5. Can you manage by testing only two? For example, could you test A7 A5 = 00? No, because A7 A6 A5 = 000 will select the first chip, and so both chips would be selected simultaneously. Could you test A6 A5 = 10? No, because A7 A6 A5 A4 A3 = 11010 selects the eight-port I/O chip, and again you would have a conflict. However, by inspection of the columns, none of the other chips have 01 as their first two bits. So you can test for A7 A6 = 01 to select the second chip.

Similar reasoning shows that you can select the third chip by testing A7 A6 A5 = 110 and the fourth chip by testing A7 A6 A5 = 111. FIGURE 11.50 shows the final result of the minimization. Compared to Figure 11.49, we have eliminated one two-input AND gate and three inverters, and decreased the number of inputs of one AND gate from three to two and of another from five to three.

The question naturally arises as to the difference in behavior between the memory modules of Figures 11.49 and 11.50. There is no difference when the CPU accesses one of the shaded regions in the memory map of Figure 11.47. With full address decoding, if the CPU accesses an address outside the shaded areas, no chip will be selected and the data on the data bus is unpredictable. However, with partial address decoding, the CPU might access a chip.

Consider the 64 × 8 RAM, which has a general address of 00xx xxxx, but is selected if A6 = 0. It will be selected in two cases—with an address request of 00xx xxxx and of 10xx xxxx. The first address range is by design, but the second is a side effect of partial address decoding. In effect, you have mapped one physical device into two separate regions of the address space. The CPU sees a clone of the chip at address 0 and address 128. Similar reasoning shows that the 32 × 8 RAM is duplicated once more at address 96, and the eight-port I/O chip at three more locations. The ROM is not duplicated, as its address decode circuitry is the same for full and partial addressing. FIGURE 11.51 shows the memory map with partial address decoding.

FIGURE 11.50
Partial address decoding for the memory map of Figure 11.47.

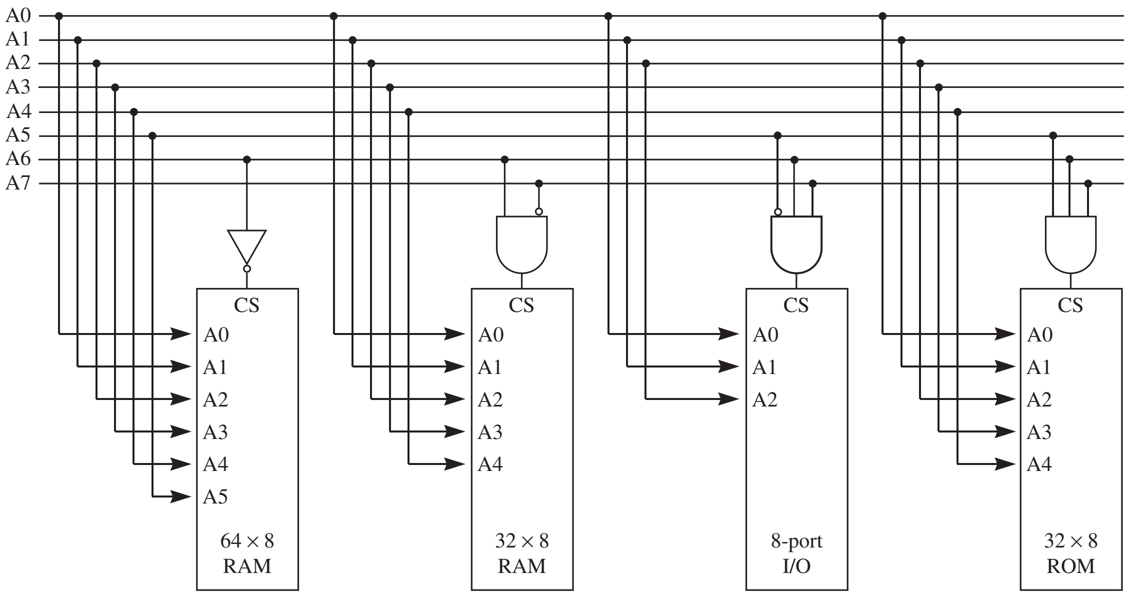
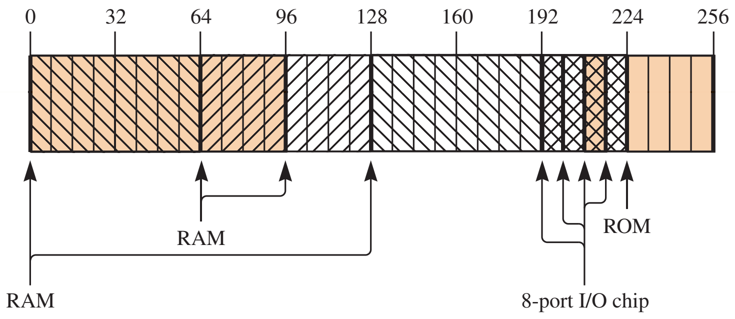


FIGURE 11.51
The memory map with the partial address decoding of Figure 11.49.



You must be careful with partial address decoding. In this example, the chips were duplicated in such a way as to completely fill the memory map with no gaps and no overlaps. If your decoding leaves gaps in the resulting map, no harm is done. If the CPU accesses a gap, then no chip is selected. If your decoding produces overlaps, however, it means that more than one chip will be selected by the CPU if it ever tries to access that region of the address space. The result could be hazardous to the system. Of course, the premise is that the CPU should never access that region, as it has no need to access a duplicated chip when it can access the original. But when is the last time you were sure there were no bugs in your program, only to discover later that there were?

A Two-Port Register Bank

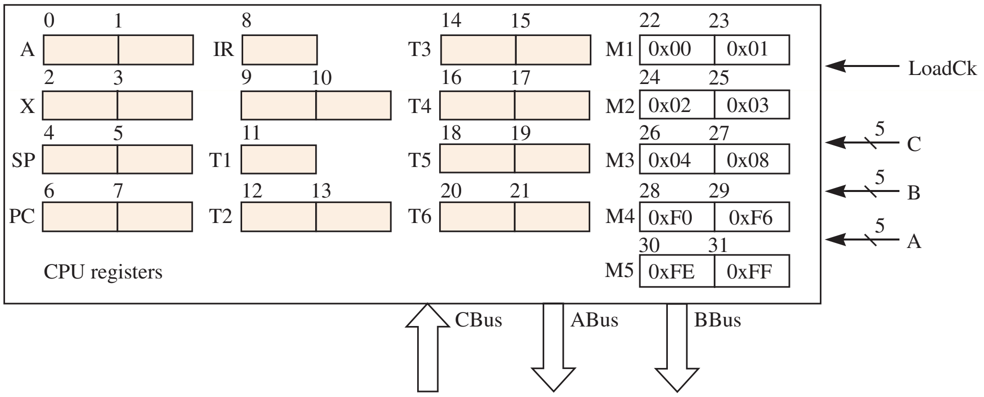
The memory subsystems of the previous section all have just one set of data lines that correspond to one set of address lines. That organization is appropriate for the main memory subsystem of a computer, which normally does not reside on the same integrated circuit as the CPU. Figure 4.2 shows the registers in the Pep/9 CPU. They are organized much like a memory subsystem but are stored in a register bank in the CPU itself. The register bank in the CPU differs from the memory organization of a memory chip in two respects:

- The data buses are unidirectional instead of bidirectional.
- There are two output ports instead of one.

FIGURE 11.52 shows the 32 8-bit registers, addressed from 0 to 31. The first five registers are the ones visible to the programmer at the ISA level. Each 16-bit register is divided into two 8-bit registers. That division is invisible to the machine-level programmer.

The remaining registers, addressed from 11 to 31, are not visible to the machine-level programmer. Registers 11 to 21 comprise a group of registers for storing temporary values. Registers 22 to 31 are read-only registers that contain fixed values. They are similar to ROM in that if you try to store to them, the value in the register will not change. Their constant values are given in hexadecimal. The read-only registers are not shaded because they

FIGURE 11.52
The 32 eight-bit registers in the Pep/9 CPU.

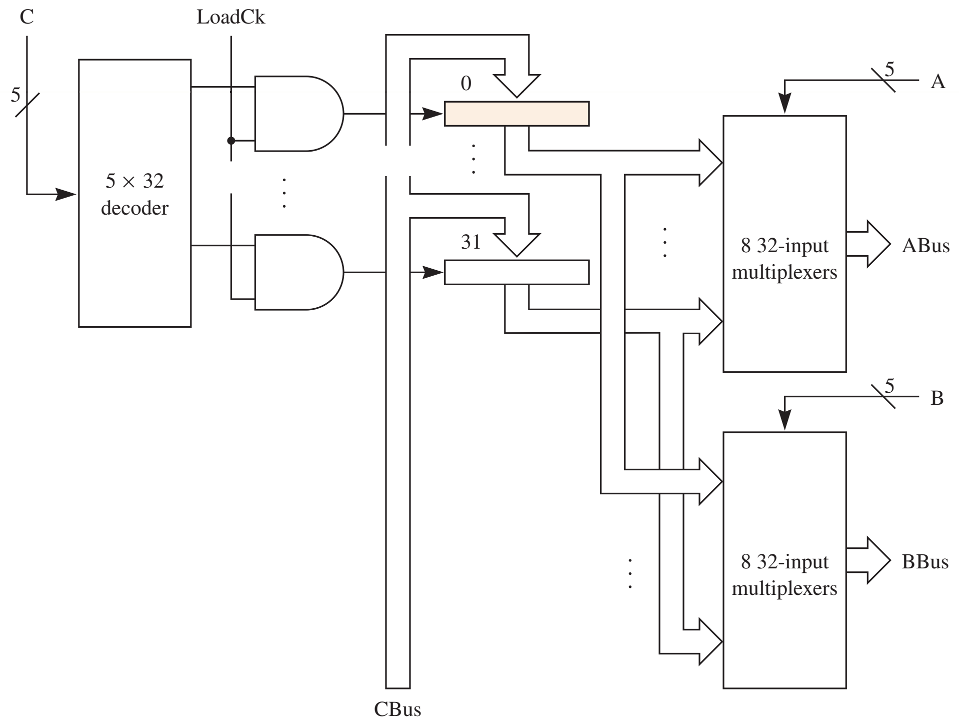


are not really sequential circuits. They act more like combinational circuits because they have no states that can change.

Where a main memory chip has one set of address lines and one set of bidirectional data lines, this register bank has three sets of addresses—A, B, and C—and three unidirectional data buses—ABus, BBus, and CBus. ABus and BBus are the two output ports, and CBus is the input port. Each data bus is eight bits wide, and each set of address lines contains five wires, capable of accessing any of the 2^5 registers. To store a value to a register, you place the address of the register on C, the data to store on the CBus, and clock the control line labeled *LoadCk*. The two output ports allow you to read two different registers simultaneously. You can put any address on A and any address on B, and the data from those two registers will appear simultaneously on ABus and BBus. You are allowed to put the same address on A and B, in which case the data from the one register will appear on both the ABus and the BBus.

FIGURE 11.53 shows the implementation of the two-port register bank. The input follows the same basic organization as a main memory chip. The

FIGURE 11.53
Implementation of the two-port register bank of Figure 11.52.



five address lines from C make one decoder output line 1 and the rest 0. CBus is connected to each of the 32 registers. When you pulse the LoadCk control line, the value on CBus is clocked into one of the registers.

The two output ports are 32-input multiplexers, each capable of routing an eight-bit quantity. Each of these multiplexers is a bank of eight individual 32-input multiplexers like the one in Figure 10.43. The five lines of A connect to the five select lines of all eight individual multiplexers in the first port. The first multiplexer routes the first bit from all 32 registers, the second multiplexer routes the second bit from all 32 registers, and so on.

Chapter Summary

A sequential circuit, constructed from logic gates with feedback loops, can remember its state. The four basic sequential devices are the SR flip-flop (set, reset), the JK flip-flop, the D flip-flop (data or delay), and the T flip-flop (toggle). The S input to the SR flip-flop sets the state to 1, and the R input resets the state to 0. The input condition SR = 11 is undefined. The JK inputs correspond to SR, except that the input condition JK = 11 is defined and toggles the state. The D input transfers directly to the state. If the T input is 1, the state toggles; otherwise, it remains unchanged. Each of these flip-flops may be constructed as master–slave devices to solve the instability problem that external feedback would produce.

A general sequential circuit consists of a combinational circuit whose output feeds into a set of state registers. The output from the state registers feeds back to the input of the combinational circuit. A sequential circuit can be characterized by a state transition diagram, which is a manifestation of a finite-state machine. When you analyze a sequential circuit, you are given the input and the sequential circuit, and you determine the output.

When you design a sequential circuit, you are given the input and the desired output, and you determine the sequential circuit. Excitation tables aid the design process. An excitation table for a flip-flop consists of the four possible state changes of the device—0 to 0, 0 to 1, 1 to 0, and 1 to 1—and the input conditions necessary to produce the change. The design process consists of tabulating the input conditions necessary to produce the given state transition diagram, and then designing the combinational circuit to produce those input conditions.

A register is a sequence of D flip-flops. The tri-state buffer makes possible the implementation of the bidirectional bus. A memory chip is (conceptually) an array of D flip-flops with a set of address lines, data lines, and control lines. The control lines usually consist of *CS* for *chip select*, *WE* for *write enable*, and *OE* for *output enable*. Address decoding is a technique for using the CS lines to construct a memory module from a set of memory chips. Partial address decoding minimizes the number of gates in the select circuitry. The two-port register bank in the Pep/9 CPU implements the registers visible to the ISA programmer, as well as the temporary registers and constant registers that are not visible.

Exercises

Section 11.1

*__1.__ Under what circumstances will a string of an arbitrary number of inverters with a feedback loop, as in Figure 11.1, produce a stable network?

2. Construct tables analogous to Figures 11.3 and 11.4 to show that changing R to 1 and back to 0 resets the SR latch to Q = 0 if it starts in state Q = 1.

3. Define the following points in Figure 11.10: (1) A is the output of the top master AND gate. (2) B is the output of the bottom master AND gate. (3) C is the output of the inverter. (4) D is the output of the top slave AND gate. (5) E is the output of the bottom slave AND gate. Suppose SR = 10 and Q = 0 before the arrival of a clock pulse. Construct a table that shows the values of A, B, C, D, E, R2, S2, Q, and $\overline{Q}$ during each of the following intervals of Figure 11.11, assuming zero gate delay:

*__(a)__ before t_1 *__(b)__ between t_1 and t_2 **(c)** between t_2 and t_3
(d) between t_3 and t_4 **(e)** after t_4

4. Do Exercise 3 with SR = 01 and Q = 1 before the arrival of the clock pulse.

5. Draw the state transition diagram, as in Figure 11.14, for the following flip-flops:

(a) JK *__(b)__ D **(c)** T

6. Draw the timing diagram of Figure 11.21(c) for the toggle flip-flop with the D input replaced by T.

7. Construct the T flip-flop from an SR flip-flop.

8. This section shows how the JK and D flip-flops can be constructed from the SR flip-flop and a few gates. In fact, any flip-flop can be constructed from any other with the help of a few gates. Construct the following flip-flops from a JK flip-flop:

*(**a**) D (**b**) SR (**c**) T

Construct the following flip-flops from a D flip-flop:

(**d**) SR (**e**) JK (**f**) T

Construct the following flip-flops from a T flip-flop:

(**g**) SR (**h**) JK (**i**) D

Section 11.2

9. Modify Figure 11.10, the implementation of the SR master–slave flip-flop, to provide asynchronous preset and clear inputs, as in Figure 11.32. When preset and clear are both 0, the device should operate normally. When preset is 1, both the master state Q2 and the slave state Q should be forced to 1 independent of the clock Ck. When clear is 1, both the master state, Q2, and the slave state, Q, should be forced to 0. You may assume that preset and clear will not both be 1 simultaneously. You can design the circuit with no extra gates if you assume that existing AND and OR gates may have three inputs instead of two.

10. Draw the logic diagram and the state transition diagram for a sequential circuit with two JK flip-flops, FFA and FFB, and two inputs, X1 and X2, with flip-flop inputs

$JA = X1\ B$ $JB = X1\ \overline{A}$

$KA = X2 + X1\ A\overline{B}$ $KB = X2 + X1\ A$

There is no output other than the flip-flop states.

11. Draw the logic diagram and the state transition diagram for a sequential circuit with one JK flip-flop, FFA; one T flip-flop, FFB; and one input, X, with flip-flop inputs

$J = X \oplus B$ $T = X \oplus A$

$K = \overline{X}\ B$

and output

$Z = A\ B$

12. Draw the logic diagram and the state transition diagram for a sequential circuit with two SR flip-flops, FFA and FFB; two inputs, X1 and X2, with flip-flop inputs

$SA = X1$ $SB = \overline{X1}\ \overline{X2}\ \overline{A}$

$RA = \overline{X1}\ X2$ $RB = X1\ A + X2$

and output

$Z = X1\ \overline{A}$

13. Design the sequential circuit of Figure 11.33 using the following flip-flops:

*(**a**) D (**b**) T

14. FIGURE 11.54 is a state transition diagram for a sequential circuit with three flip-flops and one input. It counts up in binary when the input is 1 and remains in the same state when the input is 0. Design the circuit and draw the logic diagram using the following flip-flops:

(**a**) JK (**b**) SR (**c**) D (**d**) T

15. FIGURE 11.55 is a state transition diagram for a sequential circuit with three flip-flops and one input. It counts up in binary when the input is 1 and counts down when the input is 0. Design the circuit and draw the logic diagram using the following flip-flops:

*(**a**) JK (**b**) SR (**c**) D (**d**) T

16. FIGURE 11.56 is a state transition diagram for a sequential circuit with two flip-flops and two inputs. It counts up in binary when the input is 01, counts down in binary when the input is 10, and does not change

FIGURE 11.54
The state transition diagram for Exercise 14.

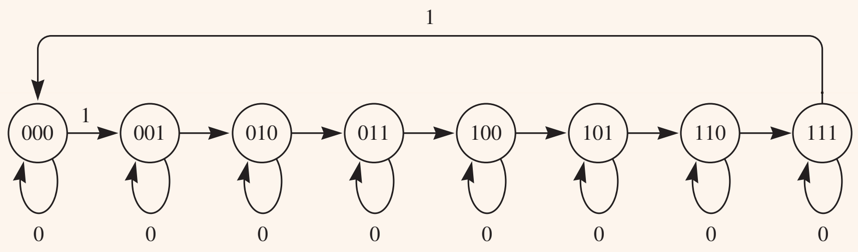


FIGURE 11.55
The state transition diagram for Exercise 15.

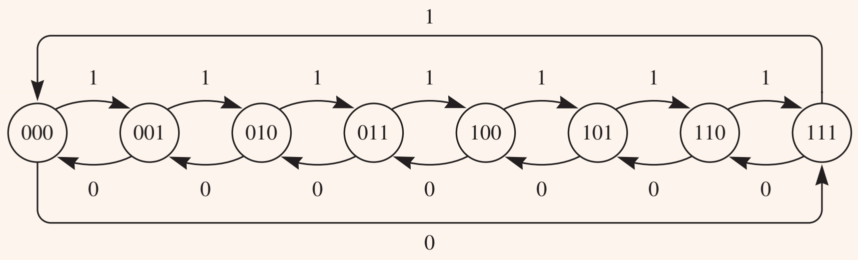
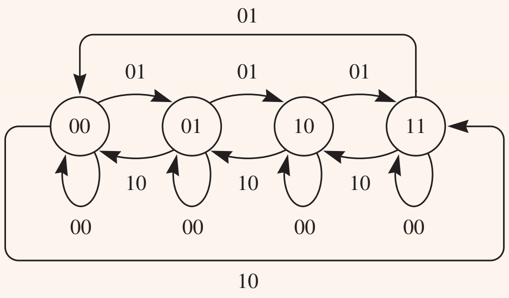


FIGURE 11.56
The state transition diagram for Exercise 16.



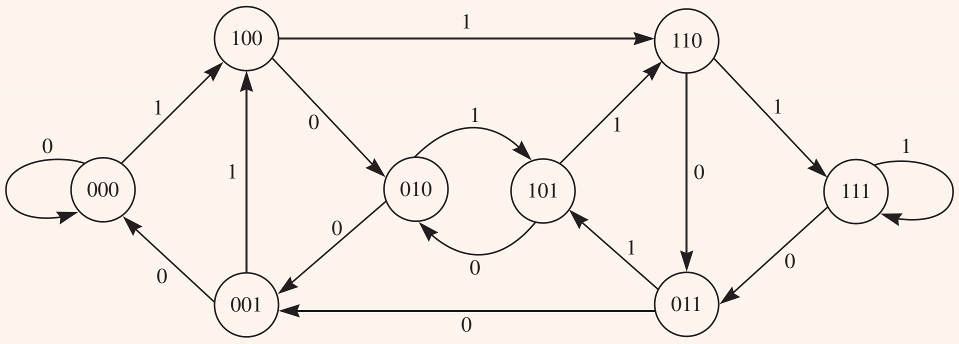
state when the input is 00. An input of 11 will never occur and can be treated as a don't-care condition. Design the circuit and draw the logic diagram using the following flip-flops:

(a) JK **(b)** SR **(c)** D **(d)** T

17. FIGURE 11.57 is a state transition diagram for a sequential circuit with three flip-flops and one input. It is a three-bit shift right register. If the input is 0, a 0 is shifted into the most significant bit. If the input is 1, a 1 is shifted into the most significant bit. Design the circuit and draw the logic diagram using the following flip-flops:

(a) JK **(b)** SR **(c)** D **(d)** T

FIGURE 11.57
The state transition diagram for Exercise 17.



18. A sequential circuit has six state registers and three input lines. **(a)** How many states does it have? **(b)** How many transitions from each state does it have? **(c)** How many total transitions does it have?

Section 11.3

19. **(a)** How many AND gates, OR gates, and inverters are in the memory chip of Figure 11.42? Include the gates in the decoder but not in the D flip-flops. **(b)** How many of each element are in the memory chip of Figure 11.41(a)? **(c)** How many are in the memory chip of Figure 11.41(b)?

20. In practice, the chip select line of a memory chip is asserted low, and the line is labeled $\overline{\text{CS}}$ instead of CS to indicate that fact. How would Figure 11.49 change if the chip select lines on all the chips were asserted low?

21. A computer system has a 16-bit wide data bus. **(a)** If you have a box of 1 Ki × 1 dynamic RAM chips, what is the smallest number of bytes of memory this computer can have? **(b)** Answer part (a) if the 1 KiB chips you have are configured as 256 × 4 devices.

22. You have a small CPU with a 10-bit address bus. You need to connect a 64-byte PROM, a 32-byte RAM, and a 4-port I/O chip with two address lines. Chip selects on all chips are asserted high. **(a)** Show the connection for full address decoding with the PROM at address 0, the RAM at address 384, and the PIO at address 960. (These addresses are decimal.) **(b)** Show the connection for partial address decoding with the chips at the same locations. Show the memory map with partial address decoding and ensure that no duplicate regions overlap. For each chip, state (1) how many clones are produced and (2) the starting address of each clone.

23. Show how the individual multiplexers in the top port of Figure 11.53 are connected. You may use ellipses (. . .).

24. How many AND gates, OR gates, and inverters are in the two-port register bank of Figure 11.53? Include the gates necessary to construct the decoder and the multiplexers, but not the ones in the D flip-flops that make up the registers.