In previous chapters we considered combinational circuits where the value of each output depends solely on the values of signals applied to the inputs. There exists another class of logic circuits in which the values of the outputs depend not only on the present values of the inputs but also on the past behavior of the circuit. Such circuits include storage elements that store the values of logic signals. The contents of the storage elements are said to represent the *state* of the circuit. When the circuit's inputs change values, the new input values either leave the circuit in the same state or cause it to change into a new state. Over time the circuit changes through a sequence of states as a result of changes in the inputs. Circuits that behave in this way are referred to as *sequential circuits*.

In this chapter we will introduce circuits that can be used as storage elements. But first, we will motivate the need for such circuits by means of a simple example. Suppose that we wish to control an alarm system, as shown in Figure 7.1. The alarm mechanism responds to the control input $On/\overline{Off}$. It is turned on when $On/\overline{Off} = 1$, and it is off when $On/\overline{Off} = 0$. The desired operation is that the alarm turns on when the sensor generates a positive voltage signal, *Set*, in response to some undesirable event. Once the alarm is triggered, it must remain active even if the sensor output goes back to zero. The alarm is turned off manually by means of a *Reset* input. The circuit requires a memory element to remember that the alarm has to be active until the *Reset* signal arrives.

Figure 7.2 gives a rudimentary memory element, consisting of a loop that has two inverters. If we assume that $A = 0$, then $B = 1$. The circuit will maintain these values indefinitely. We say that the circuit is in the *state* defined by these values. If we assume that $A = 1$, then $B = 0$, and the circuit will remain in this second state indefinitely. Thus the circuit has two possible states. This circuit is not useful, because it lacks some practical means for changing its state.

A more useful circuit is shown in Figure 7.3. It includes a mechanism for changing the state of the circuit in Figure 7.2, using two transmission gates of the type discussed in section 3.9. One transmission gate, *TG*1, is used to connect the *Data* input terminal to point

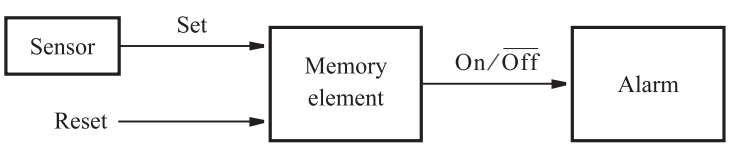


Figure 7.1 Control of an alarm system.

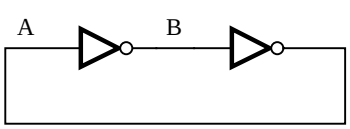


Figure 7.2 A simple memory element.

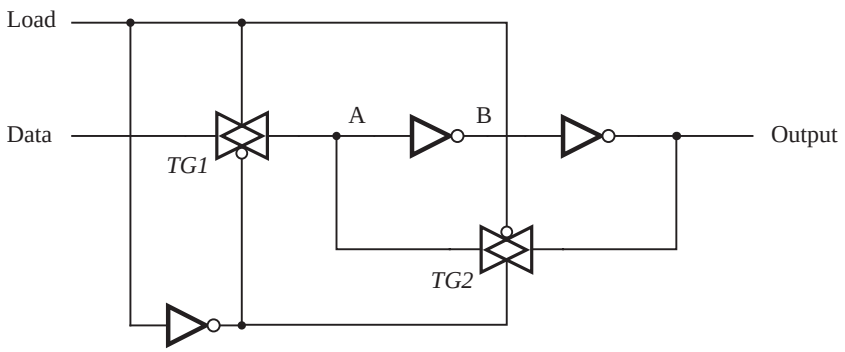


Figure 7.3 A controlled memory element.

A in the circuit. The second, *TG*2, is used as a switch in the *feedback loop* that maintains the state of the circuit. The transmission gates are controlled by the *Load* signal. If *Load* = 1, then *TG*1 is on and the point *A* will have the same value as the *Data* input. Since the value presently stored at *Output* may not be the same value as *Data*, the feedback loop is broken by having *TG*2 turned off when *Load* = 1. When *Load* changes to zero, then *TG*1 turns off and *TG*2 turns on. The feedback path is closed and the memory element will retain its state as long as *Load* = 0. This memory element cannot be applied directly to the system in Figure 7.1, but it is useful for many other applications, as we will see later.

7.1 BASIC LATCH

Instead of using the transmission gates, we can construct a similar circuit using ordinary logic gates. Figure 7.4 presents a memory element built with NOR gates. Its inputs, *Set* and *Reset*, provide the means for changing the state, Q, of the circuit. A more usual way of drawing this circuit is given in Figure 7.5*a*, where the two NOR gates are said to be connected in cross-coupled style. The circuit is referred to as a *basic latch*. Its behavior is described by the table in Figure 7.5*b*. When both inputs, *R* and *S*, are equal to 0 the latch maintains its existing state. This state may be either $Q_a = 0$ and $Q_b = 1$, or $Q_a = 1$ and $Q_b = 0$, which is indicated in the table by stating that the Q_a and Q_b outputs have values

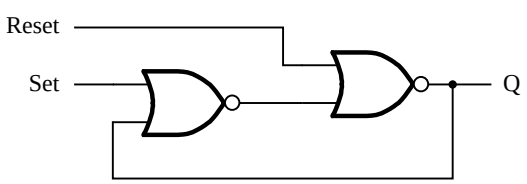
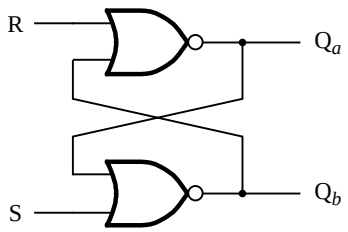


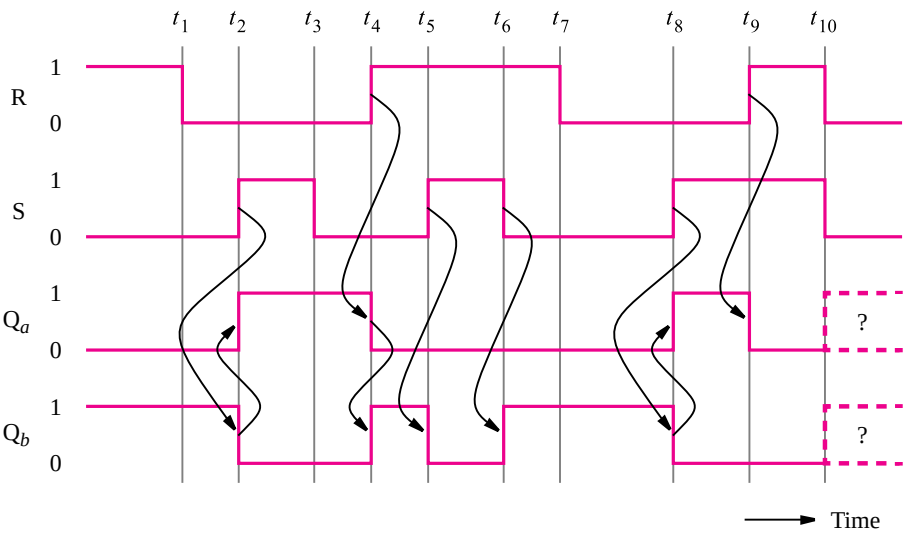
Figure 7.4 A memory element with NOR gates.



(a) Circuit

S	R	Q_a	Q_b	
0	0	0/1	1/0	(no change)
0	1	0	1	
1	0	1	0	
1	1	0	0	

(b) Characteristic table



(c) Timing diagram

Figure 7.5 A basic latch built with NOR gates.

0/1 and 1/0, respectively. Observe that Q_a and Q_b are complements of each other in this case. When $R = 0$ and $S = 1$, the latch is *set* into a state where $Q_a = 1$ and $Q_b = 0$. When $R = 1$ and $S = 0$, the latch is *reset* into a state where $Q_a = 0$ and $Q_b = 1$. The fourth possibility is to have $R = S = 1$. In this case both Q_a and Q_b will be 0. The table in Figure 7.5*b* resembles a truth table. However, since it does not represent a combinational circuit in which the values of the outputs are determined solely by the current values of the inputs, it is often called a *characteristic table* rather than a truth table.

Figure 7.5*c* gives a timing diagram for the latch, assuming that the propagation delay through the NOR gates is negligible. Of course, in a real circuit the changes in the waveforms would be delayed according to the propagation delays of the gates. We assume that initially $Q_a = 0$ and $Q_b = 1$. The state of the latch remains unchanged until time t_2, when S becomes equal to 1, causing Q_b to change to 0, which in turn causes Q_a to change to 1.

The causality relationship is indicated by the arrows in the diagram. When S goes to 0 at t_3, there is no change in the state because both S and R are then equal to 0. At t_4 we have $R = 1$, which causes Q_a to go to 0, which in turn causes Q_b to go to 1. At t_5 both S and R are equal to 1, which forces both Q_a and Q_b to be equal to 0. As soon as S returns to 0, at t_6, Q_b becomes equal to 1 again. At t_8 we have $S = 1$ and $R = 0$, which causes $Q_b = 0$ and $Q_a = 1$. An interesting situation occurs at t_{10}. From t_9 to t_{10} we have $Q_a = Q_b = 0$ because $R = S = 1$. Now if both R and S change to 0 at t_{10}, both Q_a and Q_b will go to 1. But having both Q_a and Q_b equal to 1 will immediately force $Q_a = Q_b = 0$. There will be an oscillation between $Q_a = Q_b = 0$ and $Q_a = Q_b = 1$. If the delays through the two NOR gates are exactly the same, the oscillation will continue indefinitely. In a real circuit there will invariably be some difference in the delays through these gates, and the latch will eventually settle into one of its two stable states, but we don't know which state it will be. This uncertainty is indicated in the waveforms by dashed lines.

The oscillations discussed above illustrate that even though the basic latch is a simple circuit, careful analysis has to be done to fully appreciate its behavior. In general, any circuit that contains one or more feedback paths, such that the state of the circuit depends on the propagation delays through logic gates, has to be designed carefully. We discuss timing issues in detail in Chapter 9.

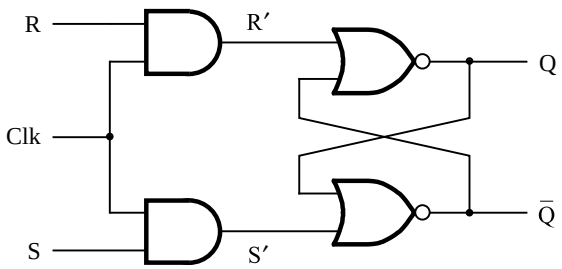
The latch in Figure 7.5*a* can perform the functions needed for the memory element in Figure 7.1, by connecting the *Set* signal to the S input and *Reset* to the R input. The Q_a output provides the desired $On/\overline{Off}$ signal. To initialize the operation of the alarm system, the latch is reset. Thus the alarm is off. When the sensor generates the logic value 1, the latch is set and Q_a becomes equal to 1. This turns on the alarm mechanism. If the sensor output returns to 0, the latch retains its state where $Q_a = 1$; hence the alarm remains turned on. The only way to turn off the alarm is by resetting the latch, which is accomplished by making the *Reset* input equal to 1.

7.2 GATED SR LATCH

In section 7.1 we saw that the basic SR latch can serve as a useful memory element. It remembers its state when both the S and R inputs are 0. It changes its state in response to changes in the signals on these inputs. The state changes occur at the time when the changes in the signals occur. If we cannot control the time of such changes, then we don't know when the latch may change its state.

In the alarm system of Figure 7.1, it may be desirable to be able to enable or disable the entire system by means of a control input, *Enable*. Thus when enabled, the system would function as described above. In the disabled mode, changing the *Set* input from 0 to 1 would not cause the alarm to turn on. The latch in Figure 7.5*a* cannot provide the desired operation. But the latch circuit can be modified to respond to the input signals S and R only when $Enable = 1$. Otherwise, it would maintain its state.

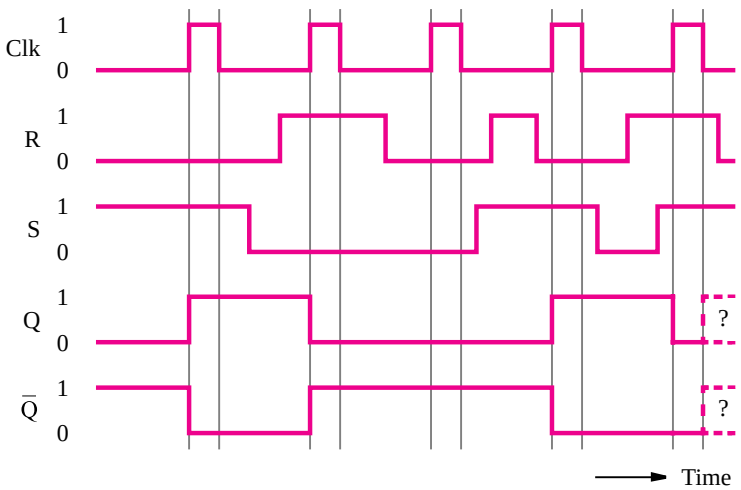
The modified circuit is depicted in Figure 7.6*a*. It includes two AND gates that provide the desired control. When the control signal *Clk* is equal to 0, the S' and R' inputs to the latch will be 0, regardless of the values of signals S and R. Hence the latch will maintain its



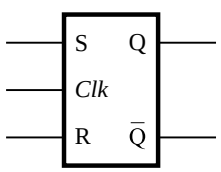
(a) Circuit

Clk	S	R	Q(*t* + 1)
0	x	x	Q(*t*) (no change)
1	0	0	Q(*t*) (no change)
1	0	1	0
1	1	0	1
1	1	1	x

(b) Characteristic table



(c) Timing diagram



(d) Graphical symbol

Figure 7.6 Gated SR latch.

existing state as long as $Clk = 0$. When *Clk* changes to 1, the S' and R' signals will be the same as the *S* and *R* signals, respectively. Therefore, in this mode the latch will behave as we described in section 7.1. Note that we have used the name *Clk* for the control signal that allows the latch to be set or reset, rather than call it the *Enable* signal. The reason is that such circuits are often used in digital systems where it is desirable to allow the changes in

the states of memory elements to occur only at well-defined time intervals, as if they were controlled by a clock. The control signal that defines these time intervals is usually called the *clock* signal. The name *Clk* is meant to reflect this nature of the signal.

Circuits of this type, which use a control signal, are called *gated latches*. Because our circuit exhibits set and reset capability, it is called a *gated SR latch*. Figure 7.6*b* describes its behavior. It defines the state of the Q output at time $t+1$, namely, $Q(t+1)$, as a function of the inputs S, R, and *Clk*. When $Clk = 0$, the latch will remain in the state it is in at time t, that is, $Q(t)$, regardless of the values of inputs S and R. This is indicated by specifying S = x and R = x, where x means that the signal value can be either 0 or 1. (Recall that we already used this notation in Chapter 4.) When $Clk = 1$, the circuit behaves as the basic latch in Figure 7.5. It is set by $S = 1$ and reset by $R = 1$. The last row of the table, where $S = R = 1$, shows that the state $Q(t + 1)$ is undefined because we don't know whether it will be 0 or 1. This corresponds to the situation described in section 7.1 in conjunction with the timing diagram in Figure 7.5 at time t_{10}. At this time both S and R inputs go from 1 to 0, which causes the oscillatory behavior that we discussed. If $S = R = 1$, this situation will occur as soon as *Clk* goes from 1 to 0. To ensure a meaningful operation of the gated SR latch, it is essential to avoid the possibility of having both the S and R inputs equal to 1 when *Clk* changes from 1 to 0.

A timing diagram for the gated SR latch is given in Figure 7.6*c*. It shows *Clk* as a periodic signal that is equal to 1 at regular time intervals to suggest that this is how the clock signal usually appears in a real system. The diagram presents the effect of several combinations of signal values. Observe that we have labeled one output as Q and the other as its complement $\overline{Q}$, rather than Q_a and Q_b as in Figure 7.5. Since the undefined mode, where $S = R = 1$, must be avoided in practice, the normal operation of the latch will have the outputs as complements of each other. Moreover, we will often say that the latch is *set* when $Q = 1$, and it is *reset* when $Q = 0$. A graphical symbol for the gated SR latch is given in Figure 7.6*d*.

7.2.1 GATED SR LATCH WITH NAND GATES

So far we have implemented the basic latch with cross-coupled NOR gates. We can also construct the latch with NAND gates. Using this approach, we can implement the gated SR latch as depicted in Figure 7.7. The behavior of this circuit is described by the table in Figure 7.6*b*. Note that in this circuit, the clock is gated by NAND gates, rather than by

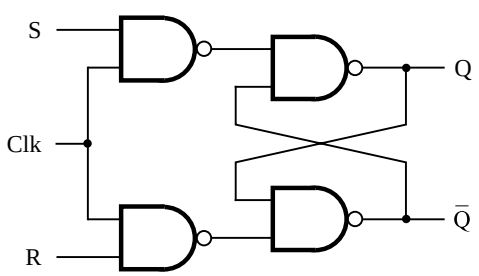


Figure 7.7 Gated SR latch with NAND gates.

AND gates. Note also that the *S* and *R* inputs are reversed in comparison with the circuit in Figure 7.6*a*. The circuit with NAND gates requires fewer transistors than the circuit with AND gates. We will use the circuit in Figure 7.7, in preference to the circuit in Figure 7.6*a*.

7.3 GATED D LATCH

In section 7.2 we presented the gated SR latch and showed how it can be used as the memory element in the alarm system of Figure 7.1. This latch is useful for many other applications. In this section we describe another gated latch that is even more useful in practice. It has a single data input, called *D*, and it stores the value on this input, under the control of a clock signal. It is called a *gated D latch*.

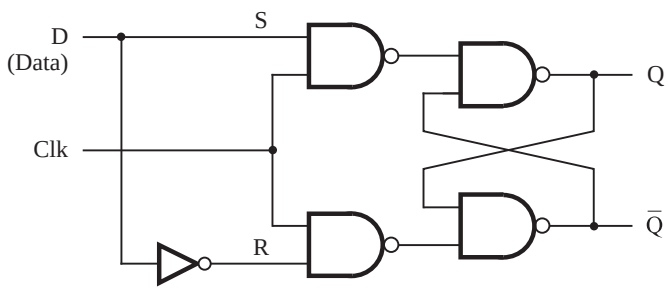
To motivate the need for a gated D latch, consider the adder/subtractor unit discussed in Chapter 5 (Figure 5.13). When we described how that circuit is used to add numbers, we did not discuss what is likely to happen with the sum bits that are produced by the adder. Adder/subtractor units are often used as part of a computer. The result of an addition or subtraction operation is often used as an operand in a subsequent operation. Therefore, it is necessary to be able to remember the values of the sum bits generated by the adder until they are needed again. We might think of using the basic latches to remember these bits, one bit per latch. In this context, instead of saying that a latch remembers the value of a bit, it is more illuminating to say that the latch *stores* the value of the bit or simply "stores the bit." We should think of the latch as a storage element.

But can we obtain the desired operation using the basic latches? We can certainly reset all latches before the addition operation begins. Then we would expect that by connecting a sum bit to the *S* input of a latch, the latch would be set to 1 if the sum bit has the value 1; otherwise, the latch would remain in the 0 state. This would work fine if all sum bits are 0 at the start of the addition operation and, after some propagation delay through the adder, some of these bits become equal to 1 to give the desired sum. Unfortunately, the propagation delays that exist in the adder circuit cause a big problem in this arrangement. Suppose that we use a ripple-carry adder. When the *X* and *Y* inputs are applied to the adder, the sum outputs may alternate between 0 and 1 a number of times as the carries ripple through the circuit. This situation was illustrated in the timing diagram in Figure 5.21. The problem is that if we connect a sum bit to the *S* input of a latch, then if the sum bit is temporarily a 1 and then settles to 0 in the final result, the latch will remain set to 1 erroneously.

The problem caused by the alternating values of the sum bits in the adder could be solved by using the gated SR latches, instead of the basic latches. Then we could arrange that the clock signal is 0 during the time needed by the adder to produce a correct sum. After allowing for the maximum propagation delay in the adder circuit, the clock should go to 1 to store the values of the sum bits in the gated latches. As soon as the values have been stored, the clock can return to 0, which ensures that the stored values will be retained until the next time the clock goes to 1. To achieve the desired operation, we would also have to reset all latches to 0 prior to loading the sum-bit values into these latches. This is an awkward way of dealing with the problem, and it is preferable to use the gated D latches instead.

Figure 7.8*a* shows the circuit for a gated D latch. It is based on the gated SR latch, but instead of using the S and R inputs separately, it has just one data input, D. For convenience we have labeled the points in the circuit that are equivalent to the S and R inputs. If $D = 1$, then $S = 1$ and $R = 0$, which forces the latch into the state $Q = 1$. If $D = 0$, then $S = 0$ and $R = 1$, which causes $Q = 0$. Of course, the changes in state occur only when $Clk = 1$.

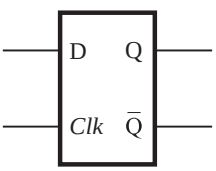
It is important to observe that in this circuit it is impossible to have the troublesome situation where $S = R = 1$. In the gated D latch, the output Q merely tracks the value of the input D while $Clk = 1$. As soon as Clk goes to 0, the state of the latch is frozen until the next time the clock signal goes to 1. Therefore, the gated D latch stores the value of the D



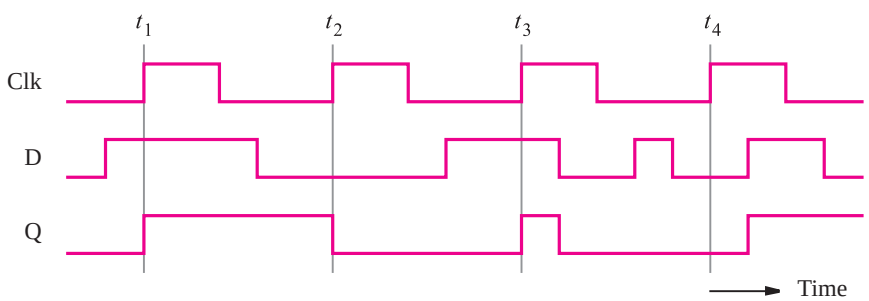
(a) Circuit

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

(b) Characteristic table



(c) Graphical symbol



(d) Timing diagram

Figure 7.8 Gated D latch.

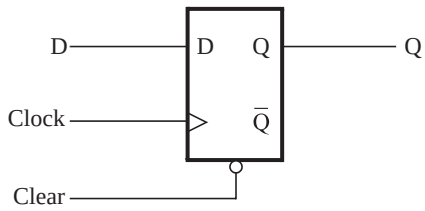
input seen at the time the clock changes from 1 to 0. Figure 7.8 also gives the characteristic table, the graphical symbol, and the timing diagram for the gated D latch.

The timing diagram illustrates what happens if the D signal changes while $Clk = 1$. During the third clock pulse, starting at t_3, the output Q changes to 1 because $D = 1$. But midway through the pulse D goes to 0, which causes Q to go to 0. This value of Q is stored when *Clk* changes to 0. Now no further change in the state of the latch occurs until the next clock pulse, at t_4. The key point to observe is that as long as the clock has the value 1, the Q output follows the D input. But when the clock has the value 0, the Q output cannot change. In Chapter 3 we saw that the logic values are implemented as low and high voltage levels. Since the output of the gated D latch is controlled by the level of the clock input, the latch is said to be *level sensitive*. The circuits in Figures 7.6 through 7.8 are level sensitive. We will show in section 7.4 that it is possible to design storage elements for which the output changes only at the point in time when the clock changes from one value to the other. Such circuits are said to be *edge triggered*.

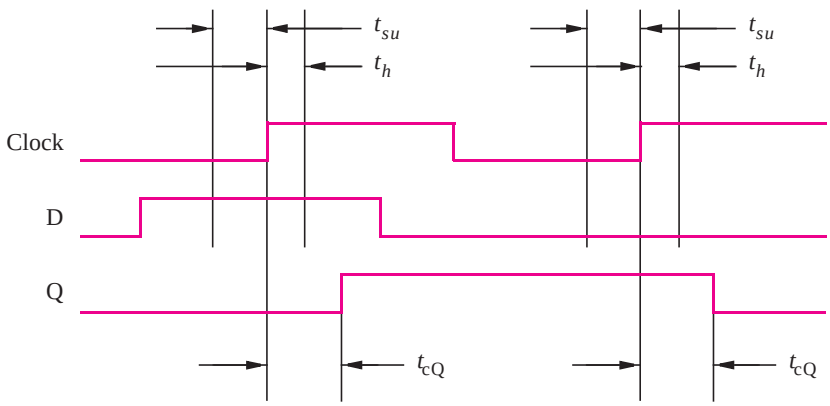
At this point we should reconsider the circuit in Figure 7.3. Careful examination of that circuit shows that it behaves in exactly the same way as the circuit in Figure 7.8*a*. The *Data* and *Load* inputs correspond to the D and *Clk* inputs, respectively. The *Output*, which has the same signal value as point A, corresponds to the Q output. Point B corresponds to $\overline{Q}$. Therefore, the circuit in Figure 7.3 is also a gated D latch. An advantage of this circuit is that it can be implemented using fewer transistors than the circuit in Figure 7.8*a*.

7.4.2 EDGE-TRIGGERED D FLIP-FLOP

The output of the master-slave D flip-flop in Figure 7.10*a* responds on the negative edge of the clock signal. The circuit can be changed to respond to the positive clock edge by connecting the slave stage directly to the clock and the master stage to the complement of



(a) D flip-flop with asynchronous clear



(b) Timing diagram

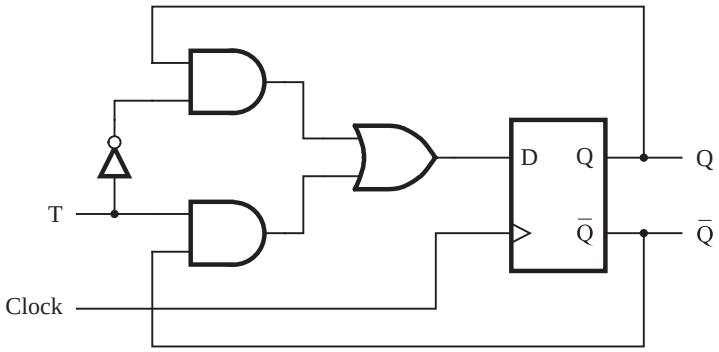
Figure 7.15 Flip-flop timing parameters.

process. In section 7.15 we provide some examples that illustrate the effects of flip-flop timing parameters on the operation of circuits.

7.5 T FLIP-FLOP

The D flip-flop is a versatile storage element that can be used for many purposes. By including some simple logic circuitry to drive its input, the D flip-flop may appear to be a different type of storage element. An interesting modification is presented in Figure 7.16*a*. This circuit uses a positive-edge-triggered D flip-flop. The *feedback* connections make the input signal D equal to either the value of Q or $\overline{Q}$ under the control of the signal that is labeled T. On each positive edge of the clock, the flip-flop may change its state $Q(t)$. If $T = 0$, then $D = Q$ and the state will remain the same, that is, $Q(t + 1) = Q(t)$. But if $T = 1$, then $D = \overline{Q}$ and the new state will be $Q(t + 1) = \overline{Q}(t)$. Therefore, the overall operation of the circuit is that it retains its present state if $T = 0$, and it reverses its present state if $T = 1$.

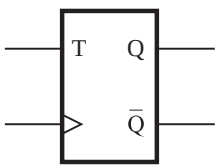
The operation of the circuit is specified in the form of a characteristic table in Figure 7.16*b*. Any circuit that implements this table is called a *T flip-flop*. The name T flip-flop



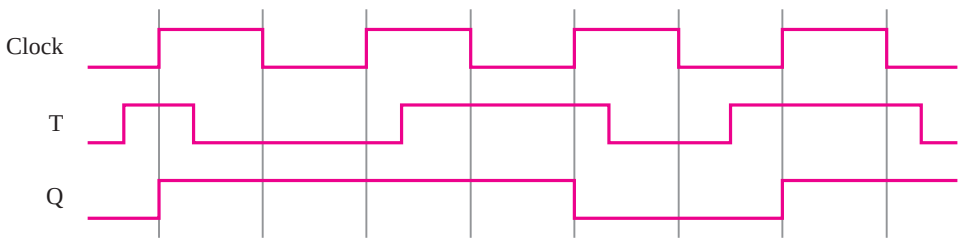
(a) Circuit

T	Q(t + 1)
0	Q(t)
1	$\overline{Q}(t)$

(b) Characteristic table



(c) Graphical symbol



(d) Timing diagram

Figure 7.16 T flip-flop.

derives from the behavior of the circuit, which "toggles" its state when $T = 1$. The toggle feature makes the T flip-flop a useful element for building counter circuits, as we will see in section 7.9.

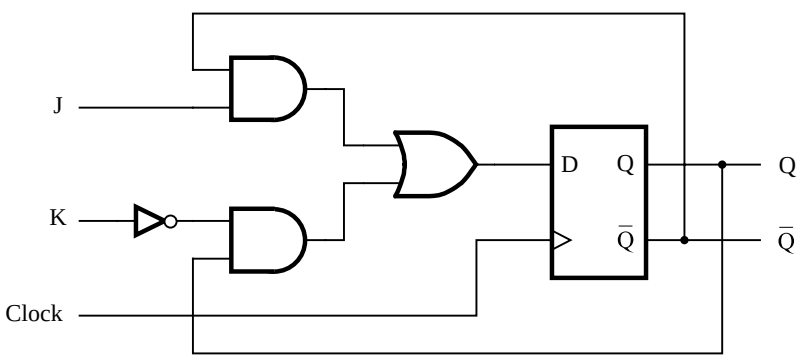
7.6 JK FLIP-FLOP

Another interesting circuit can be derived from Figure 7.16*a*. Instead of using a single control input, T, we can use two inputs, J and K, as indicated in Figure 7.17*a*. For this circuit the input D is defined as

$$D = J\overline{Q} + \overline{K}Q$$

A corresponding characteristic table is given in Figure 7.17*b*. The circuit is called a *JK flip-flop*. It combines the behaviors of SR and T flip-flops in a useful way. It behaves as the SR flip-flop, where $J = S$ and $K = R$, for all input values except $J = K = 1$. For the latter case, which has to be avoided in the SR flip-flop, the JK flip-flop toggles its state like the T flip-flop.

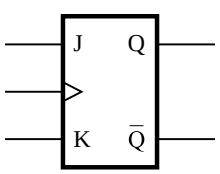
The JK flip-flop is a versatile circuit. It can be used for straight storage purposes, just like the D and SR flip-flops. But it can also serve as a T flip-flop by connecting the J and K inputs together.



(a) Circuit

J	K	Q(t + 1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q}(t)$

(b) Characteristic table



(c) Graphical symbol

Figure 7.17 JK flip-flop.

7.7 SUMMARY OF TERMINOLOGY

We have used the terminology that is quite common. But the reader should be aware that different interpretations of the terms *latch* and *flip-flop* can be found in the literature. Our terminology can be summarized as follows:

Basic latch is a feedback connection of two NOR gates or two NAND gates, which can store one bit of information. It can be set to 1 using the S input and reset to 0 using the R input.

Gated latch is a basic latch that includes input gating and a control input signal. The latch retains its existing state when the control input is equal to 0. Its state may be changed when the control signal is equal to 1. In our discussion we referred to the control input as the clock. We considered two types of gated latches:

- **Gated SR latch** uses the S and R inputs to set the latch to 1 or reset it to 0, respectively.
- **Gated D latch** uses the D input to force the latch into a state that has the same logic value as the D input.

A **flip-flop** is a storage element based on the gated latch principle, which can have its output state changed only on the edge of the controlling clock signal. We considered two types:

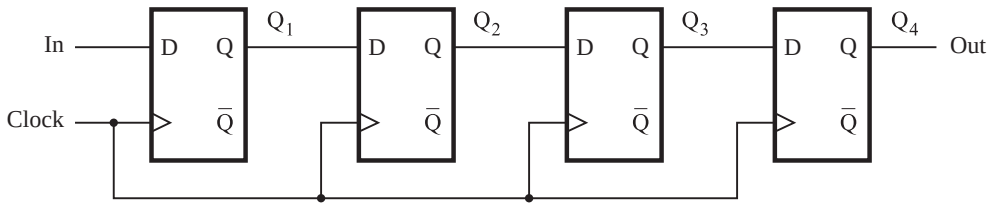
- **Edge-triggered flip-flop** is affected only by the input values present when the active edge of the clock occurs.
- **Master-slave flip-flop** is built with two gated latches. The master stage is active during half of the clock cycle, and the slave stage is active during the other half. The output value of the flip-flop changes on the edge of the clock that activates the transfer into the slave stage.

7.8 REGISTERS

A flip-flop stores one bit of information. When a set of n flip-flops is used to store n bits of information, such as an n-bit number, we refer to these flip-flops as a *register*. A common clock is used for each flip-flop in a register, and each flip-flop operates as described in the previous sections. The term register is merely a convenience for referring to n-bit structures consisting of flip-flops.

7.8.1 SHIFT REGISTER

In section 5.6 we explained that a given number is multiplied by 2 if its bits are shifted one bit position to the left and a 0 is inserted as the new least-significant bit. Similarly, the number is divided by 2 if the bits are shifted one bit-position to the right. A register that provides the ability to shift its contents is called a *shift register*.



(a) Circuit

	In	Q_1	Q_2	Q_3	Q_4 = Out
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	1	1	1	0	1
t_5	0	1	1	1	0
t_6	0	0	1	1	1
t_7	0	0	0	1	1

(b) A sample sequence

Figure 7.18 A simple shift register.

Figure 7.18*a* shows a four-bit shift register that is used to shift its contents one bit-position to the right. The data bits are loaded into the shift register in a serial fashion using the *In* input. The contents of each flip-flop are transferred to the next flip-flop at each positive edge of the clock. An illustration of the transfer is given in Figure 7.18*b*, which shows what happens when the signal values at *In* during eight consecutive clock cycles are 1, 0, 1, 1, 1, 0, 0, and 0, assuming that the initial state of all flip-flops is 0.

To implement a shift register, it is necessary to use either edge-triggered or master-slave flip-flops. The level-sensitive gated latches are not suitable, because a change in the value of *In* would propagate through more than one latch during the time when the clock is equal to 1.

7.8.2 PARALLEL-ACCESS SHIFT REGISTER

In computer systems it is often necessary to transfer n-bit data items. This may be done by transmitting all bits at once using n separate wires, in which case we say that the transfer is performed in *parallel*. But it is also possible to transfer all bits using a single wire, by

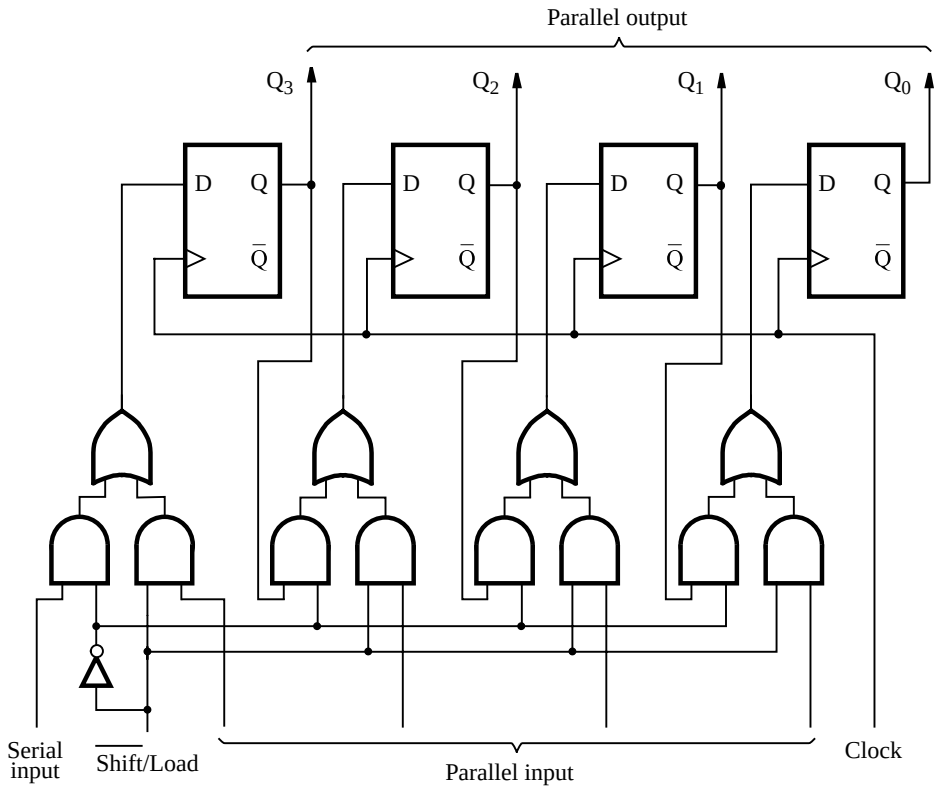


Figure 7.19 Parallel-access shift register.

performing the transfer one bit at a time, in n consecutive clock cycles. We refer to this scheme as *serial* transfer. To transfer an n-bit data item serially, we can use a shift register that can be loaded with all n bits in parallel (in one clock cycle). Then during the next n clock cycles, the contents of the register can be shifted out for serial transfer. The reverse operation is also needed. If bits are received serially, then after n clock cycles the contents of the register can be accessed in parallel as an n-bit item.

Figure 7.19 shows a four-bit shift register that allows the parallel access. Instead of using the normal shift register connection, the D input of each flip-flop is connected to two different sources. One source is the preceding flip-flop, which is needed for the shift-register operation. The other source is the external input that corresponds to the bit that is to be loaded into the flip-flop as a part of the parallel-load operation. The control signal $\overline{Shift}/Load$ is used to select the mode of operation. If $\overline{Shift}/Load = 0$, then the circuit operates as a shift register. If $\overline{Shift}/Load = 1$, then the parallel input data are loaded into the register. In both cases the action takes place on the positive edge of the clock.

In Figure 7.19 we have chosen to label the flip-flops outputs as $Q_3, \ldots, Q_0$ because shift registers are often used to hold binary numbers. The contents of the register can be accessed in parallel by observing the outputs of all flip-flops. The flip-flops can also be accessed serially, by observing the values of Q_0 during consecutive clock cycles while the

contents are being shifted. A circuit in which data can be loaded in series and then accessed in parallel is called a series-to-parallel converter. Similarly, the opposite type of circuit is a parallel-to-series converter. The circuit in Figure 7.19 can perform both of these functions.

7.9 COUNTERS

In Chapter 5 we dealt with circuits that perform arithmetic operations. We showed how adder/subtractor circuits can be designed, either using a simple cascaded (ripple-carry) structure that is inexpensive but slow or using a more complex carry-lookahead structure that is both more expensive and faster. In this section we examine special types of addition and subtraction operations, which are used for the purpose of counting. In particular, we want to design circuits that can increment or decrement a count by 1. Counter circuits are used in digital systems for many purposes. They may count the number of occurrences of certain events, generate timing intervals for control of various tasks in a system, keep track of time elapsed between specific events, and so on.

Counters can be implemented using the adder/subtractor circuits discussed in Chapter 5 and the registers discussed in section 7.8. However, since we only need to change the contents of a counter by 1, it is not necessary to use such elaborate circuits. Instead, we can use much simpler circuits that have a significantly lower cost. We will show how the counter circuits can be designed using T and D flip-flops.

7.9.1 ASYNCHRONOUS COUNTERS

The simplest counter circuits can be built using T flip-flops because the toggle feature is naturally suited for the implementation of the counting operation.

Up-Counter with T Flip-Flops

Figure 7.20*a* gives a three-bit counter capable of counting from 0 to 7. The clock inputs of the three flip-flops are connected in cascade. The T input of each flip-flop is connected to a constant 1, which means that the state of the flip-flop will be reversed (toggled) at each positive edge of its clock. We are assuming that the purpose of this circuit is to count the number of pulses that occur on the primary input called *Clock*. Thus the clock input of the first flip-flop is connected to the *Clock* line. The other two flip-flops have their clock inputs driven by the $\overline{Q}$ output of the preceding flip-flop. Therefore, they toggle their state whenever the preceding flip-flop changes its state from $Q = 1$ to $Q = 0$, which results in a positive edge of the $\overline{Q}$ signal.

Figure 7.20*b* shows a timing diagram for the counter. The value of Q_0 toggles once each clock cycle. The change takes place shortly after the positive edge of the *Clock* signal. The delay is caused by the propagation delay through the flip-flop. Since the second flip-flop is clocked by $\overline{Q}_0$, the value of Q_1 changes shortly after the negative edge of the Q_0 signal. Similarly, the value of Q_2 changes shortly after the negative edge of the Q_1 signal. If we look at the values $Q_2Q_1Q_0$ as the count, then the timing diagram indicates that the counting sequence is 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, and so on. This circuit is a modulo-8 counter. Because it counts in the upward direction, we call it an *up-counter*.