7. Latches and Flip-Flops

Latches and *flip-flops* are the basic elements for storing information. One latch or flip-flop can store one bit of information. The main difference between latches and flip-flops is that for latches, their outputs are constantly affected by their inputs as long as the enable signal is asserted. In other words, when they are enabled, their content changes immediately when their inputs change. Flip-flops, on the other hand, have their content change only either at the rising or falling edge of the enable signal. This enable signal is usually the controlling clock signal. After the rising or falling edge of the clock, the flip-flop content remains constant even if the input changes.

There are basically four main types of latches and flip-flops: SR, D, JK, and T. The major differences in these flip-flop types are the number of inputs they have and how they change state. For each type, there are also different variations that enhance their operations. In this chapter, we will look at the operations of the various latches and flip-flops.
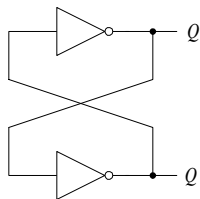
7.1 Bistable Element

The simplest sequential circuit or storage element is a *bistable element*, which is constructed with two inverters connected sequentially in a loop as shown in Figure 1. It has no inputs and two outputs labeled Q and Q'. Since the circuit has no inputs, we cannot change the values of Q and Q'. However, Q will take on whatever value it happens to be when the circuit is first powered up. Assume that $Q = 0$ when we switch on the power. Since Q is also the input to the bottom inverter, Q', therefore, is a 1. A 1 going to the input of the top inverter will produce a 0 at the output Q, which is what we started off with. Similarly, if we start the circuit with $Q = 1$, we will get $Q' = 0$, and again we get a stable situation.

A bistable element has memory in the sense that it can remember the content (or state) of the circuit indefinitely. Using the signal Q as the state variable to describe the state of the circuit, we can say that the circuit has two stable states: $Q = 0$, and $Q = 1$; hence the name "bistable."
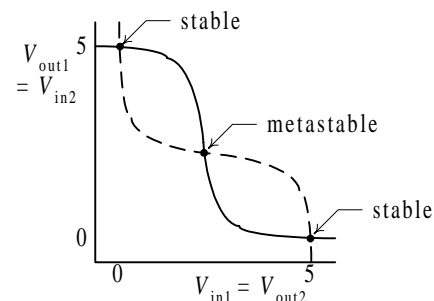
An analog analysis of a bistable element, however, reveals that it has three equilibrium points and not two as found from the digital analysis. Assuming again that $Q = 1$, and we plot the output voltage (V_{out1}) versus the input voltage (V_{in1}) of the top inverter, we get the solid line in Figure 2. The dotted line shows the operation of the bottom inverter where V_{out2} and V_{in2} are the output and input voltages respectively for that inverter.

Figure 2 shows that there are three intersection points, two of which corresponds to the two stable states of the circuit where Q is either 0 or 1. The third intersection point labeled *metastable*, is at a voltage that is neither a logical 1 nor a logical 0 voltage. Nevertheless, if we can get the circuit to operate at this voltage, then it can stay at that point indefinitely. Practically, however, we can never operate a circuit at precisely a certain voltage. A slight deviation from the metastable point as cause by noise in the circuit or other stimulants will cause the circuit to go to one of the two stable points. Once at the stable point, a slight deviation, however, will not cause the circuit to go away from the stable point but rather back towards the stable point because of the feedback effect of the circuit.

An analogy of the metastable behavior is a ball on top of a symmetrical hill as depicted in Figure 3. The ball can stay indefinitely in that precarious position as long as there is absolutely no movement whatsoever. With any slight force, the ball will roll down to either of the two sides. Once at the bottom of the hill, the ball will stay there until an external force is applied to it. The strength of this external force will cause the ball to do one of three things. If a

Figure 1. Bistable element.

Figure 2. Analog analysis of bistable element.
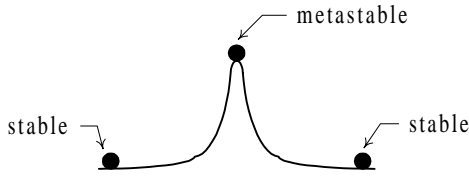
**Figure 3.** Ball and hill analogy for metastable behavior.

small force is applied to the ball, it will go partly up the hill and then rolls back down to the same side. If a big enough force is applied to it, it will go over the top and down the other side of the hill. We can also apply a force that is just strong enough to push the ball to the top of the hill. Again at this precarious position, it can roll down either side.

We will find that all latches and flip-flops have this metastable behavior. In order for the element to change state, we need to apply a strong enough pulse satisfying a given minimum width requirement. Otherwise, the element will either remain at the current state or go into the metastable state in which case unpredictable results can occur.
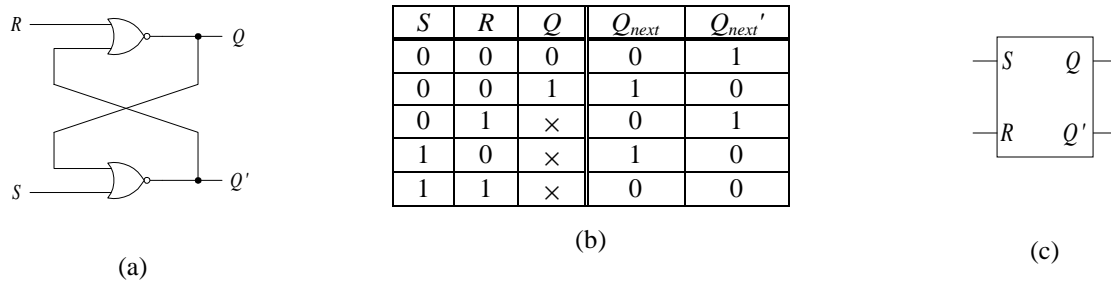
7.2 SR Latch

The bistable element is able to remember or store one bit of information. However, because it does not have any inputs, we cannot change the information bit that is stored in it. In order to change the information bit, we need to add inputs to the circuit. The simplest way to add inputs is to replace the two inverters with two NAND gates as shown in Figure 4(a). This circuit is called a *SR latch*. In addition to the two outputs Q and Q', there are two inputs S' and R' for *set* and *reset* respectively. Following the convention, the prime in S and R denotes that these inputs are active low. The SR latch can be in one of two states: a set state when $Q = 1$, or a reset state when $Q = 0$.

To make the SR latch go to the set state, we simply assert the S' input by setting it to 0. Remember that 0 NAND anything gives a 1, hence $Q = 1$ and the latch is set. If R' is not asserted ($R' = 1$), then the output of the bottom NAND gate will give a 0, and so $Q' = 0$. This situation is shown in Figure 4 (d) at time t_0. If we de-assert S' so that $S' = R' = 1$, the latch will remain at the set state because Q', the second input to the top NAND gate, is 0 which will keep $Q = 1$ as shown at time t_1. At time t_2 we reset the latch by making $R' = 0$. Now, Q' goes to 1 and this will force Q to go to a 0. If we de-assert R' so that again we have $S' = R' = 1$, this time the latch will remain at the reset state as shown at time t_3. Notice the two times (at t_1 and t_3) when both S' and R' are de-asserted. At t_1, Q is at a 1, whereas, at t_3, Q is at

(a)

S	R	Q	Q_{next}	Q_{next}'
0	0	×	1	1
0	1	×	1	0
1	0	×	0	1
1	1	0	0	1
1	1	1	1	0

(b)

(c)

(d)
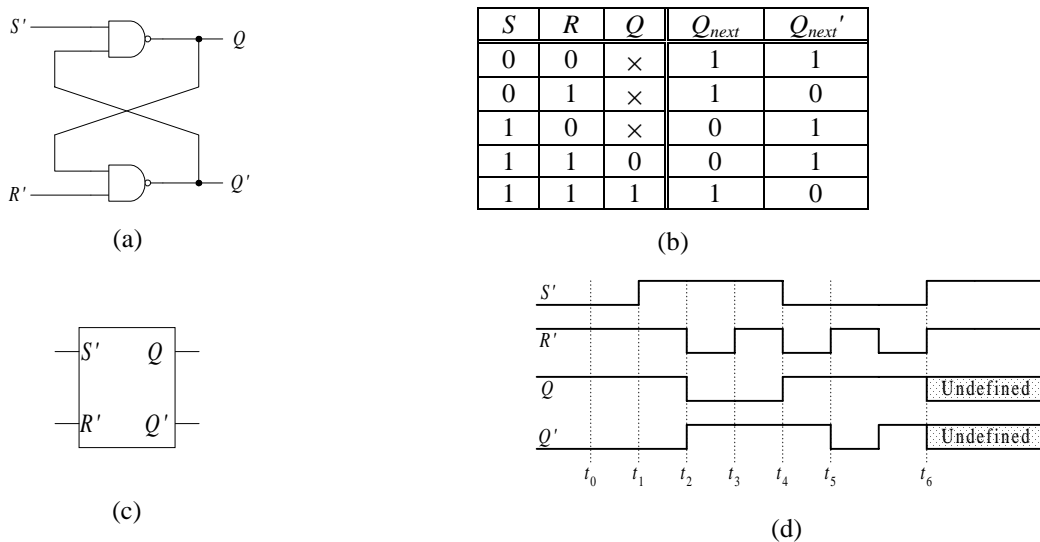
Figure 4.   SR latch: (a) circuit using NAND gates; (b) truth table; (c) logic symbol; (d) timing diagram.

(a)

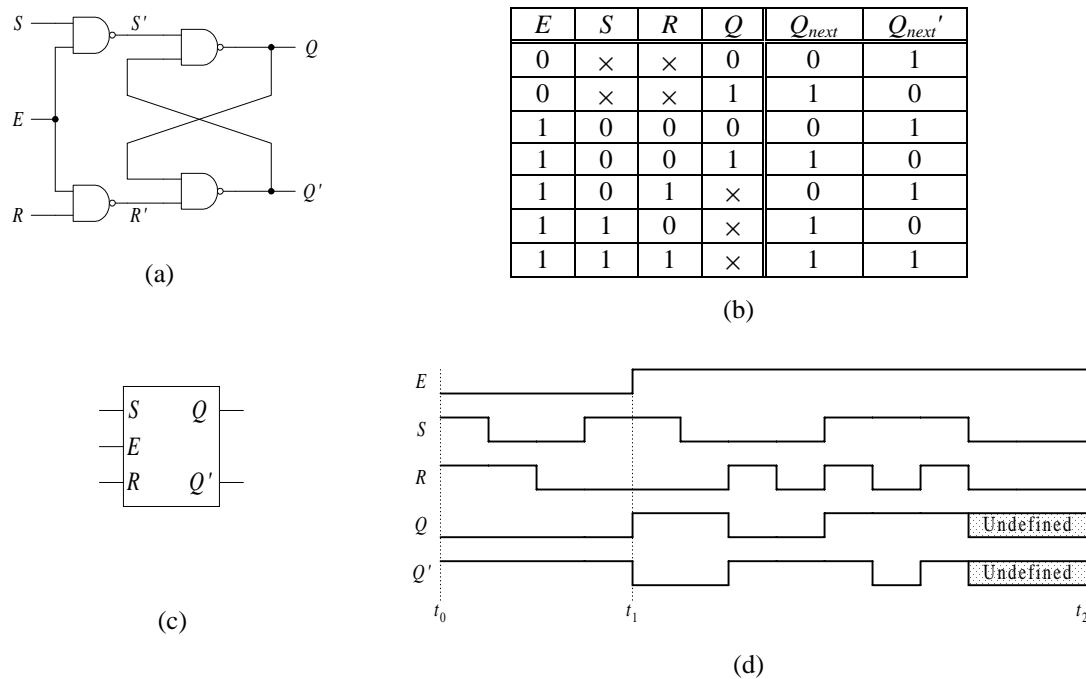| S	R	Q	Q_{next}	Q_{next}'
0	0	0	0	1
0	0	1	1	0
0	1	×	0	1
1	0	×	1	0
1	1	×	0	0

(b)

(c)

Figure 5. SR latch: (a) circuit using NOR gates; (b) truth table; (c) logic symbol.

a 0. When both inputs are de-asserted, the SR latch maintains its previous state. Previous to t_1, Q has the value 1, so at t_1, Q remains at a 1. Similarly, previous to t_3, Q has the value 0, so at t_3, Q remains at a 0.

If both S' and R' are asserted, then both Q and Q' are equal to 1 as shown at time t_4. If one of the input signals is de-asserted earlier than the other, the latch will end up in the state forced by the signal that was de-asserted later as shown at time t_5. At t_5, R' is de-asserted first, so the latch goes into the normal set state with $Q = 1$ and $Q' = 0$.

A problem exists if both S' and R' are de-asserted at exactly the same time as shown at time t_6. If both gates have exactly the same delay then they will both output a 0 at exactly the same time. Feeding the zeros back to the gate input will produce a 1, again at exactly the same time, which again will produce a 0, and so on and on. This oscillating behavior, called the *critical race*, will continue forever. If the two gates do not have exactly the same delay then the situation is similar to de-asserting one input before the other, and so the latch will go into one state or the other. However, since we do not know which is the faster gate, therefore, we do not know which state the latch will go into. Thus, the latch's next state is undefined.

In order to avoid this indeterministic behavior, we must make sure that the two inputs are never de-asserted at the same time. Note that both of them can be de-asserted, but just not at the same time. In practice, this is guaranteed by not having both of them asserted. Another reason why we do not want both inputs to be asserted is that when they are both asserted, Q is equal to Q', but we usually want Q to be the inverse of Q'.

(a)

E	S	R	Q	Q_{next}	Q_{next}'
0	×	×	0	0	1
0	×	×	1	1	0
1	0	0	0	0	1
1	0	0	1	1	0
1	0	1	×	0	1
1	1	0	×	1	0
1	1	1	×	1	1

(b)

(c)

(d)

Figure 6. SR latch with enable: (a) circuit using NAND gates; (b) truth table; (c) logic symbol; (d) timing diagram.

(a)

E	S	R	Q	Q_{next}	Q_{next}'
0	×	×	0	0	1
0	×	×	1	1	0
1	0	0	0	0	1
1	0	0	1	1	0
1	0	1	×	0	1
1	1	0	×	1	0
1	1	1	×	0	0

(b)
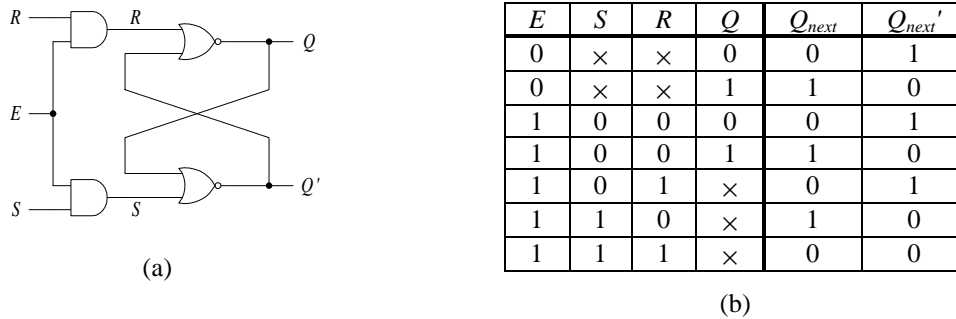
Figure 7. SR latch with enable: (a) circuit using NOR gates; (b) truth table.

From the above analysis, we obtain the truth table in Figure 4(b) for the NAND implementation of the SR latch. Q is the current state or the current content of the latch and Q_{next} is the value to be updated in the next state. Figure 4(c) shows the logic symbol for the SR latch.

The SR latch can also be implemented using NOR gates as shown in Figure 5(a). The truth table for this implementation is shown in Figure 5(b). From the truth table, we see that the main difference between this implementation and the NAND implementation is that for the NOR implementation, the S and R inputs are active high, so that setting S to 1 will set the latch and setting R to 1 will reset the latch. However, just like the NAND implementation, the latch is set when $Q = 1$ and reset when $Q = 0$. The latch remembers its previous state when $S = R = 0$. When $S = R = 1$, both Q and Q' are 0. The logic symbol for the SR latch using NOR implementation is shown in Figure 5(c).

7.3   SR Latch with Enable

The SR latch is sensitive to its inputs all the time. It is sometimes useful to be able to disable the inputs. The *SR latch with enable* (also known as a *gated SR latch*) accomplishes this by adding an enable input, *E*, to the original implementation of the latch that allows the latch to be enabled or disabled. The circuit for the SR latch with enable using NAND gates is shown in Figure 6(a), its truth table in Figure 6(b), and logic symbol in Figure 6(c). When $E = 1$, the circuit behaves like the normal NAND implementation of the SR latch except that the S and R inputs are active high rather than low. When $E = 0$, the latch remains in its previous state regardless of the S and R inputs. In actual circuits, the enable input can either be active high or low, and may be named *ENABLE*, *CLK*, or *CONTROL*. A typical operation of the latch is shown in the timing diagram in Figure 6(d). Between t_0 and t_1, $E = 0$ so changing the S and R inputs do not affect the output. Between t_1 and t_2, $E = 1$ and the trace is similar to the trace of Figure 4(d) except that the input signals are inverted.

The SR latch with enable can also be implemented using NOR gates as shown Figure 7.

7.4   D Latch

D	Q	Q_{next}	Q_{next}'
0	×	0	1
1	×	1	0

(c)

(a)

(b)

(d)
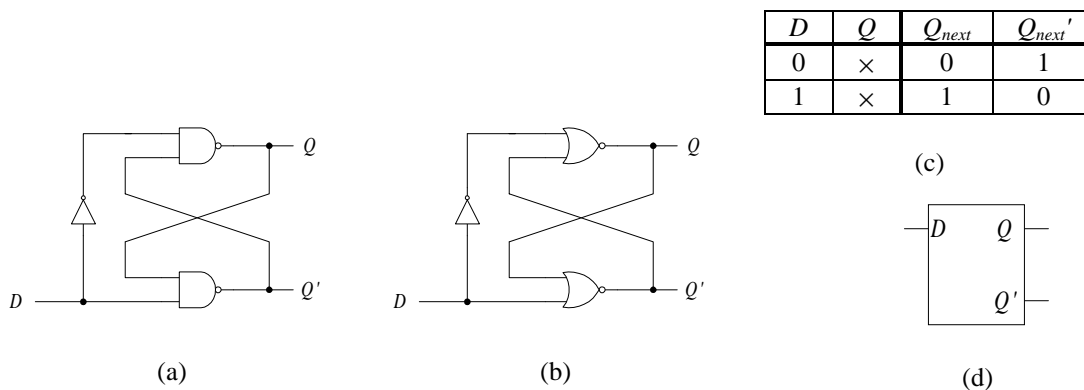
Figure 8. D latch: (a) circuit using NAND gates; (b) circuit using NOR gates; (c) truth table; (d) logic symbol.

E	D	Q	Q_{next}	Q_{next}'
0	×	0	0	1
0	×	1	1	0
1	0	×	0	1
1	1	×	1	0

(a)

(b)

(c)

(d)
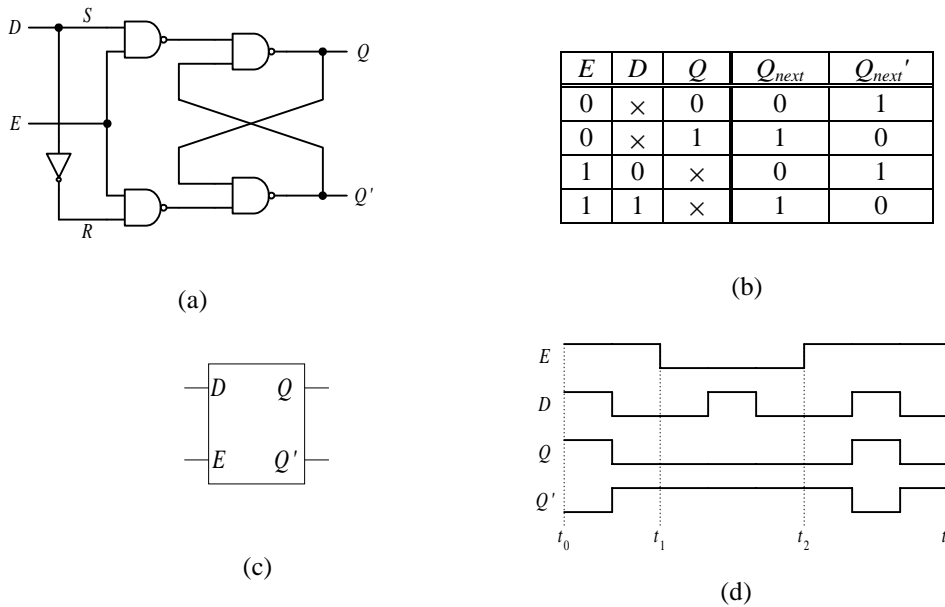
Figure 9. D latch with enable: (a) circuit using NAND gates; (b) truth table; (c) logic symbol; (d) timing diagram.

The disadvantage with the SR latch is that we need to ensure that the two inputs, S and R, are never de-asserted at the same time. This situation is prevented in the *D latch* by adding an inverter between the original S and R inputs and replacing them with just one input D (for *data*) as shown in Figure 8(a) and (b).

Notice that the placement of the inverter with respect to the Q output is such that the Q output value follows the D input. This feature is useful because, whereas the SR latch is useful for setting or resetting a flag on a given condition, the D latch is useful for simply storing a bit of information that is presented on a line. Figure 8(c) shows the truth table for the D latch, and Figure 8(d) shows the graphic symbol.

7.5   D Latch with Enable

Just like the SR latch with an enable input, the D latch can also have an enable input as shown in Figure 9(a). When the E input is asserted ($E = 1$), the Q output follows the D input. In this situation, the latch is said to be "open" and the path from the input D to the output Q is "transparent". Hence the circuit is often referred to as a *transparent latch*. When E is de-asserted ($E = 0$), the latch is disabled or "closed", and the Q output retains its last value independent of the D input. A sample timing diagram for the operation of the D latch with enable is shown in Figure 9(d). Between t_0 and t_1, the latch is enabled with $E = 1$ so the output Q follows the input D. Between t_1 and t_2, the latch is disabled, so Q remains stable even when D changes.
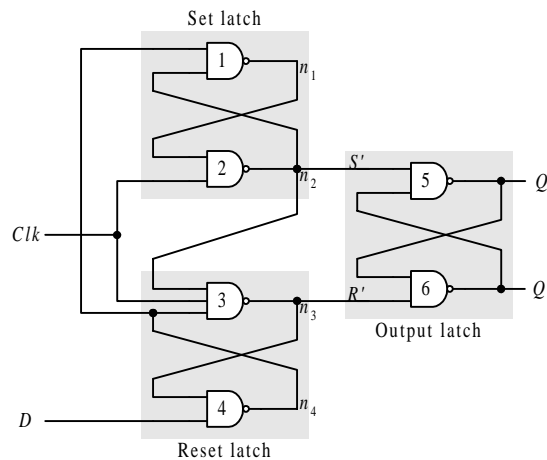
7.6   D Flip-Flop

Latches are often called *level-sensitive* because their output follows their inputs as long as they are enabled. They are transparent during this entire time when the enable signal is asserted. There are situations when it is more useful to have the output change only at the rising or falling edge of the enable signal. This enable signal is usually the controlling clock signal. Thus, we can have all changes synchronized to the rising or falling edge of the clock. An *edge-triggered flip-flop* achieves this by combining in series a pair of latches. Figure 10(a) shows a *positive-edge-triggered D flip-flop* where two D latches are connected in series and a clock signal Clk is connected to the E input of the latches, one directly, and one through an inverter. The first latch is called the *master* latch. The master latch is enabled when $Clk = 0$ and follows the primary input D. When Clk is a 1, the master latch is disabled but the second latch, called the *slave* latch, is enabled so that the output from the master latch is transferred to the slave latch. The slave latch is enabled all the while that $Clk = 1$, but its content changes only at the beginning of the cycle, that is, only at the rising edge of the signal because once Clk is 1, the master latch is disabled and so the input to the

slave latch will not change. The circuit of Figure 10(a) is called a *positive* edge-triggered flip-flop because the output Q on the slave latch changes only at the rising edge of the clock. If the slave latch is enabled when the clock is low, then it is referred to as a *negative* edge-triggered flip-flop. The circuit of Figure 10(a) is also referred to as a *master-slave* D flip-flop because of the two latches used in the circuit. Figure 10(b) and (c) show the truth table and the logic symbol respectively. Figure 10(d) shows the timing diagram for the D flip-flop.

Another way of constructing a positive-edge-triggered flip-flop is to use three interconnected SR latches rather than a master and slave D latch with enable. The circuit is shown in Figure 11. The advantage of this circuit is that it uses only 6 NAND gates (26 transistors) as opposed to 10 gates (46 transistors) for the master-slave D flip-flop of Figure 10(a). The operation of the circuit is as follows. When $E = 0$, the outputs of gates 2 and 3 are high (0 NAND x = 1). Thus $n_2 = n_3 = 1$, which maintains the output latch, comprising gates 5 and 6, in its current state. At the same time $n_4 = D'$ since one input to gate 4 is n_3 which is a 1 (1 NAND $x = x'$). Similarly, $n_1 = D$. When E changes to 1, n_2 will be equal to $n_1' = D'$, while n_3 will be equal to D. So if $D = 0$, then n_3 will be 0, thus asserting R' and resetting the output latch Q to 0. On the other hand, if $D = 1$, then n_2 will be 0, thus asserting S' and setting the output latch Q to 1. Once $E = 1$, changing D will not change n_2 or n_3, so Q will remain stable during the remaining time that E is asserted.

(a)

Clk	D	Q	Q_{next}	Q_{next}'
0	×	0	0	1
0	×	1	1	0
1	×	0	0	1
1	×	1	1	0
↑	0	×	0	1
↑	1	×	1	0

(b)

(c)
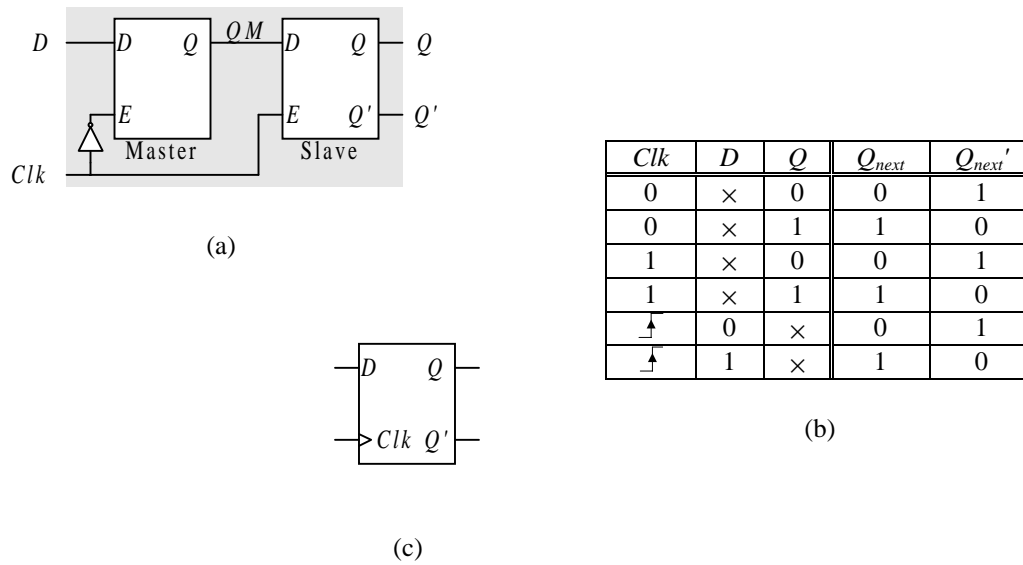
Figure 10.   Master-slave positive-edge-triggered D flip-flop: (a) circuit using D latches; (b) truth table; (c) logic symbol; (d) timing diagram.

Figure 11.   Positive-edge-triggered D flip-flop.

(a)   (b)
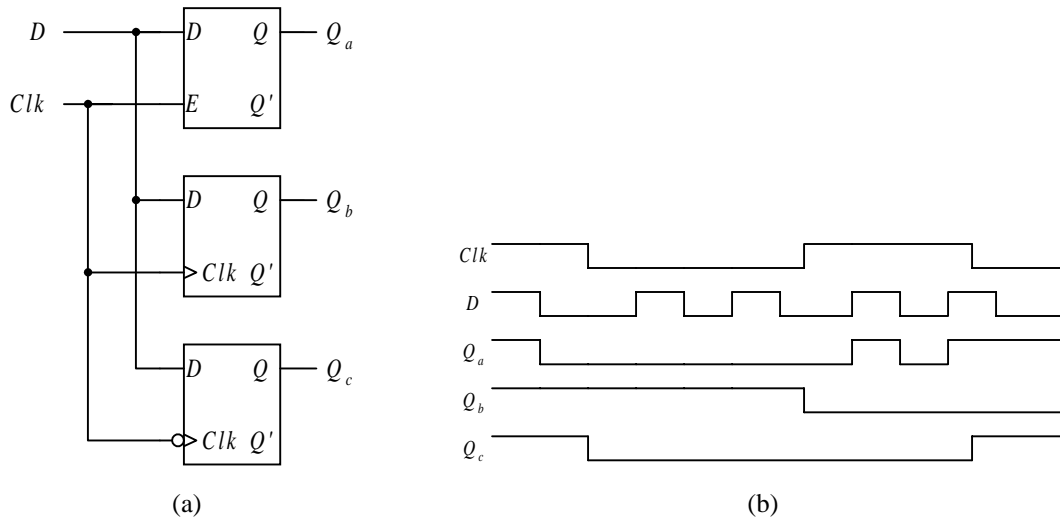
Figure 12.   Comparison of a gated latch, a positive-edge-triggered flip-flop, and a negative-edge-triggered flip-flop: (a) circuit; (b) timing diagram.

Figure 12 compares the different operations between a latch and a flip-flop. In (a), we have a gated D latch, a positive-edge-triggered D flip-flop and a negative-edge-triggered D flip-flop, all having the same D input and controlled by the same clock signal. (b) shows a sample trace of the circuit's operations. Notice that the gated D latch Q_a follows the D input as long as the clock is high. The positive-edge-triggered flip-flop Q_b responds to the D input only at the rising edge of the clock while the negative-edge-triggered flip-flop Q_c responds to the D input only at the falling edge of the clock.

7.7   D Flip-Flop with Enable

A commonly desired function in D flip-flops is the ability to hold the last value stored rather than load in a new value at the clock edge. This is accomplished by adding an enable input called *EN* or *CE* (clock enable) through a multiplexer as shown in Figure 13(a). When *EN* = 1, the primary D signal will pass to the D input of the flip-flop, thus updating the content of the flip-flop. When *EN* = 0, the bottom AND gate is enabled and so the current content of the flip-flop, Q, is passed back to the input, thus, keeping its current value. Notice that changes to the flip-flop value occur only at the rising edge of the clock. The truth table and the logic symbol for the D flip-flop with enabled is shown in (b) and (c) respectively.

Clk	EN	D	Q	Q_{next}	Q_{next}'
0	×	×	0	0	1
0	×	×	1	1	0
1	×	×	0	0	1
1	×	×	1	1	0
⤒	0	×	0	0	1
⤒	0	×	1	1	0
⤒	1	0	×	0	1
⤒	1	1	×	1	0

(a)   (b)   (c)
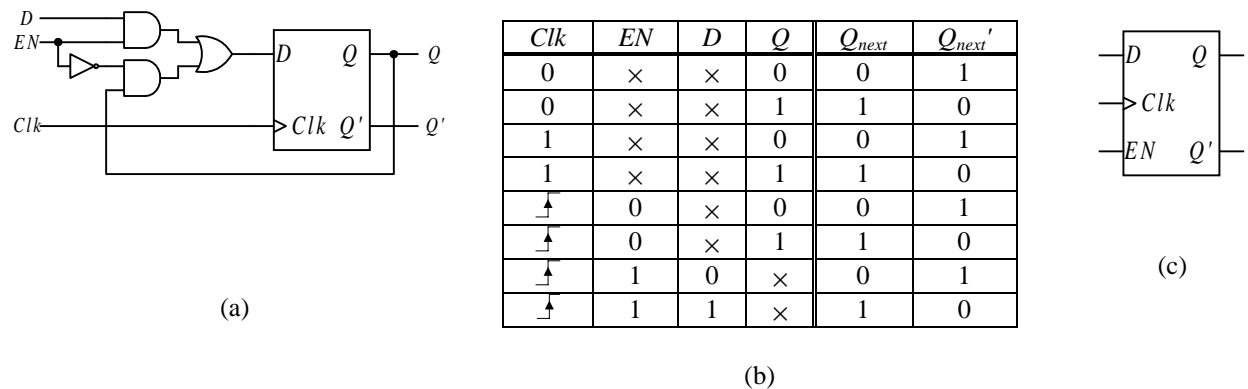
Figure 13.   D flip-flop with enable: (a) circuit; (b) truth table; (c) logic symbol.

7.8   Asynchronus Inputs

Flip-flops, as we have seen so far, change states at the edge of a synchronizing clock signal. Many circuits require the initialization of flip-flops to a known state independent of the clock signal. Sequential circuits that change states whenever a change in input values occurs independent of the clock are referred to as *asynchronous sequential circuits*. *Synchronous sequential circuits*, on the other hand, change states only at the edge of the clock signal. Asynchronous inputs are usually available for both flip-flops and latches, and they are used to either set or clear the storage element's content independent of the clock.

Figure 14(a) shows a D latch with asynchronous *PRESET'* and *CLEAR'* inputs, and (b) is the logic symbol for it. (c) is the circuit for the D edge-triggered flip-flop with asynchronous *PRESET'* and *CLEAR'* inputs, and (d) is the logic symbol for it. When *PRESET'* is asserted (set to 0) the content of the storage element is set to a 1 immediately, and when *CLEAR'* is asserted (set to 0) the content of the storage element is set to a 0 immediately.
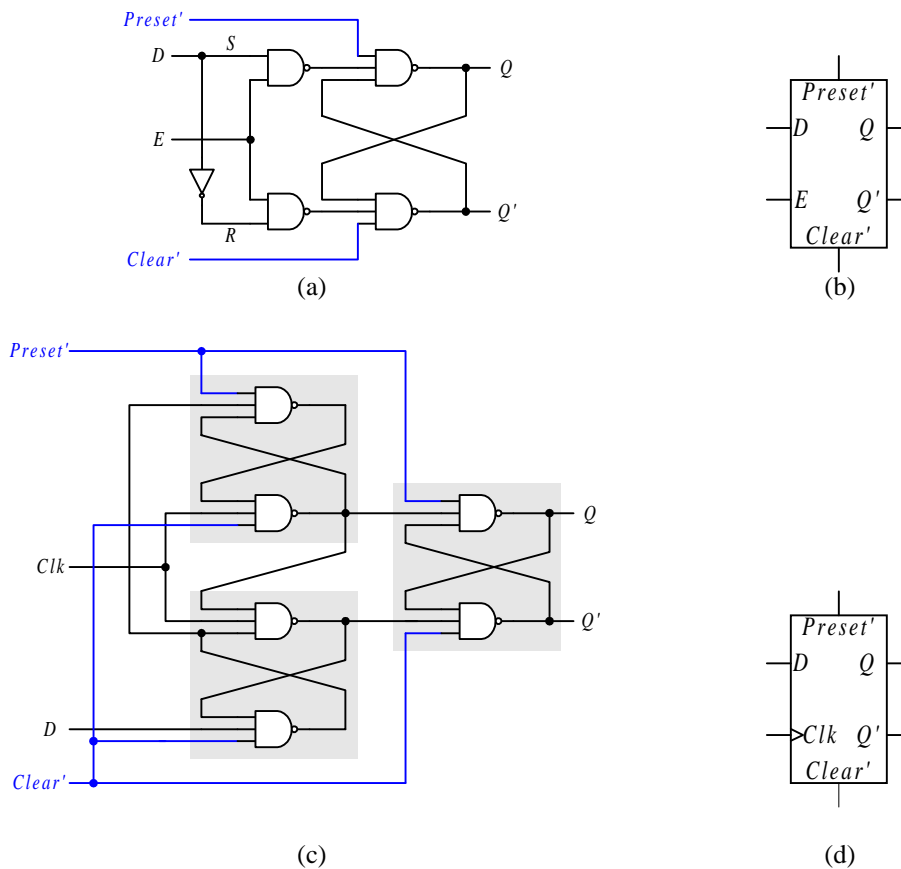
Figure 14.   Storage elements with asynchronous inputs: (a) D latch with preset and clear; (b) logic symbol for (a); (c) D edge-triggered flip-flop with preset and clear; (d) logic symbol for (c).

7.9   Flip-Flop Types

There are basically four main types of flip-flops: SR, D, JK, and T. The major differences in these flip-flop types are in the number of inputs they have and how they change state. Each type can have different variations such as active high or low inputs, whether they change state at the rising or falling edge of the clock signal, and whether they have asynchronous inputs or not. The flip-flops can be described fully and uniquely by its logic symbol, characteristic table, characteristic equation, state diagram, or excitation table, and are summarized in Figure 15.
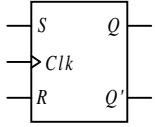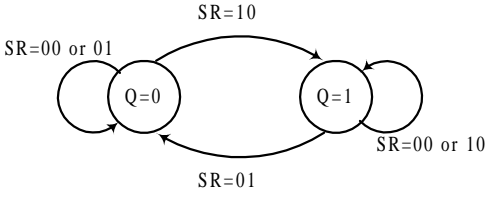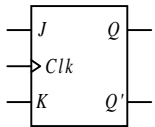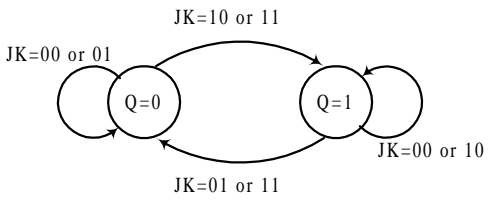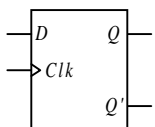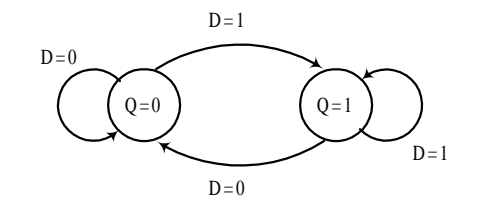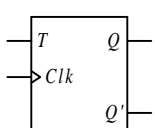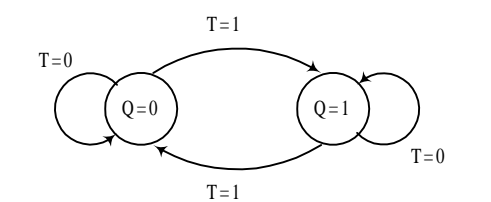
Name / Symbol	Characteristic (Truth) Table	State Diagram / Characteristic Equations	Excitation Table

SR

Symbol:
- S — Q
- Clk (clock)
- R — Q'

Characteristic (Truth) Table:

S	R	Q	Q_{next}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	$\times$
1	1	1	$\times$

State Diagram:

SR=10, SR=00 or 01, Q=0, Q=1, SR=00 or 10, SR=01

$$Q_{next} = S + R'Q$$
$$SR = 0$$

Excitation Table:

Q	Q_{next}	S	R
0	0	0	$\times$
0	1	1	0
1	0	0	1
1	1	$\times$	0

JK

Symbol:
- J — Q
- Clk
- K — Q'

Characteristic (Truth) Table:

J	K	Q	Q_{next}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

State Diagram:

JK=10 or 11, JK=00 or 01, Q=0, Q=1, JK=00 or 10, JK=01 or 11

$$\begin{aligned}
Q_{next} &= J'K'Q + JK' + JKQ' \\
&= J'K'Q + JK'Q + JK'Q' + JKQ' \\
&= K'Q(J'+J) + JQ'(K'+K) \\
&= K'Q + JQ'
\end{aligned}$$

Excitation Table:

Q	Q_{next}	J	K
0	0	0	$\times$
0	1	1	$\times$
1	0	$\times$	1
1	1	$\times$	0

D

Symbol:
- D — Q
- Clk
- Q'

Characteristic (Truth) Table:

D	Q	Q_{next}
0	$\times$	0
1	$\times$	1

State Diagram:

D=1, D=0, Q=0, Q=1, D=1, D=0

$$Q_{next} = D$$

Excitation Table:

Q	Q_{next}	D
0	0	0
0	1	1
1	0	0
1	1	1

T

Symbol:
- T — Q
- Clk
- Q'

Characteristic (Truth) Table:

T	Q	Q_{next}
0	0	0
0	1	1
1	0	1
1	1	0

State Diagram:

T=1, T=0, Q=0, Q=1, T=0, T=1

$$Q_{next} = TQ' + T'Q = T \oplus Q$$

Excitation Table:

Q	Q_{next}	T
0	0	0
0	1	1
1	0	1
1	1	0

Figure 15. Flip-flop types.

7.9.1 SR Flip-Flop

We can replace the D latches in the D flip-flop of Figure 10(a) with SR latches to get a master-slave SR flip-flop shown in Figure 16. Like SR latches, SR flip-flops are useful in control applications where we want to be able to set or reset the data bit. However, unlike SR latches, SR flip-flops change their content only at the active edge of the clock signal. Similar to SR latches, SR flip-flops can enter an undefined state when both inputs are asserted simultaneously.

S	R	Q	Q_{next}	Q_{next}'
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	$\times$	$\times$
1	1	1	$\times$	$\times$

(a)

(b)

(c)
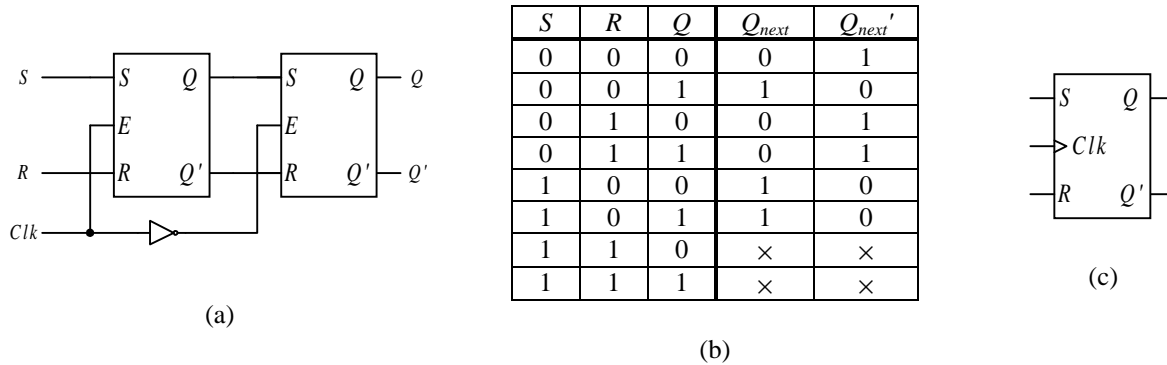
Figure 16. SR flip-flop: (a) circuit; (b) truth table; (c) logic symbol.

7.9.2 JK Flip-Flop

JK flip-flops are very similar to SR flip-flops. The J input is just like the S input in that when asserted, it sets the flip-flop. Similarly, the K input is like the R input where it clears the flip-flop when asserted. The only difference is when both inputs are asserted. For the SR flip-flop, the next state is undefined, whereas, for the JK flip-flop, the next state is the inverse of the current state. In other words, the JK flip-flop toggles its state when both inputs are asserted. The circuit, truth table and the logic symbol for the JK flip-flop is shown in Figure 17.

J	K	Q	Q_{next}	Q_{next}'
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

(a)

(b)

(c)
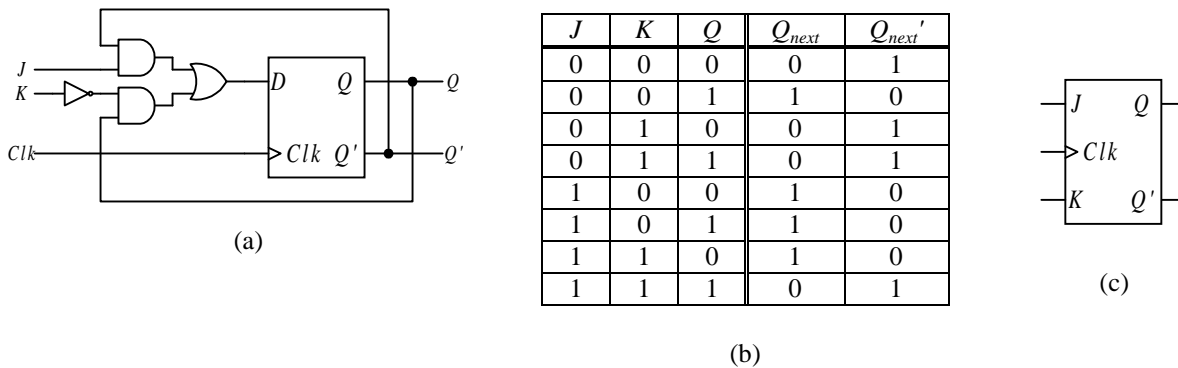
Figure 17. JK flip-flop: (a) circuit; (b) truth table; (c) logic symbol.

7.9.3 T Flip-Flop

The T flip-flop has one input in addition to the clock. T stands for toggle for the obvious reason. When T is asserted ($T = 1$), the flip-flop state toggles back and forth, and when T is de-asserted, the flip-flop keeps its current state. The T flip-flop can be constructed using a D flip-flop with the two outputs Q and Q' feedback to the D input through a multiplexer that is controlled by the T input as shown in Figure 18.
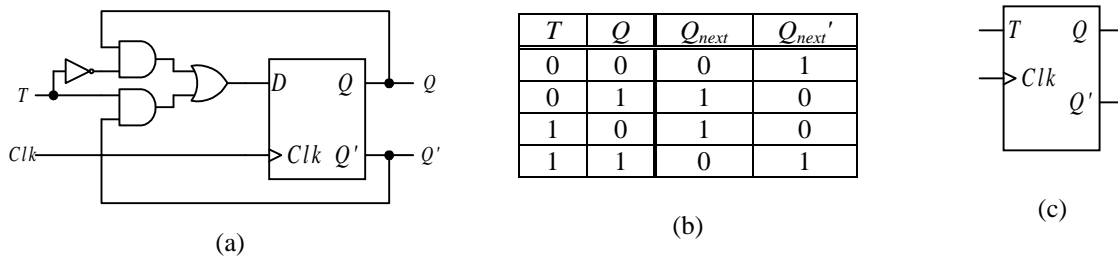
Figure 18.   T flip-flop: (a) circuit; (b) truth table; (c) logic symbol.

7.9.4 Logic Symbol

The *logic* or *graphical symbol* describes the flip-flop's inputs and outputs, the names given to these signals, and whether they are active high or low. All the flip-flops have Q and Q' as their outputs. All of them also have a *CLK* input. The small triangle at the clock input indicates that the circuit is a flip-flop and so it is triggered by the edge of the clock signal; if there is a circle in front, then it is the falling edge, otherwise, it is the rising edge of the clock
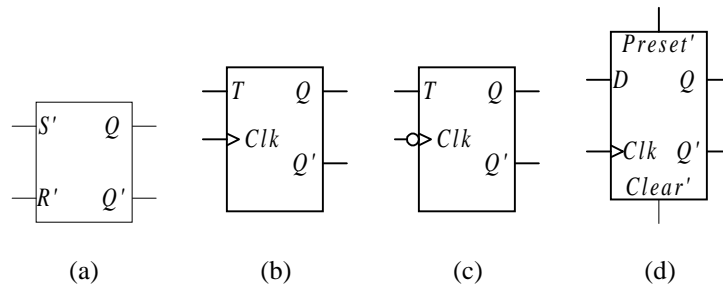
Figure 19.   Various logic symbols: (a) Active low SR latch; (b) positive-edge-triggered active
high T flip-flop; (c) negative-edge-triggered T flip-flop; (d) positive-edge-triggered
D flip-flop with asynchronous active low preset and clear.

signal. Without the small triangle, the circuit is a latch. In addition, the flip-flops have one or two more inputs that characterize the flip-flop and give it its name. Figure 19 shows several sample logic symbols for various memory elements.

7.9.5 Characteristic Table

The *characteristic table* is just the truth table but usually written in a shorter format. For example, compare the characteristic table for the JK flip-flop in Figure 20 with the truth table in Figure 17(b). The truth table, as we have seen, simply lists all possible combinations of the input signals, the current state (or content) of the flip-flop, and the next state that the flip-flop will go to at the next active edge of the clock signal. The characteristic table answers the question of what is the next state when given the inputs and the current state, and is used in the analysis of sequential circuits.

J	K	Q_{next}
0	0	Q
0	1	0
1	0	1
1	1	Q'

Figure 20.   JK flip-flop characteristic table.

7.9.6 Characteristic Equation

The *characteristic equation* is the functional Boolean equation that is derived from the characteristic table. This equation formally describes the functional behavior of the flip-flop. Like the characteristic table, it specifies the flip-flop's next state as a function of its current state and inputs. For example, the characteristic equation for the JK flip-flop can be derived from the truth table as follows:

$$Q_{next} \quad = J'K'Q + JK'Q + JK'Q' + JKQ'$$
$$= K'Q(J'+J) + JQ'(K'+K)$$
$$= K'Q + JQ'$$

The characteristic equation can also be obtained from the truth table using the K-map method as follows for the SR flip-flop:

Thus, the characteristic equation for the SR flip-flop is

$$Q_{next} \quad = S + R'Q.$$

7.9.7 State Diagram

A *state diagram* is a graph that shows the flip-flop's operations in terms of how it transitions from one state to another. The nodes are labeled with the states and the directed arcs are labeled with the input signals that cause the transition to go from one state to the next. Figure 21 shows the state diagram for the SR flip-flop. For example, to go from state $Q = 0$ to the state $Q = 1$, the two inputs S and R have to be 1 and 0 respectively. Similarly, if the current state is $Q = 0$ and we want to remain in that state, then SR need to be 00 or 01.
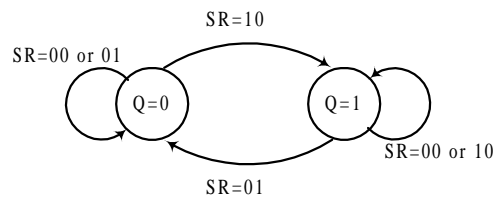
Figure 21.   State diagram for the SR flip-flop.

7.9.8 Excitation Table

The *excitation table* gives the value of the flip-flop's inputs that are necessary to change the flip-flop's current state to the desired next state at the next active edge of the clock signal. The excitation table answers the question of what should the inputs be when given the current state that the flip-flop is in and the next state that we want the flip-flop to go to. This table is used in the synthesis of sequential circuits.

Figure 22 shows the excitation table for the SR flip-flop. As can be seen, this table can be obtained directly from the state diagram. For example, if the current state is $Q = 0$ and we want the next state to be $Q = 1$, then the two inputs must be $SR = 10$.

Q	Q_{next}	S	R
0	0	0	$\times$
0	1	1	0
1	0	0	1
1	1	$\times$	0

Figure 22.   SR flip-flop excitation table.

7.10 VHDL for Latches and Flip-Flops

7.10.1 Implied Memory Element

VHDL does not have any explicit object for defining a memory element. Instead, the semantics of the language provides for signals to be interpreted as a memory element. In other words, memory element is declared depending on how these signals are assigned. Consider the code in Figure 23.

```
ENTITY no_memory_element IS
  PORT (A, B : IN STD_LOGIC;
        C : OUT STD_LOGIC);
END no_memory_element;

ARCHITECTURE Behavior OF no_memory_element IS
BEGIN
  PROCESS(A, B)
  BEGIN
    C <= '1';   -- assigns default value to C
    IF A = B THEN
      C <= '0';
    END IF;
  END PROCESS;
END Behavior;
```

Figure 23.   Sample VHDL description of a combinational circuit.

The process assigns the default value of 1 to C and then if A is equal to B then it changes the value of C to a 0. In this code, C will be assigned a value for all possible outcomes of the test A = B. With this construct, a combinational circuit is produced.

If we simply remove the statement that assigns the default value to C, then we have a situation where no value will be assigned to C if A is not equal to B. The key point here is that the VHDL semantics stipulate that in cases where the code does not specify a value of a signal, the signal should retain its current value. In other words, the signal must remember its current value, and in order to do so, a memory element is implied.

7.10.2 VHDL Code for a D Latch

Figure 24 shows the VHDL code for a D latch with enable. If *Enable* is 1 then *Q* follows *D*. However, if *Enable* is not 1, the code does not specify what *Q* should be, therefore, *Q* retains its current value. This code produces a latch and not a flip-flop because *Q* follows *D* as long as *Enable* is 1, and not only at the active edge of the signal. The process sensitivity list includes both *D* and *Enable* because either one of these signals can cause a change in the value of the *Q* output.

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY D_latch_with_enable IS
   PORT(D, Enable : IN STD_LOGIC;
        Q : OUT STD_LOGIC);
END D_latch_with_enable;

ARCHITECTURE Behavior OF D_latch_with_enable IS
BEGIN
   PROCESS(D, Enable)
   BEGIN
     IF Enable = '1' THEN
       Q <= D;
     END IF;
   END PROCESS;
END Behavior;
```

Figure 24. VHDL code for a gated D latch.

7.10.3 VHDL Code for a D Flip-Flop

Figure 25 shows the behavioral VHDL code for a positive-edge-triggered D flip-flop. The only difference here is that *Q* follows *D* only at the rising edge of the clock, and it is specified here by the condition "Clock'EVENT AND Clock = '1'." The 'EVENT attribute refers to any changes in the qualifying clock signal. So when this happens and the resulting clock value is a one, we have in effect, a condition for a positive or rising clock edge. Note also that the process sensitivity list contains only the clock signal because it is the only signal that can cause a change in the *Q* output.

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY D_flipflop IS
   PORT(D, Clock : IN STD_LOGIC;
        Q : OUT STD_LOGIC);
END D_flipflop;

ARCHITECTURE Behavior OF D_flipflop IS
BEGIN
   PROCESS(Clock)
   BEGIN
     IF Clock'EVENT AND Clock = '1' THEN
       Q <= D;
     END IF;
   END PROCESS;
END Behavior;
```

Figure 25. VHDL code for a positive-edge-triggered D flip-flop using an IF statement.

Another way to describe a flip-flop is to use the WAIT statement instead of the IF statement as shown in Figure 26. When execution reaches the WAIT statement, it stops until the condition in the statement is true before proceeding. Note also that the process sensitivity list is omitted because the WAIT statement implies that the sensitivity list contains only the clock signal.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY D_flipflop IS
  PORT(D, Clock : IN STD_LOGIC;
       Q : OUT STD_LOGIC);
END D_flipflop;

ARCHITECTURE Behavior OF D_flipflop IS
BEGIN
  PROCESS
  BEGIN
    WAIT UNTIL Clock'EVENT AND Clock = '0'     -- negative edge triggered
    Q <= D;
  END PROCESS;
END Behavior;
```

Figure 26. VHDL code for a negative-edge-triggered D flip-flop using a WAIT statement.

Alternatively, we can write a structural VHDL description for the positive-edge-triggered D flip-flop as shown in Figure 27. This VHDL code is based on the circuit for a positive-edge-triggered D flip-flop as given in Figure 11.

```
-- define the behavioral operation of the 2-input NAND gate
LIBRARY ieee;
USE IEEE.std_logic_1164.all;

ENTITY NAND2 IS
  PORT(I0, I1 : IN STD_LOGIC;
       O : OUT STD_LOGIC);
END NAND2;

ARCHITECTURE Behavioral_NAND2 OF NAND2 IS
BEGIN
  O <= I1 NAND I2;
END Behavioral_NAND2;

-- define the behavioral operation of the 3-input NAND gate
LIBRARY ieee;
USE IEEE.std_logic_1164.all;

ENTITY NAND3 IS
  PORT(I0, I1, I2 : IN STD_LOGIC;
       O : OUT STD_LOGIC);
END NAND3;

ARCHITECTURE Behavioral_NAND3 OF NAND3 IS
BEGIN
  O <= NOT (I1 AND I2 AND I3);
END Behavioral_NAND3;
```

Figure 27. Structural VHDL code for a positive-edge-triggered D flip-flop.

```
-- define the structural operation of the SR latch
LIBRARY ieee;
USE IEEE.std_logic_1164.all;

ENTITY SRlatch IS
  PORT(SN, RN : IN STD_LOGIC;
      Q, QN : OUT STD_LOGIC);
END SRlatch;

ARCHITECTURE Structural_SRlatch OF SRlatch IS
  COMPONENT NAND2 PORT (I0, I1 : IN STD_LOGIC;
                        O : OUT STD_LOGIC);
  END COMPONENT;
BEGIN
  U1: NAND2 PORT MAP (SN, QN, Q);
  U2: NAND2 PORT MAP (Q, RN, QN);
END Structural_SRlatch;


-- define the structural operation of the positive edge triggered
-- D flip-flop
LIBRARY ieee;
USE IEEE.std_logic_1164.all;

ENTITY positive_edge_triggered_D_flipflop IS
  PORT(D, Clock : IN STD_LOGIC;
    Q, QN : OUT STD_LOGIC);
END positive_edge_triggered_D_flipflop;

ARCHITECTURE Structural OF positive_edge_triggered_D_flipflop IS
  SIGNAL N1, N2, N3, N4 : STD_LOGIC;

  COMPONENT SRlatch PORT (SN, RN : IN STD_LOGIC;
                          Q, QN  : OUT STD_LOGIC);
  END COMPONENT;
  COMPONENT NAND2 PORT (I0, I1 : IN STD_LOGIC;
                        O : OUT STD_LOGIC);
  END COMPONENT;
  COMPONENT NAND3 PORT (I0, I1, I2 : IN STD_LOGIC;
                        O : OUT STD_LOGIC);
  END COMPONENT;

BEGIN
  U1: SRlatch PORT MAP (N4, Clock, N1, N2);     -- set latch
  U2: SRlatch PORT MAP (N2, N3, Q, QN);         -- output latch
  U3: NAND3 PORT MAP (N2, Clock, N4, N3);       -- reset latch
  U4: NAND2 PORT MAP (N3, D, N4);
END Structural;
```

Figure 27 (continue). Structural VHDL code for a positive-edge-triggered D flip-flop.

7.10.4 VHDL Code for a D Flip-Flop with Asynchronous Inputs

Figure shows the VHDL code for a positive-edge-triggered D flip-flop with asynchronous active low reset and clear inputs. The two asynchronous inputs are checked for independently of the clock event. When either the *Reset* or the *Clear* input is asserted, *Q* is set to a 1 or 0 respectively immediately. Otherwise *Q* follows *D* at the rising edge of the clock.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY D_flipflop IS
  PORT(D, Clock, Reset, Clear : IN STD_LOGIC;
        Q : OUT STD_LOGIC);
END D_flipflop;

ARCHITECTURE Behavior OF D_flipflop IS
BEGIN
  PROCESS(Clock, Reset, Clear)
  BEGIN
    IF Reset = '0' THEN
      Q <= '1';
    ELSIF Clear = '0' THEN
      Q <= '0';
    ELSIF Clock'EVENT AND Clock = '1' THEN
      Q <= D;
    END IF;
  END PROCESS;
END Behavior;
```

Figure 28. VHDL code for a D flip-flop with asynchronous inputs.