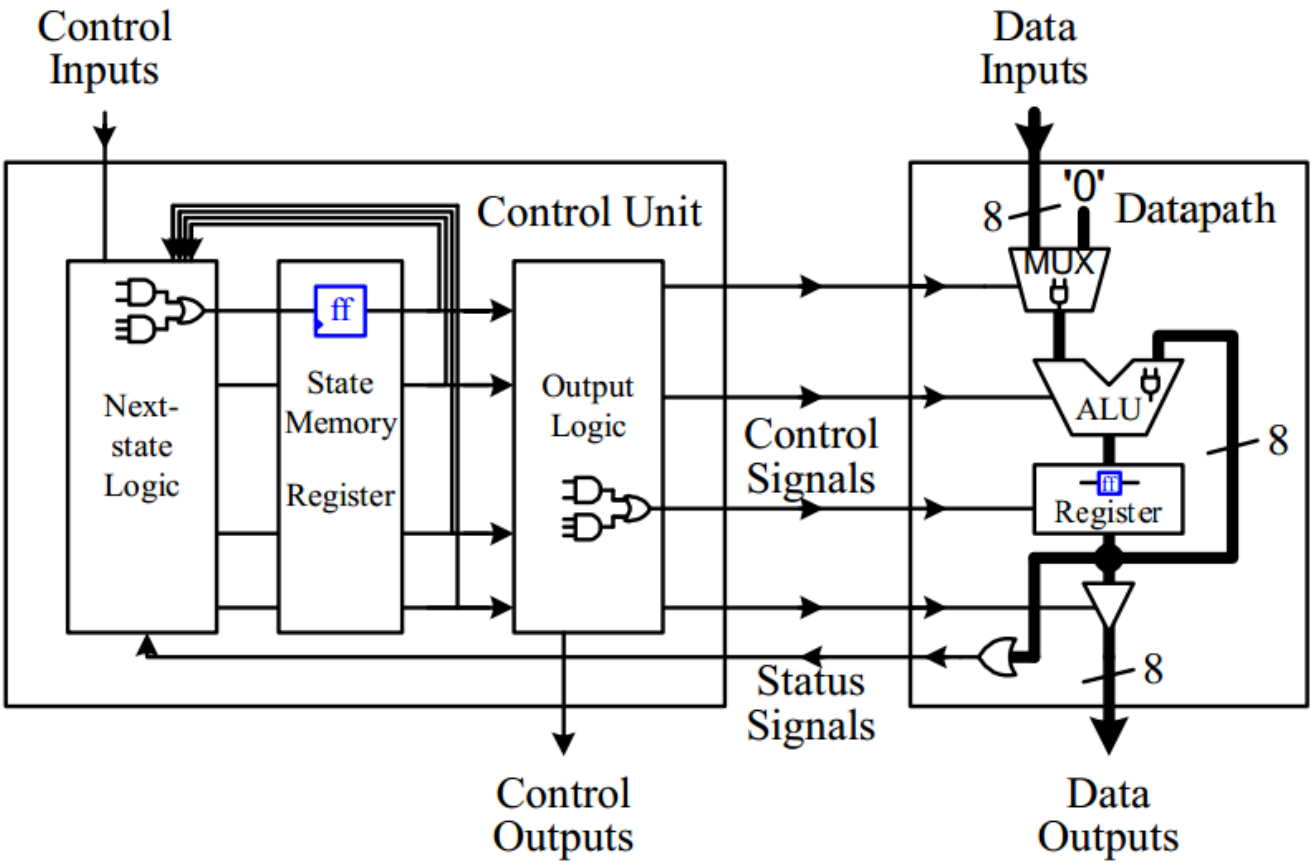
Flip - Flop


Control Inputs
Data Inputs
Control Unit
Next-state Logic
ff
State Memory Register
Output Logic
Control Signals
Status Signals
8
'0'
Datapath
MUX
ALU
Register
8
8
Control Outputs
Data Outputs

So far, we have been looking at the design of combinational circuits. We will now turn our attention to the design of sequential circuits. Recall that the outputs of sequential circuits are dependent on not only their current inputs (as in combinational circuits), but also on all their past inputs. Because of this necessity to remember the history of inputs, sequential circuits must contain memory elements. The car security system from Section 2.9 is an example of a combinational circuit. In that example, the siren is turned on when the master switch is on and someone opens the door. If you close the door afterwards, then the siren will turn off immediately. For a more realistic car security system, we would like the siren to remain on even if you close the door after it was first triggered. In order for this modified system to work correctly, the siren must be dependent on not only the master switch and the door switch but also on whether the siren is currently on or off. In other words, this modified system is a sequential circuit that is dependent on both the current and on the past inputs to the system. In order to remember this history of inputs, sequential circuits must have memory elements. Memory elements, however, are just like combinational circuits in the sense that they are made up of the same basic logic gates. What makes them different is in the way these logic gates are connected together. In order for a circuit to "remember" its current value, we have to connect the output of a logic gate directly or indirectly back to the input of that same gate.

We call this a feedback loop circuit, and it forms the basis for all memory elements. Combinational circuits do not have any feedback loops. Latches and flip-flops are the basic memory elements for storing information. Hence, they are the fundamental building blocks for all sequential circuits. A single latch or flip-flop can store only one bit of information. This bit of information that is stored in a latch or flip-flop is referred to as the state of the latch or flip-flop. Hence, a single latch or flip-flop can be in either one of two states: 0 or 1. We say that a latch or a flip-flop changes state when its content changes from a 0 to a 1 or vice versa. This state value is always available at the output. Consequently, the content of a latch or a flip-flop is the state value, and is always equal to its output value. The main difference between a latch and a flip-flop is that for a latch, its state or output is constantly affected by its input as long as its enable signal is asserted. In other words, when a latch is enabled, its state changes immediately when its input changes. When a latch is disabled, its state remains constant, thereby, remembering its previous value. On the other hand, a flip-flop changes state only at the active edge of its enable signal, i.e., at precisely the moment when either its enable signal rises from a 0 to a 1 (referred to as the rising edge of the signal), or from a 1 to a 0

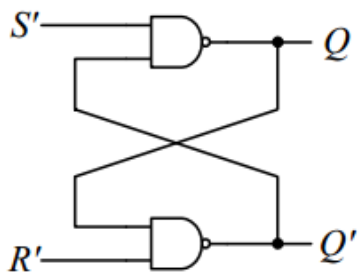
(the falling edge). However, after the rising or falling edge of the enable signal, and during the time when the enable signal is at a constant 1 or 0, the flip-flop's state remains constant even if the input changes. In a microprocessor system, we usually want changes to occur at precisely the same moment. Hence, flip-flops are used more often than latches, since they can all be synchronized to change only at the active edge of the enable signal. This enable signal for the flip-flops is usually the global controlling clock signal. Historically, there are basically four main types of flip-flops: SR, D, JK, and T. The major differences between them are the number of inputs they have and how their contents change. Any given sequential circuit can be built using any of these types of flip-flops (or combinations of them). However, selecting one type of flip-flop over another type to use in a particular sequential circuit can affect the overall size of the circuit. Today, sequential circuits are designed mainly with D flip-flops because of their ease of use. This is simply a tradeoff issue between ease of circuit design versus circuit size. Thus, we will focus mainly on the D flip-flop. Discussions about the other types of flip-flops can be found in Section 6.14. In this chapter, we will look at how latches and flip-flops are designed and how they work. Since flip-flops are at the heart of all sequential circuits, a good understanding of their design and operation is very important in the design of microprocessors.

1- SR Latch

In order to change the state for the bistable element, we need to add external inputs to the circuit. The simplest way to add extra inputs is to replace the two inverters with two NAND gates, as shown in Figure 1.(a). This circuit is called an SR latch. In addition to the two outputs Q and Q', there are two inputs S 'and R 'for set and reset, respectively. Just like the bistable element, the SR latch can be in one of two states: a set state when Q= 1, or a reset state when Q= 0. Following the convention, the primes in Sand R denote that these inputs are active-low (i.e., a 0 asserts them, and a 1 de-asserts them). To make the SR latch go to the set state, we simply assert the S 'input by setting it to 0 (and de-asserting R'). It doesn't matter what the other NAND gate input is, because 0 NAND anything gives a 1, hence Q= 1, and the latch is set. If S 'remains at 0 so that Q(which is connected to one input of the bottom NAND gate) remains at 1, and if we now de-assert R'(i.e., set R' to a 1) then the output of the bottom NAND gate will be 0, and so, Q'= 0. This situation is shown in Figure 6.2(d) at time t0. From this current situation, if we now de-assert S 'so that S'= R'= 1, the latch will remain in the set state because Q'(the second input to the top NAND gate) is 0, which will keep Q= 1, as shown at time t1. At time t2,

we reset the latch by making R'= 0 (and S'= 1). With R 'being a 0, Q 'will go to a 1. At the top NAND gate, 1 NAND1 is 0, thus forcing Q to go to 0. If we de-assert R 'next so that, again, we have S'= R'= 1, this time the latch will remain in the reset state, as shown at time t3.

Notice the two times (at t1and t3) when both S' and R 'are de-asserted (i.e., S'= R'= 1). At t1, Q is at a 1; whereas, at t3, Q is at a 0. Why is this so? What is different between these two times? The difference is in the value of Q immediately before those times. The value of Q right before t1is 1; whereas, the value of Q right before t3is 0. When both inputs are de-asserted, the SR latch remembers its previous state. Previous to t1, Q has the value 1, so at t1, Q remains at a 1. Similarly, previous to t3, Q has the value 0, so at t3, Q remains at a 0.

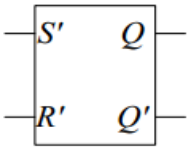


(a)

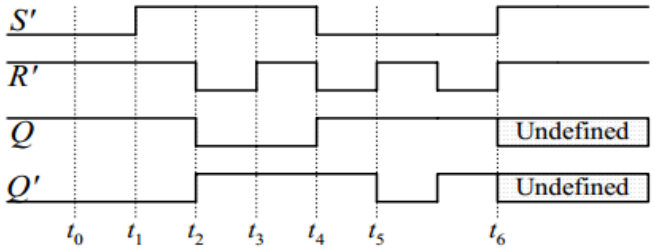
R	S	Q	Q'
0	0	Not allowed	
0	1	0	1
1	0	1	0
1	1	No change	

Reset
Set

(b)



(c)



(d)

Fig. 1 SR latch: (a) circuit using NAND gate; (b) Truth table; (c) logic symbol; (d) sample trace.

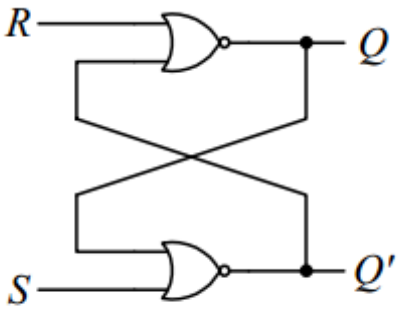
If both S' and R' are asserted (i.e., S'= R'= 0), then both Q and Q' are equal to a 1, as shown at time t4, since 0 NAND anything gives a 1. Note that there is nothing wrong with having Q equal to Q'. It is just because we named these two points Q and Q' that we don't like them to be equal. However, we could have used another name say, Pin stead of Q'. 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 is de-asserted later, as shown at time t5. At t5, R' is de-asserted first, so the latch goes into the 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 t6. Let us assume for a moment that both gates have exactly the same delay and that the two wire connections between the output of one gate to the input of the other gate also have exactly the same delay. Currently, both Q and Q' are at a 1. If we set S 'and R' to a 1 at exactly the same time, then both NAND gates will perform a 1 NAND1 and will both output a 0 at exactly the same time. The two 0's will be fed back to the two gate inputs at exactly the same time, because the two wire connections have the same delay. This time around, the two NAND gates will perform a 1 NAND0 and will both produce a 1 again at exactly the same time. This time, two 1's will be fed back to the inputs, which again will produce a 0 at the outputs, and so on and on. This oscillating behavior, called the critical race, will continue indefinitely until one outpaces the other. 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 end up in. Thus, the latch's next state is undefined.

Of course, in practice, it is next to impossible to manufacture two gates and make the two connections with precisely the same delay. In addition, both S'and R'need to be de-asserted at exactly the same time. Nevertheless, if this circuit is used in controlling some mission-critical device, we don't want even this slim chance to happen. In order to avoid this non-deterministic behavior, we must make sure that the two inputs are never de-asserted at the same time. Note that we do want the situation when both of them are de-asserted, as in times t1and t3, so that the circuit can remember its current content. We want to de-assert one input after de-asserting the other, but just not de-asserting both of them at exactly the same time. In practice, it is very difficult to guarantee that these two signals are never de-asserted at the same time, so we relax the condition slightly by not having both of them asserted together. In other words, if one is asserted, then the other one cannot be asserted. Therefore, if both of them are never asserted simultaneously, then they cannot be de-asserted at the same time. A minor side benefit for not having

both of them asserted together is that Q and Q' are never equal to each other. Recall that, from the names that we have given these two nodes, we do want them to be inverses of each other.

From the above analysis, we obtain the truth table in Figure 1.(b) for the NAND implementation of the SR latch. In the truth table, Q and Q next actually represent the same point in the circuit. The difference is that Q is the current value at that point, while Q next is the new value to be updated in the next time period. Another way of looking at it is that Q is the input to a gate, and Q next is the output from a gate. In other words, the signal Q goes into a gate, propagates through the two gates, and arrives back at Q as the new signal Q ext. Figure 1.(c) shows the logic symbol for the SR latch.

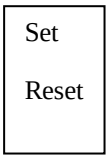
The SR latch can also be implemented using NOR gates, as shown in Figure 2.(a). The truth table for this implementation is shown in Figure 2.(b). From the truth table, we see that the main difference between this implementation and the NAND implementation is that for the NOR implementation, the Sand 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 2.(c).

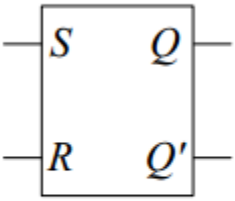


(a)

R	S	Q	Q'
0	0	No change	
0	1	1	0
1	0	0	1
1	1	Not allowed	

(b)





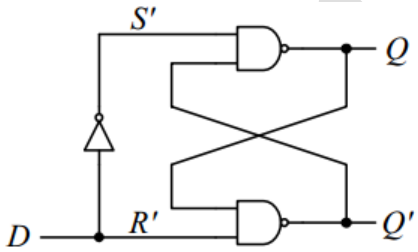
(c)

Figure 2.SR latch: (a) circuit using NOR gates; (b) truth table; (c) logic symbol.

D Latch

Recall from Section 1 that the disadvantage with the SR latch is that we need to ensure that the two inputs, S and R, are never de-asserted at exactly the same time, and we said that we can guarantee this by not having both of them asserted. This situation is prevented in the D latch by adding an inverter between the original S' and R' inputs. This way, S' and R' will always be inverses of each other, and so, they will never be asserted together. The circuit using NAND gates and the inverter is shown in Figure 3(a). There is now only one input D(for data). When D= 0, then S'= 1 and R'= 0, so this is similar to resetting the SR latch by making Q= 0. Similarly, when D= 1, then S'= 0 and R'= 1, and Q will be set to 1. From this observation, we see that Q next always gets the same value as the input D, and is independent of the current value of Q. Hence, we obtain the truth table for the D latch, as shown in Figure 3(b).

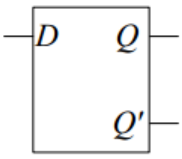
Comparing the truth table for the D latch shown in Figure 3(b) with the truth table for the SR latch shown in Figure 1(b), it is obvious that we have eliminated not just one, but three rows, where S'= R'. The reason for adding the inverter to the SR latch circuit was to eliminate the row where S'= R'= 0. However, we still need to have the other two rows where S'= R'= 1 in order for the circuit to remember its current value. By not being able to set both S' and R' to 1, this D latch circuit has now lost its ability to remember. Q next cannot remember the current value of Q, instead it will always follow D. The end result is like having a piece of wire where the output is the same as the input!



(a)

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

(b)



(c)

Figure 3.D latch: (a) circuit using NAND gates; (b) truth table; (c) logic symbol.

Clock

Latches are known as level-sensitive because their outputs are affected by their inputs as long as they are enabled. Their memory state can change during this entire time when the enable signal is asserted. In a computer circuit, however, we do not want the memory state to change at various times when the enable signal is asserted. Instead, we like to synchronize all of the state changes to happen at precisely the same moment and at regular intervals. In order to achieve this, two things are needed:

1) a synchronizing signal, and 2) a memory circuit that is not level-sensitive. The synchronizing signal, of course, is the clock, and the non-level-sensitive memory circuit is The Flip-Flop.

The clock is simply a very regular square wave signal, as shown in Figure 3. We call the edge of the clock signal when it changes from 0 to 1 the rising edge. Conversely, the falling edge of the clock is the edge when the signal changes from 1 to 0. We will use the symbol to denote the rising edge and for the falling edge. In a computer circuit, either the rising edge or the falling edge of the clock can be used as the synchronizing signal for writing data into a memory element. This edge signal is referred to as the active edge of the clock. In all of our examples, we will use the rising clock edge as the active edge. Therefore, at every rising edge, data will be clocked or stored into the memory element.

A clock cycle is the time from one rising edge to the next rising edge or from one falling edge to the next falling edge. The speed of the clock, measured in hertz (Hz), is the number of cycles per second. Typically, the clock speed for a microprocessor in an embedded system runs around 20 MHz, while the microprocessor in a personal computer runs upwards of 2 GHz and higher. A clock period is the time for one clock cycle (seconds per cycle), so it is just the inverse of the clock speed. The speed of the clock is determined by how fast a circuit can produce valid results. For example, a two-level combinational circuit will have valid results at its output much sooner than, say, an ALU can. Of course, we want the clock speed to be as fast as possible, but it can only be as fast as the slowest circuit in the entire system. We want the clock period to be the time it takes for the slowest circuit to get its input from a memory element, operate on the data, and then write the data back into a memory element. More will be said on this in later sections.

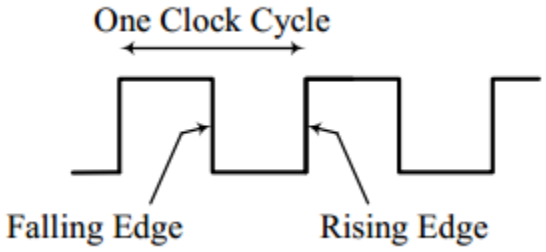
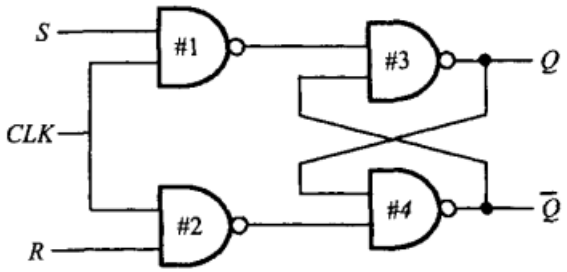


Figure 4. Clock signal.

RS Flip-Flop

An RS flip-flop is a clocked SR latch. This means that the RS flip-flop is same as the SR latch with a clock input. The SR flip-flop is an important circuit because all other flip-flops are built from it. Figure 5. shows an RS flip-flop.

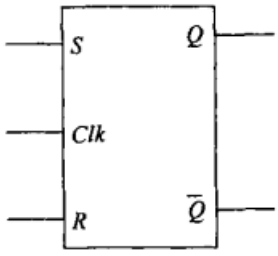
The RS flip-flop contains an SR latch with two more NAND gates. It has three inputs (S, CLK, R) and two outputs (Q and D). When S = 0 and R = 0 and CLK = 1, the outputs of both NAND gates #1 and #2 are 1. This means that the output of NAND gate #3 is 0 if = 1 and is 1 if D = 0. This means that Q is unchanged as long as S = 0 and R = 0. On the other hand, the output of NAND gate #4 is 0 if Q = 1 and is 1 if Q = 0. Thus, Q is also unchanged. Suppose that S = 1, R = 0, and CLK = 1. This will produce 0 and 1



(a) NAND gate implementation

S	R	Clk	Q^+	$\overline{Q^+}$
0	0	1	Q	$\overline{Q}$
0	1	1	0	1
1	0	1	1	0
1	1	1	?	?
X	X	0	Q	$\overline{Q}$

(b) Truth Table

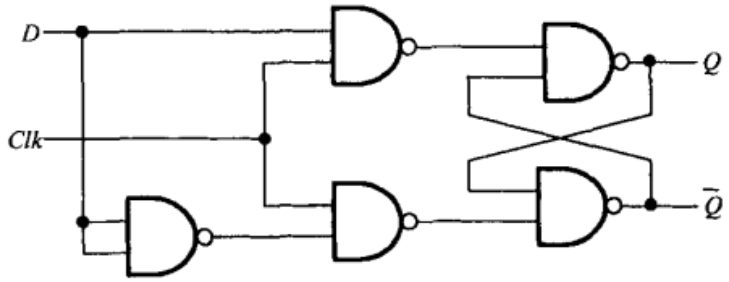


(c) Logic Symbol

Figure 5. RS Flip-Flop with Clock.

D Flip-Flop

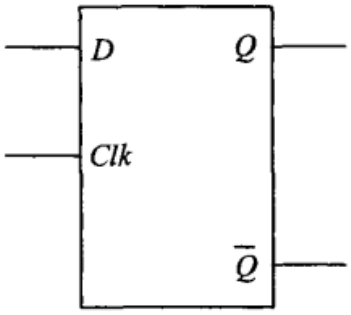
Figure 6. shows the logic diagram, truth table and the logic symbol of a D flip-flop (Delay flip-flop). This type of flip-flop ensures that the invalid input combinations S = 1 and R = 1 for the RS flip-flop can never occur. The D flip-flop has two inputs (D and CLK) and two outputs (Q and Q). The D input is same as the S input and the complement of D is applied to the R input. Thus, R and S can never be equal to 1 simultaneously. The D flip-flop (called gated D flip-flop) transfers the D input to output Q when CLK = 1. Note that if CLK = 0, one of the inputs to each of the last two NAND gates will be 1 ; thus, outputs of the D flip-flop remain unchanged regardless of the values of the D input. The D flip-flop is also called a "transparent latch." The term "transparent" is on the fact that the output Q follows the D input when CLK = 1. Therefore, transfer of input to outputs is transparent, as if the flip-flop were not present.



(a) NAND gate implementation

D	Clk	Q^+	$\overline{Q^+}$
0	1	0	1
1	1	1	0
X	0	Q	$\bar{Q}$

(b) Truth Table



(c) Logic Symbol

Figure 6. D Flip-Flop with Clock.

JK Flip-Flop

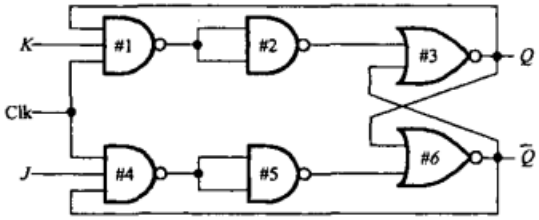
The JK flip-flop is a modified version of the RS flip-flop such that the Sand R inputs of the RS flip-flop correspond to the J and K inputs of the JK flip-flop. Furthermore, the invalid inputs S = 1 and R = 1 are allowed in the JK flip-flop. When J = 1, K = 1, and Clk = 1, the JK flip-flop complements its output. Otherwise, the meaning of the J and K inputs is the same as that of the S and R inputs respectively. Figure 5.6 shows a logic diagram of JK flip- flop along with its truth table. This is a NANDNOR implementation, and is called a gated JK flip-flop. The circuit operation of Figure 7.(a) is discussed in the following:

i) Suppose Q = 1, = 0, and CLK = 1. With J = 0 and K = 0, the outputs of inverters 140 Fundamentals of Digital Logic and Microcomputer Design #2 and #5 are both 0. This means that the outputs of NOR gates #3 and #6 are 1 and 0 respectively. Therefore, the outputs of the flip-flop are unchanged

ii) Suppose Q = 0, e = 1, and CLK = 1. With J= 1 and K = 0, the outputs of inverters #2 and #5 are 0 and 1 respectively. This means that a 0 is produced at the output of NOR gate #6 (e = 0). Thus, apply a 0 at one of the inputs of NOR gate #3 generating a 1 at its output (Q = 1). The JK flip-flop is therefore set to 1 (Q = 1 and = 0 and CLK = 1. With J= 0 and K = 1, the outputs of the inverter #2 and #5 are 1 and 0 respectively. This means that the output of NOR gate #3 is 0. This will produce a 1 at the output of NOR gate #6. Thus, the flip-flop is cleared to zero (Q =O ande= 1).

iv) Suppose Q = 1, = 0, and CLK = 1. With J = 1 and K = 1, the outputs of inverters #2 and #5 are 1 and 0 respectively. This will produce a 0 at the output of NOR gate #3 (Q = 0). This in turn will apply 0 at one of the inputs of NOR gate #6, making its output HIGH (e = 1). Thus, the output of the JK flip-flop is complemented. The other rows in the truth table of the JK flip-flop can similarly be verified. JK flip-flops are never built using the schematic of figure 5.6(a). This is because the schematic of Figure 5.6(a) will generate oscillations. For example, when J=1, K=l, and Clk =1, the outputs (Q and g) are complemented with the clock staying high after the first transition ofthe outputs. Since the outputs are fed back, the outputs will change continuously after being complemented once, causing oscillations. This undesirable behavior can be avoided using master-slave (edge-triggered) flip-flops discussed in the next section. = 0).

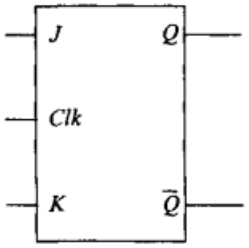
iii) Suppose Q = 1,



(a) NAND gate implementation

J	K	Clk	Q+	$\bar{Q}$+
0	0	1	Q	$\bar{Q}$
1	0	1	1	0
0	1	1	0	1
1	1	1	$\bar{Q}$	Q
X	X	0	Q	$\bar{Q}$

(b) Truth Table

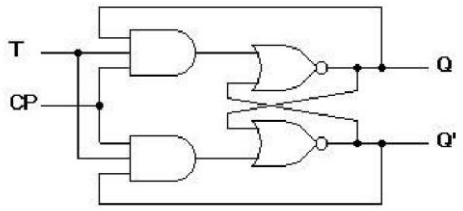


(c) Logic Symbol

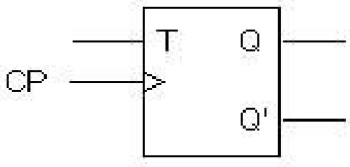
Figure 7. JK Flip-Flop with Clock.

T Flip-Flop

The T (Toggle) flip-flop complements its output when the clock input is applied with T = 1 ; the output remains unchanged when T = 0. The name "toggle" is based on the fact that the T flip-flop toggles or complements its output when the clock input is 1 with T = 1. T flip-flop is not available commercially. However, T flip-flop can be obtained from JK flip- flop in two ways. In the first approach, the J and K inputs of the JK flip-flop can be tied together to provide the T input; the output is complemented when T = 1 at the clock while the output remains unchanged when T = 0 at the clock. In the second approach, the J and K inputs can be tied to high; in this case, T is the clock input.



(a) Logic diagram



(b) Graphical symbol

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

(c) Transition table

Figure 8. T Flip-Flop with Clock.

Master-Slave Flip-Flop

As mentioned before, sequential circuits contain combinational circuits with flip-flops in the feedback loop. These flip-flops generate outputs at the clock based on the inputs from

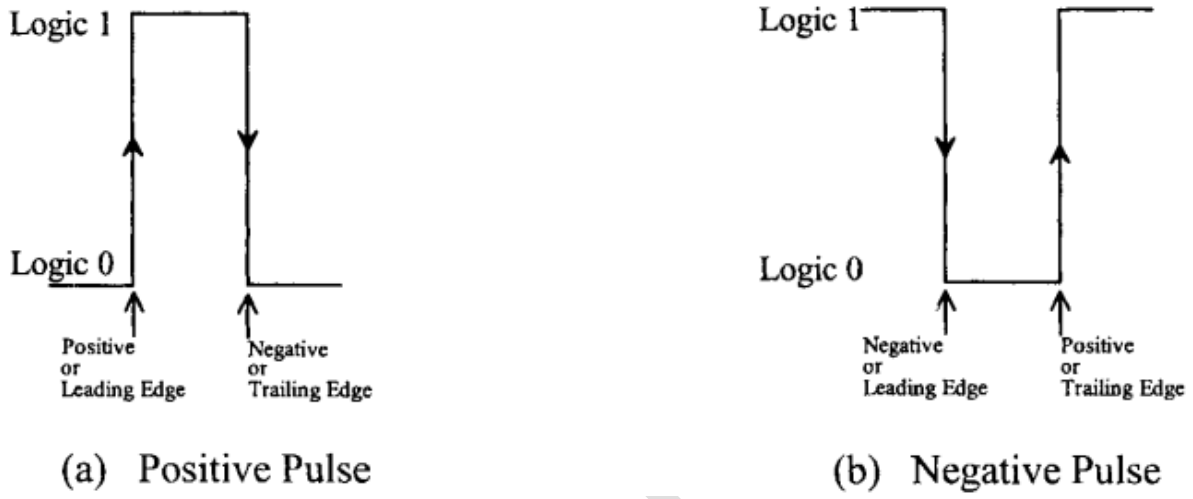


Figure 9. Clock Pulses

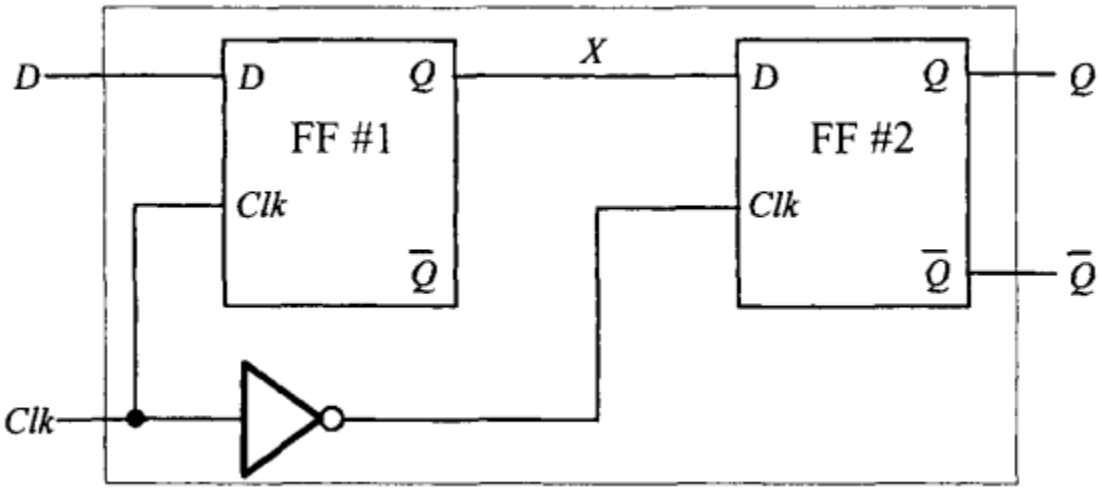


Figure 10. Typical Master-Slave D Flip-Flop

The combinational circuits. The feedback loop can create an undesirable situation if the outputs from the combinational circuits that are connected to the flip-flop inputs change values at the clock pulse simultaneously when flip-flops change outputs. This situation can be avoided if the flip-flop outputs do not change until the clock pulse goes back to 0. One way of accomplishing this is to ensure that the outputs of the flip-flops are affected by the pulse transition rather than pulse duration of the clock input. To understand this concept, consider the clock pulses shown in Figure 9. There are two types of clock pulses: positive and negative. A positive pulse includes two transitions: logic 0 to logic 1 and logic 1 to logic 0. A negative pulse also goes through two transitions: logic 1 to logic 0 and logic 0 to logic 1. Assume that a positive pulse is used as the clock input of a D flip-flop. With the D input = 1, the output of the flip-flop will become 1 when the clock pulse reaches logic 1. Now, suppose that the D input changes to zero but the clock pulse is still 1. This means that the flip-flop will have a new output, 0. In this situation, the output of one flip-flop cannot be connected to the input of another when both flip-flops are enabled simultaneously by the same clock input. This problem can be avoided if the flip-flop is clocked by either the leading or the trailing edge rather than the signal level of the pulse. A master-slave flip-flop is used to accomplish this. Figure 10. shows a typical master-slave D flip-flop. A master-slave flip-flop contains two independent flip-flops. Flip-flop #1 (FF #1) works as a master flip-flop, whereas the flip-flop (FF #2) is a slave. An inverter is used to invert the clock input to the slave flip-flop. Assume that the CLK is a positive pulse. Suppose that the D input of the master flip-flop (FF #I) is 1 and the CLK input = 1 (leading edge). The output of the inverter will apply a 0 at the CLK input of the slave flip-flop (FF #2). Thus, FF #2 is disabled. The master flip-flop will transfer a 1 to its Q output. Thus, X will be 1. At the trailing edge of the CLK input, the CLK input of the master flip-flop is 0. Thus, FF #1 is disabled. The inverter will apply a 1 at the CLK input of the FF #2. Thus, 1 at the X input (D input of FF #2) will be transferred to the Q output of FF #2. When the CLK goes back to 0, the master flip-flop is separated. This avoids any change in the other inputs to affect the master flip-flop. The slave flip-flop will have the same output as the master.

Sources:

1. Digital Logic and Microprocessor Design With VHDL.
2. Fundamentals of Digital Logic and Microcomputer Design, 5th Ed.