

UNIT IV

Sequential circuit

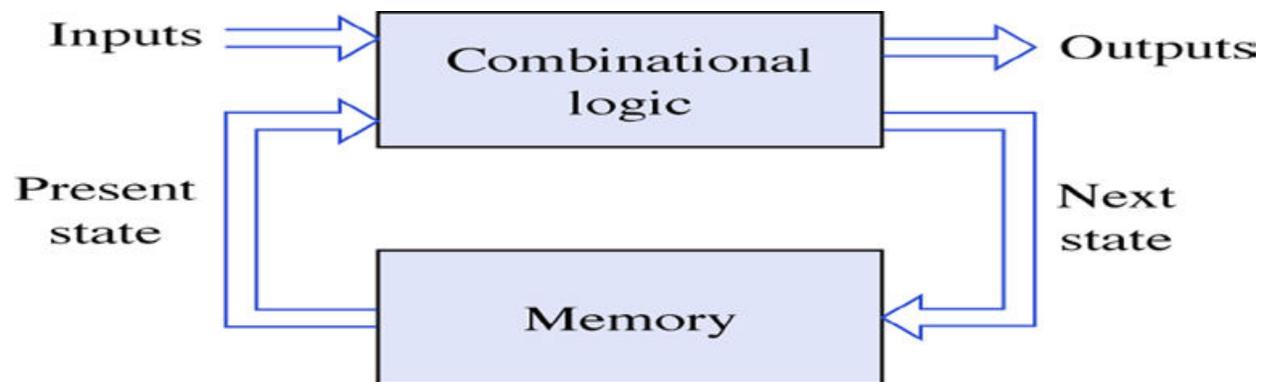
Introduction

In the previous session, we said that the output of a combinational circuit depends solely upon the input. The implication is that combinational circuits have no memory. In order to build sophisticated digital logic circuits, including computers, we need more a powerful model. We need circuits whose output depends upon both the input of the circuit and its previous state. In other words, we need circuits that have *memory*. For a device to serve as a memory, it must have three characteristics:

- the device must have two stable states
- there must be a way to read the state of the device
- there must be a way to set the state at least once.

It is possible to produce circuits with memory using the digital logic gates we've already seen. To do that, we need to introduce the concept of *feedback*. So far, the logical flow in the circuits we've studied has been from input to output. Such a circuit is called *acyclic*. Now we will introduce a circuit in which the output is fed back to the input, giving the circuit memory. (There are other memory technologies that store electric charges or magnetic fields; these do not depend on feedback.)

Latches and flip-flops

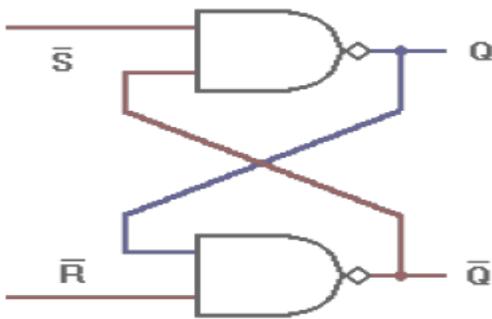


Latches and flip-flops

In the same way that gates are the building blocks of combinatorial circuits, *latches* and *flip-flops* are the building blocks of sequential circuits. While gates had to be built directly from transistors, latches can be built from gates, and flip-flops can be built from latches. This fact will make it somewhat easier to understand latches and flip-flops. Both latches and flip-flops are circuit elements whose output depends not only on the current inputs, but also on previous inputs and outputs. The difference between a latch and a flip-flop is that a latch does not have a *clock signal*, whereas a flip-flop always does.

Latches

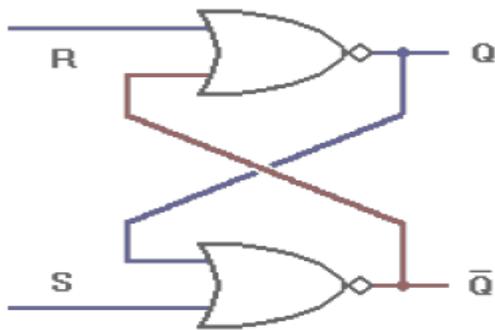
How can we make a circuit out of gates that is not combinatorial? The answer is *feed-back*, which means that we create *loops* in the circuit diagrams so that output values depend, indirectly, on themselves. If such feed-back is *positive* then the circuit tends to have stable states, and if it is *negative* the circuit will tend to oscillate. In order for a logical circuit to "remember" and retain its logical state even after the controlling input signal(s) have been removed, it is necessary for the circuit to include some form of feedback. We might start with a pair of inverters, each having its input connected to the other's output. The two outputs will always have opposite logic. The problem with this is that we don't have any additional inputs that we can use to change the logic states if we want. We can solve this problem by replacing the inverters with NAND or NOR gates, and using the extra input lines to control the circuit. The circuit shown below is a basic NAND latch. The inputs are generally designated "S" and "R" for "Set" and "Reset" respectively. Because the NAND inputs must normally be logic 1 to avoid affecting the latching action, the inputs are considered to be inverted in this circuit. The outputs of any single-bit latch or memory are traditionally designated Q and Q'. In a commercial latch circuit, either or both of these may be available for use by other circuits. In any case, the circuit itself is:



For the NAND latch circuit, both inputs should normally be at a logic 1 level. Changing an input to a logic 0 level will force that output to a logic 1. The same logic 1 will also be applied to the second input of the other NAND gate, allowing that output to fall to a logic 0 level. This in turn feeds back to the second input of the original gate, forcing its output to remain at logic 1. Applying another logic 0 input to the same gate will have no further effect on this circuit. However, applying a logic 0 to the *other* gate will cause the same reaction in the other direction, thus changing the state of the latch circuit the other way. Note that it is forbidden to have both inputs at a logic 0 level at the same time. That state will force both outputs to a logic 1, overriding the feedback latching action. In this condition, whichever input goes to logic 1 first will lose control, while the other input (still at logic 0) controls the resulting state of the latch. If both inputs go to logic 1 simultaneously, the result is a "race" condition, and the final state of the latch cannot be determined ahead of time. The same functions can also be performed using NOR gates. A few adjustments must be made to allow for the difference in the logic function, but the logic involved is quite similar. The circuit shown below is a basic NOR latch. The inputs are generally designated "S" and "R" for "Set" and "Reset" respectively. Because the NOR inputs must normally be logic 0 to avoid overriding the latching action, the inputs are not inverted in this circuit. The NOR-based latch circuit is: For the NOR latch circuit, both inputs should normally be at a logic 0 level. Changing an input to a logic 1 level will force that output to a logic 0. The same logic 0 will also be applied to the second input of the other NOR gate, allowing that output to rise to a logic 1 level. This in turn feeds back to the second input of the original gate, forcing its output to remain at logic 0 even after the external

input is removed. Applying another logic 1 input to the same gate will have no further effect on this circuit. However, applying a logic 1 to the *other* gate will cause the same reaction in the other direction, thus changing the state of the latch circuit the other way.

Note that it is forbidden to have both inputs at a logic 1 level at the same time. That state will force both outputs to a logic 0, overriding the feedback latching action. In this condition, whichever input goes to logic 0 first will lose control, while the other input (still at logic 1) controls the resulting state of the latch. If both inputs go to logic 0 simultaneously, the result is a "race" condition, and the final state of the latch cannot be determined ahead of time.



One problem with the basic RS NOR latch is that the input signals actively drive their respective outputs to a logic 0, rather than to a logic 1. Thus, the S input signal is applied to the gate that produces the Q' output, while the R input signal is applied to the gate that produces the Q output. The circuit works fine, but this reversal of inputs can be confusing when you first try to deal with NOR-based circuits.

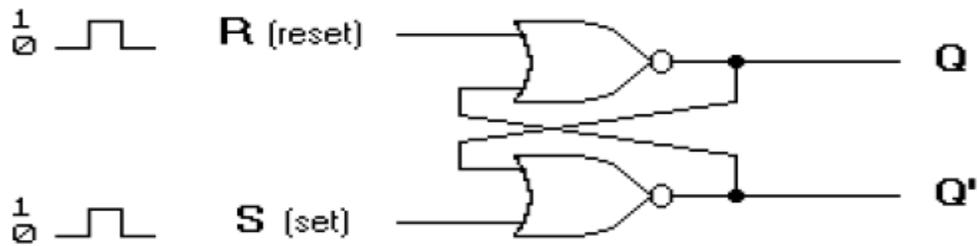
Flip-flops

Latches are asynchronous, which means that the output changes very soon after the input changes. Most computers today, on the other hand, are synchronous, which means that the outputs of all the sequential circuits change simultaneously to the rhythm of a global clock signal.

A flip-flop is a synchronous version of the latch.

A flip-flop circuit can be constructed from two NAND gates or two NOR gates. These flip-flops are shown in Figure 2 and Figure 3. Each flip-flop has two outputs, Q and Q', and two inputs, set and reset. This type of flip-flop is referred to as an SR flip-flop or SR latch. The flip-flop in Figure 2 has two useful states. When Q=1 and Q'=0, it is in the set state (or 1-state). When Q=0 and Q'=1, it is in the clear state (or 0-state). The outputs Q and Q' are complements of each other and are referred to as the normal and complement outputs, respectively. The binary state of the flip-flop is taken to be the value of the normal output.

When a 1 is applied to both the set and reset inputs of the flip-flop in Figure 2, both Q and Q' outputs go to 0. This condition violates the fact that both outputs are complements of each other. In normal operation this condition must be avoided by making sure that 1's are not applied to both inputs simultaneously.

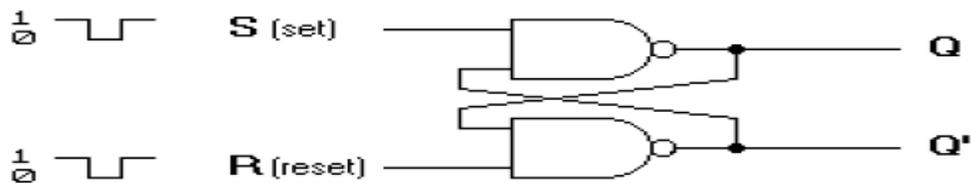


(a) Logic diagram

S	R	Q	Q'	
1	0	1	0	
0	0	1	0	(after S=1, R=0)
0	1	0	1	
0	0	0	1	(after S=0, R=1)
1	1	0	0	

(b) Truth table

Figure 2. Basic flip-flop circuit with NOR gates

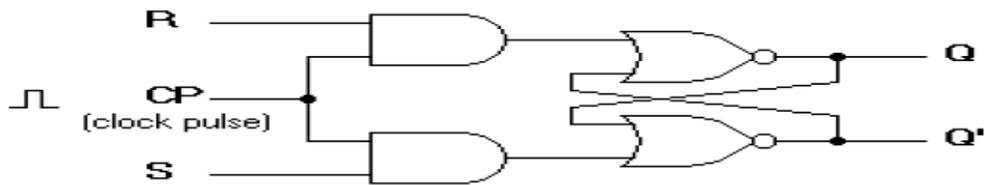


(a) Logic diagram

S	R	Q	Q'	
1	0	0	1	
1	1	0	1	(after S=1, R=0)
0	1	1	0	
1	1	1	0	(after S=0, R=1)
0	0	1	1	

Clocked SR Flip-Flop

The clocked SR flip-flop shown in Figure 4 consists of a basic NOR flip-flop and two AND gates. The outputs of the two AND gates remain at 0 as long as the clock pulse (or CP) is 0, regardless of the S and R input values. When the clock pulse goes to 1, information from the S and R inputs passes through to the basic flip-flop. With both S=1 and R=1, the occurrence of a clock pulse causes both outputs to momentarily go to 0. When the pulse is removed, the state of the flip-flop is indeterminate, i.e., either state may result, depending on whether the set or reset input of the flip-flop remains a 1 longer than the transition to 0 at the end of the pulse.

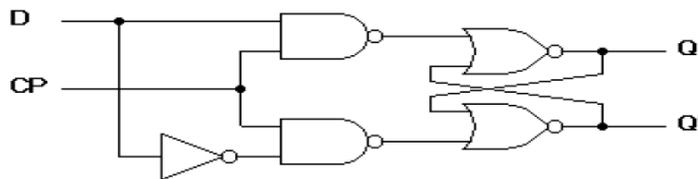


(a) Logic diagram

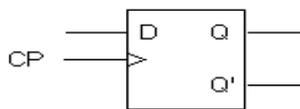
Q	S	R	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	indeterminate
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	indeterminate

D Flip-Flop

The D flip-flop shown in Figure 5 is a modification of the clocked SR flip-flop. The D input goes directly into the S input and the complement of the D input goes to the R input. The D input is sampled during the occurrence of a clock pulse. If it is 1, the flip-flop is switched to the set state (unless it was already set). If it is 0, the flip-flop switches to the clear state.



(a) Logic diagram with NAND gates



(b) Graphical symbol

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

(c) Transition table

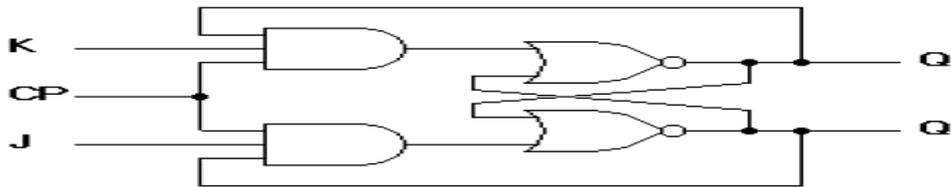
Figure 5. Clocked D flip-flop

JK Flip-Flop

A JK flip-flop is a refinement of the SR flip-flop in that the indeterminate state of the SR type is defined in the JK type. Inputs J and K behave like inputs S and R to set and clear the flip-flop (note that in a JK flip-flop, the letter J is for set and the letter K is for clear). When logic 1 inputs are applied to both J and K simultaneously, the flip-flop switches to its complement state, i.e., if $Q=1$, it switches to $Q=0$ and vice versa.

A clocked JK flip-flop is shown in Figure 6. Output Q is ANDed with K and CP inputs so that the flip-flop is cleared during a clock pulse only if Q was previously 1. Similarly, output Q' is ANDed with J and CP inputs so that the flip-flop is set with a clock pulse only if Q' was previously 1. Note that because of the feedback connection in the JK flip-flop, a CP signal which remains a 1 (while $J=K=1$) after the outputs

have been complemented once will cause repeated and continuous transitions of the outputs. To avoid this, the clock pulses must have a time duration less than the propagation delay through the flip-flop. The restriction on the pulse width can be eliminated with a master-slave or edge-triggered construction. The same reasoning also applies to the T flip-flop presented next.



(a) Logic diagram

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

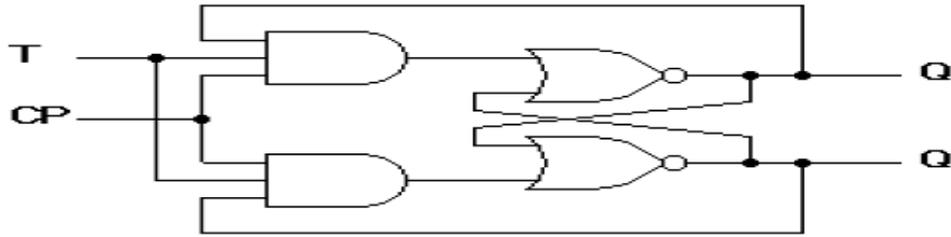
(c) Transition table

T Flip-Flop

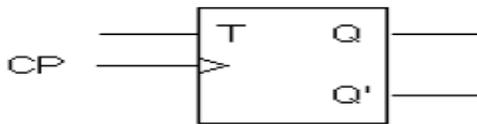
The T flip-flop is a single input version of the JK flip-flop. As shown in Figure 7, the T flip-flop is obtained from the JK type if both inputs are tied together. The output of the T flip-flop "toggles" with each clock pulse.

(a) Logic diagram

(b) Graphical symbol



(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

Triggering of Flip-flops

The state of a flip-flop is changed by a momentary change in the input signal. This change is called a trigger and the transition it causes is said to trigger the flip-flop. The basic circuits of Figure 2 and Figure 3 require an input trigger defined by a change in signal level. This level must be returned to its initial level before a second trigger is applied. Clocked flip-flops are triggered by pulses.

The feedback path between the combinational circuit and memory elements in Figure 1 can produce instability if the outputs of the memory elements (flip-flops) are changing while the outputs of the combinational circuit that go to the flip-flop inputs are being sampled by the clock pulse. A way to solve the feedback timing problem is to make the flip-flop sensitive to the pulse transition rather than the pulse duration. The clock pulse goes through two signal transitions: from 0 to 1 and the return from 1 to 0. As shown in Figure 8 the positive transition is defined as the positive edge and the negative transition as the negative edge.

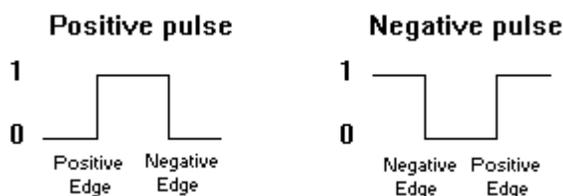
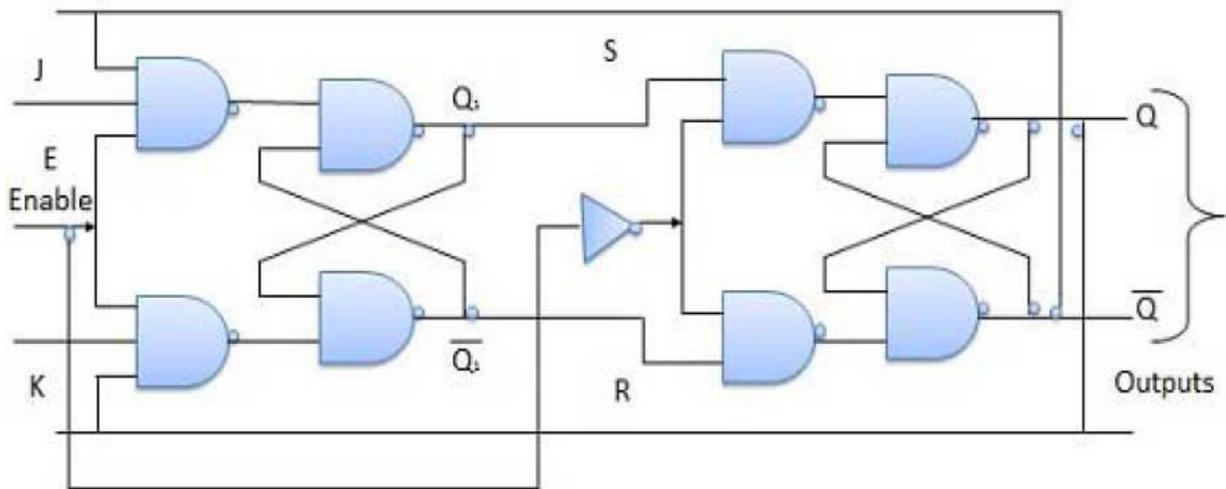
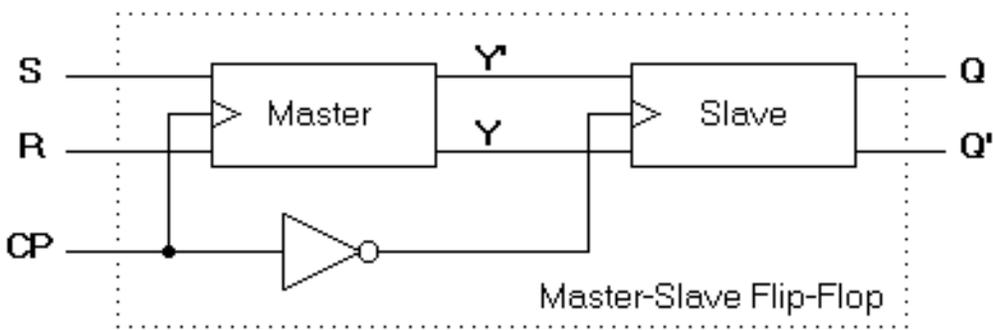


Figure 8. Definition of clock pulse transition The clocked flip-flops already introduced are triggered during the positive edge of the pulse, and the state transition starts as soon as the pulse reaches the logic-1 level. If the other inputs change while the clock is still 1, a new output state may occur. If the flip-

flop is made to respond to the positive (or negative) edge transition only, instead of the entire pulse duration, then the multiple-transition problem can be eliminated.

Master-Slave Flip-Flop

A master-slave flip-flop is constructed from two separate flip-flops. One circuit serves as a master and the other as a slave. The logic diagram of an SR flip-flop is shown in Figure 9. The master flip-flop is enabled on the positive edge of the clock pulse CP and the slave flip-flop is disabled by the inverter. The information at the external R and S inputs is transmitted to the master flip-flop. When the pulse returns to 0, the master flip-flop is disabled and the slave flip-flop is enabled. The slave flip-flop then goes to the same state as the master flip-flop.



Master slave RS flip flop

The timing relationship is shown in Figure 10 and is assumed that the flip-flop is in the clear state prior to the occurrence of the clock pulse. The output state of the master-slave flip-flop occurs on the negative transition of the clock pulse. Some master-slave flip-flops change output state on the positive transition of the clock pulse by having an additional inverter between the CP terminal and the input of the master.

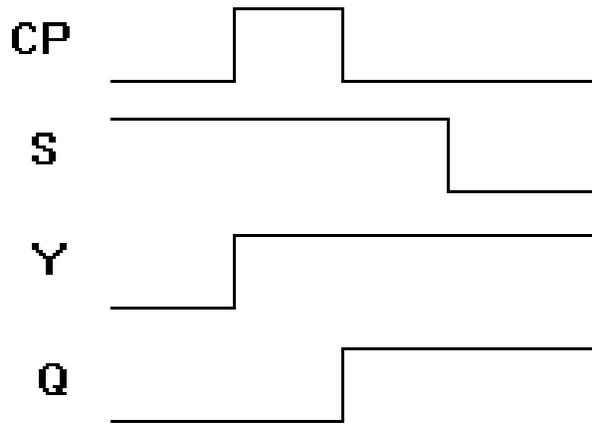


Figure 10. Timing relationship in a master slave flip-flop

Edge Triggered Flip-Flop

Another type of flip-flop that synchronizes the state changes during a clock pulse transition is the edge-triggered flip-flop. When the clock pulse input exceeds a specific threshold level, the inputs are locked out and the flip-flop is not affected by further changes in the inputs until the clock pulse returns to 0 and another pulse occurs. Some edge-triggered flip-flops cause a transition on the positive edge of the clock pulse (positive-edge-triggered), and others on the negative edge of the pulse (negative-edge-triggered). The logic diagram of a D-type positive-edge-triggered flip-flop is shown in Figure 11.

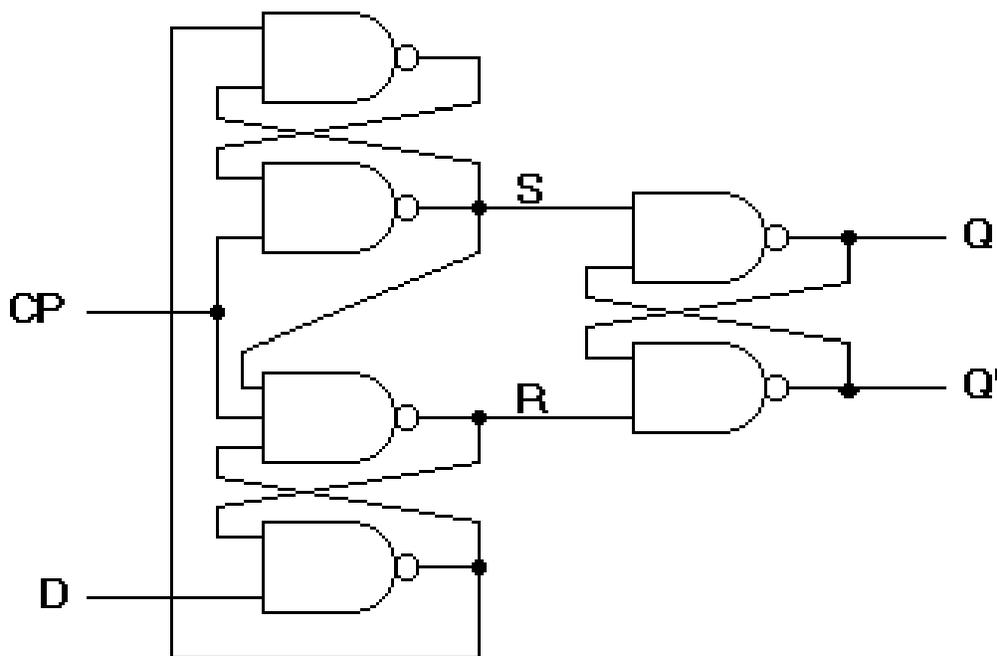
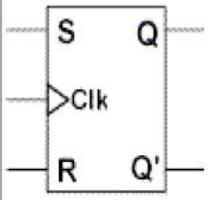
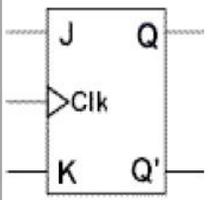
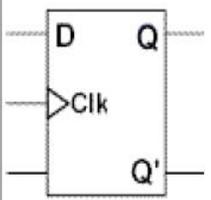
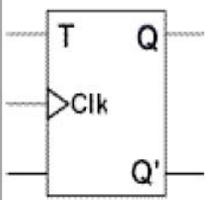


Figure 11. D-type positive-edge triggered flip-flop When using different types of flip-flops in the same circuit, one must ensure that all flip-flop outputs make their transitions at the same time, ie., during either the negative edge or the positive edge of the clock pulse.

Table 1. Flip-flop Types

TYPE	FLIP-FLOP SYMBOL	CHARACTERISTIC TABLE	CHARACTERISTIC EQUATION	EXCITATION TABLE																																			
SR		<table border="1"> <thead> <tr> <th>S</th> <th>R</th> <th>Q(next)</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>Q</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>?</td> </tr> </tbody> </table>	S	R	Q(next)	0	0	Q	0	1	0	1	0	1	1	1	?	$Q(\text{next}) = S + R'Q$ $SR = 0$	<table border="1"> <thead> <tr> <th>Q</th> <th>Q(next)</th> <th>S</th> <th>R</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>X</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>X</td> <td>0</td> </tr> </tbody> </table>	Q	Q(next)	S	R	0	0	0	X	0	1	1	0	1	0	0	1	1	1	X	0
S	R	Q(next)																																					
0	0	Q																																					
0	1	0																																					
1	0	1																																					
1	1	?																																					
Q	Q(next)	S	R																																				
0	0	0	X																																				
0	1	1	0																																				
1	0	0	1																																				
1	1	X	0																																				
JK		<table border="1"> <thead> <tr> <th>J</th> <th>K</th> <th>Q(next)</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>Q</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>Q'</td> </tr> </tbody> </table>	J	K	Q(next)	0	0	Q	0	1	0	1	0	1	1	1	Q'	$Q(\text{next}) = JQ' + K'Q$	<table border="1"> <thead> <tr> <th>Q</th> <th>Q(next)</th> <th>J</th> <th>K</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>X</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>X</td> </tr> <tr> <td>1</td> <td>0</td> <td>X</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>X</td> <td>0</td> </tr> </tbody> </table>	Q	Q(next)	J	K	0	0	0	X	0	1	1	X	1	0	X	1	1	1	X	0
J	K	Q(next)																																					
0	0	Q																																					
0	1	0																																					
1	0	1																																					
1	1	Q'																																					
Q	Q(next)	J	K																																				
0	0	0	X																																				
0	1	1	X																																				
1	0	X	1																																				
1	1	X	0																																				
D		<table border="1"> <thead> <tr> <th>D</th> <th>Q(next)</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> </tr> </tbody> </table>	D	Q(next)	0	0	1	1	$Q(\text{next}) = D$	<table border="1"> <thead> <tr> <th>Q</th> <th>Q(next)</th> <th>D</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	Q	Q(next)	D	0	0	0	0	1	1	1	0	0	1	1	1														
D	Q(next)																																						
0	0																																						
1	1																																						
Q	Q(next)	D																																					
0	0	0																																					
0	1	1																																					
1	0	0																																					
1	1	1																																					
T		<table border="1"> <thead> <tr> <th>T</th> <th>Q(next)</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Q</td> </tr> <tr> <td>1</td> <td>Q'</td> </tr> </tbody> </table>	T	Q(next)	0	Q	1	Q'	$Q(\text{next}) = TQ' + T'Q$	<table border="1"> <thead> <tr> <th>Q</th> <th>Q(next)</th> <th>T</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	Q	Q(next)	T	0	0	0	0	1	1	1	0	1	1	1	0														
T	Q(next)																																						
0	Q																																						
1	Q'																																						
Q	Q(next)	T																																					
0	0	0																																					
0	1	1																																					
1	0	1																																					
1	1	0																																					

Each of these flip-flops can be uniquely described by its graphical symbol, its characteristic table, its characteristic equation or excitation table. All flip-flops have output signals Q and Q'. The characteristic table in the third column of Table 1 defines the state of each flip-flop as a function of its inputs and previous state. **Q** refers to the present state and **Q(next)** refers to the next state after the occurrence of the clock pulse. The characteristic table for the RS flip-flop shows that the next state is equal to the present state when both inputs S and R are equal to 0. When R=1, the next clock pulse clears the flip-flop. When S=1, the flip-flop output Q is set to 1. The equation mark (?) for the next state when S and R are both equal to 1 designates an indeterminate next state. The characteristic table for the JK flip-flop is the same as that of the RS when J and K are replaced by S and R respectively, except for the indeterminate case. When both J and K are equal to 1, the next state is equal to the complement of the

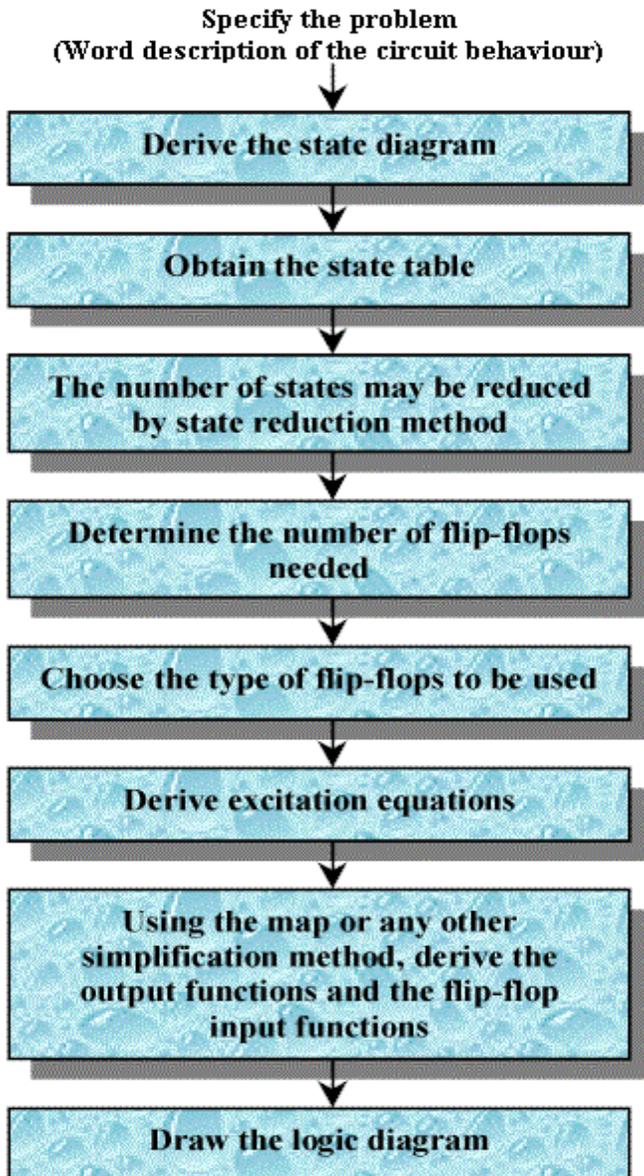
present state, that is, $Q(\text{next}) = Q'$. The next state of the D flip-flop is completely dependent on the input D and independent of the present state. The next state for the T flip-flop is the same as the present state Q if $T=0$ and complemented if $T=1$. The characteristic table is useful during the analysis of sequential circuits when the value of flip-flop inputs are known and we want to find the value of the flip-flop output Q after the rising edge of the clock signal. As with any other truth table, we can use the map method to derive the characteristic equation for each flip-flop, which are shown in the third column of Table 1. During the design process we usually know the transition from present state to the next state and wish to find the flip-flop input conditions that will cause the required transition. For this reason we will need a table that lists the required inputs for a given change of state. Such a list is called the excitation table, which is shown in the fourth column of Table 1. There are four possible transitions from present state to the next state. The required input conditions are derived from the information available in the characteristic table. The symbol X in the table represents a "don't care" condition, that is, it does not matter whether the input is 1 or 0.

Synchronous and asynchronous sequential circuit

asynchronous system is a system whose outputs depend upon the order in which its input variables change and can be affected at any instant of time. Gate-type asynchronous systems are basically combinational circuits with feedback paths. Because of the feedback among logic gates, the system may, at times, become unstable. Consequently they are not often used. Synchronous type of system uses storage elements called flip-flops that are employed to change their binary value only at discrete instants of time. Synchronous sequential circuits use logic gates and flip-flop storage devices. Sequential circuits have a clock signal as one of their inputs. All state transitions in such circuits occur only when the clock value is either 0 or 1 or happen at the rising or falling edges of the clock depending on the type of memory elements used in the circuit. Synchronization is achieved by a timing device called a clock pulse generator. Clock pulses are distributed throughout the system in such a way that the flip-flops are affected only with the arrival of the synchronization pulse. Synchronous sequential circuits that use clock pulses in the inputs are called clocked-sequential circuits. They are stable and their timing can easily be broken down into independent discrete steps, each of which is considered separately. A clock signal is a periodic square wave that indefinitely switches from 0 to 1 and from 1 to 0 at fixed intervals. Clock cycle time or clock period: the time interval between two consecutive rising or falling edges of the clock.

Design of Sequential Circuits

The design of a synchronous sequential circuit starts from a set of specifications and culminates in a logic diagram or a list of Boolean functions from which a logic diagram can be obtained. In contrast to a combinational logic, which is fully specified by a truth table, a sequential circuit requires a state table for its specification. The first step in the design of sequential circuits is to obtain a state table or an equivalence representation, such as a state diagram. A synchronous sequential circuit is made up of flip-flops and combinational gates. The design of the circuit consists of choosing the flip-flops and then finding the combinational structure which, together with the flip-flops, produces a circuit that fulfils the required specifications. The number of flip-flops is determined from the number of states needed in the circuit. The recommended steps for the design of sequential circuits are set out below:



Analysis of a sequential circuit

We have examined a general model for sequential circuits. In this model the effect of all previous inputs on the outputs is represented by a state of the circuit. Thus, the output of the circuit at any time depends upon its current state and the input. These also determine the next state of the circuit. The relationship that exists among the inputs, outputs, present states and next states can be specified by either the **state table** or the **state diagram**.

State Table

The state table representation of a sequential circuit consists of three sections labelled *present state*, *next state* and *output*. The present state designates the state of flip-flops before the occurrence of a clock pulse. The next state shows the states of flip-flops after the clock pulse, and the output section lists the value of the output variables during the present state.

State Diagram

In addition to graphical symbols, tables or equations, flip-flops can also be represented graphically by a state diagram. In this diagram, a state is represented by a circle, and the transition between states is indicated by directed lines (or arcs) connecting the circles. An example of a state diagram is shown in Figure 3 below.

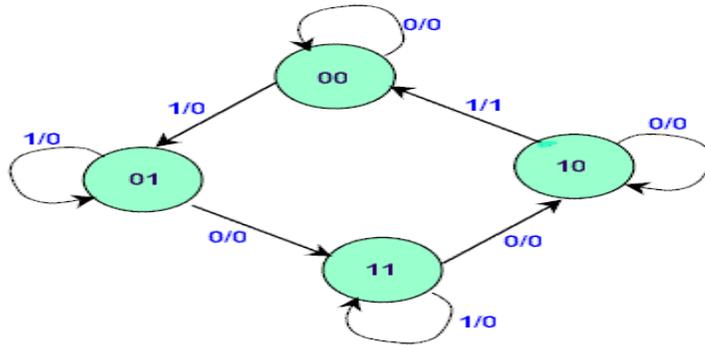


Figure 3. State Diagram The binary number inside each circle identifies the state the circle represents. The directed lines are labelled with two binary numbers separated by a slash (/). The input value that causes the state transition is labelled first. The number after the slash symbol / gives the value of the output. For example, the directed line from state 00 to 01 is labelled 1/0, meaning that, if the sequential circuit is in a present state and the input is 1, then the next state is 01 and the output is 0. If it is in a present state 00 and the input is 0, it will remain in that state. A directed line connecting a circle with itself indicates that no change of state occurs. The state diagram provides exactly the same information as the state table and is obtained directly from the state table.

Example: Consider a sequential circuit shown in Figure 4. It has one input x , one output Z and two state variables Q_1Q_2 (thus having four possible present states 00, 01, 10, 11).

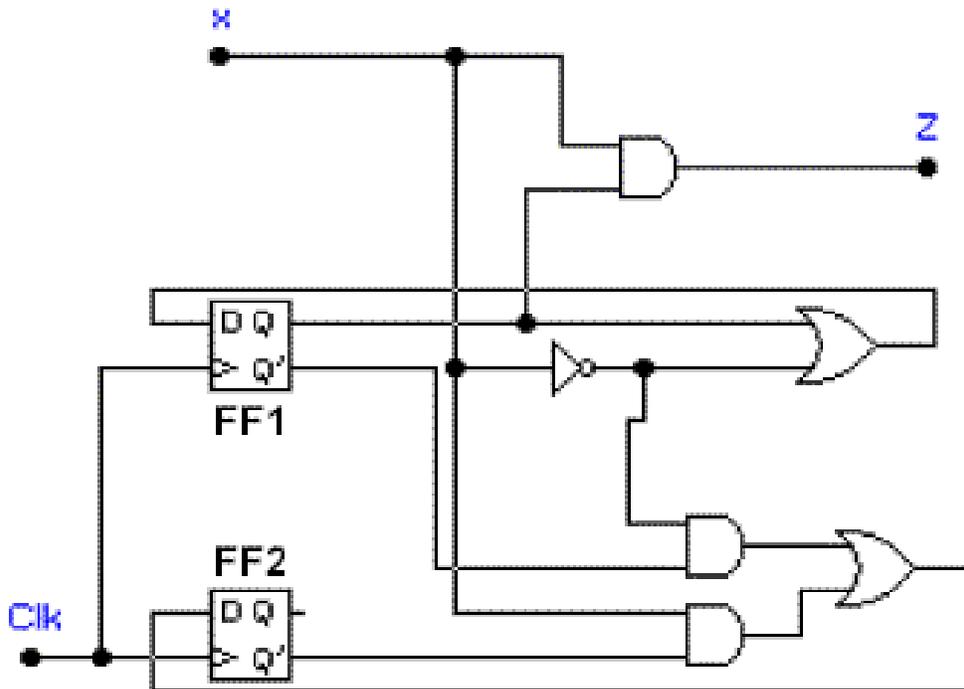


Figure 4. A Sequential Circuit

The behaviour of the circuit is determined by the following Boolean expressions:

$$Z = xQ_1$$

$$D_1 = x' + Q_1$$

$$D_2 = xQ_2' + x'Q_1'$$

These equations can be used to form the state table. Suppose the present state (i.e. Q_1Q_2) = 00 and input $x = 0$. Under these conditions, we get $Z = 0$, $D_1 = 1$, and $D_2 = 1$. Thus the next state of the circuit $D_1D_2 = 11$, and this will be the present state after the clock pulse has been applied. The output of the circuit corresponding to the present state $Q_1Q_2 = 00$ and $x = 1$ is $Z = 0$. This data is entered into the state table as shown in Table 2.

Present State Q1 Q2	Next State		Output (Z)	
	X = 0 Q1 Q0	X = 1 Q1 Q0	x = 0	x = 1
0 0	1 1	0 1	0	0
0 1	1 1	0 0	0	0
1 0	1 0	1 1	0	1
1 1	1 0	1 0	0	1

Table 2. State table for the sequential circuit in Figure 4.

The state diagram for the sequential circuit in Figure 4 is shown in Figure 5.

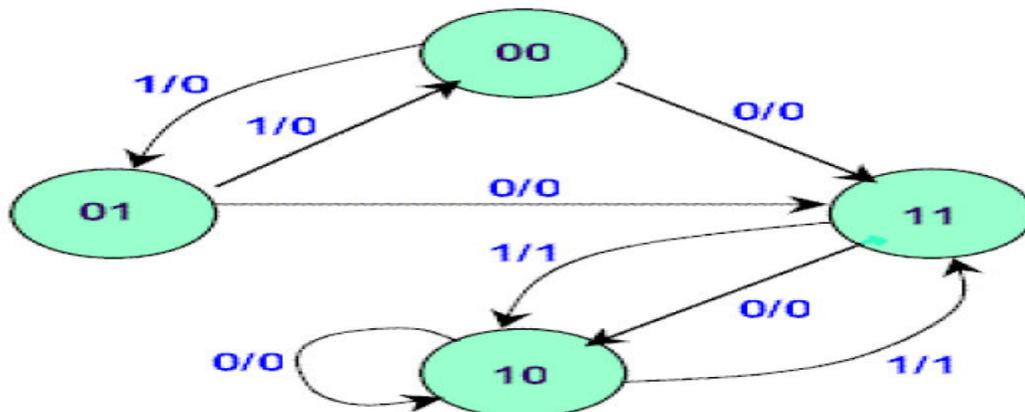


Figure 5. State Diagram of circuit in Figure 4.

State Diagrams of Various Flip-flops

Table 3 shows the state diagrams of the four types of flip-flops.

NAME	STATE DIAGRAM
SR	<p>State diagram for SR flip-flop. States: $Q=0$ and $Q=1$. Transitions: $S,R=0,0$ (self-loops), $S,R=1,0$ ($Q=0 \rightarrow Q=1$), $S,R=0,1$ ($Q=1 \rightarrow Q=0$).</p>
JK	<p>State diagram for JK flip-flop. States: $Q=0$ and $Q=1$. Transitions: $J,K=0,0$ (self-loops), $J,K=1,0$ or $1,1$ ($Q=0 \rightarrow Q=1$), $J,K=0,1$ or $1,1$ ($Q=1 \rightarrow Q=0$).</p>
D	<p>State diagram for D flip-flop. States: $Q=0$ and $Q=1$. Transitions: $D=1$ (self-loops), $D=1$ ($Q=0 \rightarrow Q=1$), $D=0$ ($Q=1 \rightarrow Q=0$).</p>
T	<p>State diagram for T flip-flop. States: $Q=0$ and $Q=1$. Transitions: $T=0$ (self-loops), $T=1$ ($Q=0 \rightarrow Q=1$), $T=1$ ($Q=1 \rightarrow Q=0$).</p>

Table 3. State diagrams of the four types of flip-flops. You can see from the table that all four flip-flops have the same number of states and transitions. Each flip-flop is in the set state when $Q=1$ and in the reset state when $Q=0$. Also, each flip-flop can move from one state to another, or it can re-enter the same state. The only difference between the four types lies in the values of input signals that cause these transitions. A state diagram is a very convenient way to visualise the operation of a flip-flop or even of large sequential components.

Example 1.1 Derive the state table and state diagram for the sequential circuit shown in Figure 7.

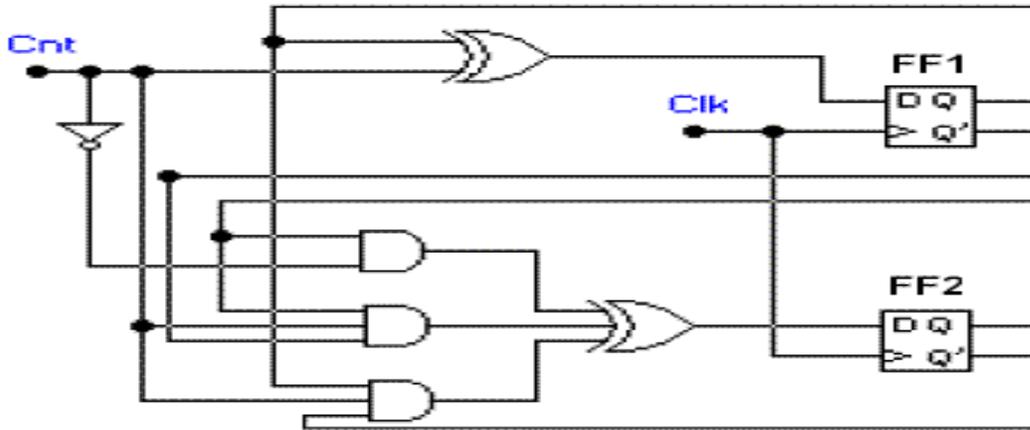


Figure 7. Logic schematic of a sequential circuit.

SOLUTION: STEP 1: First we derive the Boolean expressions for the inputs of each flip-flops in the schematic, in terms of external input Cnt and the flip-flop outputs Q_1 and Q_0 . Since there are two D flip-flops in this example, we derive two expressions for D_1 and D_0 :

$$D_0 = Cnt \oplus Q_0 = Cnt'Q_0 + CntQ_0'$$

$$D_1 = Cnt'Q_1 + CntQ_1'Q_0 + CntQ_1Q_0'$$

These Boolean expressions are called **excitation equations** since they represent the inputs to the flip-flops of the sequential circuit in the next clock cycle. **STEP 2:** Derive the next-state equations by converting these excitation equations into flip-flop characteristic equations. In the case of D flip-flops, $Q(\text{next}) = D$. Therefore the next state equal the excitation equations.

$$Q_0(\text{next}) = D_0 = Cnt'Q_0 + CntQ_0'$$

$$Q_1(\text{next}) = D_1 = Cnt'Q_1 + CntQ_1'Q_0 + CntQ_1Q_0'$$

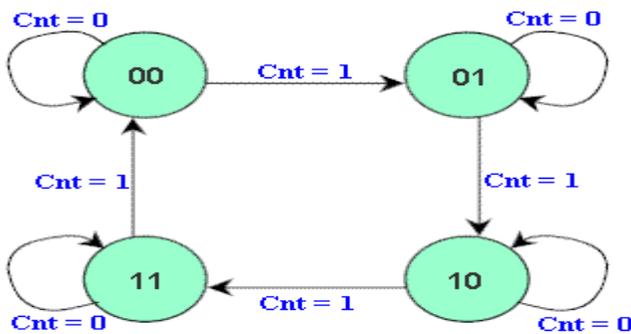
STEP 3: Now convert these next-state equations into tabular form called the next-state table.

Present State $Q_1 \quad Q_0$	Next State	
	$Cnt = 0$ $Q_1 \quad Q_0$	$Cnt = 1$ $Q_1 \quad Q_0$
0 0	0 0	0 1
0 1	0 1	1 0
1 0	1 0	1 1
1 1	1 1	0 0

Each row is corresponding to a state of the sequential circuit and each column represents one set of input values. Since we have two flip-flops, the number of possible states is four - that is, Q_1Q_0 can be equal to 00, 01, 10, or 11. These are present states as shown in the table.

For the next state part of the table, each entry defines the value of the sequential circuit in the next clock cycle after the rising edge of the **Clk**. Since this value depends on the present state and the value of the input signals, the next state table will contain one column for each assignment of binary values to the input signals. In this example, since there is only one input signal, **Cnt**, the next-state table shown has only two columns, corresponding to **Cnt = 0** and **Cnt = 1**. Note that each entry in the next-state table indicates the values of the flip-flops in the next state if their value in the present state is in the row header and the input values in the column header. Each of these next-state values has been computed from the next-state equations in STEP 2.

STEP 4: The state diagram is generated directly from the next-state table, shown in Figure 8.



Each arc is labelled with the values of the input signals that cause the transition from the present state (the source of the arc) to the next state (the destination of the arc).

Example 1.2

Derive the next state, the output table and the state diagram for the sequential circuit shown in Figure 10.

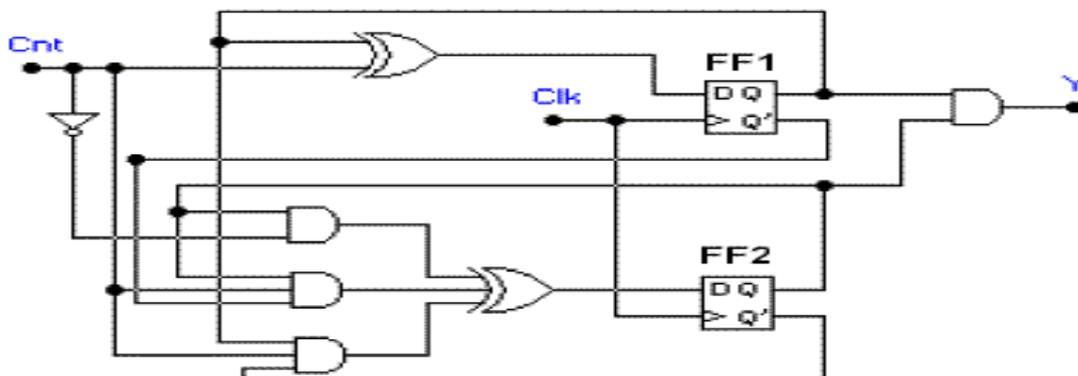


Figure 10. Logic schematic of a sequential circuit.

SOLUTION: The input combinational logic in Figure 10 is the same as in example1.1 so the excitation and the next-state equations will be the same as in

Example 1.1. **Excitation equations:**

$$D0 = \text{Cnt}Q0 = \text{Cnt}'Q0 + \text{Cnt}Q0'$$

$$D1 = \text{Cnt}'Q1 + \text{Cnt}Q1'Q0 + \text{Cnt}Q1Q0'$$

Next-state equations:

$$Q0(\text{next}) = D0 = \text{Cnt}'Q0 + \text{Cnt}Q0'$$

$$Q1(\text{next}) = D1 = \text{Cnt}'Q1 + \text{Cnt}Q1'Q0 + \text{Cnt}Q1Q0'$$

In addition, however, we have computed the output equation. **Output equation: $Y = Q1Q0$** As this equation shows, the output Y will equal to 1 when the counter is in state $Q1Q0 = 11$, and it will stay 1 as long as the counter stays in that state.

Next-state and output table:

Present State Q1 Q0	Next State		Output Z
	Cnt = 0 Q1 Q0	Cnt = 1 Q1 Q0	
0 0	0 0	0 1	0
0 1	0 1	1 0	0
1 0	1 0	1 1	0
1 1	1 1	0 0	1

State diagram:

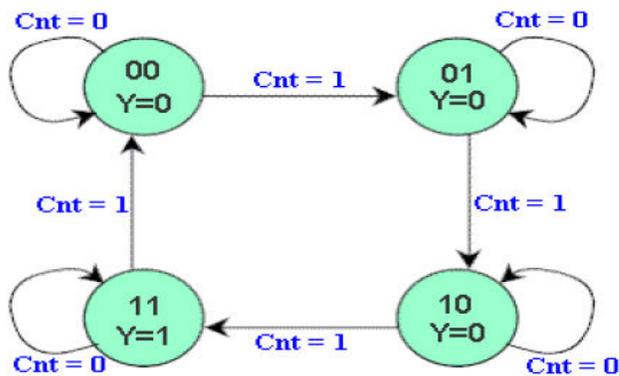


Figure 11. State diagram of sequential circuit in Figure 10.

State Reduction

Any design process must consider the problem of minimising the cost of the final circuit. The two most obvious cost reductions are reductions in the number of flip-flops and the number of gates.

The number of states in a sequential circuit is closely related to the complexity of the resulting circuit. It is therefore desirable to know when two or more states are equivalent in all aspects. The process of

eliminating the equivalent or redundant states from a state table/diagram is known as **state reduction**.

Example: Let us consider the state table of a sequential circuit shown in Table 6.

Present State	Next State		Output	
	x = 0	x = 1	x = 0	x = 1
A	B	C	1	0
B	F	D	0	0
C	D	E	1	1
D	F	E	0	1
E	A	D	0	0
F	B	C	1	0

Table 6. State table

It can be seen from the table that the present state A and F both have the same next states, B (when $x=0$) and C (when $x=1$). They also produce the same output 1 (when $x=0$) and 0 (when $x=1$). Therefore states A and F are equivalent. Thus one of the states, A or F can be removed from the state table. For example, if we remove row F from the table and replace all F's by A's in the columns, the state table is modified as shown in Table 7.

Present State	Next State		Output	
	x = 0	x = 1	x = 0	x = 1
A	B	C	1	0
B	A	D	0	0
C	D	B	1	1
D	A	B	0	1

Table 7. State F removed It is apparent that states B and E are equivalent. Removing E and replacing E's by B's results in the reduce table shown

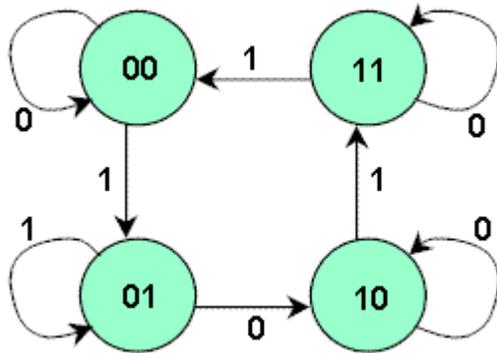
Present State	Next State		Output	
	x = 0	x = 1	x = 0	x = 1
A	B	C	1	0
B	A	D	0	0
C	D	E	1	1
D	A	E	0	1
E	A	D	0	0

Table 8. Reduced state table

The removal of equivalent states has reduced the number of states in the circuit from six to four. Two states are considered to be **equivalent** if and only if for every input sequence the circuit produces the same output sequence irrespective of which one of the two states is the starting state.

Example 1.3

We wish to design a synchronous sequential circuit whose state diagram is shown in Figure 13. The type of flip-flop to be use is J-K.



From the state diagram, we can generate the state table shown in Table 9. Note that there is no output section for this circuit. Two flip-flops are needed to represent the four states and are designated Q0Q1. The input variable is labelled x.

Present State Q0 Q1	Next State	
	X = 0 Q0 Q1	X = 1 Q0 Q1
0 0	0 0	0 1
0 1	1 0	0 1
1 0	1 0	1 1
1 1	1 1	0 0

Table 9. State table.

We shall now derive the excitation table and the combinational structure. The table is now arranged in a different form shown in Table 11, where the present state and input variables are arranged in the form of a truth table. Remember, the excitable for the JK flip-flop was derive in table 1

Table 10. Excitation table for JK flip-flop

Output Transitions	Flip-flop inputs
Q → Q(next)	J K
0 → 0	0 X
0 → 1	1 X
1 → 0	X 1
1 → 1	X 0

Example 1.4 Design a sequential circuit whose state tables are specified in Table 12, using D flip-flops.

Table 12. State table of a sequential circuit.

Present State Q ₀ Q ₁	Next State		Output	
	X = 0 Q ₀ Q ₁	X = 1 Q ₀ Q ₁	x = 0	x = 1
0 0	0 0	0 1	0	0
0 1	0 0	1 0	0	0
1 0	1 1	1 0	0	0
1 1	0 0	0 1	0	1

Table 13. Excitation table for a D flip-flop.

Output Transitions Q → Q(next)	Flip-flop inputs D
0 → 0	0
0 → 1	1
1 → 0	0
1 → 1	1

Next step is to derive the excitation table for the design circuit, which is shown in Table 14. The output of the circuit is labelled Z.

Present State Q ₀ Q ₁	Next State Q ₀ Q ₁	Input x	Flip-flop Inputs		Output Z
			D ₀	D ₁	
0 0	0 0	0	0	0	0
0 0	0 1	1	0	1	0
0 1	0 0	0	0	0	0
0 1	1 0	1	1	0	0
1 0	1 1	0	1	1	0
1 0	1 0	1	1	0	0
1 1	0 0	0	0	0	0
1 1	0 1	1	0	1	1

Table 14. Excitation table

Now plot the flip-flop inputs and output functions on the Karnaugh map to derive the Boolean expressions, which is shown in Figure 16.

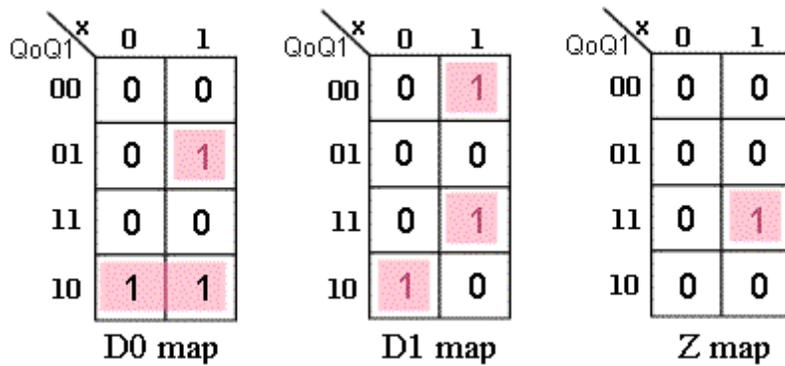


Figure 16. Karnaugh maps

The simplified Boolean expressions

are: $D0 = Q0*Q1' + Q0'*Q1*x$

$D1 = Q0'*Q1'*x + Q0*Q1*x + Q0*Q1'*x'$

$Z = Q0*Q1*x$

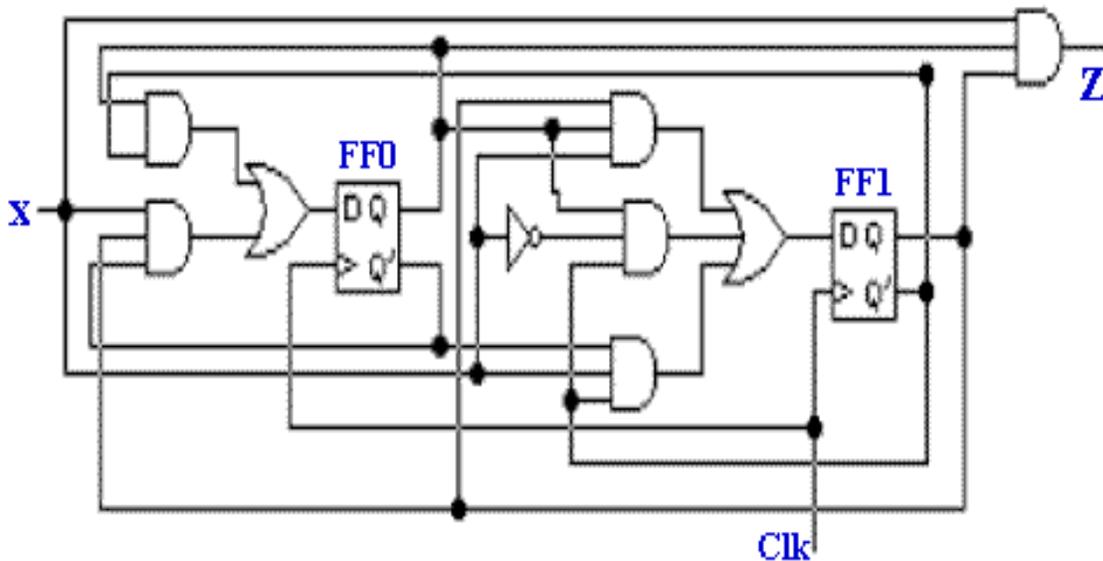


Figure 17. Logic diagram of the sequential circuit.

Register

A register is a sequential circuit with $n + 1$ (not counting the clock) inputs and n output. To each of the outputs corresponds an input. The first n inputs will be called x_0 through x_{n-1} and the last input will be called ld (for load). The n outputs will be called y_0 through y_{n-1} . When the ld input is 0, the outputs are unaffected by any clock transition. When the ld input is 1, the x inputs are stored in the register at the next clock transition, making the y outputs into copies of the x inputs before the clock transition. We can

explain this behavior more formally with a state table. As an example, let us take a register with $n = 4$. The left side of the state table contains 9 columns, labeled $x_0, x_1, x_2, x_3, Id, y_0, y_1, y_2,$ and y_3 . This means that the state table has 512 rows. We will therefore abbreviate it.

Id	x_3	x_2	x_1	x_0	y_3	y_2	y_1	y_0	y_3'	y_2'	y_1'	y_0'
0	--	--	--	--	c_3	c_2	c_1	c_0	c_3	c_2	c_1	c_0
1	c_3	c_2	c_1	c_0	--	--	--	--	c_3	c_2	c_1	c_0

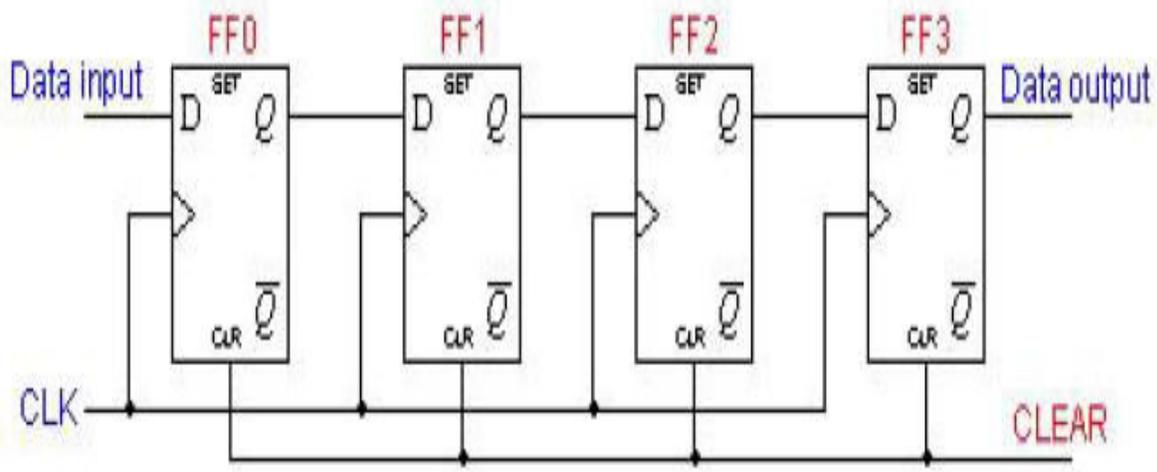
As you can see, when Id is 0 (the top half of the table), the right side of the table is a copy of the values of the old outputs, independently of the inputs. When Id is 1, the right side of the table is instead a copy of the values of the inputs, independently of the old values of the outputs. Registers play an important role in computers. Some of them are visible to the programmer, and are used to hold variable values for later use. Some of them are hidden to the programmer, and are used to hold values that are internal to the central processing unit, but nevertheless important.

Shift registers

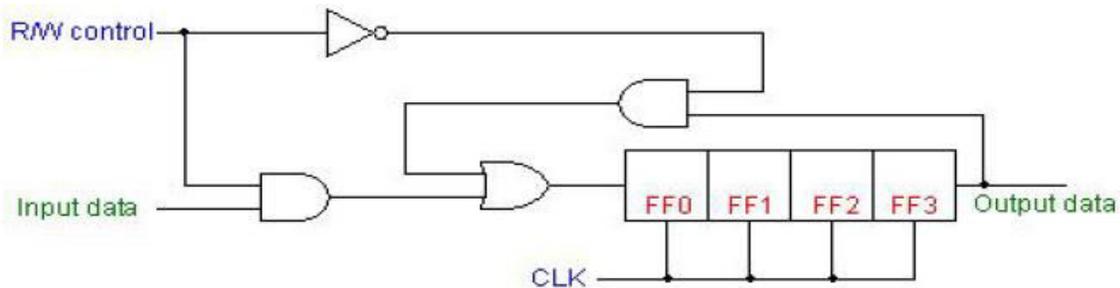
Shift registers are a type of sequential logic circuit, mainly for storage of digital data. They are a group of flip-flops connected in a chain so that the output from one flip-flop becomes the input of the next flip-flop. Most of the registers possess no characteristic internal sequence of states. All the flip-flops are driven by a common clock, and all are set or reset simultaneously. In this section, the basic types of shift registers are studied, such as Serial In - Serial Out, Serial In - Parallel Out, Parallel In - Serial Out, Parallel In - Parallel Out, and bidirectional shift registers. A special form of counter - the shift register counter, is also introduced.

Serial In - Serial Out Shift Registers

A basic four-bit shift register can be constructed using four D flip-flops, as shown below. The operation of the circuit is as follows. The register is first cleared, forcing all four outputs to zero. The input data is then applied sequentially to the D input of the first flip-flop on the left (FF0). During each clock pulse, one bit is transmitted from left to right. Assume a data word to be 1001. The least significant bit of the data has to be shifted through the register from FF0 to FF3.



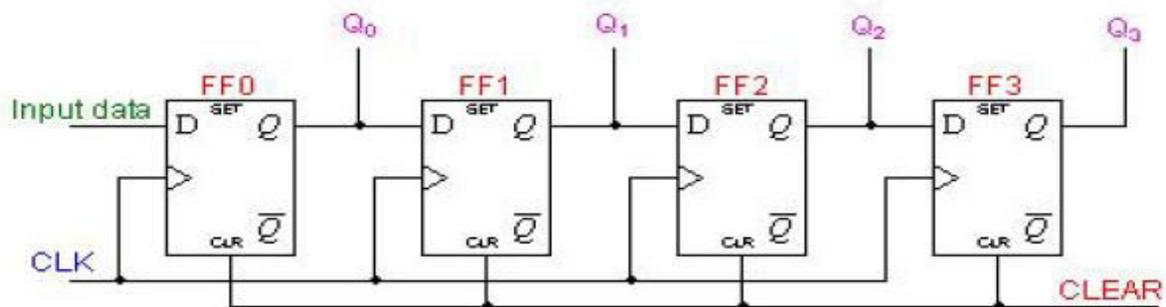
order to get the data out of the register, they must be shifted out serially. This can be done destructively or non-destructively. For destructive readout, the original data is lost and at the end of the read cycle, all flip-flops are reset to zero. To avoid the loss of data, an arrangement for a non-destructive reading can be done by adding two AND gates, an OR gate and an inverter to the system. The construction of this circuit is shown below.



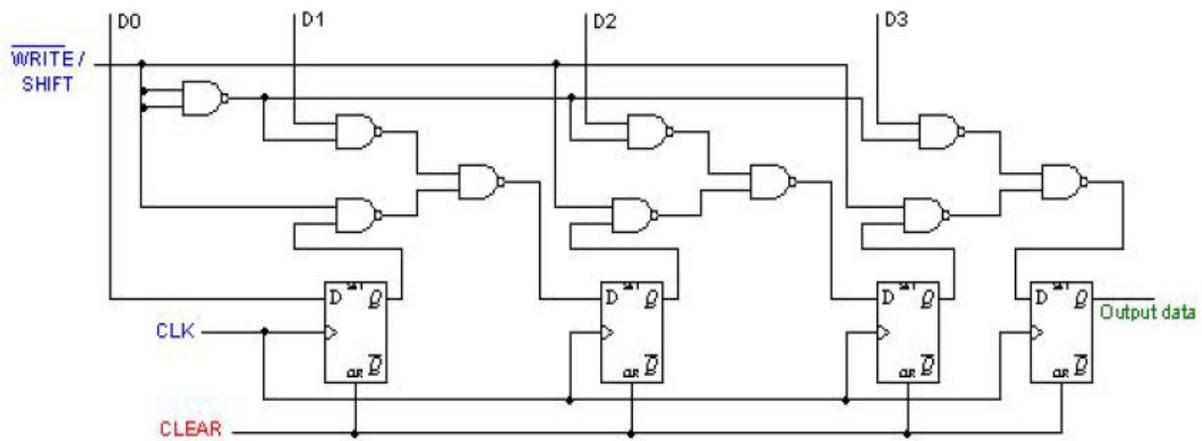
The data is loaded to the register when the control line is HIGH (ie WRITE). The data can be shifted out of the register when the control line is LOW (ie READ)

Serial In - Parallel Out Shift Registers

For this kind of register, data bits are entered serially in the same manner as discussed in the last section. The difference is the way in which the data bits are taken out of the register. Once the data are stored, each bit appears on its respective output line, and all bits are available simultaneously. A construction of a four-bit serial in - parallel out register is shown below.



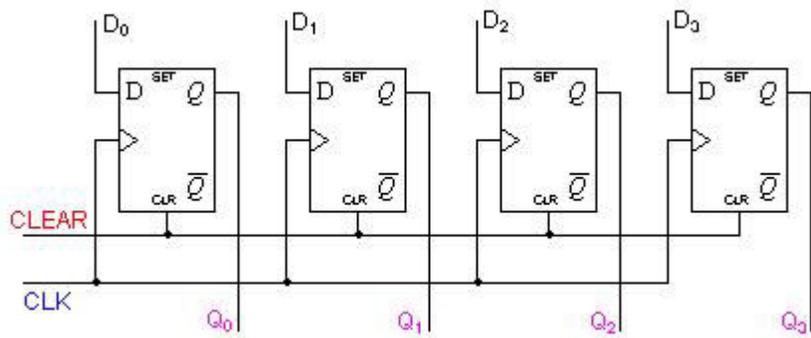
A four-bit parallel in - serial out shift register is shown below. The circuit uses D flip-flops and NAND gates for entering data (ie writing) to the register.



D0, D1, D2 and D3 are the parallel inputs, where D0 is the most significant bit and D3 is the least significant bit. To write data in, the mode control line is taken to LOW and the data is clocked in. The data can be shifted when the mode control line is HIGH as SHIFT is active high

Parallel In - Parallel Out Shift Registers

For parallel in - parallel out shift registers, all data bits appear on the parallel outputs immediately following the simultaneous entry of the data bits. The following circuit is a four-bit parallel in - parallel out shift register constructed by D flip-flops.



The D's are the parallel inputs and the Q's are the parallel outputs. Once the register is clocked, all the data at the D inputs appear at the corresponding Q outputs simultaneously.

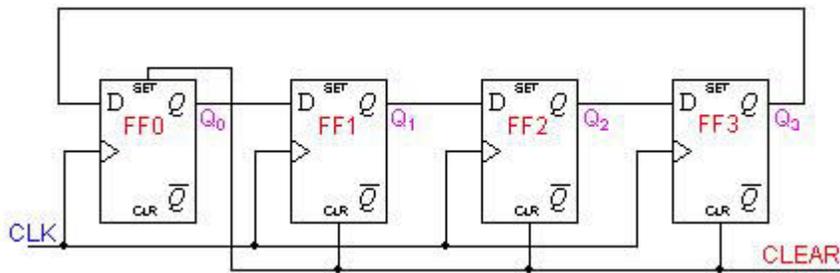
Shift Register Counters

Two of the most common types of shift register counters are introduced here: the Ring counter and the Johnson counter. They are basically shift registers with the serial outputs connected back to the serial inputs in order to produce particular sequences. These registers are classified as counters because they exhibit a specified sequence of states.

Ring Counters

A ring counter is basically a circulating shift register in which the output of the most significant stage is fed back to the input of the least significant stage. The following is a 4-bit ring counter constructed from D flip-flops. The output of each stage is shifted into the next stage on the positive edge of a clock pulse.

If the CLEAR signal is high, all the flip-flops except the first one FF0 are reset to 0. FF0 is preset to 1 instead.

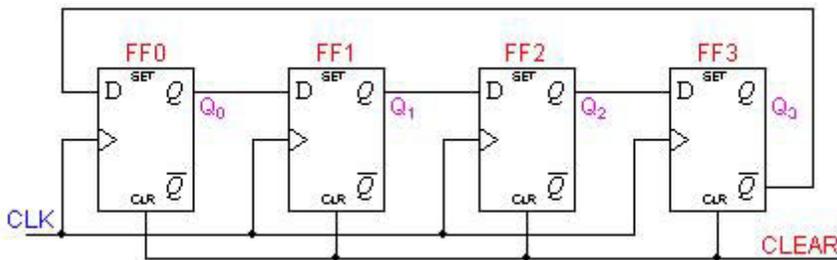


Since the count sequence has 4 distinct states, the counter can be considered as a mod-4 counter. Only 4 of the maximum 16 states are used, making ring counters very inefficient in terms of state usage. But the major advantage of a ring counter over a binary counter is that it is self-decoding. No extra decoding circuit is needed to determine what state the counter is in.

Clock Pulse	Q3	Q2	Q1	Q0
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	1	0	0	0

Johnson Counters

Johnson counters are a variation of standard ring counters, with the inverted output of the last stage fed back to the input of the first stage. They are also known as twisted ring counters. An n -stage Johnson counter yields a count sequence of length $2n$, so it may be considered to be a mod- $2n$ counter. The circuit above shows a 4-bit Johnson counter. The state sequence for the counter is given in the table



Clock Pulse	Q3	Q2	Q1	Q0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	1	1	1
4	1	1	1	1
5	1	1	1	0
6	1	1	0	0
7	1	0	0	0

Again, the apparent disadvantage of this counter is that the maximum available states are not fully utilized. Only eight of the sixteen states are being used.

Counters

A sequential circuit that goes through a prescribed sequence of states upon the application of input pulses is called a **counter**. The input pulses, called **count pulses**, may be clock pulses. In a counter, the sequence of states may follow a binary count or any other sequence of states. Counters are found in almost all equipment containing digital logic. They are used for counting the number of occurrences of an even and are useful for generating timing sequences to control operations in a digital system. A *counter* is a sequential circuit with 0 inputs and n outputs. Thus, the value after the clock transition depends only on old values of the outputs. For a counter, the values of the outputs are interpreted as a sequence of *binary digits* (see the section on binary arithmetic). We shall call the outputs o_0, o_1, \dots, o_{n-1} . The value of the outputs for the counter after a clock transition is a binary number which is *one plus* the binary number of the outputs before the clock transition. We can explain this behavior more formally with a state table. As an example, let us take a counter with $n = 4$. The left side of the state table contains 4 columns, labeled $o_0, o_1, o_2,$ and o_3 . This means that the state table has 16 rows. Here it is in full:

o_3	o_2	o_1	o_0	o_3'	o_2'	o_1'	o_0'
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	1	0	1	0
1	0	1	0	1	0	1	1
1	0	1	1	1	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	0	0	0	0