

COURSE MATERIAL

UNIT 3

SEC1207-DIGITAL LOGIC CIRCUITS

SYLLABUS

UNIT III DESIGN OF SEQUENTIAL CIRCUITS

9 Hrs.

Introduction to Sequential circuits - Flip flops - SR, JK, D and T flip flops, Master Slave flip flops, Characteristic and excitation table - Realization of one flip flop with other flip flops - Registers - Shift registers - Counters - Synchronous and Asynchronous counters - Modulus counters, Up/Down counters - Ring Counter - Johnson Counter - State diagram, State table, State minimization - Hazards.

TABLE OF TOPICS

S.NO	TOPIC	PAGE NO.
3.1	Introduction to Sequential circuits	
3.2	Flip flops	
3.2.1	SR flip flops	
3.2.2	JK flip flops	
3.2.3	D flip flops	
3.2.4	T flip flops	
3.3	Master Slave flip flops	
3.4	Characteristic and excitation table	
3.5	Realization of one flip flop with other flip flops	
3.6	Registers	
3.6.1	Shift registers	
3.7	Counters	
3.7.1	Synchronous and Asynchronous counters	
3.7.1.1	<i>Modulus counters</i>	
3.7.1.2	<i>Up/Down counters</i>	
3.7.1.3	<i>Ring Counter</i>	
3.7.1.4	<i>Johnson Counter</i>	
3.8	State diagram, State table, State minimization.	
3.9	Hazards	

3.1 Introduction to Sequential circuits

Digital electronics is classified into **combinational logic** and **sequential logic**.

Combinational logic output depends on the present inputs levels, whereas sequential logic output not only depends on the input levels, but also stored levels (previous output history).

Combinational Circuits

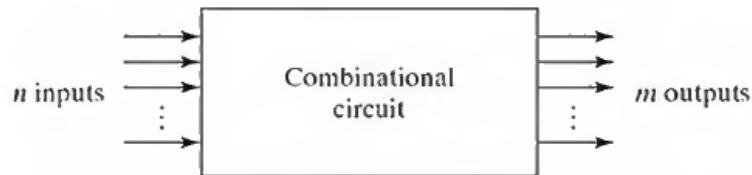
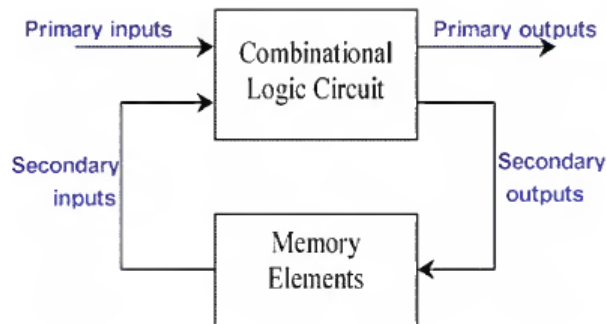


Fig. Block Diagram of Combinational Circuit

Sequential Circuits



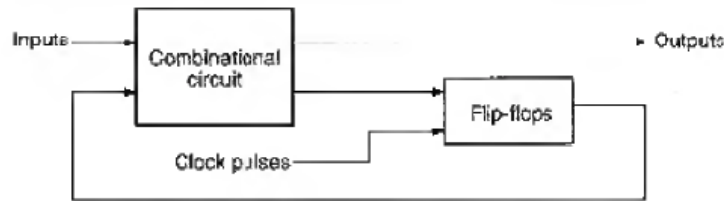
The memory elements are devices capable of storing binary info. The binary info stored in the memory elements at any given time defines the state of the sequential circuit. The input and the present state of the memory element determine the output. Memory elements next state is also a function of external inputs and present state. A sequential circuit is specified by a time sequence of inputs, outputs, and internal states.

There are two types of sequential circuits. Their classification depends on the timing of their signals:

- ❖ Synchronous sequential circuits
- ❖ Asynchronous sequential circuits

Asynchronous sequential circuit

This is a system whose outputs depend upon the order in which its input variables change and can be affected at **any instant of time**.



(a) Block diagram

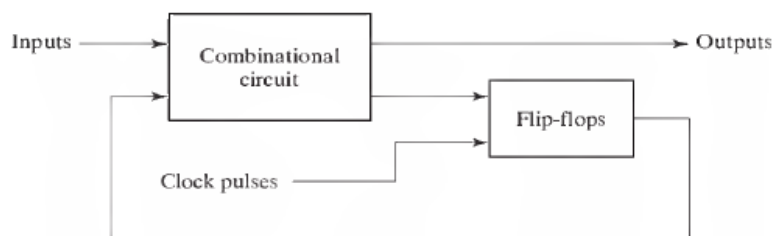


(b) Timing diagram of clock pulses

Synchronous sequential circuits

This 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.

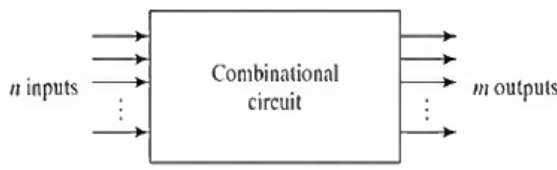
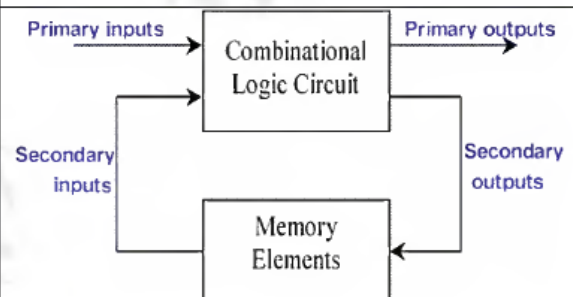


(a) Block diagram



(b) Timing diagram of clock pulses

Fig. 5-2 Synchronous Clocked Sequential Circuit

Combinational Circuits	Sequential Circuits
1. The circuit whose output at any instant depends only on the input present at that instant only is known as combinationational circuit.	1. The circuit whose output at any instant depends not only on the input present but also on the past output a is known as sequential circuit
2. This type of circuit has no memory unit.	2. This type of circuit has memory unit for store past output.
3. Examples of combinational circuits are half adder, full adder, magnitude comparator, multiplexer, demultiplexer e.t.c.	3. Examples of sequential circuits are Flip flop, register, counter e.t.c.
4. Faster in Speed	Slower compared to Combinational Circuit
<p style="text-align: center;">Combinational Circuits</p>  <p style="text-align: center;">Fig. Block Diagram of Combinational Circuit</p>	

A sequential circuit can further be categorized into **Synchronous** and **Asynchronous**.

Here is the difference between synchronous and asynchronous sequential circuits:

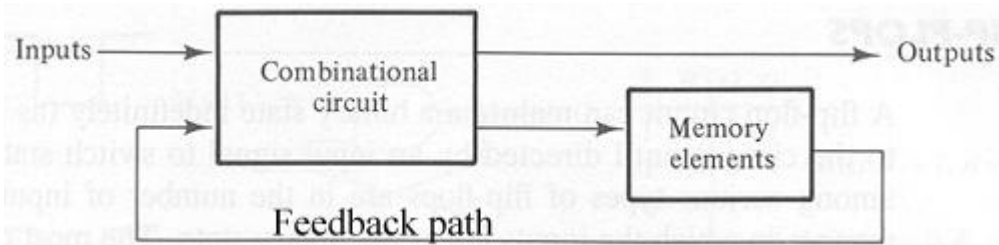
Synchronous Sequential Circuit: Output changes at discrete interval of time. It is a circuit based on an equal state time or a state time defined by external means such as clock. Examples of synchronous sequential circuit are Flip Flops, Synchronous Counter.

Asynchronous Sequential Circuit: Output can be changed at any instant of time by changing the input. It is a circuit whose state time depends solely upon the internal logic circuit delays. Example of asynchronous sequential circuit is Asynchronous Counter.

A sequential circuit is a combinational circuit with some feedback from the outputs. In a sequential circuit, the output state depends on both the inputs and the outputs. The term "sequential" comes from the fact that the output depends not only on the current states, but on the states immediately preceding.

There are two types of sequential circuit, synchronous and asynchronous. Asynchronous sequential circuits do not use a clock signal as synchronous circuits do. Instead the circuit is

driven by the pulses of the inputs. Synchronous types use pulsed or level inputs and a clock input to drive the circuit.



3.2 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.

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

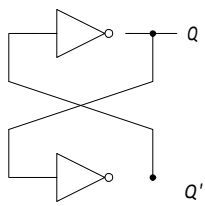


Figure 1. Bistable element.

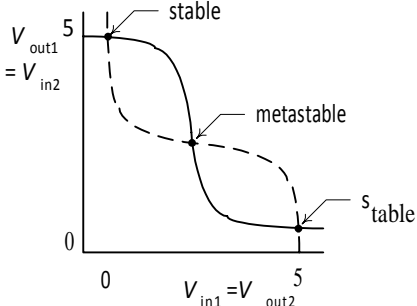


Figure 2. Analog analysis of bistable element

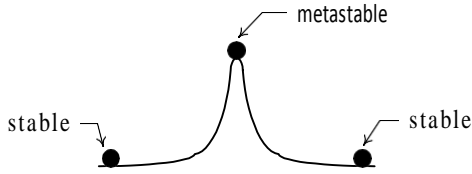


Figure 3. Ball and hill analogy for metastable behavior.

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.

SRLatch

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

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

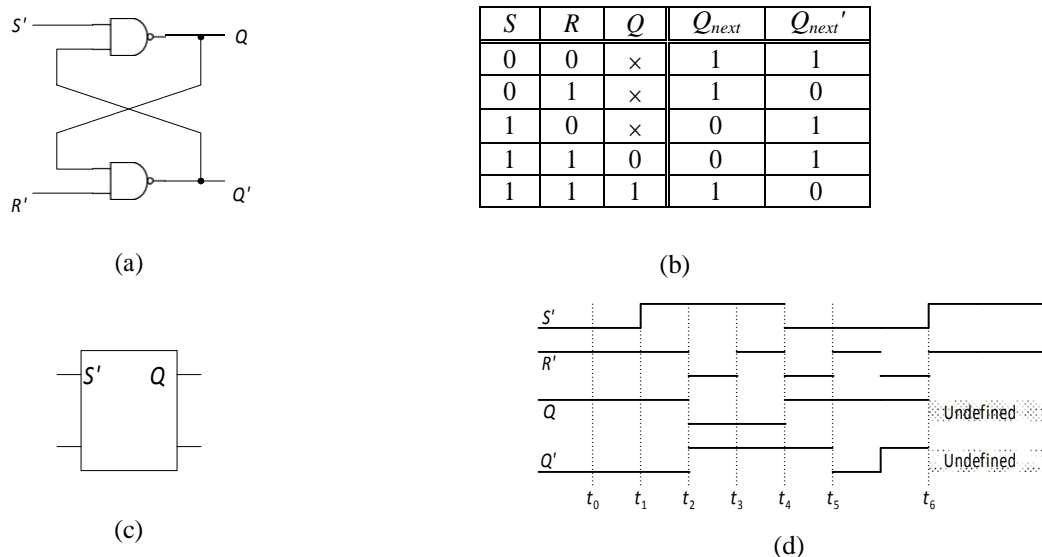


Figure 4. SR latch: (a) circuit using NAND gates; (b) truth table; (c) logic symbol; (d) timing diagram.

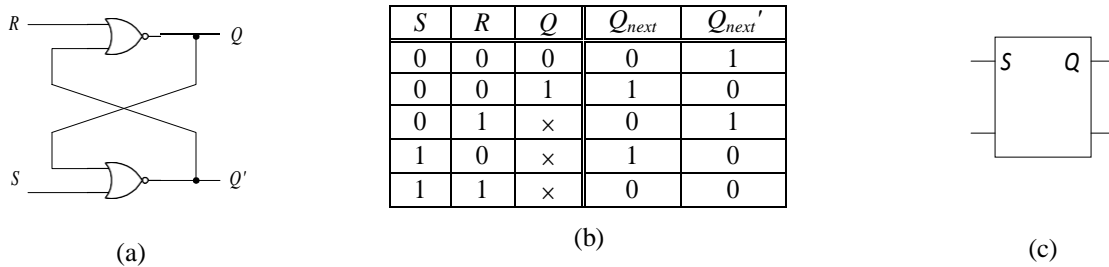


Figure 5. SR latch: (a) circuit using NOR gates; (b) truth table; (c) logic symbol.

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'.

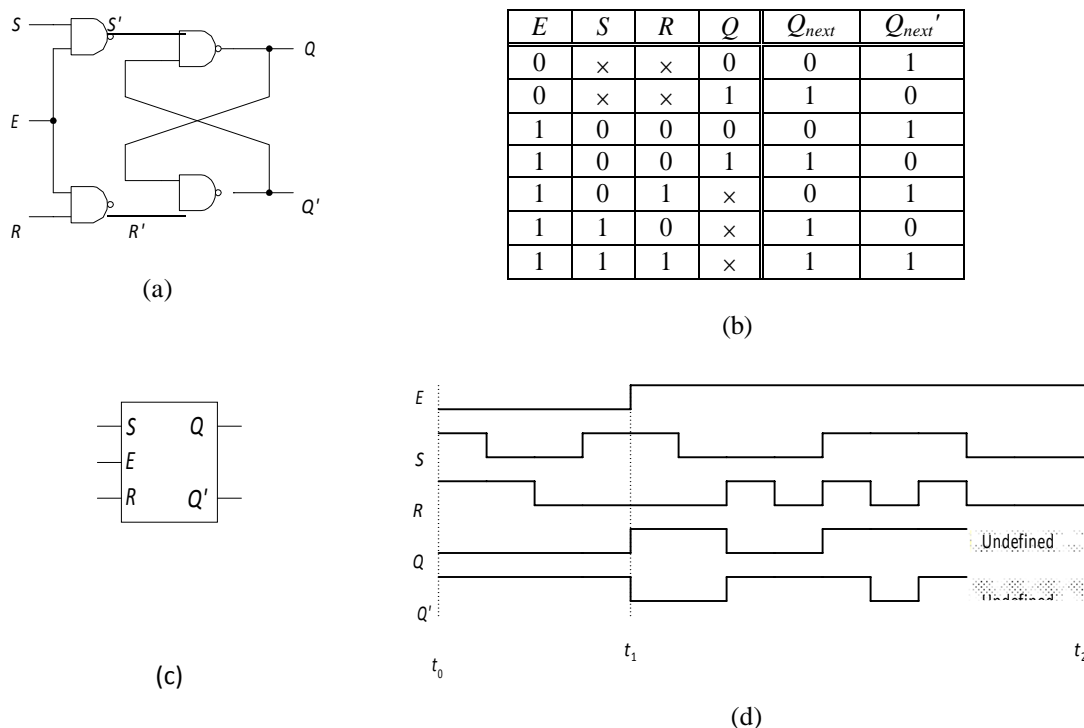
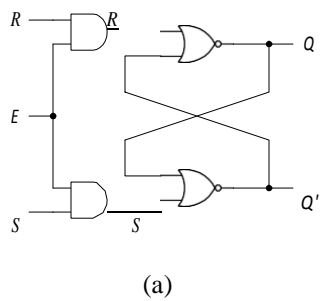


Figure 6. SR latch with enable: (a) circuit using NAND gates; (b) truth table; (c) logic symbol; (d) timing diagram.



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

Figure7. SR latch with enable: (a) circuit using NOR gates; (b) truthtable.

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).

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.

D Latch

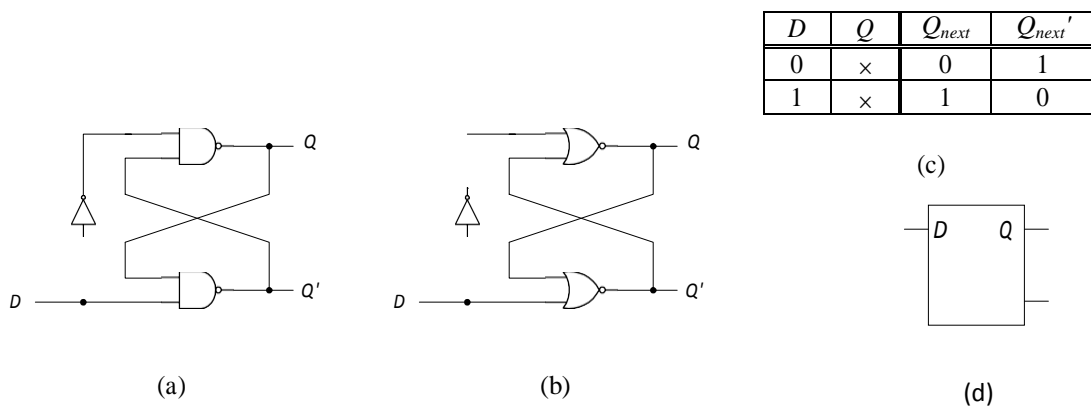
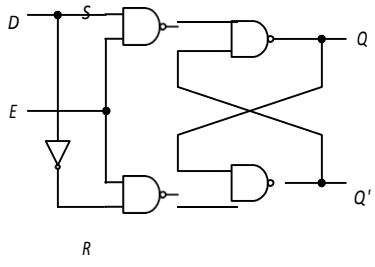


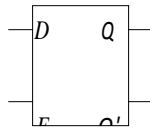
Figure8. D latch: (a) circuit using NAND gates; (b) circuit using NOR gates; (c) truth table; (d) logic symbol.



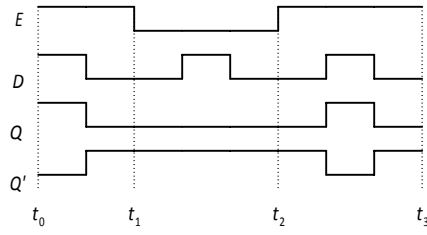
(a)

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

(b)



(c)



(d)

Figure9. D latch with enable: (a) circuit using NAND gates; (b) truth table; (c) logic symbol;(d)timingdiagram.

3.2.1 SR flip flops

2. S-R Flip Flop:

The SET-RESET flip flop is not designed with the help of two NOR gates and also two NAND gates. These flip flops are also called S-R Latch.

S-R Flip Flop using NOR Gate

The design of such a flip flop includes two inputs, called the SET [S] and RESET [R]. There are also two outputs, Q and Q'. The diagram and truth table is shown below.

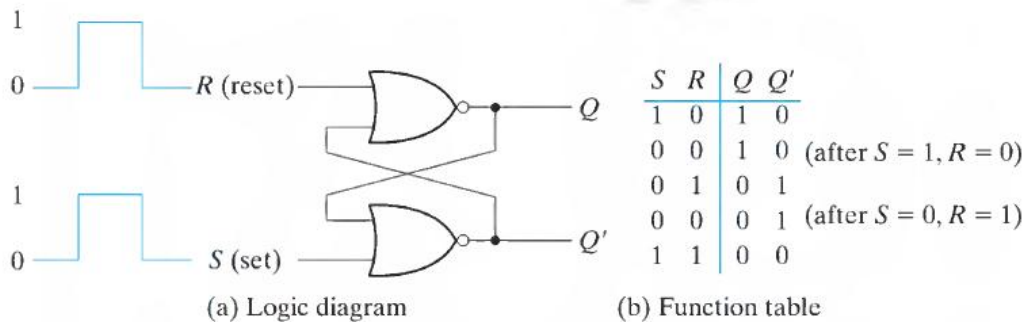


Fig. 5-3 SR Latch with NOR Gates

The operation has to be analyzed with the 4 inputs combinations together with the 2 possible previous states.

From the diagram it is evident that the flip flop has mainly four states. They are

1. When S=1, R=0 the output becomes Q=1, Q'=0

This SR flip flop function table is constructed based on the XOR gate. In XOR gate if any of the input is 1 the output becomes 1.

In this state when S=1 and R=0 the output Q becomes set (1). So this state is also called the SET state.

2. When S=0, R=1, the output becomes Q=0, Q'=1

In this state When R=1 it resets the output. So this state is known as the RESET state.

In both the states you can see that the outputs are just compliments of each other and that the value of Q follows the value of S.

3. When S=0, R=0 the output is Q & Q' = Remember (memory)

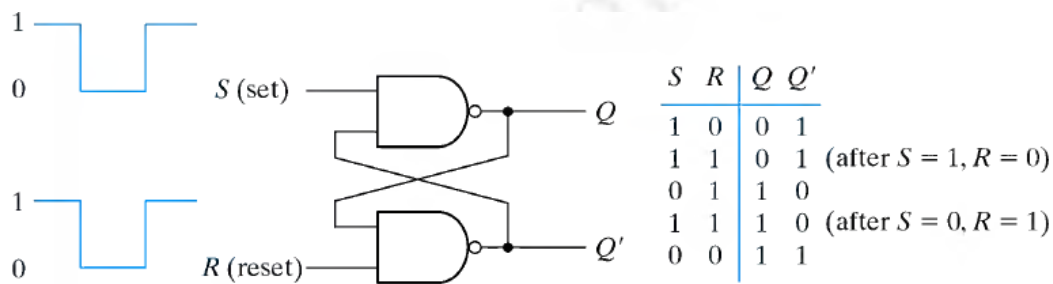
If both the values of S and R are switched to 0, then the circuit remembers the value of S and R in their previous state.

4. When S=1, R=1 the output Q=0, Q'=0 [Invalid]

This is an invalid state because the values of both Q and Q' are 0. They are supposed to be compliments of each other. Normally, this state must be avoided.

S-R Flip Flop using NAND Gate

The above SR flip flop can be constructed using NAND gate.



(a) Logic diagram

(b) Function table

Fig. 5-4 SR Latch with NAND Gates

Like the NOR Gate S-R flip flop, this one also has four states. They are

1. S=1, R=0, Q=0, Q'=1

This state is also called the SET state.

2. S=0, R=1, Q=1, Q'=0

This state is known as the RESET state.

In both the states you can see that the outputs are just compliments of each other and that the value of Q follows the compliment value of S.

3. $S=0, R=0, Q=1, \& Q' =1$ [Invalid]

If both the values of S and R are switched to 0 it is an invalid state because the values of both Q and Q' are 1. They are supposed to be compliments of each other. Normally, this state must be avoided.

4. $S=1, R=1, Q \& Q' =$ Remember

If both the values of S and R are switched to 1, then the circuit remembers the value of S and R in their previous state.

Clocked S-R Flip Flop

- ❖ It is also called a Gated S-R flip flop.
- ❖ The problems with S-R flip flops using NOR and NAND gate is the invalid state.
- ❖ This problem can be overcome by using a bistable SR flip-flop that can change outputs when certain invalid states are met, regardless of the condition of either the Set or the Reset inputs.

3. $S=0, R=0, Q=1, \& Q' =1$ [Invalid]

If both the values of S and R are switched to 0 it is an invalid state because the values of both Q and Q' are 1. They are supposed to be compliments of each other. Normally, this state must be avoided.

4. $S=1, R=1, Q \& Q' =$ Remember

If both the values of S and R are switched to 1, then the circuit remembers the value of S and R in their previous state.

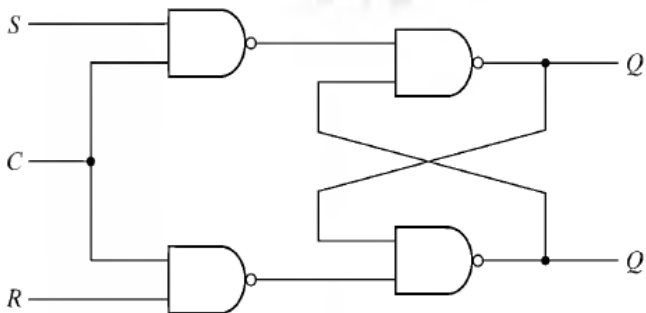
Clocked S-R Flip Flop

- ❖ It is also called a Gated S-R flip flop.
- ❖ The problems with S-R flip flops using NOR and NAND gate is the invalid state.
- ❖ This problem can be overcome by using a bistable SR flip-flop that can change outputs when certain invalid states are met, regardless of the condition of either the Set or the Reset inputs.

❖ For this, a clocked S-R flip flop is designed by adding two AND gates to a basic NOR Gate flip flop.

❖ The circuit diagram and truth table is shown below.

The circuit of the S-R flip flop using NAND Gate and its truth table is shown below.



(a) Logic diagram

C	S	R	Next state of Q
0	X	X	No change
1	0	0	No change
1	0	1	Q = 0; Reset state
1	1	0	Q = 1; set state
1	1	1	Indeterminate

(b) Function table

Fig. 5-5 SR Latch with Control Input

- A clock pulse [CP] is given to the inputs of the AND Gate.
- When the value of the clock pulse is '0', the outputs of both the AND Gates remain '0'.
- As soon as a pulse is given the value of CP turns '1'.
- This makes the values at S and R to pass through the NOR Gate flip flop. But when the values of both S and R values turn '1', the HIGH value of CP causes both of them to turn to '0' for a short moment.
- As soon as the pulse is removed, the flip flop state becomes intermediate.
- Thus either of the two states may be caused, and it depends 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. Thus the invalid states can be eliminated.

Excitation Table of the SR Latch

- During the design process we usually know the transition from present state to next state and wish to find the latch input conditions that will cause the required transition.
- For this reason, we need a table that lists the required inputs for a given change of state. Such a table is called an excitation table, and it specifies the excitation behavior of the sequential circuits. These are used in the synthesis (design) of sequential circuits, which we shall see later.

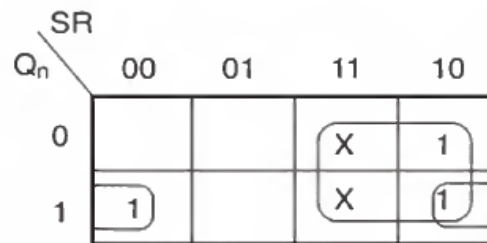
- The excitation of the SR latch is as follows:

Excitation Table:

Q _n	S	R	Q _{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	Indeter
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	Indeter

Note: Indeter = not used

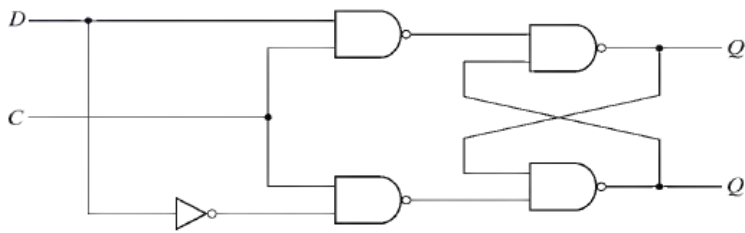
K Map for Q_{n+1}:



$$Q_{n+1} = S + \bar{R} \cdot Q_n$$

3. D Flip Flop

- D flip flop is actually a slight modification of the above explained clocked SR flip-flop. From the figure you can see that the D input is connected to the S input and the complement of the D input is connected to the R input.
- The D input is passed on to the flip flop when the value of CP is '1'.
- When CP is HIGH, the flip flop moves to the SET state. If it is '0', the flip flop switches to the CLEAR state.
- As long as the clock input C = 0, the SR latch has both inputs equal to 0 and it can't change its state regardless of the value of D
- When C is 1, the latch is placed in the set or reset state based on the value of D.
 - If D = 1, the Q output goes to 1.
 - If D = 0, the Q output goes to 0.



(a) Logic diagram

C	D	Next state of Q
0	X	No change
1	0	Q = 0; Reset state
1	1	Q = 1; Set state

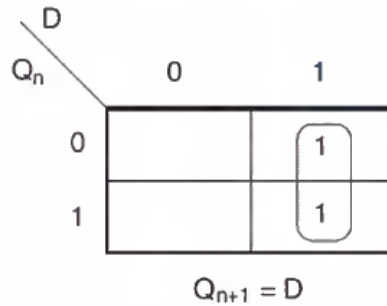
(b) Function table

Fig. 5-6 D Latch

Excitation Table:

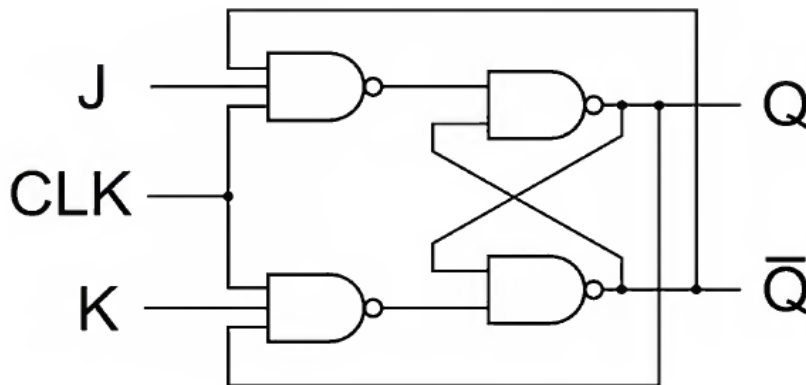
Q_n	D	Q_{n+1}
0	0	0
0	1	1
1	0	0
1	1	1

K- Map for Q_{n+1} :



3. J-K Flip Flop

❖ A J-K flip flop can also be defined as a modification of the S-R flip flop. The only difference is that the intermediate state is more refined and precise than that of a S-R flip flop.



- ❖ The behavior of inputs J and K is same as the S and R inputs of the S-R flip flop. The letter J stands for SET and the letter K stands for CLEAR.
- ❖ When both the inputs J and K have a HIGH state, the flip-flop switches to the complement state. So, for a value of $Q = 1$, it switches to $Q=0$ and for a value of $Q = 0$, it switches to $Q=1$.

- ❖ The circuit includes two 3-input AND gates. The output Q of the flip flop is returned back as a feedback to the input of the AND along with other inputs like K and clock pulse [CP].
- ❖ So, if the value of CP is '1', the flip flop gets a CLEAR signal and with the condition that the value of Q was earlier 1.
- ❖ Similarly output Q' of the flip flop is given as a feedback to the input of the AND along with other inputs like J and clock pulse [CP].
- ❖ So the output becomes SET when the value of CP is 1 only if the value of Q' was earlier 1.
- ❖ The output may be repeated in transitions once they have been complimented for J=K=1 because of the feedback connection in the JK flip-flop.
- ❖ This can be avoided by setting a time duration lesser 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.

Characteristic table:

Clk	J	K	Q _{n+1}
0	X	X	Memory
1	0	0	Memory
1	0	1	0
1	1	0	1 (Set)
1	1	1	Toggle

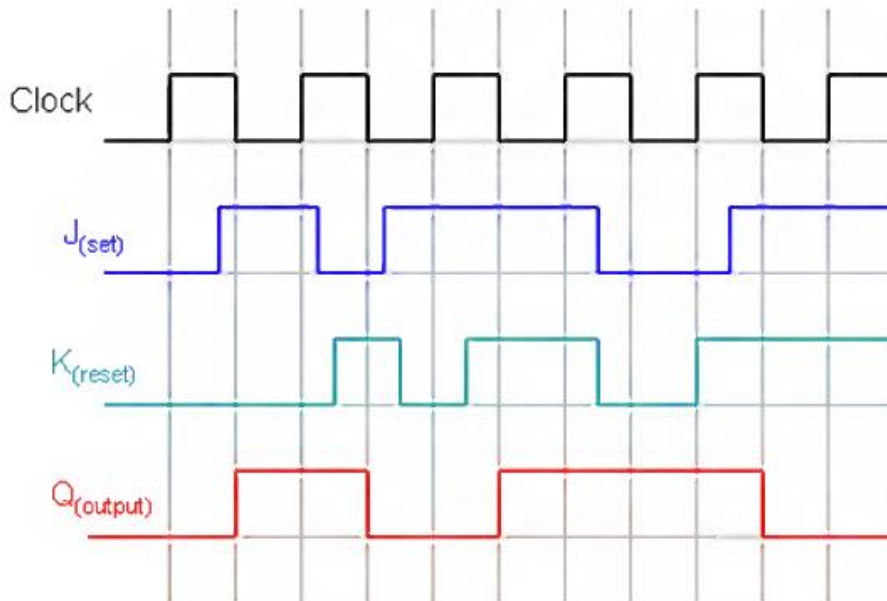
Excitation table for JK Flipflop K map for Q_{n+1}:

Q _n	J	K	Q _{n+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

Q _n	00	01	11	10
0			1	1
1	1			1

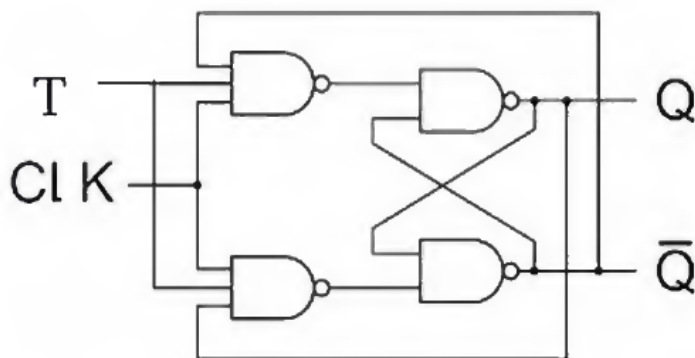
$$Q_{n+1} = J \cdot \overline{Q_n} + \overline{K} \cdot Q_n$$

Timing Diagram:



4. T Flip Flop

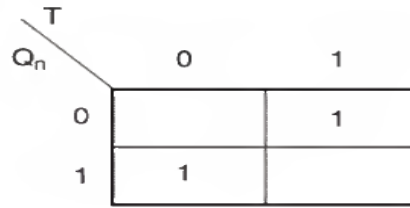
- ❖ This is a much simpler version of the J-K flip flop.
- ❖ Both the J and K inputs are connected together and thus are also called a single input J-K flip flop.
- ❖ When clock pulse is given to the flip flop, the output begins to toggle.
- ❖ Here also the restriction on the pulse width can be eliminated with a master-slave or edge-triggered construction. Take a look at the circuit and truth table below.



Excitation Table for T Flip Flop:

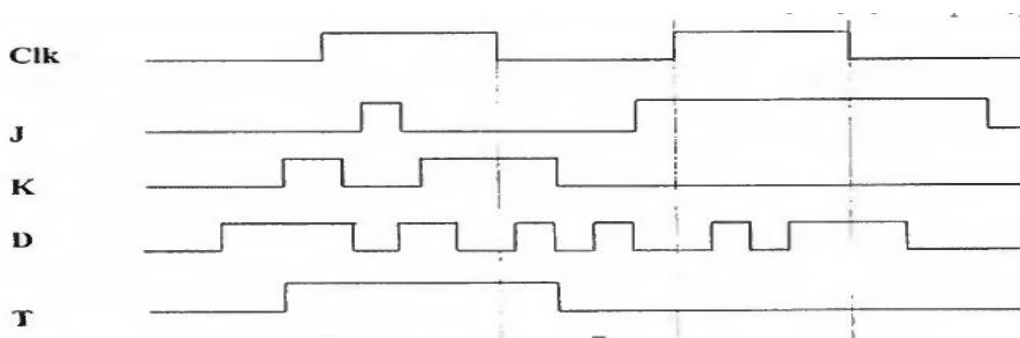
Q_n	T	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

K map for T Flip Flop:



Characteristic Equation:

$$Q_{n+1} = T \cdot \overline{Q_n} + \overline{T} \cdot Q_n$$

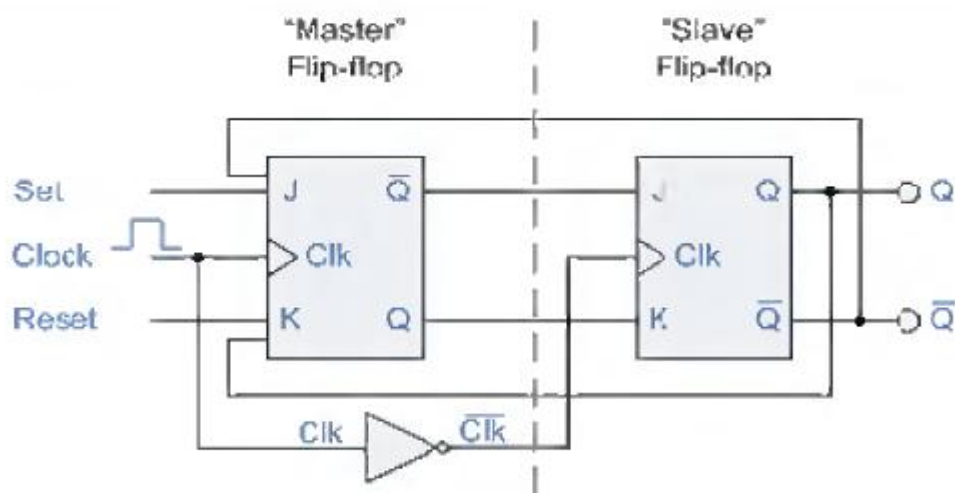
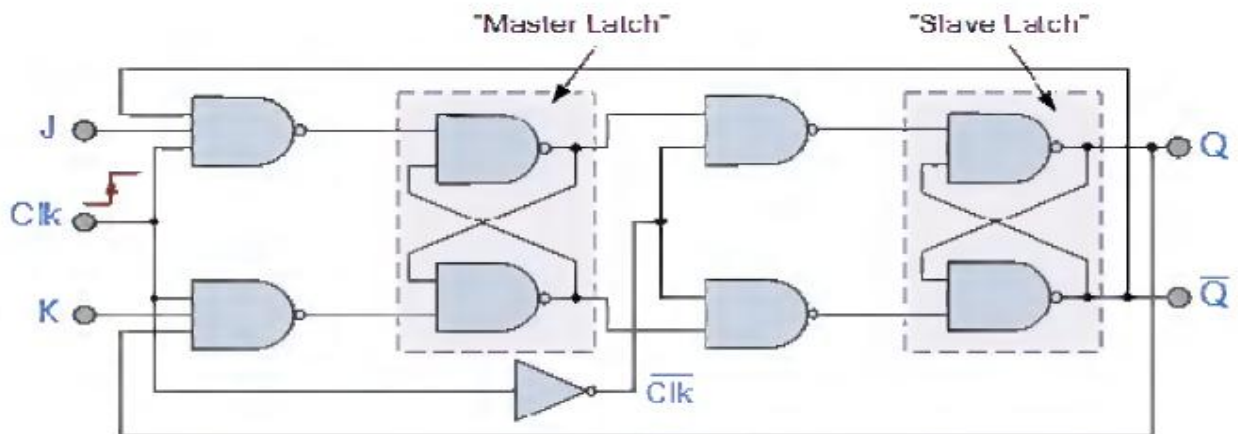


3.3 Master Slave flip flops

Before knowing more about the master-slave flip flop you have to know more on the basics of a J-K flip flop and S-R flip flop. To know more about the flip flops, click on the link below.

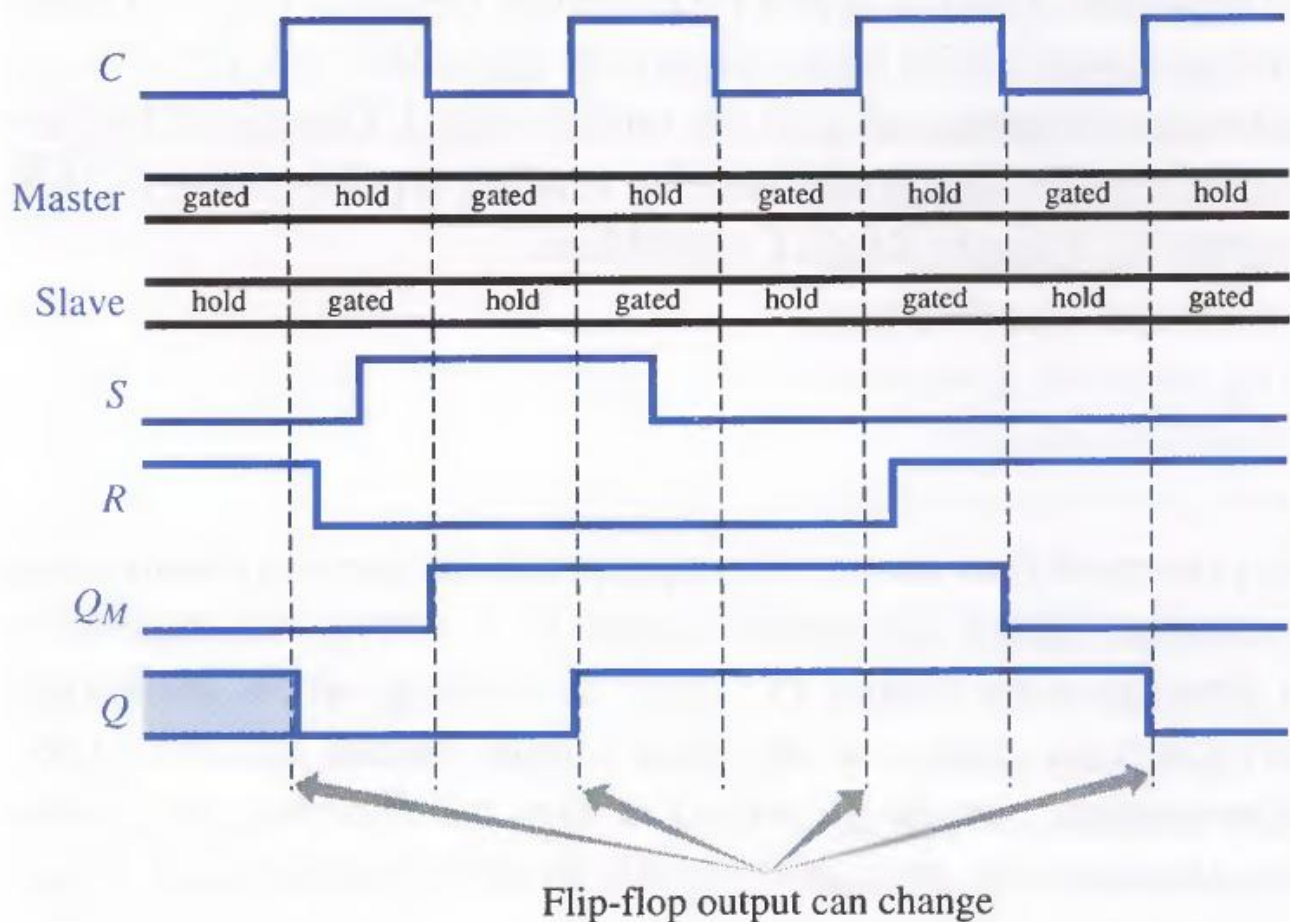
Master-slave flip flop is designed using two separate flip flops. Out of these, one acts as the master and the other as a slave. The figure of a master-slave J-K flip flop is shown below.

From the below figure you can see that both the J-K flip flops are presented in a series connection. The output of the master J-K flip flop is fed to the input of the slave J-K flip flop. The output of the slave J-K flip flop is given as a feedback to the input of the master J-K flip flop. The clock pulse [Clk] is given to the master J-K flip flop and it is sent through a NOT Gate and thus inverted before passing it to the slave J-K flip flop.



Working

When Clock=1, the master J-K flip flop gets disabled. The Clock input of the master input will be the opposite of the slave input. So the master flip flop output will be recognized by the slave flip flop only when the Clock value becomes 0. Thus, when the clock pulse makes a transition from 1 to 0, the locked outputs of the master flip flop are fed through to the inputs of the slave flip-flop making this flip flop edge or pulse-triggered. To understand better take a look at the timing diagram illustrated below.



Thus, the circuit accepts the value in the input when the clock is HIGH, and passes the data to the output on the falling-edge of the clock signal. This makes the Master-Slave J-K flip flop a Synchronous device as it only passes data with the timing of the clock signal.

Triggering of Flip Flops

The output of a flip flop can be changed by bring a small change in the input signal. This small change can be brought with the help of a clock pulse or commonly known as a trigger pulse.

Triggering of Flip Flops

The output of a flip flop can be changed by bring a small change in the input signal. This small change can be brought with the help of a clock pulse or commonly known as a trigger pulse.

When such a trigger pulse is applied to the input, the output changes and thus the flip flop is said to be triggered. Flip flops are applicable in designing counters or registers which stores data in the form of multi-bit numbers. But such registers need a group of flip flops connected to each other as sequential circuits. And these sequential circuits require trigger pulses.

The number of trigger pulses that is applied to the input of the circuit determines the number in a counter. A single pulse makes the bit move one position, when it is applied onto a register that stores multi-bit data.

In the case of SR Flip Flops, the change in signal level decides the type of trigger that is to be given to the input. But the original level must be regained before giving a second pulse to the circuit.

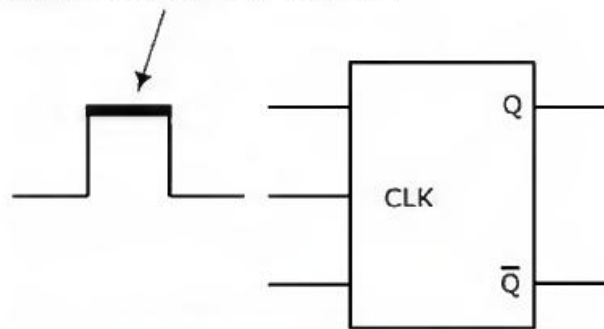
If a clock pulse is given to the input of the flip flop at the same time when the output of the flip flop is changing, it may cause instability to the circuit. The reason for this instability is the feedback that is given from the output combinational circuit to the memory elements. This problem can be solved to a certain level by making the flip flop more sensitive to the pulse transition rather than the pulse duration.

There are mainly four types of pulse-triggering methods. They differ in the manner in which the electronic circuits respond to the pulse. They are

1. High Level Triggering

When a flip flop is required to respond at its HIGH state, a HIGH level triggering method is used. It is mainly identified from the straight lead from the clock input. Take a look at the symbolic representation shown below.

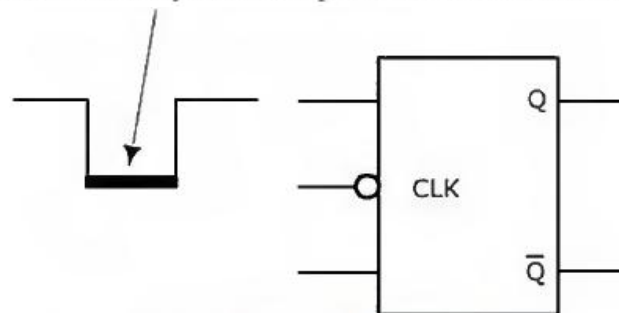
Triggers on high clock level



High Level Triggering

4. Low Level Triggering

When a flip flop is required to respond at its LOW state, a LOW level triggering method is used.. It is mainly identified from the clock input lead along with a low state indicator bubble. Take a look at the symbolic representation shown below.

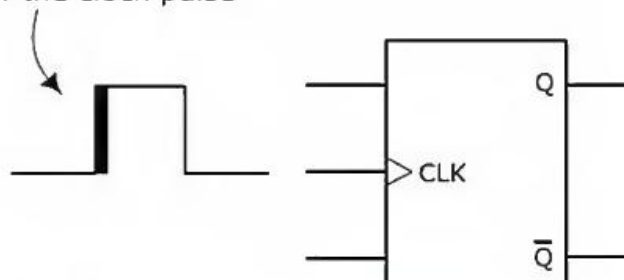


Low Level Triggering

3. Positive Edge Triggering

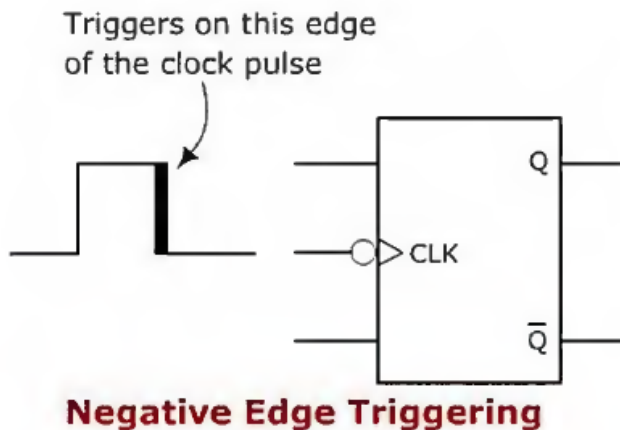
When a flip flop is required to respond at a LOW to HIGH transition state, POSITIVE edge triggering method is used. It is mainly identified from the clock input lead along with a triangle. Take a look at the symbolic representation shown below.

Triggers on this edge
of the clock pulse



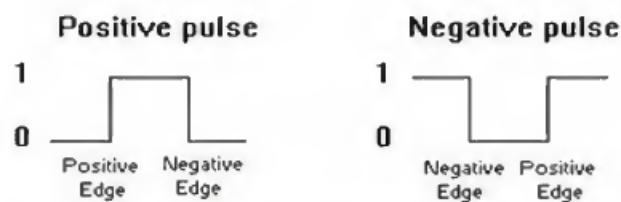
4. Negative Edge Triggering

When a flip flop is required to respond during the HIGH to LOW transition state, a NEGATIVE edge triggering method is used. It is mainly identified from the clock input lead along with a low-state indicator and a triangle. Take a look at the symbolic representation shown below.



Clock Pulse Transition

The movement of a trigger pulse is always from a 0 to 1 and then 1 to 0 of a signal. Thus it takes two transitions in a single signal. When it moves from 0 to 1 it is called a positive transition and when it moves from 1 to 0 it is called a negative transition. To understand more take a look at the images below.

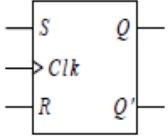
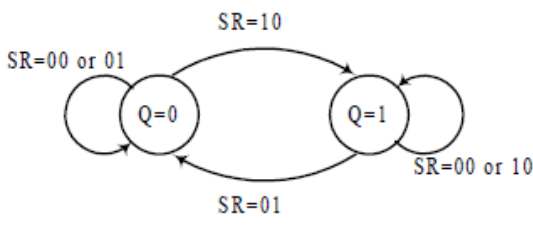
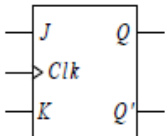
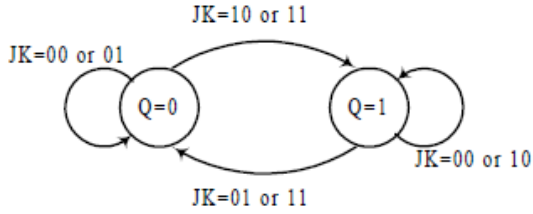
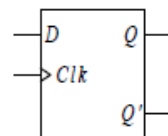
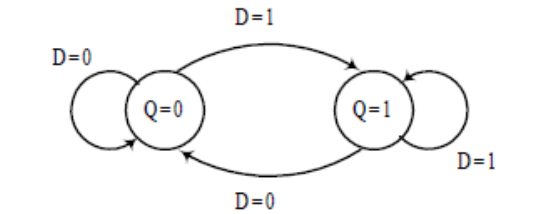
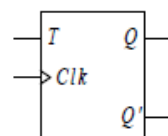
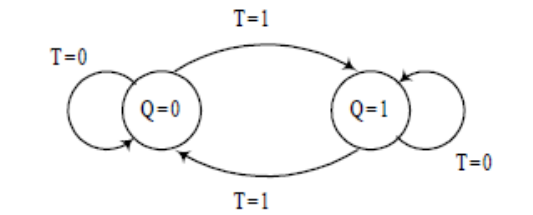


Definition of clock pulse transition

The clocked flip-flops already introduced are triggered during the 0 to 1 transition of the pulse, and the state transition starts as soon as the pulse reaches the HIGH 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 then the multiple-transition problem can be eliminated.

The multi-transition problem can be stopped is the flip flop is made to respond to the positive or negative edge transition only, other than responding to the entire pulse duration.

3.4 Characteristic and excitation table

Name / Symbol	Characteristic (Truth) Table	State Diagram / Characteristic Equations	Excitation Table																																																								
<p>SR</p> 	<table border="1"> <thead> <tr> <th>S</th> <th>R</th> <th>Q</th> <th>Q_{next}</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td><td>0</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>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>×</td></tr> <tr><td>1</td><td>1</td><td>1</td><td>×</td></tr> </tbody> </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	×	1	1	1	×	 <p> $Q_{next} = S + R'Q$ $SR = 0$ </p>	<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>×</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>×</td><td>0</td></tr> </tbody> </table>	Q	Q _{next}	S	R	0	0	0	×	0	1	1	0	1	0	0	1	1	1	×	0
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	×																																																								
1	1	1	×																																																								
Q	Q _{next}	S	R																																																								
0	0	0	×																																																								
0	1	1	0																																																								
1	0	0	1																																																								
1	1	×	0																																																								
<p>JK</p> 	<table border="1"> <thead> <tr> <th>J</th> <th>K</th> <th>Q</th> <th>Q_{next}</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td><td>0</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>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td><td>0</td></tr> </tbody> </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	 <p> $Q_{next} = J'K'Q + JK' + JKQ'$ $= J'K'Q + JK'Q + JK'Q' + JKQ'$ $= K'Q(J'+J) + JQ'(K'+K)$ $= K'Q + JQ'$ </p>	<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>×</td></tr> <tr><td>0</td><td>1</td><td>1</td><td>×</td></tr> <tr><td>1</td><td>0</td><td>×</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>×</td><td>0</td></tr> </tbody> </table>	Q	Q _{next}	J	K	0	0	0	×	0	1	1	×	1	0	×	1	1	1	×	0
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																																																								
Q	Q _{next}	J	K																																																								
0	0	0	×																																																								
0	1	1	×																																																								
1	0	×	1																																																								
1	1	×	0																																																								
<p>D</p> 	<table border="1"> <thead> <tr> <th>D</th> <th>Q</th> <th>Q_{next}</th> </tr> </thead> <tbody> <tr><td>0</td><td>×</td><td>0</td></tr> <tr><td>1</td><td>×</td><td>1</td></tr> </tbody> </table>	D	Q	Q _{next}	0	×	0	1	×	1	 <p> $Q_{next} = D$ </p>	<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	Q _{next}																																																									
0	×	0																																																									
1	×	1																																																									
Q	Q _{next}	D																																																									
0	0	0																																																									
0	1	1																																																									
1	0	0																																																									
1	1	1																																																									
<p>T</p> 	<table border="1"> <thead> <tr> <th>T</th> <th>Q</th> <th>Q_{next}</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>	T	Q	Q _{next}	0	0	0	0	1	1	1	0	1	1	1	0	 <p> $Q_{next} = TQ' + T'Q = T \oplus Q$ </p>	<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	Q _{next}																																																									
0	0	0																																																									
0	1	1																																																									
1	0	1																																																									
1	1	0																																																									
Q	Q _{next}	T																																																									
0	0	0																																																									
0	1	1																																																									
1	0	1																																																									
1	1	0																																																									

3.5 Realization of one flip flop with other flip flops

For the conversion of one flip flop to another, a combinational circuit has to be designed first. If a JK Flip Flop is required, the inputs are given to the combinational circuit and the output of the combinational circuit is connected to the inputs of the actual flip flop. Thus, the output of the actual flip flop is the output of the required flip flop. The following flip flop conversions will be explained.

- **SR Flip Flop to JK Flip Flop**
- **JK Flip Flop to SR Flip Flop**
- **SR Flip Flop to D Flip Flop**
- **D Flip Flop to SR Flip Flop**
- **JK Flip Flop to T Flip Flop**
- **JK Flip Flop to D Flip Flop**
- **D Flip Flop to JK Flip Flop**

SR Flip Flop to JK Flip Flop

As told earlier, J and K will be given as external inputs to S and R. As shown in the logic diagram below, S and R will be the outputs of the combinational circuit.

The truth tables for the flip flop conversion are given below. The present state is represented by Q_p and Q_{p+1} is the next state to be obtained when the J and K inputs are applied.

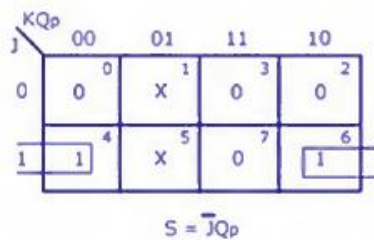
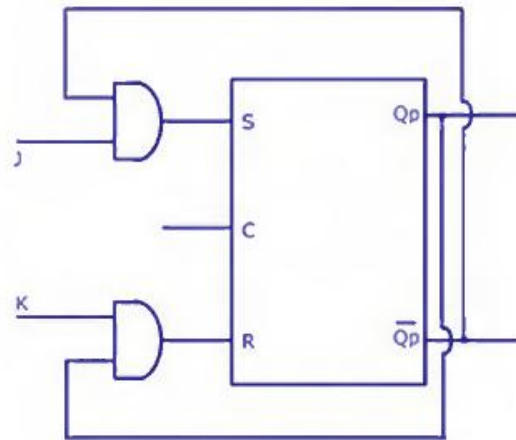
For two inputs J and K, there will be eight possible combinations. For each combination of J, K and Q_p , the corresponding Q_{p+1} states are found. Q_{p+1} simply suggests the future values to be obtained by the JK flip flop after the value of Q_p . The table is then completed by writing the values of S and R required getting each Q_{p+1} from the corresponding Q_p . That is, the values of S and R that are required to change the state of the flip flop from Q_p to Q_{p+1} are written.

S-R Flip Flop to J-K Flip Flop

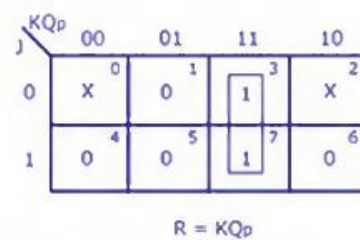
Conversion Table

J-K Inputs		Outputs		S-R Inputs	
J	K	Q_p	Q_{p+1}	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

Logic Diagram



K-Map



$$R = KQ_p$$

JK Flip Flop to SR Flip Flop

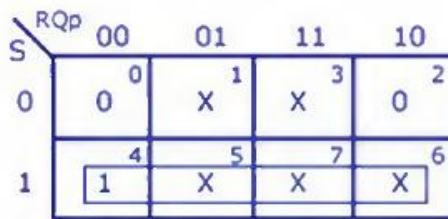
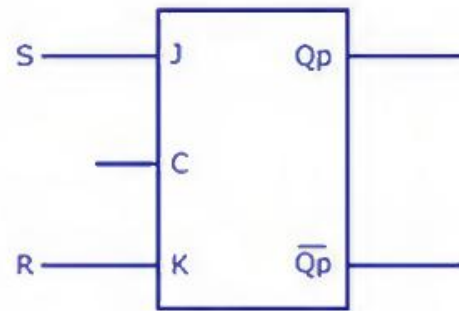
This will be the reverse process of the above explained conversion. S and R will be the external inputs to J and K. As shown in the logic diagram below, J and K will be the outputs of the combinational circuit. Thus, the values of J and K have to be obtained in terms of S, R and Q_p . The logic diagram is shown below.

A conversion table is to be written using S, R, Q_p , Q_{p+1} , J and K. For two inputs, S and R, eight combinations are made. For each combination, the corresponding Q_{p+1} outputs are found out. The outputs for the combinations of $S=1$ and $R=1$ are not permitted for an SR flip flop. Thus the outputs are considered invalid and the J and K values are taken as “don’t cares”.

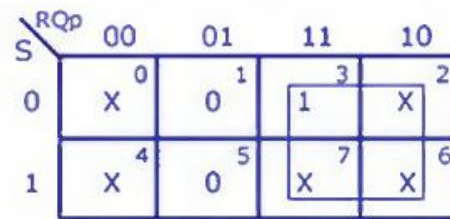
Conversion Table

S-R Inputs		Outputs		J-K Inputs	
S	R	Q _p	Q _{p+1}	J	K
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	X	1
1	0	0	1	1	X
1	0	1	1	X	0
1	1	Invalid		Dont care	
1	1	Invalid		Dont care	

Logic Diagram



J=S



K-maps

K=R

SR Flip Flop to D Flip Flop

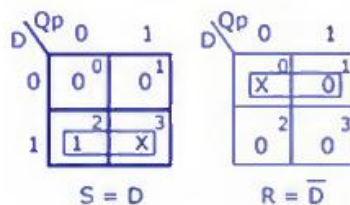
As shown in the figure, S and R are the actual inputs of the flip flop and D is the external input of the flip flop. The four combinations, the logic diagram, conversion table, and the K-map for S and R in terms of D and Q_p are shown below.

S-R Flip Flop to D Flip Flop

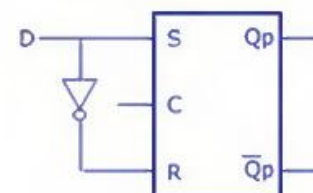
Conversion Table

D Input	Outputs		S-R Inputs	
	Q _p	Q _{p+1}	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

K-maps



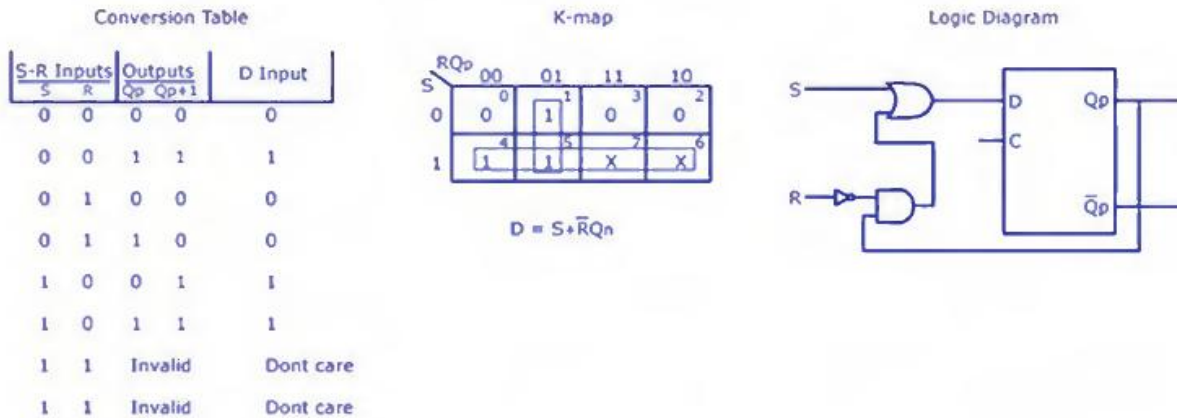
Logic Diagram



D Flip Flop to SR Flip Flop

D is the actual input of the flip flop and S and R are the external inputs. Eight possible combinations are achieved from the external inputs S, R and Q_p . But, since the combination of $S=1$ and $R=1$ are invalid, the values of Q_{p+1} and D are considered as “don’t cares”. The logic diagram showing the conversion from D to SR, and the K-map for D in terms of S, R and Q_p are shown below.

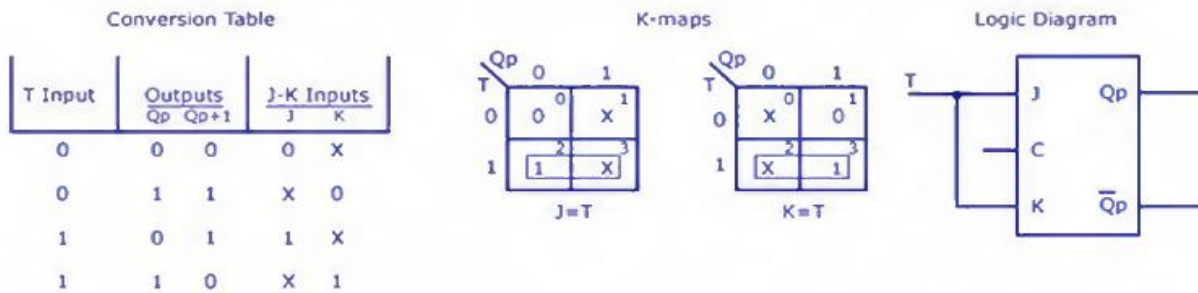
D Flip Flop to S-R Flip Flop



JK Flip Flop to T Flip Flop

J and K are the actual inputs of the flip flop and T is taken as the external input for conversion. Four combinations are produced with T and Q_p . J and K are expressed in terms of T and Q_p . The conversion table, K-maps, and the logic diagram are given below.

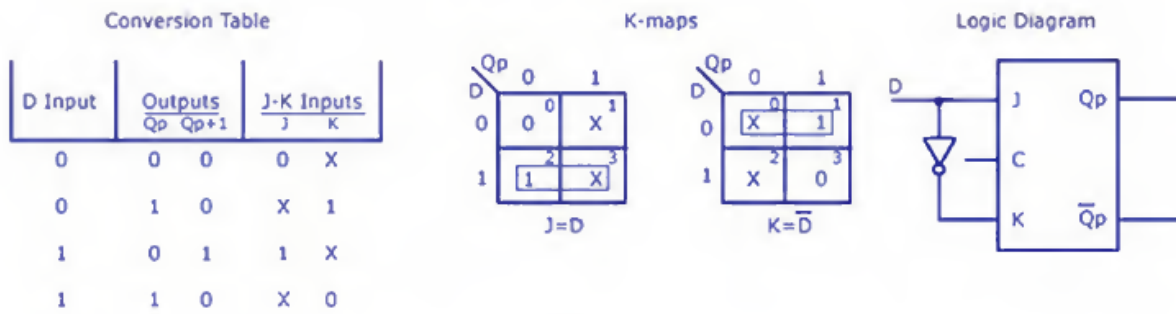
J-K Flip Flop to T Flip Flop



JK Flip Flop to D Flip Flop

D is the external input and J and K are the actual inputs of the flip flop. D and Q_p make four combinations. J and K are expressed in terms of D and Q_p . The four combination conversion table, the K-maps for J and K in terms of D and Q_p , and the logic diagram showing the conversion from JK to D are given below.

J-K Flip Flop to D Flip Flop

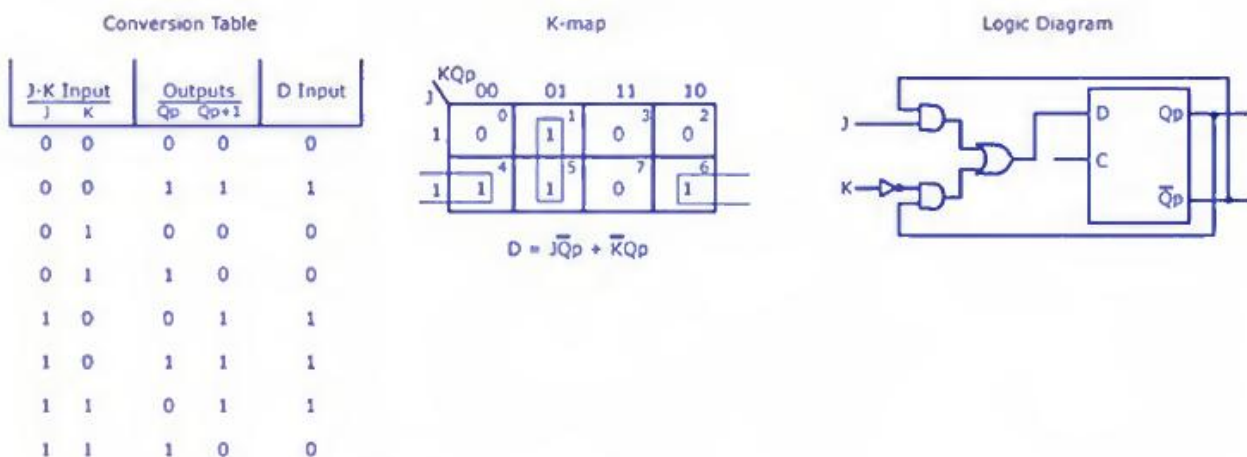


▪ D Flip Flop to JK Flip Flop

In this conversion, D is the actual input to the flip flop and J and K are the external inputs. J, K and Q_p make eight possible combinations, as shown in the conversion table below. D is expressed in terms of J, K and Q_p.

The conversion table, the K-map for D in terms of J, K and Q_p and the logic diagram showing the conversion from D to JK are given in the figure below.

D Flip Flop to J-K Flip Flop



3.6 Registers

Flip-flop is a 1 bit memory cell which can be used for storing the digital data. To increase the storage capacity in terms of number of bits, we have to use a group of flip-flop. Such a group of flip-flop is known as a Register. The n-bit register will consist of n number of flip-flop and it is capable of storing an n-bit word.

3.6.1 Shift registers

Introduction:

A group of flip-flops connected together forms a **register**. A register is used solely for storing and shifting data which is in the form of 1's and 0's entered from an external source. The Binary information in a register can be moved from stage to stage within the register or into or out of the register upon application of clock pulses. This type of bit movement or shifting essential for certain arithmetic and logic operations used in microprocessors. This gives rise to a group of registers called Shift Registers. They are very important in applications involving the storage and transfer of data in a digital system.

Shift Registers:

Shift Registers consists of a number of single bit "D-Type Data Latches" connected together in a chain arrangement so that the output from one data latch becomes the input of the next latch and so on, thereby moving the stored data serially from either the left or the right direction. The number of individual Data Latches used to make up **Shift Registers** are determined by the number of bits to be stored with the most common being 8-bits wide. Shift Registers are mainly used to store data and to convert data from either a serial to parallel or parallel to serial format with all the latches being driven by a common clock (Clk) signal making them Synchronous devices. They are generally provided with a Clear or Reset connection so that they can be "SET" or "RESET" as required.

Generally, Shift Registers operate in one of four different modes:

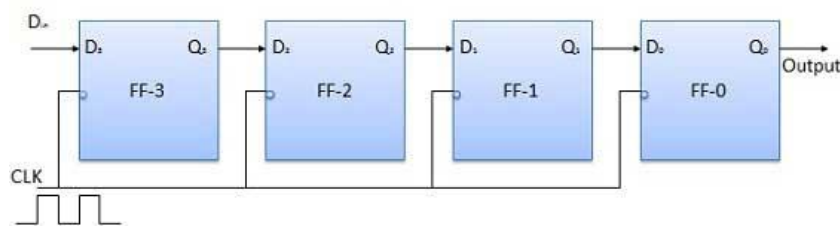
Generally, Shift Registers operate in one of four different modes:

- Serial-in to Parallel-out (SIPO)
- Serial-in to Serial-out (SISO)
- Parallel-in to Parallel-out (PIPO)
- Parallel-in to Serial-out (PISO)

Serial Input Serial Output

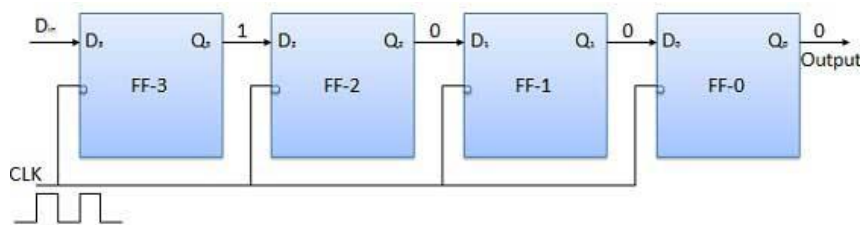
Let all the flip-flop be initially in the reset condition i.e. $Q_3 = Q_2 = Q_1 = Q_0 = 0$. If an entry of a four bit binary number 1 1 1 1 is made into the register, this number should be applied to Din bit with the LSB bit applied first. The D input of FF-3 i.e. D_3 is connected to serial data input Din. Output of FF-3 i.e. Q_3 is connected to the input of the next flip-flop i.e. D_2 and so on.

Block Diagram

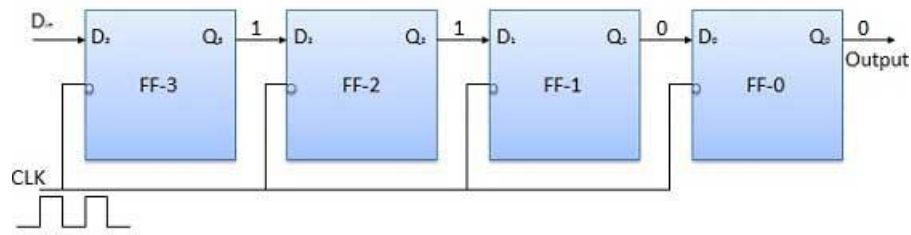


Operation

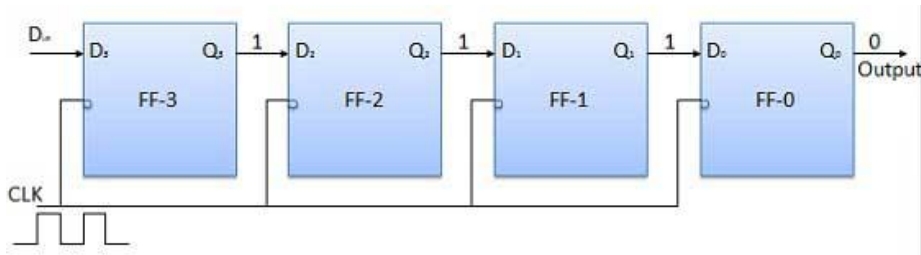
Before application of clock signal, let $Q_3 Q_2 Q_1 Q_0 = 0000$ and apply LSB bit of the number to be entered to Din. So $D_{in} = D_3 = 1$. Apply the clock. On the first falling edge of clock, the FF-3 is set, and stored word in the register is $Q_3 Q_2 Q_1 Q_0 = 1000$.



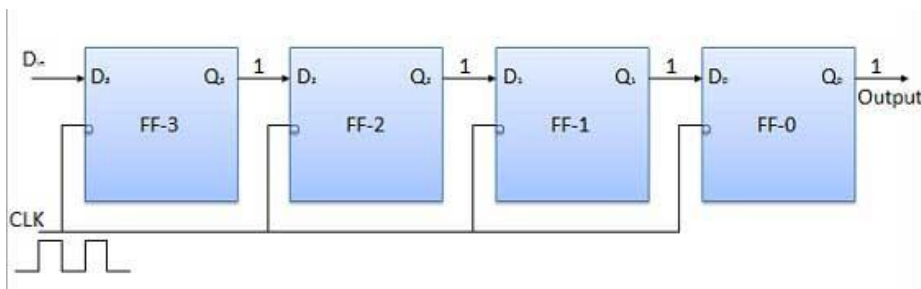
Apply the next bit to Din. So $D_{in} = 1$. As soon as the next negative edge of the clock hits, FF-2 will set and the stored word change to $Q_3 Q_2 Q_1 Q_0 = 1100$.



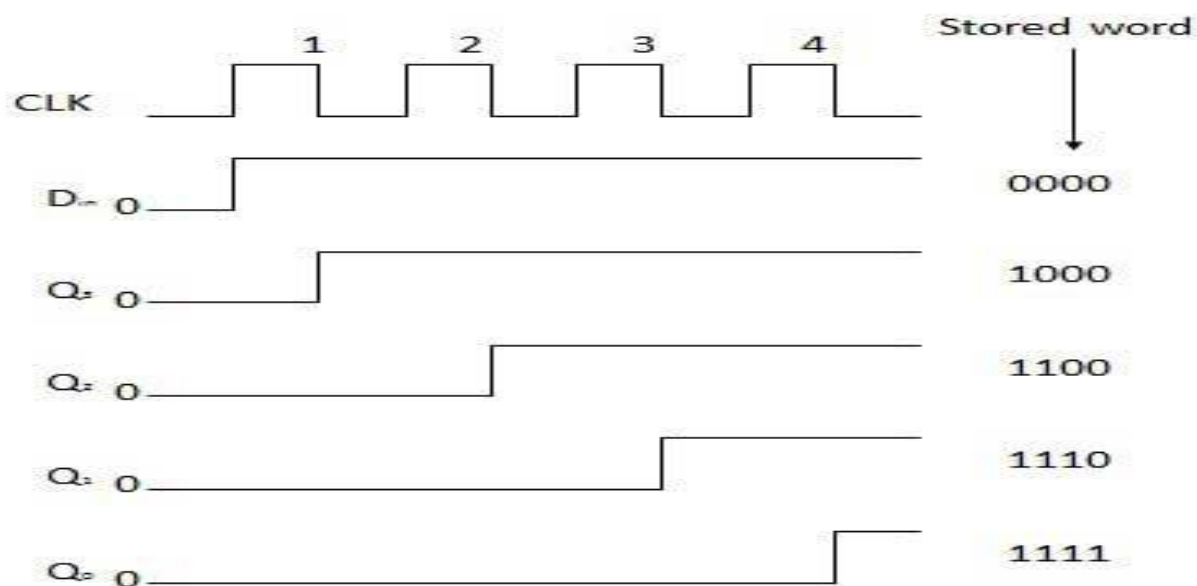
Apply the next bit to be stored i.e. 1 to Din. Apply the clock pulse. As soon as the third negative clock edge hits, FF-1 will be set and output will be modified to Q3 Q2 Q1 Q0 = 1110.



Similarly with Din = 1 and with the fourth negative clock edge arriving, the stored word in the register is Q3 Q2 Q1 Q0 = 1111.



Waveforms



Serial Input Parallel Output

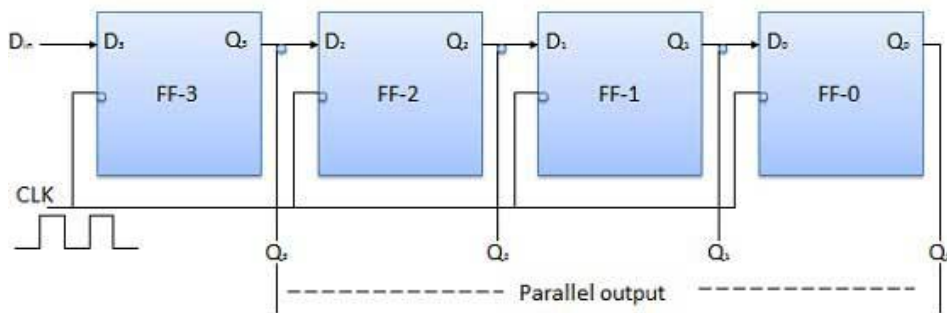
In such types of operations, the data is entered serially and taken out in parallel fashion.

Data is loaded bit by bit. The outputs are disabled as long as the data is loading.

As soon as the data loading gets completed, all the flip-flops contain their required data, the outputs are enabled so that all the loaded data is made available over all the output lines at the same time.

4 clock cycles are required to load a four bit word. Hence the speed of operation of SIPO mode is same as that of SISO mode.

Block Diagram



Parallel Input Serial Output (PISO)

Data bits are entered in parallel fashion.

The circuit shown below is a four bit parallel input serial output register.

Output of previous Flip Flop is connected to the input of the next one via a combinational circuit.

The binary input word B0, B1, B2, B3 is applied through the same combinational circuit.

There are two modes in which this circuit can work namely - shift mode or load mode.

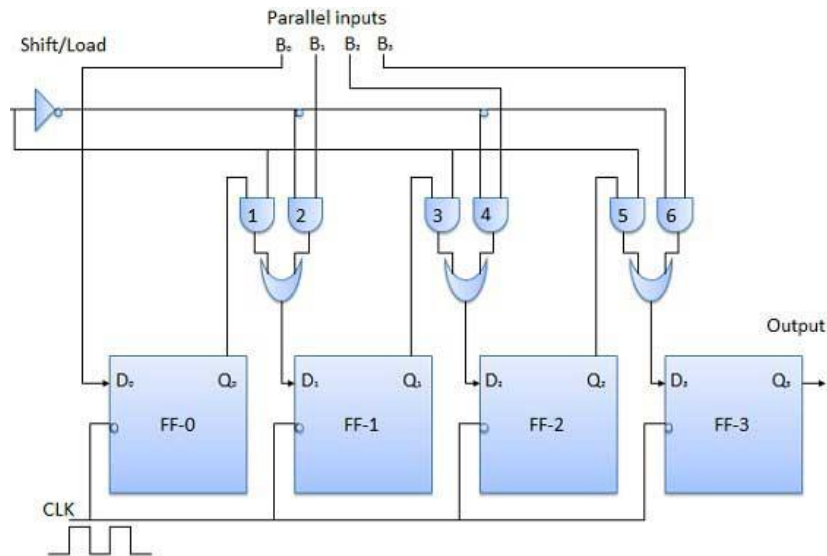
Load mode

When the shift/load bar line is low (0), the AND gate 2, 4 and 6 become active they will pass B1, B2, B3 bits to the corresponding flip-flops. On the low going edge of clock, the binary input B0, B1, B2, B3 will get loaded into the corresponding flip-flops. Thus parallel loading takes place.

Shift mode

When the shift/load bar line is high (1), the AND gate 2, 4 and 6 become inactive. Hence the parallel loading of the data becomes impossible. But the AND gate 1,3 and 5 become active. Therefore the shifting of data from left to right bit by bit on application of clock pulses. Thus the parallel in serial out operation takes place.

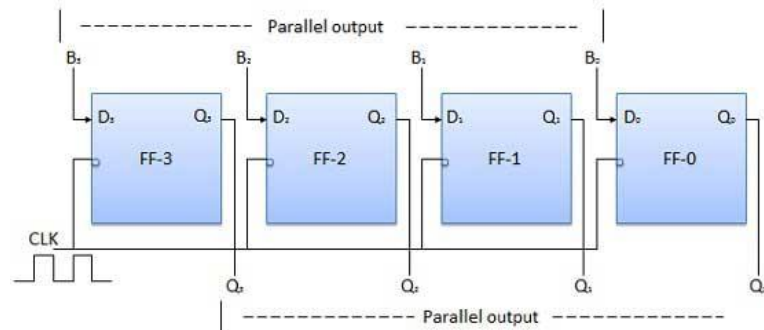
Block Diagram



Parallel Input Parallel Output (PIPO)

In this mode, the 4 bit binary input B_0, B_1, B_2, B_3 is applied to the data inputs D_0, D_1, D_2, D_3 respectively of the four flip-flops. As soon as a negative clock edge is applied, the input binary bits will be loaded into the flip-flops simultaneously. The loaded bits will appear simultaneously to the output side. Only clock pulse is essential to load all the bits.

Block Diagram



Counter is a sequential circuit. A digital circuit which is used for counting pulses is known counter. Counter is the widest application of flip-flops. It is a group of flip-flops with a clock signal applied. Counters are of two types.

- Asynchronous or ripple counters.
- Synchronous counters.

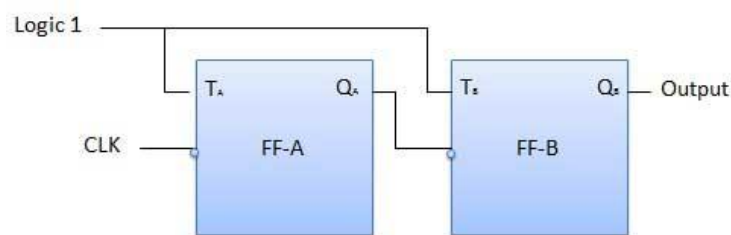
Asynchronous or ripple counters

RING COUNTER (shift register sequencer):

- The ring counter (shift register sequencer) is a unique type of shift register that incorporates feedback from the output of the last flip-flop to the input of the first flip-flop.
- Figure shows a 4-bit ring counter made from D-type flip-flops.
- In this circuit, when the LOAD' input is set low, Q_0 is forced high by the active-low preset, while Q_1 , Q_2 , and Q_3 are forced low (cleared) by the active-low clear.
- This causes the binary word 1000 to be stored within the register.
- When the LOAD line is brought low, the data bits stored in the flip-flops are shifted right with each positive clock edge.
- The data bit from the last flip-flop is sent to the D input of the first flip-flop. The shifting cycle will continue to recirculate while the clock is applied.
- To start fresh cycle, the LOAD line is momentarily brought low.

The logic diagram of a 2-bit ripple up counter is shown in figure. The toggle (T) flip-flop is being used. But we can use the JK flip-flop also with J and K connected permanently to logic 1. External clock is applied to the clock input of flip-flop A and Q_A output is applied to the clock input of the next flip-flop i.e. FF-B.

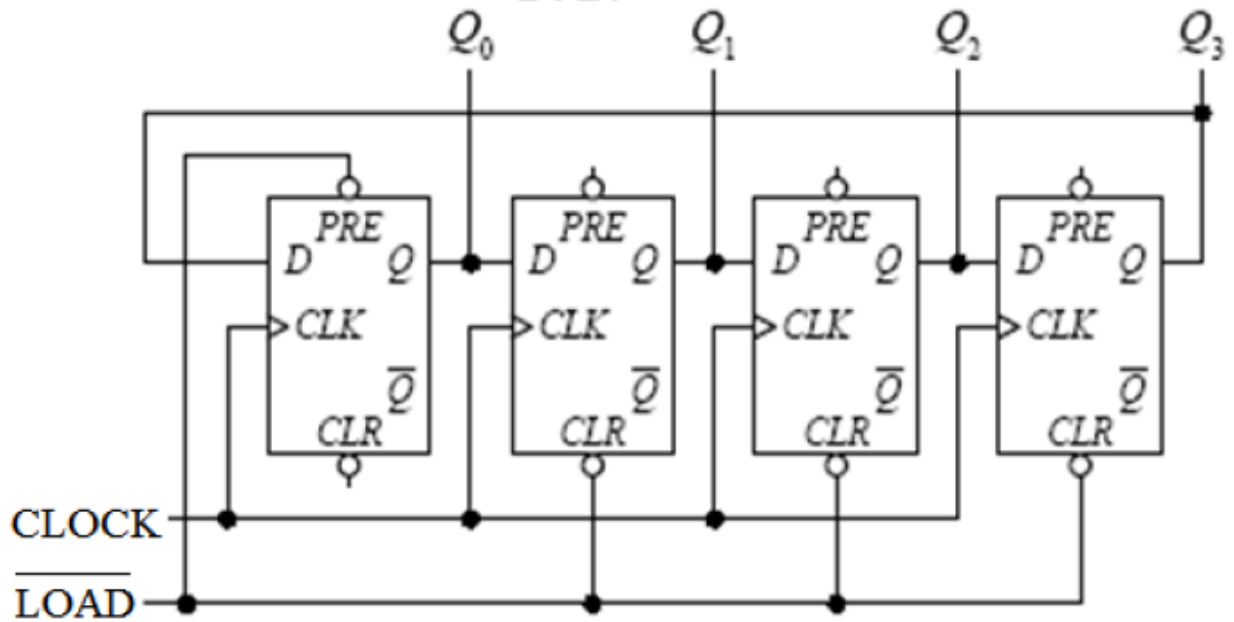
Logical Diagram



Initially let both the FFs be in the reset state, $QBQA = 00$ initially. After 1st negative clock edge, As soon as the first negative clock edge is applied, FF-A will toggle and QA will be equal to 1. QA is connected to clock input of FF-B. Since QA has changed from 0 to 1, it is treated as the positive clock edge by FF-B. There is no change in QB because FF-B is a negative edge triggered FF. $QBQA = 01$ after the first clock pulse. After 2nd negative clock edge On the arrival of second negative clock edge, FF-A toggles again and $QA = 0$. The change in QA acts as a negative clock edge for FF-B. So it will also toggle, and QB will be 1. $QBQA = 10$ after the second clock pulse. After 3rd negative clock edge, On the arrival of 3rd negative clock edge, FF-A toggles again and QA become 1 from 0. Since this is a positive going change, FF-B does not respond to it and remains inactive. So QB does not change and continues to be equal to 1. $QBQA = 11$ after the third clock pulse. After 4th negative clock edge, On the arrival of 4th negative

clock edge, FF-A toggles again and QA becomes 1 from 0. This negative change in QA acts as clock pulse for FF-B. Hence it toggles to change QB from 1 to 0. QBQA = 00 after the fourth clock pulse.

Ring counter using positive edge-triggered D flip-flops



Truth Table

Clock	Counter output		State number	Decimal Counter output
	Q ₁	Q ₀		
Initially	0	0	—	0
1st	0	1	1	1
2nd	1	0	2	2
3rd	1	1	3	3
4th	0	0	4	0

3.7 Counters

A **digital counter**, or simply counter, is a semiconductor device that is used for counting the number of times that a digital event has occurred. The counter's output is indexed by one LSB every time the counter is clocked.

A counter is a register capable of counting the number of clock pulses arriving at its clock input. Count represents the number of clock pulses arrived. A counter that follows a binary number is called binary counter.

There are two types of counters

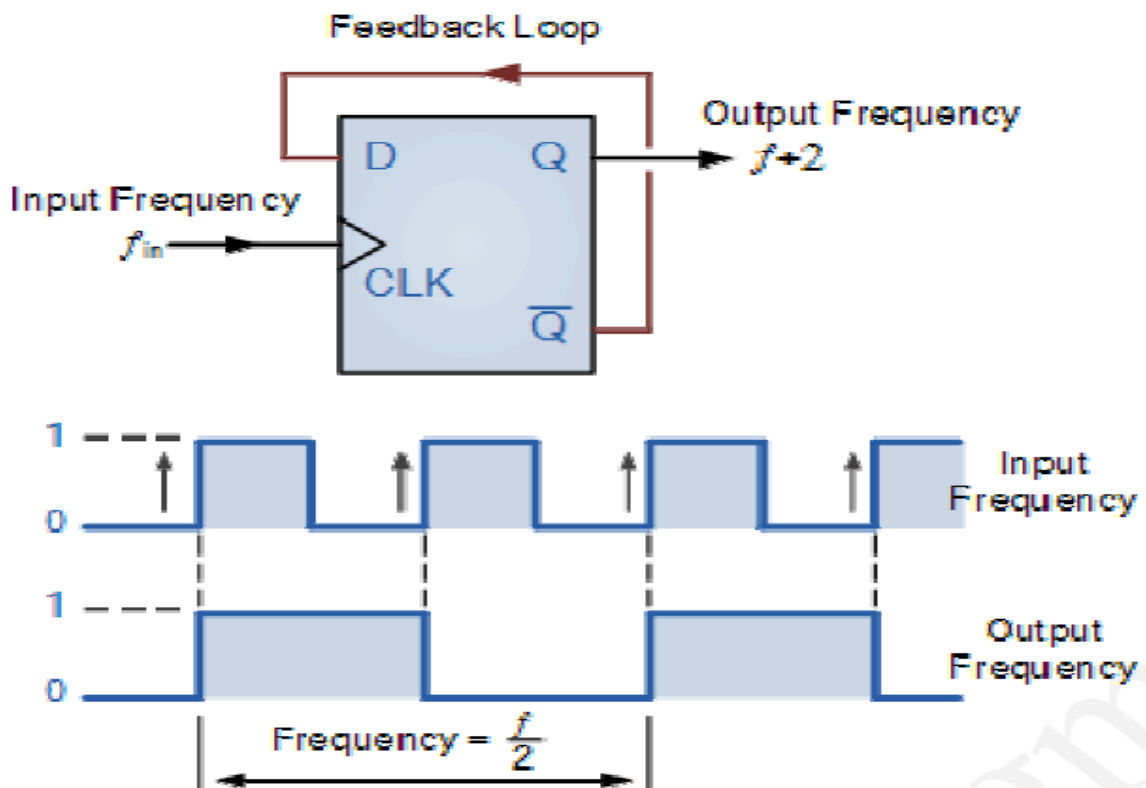
1. Asynchronous Counter
2. Synchronous Counter

In *Asynchronous counters*, (ripple counter) the first flip-flop is clocked by the external clock pulse and then each successive flip-flop is clocked by the output of the preceding flip-flop. In *Synchronous counters*, the clock input is connected to all of the flip-flop so that they are clocked simultaneously.

Frequency Division

First, let us discuss about principle behind the counters. In the **Sequential Logic** tutorials we discussed how D-type Flip-Flop's work and how they can be connected together to form a Data Latch. Another useful feature of the D-type Flip-Flop is as a binary divider, for **Frequency Division** or as a "divide-by-2" counter. Here the inverted output terminal Q (NOT-Q) is connected directly back to the Data input terminal D giving the device "feedback" as shown below.

Divide-by-2 Counter



It can be seen from the frequency waveforms above, that by "feeding back" the output from Q to the input terminal D, the output pulses at Q have a frequency that are exactly one half ($f/2$) that of the input clock frequency. In other words the circuit produces **Frequency Division** as it now divides the input frequency by a factor of two (an octave). This then produces a type of counter called a "ripple counter" and in ripple counters, the clock pulse triggers the first flip-flop whose output triggers the second flip-flop, which in turn triggers the third flip-flop and so on through the chain.

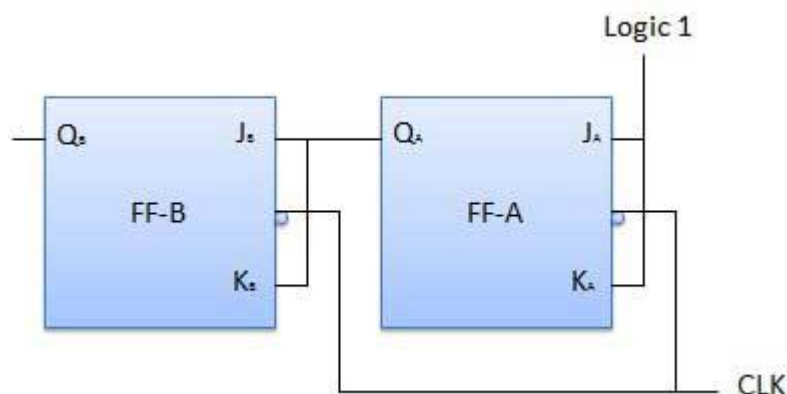
3.7.1 Synchronous and Asynchronous counters

If the "clock" pulses are applied to all the flip-flops in a counter simultaneously, then such a counter is called as synchronous counter.

2-bit Synchronous up counter

The JA and KA inputs of FF-A are tied to logic 1. So FF-A will work as a toggle flip-flop. The JB and KB inputs are connected to QA.

Logical Diagram



Initially let both the FFs be in the reset state, QBQA = 00 initially. After 1st negative clock edge, As soon as the first negative clock edge is applied, FF-A will toggle and QA will change from 0 to 1. But at the instant of application of negative clock edge, QA, JB = KB = 0. Hence FF-B will not change its state. So QB will remain 0. QBQA = 01 after the first clock pulse. After 2nd negative clock edge, on the arrival of second negative clock edge, FF-A toggles again and QA changes from 1 to 0. But at this instant QA was 1. So JB = KB = 1 and FF-B will toggle. Hence QB changes from 0 to 1. QBQA = 10 after the second clock pulse. After 3rd negative clock edge, On application of the third falling clock edge, FF-A will toggle from 0 to 1 but there is no change of state for FF-B. QBQA = 11 after the third clock pulse. After 4th negative clock edge, On application of the next clock pulse, QA will change from 1 to 0 as QB will also change from 1 to 0. QBQA = 00 after the fourth clock pulse

3.7.1.1 Modulus counters

A counter is nothing more than a specialized register or pattern generator that produces a specified output pattern or sequence of binary values (or states) upon the application of an input pulse signal called the "Clock". The clock is actually used for data transfer in these applications. Typically, counters are logic circuits that can increment or decrement a count by one but when used as asynchronous divide-by-n counters they are able to divide these input pulses producing a clock division signal.

Counters are formed by connecting flip-flops together and any number of flip-flops can be connected or "cascaded" together to form a "divide-by-n" binary counter where "n" is the number of counter stages used and which is called the **Modulus**. The modulus or simply "MOD" of a counter is the number of output states the counter goes through before returning itself back to zero, i.e., one complete cycle.

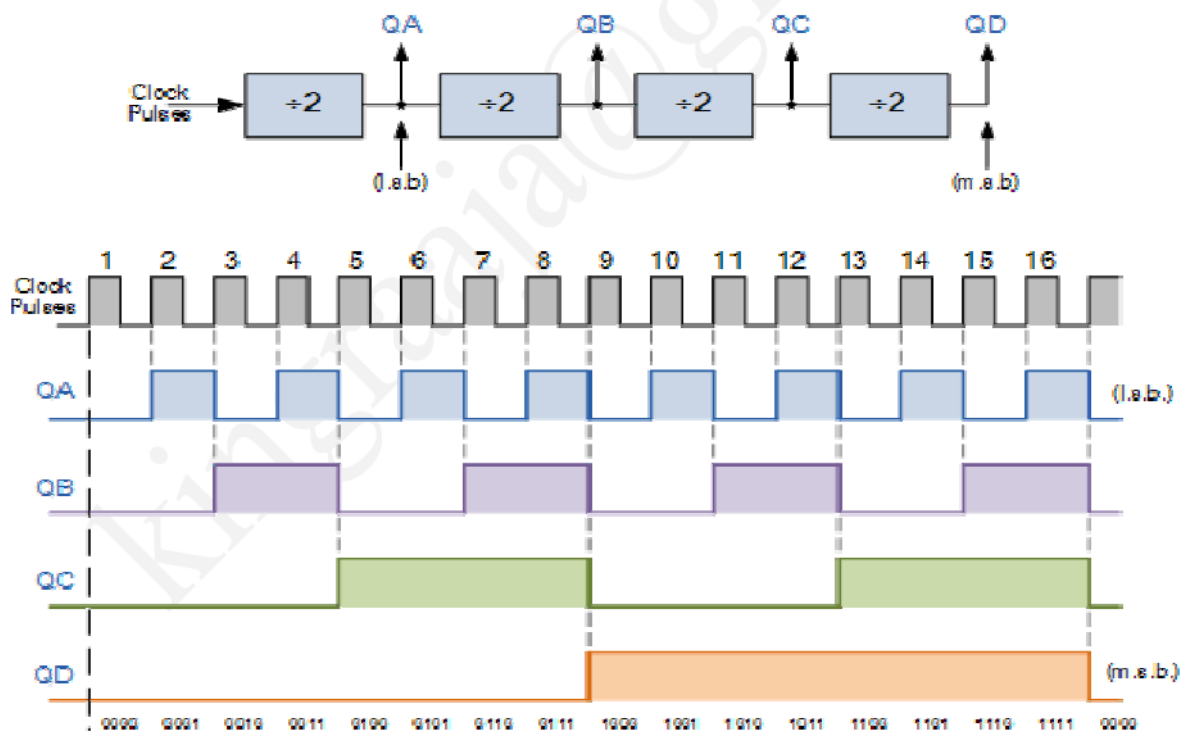
Then a counter with three flip-flops like the circuit above will count from 0 to 7 i.e., 2^3-1 . It has eight different output states representing the decimal numbers 0 to 7 and is called a **Modulo-8** or **MOD-8** counter. A counter with four flip-flops will count from 0 to 15 and is therefore called a **Modulo-16** counter and so on.

An example of this is given as.

- 3-bit Binary Counter = $2^3 = 8$ (modulo-8 or MOD-8)
- 4-bit Binary Counter = $2^4 = 16$ (modulo-16 or MOD-16)
- 8-bit Binary Counter = $2^8 = 256$ (modulo-256 or MOD-256)

The Modulo number can be increased by adding more flip-flops to the counter and cascading is a method of achieving higher modulus counters. Then the modulo or MOD number can simply be written as: MOD number = 2^n

4-bit Modulo-16 Counter



Multi-bit asynchronous counters connected in this manner are also called "**Ripple Counters**" or ripple dividers because the change of state at each stage appears to "ripple" itself through the counter from the LSB output to its MSB output connection.

Frequency Division Summary

For **frequency division**, toggle mode flip-flops are used in a chain as a divide by two counter. One flip-flop will divide the clock, f_{in} by 2, two flip-flops will divide f_{in} by 4 (and so on). One benefit of using toggle flip-flops for frequency division is that the output at any point has an exact 50% duty cycle.

The final output clock signal will have a frequency value equal to the input clock frequency divided by the MOD number of the counter. Such circuits are known as "divide-by-n" counters. Counters can be formed by connecting individual flip-flops together and are classified according to the way they are clocked.

In *Asynchronous counters*, (ripple counter) the first flip-flop is clocked by the external clock pulse and then each successive flip-flop is clocked by the output of the preceding flip-flop. In *Synchronous counters*, the clock input is connected to all of the flip-flop so that they are clocked simultaneously.

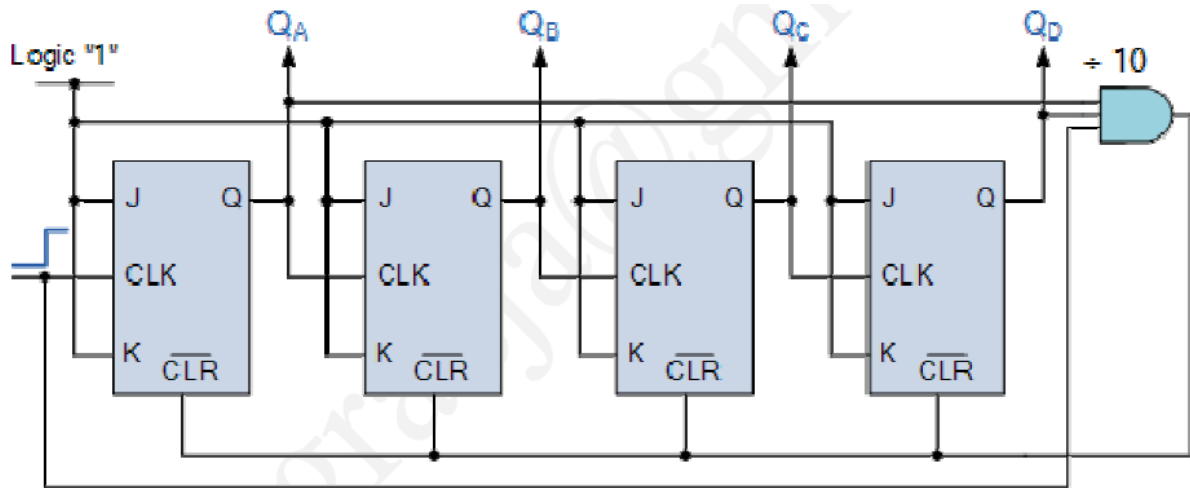
Asynchronous Counter

In the previous tutorial we discussed that an **Asynchronous counter** can have $2^n - 1$ possible counting states e.g. MOD-16 for a 4-bit counter, (0-15) making it ideal for use in **Frequency Division**. But it is also possible to use the basic asynchronous counter to construct special counters with counting states less than their maximum output number by forcing the counter to reset itself to zero at a pre-determined value producing a type of asynchronous counter that has truncated sequences. Then an n-bit counter that counts up to its maximum modulus (2^n) is called a full sequence counter and a n-bit counter whose modulus is less than the maximum possible is called a **truncated counter**.

If we take the modulo-16 asynchronous counter and modified it with additional logic gates it can be made to give a decade (divide-by-10) counter output for use in standard decimal counting and arithmetic circuits.

Such counters are generally referred to as **Decade Counters**. A decade counter requires resetting to zero when the output count reaches the decimal value of 10, i.e. when DCBA = 1010 and to do this we need to feed this condition back to the reset input. A counter with a count sequence from binary "0000" (BCD = "0") through to "1001" (BCD = "9") is generally referred to as a BCD binary-coded-decimal counter because its ten state sequence is that of a BCD code but binary decade counters are more common.

Asynchronous Decade Counter (mod-10)



This type of asynchronous counter counts upwards on each leading edge of the input clock signal starting from "0000" until it reaches an output "1010" (decimal 10). Both outputs Q_B and Q_D are now equal to logic "1" and the output from the NAND gate changes state from logic "1" to a logic "0" level and whose output is also connected to the CLEAR (CLR) inputs of all the J-K Flip-flops.

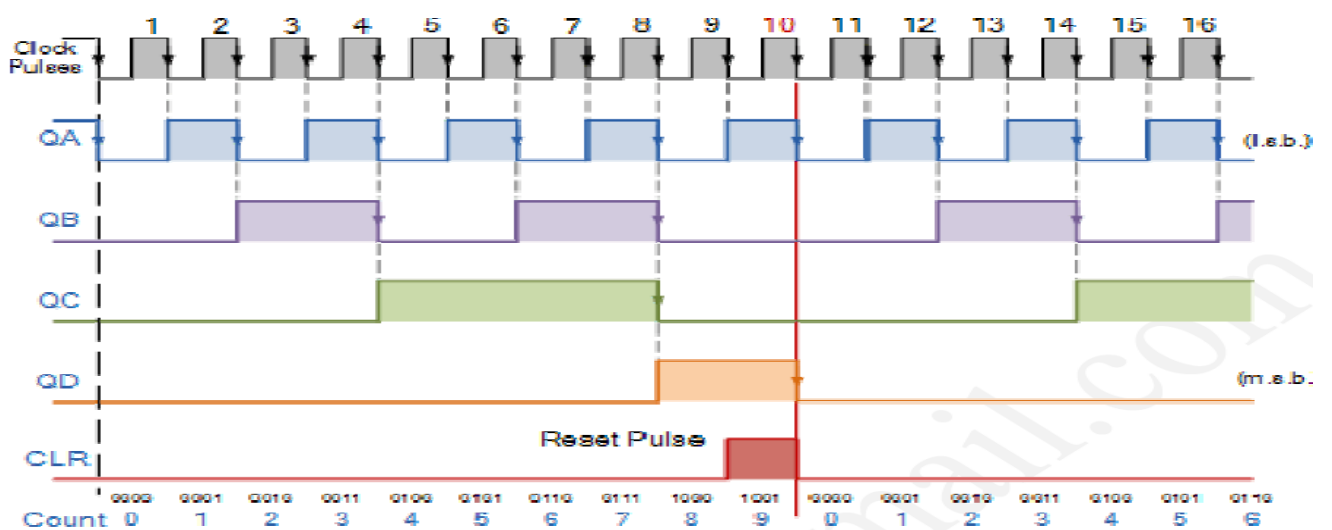
This signal causes all of the Q outputs to be reset back to binary "0000" on the count of 10. Once Q_B and Q_D are both equal to logic "0" the output of the NAND gate

returns back to a logic level "1" and the counter restarts again from "0000". We now have a decade or **Modulo-10** counter.

Decade Counter Truth Table

Clock Count	Output bit Pattern				Decimal Value
	Q _D	Q _C	Q _B	Q _A	
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	1	0	2
4	0	0	1	1	3
5	0	1	0	0	4
6	0	1	0	1	5
7	0	1	1	0	6
8	0	1	1	1	7
9	1	0	0	0	8
10	1	0	0	1	9
11	Counter Resets its Outputs back to Zero				

Decade Counter Timing Diagram



Using the same idea of truncating counter output sequences, the above circuit could easily be adapted to other counting cycles by simply changing the connections to the AND gate. For example, a scale-of-twelve (modulo-12) can easily be made by simply taking the inputs to the AND gate from the outputs at "Q_C" and "Q_D", noting that the binary equivalent of 12 is "1100" and that output "Q_A" is the least significant bit (LSB).

Since the maximum modulus that can be implemented with n flip-flops is 2^n , this means that when you are designing truncated asynchronous counters you should determine the lowest power of two that is greater than or equal to your desired modulus. For example, if you wish to count from 0 to 39, or mod-40. Then the highest number of flip-flops required would be six, $n = 6$ giving a maximum MOD of 64 as five flip-flops would only equal MOD-32.

Unfortunately one of the main disadvantages with asynchronous counters is that there is a small delay between the arrival of the clock pulse at its input and it being present at its output due to the internal circuitry of the gate. In asynchronous circuits this delay is called the **Propagation Delay** giving the asynchronous ripple counter the nickname of "**propagation counter**" and in some high frequency cases this delay can produce false output counts.

In large bit ripple counter circuits, if the delay of the separate stages are all added together to give a summed delay at the end of the counter chain the difference in time between the input signal and the counted output signal can be very large. This is why the **Asynchronous Counter** is generally not used in high frequency counting circuits where large numbers of bits are involved.

Also, the outputs from the counter do not have a fixed time relationship with each other and do not occur at the same instant in time due to their clocking sequence. In other words the output frequencies become available one by one, a sort of domino effect. Then,

the more flip-flops that are added to an asynchronous counter chain the lower the maximum operating frequency becomes to ensure accurate counting. To overcome the problem of propagation delay Synchronous Counters were developed.

Summary:

- **Asynchronous Counters** can be made from Toggle or D-type flip-flops.
- They are called asynchronous counters because the clock input of the flip-flops are not all driven by the same clock signal.
- Each output in the chain depends on a change in state from the previous flip-flops output.
- Asynchronous counters are sometimes called ripple counters because the data appears to "ripple" from the output of one flip-flop to the input of the next.

Disadvantages of Asynchronous Counters:

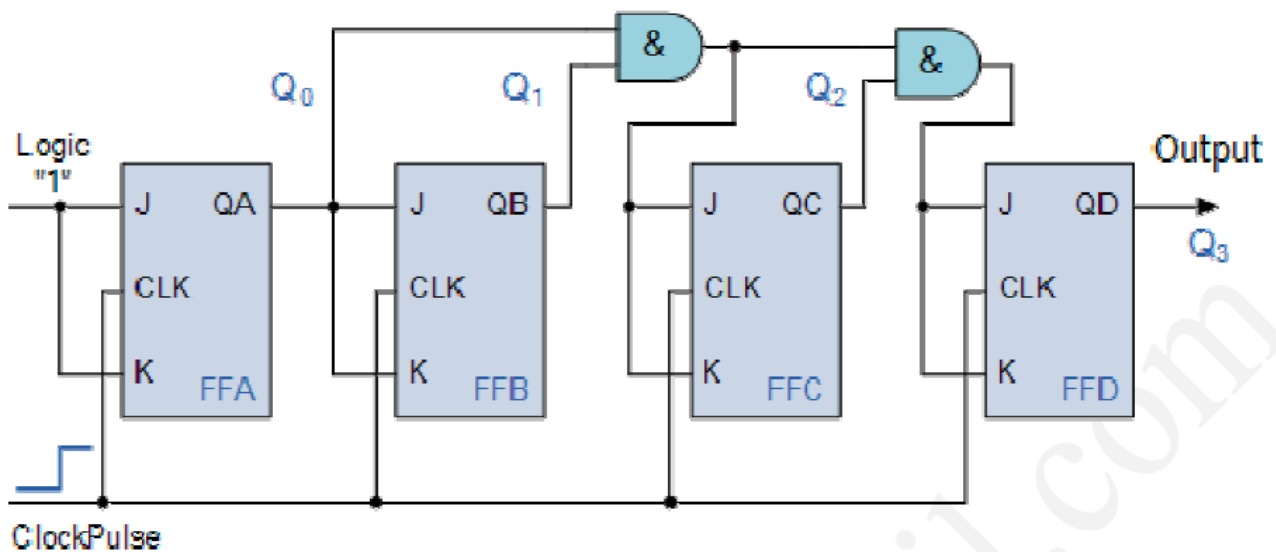
- An extra "re-synchronizing" output flip-flop may be required.
- To count a truncated sequence not equal to 2^n , extra feedback logic is required.
- Counting a large number of bits, propagation delay by successive stages may become undesirably large.
- This delay gives them the nickname of "Propagation Counters".
- Counting errors at high clocking frequencies.
- Synchronous Counters are faster using the same clock signal for all flip-flops.

Synchronous Counter

In the previous Asynchronous binary counter tutorial, we discussed that the output of one counter stage is connected directly to the clock input of the next counter stage and so on along the chain, and as a result the asynchronous counter suffers from what is known as "Propagation Delay" in which the timing signal is delayed a fraction through each flip-flop.

However, with the **Synchronous Counter**, the external clock signal is connected to the clock input of EVERY individual flip-flop within the counter so that all of the flip-flops are clocked together simultaneously (in parallel) at the same time giving a fixed time relationship. In other words, changes in the output occur in "synchronization" with the clock signal. This results in all the individual output bits changing state at exactly the same time in response to the common clock signal with no ripple effect and therefore, no

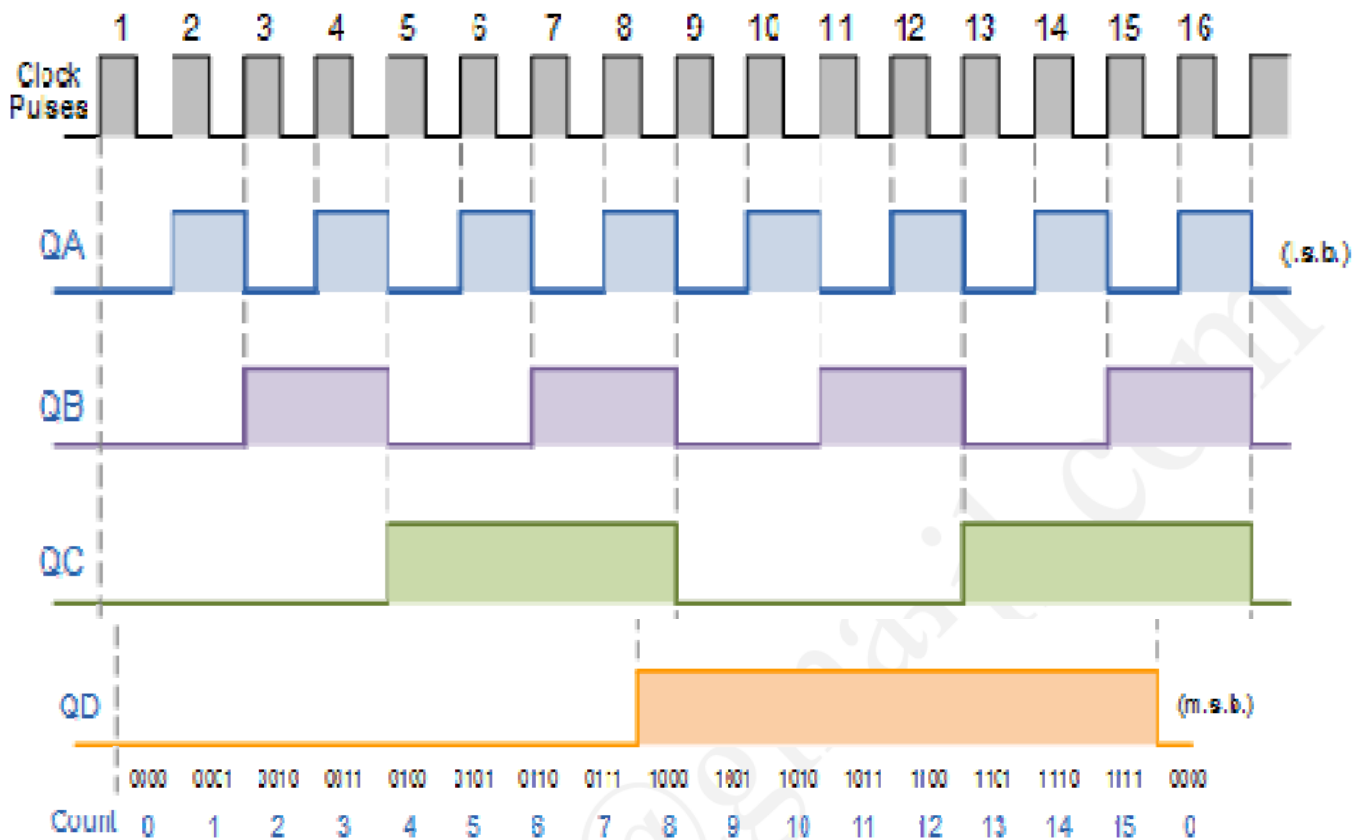
Binary 4-bit Synchronous Counter



It can be seen that the external clock pulses (pulses to be counted) are fed directly to each **J-K flip-flop** in the counter chain and that both the J and K inputs are all tied together in toggle mode, but only in the first flip-flop, flip-flop A (LSB) are they connected HIGH, logic "1" allowing the flip-flop to toggle on every clock pulse. Then the synchronous counter follows a predetermined sequence of states in response to the common clock signal, advancing one state for each pulse.

The J and K inputs of flip-flop B are connected to the output "Q" of flip-flop A, but the J and K inputs of flip-flops C and D are driven from AND gates which are also supplied with signals from the input and output of the previous stage. If we enable each J-K flip-flop to toggle based on whether or not all preceding flip-flop outputs (Q) are "HIGH" we can obtain the same counting sequence as with the asynchronous circuit but without the ripple effect, since each flip-flop in this circuit will be clocked at exactly the same time. As there is no propagation delay in synchronous counters because all the

4-bit Synchronous Counter Timing Diagram.



Because this 4-bit synchronous counter counts sequentially on every clock pulse the resulting outputs count upwards from 0 ("0000") to 15 ("1111"). Therefore, this type of counter is also known as a **4-bit Synchronous Up Counter**.

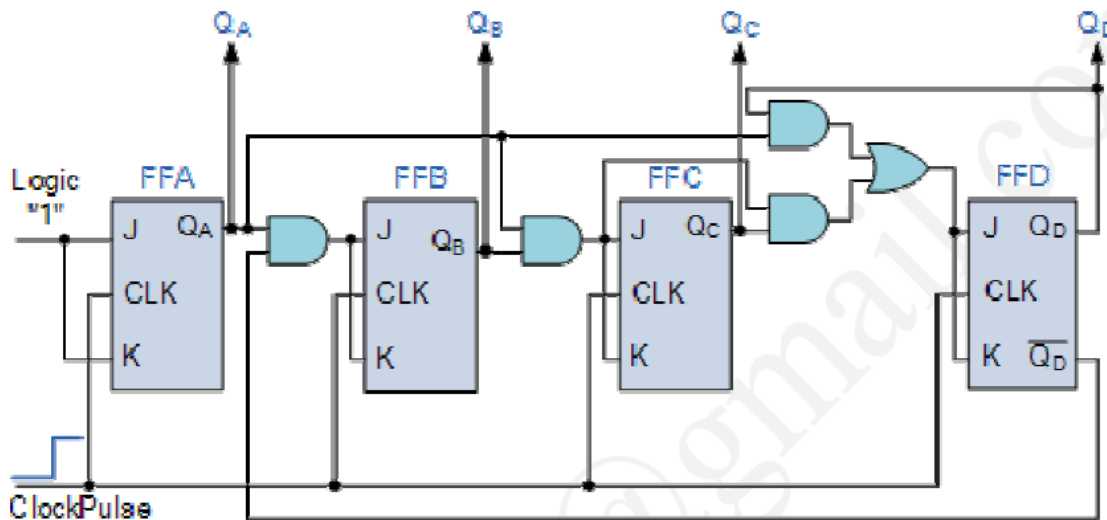
As synchronous counters are formed by connecting flip-flops together and any number of flip-flops can be connected or "cascaded" together to form a "divide-by-n" binary counter, the modulo's or "MOD" number still applies as it does for asynchronous counters so a Decade counter or BCD counter with counts from 0 to 2^n-1 can be built along with truncated sequences.

Decade 4-bit Synchronous Counter

Another name for Decade Counter is Modulo 10 counter. A 4-bit decade synchronous counter can also be built using synchronous binary counters to produce a

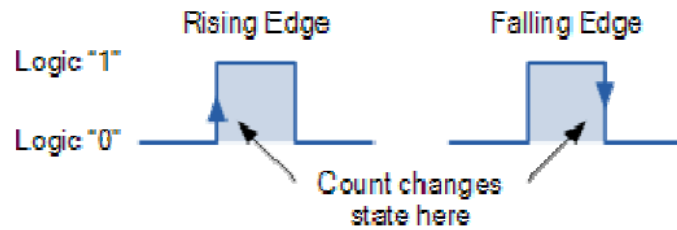
count sequence from 0 to 9. A standard binary counter can be converted to a decade (decimal 10) counter with the aid of some additional logic to implement the desired state sequence. After reaching the count of "1001", the counter recycles back to "0000". We now have a decade or **Modulo-10** counter.

Decade 4-bit Synchronous Counter



The additional AND gates detect when the sequence reaches "1001", (Binary 10) and causes flip-flop FF3 to toggle on the next clock pulse. Flip-flop FF0 toggles on every clock pulse. Thus, the count starts over at "0000" producing a synchronous decade counter. We could quite easily re-arrange the additional AND gates to produce other counters such as a Mod-12 Up counter which counts 12 states from "0000" to "1011" (0 to 11) and then repeats making them suitable for clocks.

Synchronous Counters use edge-triggered flip-flops that change states on either the "positive-edge" (rising edge) or the "negative-edge" (falling edge) of the clock pulse on the control input resulting in one single count when the clock input changes state. Generally, synchronous counters count on the rising-edge which is the low to high transition of the clock signal and asynchronous ripple counters count on the falling-edge which is the high to low transition of the clock signal.



It may seem unusual that ripple counters use the falling-edge of the clock cycle to change state, but this makes it easier to link counters together because the most significant bit (MSB) of one counter can drive the clock input of the next. This works because the next bit must change state when the previous bit changes from high to low - the point at which a carry must occur to the next bit. Synchronous counters usually have a carry-out and a carry-in pin for linking counters together without introducing any propagation delays.

Summary:

- **Synchronous Counters** can be made from Toggle or D-type flip-flops.
- They are called synchronous counters because the clock input of the flip-flops are clocked with the same clock signal.
- Due to the same clock pulse all outputs change simultaneously.
- Synchronous counters are also called parallel counters as the clock is fed in parallel to all flip-flops.
- Synchronous binary counters use both sequential and combinational logic elements.
- The memory section keeps track of the present state.
- The sequence of the count is controlled by combinational logic.

Advantages of Synchronous Counters:

- Synchronous counters are easier to design.

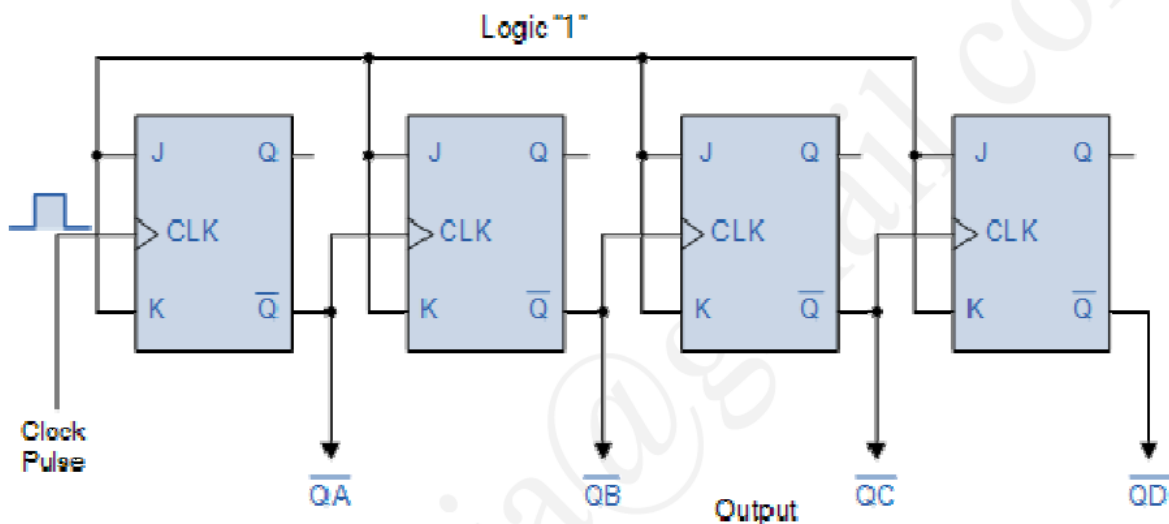
Advantages of Synchronous Counters:

- Synchronous counters are easier to design.
- With all clock inputs wired together there is no inherent propagation delay.
- Overall faster operation may be achieved compared to Asynchronous counters.

Count Down Counter

As well as counting "up" from zero and increase, or increment to some value, it is sometimes necessary to count "down" from a predetermined value to zero and to produce an output that activates when the zero count or other pre-set value is reached. This type of counter is normally referred to as a **Down Counter**, (CTD).

4-bit Count Down Counter

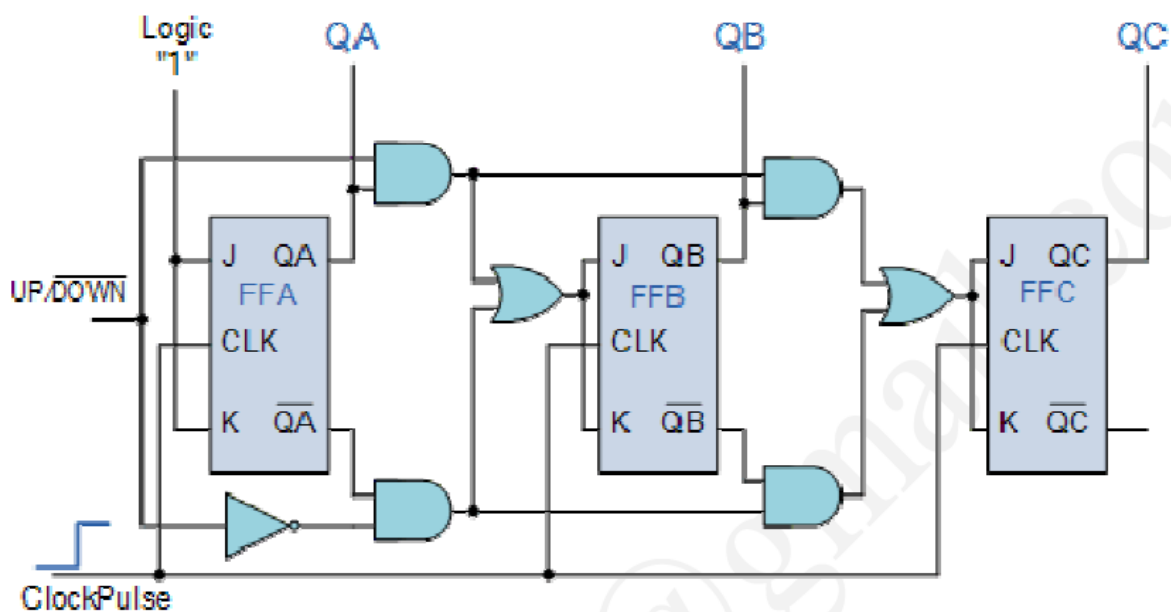


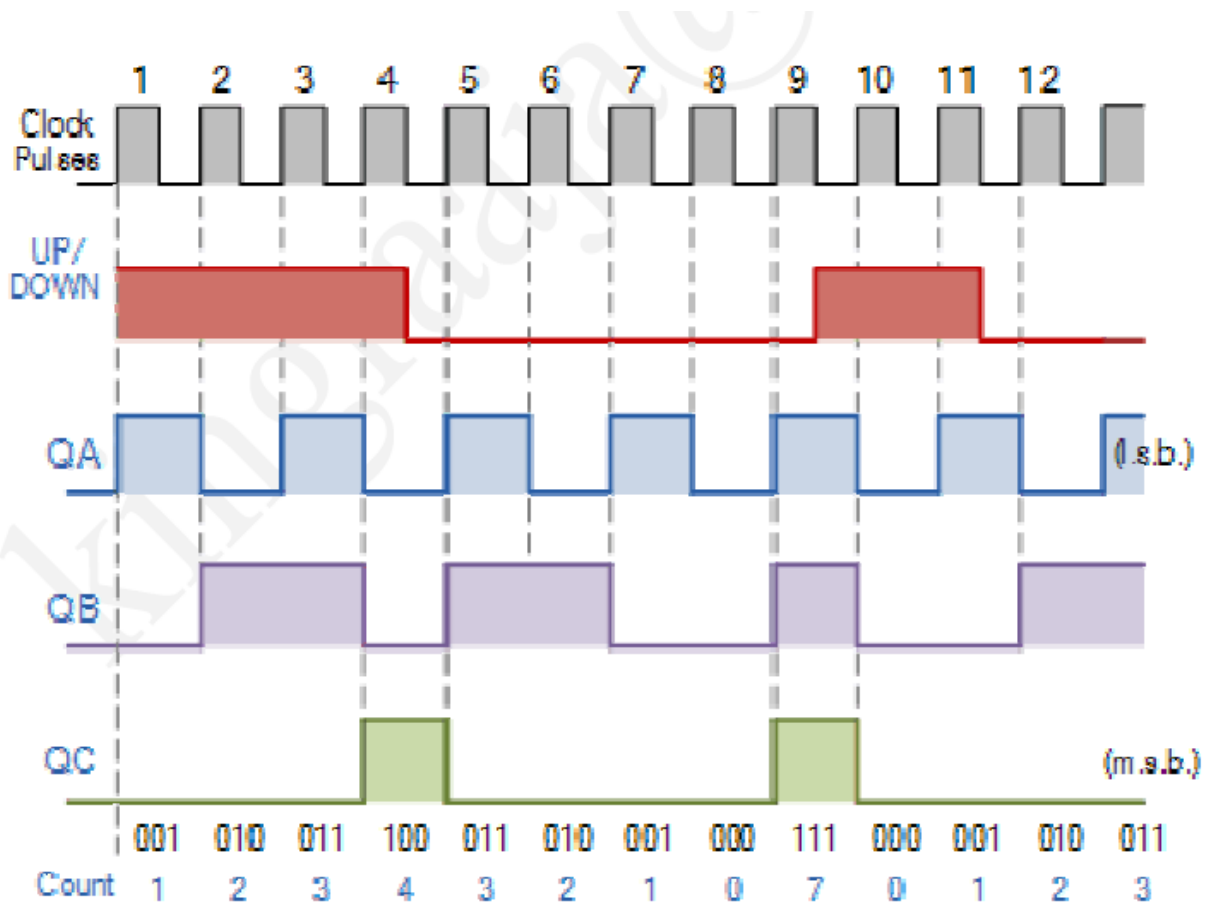
In the 4-bit counter above the output of each flip-flop changes state on the falling edge (1-to-0 transition) of the CLK input which is triggered by the Q output of the previous flip-flop, rather than by the Q output as in the up counter configuration. As a result, each flip-flop will change state when the previous one changes from 0 to 1 at its output,

Bidirectional Counter(up/down)

Both Synchronous and Asynchronous counters are capable of counting "Up" or counting "Down", but there is another more "Universal" type of counter that can count in both directions either Up or Down depending on the state of their input control pin and these are known as **Bidirectional Counters**. Bidirectional counters, also known as Up/Down counters, are capable of counting in either direction through any given count sequence and they can be reversed at any point within their count sequence by using an additional control input as shown below.

Synchronous 3-bit Up/Down Counter





The circuit above is of a simple 3-bit Up/Down synchronous counter using JK flip-flops configured to operate as toggle or T-type flip-flops giving a maximum count of zero (000) to seven (111) and back to zero again. Then the 3-Bit counter advances upward in sequence (0,1,2,3,4,5,6,7) or downwards in reverse sequence (7,6,5,4,3,2,1,0) but generally, bidirectional counters can be made to change their count direction at any point in the counting sequence. An additional input determines the direction of the count, either Up or Down and the timing diagram gives an example of the counters operation as this Up/Down input changes state.

Nowadays, both up and down counters are incorporated into single IC that is fully programmable to count in both an "Up" and a "Down" direction from any preset value producing a complete **Bidirectional Counter** chip.

3.7.1.2 Up/Down counters

UP/DOWN Counter

Up counter and down counter is combined together to obtain an UP/DOWN counter. A mode control (M) input is also provided to select either up or down mode. A combinational circuit is required to be designed and used between each pair of flip-flop in order to achieve the up/down operation.

- Type of up/down counters
- UP/DOWN ripple counters
- UP/DOWN synchronous counter

UP/DOWN Ripple Counters

In the UP/DOWN ripple counter all the FFs operate in the toggle mode. So either T flip-flops or JK flip-flops are to be used. The LSB flip-flop receives clock directly. But the clock to every other FF is obtained from ($Q = \bar{Q}$) output of the previous FF.

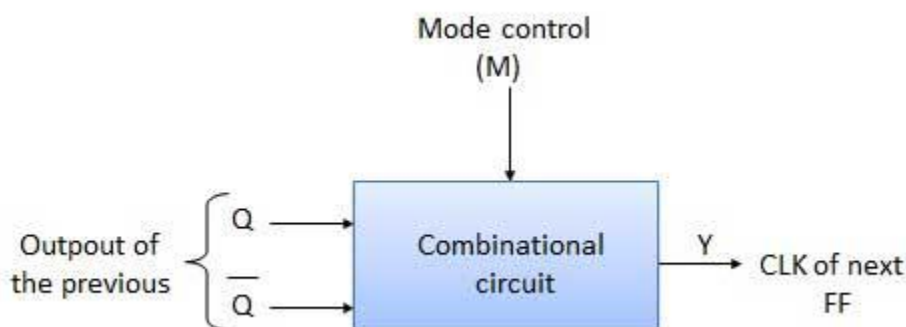
- **UP counting mode (M=0)** – The Q output of the preceding FF is connected to the clock of the next stage if up counting is to be achieved. For this mode, the mode select input M is at logic 0 (M=0).
- **DOWN counting mode (M=1)** – If M = 1, then the Q bar output of the preceding FF is connected to the next FF. This will operate the counter in the counting mode.

Example

3-bit binary up/down ripple counter.

- 3-bit – hence three FFs are required.
- UP/DOWN – So a mode control input is essential.
- For a ripple up counter, the Q output of preceding FF is connected to the clock input of the next one.
- For a ripple up counter, the Q output of preceding FF is connected to the clock input of the next one.
- For a ripple down counter, the Q bar output of preceding FF is connected to the clock input of the next one.
- Let the selection of Q and Q bar output of the preceding FF be controlled by the mode control input M such that, If M = 0, UP counting. So connect Q to CLK. If M = 1, DOWN counting. So connect Q bar to CLK.

Block Diagram



Truth Table

Inputs			Outputs
M	Q	\overline{Q}	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Y = Q for up counter

Y = \overline{Q} for up counter

Case 1 – With M = 0 (Up counting mode)

, If $M = 0$ and $\overline{M} = 1$, then the AND gates 1 and 3 in fig. will be enabled whereas the AND gates 2 and 4 will be disabled.

Hence Q_A gets connected to the clock input of FF-B and Q_B gets connected to the clock input of FF-C.

These connections are same as those for the normal up counter. Thus with $M = 0$ the circuit work as an up counter.

Case 2: With M = 1 (Down counting mode)

If $M = 1$, then AND gates 2 and 4 in fig. are enabled whereas the AND gates 1 and 3 are disabled.

Hence $\overline{Q_A}$ gets connected to the clock input of FF-B and $\overline{Q_B}$ gets connected to the clock input of FF-C.

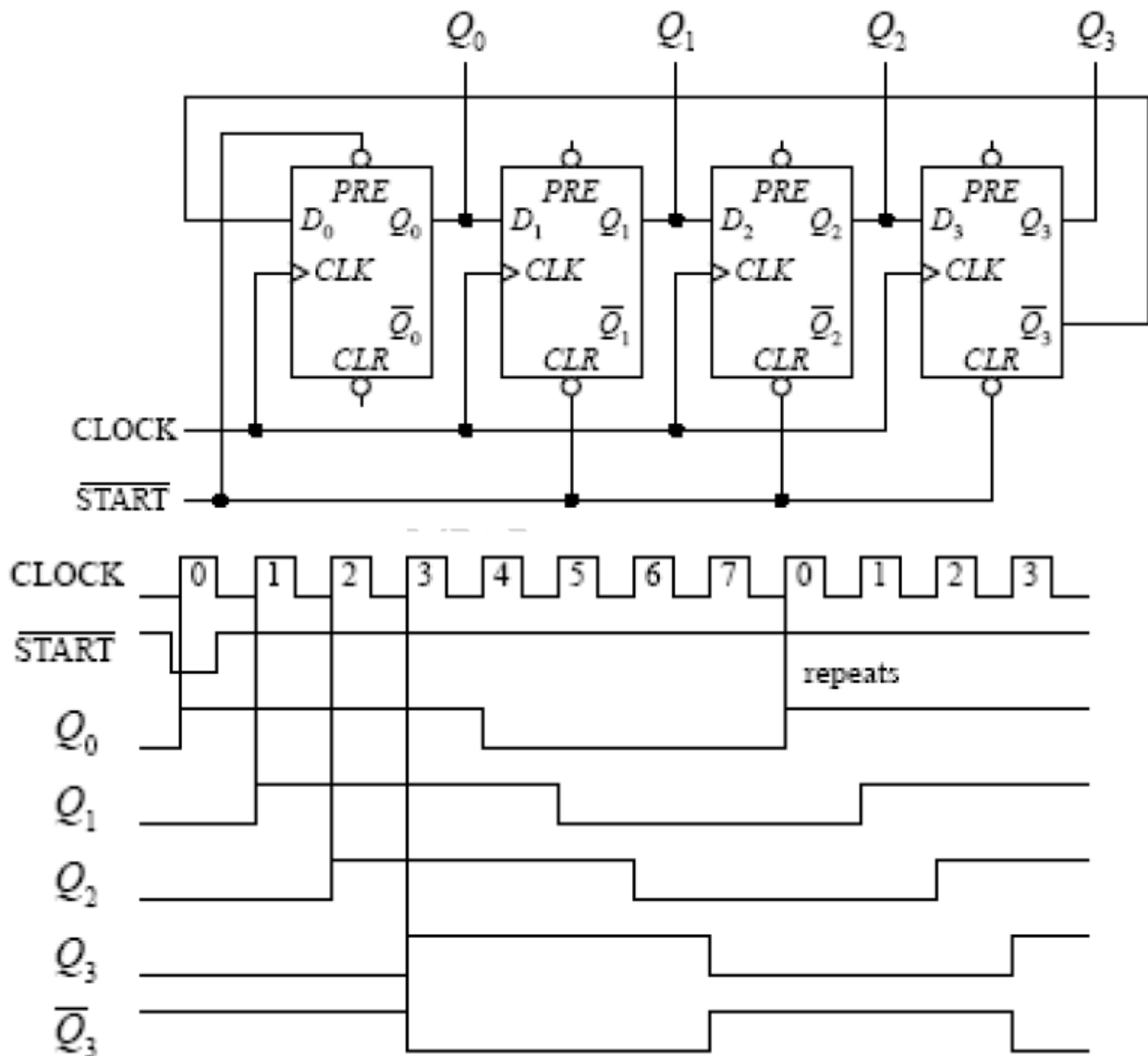
These connections will produce a down counter. Thus with $M = 1$ the circuit works as a down counter.

3.7.1.3 **Ring Counter**

3.7.1.4 Johnson Counter

- The Johnson shift counter is similar to the ring counter except that its last flip-flop feeds data back to the first flip-flop from its inverted output (Q).
- In the simple 4-bit Johnson shift counter shown below, we start out by applying a low to the START line, which sets Q_0 high and Q_1 , Q_2 , and Q_3 low $\rightarrow Q_3$ high.
- In other words, you load the register with the binary word 1000, as you did with the ring counter. Now, when you bring START line low, data will shift through the register.
- However, unlike the ring counter, the first bit sent back to the D_0 input of the first flip-flop will be high because feedback is from Q_3 not Q_3 .
- At the next clock edge, another high is fed back to D_0 ; at the next clock edge, another high is fed back; at the next edge, another high is fed back.
- Only after the fourth clock edge does a low get fed back (the 1 has shifted down to the last flip-flop and Q_3 goes high).

- At this point, the shift register is full of 1s. As more clock pulses arrive, the feedback loop supplies lows to D_0 for the next four clock pulses.
- After that, the Q outputs of all the flip-flops are low while Q_3 goes high. This high from Q_3 is fed back to D_0 during the next positive clock edge, and the cycle repeats.
- As you can see, the 4-bit Johnson shift counter has 8 output stages (which require 8 clock pulses to recycle), not 4, as was the case with the ring counter. Johnson counter using positive edge-triggered D flip-flops



This application can use as LAMP decoration or Advertiser. You can make more output by those connection use mode D- Flip flop.

DESIGN PROCEDURE OF A SYNCHRONOUS COUNTER

Following certain general steps, synchronous counters of any given count sequence and modulus can be designed. The steps are listed below:

Step 1. From the given word description of the problem, draw a state diagram that describes the operation of the counter.

Step 2. From the state table, write the count sequences in the form of a table as shown in Table 9.10.

Step 3. Find the number of flip-flops required.

Step 4. Decide the type of flip-flop to be used for the design of the counter. Then determine the flip-flop inputs that must be present for the desired next state from the present state using the excitation table of the flip-flops.

Step 5. Prepare K-maps for each flip-flop input in terms of flip-flop outputs as the input variables. Simplify the K-maps and obtain the minimized expressions.

Step 6. Connect the circuit using flip-flops and other gates corresponding to the minimized expressions.

3.8 State diagram, State table, State minimization

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.

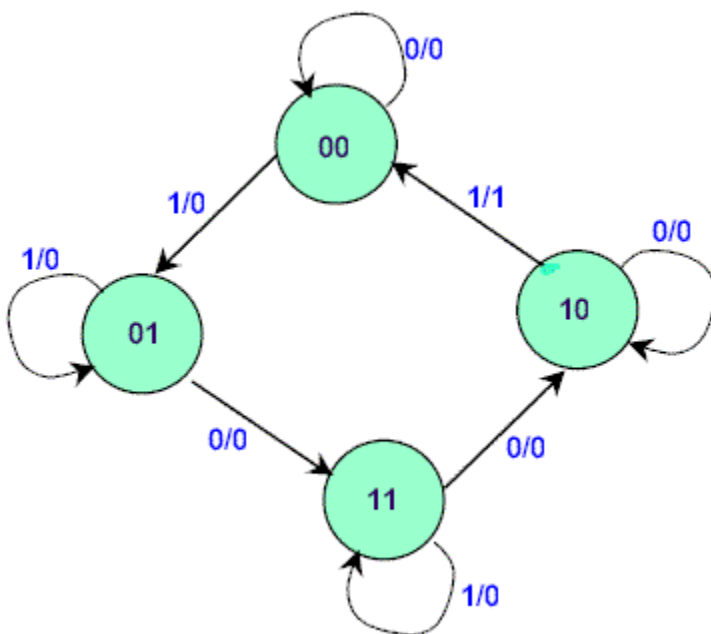


Figure 3. State Diagram

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: This example is taken from P. K. Lala, Practical Digital Logic Design and Testing, Prentice Hall, 1996, p.155.

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).

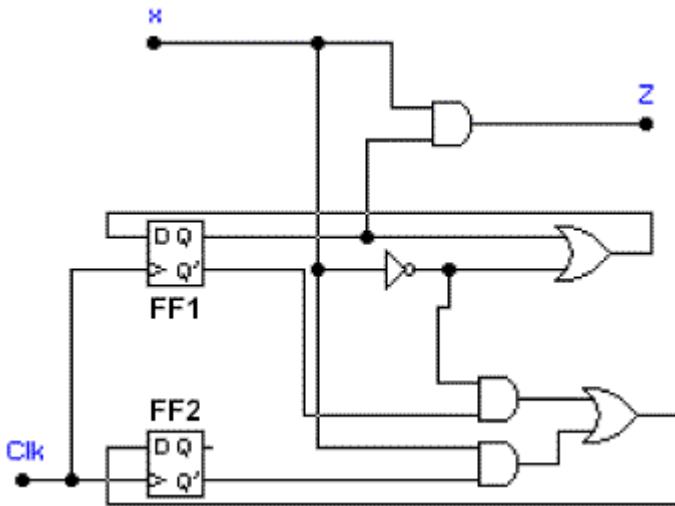


Figure 4. A Sequential Circuit

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

$$Z = x \cdot Q_1$$

$$D_1 = x' + Q_1$$

$$D_2 = x \cdot Q_2' + x' \cdot Q_1'$$

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 Q ₁ Q ₂	Next State		Output	
	x = 0	x = 1	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.

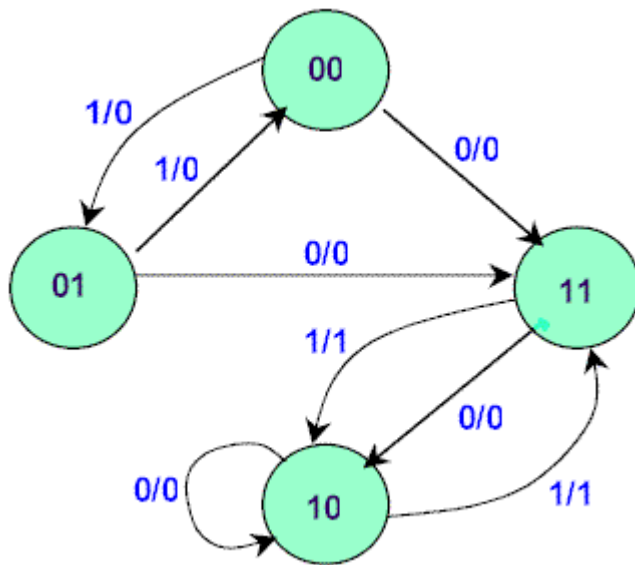


Figure 5. State Diagram of circuit in Figure 4.

State Diagrams of Various Flip-flops

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

NAME	STATE DIAGRAM
SR	<p>SR flip-flop state diagram showing two states: $Q = 0$ and $Q = 1$. Transitions are labeled with S, R values:</p> <ul style="list-style-type: none"> $Q = 0$ to $Q = 0$: $S, R = 0, 0$ $Q = 1$ to $Q = 1$: $S, R = 0, 0$ $Q = 0$ to $Q = 1$: $S, R = 1, 0$ $Q = 1$ to $Q = 0$: $S, R = 0, 1$
JK	<p>JK flip-flop state diagram showing two states: $Q = 0$ and $Q = 1$. Transitions are labeled with J, K values:</p> <ul style="list-style-type: none"> $Q = 0$ to $Q = 0$: $J, K = 0, 0$ $Q = 1$ to $Q = 1$: $J, K = 0, 0$ $Q = 0$ to $Q = 1$: $J, K = 1, 0$ or $1, 1$ $Q = 1$ to $Q = 0$: $J, K = 0, 1$ or $1, 1$
D	<p>D flip-flop state diagram showing two states: $Q = 0$ and $Q = 1$. Transitions are labeled with D values:</p> <ul style="list-style-type: none"> $Q = 0$ to $Q = 0$: $D = 1$ $Q = 1$ to $Q = 1$: $D = 1$ $Q = 0$ to $Q = 1$: $D = 1$ $Q = 1$ to $Q = 0$: $D = 0$

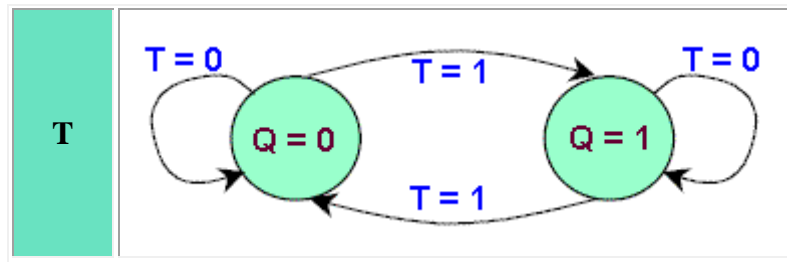


Table 3. State diagrams of the four types of flip-flops.

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.

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 1.

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 1. 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 2.

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 2. 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 in Table 3.

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

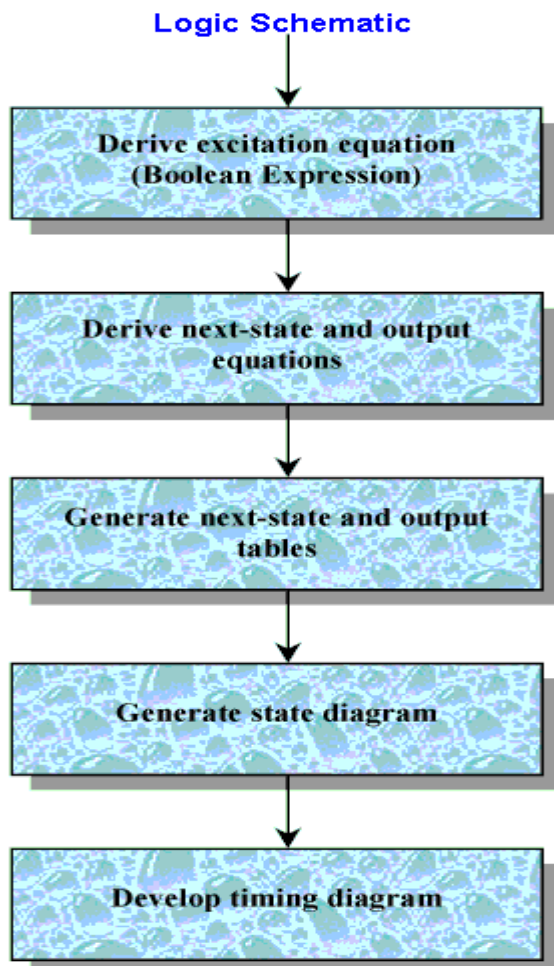
Table 3. Reduced state table

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.

Analysis of Sequential Circuits

The behaviour of a sequential circuit is determined from the inputs, the outputs and the states of its flip-flops. Both the output and the next state are a function of the inputs and the present state.

The suggested analysis procedure of a sequential circuit is set out in Figure 6 below.



We start with the logic schematic from which we can derive excitation equations for each flip-flop input. Then, to obtain next-state equations, we insert the excitation equations into the characteristic equations. The output equations can be derived from the schematic, and once we have our output and next-state equations, we can generate the next-state and output tables as well as state diagrams. When we reach this stage, we use either the table or the state diagram to develop a timing diagram which can be verified through simulation.

Figure . Analysis procedure of sequential circuits.

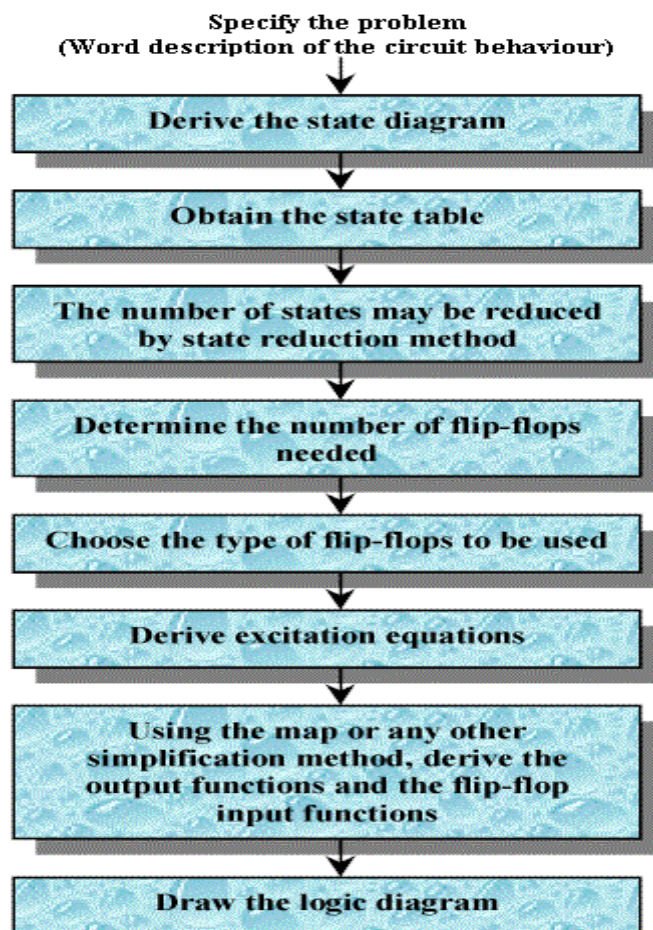
Now let's look at some examples, using these procedures to analyse a sequential circuit.

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.



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

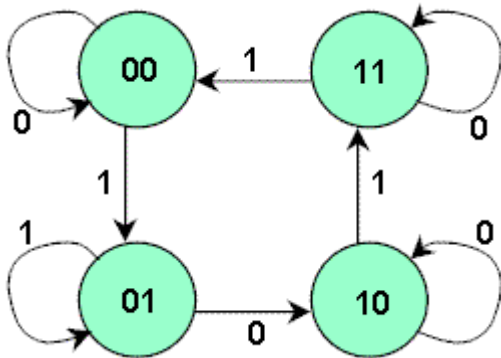


Figure . State diagram

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	x = 1
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 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, where the present state and input variables are arranged in the form of a truth table. Remember, the excitation for the JK flip-flop was derived in Table .

Table Excitation table for JK flip-flop

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

Table Excitation table of the circuit

Present State	Next State	Input	Flip-flop Inputs	
Q0 Q1	Q0 Q1	x	J0K0	J1K1
0 0	0 0	0	0 X	0 X
0 0	0 1	1	0 X	1 X
0 1	1 0	0	1 X	X 1
0 1	0 1	1	0 X	X 0
1 0	1 0	0	X 0	0 X
1 0	1 1	1	X 0	1 X
1 1	1 1	0	X 0	X 0
1 1	0 0	1	X 1	X 1

In the first row of Table , we have a transition for flip-flop Q0 from 0 in the present state to 0 in the next state. In Table 10 we find that a transition of states from 0 to 0 requires that input J = 0 and input K = X. So 0 and X are copied in the first row under J0 and K0 respectively. Since the first row also shows a transition for the flip-flop Q1 from 0 in the present state to 0 in the next state, 0 and X are copied in the first row under J1 and K1. This process is continued for each row of the table and for each flip-flop, with the input conditions as specified in Table 10.

The simplified Boolean functions for the combinational circuit can now be derived. The input variables are Q0, Q1, and x; the output are the variables J0, K0, J1 and K1. The information from the truth table is plotted on the Karnaugh maps shown in Figure .

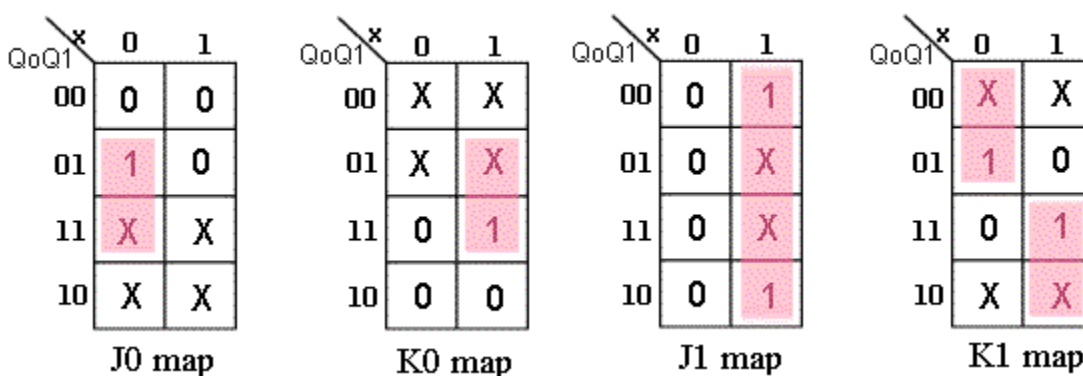


Figure . Karnaugh Maps

The flip-flop input functions are derived:

$$\begin{aligned}
 J0 &= Q1*x' & K0 &= Q1*x \\
 J1 &= x & K1 &= Q0'*x' + Q0*x = Q0 \oplus x
 \end{aligned}$$

Note: the symbol \boxtimes is exclusive-NOR.

The logic diagram is drawn in Figure .

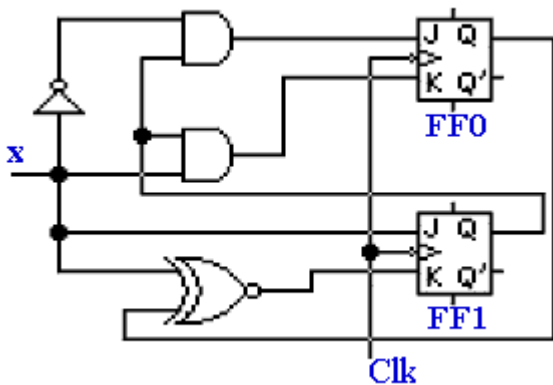


Figure . Logic diagram of the sequential circuit.

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

Table . State table of a sequential circuit.

Present State Q0 Q1	Next State		Output	
	x = 0	x = 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 . Excitation table for a D flip-flop.

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

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

Present State Q0 Q1	Next State Q0 Q1	Input x	Flip-flop Inputs	Output Z
------------------------	---------------------	------------	---------------------	-------------

				D0	D1	
00	00	0	0	0	0	0
00	01	1	0	0	1	0
01	00	0	0	0	0	0
01	10	1	1	0	0	0
10	11	0	1	1	0	0
10	10	1	1	0	0	0
11	00	0	0	0	0	0
11	01	1	0	1	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 .

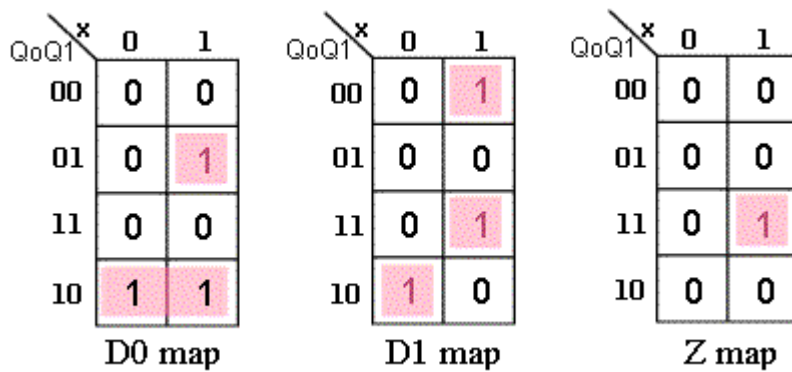


Figure 16. Karnaugh maps

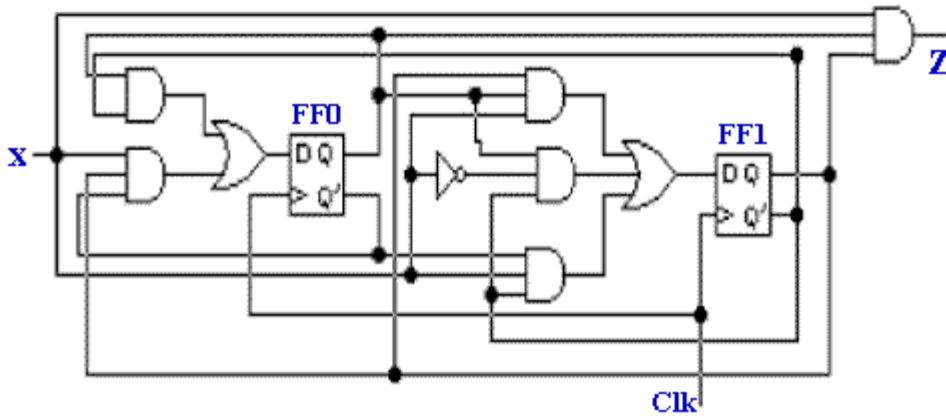
The simplified Boolean expressions are:

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

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

$$Z = Q0*Q1*x$$

Finally, draw the logic diagram.



Design of Counters

A counter is first described by a state diagram, which shows the sequence of states through which the counter advances when it is clocked. Figure shows a state diagram of a 3-bit binary counter.

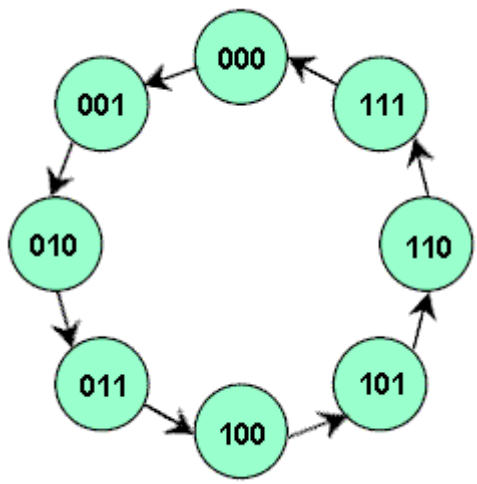
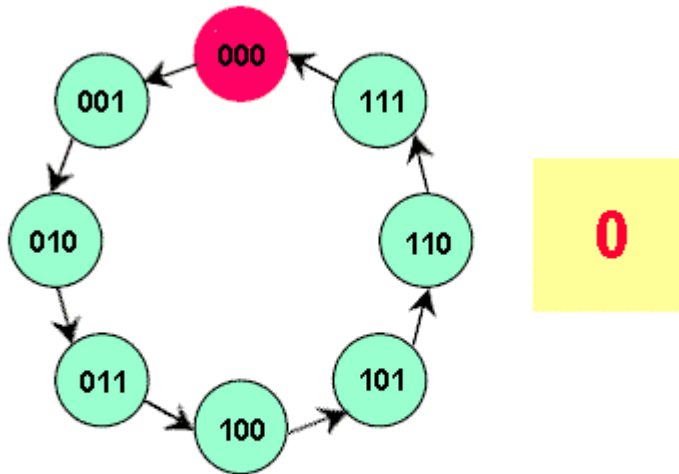


Figure . State diagram of a 3-bit binary counter.

The circuit has no inputs other than the clock pulse and no outputs other than its internal state (outputs are taken off each flip-flop in the counter). The next state of the counter depends entirely on its present state, and the state transition occurs every time the clock pulse occurs. Figure 19 shows the sequences of count after each clock pulse.



clk

Once the sequential circuit is defined by the state diagram, the next step is to obtain the next-state table, which is derived from the state diagram in Figure and is shown in Table

Table . State table

Present State			Next State		
Q2	Q1	Q0	Q2	Q1	Q0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

Since there are eight states, the number of flip-flops required would be three. Now we want to implement the counter design using JK flip-flops.

Next step is to develop an excitation table from the state table, which is shown in Table .

Table Excitation table

Output State Transitions			Flip-flop inputs					
Present State	Next State							
Q2	Q1	Q0	J2	K2	J1	K1	J0	K0
Q2	Q1	Q0	Q2	Q1	Q0			

0 0 0	0 0 1	0 X	0 X	1 X
0 0 1	0 1 0	0 X	1 X	X 1
0 1 0	0 1 1	0 X	X 0	1 X
0 1 1	1 0 0	1 X	X 1	X 1
1 0 0	1 0 1	X 0	0 X	1 X
1 0 1	1 1 0	X 0	1 X	X 1
1 1 0	1 1 1	X 0	X 0	1 X
1 1 1	0 0 0	X 1	X 1	X 1

Now transfer the JK states of the flip-flop inputs from the excitation table to Karnaugh maps to derive a simplified Boolean expression for each flip-flop input. This is shown in Figure .

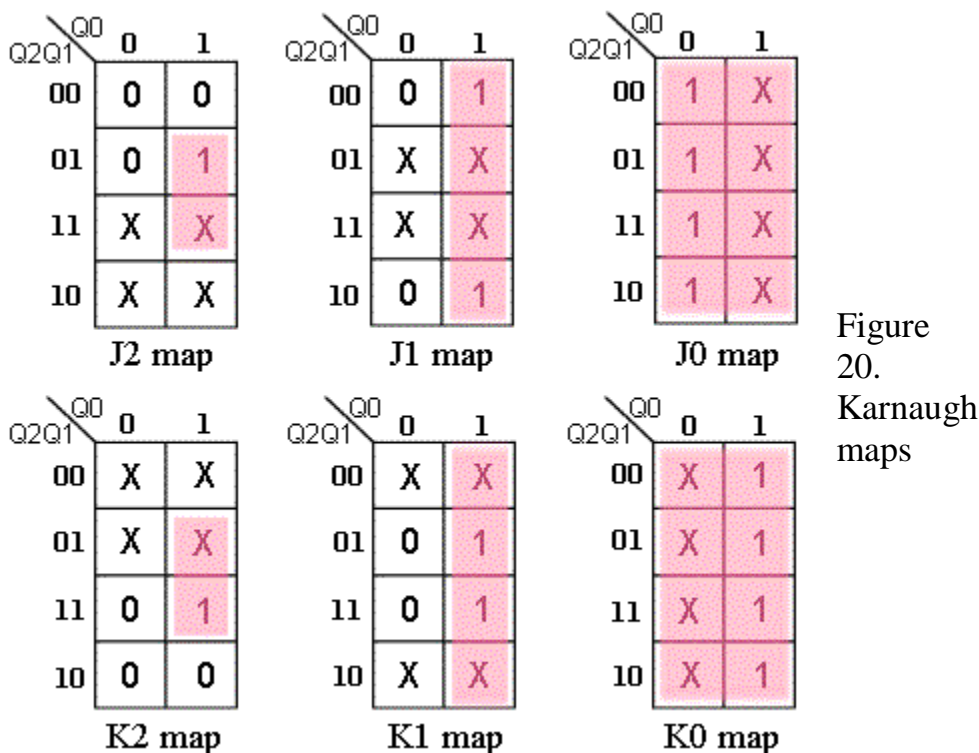


Figure 20. Karnaugh maps

The 1s in the Karnaugh maps of Figure are grouped with "don't cares" and the following expressions for the J and K inputs of each flip-flop are obtained:

$$\begin{aligned}
 \mathbf{J0} &= \mathbf{K0} = \mathbf{1} \\
 \mathbf{J1} &= \mathbf{K1} = \mathbf{Q0} \\
 \mathbf{J2} &= \mathbf{K2} = \mathbf{Q1*Q0}
 \end{aligned}$$

The final step is to implement the combinational logic from the equations and connect the flip-flops to form the sequential circuit. The complete logic of a 3-bit binary counter is shown in Figure .

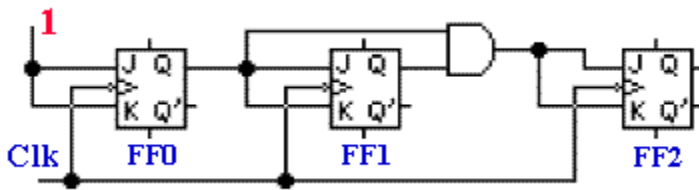


Figure .
Logic diagram of a 3-bit binary counter.

Design a counter specified by the state diagram using T flip-flops. The state diagram is shown here again in Figure .

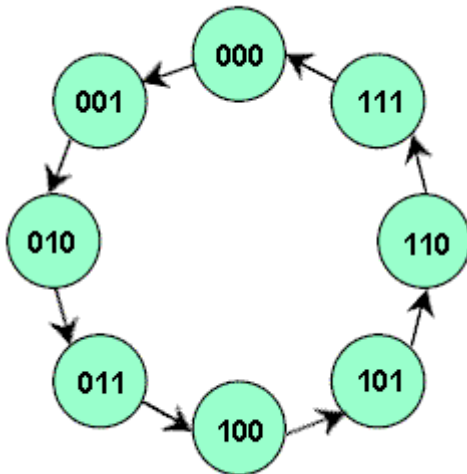


Figure . State diagram of a 3-bit binary counter.

The state table will be the same as in Example 1.5.

Now derive the excitation table from the state table, which is shown in Table .

Table . Excitation table.

Output State Transitions			Flip-flop inputs		
Present State	Next State				
Q2 Q1 Q0	Q2	Q1 Q0	T2	T1	T0
0 0 0	0	0 1	0	0	1
0 0 1	0	1 0	0	1	1
0 1 0	0	1 1	0	0	1
0 1 1	1	0 0	1	1	1
1 0 0	1	0 1	0	0	1
1 0 1	1	1 0	0	1	1
1 1 0	1	1 1	0	0	1
1 1 1	0	0 0	1	1	1

Next step is to transfer the flip-flop input functions to Karnaugh maps to derive a simplified Boolean expressions, which is shown in Figure .

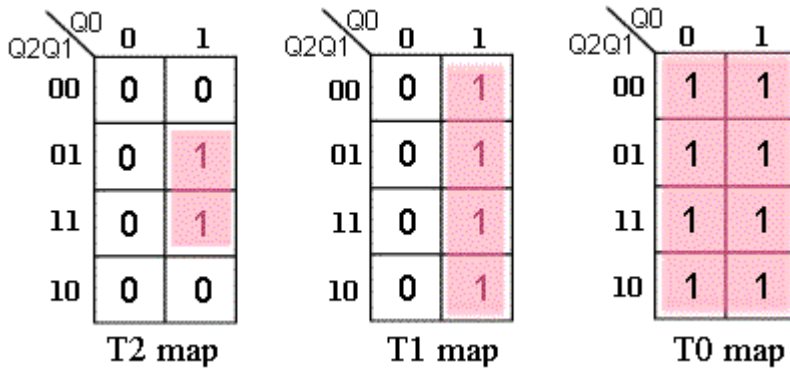


Figure 23. Karnaugh maps

The following expressions are obtained:

$$T0 = 1; \quad T1 = Q0; \quad T2 = Q1 * Q0$$

Finally, draw the logic diagram of the circuit from the expressions obtained. The complete logic diagram of the counter is shown in Figure .

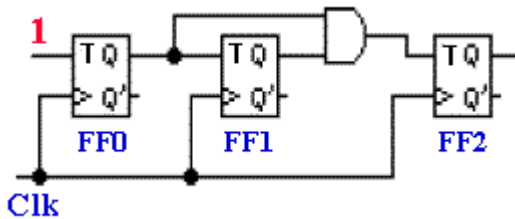


Figure . Logic diagram of 3-bit binary counter.

3.9 Hazards

What are Hazards ?

Hazards in any system are obviously an un-desirable effect caused by either a deficiency in the system or external influences. In digital logic hazards are usually referred to in one of three ways:

- [Static Hazards](#)

- [Dynamic Hazards](#)
- [Function Hazards](#)

These logic hazards are all subsets of the same problem: - When changes in the input variables do not change the output due to some form of delay caused by logic elements (NOT, AND, OR gates etc), this results in the logic not performing its function properly.

This is however a temporary problem, and the logic will finally come to the desired function. Despite the logic arriving at the correct output, it is imperative that hazards be eliminated as they can have an effect on other systems. Imagine hazards like this in a piece of hospital equipment.

Static Hazards

Definition:- "When one input variable changes, the output changes momentarily when it shouldn't"

This particular type of hazard is usually due to a NOT gate within the logic. We can see the effects of the delay in the circuit from the following flash animation.

The hazard can be dealt with in two ways:

Insert another (additional) delay to the circuit. This then eliminates the static hazard.

Eliminate the hazard by inserting more logic to counteract the effects (Note this makes assumptions that the logic will fail)

The first case is the most used of the two options. This is because it does not make assumptions about the logic, instead the method adds redundancy to overcome the hazard.

To solve the hazard we shall use our previous example and apply a theory that 'Huffman' discovered. The insertion of a redundant loop can eliminate a static hazard.

In the next example, it will also be evident that there will not be a situation where a static '0' occurs. A static '0' hazard is one which briefly goes to '1' when it should remain at '0'. A static '1' hazard is the reverse of this situation, i.e. the output should remain at '1' yet under some condition it briefly changes state to '0' (something we shall see in the following example)..

Example of Static Hazards

The Static '1' Hazard.

Let us consider an imperfect circuit that suffers from a delay in the physical logic elements i.e. AND gates etc.

The simple circuit performs the function:

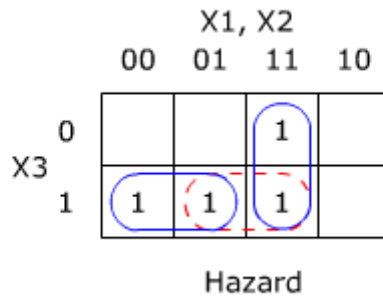
$$f = X1.X2 + X1'.X3$$

and the logic diagram can be shown as follows:

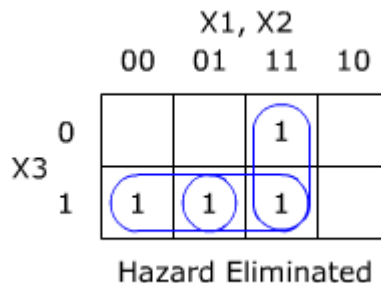
<insert flash logic animation>

If we first look at the starting diagram, it is clear that if no delays were to occur, then the circuit would function normally. However since this isn't a perfect circuit, and an error occurs when the input changes from 111 to 011. i.e. When X1 changes state.

Now we know roughly how the hazard is occurring, for a clearer picture and the solution on how to solve this problem, we look to the Karnaugh Map:



This Karnaugh Map shows the circuit. The two gates are shown by solid rings, and the hazard can be seen under the dashed ring. The theory proved by Huffman tells us that by adding a redundant loop 'X2X3' this will eliminate the hazard. So the resulting logic is of the form shown in the next figure.



So our original function is now: $f = X1.X2 + X1'.X3 + X2.X3$

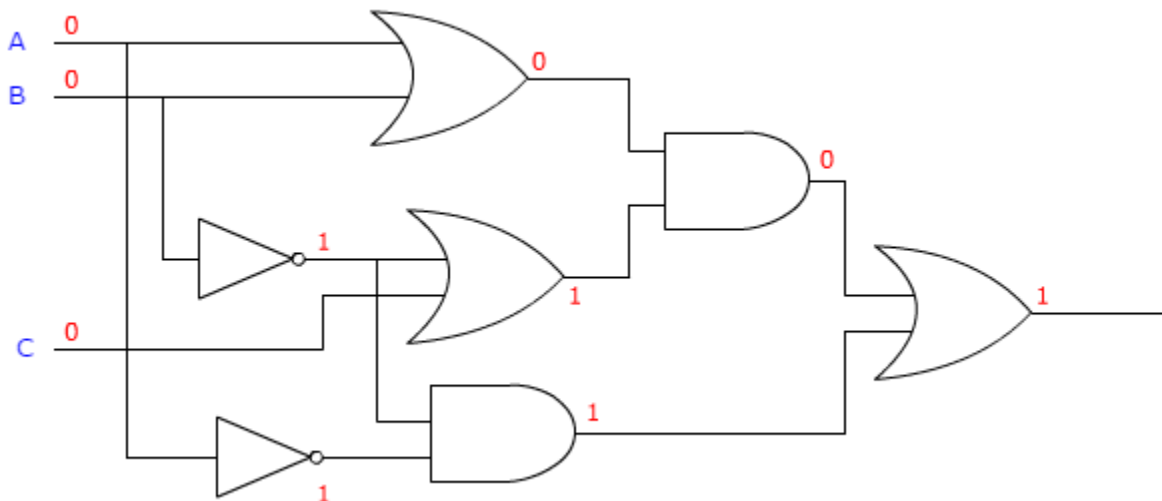
Now we can see that even with imperfect logic elements, our example will not show signs of hazards when X1 changes state. This theory can be applied to any logic system. Computer programs deal with most of this work now, but for simple examples it is quicker to do the debugging by hand. When there are many input variables (say 6 or more) it will become quite difficult to 'see' the errors on a Karnaugh map.

Dynamic Hazards

Definition:- "A dynamic hazard is the possibility of an output changing more than once as a result of a single input change"

Dynamic hazards often occur in larger logic circuits where there are different routes to the output (from the input). If each route has a different delay, then it quickly becomes clear that there is the potential for changing output values that differ from the required / expected output.

e.g. A logic circuit is meant to change output state from '1' to '0', but instead changes from '1' to '0' then '1' and finally rests at the correct value '0'. This is a dynamic hazard.

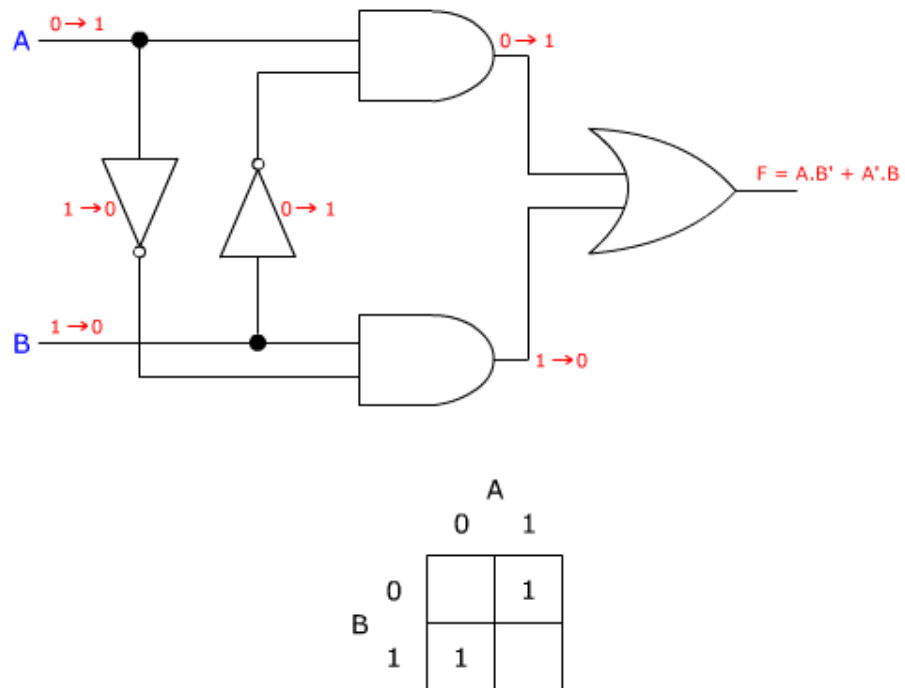


Function Hazards

Function hazards are non-solvable hazards which occurs when more than one input variable changes at the same time. Hazards such as function hazards can not be logically eliminated as the problem lies with actual specification of the circuit. The only real way to avoid such problems is to restrict the changing of input variables so that only one input should change at any given time.

Restrictions are not always possible, for instance let us imagine some logic circuit that has two inputs. One input is used for a clock signal, and the other is connected to a random noise source that we wish to measure. It should be clear that restrictions in this case would not be an effective solution.

The simplest example of this is the exclusive-or function.



In this scenario it is quite difficult to see how a hazard could occur if the circuit is built up on the same couple of chips. However let us imagine that some circuit designer has split this function across different chips (i.e. one NOT gate on one chip and the other NOT gate is implemented on another chip across the PCB somewhere)

Let us setup the initial state of our circuit. $A = 1, B = 0$. Now let's say there is a delay in the NOT gate marked (X). The inputs now change simultaneously so that $A = 0$ and $B = 1$ (remember in an equally delayed circuit or a perfect circuit, the circuit output would match the specification). If we observe what the circuit should do, and do not change the output of the NOT gate X (this simulates a delay in gate X), it should be clear that the output of the circuit changes. Now we change the output of NOT gate X and the circuit goes back to the proper state.

The most effective way to solve this hazard would be to carefully design the PCB so that delays are all equal, or at least match the delays on each path. i.e. Delay of A's path = Delay of B's path. Yet adding more gates to the circuit by the same methods as described in dynamic and static hazards will not work as Huffman's method cannot be applied.