UNIT 4:

Sequential logic circuits

Introduction of Sequential Circuits

Sequential circuits are digital circuits that store and use the previous state information to determine their next state. Unlike combinational circuits, which only depend on the current input values to produce outputs, sequential circuits depend on both the current inputs and the previous state stored in memory elements.

1. Sequential circuits are commonly used in digital systems to implement state machines, timers, counters, and memory elements. The memory elements in sequential circuits can be implemented using flip-flops, which are circuits that store binary values and maintain their state even when the inputs change.
2. There are two types of sequential circuits: finite state machines (FSMs) and synchronous sequential circuits. FSMs are designed to have a limited number of states and are typically used to implement state machines and control systems. Synchronous sequential circuits, on the other hand, are designed to have an infinite number of states and are typically used to implement timers, counters, and memory elements.

In summary, sequential circuits are digital circuits that store and use previous state information to determine their next state. They are commonly used in digital systems to implement state machines, timers, counters, and memory elements and are essential components in digital systems design.

Sequential circuit is a combinational logic circuit that consists of inputs variable (X), logic gates (Computational circuit), and output variable (Z).

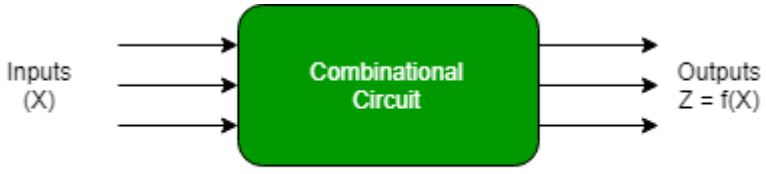


Figure: Combinational Circuits

A combinational circuit produces an output based on input variables only, but a **sequential circuit** produces an output based on **current input and previous output variables**. That means sequential circuits include memory elements that are capable of storing binary information. That binary information defines the state of the sequential

circuit at that time. A latch capable of storing one bit of information.

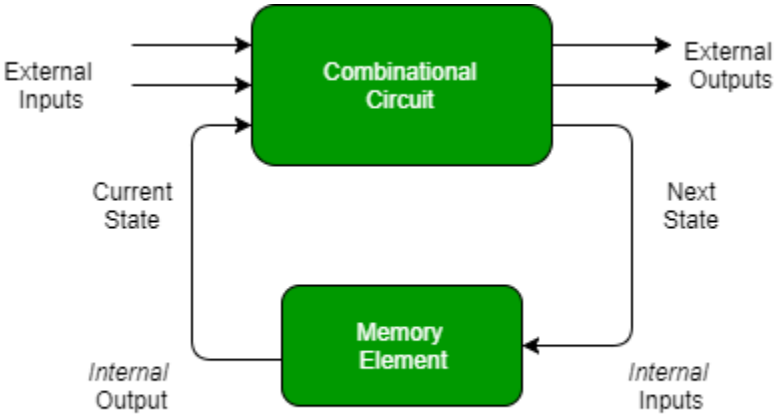


Figure: Sequential Circuit

As shown in the figure, there are two types of input to the combinational logic :

1. External inputs which are not controlled by the circuit.
2. Internal inputs, which are a function of a previous output state.

Secondary inputs are state variables produced by the storage elements, whereas secondary outputs are excitations for the storage elements.

Types of Sequential Circuits:
There are two types of sequential circuits:

Type 1: Asynchronous sequential circuit: These circuits **do not use a clock signal** but uses the pulses of the inputs. These circuits are **faster** than synchronous sequential circuits because there is clock pulse and change their state immediately when there is a change in the input signal. We use asynchronous sequential circuits when speed of operation is important and **independent** of internal clock pulse.

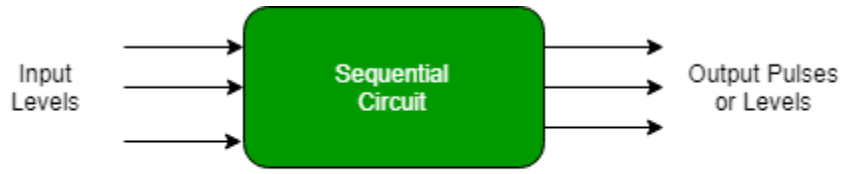


Figure: Asynchronous Sequential Circuit

But these circuits are more **difficult** to design and their output is **uncertain**.

Type2: Synchronous sequential circuit: These circuits **uses clock signal** and level inputs (or pulsed) (with restrictions on pulse width and circuit propagation). The output pulse is the same duration as the clock pulse for the clocked sequential circuits. Since they

wait for the next clock pulse to arrive to perform the next operation, so these circuits are bit **slower** compared to asynchronous. Level output changes state at the start of an input pulse and remains in that until the next input or clock pulse.

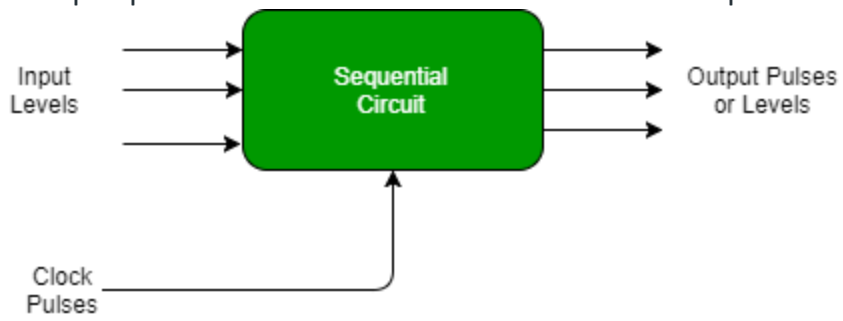


Figure: Synchronous Sequential Circuit

We use synchronous sequential circuit in synchronous counters, flip flops, and in the design of MOORE-MEALY state management machines. We use sequential circuits to design Counters, Registers, RAM, MOORE/MEALY Machine and other state retaining machines.

Advantages of Sequential Circuits:

1. Memory: Sequential circuits have the ability to store binary values, which makes them ideal for applications that require memory elements, such as timers and counters.
2. Timing: Sequential circuits are commonly used to implement timing and synchronization in digital systems, making them essential for real-time control applications.
3. State machine implementation: Sequential circuits can be used to implement state machines, which are useful for controlling complex digital systems and ensuring that they operate as intended.
4. Error detection: Sequential circuits can be designed to detect errors in digital systems and respond accordingly, improving the reliability of digital systems.

Disadvantages of Sequential Circuits:

1. Complexity: Sequential circuits are typically more complex than combinational circuits and require more components to implement.
2. Timing constraints: The design of sequential circuits can be challenging due to the need to ensure that the timing of the inputs and outputs is correct.

3. Testing and debugging: Testing and debugging sequential circuits can be more difficult compared to combinational circuits due to their complex structure and state-dependant outputs.

Latches

Latches are digital circuits that store a single bit of information and hold its value until it is updated by new input signals. They are used in digital systems as temporary storage elements to store binary information. Latches can be implemented using various digital logic gates, such as AND, OR, NOT, NAND, and NOR gates.

There are two types of latches:

1. S-R (Set-Reset) Latches: S-R latches are the simplest form of latches and are implemented using two inputs: S (Set) and R (Reset). The S input sets the output to 1, while the R input resets the output to 0. When both S and R are at 1, the latch is said to be in an "undefined" state.
2. D (Data) Latches: D latches are also known as transparent latches and are implemented using two inputs: D (Data) and a clock signal. The output of the latch follows the input at the D terminal as long as the clock signal is high. When the clock signal goes low, the output of the latch is stored and held until the next rising edge of the clock.
3. Latches are widely used in digital systems for various applications, including data storage, control circuits, and flip-flop circuits. They are often used in combination with other digital circuits to implement sequential circuits, such as state machines and memory elements.
4. In summary, latches are digital circuits that store a single bit of information and hold its value until it is updated by new input signals. There are two types of latches: S-R (Set-Reset) Latches and D (Data) Latches, and they are widely used in digital systems for various applications.

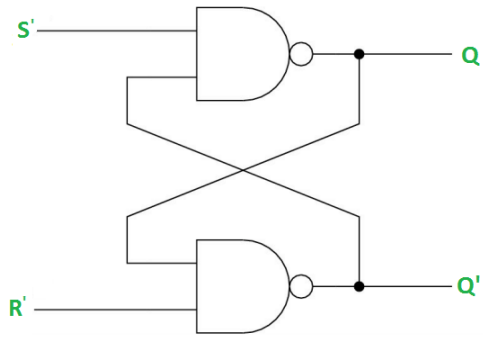
Latches are basic storage elements that operate with signal levels (rather than signal transitions). Latches controlled by a clock transition are flip-flops. Latches are level-sensitive devices. Latches are useful for the design of the asynchronous sequential circuit. Latches are sequential circuit with two stable states. These are sensitive to the input voltage applied and does not depend on the clock pulse. Flip flops that do not use clock pulse are referred to as latch.

SR (Set-Reset) Latch – They are also known as preset and clear states. The SR latch forms the basic building blocks of all other types of flip-flops.

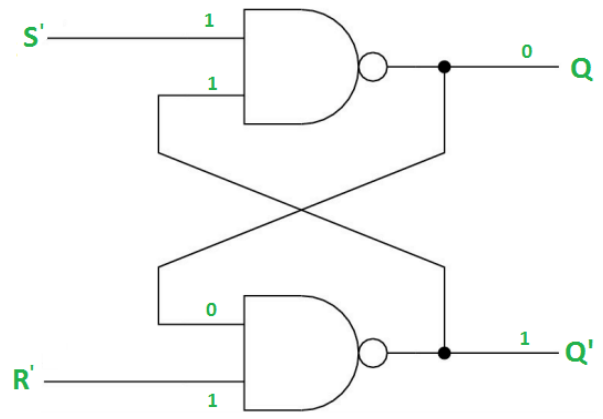
SR Latch is a circuit with:
(i) 2 cross-coupled NOR gate or 2 cross-coupled NAND gate.
(ii) 2 input S for SET and R for RESET.
(iii) 2 output Q, Q'.

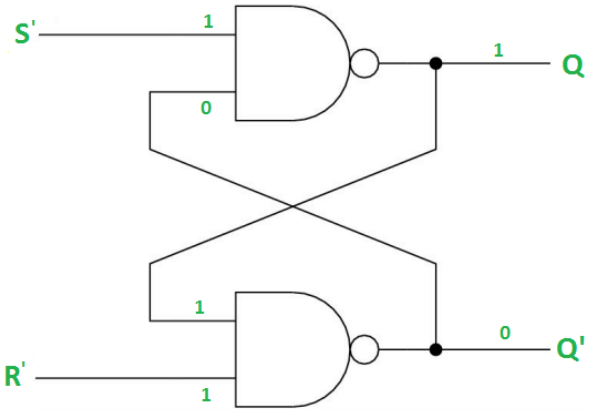
Q	Q'	STATE
1	0	Set
0	1	Reset

Under normal conditions, both the input remains 0. The following is the RS Latch with NAND gates:

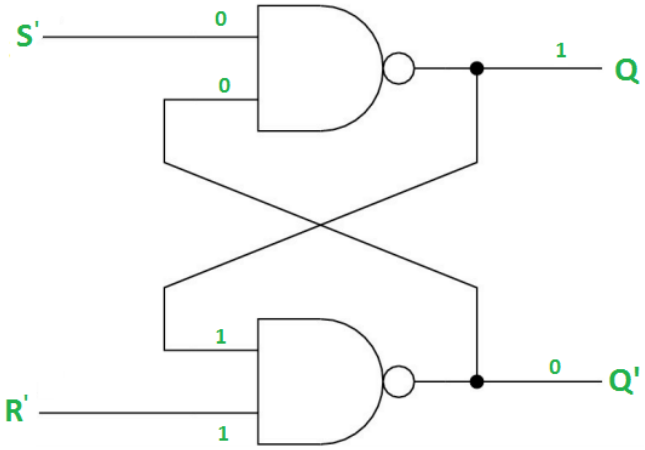


Case-1: S'=R'=1 (S=R=0) –
If Q = 1, Q and R' inputs for 2nd NAND gate are both 1.
If Q = 0, Q and R' inputs for 2nd NAND gate are 0 and 1 respectively.

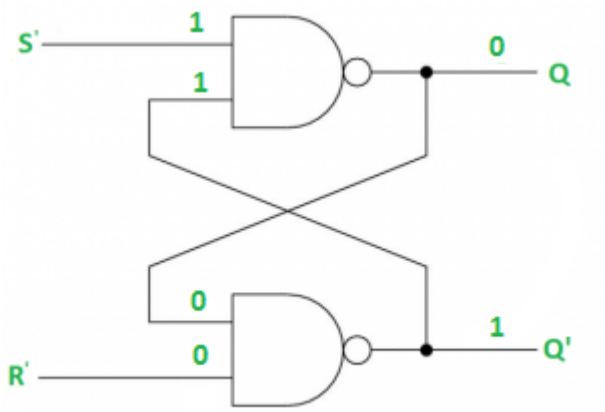




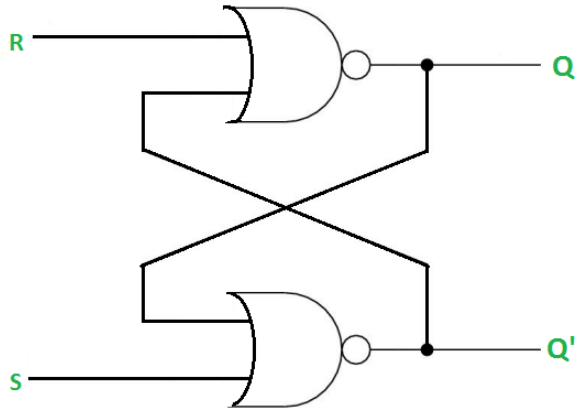
Case-2: S'=0, R'=1 (S=1, R=0) –
As S'=0, the output of 1st NAND gate, Q = 1(**SET state**). In 2nd NAND gate, as Q and R' inputs are 1, Q'=0.



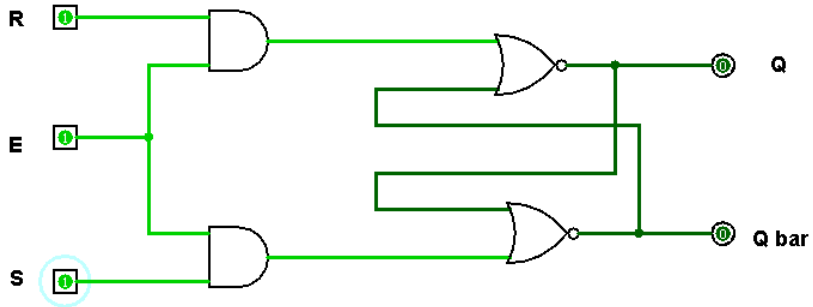
Case-3: S'= 1, R'= 0 (S=0, R=1) –
As R'=0, the output of 2nd NAND gate, Q' = 1. In 1st NAND gate, as Q and S' inputs are 1, Q=0(**RESET state**).



Case-4: S'= R'= 0 (S=R=1) –
When S=R=1, both Q and Q' becomes 1 which is not allowed. So, the input condition is prohibited.
The SR Latch using NOR gate is shown below:



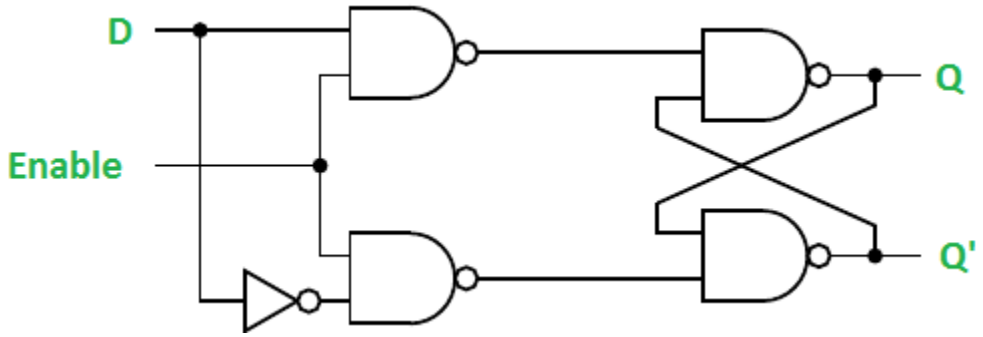
Gated SR Latch –
A Gated SR latch is a SR latch with enable input which works when enable is 1 and retain the previous state when enable is 0.



Gated D Latch –

D latch is similar to SR latch with some modifications made. Here, the inputs are complements of each other. The letter in the D latch stands for “data” as this latch stores single bit temporarily.

The design of D latch with Enable signal is given below:



The truth table for the D-Latch is shown below:

Enable	D	Q(n)	Q(n+1)	STATE
1	0	x	0	RESET
1	1	x	1	SET
0	x	x	Q(n)	No Change

As the output is same as the input D, D latch is also called as *Transparent Latch*.

Considering the truth table, the characteristic equation for D latch with enable input can be given as:

$Q(n+1) = EN.D + EN'.Q(n)$

Advantages of Latches:

1. Easy to Implement: Latches are simple digital circuits that can be easily implemented using basic digital logic gates.
2. Low Power Consumption: Latches consume less power compared to other sequential circuits such as flip-flops.
3. High Speed: Latches can operate at high speeds, making them suitable for use in high-speed digital systems.
4. Low Cost: Latches are inexpensive to manufacture and can be used in low-cost digital systems.
5. Versatility: Latches can be used for various applications, such as data storage, control circuits, and flip-flop circuits.

Disadvantages of Latches:

1. No Clock: Latches do not have a clock signal to synchronize their operations, making their behavior unpredictable.
2. Unstable State: Latches can sometimes enter into an unstable state when both inputs are at 1. This can result in unexpected behavior in the digital system.
3. Complex Timing: The timing of latches can be complex and difficult to specify, making them less suitable for real-time control applications.

Switch Debounce

We all know how a switch or a push button works, you simply press it to toggle its state. In electronics, switches are used to drive or represent many things, and these are mainly in the form of different voltage level, thus making the system binary, i.e either ON or OFF and in voltage levels High or Low. Thus, these voltage levels (e.g. 5 Volts = High = ON = closed circuit and 0 Volts = Low = OFF = open circuit) helps us to represent the binary logic of 0s (zeros) and 1s (ones).

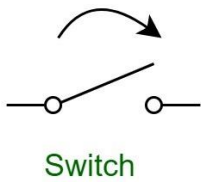


Figure – Normal Switch Diagram

But a lot is happening under the simple push buttons on our keyboards and various other devices. A simple button is basically two metal contacts that touch under the user's input, i.e. pressing action on the button. These metal contacts then make the underlying circuit complete and informs the sensing element (in most cases a micro-controller) that the button is pressed.

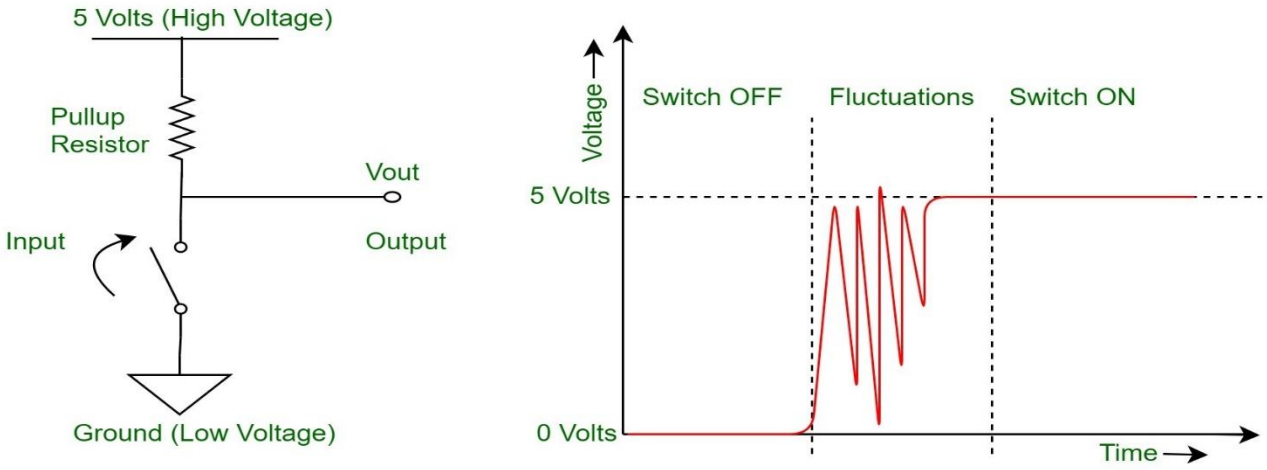


Figure – Switch Bounce Graph

This action of touching metal contacts physically works in a different manner, i.e. the metal contacts bounce upon each other making the switch on and off for a time after impact from pressing action by user. Thus, if the sensing element is sensitive enough, it registers multiple button press from a single action. This can introduce errors in the system you are using and can hinder a lot of processes.

Thus, to avoid this miss-interpretation of button press, the concept of switch debounce is needed. This helps in registering the switch's action correctly. Switch debounce can be achieved through software programming as well as hardware circuits' use. Let's look into these methods in brief.

Software Switch Debouncing:

In this method, the switch's bouncing state effect is eliminated using various algorithms and filters. The programmer can design an algorithm with use of shift register and counters such that it will register the switch's state after a delay. Another method is to use filter algorithms on the sampled input from the switch and determine the state of switch based on the output of such digital filter. All this can make the software slightly inefficient, adding to delay in performance if not implemented correctly.

Hardware Switch Debouncing:

In this category, there are various implementations of circuits which can be used for eliminating the effect of switch debouncing right at the hardware level. The different types of circuits used are:

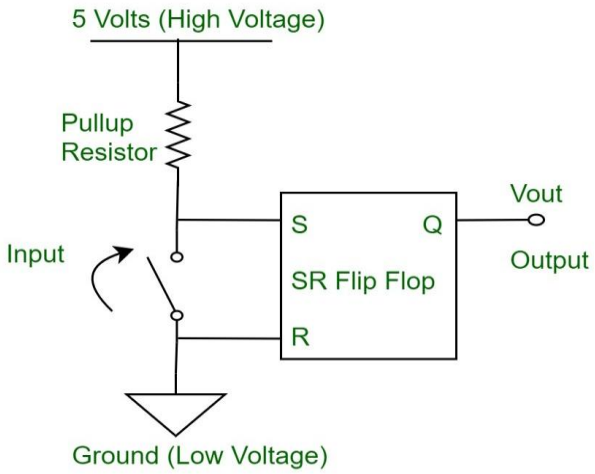


Figure – Switch Debounce using SR Flip Flop Latch

Use of S-R Flip Flop Latch circuit. The circuit when introduced in the output part of the switch, it will retain the voltage level of the input as the output state. Thus, latching to the input, when change in state is introduced. This method is useful, but adds to the bulkiness of the simple circuit.

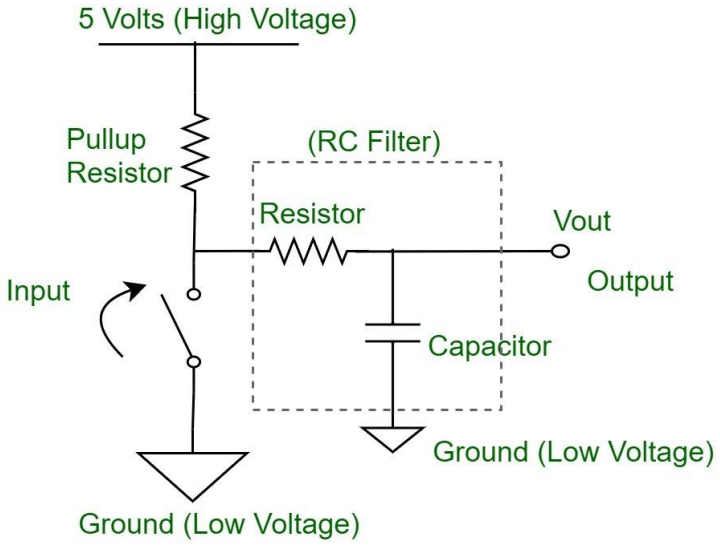


Figure – Switch Debounce using RC Filter

Use of R-C circuit. This circuit involves the combination of a resistor and a capacitor circuit to act as a filter to smooth out the output glitch for the switch.

Use of dedicated ICs. There are various Integrated Circuits available in market specifically designed to eliminate the switch bouncing action. These implement the use of combination circuits to eliminate the fluctuating output of the switch.

Flip-flops and their Types

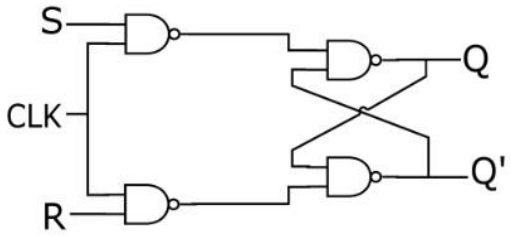
A flip-flop is a sequential digital electronic circuit having two stable states that can be used to store one bit of binary data. Flip-flops are the fundamental building blocks of all memory devices.

Types of Flip–Flops

- S-R flip-flop
- J-K flip-flop
- D flip-flop
- T flip-flop

S-R Flip-flop

- This is the simplest flip-flop circuit. It has a set input (S) and a reset input (R). When in this circuit when S is set as active, the output Q would be high and the Q' will be low. If R is set to active then the output Q is low and the Q' is high. Once the outputs are established, the results of the circuit are maintained until S or R get changed, or the power is turned off.



- **Truth table of S-R flip-flop**

S	R	Q	State
0	0	0	No Change
0	1	0	Reset
1	0	1	Set

S	R	Q	State
1	1	X	

- **Characteristics Table of S-R flip-flop**

S	R	Q(t)	Q(t+1)
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	X
1	1	1	X

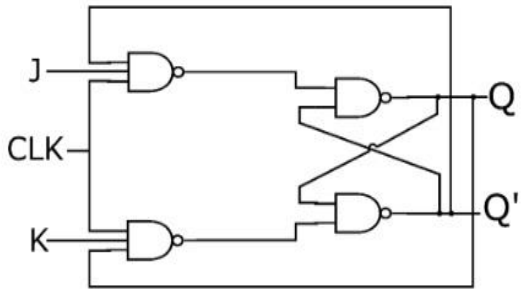
- **Characteristics equation of S-R flip-flop**

$Q(t+1)=S+R'Q(t)$Q(t+1)=S+R‘Q(t)

J-K Flip-flop

- Because of the invalid state corresponding to S=R=1 in the SR flip-flop, there is a need of another flip-flop. The JK flip-flop operates with only positive or negative clock transitions. The operation of the JK flip-flop is similar to the SR flip-flop. When the input J and K are different then the output Q takes the value of J at the next clock edge. When J

and K both are low then NO change occurs at the output. If both J and K are high, then at the clock edge, the output will toggle from one state to the other.



- **Truth table of JK flip-flop**

J	K	Q	State
0	0	0	No Change
0	1	0	Reset
1	0	1	Set
1	1	Toggles	Toggle

- **Characteristics table of JK flip-flop**

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

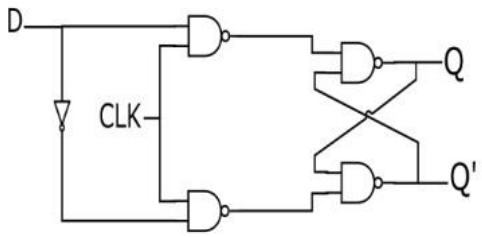
J	K	Q(t)	Q(t+1)
1	0	1	1
1	1	0	1
1	1	1	0

- **Characteristics equation of JK flip-flop**

$Q(t+1)=jkQ(t)'+K'Q(t)$Q(t+1)=jkQ(t)‘+K‘Q(t)

D Flip-flop

- In a D flip-flop, the output can only be changed at positive or negative clock transitions, and when the inputs changed at other times, the output will remain unaffected. The D flip-flops are generally used for shift-registers and counters. The change in output state of D flip-flop depends upon the active transition of clock. The output (Q) is same as input and changes only at active transition of clock



- **Truth table of D flip-flop**

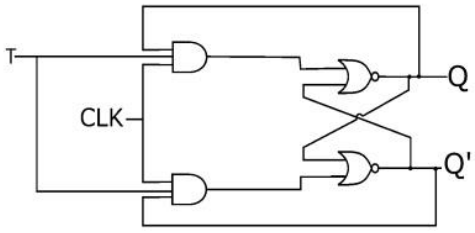
D	Q
0	0
1	1

- **Characteristics equation of D flip-flop**

$Q(t+1)=D$Q(t+1)=D

T Flip-flop

- A T flip-flop (Toggle Flip-flop) is a simplified version of JK flip-flop. The T flop is obtained by connecting the J and K inputs together. The flip-flop has one input terminal and clock input. These flip-flops are said to be T flip-flops because of their ability to toggle the input state. Toggle flip-flops are mostly used in counters.



- **Truth Table of T flip-flop**

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

- **Characteristics equation of T flip-flop**

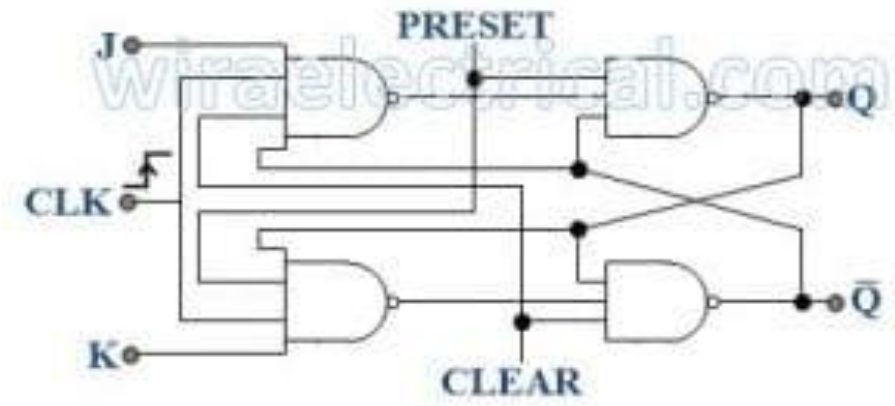
$Q(t+1)=T'Q(t)+TQ(t)'=T\oplus Q(t)$$Q(t+1)=T'Q(t)+TQ(t)'=T\oplus Q(t)$

Applications of Flip-flops

- Counters
- Shift Registers
- Storage Registers, etc.

JK Flip Flop with PRESET and CLEAR Inputs

Often we need to CLEAR the flip flop to logic state “0” ($Q_n = 0$) or PRESET it to logic state “1” ($Q_n = 1$). There is an example in the figure below. It will show how we do it.

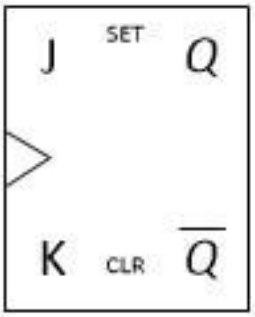


JK flip flop with PRESET and CLEAR

There are two conditions:

- The flip flop will be cleared ($Q_n = 0$) if we give logic state “0” to the CLEAR inputs and logic state “1” to the PRESET input.
- The flip flop is in preset logic state “1” condition ($Q_n = 1$) if we give logic state “1” to the CLEAR inputs and logic state “0” to the PRESET inputs.

Here, the PRESET and CLEAR inputs are active when low.



JK flip flop with PRESET and CLEAR

The image above is the circuit symbol of clocked JK flip flop which is presettable and clearable.

PR	CL	CLK	J	K	Q_{n+1}	$\overline{Q_{n+1}}$
0	1	X	X	X	1	0
1	0	X	X	X	0	1
0	0	X	X	X	-	-
1	1	1	0	0	Q_n	$\overline{Q_n}$
1	1	1	1	0	1	0
1	1	1	0	1	0	1
1	1	1	1	1	Toggle	
1	1	0	X	X	Q_n	$\overline{Q_n}$

The truth table of JK flip flop with PRESET and CLEAR

The table above is the truth table of JK flip flop with PRESET and CLEAR.

From the table, we conclude that, if the PRESET input is active, the output changes to logic state “1” regardless of the status of the clock, J, and K inputs.

Otherwise, if the CLEAR input is active, the output changes to logic state “0” regardless of the status of the clock, J, and K inputs.

There is an exception for this JK flip flop with PRESET and CLEAR: both of the PRESET and CLEAR inputs should not be activated at the same time.

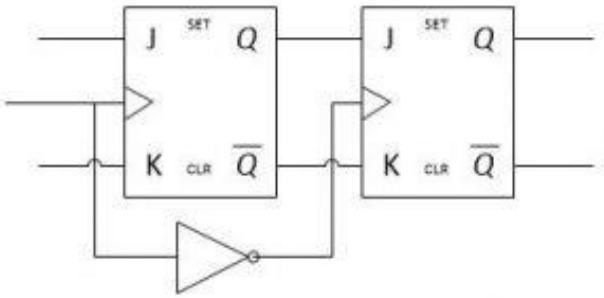
Master Slave of JK Flip Flop

When the width of the clock pulse of the flip flop is greater than the delay of the flip flop’s propagation, the change of the flip flop’s output is not reliable.

To overcome this problem, we will use the pulse generated by the edge-triggered flip flop. This pulse generated by the edge-detector portion of the flip flop would be the trigger, instead of the pulse width generated by the clock input signal.

This phenomenon is referred to as a race problem. Because the propagation delay is usually very small, the likelihood of race conditions occurring is quite high.

The most known solution to solve this problem is to use the slave-master flip flop configuration.



Master-slave J-K flip flop

Above is the master-slave J-K flip flop built with two J-K flip flops. There are two parts of this type of flip flop:

- The first flip flop = the master flip flop
- The second flip flop = the slave flip flop

The clock signal input will be complemented to the slave flip flop, while the master receives the clock input signal directly.

The operation steps of this master-slave J-K flip flop are:

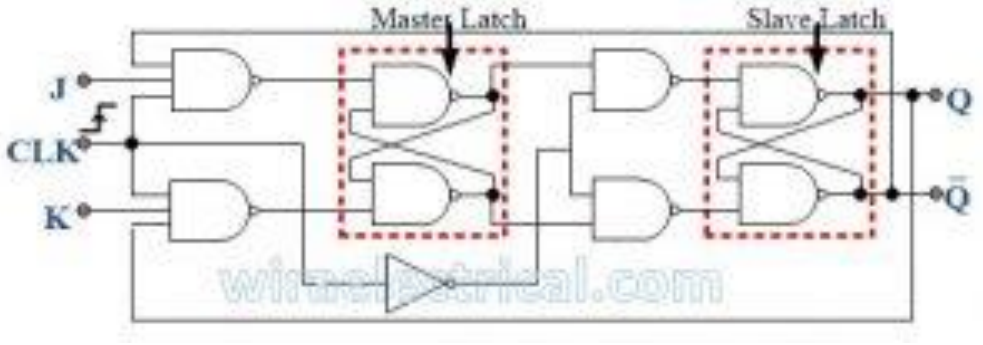
1. The clock signal pulse is HIGH,
2. The master flip flop is enabled, but the slave flip flop is disabled,
3. As the result, the master flip flop is able to change its output logic state, but the slave flip flop is unable,
4. The clock signal pulse is LOW,
5. The master flip flop is disabled, but the slave flip flop is enabled,
6. Hence, the logic state of the slave J-K flip flop changes as per logic state J-K logic inputs.
7. The logic state of the master flip flop is transferred to the slave flip flop, and the disabled master flip flop can acquire new inputs without affecting the output.

From the steps above, it should be clear that a master-slave flip flop is a pulse-triggered flip flop, not an edge-triggered flip flop.

The table below will show us the truth table of a master-slave J-K flip flop along with active LOW PRESET and CLEAR inputs, and also the active HIGH J and K inputs.

But, the master-slave J-K flip flop has become obsolete. The modern IC such as 74LS, 74AL, 74ALS, 74HC, and 74HCT don't have master-slave flip flops in their series.

Below we will observe how the master-slave of J-K flip flop works using its circuit diagram.



Master Slave J-K flip flop

Both input signals J, K, and clock input are connected to the "master" R-S flip flop which is able to lock the inputs when the clock input 'CLK' signal is HIGH or at logic state "1".

The CLK signal is complemented as the timing pulse for the "slave" R-S flip flop. This will make both flip flops work alternately.

Looking from the circuit diagram above, we can conclude the steps as:

1. CLK is HIGH or at logic state "1"
2. CLK input is at logic state "1" for the "master" and "0" for the "slave"
3. The inputs of the "master" are locked, but the outputs are only seen by the "slave" flip flop.
4. CLK is LOW or at logic state "0"
5. CLK input is at logic state "0" for the "master" and "1" for the "slave"

6. The outputs from the “master” latched and the flip flop does not read any inputs.
7. The “slave” flip flop is reading its input from the transferred outputs from the “master”

It is quite interesting that the “LOW to HIGH” transition of the clock input signal will play a huge role in this J-K flip flop.

’LOW to HIGH’: the “master” will transfer its outputs. This transition is complemented to the “slave” as ‘HIGH to LOW’ and makes the inputs processed by the “slave”.

This timing operation makes this flip flop as edge or pulse-triggered.

The flip flop receives input logic state when the CLK is HIGH and sends the data to the output when the clock signal is in falling-edge.

Hence, we can assume that the Master-Slave J-K flip flop is a “Synchronous” electric device because it only sends data at specific clock input timing.

The Drawback of J-K Flip Flop

The main and the only drawback of the J-K flip flop has been mentioned above, the Race Around Condition. This problem occurs when the J and K inputs are in logic state “1”.

The race around condition is when the output toggles the outputs more than one time after the output is complemented once.

If this problem happens, it will be very difficult to predict the next outputs. Assume if we give J and K a logic state “1”, in the next clock pulse the output will toggle.

What will happen if the J and K remain same at logic state “1”?

The output will toggle one more time and continue the pattern 0101010 in real scenario.. We need the master slave J-K flip flop in order to prevent this drawback.

We also need the clock interval is less than the delay propagation of the flip flop. If this is not achieved, the inputs won't be able to read the inputs before the clock pulse changes.

Counters

A **Counter** is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal. Counters are used in digital electronics for counting purpose, they can count specific event happening in the circuit. For example, in UP counter a counter increases count for every rising edge of clock. Not only counting, a counter can follow the certain sequence based on our design like any random sequence 0,1,3,2… .They can also be designed with the help of flip flops. They are used as frequency dividers where the frequency of given pulse waveform is divided. Counters are sequential circuit that count the number of pulses can be either in binary code or BCD form. The main properties of a counter are timing , sequencing , and counting. Counter works in two modes

Up counter

Down counter

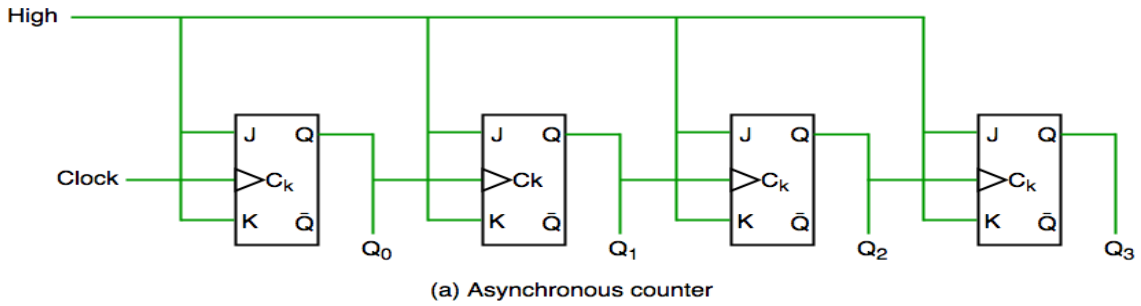
Counter Classification

Counters are broadly divided into two categories

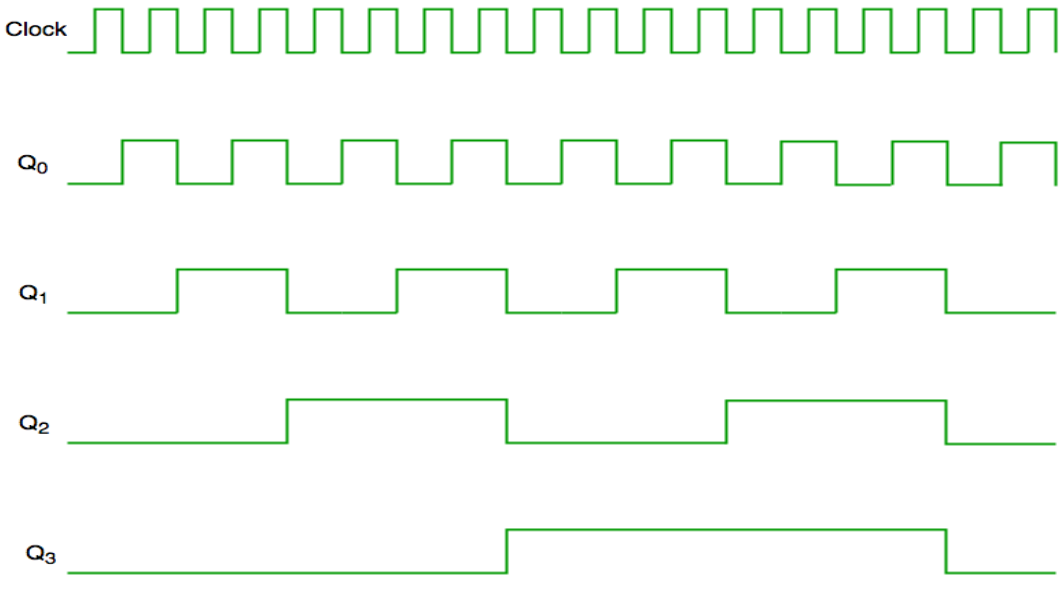
1. Asynchronous counter
2. Synchronous counter

1. **Asynchronous Counter**

In asynchronous counter we don't use universal clock, only first flip flop is driven by main clock and the clock input of rest of the following flip flop is driven by output of previous flip flops. We can understand it by following diagram-



(a) Asynchronous counter

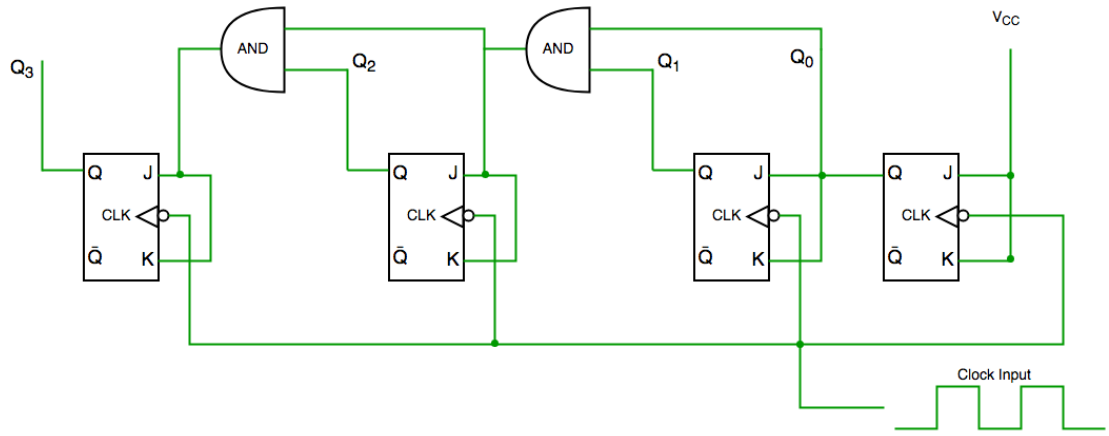


(b) Timing Diagram

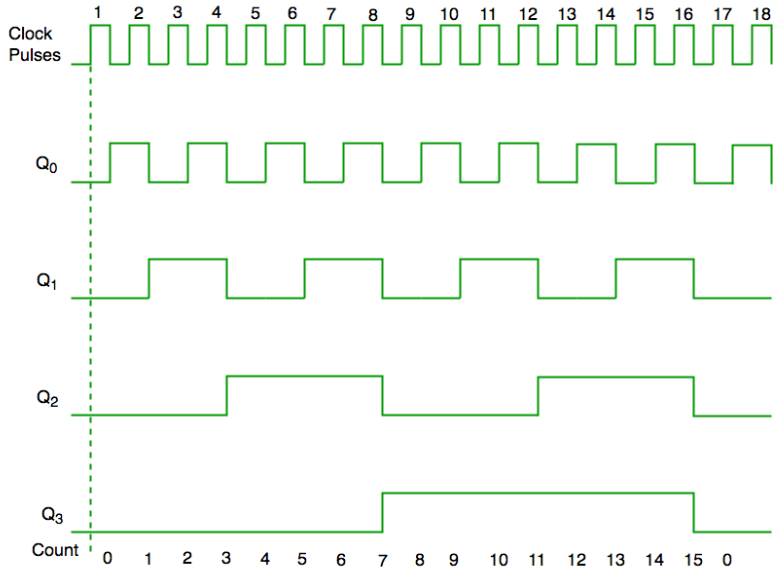
It is evident from timing diagram that Q0 is changing as soon as the rising edge of clock pulse is encountered, Q1 is changing when rising edge of Q0 is encountered(because Q0 is like clock pulse for second flip flop) and so on. In this way ripples are generated through Q0,Q1,Q2,Q3 hence it is also called **RIPPLE counter and serial counter.** A ripple counter is a cascaded arrangement of flip flops where the output of one flip flop drives the clock input of the following flip flop

2. **Synchronous Counter**

Unlike the asynchronous counter, synchronous counter has one global clock which drives each flip flop so output changes in parallel. The one advantage of synchronous counter over asynchronous counter is, it can operate on higher frequency than asynchronous counter as it does not have cumulative delay because of same clock is given to each flip flop. It is also called as parallel counter.



Synchronous counter circuit



Timing diagram synchronous counter

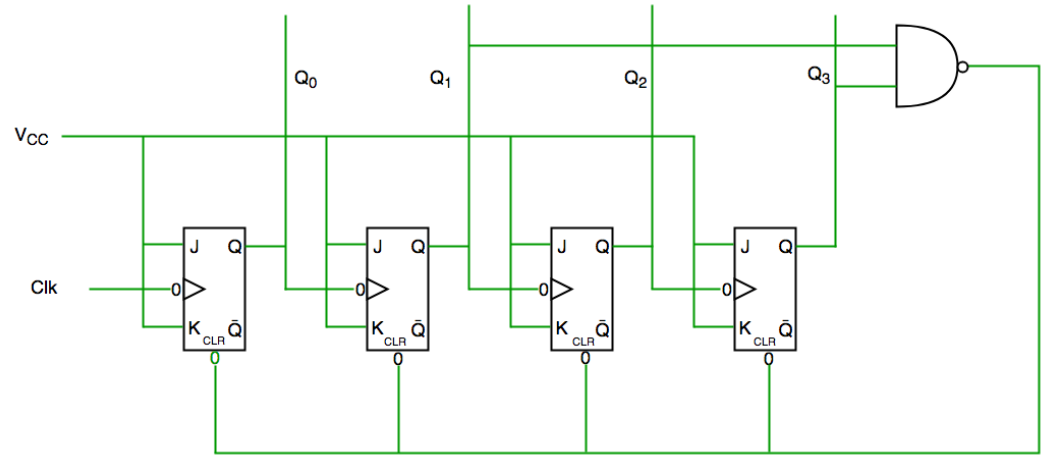
From circuit diagram we see that Q0 bit gives response to each falling edge of clock while Q1 is dependent on Q0, Q2 is dependent on Q1 and Q0 , Q3 is dependent on Q2,Q1 and Q0.

Decade Counter

A decade counter counts ten different states and then reset to its initial states. A simple decade counter will count from 0 to 9 but we can also make the decade counters which can go through any ten states between 0 to 15(for 4 bit counter).

Clock pulse	Q3	Q2	Q1	Q0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	0	0	0	0

Truth table for simple decade counter



Decade counter circuit diagram

We see from circuit diagram that we have used nand gate for Q3 and Q1 and feeding this to clear input line because binary representation of 10 is—

1010

And we see Q3 and Q1 are 1 here, if we give NAND of these two bits to clear input then counter will be clear at 10 and again start from beginning.

Important point: Number of flip flops used in counter are always greater than equal to ($\log_2 n$) where n=number of states in counter.

Binary Counter

In digital electronics, a **binary counter** is a type of sequential logic circuit which is able to count in binary numbers. A binary counter can counter from 0 to 2(n-1), where n is the total number of bits in the counter.

Basically, a binary counter is a type of digital circuit which counts the number of clock pulses that occur over a time period.

The binary counters are built up of flip flops, where a flip flop is a most elementary memory element that can store 1-bit of information. In a binary counter, each flip flop represents one bit of the binary number. The counter increases its count by one whenever a clock pulse occurs.

For example, a 3-bit binary counter can count from 000 (0) to 111 (7) before wrapping around to 000 again. We can design a binary counter to count up or down. Also, a binary counter has more advanced features such as ability to reset the count to zero, to load a specific count, etc.

Now, let us discuss different types of binary counters.

Types of Binary Counters

There are many types of binary counters present. Some common types of binary counters are defined as follows −

- **Asynchronous Counter** − The type of binary counter in which the flip flops do not receive the same clock pulse at the same time is called an asynchronous counter. The asynchronous counter is also known as **ripple counter**. It is the simplest type of binary counter. In the case of asynchronous binary counter, each flip flop is triggered by the output of the previous flip flop. Therefore, the asynchronous counters suffer from propagation delay.
- **Synchronous Counter** − The type of binary counter in which all the flip flops receive the same clock pulse at the same time is known as a synchronous counter. Since, all the flip flops of the synchronous counter are triggered by the same clock pulse, therefore, their outputs change simultaneously. This will result in the no propagation delay between the flip flops.
- **Up Counter** − The type of binary counter that counts upwards from zero to its maximum count value is known as up counter. In the case of up counter, the count is increased by one on each clock pulse.
- **Down Counter** − The type of binary counter that counts downwards from its maximum count value to zero is known as a down counter. In the down counter, the count value of the counter is decreased by one on each clock pulse.
- **Up/Down Counter** − The type of binary counter that can count in both upward and downward directions is known as a up/down counter. In the up/down counter, the direction of count is determined by a control input signal.

Design of Binary Counter

The general procedure that is followed for designing a binary counter is described in the following steps −

Step (1) – Determine the count range

Firstly, we have to determine the count range of the counter. It involves determining the minimum and maximum values that our binary counter has to count. This count range depends upon the application requirements.

Step (2) – Select the number of bits

In this step, we have to select the number of bits required for the counter. The number of bits depends upon the requirement of the count range. The count range of a binary counter is given by 2n, where n is the number of bits. For example, a 3-bit counter can count up to 8 different values, ranging from 000 (0) to 111 (7).

Step (3) – Select a proper counter type

Depending on the application requirements, select a proper type of counter. The chosen counter should have speed and accuracy required by the application. For example, asynchronous counters are used in simple and less expensive applications, whereas synchronous counters are used in applications where timing is critical.

Step (4) – Select the flip-flops

In this step, we have to choose the flip-flops used to implement the counter. The selected flip flops must be able to handle the desired count range and the clock frequency. In binary counters, the D flip flop is most commonly used type of the flip flop.

Step (5) – Write the excitation table and derive the minimal expression

Write the excitation table of the flip-flop for the counter as per the given state diagram. And derive the minimal expressions using K-map.

Step (6) – Design the counter circuit and test it

Connect the flip flops as per the expressions and test the counter to ensure that it counts correctly and reliably.

Advantages of Binary Counter

The major advantages of binary counters are listed as follows −

- Binary counters have high accuracy, i.e. they count number of clock pulses that occur over a time accurately.
- Binary counters consume less power because they are generally designed by using low power logic gates and flip flops.
- Binary counters are easy to design as they can be realized by using standard logic gates and flip flops.
- Binary counters have fast response. Therefore, they can operate at high clock frequencies.
- Binary counters are reliable. Thus, they can work for a long period of time without requiring maintenance.
- Binary counter is a versatile device as it can be used in a wide range of applications such as frequency dividers, digital clocks, etc.

Limitations of Binary Counters

The following are the major limitations of binary counters −

- Binary counter can count in a limited range, where the maximum limit of the counter is determined by the number of bits.
- Binary counter produces an output signal in the binary form which is limited to some applications and for more applications, it requires an additional circuitry to convert it in a suitable form.
- Binary counters are susceptible to electronic noise which may cause errors in the counting.

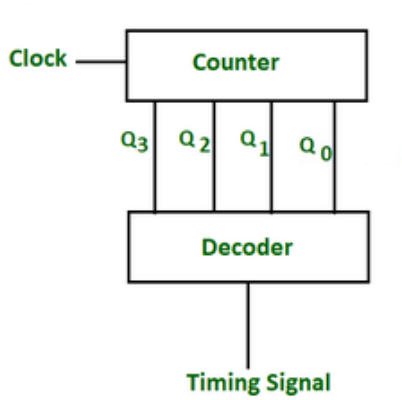
Applications of Binary Counters

Binary counters are used in numerous digital systems. Some common applications of binary counters are listed below.

- Binary counters are used in digital clocks and other digital timing devices.
- Binary counter can be used as a frequency divider, where it divides the frequency of the input signal by a fixed value.
- Binary counter can also be used as a shift register.
- In digital systems like computers, binary counters can be used as memory address decoders.
- Binary counter can also be used as a sequence generator, where it can generate sequences of binary codes.
- Binary counters can be used in error detection and correction applications.

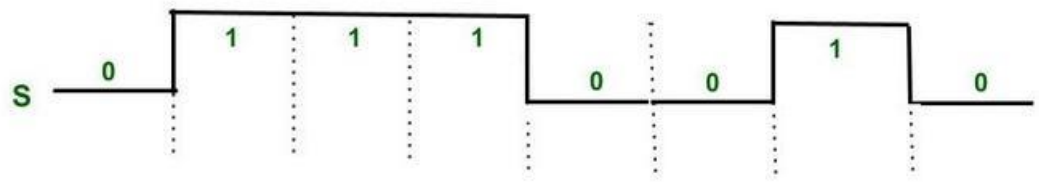
Control Signals Generation using Counter

- In various digital applications(For example : hardwired control unit) control signals are needed to start, execute and step various operations in a particular time sequence.
- For this control signals are required and for the generation of control signals, a counter circuit is designed whose outputs are connected to a decoder. The decoder provides the required control signal.
- The counter can be synchronous or asynchronous.
- The procedure for designing the counter is the same refer (this).



Block diagram for control signal generation

The design of control signal can be understood by considering the example.
Example – Generate a control signal which can deliver the following pulse train. The pulse train will repeat after 7 pulses.



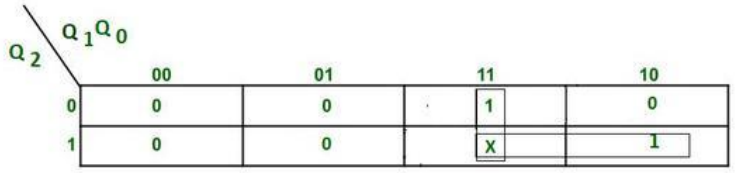
Control Signal

Here we need to generate a control signal (say S) which can generate periodic pulse train of **0111001** and then repeats. The pulse train repeats after 7 seven pulses. Therefore, **mod -7 counter is needed.** The outputs of Mod-7 counter will be connected to a decoder circuit. For this a mod –7 counter is to be designed. Three T flips are required(because we need to count up to 000 to 110, therefore 3 bits are needed).
For the design for Mod — N counter please refer this article.
Design for mod-7 counter –
Here Q is previous state and Q* is next state.

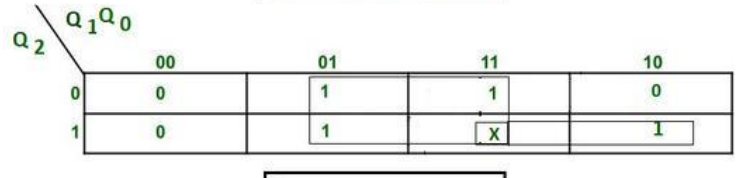
Q_2	Q_1	Q_0	Q_2^*	Q_1^*	Q_0^*	T_2	T_1	T_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	0	0	0	1	1	0

Circuit excitation table for mod- 7 Counter

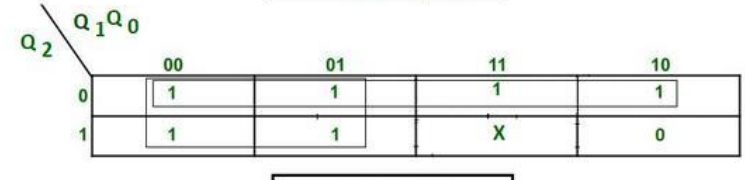
The expressions for inputs of T flip-flops obtained from the K – maps shown below.



$T_2 = Q_1Q_0 + Q_2Q_1$



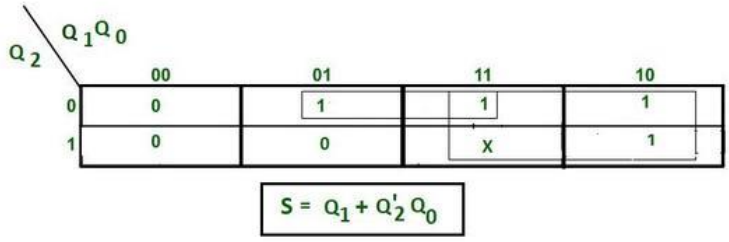
$T_1 = Q_0 + Q_2Q_1$



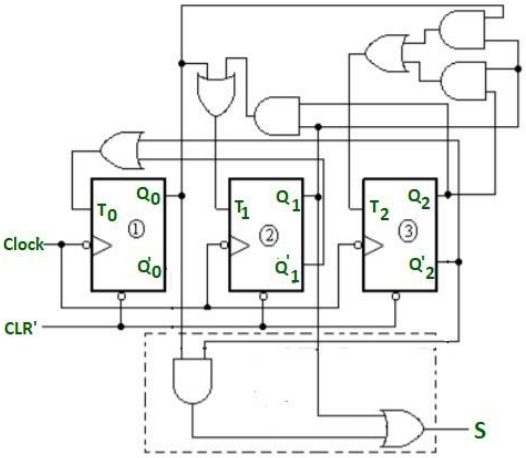
$T_0 = Q'_2 + Q'_1$

The truth table for the **decoder is obtained by observing of the given timing sequence(0111001**). The simplified expression for the output S of decoder is obtained using the K-map. The not used counts are considered as don't care.
Each counting sequence is mapped to one control signal bit.
The combinational logic of decoder can be found by solving K map.

Q_2	Q_1	Q_0	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1



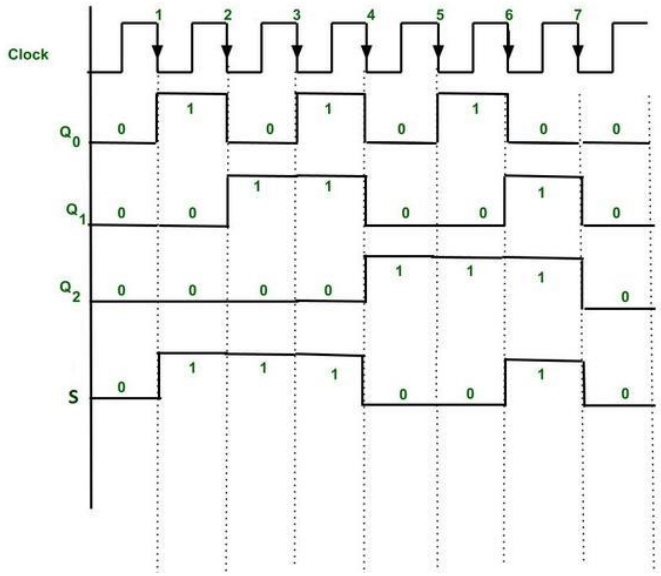
Simplified expression for decoder circuit is $S= Q_1+ Q'_2Q_0$.
Example — When we are in state $Q_2 =0$ $Q_1= 0$ $Q_0 = 0$, then the value of S = 0+0.1= 0.



Complete logic diagram with Decoder.

The decoder circuit generates output after every clock(-ve edge triggered) pulse. The output i.e S will be act as Control signal for other circuitry.

The generation of a control signal takes place after each negative edge clock and timing diagram for such control signal may also be drawn as shown below. Each counter state is used for generating one control signal.



Timing diagram for Control signal S generation by counter

Up-Down Counter

Counters are used in many different applications. Some count up from zero and provide a change in state of output upon reaching a predetermined value; others count down from a preset value to zero to provide an output state change.

However, some counters can operate in both up and down count mode, depending on the state of an up/down count mode input pin. They can be reversed at any point within their count sequence. Dual purpose ICs such as the TTL 74LS190 and 75LS191 are available which implement both Up and Down count functions.

The TTL 74LS190 is a 4-bit device that can be switched between Up and Down modes, and provides a BCD decade output; the 74LS191 is a binary counter. The counters are synchronous, but they are asynchronously presettable. Four data inputs (A – D) allow the preset target to be loaded. The counter is decremented or incremented synchronously with the low to high transition of the clock. The counters can be cascaded in high-speed mode.

A simple three-bit Up/Down synchronous counter can be built using JK flip-flops configured to operate as toggle or T-type flip-flops giving a maximum count of zero (000), advancing through 001, 010 to seven (111) and back to zero again.

Shift Register

A group of flip flops which is used to store multiple bits of data and the data is moved from one flip flop to another is known as **Shift Register**. The bits stored in registers shifted when the clock pulse is applied within and inside or outside the registers. To form an n-bit shift register, we have to connect n number of flip flops. So, the number of bits of the binary number is directly proportional to the number of flip flops. The flip flops are connected in such a way that the first flip flop's output becomes the input of the other flip flop.

A **Shift Register** can shift the bits either to the left or to the right. A **Shift Register**, which shifts the bit to the left, is known as **"Shift left register"**, and it shifts the bit to the right, known as **"Right left register"**.

The shift register is classified into the following types:

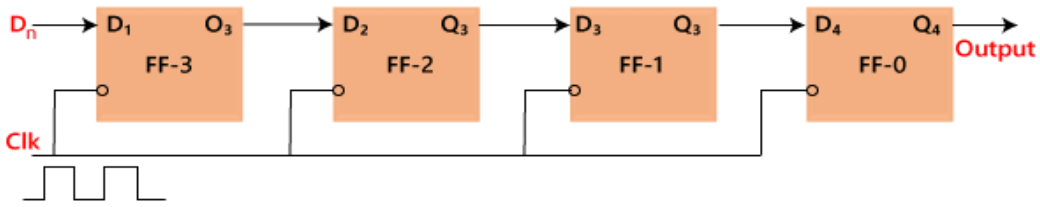
- Serial In Serial Out
- Serial In Parallel Out
- Parallel In Serial Out
- Parallel In Parallel Out
- Bi-directional Shift Register
- Universal Shift Register

Serial IN Serial OUT

In "Serial Input Serial Output", the data is shifted "IN" or "OUT" serially. In SISO, a single bit is shifted at a time in either right or left direction under clock control.

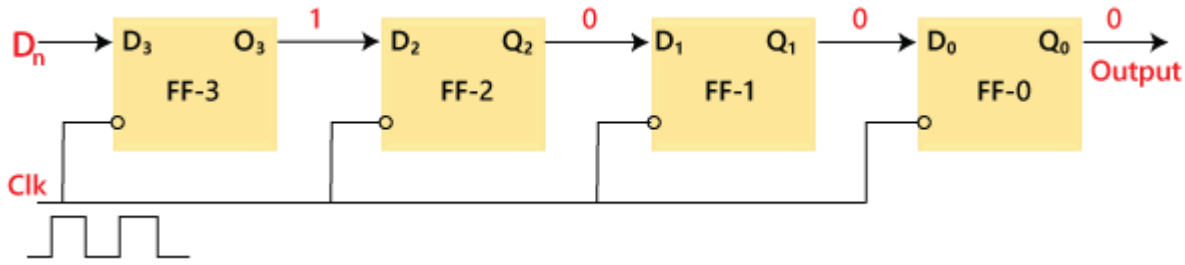
Initially, all the flip-flops are set in "reset" condition i.e. $Y_3 = Y_2 = Y_1 = Y_0 = 0$. If we pass the binary number 1111, the LSB bit of the number is applied first to the Din bit. The D3 input of the third flip flop, i.e., FF-3, is directly connected to the serial data input D3. The output Y_3 is passed to the data input d_2 of the next flip flop. This process remains the same for the remaining flip flops. The block diagram of the **"Serial IN Serial OUT"** is given below.

Block Diagram:

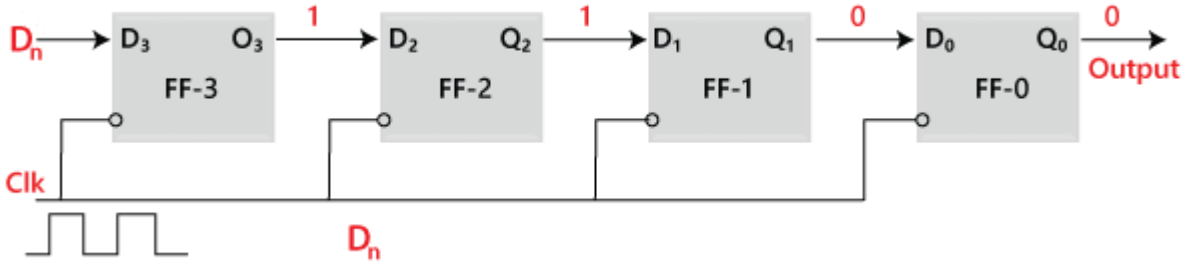


Operation

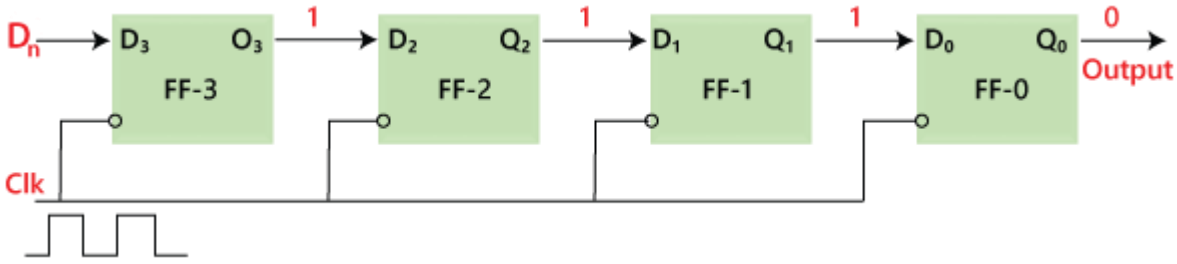
When the clock signal application is disabled, the outputs $Y_3\ Y_2\ Y_1\ Y_0 = 0000$. The LSB bit of the number is passed to the data input D_{in}, i.e., D_3. We will apply the clock, and this time the value of D_3 is 1. The first flip flop, i.e., FF-3, is set, and the word is stored in the register at the first falling edge of the clock. Now, the stored word is 1000.



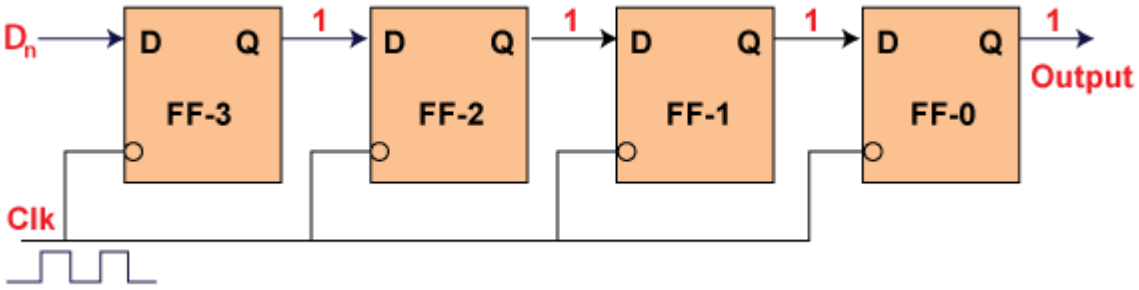
The next bit of the binary number, i.e., 1, is passed to the data input D_2. The second flip flop, i.e., FF-2, is set, and the word is stored when the next negative edge of the clock hits. The stored word is changed to 1100.



The next bit of the binary number, i.e., 1, is passed to the data input D_1, and the clock is applied. The third flip flop, i.e., FF-1, is set, and the word is stored when the negative edge of the clock hits again. The stored word is changed to 1110.



Similarly, the last bit of the binary number, i.e., 1, is passed to the data input D_0, and the clock is applied. The last flip flop, i.e., FF-0, is set, and the word is stored when the clock's negative edge arrives. The stored word is changed to 1111.

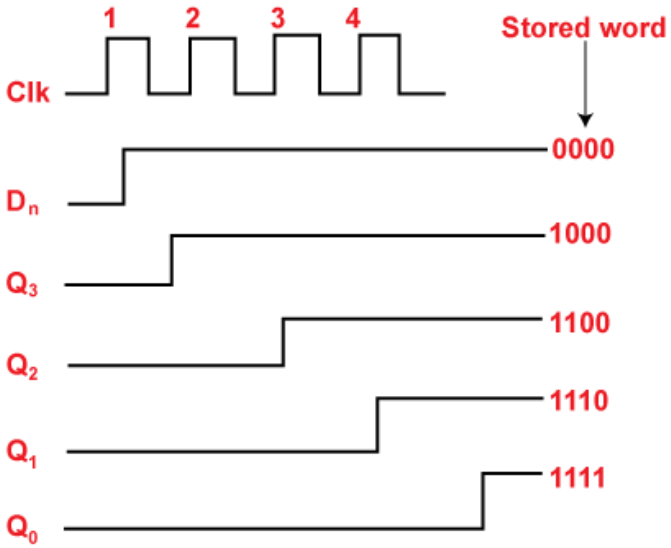


Truth Table

	Clk	$D_n=Q_3$	$Q_3=D_2$	$Q_2=D_1$	$Q_1=D_0$	Q_0
Initially			0	0	0	0
(1)	↓	1	1	0	0	0
(2)	↓	1	1	1	0	0
(3)	↓	1	1	1	1	0
(4)	↓	1	1	1	1	1

→ Direction of data travel

Waveforms

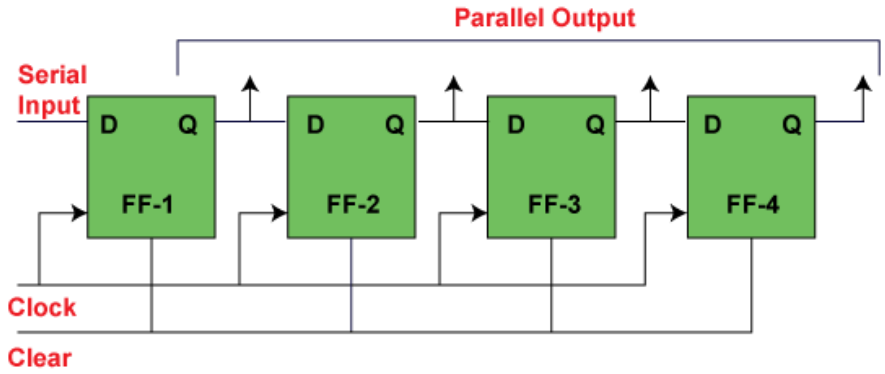


Serial IN Parallel OUT

In the **"Serial IN Parallel OUT"** shift register, the data is passed serially to the flip flop, and outputs are fetched in a parallel way. The data is passed bit by bit in the register, and the output remains disabled until the data is not passed to the data input. When the data is passed to the register, the outputs are enabled, and the flip flops contain their return value

Below is the block diagram of the 4-bit **serial in the parallel-out** shift register. The circuit having four D flip-flops contains a clear and clock signal to reset these four flip flops. In **SIPO**, the input of the second flip flop is the output of the first flip flop, and so on. The same clock signal is applied to each flip flop since the flip flops synchronize each other. The parallel outputs are used for communication.

Block Diagram



Parallel IN Serial OUT

In the **"Parallel IN Serial OUT"** register, the data is entered in a parallel way, and the outcome comes serially. A four-bit **"Parallel IN Serial OUT"** register is designed below. The input of the flip flop is the output of the previous Flip Flop. The input and outputs are connected through the combinational circuit. Through this combinational circuit, the binary input B_0, B_1, B_2, B_3 are passed. The **shift mode** and the **load mode** are the two modes in which the **"PISO"** circuit works.

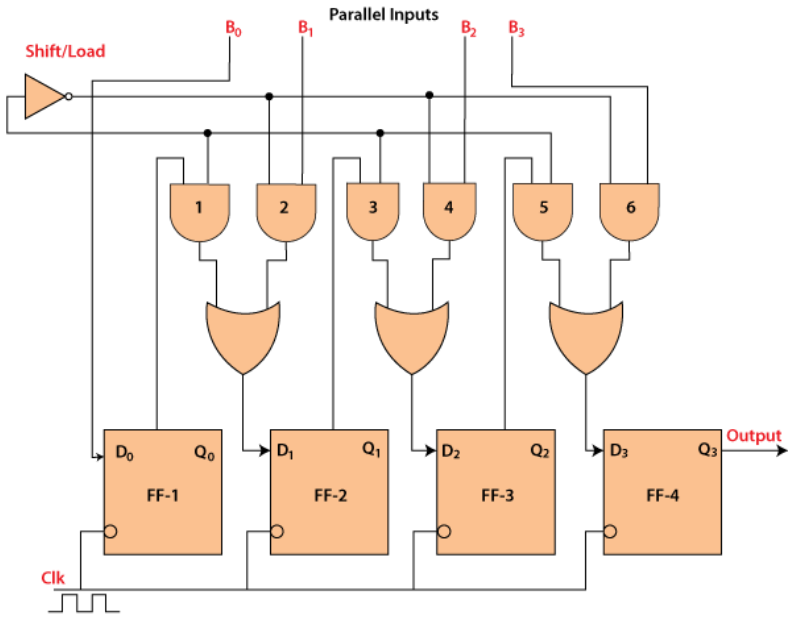
Load mode

The bits B_0, B_1, B_2, and B_3 are passed to the corresponding flip flops when the second, fourth, and sixth "AND" gates are active. These gates are active when the shift or load bar line set to 0. The binary inputs B0, B1, B2, and B3 will be loaded into the respective flip-flops when the edge of the clock is low. Thus, parallel loading occurs.

Shift mode

The second, fourth, and sixth gates are inactive when the load and shift line set to 0. So, we are not able to load data in a parallel way. At this time, the first, third, and fifth gates will be activated, and the shifting of the data will be left to the right bit. In this way, the **"Parallel IN Serial OUT"** operation occurs.

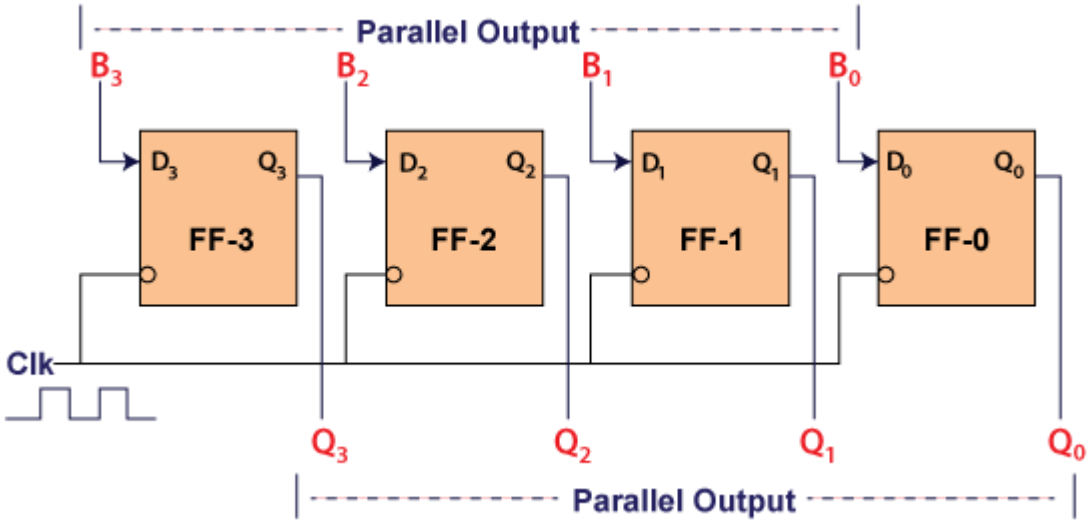
Block Diagram



Parallel IN Parallel OUT

In **"Parallel IN Parallel OUT"**, the inputs and the outputs come in a parallel way in the register. The inputs A_0, A_1, A_2, and A_3, are directly passed to the data inputs D_0, D_1, D_2, and D_3 of the respective flip flop. The bits of the binary input is loaded to the flip flops when the negative clock edge is applied. The clock pulse is required for loading all the bits. At the output side, the loaded bits appear.

Block Diagram



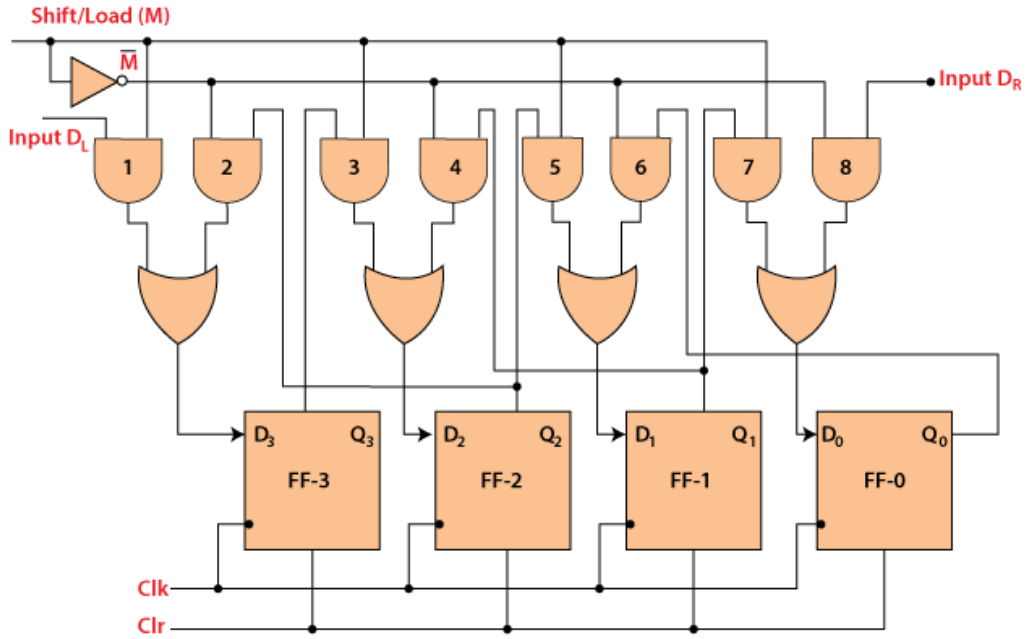
Bidirectional Shift Register

The binary number after shifting each bit of the number to the left by one position will be equivalent to the number produced by multiplying the original number by 2. In the same way, the binary number after shifting each bit of the number to the right by one position will be equivalent to the number produced by dividing the original number by 2.

For performing the multiplication and division operation using the shift register, it is required that the data should be moved in both the direction, i.e., left or right in the register. Such registers are called the **"Bidirectional"** shift register.

Below is the diagram of 4-bit **"bidirectional"** shift register where $\mathbf{D_R}$ is the **"serial right shift data input"**, $\mathbf{D_L}$ is the **"left shift data input"**, and M is the **"mode select input"**.

Block Diagram



Operations

1) Shift right operation(M=1)

- The first, third, fifth, and seventh AND gates will be enabled, but the second, fourth, sixth, and eighth AND gates will be disabled.
- The data present on the data input **DR** is shifted bit by bit from the fourth flip flop to the first flip flop when the clock pulse is applied. In this way, the shift right operation occurs.

2) Shift left operation(M=0)

- The second, fourth, sixth and eighth AND gates will be enabled, but the AND gates first, third, fifth, and seventh will be disabled.
- The data present on the data input DR is shifted bit by bit from the first flip flop to the fourth flip flop when the clock pulse is applied. In this way, the shift right operation occurs.

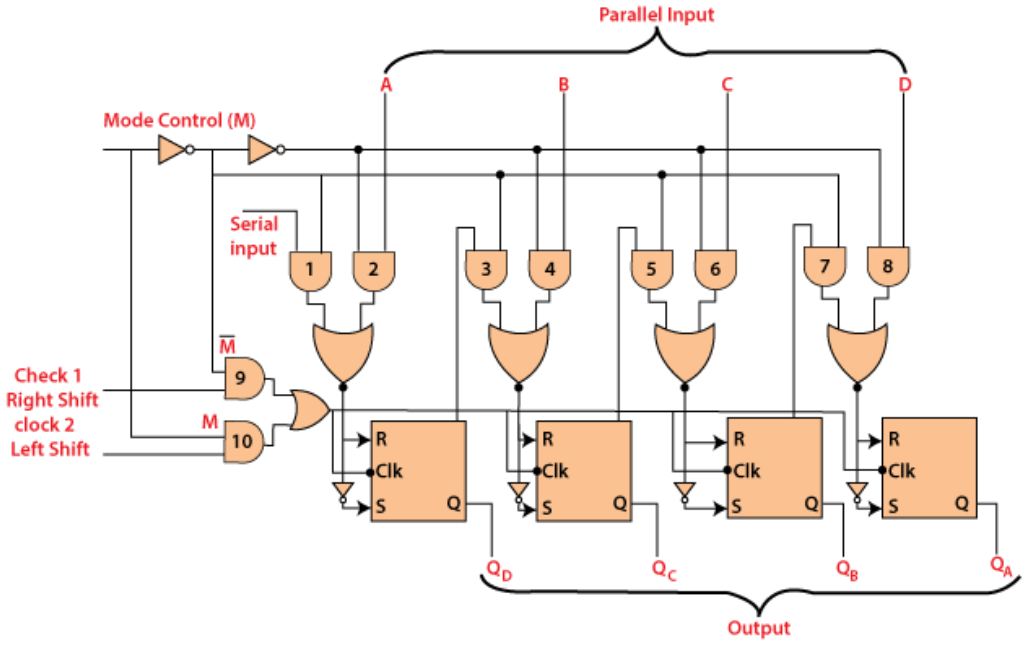
Universal Shift Register

A register where the data is shifted in one direction is known as the **"uni-directional"** shift register. A register in which the data is shifted in both the direction is known as **"bi-**

directional" shift register. A **"Universal"** shift register is a special type of register that can load the data in a parallel way and shift that data in both directions, i.e., right and left.

The input M, i.e., the mode control input, is set to 1 to perform the parallel loading operation. If this input set to 0, then the serial shifting operation is performed. If we connect the mode control input with the ground, then the circuit will work as a **"bi-directional"** register. The diagram of the universal shift register is given below. When the input is passed to the **serial input**, the register performs the "serial left" operation. When the input is passed to the input **D**, the register performs the serial right operation.

Block Diagram



What is Parity Generator and Parity Checker : Types & Its Logic Diagrams

The parity generator and parity checker's main function is to detect errors in data transmission and this concept is introduced in 1922. In RAID technology the parity bit and the parity checker are used to guard against data loss. The parity bit is an extra bit that is set at the transmission side to either '0' or '1', it is used to detect only single bit error and it is the easiest method for detecting errors. There are different types of error detection codes used to detect the errors they are parity, ring counter, block parity code, Hamming code, biquinary, etc. The brief explanation about parity bit, parity generator and checker are explained below.

What is Parity Bit?

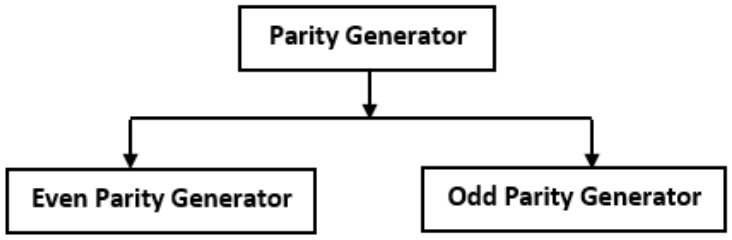
Definition: The parity bit or check bit are the bits added to the binary code to check whether the particular code is in parity or not, for example, whether the code is in even parity or odd parity is checked by this check bit or parity bit. The parity is nothing but number of 1's and there are two types of parity bits they are even bit and odd bit.
In odd parity bit, the code must be in an odd number of 1's, for example, we are taking 5-bit code 100011, this code is said to be odd parity because there is three number of 1's in the code which we have taken. In even parity bit the code must be in even number of 1's, for example, we are taking 6-bit code 101101, this code is said to be even parity because there are four number of 1's in the code which we have taken

What is the Parity Generator?

Definition: The parity generator is a combination circuit at the transmitter, it takes an original message as input and generates the parity bit for that message and the transmitter in this generator transmits messages along with its parity bit.

Types of Parity Generator

The classification of this generator is shown in the below figure



types-of-parity-generator

Even Parity Generator

The even parity generator maintains the binary data in even number of 1's, for example, the data taken is in odd number of 1's, this even parity generator is going to maintain the data as even number of 1's by adding the extra 1 to the odd number of 1's. This is also a combinational circuit whose output is dependent upon the given input data, which means the input data is binary data or binary code given for parity generator.

Let us consider three input binary data, that three bits are considered as A, B, and C. We can write 2^3 combinations using the three input binary data that is from 000 to 111 (0 to 7), total eight combinations will get from the given three input binary data which we have considered. The truth table of even parity generator for three input binary data is shown below.

0 0 0 – In this input binary code the even parity is taken as '0' because the input is already in even parity, so no need to add even parity once again for this input.

0 0 1 – – In this input binary code there is only a single number of '1' and that single number of '1' is an odd number of '1'. If an odd number of '1' is there, then even parity generator must generate another '1' to make it as even parity, so even parity is taken as 1 to make the 0 0 1 code into even parity.

0 1 0 – This bit is in odd parity so even parity is taken as 1 to make the 0 1 0 code into even parity.

0 1 1 – This bit is already in even parity so even parity is taken as 0 to make the 0 1 1 code into even parity.

1 0 0 – This bit is in odd parity so even parity is taken as 1 to make the 1 0 0 code into even parity.

1 0 1 – This bit is already in even parity so even parity is taken as 0 to make the 1 0 1 code into even parity.

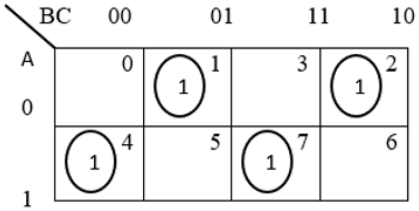
1 1 0 – This bit is also in even parity so even parity is taken as 0 to make the 1 1 0 code into even parity.

1 1 1 – This bit is in odd parity so even parity is taken as 1 to make the 1 1 1 code into even parity.

Even Parity Generator Truth Table

A B C	Even Parity
0 0 0	0
0 0 1	1
0 1 0	1
0 1 1	0
1 0 0	1
1 0 1	0
1 1 0	0
1 1 1	1

The karnaugh map (k-map) simplification for three-bit input even parity is



k-map-for-even-parity-generator

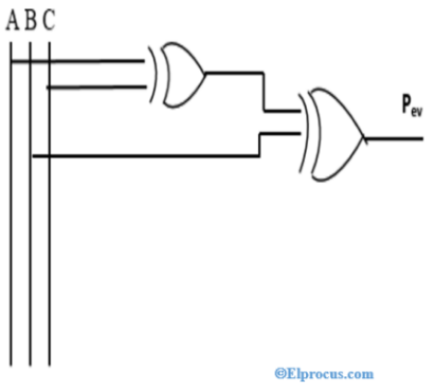
From the above even parity truth table, the parity bit simplified expression is written as

$$P_{ev} = (A\,\bar{B}\,\bar{C} + \bar{A}\,\bar{B}\,C) + (A\,B\,C + \bar{A}\,B\,\bar{C})$$

$$= \bar{B}\,(A\,\bar{C} + \bar{A}\,C) + B\,(A\,C + \bar{A}\,\bar{C})$$

$$= \bar{B}\,(A \oplus C) + B\,(\overline{A \oplus C}) = \bar{B}x + B\bar{x} = \mathrm{B} \oplus \mathrm{x} = \mathrm{B} \oplus \mathrm{A} \oplus \mathrm{C}$$

The even parity expression implemented by using two Ex-OR gates and the logic diagram of this even parity using the Ex-OR logic gate is shown below.



even-parity-logic-circuit

In this way, the even parity generator generates an even number of 1's by taking the input data.

Odd Parity Generator

The odd parity generator maintains the binary data in an odd number of 1's, for example, the data taken is in even number of 1's, this odd parity generator is going to maintain the data as an odd number of 1's by adding the extra 1 to the even number of 1's. This is the combinational circuit whose output is always dependent upon the given input data. If there is an even number of 1's then only parity bit is added to make the binary code into an odd number of 1's.

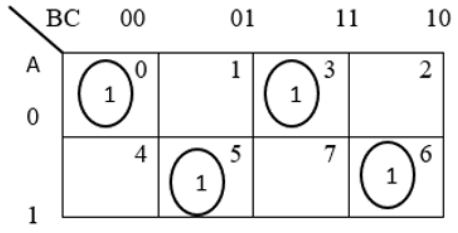
Let us consider three input binary data, that three bits are considered as A, B, and C. The truth table of odd parity generator for three input binary data is shown below.

0 0 0 – In this input binary code the odd parity is taken as '1' because the input is in even parity.
0 0 1 – This binary input is already in odd parity, so odd parity is taken as 0.
0 1 0 – This binary input is also in odd parity, so odd parity is taken as 0.
0 1 1 – This bit is in even parity so odd parity is taken as 1 to make the 0 1 1 code into odd parity.
1 0 0 – This bit is already in odd parity, so odd parity is taken as 0 to make the 1 0 0 code into odd parity.
1 0 1 – This input bit is in even parity, so odd parity is taken as 1 to make the 1 0 1 code into odd parity.
1 1 0 – This bit is in even parity, so odd parity is taken as 1.
1 1 1 – This input bit is in odd parity, so odd parity is taken as o.

Odd Parity Generator Truth Table

A B C	Odd Parity
0 0 0	1
0 0 1	0
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	0

The Kavanaugh map (k-map) simplification for three-bit input odd parity is



k-map-for-odd-parity-generator

From the above odd parity truth table, the parity bit simplified expression is written as

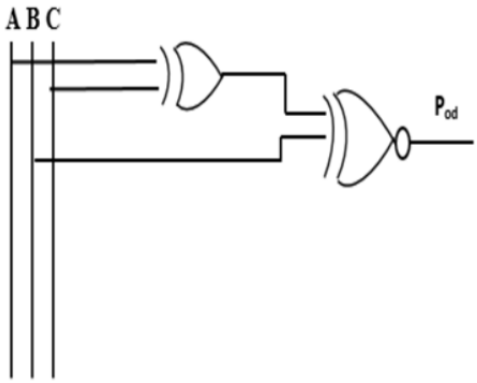
$$P_{od} = (\bar{A}\,\bar{B}\,\bar{C} + A\,\bar{B}\,C) + (\bar{A}\,B\,C + A\,B\,\bar{C})$$

$$= \bar{C}\,(A\,B + \bar{A}\,\bar{B}) + B\,(A\,\bar{B} + \bar{A}\,B)$$

$$= \bar{C}\,(A \oplus B) + C(A \oplus B)$$

$$= \bar{C}\,\bar{X} + C\,X = C \odot X = C \odot \overline{A \oplus B}$$

The logic diagram of this odd parity generator is shown below.



logic-circuit

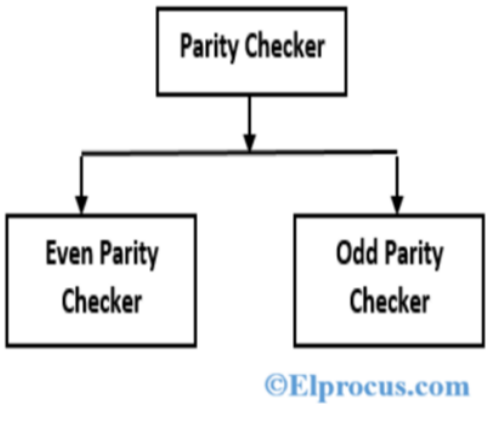
In this way, the odd parity generator generates an odd number of 1's by taking the input data.

What is the Parity Check?

Definition: The combinational circuit at the receiver is the parity checker. This checker takes the received message including the parity bit as input. It gives output '1' if there is some error found and gives output '0' if no error is found in the message including the parity bit.

Types of Parity Checker

The classification of the parity checker is shown in the below figure



types-of-parity-checker

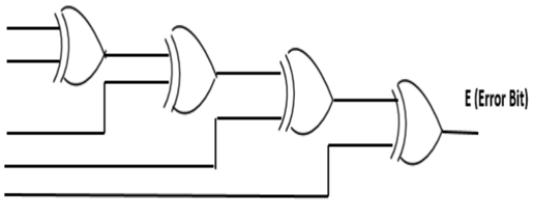
Even Parity Checker

In even parity checker if the error bit (E) is equal to '1', then we have an error. If error bit E=0 then indicates there is no error.

Error Bit (E) =1, error occurs

Error Bit (E) =0, no error

The parity checker circuit is shown in the below figure



 logic-circuit

Odd Parity Checker

In odd parity checker if an error bit (E) is equal to '1', then it indicates there is no error. If an error bit E=0 then indicates there is an error.

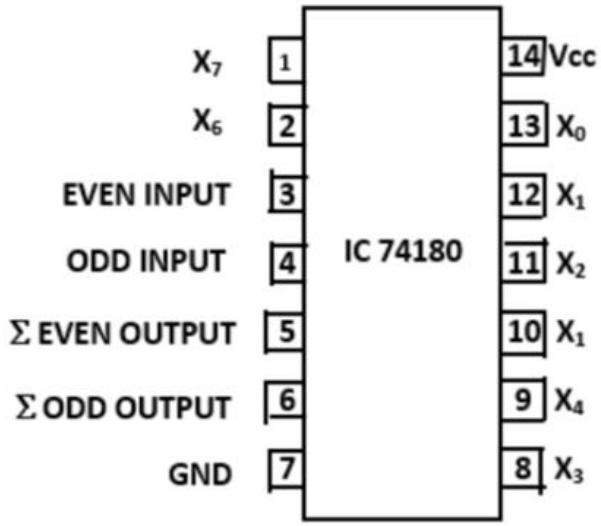
Error Bit (E) =1, no error

Error Bit (E) =0, error occurs

The parity checker won't be able to detect if there are errors in more than '1' bit and the correct of data is also not possible, these are the main disadvantages of the parity checker.

Parity Generator/Checker using IC's

The IC 74180 does the function of parity generation as well as checking. The 9 bit (8 data bits, 1 parity bit) Parity Generator/Checker is shown in the below figure.



ic-74180

The IC 74180 contains eight data bits (X_0 to X_7), $V_{cc,}$ even input, odd input, Seven output, S odd output, and ground pin.
If the given even and odd input both are high (H), then the even and odd outputs both are low (L), similarly, if the given inputs both are Low (L), then the even and odd outputs both becomes high (H).

Advantages of Parity

The advantages of parity are

- Simplicity
- Easy to use

Applications **of Parity**

The applications of parity are

- In digital systems and many hardware applications, this parity is used
- The parity bit is also used in Small Computer System Interface (SCSI) and also in Peripheral Component Interconnect (PCI) to detect the errors