**Experiment 1**

**AIM:** Implementing HALF ADDER, FULL ADDER using basic logic gates

**Theory:** The most basic arithmetic operation is the addition of two binary digits. There are four possible elementary operations, namely,

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 0 (with 1 as carry)

The first three operations produce a sum of whose length is one digit, but when the last operation is performed the sum is two digits. The higher significant bit of this result is called a carry and lower significant bit is called the sum.

**Half Adder**: A combinational circuit which performs the addition of two bits is called half adder. The input variables designate the augend and the addend bit, whereas the output variables produce the sum and carry bits.

**Block Diagram:**

HA

**X HS**

**Y CO**

**Truth Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.No | Input | | Output | |
| X | Y | HS | CO |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 1 | 0 |
| 3 | 1 | 0 | 1 | 0 |
| 4 | 1 | 1 | 0 | 1 |

**Logic Diagram:**

****

**Full Adder**: A combinational circuit which performs the arithmetic sum of three input bits is called full adder. The three input bits include two significant bits and a previous carry bit. A full adder circuit can be implemented with two half adders and one OR gate.

**Block Diagram:**



**Truth Table:**

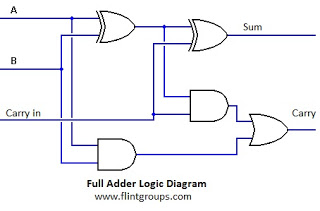
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Inputs** | | | **Outputs** | |
| **X** | **Y** | Cin | **S** | **Cout** |
| **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **1** | **1** | **0** |
| **0** | **1** | **0** | **1** | **0** |
| **0** | **1** | **1** | **0** | **1** |
| **1** | **0** | **0** | **1** | **0** |
| **1** | **0** | **1** | **0** | **1** |
| **1** | **1** | **0** | **0** | **1** |
| **1** | **1** | **1** | **1** | **1** |

From the truth table, the expression for sum and carry bits of the output can be obtained as,

SUM = A’B’C + A’BC’ + AB’C’ + ABC= ABC

CARRY = A’BC + AB’C + ABC’ +ABC=AB+(AB)C

**Logic Diagram:**



**Simulation:-**

Using simulator produced by IIT-Khadagpur we can simulator to implement half adder & full adder can verify results/truth table of half/full adder.

1. Half Adder Simulation :- First make Half adder using logic gates in editor window then connect input and output display element for verifying truth table of adder as shown in screenshot of simulator edit window.

1. **Full Adder Simulation**:- Make full adder using logic gates in simulator editor window as shown below in screenshot and verify truth table of full adder by simulating diagram.

**Result:-** Implementation of HALF ADDER & FULL ADDER using basic logic gates has been done in simulator.

Questions:

Q.1 What is a half Adder?

Q.2 What is a Full Adder

Q.3 Make full adder with two Half Adder.

## EXPERIMENT NO. – 2

**EXPERIMENT NAME:** Implementing Binary -to -Gray, Gray -to -Binary code conversions.

**PROBLEM STATEMENT:** In this we have to convert binary number into gray code and vice versa.

**OBJECTIVE:** . **:** Implementing Binary -to -Gray, Gray -to -Binary code conversions. Using COA Simulator

### SOFTWARE REQUIRED:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sr. no. | Name of  equipment’s/components/software |  | Specification/range/rating/version | Quantity |
| 1 | COA SIMULATOR BY IIT KHARAGPUR |  |  |  |

**Definition of Binary and Gray code:**

**Gray code** – also known as **Cyclic Code**, **Reflected Binary Code** (RBC), **Reflected Binary** (RB) or **Grey code** – is defined as an ordering of the binary number system such that each incremental value can only differ by one bit. In gray code, while traversing from one step to another step only one bit in the code group changes. That is to say that two adjacent code numbers differ from each other by only one bit.

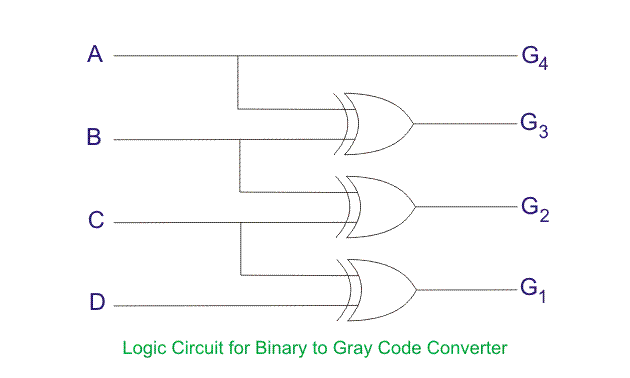
Gray code is the most popular of the unit distance codes, but it is not suitable for arithmetic operations. Gray code has some applications in analog to digital converters, as well as being used for error correction in digital communication. Gray code can be difficult to understand initially, but becomes much easier to understand when looking at the gray code tables below.

## Binary to Gray Code Converter

The logical circuit which converts the binary code to equivalent gray code is known as **binary to gray code converter**. An n-bit gray code can be obtained by reflecting an n-1 bit code about an axis after 2n-1 rows and putting the MSB (Most Significant Bit) of 0 above the axis and the MSB of 1 below the axis. Reflection of Gray codes is shown below.

The 4 bit binary to gray code conversion table is given below:

|  |  |  |
| --- | --- | --- |
| Decimal | Binary Number | Gray code |
| 0 | 0000 | 0000 |
| 1 | 0001 | 0001 |
| 2 | 0010 | 0011 |
| 3 | 0011 | 0010 |
| 4 | 0100 | 0110 |
| 5 | 0101 | 0111 |
| 6 | 0110 | 0101 |
| 7 | 0111 | 0100 |
| 8 | 1000 | 1100 |
| 9 | 1001 | 1101 |
| 10 | 1010 | 1111 |
| 11 | 1011 | 1110 |
| 12 | 1100 | 1010 |
| 13 | 1101 | 1011 |
| 14 | 1110 | 1001 |
| 15 | 1111 | 1000 |

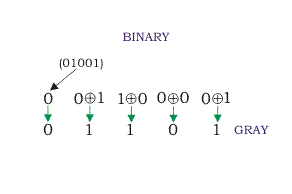


### How to Convert Binary to Gray Code

1. The MSB (Most Significant Bit) of the gray code will be exactly equal to the first bit of the given binary number.
2. The second bit of the code will be exclusive-or (XOR) of the first and second bit of the given binary number, i.e if both the bits are same the result will be 0 and if they are different the result will be 1.
3. The third bit of gray code will be equal to the exclusive-or (XOR) of the second and third bit of the given binary number. Thus the binary to gray code conversion goes on. An example is given below to illustrate these steps.

### Binary to Gray Code Conversion Example

1. The MSB is kept the same. As the MSB of the binary is 0, the MSB of the gray code will be 0 as well (first gray bit)
2. Next, take the XOR of the first and the second binary bit. The first bit is 0, and the second bit is 1. The bits are different so the resultant gray bit will be 1 (second gray bit)
3. Next, take the XOR of the second and third binary bit. The second bit is 1, and the third bit is 0. These bits are again different so the resultant gray bit will be 1 (third gray bit)
4. Next, take the XOR of third and fourth binary bit. The third bit is 0, and the fourth bit is 0. As these are the same, the resultant gray bit will be 0 (fourth gray bit)
5. Lastly, take the XOR of the fourth and fifth binary bit. The fourth bit is 0, and the fifth bit is 1. These bits are different so the resultant gray bit will be 1 (fifth gray bit)
6. Hence the result of binary to gray code conversion of 01001 is complete, and the equivalent gray code is 01101.

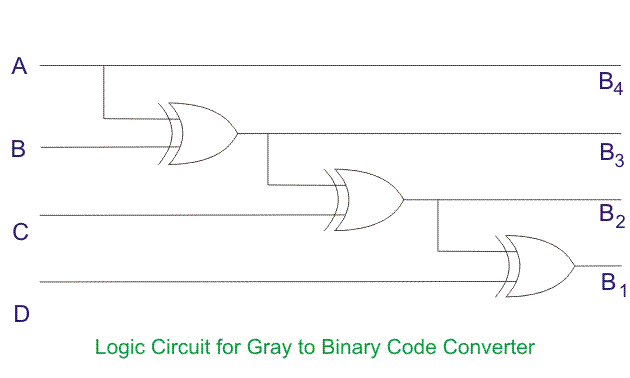


## Gray to Binary Code Converter

In a **gray to binary code converter**, the input is gray code and output is its equivalent binary code.

Gray to binary Code Conversion Table

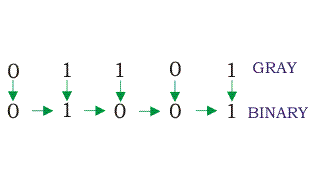
|  |  |  |
| --- | --- | --- |
| Decimal Number | Gray code | Binary code |
| 0 | 0000 | 0000 |
| 1 | 0001 | 0001 |
| 2 | 0011 | 0010 |
| 3 | 0010 | 0011 |
| 4 | 0110 | 0100 |
| 5 | 0111 | 0101 |
| 6 | 0101 | 0110 |
| 7 | 0100 | 0111 |
| 8 | 1100 | 1000 |
| 9 | 1101 | 1001 |
| 10 | 1111 | 1010 |
| 11 | 1110 | 1011 |
| 12 | 1010 | 1100 |
| 13 | 1011 | 1101 |
| 14 | 1001 | 1110 |
| 15 | 1000 | 1111 |



### Gray Code to Binary Conversion

**Gray code to binary conversion** is again a very simple and easy process. Following steps can make your idea clear on this type of conversions.

1. The MSB of the binary number will be equal to the MSB of the given gray code.
2. Now if the second gray bit is 0, then the second binary bit will be the same as the previous or the first bit. If the gray bit is 1 the second binary bit will alter. If it was 1 it will be 0 and if it was 0 it will be 1.
3. This step is continued for all the bits to do **Gray code to binary conversion**.



Some other applications of gray code:

* Boolean circuit minimization
* Communication between clock domains
* Error correction
* Genetic algorithms
* Mathematical puzzles
* Position encoders

## Advantages of Gray Code

* Better for error minimization in converting analog signals to digital signals
* Reduces the occurrence of “Hamming Walls” (an undesirable state) when used in genetic algorithms
* Can be used to in to minimize a logic circuit
* Useful in clock domain crossing

## Disadvantages of Gray Code

* Not suitable for arithmetic operations
* Limited practical use outside of a few specific applications

Questions for Practice:

Design binary to gray code converter using Universal Gate Nand.

1. Design Binary to gray code converter using Universal Gate Nor.
2. Design gray to binary code converter using Universal Gates Nand and Nor.

Questions:

Q.1 What is a code converter?

Q.2 Differentiate between translator and code converter.

Q.3 Explain the primary usage of grey code.

Q.4 Illustrate the reasons for using grey code.

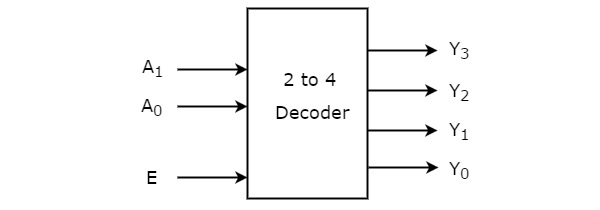
**Experiment 3**

**AIM:-** Implementing 3-8 line Decoder.

**Theory: -** Decoder is a combinational circuit that has ‘n’ input lines and maximum of 2n output lines. One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code. The outputs of the decoder are nothing but the **min terms** of ‘n’ input variables lines, when it is enabled.

## 2 to 4 Decoder

Let 2 to 4 Decoder has two inputs A1 & A0 and four outputs Y3, Y2, Y1 & Y0. The **block diagram** of 2 to 4 decoder is shown in the following figure.



One of these four outputs will be ‘1’ for each combination of inputs when enable, E is ‘1’. The **Truth table** of 2 to 4 decoder is shown below.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Enable | Inputs | | Outputs | | | |
| E | A1 | A0 | Y3 | Y2 | Y1 | Y0 |
| 0 | x | x | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |

From Truth table, we can write the **Boolean functions** for each output as

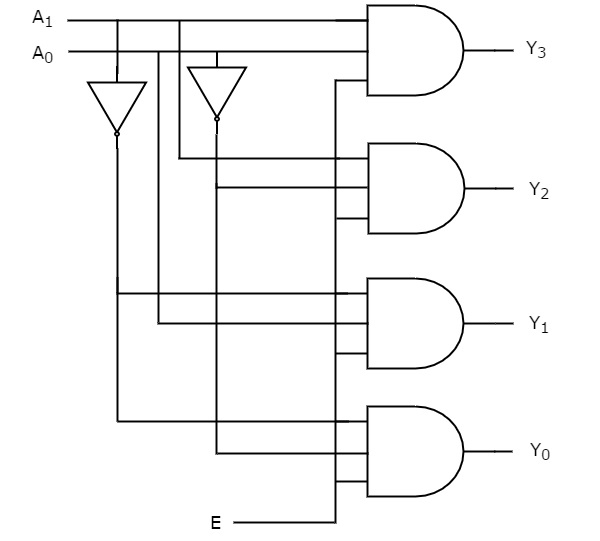
Y0=EA’1.A’0

Y1=EA’1.A0

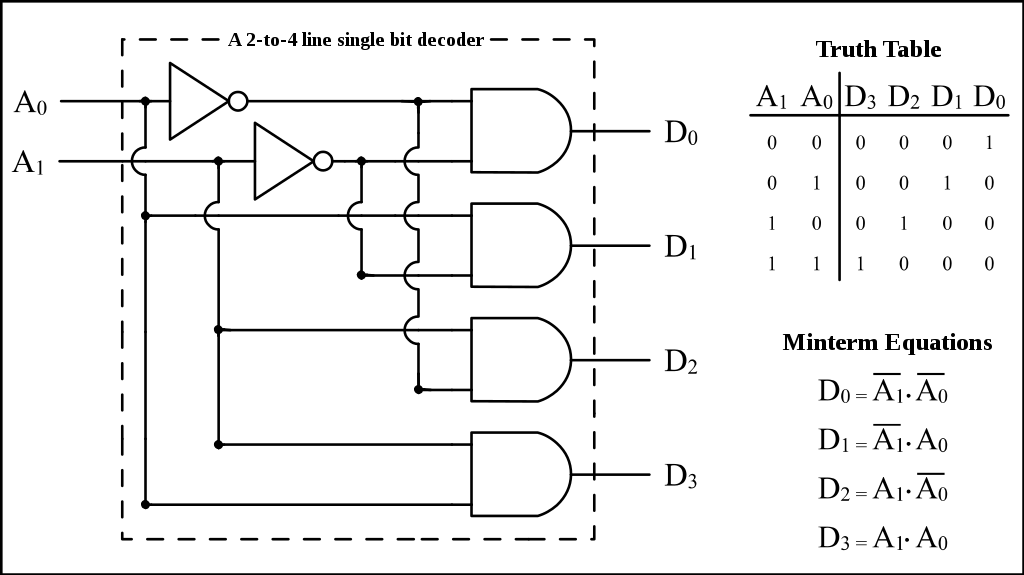
Y2=EA1.A’0

Y3=EA1.A0

Each output is having one product term. So, there are four product terms in total. We can implement these four product terms by using four AND gates having three inputs each & two inverters. The **circuit diagram** of 2 to 4 decoder is shown in the following figure.



By neglecting Enable bit we can design 2-4 Decoder as shown below-



Therefore, the outputs of 2 to 4 decoder are nothing but the **min terms** of two input variables A1 & A0, when enable, E is equal to one. If enable, E is zero, then all the outputs of decoder will be equal to zero.

Similarly, 3 to 8 decoder produces eight min terms of three input variables A2, A1 & A0 and 4 to 16 decoder produces sixteen min terms of four input variables A3, A2, A1 & A0.

### **3 to 8 Decoder**

Similarly 3 to 8 Decoder has three inputs A2, A1 & A0 and eight outputs, Y7 to Y0.

We can find the number of lower order decoders required for implementing higher order decoder using the following formula.

The **Truth table** of 3 to 8 decoder is shown below.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Enable | Inputs | | | Outputs | | | | | | | |
| E | A2 | A1 | A0 | Y7 | Y6 | Y5 | Y4 | Y3 | Y2 | Y1 | Y0 |
| 0 | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

From Truth table, we can write the **Boolean functions** for each output as

Y0=E A’2.A’1.A’0

Y1=E A’2A’1.A0

Y2=E A’2A1.A’0

Y3=E A’2A1.A0

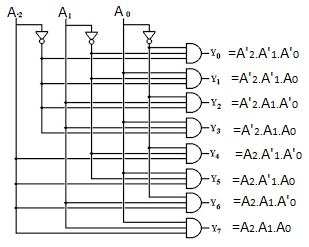
Y4=E A2A’1.A’0

Y5=E A2A’1.A0

Y6=E A2A1.A’0

Y7=E A2A1.A0

By ignoring Enable bit we can design 3-8 line Decoder as shown in Diagram below-



**Simulation:- 2-4 Decoder & 3-8 Decoders are simulated in simulator, screenshots are given below as-**

## EXPERIMENT NO. 4

**EXPERIMENT NAME: Implementing 4\*1 and 8\*1 Multiplexers**

**PROBLEM STATEMENT: Design 4\*1 and 8\*1 Multiplexers using COA simulator.**

## 

**OBJECTIVE:** To implement the working of 4\*1 and 8\*1 Multiplexer.

### APPARATUS REQUIRED:

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. no. | Name of  Equipments/components/software | Specification/range/rating/ version | Quantity |
| 1 | COA Simulator by IIT Kharagpur |  |  |

Multiplexers:

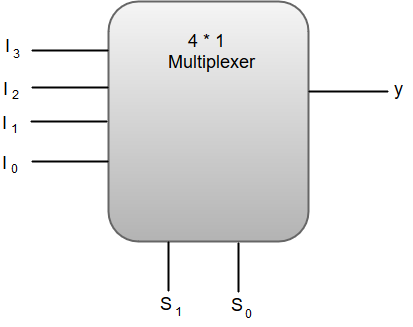
A Multiplexer (MUX) can be described as a combinational circuit that receives binary information from one of the 2^n input data lines and directs it to a single output line.

The selection of a particular input data line for the output is decided on the basis of selection lines.

The multiplexer is often called as data selector since it selects only one of many data inputs.

Note: A 2^n-to-1 multiplexer has 2^n input data lines and n input selection lines whose bit combinations determine which input data are selected for the output.

Block diagram of 4\*1 Multiplexer



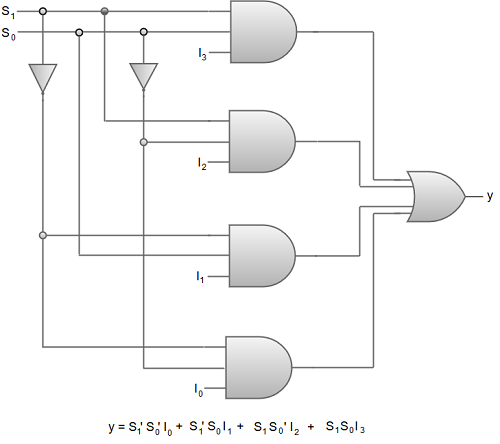
Out of these four input data lines, a particular input data line will be connected to the output based on the combination of inputs present at these two selection lines.

Truth Table of 4\*1 Multiplexer

|  |  |  |
| --- | --- | --- |
| S1 | S0 | Y |
| 0 | 0 | I0 |
| 0 | 1 | I1 |
| 1 | 0 | I2 |
| 1 | 1 | I3 |

From the function table, we can write the Boolean function for the output (y) as:

y = S1'S0'I0 + S1' S0'I1 + S1S0'I2 + S1S0I3



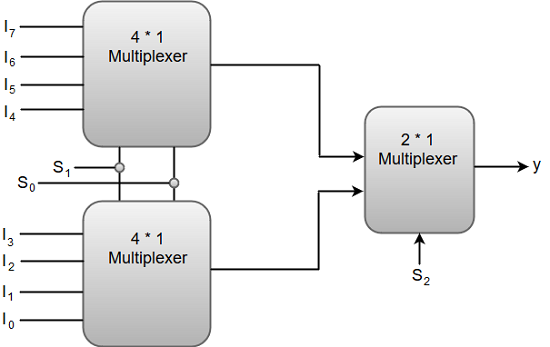
Digital Circuit Diagram of 4\*1 Multiplexer.

We can also implement higher order multiplexers using lower order multiplexers. For instance, let us implement an **8 \*1 multiplexer** using two 4\*1 multiplexers and a 2\*1 multiplexer.

The two 4\*1 multiplexers are required in the first stage to get the eight input data lines.

A 2\*1 multiplexer is required in the second stage to converge the outputs generated at first stage into a single output.

The following image shows the block diagram of an 8\*1 multiplexer designed using two 4\*1 multiplexers and a single 2\*1 multiplexer.



|  |  |  |  |
| --- | --- | --- | --- |
|  | **S1** | **S0** | **y** |
| 0 | 0 | 0 | 10 |
| 0 | 0 | 1 | I1 |
| 0 | 1 | 0 | I2 |
| 0 | 1 | 1 | I3 |
| 1 | 0 | 0 | I4 |
| 1 | 0 | 1 | I5 |
| 1 | 1 | 0 | I6 |
| 1 | 1 | 1 | 17 |

Function Table of 8\*1 Multiplexer.

Questions for Practice:

* 1. Design 4\*1 MUX using Universal Gates Nand and Nor.
  2. Design 8\*1 MUX using Universal Gates Nand and Nor.

Questions:

1. Describe Multiplexers.
2. State the Applications of Multiplexers.
3. List Applications and usage of Multiplexers.

**Experiment 5**

**AIM:-** Verify the excitation tables of various FLIP-FLOPS

**Theory:** Logic circuits for digital systems are either combinational or sequential. The output of combinational circuits depends only on the current inputs. In contrast, sequential circuit depends not only on the current value of the input but also upon the internal state of the circuit. Basic building blocks (memory elements) of a sequential circuit are the flip-flops (FFs). A flip-flop is a device which stores a single *bit* (binary digit) of data; one of its two states represents a "one" and the other represents a "zero". Such data storage can be used for storage of *state*, and such a circuit is described as sequential logic

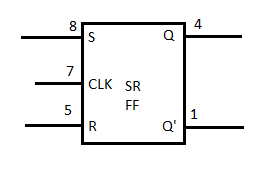
There are four Flip-Flops as follows-

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

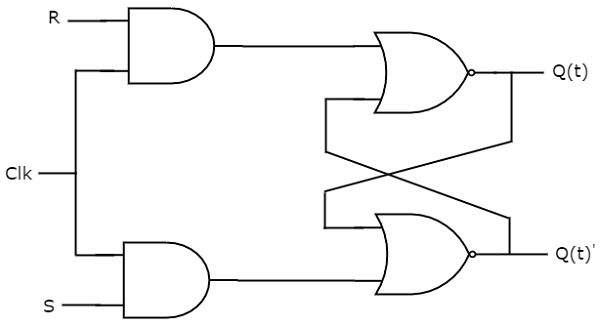
## SR Flip-Flop

SR flip-flop operates with only positive clock transitions or negative clock transitions. Whereas, SR latch operates with enable signal.

**Block Diagram of SR Flip Flop & Pin Diagram:**

****

The **circuit diagram** of SR flip-flop is shown in the following figure.



This circuit has two inputs S & R and two outputs Qt & Qt’. The operation of SR flipflop is similar to SR Latch. But, this flip-flop affects the outputs only when positive transition of the clock signal is applied instead of active enable.

The following table shows the **state table** of SR flip-flop.

|  |  |  |
| --- | --- | --- |
| **S** | **R** | **Q**t+1 |
| 0 | 0 | Qt |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | - |

Here, Qt & Qt+1 are present state & next state respectively. So, SR flip-flop can be used for one of these three functions such as Hold, Reset & Set based on the input conditions, when positive transition of clock signal is applied. The following table shows the **characteristic table** of SR flip-flop.

|  |  |  |  |
| --- | --- | --- | --- |
| **Present Inputs** | | **Present State** | **Next State** |
| **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 |

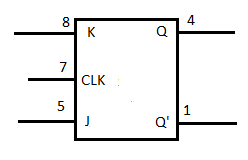
The **simplified expression** for next state Qt+1 is

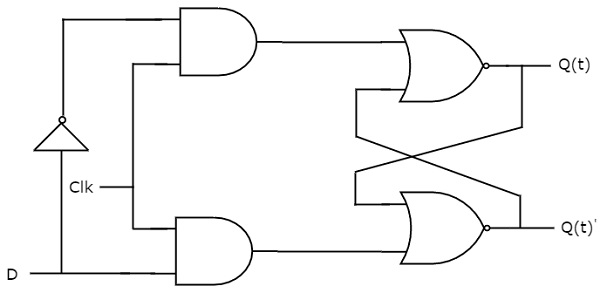
Q(t+1)=S+R′Q(t)Q(t+1)=S+R′Q(t)

## D Flip-Flop

D flip-flop operates with only positive clock transitions or negative clock transitions. Whereas, D latch operates with enable signal. That means, the output of D flip-flop is insensitive to the changes in the input, D except for active transition of the clock signal.

**Block Diagram of SR Flip Flop & Pin Diagram**



The **circuit diagram** of D flip-flop is shown in the following figure.

This circuit has single input D and two outputs Qtt & Qtt’. The operation of D flip-flop is similar to D Latch. But, this flip-flop affects the outputs only when positive transition of the clock signal is applied instead of active enable.

The following table shows the **state table** of D flip-flop.

|  |  |
| --- | --- |
| **D** | **Qt + 1t + 1** |
| 0 | 0 |
| 1 | 1 |

Therefore, D flip-flop always Hold the information, which is available on data input, D of earlier positive transition of clock signal. From the above state table, we can directly write the next state equation as

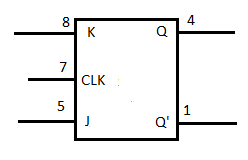
Qt+1t+1 = D

Next state of D flip-flop is always equal to data input, D for every positive transition of the clock signal. Hence, D flip-flops can be used in registers, **shift registers** and some of the counters.

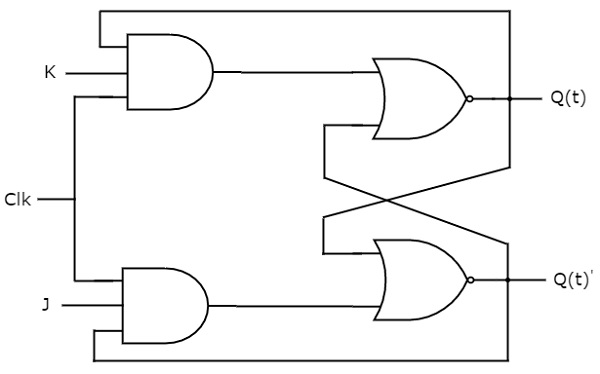
## JK Flip-Flop

JK flip-flop is the modified version of SR flip-flop. It operates with only positive clock transitions or negative clock transitions.

**Block Diagram of SR Flip Flop & Pin Diagram**



The **circuit diagram** of JK flip-flop is shown in the following figure.



This circuit has two inputs J & K and two outputs Qt & Qt’. The operation of JK flip-flop is similar to SR flip-flop. Here, we considered the inputs of SR flip-flop as **S = J Q**t**’** and **R = KQ**t in order to utilize the modified SR flip-flop for 4 combinations of inputs.

The following table shows the **state table** of JK flip-flop.

|  |  |  |
| --- | --- | --- |
| **J** | **K** | **Q**t+1 |
| 0 | 0 | Qt |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | Qt' |

Here, Qt& Qt+1 are present state & next state respectively. So, JK flip-flop can be used for one of these four functions such as Hold, Reset, Set & Complement of present state based on the input conditions, when positive transition of clock signal is applied. The following table shows the **characteristic table** of JK flip-flop.

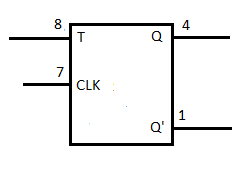
|  |  |  |  |
| --- | --- | --- | --- |
| **Present Inputs** | | **Present State** | **Next State** |
| **J** | **K** | **Q**t | **Q**t+1 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

The **simplified expression** for next state Qt+1 is

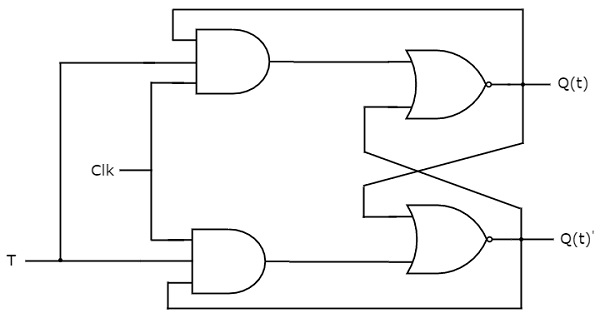
Q(t+1)=JQ(t)′+K′Q(t)Q(t+1)=JQ(t)′+K′Q(t)

## T Flip-Flop

T flip-flop is the simplified version of JK flip-flop. It is obtained by connecting the same input ‘T’ to both inputs of JK flip-flop. It operates with only positive clock transitions or negative clock transitions.



The **circuit diagram** of T flip-flop is shown in the following figure.



This circuit has single input T and two outputs Qtt & Qtt’. The operation of T flip-flop is same as that of JK flip-flop. Here, we considered the inputs of JK flip-flop as **J = T** and **K = T** in order to utilize the modified JK flip-flop for 2 combinations of inputs. So, we eliminated the other two combinations of J & K, for which those two values are complement to each other in T flip-flop.

The following table shows the **state table** of T flip-flop.

|  |  |
| --- | --- |
| **T** | **Q**t+1 |
| 0 | Qt |
| 1 | Qt’ |

Here, Qt & Qt+1 are present state & next state respectively. So, T flip-flop can be used for one of these two functions such as Hold, & Complement of present state based on the input conditions, when positive transition of clock signal is applied. The following table shows the **characteristic table** of T flip-flop.

|  |  |  |
| --- | --- | --- |
| **Inputs** | **Present State** | **Next State** |
| **T** | **Q**t | **Q**t+1 |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

From the above characteristic table, we can directly write the **next state equation** as

Q(t+1)=T′Q(t)+TQ(t)′Q(t+1)=T′Q(t)+TQ(t)′

⇒Q(t+1)=T⊕Q(t)⇒Q(t+1)=T⊕Q(t)

The output of T flip-flop always toggles for every positive transition of the clock signal, when input T remains at logic High 11. Hence, T flip-flop can be used in **counters**.

In this chapter, we implemented various flip-flops by providing the cross coupling between NOR gates. Similarly, you can implement these flip-flops by using NAND gates.