Topic 2 Notes

Discrete logic is a chip that contains one logic or a number of logic gates. **A logic gate is** a collection of transistors and resistors that implement boolean logic operations in a circuit. Transistors make up logic gate, logic gates make circuits and circuits make up electronic system.

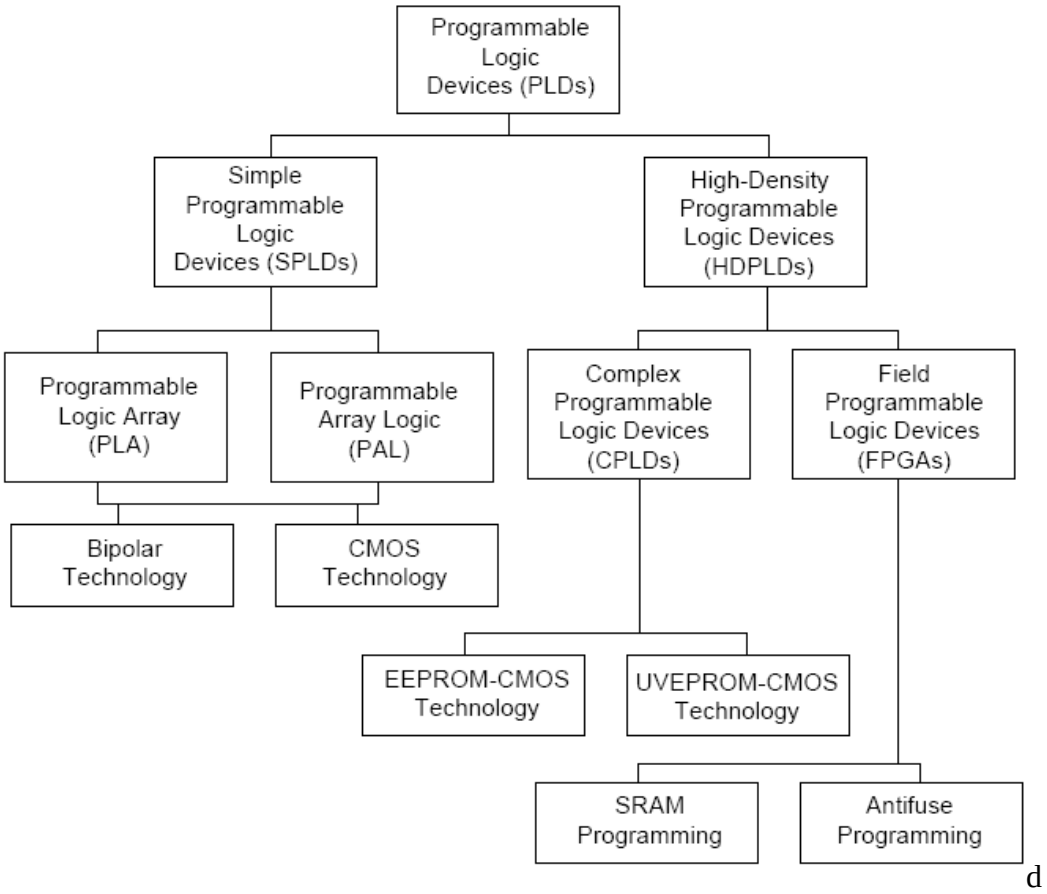
Programmable Logic Devices

A programmable Logic device refers to any type of integrated circuit that a logic design can be implemented and reconfigured in the field by the end user. Since these logic devices can be programmed in the field they are also called **Field Programmable Logic Devices (FPLDs).** The PLD provides flexibility for designers to implement many different designs in varying complexities for many different applications. One of the most common PLDs is the one time Programmable Read-only Memory (PROM). This comes in two different types: (a) mask programmable devices programmed by the vendor using a custom mask and interconnects and (b) field programmable devices that are configured by the user. One of the great advantages of PLDs is that they are very inexpensive at low quantities.

A device that was a follow on from the PROM technology that can be used for logic designs was the Programmable Logic Array (PLA). The PLA using the PROM structure turned out to be the first Field Programmable Logic Array (FPLA). The first FPLA was introduced in the mid-1970s. The FPLA had a fixed number of inputs, outputs and product terms that consisted of AND and OR arrays that contained programmable inputs. The FPLA did not have great success because they were very slow and complicated to use. The designer had to design to a fuse map instead of conventional boolean equations or schematic capture.

In the late 1970s the Programmable Array Logic (PAL) architecture was introduced that increased the use of programmable logic. The PAL architecture consisted of a programmable AND array and a fixed OR array so that each output is the sum of a specific set of product terms. The design entry tool for the earlier PAL was in the form of Boolean equations making it very easy to learn and implement. PAL devices are now available in different varieties from different vendors providing flexibility inputs/outputs, size of the OR-gate, and flip-flops. Some PALs are even provided in either NAND/NAND or NOR/NOR structure to increase design flexibility instead of the AND/OR structure.

PLDs can be divided into two groups, Simple Programmable Logic Devices (SPLDs) and High-Density Programmable Logic Devices (HDPLDs). SPLDs come in the PAL and PLA architecture, while HDPLDs include CPLDs and FPGAs. Figure 4.1 contains a hierarchical block diagram of the PLD architectures, subfamilies and programming technologies.



d

PLDs can be erased electrically and reprogrammed with a new design, making them very well suited for academic and prototyping

Types of Programmable Logic Devices

SPLDs (Simple Programmable Logic Devices)
ROM (Read-Only Memory)
PLA (Programmable Logic Array)
PAL (Programmable Array Logic)
GAL (Generic Array Logic)
CPLD (Complex Programmable Logic Device)
FPGA (Field-Programmable Gate Array)

Programmable Logic Devices

Programmable Logic Devices ***PLDs*** are the integrated circuits. They contain an array of AND gates & another array of OR gates. There are three kinds of PLDs based on the type of array*s*

, which has programmable feature.

- Programmable Read Only Memory
- Programmable Array Logic
- Programmable Logic Array

The process of entering the information into these devices is known as **programming**. Basically, users can program these devices or ICs electrically in order to implement the Boolean functions based on the requirement. Here, the term programming refers to hardware programming but not software programming.

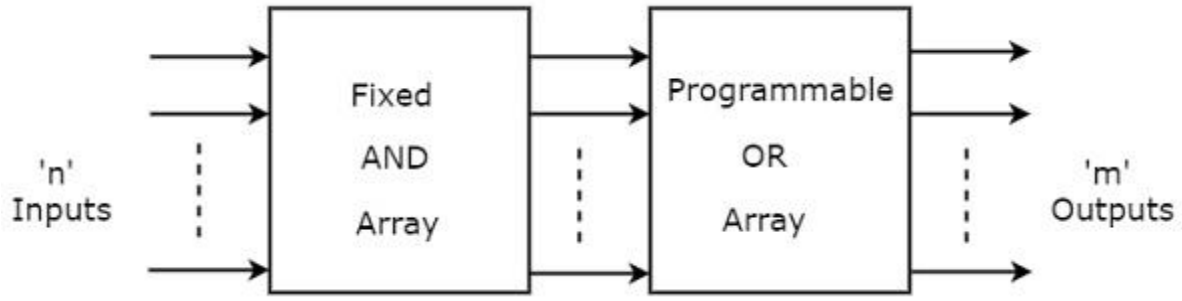
Programmable Read Only Memory *PROM*

Read Only Memory *ROM*

is a memory device, which stores the binary information permanently. That means, we can't change that stored information by any means later. If the ROM has programmable feature, then it is called as **Programmable ROM *PROM***

. The user has the flexibility to program the binary information electrically once by using PROM programmer.

PROM is a programmable logic device that has fixed AND array & Programmable OR array. The **block diagram** of PROM is shown in the following figure.



Here, the inputs of AND gates are not of programmable type. So, we have to generate 2^n product terms by using 2^n AND gates having n inputs each. We can implement these product terms by using $nx2^n$ decoder. So, this decoder generates 'n' **min terms**.

Here, the inputs of OR gates are programmable. That means, we can program any number of required product terms, since all the outputs of AND gates are applied as inputs to each OR gate. Therefore, the outputs of PROM will be in the form of **sum of min terms**.

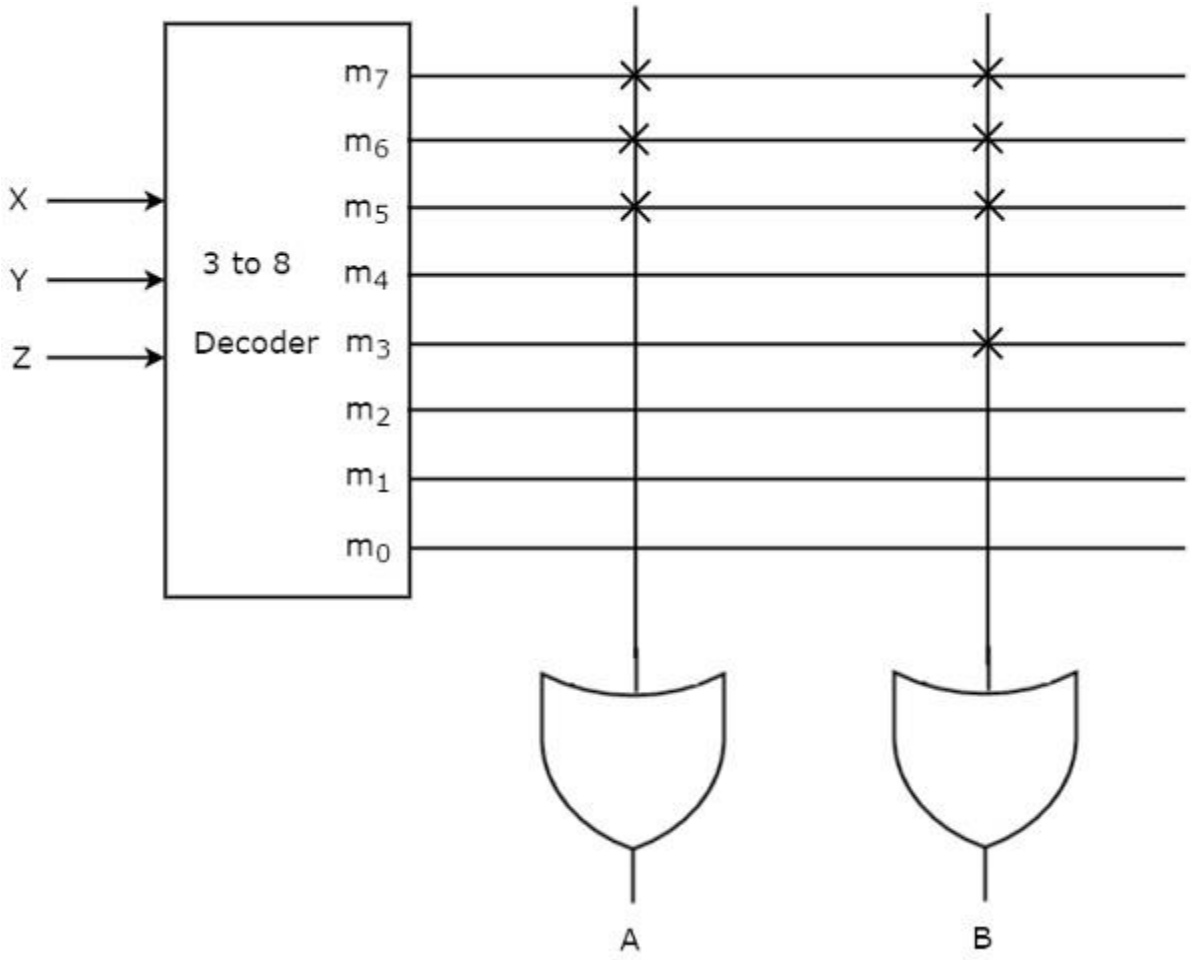
Example

Let us implement the following **Boolean functions** using PROM.

$$A(X,Y,Z)=\sum m(5,6,7)$$

$$B(X,Y,Z)=\sum m(3,5,6,7)$$

The given two functions are in sum of min terms form and each function is having three variables X, Y & Z. So, we require a 3 to 8 decoder and two programmable OR gates for producing these two functions. The corresponding **PROM** is shown in the following figure.

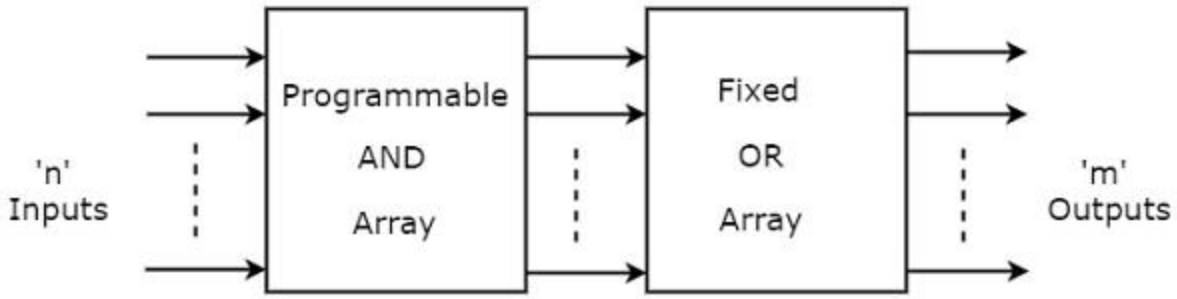


Here, 3 to 8 decoder generates eight min terms. The two programmable OR gates have the access of all these min terms. But, only the required min terms are programmed in order to produce the respective Boolean functions by each OR gate. The symbol 'X' is used for programmable connections.

Programmable Array Logic *PAL*

PAL is a programmable logic device that has Programmable AND array & fixed OR array. The advantage of PAL is that we can generate only the required product terms of Boolean function

instead of generating all the min terms by using programmable AND gates. The **block diagram** of PAL is shown in the following figure.



Here, the inputs of AND gates are programmable. That means each AND gate has both normal and complemented inputs of variables. So, based on the requirement, we can program any of those inputs. So, we can generate only the required **product terms** by using these AND gates.

Here, the inputs of OR gates are not of programmable type. So, the number of inputs to each OR gate will be of fixed type. Hence, apply those required product terms to each OR gate as inputs. Therefore, the outputs of PAL will be in the form of **sum of products form**.

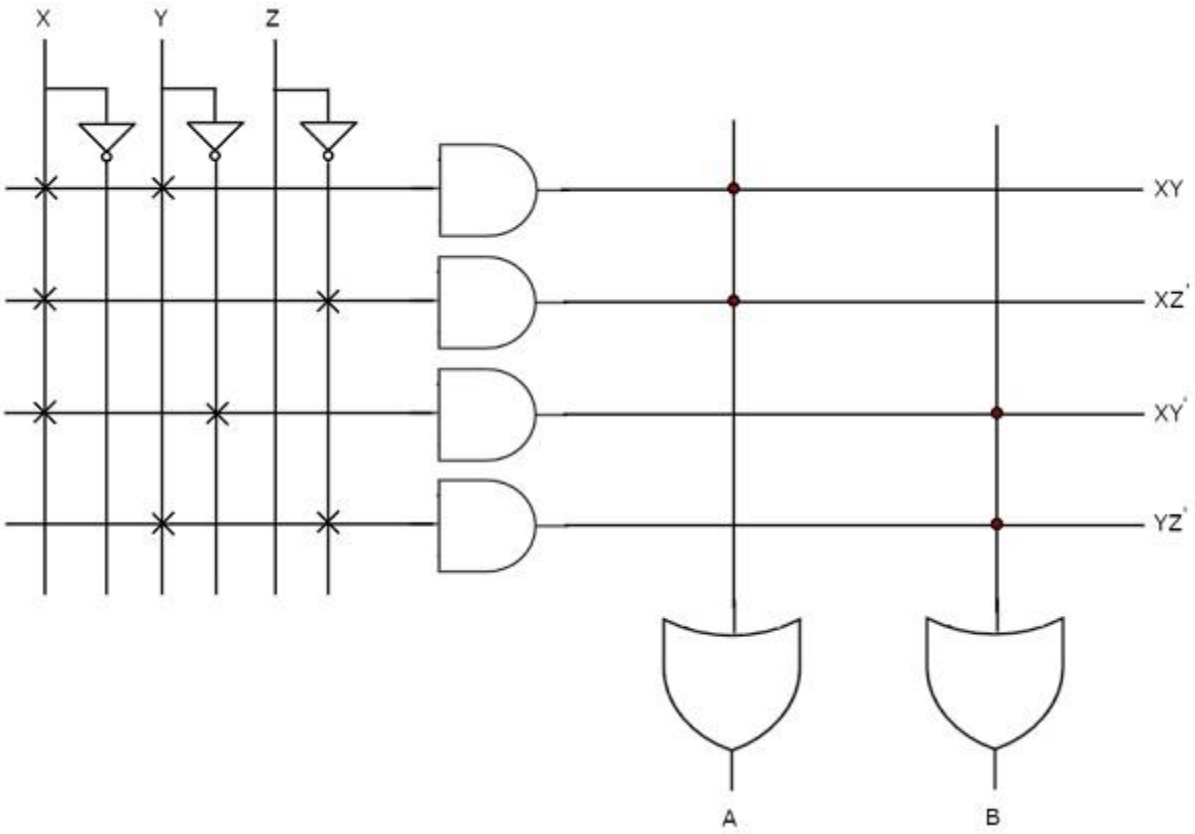
Example

Let us implement the following **Boolean functions** using PAL.

$$A=XY+XZ'$$

$$A=XY'+YZ'$$

The given two functions are in sum of products form. There are two product terms present in each Boolean function. So, we require four programmable AND gates & two fixed OR gates for producing those two functions. The corresponding **PAL** is shown in the following figure.



The **programmable AND gates** have the access of both normal and complemented inputs of variables. In the above figure, the inputs X, X'

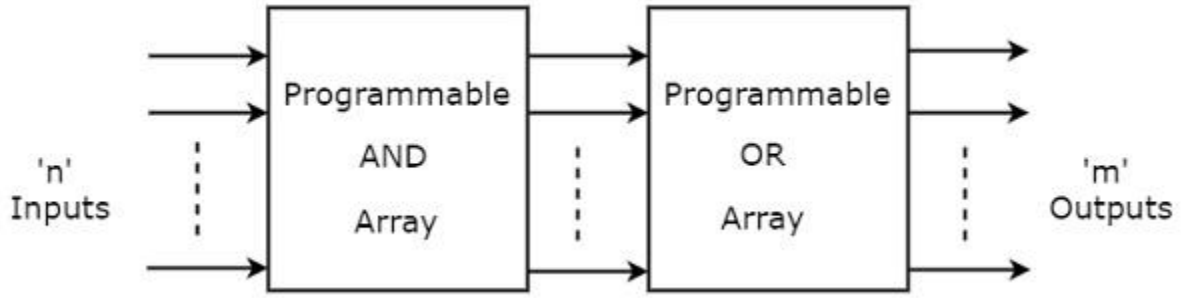
, Y, Y', Z & Z'

, are available at the inputs of each AND gate. So, program only the required literals in order to generate one product term by each AND gate. The symbol 'X' is used for programmable connections.

Here, the inputs of OR gates are of fixed type. So, the necessary product terms are connected to inputs of each **OR gate**. So that the OR gates produce the respective Boolean functions. The symbol '.' is used for fixed connections.

Programmable Logic Array *PLA*

PLA is a programmable logic device that has both Programmable AND array & Programmable OR array. Hence, it is the most flexible PLD. The **block diagram** of PLA is shown in the following figure.



Here, the inputs of AND gates are programmable. That means each AND gate has both normal and complemented inputs of variables. So, based on the requirement, we can program any of those inputs. So, we can generate only the required **product terms** by using these AND gates.

Here, the inputs of OR gates are also programmable. So, we can program any number of required product terms, since all the outputs of AND gates are applied as inputs to each OR gate. Therefore, the outputs of PAL will be in the form of **sum of products form**.

Example

Let us implement the following **Boolean functions** using PLA.

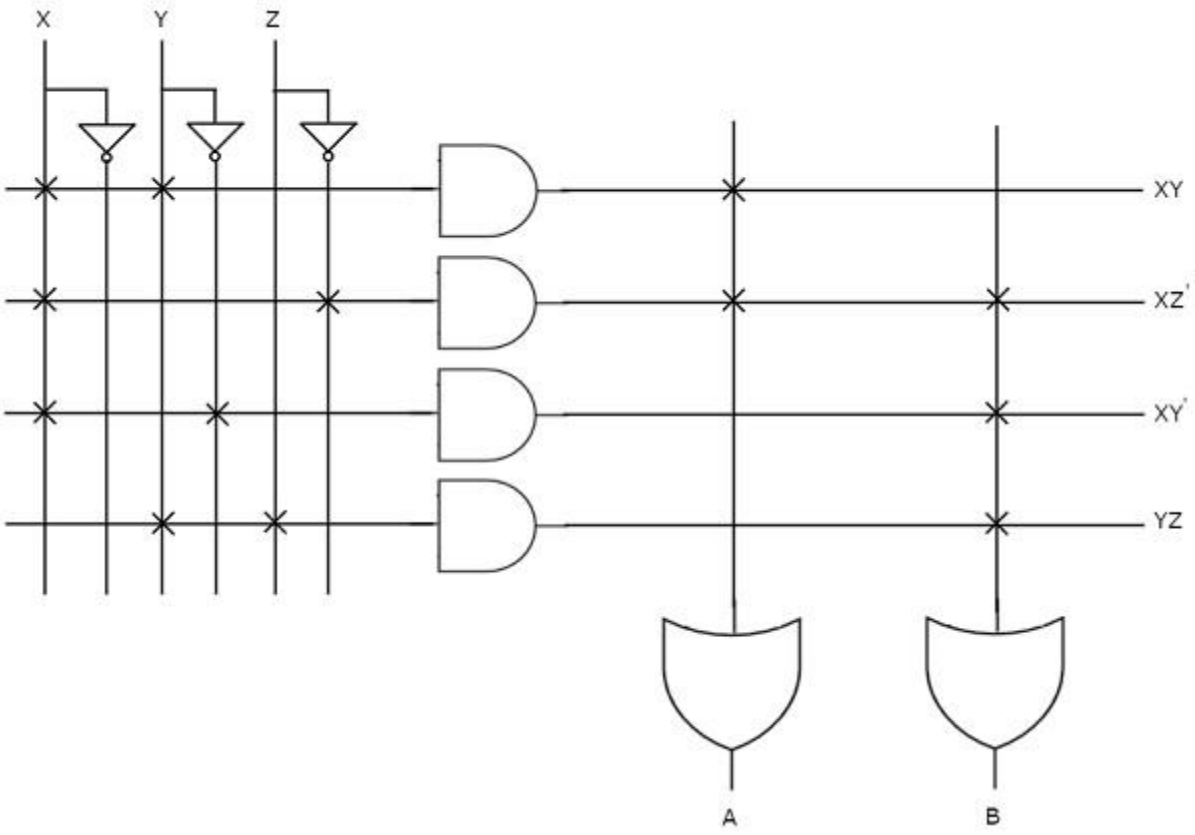
$$A=XY+XZ'$$

$$B=XY'+YZ+XZ'$$

The given two functions are in sum of products form. The number of product terms present in the given Boolean functions A & B are two and three respectively. One product term, $Z'X$

is common in each function.

So, we require four programmable AND gates & two programmable OR gates for producing those two functions. The corresponding **PLA** is shown in the following figure.



The **programmable AND gates** have the access of both normal and complemented inputs of variables. In the above figure, the inputs X, X'

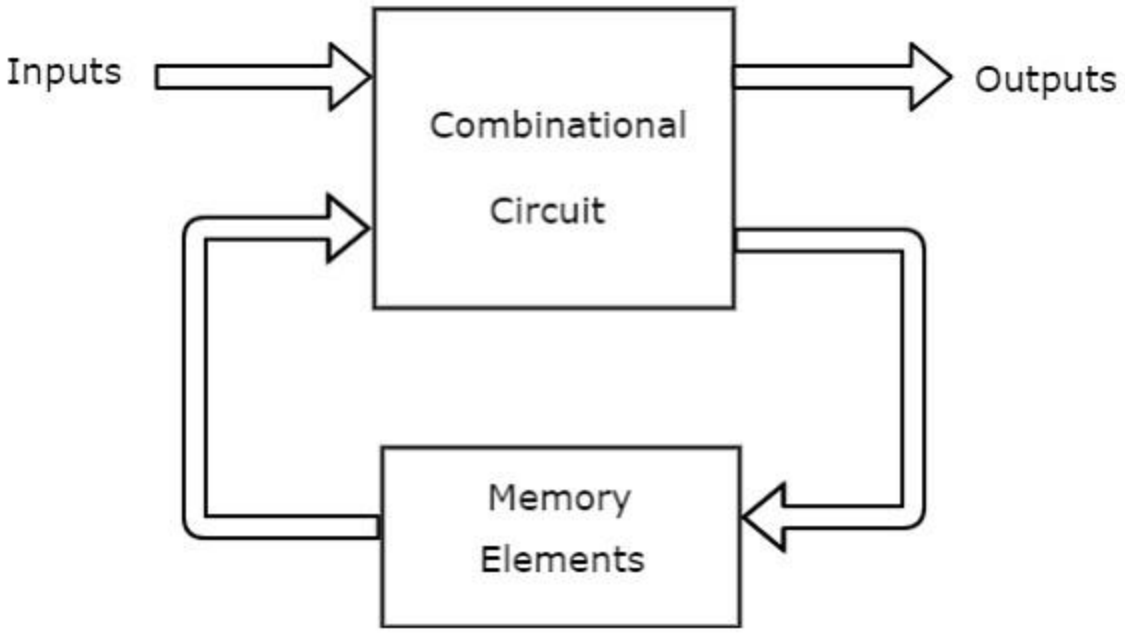
, Y, Y', Z & Z'

, are available at the inputs of each AND gate. So, program only the required literals in order to generate one product term by each AND gate.

All these product terms are available at the inputs of each **programmable OR gate**. But, only program the required product terms in order to produce the respective Boolean functions by each OR gate. The symbol 'X' is used for programmable connections.

We discussed various combinational circuits in earlier chapters. All these circuits have a set of output$S$

, which depends only on the combination of present inputs. The following figure shows the **block diagram** of sequential circuit.



This sequential circuit contains a set of inputs and output*S*

. The output*S* of sequential circuit depends not only on the combination of present inputs but also on the previous output*S*. Previous output is nothing but the **present state**. Therefore, sequential circuits contain combinational circuits along with memory *storage*

elements. Some sequential circuits may not contain combinational circuits, but only memory elements.

Following table shows the **differences** between combinational circuits and sequential circuits.

Combinational Circuits	**Sequential Circuits**
1. Outputs depend only on present inputs.	1. Outputs depend on both present inputs and present state.
2. Feedback path is not present.	2. Feedback path is present.
3. Memory elements are not required.	3. Memory elements are required.
4. Clock signal is not required.	4. Clock signal is required.
5. Easy to design.	5. Difficult to design.

Types of Sequential Circuits

Following are the two types of sequential circuits −

- Asynchronous sequential circuits
- Synchronous sequential circuits

Asynchronous sequential circuits

If some or all the outputs of a sequential circuit do not change *affect*

with respect to active transition of clock signal, then that sequential circuit is called as **Asynchronous sequential circuit**. That means, all the outputs of asynchronous sequential circuits do not change *affect*

at the same time. Therefore, most of the outputs of asynchronous sequential circuits are **not in synchronous** with either only positive edges or only negative edges of clock signal.

Synchronous sequential circuits

If all the outputs of a sequential circuit change *affect*

with respect to active transition of clock signal, then that sequential circuit is called as **Synchronous sequential circuit**. That means, all the outputs of synchronous sequential circuits change *affect*

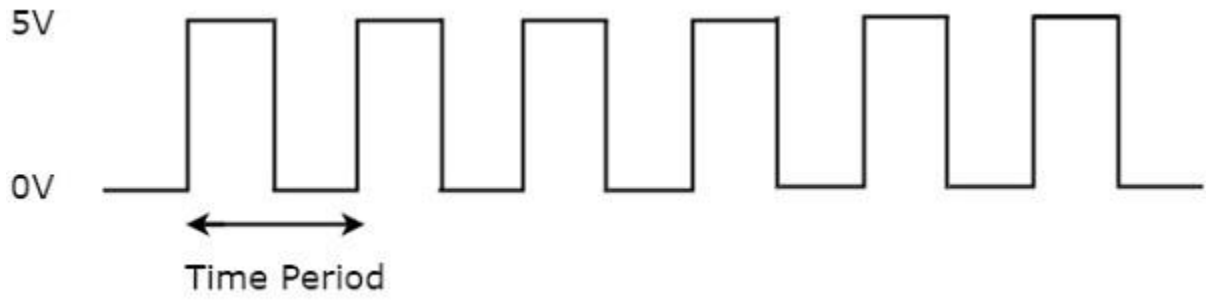
at the same time. Therefore, the outputs of synchronous sequential circuits are in synchronous with either only positive edges or only negative edges of clock signal.

Clock Signal and Triggering

In this section, let us discuss about the clock signal and types of triggering one by one.

Clock signal

Clock signal is a periodic signal and its ON time and OFF time need not be the same. We can represent the clock signal as a **square wave**, when both its ON time and OFF time are same. This clock signal is shown in the following figure.

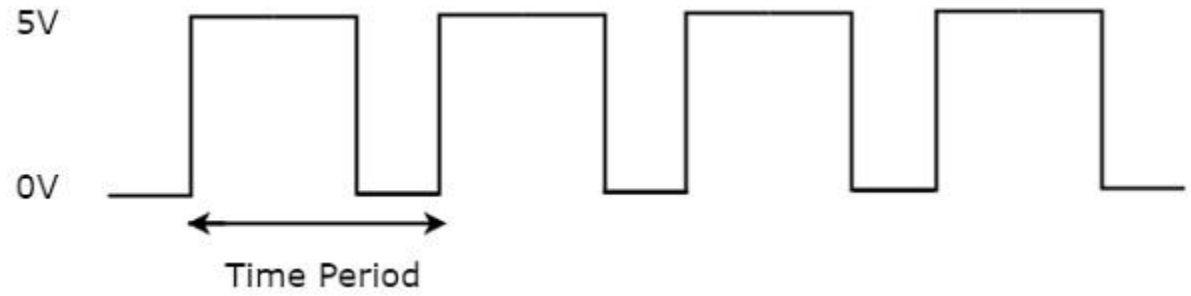


n the above figure, square wave is considered as clock signal. This signal stays at logic High $5V$

for some time and stays at logic Low $0V$

for equal amount of time. This pattern repeats with some time period. In this case, the **time period** will be equal to either twice of ON time or twice of OFF time.

We can represent the clock signal as **train of pulses**, when ON time and OFF time are not same. This clock signal is shown in the following figure.



In the above figure, train of pulses is considered as clock signal. This signal stays at logic High $5V$

for some time and stays at logic Low $0V$

for some other time. This pattern repeats with some time period. In this case, the **time period** will be equal to sum of ON time and OFF time.

The reciprocal of the time period of clock signal is known as the **frequency** of the clock signal. All sequential circuits are operated with clock signal. So, the frequency at which the sequential circuits can be operated accordingly the clock signal frequency has to be chosen.

Types of Triggering

Following are the two possible types of triggering that are used in sequential circuits.

- Level triggering
- Edge triggering

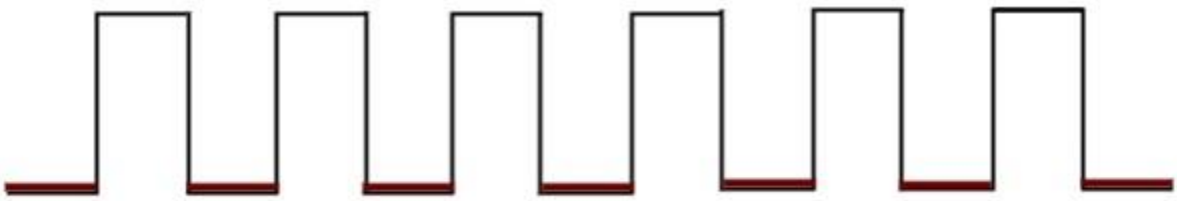
Level triggering

There are two levels, namely logic High and logic Low in clock signal. Following are the two **types of level triggering**.

- Positive level triggering
- Negative level triggering

If the sequential circuit is operated with the clock signal when it is in **Logic High**, then that type of triggering is known as **Positive level triggering**. It is highlighted in below figure.

If the sequential circuit is operated with the clock signal when it is in **Logic Low**, then that type of triggering is known as **Negative level triggering**. It is highlighted in the following figure.



Edge triggering

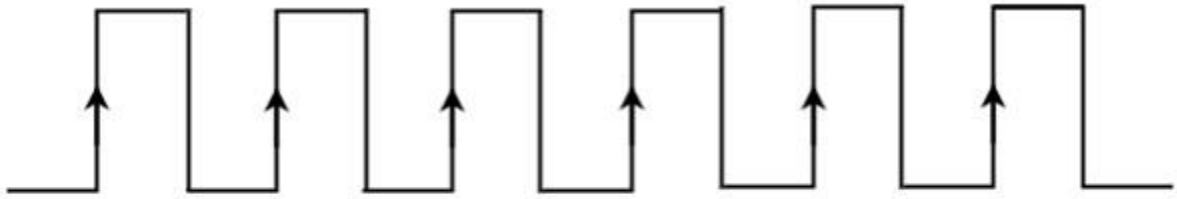
There are two types of transitions that occur in clock signal. That means, the clock signal transitions either from Logic Low to Logic High or Logic High to Logic Low.

Following are the two **types of edge triggering** based on the transitions of clock signal.

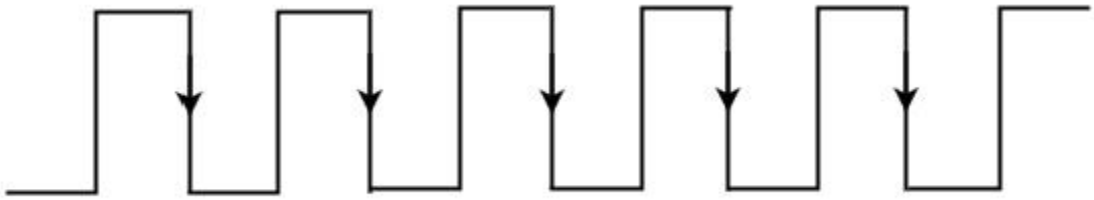
- Positive edge triggering

- Negative edge triggering

If the sequential circuit is operated with the clock signal that is transitioning from Logic Low to Logic High, then that type of triggering is known as **Positive edge triggering**. It is also called as rising edge triggering. It is shown in the following figure.



If the sequential circuit is operated with the clock signal that is transitioning from Logic High to Logic Low, then that type of triggering is known as **Negative edge triggering**. It is also called as falling edge triggering. It is shown in the following figure.



In coming chapters, we will discuss about various sequential circuits based on the type of triggering that can be used in it.

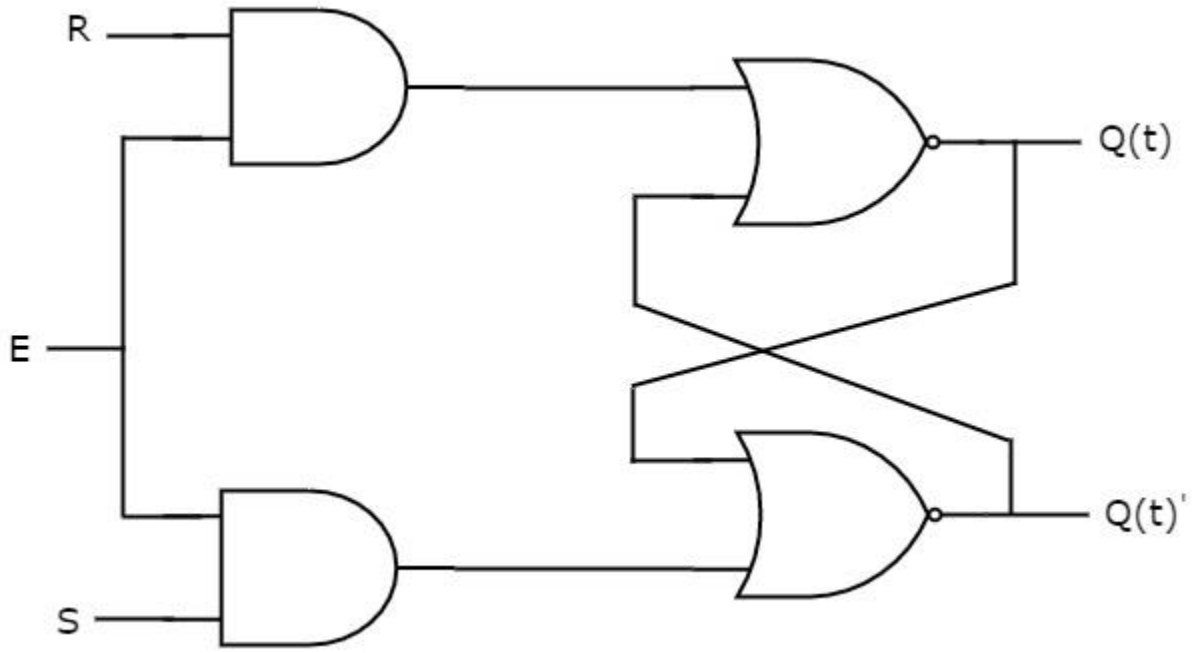
There are two types of memory elements based on the type of triggering that is suitable to operate it.

- Latches
- Flip-flops

Latches operate with enable signal, which is **level sensitive**. Whereas, flip-flops are edge sensitive. We will discuss about flip-flops in next chapter. Now, let us discuss about SR Latch & D Latch one by one.

SR Latch

SR Latch is also called as **Set Reset Latch**. This latch affects the outputs as long as the enable, E is maintained at '1'. The **circuit diagram** of SR Latch is shown in the following figure.



This circuit has two inputs S & R and two outputs Qt

& Qt'. The **upper NOR gate** has two inputs R & complement of present state, Qt' and produces next state, Qt+1

when enable, E is '1'.

Similarly, the **lower NOR gate** has two inputs S & present state, Qt

and produces complement of next state, Qt+1

' when enable, E is '1'.

We know that a **2-input NOR gate** produces an output, which is the complement of another input when one of the input is '0'. Similarly, it produces '0' output, when one of the input is '1'.

- If S = 1, then next state Qt+1

will be equal to '1' irrespective of present state, Qt

- values.

- If R = 1, then next state Qt+1

will be equal to ‘0’ irrespective of present state, Q_t

- values.

At any time, only of those two inputs should be ‘1’. If both inputs are ‘1’, then the next state Q_{t+1}

value is undefined.

The following table shows the **state table** of SR latch.

S	R	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	-

Therefore, SR Latch performs three types of functions such as Hold, Set & Reset based on the input conditions.

D Latch

There is one drawback of SR Latch. That is the next state value can’t be predicted when both the inputs S & R are one. So, we can overcome this difficulty by D Latch. It is also called as Data Latch. The **circuit diagram** of D Latch is shown in the following figure.

In previous chapter, we discussed about Latches. Those are the basic building blocks of flip-flops. We can implement flip-flops in two methods.

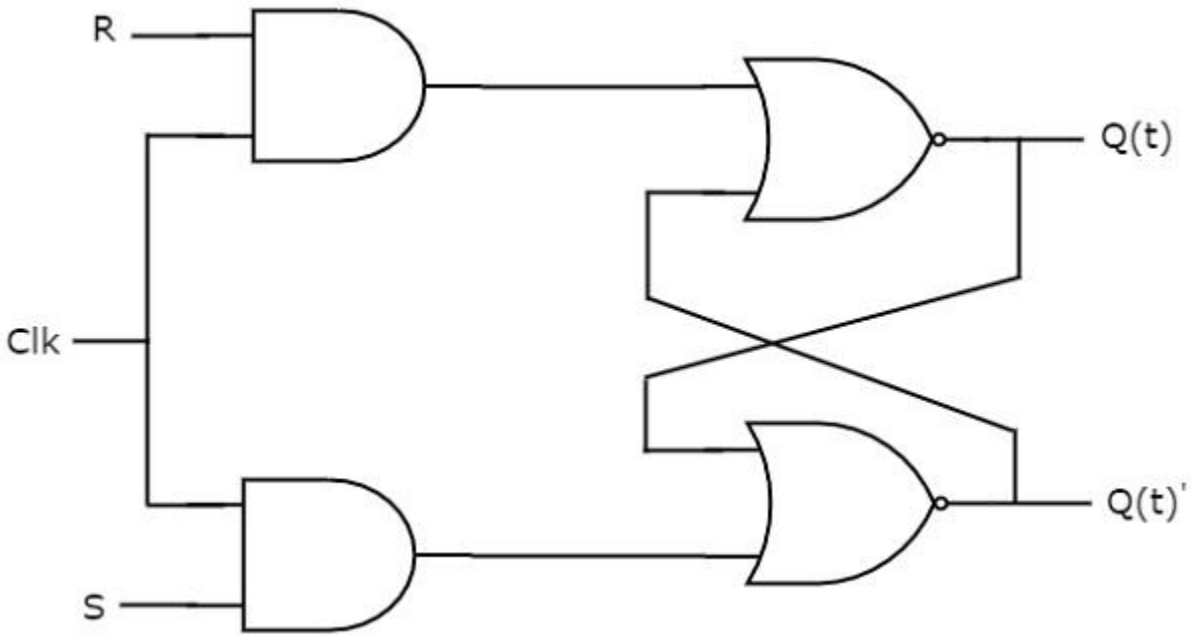
In first method, **cascade two latches** in such a way that the first latch is enabled for every positive clock pulse and second latch is enabled for every negative clock pulse. So that the combination of these two latches become a flip-flop.

In second method, we can directly implement the flip-flop, which is edge sensitive. In this chapter, let us discuss the following **flip-flops** using second method.

- 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. 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	Qt+1
0	0	Qt

0	1	0
1	0	1
1	1	-

Here, Qt

& $Qt+1$

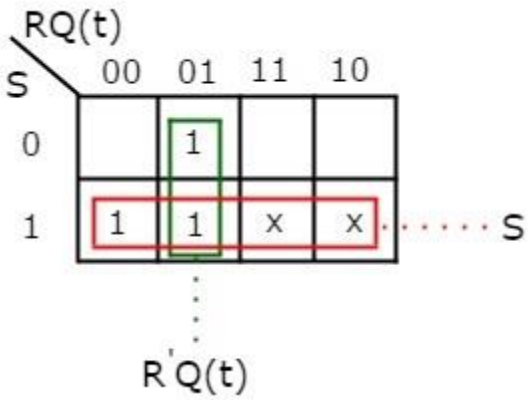
are present state & next state respectively. So, SR flip-flop can be used for one of these three functions such as Hold, Reset & Set based on the input conditions, when positive transition of clock signal is applied. The following table shows the **characteristic table** of SR flip-flop.

Present Inputs		**Present State**	**Next State**
S	**R**	**Qt**	**$Qt+1$**
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

By using three variable K-Map, we can get the simplified expression for next state, $Qt+1$

. The **three variable K-Map** for next state, $Qt+1$

is shown in the following figure.



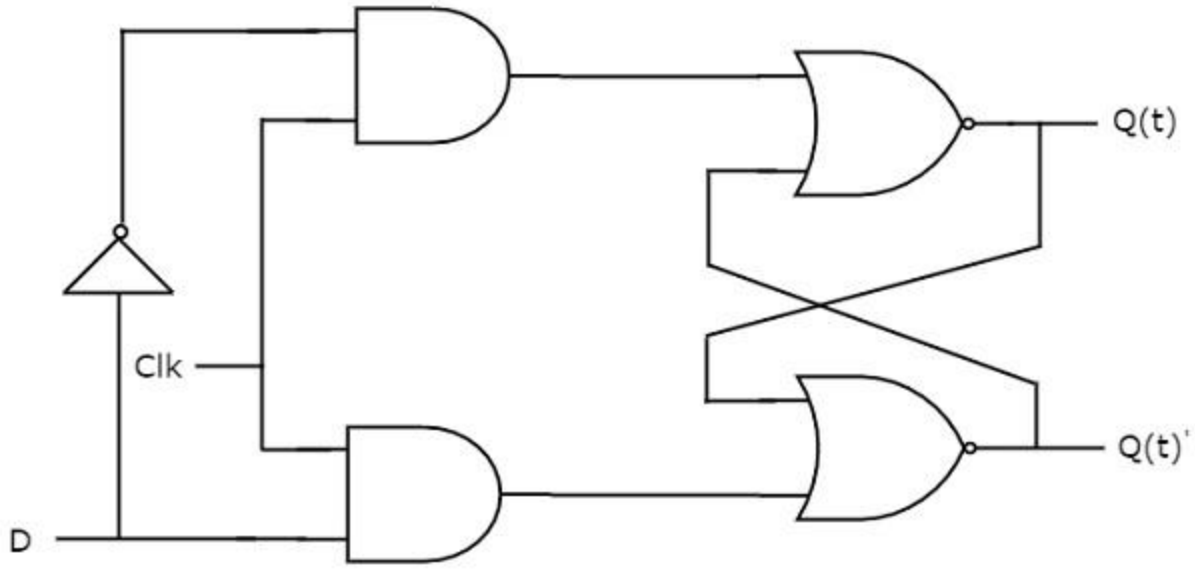
The maximum possible groupings of adjacent ones are already shown in the figure. Therefore, the **simplified expression** for next state $Qt+1$

is

$$Q(t+1)=S+R'Q(t)$$

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



This circuit has single input D and two outputs Qt

& Q*t*

’. 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

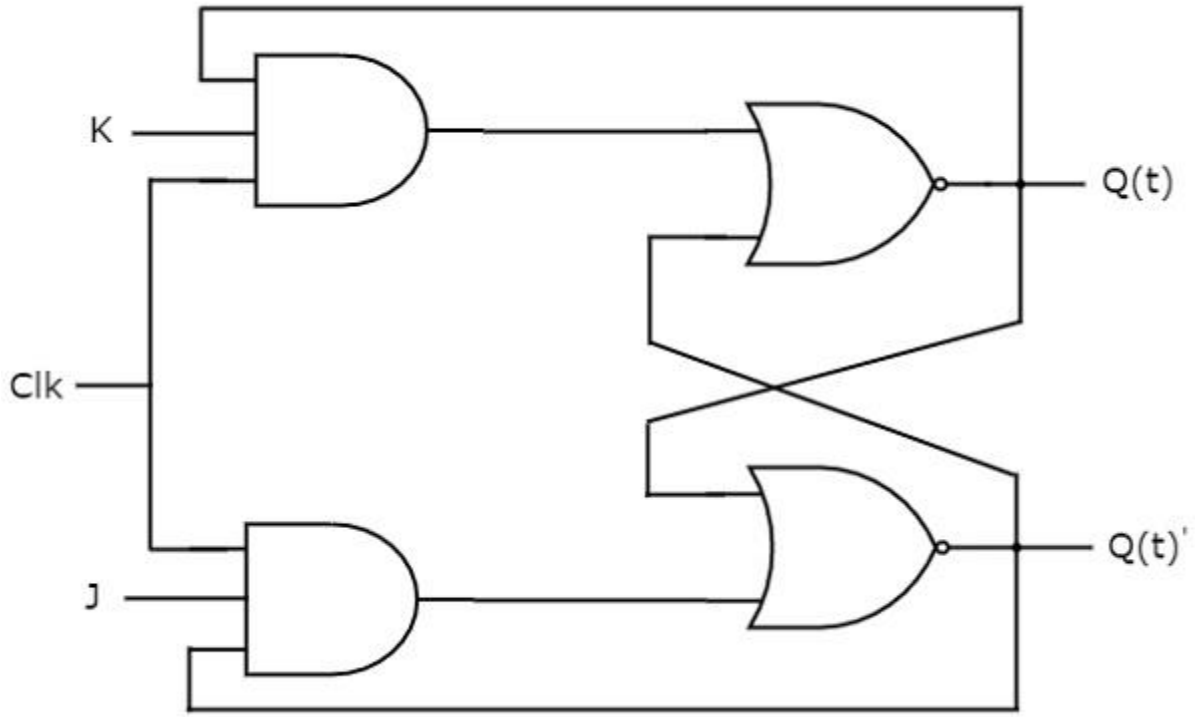
Therefore, D flip-flop always Hold the information, which is available on data input, D of earlier positive transition of clock signal. From the above state table, we can directly write the next state equation as

$$Q_{t+1} = D$$

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. 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 Qt'** and **R = KQt**

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	Qt+1
0	0	Qt
0	1	0
1	0	1
1	1	Qt

'

Here, Q_t

& Q_{t+1}

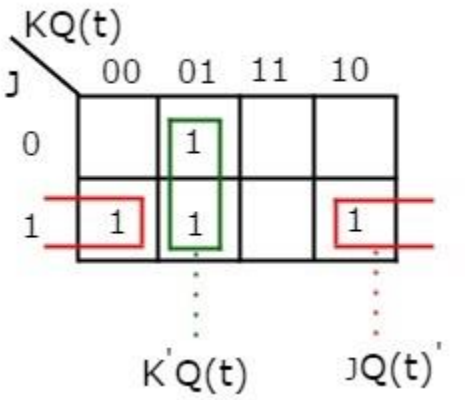
are present state & next state respectively. So, JK flip-flop can be used for one of these four functions such as Hold, Reset, Set & Complement of present state based on the input conditions, when positive transition of clock signal is applied. The following table shows the **characteristic table** of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

By using three variable K-Map, we can get the simplified expression for next state, Q_{t+1}

. **Three variable K-Map** for next state, Q_{t+1}

is shown in the following figure.



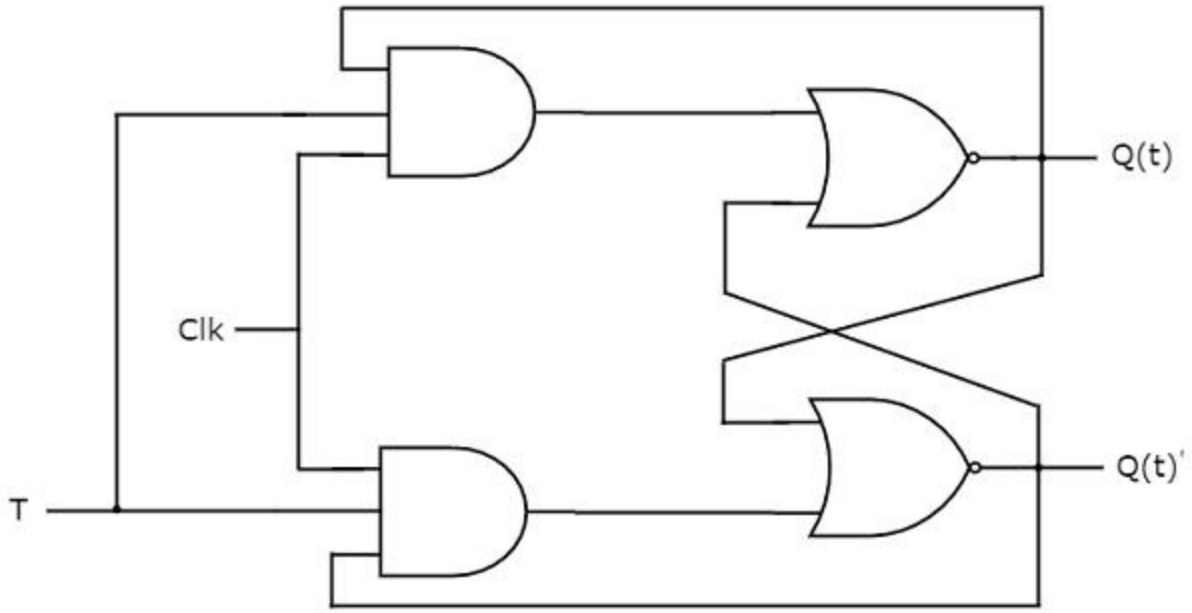
The maximum possible groupings of adjacent ones are already shown in the figure. Therefore, the **simplified expression** for next state Qt+1

is

$$Q(t+1)=JQ(t)'+K'Q(t)$$

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 Qt

& Q*t*

’. 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.

D	**Q*t*+1**
0	Q*t*
1	Q*t*

,

Here, Q*t*

& Q*t*+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

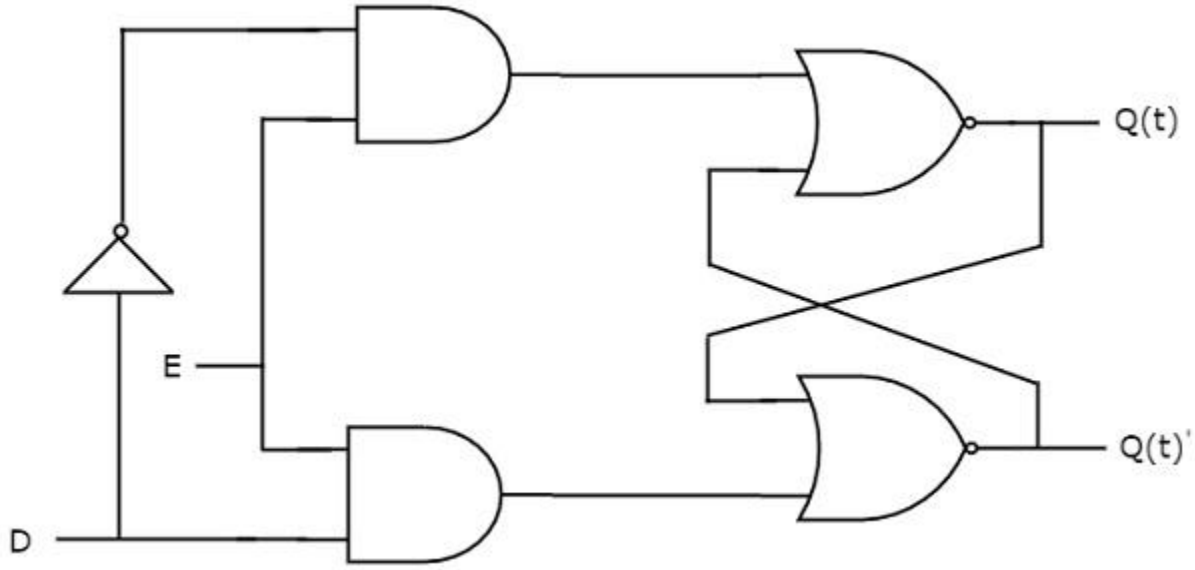
$$Q(t+1)=T'Q(t)+TQ(t)'$$

$$\Rightarrow Q(t+1)=T\oplus Q(t)$$

The output of T flip-flop always toggles for every positive transition of the clock signal, when input T remains at logic High 1

. 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.



This circuit has single input D and two outputs Qt

& Qt

'. D Latch is obtained from SR Latch by placing an inverter between S amp;& R inputs and connect D input to S. That means we eliminated the combinations of S & R are of same value.

- If D = 0 → S = 0 & R = 1, then next state Qt+1

will be equal to '0' irrespective of present state, Qt

- values. This is corresponding to the second row of SR Latch state table.

- If D = 1 → S = 1 & R = 0, then next state Q_{t+1}

will be equal to '1' irrespective of present state, Q_t

- values. This is corresponding to the third row of SR Latch state table.

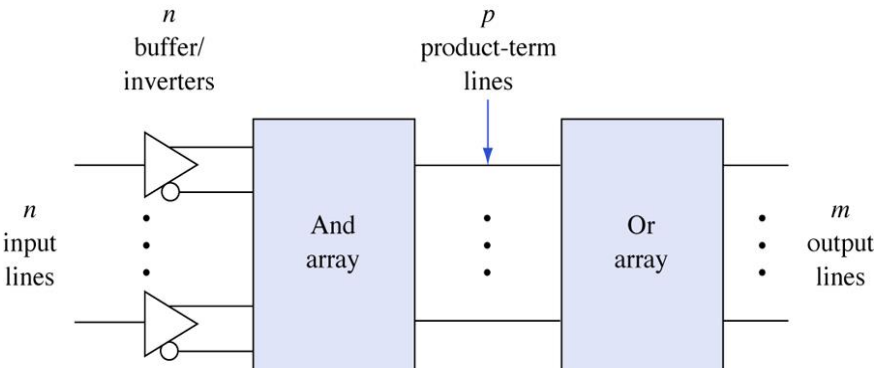
The following table shows the **state table** of D latch.

D	Q_{t+1}
0	0
1	1

Therefore, D Latch Hold the information that is available on data input, D. That means the output of D Latch is sensitive to the changes in the input, D as long as the enable is High.

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

General archtechture of PLDs



Revision Questions

i. Giving examples, explain types of Programmable Logic Devices?

ii. PLDs can be divided into two groups, outline and explain?

What is Discrete logic?