JHARSUGUDA ENGINEERING SCHOOL, JHARSUGUDA

LECTURE NOTES

ON

DIGITAL ELECTRONICS

(3rd Sem. ETC)

Prepared by: RAJENDRA DORA

(Lect. in E&TC)

DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING

Session-2022-23(Winter)

SYLLABUS:

Unit-1: Basics of Digital Electronics: Number System-Binary, Octal, Decimal, Hexadecimal - Conversion from one system to another number system. Arithmetic Operation-Addition, Subtraction, Multiplication, Division, 1's & 2's complement of Binary numbers& Subtraction using complements method, Digital Code & its application & distinguish between weighted & non-weight Code, Binary codes, excess-3 and Gray codes. Logic gates: AND, OR, NOT, NAND, NOR, Exclusive-OR, Exclusive-NOR-- Symbol, Function, expression, truth table & timing diagram, Universal Gates& its Realisation, Boolean algebra, Boolean expressions, DE Morgan's Theorems. Represent Logic Expression: SOP & POS forms, Karnaugh map (3 & 4 Variables) &Minimization of logical expressions, don't care conditions

Unit-2: Combinational logic circuits: Half adder, Full adder, Half Subtractor, Full Subtractor, Serial and Parallel Binary 4-bit adder, Multiplexer (4:1), De- multiplexer (1:4), Decoder, Encoder, Digital comparator (3 Bit), Seven segment Decoder (Definition, relevance, gate level of circuit Logic circuit, truth table, Applications of above)

Unit-3: Sequential logic Circuits: Principle of flip-flops operation, its Types, SR Flip Flop using NAND, NOR Latch (un clocked), Clocked SR, D, JK, T, JK Master Slave flip-flops-Symbol, logic Circuit, truth table and applications, Concept of Racing and how it can be avoided.

Unit-4: Registers, Memories & PLD: Shift Registers-Serial in Serial -out, Serial- in Parallel-out, Parallel in serial out and Parallel in parallel out, Universal shift registers-Applications. Types of Counter & applications, Binary counter, Asynchronous ripple counter (UP & DOWN), Decade counter. Synchronous counter, Ring Counter. Concept of memories-RAM, ROM, static RAM, dynamic RAM, PS RAM, Basic concept of PLD & applications

Unit-5: A/D and D/A Converters: Necessity of A/D and D/A converters. D/A conversion using weighted resistors methods. D/A conversion using R-2R ladder (Weighted resistors) network. A/D conversion using counter method. A/D conversion using Successive approximate method

Unit-6: LOGIC FAMILIES: Various logic families &categories according to the IC fabrication process, Characteristics of Digital ICs- Propagation Delay, fan-out, fan-in, Power Dissipation, Noise Margin, Power Supply requirement &Speed with Reference to logic families. Features, circuit operation &various applications of TTL(NAND), CMOS (NAND & NOR)

Unit-1

Basics of Digital Electronics

Number System-Binary, Octal, Decimal, Hexadecimal

If base or radix of a number system is 'r', then the numbers present in that number system are ranging from zero to r-1. The total numbers present in that number system is 'r'. So, we will get various number systems, by choosing the values of radix as greater than or equal to two.

The following number systems are the most commonly used.

- Decimal Number system
- Binary Number system
- Octal Number system
- Hexadecimal Number system

Decimal Number System:

The **base** or radix of Decimal number system is **10**. So, the numbers ranging from 0 to 9 are used in this number system. The part of the number that lies to the left of the **decimal point** is known as integer part. Similarly, the part of the number that lies to the right of the decimal point is known as fractional part.

In this number system, the successive positions to the left of the decimal point having weights of 10^0, 10^1, 10^2, 10^3 and so on. Similarly, the successive positions to the right of the decimal point having weights of 10^{-1}, 10^{-2}, 10^{-3} and so on. That means, each position has specific weight, which is **power of base 10**

Example

Consider the **decimal number 1358.246**. Integer part of this number is 1358 and fractional part of this number is 0.246. The digits 8, 5, 3 and 1 have weights of 100, 101, 10^2 and 10^3 respectively. Similarly, the digits 2, 4 and 6 have weights of 10^{-1}, 10^{-2} and 10^{-3} respectively.

Mathematically, we can write it as

$1358.246 = (1 \times 10^3) + (3 \times 10^2) + (5 \times 10^1) + (8 \times 10^0) + (2 \times 10^{-1}) +$

$(4 \times 10^{-2}) + (6 \times 10^{-3})$

After simplifying the right-hand side terms, we will get the decimal number, which is on left hand side.

Binary Number System:

All digital circuits and systems use this binary number system. The **base** or radix of this number system is **2**. So, the numbers 0 and 1 are used in this number system.

The part of the number, which lies to the left of the **binary point** is known as integer part. Similarly, the part of the number, which lies to the right of the binary point is known as fractional part.

In this number system, the successive positions to the left of the binary point having weights of 2^0, 2^1, 2^2, 2^3 and so on. Similarly, the successive positions to the right of the binary point having weights of 2^{-1}, 2^{-2}, 2^{-3} and so on. That means, each position has specific weight, which is **power of base 2**.

Example

Consider the **binary number 1101.011**. Integer part of this number is 1101 and fractional part of this number is 0.011. The digits 1, 0, 1 and 1 of integer part have weights of 2^0, 2^1, 2^2, 2^3 respectively. Similarly, the digits 0, 1 and 1 of fractional part have weights of 2^{-1}, 2^{-2}, 2^{-3} respectively.

Mathematically, we can write it as

$1101.011 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (0 \times 2^{-1}) +$

$(1 \times 2^{-2}) + (1 \times 2^{-3})$

After simplifying the right-hand side terms, we will get a decimal number, which is an equivalent of binary number on left hand side.

Octal Number System:

The **base** or radix of octal number system is **8**. So, the numbers ranging from 0 to 7 are used in this number system. The part of the number that lies to the left of the **octal point** is known as integer part. Similarly, the part of the number that lies to the right of the octal point is known as fractional part.

In this number system, the successive positions to the left of the octal point having weights of 8^0, 8^1, 8^2, 8^3 and so on. Similarly, the successive positions to the right of the octal point having weights of 8^{-1}, 8^{-2}, 8^{-3} and so on. That means, each position has specific weight, which is **power of base 8**.

Example

Consider the **octal number 1457.236**. Integer part of this number is 1457 and fractional part of this number is 0.236. The digits 7, 5, 4 and 1 have weights of 8^0, 8^1, 8^2 and 8^3 respectively. Similarly, the digits 2, 3 and 6 have weights of 8^{-1}, 8^{-2}, 8^{-3} respectively.

Mathematically, we can write it as

$1457.236 = (1 \times 8^3) + (4 \times 8^2) + (5 \times 8^1) + (7 \times 8^0) + (2 \times 8^{-1}) +$

$(3 \times 8^{-2}) + (6 \times 8^{-3})$

After simplifying the right-hand side terms, we will get a decimal number, which is an equivalent of octal number on left hand side.

Hexadecimal Number System:

The **base** or radix of Hexa-decimal number system is **16**. So, the numbers ranging from 0 to 9 and the letters from A to F are used in this number system. The decimal equivalent of Hexa-decimal digits from A to F are 10 to 15.

The part of the number, which lies to the left of the **hexadecimal point** is known as integer part. Similarly, the part of the number, which lies to the right of the Hexa-decimal point is known as fractional part.

In this number system, the successive positions to the left of the Hexa-decimal point having weights of 16^0, 16^1, 16^2, 16^3 and so on. Similarly, the successive positions to the right of the Hexa-decimal point having weights of 16^{-1}, 16^{-2}, 16^{-3} and so on. That means, each position has specific weight, which is **power of base 16**.

Example

Consider the **Hexa-decimal number 1A05.2C4**. Integer part of this number is 1A05 and fractional part of this number is 0.2C4. The digits 5, 0, A and 1 have weights of 16^0, 16^1, 16^2 and 16^3 respectively. Similarly, the digits 2, C and 4 have weights of 16^{-1}, 16^{-2} and 16^{-3} respectively.

Mathematically, we can write it as

$1A05.2C4 = (1 \times 16^3) + (10 \times 16^2) + (0 \times 16^1) + (5 \times 16^0) + (2 \times 16^{-1}) +$

$(12 \times 16^{-2}) + (4 \times 16^{-3})$

After simplifying the right-hand side terms, we will get a decimal number, which is an equivalent of Hexa-decimal number on left hand side.

Conversion from one system to another number system:

Decimal Number to other Bases Conversion:

If the decimal number contains both integer part and fractional part, then convert both the parts of decimal number into other base individually. Follow these steps for converting the decimal number into its equivalent number of any base 'r'.

- Do **division** of integer part of decimal number and **successive quotients** with base 'r' and note down the remainders till the quotient is zero. Consider the remainders in reverse order to get the integer part of equivalent number of base 'r'. That means, first and last remainders denote the least significant digit and most significant digit respectively.
- Do **multiplication** of fractional part of decimal number and **successive fractions** with base 'r' and note down the carry till the result is zero or the desired number of equivalent digits is obtained. Consider the normal sequence of carry in order to get the fractional part of equivalent number of base 'r'.

Decimal to Binary Conversion

The following two types of operations take place, while converting decimal number into its equivalent binary number.

- Division of integer part and successive quotients with base 2.
- Multiplication of fractional part and successive fractions with base 2.

Example

Consider the **decimal number 58.25**. Here, the integer part is 58 and fractional part is 0.25.

Step 1 − Division of 58 and successive quotients with base 2.

Operation	Quotient	Remainder
58/2	29	**0** LSBLSB
29/2	14	**1**
14/2	7	**0**
7/2	3	**1**
3/2	1	**1**
1/2	0	**1**MSBMSB

$$\Rightarrow 5858_{10} = 111010111010_2$$

Therefore, the **integer part** of equivalent binary number is **111010**.

Step 2 − Multiplication of 0.25 and successive fractions with base 2.

Operation	Result	Carry
0.25 x 2	0.5	0
0.5 x 2	1.0	1
-	0.0	-

$$\Rightarrow .25.25_{10} = .01.01_2$$

Therefore, the **fractional part** of equivalent binary number is **.01**

$$\Rightarrow 58.2558.25_{10} = 111010.01111010.01_2$$

Therefore, the **binary equivalent** of decimal number 58.25 is 111010.01.

Decimal to Octal Conversion

The following two types of operations take place, while converting decimal number into its equivalent octal number.

- Division of integer part and successive quotients with base 8.
- Multiplication of fractional part and successive fractions with base 8.

Example

Consider the **decimal number 58.25**. Here, the integer part is 58 and fractional part is 0.25.

Step 1 − Division of 58 and successive quotients with base 8.

Operation	Quotient	Remainder
58/8	7	**2**
7/8	0	**7**

$$\Rightarrow 5858_{10} = 7272_{8}$$

Therefore, the **integer part** of equivalent octal number is **72**.

Step 2 − Multiplication of 0.25 and successive fractions with base 8.

Operation	Result	Carry
0.25 x 8	2.00	2
-	0.00	-

$$\Rightarrow .25.25_{10} = .2.2_{8}$$

Therefore, the **fractional part** of equivalent octal number is .2

$$\Rightarrow 58.2558.25_{10} = 72.272.2_{8}$$

Therefore, the **octal equivalent** of decimal number 58.25 is 72.2.

Decimal to Hexa-Decimal Conversion

The following two types of operations take place, while converting decimal number into its equivalent hexa-decimal number.

- Division of integer part and successive quotients with base 16.
- Multiplication of fractional part and successive fractions with base 16.

Example

Consider the **decimal number 58.25**. Here, the integer part is 58 and decimal part is 0.25.

Step 1 − Division of 58 and successive quotients with base 16.

Operation	Quotient	Remainder
58/16	3	10=A
3/16	0	3

$$\Rightarrow 5858_{10} = 3A3A_{16}$$

Therefore, the **integer part** of equivalent Hexa-decimal number is 3A.

Step 2 − Multiplication of 0.25 and successive fractions with base 16.

Operation	Result	Carry
0.25 x 16	4.00	4
-	0.00	-

$$\Rightarrow .25.25_{10} = .4.4_{16}$$

Therefore, the **fractional part** of equivalent Hexa-decimal number is .4.

$$\Rightarrow 58.2558.25_{10} = 3A.43A.4_{16}$$

Therefore, the **Hexa-decimal equivalent** of decimal number 58.25 is 3A.4.

Binary Number to other Bases Conversion

The process of converting a number from binary to decimal is different to the process of converting a binary number to other bases. Now, let us discuss about the conversion of a binary number to decimal, octal and Hexa-decimal number systems one by one.

Binary to Decimal Conversion

For converting a binary number into its equivalent decimal number, first multiply the bits of binary number with the respective positional weights and then add all those products.

Example

Consider the **binary number 1101.11**.

Mathematically, we can write it as

$1101.111101.11_2 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) +$

(1×2^{-2})

$\Rightarrow 1101.111101.11_2 = 8 + 4 + 0 + 1 + 0.5 + 0.25 = 13.75$

$\Rightarrow 1101.111101.11_2 = 13.7513.75_{10}$

Therefore, the **decimal equivalent** of binary number 1101.11 is 13.75.

Binary to Octal Conversion

We know that the bases of binary and octal number systems are 2 and 8 respectively. Three bits of binary number is equivalent to one octal digit, since $2^3 = 8$.

Follow these two steps for converting a binary number into its equivalent octal number.

- Start from the binary point and make the groups of 3 bits on both sides of binary point. If one or two bits are less while making the group of 3 bits, then include required number of zeros on extreme sides.
- Write the octal digits corresponding to each group of 3 bits.

Example

Consider the **binary number 101110.01101**.

Step 1 − Make the groups of 3 bits on both sides of binary point.

101 110.011 01

Here, on right side of binary point, the last group is having only 2 bits. So, include one zero on extreme side in order to make it as group of 3 bits.

⇒ 101 110.011 010

Step 2 − Write the octal digits corresponding to each group of 3 bits.

$$\Rightarrow 101110.011010101110.011010_2 = 56.3256.32_8$$

Therefore, the **octal equivalent** of binary number 101110.01101 is 56.32.

Binary to Hexa-Decimal Conversion

We know that the bases of binary and Hexa-decimal number systems are 2 and 16 respectively. Four bits of binary number is equivalent to one Hexa-decimal digit, since $2^4 = 16$.

Follow these two steps for converting a binary number into its equivalent Hexa-decimal number.

- Start from the binary point and make the groups of 4 bits on both sides of binary point. If some bits are less while making the group of 4 bits, then include required number of zeros on extreme sides.
- Write the Hexa-decimal digits corresponding to each group of 4 bits.

Example

Consider the **binary number 101110.01101**

Step 1 − Make the groups of 4 bits on both sides of binary point.

10 1110.0110 1

Here, the first group is having only 2 bits. So, include two zeros on extreme side in order to make it as group of 4 bits. Similarly, include three zeros on extreme side in order to make the last group also as group of 4 bits.

⇒ 0010 1110.0110 1000

Step 2 − Write the Hexa-decimal digits corresponding to each group of 4 bits.

$$\Rightarrow 00101110.01101000001011 10.01101000_2 = 2E.682E.68_{16}$$

Therefore, the **Hexa-decimal equivalent** of binary number 101110.01101 is 2E.682E.68.

Octal Number to other Bases Conversion

The process of converting a number from octal to decimal is different to the process of converting an octal number to other bases. Now, let us discuss about the conversion of an octal number to decimal, binary and Hexa-decimal number systems one by one.

Octal to Decimal Conversion

For converting an octal number into its equivalent decimal number, first multiply the digits of octal number with the respective positional weights and then add all those products.

Example

Consider the **octal number 145.23**.

Mathematically, we can write it as

$145.23145.23_8 = (1 \times 8^2) + (4 \times 8^1) + (5 \times 8^0) + (2 \times 8^{-1}) + (3 \times 8^{-2})$
$\Rightarrow 145.23145.23_8 = 64 + 32 + 5 + 0.25 + 0.05 = 101.3$
$\Rightarrow 145.23145.23_8 = 101.3101.3_{10}$

Therefore, the **decimal equivalent** of octal number 145.23 is 101.3.

Octal to Binary Conversion

The process of converting an octal number to an equivalent binary number is just opposite to that of binary to octal conversion. By representing each octal digit with 3 bits, we will get the equivalent binary number.

Example

Consider the **octal number 145.23**.

Represent each octal digit with 3 bits.

$$145.23145.23_8 = 001100101.010011001100101.010011_2$$

The value doesn't change by removing the zeros, which are on the extreme side.

$$\Rightarrow 145.23145.23_8 = 1100101.0100111100101.010011_2$$

Therefore, the **binary equivalent** of octal number 145.23 is 1100101.010011.

Octal to Hexa-Decimal Conversion

Follow these two steps for converting an octal number into its equivalent Hexa-decimal number.

- Convert octal number into its equivalent binary number.
- Convert the above binary number into its equivalent Hexa-decimal number.

Example

Consider the **octal number 145.23**

In previous example, we got the binary equivalent of octal number 145.23 as 1100101.010011.

By following the procedure of binary to Hexa-decimal conversion, we will get

$$1100101.0100111100101.010011_2 = 65.4C65.4C16$$
$$\Rightarrow 145.23145.23_8 = 65.4C65.4C_{16}$$

Therefore, the **Hexa-decimal equivalent** of octal number 145.23 is 65.4*C*.

Hexa-Decimal Number to other Bases Conversion

The process of converting a number from Hexa-decimal to decimal is different to the process of converting Hexa-decimal number into other bases. Now, let us discuss about the conversion of Hexa-decimal number to decimal, binary and octal number systems one by one.

Hexa-Decimal to Decimal Conversion

For converting Hexa-decimal number into its equivalent decimal number, first multiply the digits of Hexa-decimal number with the respective positional weights and then add all those products.

Example

Consider the **Hexa-decimal number 1A5.2**

Mathematically, we can write it as

$$1A5.21A5.2_{16} = (1 \times 16^2) + (10 \times 16^1) + (5 \times 16^0) + (2 \times 16^{-1})$$
$$\Rightarrow 1A5.21A5.2_{16} = 256 + 160 + 5 + 0.125 = 421.125$$
$$\Rightarrow 1A5.21A5.2_{16} = 421.125421.125_{10}$$

Therefore, the **decimal equivalent** of Hexa-decimal number 1A5.2 is 421.125.

Hexa-Decimal to Binary Conversion

The process of converting Hexa-decimal number into its equivalent binary number is just opposite to that of binary to Hexa-decimal conversion. By representing each Hexa-decimal digit with 4 bits, we will get the equivalent binary number.

Example

Consider the **Hexa-decimal number 65.4C**

Represent each Hexa-decimal digit with 4 bits.

$$65.4C65.4C_6 = 01100101.0100110001100101.01001100_2$$

The value doesn't change by removing the zeros, which are at two extreme sides.

$$\Rightarrow 65.4C65.4C_{16} = 1100101.0100111100101.010011_2$$

Therefore, the **binary equivalent** of Hexa-decimal number 65.4C is 1100101.010011.

Hexa-Decimal to Octal Conversion

Follow these two steps for converting Hexa-decimal number into its equivalent octal number.

- Convert Hexa-decimal number into its equivalent binary number.
- Convert the above binary number into its equivalent octal number.

Example

Consider the **Hexa-decimal number 65.4C**

In previous example, we got the binary equivalent of Hexa-decimal number 65.4C as 1100101.010011.

By following the procedure of binary to octal conversion, we will get

$$1100101.0100111100101.010011_2 = 145.23145.23_8$$
$$\Rightarrow 65.4C65.4C_{16} = 145.23145.23_{\mathbf{8}}$$

Therefore, the **octal equivalent** of Hexa-decimal number 65.4*C* is 145.23.

Arithmetic Operation-Addition, Subtraction, Multiplication, Division:

The binary numbers can be divided into two groups – **Unsigned numbers** and **Signed numbers**.

Unsigned Numbers

Unsigned numbers contain only magnitude of the number. They don't have any sign. That means all unsigned binary numbers are positive. As in decimal number system, the placing of positive sign in front of the number is optional for representing positive numbers. Therefore, all positive numbers including zero can be treated as unsigned numbers if positive sign is not assigned in front of the number.

Signed Numbers

Signed numbers contain both sign and magnitude of the number. Generally, the sign is placed in front of number. So, we have to consider the positive sign for positive numbers and negative sign for negative numbers. Therefore, all numbers can be treated as signed numbers if the corresponding sign is assigned in front of the number.

If sign bit is zero, which indicates the binary number is positive. Similarly, if sign bit is one, which indicates the binary number is negative.

Representation of Un-Signed Binary Numbers:

The bits present in the un-signed binary number holds the **magnitude** of a number. That means, if the un-signed binary number contains **'N'** bits, then all **N** bits represent the magnitude of the number, since it doesn't have any sign bit.

Example

Consider the **decimal number 108**. The binary equivalent of this number is **1101100**. This is the representation of unsigned binary number.

$$108108_{10} = 11011001101100_2$$

It is having 7 bits. These 7 bits represent the magnitude of the number 108.

Representation of Signed Binary Numbers:

The Most Significant Bit MSBMSB of signed binary numbers is used to indicate the sign of the numbers. Hence, it is also called as **sign bit**. The positive sign is represented by placing '0' in the sign bit. Similarly, the negative sign is represented by placing '1' in the sign bit. If the signed binary number contains 'N' bits, then $N-1N-1$ bits only represent the magnitude of the number since one-bit MSB, MSB is reserved for representing sign of the number.

There are three **types of representations** for signed binary numbers

- Sign-Magnitude form
- 1's complement form
- 2's complement form

Representation of a positive number in all these 3 forms is same. But, only the representation of negative number will differ in each form.

Example

Consider the **positive decimal number +108**. The binary equivalent of magnitude of this number is 1101100. These 7 bits represent the magnitude of the number 108. Since it is positive number, consider the sign bit as zero, which is placed on left most side of magnitude.

$$+108+108_{10} = 0110110001101100_2$$

Therefore, the **signed binary representation** of positive decimal number +108 is **01101100**. So, the same representation is valid in sign-magnitude form, 1's complement form and 2's complement form for positive decimal number +108.

Sign-Magnitude form

In sign-magnitude form, the MSB is used for representing **sign** of the number and the remaining bits represent the **magnitude** of the number. So, just include sign bit at the left most side of unsigned binary number. This representation is similar to the signed decimal numbers representation.

Example

Consider the **negative decimal number -108**. The magnitude of this number is 108. We know the unsigned binary representation of 108 is 1101100. It is having 7 bits. All these bits represent the magnitude.

Since the given number is negative, consider the sign bit as one, which is placed on left most side of magnitude.

$$-108_{10} = 11101100_2$$

Therefore, the sign-magnitude representation of -108 is **11101100**.

1's complement form

The 1's complement of a number is obtained by **complementing all the bits** of signed binary number. So, 1's complement of positive number gives a negative number. Similarly, 1's complement of negative number gives a positive number.

That means, if you perform two times 1's complement of a binary number including sign bit, then you will get the original signed binary number.

Example

Consider the **negative decimal number -108**. The magnitude of this number is 108. We know the signed binary representation of 108 is 01101100.

It is having 8 bits. The MSB of this number is zero, which indicates positive number. Complement of zero is one and vice-versa. So, replace zeros by ones and ones by zeros in order to get the negative number.

$$-108_{10} = 10010011_2$$

Therefore, the **1's complement of** 108_{10} is 10010011_2.

2's complement form

The 2's complement of a binary number is obtained by **adding one to the 1's complement** of signed binary number. So, 2's complement of positive number gives a negative number. Similarly, 2's complement of negative number gives a positive number.

That means, if you perform two times 2's complement of a binary number including sign bit, then you will get the original signed binary number.

Example

Consider the **negative decimal number -108**.

We know the 1's complement of $(\mathbf{108})_{10}$ is $(\mathbf{10010011})_2$

2's compliment of 108_{10} *= 1's compliment of* 108_{10} *+ 1.*

= 10010011 + 1

= 10010100

Therefore, the **2's complement of** 108_{10} is 10010100_2.

Addition of two Signed Binary Numbers

Consider the two signed binary numbers A & B, which are represented in 2's complement form. We can perform the **addition** of these two numbers, which is similar to the addition of two unsigned binary numbers. But, if the resultant sum contains carry out from sign bit, then discard ignore it in order to get the correct value.

If resultant sum is positive, you can find the magnitude of it directly. But, if the resultant sum is negative, then take 2's complement of it in order to get the magnitude.

Example 1

Let us perform the **addition** of two decimal numbers **+7 and +4** using 2's complement method.

The **2's complement** representations of +7 and +4 with 5 bits each are shown below.

$$+7_{10} = 00111_2$$
$$+4_{10} = 00100_2$$

The addition of these two numbers is

$$+7_{10} + +4_{10} = 00111_2 + 00100_2$$
$$\Rightarrow +7_{10} + +4_{10} = 01011_2.$$

The resultant sum contains 5 bits. So, there is no carry out from sign bit. The sign bit '0' indicates that the resultant sum is **positive**. So, the magnitude of sum is 11 in decimal number system. Therefore, addition of two positive numbers will give another positive number.

Example 2

Let us perform the **addition** of two decimal numbers **-7** and **-4** using 2's complement method.

The **2's complement** representation of -7 and -4 with 5 bits each are shown below.

$$-7_{10} = 11001_2$$
$$-4_{10} = 11100_2$$

The addition of these two numbers is

$$-7_{10} + -4_{10} = 11001_2 + 11100_2$$
$$\Rightarrow -7_{10} + -4_{10} = 110101_2.$$

The resultant sum contains 6 bits. In this case, carry is obtained from sign bit. So, we can remove it

Resultant sum after removing carry is $-7_{10} + -4_{10} = 10101_2$.

The sign bit '1' indicates that the resultant sum is **negative**. So, by taking 2's complement of it we will get the magnitude of resultant sum as 11 in decimal number system. Therefore, addition of two negative numbers will give another negative number.

Subtraction of two Signed Binary Numbers

Consider the two signed binary numbers A & B, which are represented in 2's complement form. We know that 2's complement of positive number gives a negative number. So, whenever we have to subtract a number B from number A, then take 2's complement of B and add it to A. So, **mathematically** we can write it as

A - B = A + 2′s complement of B

Similarly, if we have to subtract the number A from number B, then take 2's complement of A and add it to B. So, **mathematically** we can write it as

B - A = B + 2′s complement of A

So, the subtraction of two signed binary numbers is similar to the addition of two signed binary numbers. But we have to take 2's complement of the number, which is supposed to be subtracted. This is the **advantage** of 2's complement technique. Follow, the same rules of addition of two signed binary numbers.

Example 3

Let us perform the **subtraction** of two decimal numbers **+7 and +4** using 2's complement method.

The subtraction of these two numbers is

$$+7_{10} - +4_{10} = +7_{10} + -4_{10}.$$

The **2's complement** representation of +7 and -4 with 5 bits each are shown below.

$$+7_{10} = 00111_2$$
$$+4_{10} = 11100_2$$
$$\Rightarrow +7_{10} + +4_{10} = 00111_2 + 11100_2 = 00011_2$$

Here, the carry obtained from sign bit. So, we can remove it. The resultant sum after removing carry is

$$+7_{10} + +4_{10} = 00011_2$$

The sign bit '0' indicates that the resultant sum is **positive**. So, the magnitude of it is 3 in decimal number system. Therefore, subtraction of two decimal numbers +7 and +4 is +3.

Example 4

Let us perform the **subtraction of** two decimal numbers **+4** and **+7** using 2's complement method.

The subtraction of these two numbers is

$$+4_{10} - +7_{10} = +4_{10} + -7_{10}.$$

The **2's complement** representation of +4 and -7 with 5 bits each are shown below.

$$+4_{10} = 00100_2$$
$$-7_{10} = 11001_2$$
$$\Rightarrow +4_{10} + -7_{10} = 00100_2 + 11001_2 = 11101_2$$

Here, carry is not obtained from sign bit. The sign bit '1' indicates that the resultant sum is **negative**. So, by taking 2's complement of it we will get the magnitude of resultant sum as 3 in decimal number system. Therefore, subtraction of two decimal numbers +4 and +7 is -3.

Digital Code & its application:

In the coding, when numbers or letters are represented by a specific group of symbols, it is said to be that number or letter is being encoded. The group of symbols is called as **code**.

The digital data is represented, stored and transmitted as group of bits. This group of bits is also called as **binary code**.

Binary codes can be classified into two types.

- Weighted codes
- Unweighted codes

If the code has positional weights, then it is said to be **weighted code**. Otherwise, it is an unweighted code. Weighted codes can be further classified as positively weighted codes and negatively weighted codes.

Binary Codes for Decimal digits

The following table shows the various binary codes for decimal digits 0 to 9.

Decimal Digit	8421 Code	2421 Code	84-2-1 Code	Excess 3 Code
0	0000	0000	0000	0011
1	0001	0001	0111	0100
2	0010	0010	0110	0101
3	0011	0011	0101	0110
4	0100	0100	0100	0111
5	0101	1011	1011	1000
6	0110	1100	1010	1001
7	0111	1101	1001	1010
8	1000	1110	1000	1011
9	1001	1111	1111	1100

We have 10 digits in decimal number system. To represent these 10 digits in binary, we require minimum of 4 bits. But, with 4 bits there will be 16 unique combinations of zeros and ones. Since, we have only 10 decimal digits, the other 6 combinations of zeros and ones are not required.

8 4 2 1 code

- The weights of this code are 8, 4, 2 and 1.
- This code has all positive weights. So, it is a **positively weighted code**.
- This code is also called as **natural BCD** Binary Coded Decimal **code**.

Example

Let us find the BCD equivalent of the decimal number 786. This number has 3 decimal digits 7, 8 and 6. From the table, we can write the BCD 84218421 codes of 7, 8 and 6 are 0111, 1000 and 0110 respectively.

$$\therefore 786_{10} = 0011110000110_{BCD}$$

There are 12 bits in BCD representation, since each BCD code of decimal digit has 4 bits.

2 4 2 1 code

- The weights of this code are 2, 4, 2 and 1.
- This code has all positive weights. So, it is a **positively weighted code**.
- It is an **unnatural BCD** code. Sum of weights of unnatural BCD codes is equal to 9.
- It is a **self-complementing** code. Self-complementing codes provide the 9's complement of a decimal number, just by interchanging 1's and 0's in its equivalent 2421 representation.

Example

Let us find the 2421 equivalent of the decimal number 786. This number has 3 decimal digits 7, 8 and 6. From the table, we can write the 2421 codes of 7, 8 and 6 are 1101, 1110 and 1100 respectively.

Therefore, the 2421 equivalent of the decimal number 786 is **110111101100**.

8 4 -2 -1 code

- The weights of this code are 8, 4, -2 and -1.
- This code has negative weights along with positive weights. So, it is a **negatively weighted code**.
- It is an **unnatural BCD** code.
- It is a **self-complementing** code.

Example

Let us find the 8 4-2-1 equivalent of the decimal number 786. This number has 3 decimal digits 7, 8 and 6. From the table, we can write the 8 4 -2 -1 codes of 7, 8 and 6 are 1001, 1000 and 1010 respectively.

Therefore, the 8 4 -2 -1 equivalent of the decimal number 786 is **100110001010**.

Excess 3 code

- This code doesn't have any weights. So, it is an **un-weighted code**.
- We will get the Excess 3 code of a decimal number by adding three 00110011 to the binary equivalent of that decimal number. Hence, it is called as Excess 3 code.
- It is a **self-complementing** code.

Example

Let us find the Excess 3 equivalent of the decimal number 786. This number has 3 decimal digits 7, 8 and 6. From the table, we can write the Excess 3 codes of 7, 8 and 6 are 1010, 1011 and 1001 respectively.

Therefore, the Excess 3 equivalent of the decimal number 786 is **101010111001**

Gray Code

The following table shows the 4-bit Gray codes corresponding to each 4-bit binary code.

Decimal Number	Binary Code	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

- This code doesn't have any weights. So, it is an **un-weighted code**.
- In the above table, the successive Gray codes are differed in one bit position only. Hence, this code is called as **unit distance** code.

Binary code to Gray Code Conversion

Follow these steps for converting a binary code into its equivalent Gray code.

- Consider the given binary code and place a zero to the left of MSB.
- Compare the successive two bits starting from zero. If the 2 bits are same, then the output is zero. Otherwise, output is one.

- Repeat the above step till the LSB of Gray code is obtained.

Example

From the table, we know that the Gray code corresponding to binary code 1000 is 1100. Now, let us verify it by using the above procedure.

Given, binary code is 1000.

Step 1 − By placing zero to the left of MSB, the binary code will be 01000.

Step 2 − By comparing successive two bits of new binary code, we will get the gray code as **1100**.

Logic gates:

Digital electronic circuits operate with voltages of **two logic levels** namely Logic Low and Logic High. The range of voltages corresponding to Logic Low is represented with '0'. Similarly, the range of voltages corresponding to Logic High is represented with '1'.

The basic digital electronic circuit that has one or more inputs and single output is known as **Logic gate**. Hence, the Logic gates are the building blocks of any digital system. We can classify these Logic gates into the following three categories.

- Basic gates
- Universal gates
- Special gates

Now, let us discuss about the Logic gates come under each category one by one.

Basic Gates

In earlier chapters, we learnt that the Boolean functions can be represented either in sum of products form or in product of sums form based on the requirement. So, we can implement these Boolean functions by using basic gates. The basic gates are AND, OR & NOT gates.

AND gate

An AND gate is a digital circuit that has two or more inputs and produces an output, which is the **logical AND** of all those inputs. It is optional to represent the **Logical AND** with the symbol '.'.

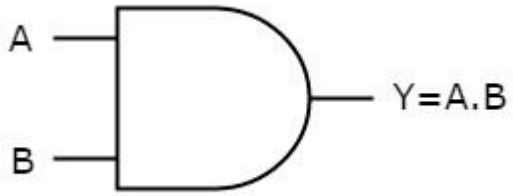
The following table shows the **truth table** of 2-input AND gate.

A	B	Y = A.B
0	0	0
0	1	0
1	0	0

1	1	1

Here A, B are the inputs and Y is the output of two input AND gate. If both inputs are '1', then only the output, Y is '1'. For remaining combinations of inputs, the output, Y is '0'.

The following figure shows the **symbol** of an AND gate, which is having two inputs A, B and one output, Y.



This AND gate produce an output Y, which is the **logical AND** of two inputs A, B. Similarly, if there are 'n' inputs, then the AND gate produces an output, which is the logical AND of all those inputs. That means, the output of AND gate will be '1', when all the inputs are '1'.

OR gate

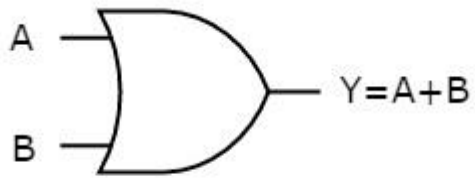
An OR gate is a digital circuit that has two or more inputs and produces an output, which is the logical OR of all those inputs. This **logical OR** is represented with the symbol '+'.

The following table shows the **truth table** of 2-input OR gate.

A	B	Y = A + B
0	0	0
0	1	1
1	0	1
1	1	1

Here A, B are the inputs and Y is the output of two input OR gate. If both inputs are '0', then only the output, Y is '0'. For remaining combinations of inputs, the output, Y is '1'.

The following figure shows the **symbol** of an OR gate, which is having two inputs A, B and one output, Y.



This OR gate produces an output YY, which is the **logical OR** of two inputs A, B. Similarly, if there are 'n' inputs, then the OR gate produces an output, which is the logical OR of all those inputs. That means, the output of an OR gate will be '1', when at least one of those inputs is '1'.

NOT gate

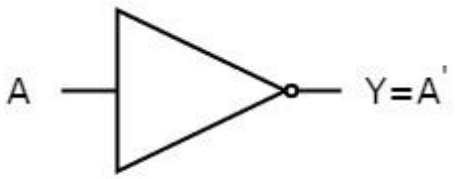
A NOT gate is a digital circuit that has single input and single output. The output of NOT gate is the **logical inversion** of input. Hence, the NOT gate is also called as inverter.

The following table shows the **truth table** of NOT gate.

A	Y = A'
0	1
1	0

Here A and Y are the input and output of NOT gate respectively. If the input, A is '0', then the output, Y is '1'. Similarly, if the input, A is '1', then the output, Y is '0'.

The following figure shows the **symbol** of NOT gate, which is having one input, A and one output, Y.



This NOT gate produces an output YY, which is the **complement** of input, A.

Universal gates

NAND & NOR gates are called as **universal gates**. Because we can implement any Boolean function, which is in sum of products form by using NAND gates alone. Similarly, we can implement any Boolean function, which is in product of sums form by using NOR gates alone.

NAND gate

NAND gate is a digital circuit that has two or more inputs and produces an output, which is the **inversion of logical AND** of all those inputs.

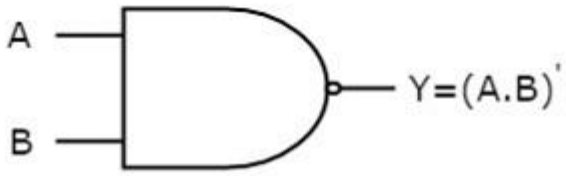
The following table shows the **truth table** of 2-input NAND gate.

A	B	Y = (A.B)'
0	0	1

0	1	1
1	0	1
1	1	0

Here A, B are the inputs and Y is the output of two input NAND gate. When both inputs are '1', the output, Y is '0'. If at least one of the input is zero, then the output, Y is '1'. This is just opposite to that of two input AND gate operation.

The following image shows the **symbol** of NAND gate, which is having two inputs A, B and one output, Y.



NAND gate operation is same as that of AND gate followed by an inverter. That's why the NAND gate symbol is represented like that.

NOR gate

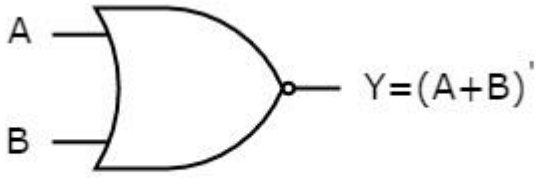
NOR gate is a digital circuit that has two or more inputs and produces an output, which is the **inversion of logical OR** of all those inputs.

The following table shows the **truth table** of 2-input NOR gate

A	**B**	**Y = (**A+B)**'**
0	0	1
0	1	0
1	0	0
1	1	0

Here A, B are the inputs and Y is the output. If both inputs are '0', then the output, Y is '1'. If at least one of the input is '1', then the output, Y is '0'. This is just opposite to that of two input OR gate operation.

The following figure shows the **symbol** of NOR gate, which is having two inputs A, B and one output, Y.



NOR gate operation is same as that of OR gate followed by an inverter. That's why the NOR gate symbol is represented like that.

Special Gates

Ex-OR & Ex-NOR gates are called as special gates. Because, these two gates are special cases of OR & NOR gates.

Ex-OR gate

The full form of Ex-OR gate is **Exclusive-OR** gate. Its function is same as that of OR gate except for some cases, when the inputs having even number of ones.

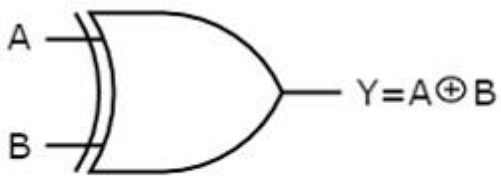
The following table shows the **truth table** of 2-input Ex-OR gate.

A	B	Y = A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

Here A, B are the inputs and Y is the output of two input Ex-OR gate. The truth table of Ex-OR gate is same as that of OR gate for first three rows. The only modification is in the fourth row. That means, the output YY is zero instead of one, when both the inputs are one, since the inputs having even number of ones.

Therefore, the output of Ex-OR gate is '1', when only one of the two inputs is '1'. And it is zero, when both inputs are same.

Below figure shows the **symbol** of Ex-OR gate, which is having two inputs A, B and one output, Y.



Ex-OR gate operation is similar to that of OR gate, except for few combinationss of inputs. That's why the Ex-OR gate symbol is represented like that. The output of Ex-OR gate is '1', when odd number of ones present at the inputs. Hence, the output of Ex-OR gate is also called as an **odd function**.

Ex-NOR gate

The full form of Ex-NOR gate is **Exclusive-NOR** gate. Its function is same as that of NOR gate except for some cases, when the inputs having even number of ones.

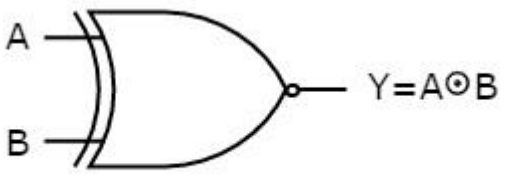
The following table shows the **truth table** of 2-input Ex-NOR gate.

A	B	Y = A⊙B
0	0	1
0	1	0
1	0	0
1	1	1

Here A, B are the inputs and Y is the output. The truth table of Ex-NOR gate is same as that of NOR gate for first three rows. The only modification is in the fourth row. That means, the output is one instead of zero, when both the inputs are one.

Therefore, the output of Ex-NOR gate is '1', when both inputs are same. And it is zero, when both the inputs are different.

The following figure shows the **symbol** of Ex-NOR gate, which is having two inputs A, B and one output, Y.



Ex-NOR gate operation is similar to that of NOR gate, except for few combinations of inputs. That's why the Ex-NOR gate symbol is represented like that. The output of Ex-NOR gate is '1', when even number of ones present at the inputs. Hence, the output of Ex-NOR gate is also called as an **even function**.

From the above truth tables of Ex-OR & Ex-NOR logic gates, we can easily notice that the Ex-NOR operation is just the logical inversion of Ex-OR operation.

Boolean algebra & Boolean expressions:

Boolean Algebra is an algebra, which deals with binary numbers & binary variables. Hence, it is also called as Binary Algebra or logical Algebra. A mathematician, named George Boole had developed this algebra in 1854. The variables used in this algebra are also called as Boolean variables.

The range of voltages corresponding to Logic 'High' is represented with '1' and the range of voltages corresponding to logic 'Low' is represented with '0'.

Postulates and Basic Laws of Boolean Algebra

In this section, let us discuss about the Boolean postulates and basic laws that are used in Boolean algebra. These are useful in minimizing Boolean functions.

Boolean Postulates
Consider the binary numbers 0 and 1, Boolean variable xx and its complement x′. Either the Boolean variable or complement of it is known as **literal**. The four possible **logical OR** operations among these literals and binary numbers are shown below.

$$x + 0 = x$$

$$x + 1 = 1$$

$$x + x = x$$

$$x + x' = 1$$

Similarly, the four possible **logical AND** operations among those literals and binary numbers are shown below.

$$x.1 = x$$

$$x.0 = 0$$

$$x.x = x$$

$$x.x' = 0$$

These are the simple Boolean postulates. We can verify these postulates easily, by substituting the Boolean variable with '0' or '1'.

Note− The complement of complement of any Boolean variable is equal to the variable itself. i.e., x''=x.

Basic Laws of Boolean Algebra

Following are the three basic laws of Boolean Algebra.

- Commutative law
- Associative law
- Distributive law

Commutative Law

If any logical operation of two Boolean variables gives the same result irrespective of the order of those two variables, then that logical operation is said to be **Commutative**. The logical OR & logical AND operations of two Boolean variables x & y are shown below

$$x + y = y + x$$

$$x.y = y.x$$

The symbol '+' indicates logical OR operation. Similarly, the symbol '.' indicates logical AND operation and it is optional to represent. Commutative law obeys for logical OR & logical AND operations.

Associative Law

If a logical operation of any two Boolean variables is performed first and then the same operation is performed with the remaining variable gives the same result, then that logical operation is said to be **Associative**. The logical OR & logical AND operations of three Boolean variables x, y & z are shown below.

$$(x + y)+z= x+(y + z)$$
$$(x.y).z = x.(y.z)$$

Associative law obeys for logical OR & logical AND operations.

Distributive Law

If any logical operation can be distributed to all the terms present in the Boolean function, then that logical operation is said to be **Distributive**. The distribution of logical OR & logical AND operations of three Boolean variables x, y & z are shown below.

$$x.(y+z) = x.y + x.z$$
$$x + (y.z) = (x+y).(x+z)$$

Distributive law obeys for logical OR and logical AND operations.

These are the Basic laws of Boolean algebra. We can verify these laws easily, by substituting the Boolean variables with '0' or '1'.

Theorems of Boolean Algebra

The following two theorems are used in Boolean algebra.

- Duality theorem
- De Morgan's theorem

Duality Theorem

This theorem states that the **dual** of the Boolean function is obtained by interchanging the logical AND operator with logical OR operator and zeros with ones. For every Boolean function, there will be a corresponding Dual function.

Let us make the Boolean equations relations that we discussed in the section of Boolean postulates and basic laws into two groups. The following table shows these two groups.

Group1	Group2
x + 0 = x	x.1 = x
x + 1 = 1	x.0 = 0
x + x = x	x.x = x
x + x’ = 1	x.x’ = 0
x + y = y + x	x.y = y.x
x + (y+z) = (x+y) +z	(x.y).z = x.(y.z)
x.(y+z) = x.y + x.z	x +(y.z) = (x+y).(x+z)

In each row, there are two Boolean equations and they are dual to each other. We can verify all these Boolean equations of Group1 and Group2 by using duality theorem.

De Morgan’s Theorem

This theorem is useful in finding the **complement of Boolean function**. It states that the complement of logical OR of at least two Boolean variables is equal to the logical AND of each complemented variable.

De Morgan’s theorem with 2 Boolean variables x and y can be represented as

$$(x+y)' = x'.y'$$

The dual of the above Boolean function is

$$(x.y)' = x' + y'$$

Therefore, the complement of logical AND of two Boolean variables is equal to the logical OR of each complemented variable. Similarly, we can apply DeMorgan’s theorem for more than 2 Boolean variables also.

Simplification of Boolean Functions

Till now, we discussed the postulates, basic laws and theorems of Boolean algebra. Now, let us simplify some Boolean functions.

Example 1

Let us **simplify** the Boolean function, f = p’qr + pq’r + pqr’ + pqr

We can simplify this function in two methods.

Method 1

Given Boolean function, f = p’qr + pq’r + pqr’ +pqr.

Step 1 − In first and second terms r is common and in third and fourth terms pq is common. So, take the common terms by using **Distributive law**.

$$\Rightarrow f = (p'q+pq')r + pq(r'+r)$$

Step 2 − The terms present in first parenthesis can be simplified to Ex-OR operation. The terms present in second parenthesis can be simplified to '1' using **Boolean postulate**

$$\Rightarrow f = (p \oplus q)r + pqx1$$

Step 3 − The first term can't be simplified further. But, the second term can be simplified to pq using **Boolean postulate**.

$$\Rightarrow f = (p \oplus q)r + pq$$

Therefore, the simplified Boolean function is **f = (p⊕q)r + pq**

Method 2

Given Boolean function, f = p'qr + pq'r + pqr' + pqr.

Step 1 − Use the **Boolean postulate**, x + x = x. That means, the Logical OR operation with any Boolean variable 'n' times will be equal to the same variable. So, we can write the last term pqr two more times.

$$\Rightarrow f = p'qr + pq'r + pqr' + pqr + pqr + pqr$$

Step 2 − Use **Distributive law** for 1st and 4th terms, 2nd and 5th terms, 3rd and 6th terms.

$$\Rightarrow f = qr(p'+p) + pr(q'+q) + pq(r'+r)$$

Step 3 − Use **Boolean postulate**, x + x' = 1 for simplifying the terms present in each parenthesis.

$$\Rightarrow f = qrx1 + prx1 + pqx1$$

Step 4 − Use **Boolean postulate**, x.1 = x for simplifying the above three terms.

$$\Rightarrow f = qr + pr + pq$$

$$\Rightarrow f = pq + qr + pr$$

Therefore, the simplified Boolean function is **f = pq + qr + pr**.

So, we got two different Boolean functions after simplifying the given Boolean function in each method. Functionally, those two Boolean functions are same. So, based on the requirement, we can choose one of those two Boolean functions.

Example 2

Let us find the **complement** of the Boolean function, f = p'q + pq'.

The complement of Boolean function is f' = (p'q+pq')'.

Step 1 − Use DeMorgan's theorem, (x+y)' = x'.y'.

$$\Rightarrow f' = (p'q)'.(pq')'$$

Step 2 − Use DeMorgan's theorem, (x.y)' = x' + y'

$$\Rightarrow f' = \{(p''+q')(p'+q'')\}$$

Step3 − Use the Boolean postulate, x'x''=x.

$$\Rightarrow f' = \{p + q'\}.\{p' + q\}$$

$$\Rightarrow f' = pp' + pq + p'q' + qq'$$

Step 4 − Use the Boolean postulate, xx'=0.

$$\Rightarrow f = 0 + pq + p'q' + 0$$

$$\Rightarrow f = pq + p'q'$$

Therefore, the **complement** of Boolean function, p'q + pq' is **pq + p'q'**.

Represent Logic Expression: SOP & POS forms:

We will get four Boolean product terms by combining two variables x and y with logical AND operation. These Boolean product terms are called as **min terms** or **standard product terms**. The min terms are x'y', x'y, xy' and xy.

Similarly, we will get four Boolean sum terms by combining two variables x and y with logical OR operation. These Boolean sum terms are called as **Max terms** or **standard sum terms**. The Max terms are x + y, x + y', x' + y and x' + y'.

The following table shows the representation of min terms and MAX terms for 2 variables.

x	y	Min terms	Max terms
0	0	m_0=x'y'	M_0=x + y
0	1	m_1=x'y	M_1=x + y'
1	0	m_2=xy'	M_2=x' + y
1	1	m_3=xy	M_3=x' + y'

If the binary variable is '0', then it is represented as complement of variable in min term and as the variable itself in Max term. Similarly, if the binary variable is '1', then it is represented as complement of variable in Max term and as the variable itself in min term.

From the above table, we can easily notice that min terms and Max terms are complement of each other. If there are 'n' Boolean variables, then there will be 2^n min terms and 2^n Max terms.

Canonical SoP and PoS forms

A truth table consists of a set of inputs and outputs. If there are 'n' input variables, then there will be 2^n possible combinations with zeros and ones. So the value of each output variable depends on the combination of input variables. So, each output variable will have '1' for some combination of input variables and '0' for some other combination of input variables.

Therefore, we can express each output variable in following two ways.

- Canonical SoP form

- Canonical PoS form

Canonical SoP form

Canonical SoP form means Canonical Sum of Products form. In this form, each product term contains all literals. So, these product terms are nothing but the min terms. Hence, canonical SoP form is also called as **sum of min terms** form.

First, identify the min terms for which, the output variable is one and then do the logical OR of those min terms in order to get the Boolean expression function corresponding to that output variable. This Boolean function will be in the form of sum of min terms.

Follow the same procedure for other output variables also, if there is more than one output variable.

Example

Consider the following **truth table**.

Inputs		Output	
p	**q**	**r**	**f**
0	0	0	0
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	1

Here, the output ff is '1' for four combinations of inputs. The corresponding min terms are p'qr, pq'r, pqr', pqr. By doing logical OR of these four min terms, we will get the Boolean function of output ff.

Therefore, the Boolean function of output is, f = p'qr + pq'r + pqr' + pqr. This is the **canonical SoP form** of output, f. We can also represent this function in following two notations.

$$f=m3+m5+m6+m7$$

$$f=\sum m(3,5,6,7)$$

In one equation, we represented the function as sum of respective min terms. In other equation, we used the symbol for summation of those min terms.

Canonical PoS form

Canonical PoS form means Canonical Product of Sums form. In this form, each sum term contains all literals. So, these sum terms are nothing but the Max terms. Hence, canonical PoS form is also called as **product of Max terms** form.

First, identify the Max terms for which, the output variable is zero and then do the logical AND of those Max terms in order to get the Boolean expression function corresponding to that output variable. This Boolean function will be in the form of product of Max terms.

Follow the same procedure for other output variables also, if there is more than one output variable.

Example

Consider the same truth table of previous example. Here, the output ff is '0' for four combinations of inputs. The corresponding Max terms are p + q + r, p + q + r', p + q' + r, p' + q + r. By doing logical AND of these four Max terms, we will get the Boolean function of output f.

Therefore, the Boolean function of output is, f = p+q+r.p+q+r′.p+q′+r.p′+q+r. This is the **canonical PoS form** of output, f. We can also represent this function in following two notations.

$$f=M0.M1.M2.M4$$

$$f=\prod M(0,1,2,4)$$

In one equation, we represented the function as product of respective Max terms. In other equation, we used the symbol for multiplication of those Max terms.

The Boolean function, f = p+q+r. p+q+r′. p+q′+r. p′+q+r is the dual of the Boolean function, f = p'qr + pq'r + pqr' + pqr.

Therefore, both canonical SoP and canonical PoS forms are **Dual** to each other. Functionally, these two forms are same. Based on the requirement, we can use one of these two forms.

Standard SoP and PoS forms

We discussed two canonical forms of representing the Boolean outputss. Similarly, there are two standard forms of representing the Boolean outputss. These are the simplified version of canonical forms.

- Standard SoP form
- Standard PoS form

We will discuss about Logic gates in later chapters. The main **advantage** of standard forms is that the number of inputs applied to logic gates can be minimized. Sometimes, there will be reduction in the total number of logic gates required.

Standard SoP form

Standard SoP form means **Standard Sum of Products** form. In this form, each product term need not contain all literals. So, the product terms may or may not be the min terms. Therefore, the Standard SoP form is the simplified form of canonical SoP form.

We will get Standard SoP form of output variable in two steps.

- Get the canonical SoP form of output variable
- Simplify the above Boolean function, which is in canonical SoP form.

Follow the same procedure for other output variables also, if there is more than one output variable. Sometimes, it may not possible to simplify the canonical SoP form. In that case, both canonical and standard SoP forms are same.

Example

Convert the following Boolean function into Standard SoP form.

$$f = p'qr + pq'r + pqr' + pqr$$

The given Boolean function is in canonical SoP form. Now, we have to simplify this Boolean function in order to get standard SoP form.

Step 1 − Use the **Boolean postulate**, x + x = x. That means, the Logical OR operation with any Boolean variable 'n' times will be equal to the same variable. So, we can write the last term pqr two more times.

$$\Rightarrow f = p'qr + pq'r + pqr' + pqr + pqr + pqr$$

Step 2 − Use **Distributive law** for 1st and 4th terms, 2nd and 5th terms, 3rd and 6th terms.

$$\Rightarrow f = qr(p'+p) + pr(q'+q) + pq(r'+r)$$

Step 3 − Use **Boolean postulate**, x + x' = 1 for simplifying the terms present in each parenthesis.

$$\Rightarrow f = qr \times 1 + pr \times 1 + pq \times 1$$

Step 4 − Use **Boolean postulate**, x.1 = x for simplifying above three terms.

$$\Rightarrow f = qr + pr + pq$$

$$\Rightarrow f = pq + qr + pr$$

This is the simplified Boolean function. Therefore, the **standard SoP form** corresponding to given canonical SoP form is **f = pq + qr + pr**

Standard PoS form

Standard PoS form means **Standard Product of Sums** form. In this form, each sum term need not contain all literals. So, the sum terms may or may not be the Max terms. Therefore, the Standard PoS form is the simplified form of canonical PoS form.

We will get Standard PoS form of output variable in two steps.

- Get the canonical PoS form of output variable

- Simplify the above Boolean function, which is in canonical PoS form.

Follow the same procedure for other output variables also, if there is more than one output variable. Sometimes, it may not possible to simplify the canonical PoS form. In that case, both canonical and standard PoS forms are same.

Example: Convert the following Boolean function into Standard PoS form.

$$f = p+q+r \,.\, p+q+r'.\, p+q'+r \,.\, p'+q+r$$

The given Boolean function is in canonical PoS form. Now, we have to simplify this Boolean function in order to get standard PoS form.

Step 1 − Use the **Boolean postulate**, x.x = x. That means, the Logical AND operation with any Boolean variable 'n' times will be equal to the same variable. So, we can write the first term p+q+r two more times.

$$\Rightarrow f = p+q+r.p+q+r.p+q+r.p+q+r'.p+q'+r.p'+q+r$$

Step 2 − Use **Distributive law,** (x + y).z = (x+y).(x+z) for 1st and 4th parenthesis, 2nd and 5th parenthesis, 3rd and 6th parenthesis.

$$\Rightarrow f = p+q+rr'.\, p+r+qq'.q+r+pp'$$

Step 3 − Use **Boolean postulate**, x.x'=0 for simplifying the terms present in each parenthesis.

$$\Rightarrow f = p+q+0 \,.\, p+r+0 \,.\, q+r+0$$

Step 4 − Use **Boolean postulate**, x + 0 = x for simplifying the terms present in each parenthesis

$$\Rightarrow f = p+q.p+r.q+r$$
$$\Rightarrow f = p+q.q+r.p+r$$

This is the simplified Boolean function. Therefore, the **standard PoS form** corresponding to given canonical PoS form is **f =** p+q **.** q+r **.** p+r. This is the **dual** of the Boolean function, f = pq + qr + pr.

Therefore, both Standard SoP and Standard PoS forms are Dual to each other.

Karnaugh map (3 & 4 Variables) &Minimization of logical expressions, don't care conditions:

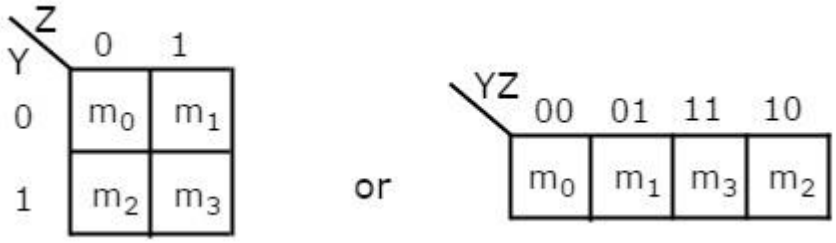
Karnaugh introduced a method for simplification of Boolean functions in an easy way. This method is known as Karnaugh map method or K-map method. It is a graphical method, which consists of 2^n cells for 'n' variables. The adjacent cells are differed only in single bit position.

K-Maps for 2 to 4 Variables

K-Map method is most suitable for minimizing Boolean functions of 2 variables to 4 variables. Now, let us discuss about the K-Maps for 2 to 4 variables one by one.

2 Variable K-Map

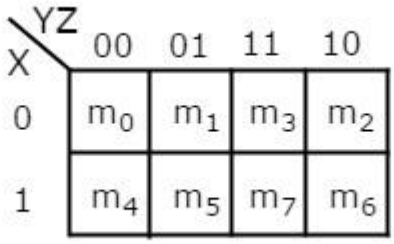
The number of cells in 2 variable K-map is four, since the number of variables is two. The following figure shows **2 variable K-Map**.



- There is only one possibility of grouping 4 adjacent min terms.
- The possible combinations of grouping 2 adjacent min terms are {(m_0, m_1), (m_2, m_3), (m_0, m_2) and (m_1, m_3)}.

3 Variable K-Map

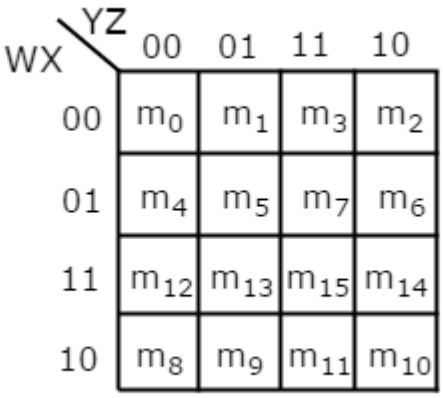
The number of cells in 3 variable K-map is eight, since the number of variables is three. The following figure shows **3 variable K-Map**.



- There is only one possibility of grouping 8 adjacent min terms.
- The possible combinations of grouping 4 adjacent min terms are {(m_0, m_1, m_3, m_2), (m_4, m_5, m_7, m_6), (m_0, m_1, m_4, m_5), (m_1, m_3, m_5, m_7), (m_3, m_2, m_7, m_6) and (m_2, m_0, m_6, m_4)}.
- The possible combinations of grouping 2 adjacent min terms are {(m_0, m_1), (m_1, m_3), (m_3, m_2), (m_2, m_0), (m_4, m_5), (m_5, m_7), (m_7, m_6), (m_6, m_4), (m_0, m_4), (m_1, m_5), (m_3, m_7) and (m_2, m_6)}.
- If x=0, then 3 variable K-map becomes 2 variable K-map.

4 Variable K-Map

The number of cells in 4 variable K-map is sixteen, since the number of variables is four. The following figure shows **4 variable K-Map**.



- There is only one possibility of grouping 16 adjacent min terms.
- Let R_1, R_2, R_3 and R_4 represents the min terms of first row, second row, third row and fourth row respectively. Similarly, C_1, C_2, C_3 and C_4 represents the min terms of first column, second column, third column and fourth column respectively. The possible combinations of grouping 8 adjacent min terms are $\{(R_1, R_2), (R_2, R_3), (R_3, R_4), (R_4, R_1), (C_1, C_2), (C_2, C_3), (C_3, C_4), (C_4, C_1)\}$.
- If w=0, then 4 variable K-map becomes 3 variable K-map.

Minimization of Boolean Functions using K-Maps

If we consider the combination of inputs for which the Boolean function is '1', then we will get the Boolean function, which is in **standard sum of products** form after simplifying the K-map.

Similarly, if we consider the combination of inputs for which the Boolean function is '0', then we will get the Boolean function, which is in **standard product of sums** form after simplifying the K-map.

Follow these **rules for simplifying K-maps** in order to get standard sum of products form.

- Select the respective K-map based on the number of variables present in the Boolean function.
- If the Boolean function is given as sum of min terms form, then place the ones at respective min term cells in the K-map. If the Boolean function is given as sum of products form, then place the ones in all possible cells of K-map for which the given product terms are valid.
- Check for the possibilities of grouping maximum number of adjacent ones. It should be powers of two. Start from highest power of two and upto least power of two. Highest power is equal to the number of variables considered in K-map and least power is zero.
- Each grouping will give either a literal or one product term. It is known as **prime implicant**. The prime implicant is said to be **essential prime implicant**, if atleast single '1' is not covered with any other groupings but only that grouping covers.

- Note down all the prime implicants and essential prime implicants. The simplified Boolean function contains all essential prime implicants and only the required prime implicants.

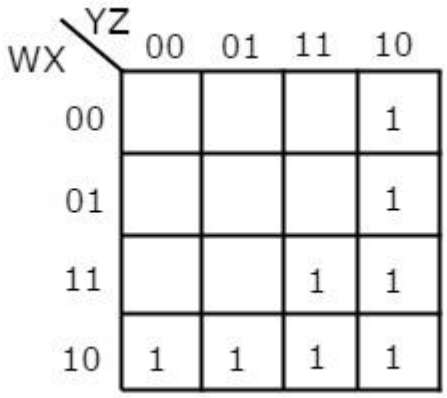
Note 1 − If outputs are not defined for some combination of inputs, then those output values will be represented with **don't care symbol 'x'**. That means, we can consider them as either '0' or '1'.

Note 2 − If don't care terms also present, then place don't cares 'x' in the respective cells of K-map. Consider only the don't cares 'x' that are helpful for grouping maximum number of adjacent ones. In those cases, treat the don't care value as '1'.

Example

Let us **simplify** the following Boolean function, **f(**W,X,Y,Z)= **WX'Y' + WY + W'YZ'** using K-map.

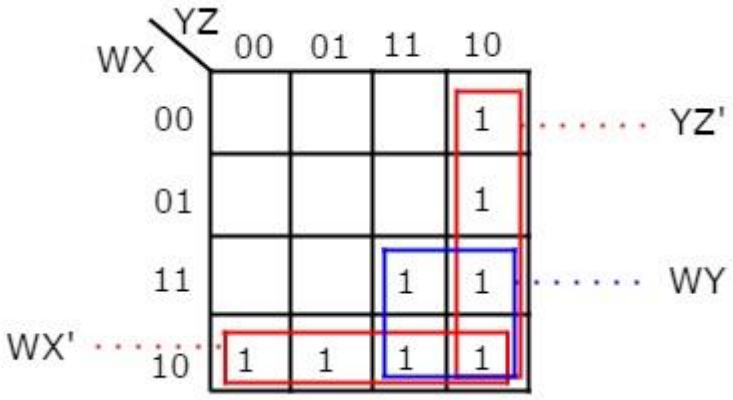
The given Boolean function is in sum of products form. It is having 4 variables W, X, Y & Z. So, we require **4 variable K-map**. The **4 variable K-map** with ones corresponding to the given product terms is shown in the following figure.



Here, 1s are placed in the following cells of K-map.

- The cells, which are common to the intersection of Row 4 and columns 1 & 2 are corresponding to the product term, **WX'Y'**.
- The cells, which are common to the intersection of Rows 3 & 4 and columns 3 & 4 are corresponding to the product term, **WY**.
- The cells, which are common to the intersection of Rows 1 & 2 and column 4 are corresponding to the product term, **W'YZ'**.

There are no possibilities of grouping either 16 adjacent ones or 8 adjacent ones. There are three possibilities of grouping 4 adjacent ones. After these three groupings, there is no single one left as ungrouped. So, we no need to check for grouping of 2 adjacent ones. The **4 variable K-map** with these three **groupings** is shown in the following figure.



Here, we got three prime implicants WX', WY & YZ'. All these prime implicants are **essential** because of following reasons.

- Two ones **(m_8 & m_9)** of fourth row grouping are not covered by any other groupings. Only fourth row grouping covers those two ones.
- Single one **(m_{15})** of square shape grouping is not covered by any other groupings. Only the square shape grouping covers that one.
- Two ones **(m_2 & m_6)** of fourth column grouping are not covered by any other groupings. Only fourth column grouping covers those two ones.

Therefore, the **simplified Boolean function** is

$$f = WX' + WY + YZ'$$

Follow these **rules for simplifying K-maps** in order to get standard product of sums form.

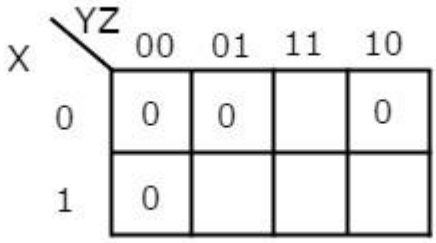
- Select the respective K-map based on the number of variables present in the Boolean function.
- If the Boolean function is given as product of Max terms form, then place the zeroes at respective Max term cells in the K-map. If the Boolean function is given as product of sums form, then place the zeroes in all possible cells of K-map for which the given sum terms are valid.
- Check for the possibilities of grouping maximum number of adjacent zeroes. It should be powers of two. Start from highest power of two and upto least power of two. Highest power is equal to the number of variables considered in K-map and least power is zero.
- Each grouping will give either a literal or one sum term. It is known as **prime implicant**. The prime implicant is said to be **essential prime implicant**, if atleast single '0' is not covered with any other groupings but only that grouping covers.
- Note down all the prime implicants and essential prime implicants. The simplified Boolean function contains all essential prime implicants and only the required prime implicants.

Note − If don't care terms also present, then place don't cares 'x' in the respective cells of K-map. Consider only the don't cares 'x' that are helpful for grouping maximum number of adjacent zeroes. In those cases, treat the don't care value as '0'.

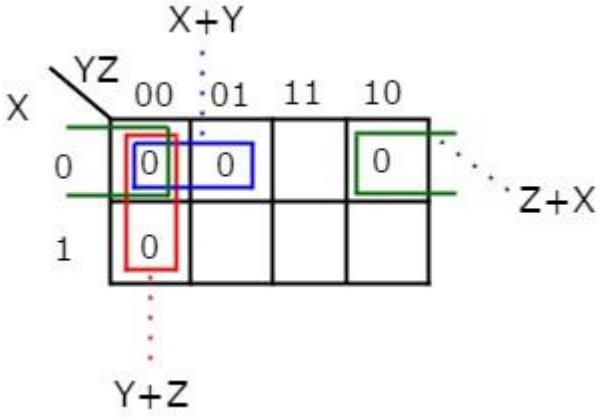
Example

Let us **simplify** the following Boolean function, f(X,Y,Z)=∏M(0,1,2,4) using K-map.

The given Boolean function is in product of Max terms form. It is having 3 variables X, Y & Z. So, we require 3 variable K-map. The given Max terms are M_0, M_1, M_2 & M_4. The 3 **variable K-map** with zeroes corresponding to the given Max terms is shown in the following figure.



There are no possibilities of grouping either 8 adjacent zeroes or 4 adjacent zeroes. There are three possibilities of grouping 2 adjacent zeroes. After these three groupings, there is no single zero left as ungrouped. The **3 variable K-map** with these three **groupings** is shown in the following figure.



Here, we got three prime implicants X + Y, Y + Z & Z + X. All these prime implicants are **essential** because one zero in each grouping is not covered by any other groupings except with their individual groupings.

Therefore, the **simplified Boolean function** is

$$\mathbf{f} = X+YX+Y.Y+ZY+Z.Z+XZ+X$$

Unit-2
Combinational logic circuits

Binary Adder

The most basic arithmetic operation is addition. The circuit, which performs the addition of two binary numbers is known as **Binary adder**. First, let us implement an adder, which performs the addition of two bits.

Half Adder

Half adder is a combinational circuit, which performs the addition of two binary numbers A and B are of **single bit**. It produces two outputs sum, S & carry, C.

The **Truth table** of Half adder is shown below.

Inputs		**Outputs**	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

When we do the addition of two bits, the resultant sum can have the values ranging from 0 to 2 in decimal. We can represent the decimal digits 0 and 1 with single bit in binary. But, we can't represent decimal digit 2 with single bit in binary. So, we require two bits for representing it in binary.

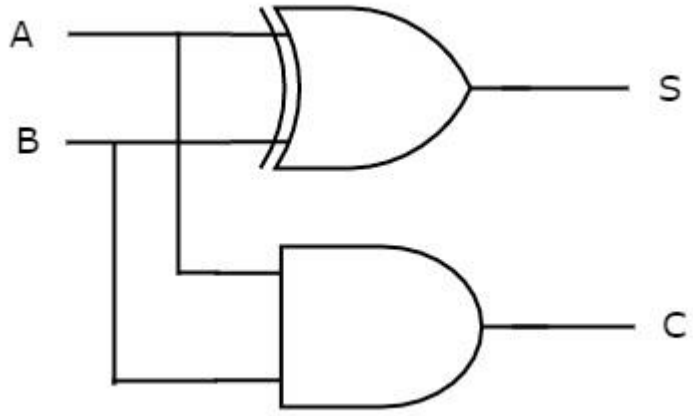
Let, sum, S is the Least significant bit and carry, C is the Most significant bit of the resultant sum. For first three combinations of inputs, carry, C is zero and the value of S will be either zero or one based on the **number of ones** present at the inputs. But, for last combination of inputs, carry, C is one and sum, S is zero, since the resultant sum is two.

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

$$S=A \oplus B$$

$$C=AB$$

We can implement the above functions with 2-input Ex-OR gate & 2-input AND gate. The **circuit diagram** of Half adder is shown in the following figure.



In the above circuit, a two input Ex-OR gate & two input AND gate produces sum, S & carry, C respectively. Therefore, Half-adder performs the addition of two bits.

Full Adder:

Full adder is a combinational circuit, which performs the **addition of three bits** A, B and C_{in}. Where, A & B are the two parallel significant bits and C_{in} is the carry bit, which is generated from previous stage. This Full adder also produces two outputs sum, S & carry, C_{out}, which are similar to Half adder.

The **Truth table** of Full adder is shown below.

Inputs			**Outputs**	
A	**B**	**C_{in}**	**C_{out}**	**S**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

When we do the addition of three bits, the resultant sum can have the values ranging from 0 to 3 in decimal. We can represent the decimal digits 0 and 1 with single bit in binary. But, we can't represent the decimal digits 2 and 3 with single bit in binary. So, we require two bits for representing those two decimal digits in binary.

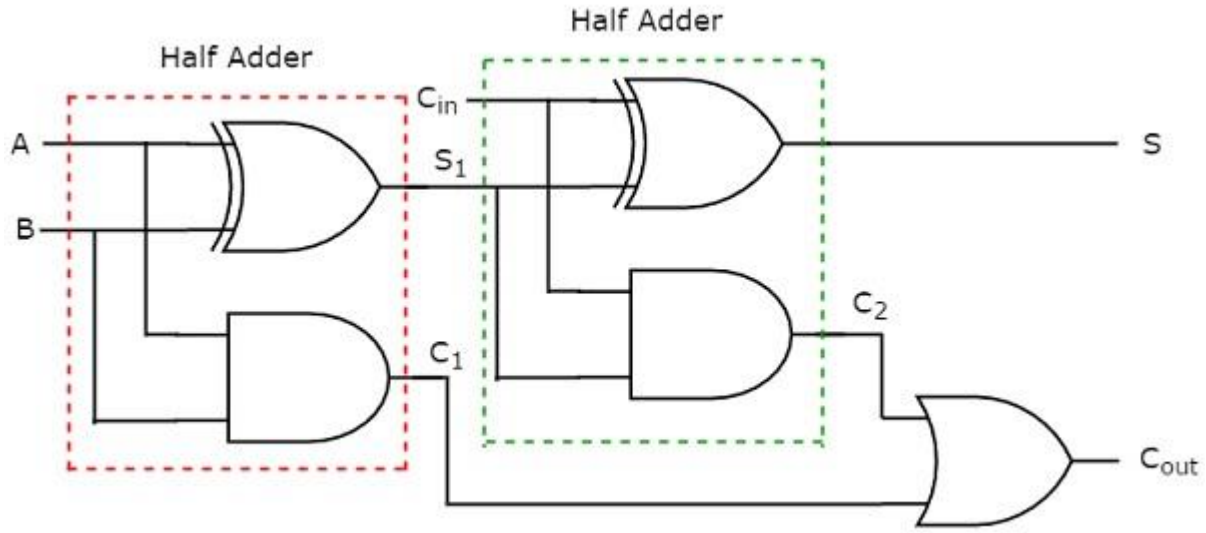
Let, sum, S is the Least significant bit and carry, C_{out} is the Most significant bit of resultant sum. It is easy to fill the values of outputs for all combinations of inputs in the truth table. Just count the **number of ones** present at the inputs and write the equivalent binary number at outputs. If C_{in} is equal to zero, then Full adder truth table is same as that of Half adder truth table.

We will get the following **Boolean functions** for each output after simplification.

$$S=A\oplus B\oplus Cin$$

$$cout=AB+(A\oplus B)cin$$

The sum, S is equal to one, when odd number of ones present at the inputs. We know that Ex-OR gate produces an output, which is an odd function. So, we can use either two 2input Ex-OR gates or one 3-input Ex-OR gate in order to produce sum, S. We can implement carry, C_{out} using two 2-input AND gates & one OR gate. The **circuit diagram** of Full adder is shown in the following figure.



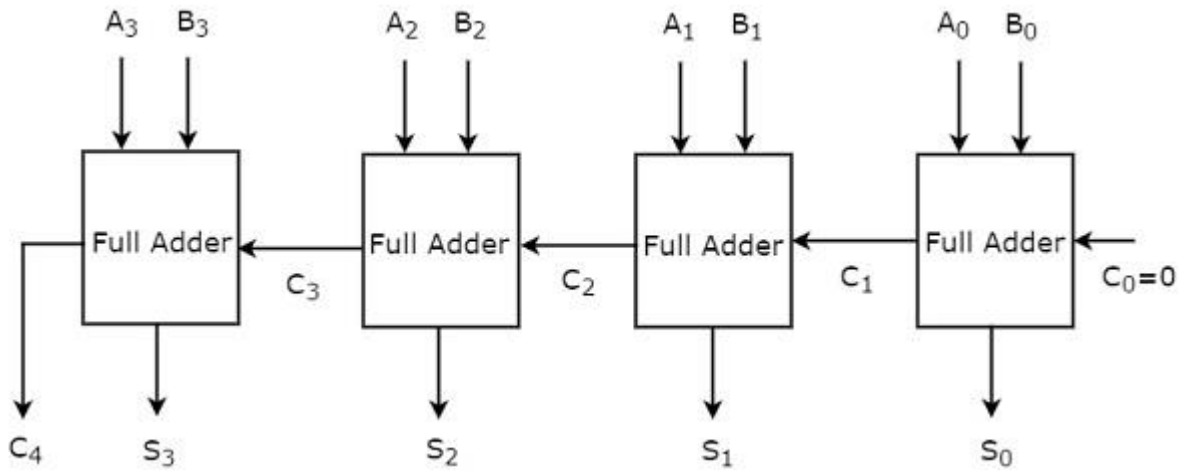
This adder is called as **Full adder** because for implementing one Full adder, we require two Half adders and one OR gate. If C_{in} is zero, then Full adder becomes Half adder. We can verify it easily from the above circuit diagram or from the Boolean functions of outputs of Full adder.

4-bit Binary Adder

The 4-bit binary adder performs the **addition of two 4-bit numbers**. Let the 4-bit binary numbers, A=A3A2A1A0 and B=B3B2B1B0. We can implement 4-bit binary adder in one of the two following ways.

- Use one Half adder for doing the addition of two Least significant bits and three Full adders for doing the addition of three higher significant bits.
- Use four Full adders for uniformity. Since, initial carry C_{in} is zero, the Full adder which is used for adding the least significant bits becomes Half adder.

For the time being, we considered second approach. The **block diagram** of 4-bit binary adder is shown in the following figure.



Here, the 4 Full adders are cascaded. Each Full adder is getting the respective bits of two parallel inputs A & B. The carry output of one Full adder will be the carry input of subsequent higher order Full adder. This 4-bit binary adder produces the resultant sum having at most 5 bits. So, carry out of last stage Full adder will be the MSB.

In this way, we can implement any higher order binary adder just by cascading the required number of Full adders. This binary adder is also called as **ripple carry** binary **adder** because the carry propagates ripples from one stage to the next stage.

Binary Subtractor

The circuit, which performs the subtraction of two binary numbers is known as **Binary subtractor**. We can implement Binary subtractor in following two methods.

- Cascade Full subtractors
- 2's complement method

In first method, we will get an n-bit binary subtractor by cascading 'n' Full subtractors. So, first you can implement Half subtractor and Full subtractor, similar to Half adder & Full adder. Then, you can implement an n-bit binary subtractor, by cascading 'n' Full subtractors. So, we will be having two separate circuits for binary addition and subtraction of two binary numbers.

In second method, we can use same binary adder for subtracting two binary numbers just by doing some modifications in the second input. So, internally binary addition operation takes place but, the output is resultant subtraction.

We know that the subtraction of two binary numbers A & B can be written as,

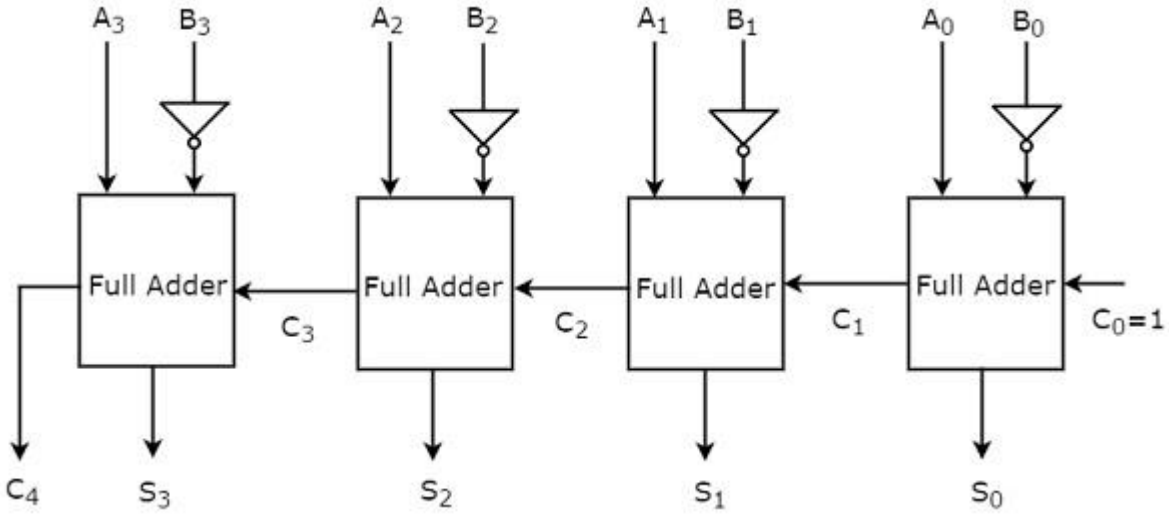
$$A-B=A+(2'\text{scomplimentof B})$$

$$\Rightarrow A-B=A+(1'\text{scomplimentof B})+1$$

4-bit Binary Subtractor

The 4-bit binary subtractor produces the **subtraction of two 4-bit numbers**. Let the 4bit binary numbers, A=A3A2A1A0 and B=B3B2B1B0. Internally, the operation of 4-bit Binary subtractor is similar to that of 4-bit Binary adder. If the normal bits of binary number

A, complemented bits of binary number B and initial carry borrow, C_{in} as one are applied to 4-bit Binary adder, then it becomes 4-bit Binary subtractor. The **block diagram** of 4-bit binary subtractor is shown in the following figure.



This 4-bit binary subtractor produces an output, which is having at most 5 bits. If Binary number A is greater than Binary number B, then MSB of the output is zero and the remaining bits hold the magnitude of A-B. If Binary number A is less than Binary number B, then MSB of the output is one. So, take the 2's complement of output in order to get the magnitude of A-B.

Binary Adder / Subtractor

The circuit, which can be used to perform either addition or subtraction of two binary numbers at any time is known as **Binary Adder / subtractor**. Both, Binary adder and Binary subtractor contain a set of Full adders, which are cascaded. The input bits of binary number A are directly applied in both Binary adder and Binary subtractor.

There are two differences in the inputs of Full adders that are present in Binary adder and Binary subtractor.

- The input bits of binary number B are directly applied to Full adders in Binary adder, whereas the complemented bits of binary number B are applied to Full adders in Binary subtractor.
- The initial carry, $C_0 = 0$ is applied in 4-bit Binary adder, whereas the initial carry borrow, $C_0 = 1$ is applied in 4-bit Binary subtractor.

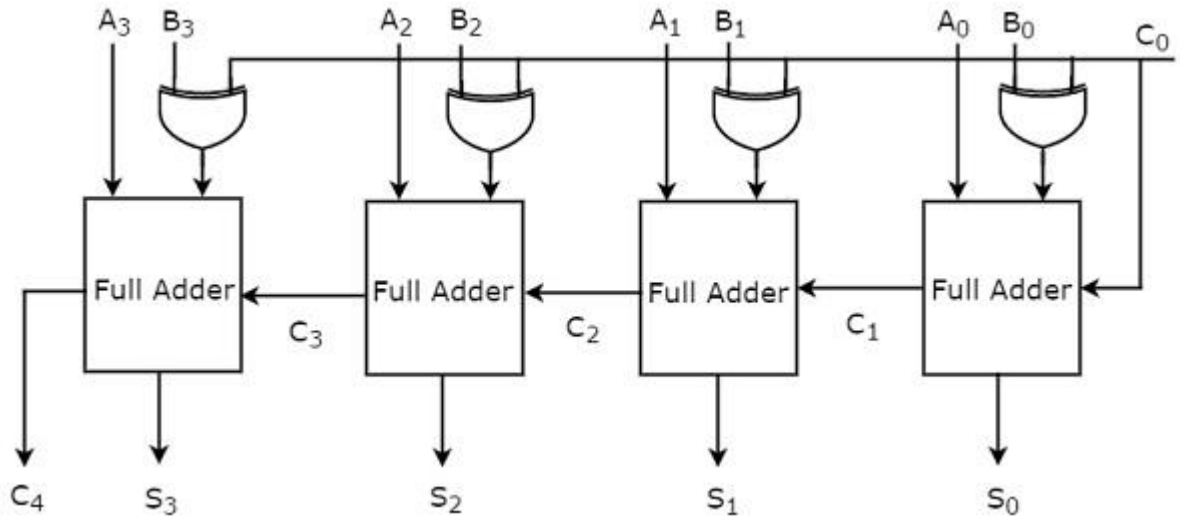
We know that a **2-input Ex-OR gate** produces an output, which is same as that of first input when other input is zero. Similarly, it produces an output, which is complement of first input when other input is one.

Therefore, we can apply the input bits of binary number B, to 2-input Ex-OR gates. The other input to all these Ex-OR gates is C_0. So, based on the value of C_0, the Ex-OR gates produce either the normal or complemented bits of binary number B.

4-bit Binary Adder / Subtractor

The 4-bit binary adder / subtractor produces either the addition or the subtraction of two 4-bit numbers based on the value of initial carry or borrow, C_0. Let the 4-bit binary numbers, A=A3A2A1A0 and B=B3B2B1B0. The operation of 4-bit Binary adder / subtractor is similar to that of 4-bit Binary adder and 4-bit Binary subtractor.

Apply the normal bits of binary numbers A and B & initial carry or borrow, C_0 from externally to a 4-bit binary adder. The **block diagram** of 4-bit binary adder / subtractor is shown in the following figure.



If initial carry, C_0 is zero, then each full adder gets the normal bits of binary numbers A & B. So, the 4-bit binary adder / subtractor produces an output, which is the **addition of two binary numbers** A & B.

If initial borrow, C_0 is one, then each full adder gets the normal bits of binary number A & complemented bits of binary number B. So, the 4-bit binary adder / subtractor produces an output, which is the **subtraction of two binary numbers** A & B.

Therefore, with the help of additional Ex-OR gates, the same circuit can be used for both addition and subtraction of two binary numbers.

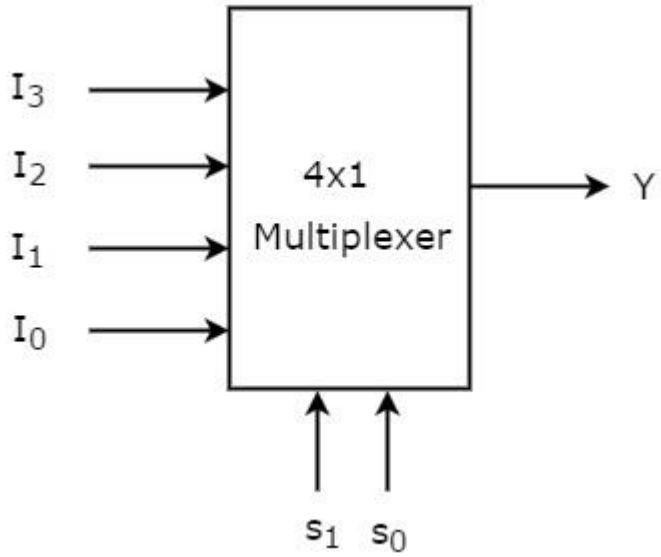
Multiplexer (4:1):

Multiplexer is a combinational circuit that has maximum of 2^n data inputs, 'n' selection lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines.

Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination will select only one data input. Multiplexer is also called as **Mux**.

4x1 Multiplexer

4x1 Multiplexer has four data inputs I_3, I_2, I_1 & I_0, two selection lines s_1 & s_0 and one output Y. The **block diagram** of 4x1 Multiplexer is shown in the following figure.



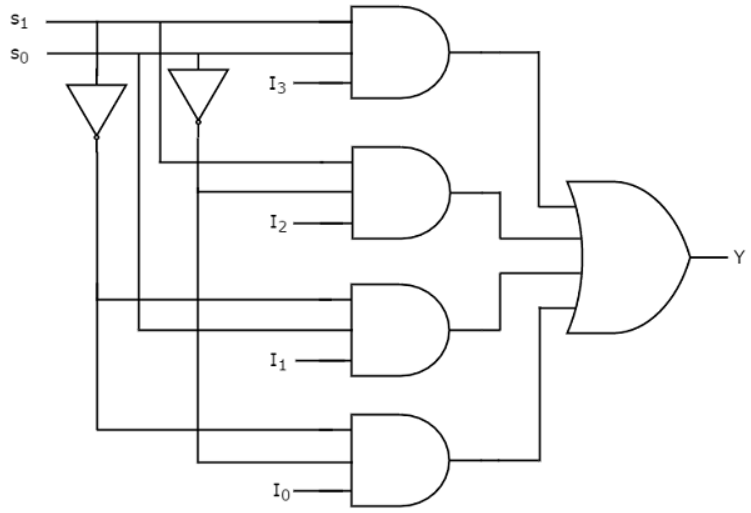
One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. **Truth table** of 4x1 Multiplexer is shown below.

Selection Lines		**Output**
S_1	**S_0**	**Y**
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

From Truth table, we can directly write the **Boolean function** for output, Y as

$$Y=S1'S0'I0+S1'S0I1+S1S0'I2+S1S0I3$$

We can implement this Boolean function using Inverters, AND gates & OR gate. The **circuit diagram** of 4x1 multiplexer is shown in the following figure.



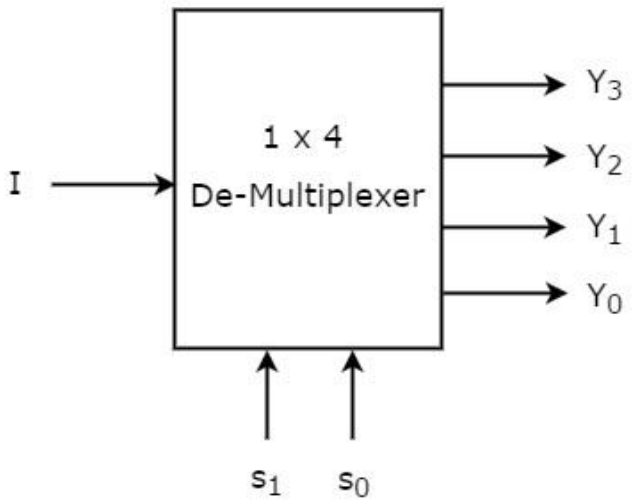
De- multiplexer (1:4):

De-Multiplexer is a combinational circuit that performs the reverse operation of Multiplexer. It has single input, 'n' selection lines and maximum of 2^n outputs. The input will be connected to one of these outputs based on the values of selection lines.

Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination can select only one output. De-Multiplexer is also called as **De-Mux**.

1x4 De-Multiplexer

1x4 De-Multiplexer has one input I, two selection lines, s_1 & s_0 and four outputs Y_3, Y_2, Y_1 &Y_0. The **block diagram** of 1x4 De-Multiplexer is shown in the following figure.



The single input 'I' will be connected to one of the four outputs, Y_3 to Y_0 based on the values of selection lines s_1 & s0. The **Truth table** of 1x4 De-Multiplexer is shown below.

Selection Inputs		Outputs			
S_1	**S_0**	**Y_3**	**Y_2**	**Y_1**	**Y_0**
0	0	0	0	0	**I**

0	1	0	0	**I**	0
1	0	0	**I**	0	0
1	1	**I**	0	0	0

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

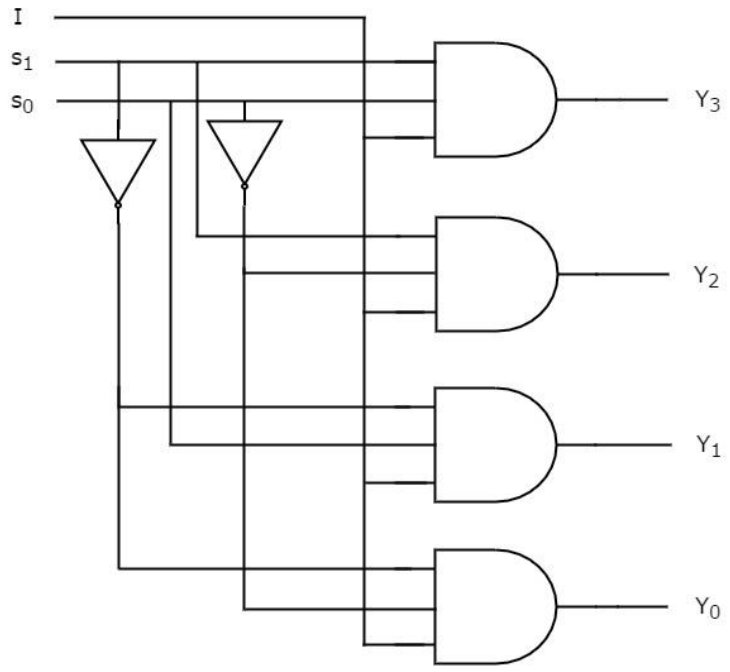
$$Y3=s1s0I$$

$$Y2=s1s0'I$$

$$Y1=s1's0I$$

$$Y0=s1's0'I$$

We can implement these Boolean functions using Inverters & 3-input AND gates. The **circuit diagram** of 1x4 De-Multiplexer is shown in the following figure.

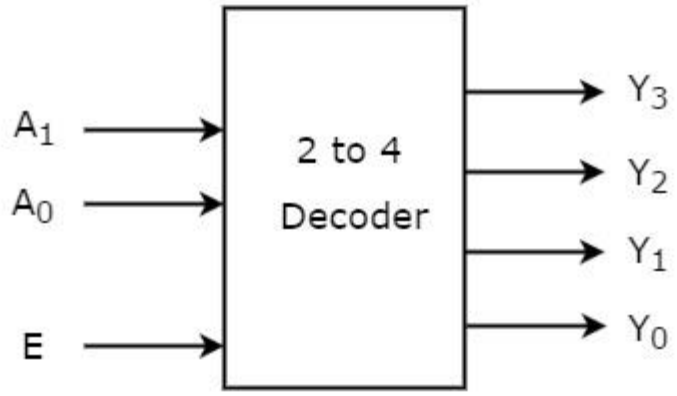


Decoder:

Decoder is a combinational circuit that has 'n' input lines and maximum of 2^n 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 lineslines, when it is enabled.

2 to 4 Decoder

Let 2 to 4 Decoder has two inputs A_1 & A_0 and four outputs Y_3, Y_2, Y_1 & Y_0. 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	**A_1**	**A_0**	**Y_3**	**Y_2**	**Y_1**	**Y_0**
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

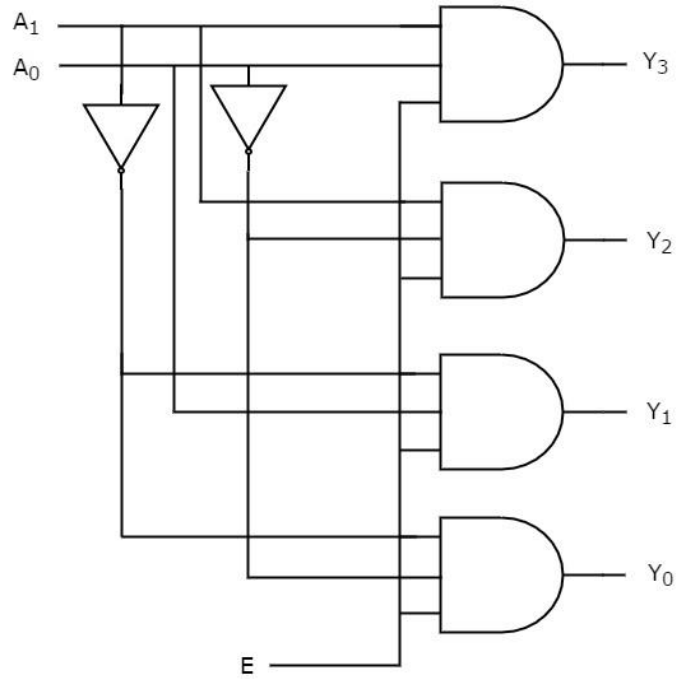
$$Y3=E.A1.A0$$

$$Y2=E.A1.A0'$$

$$Y1=E.A1'.A0$$

$$Y0=E.A1'.A0'$$

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.



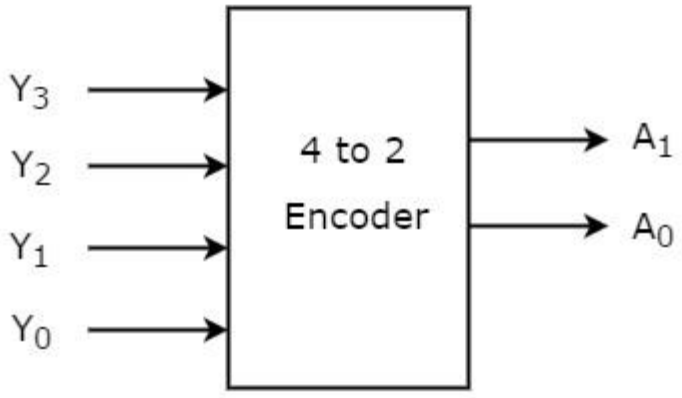
Therefore, the outputs of 2 to 4 decoder are nothing but the **min terms** of two input variables A_1 & A_0, when enable, E is equal to one. If enable, E is zero, then all the outputs of decoder will be equal to zero.

Encoder:

An **Encoder** is a combinational circuit that performs the reverse operation of Decoder. It has maximum of 2^n input lines and 'n' output lines. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes 2^n input lines with 'n' bits. It is optional to represent the enable signal in encoders.

4 to 2 Encoder

Let 4 to 2 Encoder has four inputs Y_3, Y_2, Y_1 & Y_0 and two outputs A_1 & A_0. The **block diagram** of 4 to 2 Encoder is shown in the following figure.



At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output. The **Truth table** of 4 to 2 encoder is shown below.

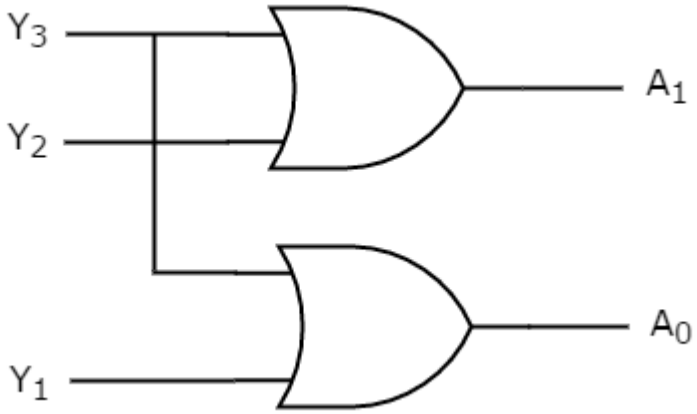
Inputs				Outputs	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

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

$$A1=Y3+Y2$$

$$A0=Y3+Y1$$

We can implement the above two Boolean functions by using two input OR gates. The **circuit diagram** of 4 to 2 encoder is shown in the following figure.



The above circuit diagram contains two OR gates. These OR gates encode the four inputs with two bits.

Digital comparator:

A magnitude digital comparator is a combinational circuit that compares two digital or binary numbers in order to find out whether one binary number is equal, less than or greater than the other binary number. We logically design a circuit for which we will have two inputs one for A and other for B and have three output terminals, one for A > B condition, one for A = B condition and one for A < B condition.

Figure-1: Block Diagram of Comparator

1-Bit Magnitude Comparator :

A comparator used to compare two bits is called a single bit comparator. It consists of two inputs each for two single bit numbers and three outputs to generate less than, equal to and greater than between two binary numbers. The truth table for a 1-bit comparator is given below :

A	B	A<B	A=B	A>B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Figure-2: Truth Table of 1-Bit Comparator

From the above truth table logical expressions for each output can be expressed as follows:

A > B : AB'

A < B : A'B

A = B : A'B' + AB

By using these Boolean expressions, we can implement a logic circuit for this comparator as given below :

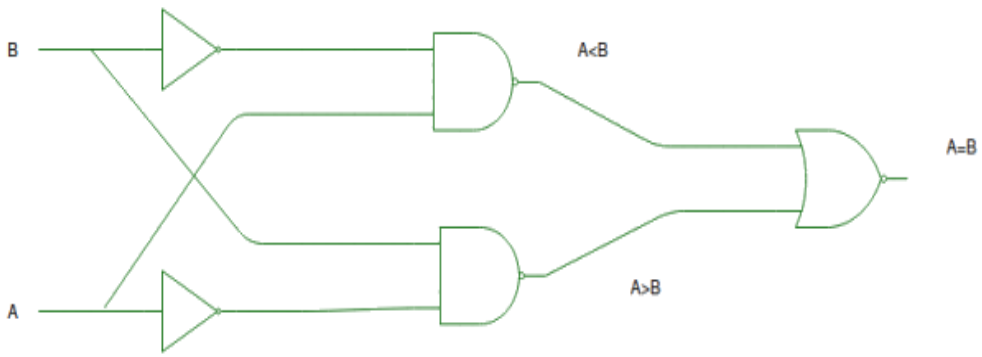


Figure-3: Logic Circuit of 1-Bit Comparator

2-Bit Magnitude Comparator :

A comparator used to compare two binary numbers each of two bits is called a 2-bit magnitude comparator. It consists of four inputs and three outputs to generate less than, equal to and greater than between two binary numbers.

The truth table for a 2-bit comparator is given below:

INPUT				OUTPUT		
A1	A0	B1	B0	A<B	A=B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

Figure-4: Truth Table of 2-Bit Comparator

From the above truth table logical expressions for each output can be expressed as follows:

$A > B : A_1B_1' + A_0B_1'B_0' + A_1A_0B_0'$

$A = B : A_1'A_0'B_1'B_0' + A_1'A_0B_1'B_0 + A_1A_0B_1B_0 + A_1A_0'B_1B_0'$

$: A_1'B_1' (A_0'B_0' + A_0B_0) + A_1B_1 (A_0B_0 + A_0'B_0')$

$: (A_0B_0 + A_0'B_0') (A_1B_1 + A_1'B_1')$

$: (A_0 \text{ Ex-Nor } B_0) (A_1 \text{ Ex-Nor } B_1)$

$A < B : A_1'B_1 + A_0'B_1B_0 + A_1'A0'B0$

By using these Boolean expressions, we can implement a logic circuit for this comparator as given below :

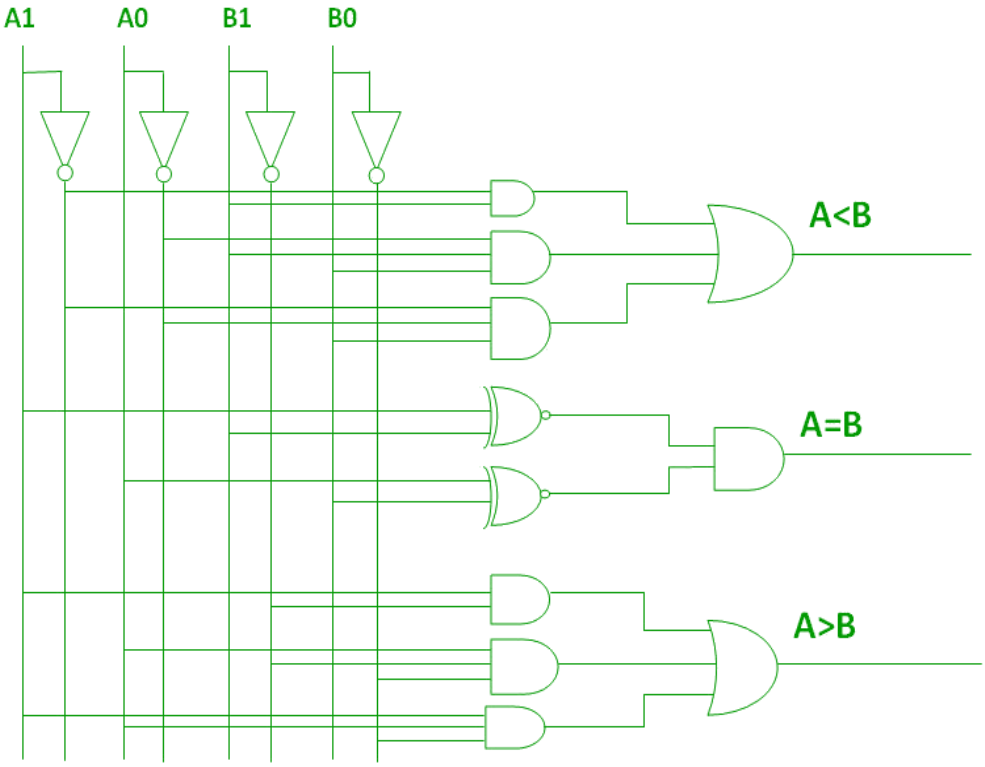


Figure-5: Logic Circuit of 2-Bit Comparator

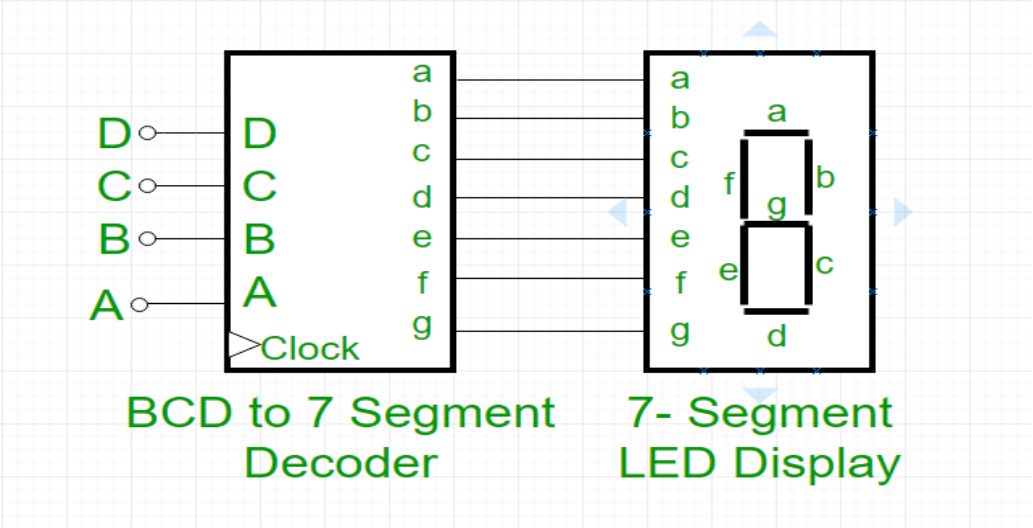
Seven segment Decoder:

In **Binary Coded Decimal (BCD)** encoding scheme each of the decimal numbers (0-9) is represented by its equivalent binary pattern (which is generally of 4-bits).
Whereas, **Seven segment** display is an electronic device which consists of seven Light Emitting Diodes (LEDs) arranged in a some definite pattern (common cathode or common anode type), which is used to display Hexadecimal numerals (in this case decimal numbers as input is BCD i.e., 0-9).
Two types of seven segment LED display:

1. **Common Cathode Type:** In this type of display all cathodes of the seven LEDs are connected together to the ground or -Vcc (hence,common cathode) and LED displays digits when some 'HIGH' signal is supplied to the individual anodes.
2. **Common Anode Type:** In this type of display all the anodes of the seven LEDs are connected to battery or +Vcc and LED displays digits when some 'LOW' signal is supplied to the individual cathodes.

But, seven segment display does not work by directly supplying voltage to different segments of LEDs. First, our decimal number is changed to its BCD equivalent signal then BCD to seven segment decoder converts those signals to the form which is fed to seven segment display.

This BCD to seven segment decoder has four input lines (A, B, C and D) and 7 output lines (a, b, c, d, e, f and g), this output is given to seven segment LED display which displays the decimal number depending upon inputs.

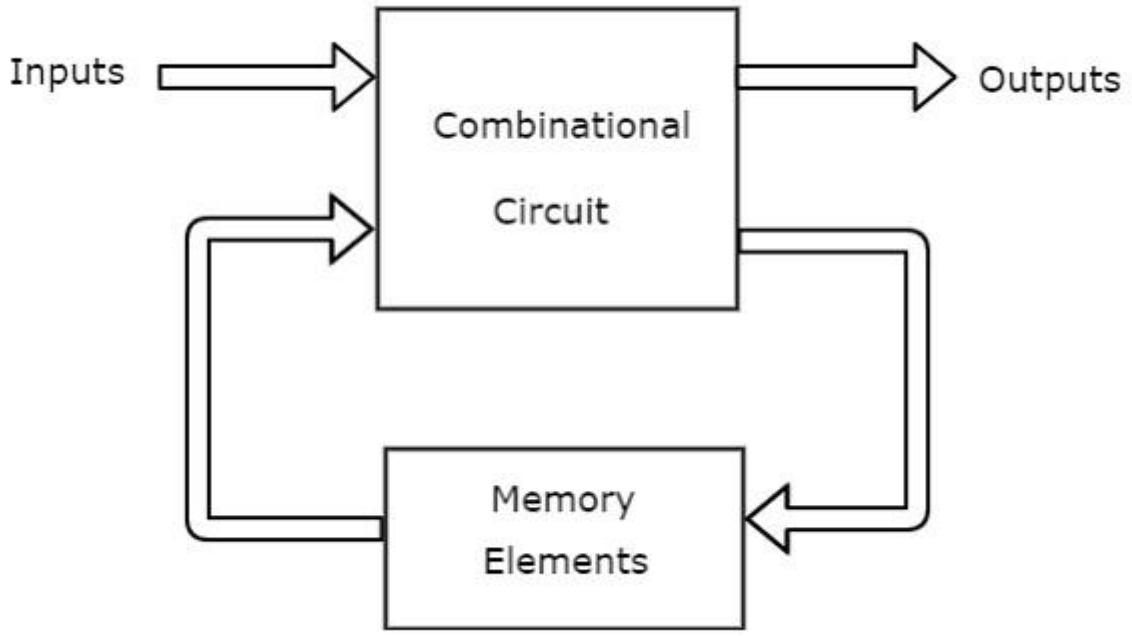


Truth Table – For common cathode type BCD to seven segment decoder:

A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

Unit-3
Sequential logic Circuits

The **block diagram** of sequential circuit is



This sequential circuit contains a set of inputs and outputs s. The outputs s of sequential circuit depends not only on the combination of present inputs but also on the previous outputs 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
Outputs depend only on present inputs.	Outputs depend on both present inputs and present state.
Feedback path is not present.	Feedback path is present.
Memory elements are not required.	Memory elements are required.
Clock signal is not required.	Clock signal is required.
Easy to design.	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:

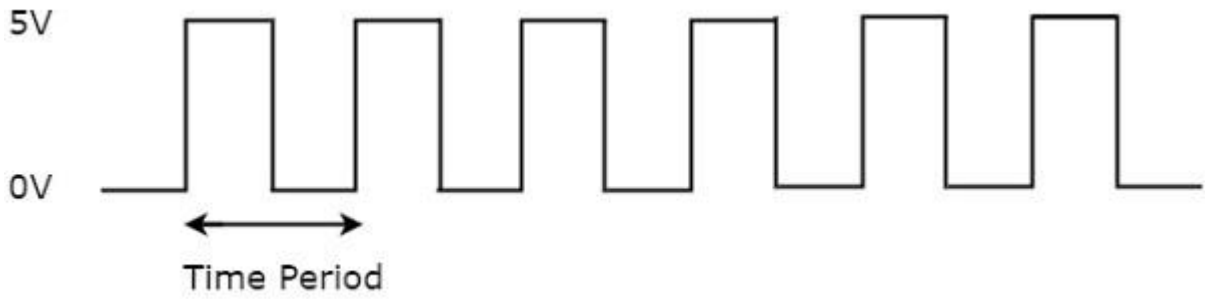
If all the outputs of a sequential circuit change affect with respect to active transition of clock signal, then that sequential circuit is called as **Synchronous sequential circuit**. That means, all the outputs of synchronous sequential circuits change affect at the same time. Therefore, the outputs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

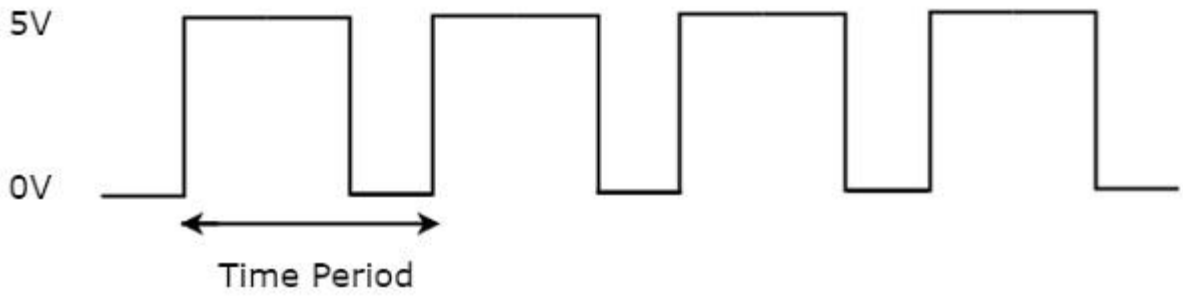
Clock signal

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 5V5V for some time and stays at logic Low 0V0V 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.

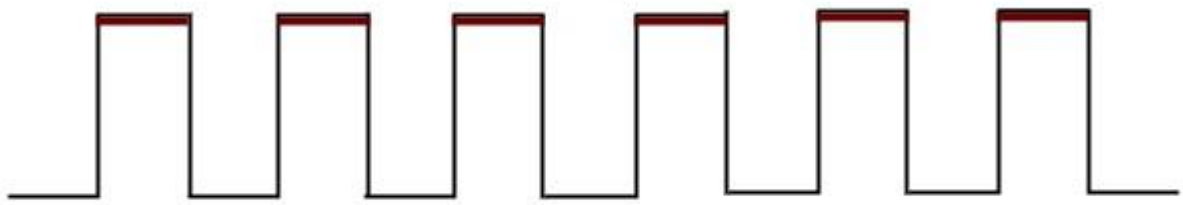
- Level triggering
- Edge triggering

Level triggering

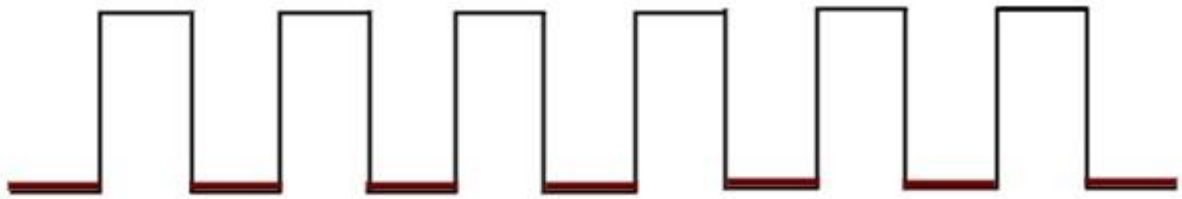
There are two levels, namely logic High and logic Low in clock signal. Following are the two **types of level triggering**.

- 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.



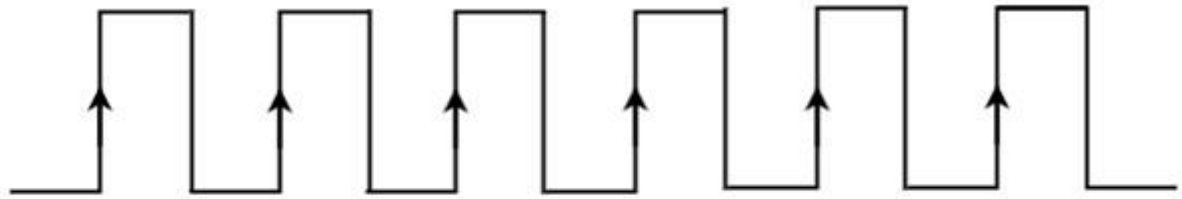
Edge triggering

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.

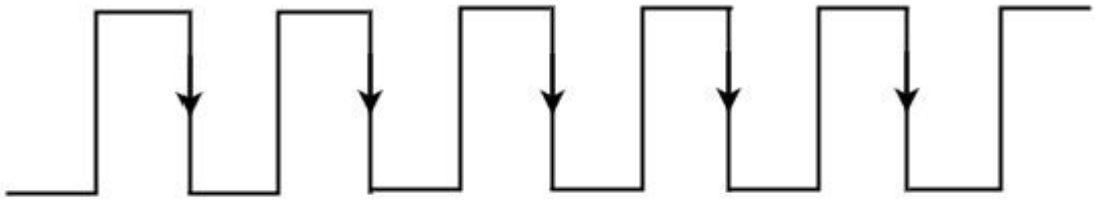
Following are the two **types of edge triggering** based on the transitions of clock signal.

- 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.



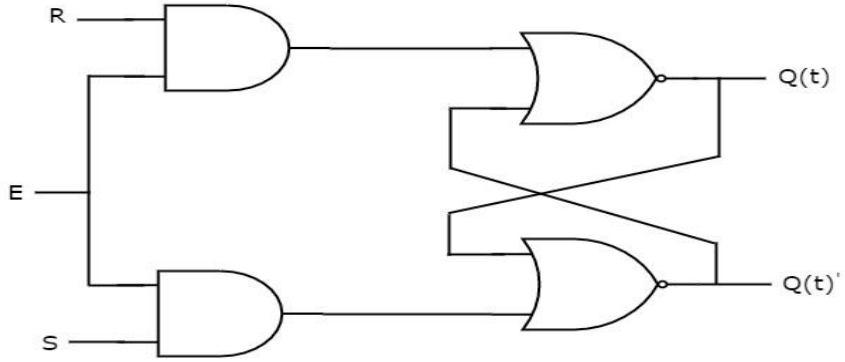
There are two types of memory elements based on the type of triggering that is suitable to operate it.

- Latches
- Flip-flops

Latches operate with enable signal, which is **level sensitive**. Whereas, flip-flops are edge sensitive.

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, Qt values.

At any time, only of those two inputs should be '1'. If both inputs are '1', then the next state Qt+1 value is undefined.

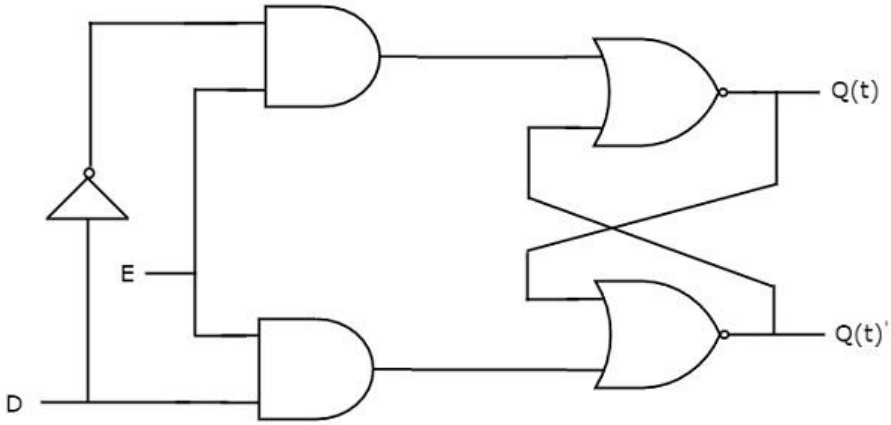
The following table shows the **state table** of SR latch.

S	**R**	**Q**t+1
0	0	Qt
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.



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

We can implement flip-flops in two methods.

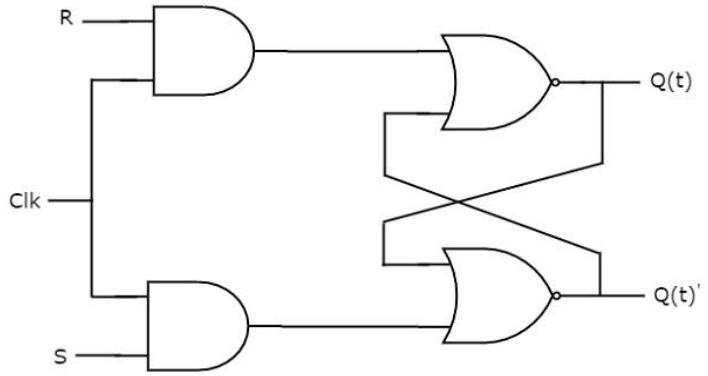
In first method, **cascade two latches** in such a way that the first latch is enabled for every positive clock pulse and second latch is enabled for every negative clock pulse. So that the combination of these two latches become a flip-flop.

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.

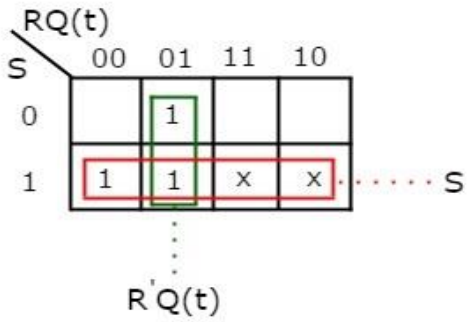
S	**R**	**Q**t+1
0	0	Qt
0	1	0
1	0	1
1	1	-

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

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.

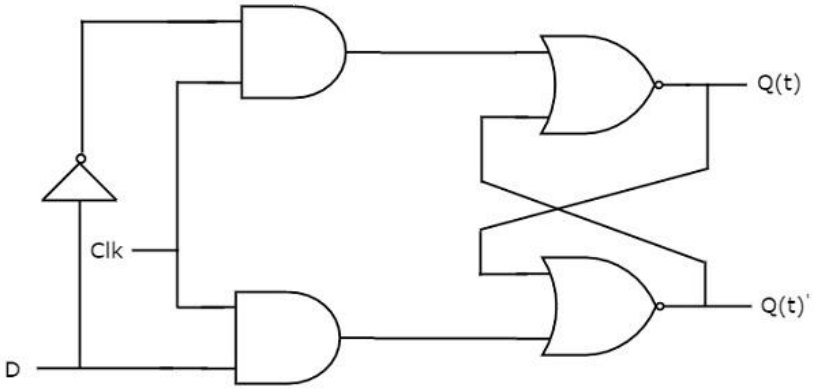


The maximum possible groupings of adjacent ones are already shown in the figure. Therefore, the **simplified expression** for next state Qt+1 is

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

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 & Qt'. 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 + 1
0	0
1	1

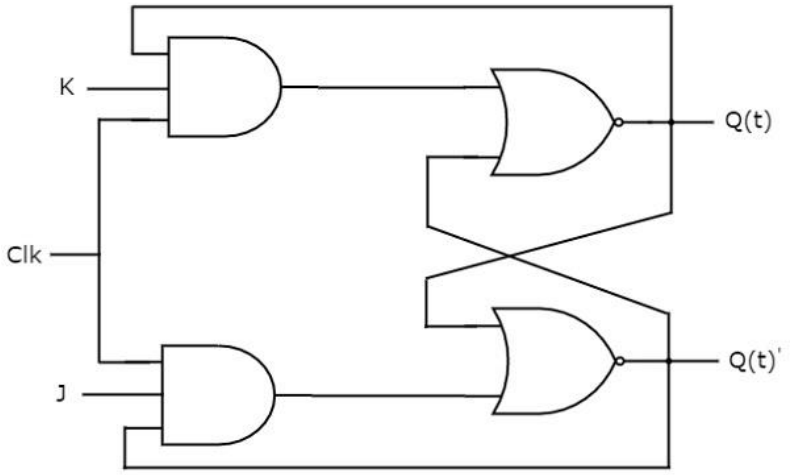
Therefore, D flip-flop always Hold the information, which is available on data input, D of earlier positive transition of clock signal. From the above state table, we can directly write the next state equation as

$$Qt+1 = D$$

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 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.

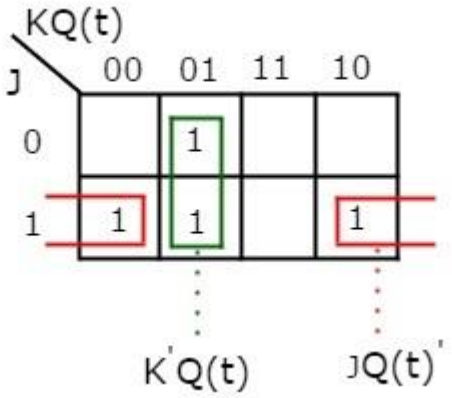
J	**K**	**Q**t+1
0	0	Qt
0	1	0
1	0	1
1	1	Qt'

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	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	1
1	1	1	0

By using three variable K-Map, we can get the simplified expression for next state, Qt+1. **Three variable K-Map** for next state, Qt+1 is shown in the following figure.

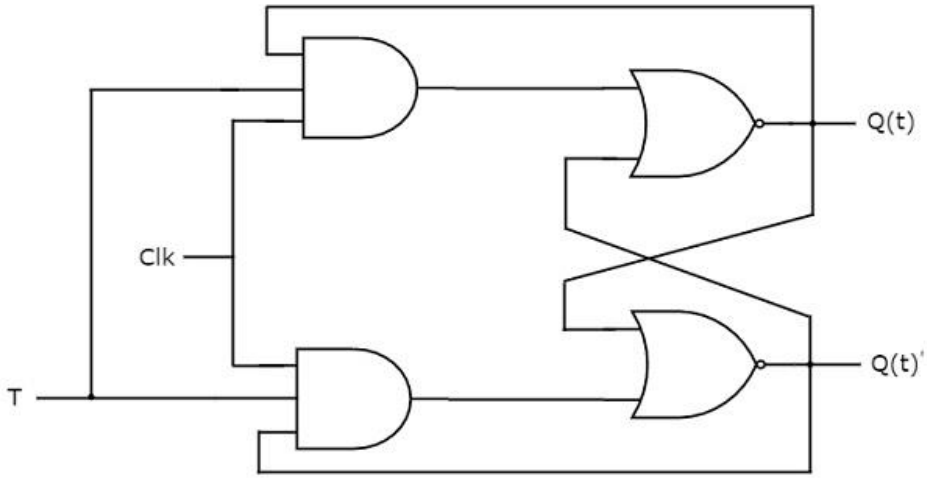


The maximum possible groupings of adjacent ones are already shown in the figure. Therefore, the **simplified expression** for next state Qt+1t+1 is

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

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 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.

D	**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

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

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

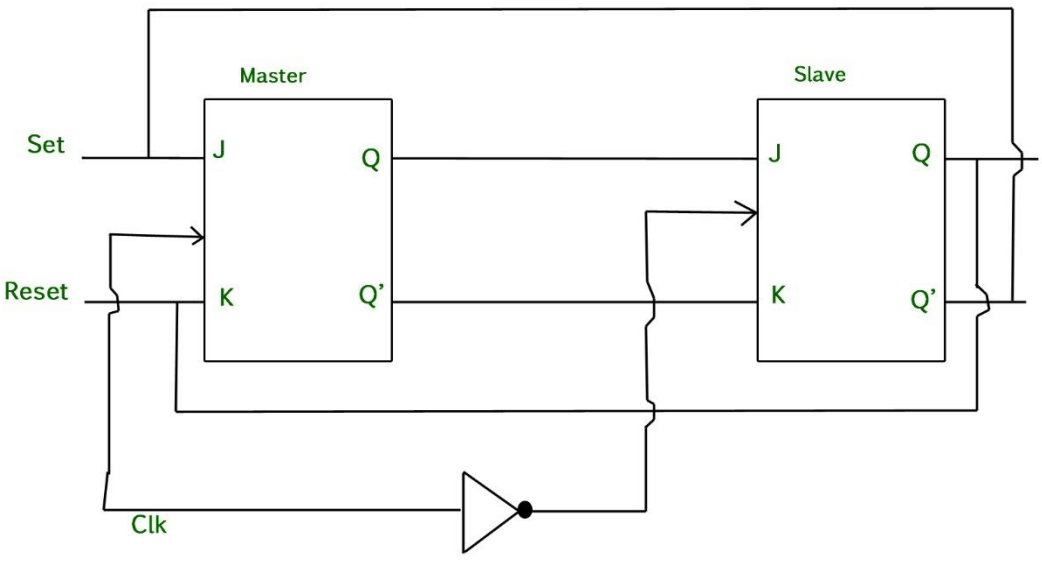
The output of T flip-flop always toggles for every positive transition of the clock signal, when input T remains at logic High 11. Hence, T flip-flop can be used in **counters**.

Race Around Condition in JK Flip-flop –

For J-K flip-flop, if J=K=1, and if clk=1 for a long period of time, then Q output will toggle as long as CLK is high, which makes the output of the flip-flop unstable or uncertain. This problem is called race around condition in J-K flip-flop. This problem (Race Around Condition) can be avoided by ensuring that the clock input is at logic "1" only for a very short time. This introduced the concept of **Master Slave JK** flip flop.

Master Slave JK flip flop –

The Master-Slave Flip-Flop is basically a combination of two JK flip-flops connected together in a series configuration. Out of these, one acts as the **"master"** and the other as a **"slave"**. The output from the master flip flop is connected to the two inputs of the slave flip flop whose output is fed back to inputs of the master flip flop.
In addition to these two flip-flops, the circuit also includes an **inverter**. The inverter is connected to clock pulse in such a way that the inverted clock pulse is given to the slave flip-flop. In other words if CP=0 for a master flip-flop, then CP=1 for a slave flip-flop and if CP=1 for master flip flop then it becomes 0 for slave flip flop.



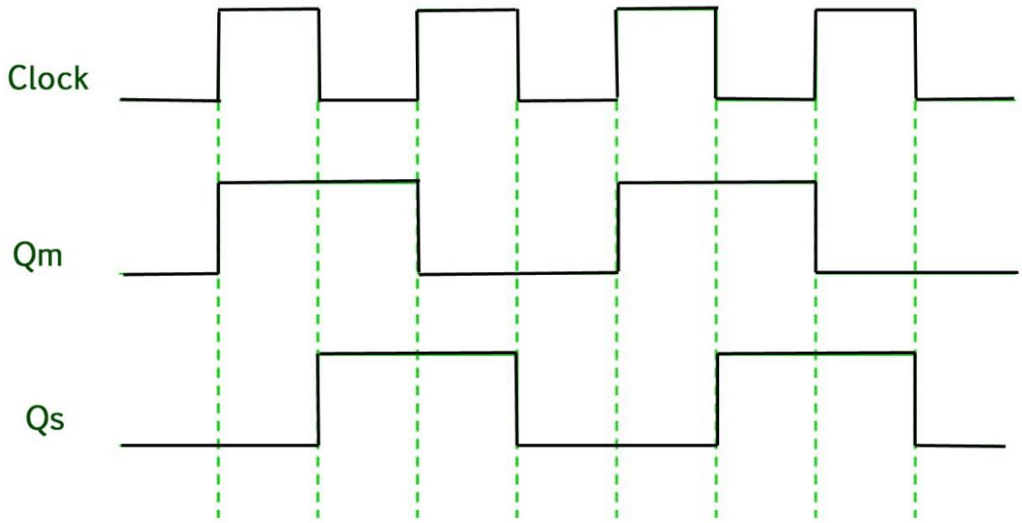
Master-slave JK Flip-Flop

Working of a master slave flip flop –

1. When the clock pulse goes to 1, the slave is isolated; J and K inputs may affect the state of the system. The slave flip-flop is isolated until the CP goes to 0. When the CP goes back to 0, information is passed from the master flip-flop to the slave and output is obtained.

2. Firstly the master flip flop is positive level triggered and the slave flip flop is negative level triggered, so the master responds before the slave.
3. If J=0 and K=1, the high Q' output of the master goes to the K input of the slave and the clock forces the slave to reset, thus the slave copies the master.
4. If J=1 and K=0, the high Q output of the master goes to the J input of the slave and the Negative transition of the clock sets the slave, copying the master.
5. If J=1 and K=1, it toggles on the positive transition of the clock and thus the slave toggles on the negative transition of the clock.
6. If J=0 and K=0, the flip flop is disabled and Q remains unchanged.

Timing Diagram of a Master flip flop –



1. When the Clock pulse is high the output of master is high and remains high till the clock is low because the state is stored.
2. Now the output of master becomes low when the clock pulse becomes high again and remains low until the clock becomes high again.
3. Thus, toggling takes place for a clock cycle.
4. When the clock pulse is high, the master is operational but not the slave thus the output of the slave remains low till the clock remains high.
5. When the clock is low, the slave becomes operational and remains high until the clock again becomes low.
6. Toggling takes place during the whole process since the output is changing once in a cycle.

This makes the Master-Slave J-K flip flop a Synchronous device as it only passes data with the timing of the clock signal.

Unit-4

Registers, Memories & PLD

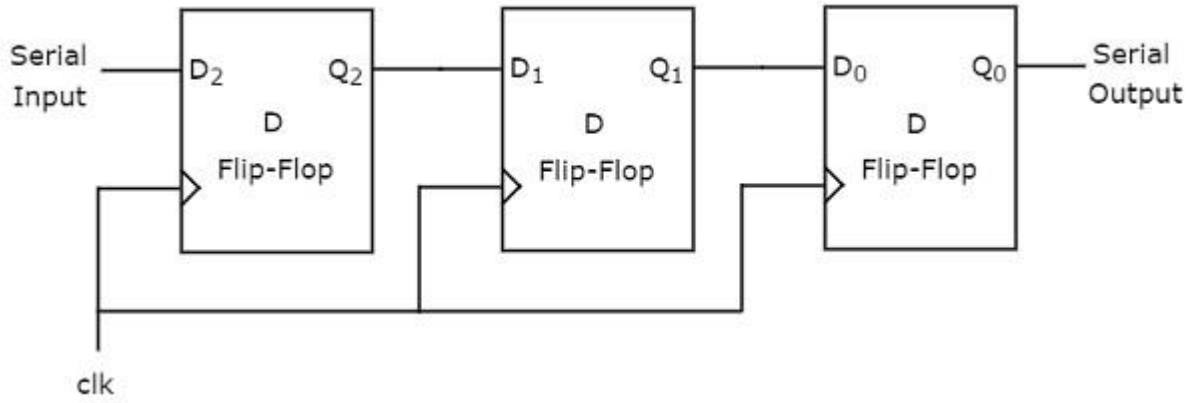
In order to store multiple bits of information, we require multiple flip-flops. The group of flip-flops, which are used to hold store the binary data is known as **register**.

If the register is capable of shifting bits either towards right hand side or towards left hand side is known as **shift register**. An 'N' bit shift register contains 'N' flip-flops. Following are the four types of shift registers based on applying inputs and accessing of outputs.

- Serial In − Serial Out shift register
- Serial In − Parallel Out shift register
- Parallel In − Serial Out shift register
- Parallel In − Parallel Out shift register

Serial In Serial Out (SISO) Shift Register

The shift register, which allows serial input and produces serial output is known as Serial In – Serial Out SISO shift register. The **block diagram** of 3-bit SISO shift register is shown in the following figure.



This block diagram consists of three D flip-flops, which are **cascaded**. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can send the bits serially from the input of left most D flip-flop. Hence, this input is also called as **serial input**. For every positive edge triggering of clock signal, the data shifts from one stage to the next. So, we can receive the bits serially from the output of right most D flip-flop. Hence, this output is also called as **serial output**.

Example

Let us see the working of 3-bit SISO shift register by sending the binary information **"011"** from LSB to MSB serially at the input.

Assume, initial status of the D flip-flops from leftmost to rightmost is Q2Q1Q0=000. We can understand the **working of 3-bit SISO shift register** from the following table.

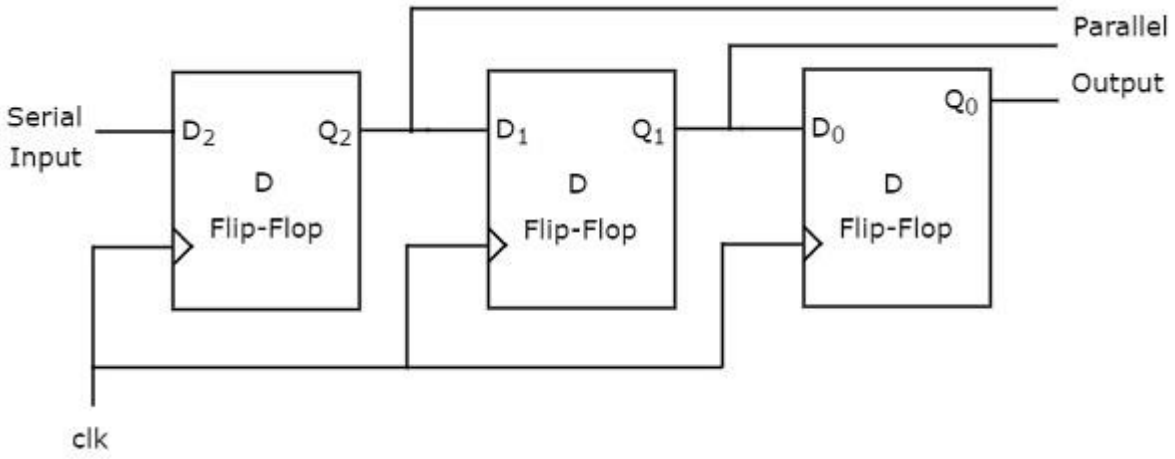
No of positive edge of Clock	Serial Input	Q_2	Q_1	Q_0
0	-	0	0	0
1	1LSBLSB	1	0	0
2	1	1	1	0
3	0MSBMSB	0	1	1LSBLSB
4	-	-	0	1
5	-	-	-	0MSBMSB

The initial status of the D flip-flops in the absence of clock signal is Q2Q1Q0=000. Here, the serial output is coming from Q0. So, the LSB 11 is received at 3rd positive edge of clock and the MSB 00 is received at 5th positive edge of clock.

Therefore, the 3-bit SISO shift register requires five clock pulses in order to produce the valid output. Similarly, the **N-bit SISO shift register** requires **2N-1** clock pulses in order to shift 'N' bit information.

Serial In - Parallel Out (SIPO) Shift Register

The shift register, which allows serial input and produces parallel output is known as Serial In – Parallel Out SIPOSIPO shift register. The **block diagram** of 3-bit SIPO shift register is shown in the following figure.



This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can send the bits serially from the input of left most D flip-flop. Hence, this input is also called as **serial input**. For every positive edge triggering of clock signal, the data shifts from one stage to the next. In this case, we can access the outputs of each D flip-flop in parallel. So, we will get **parallel outputs** from this shift register.

Example

Let us see the working of 3-bit SIPO shift register by sending the binary information **"011"** from LSB to MSB serially at the input.

Assume, initial status of the D flip-flops from leftmost to rightmost is Q2Q1Q0=000. Here, Q2 & Q0 are MSB & LSB respectively. We can understand the **working of 3-bit SIPO shift register** from the following table.

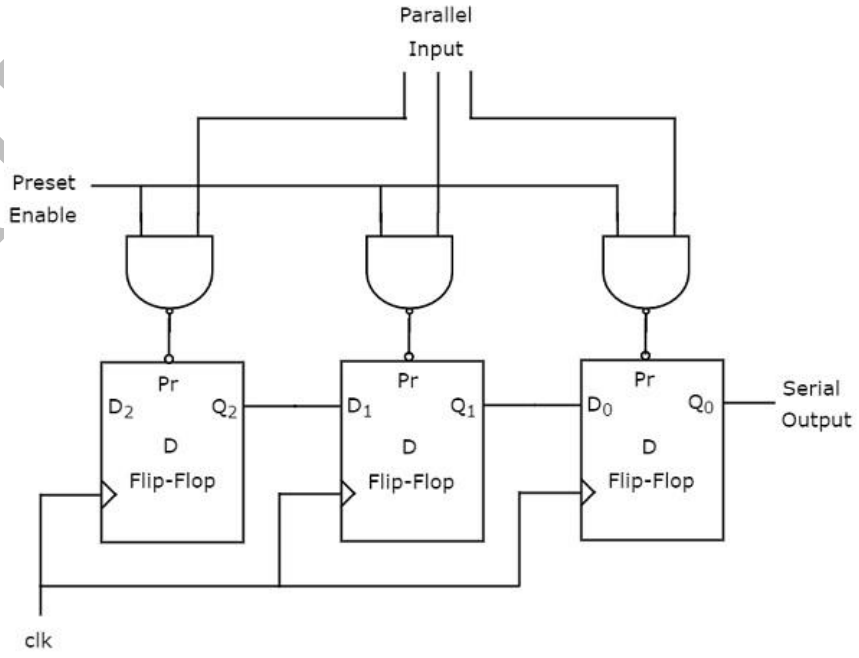
No of positive edge of Clock	Serial Input	Q_2MSBMSB	Q_1	Q_0LSBLSB
0	-	0	0	0
1	1LSBLSB	1	0	0
2	1	1	1	0
3	0MSBMSB	0	1	1

The initial status of the D flip-flops in the absence of clock signal is Q2Q1Q0=000. The binary information **"011"** is obtained in parallel at the outputs of D flip-flops for third positive edge of clock.

So, the 3-bit SIPO shift register requires three clock pulses in order to produce the valid output. Similarly, the **N-bit SIPO shift register** requires **N** clock pulses in order to shift 'N' bit information.

Parallel In − Serial Out (PISO) Shift Register

The shift register, which allows parallel input and produces serial output is known as Parallel In − Serial Out PISO shift register. The **block diagram** of 3-bit PISO shift register is shown in the following figure.



This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can apply the **parallel inputs** to each D flip-flop by making Preset Enable to 1. For every positive edge triggering of clock signal, the data shifts from one stage to the next. So, we will get the **serial output** from the right most D flip-flop.

Example

Let us see the working of 3-bit PISO shift register by applying the binary information **"011"** in parallel through preset inputs.

Since the preset inputs are applied before positive edge of Clock, the initial status of the D flip-flops from leftmost to rightmost will be Q2Q1Q0=011. We can understand the **working of 3-bit PISO shift register** from the following table.

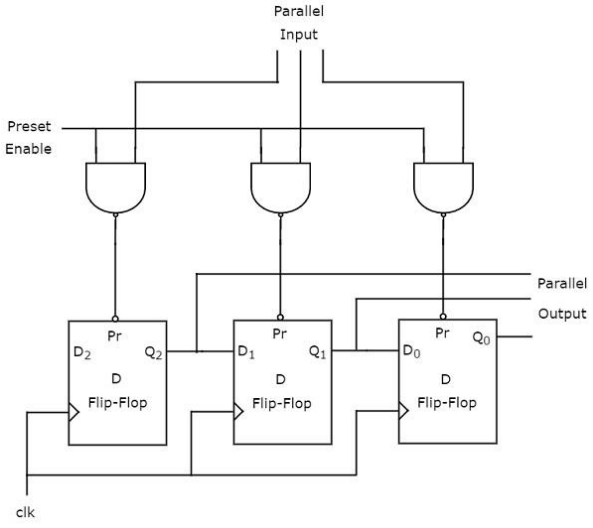
No of positive edge of Clock	Q_2	Q_1	Q_0
0	0	1	1LSB
1	-	0	1
2	-	-	0LSB

Here, the serial output is coming from Q0. So, the LSB 11 is received before applying positive edge of clock and the MSB 00 is received at 2^{nd} positive edge of clock.

Therefore, the 3-bit PISO shift register requires two clock pulses in order to produce the valid output. Similarly, the **N-bit PISO shift register** requires **N-1** clock pulses in order to shift 'N' bit information.

Parallel In - Parallel Out (PIPO) Shift Register

The shift register, which allows parallel input and produces parallel output is known as Parallel In − Parallel Out PIPOPIPO shift register. The **block diagram** of 3-bit PIPO shift register is shown in the following figure.



This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can apply the **parallel inputs** to each D flip-flop by making Preset Enable to 1. We can apply the parallel inputs through preset or clear. These two are asynchronous inputs. That means, the flip-flops produce the corresponding outputs, based on the values of asynchronous inputs. In this case, the effect of outputs is independent of clock transition. So, we will get the **parallel outputs** from each D flip-flop.

Example

Let us see the working of 3-bit PIPO shift register by applying the binary information **"011"** in parallel through preset inputs.

Since the preset inputs are applied before positive edge of Clock, the initial status of the D flip-flops from leftmost to rightmost will be Q2Q1Q0=011. So, the binary information **"011"** is obtained in parallel at the outputs of D flip-flops before applying positive edge of clock.

Therefore, the 3-bit PIPO shift register requires zero clock pulses in order to produce the valid output. Similarly, the **N-bit PIPO shift register** doesn't require any clock pulse in order to shift 'N' bit information.

Counter:

T flip-flop toggles the output either for every positive edge of clock signal or for negative edge of clock signal.

An 'N' bit binary counter consists of 'N' T flip-flops. If the counter counts from 0 to $2^N - 1$, then it is called as binary **up counter**. Similarly, if the counter counts down from $2^N - 1$ to 0, then it is called as binary **down counter**.

There are two **types of counters** based on the flip-flops that are connected in synchronous or not.

- Asynchronous counters
- Synchronous counters

Asynchronous Counters:

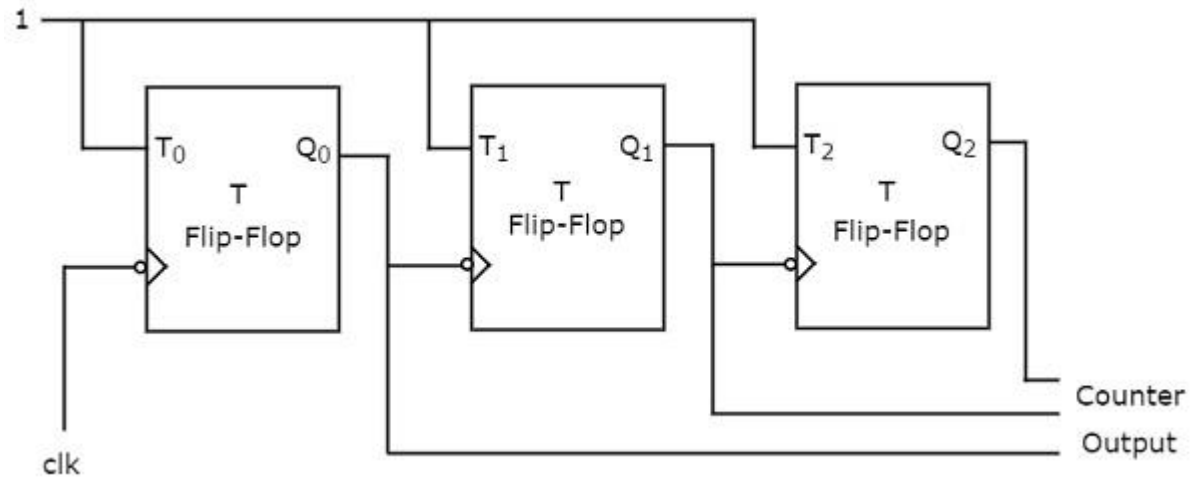
If the flip-flops do not receive the same clock signal, then that counter is called as **Asynchronous counter**. The output of system clock is applied as clock signal only to first flip-flop. The remaining flip-flops receive the clock signal from output of its previous stage flip-flop. Hence, the outputs of all flip-flops do not change affect at the same time.

Now, let us discuss the following two counters one by one.

- Asynchronous Binary up counter
- Asynchronous Binary down counter

Asynchronous Binary Up Counter:

An 'N' bit Asynchronous binary up counter consists of 'N' T flip-flops. It counts from 0 to $2^N - 1$. The **block diagram** of 3-bit Asynchronous binary up counter is shown in the following figure.



The 3-bit Asynchronous binary up counter contains three T flip-flops and the T-input of all the flip-flops are connected to '1'. All these flip-flops are negative edge triggered but the outputs change asynchronously. The clock signal is directly applied to the first T flip-flop. So, the output of first T flip-flop **toggles** for every negative edge of clock signal.

The output of first T flip-flop is applied as clock signal for second T flip-flop. So, the output of second T flip-flop toggles for every negative edge of output of first T flip-flop. Similarly, the output of third T flip-flop toggles for every negative edge of output of second T flip-flop, since the output of second T flip-flop acts as the clock signal for third T flip-flop.

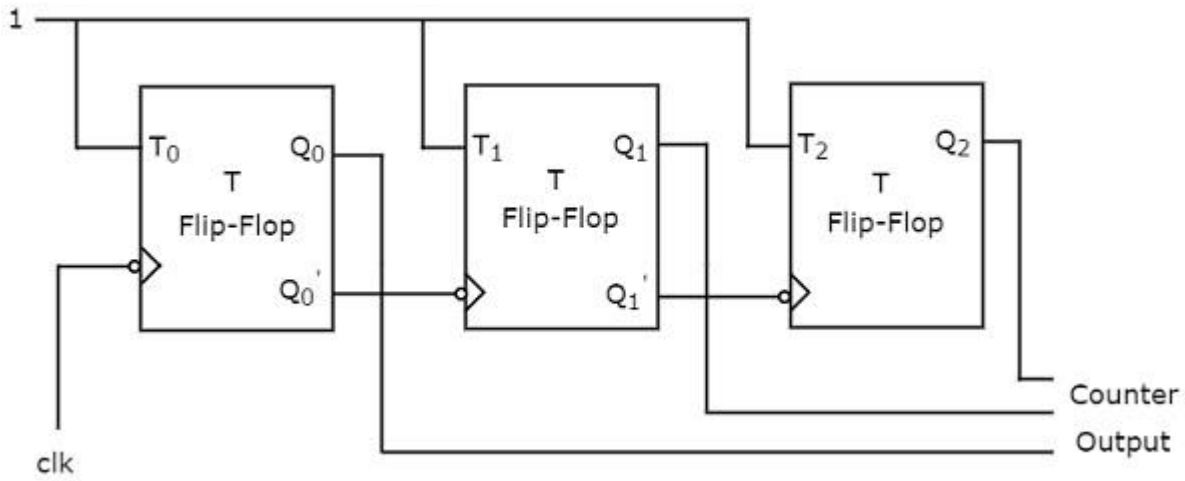
Assume the initial status of T flip-flops from rightmost to leftmost is Q2Q1Q0=000. Here, Q2 & Q0 are MSB & LSB respectively. We can understand the **working** of 3-bit asynchronous binary counter from the following table.

No of negative edge of Clock	**Q_0**LSB	**Q_1**	**Q_2**MSB
0	0	0	0
1	1	0	0
2	0	1	0
3	1	1	0
4	0	0	1
5	1	0	1
6	0	1	1
7	1	1	1

Here Q0 toggled for every negative edge of clock signal. Q1 toggled for every Q0 that goes from 1 to 0, otherwise remained in the previous state. Similarly, Q2 toggled for every Q1 that goes from 1 to 0, otherwise remained in the previous state.
The initial status of the T flip-flops in the absence of clock signal is Q2Q1Q0=000. This is incremented by one for every negative edge of clock signal and reached to maximum value at 7th negative edge of clock signal. This pattern repeats when further negative edges of clock signal are applied.

Asynchronous Binary Down Counter:

An 'N' bit Asynchronous binary down counter consists of 'N' T flip-flops. It counts from $2^N - 1$ to 0. The **block diagram** of 3-bit Asynchronous binary down counter is shown in the following figure.



The block diagram of 3-bit Asynchronous binary down counter is similar to the block diagram of 3-bit Asynchronous binary up counter. But, the only difference is that instead of connecting the normal outputs of one stage flip-flop as clock signal for next stage flip-flop, connect the **complemented outputs** of one stage flip-flop as clock signal for next stage flip-flop. Complemented output goes from 1 to 0 is same as the normal output goes from 0 to 1.

Assume the initial status of T flip-flops from rightmost to leftmost is Q2Q1Q0=000. Here, Q2 & Q0 are MSB & LSB respectively. We can understand the **working** of 3-bit asynchronous binary down counter from the following table.

No of negative edge of Clock	Q_0LSBLSB	Q_1	Q_2MSBMSB
0	0	0	0
1	1	1	1
2	0	1	1
3	1	0	1

4	0	0	1
5	1	1	0
6	0	1	0
7	1	0	0

Here Q0 toggled for every negative edge of clock signal. Q1 toggled for every Q0 that goes from 0 to 1, otherwise remained in the previous state. Similarly, Q2 toggled for every Q1 that goes from 0 to 1, otherwise remained in the previous state.
The initial status of the T flip-flops in the absence of clock signal is Q2Q1Q0=000. This is decremented by one for every negative edge of clock signal and reaches to the same value at 8th negative edge of clock signal. This pattern repeats when further negative edges of clock signal are applied.

Synchronous Counters:

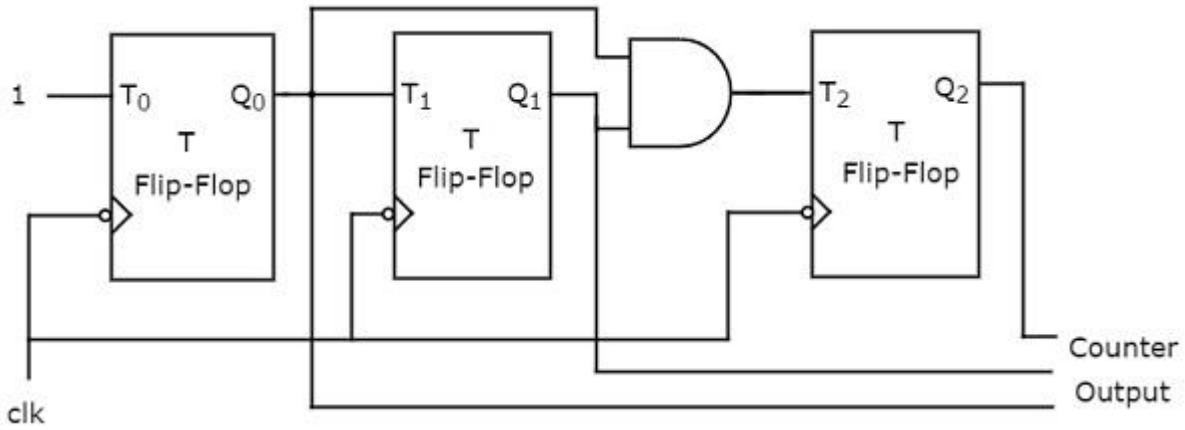
If all the flip-flops receive the same clock signal, then that counter is called as **Synchronous counter**. Hence, the outputs of all flip-flops change affect at the same time.

Now, let us discuss the following two counters one by one.

- Synchronous Binary up counter
- Synchronous Binary down counter

Synchronous Binary Up Counter:

An 'N' bit Synchronous binary up counter consists of 'N' T flip-flops. It counts from 0 to $2^N - 1$. The **block diagram** of 3-bit Synchronous binary up counter is shown in the following figure.

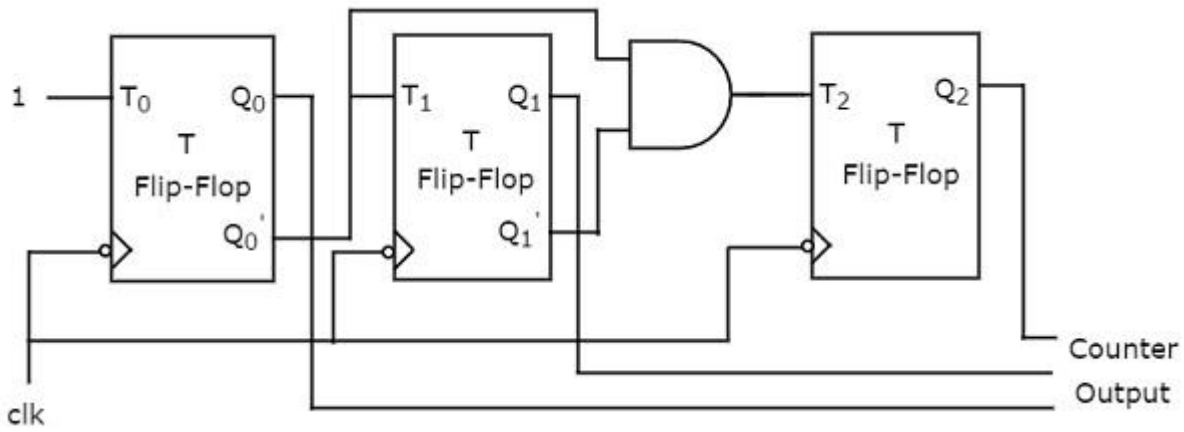


The 3-bit Synchronous binary up counter contains three T flip-flops & one 2-input AND gate. All these flip-flops are negative edge triggered and the outputs of flip-flops change affect synchronously. The T inputs of first, second and third flip-flops are 1, Q0 & Q1Q0 respectively.

The output of first T flip-flop **toggles** for every negative edge of clock signal. The output of second T flip-flop toggles for every negative edge of clock signal if Q0 is 1. The output of third T flip-flop toggles for every negative edge of clock signal if both Q0 & Q1 are 1.

Synchronous Binary Down Counter:

An 'N' bit Synchronous binary down counter consists of 'N' T flip-flops. It counts from $2^N - 1$ to 0. The **block diagram** of 3-bit Synchronous binary down counter is shown in the following figure.



The 3-bit Synchronous binary down counter contains three T flip-flops & one 2-input AND gate. All these flip-flops are negative edge triggered and the outputs of flip-flops change affect synchronously. The T inputs of first, second and third flip-flops are 1, Q0′ &' Q1′Q0′ respectively.

The output of first T flip-flop **toggles** for every negative edge of clock signal. The output of second T flip-flop toggles for every negative edge of clock signal if Q0′Q0′ is 1. The output of third T flip-flop toggles for every negative edge of clock signal if both Q1′ & Q0′ are 1.

Following are the applications of shift registers

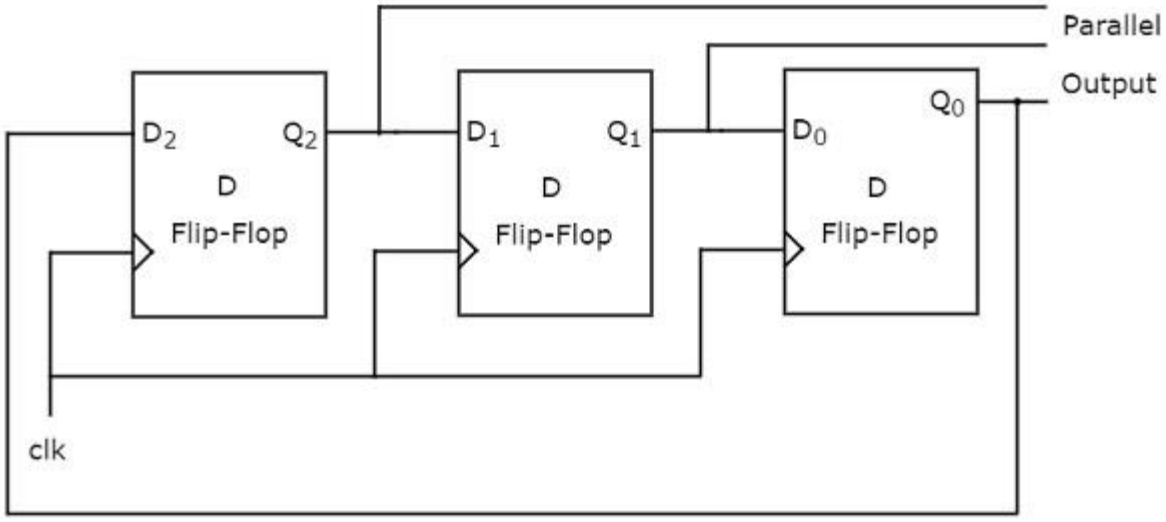
- Shift register is used as **Parallel to serial converter**, which converts the parallel data into serial data. It is utilized at the transmitter section after Analog to Digital Converter ADC block.
- Shift register is used as **Serial to parallel converter**, which converts the serial data into parallel data. It is utilized at the receiver section before Digital to Analog Converter DAC block.
- Shift register along with some additional gates s generate the sequence of zeros and ones. Hence, it is used as **sequence generator**.
- Shift registers are also used as **counters**. There are two types of counters based on the type of output from right most D flip-flop is connected to the serial input. Those are Ring counter and Johnson Ring counter.

Ring Counter:

The operation of Serial In - Parallel Out SIPO shift register. It accepts the data from outside in serial form and it requires 'N' clock pulses in order to shift 'N' bit data.

Similarly, **'N' bit Ring counter** performs the similar operation. But, the only difference is that the output of rightmost D flip-flop is given as input of leftmost D flip-flop instead of applying data from outside. Therefore, Ring counter produces a sequence of states pattern of zeros and ones pattern and it repeats for every **'N' clock cycles**.

The **block diagram** of 3-bit Ring counter is shown in the following figure.



The 3-bit Ring counter contains only a 3-bit SIPO shift register. The output of rightmost D flip-flop is connected to serial input of left most D flip-flop.

Assume, initial status of the D flip-flops from leftmost to rightmost is Q2Q1Q0=001. Here, Q2 & Q0 are MSB & LSB respectively. We can understand the **working of Ring counter** from the following table.

No of positive edge of Clock	Serial Input = Q_0	Q_2MSBMSB	Q_1	Q_0LSBLSB
0	-	0	0	1
1	1	1	0	0
2	0	0	1	0
3	0	0	0	1

The initial status of the D flip-flops in the absence of clock signal is Q2Q1Q0=001. This status repeats for every three positive edge transitions of clock signal.

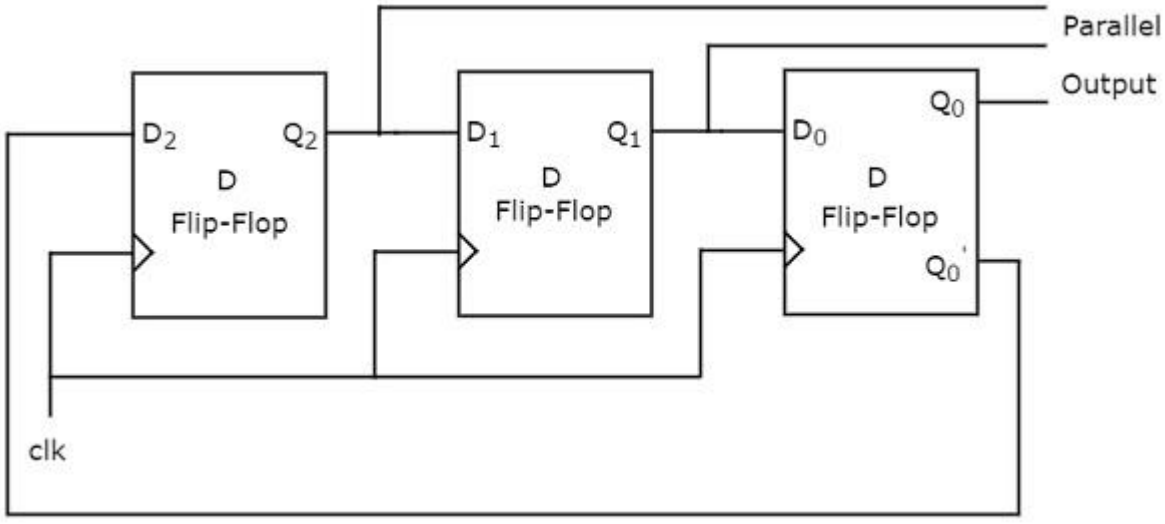
Therefore, the following **operations** take place for every positive edge of clock signal.

- Serial input of first D flip-flop gets the previous output of third flip-flop. So, the present output of first D flip-flop is equal to the previous output of third flip-flop.
- The previous outputs of first and second D flip-flops are right shifted by one bit. That means, the present outputs of second and third D flip-flops are equal to the previous outputs of first and second D flip-flops.

Johnson Ring Counter:

The operation of **Johnson Ring counter** is similar to that of Ring counter. But, the only difference is that the complemented output of rightmost D flip-flop is given as input of leftmost D flip-flop instead of normal output. Therefore, 'N' bit Johnson Ring counter produces a sequence of states pattern of zeros and ones and it repeats for every **'2N' clock cycles**.

Johnson Ring counter is also called as **Twisted Ring counter** and switch tail Ring counter. The **block diagram** of 3-bit Johnson Ring counter is shown in the following figure.



The 3-bit Johnson Ring counter also contains only a 3-bit SIPO shift register. The complemented output of rightmost D flip-flop is connected to serial input of left most D flip-flop.

Assume, initially all the D flip-flops are cleared. So, Q2Q1Q0=000. Here, Q2 & Q0 are MSB & LSB respectively. We can understand the **working** of Johnson Ring counter from the following table.

No of positive edge of Clock	Serial Input = Q_0	Q_2MSBMSB	Q_1	Q_0LSBLSB
0	-	0	0	0
1	1	1	0	0
2	1	1	1	0
3	1	1	1	1
4	0	0	1	1
5	0	0	0	1
6	0	0	0	0

The initial status of the D flip-flops in the absence of clock signal is Q2Q1Q0=000. This status repeats for every six positive edge transitions of clock signal.

Therefore, the following **operations** take place for every positive edge of clock signal.

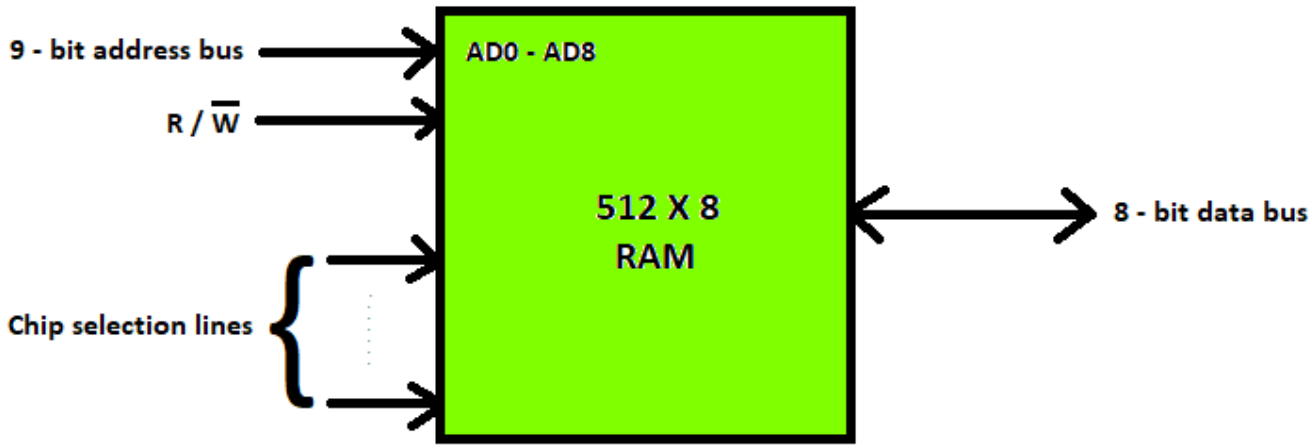
- Serial input of first D flip-flop gets the previous complemented output of third flip-flop. So, the present output of first D flip-flop is equal to the previous complemented output of third flip-flop.
- The previous outputs of first and second D flip-flops are right shifted by one bit. That means, the present outputs of second and third D flip-flops are equal to the previous outputs of first and second D flip-flops.

Concept of memories-RAM, ROM, static RAM, dynamic RAM:

RAM (Random Access Memory) is a part of computer's Main Memory which is directly accessible by CPU. RAM is used to Read and Write data into it which is accessed by CPU randomly. RAM is volatile in nature, it means if the power goes off, the stored information is lost. RAM is used to store the data that is currently processed by the CPU. Most of the programs and data that are modifiable are stored in RAM.
Integrated RAM chips are available in two form:

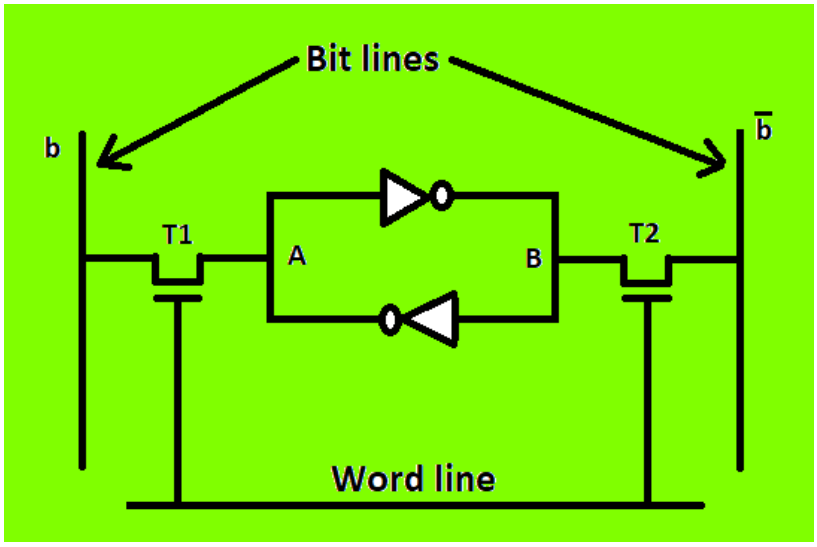
1. SRAM (Static RAM)
2. DRAM (Dynamic RAM)

The block diagram of RAM chip is given below.



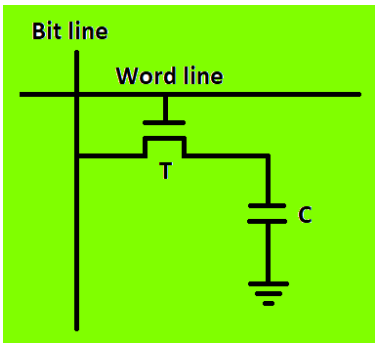
1. SRAM :
The SRAM memories consist of circuits capable of retaining the stored information as long as the power is applied. That means this type of memory requires constant power. SRAM memories are used to build Cache Memory.
SRAM Memory Cell: Static memories(SRAM) are memories that consist of circuits capable of retaining their state as long as power is on. Thus this type of memory is called volatile memory. The below figure shows a cell diagram of SRAM. A latch is formed by two inverters connected as shown in the figure. Two transistors T1 and T2 are used for

connecting the latch with two-bit lines. The purpose of these transistors is to act as switches that can be opened or closed under the control of the word line, which is controlled by the address decoder. When the word line is at 0-level, the transistors are turned off and the latch remains its information. For example, the cell is at state 1 if the logic value at point A is 1 and at point, B is 0. This state is retained as long as the word line is not activated.



For **Read operation**, the word line is activated by the address input to the address decoder. The activated word line closes both the transistors (switches) T1 and T2. Then the bit values at points A and B can transmit to their respective bit lines. The sense/write circuit at the end of the bit lines sends the output to the processor.
For **Write operation**, the address provided to the decoder activates the word line to close both the switches. Then the bit value that is to be written into the cell is provided through the sense/write circuit and the signals in bit lines are then stored in the cell.

2. DRAM :
DRAM stores the binary information in the form of electric charges applied to capacitors. The stored information on the capacitors tends to lose over a period of time and thus the capacitors must be periodically recharged to retain their usage. The main memory is generally made up of DRAM chips.
DRAM Memory Cell: Though SRAM is very fast, but it is expensive because of its every cell requires several transistors. Relatively less expensive RAM is DRAM, due to the use of one transistor and one capacitor in each cell, as shown in the below figure., where C is the capacitor and T is the transistor. Information is stored in a DRAM cell in the form of a charge on a capacitor and this charge needs to be periodically recharged. For storing information in this cell, transistor T is turned on and an appropriate voltage is applied to the bit line. This causes a known amount of charge to be stored in the capacitor. After the transistor is turned off, due to the property of the capacitor, it starts to discharge. Hence, the information stored in the cell can be read correctly only if it is read before the charge on the capacitors drops below some threshold value.

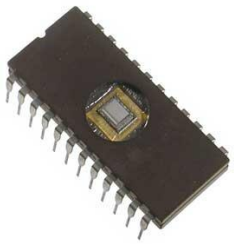


Difference between SRAM and DRAM :
Below table lists some of the differences between SRAM and DRAM:

SRAM	DRAM
1. SRAM has lower access time, so it is faster compared to DRAM.	1. DRAM has higher access time, so it is slower than SRAM.
2. SRAM is costlier than DRAM.	2. DRAM costs less compared to SRAM.
3. SRAM requires constant power supply, which means this type of memory consumes more power.	3. DRAM offers reduced power consumption, due to the fact that the information is stored in the capacitor.
4. Due to complex internal circuitry, less storage capacity is available compared to the same physical size of DRAM memory chip.	4. Due to the small internal circuitry in the one-bit memory cell of DRAM, the large storage capacity is available.
5. SRAM has low packaging density.	5. DRAM has high packaging density.

ROM:

ROM stands for **Read Only Memory**. The memory from which we can only read but cannot write on it. This type of memory is non-volatile. The information is stored permanently in such memories during manufacture. A ROM stores such instructions that are required to start a computer. This operation is referred to as **bootstrap**. ROM chips are not only used in the computer but also in other electronic items like washing machine and microwave oven.



Types of ROMs and their characteristics:

MROM (Masked ROM)

The very first ROMs were hard-wired devices that contained a pre-programmed set of data or instructions. These kind of ROMs are known as masked ROMs, which are inexpensive.

PROM (Programmable Read Only Memory)

PROM is read-only memory that can be modified only once by a user. The user buys a blank PROM and enters the desired contents using a PROM program. Inside the PROM chip, there are small fuses which are burnt open during programming. It can be programmed only once and is not erasable.

EPROM (Erasable and Programmable Read Only Memory)

EPROM can be erased by exposing it to ultra-violet light for a duration of up to 40 minutes. Usually, an EPROM eraser achieves this function. During programming, an electrical charge is trapped in an insulated gate region. The charge is retained for more than 10 years because the charge has no leakage path. For erasing this charge, ultra-violet light is passed through a quartz crystal window (lid). This exposure to ultra-violet light dissipates the charge. During normal use, the quartz lid is sealed with a sticker.

EEPROM (Electrically Erasable and Programmable Read Only Memory)

EEPROM is programmed and erased electrically. It can be erased and reprogrammed about ten thousand times. Both erasing and programming take about 4 to 10 ms (millisecond). In EEPROM, any location can be selectively erased and programmed. EEPROMs can be erased one byte at a time, rather than erasing the entire chip. Hence, the process of reprogramming is flexible but slow.

Advantages of ROM

The advantages of ROM are as follows −

- Non-volatile in nature
- Cannot be accidentally changed
- Cheaper than RAMs
- Easy to test
- More reliable than RAMs
- Static and do not require refreshing
- Contents are always known and can be verified

Basic concept of PLD & applications:

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 arrays, which has programmable feature.

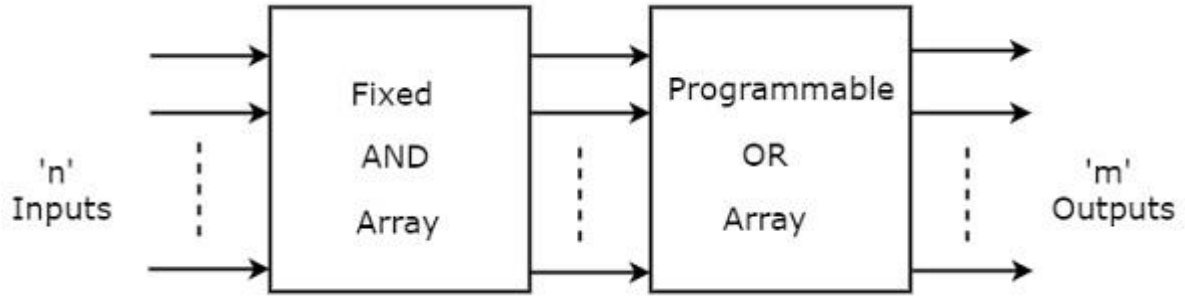
- Programmable Read Only Memory
- Programmable Array Logic
- Programmable Logic Array

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 ROMROM 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**.

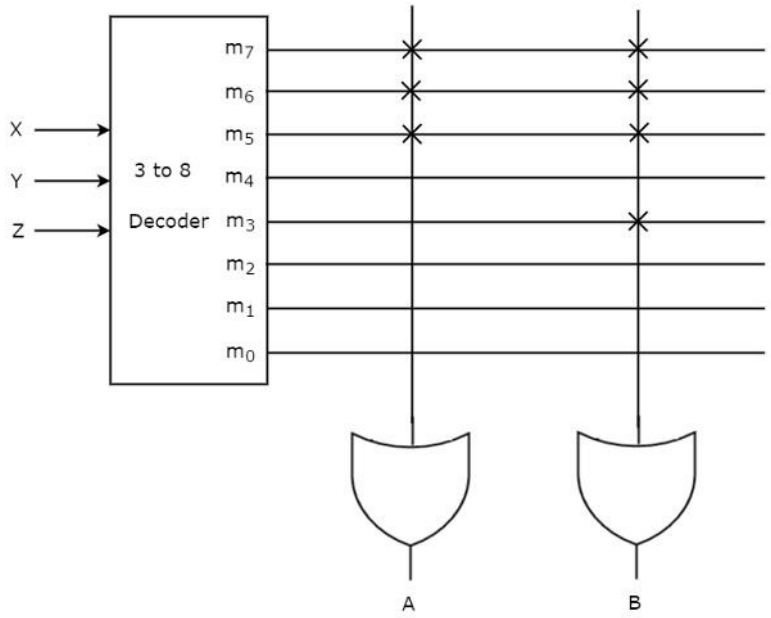
Example

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

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

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

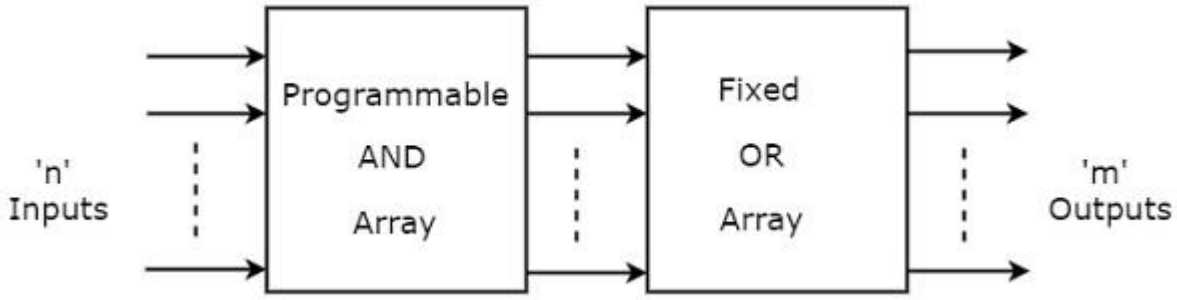
The given two functions are in sum of min terms form and each function is having three variables X, Y & Z. So, we require a 3 to 8 decoder and two programmable OR gates for producing these two functions. The corresponding **PROM** is shown in the following figure.



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**.

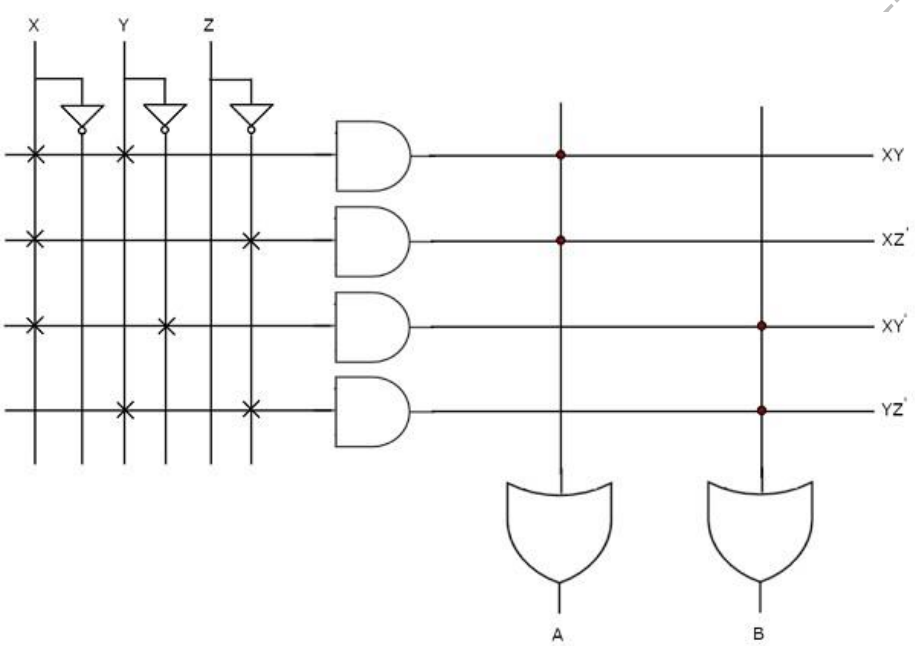
Example

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.

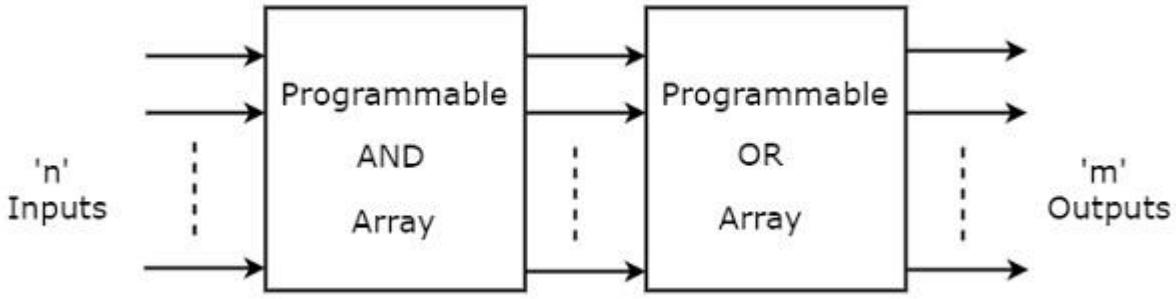


The **programmable AND gates** have the access of both normal and complemented inputs of variables. In the above figure, the inputs X, X′, Y, Y′, Z & Z′, are available at the inputs of each AND gate. So, program only the required literals in order to generate one product term by each AND gate. The symbol '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

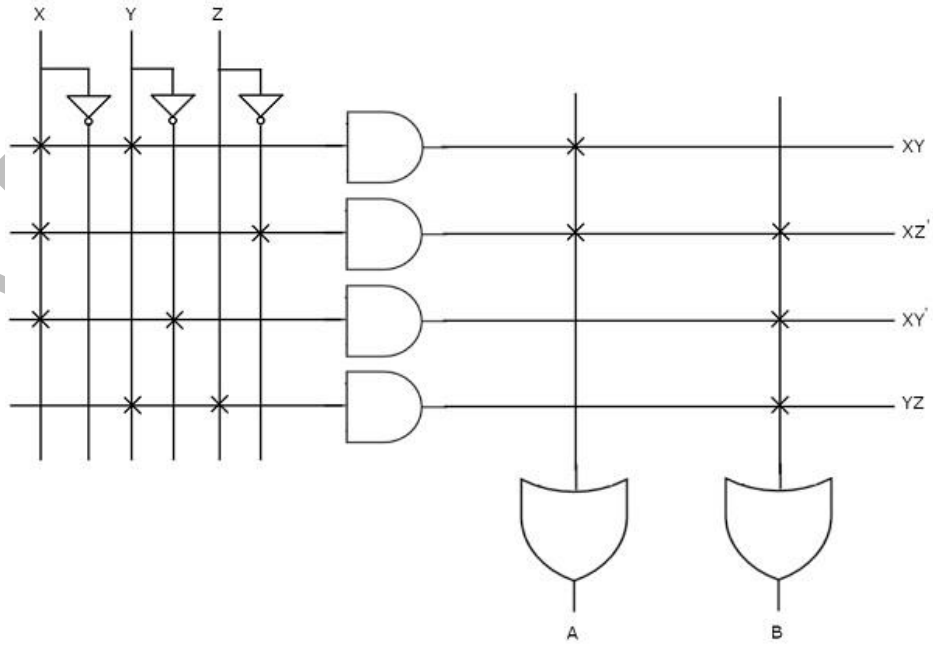
Let us implement the following **Boolean functions** using PLA.

$$A=XY+XZ'$$

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

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.

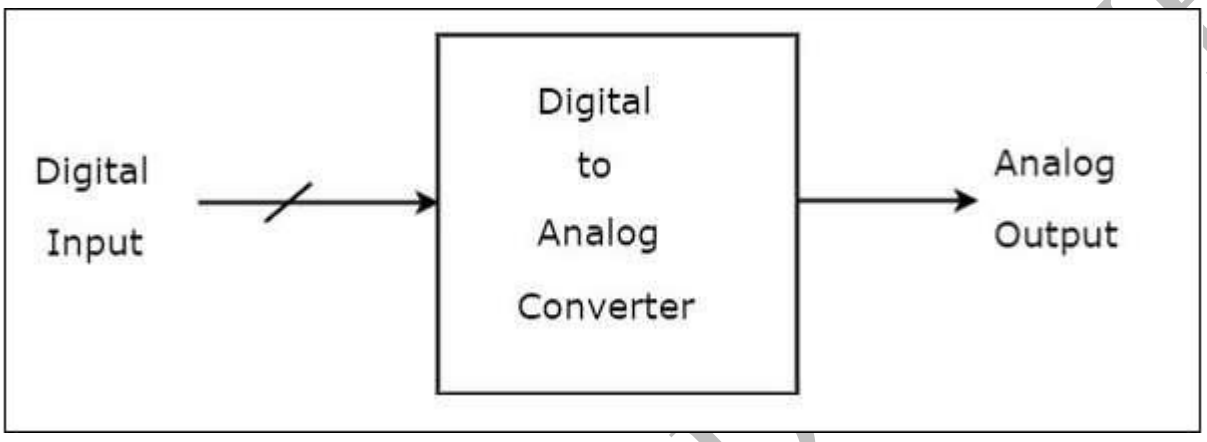
Unit-5

A/D and D/A Converters

D/A Converter

A **Digital to Analog Converter (DAC)** converts a digital input signal into an analog output signal. The digital signal is represented with a binary code, which is a combination of bits 0 and 1. This chapter deals with Digital to Analog Converters in detail.

The **block diagram** of DAC is shown in the following figure −



A Digital to Analog Converter (DAC) consists of a number of binary inputs and a single output. In general, the **number of binary inputs** of a DAC will be a power of two.

Types of DACs

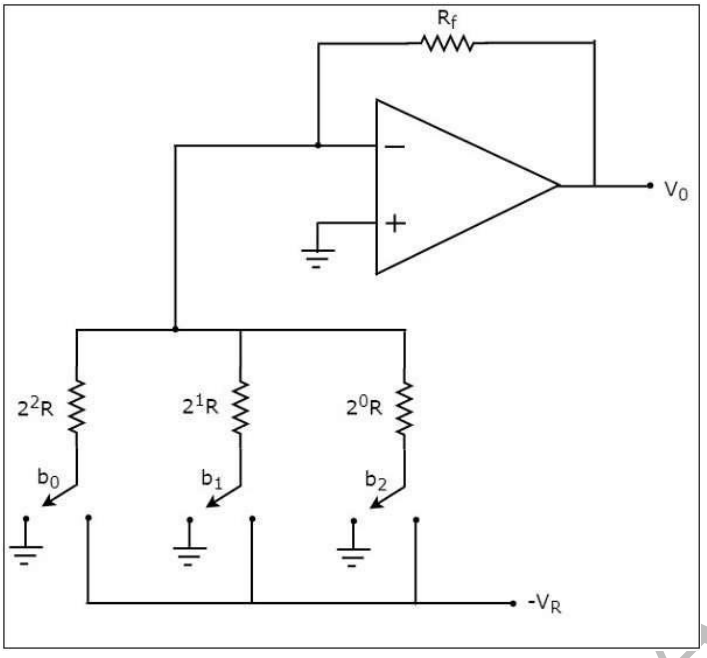
There are **two types** of DACs

- Weighted Resistor DAC
- R-2R Ladder DAC

Weighted Resistor DAC

A weighted resistor DAC produces an analog output, which is almost equal to the digital (binary) input by using **binary weighted resistors** in the inverting adder circuit. In short, a binary weighted resistor DAC is called as weighted resistor DAC.

The **circuit diagram** of a 3-bit binary weighted resistor DAC is shown in the following figure −



Recall that the bits of a binary number can have only one of the two values. i.e., either 0 or 1. Let the **3-bit binary input** is b2b1b0. Here, the bits b2 and b0 denote the **Most Significant Bit (MSB) and Least Significant Bit (LSB)** respectively.

The **digital switches** shown in the above figure will be connected to ground, when the corresponding input bits are equal to '0'. Similarly, the digital switches shown in the above figure will be connected to the negative reference voltage, −VR−VR when the corresponding input bits are equal to '1'.

In the above circuit, the non-inverting input terminal of an op-amp is connected to ground. That means zero volts is applied at the non-inverting input terminal of op-amp.

According to the **virtual short concept**, the voltage at the inverting input terminal of opamp is same as that of the voltage present at its non-inverting input terminal. So, the voltage at the inverting input terminal's node will be zero volts.

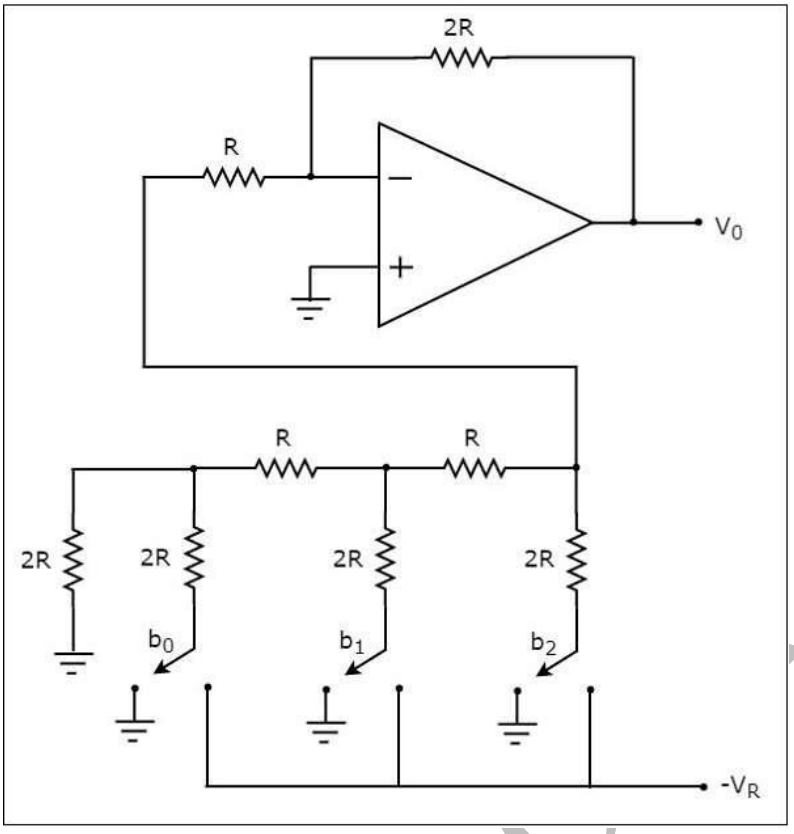
The **disadvantages** of a binary weighted resistor DAC are as follows −

- The difference between the resistance values corresponding to LSB & MSB will increase as the number of bits present in the digital input increases.
- It is difficult to design more accurate resistors as the number of bits present in the digital input increases.

R-2R Ladder DAC

The R-2R Ladder DAC overcomes the disadvantages of a binary weighted resistor DAC. As the name suggests, R-2R Ladder DAC produces an analog output, which is almost equal to the digital (binary) input by using a **R-2R ladder network** in the inverting adder circuit.

The **circuit diagram** of a 3-bit R-2R Ladder DAC is shown in the following figure −



Recall that the bits of a binary number can have only one of the two values. i.e., either 0 or 1. Let the **3-bit binary input** is b2b1b0. Here, the bits b2 and b0 denote the Most Significant Bit (MSB) and Least Significant Bit (LSB) respectively.

The digital switches shown in the above figure will be connected to ground, when the corresponding input bits are equal to '0'. Similarly, the digital switches shown in above figure will be connected to the negative reference voltage, −VR when the corresponding input bits are equal to '1'.

It is difficult to get the generalized output voltage equation of a R-2R Ladder DAC. But, we can find the analog output voltage values of R-2R Ladder DAC for individual binary input combinations easily.

The **advantages** of a R-2R Ladder DAC are as follows −

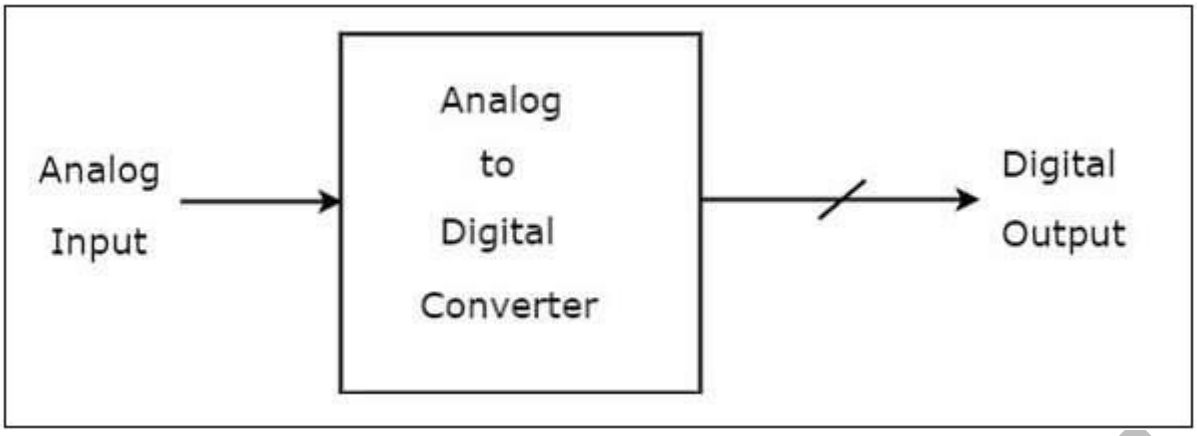
- R-2R Ladder DAC contains only two values of resistor: R and 2R. So, it is easy to select and design more accurate resistors.
- If more number of bits are present in the digital input, then we have to include required number of R-2R sections additionally.

Due to the above advantages, R-2R Ladder DAC is preferable over binary weighted resistor DAC.

A/D Converter:

An Analog to Digital Converter **(ADC)** converts an analog signal into a digital signal. The digital signal is represented with a binary code, which is a combination of bits 0 and 1.

The **block diagram** of an ADC is shown in the following figure −



Observe that in the figure shown above, an Analog to Digital Converter **(ADC)** consists of a single analog input and many binary outputs. In general, the number of binary outputs of ADC will be a power of two.

There are **two types** of ADCs: Direct type ADCs and Indirect type ADC. This chapter discusses about the Direct type ADCs in detail.

If the ADC performs the analog to digital conversion directly by utilizing the internally generated equivalent digital (binary) code for comparing with the analog input, then it is called as **Direct type ADC**.

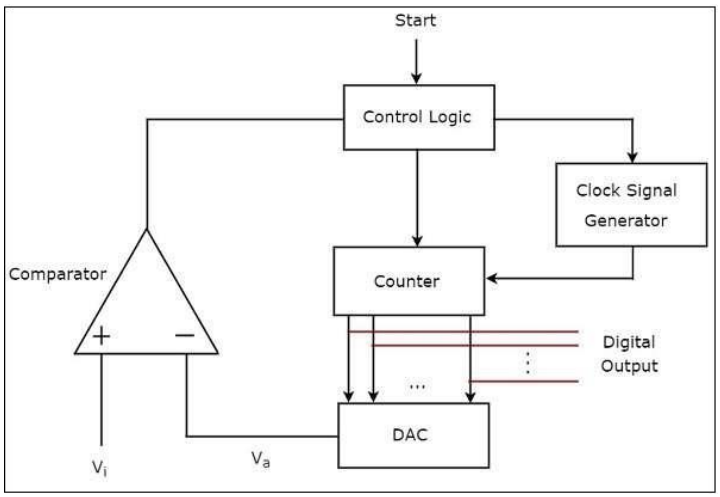
The following are the **examples** of Direct type ADCs −

- Counter type ADC
- Successive Approximation ADC

Counter type ADC

A **counter type ADC** produces a digital output, which is approximately equal to the analog input by using counter operation internally.

The **block diagram** of a counter type ADC is shown in the following figure −



The counter type ADC mainly consists of 5 blocks: Clock signal generator, Counter, DAC, Comparator and Control logic.

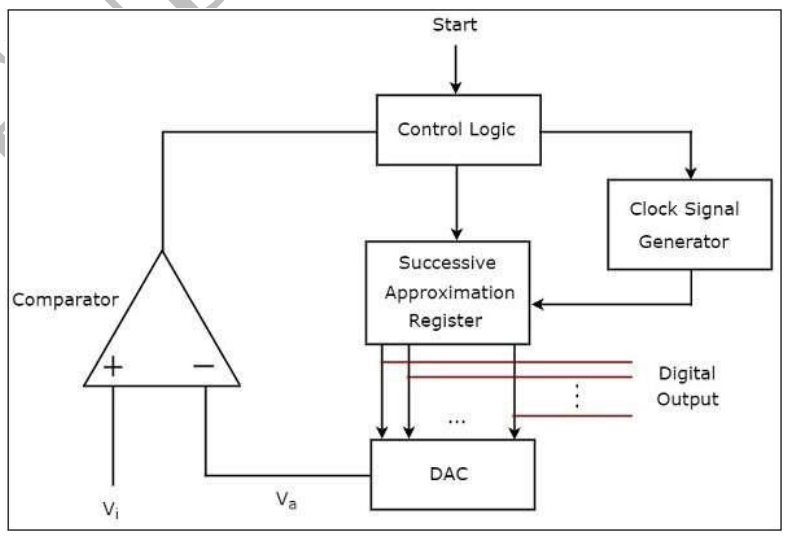
The **working** of a counter type ADC is as follows −

- The **control logic** resets the counter and enables the clock signal generator in order to send the clock pulses to the counter, when it received the start commanding signal.
- The **counter** gets incremented by one for every clock pulse and its value will be in binary (digital) format. This output of the counter is applied as an input of DAC.
- **DAC** converts the received binary (digital) input, which is the output of counter, into an analog output. Comparator compares this analog value, Va with the external analog input value Vi.
- The **output of comparator** will be **'1'** as long as Vi is greater than. The operations mentioned in above two steps will be continued as long as the control logic receives '1' from the output of comparator.
- The **output of comparator** will be **'0'** when Vi is less than or equal to Va. So, the control logic receives '0' from the output of comparator. Then, the control logic disables the clock signal generator so that it doesn't send any clock pulse to the counter.
- At this instant, the output of the counter will be displayed as the **digital output**. It is almost equivalent to the corresponding external analog input value Vi.

Successive Approximation ADC

A **successive approximation type ADC** produces a digital output, which is approximately equal to the analog input by using successive approximation technique internally.

The **block diagram** of a successive approximation ADC is shown in the following figure



The successive approximation ADC mainly consists of 5 blocks− Clock signal generator, Successive Approximation Register (SAR), DAC, comparator and Control logic.

The **working** of a successive approximation ADC is as follows −

- The **control logic** resets all the bits of SAR and enables the clock signal generator in order to send the clock pulses to SAR, when it received the start commanding signal.
- The binary (digital) data present in **SAR** will be updated for every clock pulse based on the output of comparator. The output of SAR is applied as an input of DAC.
- **DAC** converts the received digital input, which is the output of SAR, into an analog output. The comparator compares this analog value Va with the external analog input value Vi.
- The **output of a comparator** will be '1' as long as Vi is greater than Va. Similarly, the output of comparator will be '0', when Vi is less than or equal to Va.
- The operations mentioned in above steps will be continued until the digital output is a valid one.

The digital output will be a valid one, when it is almost equivalent to the corresponding external analog input value Vi.

Unit-6

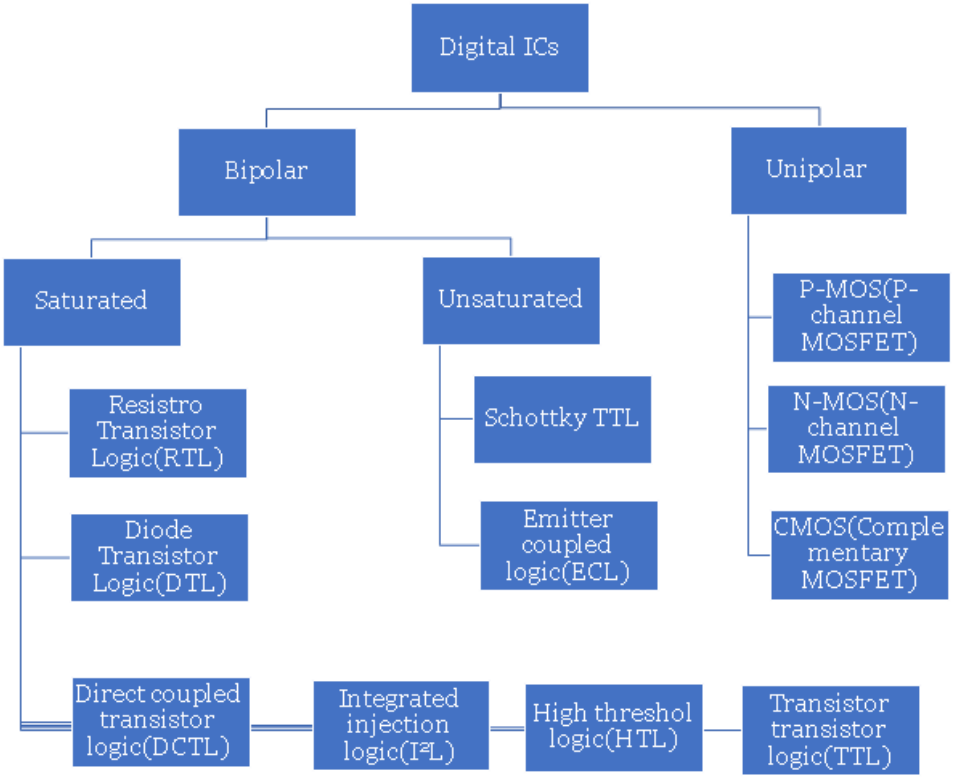
LOGIC FAMILIES

Various logic families:

Various digital ICs available in the market belong to various types. These types are known as Logic Families.Based on the components and devices internally used, digital logic families are named as

- RTL(Resistor Transistor Logic)
- TTL(Transistor Transistor Logic)
- DTL(Diode Transistor Logic)
- CMOS etc.

Classification of Logic Families:



Classification of logic families

The two basic techniques for manufacturing ICs are:

1. Bipolar Technology

2. Metal oxide semiconductor (MOS) technology.

Bipolar Families:

- The bipolar families of logic circuits construct, especially from components fabricate bipolar transistors on the chip.
- In the bipolar category, there are three basic families called Diode transistor logic(DTL), Transistor Transistor Logic(TTL), and Emitter Coupled Logic (ECL).
- DTL uses diodes and transistors, TTL uses transistors almost exclusively, TTL has become the most popular family in SSI (Small scale integration) and MSI(medium-scale integration) chips, while ECL is the fastest logic family which is used for high-speed **applications.**

MOS Families:

- The MOS family fabricates the MOS field effect transistors (MOSFETs).
- In the MOS category, there are three logic families namely PMOS(p-channel MOSFETs) family, NMOS(n-channel MOSFET) family, and CMOS(Complementary MOSFET) family.
- PMOS is the oldest and slowest type. NMOS is used for LSI(large-scale integration) field for microprocessors and memories.
- CMOS which uses a push-pull arrangement of n-channel and p-channel MOSFETs is extensively used where low power consumption is needed such as in pocket calculators.

In the "Bipolar saturated" logic families, the bipolar transistors are used as the main device. It is used at the switch and operated in the saturation and cutoff regions.

Characteristics (Parameters) of Logic Families:

Even though there are various logic families, the general characteristics, definitions, nomenclature, and terminologies used for all of them have been standardized.

So let us discuss some of the most important general characteristics first.

1. Voltage and Current Parameters:

Voltage Parameters (Threshold Levels):

Ideally, the input voltage levels of 0V and +5V (for TTL) are called logic 0 and 1 levels respectively.

However, practically we will not always obtain voltage levels matching exactly to these values. Therefore it is necessary to define the worst-case input voltages.

$V_{IL(max)}$ – worst case low level input voltage:

- This is the maximum value of input voltage which will be considered as a logic 0 level. If the input voltage is higher than $V_{IL(max)}$, then it will not be treated as a low (0) input level.

$V_{IH(min)}$ – worst-case high-level input voltage:

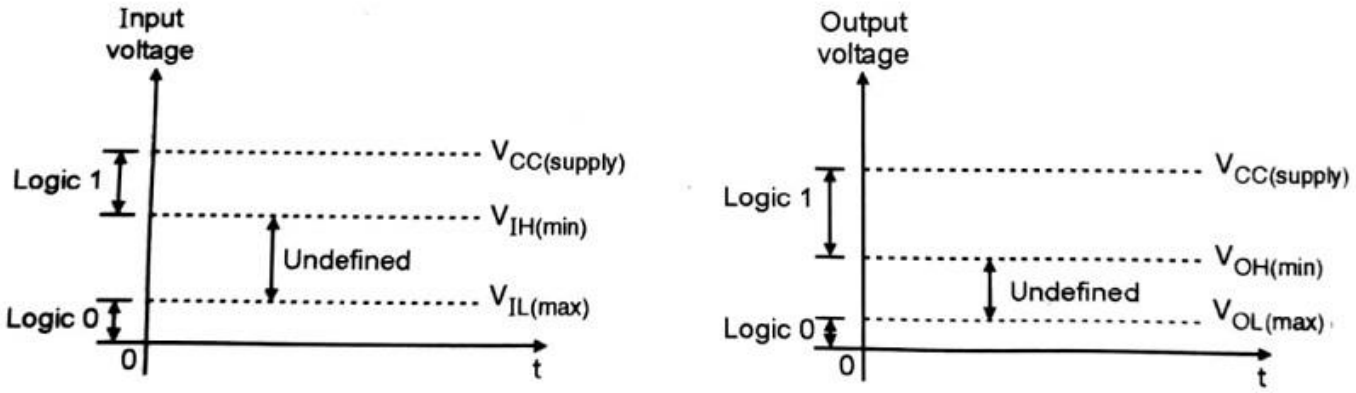
- This is the minimum value of input voltage which will be considered as a logic 1 level. If the input voltage is lower than $V_{IH(min)}$, then it will not be treated as a high (1) input level.

$V_{OH(min)}$ – worst-case high-level output voltage:

- This is the minimum value of output voltage which will be considered as a logic 1 level. If the output voltage is lower than $V_{OH(min)}$, then it will not be treated as a high (1) output level.

$V_{OL(max)}$ – worst-case low-level output voltage:

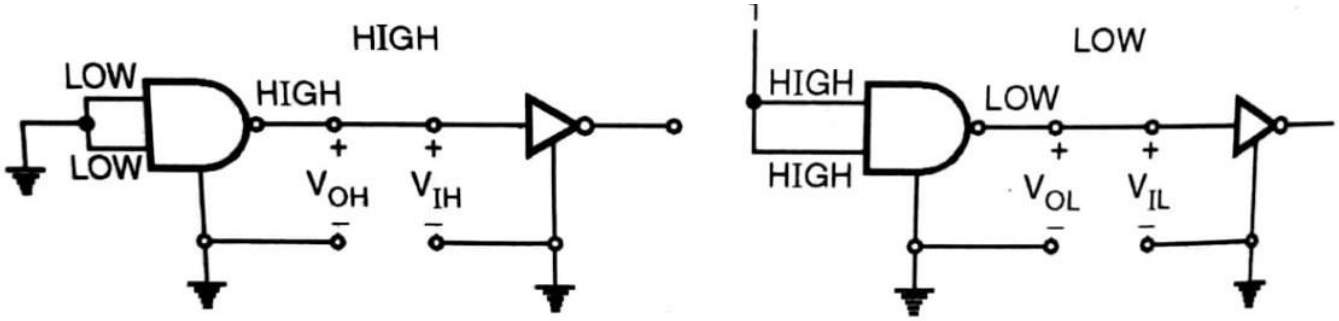
- This is the maximum value of output voltage which will be considered as a logic 0 level. If the input voltage is higher than $V_{OL(max)}$, then it will not be treated as a low (0) output level.



(a) Input voltage parameters **(b) Output voltage parameters**

Voltage parameters

The voltage parameter can be shown on the digital circuit consisting of gates as shown in the below figure. Note that the NAND and NOT gates are shown can be of TTL, ECL, CMOS or any other type.



Voltage parameters on a logic circuit

Current Parameters :

I_{IL} – Low-level input Current:

- It is the current that flows into the input when a low-level input voltage in the specified range is applied.

I_{IH} – High-level input Current:

- It is the current that flows into the input when a high-level input voltage in the specified range is applied.

I_{OL} – Low-level output Current:

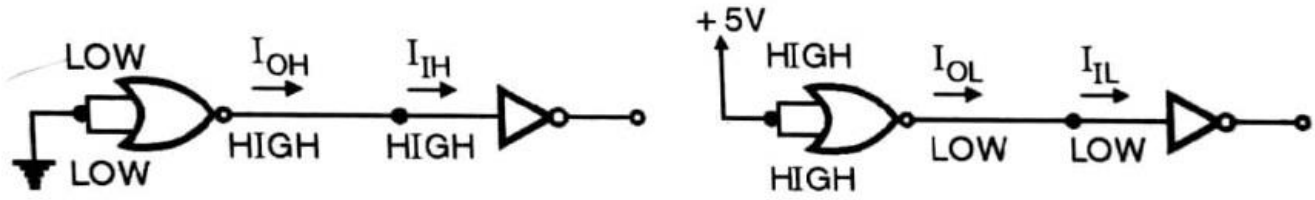
- It is the current that flows from the output when the output voltage is in the specified low voltage range and the specified load is applied.

I_{OH} – High-level Output Current:

- It is the current that flows from the output when the output voltage is in the specified high voltage range and the specified load is applied.

If the output current flows into the output terminal then it is called a **sinking current** and if the output current flows away from the output terminal then it is called a **sourcing current.**

The current parameters are displayed on the logic circuit shown in the figure below:



(a) (b)

Current parameters

Note that the actual current directions can be opposite to those shown in the figure; depending on the family.

In most data books, the current flowing into nodes or devices is considered positive, and the current flowing out from the node or device is considered as negative.

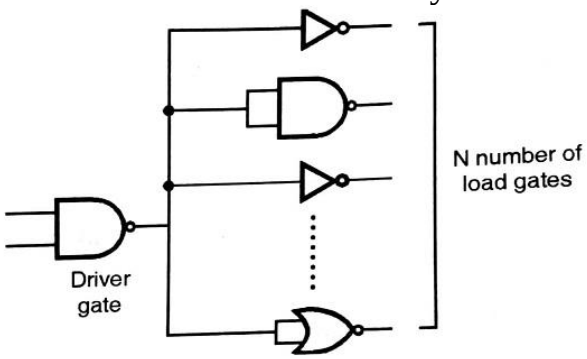
2. Fan-in and Fan-out:

Fan-in :

- The fan in is defined as the number of inputs a gate has. For example, a two-input gate will have a fan-in is equal to 2.

Fan-out:

- Fan-out is defined as the maximum number of inputs of the same IC family that a gate can drive without falling outside the specified output voltage limits.
- Higher the fan-out, the higher the current supplying capacity of a gate. For example, a fan-out of 5 indicates that the gate can drive (supply current to) at the most 5 inputs of the same IC family.
- The concept of fan-out will be more clear if you refer to the figure given below:



Fan-out

- As shown in the figure, the fan-out of the driver gate which is driving the N number of the gate is N.

- Fan-out is also called the loading factor. If the specified fan out of a gate is 10 then we should not load it with more than 10 gates. Because then the output logic level voltages can not be guaranteed.
- Fan out depends on the nature of the input devices that are connected to an output.
- Unless a different logic family is specified as the load device, fan-out is assumed to be referred to load device of the same family as the driving device.

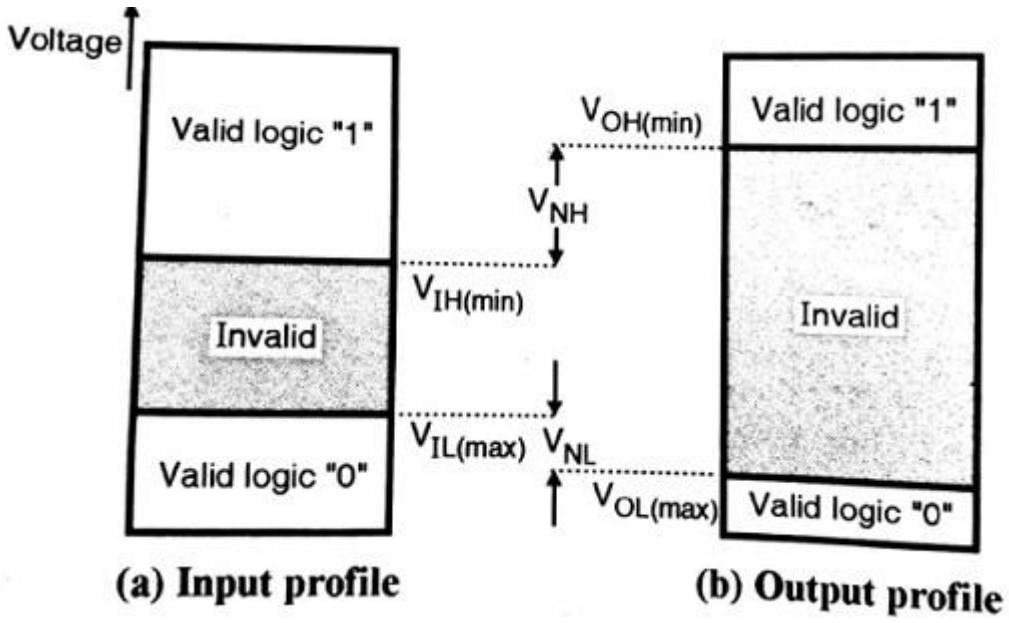
3. Noise Margin :

To understand the meaning of the term "**Noise Margin**" or "**Noise Immunity**", refer to the input and output voltage profiles shown in the figure.

Noise is unwanted electrical disturbances that may induce some voltage in the connecting wires used between two gates or from a gate output to load.

Noise immunity is defined as the ability of a logic circuit to tolerate the noise without causing any unwanted changes in the output.

A quantitative measure of noise immunity is called **noise margin**.



In order to avoid the effect of noise voltage, the voltage levels $V_{OH(min)}$ and $V_{IH(min)}$ are adjusted to different levels with some difference between them as shown in the above figure.

The difference between $V_{OH(min)}$ and $V_{IH(min)}$ is known as **high-level noise margin V_{NH}.**

similarly, the difference between VIL(max) and VOL(max) is called the **low-level noise margin V_{NL}.**

High-Level Noise Margin, $V_{NH} = V_{OH(min)} - V_{IH(min)}$

Low Level Noise Margin, $V_{NL} = V_{IL(max)} - V_{OL(max)}$

When a high logic output is driving a logic circuit input, any negative noise spike greater than V_{NH} can cause the voltage to drop into the invalid range.

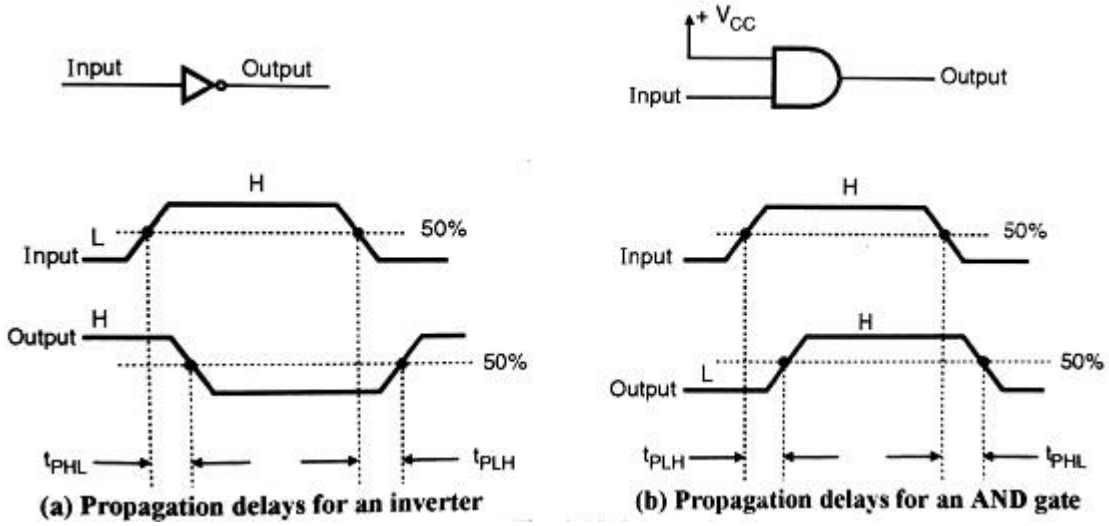
Similarly, when a low logic output s driving a logic circuit input, any positive spikes greater than V_{NL} can cause the voltage to go into the invalid range.

4. Propagation Delay(Speed of operation) :

The output of logic does not change its state instantaneously in response to the change in the state of the input.

There is a time delay between those two events, which is called propagation delay.

Thus propagation delay is defined as the time delay between the instant of application of an input pulse and the instant of occurrence of the corresponding output pulse. This is shown in the figure.



(a) Propagation delays for an inverter

(b) Propagation delays for an AND gate

From the above figure, it is observed that there are two propagation delays.

1. **t_{PHL}**: The propagation delay is measured when the output makes a transition from a HIGH (1) to a LOW (0) state.

2. **t_{PLH}**: The propagation delay is measured when the output makes a transition from a LOW (0) to a HIGH (1) state.

There are some important points that you have to remember :

1. The values of t_{PHL} and t_{PLH} are not always the same. If they are not equal then the one which is higher is considered as the propagation delay time of the gate.
2. The propagation delays are measured between the points corresponding to 50% levels as shown in the figure.

Ideally, the propagation delay should be zero and practically it should be as short as possible.

The values of propagation delays are used as a measure of the relative speed of logic circuits.

For example, a logic circuit with a propagation delay time of 5ns will be faster than the one with a 10ns propagation delay time.

5. Power Dissipation :

As a result of applied voltage and currents flowing through the logic families (ICs), some power will be dissipated in it in the form of heat.

The power is in milliwatts.

Care should be taken to reduce the power dissipation taking place in the logic ICs in order to protect the ICS against damage due to excessive heat, to reduce the loading on power supplies, etc.

Another importance of power dissipation is that the product of power dissipation and propagation time is always constant.

Therefore reduced power dissipation may lead to an increase in propagation delay.

Usually, there is only one power supply terminal on any ICs. It is denoted by V_{CC} for the TTL ICs and V_{DD} for the CMOS ICs.

The power drawn by an IC from the power supply is given by,

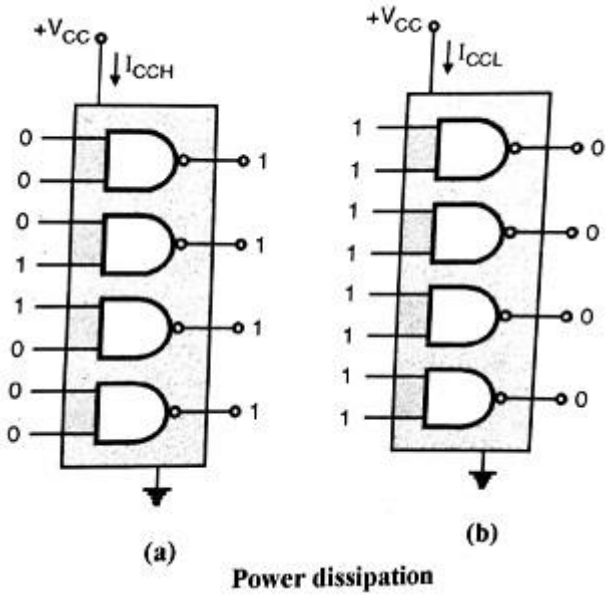
$$\mathbf{P = V_{CC} X I_{CC}}$$

- Where I_{CC} is the current drawn from the power supply.

For many ICs the current drawn from the power supply will be dependent on the logic states of the circuit on the chip.

In the below figure (a), a NAND gate IC with all its outputs is high. the current drawn from the source under such conditions is denoted by I_{CCH}.

The below figure (b), shows another extreme condition where the outputs al all the NAND gates are "0". The current drawn from the source under such conditions is denoted by I_{CCL}.



Power dissipation

The values of I_{CCH} and I_{CCL} are measured with open-circuited outputs because the load will change these values.

I_{CCH} and I_{CCL} are different values, so an average value of them is generally considered to calculate average power dissipation.

$$\mathbf{I_{CC(avg)} = (I_{CCH} + I_{CCL} / 2)}$$

$$\mathbf{P_{D(avg)} = V_{CC} X I_{CC(avg)}}$$

6.Operating Temperature:

The temperature range acceptable for consumer and industrial applications is 0^0 and 70^0 C and that for military applications is -55º to 125º C.

The performance of gates will be within the specified limits over these temperature ranges.

7. Figure of Merit (Speed Power Product SPP) :

The figure of merit of a logical family is the product of power dissipation and propagation delay.

It is called the speed-power product. The speed is specified in seconds and power is specified in watts.

Figure if Merit = Propagation delay time x Power dissipation

Practically, the value of the figure of merit should be as low as possible.

The figure of merit is always a compromise between speed and power dissipation. That means if we try to reduce the propagation delay then the power dissipation will increase and vice-versa.

The speed-power product is used as a common means for measuring and comparing the overall performance of different IC families.

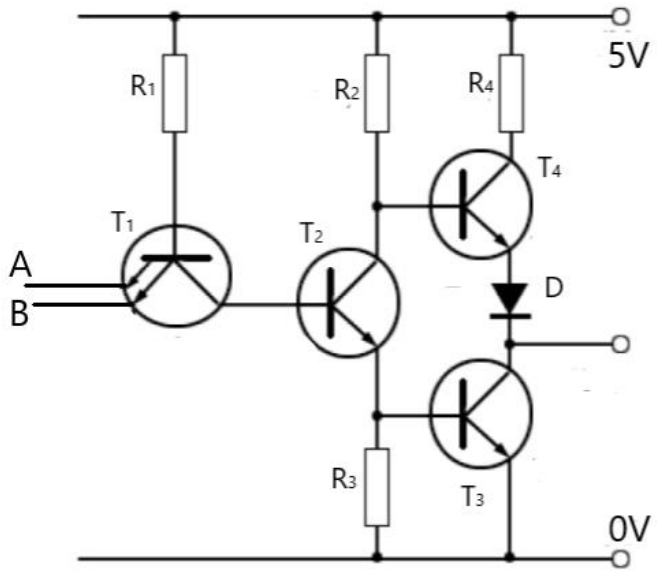
Suppose that an IC family has an average propagation delay of 20ns and an average power dissipation of 5mW, then its SPP is given by,

$$SPP=20ns \times 5\ mW = 100 \times 10^{-12}\ Watt\text{-}second$$

$$= 100\ picojoules$$

Features, circuit operation &various applications of TTL(NAND), CMOS (NAND & NOR)

A TTL NAND gate logic circuit has at least four bi polar junction transistors in which the input (transistor which receives input) has two emitters. The output is taken between the emitter and collector of two different transistor



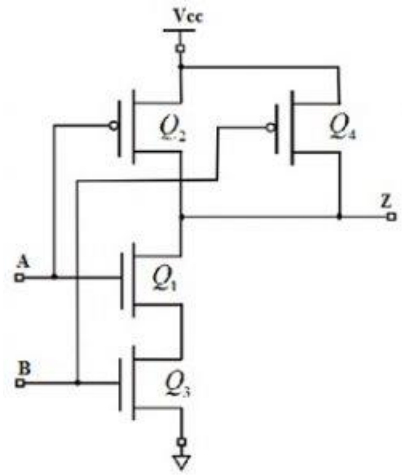
The above diagram is the circuit diagram of a TTL NAND gate.
From the diagram, we shall explain the working. Now, as seen, the transistor T1 has two emitters to allow two inputs into the transistor. Now, as connected the base voltage will be at 5V. if both inputs are logic 1 (usually means about 5V too), the potential difference across base and emitter would be zero or nearly. Hence, no current will flow and the transistor is turned off. So, the collector voltage would also be equal to about 5V. Hence, this potential can drive current through the emitter of the transistor T2. This then will allow the collector voltage of the transistor T2 to fall.
Now due to the current flowing through the emitter, there would be a voltage drop across the resistor R3. The desired voltage drop would be about 0.7V. As seen, this is the input of the transistor T3. Hence, the transistor is turned on. Due to saturation, the collector voltage will fall to about 0.2Vwhich is a logic 0. A
For the transistor T4, observe that the emitter voltage is made up of the entire voltage of the transistor T3 plus the voltage drop across the diode D about 0.7V. Hence the emitter potential would be 0.7+0.2=0.9V. Now the base voltage of the transistor T4, would be the voltage across the base-emitter of T3 and the voltage of the entire transistor (i.e.) voltage across emitter-collector. This would also be equal to about 0.9V. Hence the emitter voltage and the collector voltage are equal. So the transistor T4 will be turned off too. So the output is zero when both inputs are 1.

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

CMOS NAND Gate

The below figure shows a 2-input Complementary MOS NAND gate. It consists of two series NMOS transistors between Y and Ground and two parallel PMOS transistors between Y and VDD.

If either input A or B is logic 0, at least one of the NMOS transistors will be OFF, breaking the path from Y to Ground. But at least one of the pMOS transistors will be ON, creating a path from Y to VDD.



Two Input NAND Gate

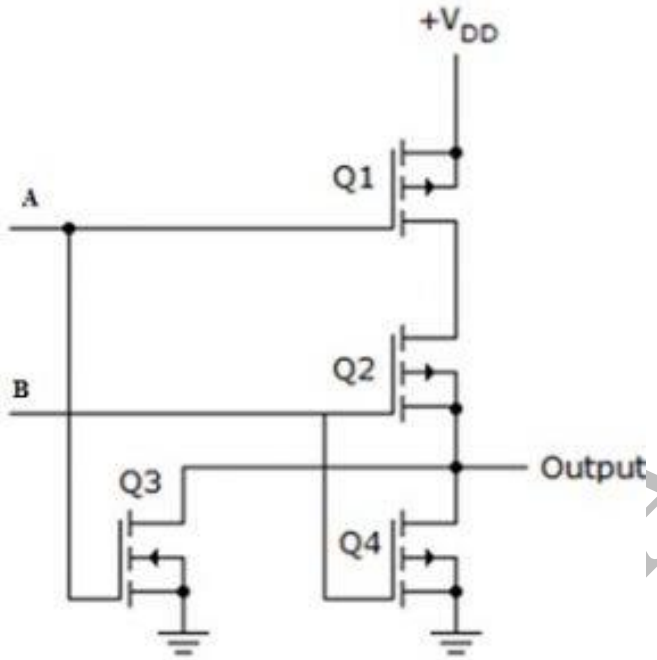
Hence, the output Y will be high. If both inputs are high, both of the nMOS transistors will be ON and both of the pMOS transistors will be OFF. Hence, the output will be logic low. The truth table of the NAND logic gate given in the below table.

A	B	Pull-Down Network	Pull-up Network	OUTPUT Y
0	0	OFF	ON	1
0	1	OFF	ON	1
1	0	OFF	ON	1
1	1	ON	OFF	0

CMOS NOR Gate

A 2-input NOR gate is shown in the figure below. The NMOS transistors are in parallel to pull the output low when either input is high. The PMOS transistors are in series to pull the

output high when both inputs are low, as given in the below table. The output is never left floating.



Two Input NOR Gate

The truth table of the NOR logic gate given in the below table.

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0