1.1 INTRODUCTION:

- Electronic systems usually deal with information. Representation of information is called a **signal.** Signal in electronics is generally in form of voltage or current. Value of a signal is proportional to some physical quantity and it gives information about it. For example, temperature represented in terms of voltage signal.

- There are two types of signals which are different in terms of their characteristics with respect to time and value.
  1. Analog Signals
  2. Digital Signals

- A signal whose value is defined at all instances of time is called **continuous time signal.** On the other hand signal whose values are defined only at discrete instances of time is called **discrete time signal.** Most of the signals that occur in nature are analog in form. A discrete time signal can be obtained from continuous time signal by process called **sampling.** This has been illustrated in Fig. 1.1.

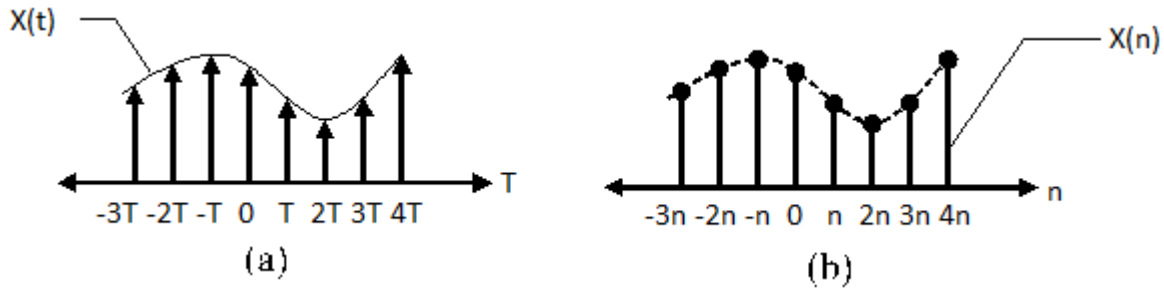


Fig. 1.1: (a) Continuous time signal x(t) sampled at every T interval, (b) Resulting discrete time signal x(n)

- Similarly if a signal can take any value in a given range between some minimum and maximum value then the signal is called **continuous value signal.** On the other hand if a signal takes only certain fixed values in a given range then it is called **discrete value signal**. The process of converting a continuous value signal to a discrete value signal is called **quantization.** This is illustrated in Fig. 1.2.

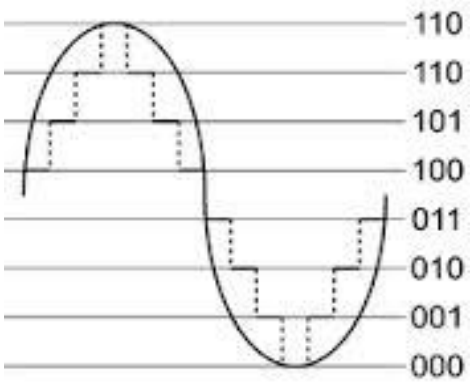


Fig. 1.2: Continuous value signal (solid line) and discrete value signal (dotted line)

Analog signal: Signals that are continuous in time and continuous in value are called analog signal.

Digital signal: Signals that are discrete in time and discrete in values are called digital signals. Digital signals are generally processed by digital systems like computers and hence their values are represented in terms of binary as shown in Fig. 1.2.

- Analog signal being continuous in time will have infinite values in any given period of time. Practically a digital system like computer cannot handle infinite values due to limited physical resources and processing power. This is the reason why a continuous time signal has to be sampled and converted to discrete time signal.

- Again analog signals are continuous in value and hence can take any value in a given range. Now ideally number of values in any given range will be infinite which cannot be represented by finite number of bits on a computer. For example, as shown in Fig. 1.2, with three bits used for representing values only eight different values can be represented. Thus a continuous value signal has to be quantized and converted to discrete value signal.

1.1.1 Levels of Integration

- Digital electronic circuits have become increasingly popular and successful due to integrated circuit (IC) technology. Advancement in IC technology has made it possible to construct large number of devices (eg. transistor, diode, resistors, capacitors, etc) on a very small chip. Classification of IC technology based on number of components per chip is as follows.

    1. Small-scale integration (SSI), containing fewer than 100 components
    2. Medium-scale integration (MSI), containing 100 to 1000 components
    3. Large-scale integration (LSI), containing 1000 to 10,000 components
    4. Very large-scale integration (VLSI), containing more than 10,000 components

1.1.2 Comparison of Analog and Digital Systems

	Analog Systems	Digital Systems
1	Analog systems operate on continuous time and continuous value signals.	Digital systems operate on discrete time and discrete value signals generally represented in binary.
2	Analog systems are difficult to design.	Digital systems are easy to design as most of the components are in form of Integrated circuits (IC).
3	Analog systems are mostly custom made and lack flexibility.	Digital systems have high degree of flexibility.
4	Less efficient in storage of information.	More efficient in storage of information.

5	Analog signal processed by these systems are affected by noise very easily.	Digital signal are more noise-immune compared to analog signals.
6	Relatively costly compared to digital system	Low cost due to mass production of components.
7	Analog systems are more sensitive to parameter variation.	Digital systems are less sensitive to parameter variation
8	No conversion of input signals are required before processing	Input signals are converted from analog to digital form before it is processed
9	As no conversion of input signal is required there is no loss of information.	Due to process of sampling and quantization there is loss of information.
10	Analog systems are more efficient for real time processing	Digital systems may offer limitations for real time processing

1.2 Introduction to Digital System:

- A digital system uses a building blocks approach. Many small operational units are interconnected to make up the overall system.
- The most basic logical unit system is gate circuit. There are several different types of gates with each perform differently from other logic gates.
- Digital signal consist of only two values, '0' and '1'. These two values have logical meaning i.e. '1' represents the existence of particular condition and '0' represents the absence of condition.

- There are two types of tables are used in digital system:

  1. **Truth Table:**
     Truth table plots inputs and outputs in terms of 1s and 0s.

  2. **Function Table:**
     Function table plots inputs and outputs in term of HIGH and LOW voltage levels.

- The design of digital system may be roughly divided into three stages;

  1. **System Design:**
     It involves breaking the overall system into subsystem and specifying the characteristics of each subsystem. For example, the system design of a digital computers involves specifying the number and type of memory, ALU and i/p – o/p devices.

  2. **Logic Design:**
     It involves how to interconnect basic logic building blocks to perform specific function. For example, to make a flip flop different logic gates are needs to be connected in specific manner.

3. **Circuit Design:**
   It involves specifying the interconnection of specific components like resistors, transistors, diodes, CMOS etc. to create a logic gates.

1.2.1 Advantages of Digital Systems

1. Digital systems are easier to design
2. Information storage is easy
3. Accuracy and precision are greater
4. Digital systems are more versatile
5. Digital circuits are less affected by noise
6. More digital circuitry can be fabricated on IC chips

1.2.2 Logic Levels and Different types of Logics

- Digital system use the binary number system. Therefore, two-state devices are used to represent the two binary digits 1s & 0s by two different voltage levels, called HIGH and LOW.
- Normally, the binary 0 and 1 are represented by the logic voltage levels 0 V and +5 V.
- Usually any voltage between 0 V to 0.8 V represents the logic 0 and any voltage between 2 V to 5 V represents the logic 1. This voltage levels can be varies according to the different logical systems.
- There are three types of logics available in digital systems.
  1. Positive Logic
  2. Negative Logic
  3. Mixed Logic

1. **Positive Logic:**
   In positive logic high voltage level is represent as logic 1 and low voltage level is represent as logic 0.

Fig. 1.3: Illustration of positive logic

2. **Negative Logic:**
   In positive logic high voltage level is represent as logic 0 and low voltage level is represent as logic 1.

Fig. 1.4: Illustration of negative logic

3. Mixed Logic:

This scheme uses positive logic in some portions (e.g inputs) of the system while applying negative logic (e.g. outputs) in other portion of the system.

- Suppose some function X = AB' + A'B for this function the representation of all the logics are as follow;

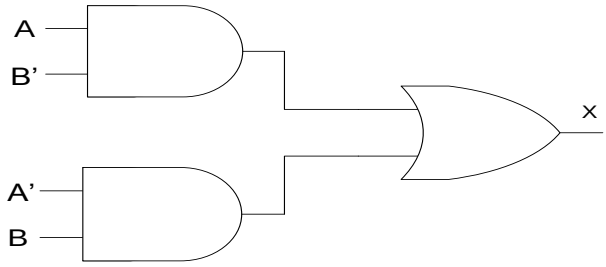


Fig. 1.5: Representation of function X = AB' + A'B

- Truth table of the given function for all the logics is shown as follow;

Table 1.1: Truth table of Positive logic, Negative logic, Mixed logic for X = AB' + A'B

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

Positive Logic

A	B	X
1	1	1
1	0	0
0	1	0
0	0	1

Negative Logic

A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

Mixed Logic

1.3 NUMBER SYSTEMS:

1.3.1 Introduction to Number System:

- **Definition & Importance**

- Number system is the basis for counting various items. On hearing the word 'number', we immediately think of the familiar decimal number system with 10 digits 0 to 9. But modern computers communicate and operate with binary numbers which use only 2 digits 0 & 1. Also different types of number systems like octal and hexadecimal are also used widely. Depending upon the type of number system, we use different digits to represent various numbers.

➢ **Few Common Aspects to All Numbering Systems**

(i) Base or Radix

The number of symbols used for the representation of numbers in a number system is known as its Base or Radix and is generally denoted by r.

(ii) Digit

Each symbol in the number system is called a Digit.

(iii) The largest value of a digit is always one less than the base

For ex, in decimal system, the largest digit is 9 (since base is 10)

(iv) Each digit position (i.e. place) represents a different multiple of base

This means that the numbers have positional importance. Hence the number systems are known as **Positional Weighted Number System**. It means that the value attached to a symbol depends on its location with respect to the decimal point.

- For example decimal number 123.4 (base 10) can actually be represented as;

  $(123.4)_{10} = 1x10^2 + 2x10^1 + 3x10^0 + 4x10^{-1}$

- In general, a number of any radix can be expressed as,

  $Nr = \ldots + D3 \times r^3 + D2 \times r^2 + D1 \times r^1 + D0 \times r^0 + D{-}1 \times r^{-1} + D{-}2 \times r^{-2} + D{-}3 \times r^{-3} + \ldots$

Where;

r is the base and D_i is any valid digit in the number system of base r.

- The digits on the left side of the decimal point form the integer part of a number and those on the right side form the fractional part.
- The left most digit in any number representation, which has the greatest positional weight out of all the digits present in that number is called the most significant digit (MSD).
- The right most digit in any number representation, which has the least positional weight out of all the digits present in that number is called the least significant digit (LSD).

➢ **Various Numbering Systems**

- Different number systems are used in various applications. The commonly used number systems along with their base, 1st digit, last digit and available digits are as shown below:

Table 1.2: Illustration of various number system

Sr. No	Number System	Base	First digit	Last digit	All digits
1	Binary	2	0	1	0,1
2	Octal	8	0	7	0,1,2,3,4,5,6,7
3	Decimal	10	0	9	0,1,2,3,4,5,6,7,8,9
4	Hexadecimal	16	0	F	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Note: In hexadecimal number system, meaning of A≈10, B≈11, C≈12, D≈13, E≈14 & F≈15.

1. DECIMAL NUMBER SYSTEM

- Decimal number system is the most familiar no. system used in day-to-day life. The decimal system consists of 10 unique symbols. Hence the **base or radix is 10**. It is a positional weighted system. In this system, any number (integer, fraction or mixed) of any magnitude can be represented by the use of these ten symbols only.

- The digits on the left side of the decimal point form the integer part of a decimal number while those on right side from the fractional part. The digits on the right of the decimal point have weights which are negative powers of 10 and the digits to the left of the decimal point have weights which are positive powers of 10. The sum of all the digits multiplied by their weights gives the total number being represented.

- In general, the value of any mixed decimal number

$$d_n\, d_{n-1}\, d_{n-2} \ldots d_1\, d_0 \,.\, d_{-1}\, d_{-2}\, d_{-3} \ldots d_{-k}$$

is given by

$$(d_n \times 10^n) + (d_{n-1} \times 10^{n-1}) + \ldots + (d_1 \times 10^1) + (d_0 \times 10^0) + (d_{-1} \times 10^{-1}) + \ldots + (d_{-k} \times 10^{-k})$$

- Consider a decimal no. 9256.26. We represent it as:

$$9256.26 = 9 \times 1000 + 2 \times 100 + 5 \times 10 + 6 \times 1 + 2 \times (1/10) + 6 \times (1/100)$$

$$= 9 \times 10^3 + 2 \times 10^2 + 5 \times 10^1 + 6 \times 10^0 + 2 \times 10^{-1} + 6 \times 10^{-2}$$

MSD . . . . 10^3 10^2 10^1 10^0 10^{-1} 10^{-2} 10^{-3} . . . . LSD

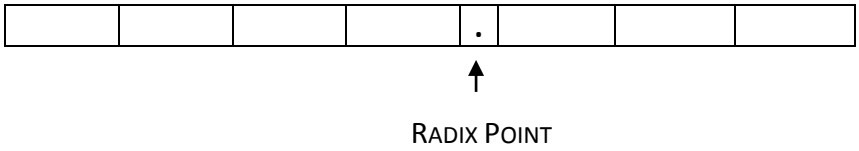


Fig. 1.6: Decimal position values as power of 10

2. BINARY NUMBER SYSTEM

- The binary number system is a positional weighted system. The **base or radix** of this number system is **2**. Hence, it has two independent symbols. The base itself cannot be a symbol. The symbols used are 0 & 1. A binary digit is called a *bit*. A binary number consists of a sequence of bits, each of which is either a 0 or a 1. The binary point separates the integer and fraction part. The weight of each bit position is one power of 2 greater than the weight of the position to its immediate right. The place values left on the binary point in binary are 64, 32, 16, 8, 4, 2 and 1.

- In general, the value of any mixed binary number

$$b_n\ b_{n-1}\ b_{n-2} \ldots b_1\ b_0\ .\ b_{-1}\ b_{-2}\ b_{-3} \ldots b_{-k}$$

is given by

$(b_n \times 2^n) + (b_{n-1} \times 2^{n-1}) + \ldots + (b_1 \times 2^1) + (b_0 \times 2^0) + (b_{-1} \times 2^{-1}) + \ldots + (b_{-k} \times 2^{-k})$

MSB .... 2^3 2^2 2^1 2^0 2^{-1} 2^{-2} 2^{-3} .... LSB

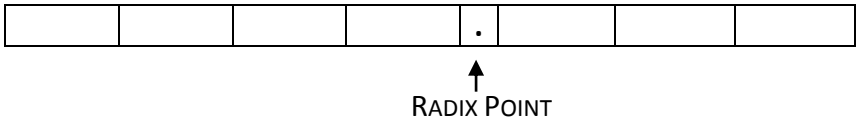


Fig. 1.7: Binary position values as power of 2

➢ Counting in Binary

- Counting in binary is very similar to decimal counting. Start counting with 0, the next count is 1. Moving ahead, we put 1 in the column to the left and continue the counting. Thus, 11 is the maximum we can count using two bits. Similarly, we can continue counting with 5, 6, ... bits.

Table 1.3: Counting in Binary

Decimal Number	Binary Number	Decimal Number	Binary Number
0	00000	11	01011
1	00001	12	01100
2	00010	13	01101
3	00011	14	01110
4	00100	15	01111
5	00101	16	10000
6	00110	17	10001
7	00111	18	10010
8	01000	19	10011
9	01001	20	10100
10	01010	21	10101

➢ **Applications**

- The binary number system is used in digital computers because the switching circuits used in these computers use two-state devices such as transistors, diodes, etc. These devices have to exist in one of the two possible states: ON of OFF, OPEN or CLOSED. So, these two states can be represented by the symbols 0 and 1, respectively.

3. OCTAL NUMBER SYSTEM

- The octal number system was extensively used by early minicomputers. It is also a positional weighted system. Its **base or radix is 8**. It has 8 independent symbols **0 to 7**.
- Since its base $8 = 2^3$, every 3-bit group of binary can be represented by an octal digit. An octal number is, thus 1/3 rd. the length of the corresponding binary number.

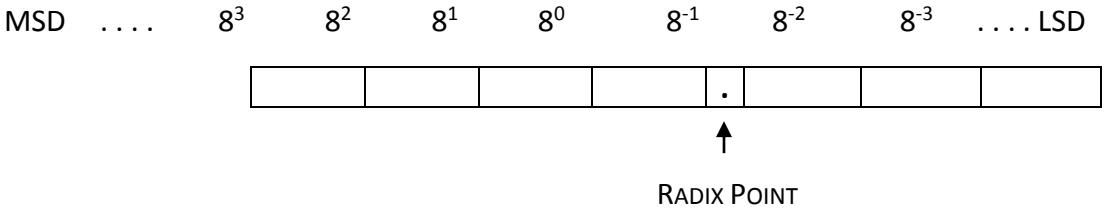


Fig. 1.8: Octal position values as power of 8

Table 1.4: Counting in Octal

Decimal Number	Octal Number	Decimal Number	Octal Number
0	0	11	13
1	1	12	14
2	2	13	15
3	3	14	16
4	4	15	17
5	5	16	20
6	6	17	21
7	7	18	22
8	10	19	23
9	11	20	24
10	12	21	25

➢ **Usefulness of the Octal System**

- In computer work, binary numbers up to 64 bits are not uncommon. These binary numbers do not always represent a numerical quantity; they often represent some type of code. While dealing with

large binary numbers, it is convenient and more efficient for us to write the numbers in octal rather than binary. The ease with which conversions can be made between octal and binary makes the octal system more attractive as a shorthand means of expressing large binary numbers.

4. HEXADECIMAL NUMBER SYSTEM

- Binary numbers are too long. These numbers are fine for machines butt are too lengthy to be handled by human beings. So, there is a need to represent the binary numbers concisely. One number system developed with this objective is the hexadecimal number system (or Hex). Although it is somewhat difficult to interpret than the octal number system, it has become the most popular means of direct data entry and retrieval in digital systems.

- The hexadecimal number system is positional weighted system. The **base or radix is 16** that means, it has 16 independent symbols. The symbols used are **0 to 9 and A to F**. since its base is 16 = 2^4, every 4 bit binary digit combination can be represented by one hexadecimal digit. So, a hexadecimal number is ¼ th the length of the corresponding binary number.

- A 4-bit group is called a *nibble*. Since computer words come in 8, 16, 32 bits and so on, they can be easily represented in hexadecimal. The hexadecimal number system is particularly used for human communications with computers. It is used in both large and small computers.

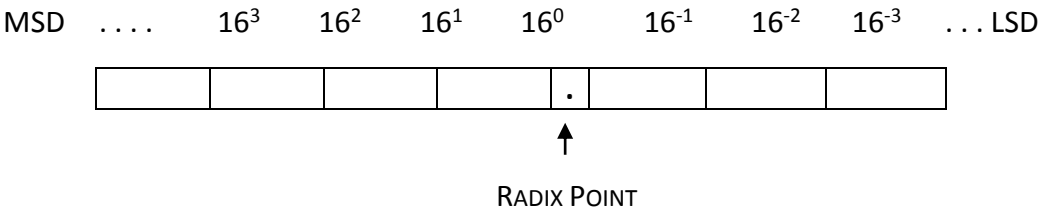


Fig. 1.9: Hexadecimal position values as power of 16

Table 1.5: Counting in Hexadecimal

Decimal Number	Hexadecimal Number	Decimal Number	Hexadecimal Number
0	0	21	15
1	1	22	16
2	2	23	17
3	3	24	18
4	4	25	19
5	5	26	1A
6	6	27	1B
7	7	28	1C
8	8	29	1D
9	9	30	1E

10	A	31	1F
11	B	32	20
12	C	33	21
13	D	34	22
14	E	35	23
15	F	36	24
16	10	37	25
17	11	38	26
18	12	39	27
19	13	40	28
20	14	41	29

1.3.2 Number Base Conversions:

➢ **Definition and Importance**

- The human beings use decimal number system while computer uses binary number system. Therefore, it is essential to convert decimal number into its equivalent binary while feeding number into computer and to convert binary number into its decimal equivalent while displaying result of operation to the human beings.

- However, dealing with a large quantity of binary numbers of many bits is inconvenient for human beings. Therefore, octal and hexadecimal numbers are used as a shorthand means of expressing large binary numbers. Hence inter conversion among different number systems is required.

➢ **Conversions between Decimal, Binary, Octal and Hexadecimal**

- The below table shows the decimal, binary, octal and hexadecimal numbers.

Table 1.6: Conversions between Decimal, Binary, Octal and Hexadecimal

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8

9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

1. Binary to Decimal Conversion

- Binary numbers can be converted to their decimal equivalents by the positional weights method. In this method, each binary digit of the number is multiplied by its position weight and the product terms are added to obtain the decimal number.

Ex 1: Convert $(10101)_2$ to decimal

Solution:

Positional weights are: $2^4\ 2^3\ 2^2\ 2^1\ 2^0$

$1\ 0\ 1\ 0\ 1 = (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$

$= 16 + 0 + 4 + 0 + 1$

$= 21$

Hence, $(10101)_2 = (21)_{10}$

Ex 2: Convert $(11011.101)_2$ to decimal

Solution:

Positional weights are: $2^4\ 2^3\ 2^2\ 2^1\ 2^0\ .\ 2^{-1}\ 2^{-2}\ 2^{-3}$

$1\ 1\ 0\ 1.1\ 0\ 1 = (1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3})$

$= 16 + 8 + 0 + 2 + 1 + 0.5 + 0 + 0.125$

$= 27.625$

Hence, $(11011.101)_2 = (27.625)_{10}$

2. Octal to Decimal Conversion

- To convert an octal number to a decimal number, multiply each digit in the octal number by the weight of its position and add all the product terms.

Ex 3: Convert $(4057.06)_8$ to decimal

Solution:

Positional weights are: $8^3\ 8^2\ 8^1\ 8^0\ .\ 8^{-1}\ 8^{-2}$

$4057.06_8 = (4 \times 8^3) + (0 \times 8^2) + (5 \times 8^1) + (7 \times 8^0) + (0 \times 8^{-1}) + (6 \times 8^{-2})$

$= 2048 + 0 + 40 + 7 + 0 + 0.0937$

$= 2095.0937$

Hence, $(4057.06)_8 = (2095.0937)_{10}$

3. Hexadecimal to Decimal Conversion

- Multiply each digit in the hex number by its position weight and add all those product terms. In this way, we get the decimal equivalent of the hexadecimal number.

Ex 4: Convert $(5C7)_{16}$ to decimal

Solution:

Positional weights are: $16^2\ 16^1\ 16^0$

$5C7_{16} = (5 \times 16^2) + (12 \times 16^1) + (7 \times 16^0)$

$= 1280 + 192 + 7$

$= 1479_{10}$

Hence, $(5C7)_{16} = (1479)_{10}$

Ex 5: Convert $A0F9.0EB_{16}$ to decimal

Solution:

Positional weights are: $16^3\ 16^2\ 16^1\ 16^0 . 16^{-1}\ 16^{-2}\ 16^{-3}$

$A0F9.0EB_{16} = (10 \times 16^3) + (0 \times 16^2) + (15 \times 16^1) + (9 \times 16^0) + (0 \times 16^{-1}) + (14 \times 16^{-2}) + (11 \times 16^{-3})$

$= 40960 + 0 + 240 + 9 + 0 + 0.0546 + 0.0026$

$= 41209.0572_{10}$

Hence, $(A0F9.0EB)_{16} = (41209.0572)_{10}$

4. Decimal to Binary Conversion

- The conversion of decimal number to binary is carried out in 2 steps. In step 1, we have to convert integer part and in step 2, we have to convert fractional part.

- For **integer part** conversion, we use successive division-by-2 method. In this method we repeatedly divide the integer part of the decimal number by 2 until the quotient is zero. The remainder of each division becomes the numeral in the new radix. The remainders are taken in the reverse order to form a new radix number. This means that the first remainder is the LSD and the last remainder is the MSD in the new radix number. Thus the integers read from **bottom to top** give the equivalent binary fraction.

- Similarly, for **fractional part**, we use successive multiplication-by-2 method. In this method, the number to be converted is multiplied by the radix of new number, producing a product that has an integer part and a fractional part. The integer part (carry) of the product becomes a numeral in the new radix number. The fractional part is again multiplied by the radix and this process is repeated until fractional part reaches 0 or until the new radix number is carried out to significant digits. The integer part (carry) of each product is read from **top to bottom** to represent the new radix number.

Ex 6: Convert 52_{10} to binary

- Here the number is integer number so we need to divide the given decimal number by 2 and read the remainders from bottom to top to get the equivalent binary number.

		Remainder
2	52	
2	26	0
2	13	0
2	6	1
2	3	0
2	1	1
	0	1

Hence, $(52)_{10} = (110100)$

Ex 7: Convert $(163.875)_{10}$ to binary

Solution:

Step 1: Separate the integer and fractional parts of the decimal number. Now for integer part, we carry successive division-by-2 method as follows:

		Remainder
2	163	
2	81	1
2	40	1
2	20	0
2	10	0

2	5	0
2	2	1
2	1	0
	0	1

So, $163_{10} = (10100011)_2$

Step 2: Now the fraction part is 0.875_{10}. Carrying out successive multiplication-by-2 as follows:

0.875 x 2 = 1.75	1
0.75 x 2 = 1.5	1
0.5 x 2 = 1.0	1

So, $0.875_{10} = 0.111_2$

Hence. $(163.875)_{10} = (10100011.111)_2$

5. Decimal to Octal Conversion

- Decimal to Octal conversion can be done in similar way as decimal to binary conversion. The integer and fractional parts are to be separated and the same procedure is carried out. But the division and multiplication are carried out by 8 as the base of octal number is 8. Following the same steps, we can get the equivalent octal number of the given decimal number.

Ex 8: Convert 378.93_{10} to octal

Solution:

Step1: Conversion of integer part by successive division-by-8 method.

8	378	Remainder
8	47	2
8	5	7
	0	5

So, $(378)_{10} = (572)_8$

Step 2: Conversion of fractional part by successive multiplication-by-8 method.

0.93 x 8 = 7.44	7
0.44 x 8 = 3.52	3
0.52 x 8 = 4.16	4
0.16 x 8 = 1.28	1

So, $0.93_{10} = 0.7341_8$

Hence, $(378.93)_{10} = (572.7341)_8$

6. Decimal to Hexadecimal Conversion

- Decimal to hexadecimal conversion is carried out by 2 steps. In the first step, the integer part of the decimal number is divided by 16 successively and the remainder is noted. The remainders read from bottom to top gives the equivalent hexadecimal integer. In the second step, the successive multiplication of fractional part by 16 is done and the integers are noted down. Reading the integers from bottom to top gives the hexadecimal fraction.

Ex 9: Convert $(3509.75)_{10}$ to hexadecimal

Solution:

Step1: Conversion of integer part by successive division-by-16 method

		Remainder
16	3509	
16	219	5
16	13	11 = B
	0	13 = D

So, $(3509)_{10} = (DB5)_{16}$

Step 2: Conversion of fractional part by successive multiplication-by-16 method.

0.75 x 16 = 12.0 12 = C

So, $(0.75)_{10} = (0.C)_{16}$

Hence, $(3509.75)_{10} = (DB5.C)_{16}$

7. Octal to Binary Conversion

- To convert a given octal number to binary, just replace each octal digit by its 3-bit binary equivalent.

Ex 9: Convert 367.52_8 to binary.

Solution:

Given octal number is	3	6	7	.	5	2
Convert each octal digit to binary	011	110	111	.	101	010

Hence, $(367.52)_8 = (11110111.101010)_2$

8. Hexadecimal to Binary Conversion

- To convert a given hexadecimal number to binary, just replace each hexadecimal digit by its 4-bit binary equivalent.

Ex 10: Convert $4BAC_{16}$ to binary.

Solution:

Given hexadecimal number is	4	B	A	C
Convert each digit to 4-bit binary	0100	1011	1010	1100

Hence, $(4BAC)_{16} = (100101110101100)_2$

Ex 11: Convert $3A9.B0D_{16}$ to binary.

Solution:

Given hexadecimal number is	3	A	9	.	B	0	D
Convert each digit to 4-bit binary	0011	1010	1001	.	1011	0000	110

Hence, $(3A9.B0D)_{16} = (1110101001.101100001101)_2$

9. Binary to Octal Conversion

- To convert a binary number to an octal number, starting from the binary point make groups of 3 bits each, on either side of the binary point and replace each 3-bit binary group by the equivalent octal digit.

Ex 12: Convert 110101.101010_2 to octal.

Solution:

Group of 3 bits are	110	101	.	101	010
Convert each group to octal	6	5	.	5	2

Hence, $(110101.101010)_2 = (65.52)_8$

Ex 13: Convert 10101111001.0111_2 to octal.

Solution:

Group of 3 bits are	10	101	111	001	.	011	1

= 010 101 111 001 . 011 100

Convert each group to octal 2 5 7 1 . 3 4

Hence, $(10101111001.0111)_2 = (2571.34)_8$

10. Binary to Hexadecimal Conversion

- To convert a binary number to an octal number, starting from the binary point make groups of 4 bits each, on either side of the binary point and replace each 4-bit binary group by the equivalent hexadecimal digit.

Ex 14: Convert 1011111011.011111_2 to hexadecimal.

Solution:

Group of 4 bits are 10 1111 1011 . 0111 11

0010 1111 1011 . 0111 1100

Convert each group to hex 2 F B . 7 C

Hence, $(1011111011.011111)_2 = (2FB.7C)_{16}$

11. Octal to Hexadecimal Conversion

- To convert an octal number to hexadecimal, the simplest way is to first convert the given octal number to binary and then the binary number to hexadecimal.

Ex 15: Convert 1245 to hex.

Solution:

Given octal number is 1 2 4 5

Convert each octal digit to binary 001 010 100 110

Group of 4 bits are 0010 1010 0110

Convert each 4–bit group to hex 2 A 6

Hence, $(1245)_8 = (2A6)_{16}$

Ex 16: Convert 756.603_8 to hex.

Solution:

Given octal number is	7	5	6	.	6	0	3
Convert each octal digit to binary	111	101	110	.	110	000	011
Group of 4 bits are	0001	1110	1110	.	1100	0001	1000
Convert each 4–bit group to hex	1	E	E	.	C	1	8

Hence, $(756.603)_8 = (1EE.C18)_{16}$

12. Hexadecimal to Octal Conversion

- To convert hexadecimal number to octal, the simplest way is to first convert the given hexadecimal number to binary and then the binary number to octal.

Ex 17: Convert $B9F.AE_{16}$ to octal.

Solution:

Given hex number is	B	9	F	.	A	E		
Convert each hex digit to binary	1011	1001	1111	.	1010	1110		
Group of 3 bits are	101	110	011	111	.	101	011	100
Convert each 3–bit group to octal	5	6	3	7	.	5	3	4

Hence, $(B9F.AE)_{16} = (5637.534)_8$

13. Any radix r number to Decimal Conversion

- We can convert a given number in radix r to decimal by multiplying each digit by its positional weights and taking sum of all the products.

Ex 18: Convert 1221_3 to decimal.

Solution:

Here, the given number is in base 3. Its positional weights are: $3^3\ 3^2\ 3^1\ 3^0$

$1\ 2\ 2\ 1 = (1 \times 3^3) + (2 \times 3^2) + (2 \times 3^1) + (1 \times 3^0)$

$= 27 + 18 + 6 + 1$

= 52

Hence, $(1221)_3 = (52)_{10}$

Ex 19: Convert 234.02_5 to decimal.

Solution:

Here, the given number is in base 5. Its positional weights are: $5^2\ 5^1\ 5^0\ .\ 5^{-1}\ 5^{-2}$

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

$= 50 + 15 + 4 + 0 + 0.08$

$= 69.08$

Hence, $(234.02)_5 = (69.08)_{10}$

14. Decimal to any radix r Conversion

- Decimal number can be converted in any radix by 2 steps. In step1, the integer part of the decimal number is divided successively by the radix r and the remainders are noted down. Taking the remainders from bottom to top gives the radix r equivalent of the integer part. Similarly, the fractional part is successively multiplied by the radix r and the integer part of the result is noted down. Noting the carry from top to bottom gives the fractional part equivalent in radix r.

Ex 20: Convert 1989.35_{10} to base 12.

Solution:

Step 1: Separate the integer and fractional parts of the decimal number. Now for integer part, we carry successive division-by-12 method as follows:

12	1989	Remainder
12	165	9
12	13	9
12	1	1
	0	1

So, $(1989)_{10} = (1199)_{12}$

Step 2: Now the fraction part is 0.35_{10}. Carrying out successive multiplication-by-12 as follows:

0.35 x12 = 4.2	4
0.2 x12 = 2.4	2
0.4 x12 = 4.8	4

So, $0.35_{10} = 0.424_{12}$

Hence. $(1989.35)_{10} = (1199.424)_{12}$

15. Find the value of unknown base

Ex 21: Determine b if $(33)_{10} = (201)_b$

Solution:

We have, $(33)_{10} = (201)_b$

$33 = 2 \times b^2 + 0 \times b^1 + 1 \times b^0$

$= 2b^2 + 1$

$2b^2 = 32$

$b^2 = 16$

$b = \pm 4$

But base of any number cannot be negative. Hence value of b = 4.

Ex 22: Determine b if $(193)_b = (623)_8$

Solution:

We have, $(193)_b = (623)_8$

$1 \times b^2 + 9 \times b^1 + 3 \times b^0 = 6 \times 8^2 + 2 \times 8^1 + 3 \times 8^0$

$b^2 + 9b + 3 = 384 + 16 + 3$

$b^2 + 9b + 3 = 403$

$b^2 + 9b - 400 = 0$

$b = 16, b = -25$

As base is always positive, value of b=16.

16. Binary Coded Decimal (BCD)

- It is a numeric code that is used to represent decimal using binary bits i.e. 1's and 0's. It is different from representation of a decimal number in binary system i.e. base 2 system.
- In BCD representation each digit of a decimal number is represented by a group of four bits. These bits are given with weights of 8-4-2-1 and hence many a times BCD code is also called 8421 code. Code for each digit of decimal is as follows.

Table 1.7: Conversions between Decimal to BCD code

Decimal Digit	BCD code 8421
0	0000
1	0001
2	0010
3	0011

4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Ex 23: $(58)_{10}$ = ($)_{BCD}$

Solution:

Decimal: 5 8

↓ ↓

0101 1000

Thus $(58)_{10} = (01011000)_{BCD}$.

- It should be noted that binary representation of $(58)_{10}$ will be $(111010)_2$ which is quite different from the BCD representation.

Ex 23: $(001001011001)_{BCD}$ = $()_{10}$

Solution:

BCD: 0010 0101 1001

↓ ↓ ↓

2 5 9

$(001001011001)_{BCD} = (259)_{10}$

- It can be observed that BCD codes are less efficient for representation compared to binary as it requires more number of bits then required in binary representation. However, it is popular because of its ease of conversion to and from decimal.

1.4 LOGIC GATES:

- Logic gates are the fundamental building blocks of digital systems. They are the physical devices that performs the basic Boolean operations of AND, OR and NOT.
- Input and outputs of logic gates (that is basically a voltage signal) can occur only in two levels. These two levels are termed as High and Low or True and False or ON and OFF or simply 1 and 0. In representation of higher of the two voltage levels is symbolized as 1 and lower symbolized as 0 the gate is said to be positive logic gate. However, if higher of the two voltage levels is symbolized as 0 and lower as 1 then it is said to be negative logic gate.
- Input output behavior of a gate is generally represented using truth table. It is a table that lists output for all possible combinations of inputs.

- There are total seven logic gates in which three are **basic logic gates** (AND, OR, NOT) and two are **universal logic gates** (NAND, NOR).

➢ Various basic gates are discussed as follows;

1. ***NOT Gate:***

- NOT gate has one inputs and one output. The output becomes logic 1 when input is at logic 0 and output becomes logic 0 when the input is at logic 1. Thus it inverts or complements the logic available at input and hence called and **inverter or complement.** It is represented by a bar over the variable " ¯ "or with a symbol " ' ". Thus, for example, X = A' or X = $\overline{A}$ read as "X is equal to Not A or A bar or A complement". NOT gate and its truth table are shown in fig. 1.10.

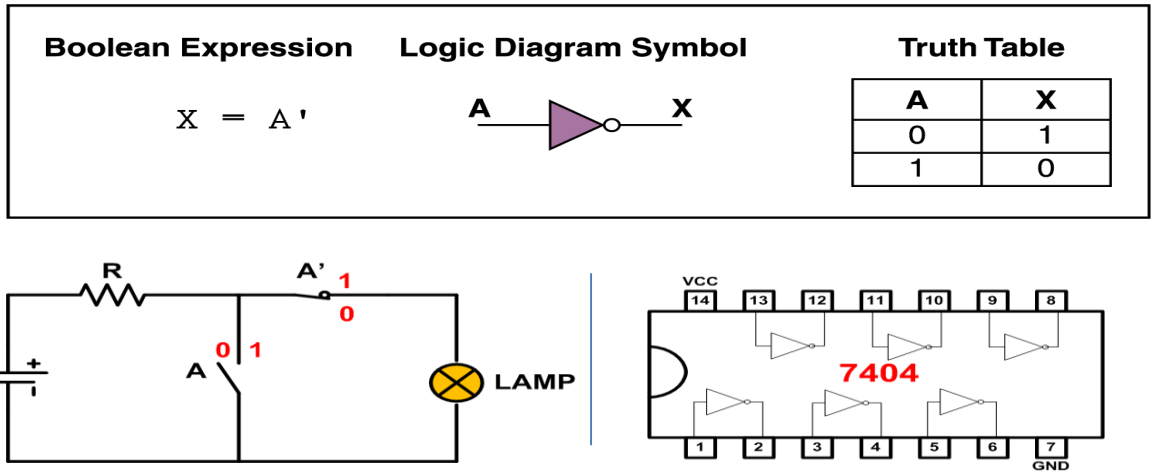
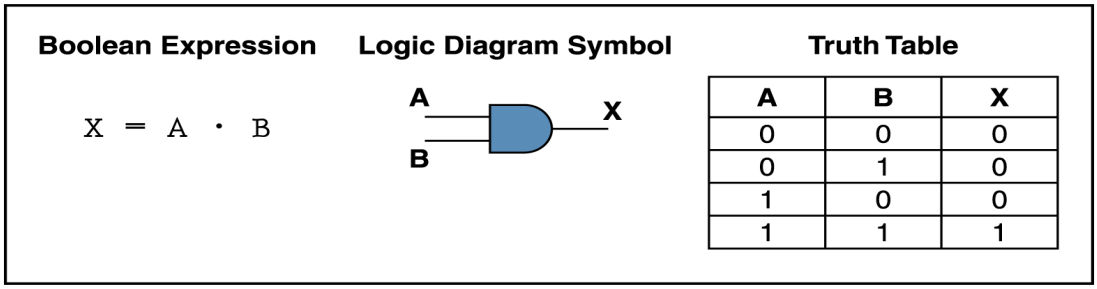


Fig. 1.10: Illustration of NOT gate

2. ***AND Gate:***

- AND gate means all or nothing logic
- AND gate has two or more inputs and one output. The output becomes logic 1 only when each one of its input is at logic 1. For all other input combinations it gives output logic 0. It is represented by a symbol •. Thus, for example, X = A · B (also written simply as X = AB) is read as "X is equal to A AND B". Two input AND gate and its truth table is shown in fig. 1.11.



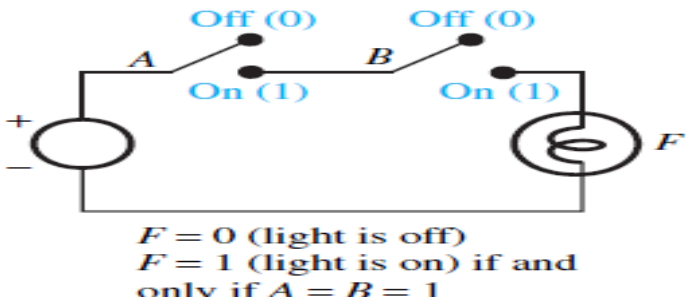


Fig. 1.11: Illustration of AND gate

- Operation of AND gate can be understood through the example of two switches connected in series as shown in fig. 1.11. Here we assume switches A and B to present logic 1 when in ON condition and logic 0 in OFF condition. Similarly, if lamp is ON we assume logic 1 and in OFF condition we assume it as logic 0. Then it can be determined that the lamp will be ON (at logic 1) only when both the switches A and B are ON (at logic 1).

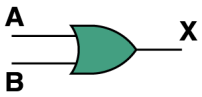
3. ***OR Gate:***

- OR gate means any or all logic
- OR gate has two or more inputs and one output. The output becomes logic 1 when at least (minimum) one of the inputs is at logic 1. It is represented by a symbol **+**. Thus, for example, X = A + B is read as "X is equal to A OR B". Two input AND gate and its truth table is shown in fig. 1.12.

Boolean Expression

$X = A + B$

Logic Diagram Symbol



Truth Table

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

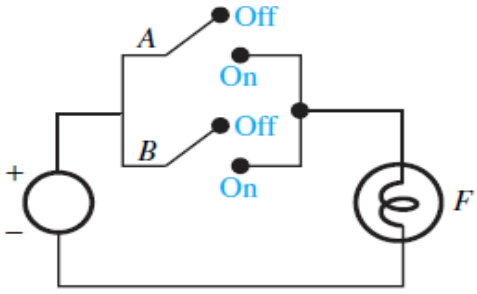
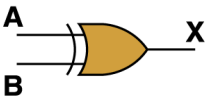


Fig. 1.12: Illustration of AND gate

4. *Exclusive OR Gate (EX-OR):*

- It also means Inequality detector because it gives output high when both inputs are different.
- Exclusive OR gate give output equal to 1 when the two inputs are exclusively different. This is the reason why it is also known as inequality gate. The schematic symbol and truth table of the gate is shown in fig. 1.13. It is represented by a symbol $\oplus$. Thus, for example, $X = A \oplus B$ is read as "X is equal to A XOR B." The logic expression this gate in terms of AND, OR and NOT operation is $X = A \oplus B = \overline{A}B + A\overline{B}$.

Boolean Expression	Logic Diagram Symbol	Truth Table
$X = A \oplus B$		

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

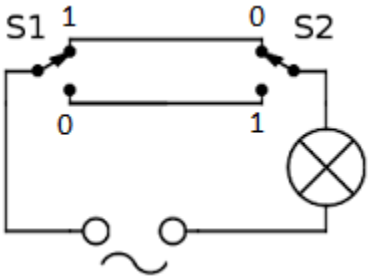
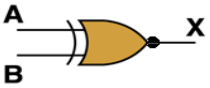


Fig. 1.13: Illustration of EX-OR gate

5. *Exclusive NOR Gate (EX-NOR):*

- It also means equality detector because it gives output high when both inputs are same.
- Exclusive NOR gate is XOR gate followed by inverter. Thus it is complement of XOR gate. This is the reason why it is also known as equality gate. The logic symbol, logic expression, schematic symbol, truth table of the gate is shown in fig. 1.14.
- **X = AB + A'B'**

Boolean Expression	Logic Diagram Symbol	Truth Table
$X = A \odot B$		

A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

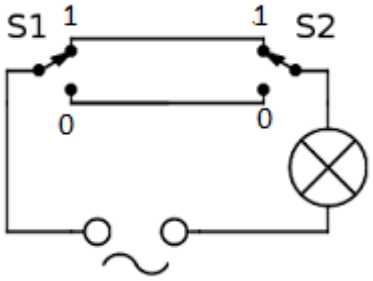


Fig. 1.14: Illustration of EX-NOR gate

➢ ***Universal Gates:***

- NAND and NOR gates are known as a universal gates because from this two gates all other gates can be constructed.

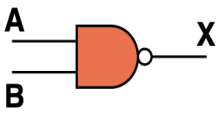
6. NAND Gate:

- NAND gate represents combination of AND gate followed by NOT gate. It represents complement of AND operation. Schematic symbol of NAND gate and its truth table are shown in fig. 1.15. The logic expression is given as $X = \overline{(A \cdot B)}$ or X = (A·B)'.

Boolean Expression

$X = (A \cdot B)'$

Logic Diagram Symbol



Truth Table

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

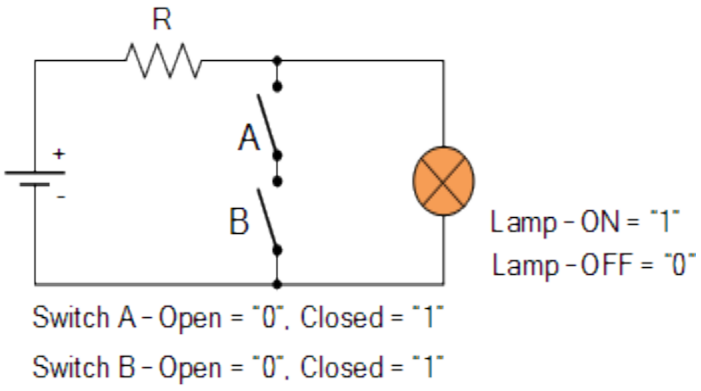
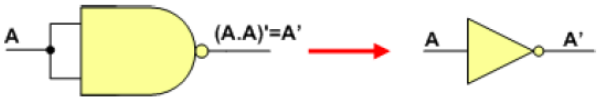


Fig. 1.15: Illustration of NAND gate

➢ **NAND gate as Universal gate**

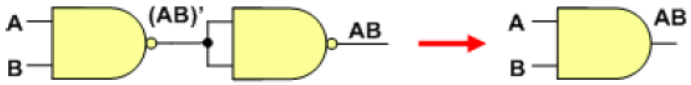
1. Implementing NOT gate

- All NAND input pins connect to the input signal A gives an output A'.



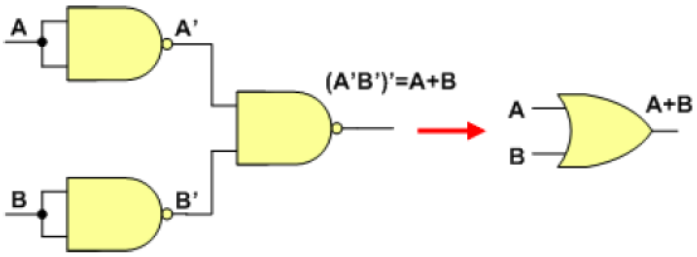
2. Implementing AND gate

- The AND is replaced by a NAND gate with its output complemented by a NAND gate inverter.



3. Implementing OR gate

- The OR gate is replaced by a NAND gate with all its inputs complemented by NAND gate inverters.

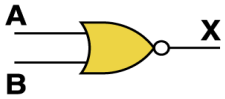


7. NOR Gate:

- NOR gate represents combination of OR gate followed by NOT gate. It represents complement of OR operation. Schematic symbol of NOR gate and its truth table are shown in fig. 4.16. The logic expression is given as $X = \overline{(A+B)}$ or *X = (A+B)'*.

Boolean Expression	Logic Diagram Symbol	Truth Table

X = (A + B)'



A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

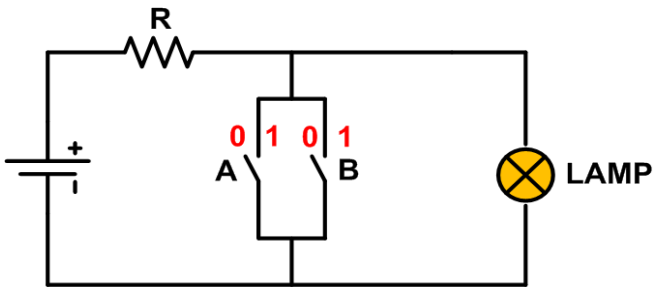
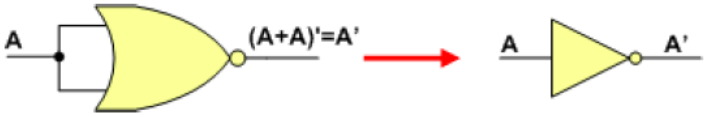


Fig. 1.16: Illustration of NOR gate

➢ **NOR gate as Universal gate.**

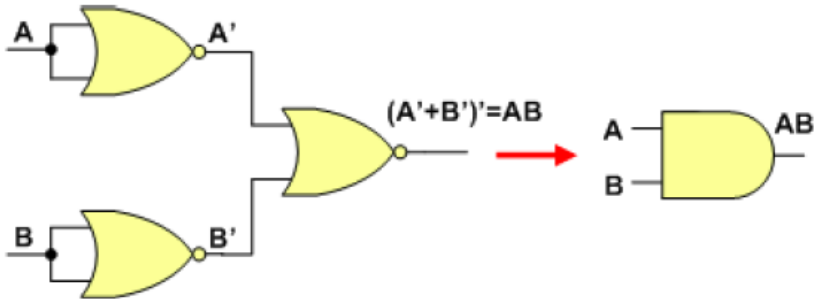
1. Implementing NOT gate

- All NOR input pins connect to the input signal A gives an output A`.



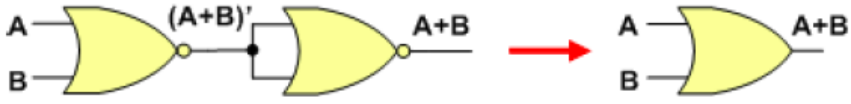
2. Implementing AND gate

- The AND gate is replaced by a NOR gate with all its inputs complemented by NOR gate inverters.



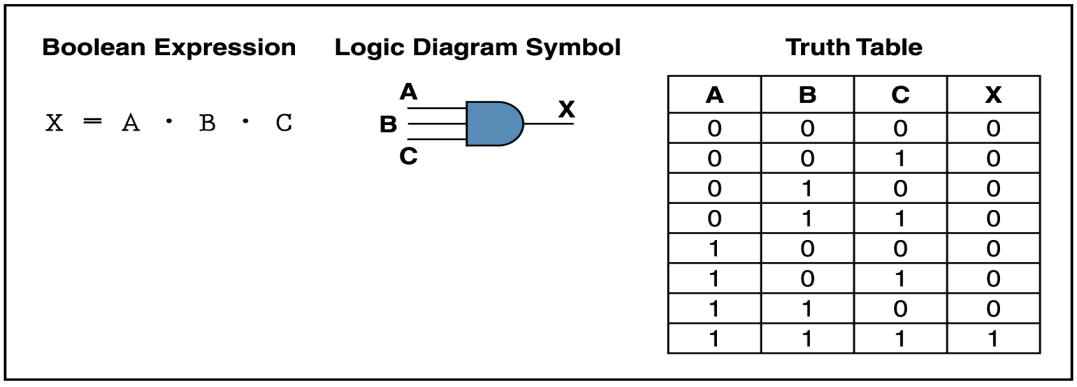
3. Implementing OR gate

- The OR is replaced by a NOR gate with its output complemented by a NOR gate inverter.



➢ **Gates with More Inputs**

- Any logic gates can be designed to accept three or more input values.
- As an example, three-input AND gate produces an output of 1 only if all input values are 1.



A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Fig. 1.17: Illustration of gates with more inputs

➢ Constructing Gates

- A transistor is a device that acts, depending on the voltage level of an input signal, either as a wire that conducts electricity or as a resistor that blocks the flow of electricity
- A transistor has no moving parts, yet acts like a switch.
- It is made of a semiconductor material, which is neither a particularly good conductor of electricity, such as copper, nor a particularly good insulator, such as rubber.
- A transistor is shown in fig. 1.18.

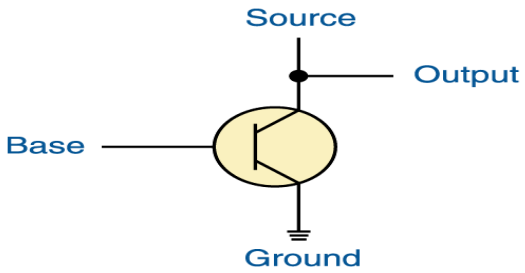


Fig. 1.18: Transistor

- A transistor has three terminals
  1. A source
  2. A base
  3. An emitter, typically connected to a ground wire
- If the electrical signal is grounded, it is allowed to flow through an alternative route to the ground (literally) where it can do no harm.
- It turns out that, because the way a transistor works, the easiest gates to create are the NOT, NAND, and NOR gates shown in fig. 1.19.

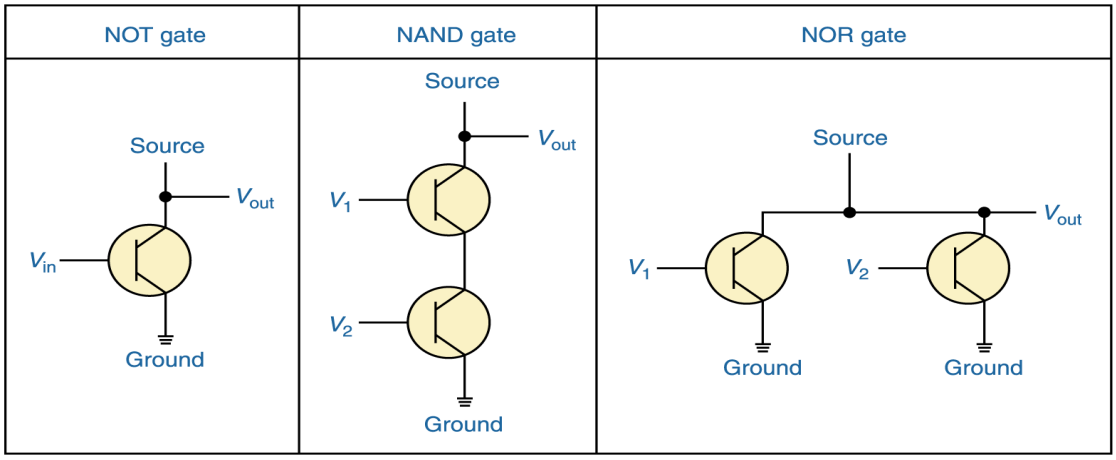


Fig. 1.19: Logic gates using transistor

1.5 Introduction to Logic Family

- It is a circuit configuration or approach used to produce a type of digital integrated circuit. The entire range of digital ICs is fabricated using either bipolar devices or MOS devices or a combination of the two.

- Different logic families falling in the first category are called bipolar families, and these include
  - Diode Logic (DL)
  - Resistor Transfer Logic (RTL)
  - Diode Transistor Logic (DTL)
  - Transistor Transistor Logic (TTL)
  - Emitter Coupled Logic (ECL)
  - Integrated Injection Logic (I^2L)

- The logic families that use MOS devices as their basis are known as MOS families, and the prominent members belonging to this category are
  - PMOS Family (using P-channel MOSFETs)
  - NMOS Family (using N-channel MOSFETs)
  - CMOS Family (using both N- and P-channel devices)
  - Bi-MOS Logic Family uses both Bipolar and MOS devices

- Of all the logic families listed above, the first three, i.e. DL, RTL and DTL have become obsolete
- Diode Logic used diode & resistors and in fact was never implemented in integrated circuits
- RTL Family used resistors and bipolar transistors
- DTL Family used resistors, diodes and bipolar transistors.
- PMOS and I^2L logic families, which were mainly intended for use in custom large-scale integrated (LSI) circuit devices, have also become more or less obsolete.
- Logic families that are still in widespread used include TTL, CMOS, ECL, NMOS and Bi-CMOS.

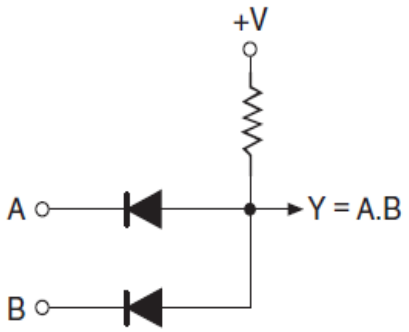


Fig. 1.20: Diode Logic for AND function

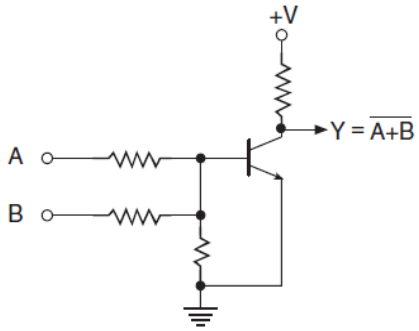


Fig. 1.21: Resistor Transfer Logic for NOR function

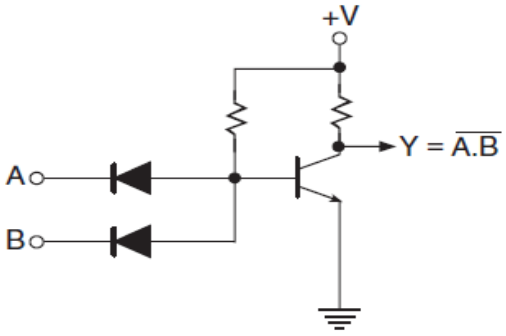


Fig. 1.22: Diode Transistor Logic for NAND function

- All circuits of a given category must have compatible operating characteristics.
- The high level voltage developed at the output of a gate must be sufficient to drive the input of any other gate to the same high level.
- The low level output must pull the input of the next stage down to an acceptable low level
- Each category is compatible to some degree with other categories of the same family.
- For perfect compatibility, a single a category is used for a given design
- In general, an entire digital system, such as computer, will use only one or two logic circuit families.
- Sometimes it is necessary to connect logic elements that are not of the same family.

- When this is done, an interface between the different elements may be required.
- An interface consists of circuits that translate the output signals from one family to the input signals required by the other family.
- A family is said to be compatible with another family when both families can be interconnected without requiring interface circuits.

1.5.1 Current & Voltage Definitions

1. V_{IHmin} = minimum input voltage that the logic element is guaranteed to interpret as high logic level
2. V_{ILmax} = maximum input voltage that the logic element is guaranteed to interpret as the low logic level
3. V_{OHmin} = minimum high logic level voltage appearing at the output terminal of the logic element
4. V_{OLmax} = maximum low logic level voltage appearing at the output terminal of the logic element
5. I_{IHmax} = maximum current that will flow into an input when a specified high logic level voltage is applied
6. I_{ILmax} = maximum current that will flow into an input when a specified low logic voltage is applied
7. I_{OH} = current flowing into the output when a specified high level output voltage is present
8. I_{OL} = current flowing into the output when a specified low level output voltage is present
9. I_{OS} = current flowing into an output when the output is shorted and input conditions are such to establish a high logic level output

1.5.2 Fan-Out

- In a digital system, a given gate may drive the inputs to several other gates.
- The no. of inputs that can be driven by the gate is referred to as ***fan-out*** of the circuit
- Most of the circuits of a family will require the same input current, but a few may require more.
- If so, the specs for such a circuit will indicate that the input is equivalent to some multiple of standard loads.
- For e.g., a circuit may present an equivalent input of two standard loads.
- If fan-out of a gate is specified as 10, only five of these circuits could be safely driven.
- Fig. 1.23 shows an AND gate loaded with 4 inputs, assuming each circuit presents one standard load to the AND gate output.

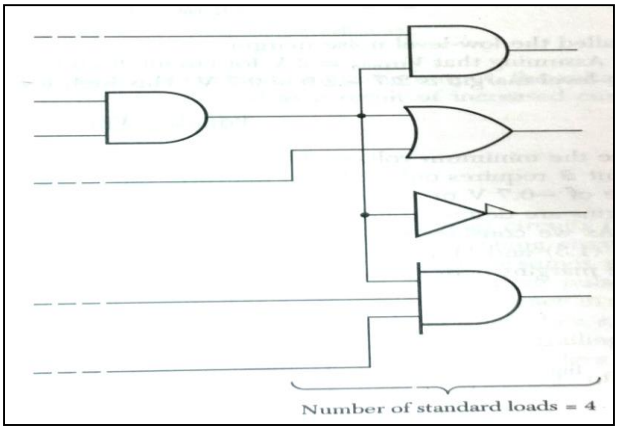


Fig. 1.23: Practical Schematic demonstrating loading

- From the current requirements, fan-out can be calculated.
- For e.g., one TTL gate has following current specs :
    - I_{IH} = 40µA, I_{IL} = -1.6 mA, I_{OH} = -400 µA, I_{OL} = 16 mA
- If this gate is to drive several other similar gates, then the output current capability of the stage is 10 times that required by the input.
- The output stage can drive 400µA into the following stages at the high level and sink 16mA at the low level. The fan-out of this gate is 10.

1.5.3 Noise Margin

- Noise margin specifies the maximum amplitude noise pulse that will not change the state of the driven stage.
- Noise margin can be evaluated from a consideration of the voltage levels V_{IHmin}, V_{ILmax}, V_{OHmin} and V_{OLmax}.
- Fig. 1.24 shows two logic circuits that are cascaded

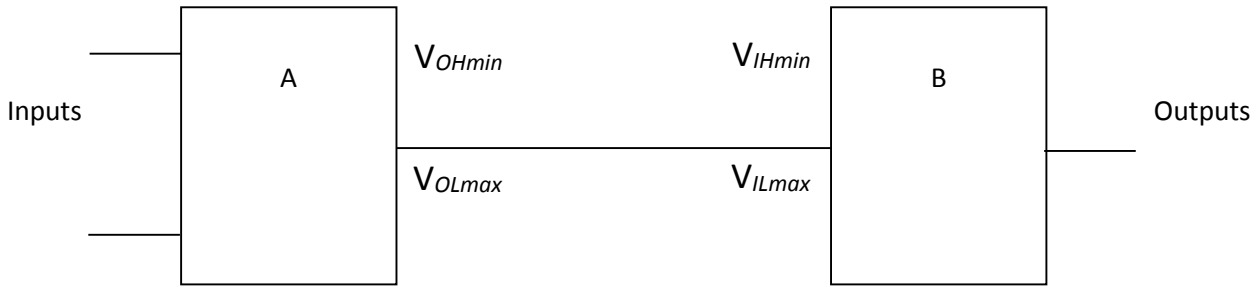


Fig. 1.24: Cascaded Stages used to calculate Noise Margin

- If V_{ILmax} = 0.8 V for circuit B, this means that the input must be less than 0.8 V to guarantee that circuit B interprets this value as a low level
- If circuit A has a value of V_{OLmax} = 0.4 V, a noise spike of less than the difference 0.8 - 0.4 = 0.4 V cannot lead to level misinterpretation by circuit b. The difference

  $V_{ILmax} – V_{OLmax}$
  is called **Low Level Noise Margin**

- Assuming that V_{IHmin} = 2 V for circuit B and V_{OHmin} = 2.7 V for circuit A, the high level margin is 2.7 – 2.0 = 0.7 V. The difference

  $V_{OHmin} – V_{IHmin}$
  is called **High Level Noise Margin**

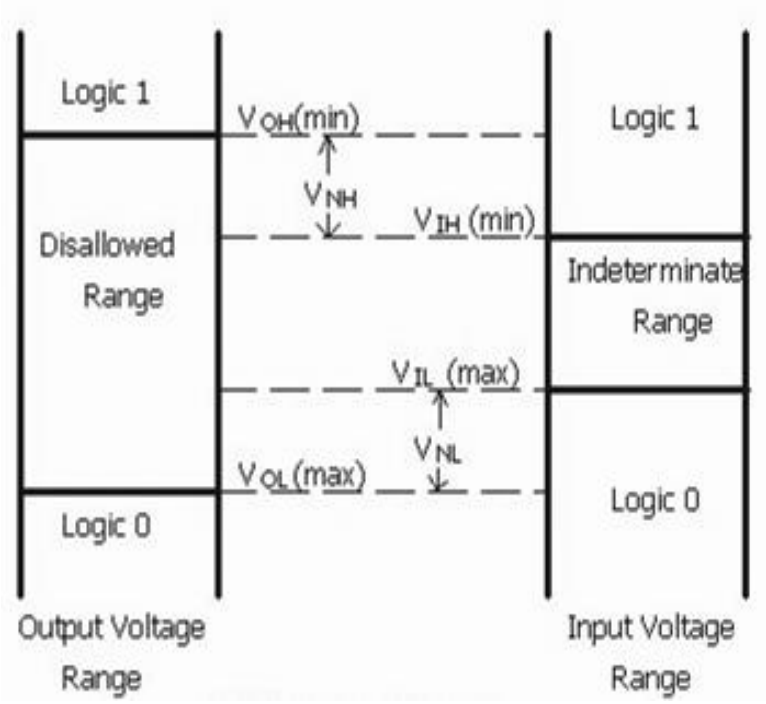


Fig. 1.25: Noise Margin Definition

- Since the minimum voltage developed by circuit A at the high level is 2.7 V, while circuit B requires only 2.0 V to interpret the signal as high level, a negative noise spike of -0.7 V or less will not result in an error.
- Both low- and high-level noise margins are demonstrated in fig. 1.25.
- As more gate inputs are connected to a given output, the voltages generated at both high and low levels are affected as a result of increased current flow. Thus, fan-out is influenced by noise margin.

1.5.4 Switching Times

- Another quantity that is used to characterize switching circuits is the speed with which the device responds to the input changes.
- For switching circuits, the graph of Fig. 1.26 shows different delay times.
- Different delay times are of an inverter gate.
- Different definitions are as follows:
  - $\mathbf{t_{pHL}}$: Difference in time between the point where e_{in} rises to 50% of its final value and the time when e_{out} falls to its 50% point
  - $\mathbf{t_{pLH}}$: Time difference between 50% points of the trailing edges of the input and output signals
  - **Propagation Delay** : The average of t_{pHL} and t_{pLH}, or

$$t_{pd} = \frac{t_{pHL} + t_{pLH}}{2}$$

- **Fall Time, t_f** : 90% to 10% values as the output voltage swing between upper and lower voltage level
- **Rise Time, t_r** : 10% to 90% values as the output voltage swing between lower and upper voltage level

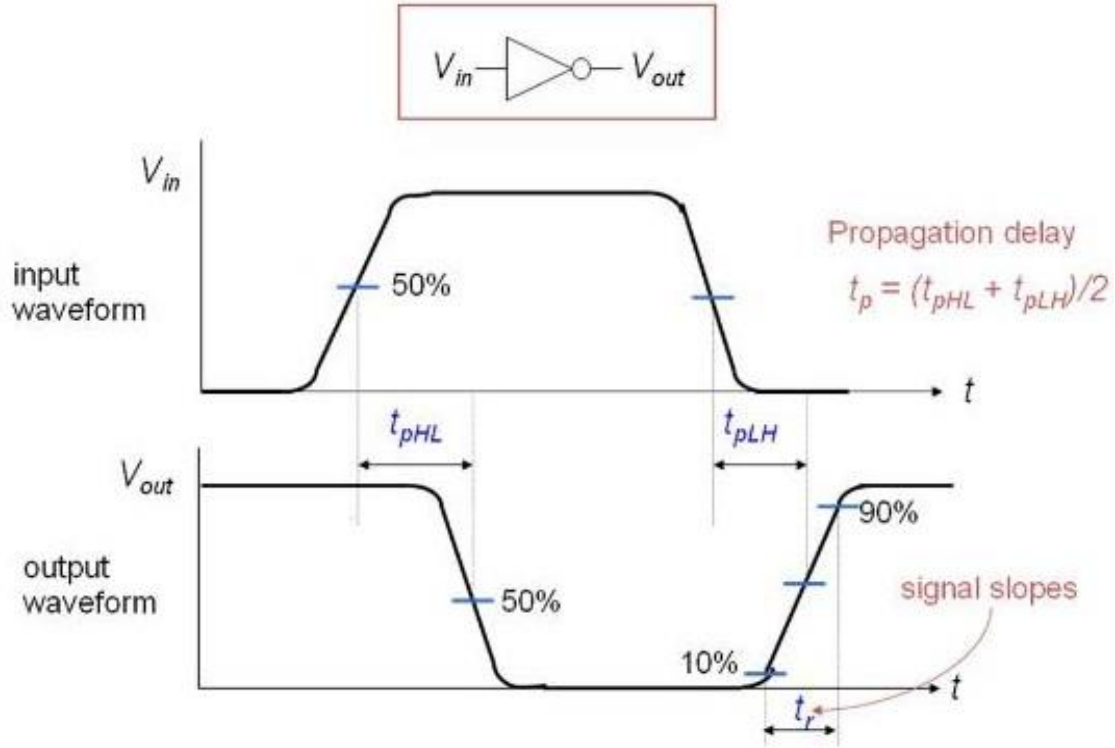


Fig. 1.26: Representation of switching time

2.1 BOOLEAN ALGEBRA:

- Inventor of Boolean algebra was George Boole (1815 - 1864).

- Designing of any digital system there are three main objectives;
  1) Build a system which operates within given specifications
  2) Build a reliable system
  3) Minimize resources

- Boolean algebra is a system of mathematical logic.
- Any complex logic can be expressed by Boolean function.
- Boolean algebra is governed by certain rules and laws.
- Boolean algebra is different from ordinary algebra & binary number system.
  In ordinary algebra; A + A = 2A and AA = A^2, here A is numeric value.
- In Boolean algebra;
  A + A = A and AA = A, here A has logical significance, but no numeric significance.

Table 2.1: Difference between Binary, Ordinary and Boolean system

Binary no. system	Ordinary no. system	Boolean algebra
1 + 1 = 10	1 + 1 = 2	1 + 1 = 1

- In Boolean algebra there is nothing like subtracting or division, no negative or fractional numbers.
- Boolean algebra represent logical operation only. Logical multiplication is same as AND operation and logical addition is same as OR operation.
- Boolean algebra has only two values 0 & 1.
- In Boolean algebra;
  If A = 0 then A ≠ 1. & If A = 1 then A ≠ 0.

- Let's introduce Boolean algebra by considering a practical problem:
- Suppose a system which transmitting 2 bit binary information over 2 line to another system.
- So with 2 line, 4 unique code could be represented.
- The receiving system may need to identify the presence of certain transmitted codes.
- As an example suppose system is to identify the occurrence of the code representing the decimal number 1 and 2. Each one of these code appear on the 2 lines, a circuit is to generate an output 1.
- When any other code present the output should be 0.

Table 2.2: 2 bit binary to decimal conversion

Binary	Decimal
00	0
01	1
10	2
11	3

- The method of implementing the system is shown in following fig.
- We want output 1 if decimal 1 & 2 otherwise 0 as defined above.
- Below circuit generate output 1 when A = 0 & B = 1 or A = 1 & B = 0.

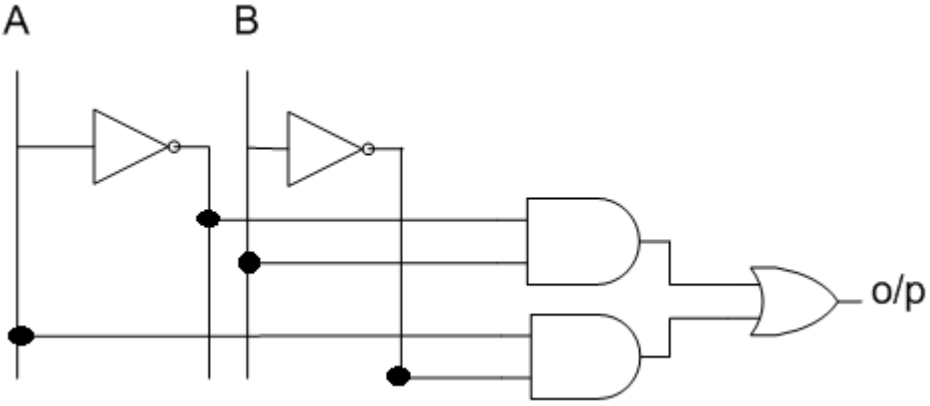


Fig. 2.1: A logic system

2.1.1 Advantages of Boolean Algebra

1. Minimize the no. of gates used in circuit.
2. Decrease the cost of circuit.
3. Minimize the resources.
4. Less fabrication area is required to design a circuit.
5. Minimize the designer's time.
6. Reducing to a simple form. Simpler the expression more simple will be hardware.
7. Reduce the complexity.

2.1.2 Axioms of Boolean Algebra

- Axioms or postulate of Boolean algebra are a set of logical expression that we accept without proof & upon which we can build a set of useful theorems.

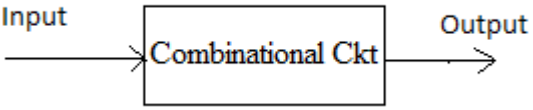
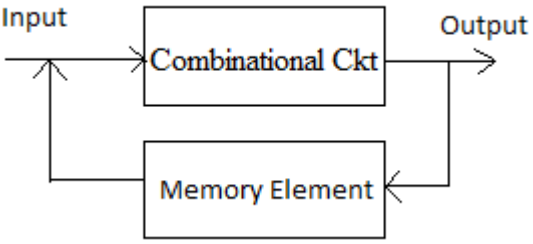
Axioms 1: $0 \cdot 0 = 0$
Axioms 2: $0 \cdot 1 = 0$
Axioms 3: $1 \cdot 0 = 0$
Axioms 4: $1 \cdot 1 = 1$
Axioms 5: $0 + 0 = 0$
Axioms 6: $0 + 1 = 1$
Axioms 7: $1 + 0 = 1$
Axioms 8: $1 \cdot 1 = 1$
Axioms 9: $1' = 0$
Axioms 10: $0' = 1$

2.1.3 Types of logic circuit

- There are two types of logic circuit;
  1) Sequential circuit
  2) Combinational circuit

2.1.3.1 Difference between combinational & sequential circuit

Table 2.3: Difference between combinational and sequential circuits

Combinational Circuit	Sequential Circuit
Output is dependent only on the input at the same instant of time	Output is dependent on present input and past output.
It does not contains memory elements. 	It contains memory element. 
Its behavior is described by the set of output function.	Its behavior is described by the set of next state function and the set of output function.
No feedback is available.	Feedback is available.
It does not contains periodic clock signal.	It contains clock signals.
Faster than sequential circuit.	Slower than combinational circuit.
e.g. half adder, full adder, etc.	e.g. Flip flop, counter, etc.

2.2 LAWS OF BOOLEAN ALGEBRA:

1. Complementation Laws:

- The term complement simply means to invert, i.e. to change 0's to 1's and 1's to 0's.
  Law 1: 0' = 1
  Law 2: 1' = 0
  Law 3: If A = 0 then A' = 1
  Law 4: If A = 1 then A' = 0
  Law 5: A'' = A

2. *AND Laws:*

Law 1: A · 0 = 0
Law 2: A · 1 = A
Law 3: A · A = A
Law 4: A · A' = 0

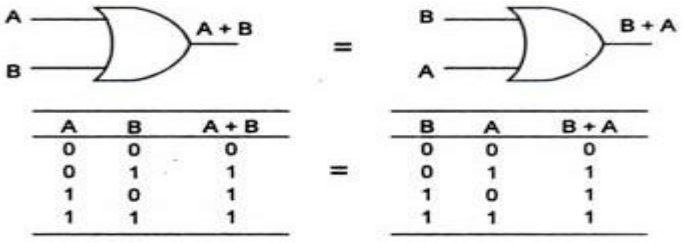
3. *OR Laws:*

Law 1: A + 0 = A
Law 2: A + 1 = 1
Law 3: A + A = A
Law 4: A + A' = 1

4. *Commutative Laws:*

- Commutative laws allow change in position of AND or OR variables.

Law 1: A + B = B + A
Proof:



A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

=

B	A	B + A
0	0	0
0	1	1
1	0	1
1	1	1

- This law can be extended to any numbers of variables for e.g.

A + B + C = B + A + C = C + B + A = C + A + B

Law 2: A · B = B · A
Proof:

A	B	A · B
0	0	0
0	1	0
1	0	0
1	1	1

=

B	A	B · A
0	0	0
0	1	0
1	0	0
1	1	1

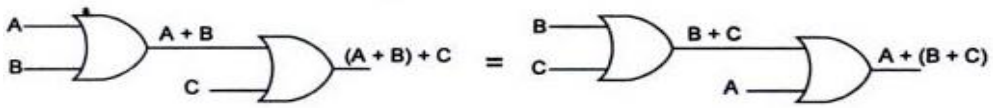
- This law can be extended to any numbers of variables for e.g.

A · B · C = B · A · C = C · B · A = C · A · B

5. Associative Laws:

- The associative laws allow grouping of variables.

Law 1: (A + B) + C = A + (B + C)
Proof:



A	B	C	A + B	(A + B) + C
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

=

A	B	C	B + C	A + (B + C)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

- This law can be extended to any no. of variables for e.g.

$$A + (B + C + D) = (A + B + C) + D = (A + B) + (C + D)$$

Law 2: (A · B) · C = A · (B · C)
Proof:

A	B	C	AB	(AB)C
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

=

A	B	C	BC	A(BC)
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

- This law can be extended to any no. of variables for e.g.

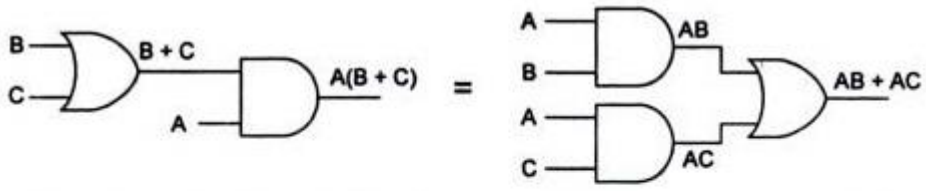
$$A \cdot (B \cdot C \cdot D) = (A \cdot B \cdot C) \cdot D = (A \cdot B) \cdot (C \cdot D)$$

6. Distributive Laws:

- The distributive laws allow factoring or multiplying out of expressions.

Law 1: A (B + C) = AB + AC

Proof:



A	B	C	B + C	A(B + C)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

=

A	B	C	AB	AC	AB + AC
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

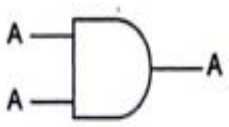
Law 2: A + BC = (A + B) (A + C)
Proof: R.H.S. = (A + B) (A + C)
= AA + AC + BA + BC
= A + AC + BA + BC
= A + BC (B'cz 1 + C + B = 1 + B = 1)
= L.H.S.

Law 3: A + A'B = A + B
Proof: L.H.S. = A + A'B
= (A + A') (A + B)
= A + B
= R.H.S.

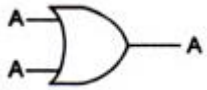
7. Idempotence Laws:

- Idempotence means the same value.

Law 1: A · A = A
Proof:
Case 1: If A = 0 → A · A = 0 · 0 = 0 = A
Case 2: If A = 1 → A · A = 1 · 1 = 1 = A

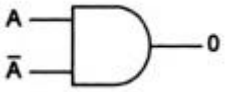


Law 2: A + A = A
Proof:
Case 1: If A = 0 → A + A = 0 + 0 = 0 = A
Case 2: If A = 1 → A + A = 1 + 1 = 1 = A

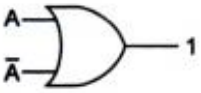


8. *Complementation Law / Negation Law:*

Law 1: A · A' = 0
Proof:
Case 1: If A = 0 → A' = 1 So, A · A' = 0 · 1 = 0
Case 2: If A = 1 → A' = 0 So, A · A' = 1 · 0 = 0



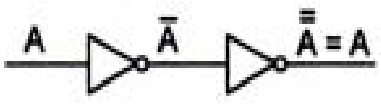
Law 2: A + A' = 1
Proof:
Case 1: If A = 0 → A' = 1 So, A + A' = 0 + 1 = 1
Case 2: If A = 1 → A' = 0 So, A + A' = 1 + 0 = 1



9. *Double Negation Law:*

- This law states that double negation of a variables is equal to the variable itself.

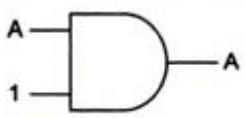
Law 1: A'' = A
Proof:
Case 1: If A = 0 → A'' = 0'' = 1' = A
Case 2: If A = 1 → A'' = 1'' = 0' = A



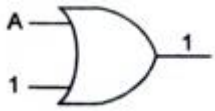
- Any odd no. of inversion is equivalent to single inversion.
- Any even no. of inversion is equivalent to no inversion at all.

10. Identity Law:

Law 1: A · 1 = A
Proof:
Case 1: If A= 1 → A · 1 = 1 · 1 = 1 = A
Case 2: If A= 0 → A · 0 = 0 · 0 = 0 = A

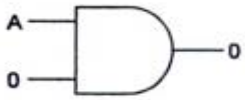


Law 2: A + 1 = 1
Proof:
Case 1: If A= 1 → A + 1 = 1 + 1 = 1 = A
Case 2: If A= 0 → A + 0 = 0 + 0 = 0 = A

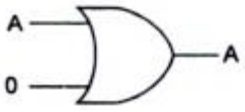


11. Null Law:

Law 1: A · 0 = 0
Proof:
Case 1: If A= 1 → A · 0 = 1 · 0 = 0 = 0
Case 2: If A= 0 → A · 0 = 0 · 0 = 0 = 0



Law 2: A + 0 = A
Proof:
Case 1: If A= 1 → A + 0 = 1 + 0 = 1 = A
Case 2: If A= 0 → A + 0 = 0 + 0 = 0 = A



12. Absorption Law:

Law 1: A + AB = A

Proof: L.H.S. = A + AB
= A (1 + B)
= A (1)
= A
R.H.S.

Law 2: A (A + B) = A
Proof: L.H.S. = A (A + B)
= A · A + AB
= A + AB
= A (1 + B)
= A (1)
= A
= L.H.S.

13. Consensus Theorem:

Theorem 1: A · B + A'C + BC = AB + A'C
Proof: L.H.S. = AB + A'C + BC
= AB + A'C + BC (A +A')
= AB + A'C + BCA + BCA'
= AB (1 + C) + A'C (1 + B)
= AB + A'C
= R.H.S.

- This theorem can be extended as,

AB + A'C + BCD = AB + A'C

Theorem 2: (A + B) (A' + C) (B + C) = (A + B) (A' + C)
Proof: L.H.S. = (A + B) (A' + C) (B + C)
= (AA' + AC + A'B + BC) (B + C)
= (0 + AC + A'B + BC) (B + C)
= ACB +ACC +A'BB + A'BC + BCB + BCC
= ABC + AC + A'B + A'BC + BC + BC
= ABC + AC + A'B + A'BC + BC
= AC (1 + B) + A'B (1 + C) + BC
= AC + A'B + BC ……(1)

R.H.S. = (A + B) (A' + C)
= AA' + AC + BA' + BC
= 0 + AC + BA' + BC
= AC + A'B + BC ……(2)

Equation (1) = Equation (2)
So. L.H.S = R.H.S.

- This theorem can be extended to any no. of variables.

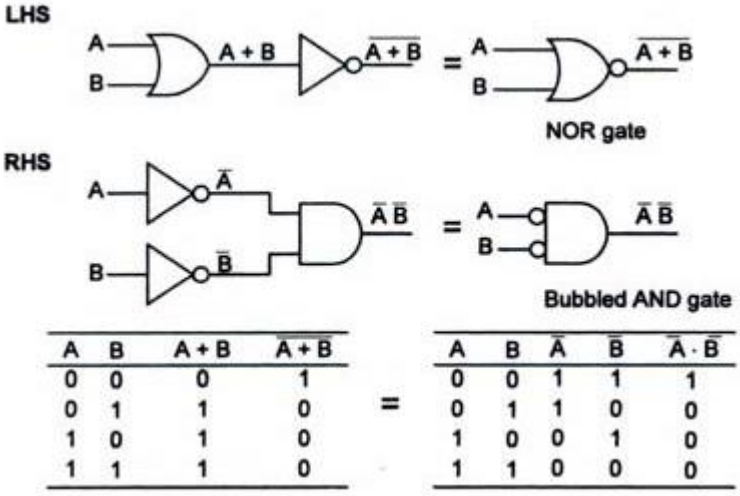
$$(A + B)(A' + C)(B + C + D) = (A + B)(A' + C)$$

14. Transposition theorem:

Theorem: $AB + A'C = (A + C)(A' + B)$
Proof: R.H.S. $= (A + C)(A' + B)$
$= AA' + AB + CA' + CB$
$= 0 + AB + CA' + CB$
$= AB + CA' + CB$
$= AB + A'C$ (B'cz of $AB + A'C + BC = AB + A'C$)
$=$L.H.S.

15. De Morgan's Theorem:

Law 1: $(A + B)' = A' \cdot B'$
Proof:



A	B	A + B	$\overline{A+B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

=

A	B	$\bar{A}$	$\bar{B}$	$\bar{A} \cdot \bar{B}$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

Law 2: $(A \cdot B)' = A' + B'$

16. Duality Theorem:

- Duality theorem arises as a result of presence of two logic system i.e. positive & negative logic system.
- This theorem helps to convert from one logic system to another.

- From changing one logic system to another following steps are taken:
  1) 0 becomes 1, 1 becomes 0.
  2) AND becomes OR, OR becomes AND.
  3) '+' becomes '·', '·' becomes '+'.
  4) Variables are not complemented in the process.

	Given Expression	*Dual*
1.	$\overline{0} = 1$	$\overline{1} = 0$
2.	$0 \cdot 1 = 0$	$1 + 0 = 1$
3.	$0 \cdot 0 = 0$	$1 + 1 = 1$
4.	$1 \cdot 1 = 1$	$0 + 0 = 0$
5.	$A \cdot 0 = 0$	$A + 1 = 1$
6.	$A \cdot 1 = A$	$A + 0 = A$
7.	$A \cdot A = A$	$A + A = A$
8.	$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$
9.	$A \cdot B = B \cdot A$	$A + B = B + A$
10.	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$
11.	$A \cdot (B + C) = AB + AC$	$A + BC = (A + B)(A + C)$
12.	$A(A + B) = A$	$A + AB = A$
13.	$A \cdot (A \cdot B) = A \cdot B$	$A + A + B = A + B$
14.	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\ \overline{B}$

2.3 REDUCTION OF BOOLEAN EXPRESSIONS:

2.3.1 Demorganize the following functions

Example 1: [(A + B') (C + D')]'

Answer: [(A + B') (C + D')]'
= (A + B')' + (C + D')'
= A' B'' + C'D''
= A'B + C'D

Example 2: [(AB)' (CD + E'F) ((AB)' + (CD)')]'

Answer: [(AB)' (CD + E'F) ((AB)' + (CD)')]'
= (AB)'' + (CD + E'F)' + ((AB)' + (CD)')'

= AB + [(CD)' (E'F)'] + [(AB)'' (CD)'']
= AB + (C' + D') (E + F') + ABCD

Example 3: [(AB)' + A' + AB]'

Answer: [(AB)' + A' + AB]'
= AB'' · A'' · AB'
= ABA (A' + B')
= AB (A' + B')
= ABA' + ABB'
= 0

Example 4: [P (Q + R)]'

Answer: [P (Q + R)]'
= P' + (Q + R)'
= P' + Q' R'

Example 5: [(P + Q') (R' + S)]'

Answer: [(P + Q') (R' + S)]'
= (P + Q')' + (R' + S)'
= P' Q'' + R'' S'
= P'Q + RS'

Example 6: [[(A + B)' (C + D)']' [(E + F)' (G + H)']']'

Answer: [[(A + B)' (C + D)']' [(E + F)' (G + H)']']'
= [(A + B)' (C + D)']'' + [(E + F)' (G + H)']''
= [(A + B)' (C + D)'] + [(E + F)' (G + H)']
= A'B'C'D' + E'F'G'H'

2.3.2 Reducing the following functions

Example 1: A [B + C' (AB + AC')']

Answer: A [B + C' (AB + AC')']

= A [B + C’ (AB)’ (AC’)’]
= A [B + C’ (A’ + B’) (A’ + C)]
= A [B + (A’ C’ + B’ C’) (A’ + C)]
= A [B + (A’ C’A’ + B’ C’A’) (A’ C’ C + B’ C’ C)]
= A [B + (A’C’ + B’ C’A’) (0 + 0)]
= A [B + A’C’ (1 + B’)]
= A [B + A’C’]
= AB + A’AC’
= AB + 0
= AB

Example 2: A + B [AC + (B + C’)D]

Answer: A + B [AC + (B + C’)D]
= A + B [AC + (BD + C’D)]
= A + ABC + BBD + BC’D
= A + ABC + BD + BC’D
= A (1 + BC) + BD (1 + C’)
= A (1) + BD (1)
= A + BD

Example 3: (A + (BC)’)’ (AB’ + ABC)

Answer: (A + (BC)’)’ (AB’ + ABC)
= (A’ (BC)’’) (AB’ + ABC)
= (A’BC) (AB’ + ABC)
= A’BCAB’ + A’BCABC
= 0 + 0
= 0

Example 4: (B + BC) (B + B’C) (B + D)

Answer: (B + BC) (B + B’C) (B + D)
= (BB + BB’C + BBC + BCB’C) (B + D)
= (B + 0 + BC + 0) (B + D)
= B (1 + C) (B + D)
= B (B + D)

= BB + BD
= B + BD
= B (1 + D)
= B

Example 5: AB + AB'C +BC'

Answer: AB + AB'C +BC'
= A (B + B'C) + BC'
= A (B + B') (B + C) + BC'
= A (1) (B + C) + BC'
= AB + AC + BC'
= CA + C'B + AB
= CA + C'B (B'cz of Consensus theorem 1)

Example 6: AB'C + B + BD' + ABD' + A'C

Answer: AB'C + B + BD' + ABD' + A'C
= AB'C + B (1 + D' + AD') + A'C
= AB'C + B + A'C
= C (A' + AB') + B
= C (A' + A) (A' + B') + B
= C (1) (A' + B') + B
= C (A' + B') + B
= A'C + CB' + B
= A'C + (C + B) (B' + B)
= A'C + (B + C) (1)
= A'C + B + C
= C (1 + A') + B
= B + C

Example 7: A'B' + A'B

Answer: A'B' + A'B = A' (B' + B) = A'

Example 8: A'B' + AB'

Answer: A'B' + AB'
= B' (A' + A)
= B'

Example 9: A'B + AB

Answer: A'B + AB
= B (A' + A)
= B (1)
= B

Example 10: A'B' + A'B + AB' + AB

Answer: A'B' + A'B + AB' + AB
= A' (B' + B) + A (B' + B)
= A' + A
= 1

Example 11: [(A + B) (A' + B)] + [(A + B) (A + B')]

Answer: [(A + B) (A' + B)] + [(A + B) (A + B')]
= [AA' + AB + BA' + BB] + [AA + AB' + BA + BB"]
= [0 + AB + A'B + B] + [A + AB' + AB + 0]
= [B (A + A' + 1)] + [A (1 + B' + B)]
= B + A
= A + B

Example 12: [(A + B') (A' + B')] + [(A' + B') (A' + B')]

Answer: [(A + B') (A' + B')] + [(A' + B') (A' + B')]
= [AA' + AB' + B'A' + B'B'] + [A'A' + A'B' + B'A' + B'B']
= [0 + AB' + A'B' + B'] + [A' + A'B' + B']
= [B' (A + A' + 1)] + [A' + B'(1)]
= B' + A' + B'
= A' + B'

Example 13: (A + B) (A + B') (A' + B) (A' + B')

Answer: (A + B) (A + B') (A' + B) (A' + B')

= (AA + AB' + BA + BB') (A'A' + A'B' + BA' + BB')

= [A (1+ B' + B)] [A' (1 + B' + B)]

= AA'

= 0

2.4 DIFFERENT FORMS OF BOOLEAN ALGEBRA:

- There are two types of boolean form
  1) Standard form
  2) Canonical form

2.4.1 Standard Forms

- In this configuration, the terms that form the function may contain one, two, or any number of literals.
- There are two types of standard forms: (i) sum of product (SOP) (ii) product of sum (POS).

➢ **Sum of Product (SOP)**

SOP is a Boolean expression containing AND terms, called product terms, of one or more literals each. The sum denote the ORing of these terms.
An example of a function expressed in sum of product is:

$$F = Y' + XY + X'YZ'$$

➢ **Product of Sum (POS)**

The OPS is a Boolean expression containing OR terms, called sum terms. Each terms may have any no. of literals. The product denotes ANDing of these terms.
An example of a function expressed in product of sum is:

$$F = X (Y' + Z) (X' + Y + Z' + W)$$

- A Boolean expression function may be expressed in a nonstandard form. For example the function:

$$F = (AB + CD) (A'B' + C'D')$$

Above function is neither sum of product nor in product sums. It can be changed to a standard form by using distributive law as below;

$$F = ABC'D' + A'B'CD$$

2.4.2 Canonical Forms

- Any boolean expression can be expressed in Sum of Product (SOP) form or Product of Sum (POS) form, they are called **canonical form**.

➢ **SOP - Sum of Product**

- A standard SOP form is one in which a no. of product terms, each one of which contains all the variables of the function either in complemented or non-complemented form, summed together.
- Each of the product term is called MINTERM.
- For minterms,
  Each non-complemented variable → 1
  Each complemented variable → 0
- Decimal equivalent is expressed in terms of lower case 'm'.

- **For example,**

1. XYZ = 111 = m_7
2. A'BC = 011 = m_3
3. P'Q'R' = 000 = m_0
4. T'S' = 00 = m_0
5. B'C = 01 = m_1

- **Example 1:** F1 = X'Y'Z + XY'Z' + XYZ
  = 001 + 100 + 111
  = $m_1 + m_4 + m_7$
  = $\Sigma m(1,4,7)$

- **Example 2:** F2 = P'Q' + PQ
  = 00 + 11
  = $m_0 + m_3$
  = $\Sigma m(0,3)$

- **Example 3:** F3 = XY'ZW + XYZ'W' + X'Y'Z'W'
  = 1011 + 1100 + 0000
  = $m_{11} + m_{12} + m_0$
  = $\Sigma m(0,11,12)$

➢ **POS - Product of Sum**

- A standard POS form is one in which a no. of sum terms, each one of which contains all the variables of the function either in complemented or non-complemented form, are multiplied together.
- Each of the product term is called MAXTERM.

- For maxterms,
  Each non-complemented variable → 0
  Each complemented variable → 1
- Decimal equivalent is expressed in terms of upper case 'M'.

- **For example,**
  1. X+Y+Z = 000 = M_0
  2. P'+Q'+R' = 111 = M_7
  3. A'+B+C'+D = 1010 = M_{10}

- **Example 1:** F1 = (P'+Q)(P+Q')
  = (10)(01) = $M_2 \cdot M_1$
  = ΠM(1,2)

- **Example 2:** F2= (X'+Y'+Z'+W)(X'+Y+Z+W')(X+Y'+Z+W')
  = (1110)(1001)(0101)
  = $M_{14} \cdot M_9 \cdot M_5$
  = ΠM(5,9,14)

- **Example 3:** F3 = (A'+B+C)(A+B'+C)(A+B+C')
  = (100) (010) (001)
  = M_4 M_2 M_1
  = ΠM(1,2,4)

➢ **Minterms & Maxterms for 3 variables**

Table 2.4: Representation of Minterms and Maxterms for 3 variables

Row No.	A B C	Minterms	Maxterms
0	0 0 0	$A'B'C' = m_0$	$A + B + C = M_0$
1	0 0 1	$A'B'C = m_1$	$A + B + C' = M_1$
2	0 1 0	$A'BC' = m_2$	$A + B' + C = M_2$
3	0 1 1	$A'BC = m_3$	$A + B' + C' = M_3$
4	1 0 0	$AB'C' = m_4$	$A' + B + C = M_4$
5	1 0 1	$AB'C = m_5$	$A' + B + C' = M_5$
6	1 1 0	$ABC' = m_6$	$A' + B' + C = M_6$
7	1 1 1	$ABC = m_7$	$A' + B' + C' = M_7$

2.4.3 Conversion between Canonical forms

- The complement of a function expressed as the sum of minterms equals the sum of minterms missing from original function
- This is because the original function is expressed by those minterms that make the function equal to 1, while its complement is 1 for those minterms that the function is 0.

- **Example 1:**

  $F(A,B,C)= \Sigma(1,4,5,6,7) \rightarrow F'(A,B,C) = \Pi M(0,2,3)$

  STEP 1:
  Take complement of the given function;
  $F'(A,B,C) = \Sigma(0,2,3) = (m_0 + m_2 + m_3)'$

  STEP 2:
  Put value of MINTERM in form of variables;
  $F'= (A'B'C' + A'BC' + A'BC)'$
  $= (A+B+C)(A+B'+C)(A+B'+C')$
  $= M_0 \cdot M_2 \cdot M_3$
  $= \Pi M(0,2,3)$
  In general, $m_j' = M_j$

- **Example 2:**

  $F(A,B,C,D)= \Pi M(0,3,7,10,14,15)$

  STEP 1:
  Take complement of the given function;
  $F'(A,B,C,D)= \Pi M(1,2,4,5,6,8,9,11,12,13) = (M_1 M_2 M_4 M_5 M_6 M_8 M_9 M_{11} M_{12} M_{13})'$

  STEP 2:
  Put value of MAXTERM in form of variables;
  $F'= [(A+B+C+D')(A+B+C'+D)(A+B'+C+D)(A+B'+C+D')(A+B'+C'+D)$
  $(A'+B+C+D)(A'+B+C+D')(A'+B+C'+D')(A'+B'+C+D)(A'+B'+C+D')]'$
  $= (A'B'C'D) + (A'B'CD') + (A'BC'D') + (A'BC'D) + (A'BCD')$
  $(AB'C'D') + (AB'C'D) + (AB'CD) + (ABC'D') + (ABC'D)$
  $= m_1 + m_2 + m_4 + m_5 + m_6 + m_8 + m_9 + m_{11} + m_{12} + m_{13}$
  $= \Sigma m(1,2,4,5,6,8,9,11,12,13)$

➢ **Convert to Minterms**

- **Example 1:**

  $F = A + B'C$

Answer: A → B & C is missing. So multiply with (B + B') & (C + C').
B'C → A is missing. So multiply with (A + A').

A = A (B + B') (C + C')
= (AB + AB') (C +C')
= ABC + AB'C + ABC' + AB'C'

B'C = B'C (A + A')
= AB'C + A'B'C

So, A + B'C = ABC + AB'C + ABC' + AB'C' + AB'C + A'B'C
= ABC + AB'C + ABC' + AB'C' + A'B'C
= 111 + 101 + 110 + 100 + 001
= $m_7 + m_6 + m_5 + m_4 + m_1$
= Σm(1,4,5,6,7)

- **Convert to Maxterms**

- **Example 1:**
F = A (B + C')

Answer: A → B & C is missing. So add BB' & CC'.
B + C' → A is missing. So add AA'.

A = A + BB' + CC'
= (A + B) (A +B') + CC
= (A + B + CC') (A + B' + CC')
= (A + B + C) (A + B + C') (A + B' + C) (A + B' + C')

B + C' = B + C' + AA' = (A + B + C') (A' + B + C')

So, A (B + C') = (A + B + C) (A + B + C') (A + B' + C) (A + B' + C')
(A + B + C') (A' + B + C')
= (A + B + C) (A + B + C') (A + B' + C) (A + B' + C')
(A' + B + C')
= (000) (001) (010) (011) (101)
= ΠM(0,1,2,3,5)

- **Example 2:**F = XY + X'Y

Answer: F = XY + X'Y
= (XY + X') (XY + Z)
= (X +X') (Y + X') (X + Z) (Y + Z)
= (Y + X') (X + Z) (Y + Z)

X' + Y = X' + Y + ZZ'
= (X' + Y + Z) (X' + Y + Z')
= (100) (101)

X + Z = X' + Z + YY'
= (X + Y + Z) (X + Y' + Z)
= (000) (010)

Y + Z = Y + Z + XX'
= (X + Y + Z) (X' + Y + Z)
= (000) (100)

So, F = XY + X'Z
= (100) (101) (000) (010)
= $M_4\ M_5\ M_0\ M_2$
F = ΠM(0,2,4,5)

2.5 KARNAUGH MAP (K-MAP):

- A Boolean expression may have many different forms.
- With the use of K-map, the complexity of reducing expression becomes easy and Boolean expression obtained is simplified.
- K-map also be said as pictorial form of truth table.

- K-map is alternative way of simplifying logic circuits.
- Instead of using Boolean algebra simplification techniques, you can transfer logic values from a Boolean statement or a truth table into a Karnaugh map (k-map)
- Tool for representing Boolean functions of up to six variables.
- K-maps are tables of rows and columns with entries represent 1's or 0's of SOP and POS representations.

- K-map cells are arranged such that adjacent cells correspond to truth rows that differ in only one bit position (*logical adjacency*)
- K-Map are often used to simplify logic problems with up to 6 variables
- ***No. of Cells = 2^n, where n is a number of variables.***

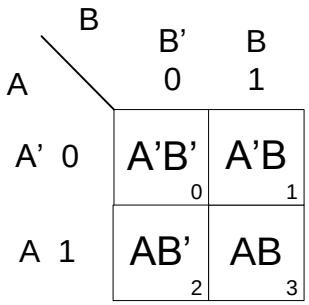
- The Karnaugh map is completed by entering a '1' (or '0') in each of the appropriate cells.
- Within the map, adjacent cells containing 1's (or 0's) are grouped together in twos, fours, or eights and so on.

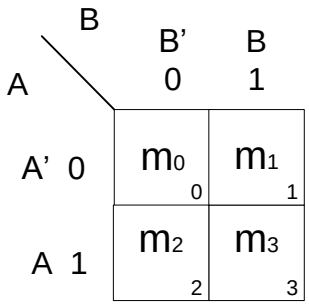
2.5.1 2 variables k-map

- For 2 variable k-map, there are $2^2 = 4$ input combinations.
- If A & B are two variables then;

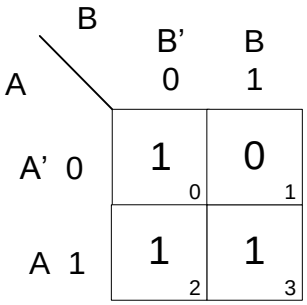
SOP → Minterms → A'B' (m_0, 00) ; A'B (m_1, 01) ; AB' (m_2, 10) ; AB (m_3, 11)
POS → Maxterms → A + B (M_0, 00) ; A + B' (M_1, 01) ; A' + B (M_2, 10) ; A' + B' (M_3, 11)

- ***Mapping of SOP Expression:***

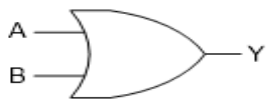




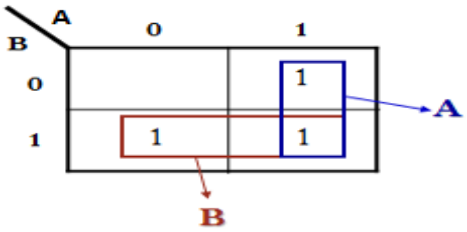
- 1 in a cell indicates that the minterm is included in Boolean expression.
- For e.g. if F = ∑m(0,2,3), then 1 is put in cell no. 0,2,3 as shown below.



Example 1: Map for a 2-input OR gate.



A	B	X
0	0	0
0	1	1
1	0	1
1	1	1



A+B

Example 2: Map for a 2-input EX-OR gate.

A	B	X
0	0	1 → $\bar{A}\bar{B}$
0	1	0
1	0	0
1	1	1 → AB

	$\bar{B}$	B
$\bar{A}$	1	0
A	0	1

F = A'B' + AB

- ***Map following SOP expressions:***

Example 1: F = AB

A \ B	B' 0	B 1
A' 0	0 (0)	0 (1)
A 1	0 (2)	1 (3)

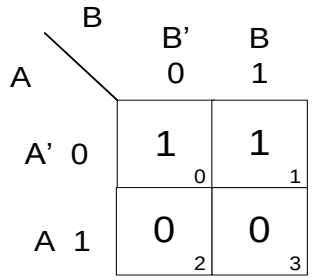
Example 2: F = AB' + A'B +A'B'

A \ B	B' 0	B 1
A' 0	1 (0)	1 (1)
A 1	1 (2)	0 (3)

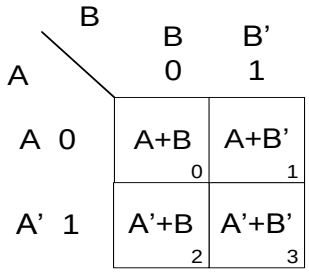
Example 3: F(A,B) = ∑(0,2)

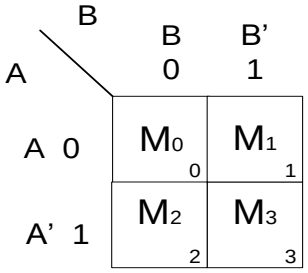
A \ B	B' 0	B 1
A' 0	1 (0)	0 (1)
A 1	1 (2)	0 (3)

Example 4: F = $m_0 + m_1$

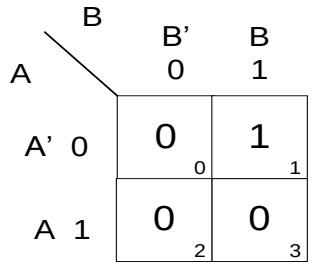


- ***Map following POS expressions:***

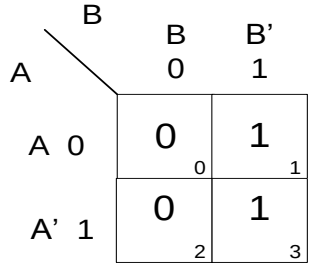




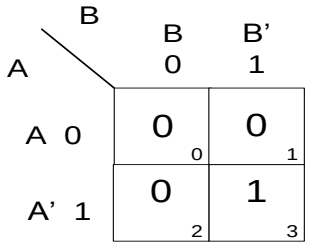
- 0 in a cell indicates that the maxterm is included in Boolean expression.
- For e.g. if F = Π M(0,2,3), then 0 is put in cell no. 0,2,3 as shown below.



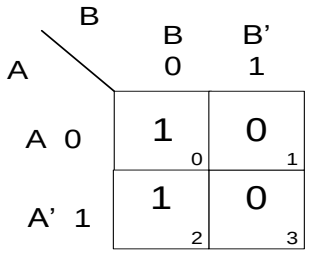
Example 1: F(A,B) = (A+B) (A'+B)



Example 2: F = $M_0 \cdot M_1 \cdot M_2$

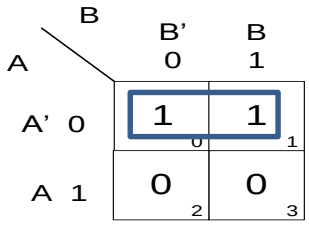


Example 3: F = ΠM(1,3)



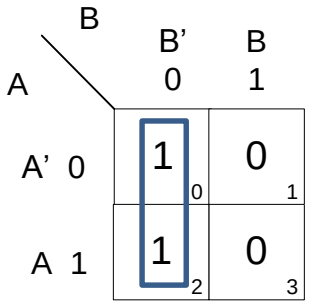
- ***Reduce following SOP expressions:***

Example 1: F = $m_0 + m_1$



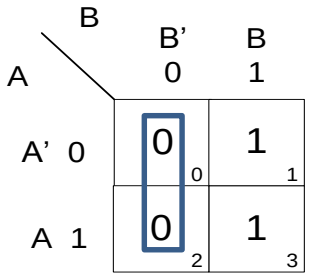
F = A'

Example 2: F = A'B' + AB'



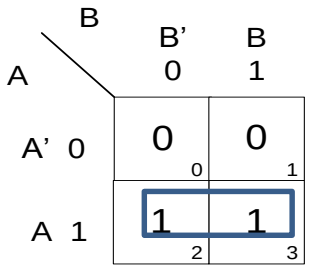
F = B'

Example 3: F = Σ(1,3)



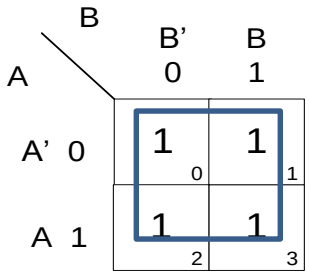
F = B

Example 4: F = $m_2 + m_3$



F = A

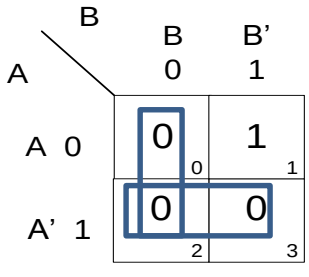
Example 5: F = ∑m(0,1,2,3)



F = 1

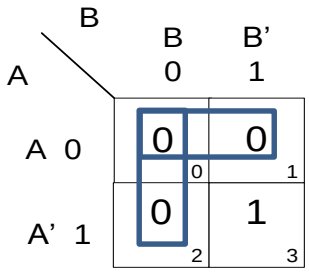
- ***Reduce following POS expressions:***

Example 1: F = Π(0,2,3)



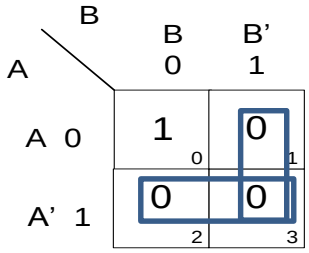
F = A'B

Example 2: F = (A+B) (A'+B) (A+B')



F = AB

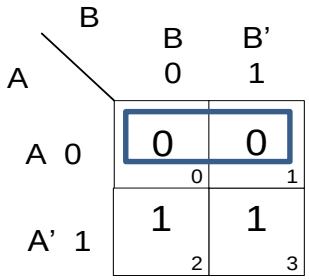
Example 3: F = $M_3 \cdot M_1 \cdot M_2$



F = A'B'

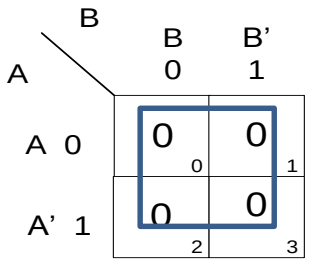
Example 4: $m_2 + m_3$

F = $m_2 + m_3 = \Pi(0,1)$



F = A

Example 5: F = ΠM(0,1,2,3)



F = 0

2.5.2 3 variables k-map

Reduce following SOP expression:

- For 3 variables, in SOP form there are 8 combinations as follow;

A'B'C'	(m_0, 000)
A'B'C	(m_1, 001)
A'BC'	(m_2, 010)
A'BC	(m_3, 011)
AB'C'	(m_4, 100)
A'BC'	(m_5, 101)
A'B'C	(m_6, 110)
ABC	(m_7, 111)

- For the case of 3 variables, we form a map consisting of 2^3=8 cells as shown in Figure

A \ BC	00	01	11	10
0	000 0	001 1	011 3	010 2
1	100 4	101 5	111 7	110 6

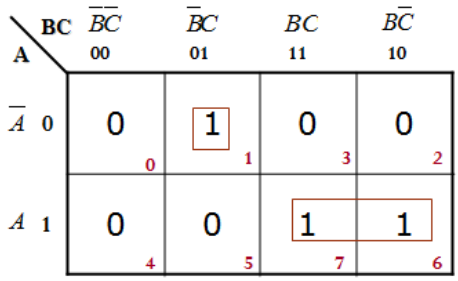
A \ BC	$\bar{B}\bar{C}$ 00	$\bar{B}C$ 01	BC 11	$B\bar{C}$ 10
$\bar{A}$ 0	$\bar{A}\bar{B}\bar{C}$ 0	$\bar{A}\bar{B}C$ 1	$\bar{A}BC$ 3	$\bar{A}B\bar{C}$ 2
A 1	$A\bar{B}\bar{C}$ 4	$A\bar{B}C$ 5	ABC 7	$AB\bar{C}$ 6

EXAMPLE 1: F = A'B'C' + ABC + A'BC'

A \ BC	$\bar{B}\bar{C}$ 00	$\bar{B}C$ 01	BC 11	$B\bar{C}$ 10
$\bar{A}$ 0	0 0	1 1	0 3	1 2
A 1	0 4	0 5	1 7	0 6

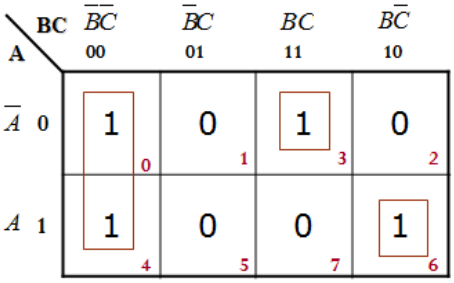
F = A'B'C + ABC + A'BC'

EXAMPLE 2: F = Σ(1,6,7)



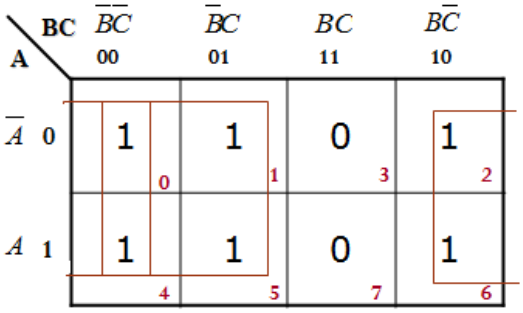
F = A'B'C' + AB

EXAMPLE 3: F = A'B'C' + ABC' + AB'C' + A'BC



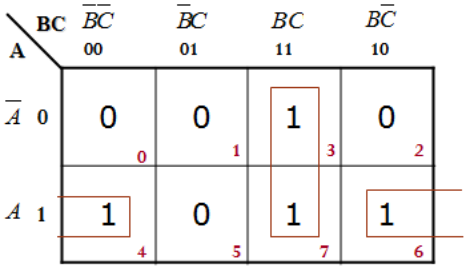
F = B'C' + AC' + A'BC

EXAMPLE 4: F = Σm(0,1,2,4,5,6)



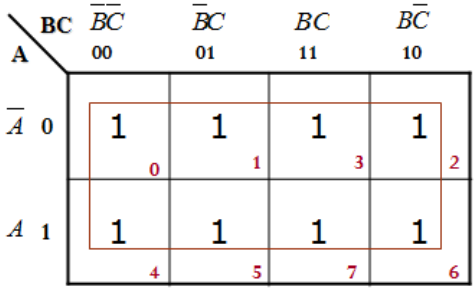
F = B' + C'

EXAMPLE 5: F = $m_3 + m_4 + m_6 + m_7$



F = BC + AC'

EXAMPLE 6: F = Σm(3,7,1,6,0,2,5,4)



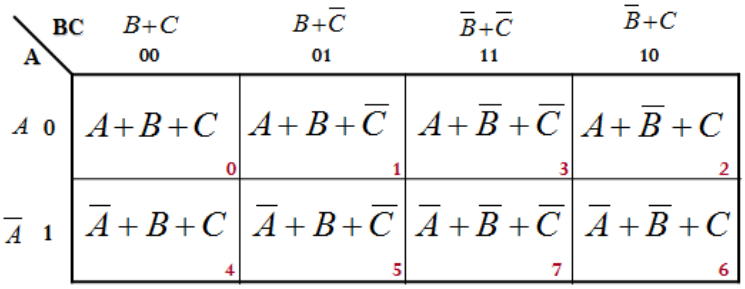
F = 1

- ***Reduce following POS expression***

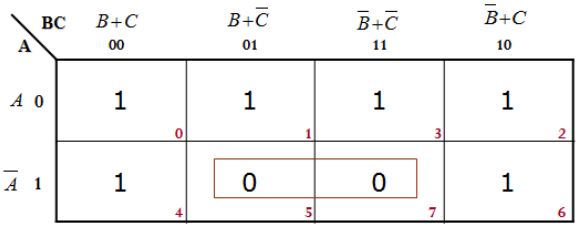
- For 3 variables, in POS form there are 8 combinations as follow;

A + B + C	(M_0, 000)
A + B + C'	(M_1, 001)
A + B' + C	(M_2, 010)
A + B' + C'	(M_3, 011)
A' + B + C	(M_4, 100)
A' + B + C'	(M_5, 101)
A' + B' + C	(M_6, 110)
A' + B' + C'	(M_7, 111)

- For the case of 3 variables, we form a map consisting of 2^3=8 cells as shown in Figure

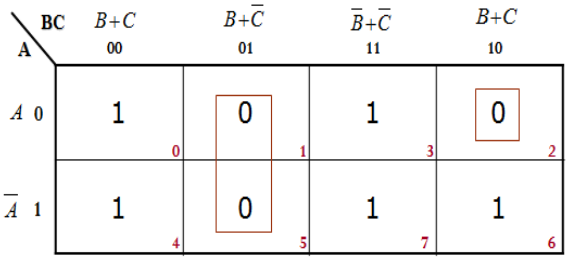


EXAMPLE 1: F = (A'+B'+C')(A'+B+C')



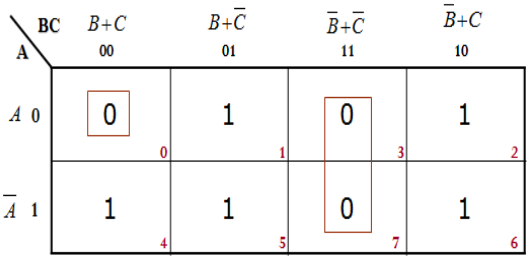
F = (A' + C')

EXAMPLE 2: F = ΠM(1,2,5)



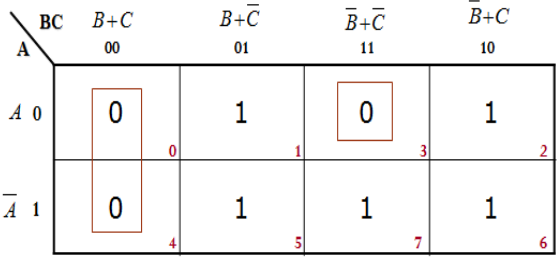
F = (B + C') (A + B + C)

EXAMPLE 3: F = M0·M3·M7



F = (A + B + C) (B' + C')

EXAMPLE 4: F = (A+B+C)(A+B'+C')(A'+B+C)



F = (B + C) (A + B' + C')

EXAMPLE 5: F = ΠM(5,7,0,3,2,4,6,1)

F = 0

2.5.3 4 variables k-map

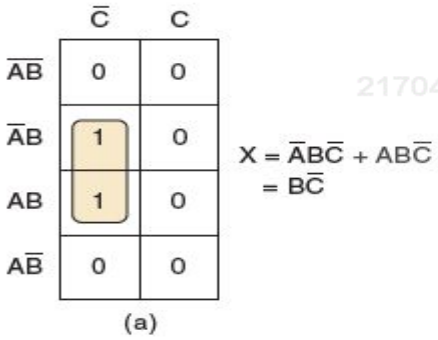
- ***Reduce following SOP expression:***

- For 4 variables, 2^4 = 16 combinations are available;

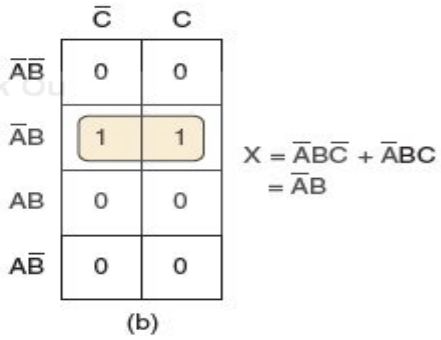
AB \ CD	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

- ***Looping:***

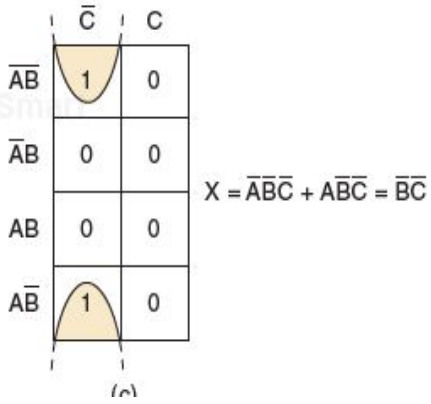
- Looping Groups of Two:



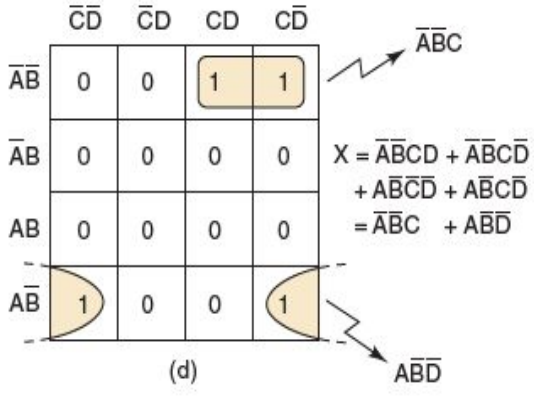
(a)



(b)

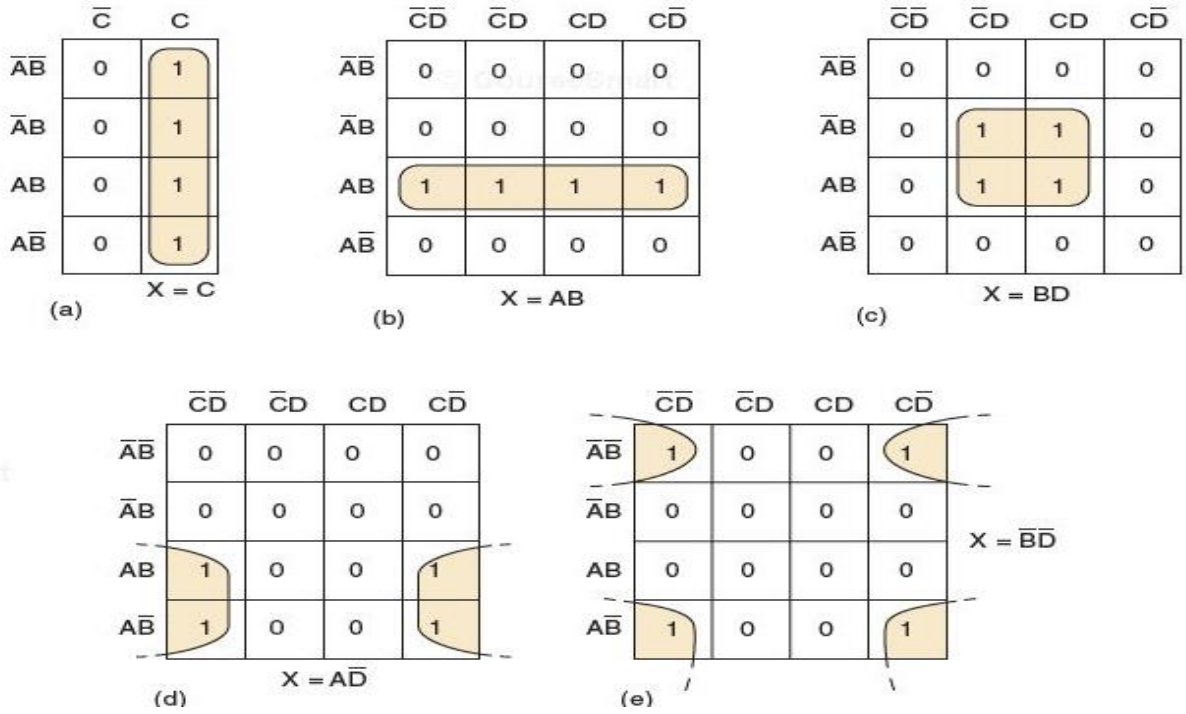


(c)

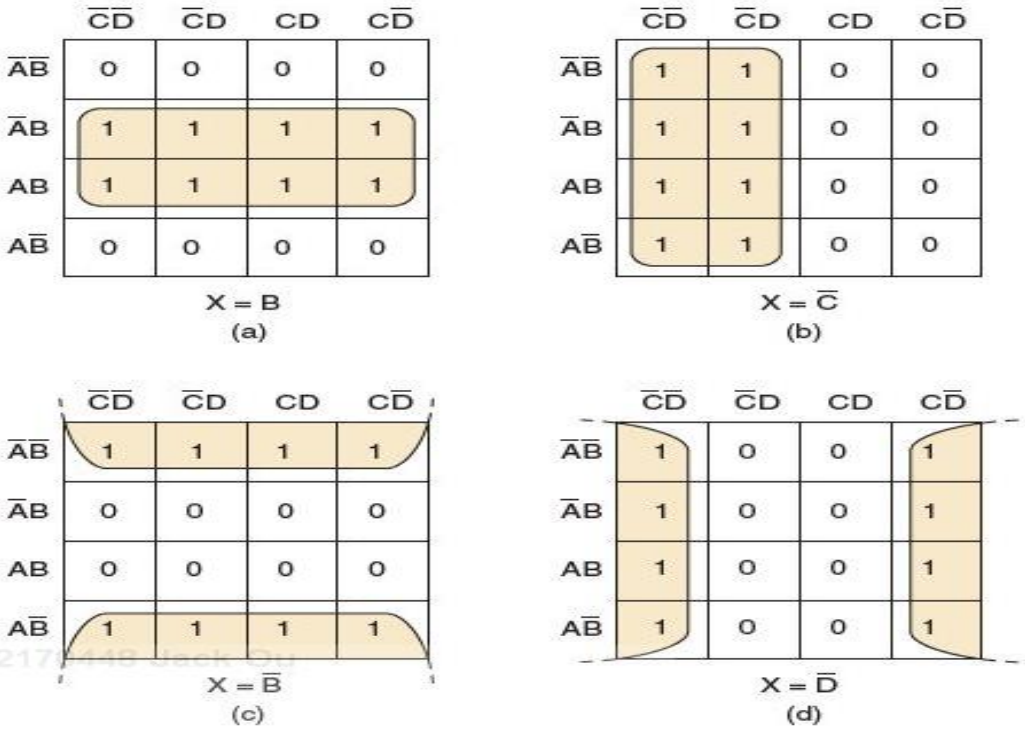


(d)

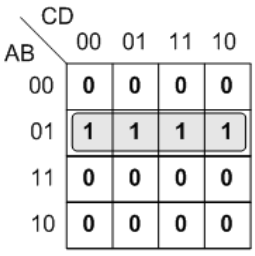
- Looping Groups of Four:



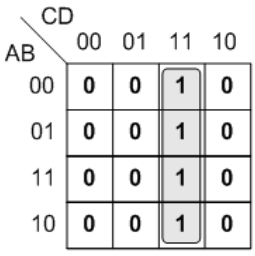
- Looping Groups of Eight:



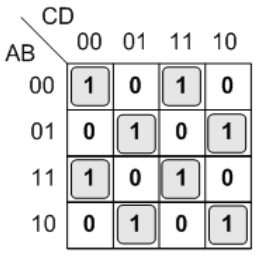
- **EXAMPLES:**



$f = \Sigma(4,5,6,7) = \bar{A} \bullet B$

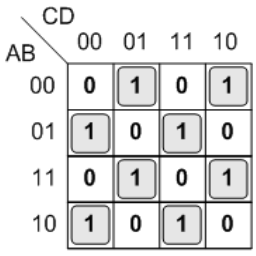


$f = \Sigma(3,7,11,15) = C \bullet D$



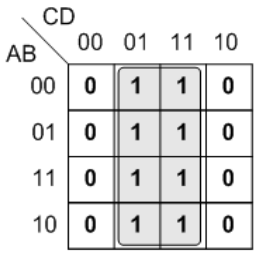
$f = \Sigma(0,3,5,6,9,10,12,15)$

$f = A \otimes B \otimes C \otimes D$



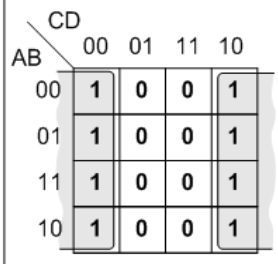
$f = \Sigma(1,2,4,7,8,11,13,14)$

$f = A \oplus B \oplus C \oplus D$



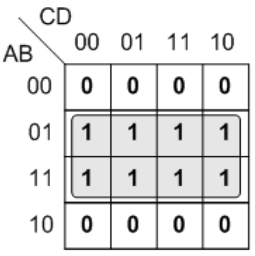
$f = \Sigma(1,3,5,7,9,11,13,15)$

$f = D$



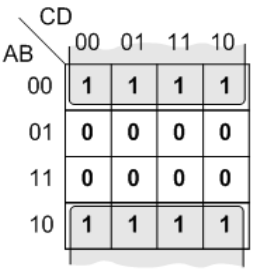
$f = \Sigma(0,2,4,6,8,10,12,14)$

$f = \bar{D}$



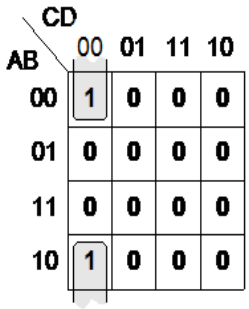
$f = \Sigma(4,5,6,7,12,13,14,15)$

$f = B$

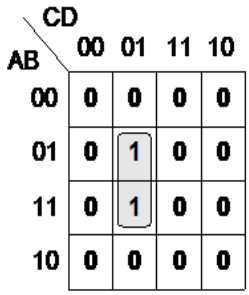


$f = \Sigma(0,1,2,3,8,9,10,11)$

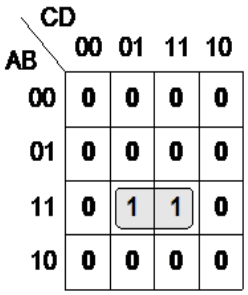
$f = \bar{B}$



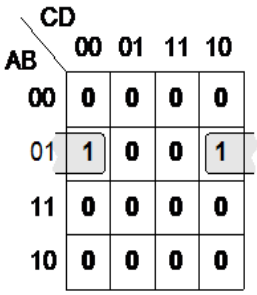
$f = \sum(0,8) = \bar{B} \bullet \bar{C} \bullet \bar{D}$



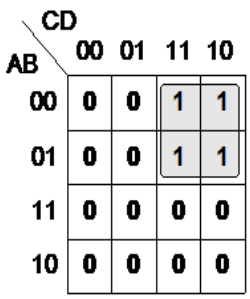
$f = \Sigma(5,13) = B \bullet \bar{C} \bullet D$



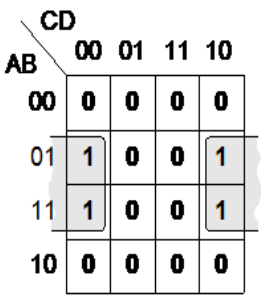
$f = \Sigma(13,15) = A \bullet B \bullet D$



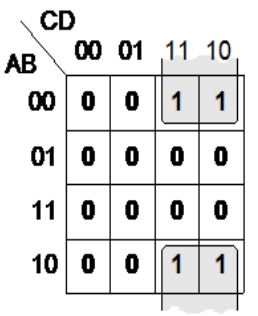
$f = \Sigma(4,6) = \bar{A} \bullet B \bullet \bar{D}$



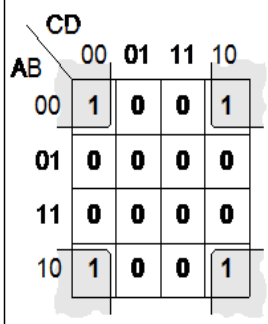
$f = \Sigma(2,3,6,7) = \bar{A} \bullet C$



$f = \Sigma(4,6,12,14) = B \bullet \bar{D}$

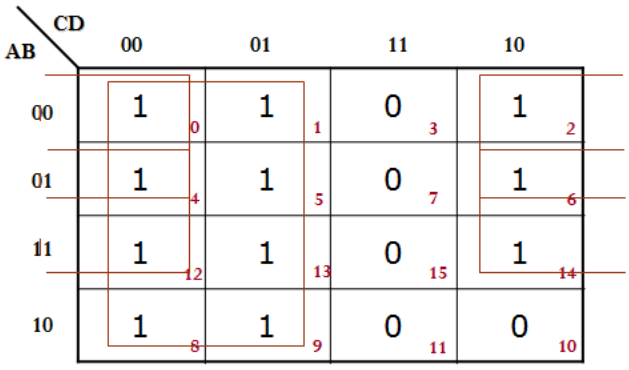


$f = \Sigma(2,3,10,11) = \bar{B} \bullet C$



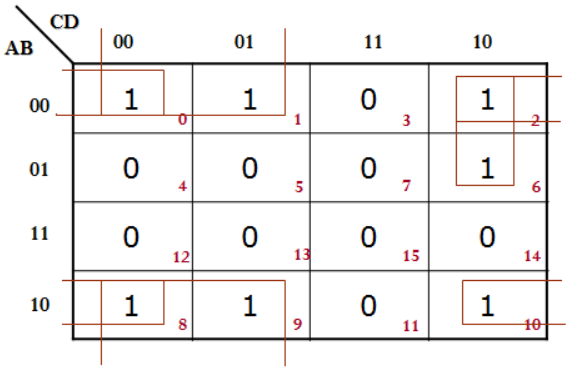
$f = \Sigma(0,2,8,10) = \bar{B} \bullet \bar{D}$

EXAMPLE 1: ∑ (0,1,2,4,5,6,8,9,12,13,14)



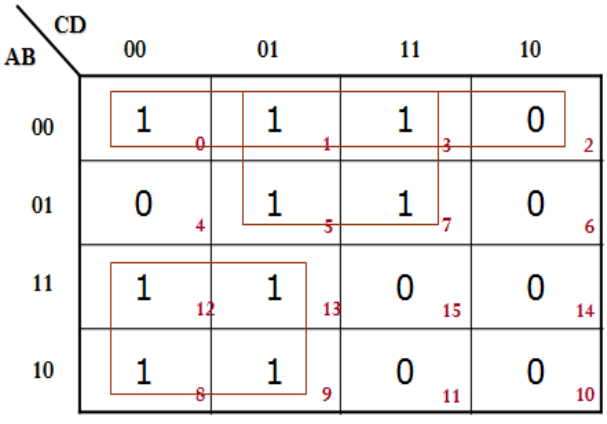
F = C' + A'D' + BD'

EXAMPLE 2: A'B'C' + B'CD' + A'BCD' + AB'C'



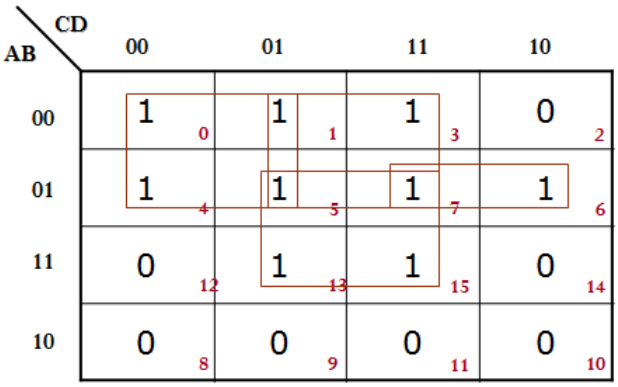
F = B'C' + B'D' + A'CD'

EXAMPLE 3: ∑ (0,1,2,3,5,7,8,9,12,13)



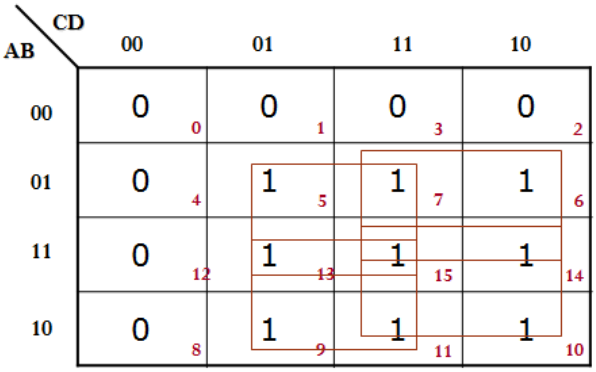
F = A'B' + AC' + A'D

EXAMPLE 4: ∑ (0,1,3,4,5,6,7,13,15)



F = A'C' + A'D + BD + A'B

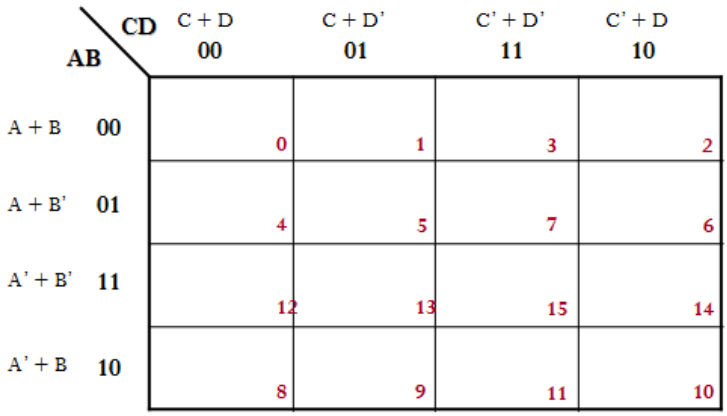
EXAMPLE 5: ∑m (5,6,7,9,10,11,13,14,15)



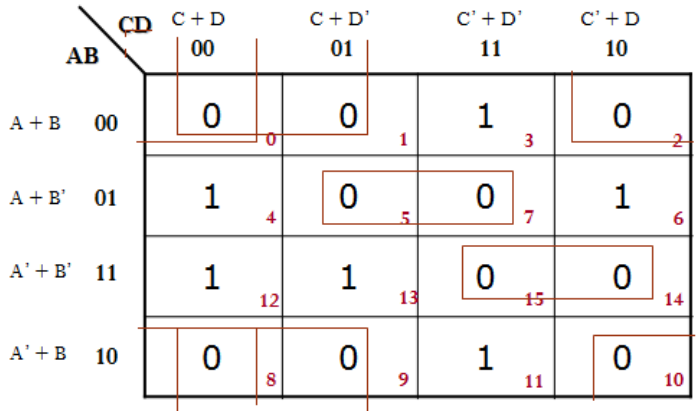
F = BD + BC + AD + AC

- ***Reduce following POS expression:***

- For 4 variables, 2^4 = 16 combinations are available;

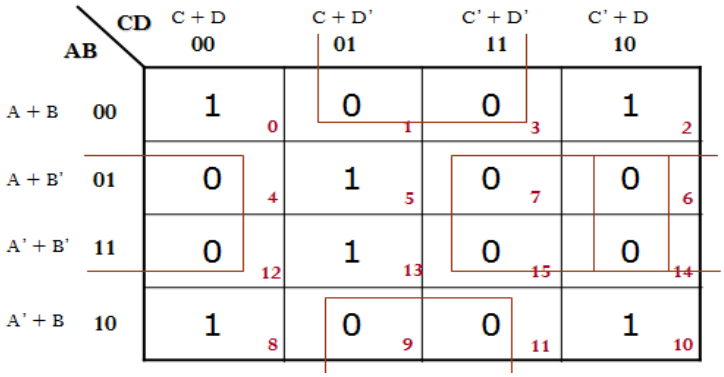


EXAMPLE 1: ΠM (0,1,2,5,7,8,9,10,14,15)



F = (B + D) (B + C) (A + B' + D') (A' + B' + C')

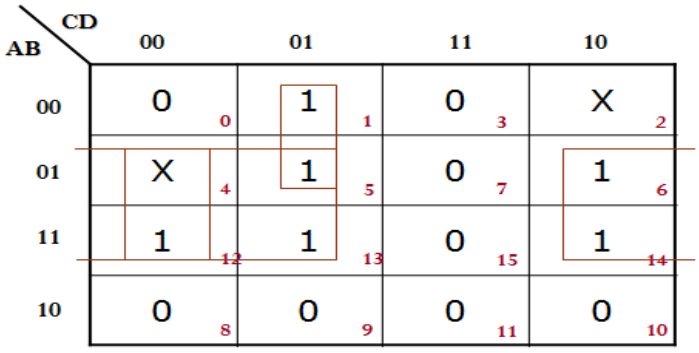
EXAMPLE 2: M_1 M_3 M_4 M_7 M_8 M_9 M_{11} M_{12} M_{14} M_{15}



F = (B' + D) (C' + B') (D' + B)

- ***Don't Care Combinations:***

EXAMPLE 1: ∑m (1,5,6,12,13,14) + d (2,4)



F = BC' + BD' + A'C'D

EXAMPLE 2: ΠM **(4,7,10,11,12,15) · d (6,8)**

AB \ CD	C + D (00)	C + D' (01)	C' + D' (11)	C' + D (10)
A + B (00)	1 (0)	1 (1)	1 (3)	1 (2)
A + B' (01)	0 (4)	1 (5)	0 (7)	X (6)
A' + B' (11)	0 (12)	1 (13)	0 (15)	1 (14)
A' + B (10)	X (8)	1 (9)	0 (11)	0 (10)

F = (B' + C + D) (B' + C' + D') (A' + B + C')

2.5.4 5 variables k-map

- ***Reduce following SOP expression:***

- For 4 variables, 2^5 = 32 combinations are available;

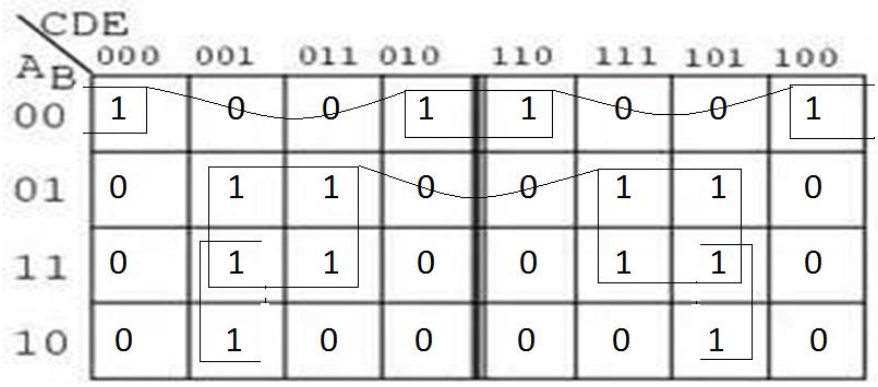
AB \ CDE	000	001	011	010	110	111	101	100
00								
01								
11								
10								

EXAMPLE 1: ∑m (0,2,3,10,11,12,13,16,17,18,19,20,21,26,27)

AB \ CDE	000	001	011	010	110	111	101	100
00	1	0	1	1	0	0	0	0
01	0	0	1	1	0	0	1	1
11	0	0	1	1	0	0	0	0
10	1	1	1	1	0	0	1	1

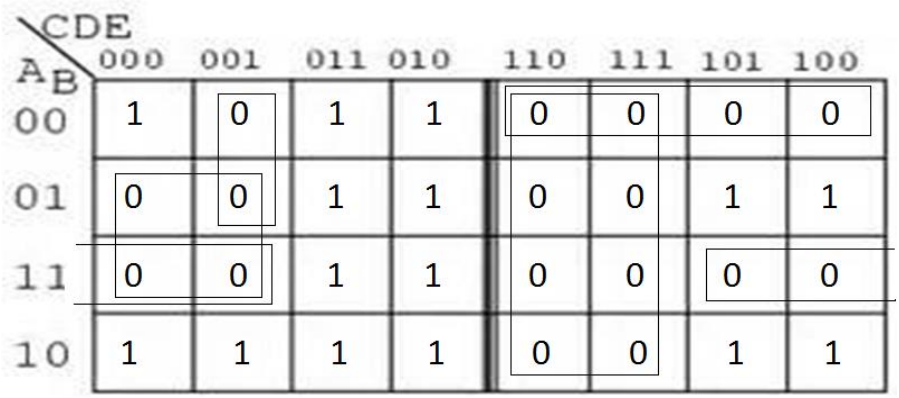
F = C'D + B'C'E' + AB'D' + A'BCD'

EXAMPLE 2: ∑m (0,2,4,6,9,11,13,15,17,21,25,27,29,31)



F = BE + AD'E + A'B'E'

EXAMPLE 3: ΠM (1,4,5,6,7,8,9,14,15,22,23,24,25,28,29,30,31)



F = (B' + C + D) (C' +D') (A + B + C') (A' + B' + D) (A + C + D + E')

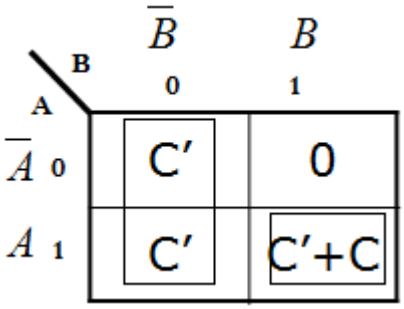
2.5.5 Variable Entered Map (VEM)

- K-map is useful tool in logic function minimization if fewer than six variables are involved.
- When no. of variables increases, the calculation becomes tough.
- The VARIABLE-ENTERED MAP can be used to plot an *n-variable* problem on an *n-1 variable* map.
- For e.g. 5 variable problem can be solved using 4 variable K-map.
- Also 5 variable problem can be solved using 3 & lesser variable K-map, but the method becomes difficult.

➢ **Plotting the Variable Entered Map (VEM)**

- Consider a function X=A'B'C' + ABC' + AB'C' + ABC.
- Easily solved using 3-variable K-map.

- But also using 2-variable K-map.
- X = A'B' (C') + AB' (C') + AB (C+C') where C is the Map-Entered Variable.

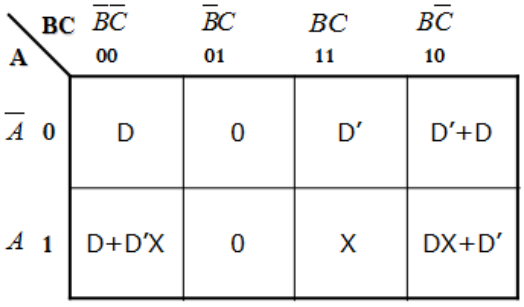


F = B' + AB

EXAMPLE 1: Plot a function F, whose truth table is given & D is MEV

A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	X
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

ANS:



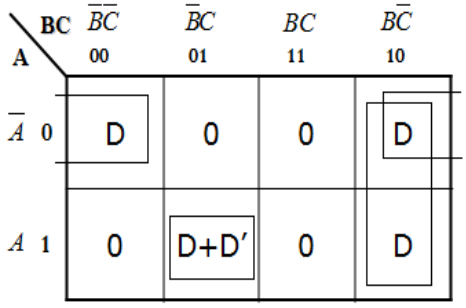
➢ **Reduce the following Functions using Variable Entered Map (VEM)**

EXAMPLE 1: F = AB'CD + A'BC'D + AB'CD' + ABC'D + A'B'C'D Consider D as MEV (Mapped Entered Variable)

ANS:

STEP 1: A'B'C'(D) + A'BC'(D) + AB'C(D + D') + ABC'(D)

STEP 2:



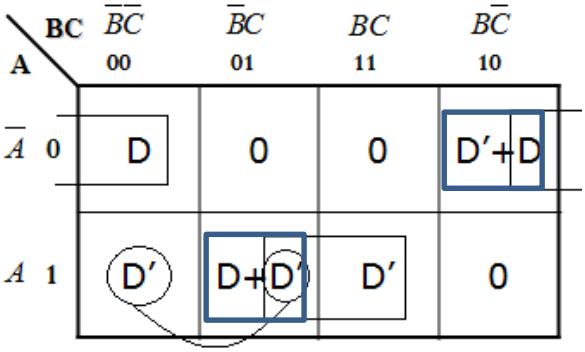
F = BC'D + A'C'D + AB'C

EXAMPLE 2: F = F = A'B'C'D + A'BC'D' + A'BC'D + AB'C'D' + AB'CD' + AB'CD + ABCD' Consider D as MEV (Mapped Entered Variable)

ANS:

STEP 1: A'B'C'(D) + A'BC'(D' + D) + AB'C'(D') + AB'C(D' + D) + ABC(D')

STEP 2:



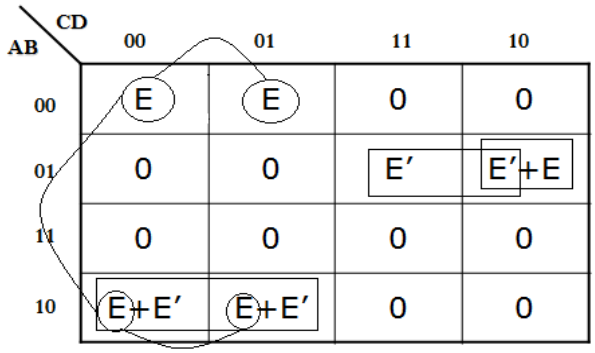
F = A'C'D + AB'D' + ACD' + AB'C + A'BC'

EXAMPLE 3: F = A'B'C'D'E + A'B'C'DE + A'BCD'E' + A'BCD'E + AB'C'D'E' + AB'C'D'E + AB'C'D + A'BCDE' Consider E as MEV (Mapped Entered Variable).

ANS:

STEP 1: **A'B'C'D'(E) + A'B'C'D(E) + A'BCD'(E' + E) + AB'C'D'(E' + E) + AB'C'D(E' + E) + A'BCDE'**

STEP 2:



F= B'C'E + AB'C'(E + E') + A'BCE' + A'BCD'(E' + E)

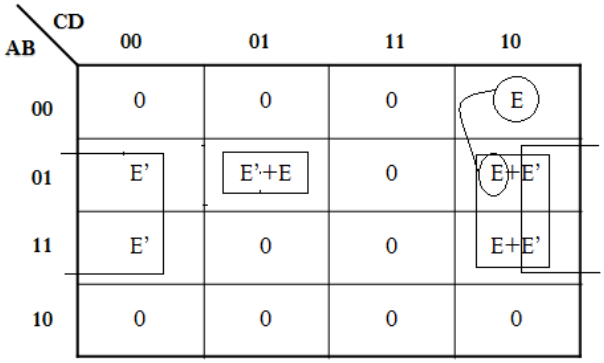
F= B'C'E + AB'C'+ A'BCE' + A'BCD'

EXAMPLE 4: F = A'BC'D'E' + ABC'D'E' + A'BC'DE' + A'BCD' + A'B'CD'E + ABCD'E' + ABCD'E' + A'BC'DE Consider E as MEV (Mapped Entered Variable).

ANS:

STEP 1: **A'BC'D'(E') + ABC'D'(E') + A'BC'D(E' + E) + A'BCD'(E + E') + A'B'CD'(E) + ABCD'(E' + E)**

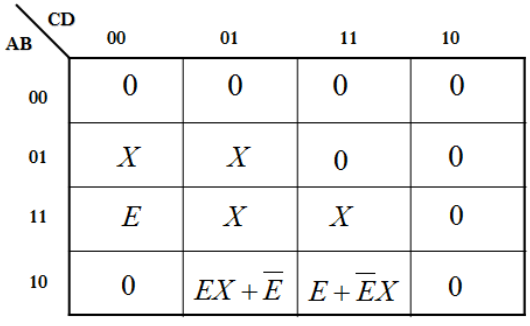
STEP 2:



F = BD'E' + BCD'(E + E') + A'CD'E + A'BC'D(E + E')

F = BD'E' + BCD' + A'CD'E + A'BC'D

EXAMPLE 5: Solve following;



ANS:

AB \ CD	00	01	11	10
00	0	0	0	0
01	X	X	0	0
11	E	X	X	0
10	0	$EX+\overline{E}$	$E+\overline{E}X$	0

F = BC'E + AD (E + E')

F = BC'E + AD

EXAMPLE 6: Solve following;

AB \ CD	00	01	11	10
00	$E+\overline{E}X$	E	EX	$E+\overline{E}$
01	X	0	0	0
11	$EX+\overline{E}$	0	X	$E+\overline{E}$
10	0	EX	E	$\overline{E}X$

ANS:

AB \ CD	00	01	11	10
00	$E+\overline{E}X$	E	EX	$E+\overline{E}$
01	X	0	0	0
11	$EX+\overline{E}$	0	X	$E+\overline{E}$
10	0	EX	E	$\overline{E}X$

Here, in E'X we have choose X as 0 So E'X = 0.

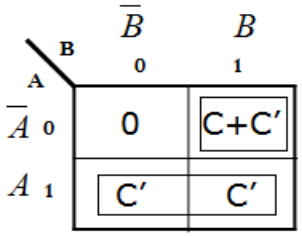
F = B'DE + A'B'D' + ABD'

EXAMPLE 7: F = A'BC' + A'BC + ABC' + AB'C' Consider C as MEV (Mapped Entered Variable).

ANS:

STEP 1: F = A'B(C' + C) + AB(C') + AB'(C')

STEP 2:



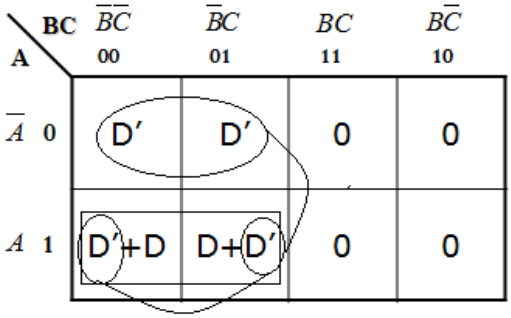
F = AC' + A'B

EXAMPLE 8: Y = A'B'C'D' + A'B'CD' + AB'C'D' + AB'C'D + AB'CD + AB'CD' Consider D as MEV (Mapped Entered Variable).

ANS:

STEP 1: F = A'B'C'(D') + A'B'C(D') + AB'C'(D'+D) + AB'C(D + D')

STEP 2:



F = AB'(D' + D) + B'D'

F = AB' + B'D'

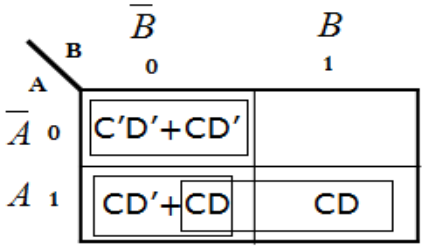
- **Variable-Entered Map with two MEV**

EXAMPLE 1: X=A'B'C'D'+A'B'CD'+ABCD+AB'CD+AB'CD' Use C & D as MEV.

ANS:

STEP 1: X = A'B'(C'D' + CD') + AB(CD) + AB'(CD + CD')

STEP 2:



F = ACD + AB'(CD' + CD) + A'B'(C'D' + CD')

F = ACD + AB'C(D' + D) + A'B'D'(C' + C)

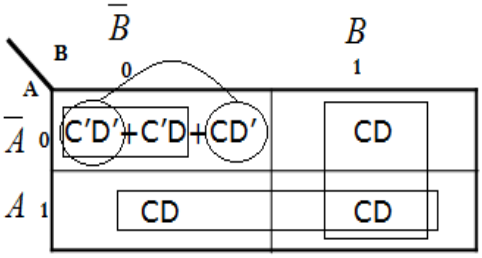
F = ACD + AB'C+ A'B'D'

EXAMPLE 2: X=A'B'C'D' + A'B'C'D + A'B'CD' + A'BCD + ABCD+ AB'CD Use C & D as MEV

ANS:

STEP 1: X = **A'B'(C'D' + C'D + CD') + A'B(CD) + AB(CD)+ AB'(CD)**

STEP 2:



F = ACD + BCD + A'B'(C'D' + C'D) + A'B'(C'D' + CD')

F = ACD + BCD + A'B'C'(D' + D) + A'B'D'(C' + C)

F = ACD + BCD + A'B'C' + A'B'D'

3.1 MULTIPLEXER:

- Multiplexer is a special type of combinational circuit.
- The figure below shows the n x 1 multiplexer and its equivalent circuit representation.
- There are 'n' data inputs, 1 output and 'm' select lines, i.e. $2^m=n$.
- A multiplexer is a digital circuit which selects one of the n data inputs and routes it to the output. The selection of one of the n inputs is done by the select inputs
- To select 'n' inputs, 'm' select lines such that $2^m=n$.
- Depending on the digital code applied at the select inputs, one out of 'n' data sources is elected and transmitted to the single output.
- As shown in the figure, the multiplexer acts like a digitally controlled single pole, multiple way switch.
- The output gets connected to only one of the 'n' data inputs at given instant of time.
- It is also called DATA SELECTOR.

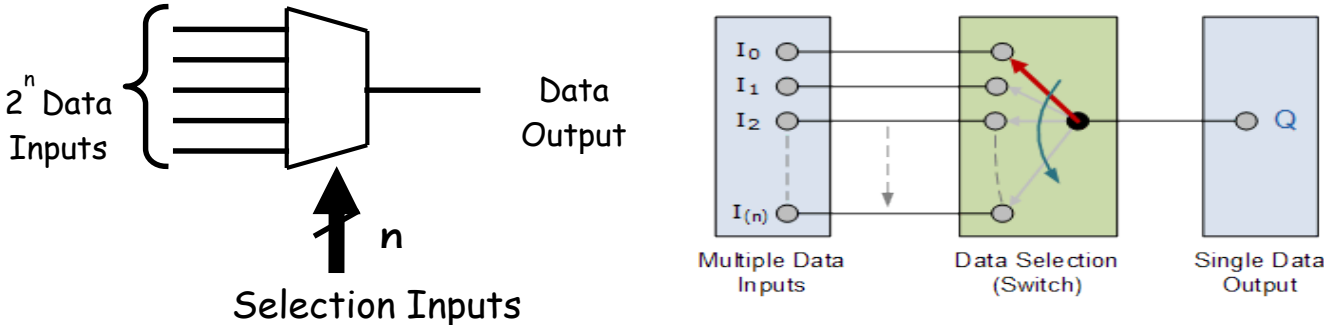


Fig. 3.1: Illustration of multiplexer

- Different types of multiplexers are available viz. 2 to 1, 4 to 1, 8 to 1, 16 to 1 and onwards.
- Multiplexers are needed in most of electronics systems, where the digital data is available on more than one lines, and it becomes necessary to route this data over a single line.
- Many logical functions can be implemented using Multiplexer.

3.1.1 2 X 1 Multiplexer

- It has two data inputs I_0 and I_1, one select input S, and one output Y.

Block Diagram

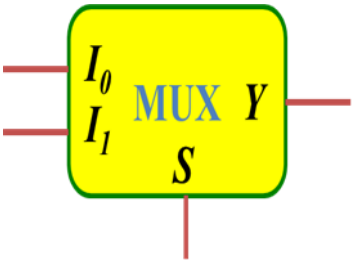


Fig. 3.2: Block diagram of 2 X 1 multiplexer

Truth Table

Table 3.1: Truth table of 2 X 1 multiplexer

Select Line S	Output Y
0	I_0
1	I_1

Boolean Equation

$$Y = S'I_0 + SI_1$$

Circuit Diagram

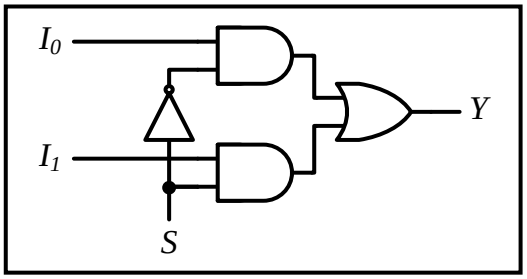


Fig. 3.3: Circuit diagram of 2 X 1 multiplexer

Working

- When S=0, the upper AND gate will turn ON and lower AND gate will turn OFF, and so the input I_0 appears in the output.
- When S=1, the upper AND gate will turn OFF and lower AND gate will turn ON, and so the input I_1 appears in the output.

3.1.2 4 X 1 Multiplexer

- It has four data inputs I_3, I_2, I_1 and I_0, two select inputs S_1 & S_0, and one output Y.
- Here 2^n=4 inputs, i.e. n=2 select lines and m = 1 output.

Block Diagram

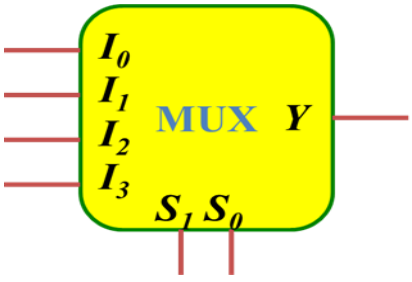


Fig. 3.4: Block diagram of 4 X 1 multiplexer

Truth Table

Table 3.2: Truth table of 4 X 1 multiplexer

Select Line		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Working

- According to the truth table, when **S_1 S_0=00**, the input I_0 is selected and routed to the output.
- When **S_1 S_0=01**, the input I_1 is selected and routed to the output.
- Similarly, when **S_1 S_0=10**, then $Y=I_2$ & when **S_1 S_0=11**, then $Y=I_3$.

Boolean Equation

$$Y = S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_3$$

Circuit Diagram

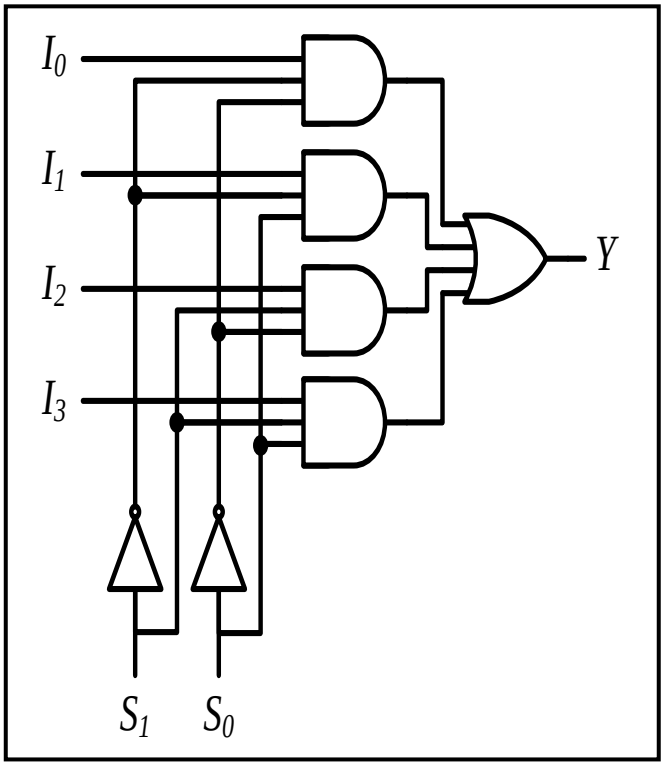


Fig. 3.5: Circuit diagram of 4 X 1 multiplexer

3.1.3 8 X 1 Multiplexer

- It has 8 data inputs I_7, I_6, I_5, I_4, I_3, I_2, I_1 and I_0, three select inputs S_2, S_1 & S_0, and one output Y.
- Here 2^n=8 inputs, i.e. n=3 select lines and m = 1 output.

Circuit Diagram

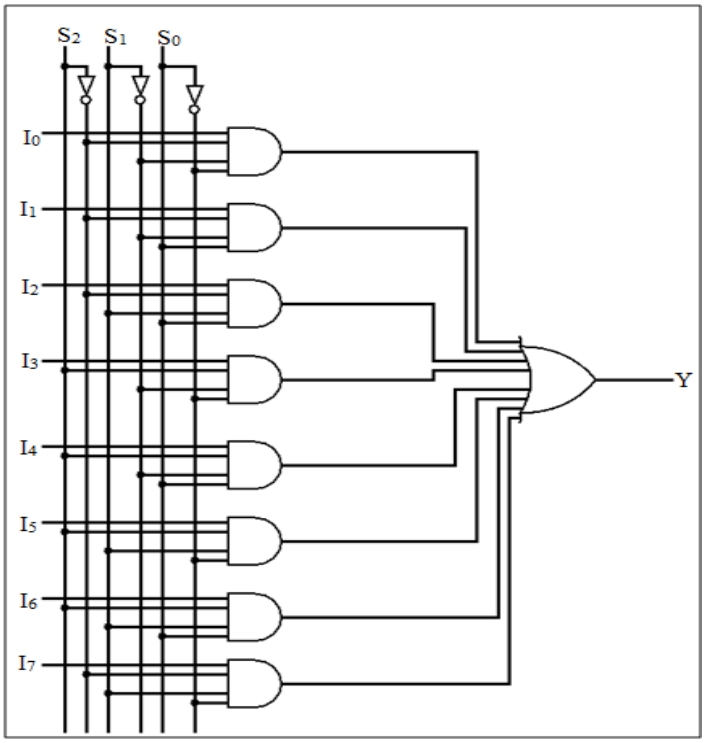


Fig. 3.6: Circuit diagram of 8 X 1 multiplexer

Truth Table

Table 3.3: Truth table of 8 X 1 multiplexer

Select Line			Output
S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

Boolean Equation

$$Y = S_2'S_1'S_0'I_0 + S_2'S_1'S_0I_1 + S_2'S_1S_0'I_2 + S_2'S_1S_0I_3 + S_2S_1'S_0'I_4 + S_2S_1'S_0I_5 + S_2S_1S_0'I_6 + S_2S_1S_0I_7$$

Operation

- According to the truth table, when **S_2S_1 S_0=000**, the input I_0 is selected and routed to the output.
- When **S_2S_1 S_0=001**, the input I_1 is selected and routed to the output.
- Similarly, for other combinations of select lines particular input is routed to the output.

3.1.4 8 X 1 Multiplexer using 4 X 1 and 2 X 1 Multiplexer

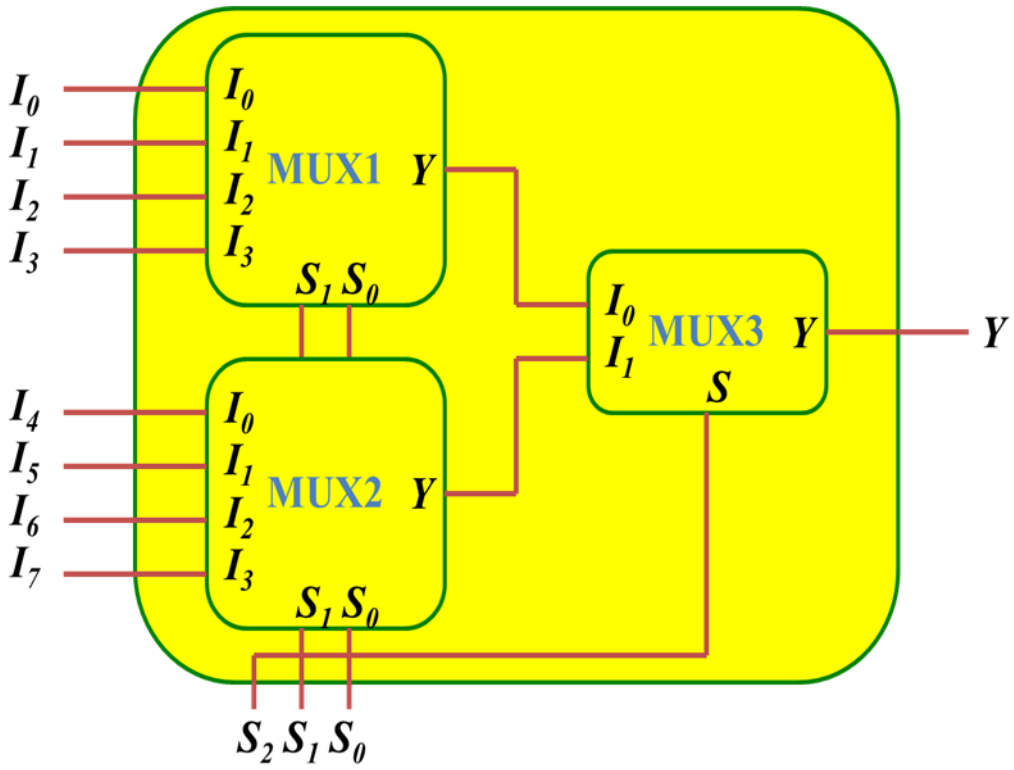
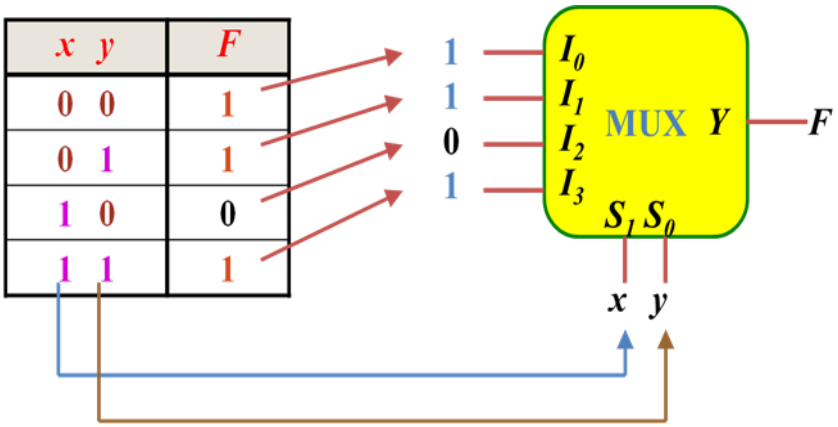


Fig. 3.7: Block diagram of 8 X 1 multiplexer using 4 X 1 and 2 X 1 multiplexer

- MUX1 & MUX2 are 4 X 1 Multiplexer & MUX3 is 2 X 1 Multiplexer.
- Assuming that input I_5 is to be routed through the output.
- So select lines will be **$S_2S_1S_0$=101.**
- Now, for MUX1 & MUX2, **S_1S_0=01**, so the inputs I_1 & I_5 will be routed through each 4 X 1 Multiplexer.
- I_1 & I_5 appears as input to 2 X 1 Multiplexer.
- The value of **S_2=1**, so the second input which is I_5 will be routed through the output Y.

Example 1: Implement the function *F*(*x*, *y*) = ∑(0, 1, 3) using multiplexer.

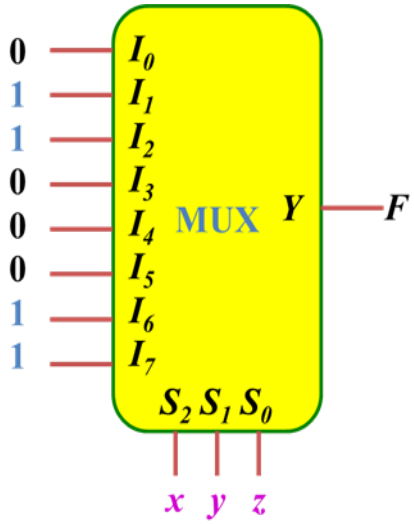
- Select multiplexer to implement this function.
- Here the function is of two variables i.e. there are two inputs, it means that two select lines will be needed.
- Therefore, $S_1 = x$, $S_0 = y$.
- 2 select lines → $2^2 = 4$ data inputs → 4 X 1 Multiplexer will be used.
- Connect the data inputs 0, 1 & 3 to logic 1 and the remaining to logic 0.



Example 2: Implement the function F(x, y, z) = ∑(1, 2, 6, 7) using multiplexer.

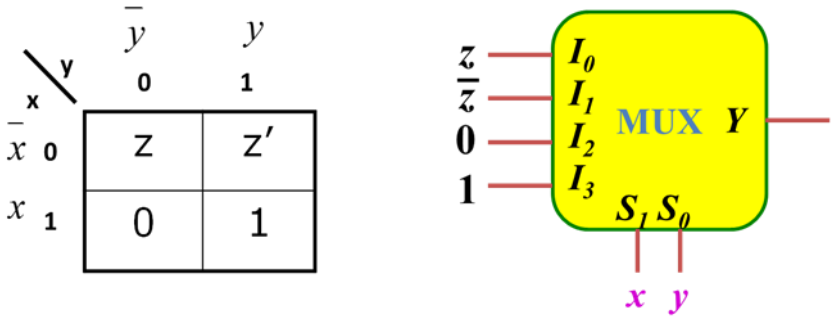
- 3 inputs → 3 select lines → 8 X 1 Multiplexer.
- Connect the data inputs 1, 2, 6 & 7 to logic 1 and the remaining to logic 0.
- Connect the select lines inputs to x, y & z respectively.

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



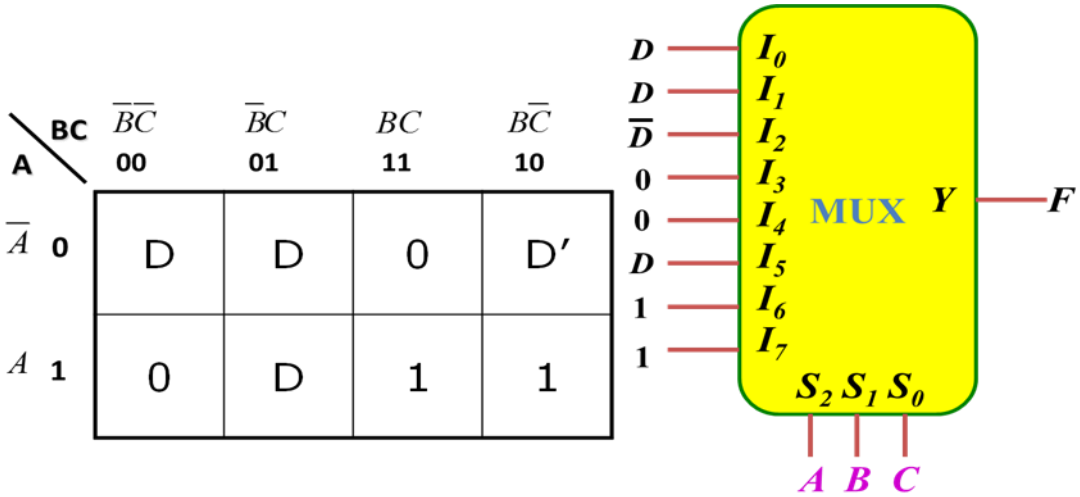
Example 3: Implement the function F(x, y, z) = ∑(1, 2, 6, 7) using 4 X 1 multiplexer.

- 3 inputs will need 8 X 1 MUX but 4 X 1 MUX is to be used.
- So Variable Entered Map method is to be used in which considering 'z' as MEV.
- Four inputs are available z, z', 0, 1 which are connected to I_0, I_1, I_2 & I_3 respectively.
- x and y are connected to select lines S_1 & S_0 respectively.



Example 4: Implement the function F (A, B, C, D) = ∑ (1, 3, 4, 11, 12, 13, 14, 15) using 8 X 1 multiplexer.

- 4 inputs will need 16 X 1 MUX but 8 X 1 MUX is to be used.
- So Variable Entered Map method is to be used in which considering 'D' as MEV.
- Eight inputs are available 1, 1, D, 0, 0, D', D & D which are connected to I_7, I_6, I_5, I_4, I_3, I_2, I_1 and I_0 respectively.
- A, B & C are connected to select lines S_2, S_1 & S_0 respectively.



3.2 DEMULTIPLEXER:

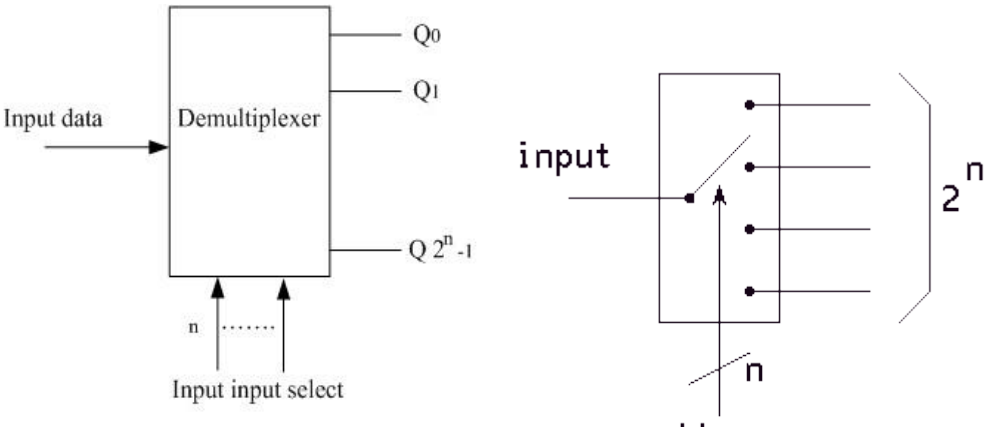


Fig. 3.8: Illustration of demultiplexer

- It has one input common data, 'n' select lines and 'm' output lines.
- A demultiplexer performs the reverse operation of a multiplexer i.e. it receives one input and distributes it over several outputs.
- At a time only one output line is selected by the select lines and the input is transmitted to the selected output line.
- Relation between 'n' output lines and m select lines is as follows :

$$n = 2^m$$

3.2.1 1 X 4 Demultiplexer

- 1 to 4 Demultiplexer has one data input F; select line inputs a,b and four outputs A, B, C & D.
- The select lines control the data to be routed. It helps in selecting the output on which the data will be routed.

Switch Representation

- Based on the switch control, the input is routed to particular output

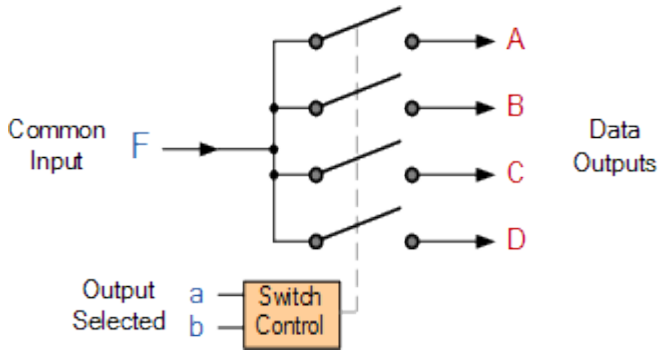


Fig. 3.9: Switching representation of 1 X 4 demultiplexer

Truth Table

Table 3.4: Truth table of 1 X 4 demultiplexer

Select Line		Output Line
b	**a**	
0	0	A
0	1	B
1	0	C
1	1	D

Boolean Equation

$$A = Fb'a'; \quad B = Fb'a;$$
$$C = Fba'; \quad D = Fba;$$

Working

- When ab=”00”, the input data F is routed to the output A
- When ab=”01”, the input data F is routed to the output B
- When ab=”10”, the input data F is routed to the output C
- When ab=”11”, the input data F is routed to the output D

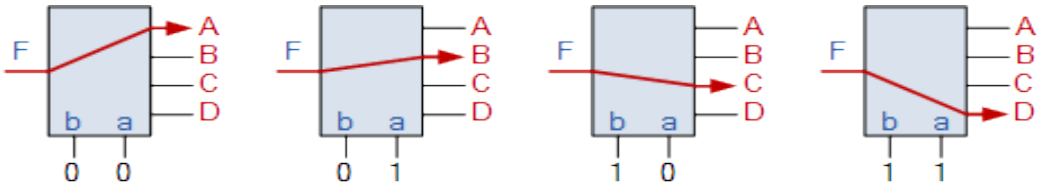


Fig. 3.10: Working of 1 X 4 demultiplexer

Circuit Diagram

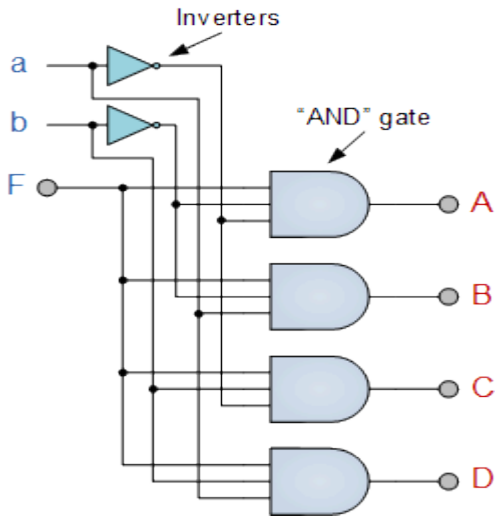


Fig. 3.11: Circuit diagram of 1 X 4 demultiplexer

3.2.2 1 X 8 Demultiplexer

- 1 to 8 Demultiplexer has one data input I; select line inputs are S_2, S_1 & S_0 and eight outputs F_0, F_1, F_2, F_3, F_4, F_5, F_6, F_7 & F_8.
- The select lines control the data to be routed. It helps in selecting the output on which the data i.e. 'I' will be routed.

Truth Table

Table 3.5: Truth table of 1 X 8 demultiplexer

Select Line			Output Line
S_2	S_1	S_0	
0	0	0	F_0
0	0	1	F_1
0	1	0	F_2
0	1	1	F_3
1	0	0	F_4
1	0	1	F_5
1	1	0	F_6
1	1	1	F_7

Boolean Equation

$$F_0 = I\overline{S_2}\overline{S_1}\overline{S_0}; \quad F_1 = I\overline{S_2}\overline{S_1}S_0;$$

$$F_2 = I\overline{S_2}S_1\overline{S_0}; \quad F_3 = I\overline{S_2}S_1S_0;$$

$$F_4 = IS_2\overline{S_1}\overline{S_0}; \quad F_5 = IS_2\overline{S_1}S_0;$$

$$F_6 = IS_2S_1\overline{S_0}; \quad F_7 = IS_2S_1S_0$$

Working

- When $S_2S_1S_0$="000", the input data routed to the output F_0
- When $S_2S_1S_0$="001", the input data routed to the output F_1
- When $S_2S_1S_0$="010", the input data routed to the output F_2
- When $S_2S_1S_0$="011", the input data routed to the output F_3
- When $S_2S_1S_0$="100", the input data routed to the output F_4
- When $S_2S_1S_0$="101", the input data routed to the output F_5
- When $S_2S_1S_0$="110", the input data routed to the output F_6
- When $S_2S_1S_0$="111", the input data routed to the output F_7

Circuit Diagram

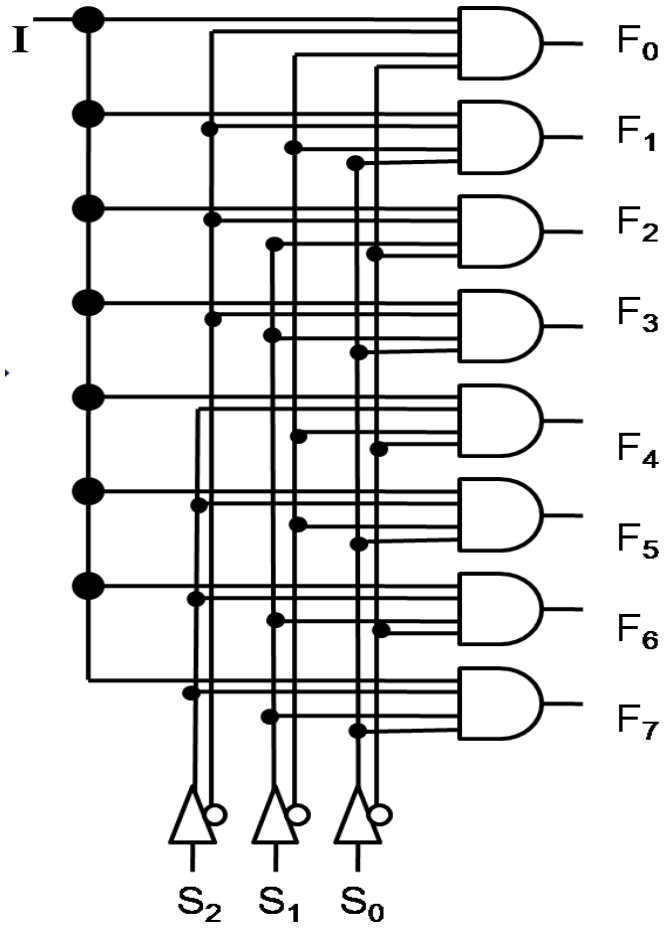


Fig. 3.12: Circuit diagram of 1 X 8 demultiplexer

3.3 ENCODER:

- It is a combinational circuit.
- It has 'n' input lines & 'm' output lines.
- An encoder produces an 'm' bit binary code corresponding to the digital input number of 'n' bits.
- Many types of Encoders – Octal to Binary (8 to 3), Decimal to BCD (10 to 4) etc.
- The block diagram is as shown below,

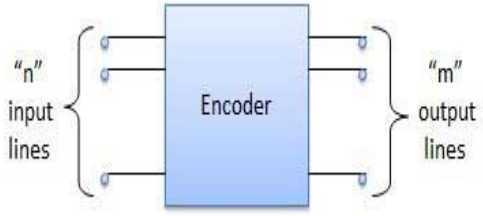


Fig. 3.13: Representation of encoder

3.3.1 Priority Encoder

- This is special type of encoder.
- Priorities are given to the input lines.
- If two or more input lines are '1' at the same time, then the input line with highest priority will be considered.
- The block diagram is shown below,

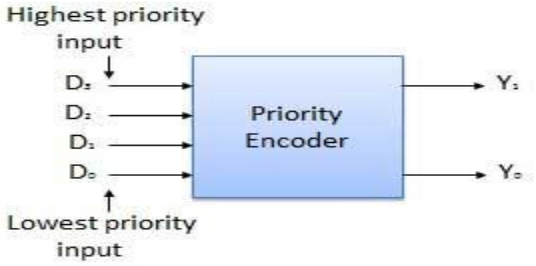


Fig. 3.14: Representation of priority encoder

- The truth table of priority encoder is as given below,
- There are four inputs, D_0 through D_3 and outputs Y_1 and Y_0. Out of the four inputs D_3 has the highest priority and D_0 has the lowest priority.
- That means if $D_3 = 1$ then $Y_1Y_0 = 11$ irrespective of the other inputs. Similarly if $D_3 = 0$ and $D_2 = 1$ then $Y_1Y_0 = 10$ irrespective of other inputs.

Table 3.6: Truth table of priority encoder

Inputs				Outputs	
D_3	**D_2**	**D_1**	D_0	**Y_1**	**Y_0**
0	0	0	0	X	X
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1

D_3D_2 \ D_1D_0	00	01	11	10
00	X	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$Y_1 = D_3 + D_2$

D_3D_2 \ D_1D_0	00	01	11	10
00	X	0	1	1
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

$Y_0 = D_3 + D_2' D_1$

Fig. 3.15: K-map representation of priority encoder

- Below figures shows the circuit diagram of Priority Encoder.

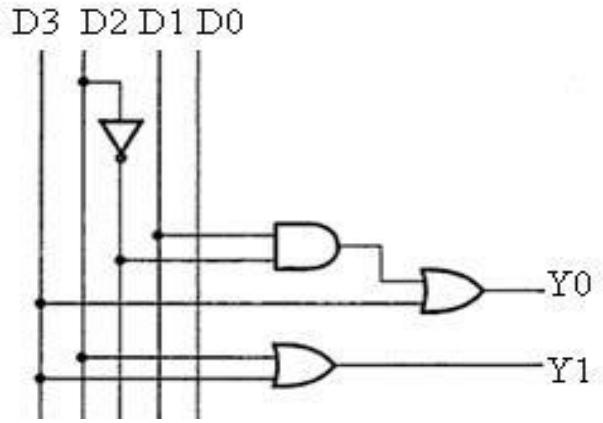


Fig. 3.16: Circuit diagram of priority encoder

3.3.2 Octal to Binary (8 to 3) Encoder

Table 3.7: Truth table of octal to binary encoder

Input								Output		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	Q_2	Q_1	Q_0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

- It has 8 input lines & 3 output lines.
- Corresponding to the eight input octal numbers we get three bit binary output.
- In encoders only one input will have a one value at any given time

Boolean Equation

$$Q_0 = D_1 + D_3 + D_5 + D_7$$
$$Q_1 = D_2 + D_3 + D_6 + D_7$$
$$Q_2 = D_4 + D_5 + D_6 + D_7$$

Circuit Diagram

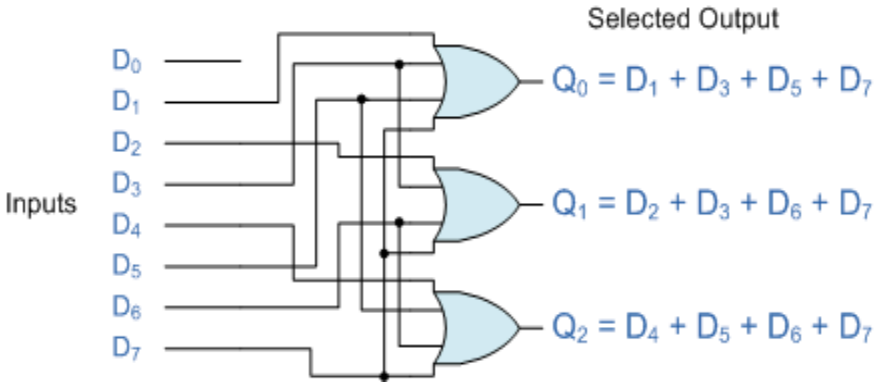


Fig. 3.17: Circuit diagram of 8 to 3 encoder

3.4 DECODER:

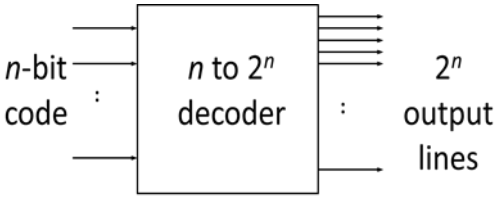


Fig. 3.18: Block diagram of decoder

- Decoder is a device which does the reverse operation of Encoder. It is a combinational circuit that converts binary information from 'n' input lines to a maximum of '2^n' unique output lines.
- Decoder is identical to a demultiplexer without any data input.
- E.g.: 2 to 4 Decoder, 3 to 8 Decoder, BCD to Seven Segment Decoder.

3.4.1 2 to 4 Line Decoder

- I_0 & I_1 are two inputs whereas y_3, y_2, y_1 & y_0 are four outputs.
- The truth table shows that each output is '1' for only a specific combination of inputs.

Block Diagram

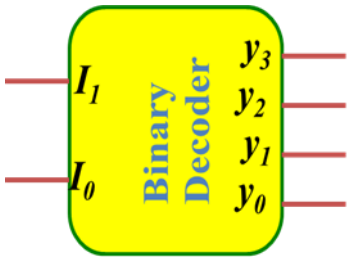


Fig. 3.19: Block diagram of 2 to 4 decoder

Truth Table

Table 3.8: Truth table of 2 to 4 decoder

Inputs		Output			
I_1	I_0	y_0	y_1	y_2	y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Working

- According to the truth table, when **I_1I_0=00**, the output Y_0 is set to '1', others are '0'
- When **I_1I_0=01**, the output Y_1 is set to '1', others are '0'
- Similarly, for other input combinations, particular output is set to '1' & others are '0'

Boolean Equation

$$y_0 = \overline{I_1}\,\overline{I_0};$$
$$y_1 = \overline{I_1} I_0;$$
$$y_2 = I_1 \overline{I_0};$$
$$y_3 = I_1 I_0$$

Circuit Diagram

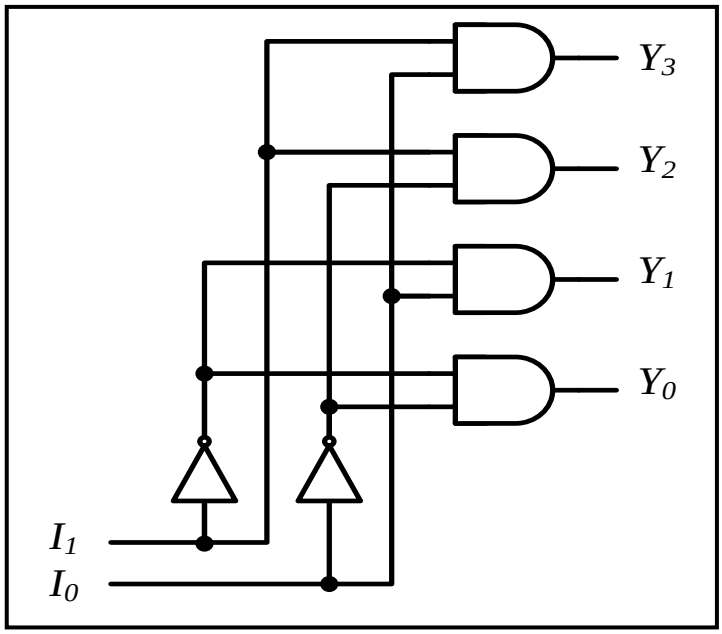


Fig. 3.20: Circuit diagram of 2 to 4 decoder

3.4.2 3 to 8 Decoder

Block Diagram

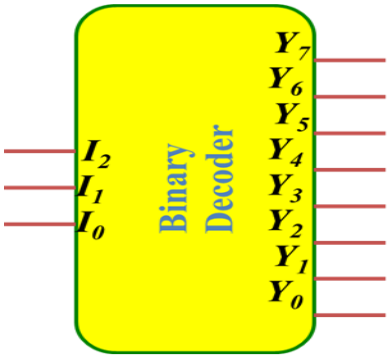


Fig. 3.21: Block diagram of 3 to 8 decoder

Truth Table

Table 3.9: Truth table of 3 to 8 decoder

Inputs			Output							
I_2	I_1	I_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Working

- According to the truth table, when **$I_2I_1I_0$=000**, the output Y_0 is set to '1', others are '0'
- When **$I_2I_1I_0$=001**, the output Y_1 is set to '1', others are '0'
- Similarly, for other input combinations, particular output is set to '1' & others are '0'

Boolean Equation

$$Y_0 = \bar{I}_2\bar{I}_1\bar{I}_0;\ Y_1 = \bar{I}_2\bar{I}_1 I_0;\ Y_2 = \bar{I}_2 I_1\bar{I}_0;\ Y_3 = \bar{I}_2 I_1 I_0$$

$$Y_4 = I_2\bar{I}_1\bar{I}_0;\ Y_5 = I_2\bar{I}_1 I_0;\ Y_6 = I_2 I_1\bar{I}_0;\ Y_7 = I_2 I_1 I_0$$

Circuit Diagram

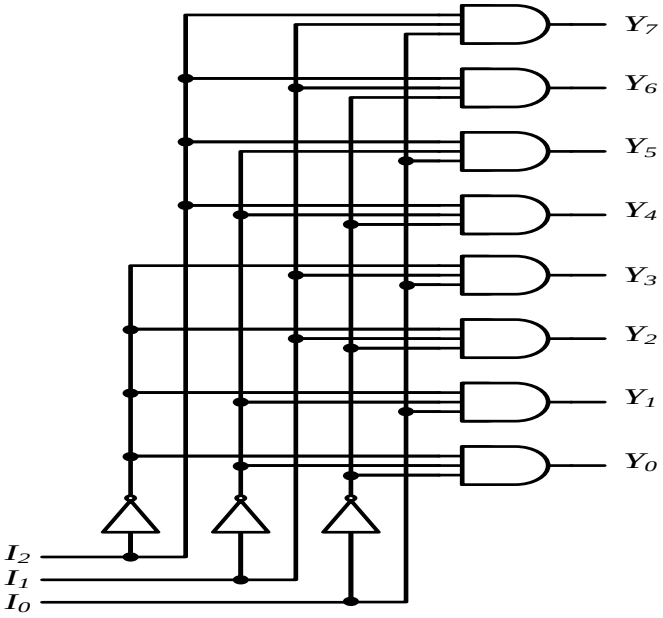


Fig. 3.22: Circuit diagram of 3 to 8 decoder

3.4.3 2 X 4 Decoder with Enable

Table 3.10: Truth table of 2 to 4 decoder

	Inputs		Output			
E	I_1	I_0	Y_0	Y_1	Y_2	Y_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

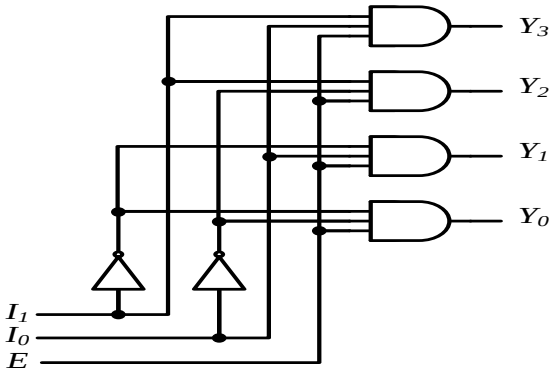
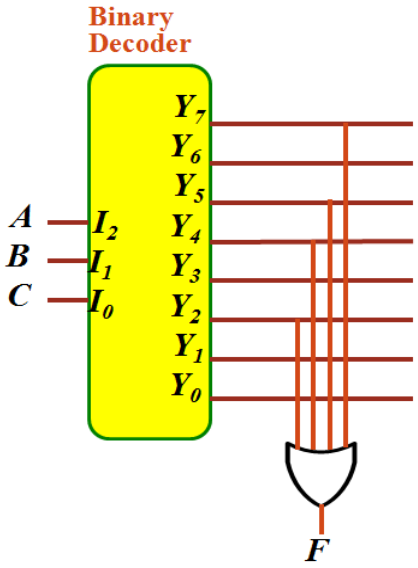


Fig. 3.23: Circuit diagram of 2 to 4 decoder with enable

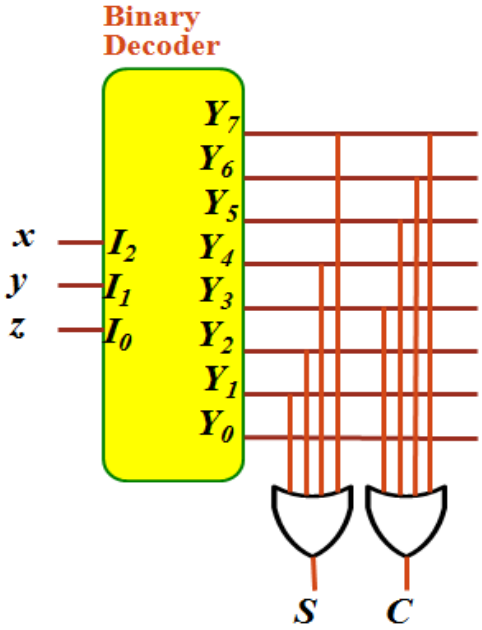
Example 1: Implement following function using decoder: F = Σ(2, 4, 5, 7).

- It is a 3 variable function. So, 3 to 8 decoder will be used to implement function F.
- The decoder produces minterms. The output Y_2, Y_4, Y_5 & Y_7 are ORed to produce the required output.

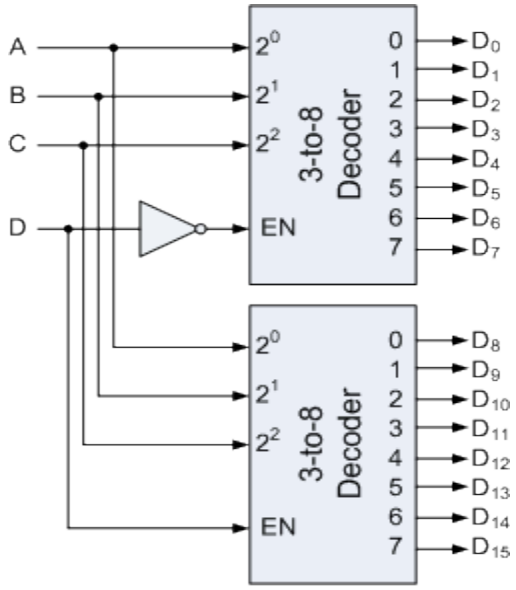


Example 2: Implement Full adder using decoder.

- Full adder has 3 inputs (x, y, z) & 2 outputs (Sum, Carry).
- So, 3 to 8 decoder will be used to implement output function Sum & Carry.
- Equation for Sum = Σ(1, 2, 4, 7) & Carry = Σ(3, 5, 6, 7).
- Connect the outputs of the decoder accordingly to OR gate for Sum & Carry.



Example 3: Design 4 to 16 decoder using 3 to 8 decoder.



3.5 MAGNITUDE COMPARATOR:

- It compares two numbers A & B.
- In process of comparison, it first compares MSB of input A to MSB of input B.
- If one of these bits is 1 and the other 0, the process is completed & the no. containing 1 as the MSB is identified as the largest number.
- If MSB of A equals the MSB of B, then the next most significant bits of A and B are compared.
- This process continues until a bit of one number differs from the corresponding bit of the other.

3.5.1 2-bit Magnitude Comparator

- There are two numbers A and B, each of two bits long.
- Magnitude comparator compares numerical values of these numbers.
- The result of comparison can be Equal (E), Greater than (G) or Less than (L).
- If A > B then E should be asserted to 1.
- If A = B then G should be asserted to 1.
- If A < B then L should be asserted to 1.

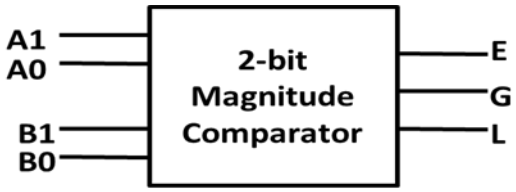


Fig. 3.24: block diagram of magnitude comparator

Truth Table

Table 3.11: Truth table of magnitude comparator

Inputs				Outputs		
A_1	A_0	B_1	B_0	G	E	L
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

Boolean Equation

For E,

A_1A_0 \ B_1B_0	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

$$E = (A_1B_1 + \overline{A_1}\,\overline{B_1})(A_0B_0 + \overline{A_0}\,\overline{B_0})$$

For L,

$A_1 A_0$ \ $B_1 B_0$	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

$$L = \overline{A_1}B_1 + \overline{A_1}\,\overline{A_0}B_0 + \overline{A_0}B_1B_0$$

For G,

$A_1 A_0$ \ $B_1 B_0$	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

$$G = A_1\overline{B_1} + A_0\,\overline{B_1}\,\overline{B_0} + A_1A_0\,\overline{B_0}$$

Fig. 3.25: K-map representation of magnitude comparator

Circuit Diagram

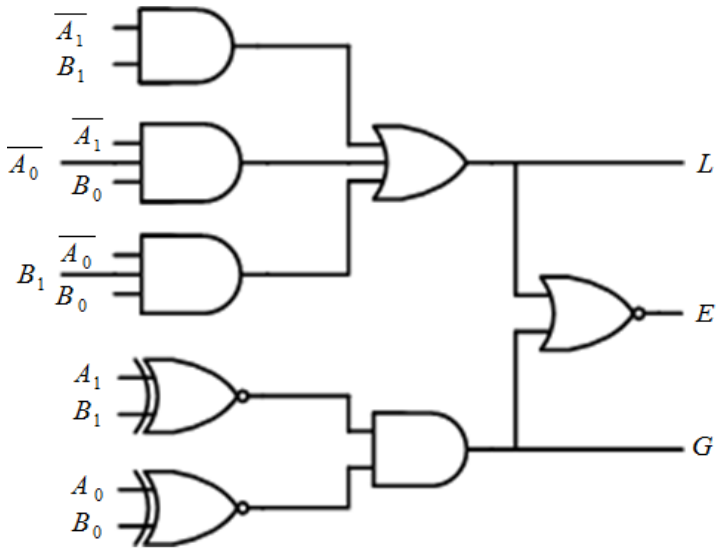


Fig. 3.26: Circuit diagram of magnitude comparator

3.5.2 4-bit Magnitude Comparator

The logic for a 4-bit magnitude comparator: Let the two 4-bit numbers be $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$.

1. If $A_3 = 1$ and $B_3 = 0$, then $A > B$. Or
2. If A_3 and B_3 coincide, and if $A_2 = 1$ and $B_2 = 0$, then $A > B$. Or
3. If A_3 and B_3 coincide, and if A_2 and B_2 coincide, and if $A_1 = 1$ and $B_1 = 0$, then $A > B$. Or
4. If A_3 and B_3 coincide, and if A_2 and B_2 coincide, and if A_1 and B_1 coincide, and if $A_0 = 1$ and $B_0 = 0$, then $A > B$.

From these statements, we see that the logic expression for $A > B$ can be written as

$$(A > B) : G = A_3\overline{B}_3 + (A_3 \odot B_3)A_2\overline{B}_2 + (A_3 \odot B_3)(A_2 \odot B_2)A_1\overline{B}_1 + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)A_0\overline{B}_0$$

Similarly, the logic expression for $A < B$ can be written as

$$(A < B): L = \overline{A}_3B_3 + (A_3 \odot B_3)\overline{A}_2B_2 + (A_3 \odot B_3)(A_2 \odot B_2)\overline{A}_1B_1 + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)\overline{A}_0B_0$$

If A_3 and B_3 coincide and if A_2 and B_2 coincide and if A_1 and B_1 coincide and if A_0 and B_0 coincide, then $A = B$.

So the expression for $A = B$ can be written as

$$(A = B): E = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$

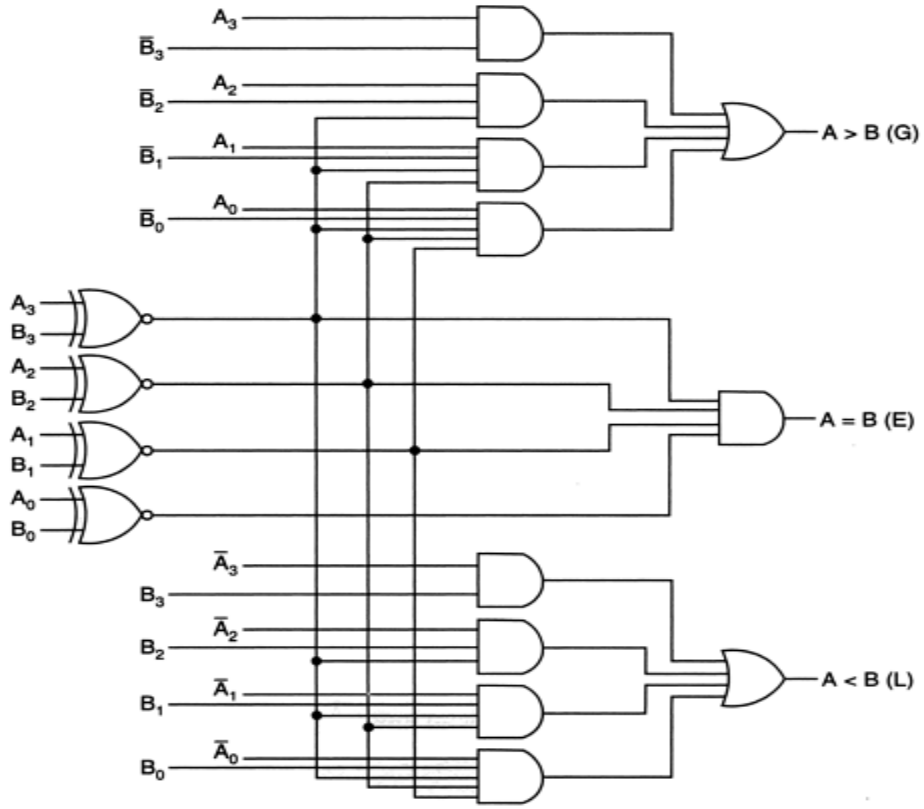


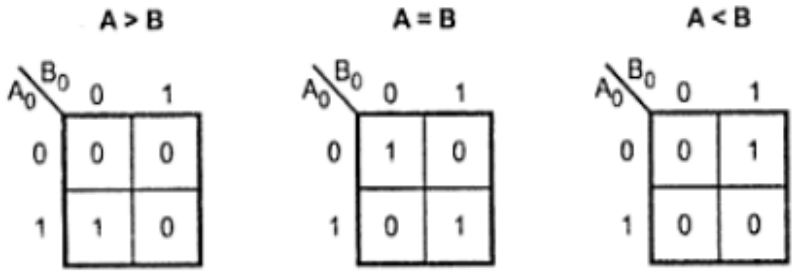
Fig. 3.27: Circuit diagram of 4 bit magnitude comparator

3.5.3 5-bit Magnitude Comparator

- Consider two numbers A = $A_0A_1A_2A_3A_4$, B = $B_0B_1B_2B_3B_4$.
- Four bit comparator IC is available i.e. 7485 which can compare four bits ($A_1A_2A_3A_4$ & $B_1B_2B_3B_4$). For remaining one bit (A_0 & B_0), the circuit can be designed using simple logic gates and can give as an input to the IC.
- The input to the IC has extra three inputs for I(A>B), I(A<B) & I(A=B).
- The circuit designed for one bit has three outputs E, G & L.

Table 3.12: Truth table of 5 bit magnitude comparator

A_0	B_0	I(A > B)	I(A = B)	I(A < B)
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0



Circuit Diagram

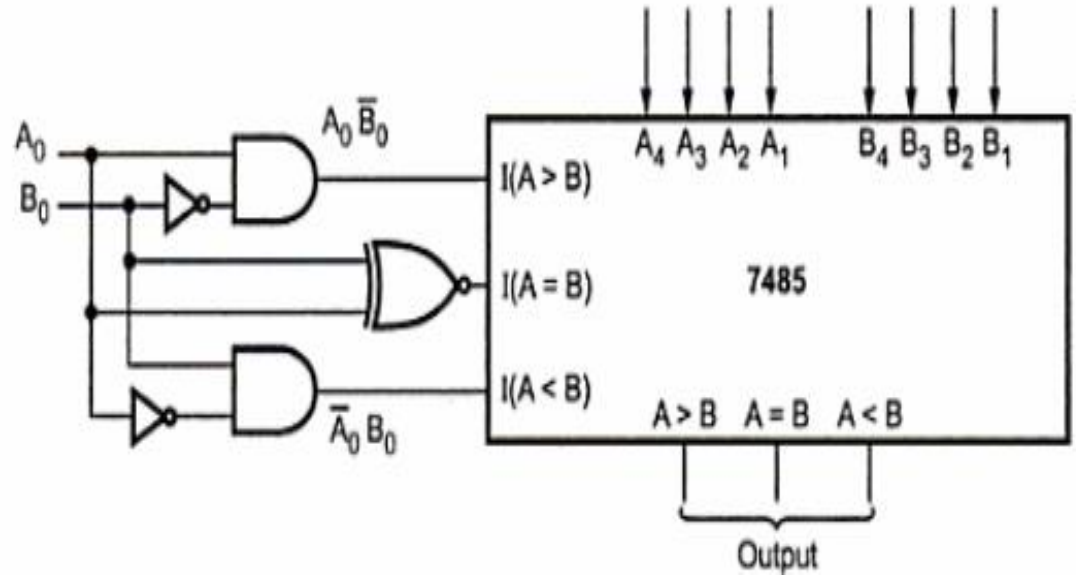


Fig. 3.28: Circuit diagram of 5 bit magnitude comparator

Example 1: Design 8-bit Magnitude Comparator using 4-bit Magnitude Comparator.

- There are two numbers each of 8 bits: $X = X_0X_1X_2X_3X_4X_5X_6X_7$, $Y = Y_0Y_1Y_2Y_3Y_4Y_5Y_6Y_7$.
- 4 bits out of 8 bits i.e. $X_0X_1X_2X_3$ & $Y_0Y_1Y_2Y_3$ are considered as input to first IC 7485. The result of which will be I(A>B), I(A<B) & I(A=B).
- These outputs are given to next cascaded IC 7485 as an input.
- Moreover, remaining 4 bits are compared i.e. $X_4X_5X_6X_7$ & $Y_4Y_5Y_6Y_7$.
- The result of this comparison will be the final comparison result of 8 bits.
- The connection of cascaded IC 7485 is as given below,

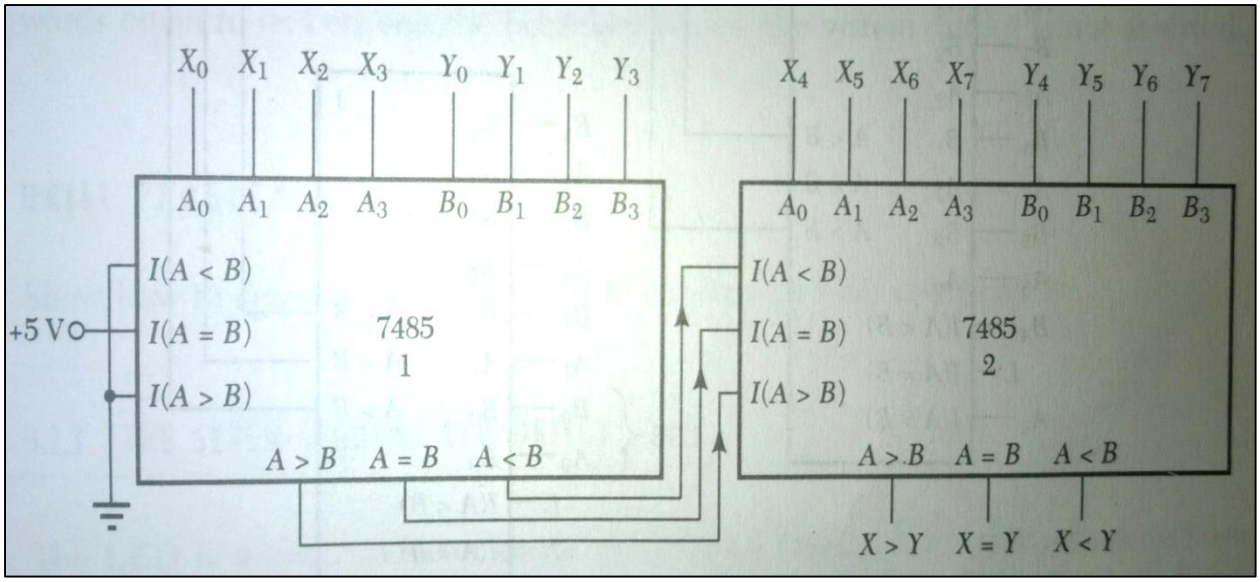


Fig. 3.29: Circuit diagram of 8 bit magnitude comparator using 4 bit bit magnitude comparator

3.6 PARITY BIT GENERATOR / CHECKER:

- Ex-OR functions are very useful in systems requiring error detection & correction codes.
- Binary data, when transmitted and processed, is susceptible to noise that can alter its 1s to 0s and 0s to 1s.
- To detect such errors, an additional bit called the *parity bit* is added to the data bits and the word containing the data bits and the parity bit is transmitted.
- At the receiving end the number of 1s in the word received are counted and the error, if any, is detected.
- This parity check detects only single bit errors.
- The circuit that generates the parity bit in the transmitter is called a parity generator.
- The circuit that checks the parity in the receiver is called parity checker.
- A parity bit, a 0 or a 1 is attached to the data bits such that the total number of 1s in the word is even for even parity and odd for odd parity.

- The parity bit can be attached to the code group either at the beginning or at the end depending on system design.
- A given system operates with either even or odd parity but not both. So, a word always contains either an even or an odd number of 1s.
- At the receiving end, if the word received has an even number of 1s in the odd parity system or an odd number of 1s in the even parity system, it implies that an error has occurred.
- In order to check or generate the proper parity bit in a given code word, the basic principle used is "the modulo sum of an even number of 1s is always a 1".
- Therefore, in order to check for an error, all the bits in the received word are added.
- If the modulo sum is a o for an odd parity system or a 1 for an even parity system, an error is detected.
- To generate an even parity, the four data bits are added using three X-OR gates. The sum bit will be the parity. Fig. (a) shows the logic diagram of an even parity generator.

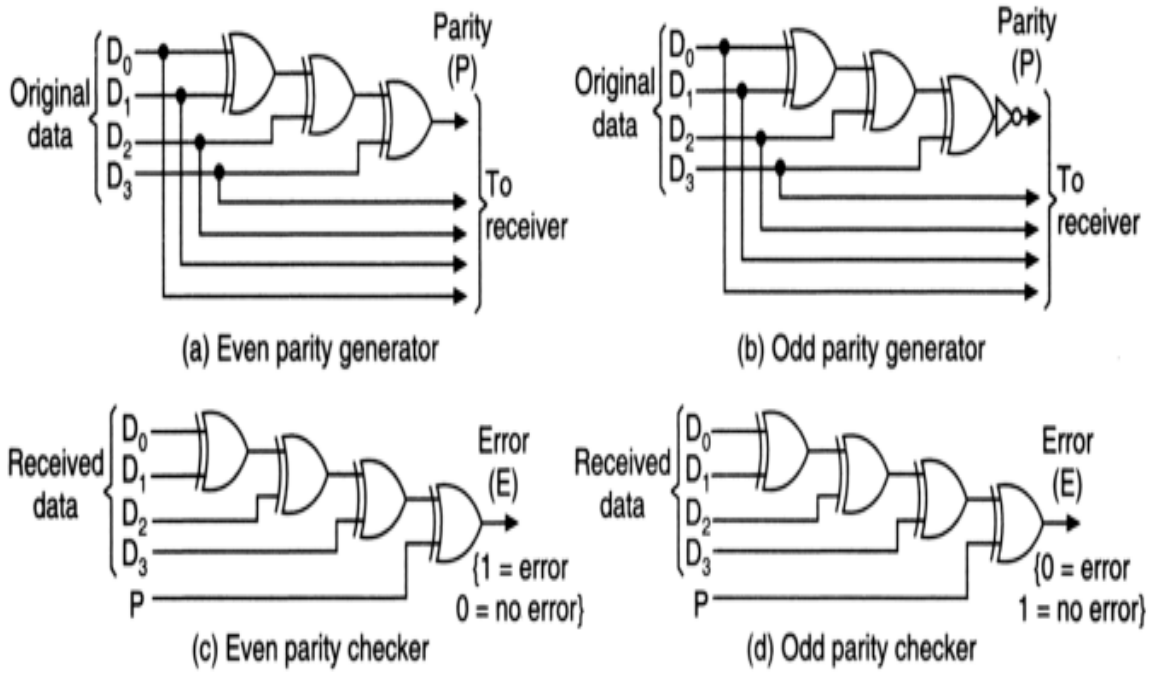


Fig. 3.30: Circuit diagram of parity bit generator / checker

- To generate an odd parity bit, the four data bits are added using three X-OR gates and the sum bit is inverter. Fig. (b) shows the logic diagram of an odd parity generator.
- Fig. (c) & (d) shows an even bit parity checker and an odd parity checker respectively

Example 1: Design Even Parity Bit Generator for 4-bit input.

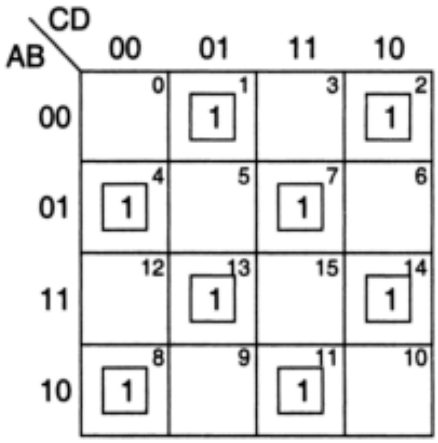
- Let the 4-bit input be A, B, C & D.
- For even parity, a parity bit 1 is added such that the total number of 1s in the 4-bit input and the parity bit together is even.

Truth Table

Table 3.13: Truth table of even parity bit generator for 4 bit input

4-bit data input				Output
A	B	C	D	parity bit (f)
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Boolean Equation



$$f = \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}BCD + AB\bar{C}D + ABC\bar{D} + A\bar{B}\bar{C}\bar{D} + A\bar{B}CD$$

$$= \bar{A}\bar{B}(C \oplus D) + \bar{A}B(\overline{C \oplus D}) + AB(C \oplus D) + A\bar{B}(\overline{C \oplus D})$$

$$= (C \oplus D)(\overline{A \oplus B}) + (\overline{C \oplus D})(A \oplus B)$$

$$= A \oplus B \oplus C \oplus D$$

Circuit Diagram

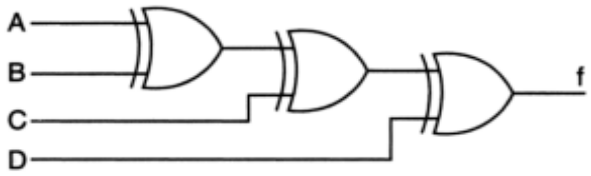


Fig. 3.31: Circuit diagram of even parity bit generator for 4 bit input

Example 2: Design Odd Parity Bit Generator for 4-bit input.

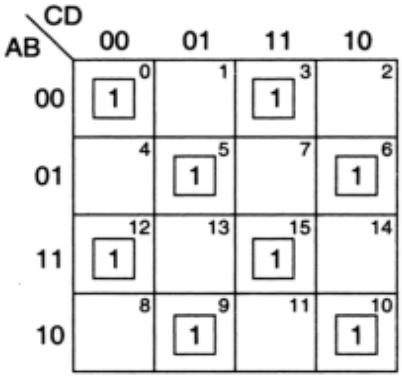
- An odd parity bit generator outputs a 1, when the number of 1s in the data bits is even, so that the total number of 1s in the data bits and parity bit together is odd.
- Since odd parity is the complement of even parity.
- So, an inverter is connected at the output of an even parity generator.
- The truth table, the K-map and the logic diagram for the odd parity generator are shown in figure below.
- From the K-map, it can be seen that no minimization is possible.
- If the expression for the parity bit is implemented as it is, 40 gate inputs are required.
- To reduce the cost, the expression may be manipulated in terms of X-OR gates and implemented.

Truth Table

Table 3.14: Truth table of odd parity bit generator for 4 bit input

4-bit data input				Output parity bit $\bar{f}$
A	B	C	D	
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Boolean Equation



$$f = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}CD + AB\bar{C}\bar{D} + ABCD + \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + A\bar{B}\bar{C}D + A\bar{B}C\bar{D}$$
$$= \bar{A}\bar{B}(\overline{C \oplus D}) + AB(\overline{C \oplus D}) + \bar{A}B(C \oplus D) + A\bar{B}(C \oplus D)$$
$$= (\overline{C \oplus D})(\overline{A \oplus B}) + (C \oplus D)(A \oplus B)$$
$$= \overline{(A \oplus B) \oplus (C \oplus D)}$$

Circuit Diagram

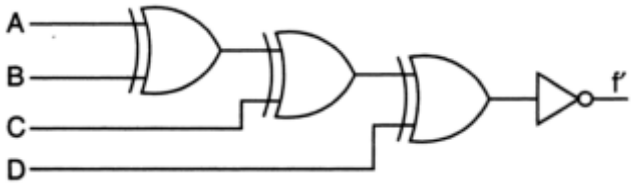


Fig. 3.32: Circuit diagram of odd parity bit generator for 4 bit input

4.1 Introduction

- **Combinational Circuit**
- A combinational circuit can be defined as a circuit whose output is dependent only on the inputs at the same instant of time. Half Adder, Full Adder, Half Subtractor, Full Subtractor are examples of combinational circuits.

- **Sequential Circuit**
- A sequential circuit can be defined as a circuit whose output depends not only on the present inputs but also on the past history of inputs.
- For a sequential circuit, the values of the variable are usually specified at certain discrete time instants rather than over the whole continuous time. Flip-Flops, Registers, Counters form the sequential circuit.
- A sequential circuit consists of combinational circuit and memory elements are connected to it to form a feedback path as shown in the block diagram below:

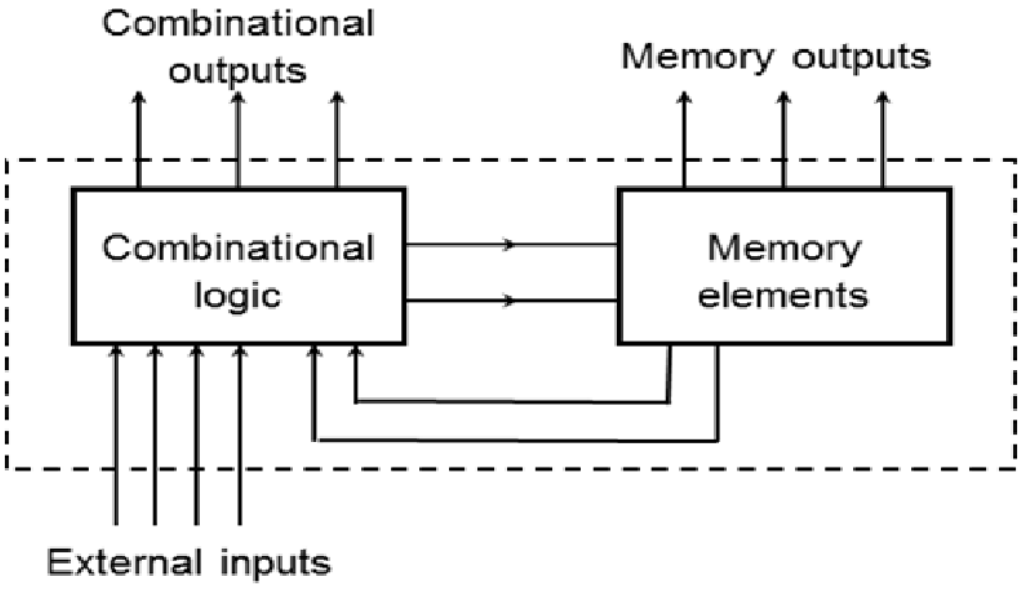


Figure 4.1 sequential circuit block diagram

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

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

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

  Further differences between **combinational** and **sequential** circuits can be listed as follows:

Combinational Circuit	Sequential Circuit
It does not contain memory elements.	It contains memory elements.
Output depends on present state of input only.	Output depends not only on the present inputs but also on the past history of inputs.
It's behavior is described by the set of output functions	It's behavior is described by the set of next-state (memory) functions and the set of output functions
Does not use Feedback path.	Use Feedback path.
Does not require clock signal.	Most of sequential circuit use clock signal.
Faster than sequential circuit.	Slower than combinational circuit.
Example: Adder, Subtractor, Multiplexers, Encoders, etc.	Flip Flops, Registers, Counters, etc.

4.2 Flip-Flops

- Memory element used in clocked sequential circuit is known as Flip Flop. It is capable of storing binary information. It is a **Binary cell** capable of storing one bit. A Flip Flop circuit can maintain a Binary state until directed by an input signal to switch into other state.

- It is also known as **Bi-stable Multivibrator**, because it has two stable states either 'Logic 0' or 'Logic 1'. It can be also known as **Latch**.

- The Flip Flop is made up of an assembly of Logic Gates. Even though, a Logic gate by itself has no storage capability, several gates can be connected in such a way that, permit the information to be stored.

- There are various types of Flip Flops, like SR Flip Flop, D Flip Flop, JK Flip Flop, T Flip Flop etc. but the major difference between various Flip Flops, are in the number of inputs they possess and the manner in which the input affect the binary states at the output.

4.2.1 Basic Flip Flop Circuit:

➢ **S-R Flip Flop (Set-Reset Flip Flop)**

- **Active High S-R Flip Flop:**

- It can be constructed using NOR gate. Which is also known as (S-R Latch) / (R-S Flip Flop) / (Set – Clear Latch)/ (S-C Latch).

- Figure 4.3 shows Basic Flip Flop circuit, upon which we can construct other type of Flip Flops. This type of Flip Flop is called direct coupled R-S Flip Flop. Here cross coupled connection constitutes the feedback path. This is an Asynchronous sequential circuit.

- S-R Flip Flop has two inputs SET and RESET as well as two outputs **Q** and $\acute{Q}$ OR **Q** and **Q'**. Both the outputs are complement of each other. For different combination of inputs, outputs are shown in Truth Table.

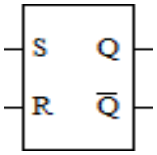
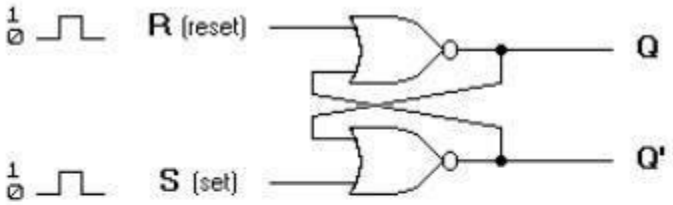


Figure 4.2 Symbol of S-R Flip Flop



(a) Logic diagram

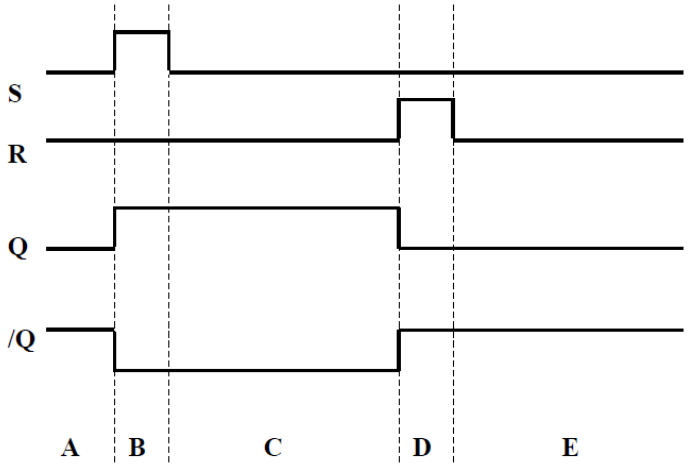
S	R	Q	Q'	
1	0	1	0	
0	0	1	0	(after S=1, R=0)
0	1	0	1	
0	0	0	1	(after S=0, R=1)
1	1	0	0	

(b) Truth table

Figure 4.3 Basic Flip-Flop Circuit with NOR Gates

- When S=1, R=0 then output is Q=1 (SET condition of Flip Flop)
- When S=0, R=1 then output is Q=0 (RESET condition of Flip Flop)
- When S=0, R=0 then output is $Q=Q_0$ (Previous condition of Flip Flop)/Memory State
- When S=1, R=1 then output is **Q=0**, and $\acute{Q}$ **=0** (Invalid condition of Flip Flop as Q and $\acute{Q}$ are always complement of each other)

- **Timing Diagram:**

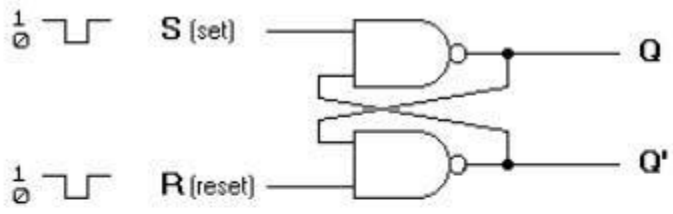


Region	S	R	Description	Q
A	0	0	Hold	0
B	1	0	Set Latch	1
C	0	0	Hold previous	1
D	0	1	Reset Latch	0
E	0	0	Hold previous	0

Figure 4.4 Timing Diagram of active high S-R Flip-Flop

- **Active Low S-R Flip Flop:**

  It can be constructed using NAND gate. Figure 4.5 shows Basic Flip Flop circuit, using NAND Gate.



(a) Logic diagram

S	R	Q	Q'	
1	0	0	1	
1	1	0	1	(after S=1, R=0)
0	1	1	0	
1	1	1	0	(after S=0, R=1)
0	0	1	1	

(b) Truth table

Figure 4.5 Basic Flip-Flop Circuit with NOR Gates

For Active Low RS Flip Flop,

- When S=1, R=0 then output is Q=0 (SET condition of Flip Flop)
- When S=0, R=1 then output is Q=1 (RESET condition of Flip Flop)
- When S=1, R=1 then output is $Q=Q_0$ (Previous condition of Flip Flop)/Memory State
- When S=0, R=0 then output is **Q=1**, and $\acute{Q}$ **=1** (Invalid condition of Flip Flop as Q and $\acute{Q}$ are always complement of each other)

Active low RS Flip Flop can be converted into Active High by connecting an Inverter at both the inputs OR by applying inverted inputs.

4.2.2 Types of Triggering:

- The output of a flip flop can be changed by bringing a small change in the input signal. This small change can be brought with the help of a clock pulse or commonly known as a trigger pulse. It is difficult to apply all the inputs simultaneously at the input terminal of flip flop, so to synchronize these inputs, a clock or triggering pulse is applied.

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

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

- There are mainly four types of pulse-triggering methods. They differ in the manner in which the Flip- Flop responds to the pulse.

1. High Level Triggering:

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

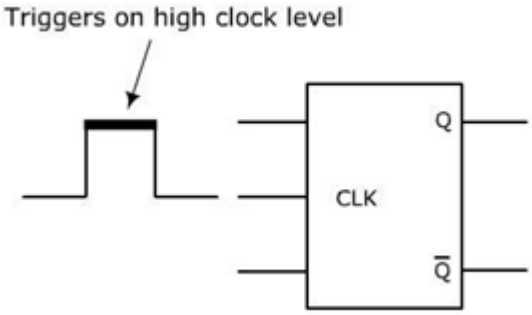


Figure 4.6 High Level Triggering

2. Low Level Triggering:

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

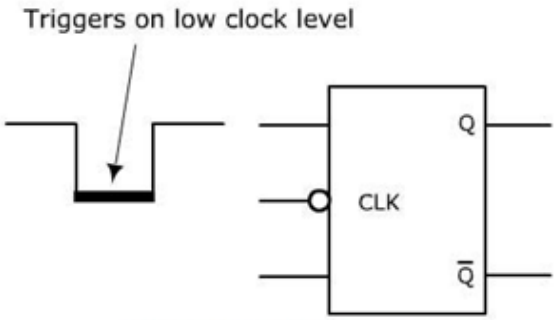


Figure 4.7 Low Level Triggering

3. Positive Edge Triggering:

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

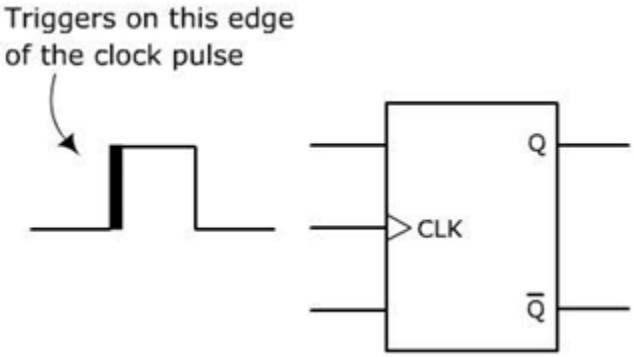


Figure 4.8 positive Edge Triggering

4. Negative Edge Triggering:

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

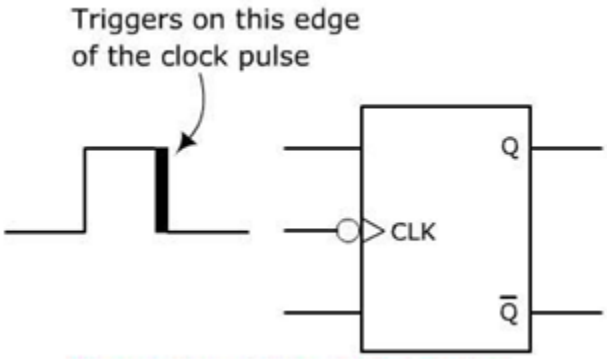


Figure 4.9 Negative Edge Triggering

4.2.3 Clocked RS Flip Flop:

- A Clocked RS Flip Flop requires a clocked pulse for synchronization purpose. It's S and R input can control the output only when ENABLE (clock) signal is high. It is also known as clocked SR latch or Synchronous SR latch.

- This type of Flip Flop responds to the changes in input only when clock is HIGH. It is also called LEVEL TRIGGERED Flip Flop.

Working of Clocked SR Flip-Flop:

- When S=1, R=0 then output is Q=1 which is SET condition of Flip Flop
- When S=0, R=1 then output is Q=0 which is RESET condition of Flip Flop
- When S=0, R=0 then output is $Q=Q_0$ which is Previous condition of Flip Flop. It is also considered as Memory State.
- When S=1, R=1 then output is **Q=0**, and $\acute{Q}$ **=0** which Invalid / indeterminate condition of Flip Flop as Q and $\acute{Q}$ are always complement of each other

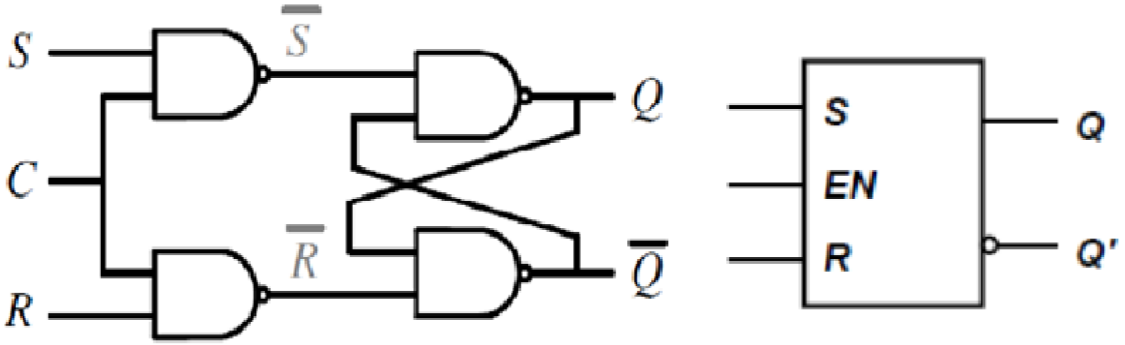


Figure 4.10 Circuit Diagram and Symbol of Clocked S-R Flip-Flop using NAND Gate

- Figure 4.10 shows the circuit diagram & symbol of clocked SR Flip Flop. Truth table is derived from the above circuit base on previous state. In truth table Q(t + 1) indicates the next state. Transition as per Truth table occurs only when clock pulse is HIGH i.e. Clock is ENABLE. The timing diagram of Level Triggered SR Flip Flop is shown in Figure 4.13.

Q	S	R	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	indeterminate
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	indeterminate

Figure 4.11 Truth Table Of Clocked S-R Flip-Flop

- K-Map derived from truth table is as follow with characteristic equation. S=1, R=1 is considered as don't care condition.

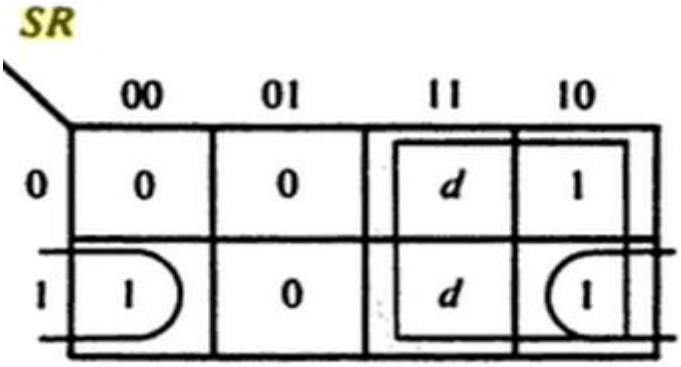


Figure 4.12 Characteristic Equation of S-R Flip-Flop

- Characteristic equation can be written as **Q (next state) = S + R'Q and SR = 0**

- **Timing Diagram:**

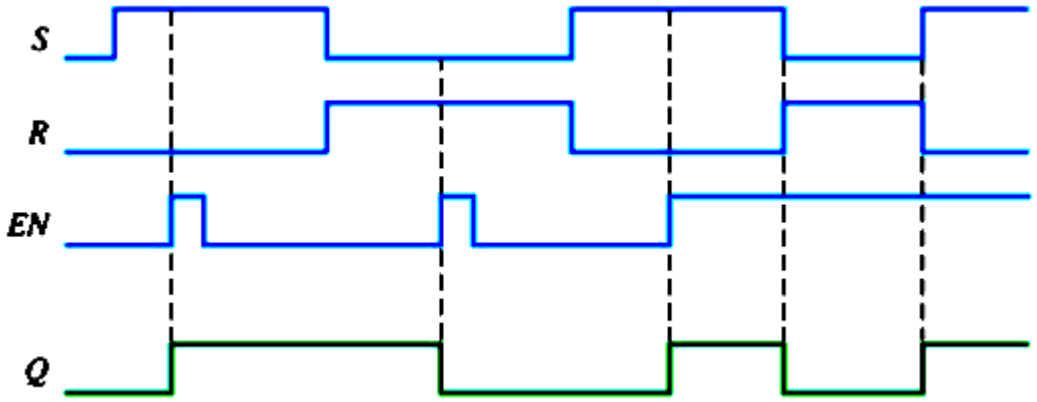


Figure 4.13 Timing Diagram of S-R Flip-Flop

- SR Flip Flop can be also derived using only NOR Gate as shown in Figure 4.14.

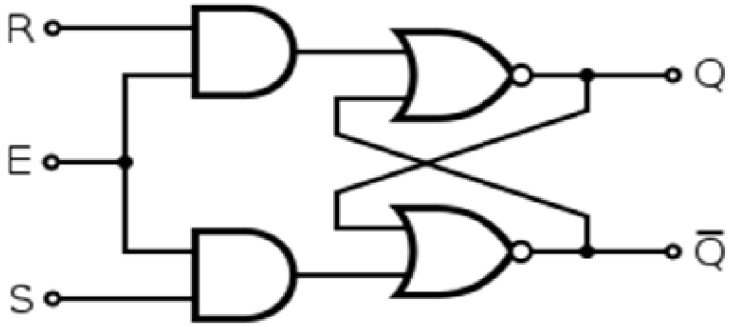


Figure 4.14 S-R Flip-Flop using NOR Gate

4.2.4 D-Flip Flop (Gated D Latch):

- D Flip Flop is a modification of Clocked SR Flip Flop. It is also known as gated D latch. The D Flip Flop receives designation 'D' from the ability to transfer 'Data' into a Flip Flop.

- It is basically a RS Flip Flop with an inverter at R Input. So number of input is only one in D Flip Flop.

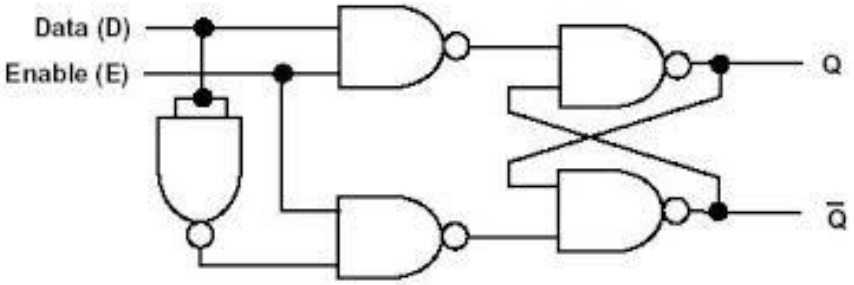
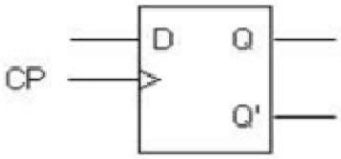


Figure 4.15 D Flip-Flop using NAND Gate



Graphical symbol

Q	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

Transition table

Figure 4.15 Symbol and Truth Table of D Flip-Flop

- **D Flip Flop : Gated D Latch using NOR Gate:**

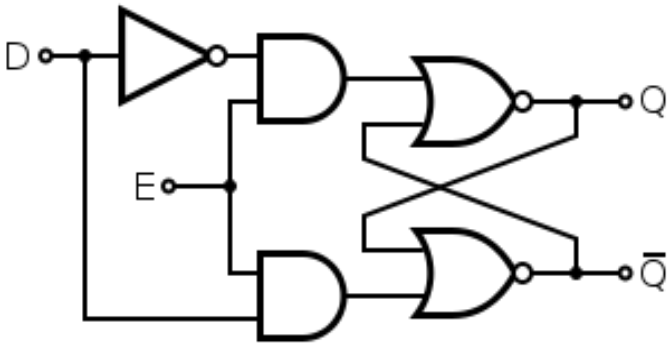


Figure 4.16 D Flip-Flop using NOR Gate

- D flip flop is actually a slight modification of the clocked SR flip-flop. From the figure you can see that the D input is connected to the S input and the complement of the D input is connected to the R input. The D input is passed on to the flip flop when the value of CP is '1'. When CP is HIGH, the flip flop moves to the SET state. If it is '0', the flip flop switches to the CLEAR state.

- **Timing Diagram:**

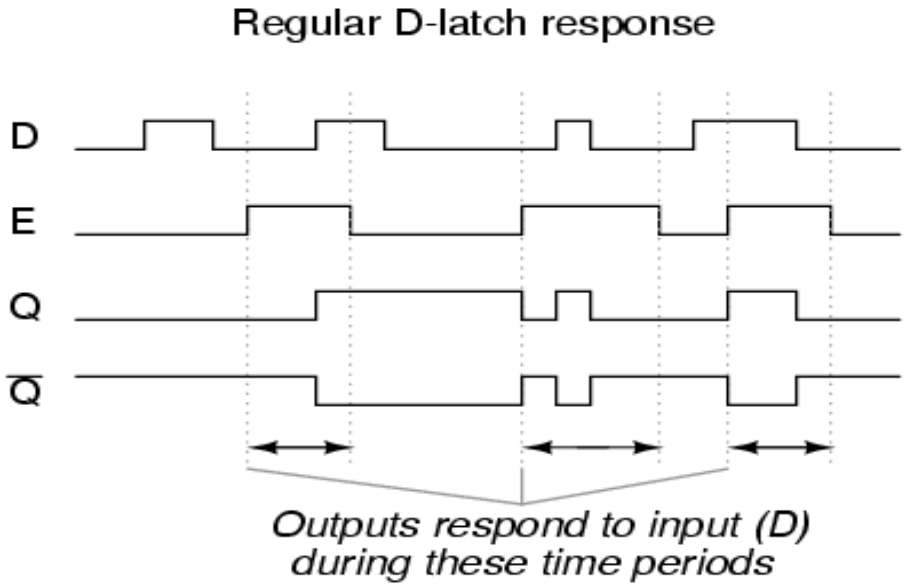


Figure 4.17 Timing Diagram for Level Triggered D Flip Flop

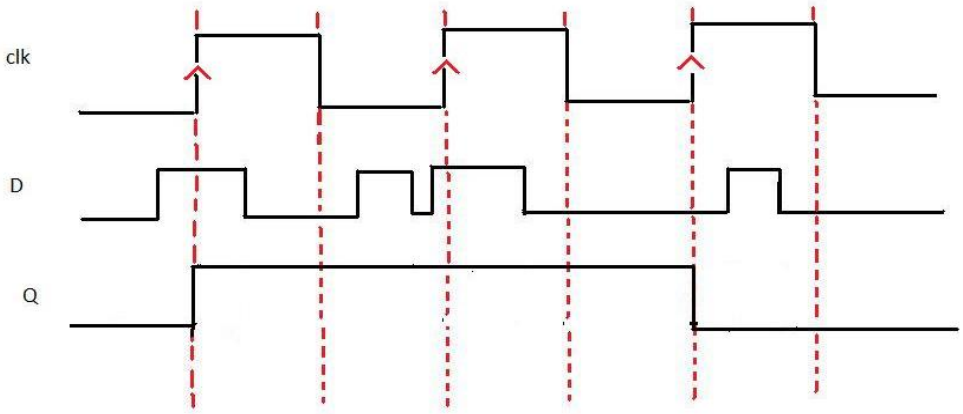


Figure 4.18 Timing Diagram for Positive Edge Triggered D Flip Flop

- **Characteristic Equation of D Flip Flop:**

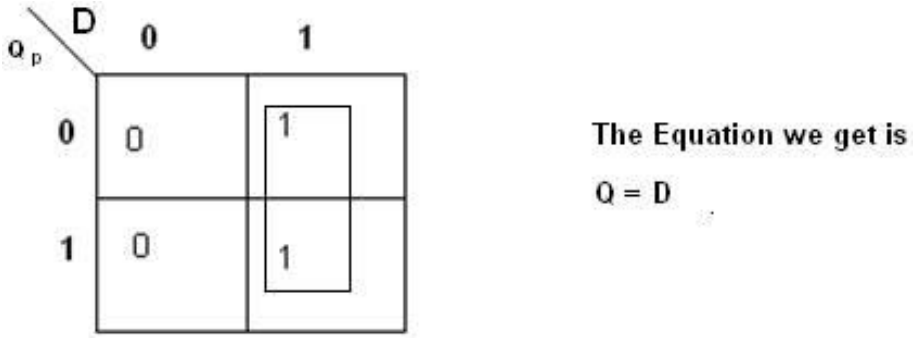


Figure 4.19 Characteristic Equation of D Flip Flop

4.2.5 J-K Flip-Flop:

- A JK flip-flop has two inputs similar to that of RS Flip-Flop. We can say JK flip-flop is a refinement of RS flip-flop. JK means Jack Kilby, a Texas instrument engineer who invented IC. The two inputs of JK Flip-flop is J (set) and K (reset). A JK flip-flop is nothing but a RS flip-flop along with two AND gates which are augmented to it.

- The flip-flop is constructed in such a way that the output Q is ANDed with K and CP. This arrangement is made so that the flip-flop is cleared during a clock pulse only if Q was previously 1. Similarly Q' is ANDed with J and CP, so that the flip-flop is cleared during a clock pulse only if Q' was previously 1.

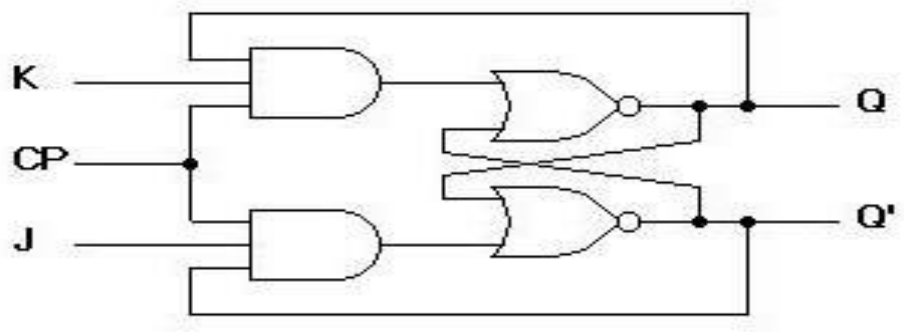
Working of JK Flip Flop:

- **When J=K=0**
  When both J and K are 0, the clock pulse has no effect on the output and the output of the flip-flop is the same as its previous value. This is because when both the J and K are 0, the output of their respective AND gate becomes 0.

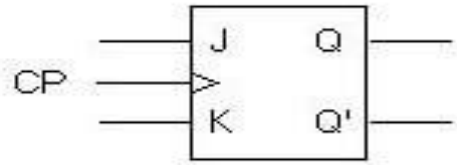
- **When J=0, K=1**
  When J=0, the output of the AND gate corresponding to J becomes 0 (i.e.) S=0 and R=1. Therefore Q' becomes 0. This condition will reset the flip-flop. This represents the RESET state of Flip-flop.

- **When J=1, K=0**
  In this case, the AND gate corresponding to K becomes 0(i.e.) S=1 and R=0. Therefore Q becomes 0. This condition will set the Flip-flop. This represents the SET state of Flip-flop.

- **When J=K=1**
  Consider the condition of CP=1 and J=K=1.
- When both the inputs J and K have a HIGH state, the flip-flop switches to the complement state. So, for a value of Q = 1, it switches to Q=0 and for a value of Q = 0, it switches to Q=1.



(a) Logic diagram



(b) Graphical symbol

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

(c) Transition table

Clocked JK flip-flop

Figure 4.20 circuit diagram of J-K Flip-Flop using NOR Gate

- This will cause the output to complement again and again. This complement operation continues until the Clock pulse goes back to 0. Since this condition is undesirable, we have to find a way to eliminate this condition. This can be avoided by setting time duration of pulse lesser than the propagation delay through the flip-flop. But it very difficult to generate the pulse with duration less than propagation delay time. So the restriction on the pulse width can be eliminated with a master-slave JK Flip Flop or Edge-triggered JK Flip Flop.

- **J-K Flip Flop using NAND Gate:**

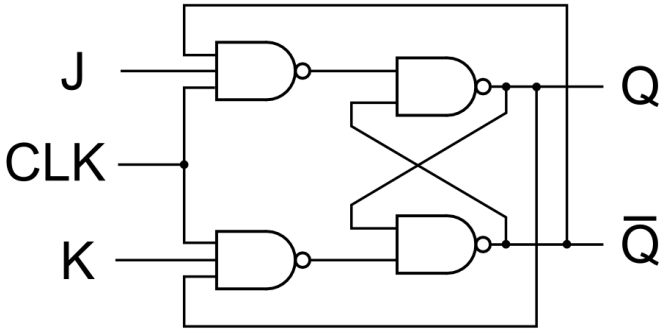


Figure 4.21 circuit diagram of J-K Flip-Flop using NAND Gate

- **Timing Diagram for positive Edge Triggered JK Flip Flop:**

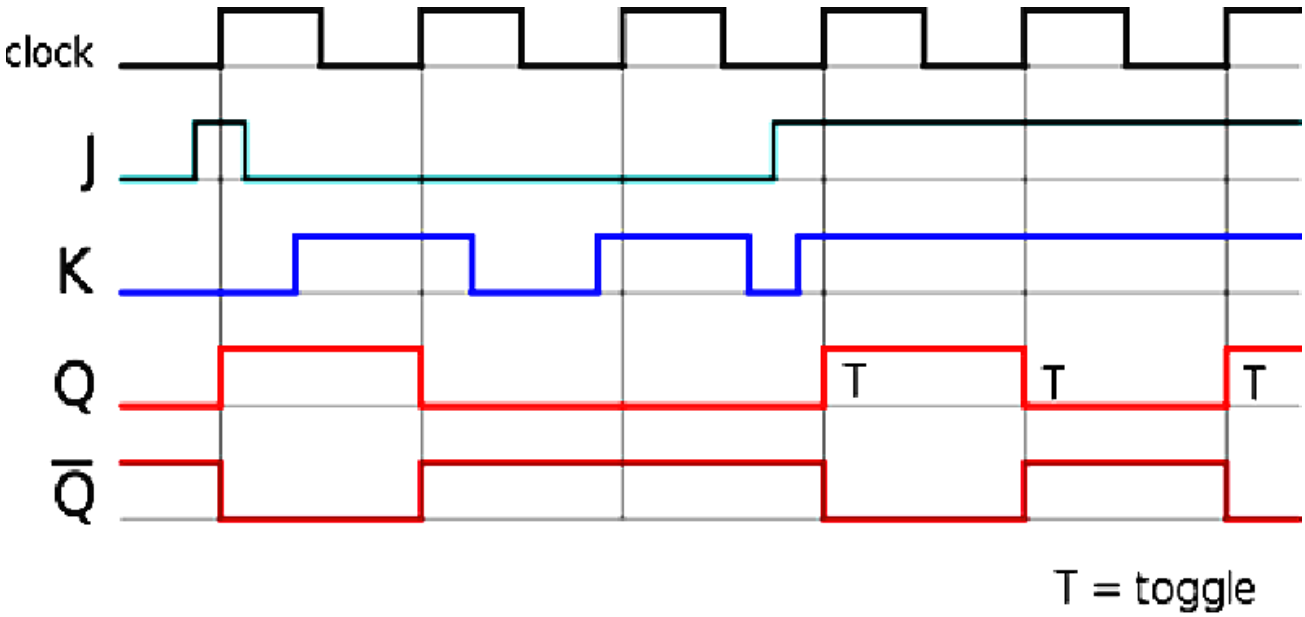
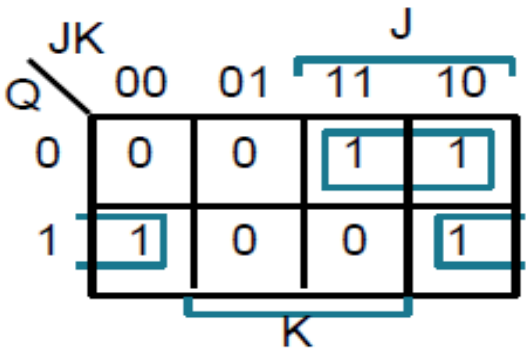


Figure 4.22 Timing Diagram of Positive Edge Triggered J-K Flip-Flop

- **Characteristic Equation for JK Flip Flop:**
  From the truth table, characteristic equation can be derived as follows using K-Map.



$$Q_{next} = J\overline{Q} + \overline{K}Q$$

Figure 4.23 Characteristic Equation for J-K Flip Flop

- **Race around condition:**

- In JK Flip Flop when J=K=1 and Clock pulse is also HIGH i.e CP=1, the flip-flop switches to the complement state. So, for a previous value of Q = 1, it switches to Q=0 and for a previous value of Q = 0, it switches to Q=1.

- This will cause the output to complement again and again. It means that output toggles between 1 and 0 when J=K=CP=1. This complement operation continues until the Clock pulse goes back to 0. This situation is called Race Around condition. Since this condition is undesirable, we have to find a way to eliminate this condition.

- This can be avoided by setting time duration of pulse lesser than the propagation delay through the flip-flop. But it very difficult to generate the pulse with duration less than propagation delay time. So the restriction on the pulse width can be eliminated with a master-slave JK Flip Flop or Edge-triggered JK Flip Flop.

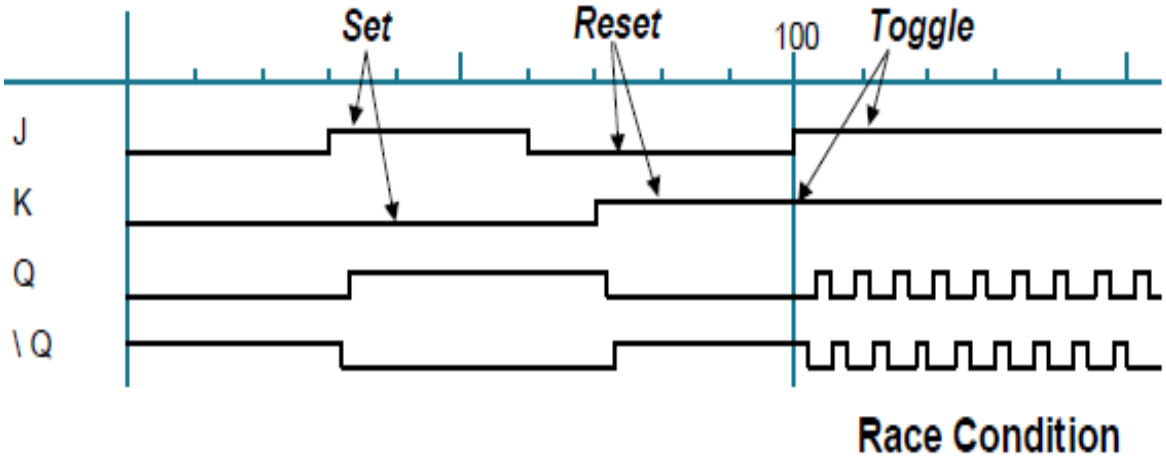


Figure 4.24 Race Around Condition in J-K Flip-Flop

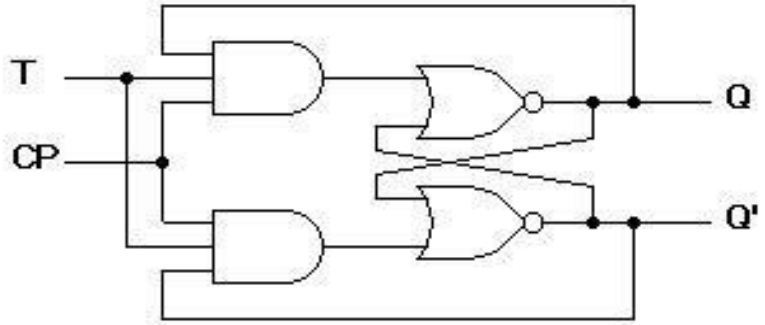
4.2.6 T-Flip Flop: Toggle Flip Flop

- T flip-flops are similar to JK flip-flops. T flip-flops are single input version of JK flip-flops. This modified form of JK flip-flop is obtained by connecting both inputs J and K together. This flip-flop has only one input along with Clock pulse. These flip-flops are called T flip-flops because of their ability to complement its state (i.e.) Toggle. So they are called as Toggle flip-flop.

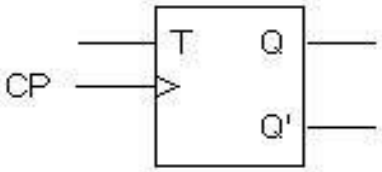
- **When J=K=0, then T=0**
  If CP=1 the output is same as previous state. It is Memory state of Flip Flop.

- **When J=K=1, then T=1**
  If Clock pulse is high (CP=1) then, the output begins to toggle. So, for a previous value of Q = 1, it switches to Q=0 and for a previous value of Q = 0, it switches to Q=1.

- Here also the restriction on the pulse width can be eliminated with a master-slave or edge-triggered construction. Take a look at the circuit and truth table given for T Flip Flop.



(a) Logic diagram



(b) Graphical symbol

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

(c) Transition table

Clocked T flip-flop

Figure 4.25 circuit diagram of T Flip-Flop using NOR Gate

- **Characteristic Equation for T Flip Flop:**
  From the truth table, characteristic equation can be derived as follows using K-Map

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

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

Figure 4.26 Characteristic Equation for T Flip Flop

4.2.7 Master Slave Flip Flop:

- Master-slave flip flop is designed using two separate flip flops. Out of these, one acts as the **Master** and the other as a **Slave.**

- In previous exercise, we discussed the JK flip-flop and its undesirable operation. When J=1 and K=1, there is a restriction on pulse width due to toggle operation (where output complements again and again until clock pulse goes to 0). This restriction can be overcome by using the Master-Slave Flip-flop.

- A master-slave flip-flop is normally constructed from two flip-flops, one is the Master flip-flop and the other is the Slave. 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 CP is given to the slave flip-flop. For example, if the CP=0 for a master flip-flop, then the output of the inverter is 1, and this value is assigned to the slave flip-flop. In other words if CP=0 for a master flip-flop, then CP=1 for a slave flip-flop.

- A master-slave flip flop can be constructed using any type of flip-flop which forms a combination with a clocked RS flip-flop, and with an inverter as slave circuit. Block diagram of Master Slave Flip Flop is as shown in Figure 4.27.

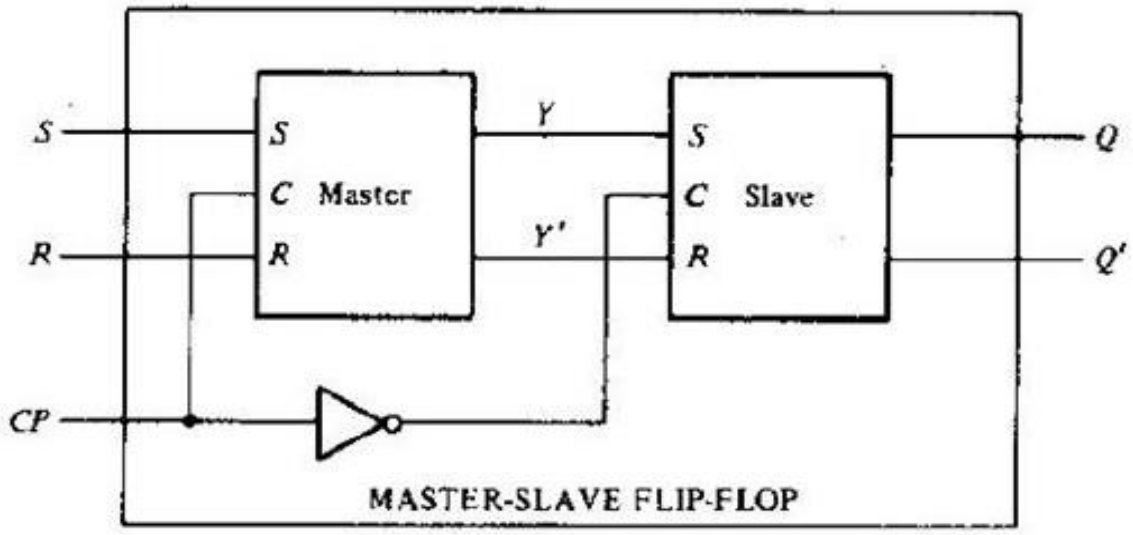


Figure 4.27 Block Diagram of Master Slave using S-R Flip Flop

- **Working of Master Slave Flip Flop:**

- **When CP=0**
  Now when CP=0, the master flip-flop is disabled. So the external inputs R and S of the master flip-flop will not affect the circuit until CP goes to 1. The inverter output goes to 1 and it enables the slave flip-flop. The output Q=Y and Q'=Y'.

- **When CP=1**
  When CP=1, the master flip-flop is enabled and the slave flip-flop remains isolated from the circuit until CP goes back to 0. Now Y and Y' depends on the external inputs R and S of the master flip-flop.

- Assume that the flip-flop is in a clear state and no clock pulse is applied to the circuit. The external inputs given are S=1 and R=0. This input will not affect the state of the system until the CP=1. Now the next clock pulse applied should change the state to SET state (S=1, R=0).

- During the clock pulse transition from 0 to 1, the master flip-flop goes to set state and changes the output Y to 1. However this does not affect the output of the system since the slave flip-flop is isolated from the system (CP=0 for slave). So no change is observed at the output of the system.

- When the CP returns to 0, the master flip-flop is disabled while the slave is enabled. So the information from the master is allowed to pass through to the slave. Since Y=1, this changes the output Q to 1.

  From this behavior of the master slave flip-flop it is quite clear that the state change in flip-flops coincide with the negative edge transition of the pulse.

- **Master Slave J-K Flip Flop using NAND Gate:**

- A JK master slave flip-flop can be constructed using NAND gates. It is constructed using two flip-flops. Gate I to 4 represents master flip-flop and gate 5 to 8 represents slave flip-flop. Gate 9 represents the inverter

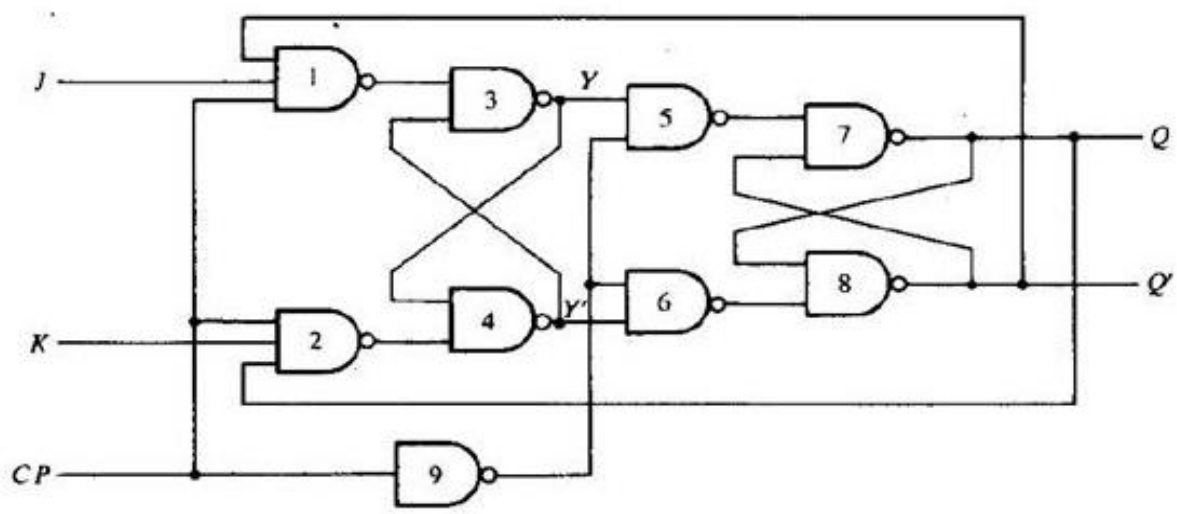


Figure 4.28 Master Slave Flip Flop Using NAND Gates

- During the positive edge of the flip-flop, the information from external inputs J and K is passed through to the master flip-flop. It is held there until the negative edge transition of the flip-flop occurs. Then the information is passed to the slave, clocked RS flip-flop, and the output is observed.

- Usually the clock pulse is 0 at start, so the values of J and K will not affect the state of the system, and the master flip-flop is isolated. But the output of gate 9 is 1, hence the slave flip-flop provides the output Q=Y and Q'=Y'. 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.

- **Timing Diagram of Master Slave Flip Flop:**

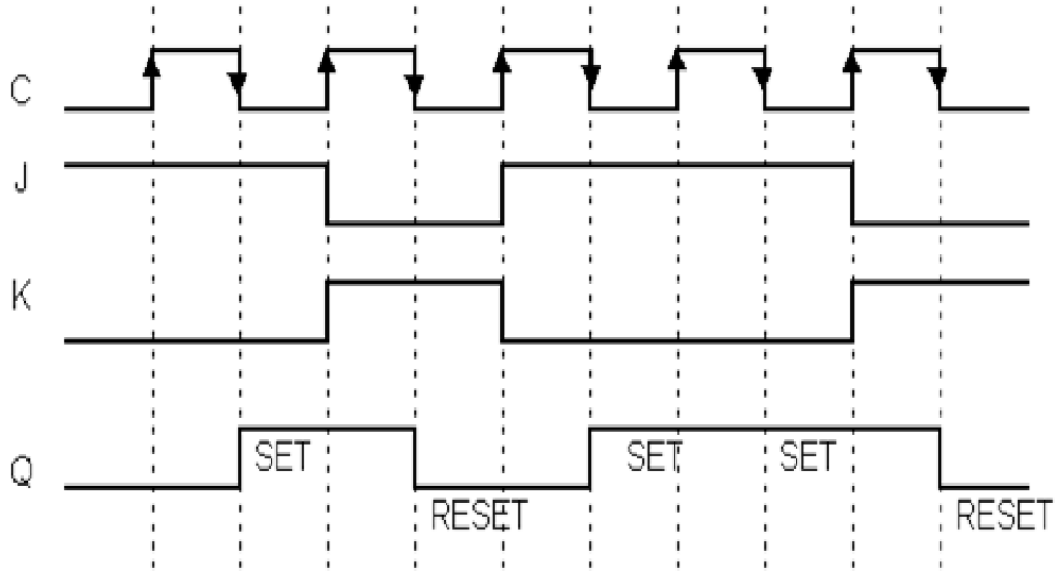


Figure 4.29 Timing Diagram of Master Slave Flip Flop

From the timing diagram of Master Slave Flip Flop it has been observed that Output Q actually makes the transition only when trailing edge of clock pulse arrives.

4.2.8 Preset & Clear Control Signal

- In Flip Flop when power is switched ON, the state of the circuit is uncertain, it may be Q=1 or Q=0.

- In many applications it is desired to initially SET or RESET the Flip Flop i.e. initial state of the flip flop is to be assigned. This is accomplished by direct or asynchronous inputs, referred as Preset (PRE) and Clear (CLR) inputs. These inputs are not in synchronism with main inputs.

- Figure 4.30 shows D flip-flop with Preset and Clear. If Preset and Clear is 1 then circuit operates in accordance with the Truth table of D Flip Flop. It has no effect on the performance of Flip Flop.

- If **Preset = 0 and Clear = 1** then Q =1, which SET the Flip Flop irrespective of main inputs, prior to its working/operation, hence the name given Preset. Once initial known state is obtained, Preset is kept now in Preset=1 condition for normal operation of Flip Flop. So Preset SET the initial condition prior to the working of Flip Flop with PRE=0.

- If **Preset = 1 and Clear = 0** then Q =0, which RESET the Flip Flop irrespective of main inputs, prior to its working/operation, hence the name given Clear. Once initial known state is obtained, Clear is kept now in Clear=1 condition for normal operation of Flip Flop. So Clear RESET the initial condition prior to the working of Flip Flop with CLR=0.

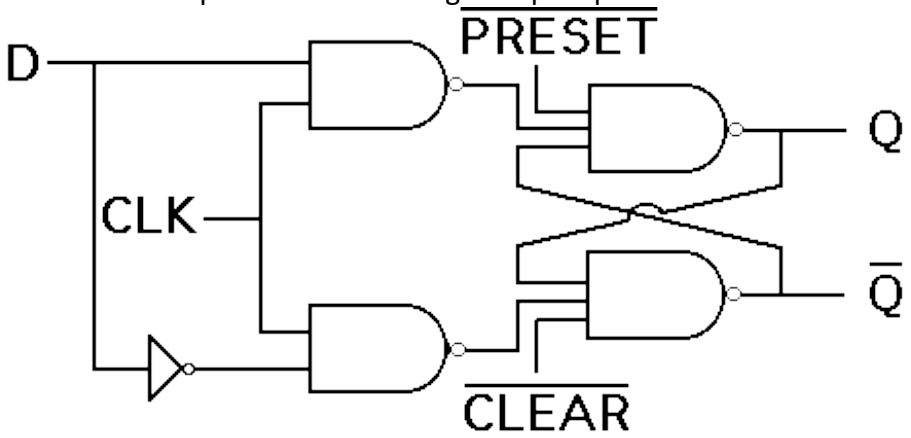


Figure 4.30 D Flip Flop with Preset and Clear

- Once the state of the flip Flop is established asynchronously, the direct inputs Preset and Clear must be connected to 1, before next clock or main inputs are applied. Figure 4.30 shows logic symbol of Flip Flop with Preset and Clear. The bubble used with Preset and Clear indicates that these are Active Low control signal i.e. its function will be performed when it is in low state 0.

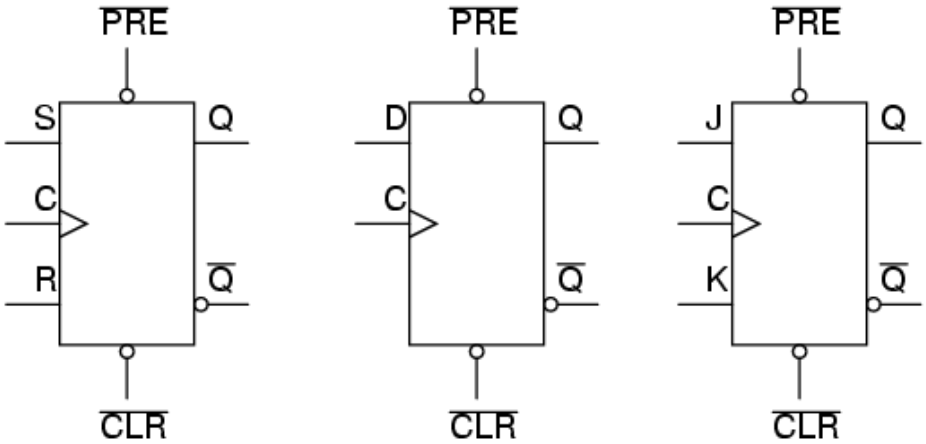


Figure 4.31 Symbol of Flip Flops with Preset and Clear signal

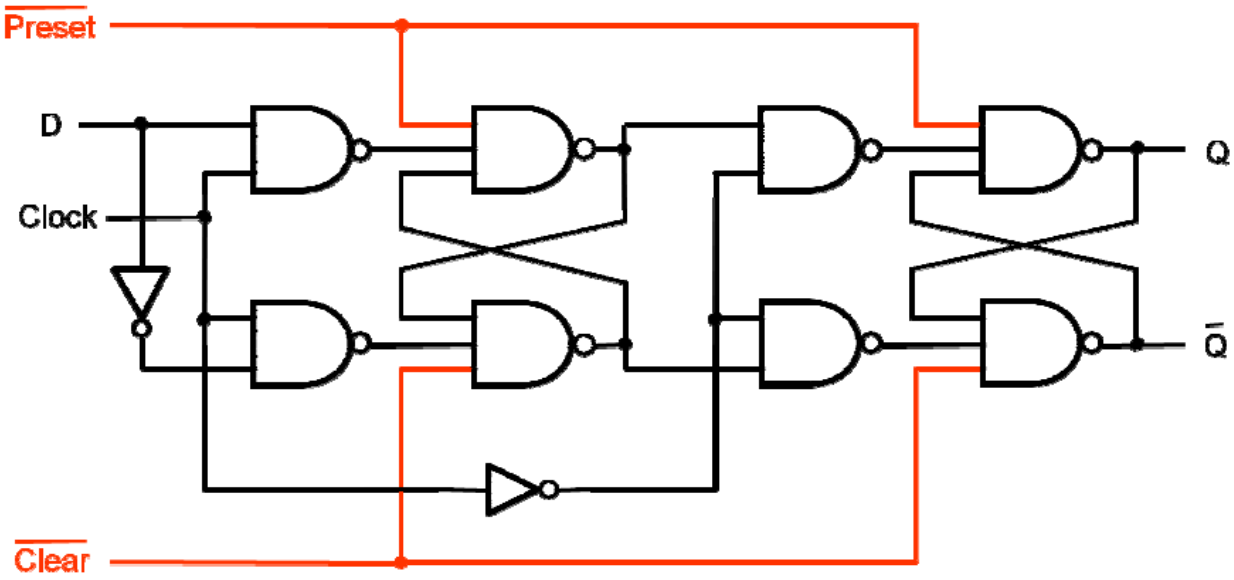


Figure 4.32 Master Slave Flip Flops with Preset and Clear signal

- **Some Important Terms related with Sequential Circuit:**

1) Propagation Delay Time:

The interval of time required after an input signal has been applied for the resulting output change to occur.

Different signals take different paths through the logic gates. Four categories of propagation delay:

a) Propagation delay for low to high transition of the output.

b) Propagation delay for high to low transition of the output.

c) Propagation delay measured from leading edge of preset input to low to high transition of the output.

d) Propagation delay measured from the leading edge of the clear input to the high to low transition of the output.

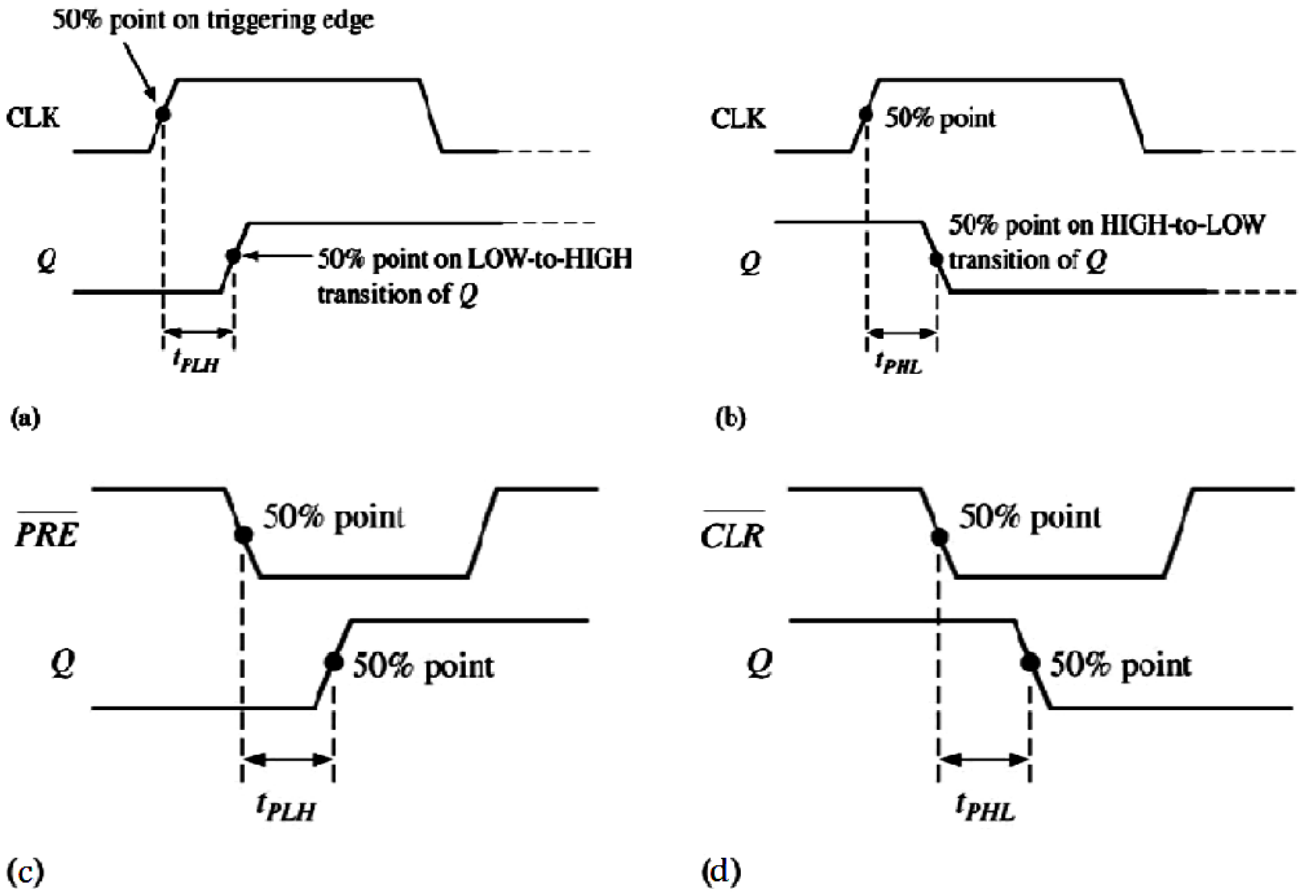


Figure 4.33 Propagation Delay Time

2) Set-up Time:

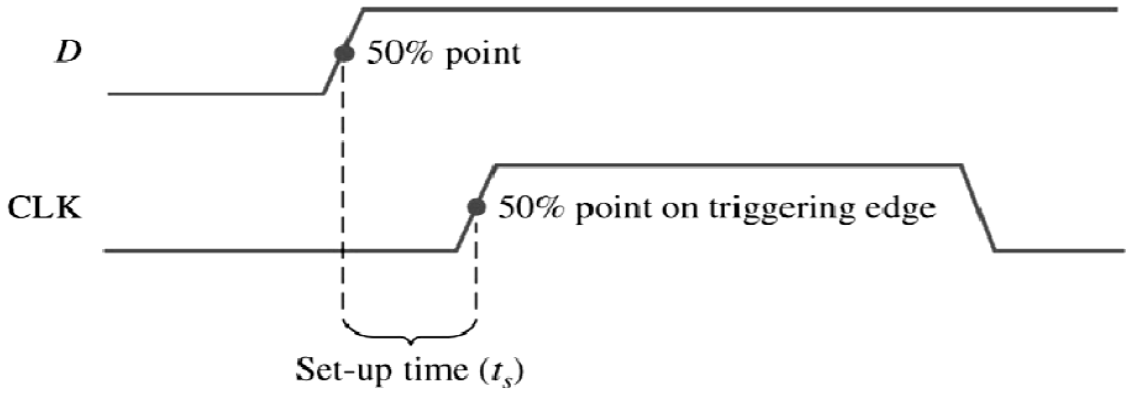


Figure 4.34 Set-up Time

The minimum amount of time required for the logic levels to be maintained constantly on the inputs (J and K, or S and R, or D) prior to the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip-flop.

3) Hold Time:

The minimum amount of time required for the logic levels to remain on the inputs after the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip-flop.

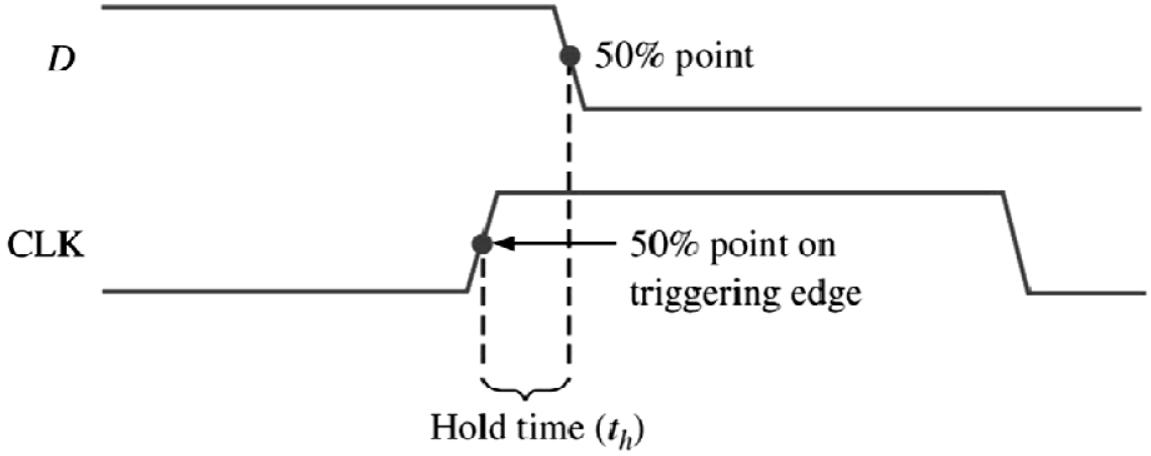


Figure 4.35 Hold Time

4.2.9 Conversion of Flip-Flops:

- To convert one type of flip-flop into another type, a combinational circuit is designed such that if the inputs of the required flip-flop (along with the outputs of the actual flip-flop if required) are fed as inputs to the combinational circuit and the output of the combinational circuit is connected to the inputs of the actual flip-flop, then the output of the actual flip-flop is the output of the required flip-flop.

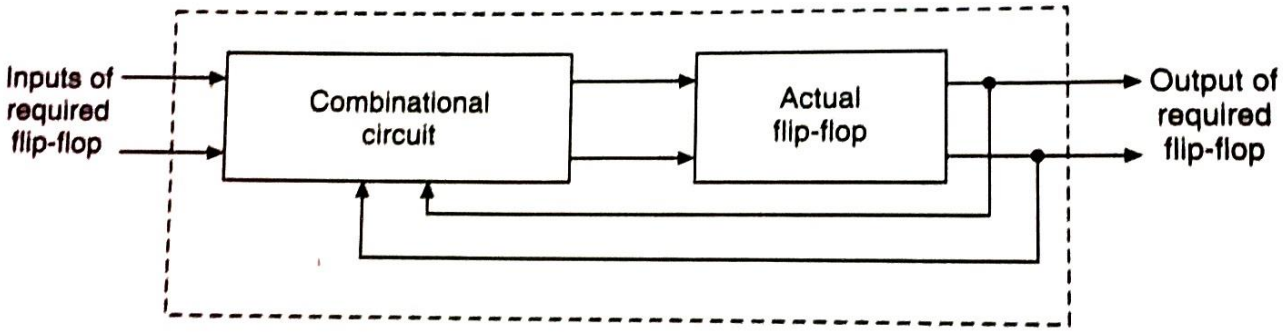


Figure 4.36 block diagram of conversion of flip-flop

- In other words, it means that, to convert one type of flip-flop into another type, we have to obtain the expressions for the inputs of the existing flip-flop in terms of the inputs of the required flip-flop and the present state variables of the existing flip-flop and implement them. The arrangement is as shown in figure 4.36.

➢ **S-R Flip-Flop To J-K Flip-Flop:**

- Here, the external inputs to the already available S-R flip-flop will be J and K. S and R are the outputs of the combinational circuit, which are also the actual inputs to the S-R flip-flop.

- We write a truth table with J, K, Q_n, Q_{n+1}, S and R, where Q_n is the present state of the flip-flop and Q_{n+1} is next state of the flip-flop before the application of the inputs and Q_{n+1} refers to the state obtained by the flip-flop after the application of inputs.

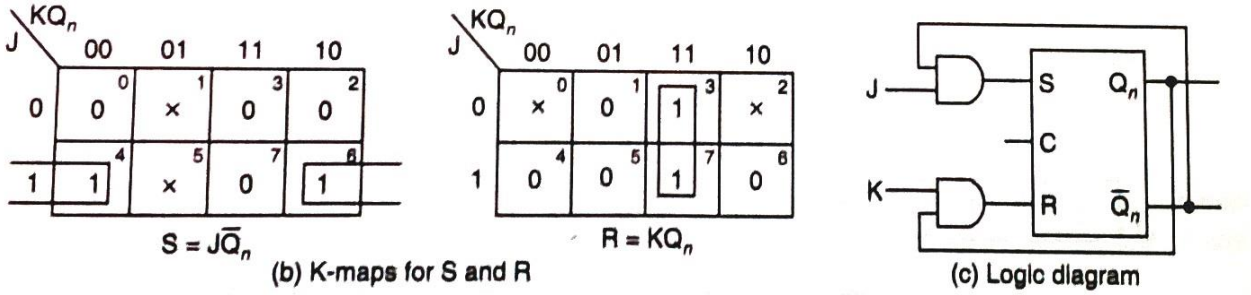
- J, K and Q_n can have eight combinations. For each combination of J, K and Q_n, find the corresponding Q_{n+1}, i.e. determine to which next state (Q_{n+1}) the J-K flip-flop will go from the present state Q_n if the present inputs J and K are applied.

- Now complete the table by writing the values of S and R required to get each Q_{n+1} from the corresponding Q_n, i.e. write what values of S and R are required to change the state of the flip-flop from Q_n to Q_{n+1}.

- The conversion table, the K-maps for S and R in terms of J, K and Q_n and the logic diagram showing the conversion from S-R to J-K are shown in figure 4.37.

External Inputs		Present State	Next State	Flip-flop Inputs	
J	K	Q_n	Q_{n+1}	S	R
0	0	0	0	0	×
0	0	1	1	×	0
0	1	0	0	0	×
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	×	0
1	1	0	1	1	0
1	1	1	0	0	1

(a) Conversion table



(b) K-maps for S and R (c) Logic diagram

Figure 4.37 conversion of S-R flip-flop to J-K flip-flop

➢ **J-K Flip-Flop To S-R Flip-Flop:**

- Here, the external inputs to the already available J-K flip-flop will be S and R. J and K are the outputs of the combinational circuit, which are also the actual inputs to the J-K flip-flop.

- So, we have to get the values of J and K in terms of S, R, and Q_n. thus, write a table using S, R, Q_n, Q_{n+1}, J and K. the external inputs S and R and the present output Q_n can make eight possible combinations. For each combination find the corresponding next state Q_{n+1}.

- In the S-R flip-flop, the combination S=1 and R=1 is not permitted. So, the corresponding output is invalid and therefore, the corresponding J and K are don't cares. Complete the table by writing the values of J and K required to get each Q_{n+1} from the corresponding Q_n.

- The conversion table, the K-maps for J and K in terms of S, R and Q_n and the logic diagram showing the conversion from J-K to S-R are shown in figure 4.38.

External Inputs		Present State	Next State	Flip-flop Inputs	
S	R	Q_n	Q_{n+1}	J	K
0	0	0	0	0	×
0	0	1	1	×	0
0	1	0	0	0	×
0	1	1	0	×	1
1	0	0	1	1	×
1	0	1	1	×	0

(a) Conversion table

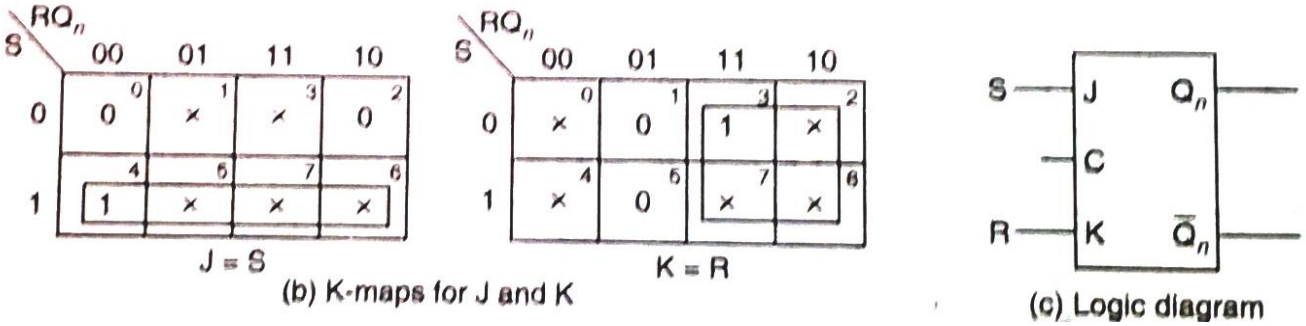


Figure 4.38 conversion of J-K flip-flop to S-R flip-flop

- **S-R Flip-Flop To D Flip-Flop:**

- Here S-R flip-flop is available and we want the operation of D flip-flop from it. So D is the external input and the outputs of the combinational circuit are the inputs to the available S-R flip-flop. Express the inputs of the existing flip-flops S and R in terms of the external input D and the present state Q_n.

- The conversion table, the K-maps for S and R in terms of D and Q_n and the logic diagram showing the conversion from S-R to D are shown in figure 4.39.

External Inputs	Present State	Next State	Flip-flop Inputs	
D	Q_n	Q_{n+1}	S	R
0	0	0	0	×
0	1	0	0	1
1	0	1	1	0
1	1	1	×	0

(a) Conversion table

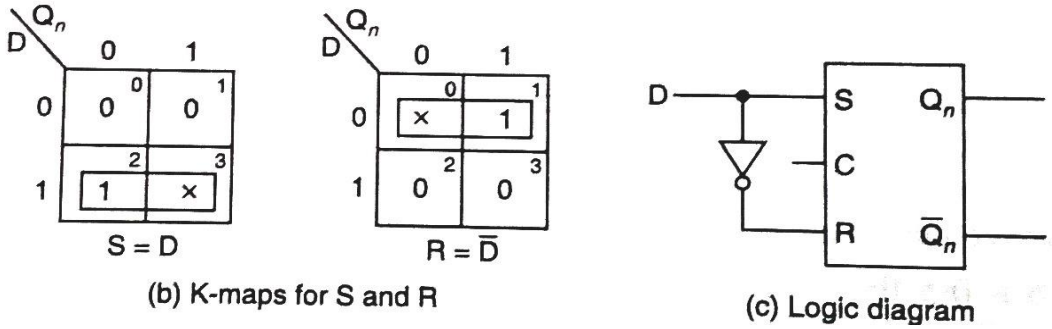


Figure 4.39 conversion of S-R flip-flop to D flip-flop

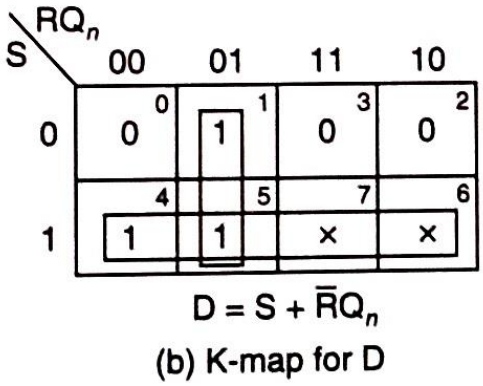
➢ **D Flip-Flop To S-R Flip-Flop:**

- Here D flip-flop is available and we want the operation of S-R flip-flop from it. So S and R is the external input and D is the actual input to the existing flip-flop. S, R and Q_n make eight possible combinations, but S=R=1 is an invalid combination. So, the corresponding entries for Q_{n+1} and D are don't cares. Express D in terms of S, R and Q_n.

- The conversion table, the K-maps for D in terms of S-R and Q_n and the logic diagram showing the conversion from D to S-R are shown in figure 4.40.

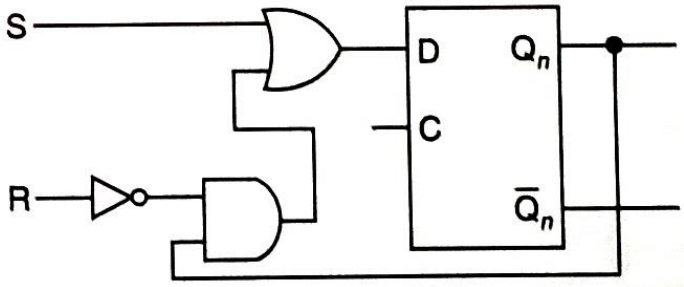
External Inputs		Present State	Next State	Flip-flop Input
S	R	Q_n	Q_{n+1}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1

(a) Conversion table



$D = S + \bar{R}Q_n$

(b) K-map for D



(c) Logic diagram

Figure 4.40 conversion of D flip-flop to S-R flip-flop

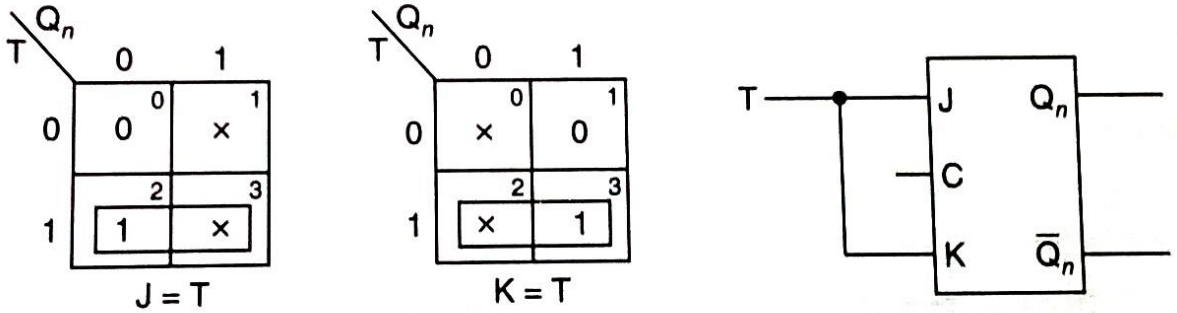
➢ **J-K Flip-Flop To T Flip-Flop:**

- Here J-K flip-flop is available and we want the operation of T flip-flop from it. So T is the external input and J and K are the actual inputs to the existing flip-flop. T and Q_n make four combinations. Express J and K in terms of T and Q_n.

- The conversion table, the K-maps for J and K in terms of T and Q_n and the logic diagram showing the conversion from J-K to T are shown in figure 4.41.

External Input	Present State	Next State	Flip-flop Inputs	
T	Q_n	Q_{n+1}	J	K
0	0	0	0	×
0	1	1	×	0
1	0	1	1	×
1	1	0	×	1

(a) Conversion table



(b) K-maps for J and K (c) Logic diagram

Figure 4.41 conversion of J-K flip-flop to T flip-flop

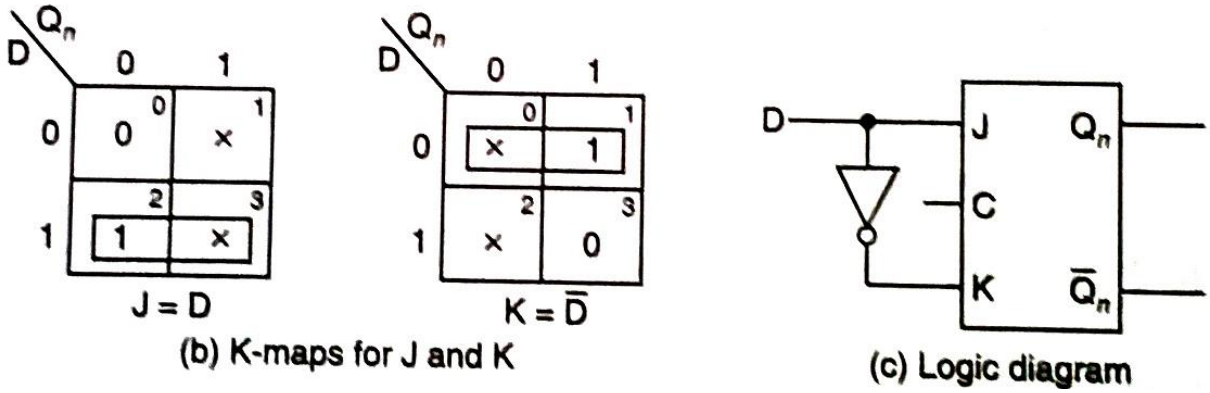
➢ **J-K Flip-Flop To D Flip-Flop:**

- Here J-K flip-flop is available and we want the operation of D flip-flop from it. So D is the external input and J and K are the actual inputs to the existing flip-flop. D and Q_n make four combinations. Express J and K in terms of D and Q_n.

- The conversion table, the K-maps for J and K in terms of D and Q_n and the logic diagram showing the conversion from J-J to D are shown in figure 4.42.

External Input	Present State	Next State	Flip-flop Inputs	
D	Q_n	Q_{n+1}	J	K
0	0	0	0	×
0	1	0	×	1
1	0	1	1	×
1	1	1	×	0

(a) Conversion table



(b) K-maps for J and K

(c) Logic diagram

Figure 4.42 conversion of J-K flip-flop to D flip-flop

- **D Flip-Flop To J-K Flip-Flop:**

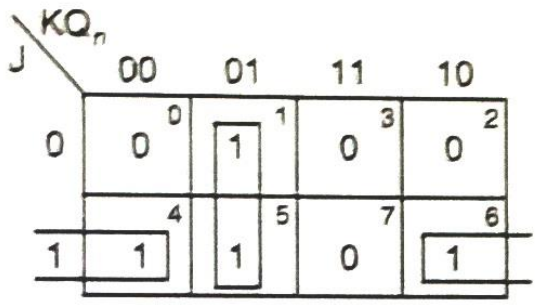
- Here D flip-flop is available and we want the operation of J-K flip-flop from it. So J-K is the external input, i.e. inputs to the combinational circuit and D is the actual input to the existing flip-flop.

- J, K and Q_n make eight combinations. Express the input of the existing flip-flop D in terms of external inputs J, K and present state Q_n.

- The conversion table, the K-maps for D in terms of J, K and Q_n and the logic diagram showing the conversion from J-J to D are shown in figure 4.43.

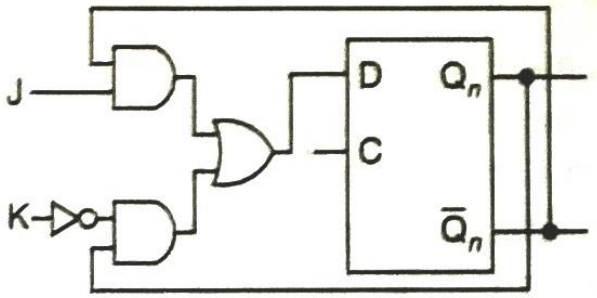
External Inputs		Present State	Next State	Flip-flop Input
J	K	Q_n	Q_{n+1}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

(a) Conversion table



$D = J\bar{Q}_n + \bar{K}Q_n$

(b) K-maps for D



(c) Logic diagram

Figure 4.43 conversion of D flip-flop to J-K flip-flop

4.3 Register

- A Register is a set of Flip Flops to store Binary information. As a Flip Flop can able to store one bit either “0” or “1”, it can be considered as one bit Register. When more bits of data are to be stored, numbers of Flip Flops are used. A group of Flip Flops constitutes a Register.

- The storage capacity of a Register is the number of bits (data) it can retain or store. To store bits (data) into the Register, these data is to be loaded either in serial or parallel. Loading of Register means SETTING or RESETTING of Flip Flops of Register. Loading may be in serial or parallel form.

- In serial loading data transferred into Register one by one in serial fashion. In parallel loading data transferred/entered into Register in parallel form i.e. all the data entered at the same time. A Register may output the data in serial or parallel form.

- A Register has two important characteristics:
  (i) Memory
  (ii) Shifting

- **Buffer Register:**
  Some registers do nothing than storing a binary information/data. Buffer register is a simplest form of registers which is simply used to store binary information. Most of buffer register use “D Flip Flop”.

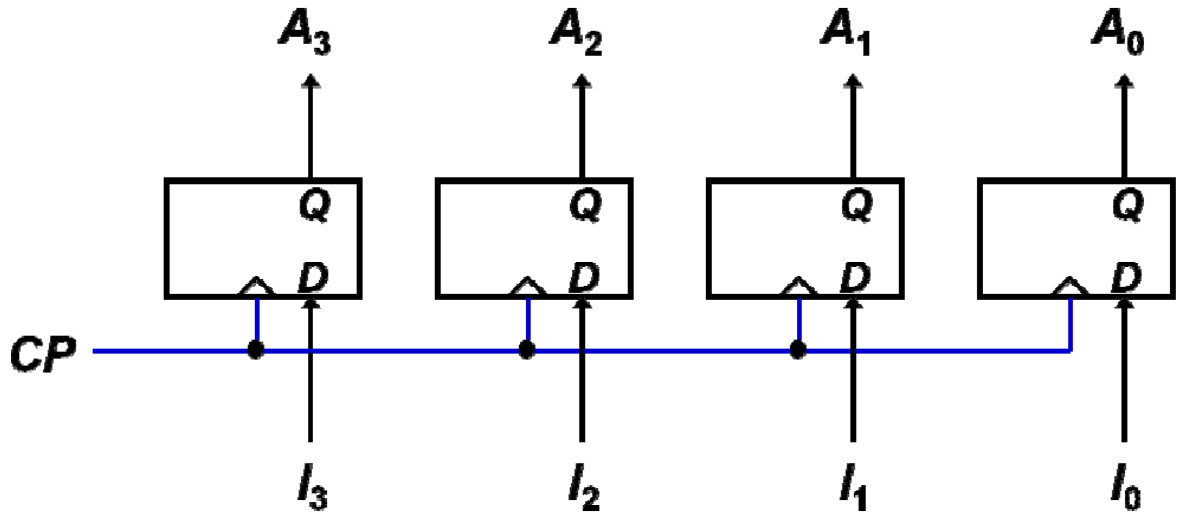


Figure 4.44 Buffer Register

- Figure 4.44 shows buffer register. The binary word which is to be stored is applied to the input terminal of D Flip Flop. On application of clock pulse all the data will be loaded into the register and because of D Flip Flop used, output will be same as input.

- Buffer register can be controlled by external combinational gates also, such that we can store the data at a particular time and also able to clear the previous data to load new data into the register.

- **Shift Register:**

- A register may input and output data in serial or parallel form. It can also shift the data left or right. A number of Flip Flops connected together in such a way that data may be shifted into and out of the register is called Shift register.

- The number of individual data Flip Flop/latches required to make up a single **Shift Register** device is usually determined by the number of bits to be stored. The most common being 8-bits (one byte) wide constructed from eight individual data latches.

- The Shift Register is used for data storage or data movement and are used in calculators or computers to store data such as two binary numbers before they are added together, or to convert the data from either a serial to parallel or parallel to serial format. The individual data latches that make up a single shift register are all driven by a common clock signal making them synchronous devices.

- Shift register IC's are generally provided with a **CLEAR** or **RESET** connection so that they can be "SET" or "RESET" as required. Generally, shift registers operate in one of four different modes with the basic movement of data through a shift register.

- Depending upon data entered into the register and output taken from register, there are four basic types of shift register.

  (i) Serial Input, Serial Output Shift Register
  (ii) Serial Input, Parallel Output Shift Register
  (iii) Parallel Input, Serial Output Shift Register
  (iv) Parallel Input Parallel Output Shift Register

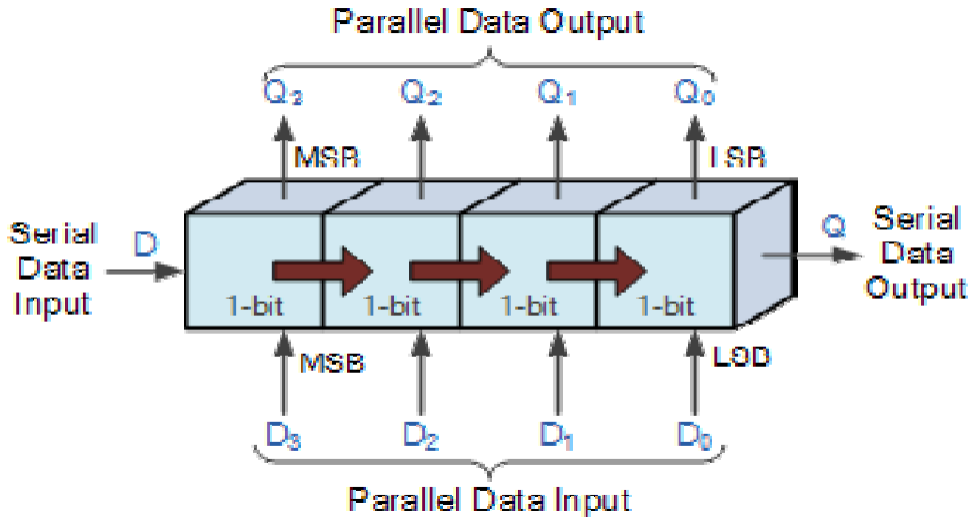
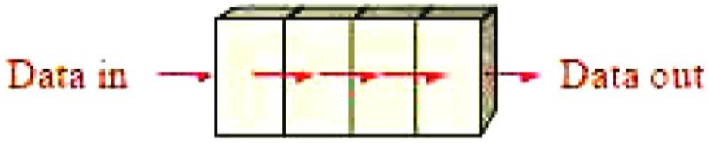
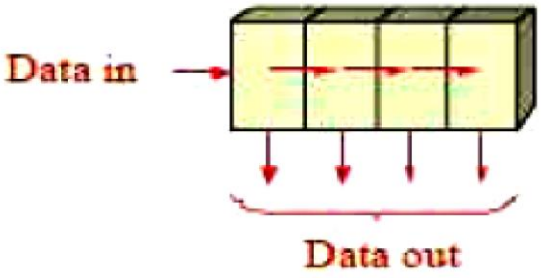


Figure 4.45 All the possible ways of input and output from shift register

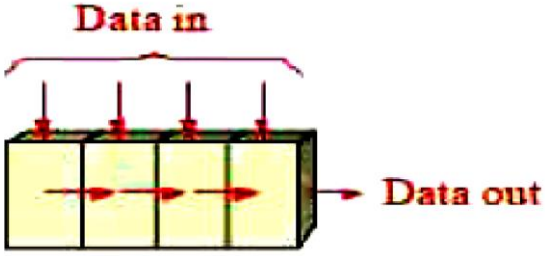
- **Serial Input, Serial Output Shift Register:**



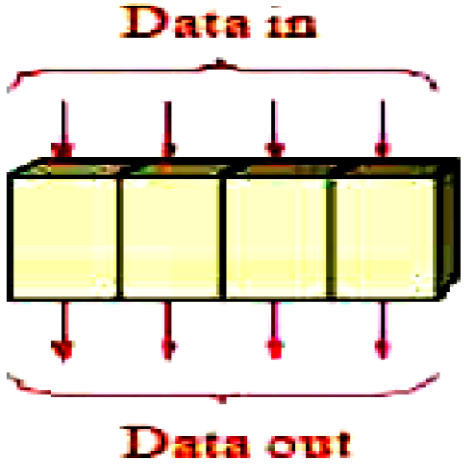
- **Serial Input, Parallel Output Shift Register:**



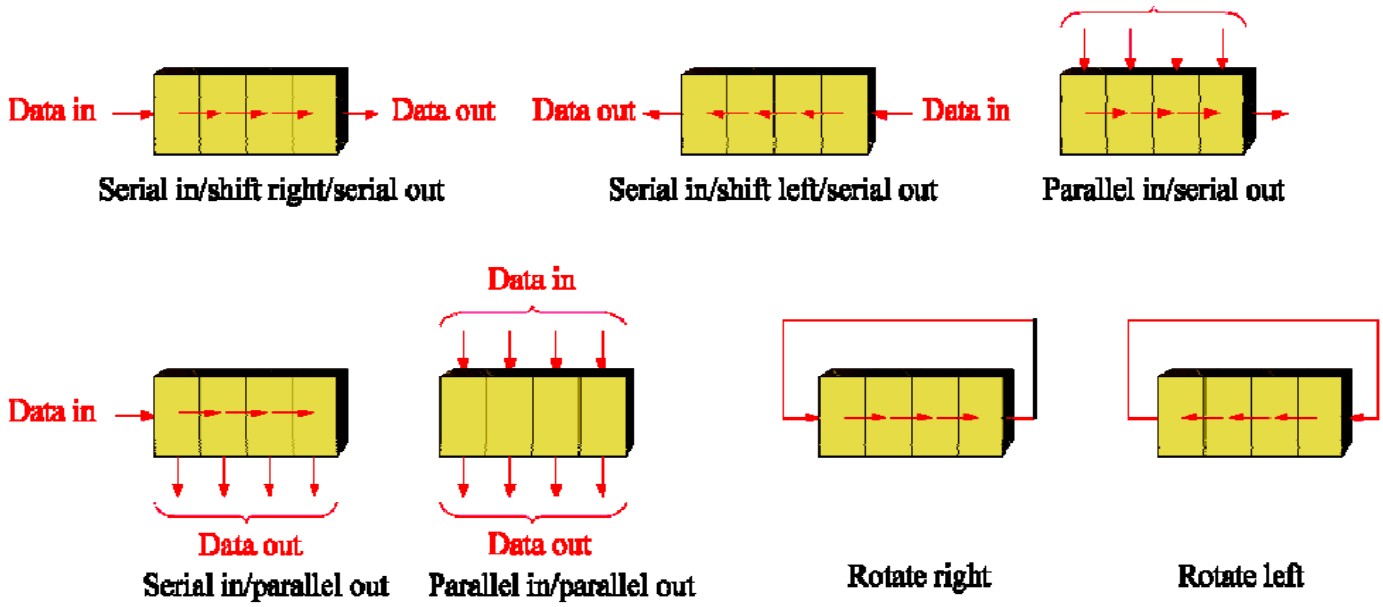
- **Parallel Input, Serial Output Shift Register:**



- **Parallel Input, Serial Output Shift Register:**



- **Various shifting operation in Shift register:**



4.3.1 Serial input serial output shift Register:

- The data is shifted serially "IN" and "OUT" of the register, one bit at a time in either a left or right direction under clock control.

- The SISO shift register is one of the simplest of the four configurations as it has only three connections, the serial input (SI) which determines what enters the left hand flip-flop, the serial

output (SO) which is taken from the output of the right hand flip-flop and the sequencing clock signal. The logic circuit diagram in Figure 4.46 shows a generalized serial-in serial-out shift register.

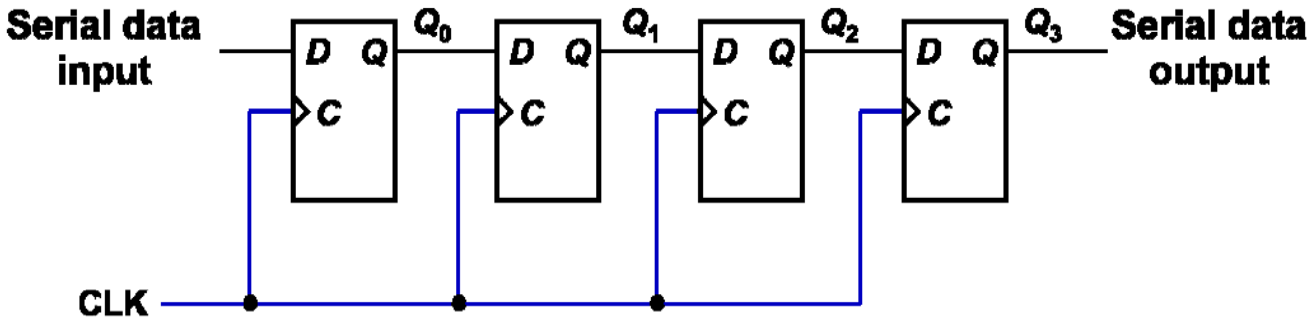


Figure 4.46 4-bit Serial-in Serial-out Shift Register

- This type of **Shift Register** also acts as a temporary storage device or as a time delay device for the data, with the amount of time delay being controlled by the number of stages in the register, 4, 8, 16 etc. or by varying the application of the clock pulses. Commonly available IC's include the 74HC595 8-bit Serial-in to Serial-out Shift Register all with 3-state outputs.

- Figure 4.47 and Figure 4.48 shows truth table and timing diagram/waveforms for **Serial input serial output shift Register,** indicating how data stored and shifted within this register.

- **Truth Table:**

	CLK	$D_{in} = Q_3$	$Q_3 = D_2$	$Q_2 = D_1$	$Q_1 = D_0$	Q_0
Initially			0	0	0	0
(i)	↓	1	1	0	0	0
(ii)	↓	1	1	1	0	0
(iii)	↓	1	1	1	1	0
(iv)	↓	1	1	1	1	1

→ Direction of data travel

Figure 4.47 Truth Table

- For example, the data input to the register is D=1111. From the truth table it has been observed that after 4 clock pulses are required to store complete binary word “1111” within the register. Now if we need output serially then another 4 clock pulses are required to obtain output serially. In total 8 clock pulses are required to obtain 4 bit data word serially, if data input is in serial form. So speed of operation is slower in SISO shift register.

- **Timing Diagram/waveform:**

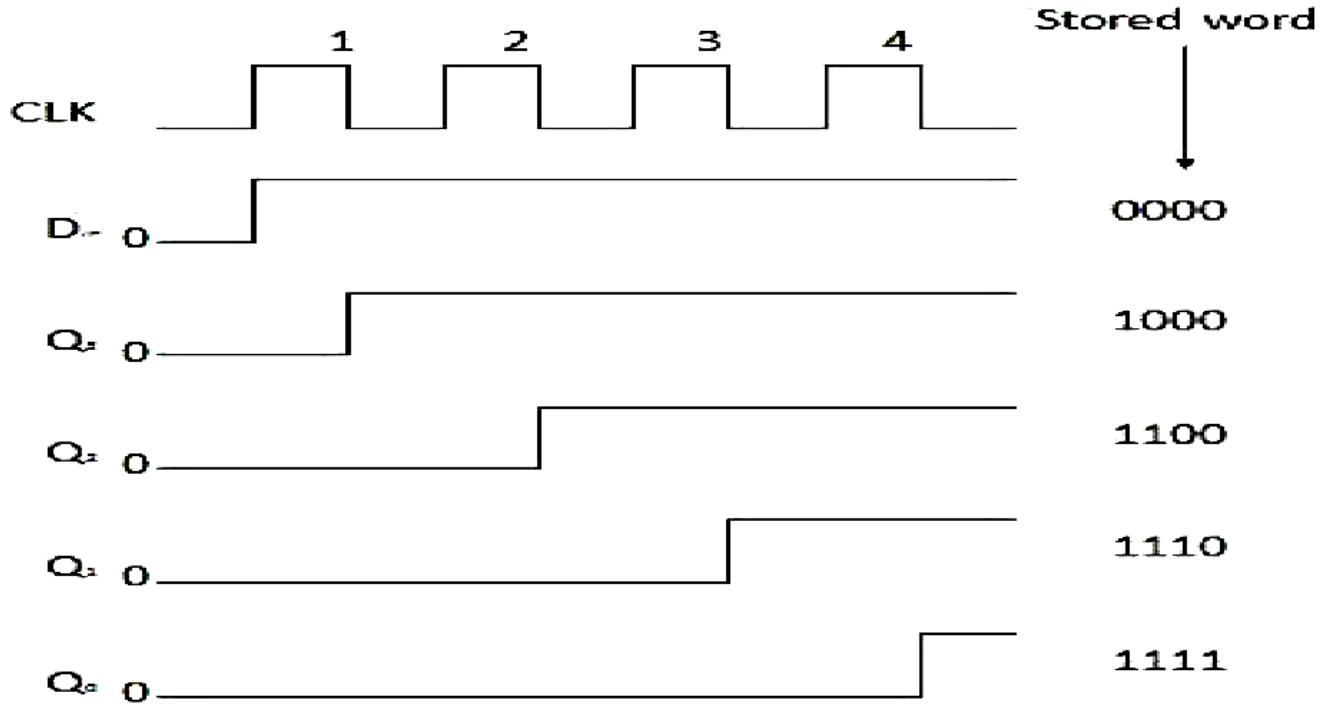


Figure 4.48 Timing Diagram/waveform

- **Working:**

- Before application of clock signal let Q0 Q1 Q2 Q3 = 0000 and apply LSB bit of the number to be entered to Din. So Din= D3=1. Apply the clock. On the first leading edge of clock, the FF-3 is set, and stored word in the register is Q0 Q1 Q2 Q3 = 1000.

- Apply the next bit to Din. So Din=1. As soon as the next positive edge of the clock hits, FF-2 will set and the stored word change to Q0 Q1 Q2 Q3 = 1100.

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

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

4.3.2 Serial input Parallel output shift Register:

- In such types of operations, the data is entered serially and taken out in parallel fashion. Data is loaded bit by bit. The outputs are disabled as long as the data is loading. As soon as the data loading gets completed, all the flip-flops contain their required data; the outputs are enabled so that all the loaded data is made available over all the output lines at the same time. 4 clock cycles are required to load a four bit word serially. After loading and shifting all the data will be available simultaneously because of parallel output.

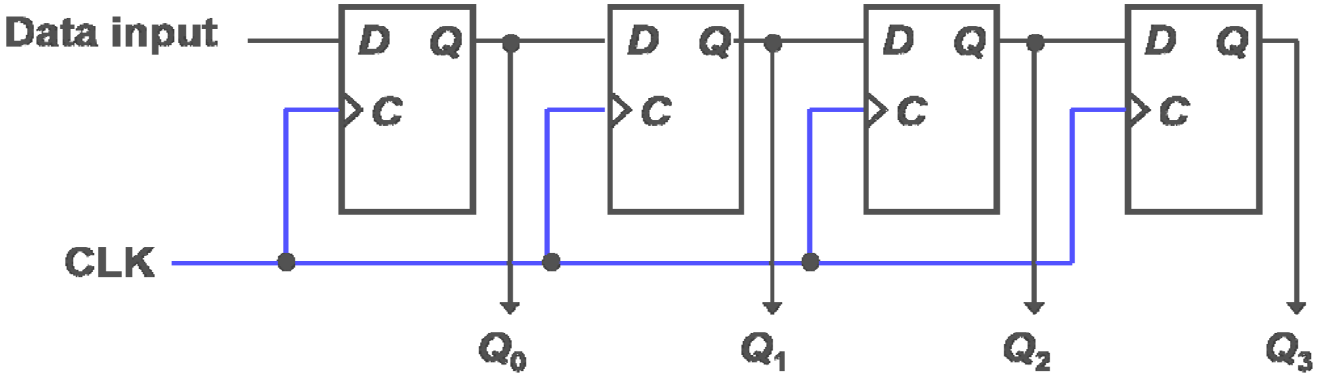


Figure 4.49 4-bit Serial-in to Parallel-out shift Register

- **Basic data movement through a Shift Register:**

Clock Pulse No	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	0	0	0	0

Figure 4.50 Truth Table of Serial-in to Parallel-out shift Register

4.3.3 Parallel input Serial output shift Register:

- The Parallel-in to Serial-out shift register acts in the opposite way to the serial-in to parallel-out one above. The data is loaded into the register in a parallel format in which all the data bits enter their inputs simultaneously, to the parallel input of the register. The data is then read out sequentially in the normal shift-right mode from the register.

- This data is outputted one bit at a time on each clock cycle in a serial format. It is important to note that with this system a clock pulse is not required to parallel load the register as it is already present, but four clock pulses are required to unload the data.

- As this type of shift register converts parallel data, into serial format, it can be used to multiplex many different input lines into a single serial DATA stream which can be sent directly to a computer or transmitted over a communications line. Commonly available IC's include the 74HC166 8-bit Parallel-in/Serial-out Shift Registers.

- The circuit shown below is a four bit parallel input serial output register. Output of previous Flip Flop is connected to the input of the next one via a **combinational circuit**. The binary input word D0, D1, D2, D3 is applied though the same combinational circuit.

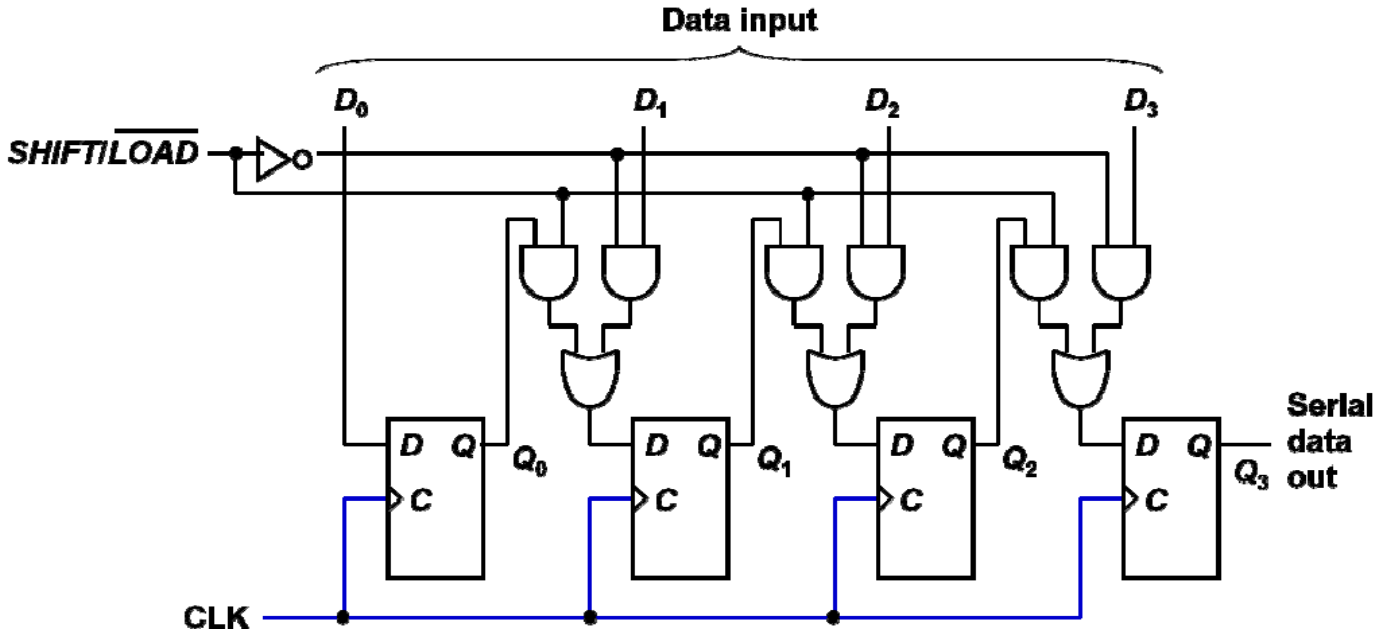


Figure 4.51 4-bit Parallel-in to Serial-out shift Register

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

(i) Load Mode:

- When the **shift/load** bar line is low (**0**), the AND gate 2, 4 and 6 become active. They will pass D1, D2, D3 bits to the corresponding flip-flops. On the positive going edge of clock, the binary input D0, D1, D2, D3 will get loaded into the corresponding flip-flops. Thus parallel loading takes place.

(ii) Shift Mode:

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

4.3.4 Parallel input Parallel output shift Register:

- In this mode, the 4 bit binary input D0, D1, D2, D3 is applied to the data inputs respectively of the four D flip-flops. As soon as a Positive clock edge is applied, the input binary bits will be loaded into the flip-flops simultaneously. The loaded bits will appear simultaneously to the output side. **Only 1 clock pulse is essential to load all the bits.**

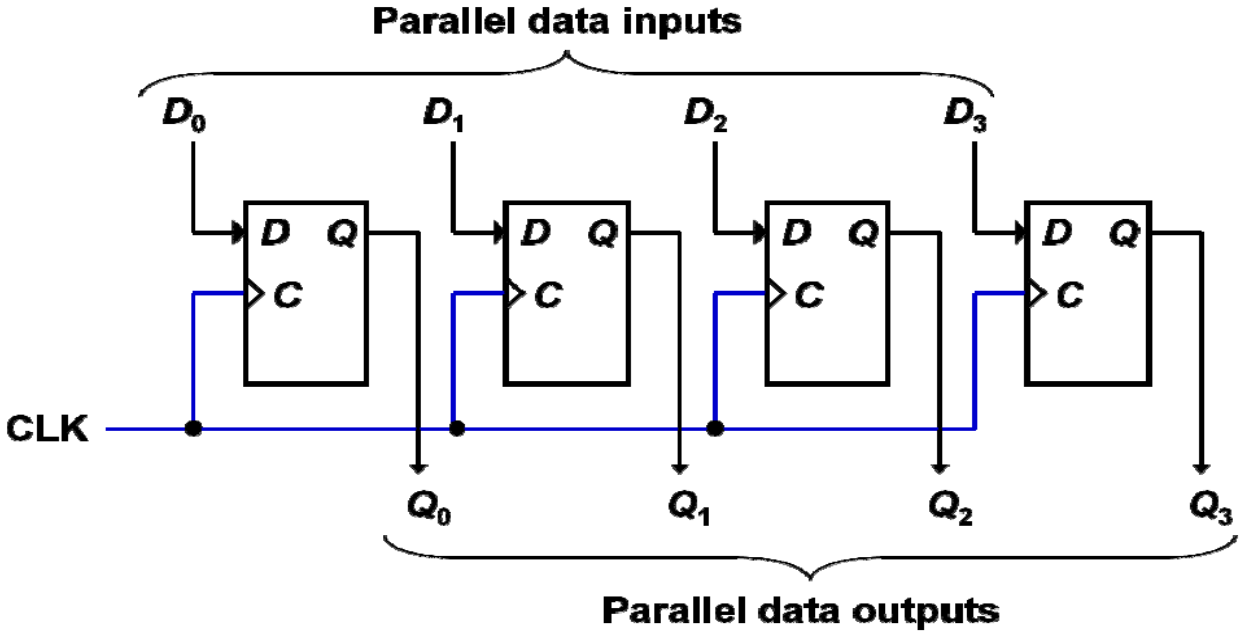


Figure 4.52 4-bit Parallel-in to Parallel-out shift Register

4.3.5 Bidirectional Shift Register:

- If a binary number is shifted left by one position then it is equivalent to multiplying the original number by 2. Similarly if a binary number is shifted right by one position then it is equivalent to dividing the original number by 2.

- Hence if we want to use the shift register to multiply and divide the given binary number, then we should be able to move the data in either left or right direction. Such a register is called as a bi-directional register. A four bit bi-directional shift register is shown in Figure 4.53

- There is one input which can load the data serially from left side or right side which is connected to AND gate 1 and 8. One control signal **RIGHT / $\overline{\text{LEFT}}$** used to operate register in either of two modes:

  (i) Shift Left
  (ii) Shift Right

- When mode selection signal **RIGHT/$\overline{\text{LEFT}}$** is **1**, shift right operation will be performed.
- When mode selection signal **RIGHT/$\overline{\text{LEFT}}$** is **0**, shift left operation will be performed.

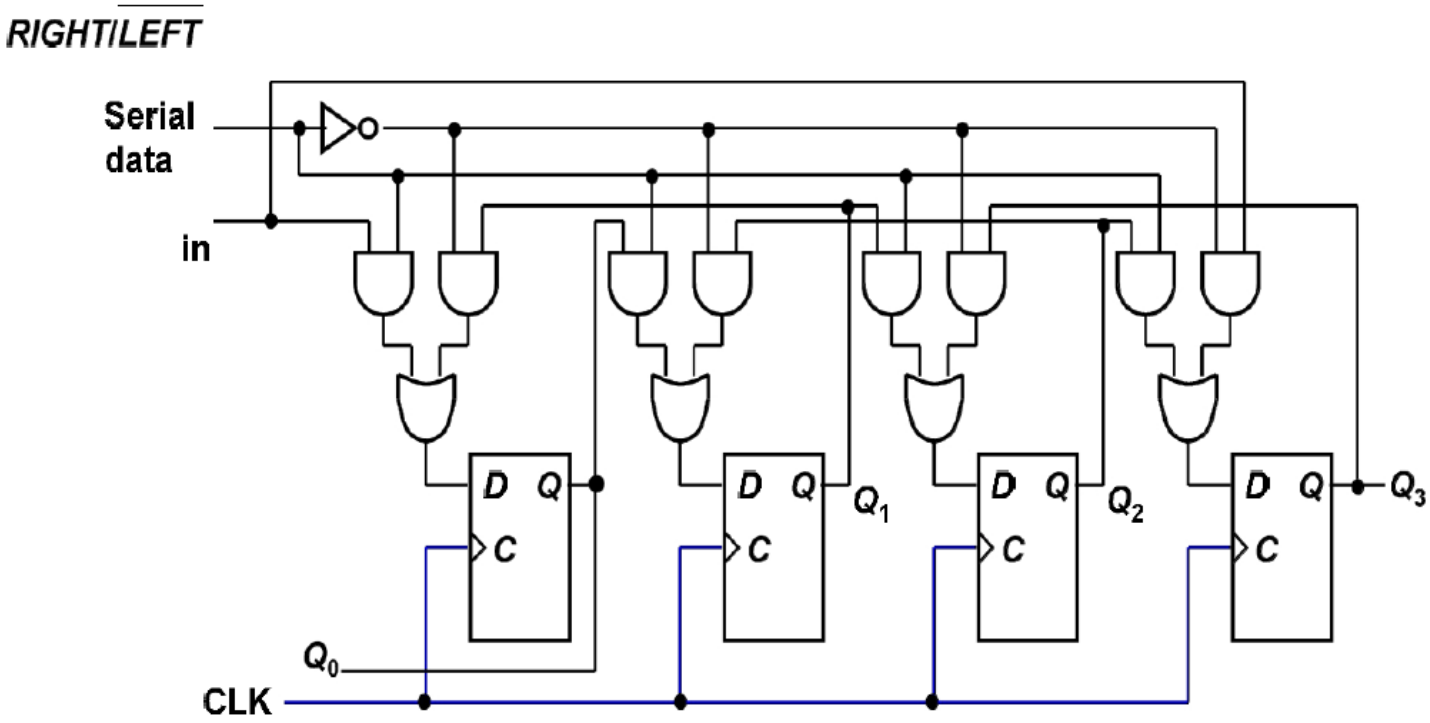


Figure 4.53 4-bit Bidirectional Shift Register

➢ **Working:**

- **With RIGHT/$\overline{\text{LEFT}}$ = 1 (Shift right operation)**

- If RIGHT/$\overline{\text{LEFT}}$ = 1, then the AND gates 1, 3, 5 and 7 are enable whereas the remaining AND gates 2, 4, 6 & 8 will be disabled
- The data at input is shifted to right bit by bit from FF-3 to FF-0 on the application of clock pulses. Thus with RIGHT/$\overline{\text{LEFT}}$ = 1 we get the serial right shift operation

- **With RIGHT/$\overline{\text{LEFT}}$ = 0 (Shift left operation)**

- When the mode control RIGHT/$\overline{\text{LEFT}}$ is connected to 0 then the AND gates 2, 4, 6 & 8 are enabled while 1, 3, 5 & 7 are disabled.
- The data at D_L is shifted left bit by bit from FF-0 to FF-3 on the application of clock pulses. Thus with RIGHT/$\overline{\text{LEFT}}$= 0 we get the serial left shift operation.

4.3.6 Universal Shift Register:

- A shift register which can shift the data in only one direction is called a uni-directional shift register. A shift register which can shift the data in both directions is called a bi-directional shift register. Applying the same logic, a shift register which can shift the data in both directions as well as load it parallelly, then it is known as a universal shift register.

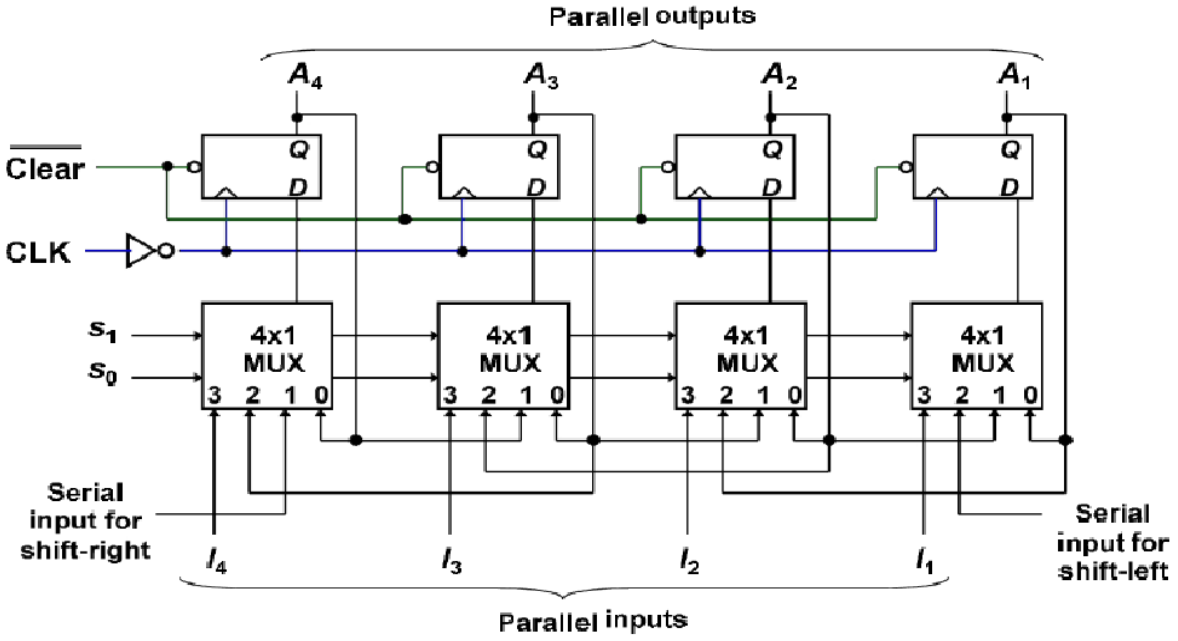


Figure 4.54 4-bit bidirectional shift register with parallel load

Mode Control		
s_1	s_0	Register Operationn
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

The shift register is capable of performing the following operation.

1. Parallel loading
2. Left shifting
3. Right shifting

- The mode control input is connected to logic 1 for parallel loading operation whereas it is connected to 0 for serial shifting. With mode control pin connected to ground, the universal shift register acts as a bi-directional register. For serial left operation, the input is applied to the serial input which goes to AND gate-1 shown in figure 4.54 whereas for the shift right operation, the serial input is applied to D input.

➢ **4-bit Universal Shift Register IC 74LS194**

- Today, there are many high speed bi-directional "universal" type **Shift Registers** available such as the TTL 74LS194, 74LS195 or the CMOS 4035 are available as a 4-bit multi-function devices that can be used in either serial-to-serial, left shifting, right shifting, serial-to-parallel, parallel-to-serial, and as a parallel-to-parallel multifunction data register, hence the name "Universal".

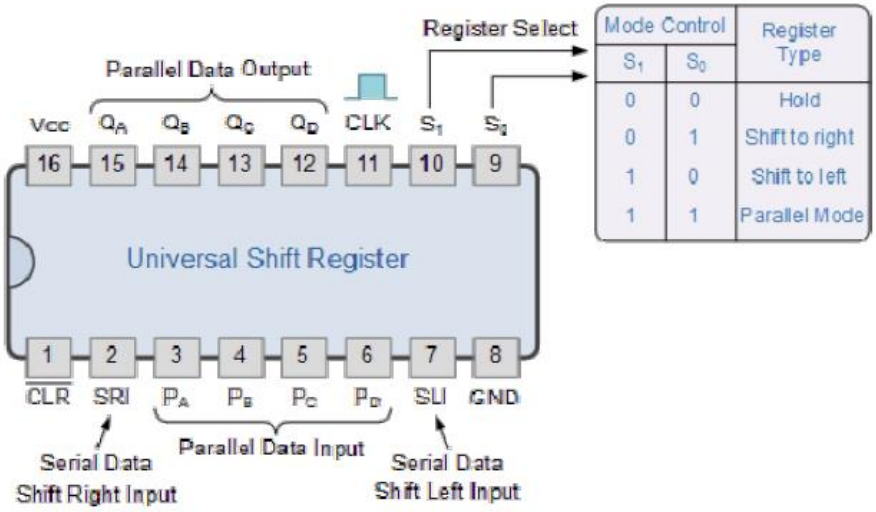


Figure 4.55 4-bit Universal Shift Register IC 74LS194

- These universal shift registers can perform any combination of parallel and serial input to output operations but require additional inputs to specify desired function and to pre-load and reset the device. A commonly used universal shift register is the TTL 74LS194 as shown in figure 4.55.

- Universal shift registers are very useful digital devices. They can be configured to respond to operations that require some form of temporary memory, delay information such as the SISO or PIPO configuration modes or transfer data from one point to another in either a serial or parallel format. Universal shift registers are frequently used in arithmetic operations to shift data to the left or right for multiplication or division.

4.4 Counter:

- A counter is a digital sequential logic device that will go through a certain predefined states (for example counting up or down) based on the application of the input pulses. They are utilized in almost all computers and digital electronics systems.

- Counting is frequently required in digital computers and other digital systems to record the number of events occurring in a specified interval of time. Normally an electronic counter is used for counting the number of pulses coming at the input line in a specified time period. The counter must possess memory since it has to remember its past states. As with other sequential logic circuits counters can be synchronous or asynchronous.

- If we broadly classify the counter then it can be classified as (1) Asynchronous counters and (2) Synchronous counters

- As the name suggests, it is a circuit which counts. The main purpose of the counter is to record the number of occurrence of some input. There are many types of counter both binary and decimal. Some of commonly used counters are for example….

  (a) Binary Ripple Counter
  (b) Ring Counter
  (c) BCD Counter
  (d) Decade counter
  (e) Up down Counter
  (f) Frequency Counter

- JK Flip Flop and T Flip Flops are preferred for the design of various counters.

4.4.1 Asynchronous Counters: (Ripple Counter)

- This counter counts the number of clock pulses applied to it. In this type of counter each Flip Flop is triggered by the output of previous flip flop (except the first Flip Flop). In this type of counter, all the Flip Flop does not change the states in exact synchronism with clock pulse. Only first Flip Flop responds to the input clock pulse. So it is known as Asynchronous counter.

- It is also known as **Ripple counter** because of the manner in which it operates. Because Flip Flop-1 responds to the external input pulse, then second Flip Flop-2 has to wait for Flip Flop-1 to change states before it toggles. Flip Flop-3 has to wait for Flip Flop-2 and so on.

- This counter can be designed using T Flip Flop or J-K Flip Flop easily. It can be also considered as serial or series counter. A counter may count up or count down or count up and down depending on the input control. So a counter may be **Up counter** or a **Down counter**.

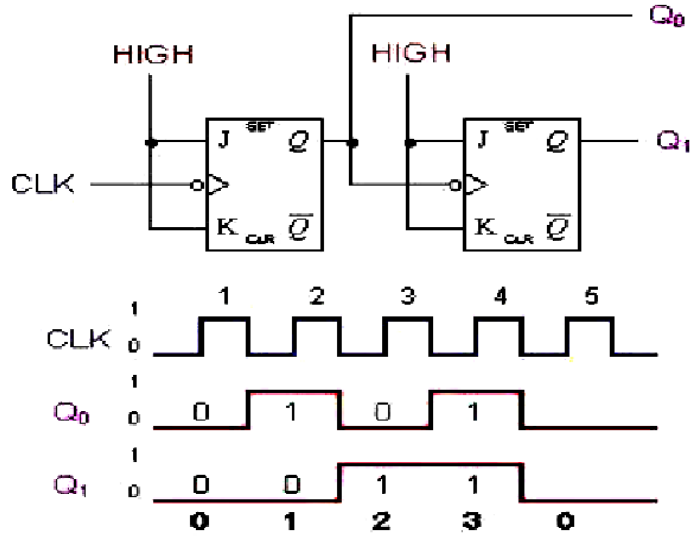
- The number of states through which counter passes before returning to the starting state is called **modules** of the counter (or **mod** of counter). Hence modules or mod of a counter is equal to the number of states (counts), a counter can store (including zero state). If n is number of Flip Flops used in counter then mod of the counter equals to 2^n.

- For example a: A two bit counter which requires two Flip Flop has four states 00, 01, 10 and 11. So it is called a Mod-4 counter. It divides the input clock frequency by 4. So it is also called **divide-by-4 counter**.

- Asynchronous counters are slower than synchronous counters because of the delay in the transmission of the pulses from flip-flop to flip-flop.

➢ **Design of 2 Bit Binary Up counter:**
(2 Bit Binary Ripple Up counter)/ (Mod-4 counter)/ (divide-by-4 counter)

- This type of counters has JK Flop-Flops arranged in a way that the output of one flip-flop feeds the clock of the following flip-flop as shown in the figure 4.56 with associated wave form. Flip Flop used is **negative edge triggered** JK Flip Flop (T Flip Flop can be also used).



Clock Pulse	Q_1	Q_0
1	0	0
2	0	1
3	1	0
4	1	1

Figure 4.56 Two bit binary Ripple Up Counter with wave form & count table

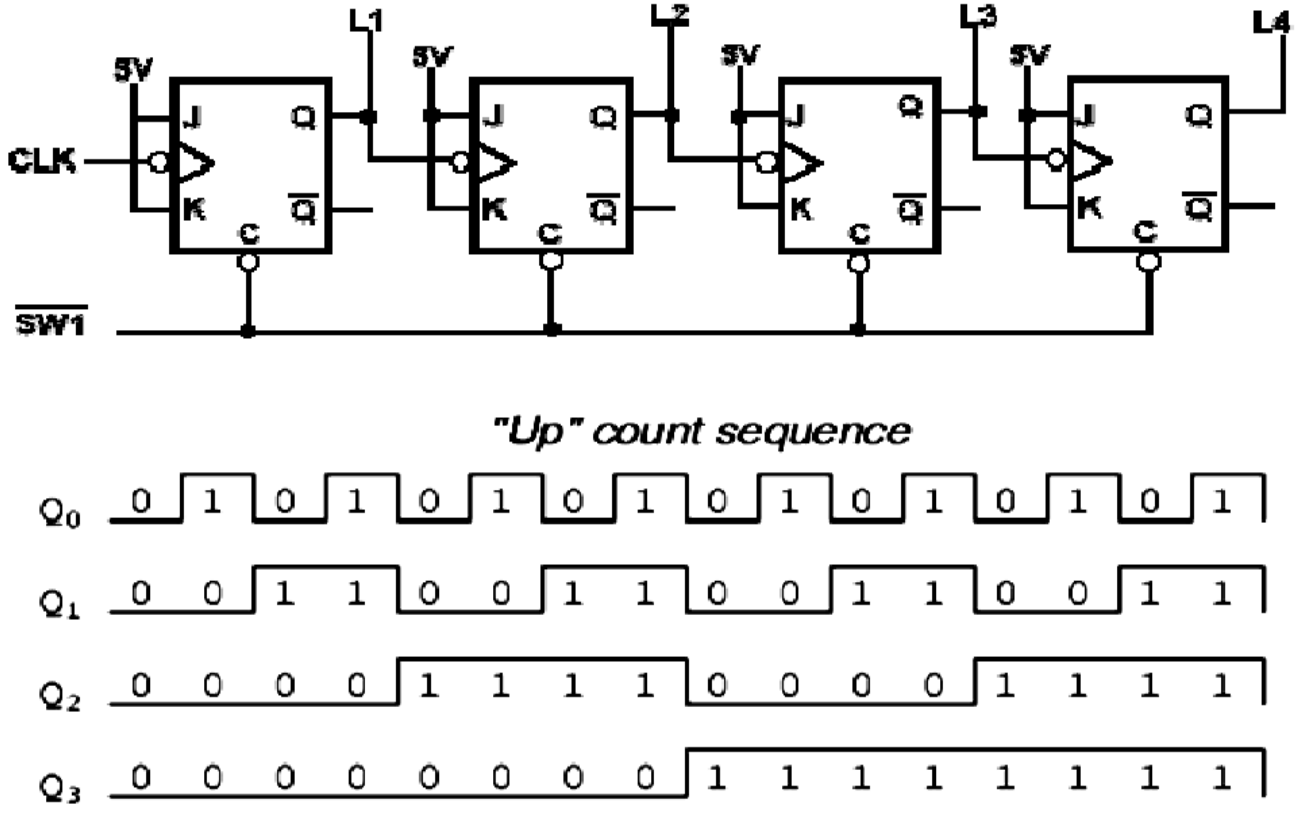


Figure 4.57 Four bit binary Ripple Up Counter with wave form

Clock Pulse	L4 (Q3)	L3 (Q2)	L2 (Q1)	L1 (Q0)
1	0	0	0	0
2	0	0	0	1
3	0	0	1	0
4	0	0	1	1
5	0	1	0	0
6	0	1	0	1
7	0	1	1	0
8	0	1	1	1
9	1	0	0	0
10	1	0	0	1
11	1	0	1	0
12	1	0	1	1
13	1	1	0	0
14	1	1	0	1
15	1	1	1	0
16	1	1	1	1

Figure 4.58 Four Bit Ripple Up Counter's count sequence

- **Design of 2 Bit Binary Up counter using positive edge triggered JK Flip Flop: (2 Bit Binary Ripple Up counter)/ (Mod-4 counter)/ (divide-by-4 counter)**

- This type of counters has JK Flop-Flops arranged in a way that the output of one flip-flop feeds the clock of the following flip-flop as shown in the figure 4.59 with associated wave form. Flip Flop used is **positive edge triggered** JK Flip Flop (T Flip Flop can be also used).

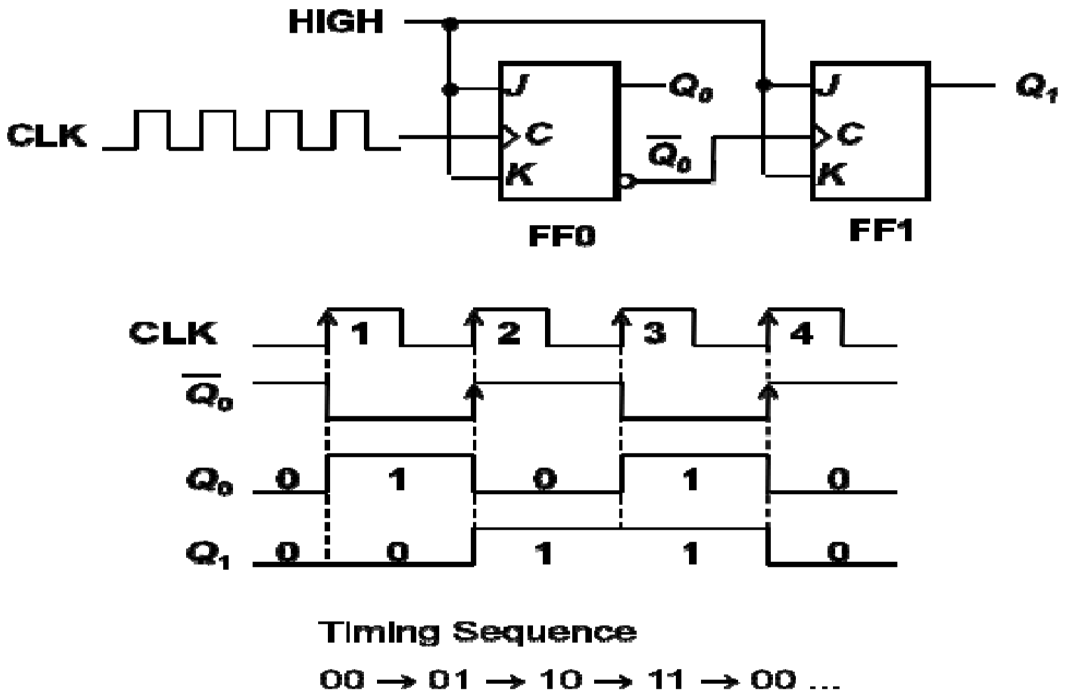


Figure 4.59 Two Bit Ripple Up Counter with wave form & count sequence

- **Design of 3 Bit Binary Up counter using positive edge triggered JK Flip Flop: (3 Bit Binary Ripple Up counter)/ (Mod-8 counter)/ (divide-by-8 counter)**

- This type of counters has JK Flop-Flops arranged in a way that the output of one flip-flop feeds the clock of the following flip-flop as shown in the figure 4.60 with associated wave form. Flip Flop used is **positive edge triggered** JK Flip Flop (T Flip Flop can be also used).

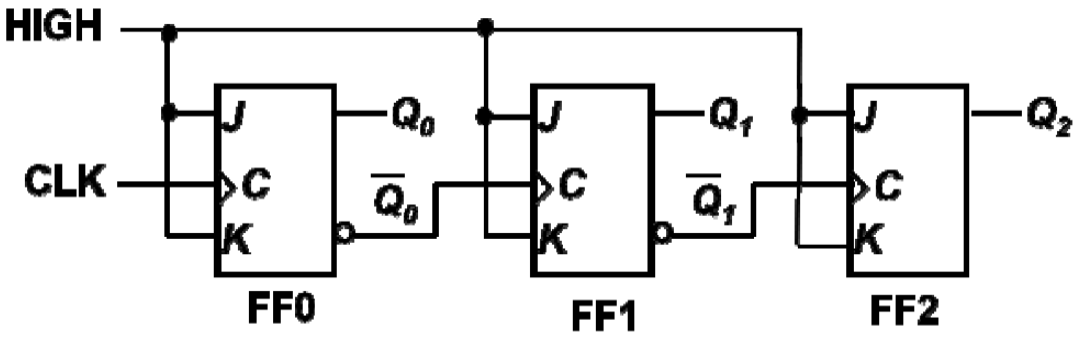
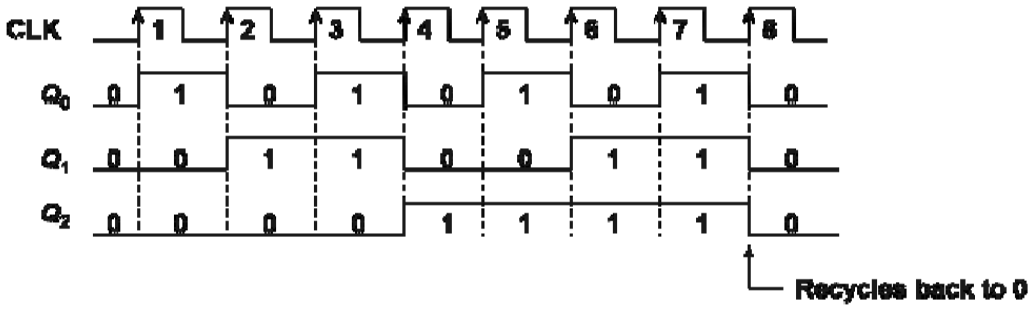


Figure 4.60 Three bit Ripple up Counter



Clock Pulse	(Q2)	(Q1)	(Q0)
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

Figure 4.61 Three Bit Ripple Up Counter with wave form & count sequence

- **Design of 3 Bit Binary Down counter:**
  (3 Bit Binary Ripple Down counter)/ (Mod-8 counter)/ (divide-by-8 counter)

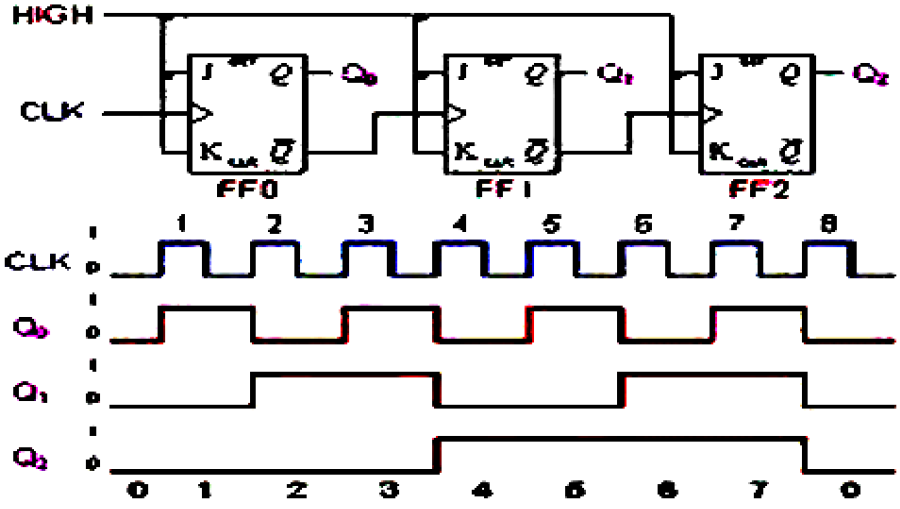


Figure 4.62 Three Bit Ripple Down Counter with wave form

- This type of counters has JK Flop-Flops arranged in a way that the output of one flip-flop feeds the clock of the following flip-flop as shown in the figure 4.62 with associated wave form. Flip Flop used is **negative edge triggered** JK Flip Flop (T Flip Flop can be also used).

Clock Pulse	(Q2)	(Q1)	(Q0)
1	1	1	1
2	1	1	0
3	1	0	1
4	1	0	0
5	0	1	1
6	0	1	0
7	0	0	1
8	0	0	0

Figure 4.63 Three Bit Ripple Down Counter - count sequence

➢ **Design of Three Bit Asynchronous Up-Down Counter:**

- In Asynchronous counter external clock is given to only first flip flop and remaining all flip flops receives clock from output of previous flip flop. Input is JK=1 common to all Flip Flops. Output frequency of last flip flop is 1/8 of the applied clock frequency so it is called frequency divider counter. In the same way a four bit counter is known as divide by 16 counters. If number of Flip-Flops are "n" the maximum counting capability is given as $N = 2^n - 1$.

- Figure 4.64 shows Up/Down Counter using **negative edge triggered** J-K Flip Flop. Counter can be designed using T Flip Flop also. It is programmable counter because When Up=1 and Down=0, it counts upward. When Up=0 and Down=1, it counts downward.

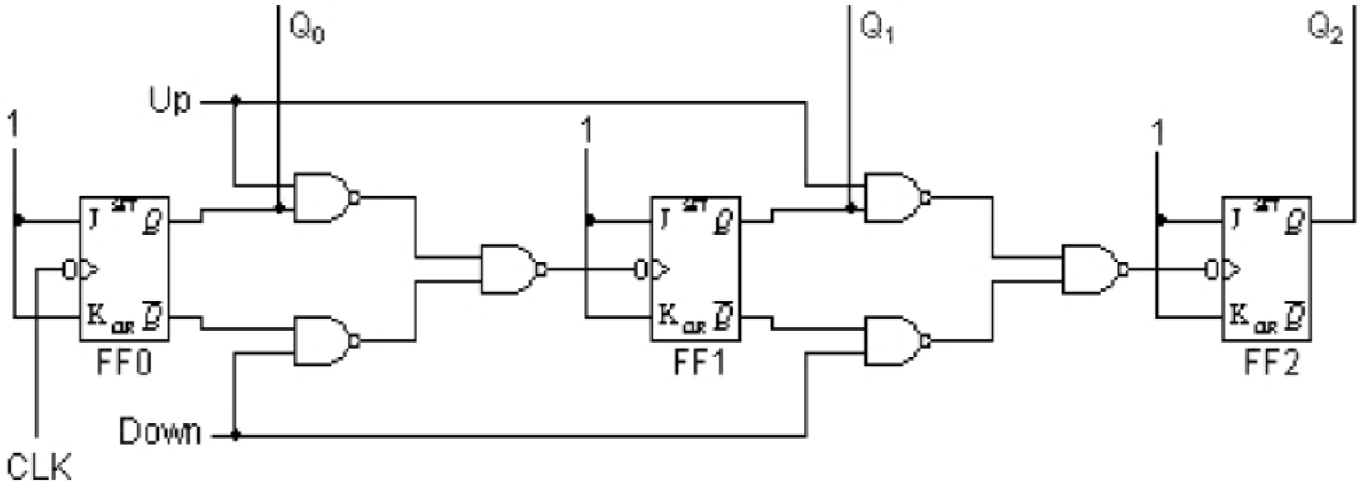
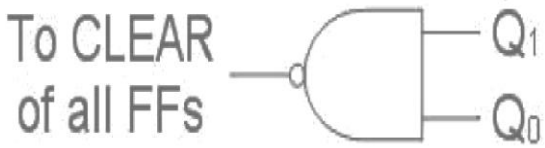


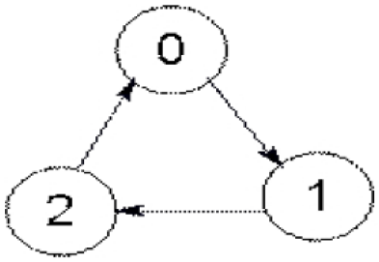
Figure 4.64 Three Bit Binary asynchronous Up/Down Counter

- **Design of Asynchronous Mod-3 Counter:**

- We can design the MOD 3 counter using 2 FFs as 3 is less than 4 i.e. 2^2 and greater than 2. We can see directly that as we have to reset the counter only after 2 i.e. when output is 3 we reset the counter and hence we need to reset only when we have Q0= 1 & Q1=1. Now firstly design MOD-4 counter using 2 FFs and then take NAND of Q0 & Q1 and feed the output to CLEAR of both FFs.



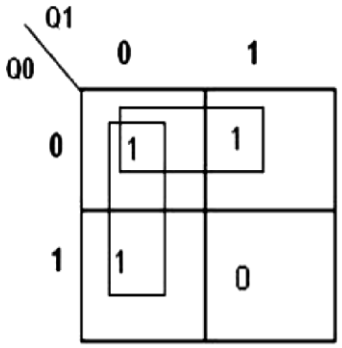
- State diagram of the counter required as



- To reset the Counter, now if we draw a table to list the different input combinations to Combinational circuit and their corresponding output as:

Q1	Q0	OUTPUT of reset logic
0	0	1
0	1	1
1	0	1
1	1	0

- And using K-map as



And equation we get is
Z=Q0 bar + Q1 bar
= NAND of (Q0,Q1)

$\mathbf{z = \overline{Q_0} + \overline{Q_1} = \overline{Q_0 Q_1}}$ (Which is NAND of Q_0 and Q_1) as per De-Morgan's Theorem

- Hence we get the whole circuit for MOD-3 counter as:

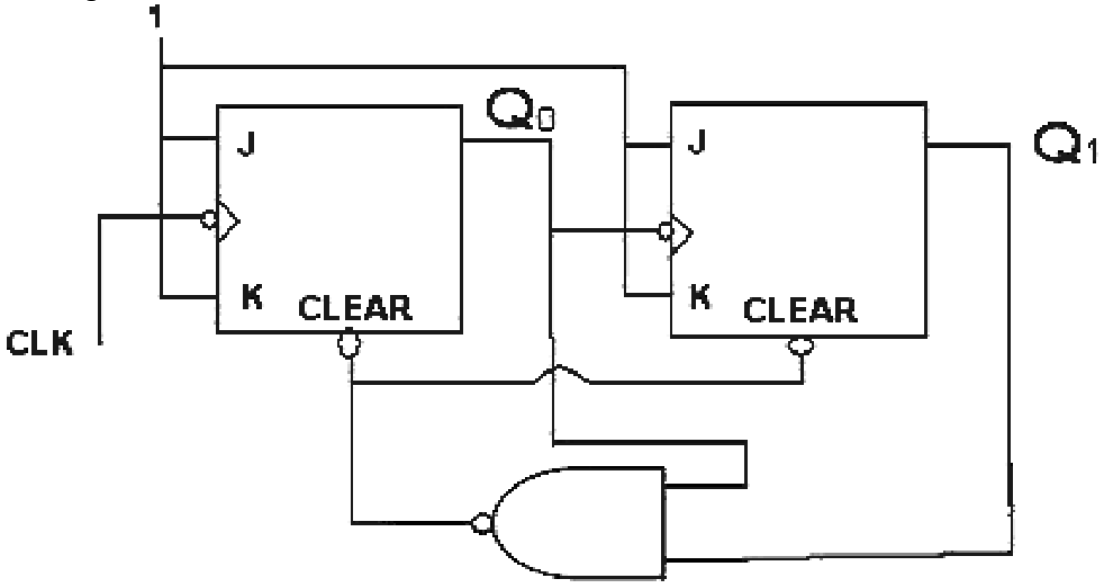
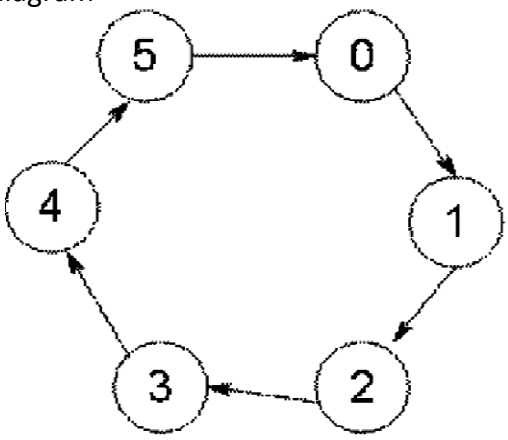


Figure 4.65 circuit diagram of Asynchronous Mod-3 Counter

➢ **Design procedure of Asynchronous Mod-6 Counter:**

- We can design the MOD 6 counter using 3 FFs as 6 is less than 8 i.e. 2^3 and greater than 4. We can see directly that as we have to reset the counter only after 5 i.e. when output is 6 (i.e. Q2Q1Q0=110) we reset the counter and hence we need to reset only when we have Q0= 0, Q1=1 & Q2=1. Now firstly design MOD-8 counter using 3 FFs and then take NAND of Q1 & Q2 and feed the output to CLEAR of all the FFs.

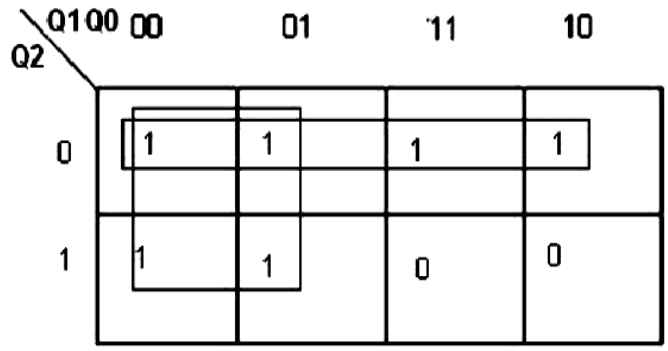
- We firstly draw the state diagram



- And now we draw the table to represent the desired output of the combinational circuit to reset FFs as:

Q2	Q1	Q0	OUTPUT
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

- And using K-map we get the combinational circuit as



And the equation we get is as
Z= Q1 bar • Q2 bar
i.e. Z= NAND (Q1, Q2)

$\mathbf{z = \overline{Q_2} + \overline{Q_1} = \overline{Q_2 Q_1}}$ (Which is NAND of Q_2 and Q_1) as per De-Morgan's Theorem

- And the complete circuit is as:

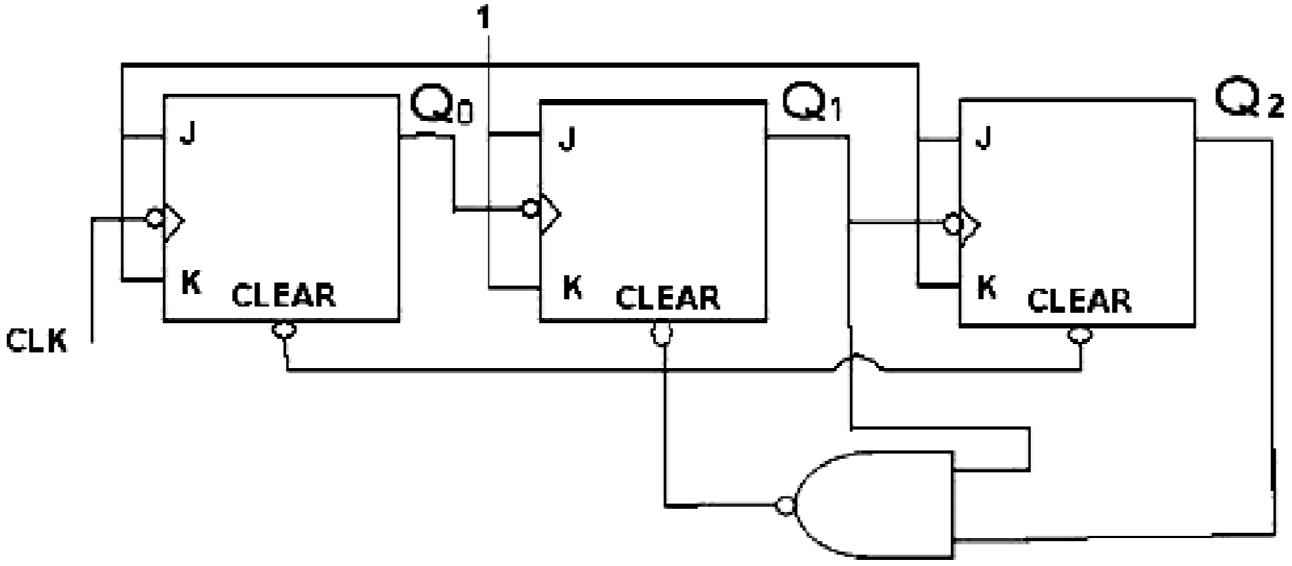


Figure 4.66 circuit diagram of Asynchronous Mod-6 Counter

➢ **Asynchronous Mod-6 Counter with wave form:**

- Mod-6 counter will count up to 6 states, starting from 000 to 101. Flip Flop used here is negative edge triggered JK Flip Flop.

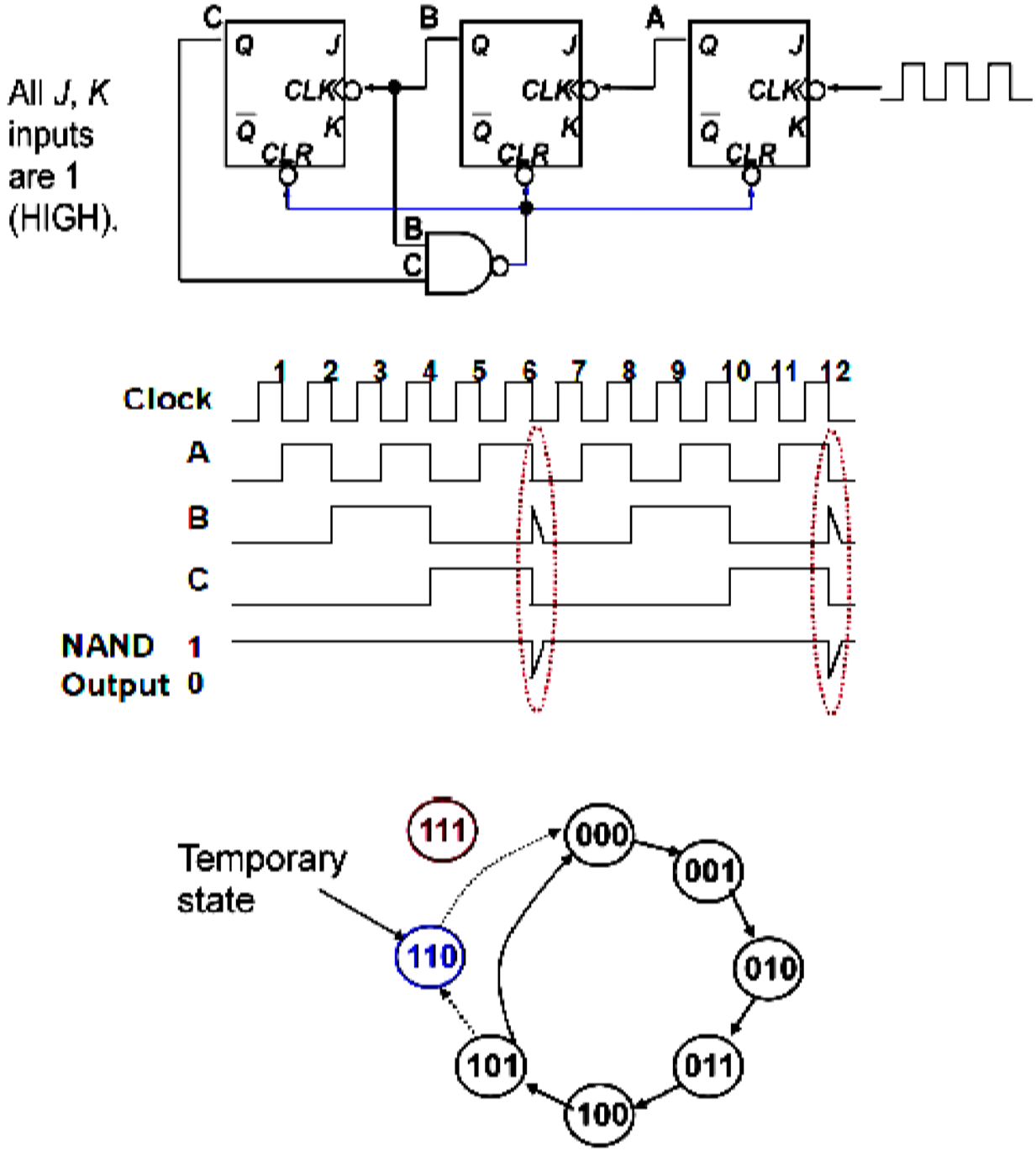


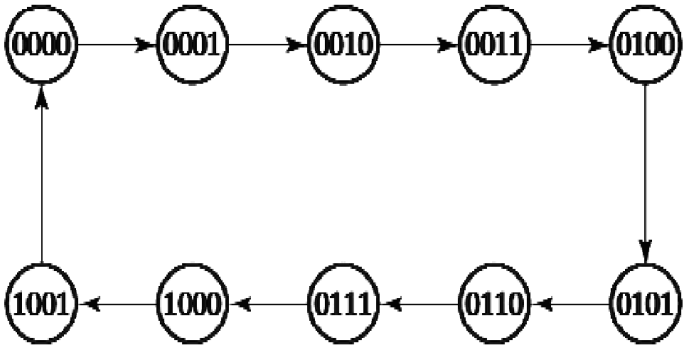
Figure 4.67 Mode-6 Ripple Counter

- **Design procedure of Asynchronous Mod-10 Counter: (BCD Counter) / (Decade Counter)**

- We can design the MOD 10 counter using 4 FFs as 10 is less than 16 i.e. 2^4 and greater than 8. We can see directly that as we have to reset the counter only after 9 i.e. when output is 10 (i.e. Q3Q2Q1Q0=1010) we reset the counter and hence we need to reset only when we have Q0= 0, Q1=1, Q2=0 & Q3=1. Now firstly design MOD-16 counter using 4 FFs and then take NAND of Q0, Q2 & Q3 and feed the output to CLEAR of all the FFs.

- A mod-10 counter is also referred as "Decade Counter". A decade counter is any counter that has 10 distinct states, no matter what is the sequence. A decade counter as shown in figure which counts sequence from 0000 (Zero) to 1001 (Decimal Nine) is also called BCD counter because it uses only 10 BCD group codes.

- State diagram for BCD Counter is as shown in below figure.



- As shown in figure output Q3=A, Q2=B, Q1=C and Q0=D

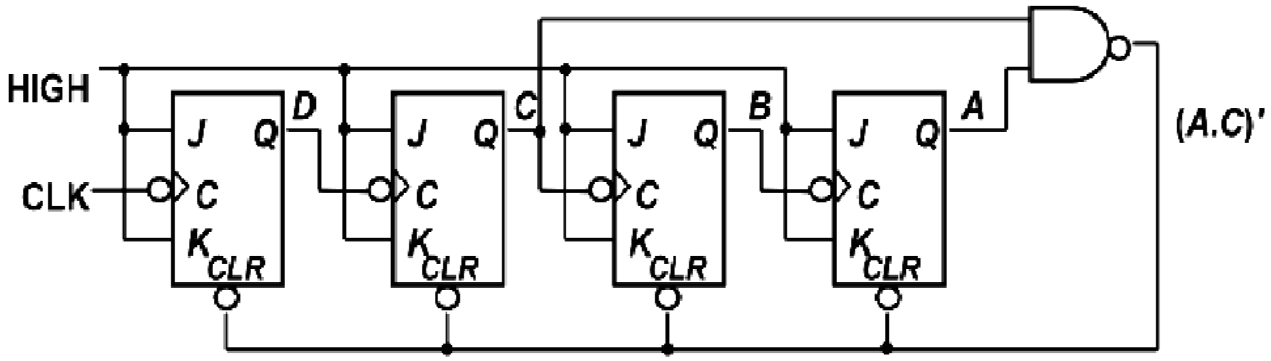


Figure 4.68 circuit diagram of Asynchronous Mod-10 Counter

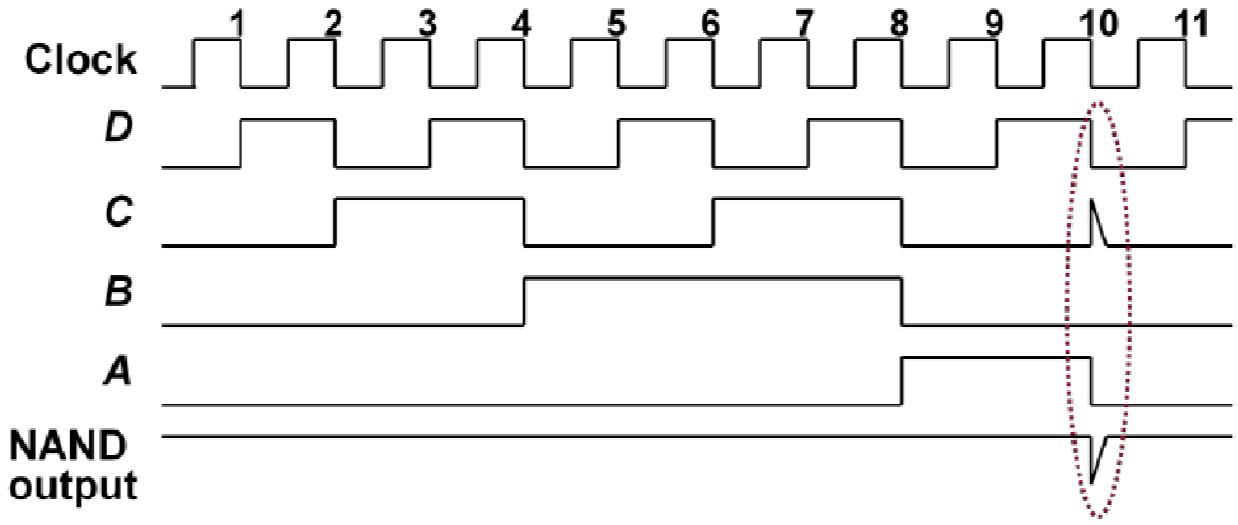


Figure 4.69 wave form of Asynchronous Mod-10 Counter

4.4.2 Synchronous Counters:

- This counter counts the number of clock pulses applied to it. In synchronous counter external clock is given to all the Flip Flop simultaneously. All the Flip Flops change the states in exact synchronism with clock pulse. So desire inputs are required to be found. Desired Inputs can be found using Excitation table of respective flip flop used in counter design. Output frequency of last flip flop is 1/n of the applied clock frequency so it is called frequency divider counter.

- It is also known as **Parallel Counter** because all the Flip Flop responds to the external input pulse simultaneously in synchronism.

- This counter can be designed using T Flip Flop or JK Flip Flop easily. A counter may count up or count down or count up and down depending on the input control. So a counter may be **Up counter** or a **Down counter**.

- The number of states through which counter passes before returning to the starting state is called **modules** of the counter (or **mod** of counter). Hence modules or mod of a counter is equal to the number of states (counts), a counter can store (including zero state). If n is number of Flip Flops used in counter then mod of the counter equals to 2^n.

- For example a: A two bit counter which requires two Flip Flop has four states 00, 01, 10 and 11. So it is called a Mod-4 counter. It divides the input clock frequency by 4. So it is also called **divide-by-4 counter**.

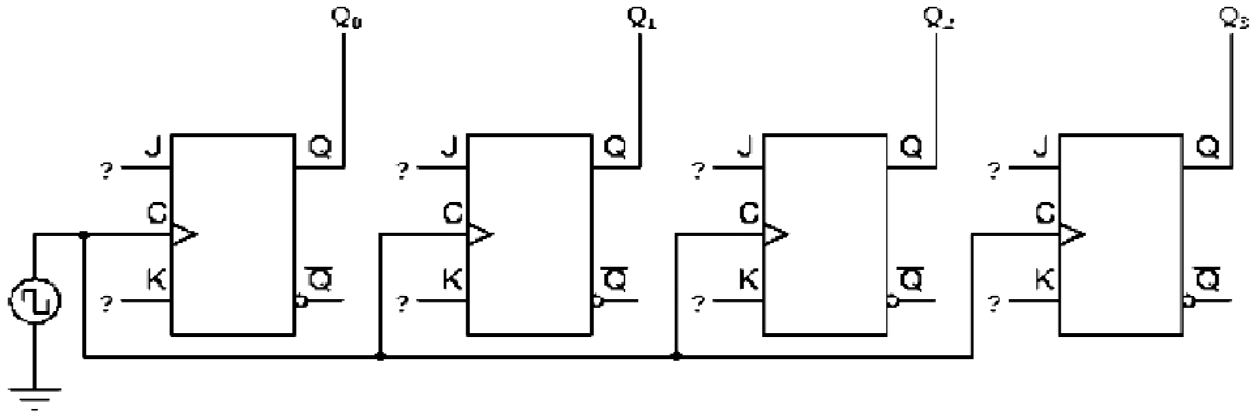


Figure 4.70 Four Bit Counter with Synchronous Clock

- **Excitation Table:**

- Characteristic tables/Truth Table provides the next state when the inputs and the present state are known. When designing digital circuits, using flip-flops one is usually given the present state and the next state. So inputs are required to be derived from present state and next state. For this reason we use excitation tables, which indicate the inputs required to change output from one state to another. Excitation table is very useful in design of counters.

- **Excitation Table of S-R Flip Flop:**

Present State	Next State	Inputs	
Q	Q(t+1)	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

- **Excitation Table of D Flip Flop:**

Present State	Next State	Inputs
Q	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

- **Excitation Table of J-K Flip Flop:**

Present State	Next State	Inputs	
Q	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

- **Excitation Table of T Flip Flop:**

Present State	Next State	Inputs
Q	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

Note: X is don't care condition.

➢ **Design of Two bit synchronous Up Counter:**

- Synchronous counter, in contrast to an asynchronous counter, is one whose output bits change state simultaneously, with no ripple. This counter can be easily designed using either J-K or T Flip Flop. Positive edged triggered Flip Flop or Negative edge triggered Flip Flop can be used. Two Flip Flops are required for 2 bit counter which counts the four states 00, 01, 10, 11 and repeat the same sequence again, so it is also called Mod-4 counter. As frequency of last flip flop is ¼ of input clock frequency, it is also called divide by 4 counter. As clock is common to all the flip flops, it is also called parallel counter. Figure.2 show two bit synchronous up counter using J-K Flip Flop.

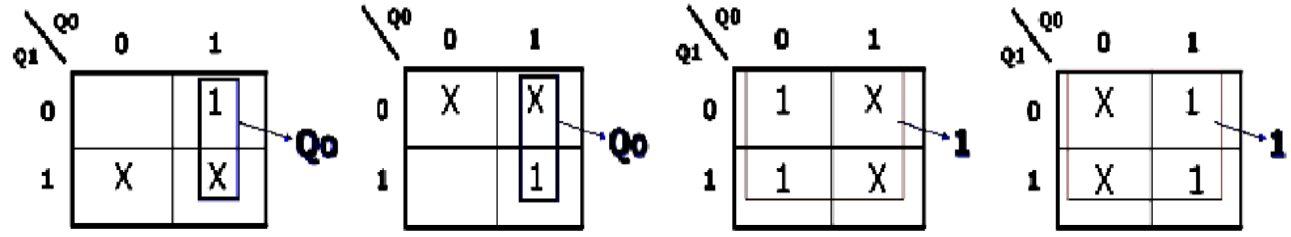
Excitation Table of J-K Flip Flop:

Present State	Next State	Inputs	
Q	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

- **Excitation Table for design of 2 bit counter:**

Present State	Next State	Inputs	
Q_1Q_0	Q_1Q_0	J_1K_1	J_0K_0
00	01	0X	1X
01	10	1X	X1
10	11	X0	1X
11	00	X1	X1

- **K-Map for inputs of both JK Flip Flops:**



$J_1 = Q_0$ $K_1 = Q_0$ $J_0 = 1$ $K_0 = 1$

- **Two bit synchronous Up Counter using Positive edge triggered J-K Flip Flop:**

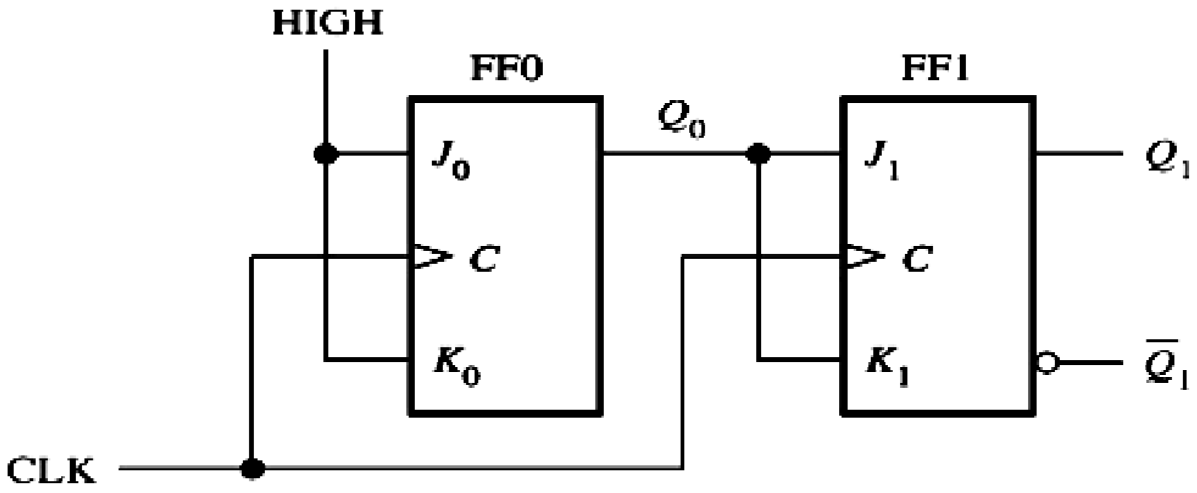


Figure 4.71 Two bit synchronous Up Counter using Positive edge triggered J-K Flip Flop

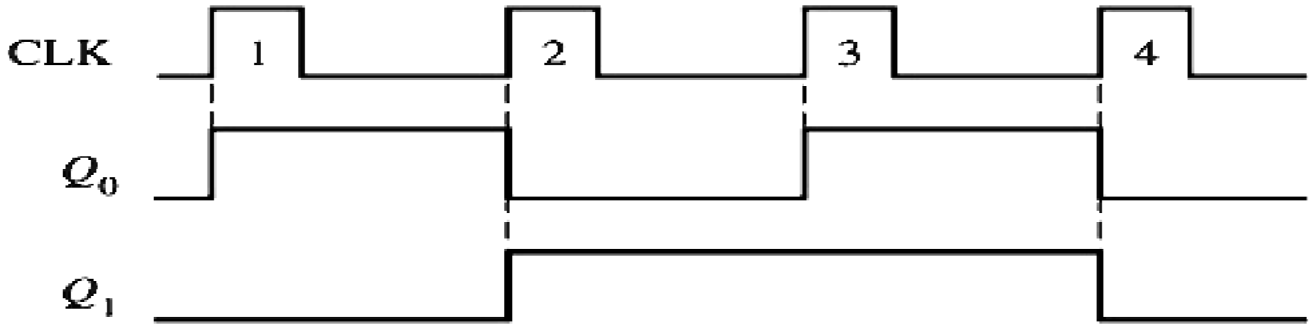


Figure 4.72 Timing Diagram for Two bit synchronous Up Counter using Positive edge triggered J-K Flip Flop

- **Two bit synchronous Up Counter using Negative edge triggered J-K Flip Flop:**

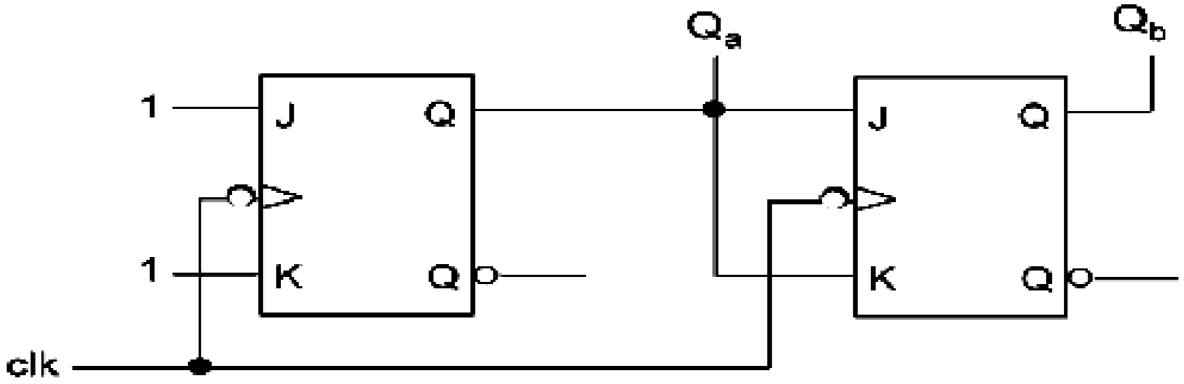


Figure 4.73 Two bit synchronous Up Counter using Negative edge triggered J-K Flip Flop

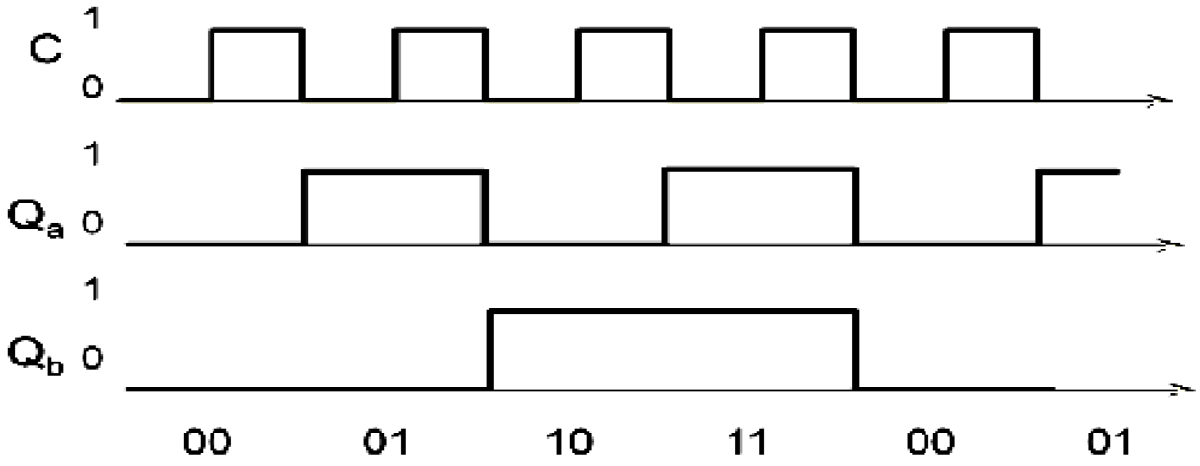


Figure 4.74 Timing Diagram for Two bit synchronous Up Counter using Negative edge triggered J-K Flip Flop

➢ **Two bit synchronous Up Counter using Negative edge triggered T Flip Flop (T flip is equivalent to JK Flip flop with J and K inputs are connected to gather) :**

- **Excitation Table of T Flip Flop:**

Present State	Next State	Inputs
Q	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

- **Excitation Table of T Flip Flops used in 2 bit counter:**

Present state		Next state		Flip-flop inputs	
A1	A0	A1+	A0+	TA1	TA0
0	0	0	1	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1

- **Equations of both the Flip Flop Using K-Map:**

$$JA_1 = KA_1 = TA_1 = A_0$$
$$JA_0 = KA_0 = TA_0 = 1$$

- **Circuit Design from Equations:**

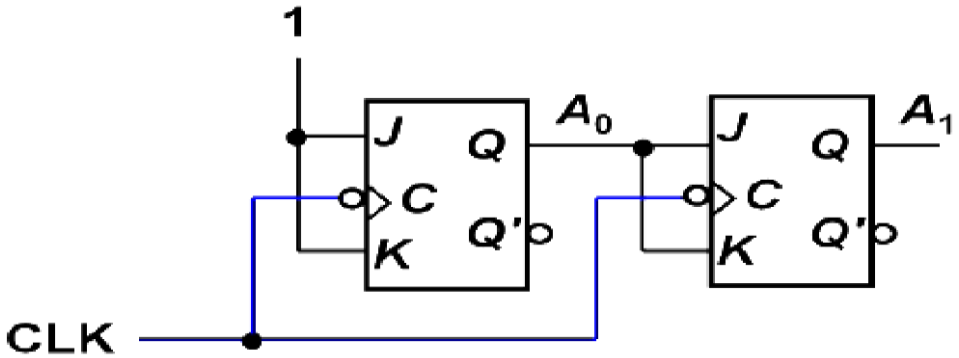


Figure 4.75 Two bit synchronous Up Counter with Negative edge triggered Flip Flop

➢ **Two Bit Synchronous Up-Down Counter:**

- In Synchronous counter clock is common to all the Flip Flops. Output frequency of last flip flop is ¼ of the applied clock frequency so it is called frequency divider counter. In the same way a four bit counter is known as divide by 16 counters. If number of Flip Flops are n the maximum counting capability is given as $N = 2^n - 1$.

- Figure 4.76 shows Up/Down Counter using **positive edge triggered** JK Flip Flop. It is programmable counter because when control signal **Up/$\overline{Down}$ = 1,** it work as Up Counter and when control signal **Up/$\overline{Down}$ = 0,** it works as Down Counter.

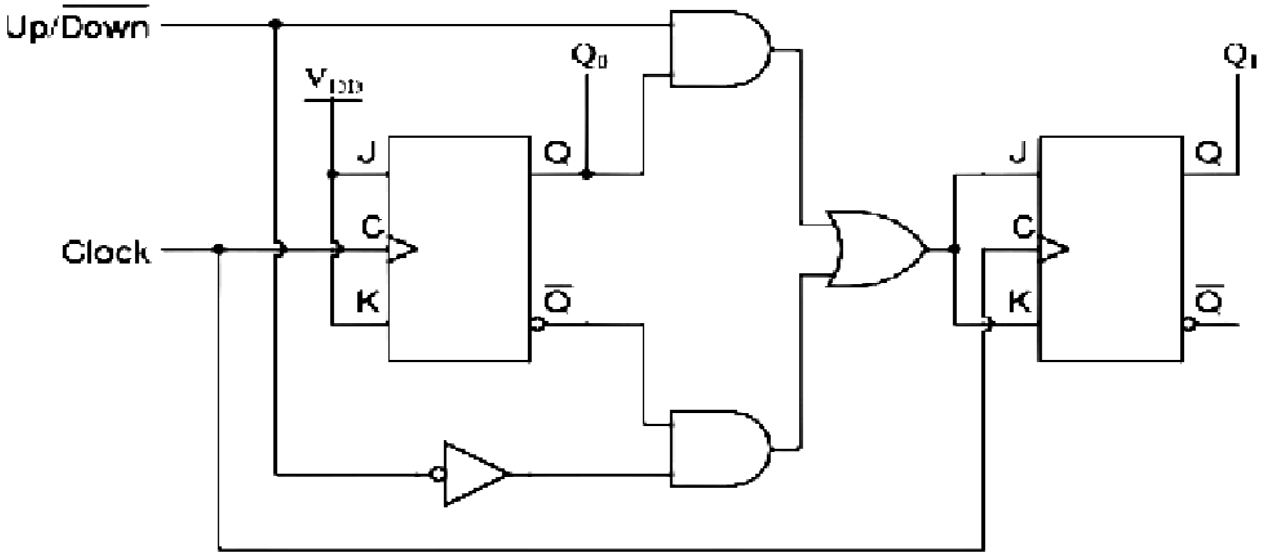


Figure 4.76 Two Bit Synchronous Up-Down Counter

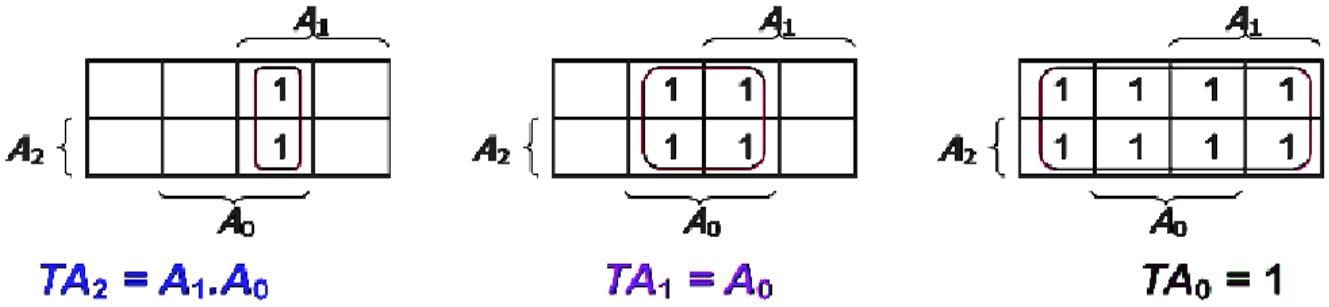
➢ **Three bit synchronous Up Counter using Positive edge triggered T Flip Flop (or using J-K Flip Flop with identical J, K inputs)**

- Synchronous counter, in contrast to an asynchronous counter, is one whose output bits change state simultaneously, with no ripple. This counter can be easily designed using either J-K or T Flip Flop. Positive edged triggered Flip Flop or Negative edge triggered Flip Flop can be used.

- Three Flip Flops are required for 3 bit counter which counts the eight states 000, 001, 010, 011, 100, 101, 110, 111 and repeat the same sequence again, so it is also called Mod-8 counter. As frequency of last flip flop is 1/8 of input clock frequency, it is also called divide by 8 counter. As clock is common to all the flip flops, it is also called parallel counter.

- Figure 4.77 shows state table and K-map derived for T Flip Flops. Equations derived from K-map are the required inputs of T Flip Flop. If JK Flip Flop is used then T Flip Flop can be derived from JK Flip Flop by shorting both J and K inputs.

Present state			Next state			Flip-flop inputs		
A_2	A_1	A_0	A_2^+	A_1^+	A_0^+	TA_2	TA_1	TA_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1



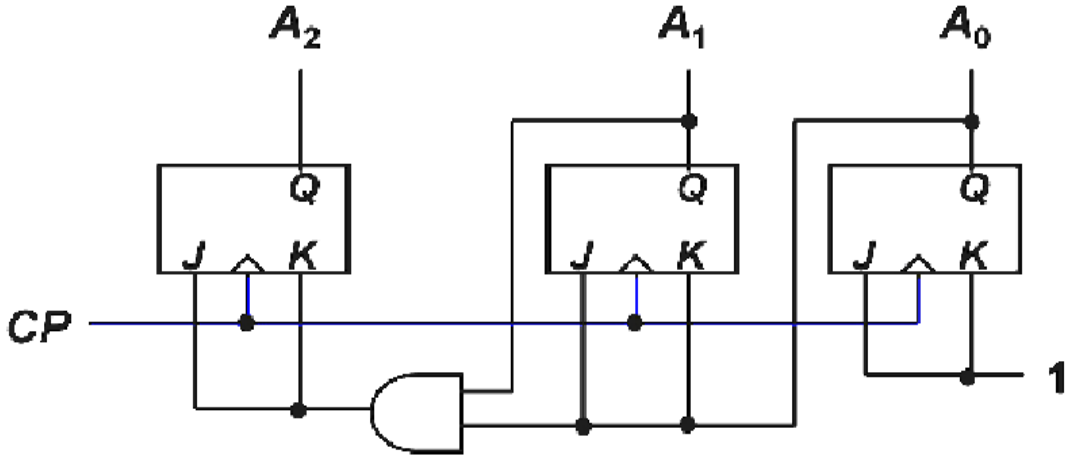


Figure 4.77 Three bit synchronous Up Counter with state stable, K-Map & design Equation

➢ **Design of Three Bit Synchronous Up-Down Counter:**

- In Synchronous counter clock is common to all the Flip Flops. Output frequency of last flip flop is 1/8 of the applied clock frequency so it is called frequency divider counter. In the same way a four bit counter is known as divide by 16 counters. If number of Flip Flops are n the maximum counting capability is given as $N = 2^n - 1$. Here it counts from 000 to 111 for up counting and 111 to 000 for down counting.

- Counter can be designed using T Flip Flop also. Figure 4.78 shows Up/Down Counter using **positive edge triggered T Flip Flop**. It is programmable counter because when control signal **Up/$\overline{Down}$ = 1,** it work as Up Counter and when control signal **Up/$\overline{Down}$ = 0,** it works as Down Counter.

- **Design equations:**

$TQ_0 = 1$
$TQ_1 = (Q_0.Up) + (Q_0'.Up')$
$TQ_2 = (Q_0.Q_1.Up) + (Q_0'.Q_1'.Up')$

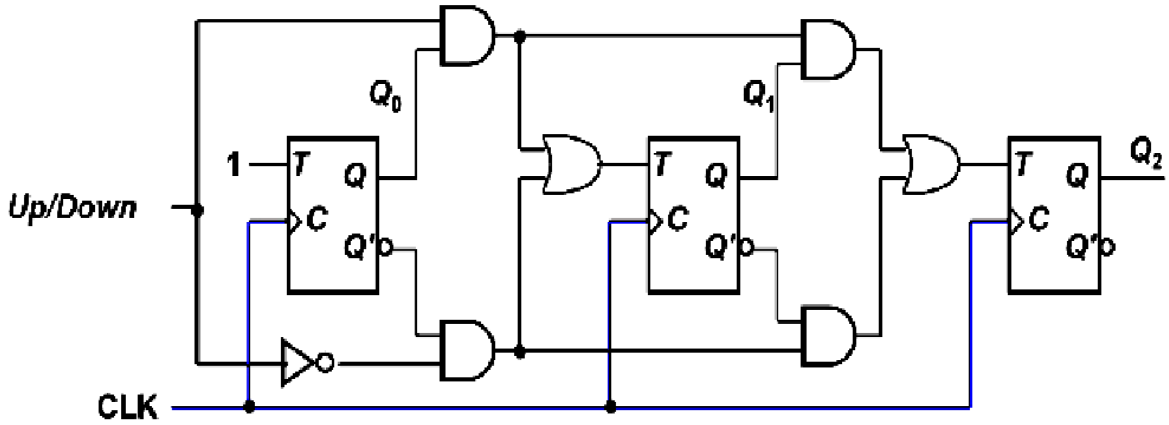


Figure 4.78 Three Bit Synchronous Up-Down Counter

➢ **Four bit synchronous Up Counter using Positive edge triggered T Flip Flop (or using J-K Flip Flop with identical J, K inputs)**

- Synchronous counter, in contrast to an asynchronous counter, is one whose output bits change state simultaneously, with no ripple. This counter can be easily designed using either J-K or T Flip Flop. Positive edged triggered Flip Flop or Negative edge triggered Flip Flop can be used. Four Flip Flops are required for 4 bit counter which counts the eight states from 0000 to 1111 and repeat the same sequence again, so it is also called Mod-16 counter. As frequency of last flip flop is 1/16 of input clock frequency, it is also called divide by 16 counter. As clock is

common to all the flip flops, it is also called parallel counter.

- Figure 4.79 shows circuit diagram and design equations derived from K-map, required at inputs of T Flip Flop. If JK Flip Flop is used then T Flip Flop can be derived from JK Flip Flop by shorting both J and K inputs.

- **Design Equations:**

$(JA_3 = KA_3) = TA_3 = A_2.A_1.A_0$
$(JA_2 = KA_2) = TA_2 = A_1.A_0$
$(JA_1 = KA_1) = TA_1 = A_0$
$(JA_0 = KA_0) = TA_0 = 1$

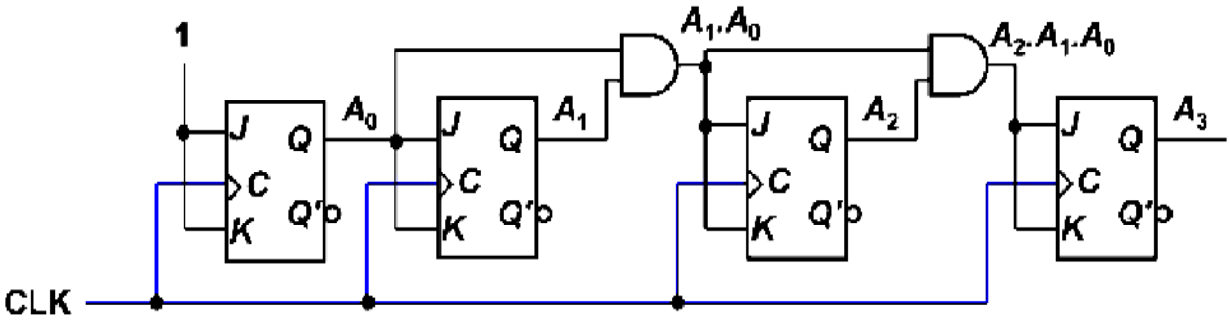


Figure 4.79 Four bit synchronous Up Counter

➢ **Design of Synchronous Decade/BCD counter:**

- Synchronous counter, in contrast to an asynchronous counter, is one whose output bits change state simultaneously, with no ripple. This counter can be easily designed using either J-K or T Flip Flop. Positive edged triggered Flip Flop or Negative edge triggered Flip Flop can be used.

- We can design the MOD 10 counter using 4 FFs as 10 is less than 16 i.e. 2^4 and greater than 8. A mod-10 counter is also referred as "Decade Counter". A decade counter is any counter that has 10 distinct states, no matter what is the sequence. A decade counter as shown in figure which counts sequence from 0000 (Zero) to 1001 (Decimal Nine) is also called BCD counter because it uses only 10 BCD group codes.

- Figure 4.80 shows circuit diagram and design equations derived from K-map, required at inputs of T Flip Flop. If JK Flip Flop is used then T Flip Flop can be derived from JK Flip Flop by shorting both J and K inputs.

Clock pulse	Q_3	Q_2	Q_1	Q_0
Initially	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10 (recycle)	0	0	0	0

- **Design Equations:**

$T_0 = 1$
$T_1 = Q_3'.Q_0$
$T_2 = Q_1.Q_0$
$T_3 = Q_2.Q_1.Q_0 + Q_3.Q_0$

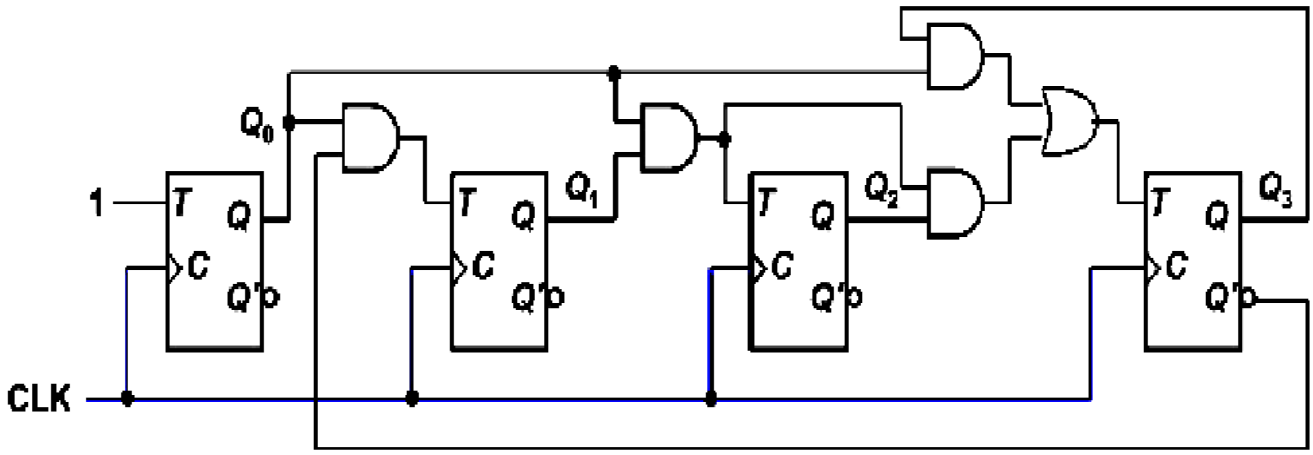


Figure 4.80 Synchronous Decade/BCD counter

- **Ring Counter & Johnson Counter:**

- **Ring Counter:**

- One flip-flop (stage) required for each state in the sequence.
- D Flip Flop is preferred for Ring Counter.

- The output of the last stage is connected to the D input of the first stage.
- An n-bit ring counter cycles through n states.
- No decoding gates are required, as there is an output that corresponds to every state the counter is in.
- Preset Control signal of 1st Flip Flop is connected to 0 to set the Flip Flop.
- Clear control signal of all the remaining Flip Flop except 1st flip flop is connected to 0 to reset the flip flop.
- Normally Preset and Clear control signals are connected to 1.

- A 6-bit (MOD-6) ring counter:

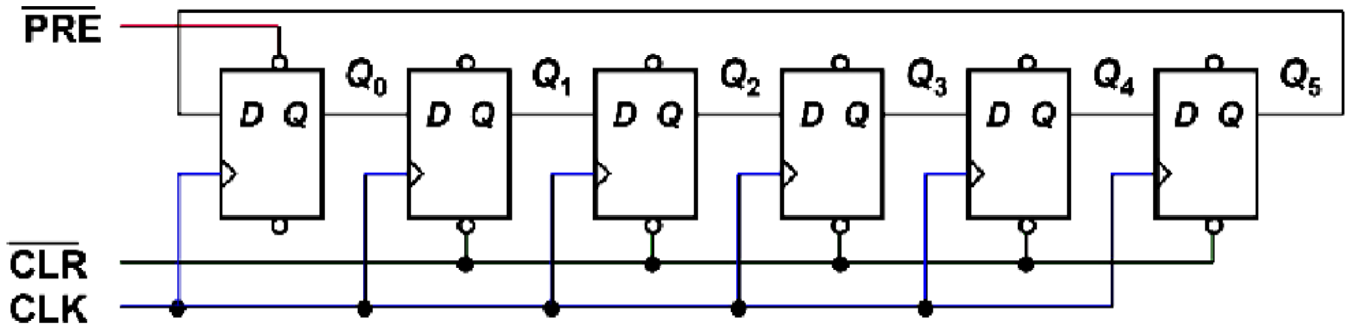


Figure 4.81 6-bit (MOD-6) Ring Counter

- State Table & State diagram of 6 bit Ring Counter:

Clock	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5
→0	1	0	0	0	0	0
1	0	1	0	0	0	0
2	0	0	1	0	0	0
3	0	0	0	1	0	0
4	0	0	0	0	1	0
5	0	0	0	0	0	1

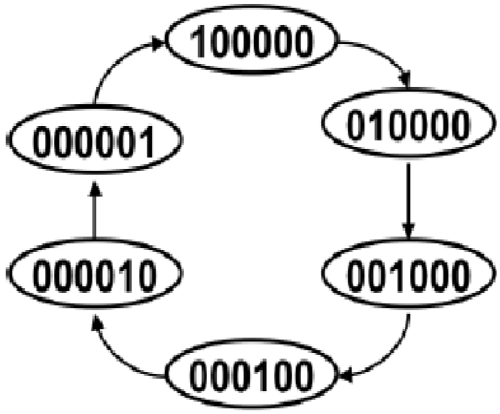


Figure 4.82 State Table & State Diagram of 6-bit Ring Counter

➢ **Johnson Counter:**

- The complement of the output of the last stage is connected back to the D input of the first stage.
- Also called the twisted-ring counter.
- Require fewer flip-flops than ring counters but more flip-flops than binary counters.
- An n-bit Johnson counter cycles through 2n states.
- Require more decoding circuitry than ring counter but less than binary counters.

- A 4-bit (MOD-8) Johnson counter.

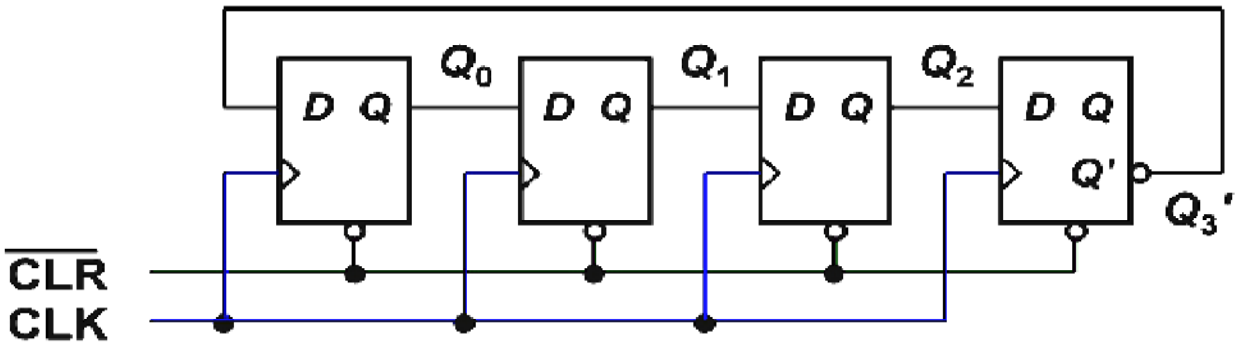


Figure 4.83 4-bit (MOD-8) Johnson counter

- State table and state diagram of Johnson's counter:

Clock	Q_0	Q_1	Q_2	Q_3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

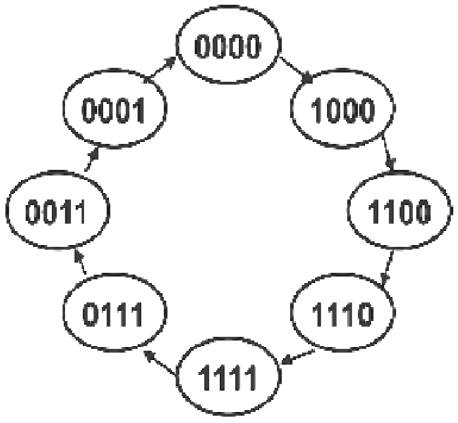


Figure 4.84 State table & state diagram of Johnson's counter

5.1 THE STATE MACHINE:

5.1.1 State Machine Definition

- **State**: Each of the outputs of a sequential circuit is called its state. For e.g. a flip-flop have 2 states: 0 & 1.

- **State Table**: The time sequence of inputs, outputs & the flip-flop states may be written in a table called state table. It is also called transition table.

Present State	Inputs	Next State	Outputs

Fig. 5.1: State table

- **State diagram**: The information available in a state table may be represented graphically in a state diagram. In this diagram, a state is represented by a circle & the transition between states is indicated by directed lines connecting the circles.

- **State equation**: It is an algebraic expression that specifies the conditions for a flip-flop state transition. It is also known as application equation.

- The left side of the equation denotes the next state of a flip-flop & the right side, a Boolean function that specifies the present state condition that makes the next state equal to 1. The state equation is derived directly from a state table.

- **Input & output equations**: The part of the circuit that generates the inputs to the flip-flops is described algebraically by a set of Boolean functions called flip-flop input function.

- The part of combinational circuit that generates external outputs is described algebraically by the circuit output functions.

  For ex: JA = BC'x + y ; KA = B + x are input functions
  y = AB'x is output function

- **State Machine:** State machine also called a sequential machine is a system that can be described in terms of set of states that the system may enter. Once in a particular state, the system may be capable of remaining in that state for some finite period of time even if the system inputs change.

As the system progresses from one state to another, the next state reached depends on the inputs & present state. The outputs also depend on inputs & present state.

- A set of n flip-flops can produce 2^n possible unique output codes and thus could have 2^n possible unique states of existence. Hence n lines driven by combinational variables can also take on 2^n different unique codes, but these are not states since they cannot exist independently of the input variable. A set of n flip-flops can exist in a particular state different from the set of input values, and this is the requirement of state machine. They are also known as Finite State Machines (FSM).

5.1.2. Classification of State Machines

- **State machines can be classified into three types:**

  (i) Synchronous state machine
  (ii) Mainly synchronous state machine
  (iii) Asynchronous state machine

  If a state machine changes state in response to clock & all inputs are synchronous, we will classify that circuit as synchronous system.

  If the state changes occur in response to clock but one or more inputs are not clock driven, the machine will be called mainly synchronous system.

  If the state changes are input-driven rather than clock-driven, the system is asynchronous one.

- **Advantages of state machine approach:**

  (i) The state machine design method leads to minimal design. This method results in the minimum no. of flip-flops & it can minimize other circuitry in the system as well.

  (ii) It is a well-developed, orderly procedure that anticipates & solves commonly occurring problems like unwanted narrow pulse or glitches & occasional oscillations can be removed.

  (iii) It reduces the time taken to debug the implemented hardware & can be applied to the solution of a wide variety of practical circuit problems.

5.1.3. General Model of a Sequential Machine.

- A sequential machine or a state machine must possess memory capability. Generally, clocked flip-flops are used as storage elements. The code that defines each state then corresponds directly to the code contained by the flip-flops.

- There are basically 2 models used of designing state machine:

  1. Mealy machine
  2. Moore machine

- A Mealy machine is the one in which the output depends on the present state as well as the inputs of the circuit. The input forming logic (IFL) and the output forming logic (OFL) are made up of combinational logic circuits. The memory section contains the state of the system. A path is provided from memory output to the IFL.

- Both input signals and the present state signals drive the IFL to determine the next state of the system. The outputs are determined by the present state & system inputs. The general model of a Mealy machine is shown as below:

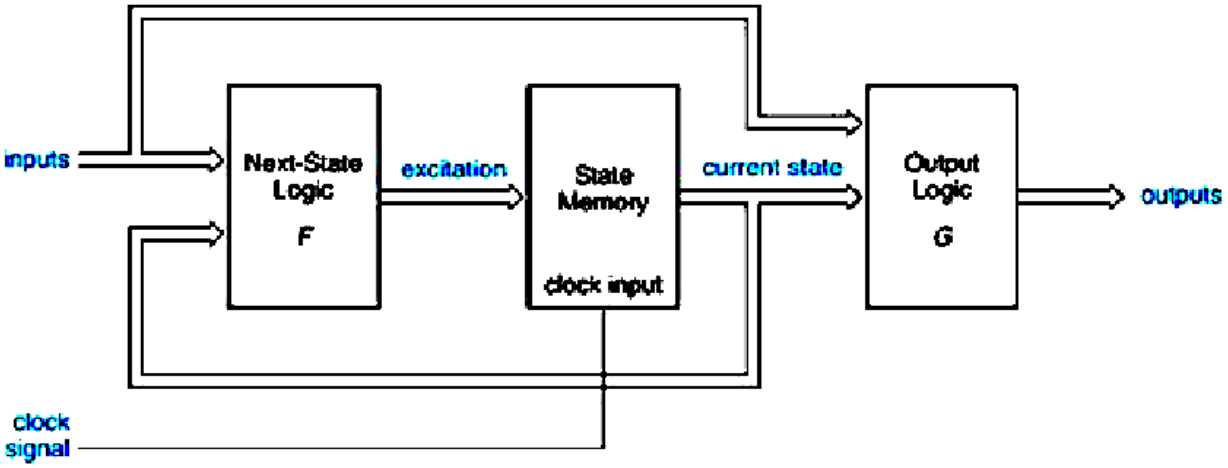


Fig. 5.2: Mealy machine

- A Moore machine is the one in which the output depends only on the present state of the system. The only difference between Mealy & Moore machine is that in Moore machine, the outputs are determined only by the present state. There is no effect of input directly on the output state. The general model of Moore machine is shown as below:

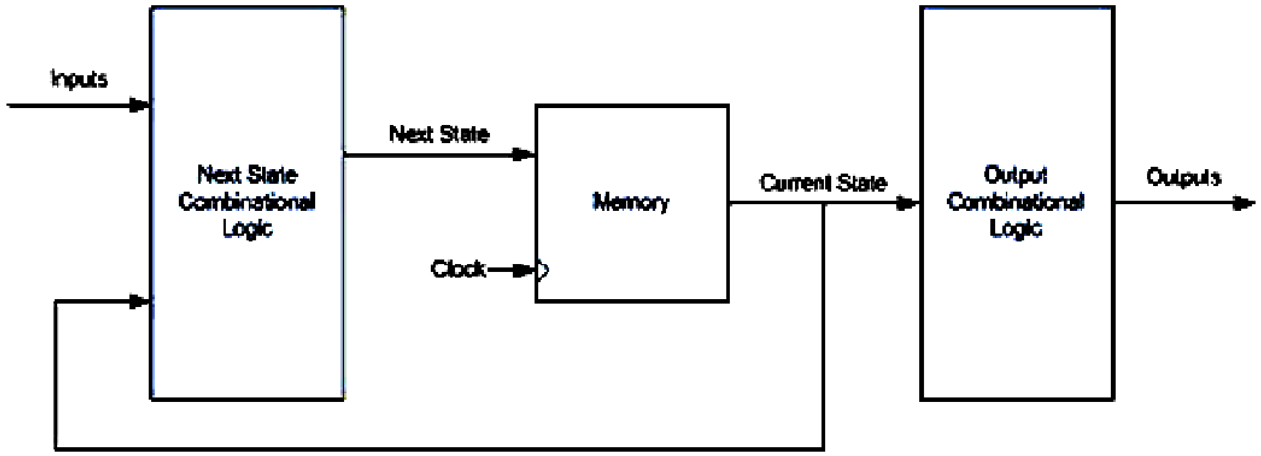


Fig 5.3: Moore machine

- **Compare Mealy & Moore State Machines:**

Moore Machine	Mealy Machine
The output depends only the present state. $Z(t)= g\{s(t)\}$	The output depends on the present state as well as input. $Z(t)= g\{s(t),x(t)\}$
Input changes do not affect the output.	Input changes may affect the output of the circuit.
It requires more number of states for implementing the same function.	It requires less number of states for implementing same function.
It requires more hardware to solve any problem.	It requires less hardware.
Problems like input transients & glitches do not affect the output.	Problem like input transients & glitches are directly conveyed to the output.

5.2 Excitation Tables for Flip-Flops

- The *characteristic table* defines the logical properties of flip-flop & completely characterizes its operation. It defines the next state when the inputs & present states are known.

- The table which gives the flip-flop input condition that will cause the required transition from present state to next state is known as *excitation table*.

(a) *JK* Flip-Flop

J	K	$Q(t+1)$	Operation
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	$\overline{Q}(t)$	Complement

(b) *SR* Flip-Flop

S	R	$Q(t+1)$	Operation
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	?	Undefined

(c) *D* Flip-Flop

D	$Q(t+1)$	Operation
0	0	Reset
1	1	Set

(d) *T* Flip-Flop

T	$Q(t+1)$	Operation
0	$Q(t)$	No change
1	$\overline{Q}(t)$	Complement

Fig. 5.4: Characteristic tables for flip-flops

Excitation tables:

(i) SR flip-flop:

Qn	Qn+1	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

(ii) JK flip-flop:

Qn	Qn+1	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

(iii) D flip-flop:

Qn	Qn+1	D
0	0	0
0	1	1
1	0	0
1	1	1

(iv) T flip-flop:

Qn	Qn+1	T
0	0	0
0	1	1
1	0	1
1	1	0

- Here, X represents don't-care condition which means the input can either be 1 or 0.

5.3 Analysis Procedure of Sequential Circuits

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

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

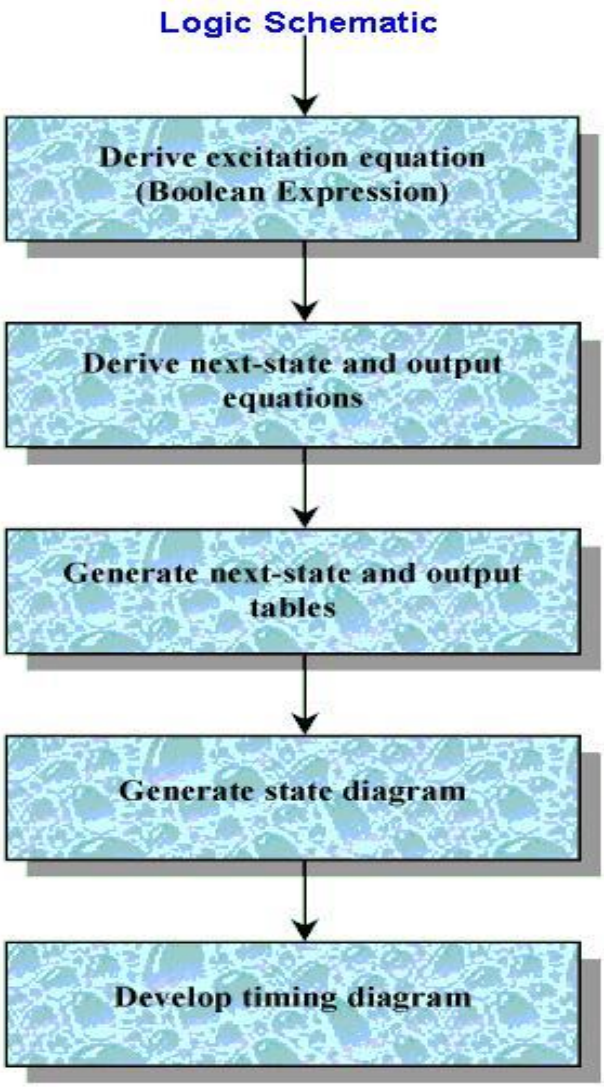


Fig. 5.5: Analysis procedure of sequential machine

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

6.1 Classification of Sequential Circuit

- Sequential circuits are divided into two main types: synchronous and asynchronous. Their classification depends on the timing of their signals.
- **Synchronous sequential circuits** change their states and output values at discrete instants of time, which are specified by the rising and falling edge of a free-running clock signal.
- The memory elements used in synchronous sequential circuits are usually flip-flops. These circuits are binary cells capable of storing one bit of information. Synchronous sequential circuits are also known as clocked sequential circuits.
- In **asynchronous sequential circuits**, the transition from one state to another is initiated by the change in the primary inputs; there is no external synchronization. The memory commonly used in asynchronous sequential circuits is time-delayed devices, usually implemented by feedback among logic gates.
- Thus, asynchronous sequential circuits may be regarded as combinational circuits with feedback. Because of the feedback among logic gates, asynchronous sequential circuits may, at times, become unstable due to transient conditions. The instability problem imposes many difficulties on the designer. Hence, they are not as commonly used as synchronous systems.

6.2 Design Procedure for Synchronous Sequential Circuits

- The design of a synchronous sequential circuit starts from a set of specifications and culminates in a logic diagram or a list of Boolean functions from which a logic diagram can be obtained. In contrast to a combinational logic, which is fully specified by a truth table, a sequential circuit requires a state table for its specification.
- The first step in the design of sequential circuits is to obtain a state table or an equivalence representation, such as a state diagram.
- A synchronous sequential circuit is made up of flip-flops and combinational gates. The design of the circuit consists of choosing the flip-flops and then finding the combinational structure which, together with the flip-flops, produces a circuit that fulfills the required specifications.
- The number of flip-flops is determined from the number of states needed in the circuit.
- The recommended steps for the design of sequential circuits are set out below.

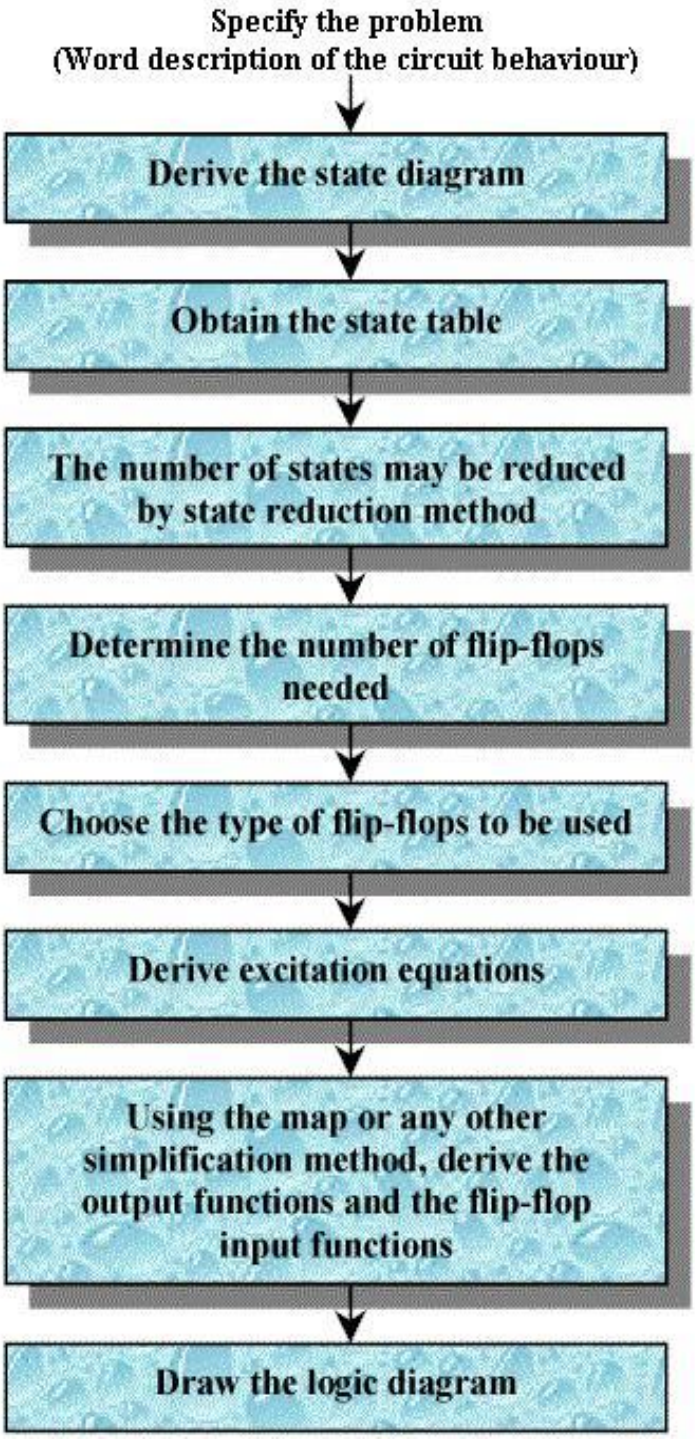


Fig. 6.1: Design procedure

6.3 Sequential Counters

- Sequential counters are state machines with no inputs other than the preset inputs that initialize the system. The sequential counter is used to generate some fixed sequence of states in a cyclic manner. This particular type of state machine has a next state that is determined by the present state only. The output may directly be taken from the flip-flops, eliminating the need of OFL. General block diagram is as shown below:

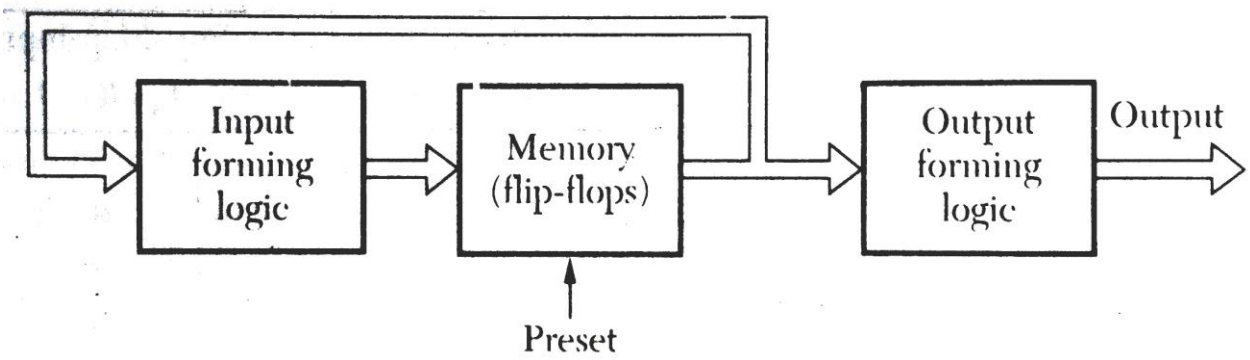


Fig. 6.2: Block diagram of sequential counter

- For example, the sequence of states desired might be 000, 011, 101, 111, 100, 000, 011 . . .

6.4 State Changes Referenced to Clock

- The flip-flops of a state machine change to a new value as a result of clock transition. At this transition, the information represented by the flip-flop inputs is set into the flip-flops. There is a requirement of data-to-clock setup time so that the inputs applied to the flip-flop inputs are stable before clock pulse arrives.

- Incorrect information is set into the flip-flops if we do not observe this specification. Thus the output of the flip-flop will be an ambiguous (not clear or decided) state. This must be avoided to achieve reliable state machine operation.

- For synchronous input signals, driven by a clock transition, the state change must take place long enough after the clock transition to allow the flip-flop inputs to settle & the data-to-clock setup time to be satisfied. There are 2 methods used to delay the state changes sufficiently:

    1. Alternate State Transition Method (AST)
    2. Delayed State Transition Method (DST)

- In the AST scheme, the state changes are caused to occur on the alternate clock transition to input data changes. If the data changes are driven on the positive clock transition, the state changes must take place on the negative clock transition. Data changes on the negative clock transition dictate positive clock transition changes.
- For a 50% duty cycle clock, the AST method allows one-half of the clock period between input change & state change. Hence in equation form,

$$\frac{T}{2} > t_{cd} + t_{plFL} + t_{su}$$

- The maximum usable clock frequency can then be found as:

$$f = \frac{1}{T} < \frac{1}{2(t_{cd} + t_{plFL} + t_{su})}$$

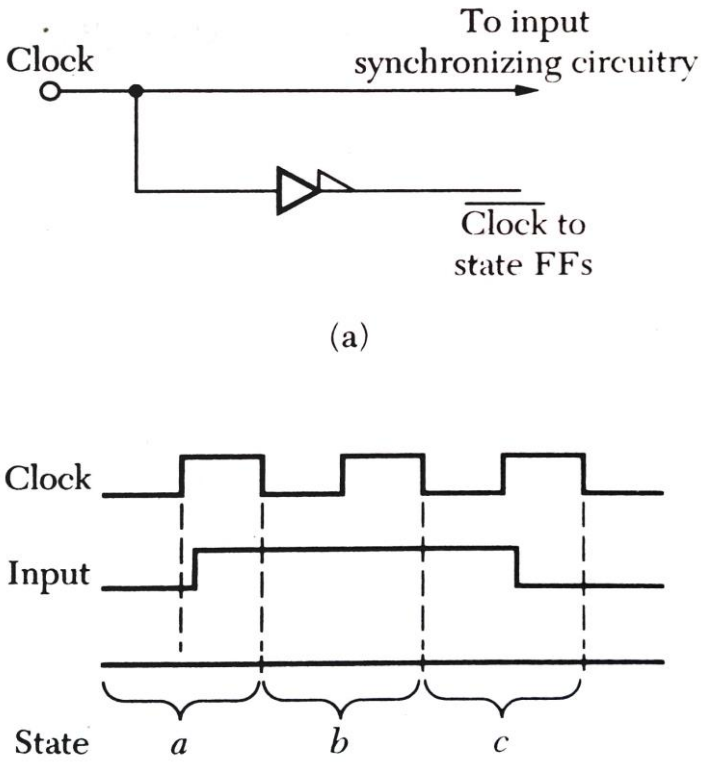


Fig. 6.3: AST method

- In the DST system, the same clock transition drives both the data changes & the state changes, but the clock signal to the state flip-flops is delayed. The number of delaying inverters used between the clock and the flip-flops is determined by two factors: the total clock delay t_t and the correct number of inversions.

- Generally the AST method is sued for synchronous state machines. This method is more popular in state machine design except for situations involving very critical timing problems.

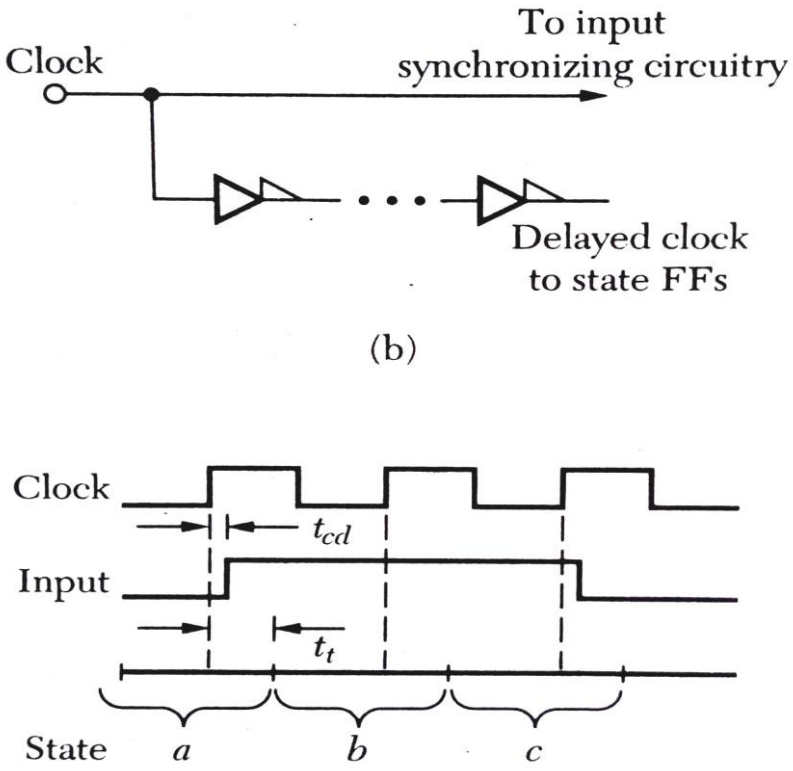


Fig. 6.4: DST method

6.5 Input Forming Logic (IFL)

The combinational circuitry that drives the state flip-flops is called the input forming logic or IFL. The IFL consists only of combinational logic. This section is driven by inputs and the outputs of the state flip-flops. Regardless of the type of flip-flop used, the IFL must generate combinational logic functions and therefore can be constructed from gates, MUXs or decoders.

a. IFL using gates

- The traditional method of IFL design uses gates. This method is very significant in applications using standard IC chips. When IFL is constructed from gates, this circuitry can be minimized by the following 2 principles:

  - Principle 1: States having the same next state for a given input condition should have logically adjacent state assignments.
  - Principle 2: States that are next states of a single state should have logically adjacent assignments.

b. IFL using direct-addressed MUX

- The IFL of the state machine can be designed with MUXs instead of gates. The general arrangement of the direct-addressed MUX system is as shown below. Although the IFL shown consists of gates & MUXs, the gates often are either very simple or not required at all.

- In this scheme, the outputs of the state flip-flops are connected to the select lines of all MUXs in parallel. Each state then addresses a different set of MUX inputs. One MUX is required for each state flip-flop and each MUX must have n select lines, where n is the number of state flip-flops. A direct-addressed MUX design requires n - 2^n : 1 MUXs to design the IFL.

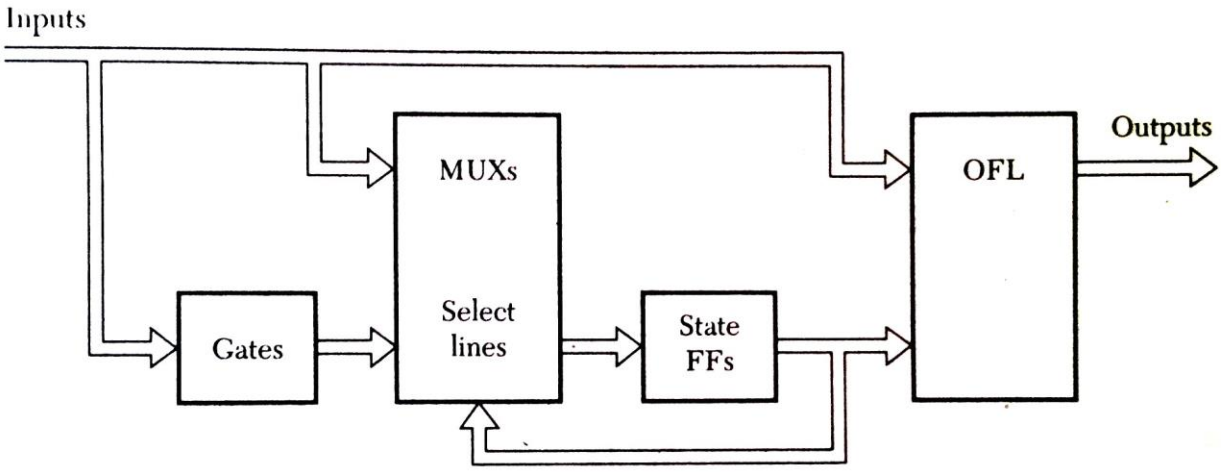


Fig. 6.5: Architecture of direct addressed MUX system

- **Advantages of designing IFL using direct-addressed MUX:**

1. The chip count can be minimized and design time is decreased.
2. No restrictions are placed on state assignment for circuit minimization, allowing more flexibility in the overall design.
3. State assignment can be used for system reliability rather than for IFL minimization.
4. This design leads to ease of troubleshooting the state machine. It is very easy to isolate problems of faulty wiring or chips with this type of system.

- **Disadvantage:**

1. Component cost may be higher which using MUX design.

6.6 Output Forming Logic (OFL)

- The combinational circuitry that the state flip-flops drive to produce the output signals is called the output forming logic of OFL. When an output is to be generated by a state machine, there are several choices of assertion time. The particular output selected will often depend on output specifications and we must design the circuit to achieve this signal.

- There are basically 2 types of outputs:

  (i) Conditional outputs
  (ii) Unconditional outputs

- A **conditional output** is the one which depend not only on the system state, but also requires certain input conditions before the output is asserted.

- An **unconditional output** is one that depends only on the state of the system. When a particular state is entered, an AND gate or IC decoder reflects this fact by asserting an output line. It is also known as immediate output.

- In OFL design, there are 2 bases upon which state assignment may be made.

  - The first relates to minimization of gates required to implement the design.
  - The second basis is elimination of output glitches.

(i) For minimizing no. of gates required to implement the design, a decoder chip is used to generate state machine outputs. Although it is often more costly, but this approach can minimize chip count especially if several outputs are generated.

- The flip-flop outputs are directly connected to the decode inputs leading to the assertion of a particular line in each state. The decoder output lines corresponding to all states producing the same output signal are ORed to generate the output.

(ii) The Output Glitch Problem:

- Glitches or slivers are very narrow pulses, often having a width of few nanoseconds, which occur during the switching of variables. They are also referred to as “runt” pulses.

- The output forming logic is driven by the state flip-flops. Although the state-changing clock transition is applied simultaneously to all state flip-flops, due to differences in clock-to-output delay times, the flip-flop outputs will not change at exactly the same time. Since these outputs drive gates or decoders, the possibility of glitches must be considered.

- There are applications in which glitches can be tolerated. When an output drives a slow-responding device, glitches may be allowable. For example, when an output is taken from a light-emitting diode (LED), a short glitch may not affect the output of the state machine. Here, glitch is tolerable and there is no requirement of further design to eliminate glitch.

- Many digital controllers require outputs to drive digital circuits such as counters o flip-flops. In such a case, the output must occur only at specified time. Here, glitch is not tolerated and glitch-free design is essential to produce a reliable system in this type of application.

- There are 2 common sources of output glitches in state machine design:

  a) The unequal switching times of the state flip-flops that drive the OFL
  b) Due to conditional outputs

6.7 Generation of A State Diagram from A Timing Chart

- A common method in describing digital systems uses a timing chart. This chart indicates the operations must take place during particular time slots in order for the system to function properly. These operations are initiated by control signals and the timing chart indicates when each control signal must be asserted.

- The state diagram is essential in the hardware design of the system and hence after the timing chart, the system design can proceed to the development of the state diagram. A trial-and-error procedure is often the most efficient method to be used.

- Let us consider the following timing chart of a simple digital controller.

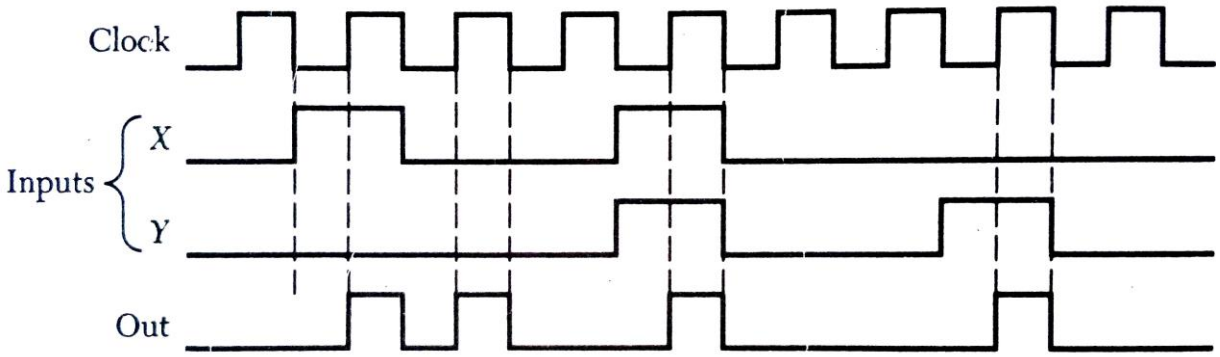


Fig. 6.6: A digital system timing chart

- Here, X and Y are the system inputs and the clock signal is also shown. The system does not give the output as long as X & Y are simultaneously 0. We have to develop a state diagram to implement this timing chart with a minimum number of states and without using counters.

  From this timing chart we can say that,

- All state changes must take place on the positive-going transitions of the clock since the inputs change on the negative transition.
- The output assertions begin at the positive transition of the clock

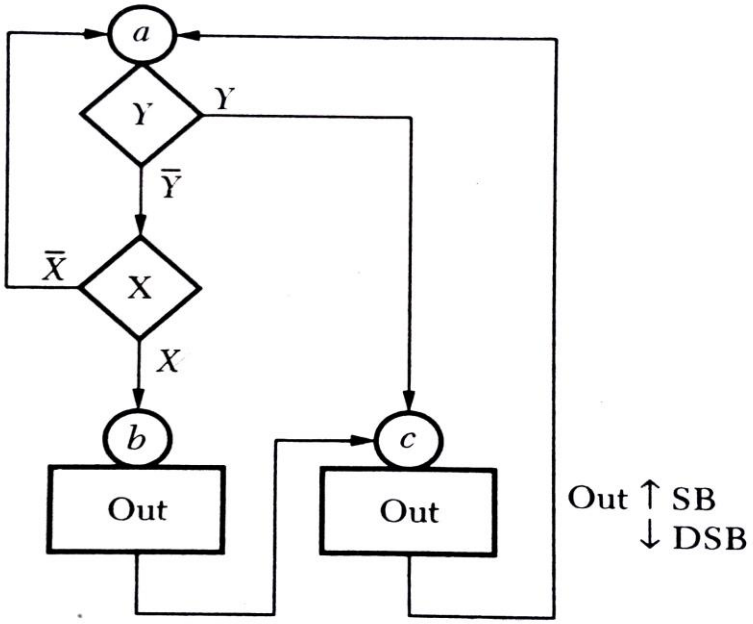


Fig. 6.7: State diagram

6.8 equivalent state & redundant state

- In state machine design, one of the first tools used is the state diagram. Some of these diagrams may contain extra or redundant states that could be eliminated to decrease the design complexity. In a state machine, two major tasks are completed during each stage:

  - Outputs are generated, if necessary
  - The signals producing the correct next state must be generated.

I **Equivalent states**: If a state machine is started from either of two states and identical output sequences are generated from every possible set of input sequences, the two states are said to be identical.

II **Redundant states**: A state that is equivalent to another state is called a redundant state.

- The redundant states can be removed. Hence a reduced state diagram contains no equivalent states. There is an orderly method available to identify and eliminate redundant states. This method is a simplified version of a more complex method that is generally applied to asynchronous system.

- For practical synchronous systems, the following method is often sufficient to create the reduced state diagram.

  a) The next state table for the state diagram is constructed
  b) Equivalent states are then identified
  c) As equivalent states are identified, all but one are considered redundant and thus be considered single state.

- Actually, the simplified method of reducing states does not guarantee a minimal no. of states. More complex method can be carried out by executing the following steps:

  a) Reduce the state table as far as possible using the simple method.
  b) Group states having equivalent outputs together.
  c) Within each group, eliminate those states that are not potentially equivalent.
  d) Within each group, assume the equivalence of every possible pair of states.

- By following this procedure, final reduced state diagram can be obtained.

6.9 General State Machine Architecture

6.9.1 Traditional State Machines

- The general block diagram of the state machine along with 2 possible variations is shown below.

  1. General state machine

     - The general state machine architecture consists of input forming logic, memory and output forming logic. The next state and the outputs are generated according to the design.

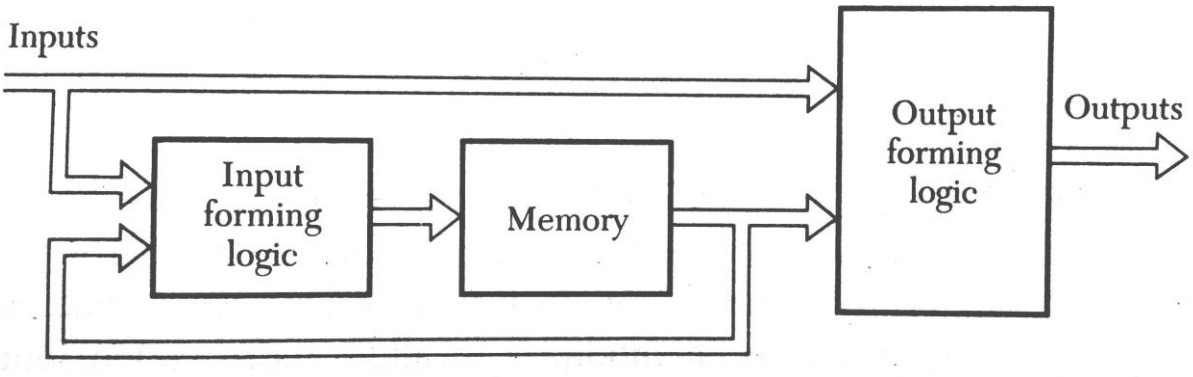


Fig. 6.8: General state machine

2. State machine with output-holding register

   - The output holding register is added to latch the outputs into this register at the midpoint of the state machine or after the state flip-flops settle. This arrangement is used to filter out the glitches from the output decoder.

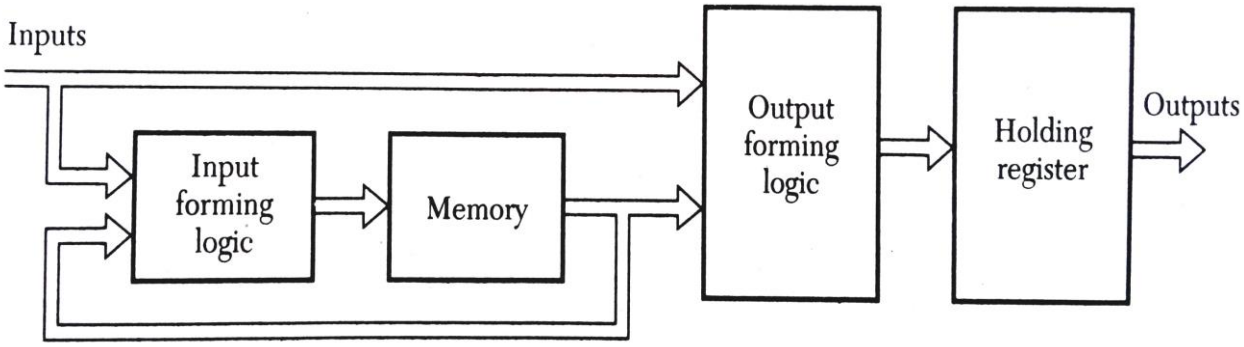


Fig. 6.9: State machine with output holding register

3. State machine with input-synchronizing register

   - A synchronizing register is used to convert asynchronous inputs to synchronous variables. The state machine is then controlled by synchronous variables. Although, no output holding register is used, it may be added to the system if needed.

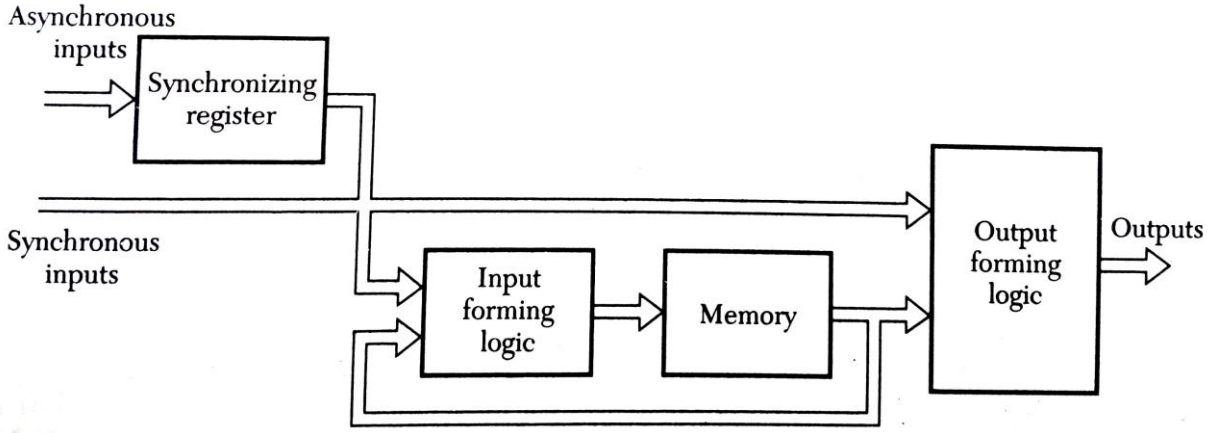


Fig 6.10: State machine with input-synchronizing register

6.9.2 The"One-Hot" state machine

- The methods of state machine design are based on the use of minimum number of state flip-flops. If m states are required to implement a sequential controller, the no of state flip-flops required n, is found by selecting the minimum value of n to satisfy: $2^n \geq m$
- This approach minimizes hardware requirements but there are certain disadvantages in certain applications.
- In sequential controllers for digital computers, there is often a need for the state machine to be in 2 states simultaneously. Since, conventional state machines always exist in a single unique state at any given time, only one series of operations can be controlled.
- The "one-hot" state machine, which associates each state with the assertion of a specific state flip-flop, can solve this problem. In a sequential system that progresses through a series of unique states, only one flip-flop is asserted during each stage. However, if two states must exist simultaneously, two flip-flops can be asserted. The output signals produced by each state flip-flop can then be used to control two operations simultaneously.
- These types of system allow the implementation of the fork and join operations in digital controllers. The figure below shows the idea of forking and jerking in a state machine controller.

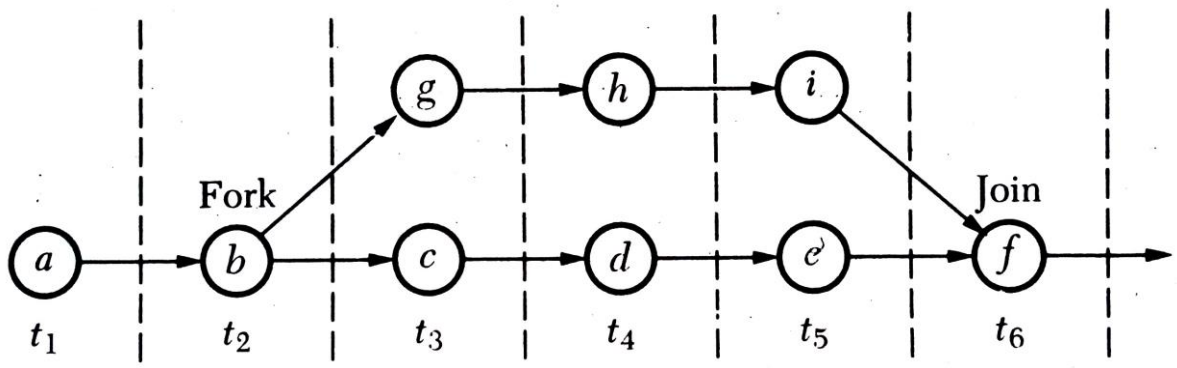


Fig. 6.11: Schematic representation of forking and jerking

- The state machine progresses from state a to b. at this point, depending on an input, the system creates two states, c and g, that exist during t_3. Two states also exist during t_4 (d and h) and during t_5 (e and i). As t_6 is entered, the system returns to a single state f. during time slots t_3 , t_4 & t_5 , two separate processes can be controlled concurrently.

- **Advantages of "one-hot" state machine design:**

  a) State machine can enter in two states simultaneously.
  b) Design methods are simpler
  c) This type of controller is more preferable for computer-aided design techniques than is the conventional state machine.
  d) This design leads to shorter development time.

7.1 Pros & Cons (Advantages & Disadvantages) of Synchronous and Asynchronous State Machines

- Synchronous and asynchronous state machines can be design according to the application. There are various advantages & disadvantages of both the systems.

- Advantages of synchronous state machines:

  1. All signals are stable when the state flip-flops change.
  2. If there is any conflict with the unstable signals, certain variables can be delayed with respect to clock signal to resolve this problem.

- Disadvantages of synchronous state machines:

  1. Additional circuitry is required to generate and control the clock signal.
  2. The state changes must wait until a clock transition occurs even though the signals determining
  3. The state change may have initialized long before the change takes place.
  4. The speed of operation is limited when the system is driven by clock.

- Advantages of asynchronous state machines:

  1. Speed: no clock involved; speed only depends upon propagation delays.
  2. Flexibility: different parts of an asynchronous system can operate at different speeds (each
  3. limited by their propagation delay)
  4. Power usage: distributing clock signal to all parts of an synchronous system adds to power
  5. usage, up to 30-40% for a high performance circuit

- Disadvantage of asynchronous state machine:

  1. Design complexity: Difficult to design and limited tool support.
  2. Glitches: race conditions, “glitches” can cause problems if circuits not carefully designed.

7.2 The Fundamental-Mode Model

- An Asynchronous Sequential Machine is a Sequential Machine without Flip-flops inside. Asynchronous Sequential Machines are based on an analysis of feedbacks in combinatorial gate networks.

- Main assumptions for Asynchronous circuits:

  1. Only 1 input changes at a time
  2. Input changes occur sufficiently far apart to allow the circuit to reach a stable state before the next change.

- The below figure demonstrates a simple asynchronous circuit.

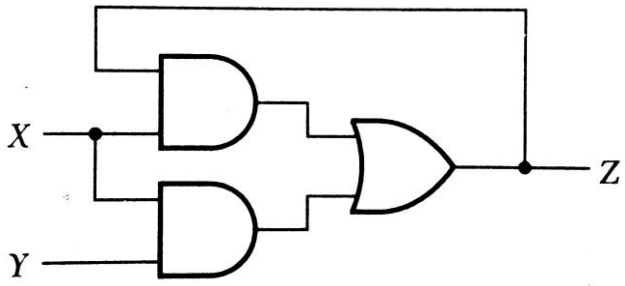


Fig. 7.1: An asynchronous circuit

- Here, X and Y are the system inputs while Z is the system output. The signal Z is fed back to a gate input to determine its new value.

- There is always a delay from change in X or Y to change in Z. the addition of delay times of both the inputs X and Y leads to change in Z signal. It is possible to force a change in feedback variable Z, by changing system input.

- The fundamental-mode model is used for the analysis of asynchronous digital circuits. The model for the above circuit is as shown. Here, the gates are considered to have no delay in this model, while the delay element has an output that follows its input after a delay of Δt.

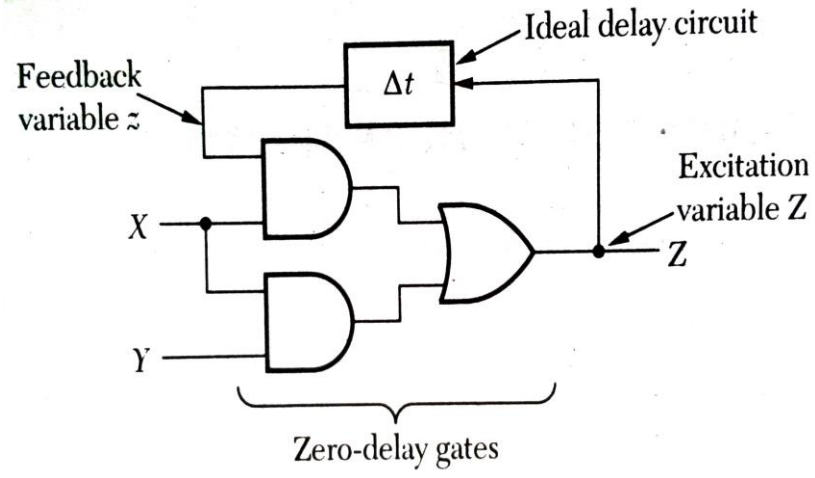


Fig. 7.2: Fundamental mode model for the an asynchronous circuit

- The variable at the input of the delay element is called the **excitation variable**, while the **feedback variable** appears at the output of the delay element. This model produces the delay of the feedback variable but predicts no delay between input and output. The overall behavior of the actual circuit can be predicted rather accurately applying this particular model.

7.3 Problems of Asynchronous Circuits

- Asynchronous state machines are not as widely used as synchronous machines since there are three problems such as hazards, oscillations and critical races.

1. Hazards

- A glitch is when a signal temporary takes on the wrong value. Glitches caused by structure of circuit and propagation delays are called hazards.

- There are two types of hazards;

a) Static Hazards

- When signal is not supposed to change its value in response to a specific change in an input, but instead momentarily does change is known as static hazard.

b) Dynamic Hazards

- This is when a signal is supposed to change value, but there is a small oscillation.

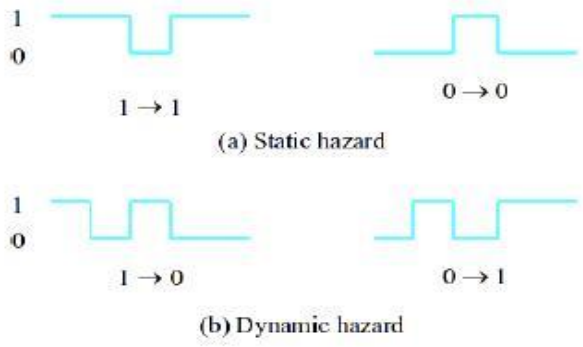


Fig. 7.3: Static and Dynamic hazard

- A change to a primary input often has more than one path of propagation to an output. When one path has a longer propagation delay than the others, we may find a static hazard.

- This can be eliminated by examining the K-map of the output. A potential hazard exists whenever two adjacent ones (or 0's if we are doing a product-of-sum implementation) are not covered by a common product term (sum term for product-of-sum).
- To guarantee no static hazards, obtain a cover such that each pair of adjacent one's (zero's) is covered by a common product term (sum term).

- A dynamic hazard is caused by the structure of a circuit. It is caused by a circuit with more than two levels, in which changes to an input have more than one path to propagate. A circuit with a dynamic hazard must also contain a static hazard

- To avoid dynamic hazards, design two-level circuits with no static hazards.

2. Oscillations

- A second problem that can occur in a poorly designed circuit is that of oscillation. Consider the excitation map as shown below.

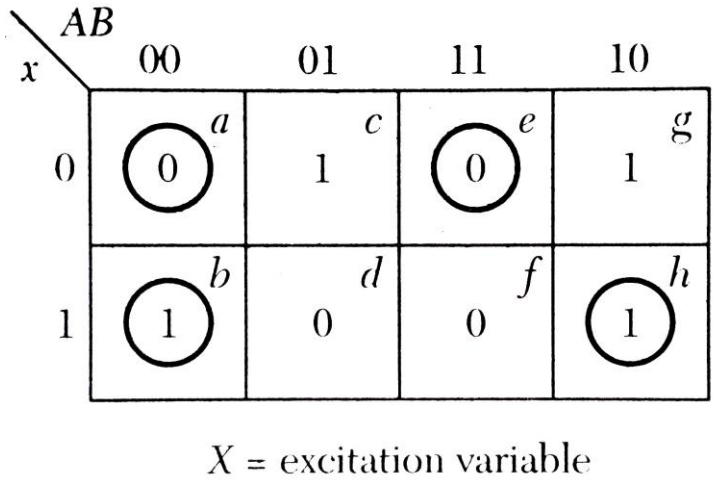


Fig. 7.4: An excitation map

- If the system is in state a, a change of input from B = 0 to B = 1 sends the system to state c. State c is the transient state, and thus the excitation variable X changes to 1.
- A short time later, *x* changes to 1, moving the system to state d. This state is also the transient state changing back X to 0, followed by change in *x* to 0.
- The system now oscillates between states c and d. this is known as oscillation and is unacceptable in most systems. Hence, the situation depicted by states c and d of the map must be avoided.

3. Critical races

- A third problem in asynchronous design is that of critical races. This situation can occur only when two or more feedback variables are present in the system. The below figure demonstrates the critical race problem.

- This system has two external inputs, A and B, and two excitation variables, X and Y, that are feedback to the input of the circuit.

- Due to unequal propagation delays, one of the excitation variables will reach a value of 1, while the other has not changed from a value of 0. The final stable state reached from this condition depends on the relative switching speeds of variables X and Y. This situation is referred to as critical race.

- In some cases, a final state is never reached. Such a situation is also called critical race. Critical races obviously must be avoided.

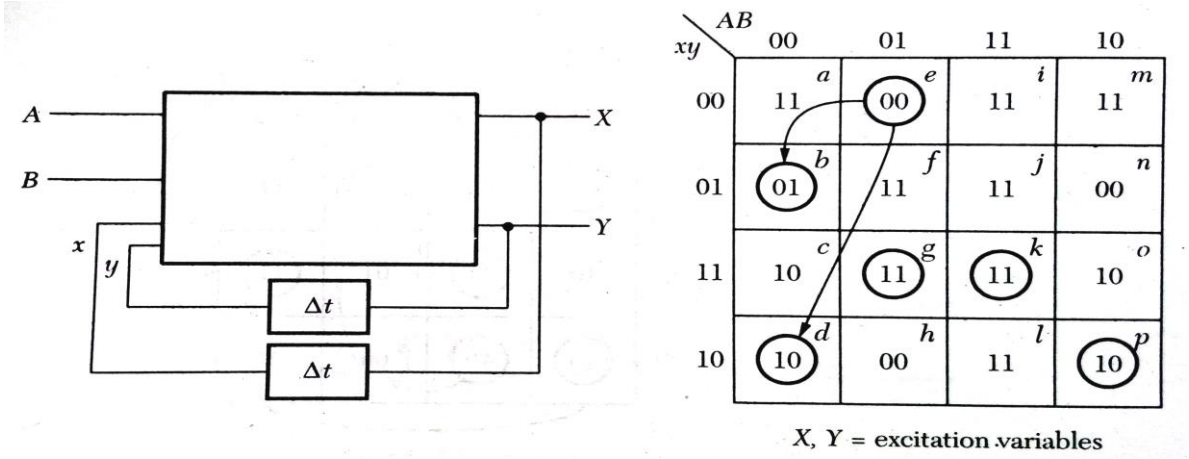


Fig. 7.5: An excitation map with critical races

- Hence, in designing asynchronous circuits, hazards, oscillations and critical races must be avoided.

7.4 Basic Design Principles

- For designing a system, we are given only the system behavior. There may be several different circuits that can satisfy the specifications. We generally can define the stable states in the system. We then must add transient or cycle states to cause the system to sequence through the proper states without introducing any critical races.

- Then, a primitive excitation table having only one stable state per row is produced. Once, the feedback variables are assigned, the excitation variables for the stable states are also determined. The transient states are not fixed at this point and can be selected to eliminate races or oscillations.

- Thereafter, expressions for the excitation variables are written and the circuit is implemented.

- To design an Asynchronous circuit, the following procedure can be used:

    1. Create a State Diagram according to the functional specification.
    2. Create a Flow Table and reduce the number of states as far as possible.
    3. Assign binary codes to the states and create the Excitation Table.
    4. Write down the expressions for the next state and output functions.
    5. Draw a circuit that implements the derived expressions.

7.5 Analysis of Asynchronous Sequential Circuits

- An asynchronous sequential circuit can be analyzed by the following steps:

    1. Cut each feedback path and insert a delay.
    2. Determine Next State and Output expressions from circuit.
    3. Derive excitation table (asynchronous state assignment table) from Next State and Output equations.
    4. Obtain a flow table (asynchronous state table) by assigning state labels to each of the state encodings.
    5. Draw FSM from flow table.

8.1 INTRODUCTION TO LOGIC FAMILY:

- Different types of logic families are as follows;
  1. Resistor Transistor Logic (RTL)
  2. Diode Transistor Logic (DTL)
  3. High Threshold Logic (HTL)
  4. Transistor Transistor Logic (TTL)
  5. Emitter Coupled Logic (ECL)
  6. Integrated Injection Logic (IIL or I^2L)
  7. Metal-Oxide Semiconductor (MOS) Logic
  8. Complementary Metal-Oxide Semiconductor (CMOS) Logic

- The logic families TTL, ECL, IIL, MOS & CMOS are currently in use.
- The TTL & CMOS are suitable for Small Scale Integration (SSI) & Medium Scale Integration (MSI).
- The MOS & CMOS are particularly suitable for Large Scale Integration (LSI).
- The IIL is mainly suitable for Very Large Scale Integration (VLSI) & Ultra Large Scale Integration (ULSI).
- The logic families currently used are compared as follows;

Table 8.1: Different types of logic family and its parameter

Logic Family	Propagation Delay Time (ns)	Power Dissipation per gate (mW)	Noise Margin	Fan-in	Fan-out	Cost
TTL	9	10	0.4	8	10	Low
ECL	1	50	0.25	5	10	High
MOS	50	0.1	1.5		10	Low
CMOS	< 50	0.01	5	10	50	Low
IIL	1	0.1	0.35	5	8	Very Low

8.2 TRANSISTOR TRANSISTOR LOGIC (TTL):

- The TTL is so named because of its independence on transistors alone to perform basic logic operations.
- The TTL uses transistors operating in saturated mode.
- It is the fastest of saturated logic families.
- The basic TTL logic circuit is the NAND gate.
- Good speed, low manufacturing cost, wide range of circuits and the availability in SSI and MSI are its advantages.
- Tight V_{CC} tolerance, relatively high power consumption, moderate packing density, generation of noise spikes and susceptibility to power transients are its disadvantages.
- TTL logic family consists of several subfamilies such as:
  - Standard TTL
  - High Speed TTL

- Low Power TTL
- Schottky TTL
- Low Power Schottky TTL
- Advanced Schottky TTL
- Advanced Low Power Schottky TTL
- F (Fast) TTL

- The difference between the various TTL subfamilies are in their electrical characteristics such as delay time, power dissipation, switching speed, fan-out, fan-in, noise margin etc.
- For standard TTL,
  - Propagation Delay time = 9 ns
  - Power dissipation per = 10 mW
  - Noise Margin = 0.4 mV
  - Fan-in = 8
  - Fan-out = 10
  - Logic '0' = 0 V to 0.8 V
  - Logic '1' = 2 V to 5 V
  - Indeterminate Range = 0.8 V to 2 V
- The voltage range at input & output side of gate are as shown below:

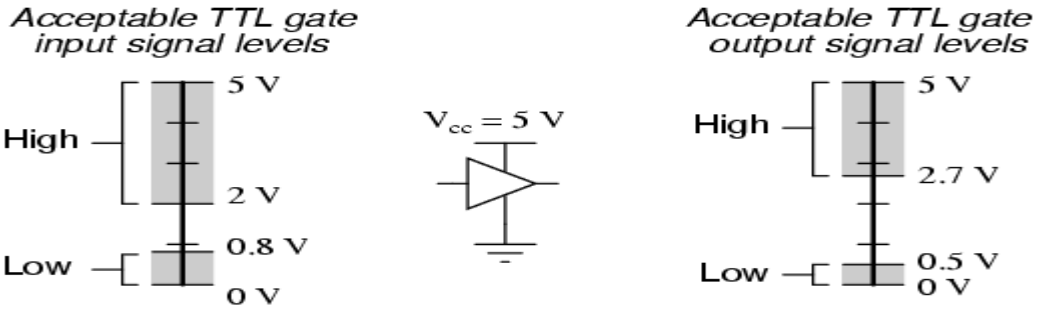


Fig. 8.1: Voltage range at input and output side of NOT gate

- The high noise margin & low noise margin are as shown below:

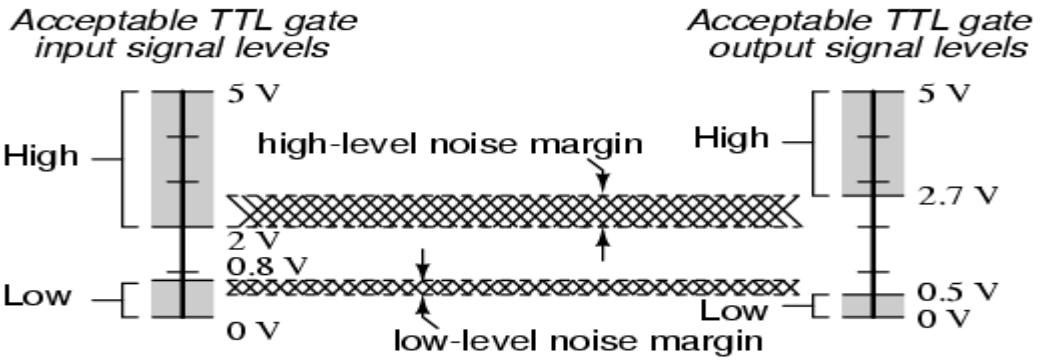


Fig. 8.2: Representation of noise margin

Two Input TTL NAND gate

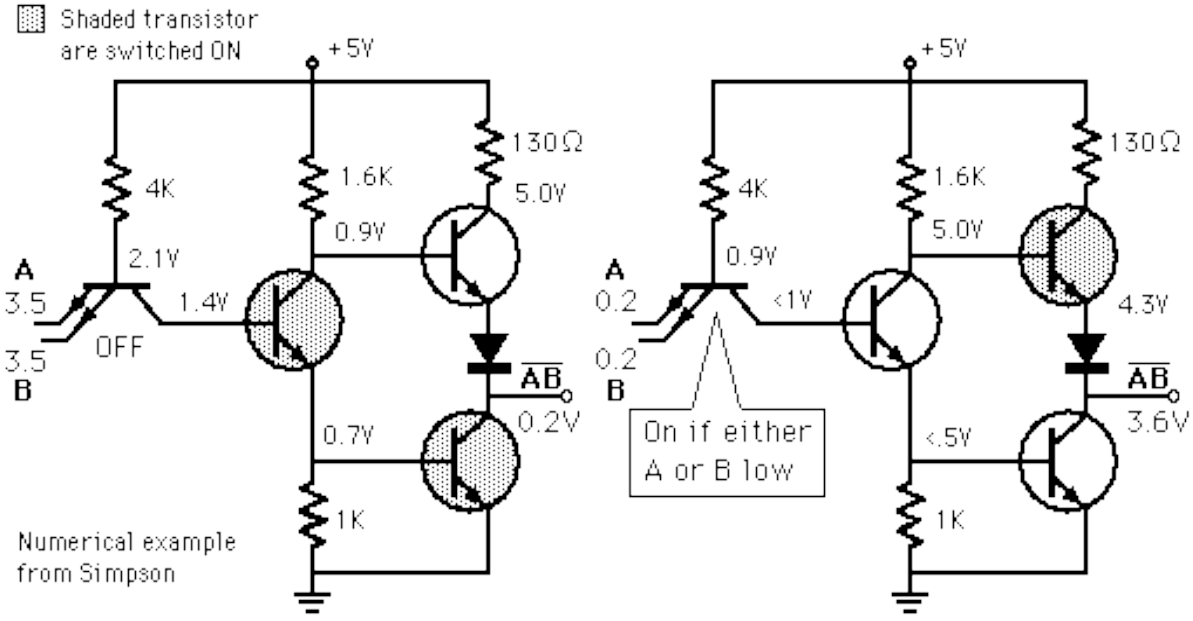


Fig. 8.3: Tow input TTL NAND gate

- The above arrangement shows two input TTL NAND gate.
- The transistor where inputs A & B are given is called Multiple Emitter Transistor.
- The transistors can act as switch depending on the type of input given to it.
- Two cases are shown in the figure above;
  - Case 1: When both the inputs A & B are given 3.5 V (i.e. Logic High or '1') then transistors with shaded region turns ON. Because of which the Vcc gets a path to Ground, as a result of which the output becomes 0.2 V (Logic Low or '0').
  - Case 2: When either of the input A or B is given 0.2 V (i.e. Logic Low or '0') then transistor with shaded region turns ON, as a result of which the Vcc is not grounded and appears at the output with 3.6 V (i.e. Logic High or '1').
- In this manner, many logical circuits can be designed using TTL.

8.3 EMITTER-COUPLED LOGIC (ECL):

- This logic family is also called Current Mode Logic or Current Steering Logic
- It is the FASTEST of all logic families.
- ECL operates on the principle of current switching, whereby a fixed bias current less than I_C(sat) is switched from one transistor's collector to another.
- Because of thie mode operation, this logic form is also referred to as Current Mode Logic (CML).
- It is also called Current Steering Logic (CSL), because current is steered from one device to another.
- The ECL family is used in very high frequency applications where its speed is superior
- The important characteristics of ECL are:

- Transistors never saturate. So, speed is high with t_{pd} = 1 ns
- Logic Levels are negative, -0.9 V for Logic 1 and -1.7 V for Logic 0.
- Noise Margin is less, about 250 mV. This makes ECL, unreliable for use in heavy industrial environment.
- ECL circuits produce the output and its complement, and therefore, eliminate the need for inverters.
- Fan-out is large because the output impedance is low. It is about 25.
- Power dissipation per gate is large, P_D = 40 Mw.
- The total current flow in ECL is more or less constant. So, no noise spikes will be internally generated.

Advantages

- It is a non-saturated logic, in the sense that the transistors are not allowed to go into saturation. So, storage time delays are eliminated and, therefore, the speed of operation is increased.
- Currents are kept high, and the output impedance is so low that circuit & stray capacitances can be quickly charged & discharged.
- It has limited voltage swing between LOW & HIGH voltage levels.

Disadvantages

- High Cost
- Low Noise Margin
- High Power Dissipation
- Its negative supply voltage and logic levels are not compatible with other logic families (making it difficult to use ECL in conjunction with TTL and MOS circuits)
- Problem of cooling

Application

- It is used in superfast computers and high-speed special purpose applications.

8.3.1 ECL NOT Gate

- The circuit shown below is of inverter/buffer which used Current Mode Logic (CML).
- This circuit has both an inverting output (OUT1) and a non-inverting output (OUT2).
- Two transistors are connected as a differential amplifier with common emitter resistor.
- There are two cases for NOT gate;
  - Case – 1: When VIN is HIGH, transistor Q1 is ON, but not saturated, and transistor Q2 is OFF. This is true because of careful choice of resistor values and voltage levels. Thus VOUT2 is pulled to 5.0 V (HIGH) through R2, and it can be shown that the voltage drop across R1 is about 0.8 V, so that VOUT1 is about 4.2 V (LOW).

- Case – 2: When VIN is LOW, as shown in figure, transistor Q2 is ON, but not saturated, and transistor Q1 is OFF. Thus, VOUT1 is pulled to 5.0 V through R1, and it can be shown that VOUT2 is about 4.2 V.

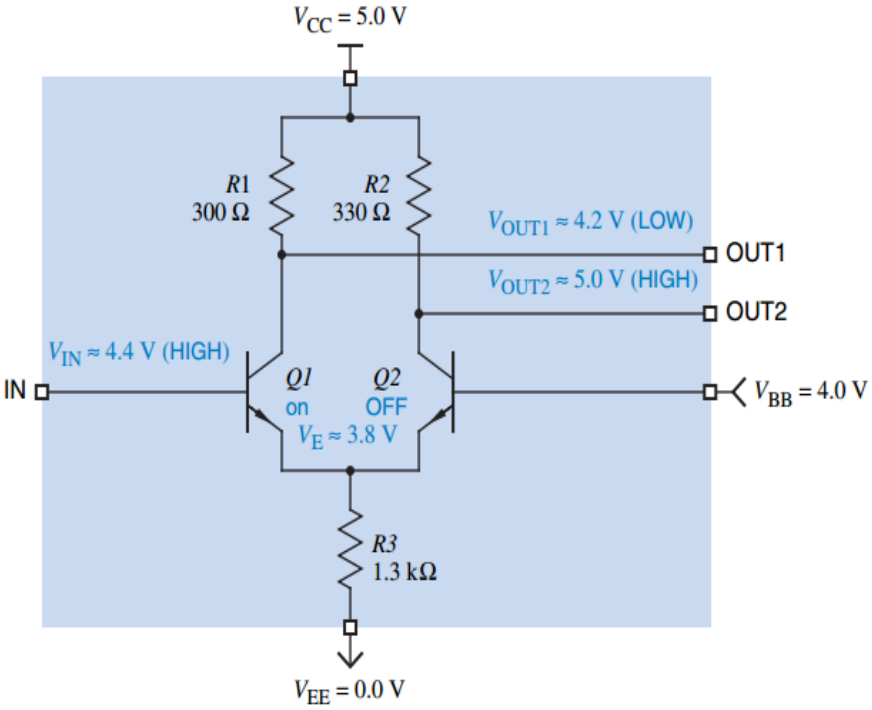


Fig. 8.4(a)

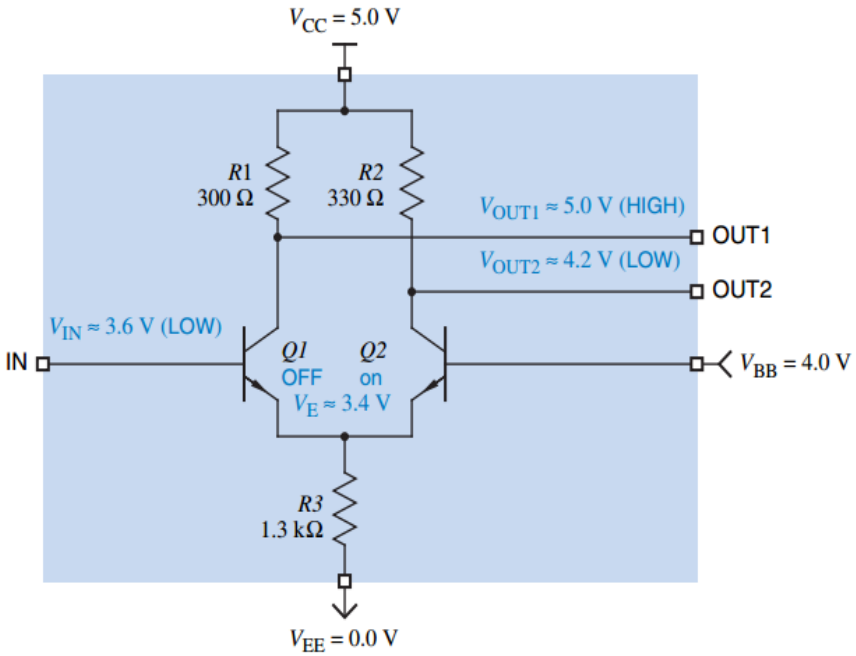


Fig. 8.4(b)

Fig. 8.4: (a) & (b) ECL NOT gate

8.4 MOSFET LOGIC (ECL):

- MOSFET family are simpler & inexpensive to fabricate, require much less power, have better noise margin, a greater supply voltage range, a higher fan-out and require much less chip area as compared to other bipolar logic families.
- For MOS logic,
  - Propagation Delay, tpd = 50 ns.
  - Noise Margin, NM = 1.5 V.
  - Power Dissipation, PD = 0.1 Mw.
  - Fan out = 50 for frequencies greater than 100 Hz and it is virtually unlimited for dc or low frequencies.
  - The propagation delay associated with MOS gates is large (50 ns) because of their high output resistance (100 k) and capacitive loading presented by the driven gates.
- There are three types of MOSFET:
  - P-channel MOSFET (PMOS)
    - Enhancement Type PMOS
    - Depletion Type PMOS
  - N-channel MOSFET (NMOS)
    - Enhancement Type NMOS
    - Depletion Type NMOS
  - Complementary MOSFET (CMOS)

Advantages

- MOS logic is the simplest to fabricate.
- It occupies very small space because it requires only one basic element-and NMOS or a PMOS transistor. It does not require other elements like resistor and diodes which occupy large space.
- Because of its ease of fabrication and lower power dissipation per gate PD, it is ideally suited for LSI, VLSI & ULSI for dedicated applications such as large memories, calculator chips, large microprocessors etc.
- The greater packing density of MOS ICs results in higher reliability because of the reduction in the number of external connections.

Disadvantages

- The operating speed of MOS is slower than TTL, so, they are hardly used in SSI and MSI applications
- Because of the very high impedance present at MOSFET's input, the MOS logic families are susceptible to static charge damage.

Difference between CMOS & NMOS/PMOS

- Both NMOS & PMOS have greater packing density than that of CMOS, and are therefore, more economical than CMOS.
- The CMOS family has more complex architecture compared to NMOS & PMOS family.
- CMOS possesses the important advantages of higher speed & much lower power dissipation compared to NMOS & PMOS family.
- CMOS family has improved noise margin compared to NMOS & PMOS family.

8.4.1 NMOS NOT Gate

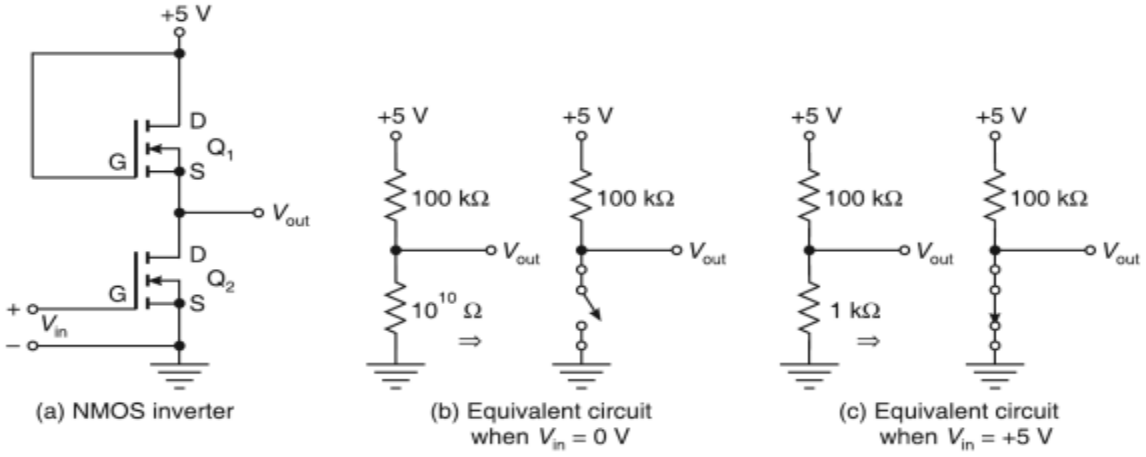
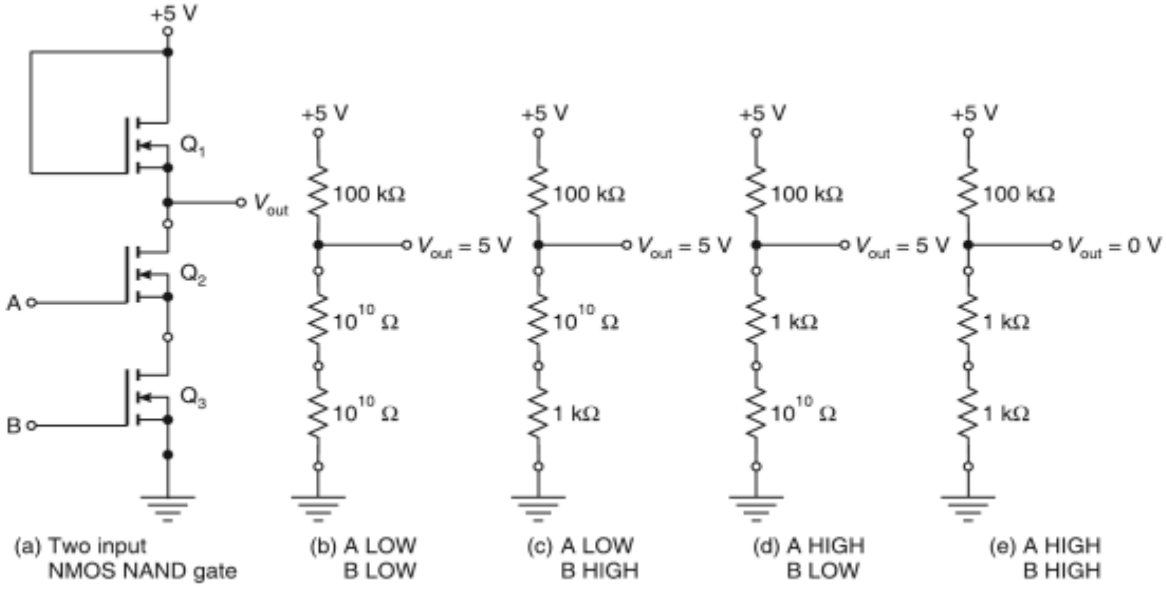


Fig. 8.5: NMOS NOT gate

8.4.2 NMOS NAND Gate

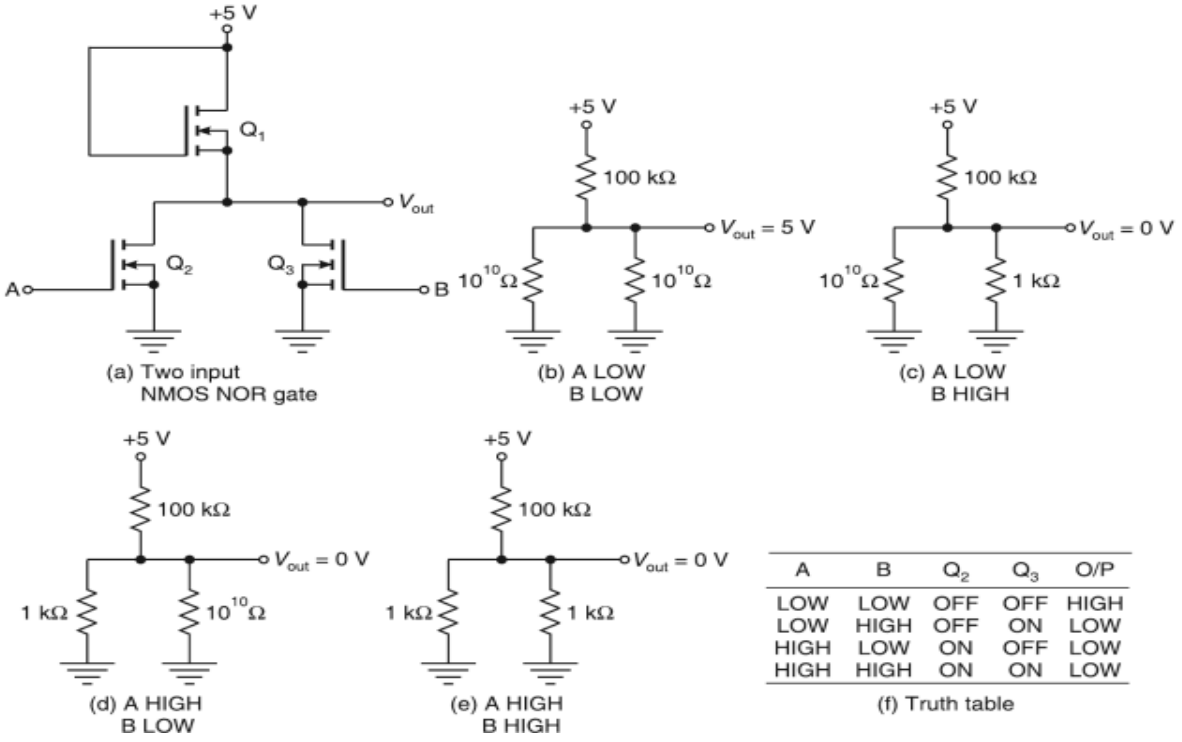


Circuit diagram and equivalent circuits for various inputs of the NMOS NAND gate.

Fig. 8.6: NMOS NAND gate

- In the N MOS NAND gate shown, Q_1 is acting as a load resistor and Q2 and Q3 as awitches controlled by input levels A and B res[ectively.
- When both A and B are 0 V, both Q2 and Q3 are OFF. So the equivalent circuit 8.6 (b) results with Vout = + 5 V.
- When both A = 0 V and B = + 5 V, both Q2 is OFF and Q3 is ON. So the equivalent circuit 8.6 (b) results with Vout = + 5 V.
- When both A = + 5 V and B = 0 V, both Q2 is ON and Q3 is OFF. So the equivalent circuit 8.6 (b) results with Vout = + 5 V.
- When both A = + 5 V and B = + 5 V, both Q2 and Q3 are ON. So the equivalent circuit 8.6 (b) results with Vout = 0 V.
- Thus the above circuit works as positive logic two inputs two input NAND gate.

8.4.3 NMOS NOR Gate



A	B	Q_2	Q_3	O/P
LOW	LOW	OFF	OFF	HIGH
LOW	HIGH	OFF	ON	LOW
HIGH	LOW	ON	OFF	LOW
HIGH	HIGH	ON	ON	LOW

(f) Truth table

Fig. 8.7: NMOS NOR gate

- When A is LOW and B is LOW, Q2 is OFF and Q3 is OFF, so the equivalent circuit (b) results with Vout = + 5 V.
- When A is LOW and B is HIGH, Q2 is OFF and Q3 is ON, so the equivalent circuit (b) results with Vout = 0 V.
- When A is LOW and B is LOW, Q2 is ON and Q3 is OFF, so the equivalent circuit (b) results with Vout = 0 V.
- When A is LOW and B is LOW, Q2 is ON and Q3 is ON, so the equivalent circuit (b) results with Vout = 0 V.

8.4.4 CMOS NOT Gate

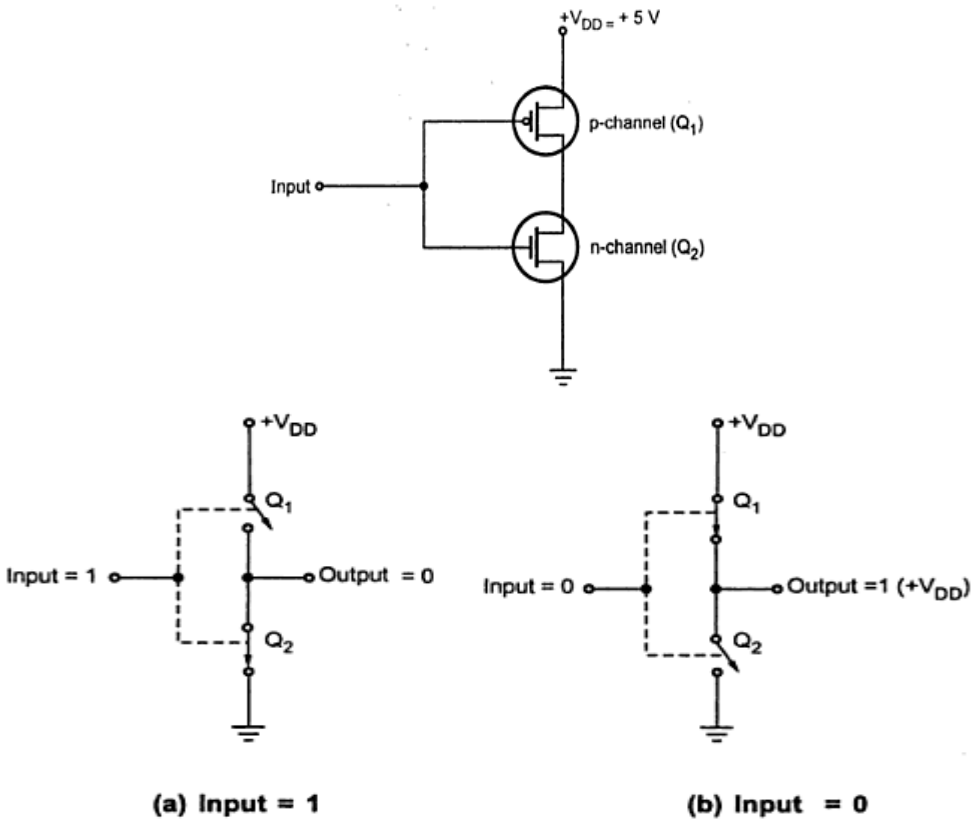
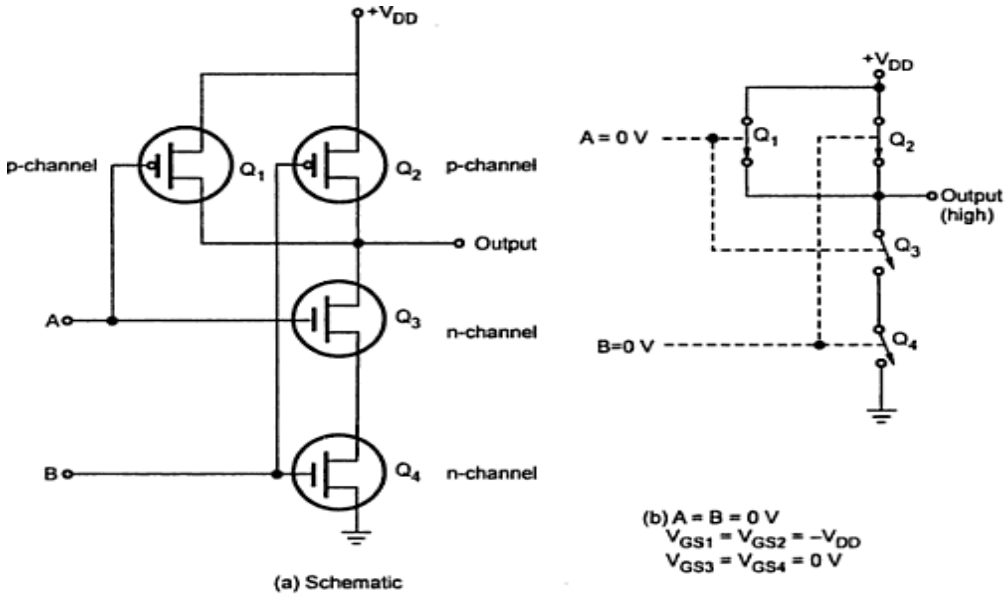


Fig. 8.8: CMOS NOT gate

8.4.5 CMOS NAND Gate



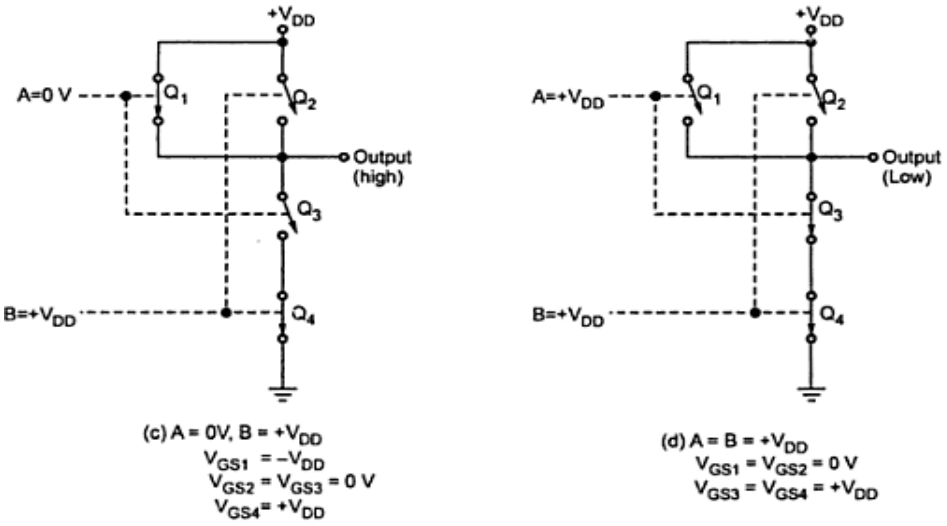


Fig. 8.9: CMOS NAND gate

8.4.6 CMOS NOR Gate

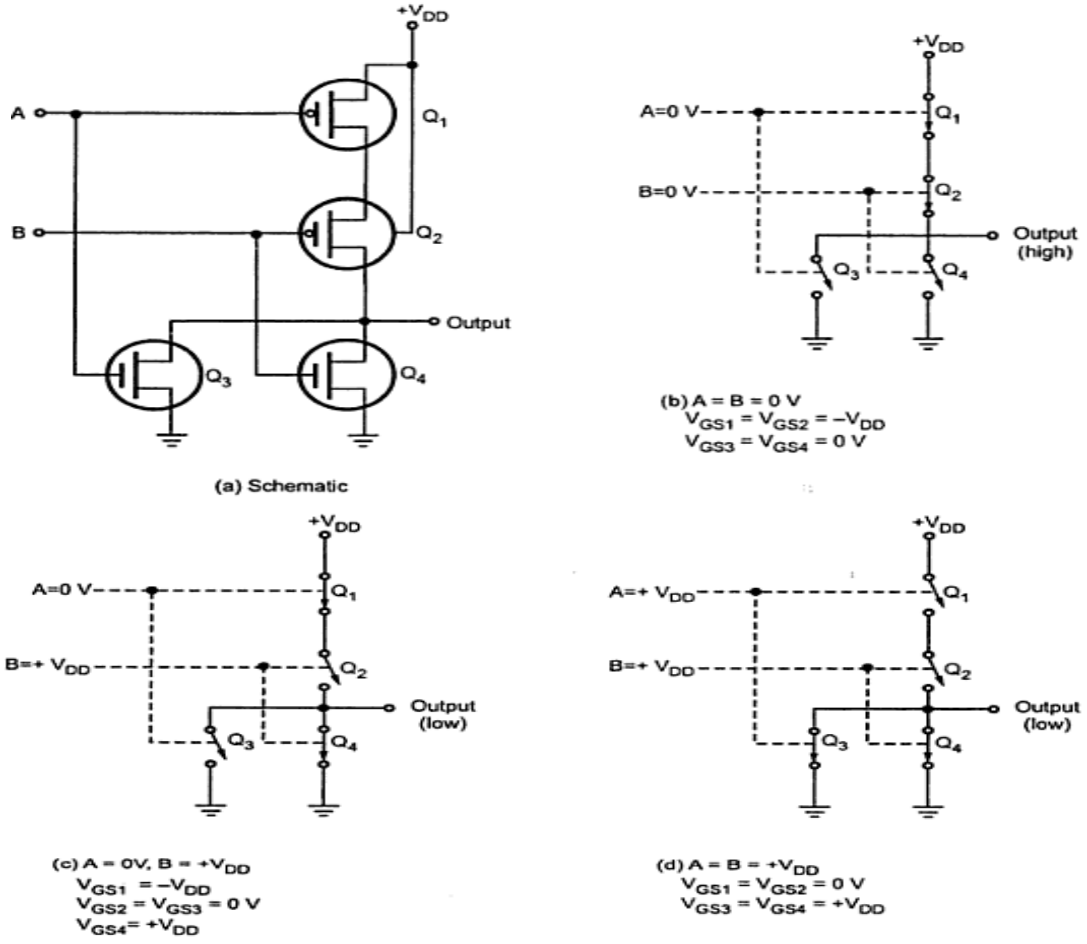


Fig. 8.10: CMOS NOR gate

9.1 INTRODUCTION TO PROGRAMMABLE LOGIC DEVICES (PLD):

9.1.1 Programmable Logic Circuit

1. Hardwire System / Circuit

- The operation of any circuit depends on IC chips used & electrical connection between chips.
- No access to internal interconnections of IC chips.
- To design circuit, internal circuit diagram is to be specified.
- Once designed, the intended function can be performed.
- If the function changes, design needs to modified.
- So the internal circuit diagram needs to be changed.
- Such systems are called **HARDWIRED SYSTEM.**

2. Programmable Circuit

- It uses programmable components.
- Device includes arrays of logic elements on a chip & allows the user to specify or program many internal connections between these components on the chip.
- Logic elements could be various gates, inverters, buffers and even flip flops.
- A system function can be created on the chip, simply by programming the chip or telling the chip where the interconnections are to be made.
- Such devices in general are called **PROGRAMMABLE LOGIC DEVICE.**

3. Programmable Logic Devices (PLD):-

- PLD is an IC chip that includes arrays of logic elements and allows a user to specify the connections among many of these elements.

9.1.2 Advantages and Disadvantages of programmable logic devices

1. Advantages:

- Chip count & physical size of a system can be minimized.
- Time from conception of system to marketing of the system can be minimized.
- Less chip count leads to integration of system on a single chip or small no. of chips.
- Low development cost.
- Less space requirement.
- High reliability.
- Easy circuit testing.
- Easy design modification.

2. **Disadvantages:**
   - Interconnections between elements on the chip must be specified or programmed.
   - PLDs also have hard wired connection but they cannot function until they are programmed while Hardwired System functions.

9.2 TYPES OF PROGRAMMABLE LOGIC DEVICES (PLD):

1. **Programmable Arrays**
   - OR Array
   - AND Array

2. **Classifications of Simple Programmable Logic Devices (SPLD)**
   - Read-Only Memory (ROM)
   - Programmable Array Logic (PAL)
   - Programmable Logic Array (PLA)
   - Programmable Logic Sequencer (PLS)

3. **More complex**
   - FPGA (Field Programmable Gate Arrays)
   - CPLD (Complex Programmable Logic Devices)

9.2.1 Programmable Array:-

9.2.1.1 OR Array:

- OR Array is as shown in below figure. In OR Array initially all the links are connected by **Fusible Link**.
- According to the required output functions one needs to program the OR Array IC.
- In OR Array all the required output functions can be taken out from the OR gates as shown in figure.
- Burn the Fusible Link in each array, which connections are not required as shown in below figure.

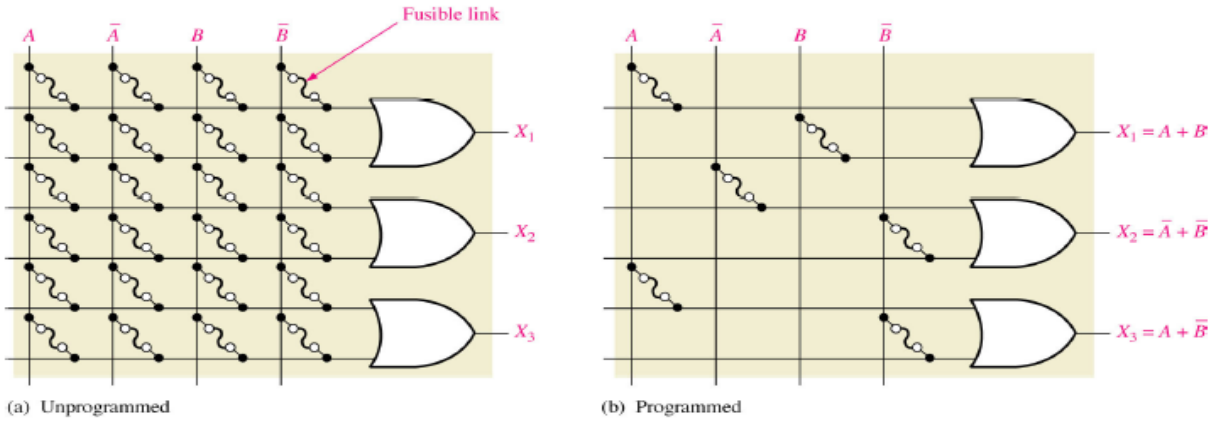


Fig. 9.1: Structure of programmed and programmed programmable OR array

9.2.1.2 AND Array:

- AND Array is as shown in below figure. In AND Array initially all the links are connected by **Fusible Link**.
- According to the required output functions one needs to program the AND Array IC.
- In AND Array all the required output functions can be taken out from the AND gates as shown in below figure.
- Burn the Fusible Link in each array, which connections are not required as shown in below figure.

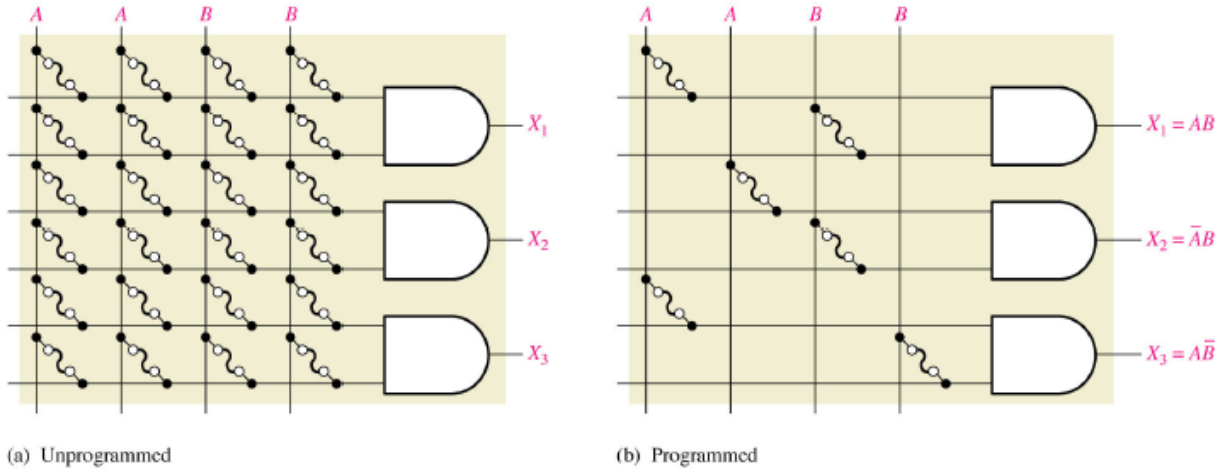


Fig. 9.2: Structure of programmed and programmed programmable AND array

9.2.2 Classifications of Simple Programmable Logic Devices (SPLD):-

9.2.2.1 Read Only Memory (ROM / PROM):

- N-input address lines, m-output data lines.
- Address lines point locations within ROM that store words of m bits.
- ROM size is defined by the no. of locations & the word size.
- A 256 X 4 ROM indicates that the device has 256 storage locations each holding a 4-bit word.
- This ROM would require eight address lines to access 256 locations.
- ROM of 32 X 8 has 32 memory locations each of 8-bit word and requires 5 address lines.
- Structure of ROM is as shown below;

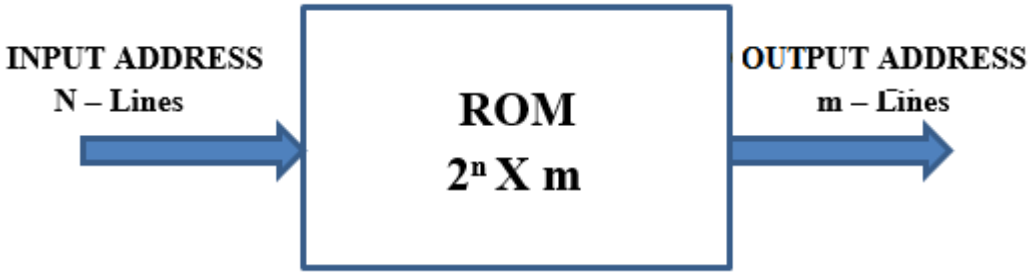


Fig. 9.3: Block diagram of ROM

- So No. of address lines (Input Lines) = n.
- No. of data lines (Output Lines) = m.
- ROM size = 2^n X m.
- Size of decode which is to be used = n X 2^n .

- ROM Consists of an array of semiconductor devices interconnected to store an array of binary data.
- Can't be changed once burned in.
- Conceptually, consist of a decoder and a memory array.

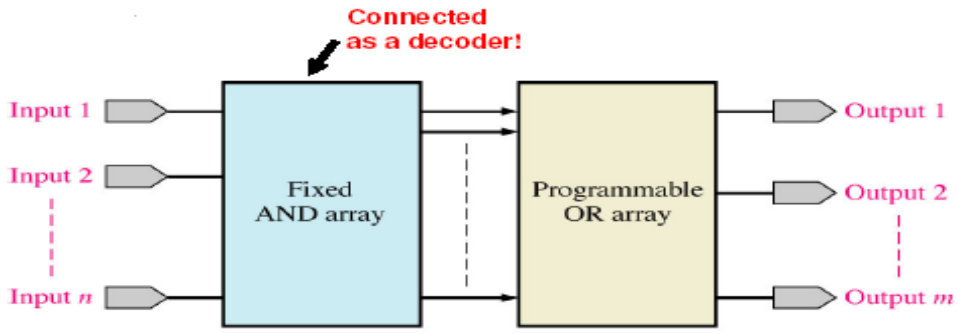


Fig. 9.4: Simplified Block diagram of ROM

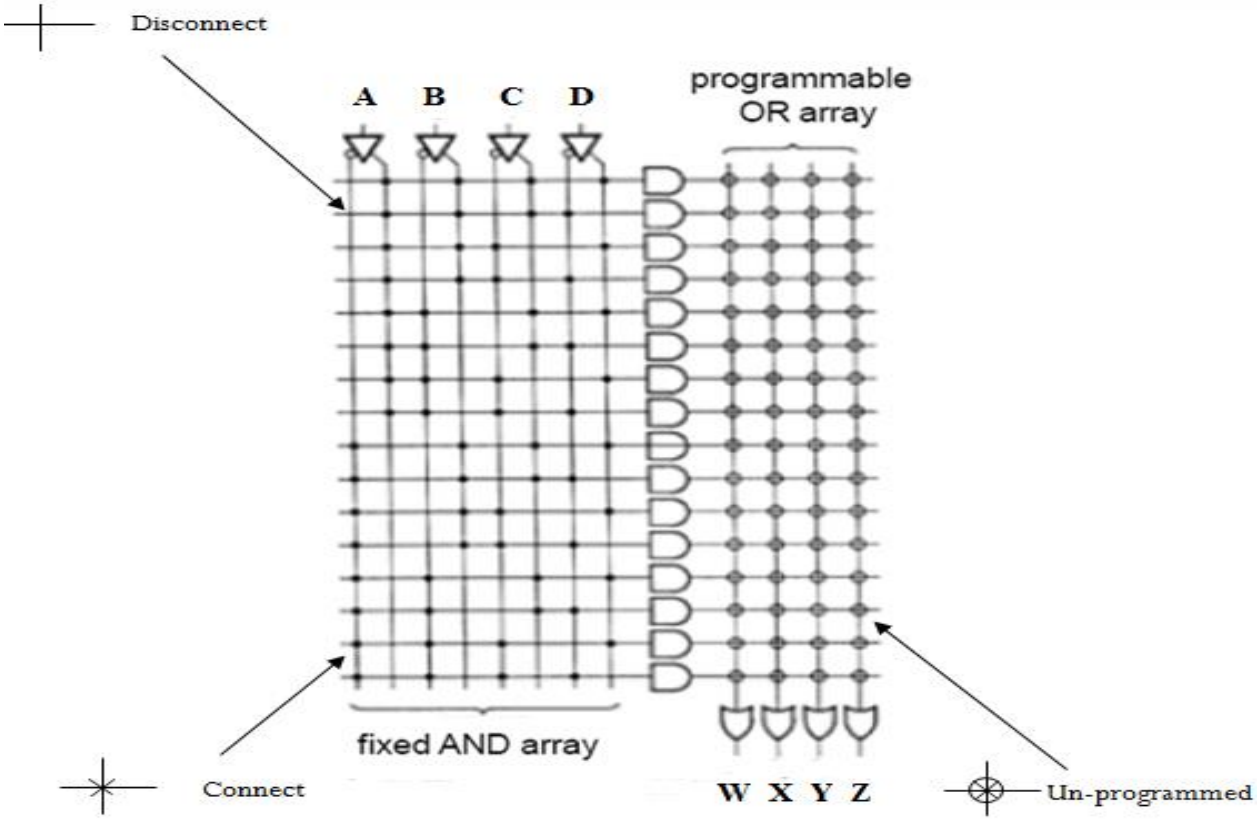


Fig. 9.5: Circuit diagram of ROM

Advantages & Disadvantages of ROM

- **Advantages:**
  1. Design become extremely easy.
  2. It is possible to change or modify the design quickly.
  3. Reduced cost.
  4. Modification takes less time than SSI/MSI circuits.

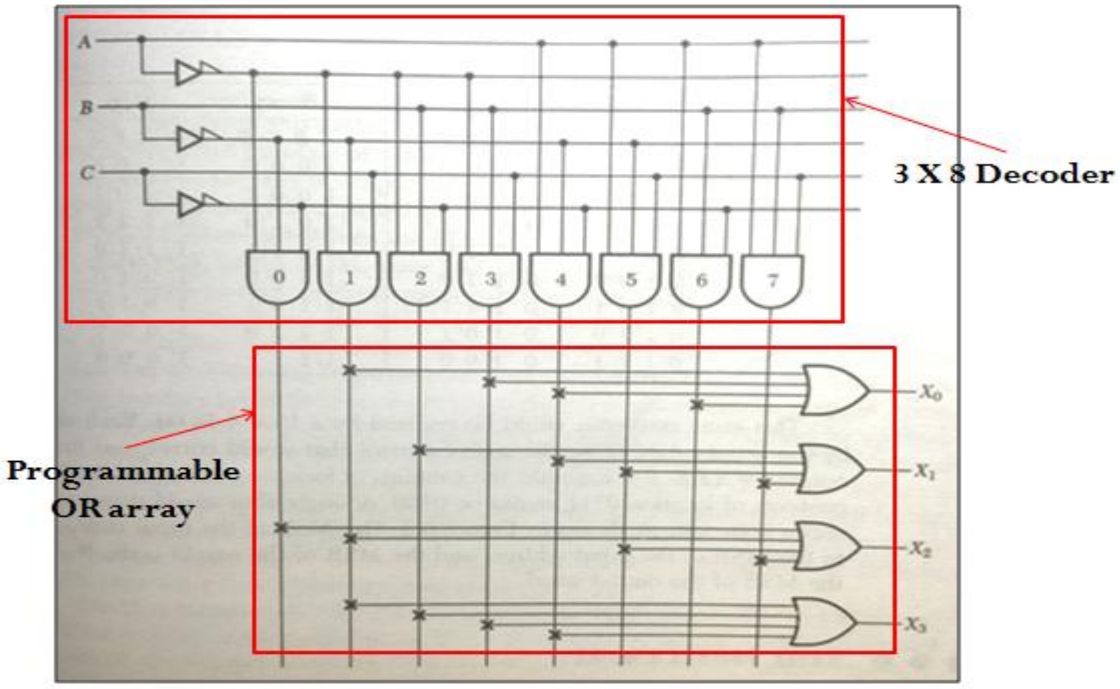
- **Disadvantages:**
  1. Increase in power requirement.
  2. Complete circuit is not utilizes
  3. Increase in size with increase in number of input variables.

- **EXAMPLE 1: Tabulate the truth for an 8 X 4 ROM / PROM that implements the following four Boolean functions:**
  $A(X,Y,Z) = \sum m(1,3,4,6)$
  $B(X,Y,Z) = \sum m(2,4,5,7)$
  $C(X,Y,Z) = \sum m(0,1,5,7)$
  $D(X,Y,Z) = \sum m(1,2,3,4)$

ANS:

Here, total No. of inputs are three → A,B,C
Total No. of outputs are four → A,B,C,D
ROM size is = 8 X 4 So, according 2^n X m → Inputs = n = 3, Output = m = 4,
Size of Decoder = n X 2^n = 3 X 8

INPUT	OUTPUT			
	A	B	C	D
X'Y'Z'(000)	0	0	1	0
X'Y'Z (001)	1	0	1	1
X'YZ' (010)	0	1	0	1
X'YZ (011)	1	0	0	1
XY'Z' (100)	1	1	0	1
XY'Z (101)	0	1	1	0
XYZ' (110)	1	0	0	0
XYZ (111)	0	1	1	0

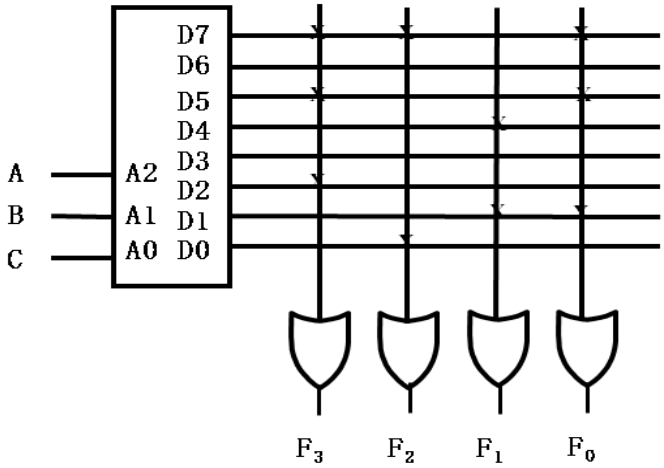


8 X 4 ROM

Here, X = Programmed Connection

• = Fixed Connection

- **EXAMPLE 2: What are functions F_3, F_2, F_1 and F_0 in terms of (A_2, A_1, A_0)?**



ANS:

- $F_3 = D_7 + D_5 + D_2 = A_2A_0 + A_2'A_1A_0'$
- $F_2 = D_7 + D_0 = A_2A_1A_0 + A_2'A_1'A_0'$
- $F_1 = D_4 + D_1 = A_2\,A_1'A_0' + A_2'A_1'A_0$
- $F_0 = D_7 + D_5 + D_1 = A_2A_0 + A_1'A_0$

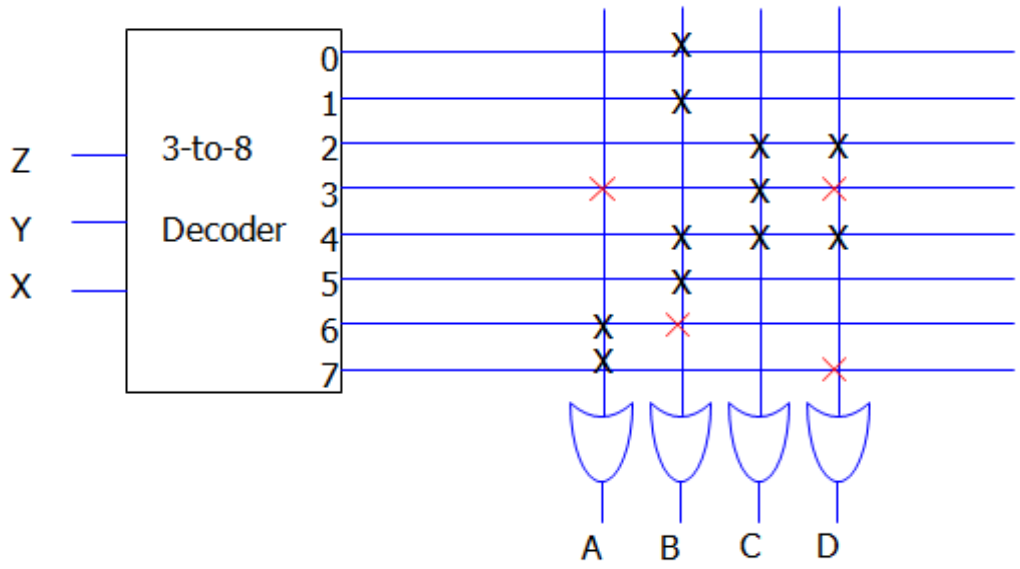
- **EXAMPLE 3: Tabulate the truth for an 8 X 4 ROM / PROM that implements the following four Boolean functions:**
  A(X,Y,Z) = ∑*m*(3,6,7);
  B(X,Y,Z) = ∑*m*(0,1,4,5,6)
  C(X,Y,Z) = ∑*m*(2,3,4);
  D(X,Y,Z) = ∑*m*(2,3,4,7)

ANS:

ROM size is = 8 X 4 So, according 2^n X m → Inputs = n = 3, Output = m = 4,
Size of Decoder = n X 2^n = 3 X 8

Inputs are → X,Y,Z
Outputs are → A,B,C,D

Inputs			Outputs			
X	Y	Z	A	B	C	D
0	0	0	0	1	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	0	0
1	1	0	1	1	0	0
1	1	1	1	0	0	1



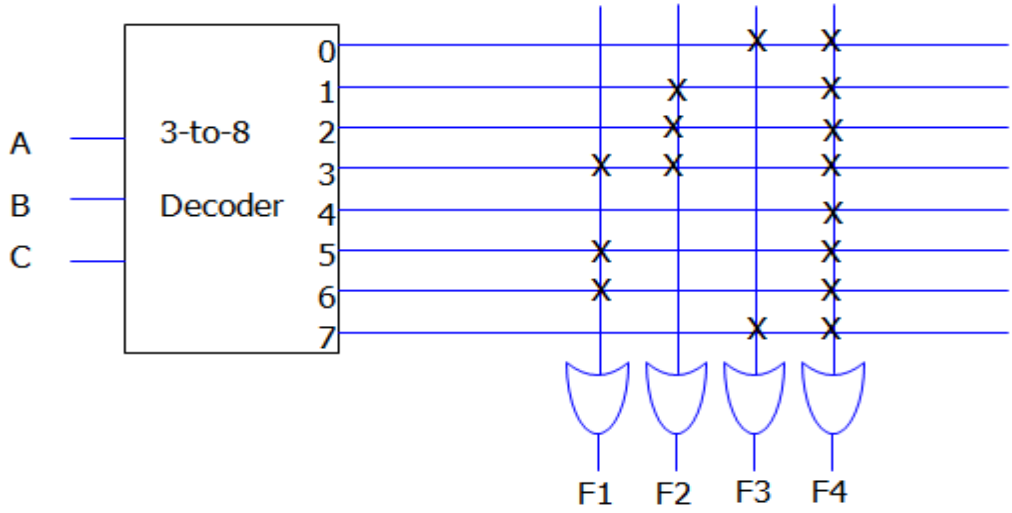
- **EXAMPLE 4: Realize the following expressions using 8 X 4 ROM / PROM**
  F1 = AB'C + ABC' + A'BC
  F2 = A'B'C + A'BC' + AB'C'
  F3 = A'B'C' + ABC

ANS:

ROM size is = 8 X 4 So, according 2^n X m → Inputs = n = 3, Output = m = 4,
Size of Decoder = n X 2^n = 3 X 8

Inputs are → A,B,C
Outputs are → F1,F2,F3,F4 Here, F4 is not given so mark all the connection for F4 with X.

INPUT	OUTPUT			
	F1	F2	F3	F4
A'B'C'(000)	0	0	1	X
A'B'C(001)	0	1	0	X
A'BC' (010)	0	1	0	X
A'BC (011)	1	1	0	X
AB'C' (100)	0	0	0	X
AB'C (101)	1	0	0	X
ABC' (110)	1	0	0	X
ABC (111)	0	0	1	X



- **EXAMPLE 5: Realize the following expressions using ROM /PROM.**
  F1 = A' + B
  F2 = AB'
  F3 = A'B + AB'

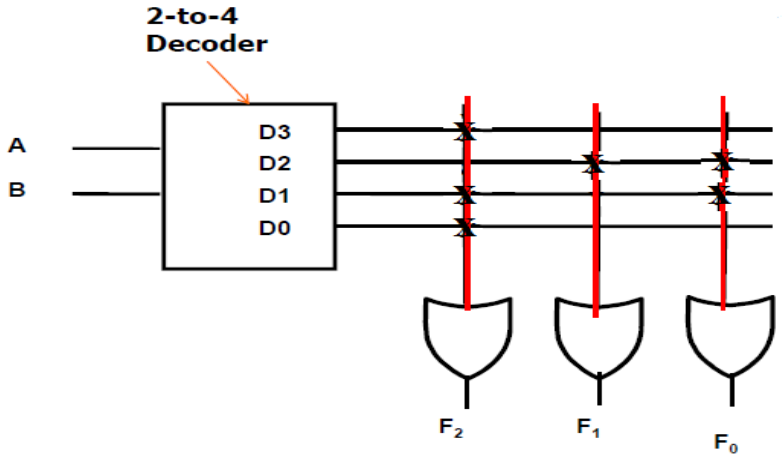
ANS:

Here, Inputs are → A,B
Outputs are → F1,F2,F3

According 2^n X m → Size of ROM = 2^2 X 3 = 4 X 3
So, Size of Decoder = n X 2^n = 2 X 4

A	B	F1	F2	F3
0	0	1	0	0
0	1	1	0	1
1	0	0	1	1
1	1	1	0	0

$F1 = D_0 + D_1 + D_3$
$F2 = D_2$
$F3 = D_1 + D_2$



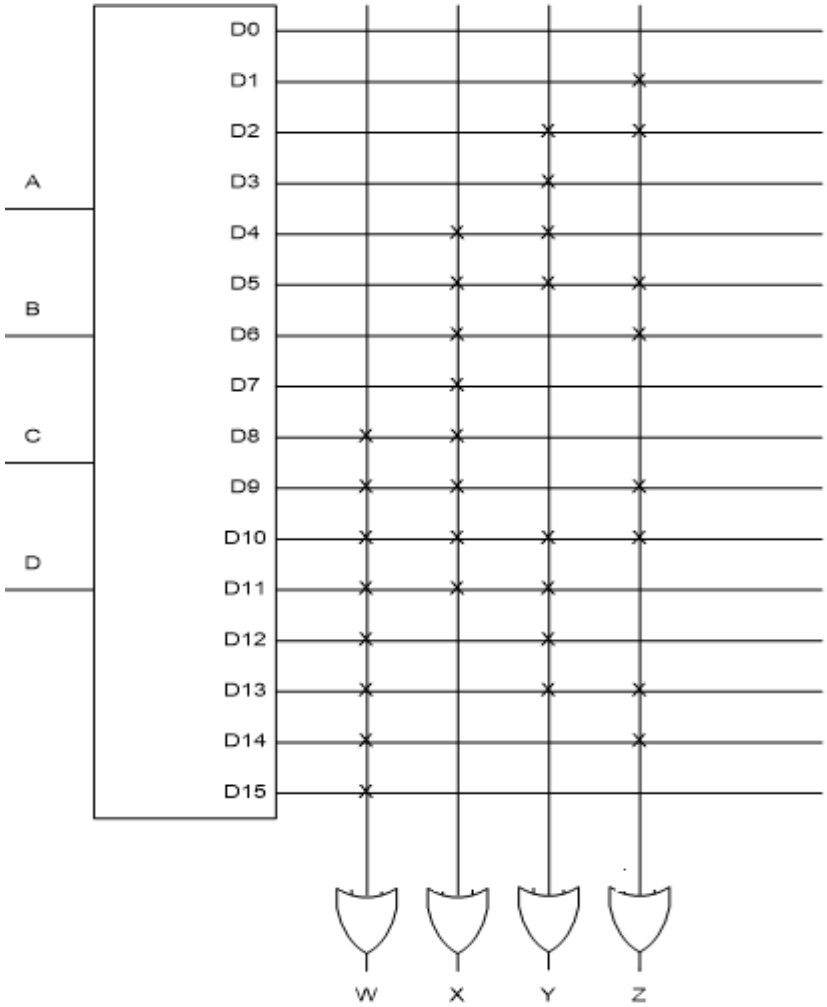
- **EXAMPLE 6: Using ROM / PROM, Implement the logic design that generates gray code for given 4 bit binary input.**

ANS:

Here, 4 bit gray code is to be designed for 4 bit binary input
So, Inputs are (n) → A,B,C,D
Outputs are (m) → W,X,Y,Z

According 2^n X m → Size of ROM = 2^4 X 4 = 16 X 4
So, Size of Decoder = n X 2^n = 4 X 16

Decimal	Binary Code ABCD	Gray Code WXYZ
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



- **EXAMPLE 7: Using 8 X 2 ROM / PROM, implement Full Adder logic design.**

ANS:

ROM size is = 8 X 2 So, according 2^n X m → Inputs = n = 3, Output = m = 2,

Size of Decoder = n X 2^n = 3 X 8

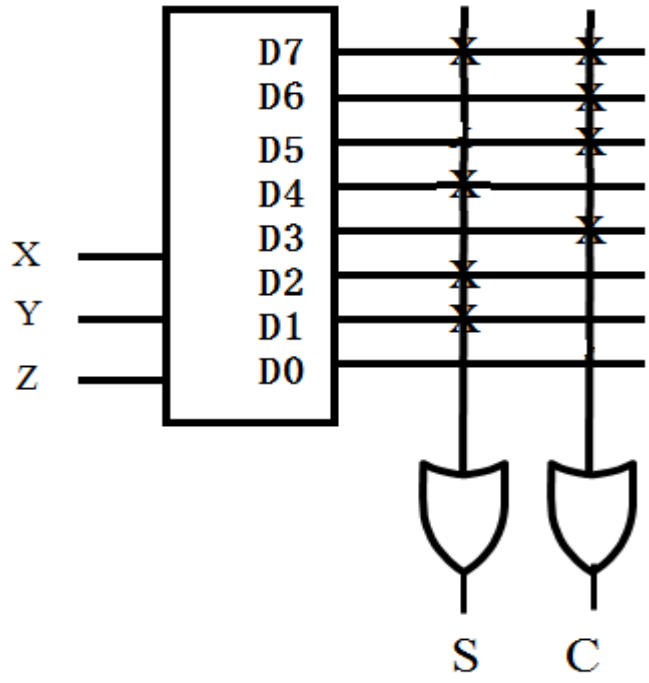
Inputs are → X,Y,Z

Outputs are → S (SUM), C (CARRY)

INPUT			OUTPUT	
X	Y	Z	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

So, $S = \sum(1,2,4,7) = D_1 + D_2 + D_4 + D_7$

$C = \sum(3,5,6,7) = D_3 + D_5 + D_6 + D_7$



- **EXAMPLE 8: Using 8 X 2 ROM / PROM, implement Full Subtractor logic design.**

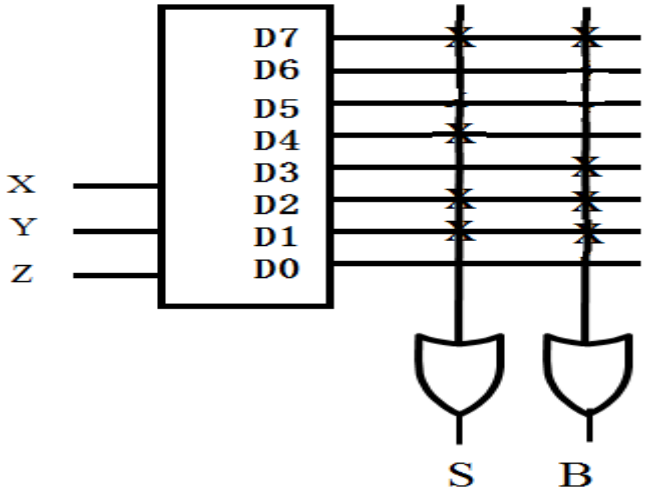
ANS:

ROM size is = 8 X 2 So, according 2^n X m → Inputs = n = 3, Output = m = 2,
Size of Decoder = n X 2^n = 3 X 8

Inputs are → X,Y,Z
Outputs are → S (SUBTRACT), B (BORROW)

INPUT			OUTPUT	
X	Y	Z	S	B
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

So, $S = \sum(1,2,4,7) = D_1 + D_2 + D_4 + D_7$
$C = \sum(3,5,6,7) = D_1 + D_2 + D_3 + D_7$



- **EXAMPLE 9: Using 8 X 6 ROM / PROM implement design that generates square of the input at the output.**

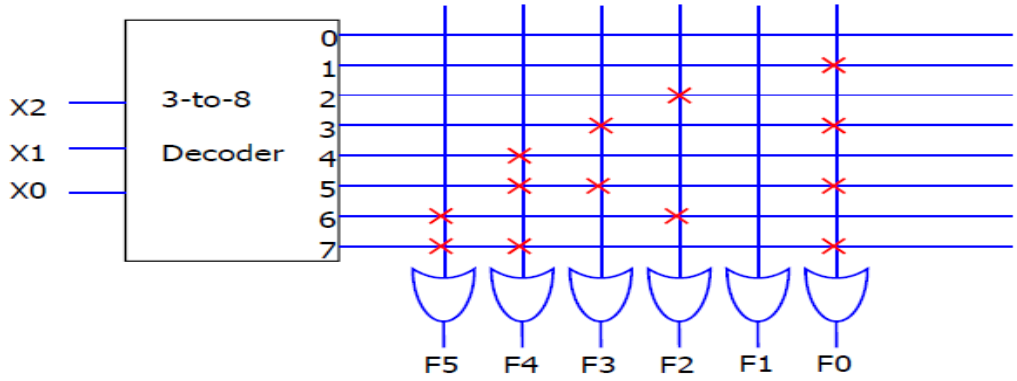
ANS:

ROM size is = 8 X 6 So, according 2^n X m → Inputs = n = 3, Output = m = 6,
Size of Decoder = n X 2^n = 3 X 8

Now design a square lookup table for **$F(X) = X^2$** using ROM.

Inputs				Outputs					
X2	X1	X0	SQ	F5	F4	F3	F2	F1	F0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	1
0	1	0	4	0	0	0	1	0	0
0	1	1	9	0	0	1	0	0	1
1	0	0	16	0	1	0	0	0	0
1	0	1	25	0	1	1	0	0	1
1	1	0	36	1	0	0	1	0	0
1	1	1	49	1	1	0	0	0	1

F5=D6+D7
F4=D4+D5+D7
F3=D3+D5
F2=D2+D6
F1=0
F0=D1+D3+D5+D7



Difference between ROPM/PROM, EPROM & EEPROM/E²PROM

ROPM/PROM	EPROM	EEPROM/E²PROM
One time programmable so erasing is not possible	Technique used for erasing is UV light	A voltage of 20 V to 25 V is applied to erasing
Erasing is not possible	Selective erasing is not possible. All the locations get erased	Selective erasing is possible. A particular locations can be erased
Erasing is not possible	Erasing can be done in 10 to 15 min.	Erasing can be done in 10 ms
Less expensive	Moderate cost	More expensive

9.2.2.2 Programmable Array Logic (PAL):

- PAL is most commonly used type of PLD. It is a programmable array of logic gates.
- The array of logic gates is on single chip and it is in the AND-OR configuration.
- The special feature of PLA is that a programmable AND array and fixed OR array.
- Also note that in each OR gate in the OR array gets input from some of the AND gates. That means output of all AND gates are not applied to any of the OR gates.
- Basic block diagram is shown as below.

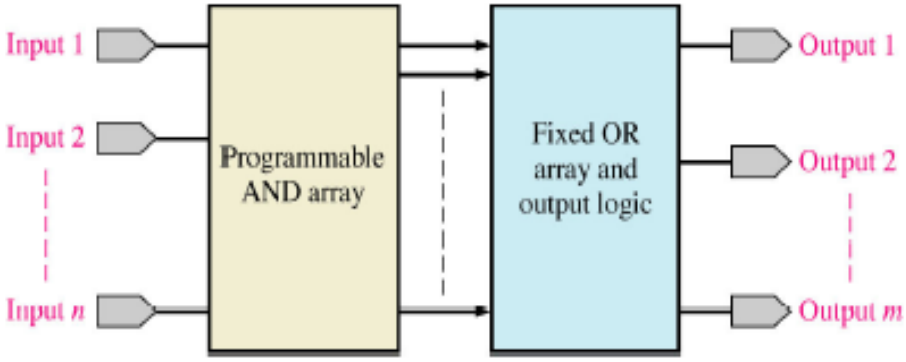
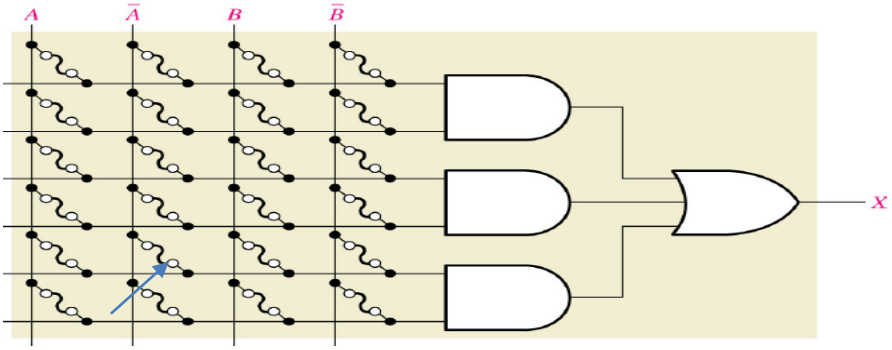


Fig. 9.6: Block diagram of ROM

- Un-programmed and programmed PAL are shown in below figure.



Fusible Link

Fig. 9.7: Un-programmed PAL

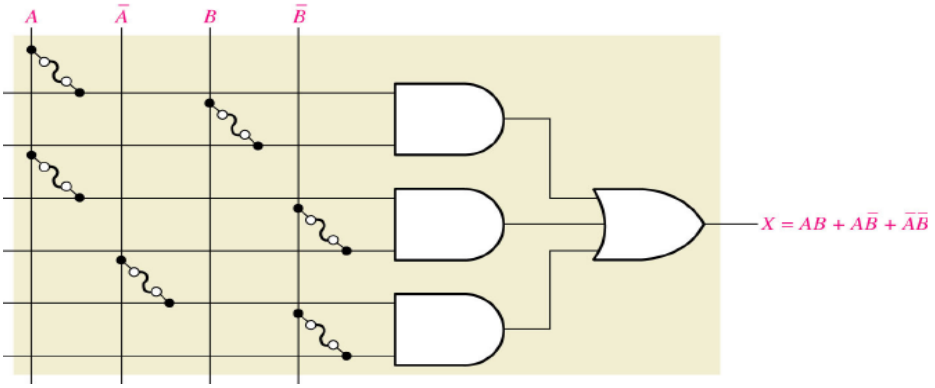


Fig. 9.8: Programmed PAL

- In Un-programmed PAL all the links are connected with Fusible Link as shown in below figure.
- As per required output function one needs to burn the fusible link and this kind of PAL is known as a programmed PAL.
- Simplified representation of PAL is shown in below figure.

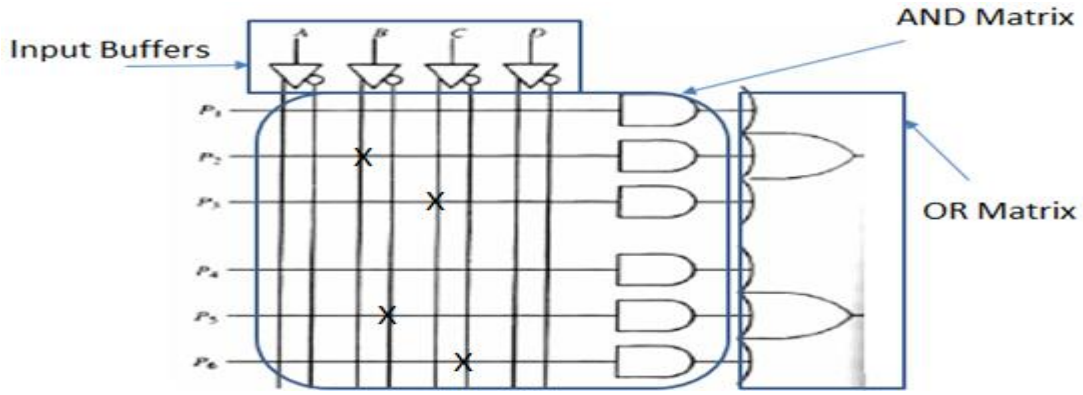


Fig. 9.9: Simplified representation programmed PAL

i. Input Buffers:

- Input buffer in a PAL is used for avoiding the loading of sources connected at the inputs.
- The buffer produce inverted and non-inverted versions of their corresponding inputs.
- One such buffer is used for each of the input lines as shown in above figure.

ii. AND Matrix:

- AND matrix is shown as above figure.
- The (X) mark indicate that a connection is present. Each AND gate has 2M input which are shown only by a single line (e.g. A,B,C, etc….). Where M is the No. of inputs.
- When a logic function is to be implemented, we have to program the array. In programming the desired connections are left with the (X) marks and such mark is not used when connection is not required.

iii. OR Matrix:

- OR matrix is shown as above figure. In PAL fixed OR array is used so there is no need to do programming to the OR array.
- No. of OR arrays are equal to the required No. of functions at the output.

iv. Input and Output Circuit:

- The input and output circuit of PAL are similar to those PLAs.
- The No. of fusible link in PAL is equal to **2M x n.** where M = No. of available inputs and n = Corresponds to No. of product terms.

Advantages & Disadvantages of PAL

- **Advantages:**
  1. For given internal complexity, a PAL can have larger N and M.
  2. Some PALs have outputs that can be complemented, adding POS functions.
  3. No multilevel circuit implementations in ROM (without external connections from output to input). PAL has outputs from OR terms as internal inputs to all AND terms, making implementation of multi-level circuits easier.

- **Disadvantages:**
  1. n x m ROM guaranteed to implement any m functions of n inputs. PAL may have too few inputs to the OR gates.

Designing steps of Combinational Circuits using PAL

- STEP 1: Prepare the Truth Table.
- STEP 2: Write a Boolean expression in SOP form.
- STEP 3: Reduce / Find the Boolean expressions using K-map or reducing Boolean expression method.
- STEP 4: List of product terms for each of the function and decide total No. of AND & OR gates required.
- STEP 5: Decide connections of AND and OR matrix & draw logic diagram.

NOTE: Out of first three steps, all three steps may not require in all examples

- **EXAMPLE 1: A combinational circuit is defined by given truth Table for that Implement the circuit using PAL.**

Truth Table

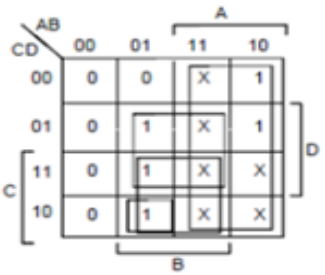
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

ANS:

- **STEP 1: Prepare the Truth Table.**
  This step is not required in this example because it is given.

- **STEP 2: Write a Boolean expression in SOP form.**
  This step is not required in this example
- **STEP 3: Find the Boolean expressions using K-map or reducing Boolean expression method.**

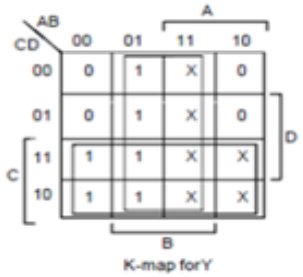
For W:



K-map forW

$$W = A + BD + BC$$

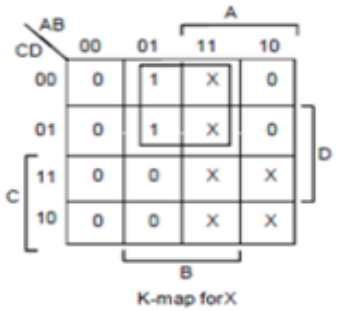
For Y:



K-map forY

$$Y = B + C$$

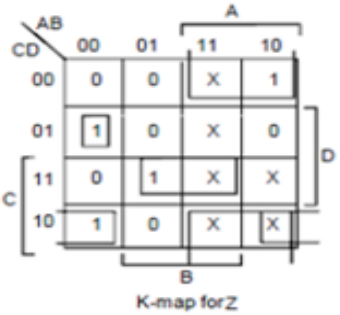
For X:



K-map forX

$$X = BC'$$

For Z:



K-map forZ

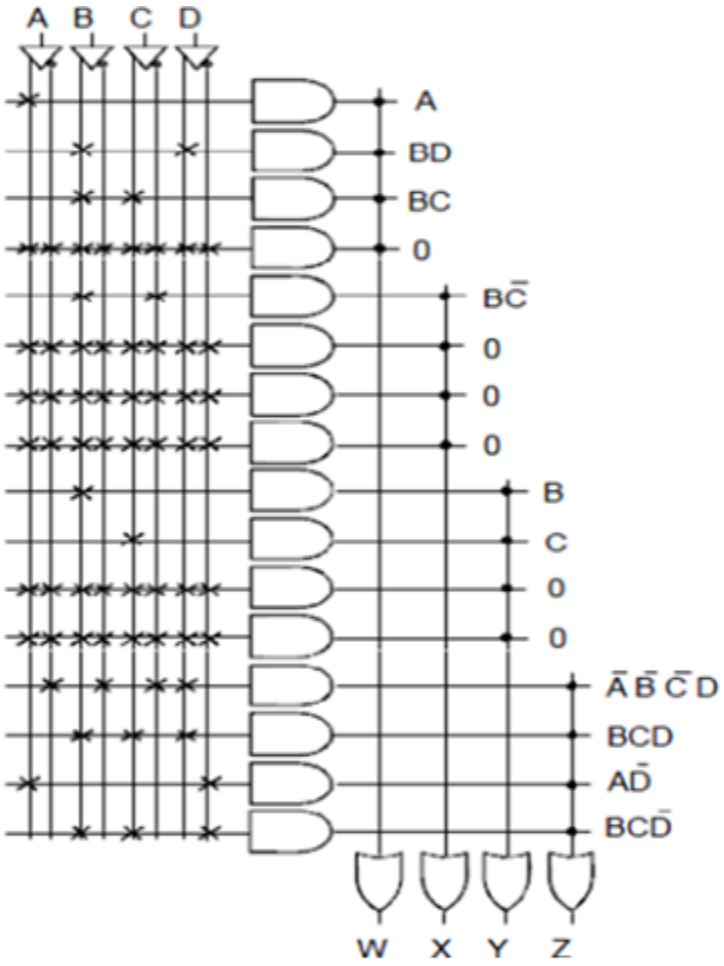
$$Z = A'B'C'D + BCD + AD' + B'CD'$$

- **STEP 4: List of product terms for each of the function and decide total No. of AND & OR gates required.**
  W = A + BD + BC
  X = BC'
  Y = B + C
  Z = A'B'C'D + BCD + AD' + B'CD'

  Total No. of AND gate = 16 (B'cz maximum No. of product terms are in function Z (4 product terms) and total No. of functions are 4 so 4 X 4 = 16).

  Total No. of OR gates = Total No. of required functions at the output side (W,X,Y,Z) = 4.

STEP 5: Decide connections of AND and OR matrix & draw logic diagram.



- **EXAMPLE 2: Implement 4 X 1 multiplexer using PAL.**

ANS:

- **STEP 1: Prepare the Truth Table.**

Select Inputs S_1 S_0	Outputs Y
0 0	D_0
0 1	D_1
1 0	D_2
1 1	D_3

- **STEP 2: Write a Boolean expression in SOP form.**
  The expression can be obtain from above truth table is,
  $$Y = S_1'S_0'D_0 + S_1'S_0D_1 + S_1S_0'D_2 + S_1S_0D_3$$

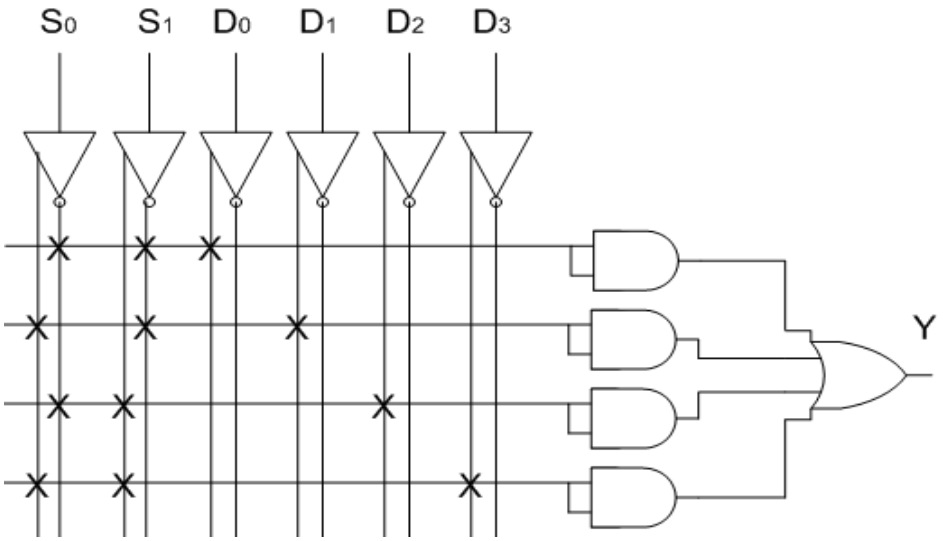
- **STEP 3: Find the Boolean expressions using K-map or reducing Boolean expression method.**
  This step is not required in this example.

- **STEP 4: List of product terms for each of the function and decide total No. of AND & OR gates required.**
  $Y = S_1'S_0'D_0 + S_1'S_0D_1 + S_1S_0'D_2 + S_1S_0D_3$

  Total No. of AND gate = 4 (B'cz No. of product terms in function Y = 4 ($S_1'S_0'D_0$, $S_1'S_0D_1$, $S_1S_0'D_2$, $S_1S_0D_3$).

  Total No. of OR gates = Total No. of required functions at the output side (Y) = 1.

- **STEP 5: Decide connections of AND and OR matrix & draw logic diagram.**



9.2.2.3 Programmable Logic Array (PLA):

- A PLD generally consist of programmable array of logic gates. Interconnections are made with the array inputs.
- PLA consist two levels of logic, an AND-plane and an OR-plane, where **both levels are programmable.**
- The outputs are connected to the device pins through inverting or non-inverting buffers and flip flops.
- The basic block diagram of a PLA is shown in below figure.

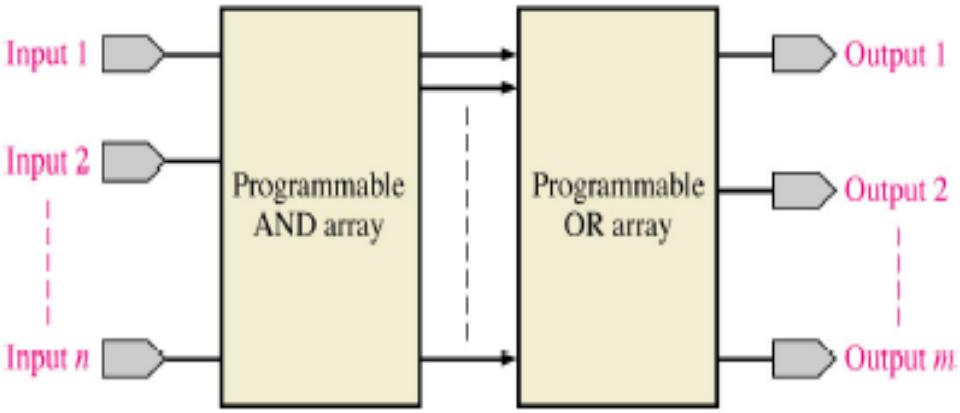


Fig. 9.10: Simplified block diagram of PLA

- Here programmable AND matrix can be used to implement the product terms in the SOP form and the programmable OR array can be used for implementing the sum of the product terms.
- Logic gates used can be two level AND-OR, NAND-NAND or NOR-NOR configuration. Sometimes AND-OR-EXOR configuration is also used. **But generally AND-OR is most preferable configuration.**
- Simplified representation of PLA is shown in below figure.

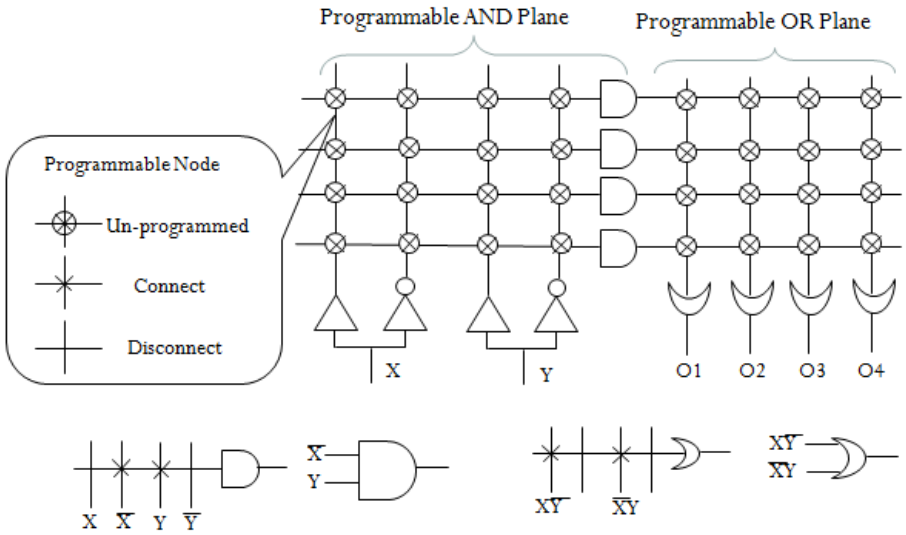


Fig. 9.11: Simplified representation of PLA

In PLA No. of outputs = No. of OR gates.
And No. of Product terms = No. of AND gates

Size of PLA is defines as M x P x N
Where M = No. of inputs, P = No. of product terms, N = No. of outputs

1. *Input Buffers:*

- Input buffer in PLA is used for avoiding the loading of sources connected at the inputs.
- Buffer of two types namely, inverted buffers and non-inverted buffers as shown in below figure.
- One such buffer is used in each of the M input lines.

Fig. 9.12: Input buffer

2. *AND Matrix:*

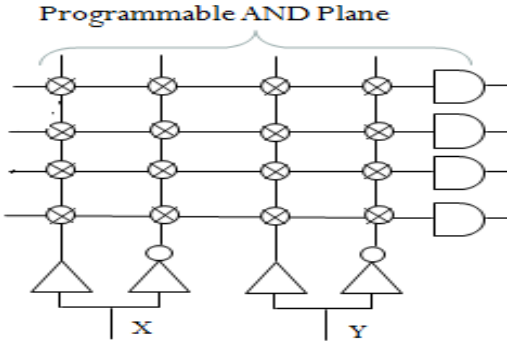


Fig. 9.13: AND matrix

- The X indicates that a connection is present. Each AND gate has 2M inputs which are shown only by single line where, M is No. of inputs (e.g. A,B,C, etc….).
- When a logic function is to be implemented, we have to program the array. In programming the desired connections are left with the (X) marks and such mark is not used when connection is not required.

3. *OR Matrix:*

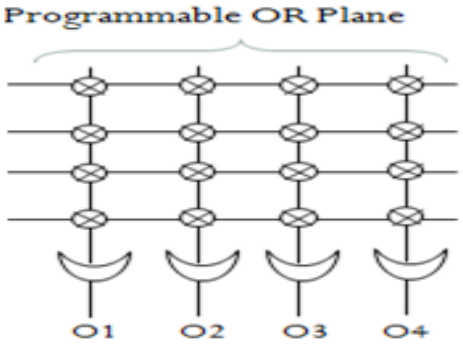


Fig. 9.14: OR matrix

- Above figure shows simplified representation of the OR matrix.
- It is possible to program the OR matrix, by open circuiting the unwanted fusible links. The open fusible links are equivalent to a '0' at the input of corresponding OR gate.

Applications of PLA:

1. We can implement combinational circuit using PLA. For this only ouput buffers are used.
2. We can also implement sequential circuit using PLA. For implement this flip flops and buffers are included in output stage.

Designing steps of Combinational Circuits using PLA

- STEP 1: Prepare the Truth Table.
- STEP 2: Write a Boolean expression in SOP form.
- STEP 3: Reduce / Find the Boolean expressions using K-map or reducing Boolean expression method.
- STEP 4: List of unique product terms.
- STEP 5: PLA table implementation.
- STEP 6: Decide connections of AND and OR matix & draw logic diagram.

NOTE: Out of first three steps, all three steps may not require in all examples

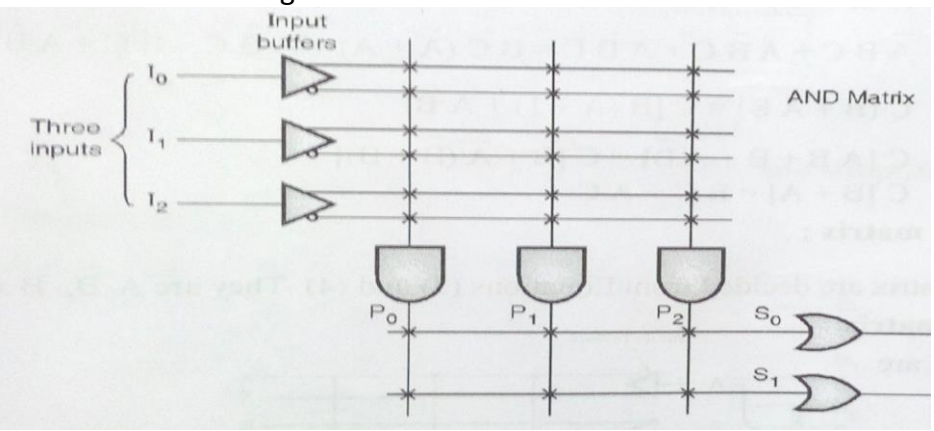
- **EXAMPLE 1: Draw combinational circuit for a PLA with three inputs, three product terms and two outputs.**

ANS:

Given No. of inputs = 3 = I_0, I_1, I_2

No. of Outputs = 2 = No. of OR gates.

NO. of product terms = No. of AND gates.



- **EXAMPLE 2: calculate the Inputs, Product terms and Outputs for 16 x 48 x 8 PLA.**

ANS:

According to the size of PLA = M x P x N = 16 x 48 x 8
So No. of inputs = M = 16
No. of output = N = 8
No. of product terms = P = 48

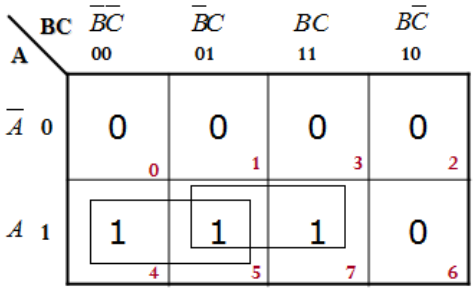
- **EXAMPLE 3: A combinational circuit is defined by the function**
  F1 (A,B,C) = ∑m (4,5,7)
  F2 (A,B,C) = ∑m (3,5,7)
  Implement the circuit using PLA.

ANS:

- **STEP 1: Prepare the Truth Table.**
  This step is not required in this example.

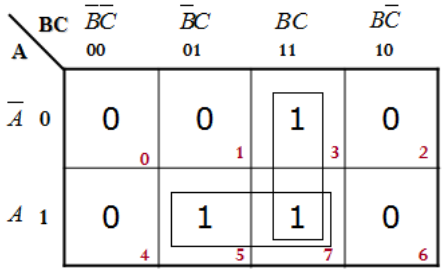
- **STEP 2: Write a Boolean expression in SOP form.**
  F1 (A,B,C) = ∑m (4,5,7) = AB'C' + AB'C + ABC
  F2 (A,B,C) = ∑m (3,5,7) = A'BC + AB'C + ABC

- **STEP 3: Reduce the Boolean expressions using K-map or reducing Boolean expression method.**
  For F1:



F1 = AB' + AC

For F2:



F2 = AC + BC

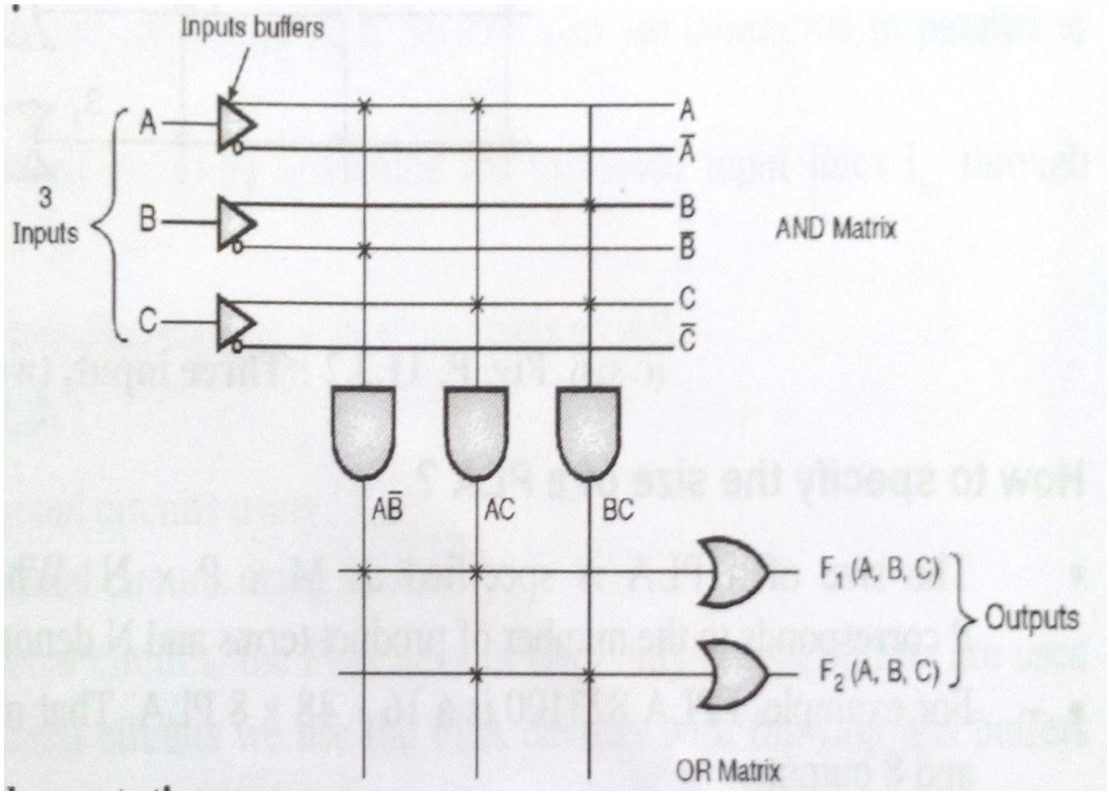
- **STEP 4: List of unique product terms.**
  Unique product terms from F1 and F2 are, AB', AC & BC

- **STEP 5: PLA table implementation.**

Product Terms	Inputs A B C	Outputs F1 F2
AB'	1 0 _	1 0
AC	1 _ 1	1 1
BC	_ 1 1	0 1

- **STEP 6: Decide connections of AND and OR matix & draw logic diagram.**

  According to the size of PLA = M x P x N
  So No. of inputs = M = 3 (A,B,C) = No. of input Buffers
  No. of output = N = 2 (F1,F2) = No. of OR gates
  No. of product terms = P = No. of AND gates = 3 (B'cz product terms are AB', AC & BC).



- **EXAMPLE 4: A combinational circuit is defined by given truth Table for that Implement the circuit using PLA.**

Truth Table

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

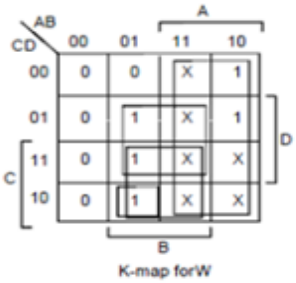
ANS:

- **STEP 1: Prepare the Truth Table.**
  This step is not required in this example because it is given.

- **STEP 2: Write a Boolean expression in SOP form.**
  This step is not required in this example

- **STEP 3: Find the Boolean expressions using K-map or reducing Boolean expression method.**

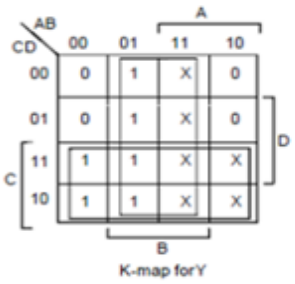
For W:



K-map forW

$W = A + BD + BC$

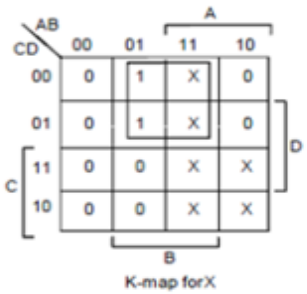
For Y:



K-map forY

$Y = B + C$

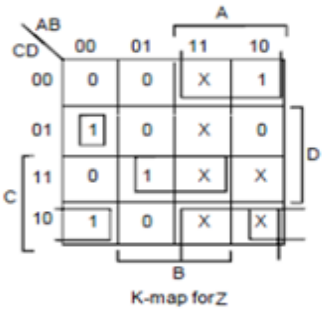
For X:



K-map forX

$X = BC'$

For Z:



K-map forZ

$Z = A'B'C'D + BCD + AD' + B'CD'$

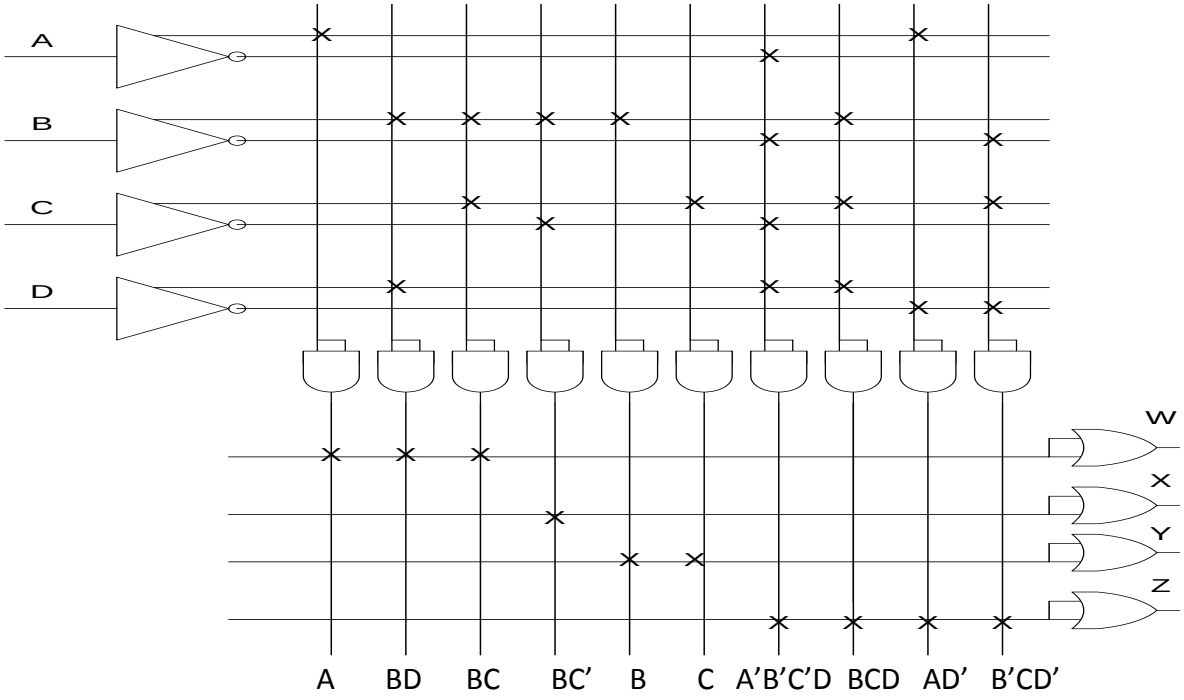
- **STEP 4: List of unique product terms.**
  Unique product terms from W,X,Y and Z are, A, BD, BC ,BC', B, C, A'B'C'D, BCD, AD', & B'CD'.

- **STEP 5: PLA table implementation.**

Product Terms	Inputs A B C D	Outputs W X Y Z
A	1 _ _ _	1 0 0 0
BD	_ 1 _ 1	1 0 0 0
BC	_ 1 1 _	1 0 0 0
BC'	_ 1 0 _	0 1 0 0
B	_ 1 _ _	0 0 1 0
C	_ _ 1 _	0 0 1 0
A'B'C'D	0 0 0 1	0 0 0 1
BCD	_ 1 1 1	0 0 0 1
AD'	1 _ _ 0	0 0 0 1
A'CD'	0 _ 1 0	0 0 0 1

- **STEP 6: Decide connections of AND and OR matix & draw logic diagram.**
  According to the size of PLA = M x P x N
  So No. of inputs = M = 4 (A,B,C,D) = No. of input Buffers

  No. of output = N = 4 (W,X,Y,Z) = No. of OR gates
  No. of product terms = P = No. of AND gates = 10 (B'cz product terms are A, BD, BC, BC', B, C, A'B'C'D, BCD, AD', B'CD').



- **EXAMPLE 5: Implement the following function using PLA.**
  F1 = A'B + AC' + A'BC'
  F2 = (AC' + B'C)'

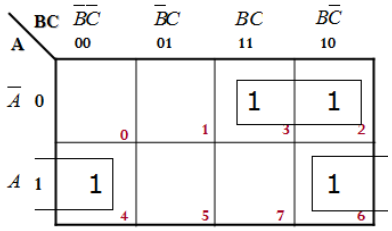
ANS:

- **STEP 1: Prepare the Truth Table.**
  This step is not required in this example.

- **STEP 2: Write a Boolean expression in SOP form.**
  This step is not required in this example because Boolean expression is given.

- **STEP 3: Reduce the Boolean expressions using K-map or reducing Boolean expression method.**

For F1:



F1 = A'B + AC'

For F2:

F2 = (AC' + B'C)'
F2 = (AC')' (B'C)'
F2 = (A' + C'') (B'' + C')
F2= (A' + C) (B + C')
F2 = A'B + A'C' + BC + CC'
F2 = A'B + A'C' + BC

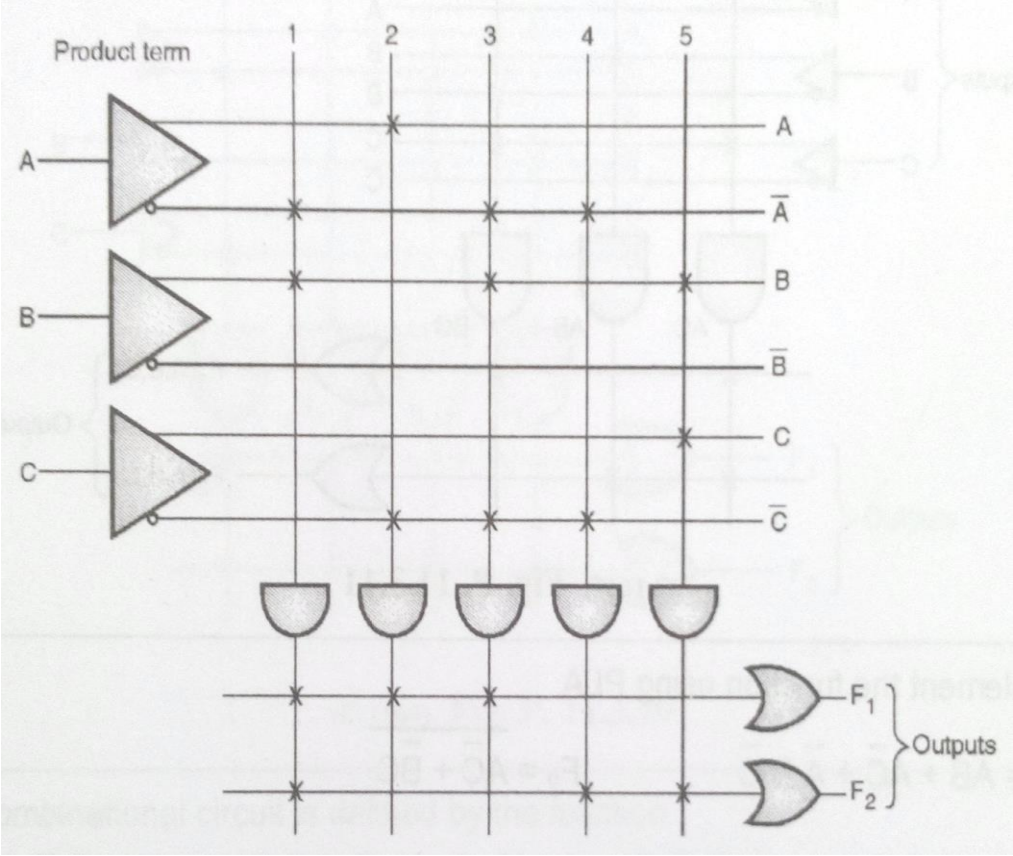
- **STEP 4: List of unique product terms.**
  Unique product terms from F1 and F2 are, A'B, AC', A'C' & BC.

- **STEP 5: PLA table implementation.**

Product Terms	Inputs A B C	Outputs F1 F2
A'B	0 1 _	1 1
AC'	1 _ 0	1 0
A'C'	0 _ 0	0 1
BC	_ 1 1	0 1

- **STEP 6: Decide connections of AND and OR matix & draw logic diagram.**

  According to the size of PLA = M x P x N
  So No. of inputs = M = 3 (A,B,C) = No. of input Buffers
  No. of output = N = 2 (F1,F2) = No. of OR gates
  No. of product terms = P = No. of AND gates = 4 (B'cz product terms are A'B, AC', A'C', BC).



- **EXAMPLE 6: Implement following equations using PLA with use of 4 inputs, 6 AND plane & 3 OR plane.**

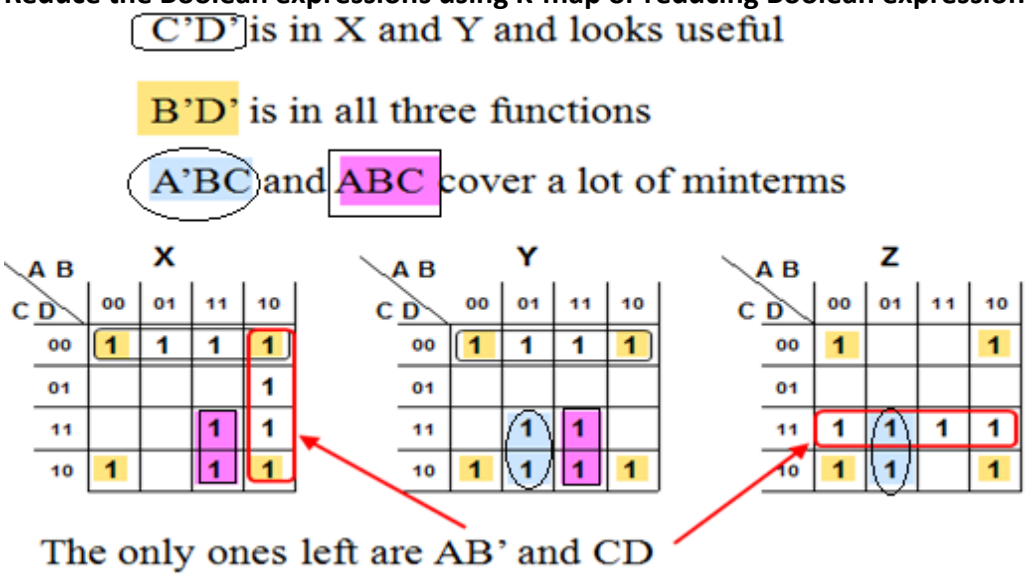
  X = ABC + B'D' + AB'D + C'D'
  Y = BC + D'
  Z = CD + B'D' + A'BC

ANS:

- **STEP 1: Prepare the Truth Table.**
  This step is not required in this example.

- **STEP 2: Write a Boolean expression in SOP form.**
  This step is not required in this example because Boolean expression is given.

- **STEP 3: Reduce the Boolean expressions using K-map or reducing Boolean expression method.**



- **STEP 4: List of unique product terms.**
  Unique product terms from X, Y & Z are C'D', B'D', A'BC, ABC, AB' & CD.

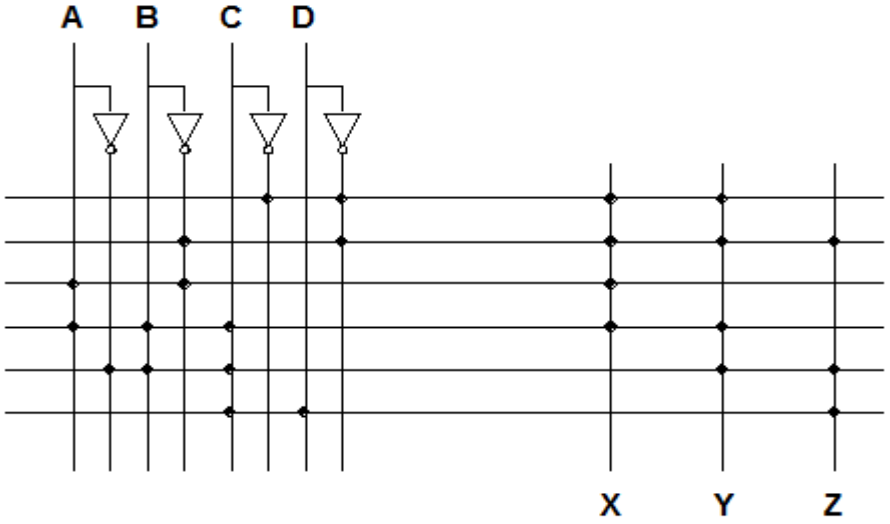
- **STEP 5: PLA table implementation.**

$$X = C'D' + B'D' + AB' + ABC$$
$$Y = C'D' + B'D' + ABC + A'BC$$
$$Z = B'D' + CD + A'BC$$

Product Term	Input A	B	C	D	Output X	Y	Z
C'D'	-	-	0	0	1	1	0
B'D'	-	0	-	0	1	1	1
AB'	1	0	-	-	1	0	0
ABC	1	1	1	-	1	1	0
A'BC	0	1	1	-	0	1	1
CD	-	-	1	1	0	0	1

- **STEP 6: Decide connections of AND and OR matix & draw logic diagram.**

According to the size of PLA = M x P x N
So No. of inputs = M = 4 (A,B,C,D) = No. of input Buffers
No. of output = N = 3 (X,Y,Z) = No. of OR gates
No. of product terms = P = No. of AND gates = 6 (B'cz product terms are C'D', B'D', A'BC, ABC, AB' & CD).



Difference between ROM, PAL and PLA.

ROM/PROM	PAL	PLA
Consist of fixed AND gate array and programmable OR array.	Consist of programmable AND gate array and fixed OR array.	Consist of programmable AND gate array and programmable OR array.
Medium speed	High speed (only one programmable gates arrays)	Slow (two programmable gates arrays)
Cheap (High-volume component)	Intermediate cost (less than PLA)	Most expensive (Most complex in design)
Not flexible	Not flexible	Offering maximum programming flexibility
It is possible to decode any minterms	We can get any desired minterms by programming the AND matrix	We can get any desired minterms by programming the AND & OR matrix
SOP function in the standard form only can be implemented	Any SOP function can be implemented	Any SOP function can be implemented

9.2.2.3 Programmable Logic Sequencer (PLS):

- The PLS is essentially a registered PLA.
- The PLS chip leads to a very compact method of state machine realization.
- As shown in figure 9.15 PLS includes a PLA along with several edge-triggered flip flops on a single chip.
- This family of programmable chip uses fusible link technology and therefore can only be programmed one time

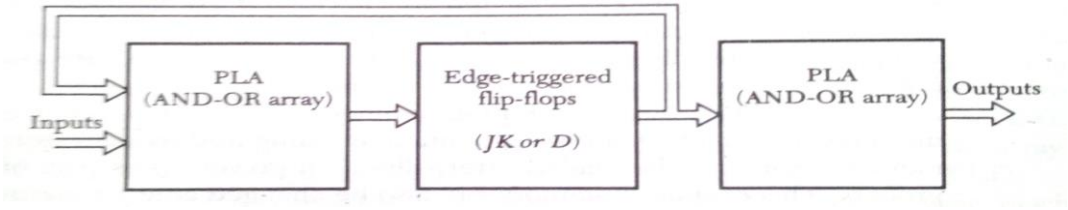


Fig 9.15: Block diagram of PLS

- Figure 9.16 represents a simplified schematic of PLS.
- The AND gate array can be driven by inputs or their components and flip flop outputs or their complements.
- One section of these AND and OR gates is allocated to the IFL and these gates are driven by inputs and the present state of the flip flops.
- A second section of AND and OR gates is allocated to OFL.
- Output can be unconditional, driven only by the state flip flops, or conditional driven by inputs and state flip flops.
- Since the outputs are coupled back to the inputs of the AND gate array, various delay can be created.
- There are several flip flops on same PLS chip that allow an output holding register to be implemented.

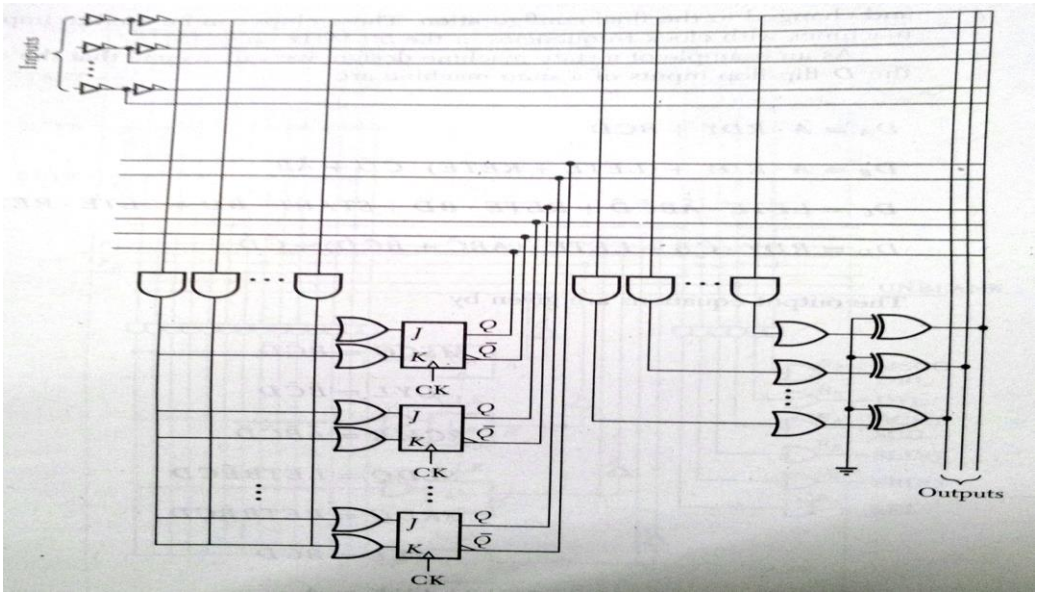


Fig 9.16: Circuit diagram of PLS

9.2.3 More Complex Logic Devices:-

9.2.3.1 FPGA (Field Programmable Gate Arrays)

- FPGA were introduced in 1985 by Xilinx.
- FPGAs are pre-fabricated silicon devices that can be electrically programmed to become almost any kind of digital circuit or system.
- Requires 3.3 V/5 V/upto 1.5 V.
- Very less cost of approx. 15$.
- Program is reloaded every time at power up i.e. need reprogramming each time they are powered up.
- The architecture is complex.
- Larger capacity in terms of gate count.
- Manufacturers – Xilinx, Altera, Atmel, Lattice Semiconductor, Actel, etc.

FPGA Basic Structure & Block Diagram

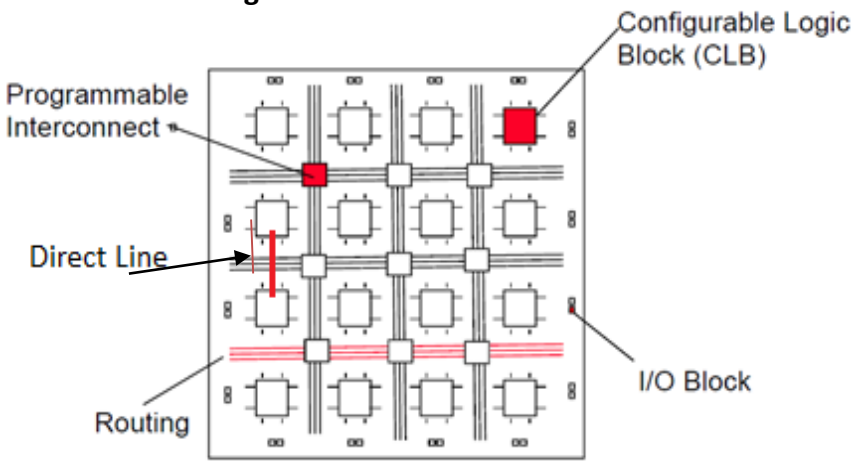


Fig 9.17: Structure and block diagram of FPGA

- FPGA is made up of three basic blocks:
  1. CLB: Configurable Logic Blocks are where the user specific functions are calculated.
  2. IOB: The Input/ Output block make it possible to connect the FPGA to the other elements of the Application.
  3. PI: Programmable Interconnect is essential for writing between CLB and from IOBs to CLBs.
- Direct Line: it is used to connect two CLBs without using switching matrix.
- The internal resources of an FPGA chip consist of a matrix of Configurable Logic Blocks (CLBs) surrounded by a periphery of I/O blocks as shown in figure.
- Signals are routed within the FPGA matrix by Programmable Interconnect (PI) switches and wire routes.

FPGA Internal Structure of CLBs

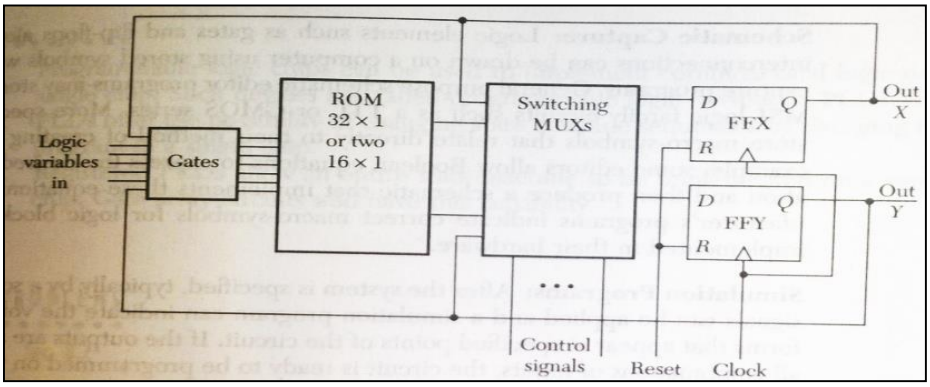


Fig 9.18: Internal structure of CLB block

- Similar to two Flip-Flop & RAM controlled State Machine.
- Combinational logic is performed by RAM that can be used as 32 X 1 or two 16 X 1 RAMs.
- Inputs to RAM are input variables and the state flip-flop outputs.
- The switching MUXs can direct the outputs of the combinational logic as well as flip-flop outputs to the inputs of the D flip-flops
- RAM & MUXs make up the IFL for the state flip-flops.

FPGA Programmable Interconnect

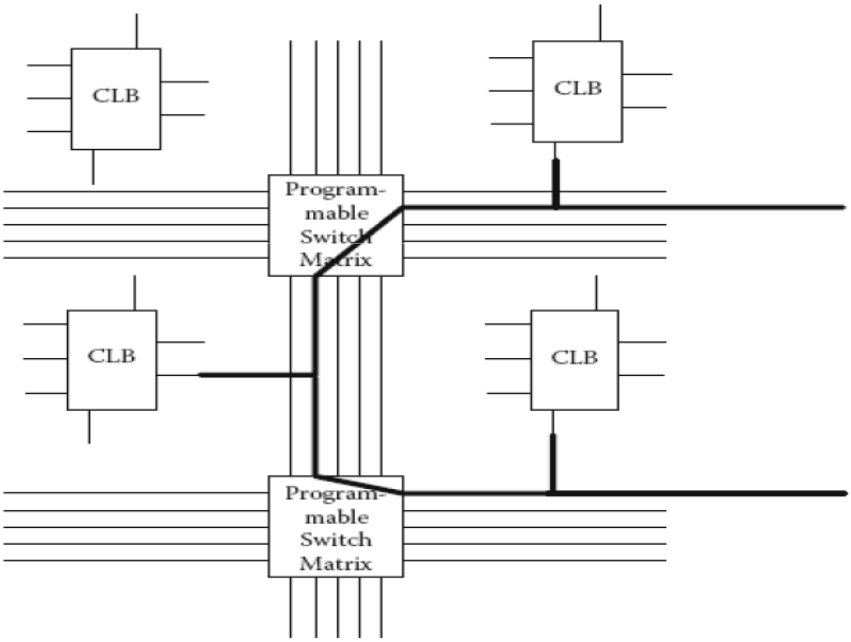


Fig 9.19: FPGA programmable interconnect

- Multiple copies of CLB slices are arranged in a matrix on the surface of the chip
- CLBs are connected column-wise & row-wise.
  At the intersections of columns & rows are programmable switch matrices.
- In this figure the output of one CLB is connected with the inputs of two other CLBs.
- The signal passes through three PIs and two PSMs.
- PSMs make the FPGA versatile, but they slow down the signals.
- There are extra routes for e.g. reset lines or clock lines.

FPGA Input / Output (I/O) Block

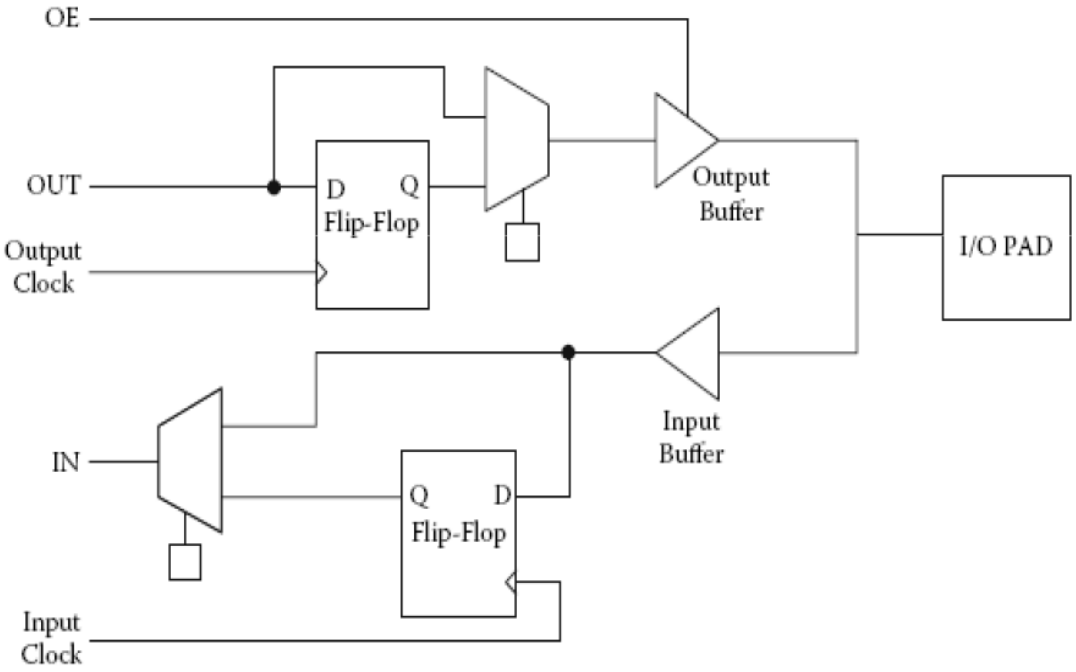


Fig 9.20: FPGA I/O block

- Input/Output Blocks are used to get the signals into the FPGA and out of the FPGA.
- Fig. shows simplified IOB. Each IOB can be used as input & output depending on the state of the Output Enable (OE).
- If OE is set to one, then IOB acts as an output, otherwise as an input.
  IOBs contain D Flip Flops for latching the input & output signals. The latches can be bypassed by appropriately programmed MUXs.

EXERCISE:

- **EXAMPLE 1:** Design a combinational logic circuit using ROM / PROM, the circuit accepts the three bit binary number and generates its equivalent excess 3 code.

- **EXAMPLE 2:** Design a 3 : 8 Decoder using PROM / ROM

- **EXAMPLE 3:** Implement the following Boolean functions using PAL.
  W (A,B,C,D) = ∑m (1,3,4,6,9,11,12,14)
  X (A,B,C,D) = ∑m (1,3,4,6,9,11,12,14,15)
  Y (A,B,C,D) = ∑m (0,2,4,6,8,12)
  Z (A,B,C,D) = ∑m (2,3,8,9,12,13)

- **EXAMPLE 4:** Implement the following Boolean functions using PAL.
  W (A,B,C,D) = ∑m (2,12,13)
  X (A,B,C,D) = ∑m (7,8,9,10,11,12,13,14,15)
  Y (A,B,C,D) = ∑m (0,2,3,4,5,6,7,8,10,11,15)
  Z (A,B,C,D) = ∑m (1,2,8,12,13)

- **EXAMPLE 5:** Implement the following Boolean functions using PAL.
  F (A,B,C,D) = ∑m (0,1,3,15)

- **EXAMPLE 6:** Design a BCD to Excess-3 code converter and implement it using PAL.

- **EXAMPLE 7:** Implement following function using PLA.
  F1 (A,B,C) = ∑m (1,3,5)
  F2 (A,B,C) = ∑m (2,4,5)

- **EXAMPLE 8:** Implement 3-bit binary to gray code & 3-bit Gray code to binary converter using PLA.

- **EXAMPLE 9:** Implement following function using PLA.
  F(A,B,C,D) = ΠM (0,3,5,7,12,15) · d (2,9)

- **EXAMPLE 10:** Implement following function using PLA.
  F1 (A,B,C) = ∑m (1,3,5)
  F2 (A,B,C) = ∑m (2,4,5)

- **EXAMPLE 11:** Implement following function using PLA.
  F1 (A,B,C) = ∑m (0,2,5,7)
  F2 (A,B,C) = ∑m (2,3,4,5)

- **EXAMPLE 12:** Implement following function using PLA.
  $F_1 = AB' + C$
  $F_2 = A'C' + BC$
  $F_3 = AB' + A'C'$
  $F_4 = A'C' + BC + AB'C$

- **EXAMPLE 13:** Implement 3-bit Gray code to binary converter using PLA.

- **EXAMPLE 14:** Implement BCD to Excess-3 code converter using PLA.

- **EXAMPLE 15:** Implement BCD to gray code converter using PLA.