Unit 1 Introduction

1.1 Digital Signals and Wave Forms

1.2 Digital Logic and Operation

1.3 Digital Computer and Integrated Circuits (IC)

1.4 Clock Wave Form [2 Hrs.]

1.1 Digital Signals and Wave Forms

A digital signal is a signal that represents data as a sequence of discrete values; at any given time, it can only take on, at most, one of a finite number of values. This contrasts with an analog signal, which represents continuous values; at any given time, it represents a real number within a continuous range of values.

Signal Vs WaveForms

The major difference between the two is the conveying of information. A signal is always talked about in terms of conveying some information about some phenomenon. Wave is talked about in terms of transferring some energy.

A wave can be considered a part of a signal, like a 1 MHz sinusoidal signal will have a million sine waves per second.

Also, a signal can be talked about in terms of images , video, speech etc. Whereas, waves are classified in terms of the material medium in which they travel , like mechanical or electromagnetic. or the wavelength.

1.2 Digital Logic and Operation

Digital logic is the underlying logic system that drives electronic circuit board design. Digital logic is the **manipulation of binary values through printed circuit board technology** that uses circuits and logic gates to construct the implementation of computer operations.

Digital logic has three basic operators, the AND, the OR and the NOT. These three operators form the basis for everything in digital logic. In fact, almost everything your computer does can be described in terms of these three operations.

1.3 Digital Computer and Integrated Circuits (IC)

There are two types of computer.

Diff b/w : Analog and Digital Computers

Analog	Digital
Analog computer works with continuous values.	Digital computers works with discrete value (**0**,**1**). It can work only with digits
It has very limited memory.	It can store large amount of data.
It has no state.	It has two states *on* and *off*
Its speed of calculation is slow	*Its speed of calculation is very high.*
It is difficult to use	*It is easy to use.*
Analog computers is used in engineering and scientific applications.	*Digital computer is widely used in almost all fields of life.*
Analog computer is used for calculations and measurement of physical quantities such as weight, height, temperature and speed.	*Digital computer is used to calculate mathematical and logical operations.*
It can perform certain types of calculations	*It can perform all types of calculations*
Examples: Thermometer, analog clock, older weighing machines. Car speedometer, voice , radio/tv signal etc.	***Examples**: digital watches, digital weighing machines, mini computers, microcomputers, mainframe computers and super computers.*

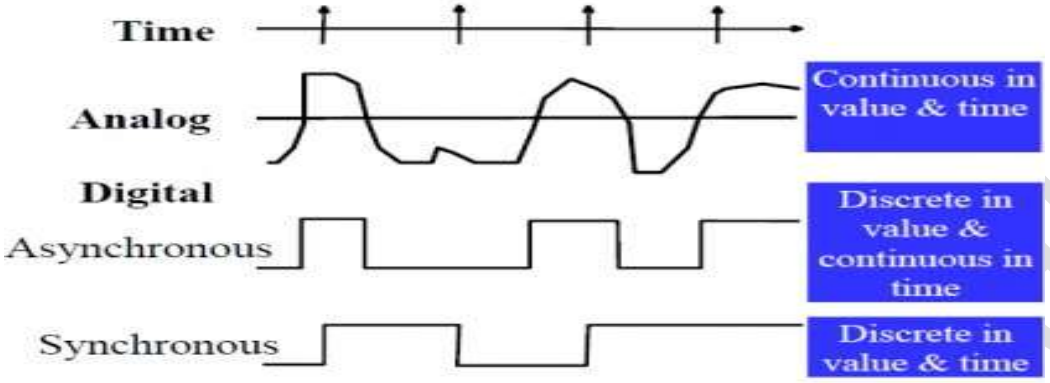
An integrated circuit or monolithic integrated circuit (also referred to as an IC, a chip, or a microchip) is a set of electronic circuits on one small flat piece (or "chip") of semiconductor material, usually silicon. Large numbers of tiny MOSFETs (metal–oxide–semiconductor field-effect transistors) integrate into a small chip. This results in circuits that are orders of magnitude smaller, faster, and less expensive than those constructed of discrete electronic components. The IC's mass production capability, reliability, and building-block approach to integrated circuit design has ensured the rapid adoption of standardized ICs in place of designs using discrete transistors. ICs are now used in virtually all electronic equipment and have revolutionized the world of electronics. Computers, mobile phones, and other digital home appliances are now inextricable parts of the structure of modern societies, made possible by the small size and low cost of ICs such as modern computer processors and microcontrollers.

1.4 Clock Wave Form

What is a clock waveform?

Digital waveforms are referenced to clock signals. ... A clock signal is **a square wave with a fixed period**. The period is measured from the edge of one clock to the next similar edge of the clock; most often is it measured from one rising edge to the next.

Signal Examples over time



Unit 2 Number Systems5 Hrs.

2.1 Binary, Octal, & Hexadecimal Number Systems and Their Conversions

2.1.1 Representation of Signed Numbers-Floating Point Number

2.1.2 Binary Arithmetic

2.2 Representation-of BCD-ASCII-Excess

3 -Gray Code —Error Detecting and Correcting Codes.

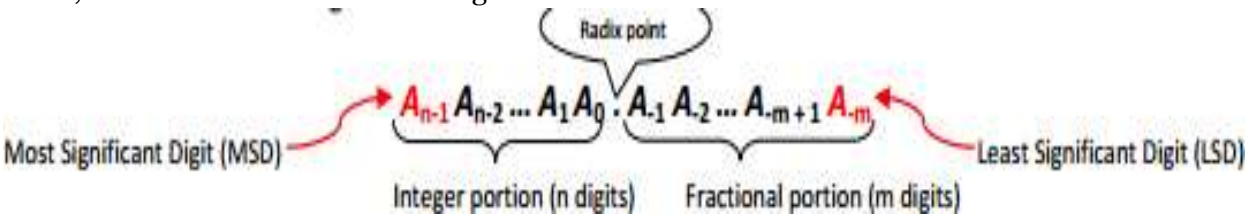
2.1 Binary, Octal, & Hexadecimal Number Systems and Their Conversions

Number Systems

Here we discuss positional number systems with Positive radix (or base) r. A number with radix r is represented by a string of digits as below i.e. wherever you guys see numbers of whatever

BCA Guides Nepal

bases, all numbers can be written in general as:



in which 0 ≤ Ai < r (since each being a symbol for particular base system viz. for r = 10 (decimal number system) Ai will be one of 0,1,2,...,8,9). Subscript i gives the position of the coefficient and, hence, the weight ri by which the coefficient must be multiplied.

HEY! Confused? Don't worry! I will describe specific number systems (r=2, 8, 10 and 16) used in digital computers later one by one, then the concept will be quite clear.

In general, a number in base r contains r digits, 0, l, 2... r- 1, and is expressed as a power series in r with the general form:

(Number)r $= A_{n-1}+r^{n-2} + \cdots + A_1 r^1 + A_0 r^0 + A_{-1} r^{-1} + \cdots + A_{-m+1} + r^{-m=+1} + A_{-m} r^{-m}$

$$(Number)_r = \left(\sum_{i=0}^{i=n-1} A_i\, r^i\right) + \left(\sum_{i=-m}^{i=-1} A_i\, r^i\right)$$

Decimal Number System (Base-10 system)

Radix (r) = 10

Symbols = 0 through r-1 = 0 through 10-1 = {0, 1, 2... 8, 9}

I am starting from base-10 system since it is used vastly in everyday arithmetic besides computers to represent numbers by strings of digits or symbols defined above, possibly with a decimal point. Depending on its position in the string, each digit has an associated value of an integer raised to the power of 10.

Example: decimal number 724.5 is interpreted to represent 7 hundreds plus 2 tens plus 4 units plus 5 tenths.

$724.5 = 7 \text{ X } 10^2 + 2 \text{ X } 10^1 + 4 \text{ X } 10^0 + 5 \text{ X } 10^{-1}$

Binary Number System (Base-2 system)

Radix (r) = 2

Symbols = 0 through r-1 = 0 through 2-1 = {0, 1}

A binary numbers are expressed with a string of 1'sand 0's and, possibly, a binary point within it. The decimal equivalent of a binary number can be found by expanding the number into a power series with a base of 2.

Example: (11010.01)2 can be interpreted using power series as:

$(11010.01)^2 = 1 \text{ X } 2^4 + 1 \text{ X } 2^3 + 0 \text{ X } 2^2 + 1 \text{ X } 2^1 + 0 \text{ X } 2^0 + 0 \text{ X } 2^{-1} + 1 \text{ X } 2^{-2} = (26.25)_{10}$

BCA Guides Nepal

Digits in a binary number are called bits (Binary digits). When a bit is equal to 0, it does not contribute to the sum during the conversion. Therefore, the conversion to decimal can be obtained by adding the numbers with powers of 2 corresponding to the bits that are equal to 1. Looking at above example,

$$(11010.01)_2 = 16 + 8 + 2 + 0.25 = (26.25)_{10}$$

In computer work,

- 2^{10} is referred to as K (kilo),
- 2^{20} as M (mega),
- 2^{30} as G (giga),
- 2^{40} as T (tera) and so on.

n	2^n	n	2^n	n	2^n
0	1	8	256	16	65,536
1	2	9	512	17	131,072
2	4	10	1,024	18	262,144
3	8	11	2,048	19	524,288
4	16	12	4,096	20	1,048,576
5	32	13	8,192	21	2,097,152
6	64	14	16,384	22	4,194,304
7	128	15	32,768	23	8,388,608

Table: Numbers obtained from 2 to the power of n

Octal Number System (Base-8 system)

Radix (r) = 8
Symbols = 0 through r-1 = 0 through 8-1 = {0, 1, 2...6, 7}
An octal numbers are expressed with a strings of symbols defined above, possibly, an octal point within it. The decimal equivalent of a octal number can be found by expanding the number into a power series with a base of 8.
Example: (40712.56)8 can be interpreted using power series as:

$$(40712.56)_8 = 4 \text{ X } 8^4 + 0 \text{ X } 8^3 + 7 \text{ X } 8\text{^}2 + 1 \text{ X } 8^1 + 2 \text{ X } 8^0 + 5 \text{ X } 8^{-1} + 6 \text{ X } 8^{-2} = (16842.1)_{10}$$

Hexadecimal Number System (Base-16 system)

Radix (r) = 16
Symbols = 0 through r-1 = 0 through 16-1 = {0, 1, 2...9, A, B, C, D, E, F}
A hexadecimal numbers are expressed with a strings of symbols defined above, possibly, a hexadecimal point with in it. The decimal equivalent of a hexadecimal number can be found by expanding the number into a power series with a base of 16.

Example: (4D71B.C6)16 can be interpreted using power series as:

$(4D71B.C6)_{16} = 4 \text{ X } 16^4 + D \text{ X } 16^3 + 7 \text{ X } 16^2 + 1 \text{ X } 16^1 + B \text{ X } 16^0 + C \text{ X } 16^{-1} + 6 \text{ X } 16^{-2}$
$= 4 \text{ X } 164 + 13 \text{ X } 163 + 7 \text{ X } 162 + 1 \text{ X } 161 + 11 \text{ X } 160 + 12 \text{ X } 16\text{-}1 + 6 \text{ X } 16\text{-}2$
$= (317211.7734375)_{10}$

Number Base Conversions

Case I: **Base-r system to Decimal**: Base-r system can be binary (r=2), octal (r=8), hexadecimal (r=16), base-60 system or any other. For decimal system as destination of conversion, we just use power series explained above with varying r and sum the result according to the arithmetic rules of base-10 system. I have already done examples for binary to decimal, octal to decimal and hexadecimal to decimal.

For refreshment let's assume base-1000 number $(458HQY)_{1000}$. Where n = 6 and m = 0. $(458HQY)_{1000} = A_{n-1} r^{n-1} + A_{n-2} r^{n-2} + \ldots + A_1 r^1 + A_0 r^0 + A_{-1} r^{-1} + A_{-2} r^{-2} + \ldots + A_{-m+1} r^{-m+1} + A_{-m} r^{-m}$

$= 4 \text{ X } 1000^5 + 5 \text{ X } 1000^4 + 8 \text{ X } 1000^3 + H \text{ X } 1000^2 + Q \text{ X } 1000^1 + Y \text{ X } 1000^0$

=Resulting number will be in decimal. Here I have supposed various symbols for base-1000 system. Don't worry, if someone gives you base-1000 number for conversion, he should also define all 1000 symbols (0-999).

Case II: Decimal to Base-r system: Conversion follows following algorithm.

1. Separate the number into integer and fraction parts if radix point is given.
2. Divide "Decimal Integer part" by base r repeatedly until quotient becomes zero andstoring remainders at each step.
3. Multiply "Decimal Fraction part" successively by r and accumulate the integer digits so obtained.
4. Combine both accumulated results and parenthesize the whole result with subscript r.

Example I: Decimal to binary

- $(41.6875)_{10} = (?)_2$

Here Integer part = 41 and fractional part = 0.6875

Integer = 41

41	
20	1
10	0
5	0
2	1
1	0
0	1

$(41)_{10} = (101001)_2$

Fraction = 0.6875

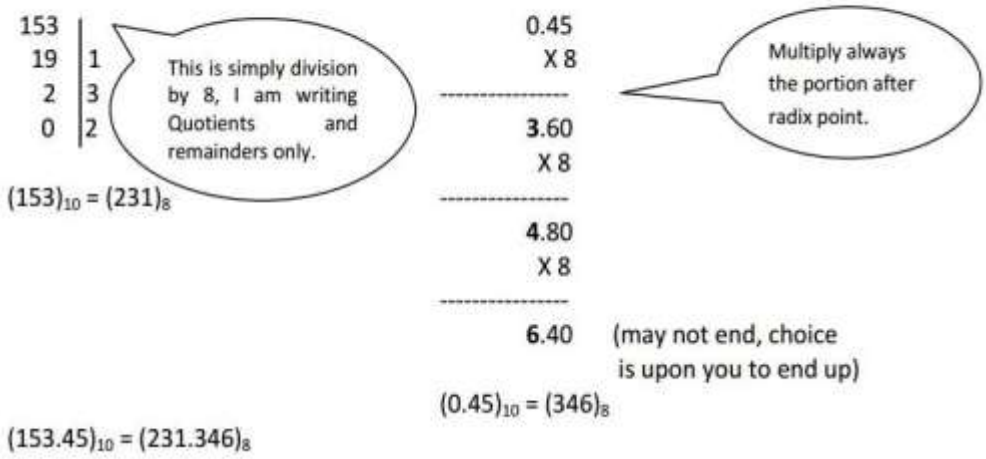
0.6875
2
1.3750
x 2
0.7500
x 2
1.5000
x 2
1.0000

$(0.6875)_{10} = (0.1011)_2$

$(41.6875)_{10} = (101001.1011)_2$

Example II: Decimal to octal

- $(153.45)_{10} = (?)_8$
  Here integer part = 153 and fractional part = 0.45



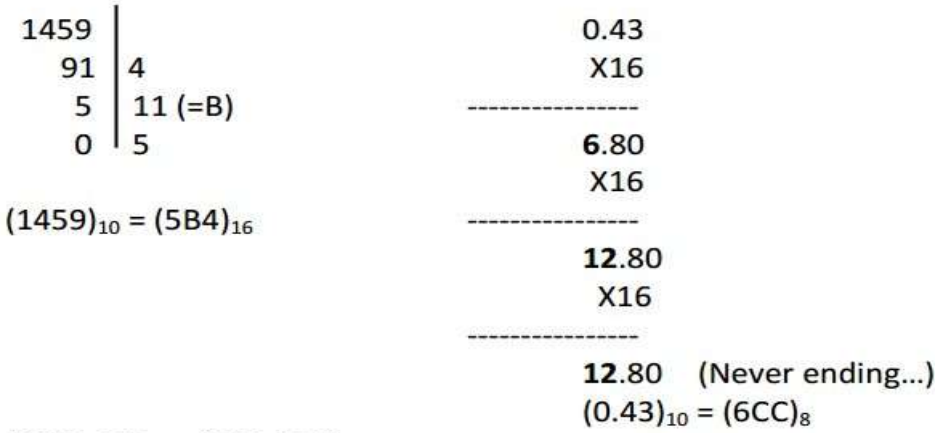
$(153)_{10} = (231)_8$

$(0.45)_{10} = (346)_8$

$(153.45)_{10} = (231.346)_8$

Example III: Decimal to Hexadecimal

- $(1459.43)_{10} = (?)_{16}$
  Here integer part = 1459 and fractional part = 0.43



$(1459)_{10} = (5B4)_{16}$

$(0.43)_{10} = (6CC)_8$

$(1459.43)_{10} = (5B4.6CC)_{16}$

Case III: Binary to octal & hexadecimal and vice-versa: Conversion from and to binary, octal and hexadecimal representation plays an important part in digital computers. Since,

- 23 = 8, octal digit can be represented by at least 3 binary digits. (We have discussed thismuch better in class). So to convert given binary number into its equivalent octal, we divide it into groups of 3 bits, give each group an octal symbol and combine the result.
  - Integer part: Group bits from right to left of an octal point. 0's can be added to make it multiple of 3 (not compulsory).
  - Fractional part: Group bits from left to right of an octal point. 0's must be added to if bits are not multiple of 3 (Note it).

- 24 = 16, each hex digit corresponds to 4 bits. So to convert given binary number into its equivalent hex, we divide it into groups of 4 bits, give each group a hex digit and combinethe result. If hex point is given, then process is similar as of octal.
- 15 numbers in 4 systems summarized below for easy reference.

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	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

Example:

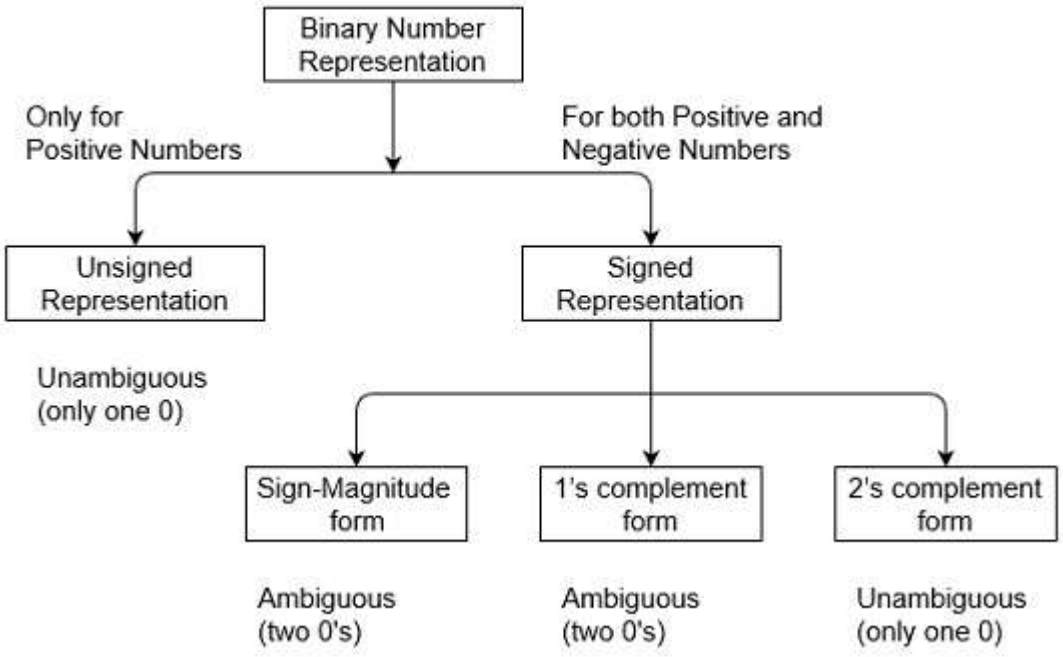
1. Binary to octal:
   $(10110001101011.11110000011)_2 = (\underbrace{010}_{2}\ \underbrace{110}_{6}\ \underbrace{001}_{1}\ \underbrace{101}_{5}\ \underbrace{011}_{3}.\ \underbrace{111}_{7}\ \underbrace{100}_{4}\ \underbrace{000}_{0}\ \underbrace{110}_{6})_2 = (26153.7406)_8$
2. Binary to hexadecimal:
   $(10110001101011.11110000011)_2 = (\underbrace{0010}_{2}\ \underbrace{1100}_{C}\ \underbrace{0110}_{6}\ \underbrace{1011}_{B}.\ \underbrace{1111}_{F}\ \underbrace{0000}_{0}\ \underbrace{0110}_{6})_2 = (2C6B.F06)_{16}$
3. From hex & octal to binary is quite easy, we just need to remember the binary of particular hex or octal digit.
   $(673.12)_8 = 110\ 111\ 011.\ 001\ 010 = (110111011.00101)_2$
   $(3A6.C)_{16} = 0011\ 1010\ 0110.\ 1100 = (1110100110.11)_2$

2.1.1 Representation of Signed Numbers-Floating Point Number

Signed numbers use **sign flag** or can be distinguish between negative values and positive values. Whereas unsigned numbers stored only positive numbers but not negative numbers.

Representation of Binary Numbers:

Binary numbers can be represented in signed and unsigned way. Unsigned binary numbers do not have sign bit, whereas signed binary numbers uses signed bit as well or these can be distinguishable between positive and negative numbers. A signed binary is a specific data type of a signed variable.



1. Unsigned Numbers:

Unsigned numbers don't have any sign, these can contain only magnitude of the number. So, representation of unsigned binary numbers are all positive numbers only. For example, representation of positive decimal numbers are positive by default. We always assume that there is a positive sign symbol in front of every number.

Representation of Unsigned Binary Numbers:

Since there is no sign bit in this unsigned binary number, so N bit binary number represent its magnitude only. Zero (0) is also unsigned number. This representation has only one zero (0), which is always positive. Every number in unsigned number representation has only one unique binary equivalent form, so this is unambiguous representation technique. The range of unsigned binary number is from 0 to (2^n-1).

Example-1: Represent decimal number 92 in unsigned binary number.

Simply convert it into Binary number, it contains only magnitude of the given number.
$= (92)_{10}$

$= (1x2^6+0x2^5+1x2^4+1x2^3+1x2^2+0x2^1+0x2^0)_{10}$

$= (1011100)_2$

It's 7 bit binary magnitude of the decimal number 92.

Example-2: Find range of 5 bit unsigned binary numbers. Also, find minimum and maximum value in this range.

Since, range of unsigned binary number is from 0 to (2^n-1). Therefore, range of 5 bit unsigned binary number is *from* 0 to (2^5-1) which is equal from minimum value 0 (i.e., 00000) to maximum value 31 (i.e., 11111).

2. Signed Numbers:

Signed numbers contain sign flag, this representation distinguish positive and negative numbers. This technique contains both sign bit and magnitude of a number. For example, in representation of negative decimal numbers, we need to put negative symbol in front of given decimal number.

Representation of Signed Binary Numbers:

There are three types of representations for signed binary numbers. Because of extra signed bit, binary number zero has two representation, either positive (0) or negative (1), so ambiguous representation. But 2's complementation representation is unambiguous representation because of there is no double representation of number 0. These are: Sign-Magnitude form, 1's complement form, and 2's complement form which are explained as following below.

2.(a) Sign-Magnitude form:

For n bit binary number, 1 bit is reserved for sign symbol. If the value of sign bit is 0, then the given number will be positive, else if the value of sign bit is 1, then the given number will be negative. Remaining (n-1) bits represent magnitude of the number. Since magnitude of number zero (0) is always 0, so there can be two representation of number zero (0), positive (+0) and negative (-0), which depends on value of sign bit. Hence these representations are ambiguous generally because of two representation of number zero (0). Generally sign bit is a most significant bit (MSB) of representation. The range of Sign-Magnitude form is from $(2^{(n-1)}-1)$ to $(2^{(n-1)}-1)$.

For example, range of 6 bit Sign-Magnitude form binary number is from (2^5-1) to (2^5-1) which is equal from minimum value -31 (i.e., 1 11111) to maximum value +31 (i.e., 0 11111). And zero (0) has two representation, -0 (i.e., 1 00000) and +0 (i.e., 0 00000).

2.(b) 1's complement form:

Since, 1's complement of a number is obtained by inverting each bit of given number. So, we represent positive numbers in binary form and negative numbers in 1's complement form. There is extra bit for sign representation. If value of sign bit is 0, then number is positive and you can directly represent it in simple binary form, but if value of sign bit 1, then number is negative and

you have to take 1's complement of given binary number. You can get negative number by 1's complement of a positive number and positive number by using 1's complement of a negative number. Therefore, in this representation, zero (0) can have two representation, that's why 1's complement form is also ambiguous form. The range of 1's complement form is *from* $(2^{(n-1)}-1)$ to $(2^{(n-1)}-1)$.

For example, range of 6 bit 1's complement form binary number is from (2^5-1) to (2^5-1) which is equal from minimum value -31 (i.e., 1 00000) to maximum value +31 (i.e., 0 11111). And zero (0) has two representation, -0 (i.e., 1 11111) and +0 (i.e., 0 00000).

2.(c) 2's complement form:

Since, 2's complement of a number is obtained by inverting each bit of given number plus 1 to least significant bit (LSB). So, we represent positive numbers in binary form and negative numbers in 2's complement form. There is extra bit for sign representation. If value of sign bit is 0, then number is positive and you can directly represent it in simple binary form, but if value of sign bit 1, then number is negative and you have to take 2's complement of given binary number. You can get negative number by 2's complement of a positive number and positive number by directly using simple binary representation. If value of most significant bit (MSB) is 1, then take 2's complement from, else not. Therefore, in this representation, zero (0) has only one (unique) representation which is always positive. The range of 2's complement form is *from* $(2^{(n-1)})$ to $(2^{(n-1)}-1)$.

For example, range of 6 bit 2's complement form binary number is from (2^5) to (2^5-1) which is equal from minimum value -32 (i.e., 1 00000) to maximum value +31 (i.e., 0 11111). And zero (0) has two representation, -0 (i.e., 1 11111) and +0 (i.e., 0 00000).

Complements

Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation. There are two types of complements for each base-r system: r's complement and the second as the (r - 1)'s complement. When the value of the base r is substituted, the two types are referred to as the 2's complement and 1's complement for binary numbers, the 10's complement and 9's complement for decimal numbers etc.

(r-1)'s Complement (diminished radix compl.)

(r-1)'s complement of a number N is defined as $(r^n-1)-N$

Where **N** is the given number
r is the base of number system
n is the number of digits in the given number

To get the (r-1)'s complement fast, subtract each digit of a number from (r-1).

Example:

- 9's complement of 835_{10} is 164_{10} (Rule: $(10^n-1)-N$)
- 1's complement of 1010_2 is 0101_2 (bit by bit complement operation)

r's Complement (radix complement)

r's complement of a number N is defined as r^n-N

Where **N** is the given number
r is the base of number system
n is the number of digits in the given number

To get the r's complement fast, add 1 to the low-order digit of its (r-1)'s complement.

Example:

- 10's complement of 835_{10} is $164_{10} + 1 = 165_{10}$
- 2's complement of 10102 is $0101_2 + 1 = 0110_2$

Subtraction with complements

The direct method of subtraction taught in elementary schools uses the borrow concept. When subtraction is implemented with digital hardware, this method is found to be less efficient than the method that uses complements.

The subtraction of two n-digit unsigned numbers M - N in base-r can be done as follows:

1. Add the minuend M to the r's complement of the subtrahend N. This performsM + (r^n – N) = M -N + r".
2. If M >= N, the sum will produce an end carry, r", which is discarded; what is left isthe result M - N.
3. If M < N, the sum does not produce an end carry and is equal to rn - (N - M), which is the r's complement of (N - M). To obtain the answer in a familiar form, take the r's complement of the sum and place a negative sign in front.

Example I:

Using 10's complement, subtract 72532 − 3250.

M =	72532
10's complement of N =	+ 96750
Sum =	169282
Discard end carry 10^5 =	−100000
Answer =	69282

HEY! *M* has 5 digits and *N* has only 4 digits. Both numbers must have the same number of digits; so we can write *N* as 03250. Taking the 10's complement of *N* produces a 9 in the most significant position. The occurrence of the end carry signifies that *M* >= *N* and the result is positive.

BCA Guides Nepal

Example II:

Using 10's complement, subtract 3250 − 72532.

$$\begin{aligned} M &= \quad 03250 \\ \text{10's complement of } N &= +\ 27468 \\ \text{Sum} &= \quad 30718 \end{aligned}$$

There is no end carry.

Answer: −(10's complement of 30718) = −69282

Example III:

Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction (a) $X - Y$ and (b) $Y - X$ using 2's complements.

(a)

$$\begin{aligned} X &= \quad 1010100 \\ \text{2's complement of } Y &= +\ 0111101 \\ \text{Discard end carry } 2^7 &= -10000000 \\ \text{Answer: } X - Y &= \quad 0010001 \end{aligned}$$

(b)

$$\begin{aligned} Y &= \quad 1000011 \\ \text{2's complement of } X &= +\ 0101100 \\ \text{Sum} &= \quad 1101111 \end{aligned}$$

There is no end carry.

Answer: $Y - X$ = −(2's complement of 1101111) = −0010001

Example IV: Repeating Example III using 1's complement.

(a) $X - Y = 1010100 - 1000011$

$$\begin{aligned} X &= \quad 1010100 \\ \text{1's complement of } Y &= +\ 0111100 \\ \text{Sum} &= \quad 10010000 \\ \text{End-around carry} &\longrightarrow +\ 1 \\ \text{Answer: } X - Y &= \quad 0010001 \end{aligned}$$

(b) $Y - X = 1000011 - 1010100$

$$\begin{aligned} Y &= \quad 1000011 \\ \text{1's complement of } X &= +\ 0101011 \\ \text{Sum} &= \quad 1101110 \end{aligned}$$

There is no end carry.

Answer: $Y - X$ = −(1's complement of 1101110) = −0010001

Binary Codes

Electronic digital systems use signals that have two distinct values and circuit elements that have two stable states. There is a direct analogy among binary signals, binary circuit elements, and binary digits. A binary number of n digits, for example, may be represented by n binary circuit elements, each having an output signal equivalent to a 0 or a 1. Digital systems represent and manipulate not only binary numbers, but also many other discrete elements of information. Any discrete element of information distinct among a group of quantities can be represented by a binary code. Binary codes play an important role in digital computers. The codes must be in binary because computers can only hold 1's and 0's.

1. Binary Coded Decimal (BCD)

The binary number system is the most natural system for a computer, but people are accustomed to the decimal system. So, to resolve this difference, computer uses decimals in coded form which the hardware understands. A binary code that distinguishes among 10 elements of decimal digits must contain at least four bits. Numerous different binary codes can be obtained by arranging four bits into 10 distinct combinations. The code most commonly used for the decimal digits is the straightforward binary assignment listed in the table below. This is called binary-coded decimal and is commonly referred to as BCD.

Decimal Symbol	BCD Digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Table: 4-bit BCD code for decimal digits

- A number with *n* decimal digits will require 4n bits in BCD. E.g. decimal 396 is represented in BCD with 12 bits as 0011 1001 0110.
- Numbers greater than 9 has a representation different from its equivalent binary number, even though both contain 1's and 0's.
- Binary combinations 1010 through 1111 are not used and have no meaning in the BCD code.
- Example:

$(185)_{10} = (0001\ 1000\ 0101)_{BCD} = (10111001)_2$

2. Error-Detection codes

Binary information can be transmitted from one location to another by electric wires or other communication medium. Any external noise introduced into the physical communication medium may change some of the bits from 0 to 1 or vice versa.

The purpose of an error-detection code is to detect such bit-reversal errors. One of the most common ways to achieve error detection is by means of a parity bit. A parity bit is the extra bit included to make the total number of 1's in the resulting code word either even or odd. A message of 4-bits and a parity bit P are shown in the table below:

Odd parity		Even parity	
Message	P	Message	P
0000	1	0000	0
0001	0	0001	1
0010	0	0010	1
0011	1	0011	0
0100	0	0100	1
0101	1	0101	0
0110	1	0110	0
0111	0	0111	1
1000	0	1000	1
1001	1	1001	0
1010	1	1010	0
1011	0	1011	1
1100	1	1100	0
1101	0	1101	1
1110	0	1110	1
1111	1	1111	0

Error Checking Mechanism:

→During the transmission of information from one location to another, an even parity bit is generated in the sending end for each message transmission. The message, together with the parity bit, is transmitted to its destination. The parity of the received data is checked in the receiving end. If the parity of the received information is not even, it means that at least one bit has changed value during the transmission.

→This method detects one, three, or any odd combination of errors in each message that is transmitted. An even combination of errors is undetected. Additional error-detection schemes may be needed to take care of an even combination of errors.

▪ A number with n decimal digits will require 4n bits in BCD. E.g. decimal 396 is represented inBCD with 12 bits as 0011 1001 0110.

3. Gray code (Reflected code)

It is a binary coding scheme used to represent digits generated from a mechanical sensor that may be prone to error. Used in telegraphy in the late 1800s, and also known as "reflected binary code". Gray code was patented by Bell Labs researcher Frank Gray in 1947. In Gray code, there is only one bit location different between two successive values, which makes mechanical transitions from one digit to the next less error prone. The following chart shows normal binary Representations from 0 to 15 and the corresponding Gray code.

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

The Gray code is used in applications where the normal sequence of binary numbers may produce an error or ambiguity during the transition from one number to the next. If

binary numbers are used, a change from 0111 to 1000 may produce an intermediate erroneous number 1001 if the rightmost bit takes more time to change than the otherthree bits. The Gray code eliminates this problem since only one bit changes in value during any transition between two numbers.

2.1.2 Binary Arithmetic

Binary Arithmetic:

Binary arithmetic includes the basic arithmetic operations of addition, subtraction, multiplication and division. The following sections present the rules that apply to these operations when they are performed on binary numbers.

Binary Addition:

Binary addition is performed in the same way as addition in the decimal- system and is, in fact, much easier to master. Binary addition obeys the following four basic rules:

0	0	1	1
+0	+1	+0	+1
0	1	1	10

The results of the last rule may seem some what strange, remember that these are binary numbers. Put into words, the last rule states that

binary one + binary one = binary two = binary "one zero"

When adding more than single-digit binary number, carry into, higher order columns as is done when adding decimal numbers. For example 11 and 10 are added as follows:

```
  11
+ 10
 101
```

In the first column (L S C or 2^0) '1 plus 0 equal 1. In the second column (2^1)1 plus 1 equals 0 with a carry of 1 into the third column (2^2).

When we add 1 + 1.+ 1 (carry) produces 11, recorded as 1 with a carry to thenext column.

Example 12: Add (a) 111 and 101 (b) 1010, 1001 and 1101.
Solution:

(a)

```
  (1) (1)
  1 1 1
  1 0 1
-------
1 1 0 0
```

(B) (2)(1)(1)(1)

1010

1001

1101

10000

Binary Subtraction:

Binary subtraction is just as simple as addition subtraction of one bit fromanother obey the following four basic rules

0 – 0 = 0

1 – 1 =0

1 – 0 = 1

10 – 1 = 1 with a transfer (borrow) of 1.

When doing subtracting, it is sometimes necessary to borrow from the next higher-order column. The only it will be necessary to borrow is when we try to subtract a 1 from a 0. In this case a 1 is borrowed from the next higher-order column, which leaves a 0 in that column and creates a 10 i.e., 2 in the column being subtracted. The following examples illustrate binary subtraction.

Example 13: Perform the following subtractions.

(a) 11 - 01 , (b) 11-10 (c) 100 - 011

Solution:

	11			11		100	
	– 01			– 10		– 011	
(a)	10	(b		01	(c)	001	

Part (c) involves to borrows, which handled as follows. Since a 1 is to be subtracted from a 0 in the first column, a borrow is required from the next higher- order column. However, it also contains a 0; therefore, the second column must borrow the 1 in the third column. This leaves a 0 in the third column and place a 10in the second column. Borrowing a 1 from 10 leaves a 1 in the second column and places a 10 i.e, 2 in the first column:

When subtracting a larger number from a smaller number, the results will benegative. To perform this subtraction, one must subtract the smaller number from the larger and prefix the results with the sign of the larger number.

Example 14: Perform the following subtraction 101 – 111.

Solution:

Subtract the smaller number from the larger.

111

– 101

010

Thus 1 0 1 – 1 1 1 = - 010 = - 10

BCA Guides Nepal

Binary multiplication:

Binary multiplication is performed in the same manner as decimal multiplication. It is much easier, since there are only two possible results ofmultiplying two bits. The Binary multiplication obeys the four basic rules.

0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1

Example 15: Multiply the following binary numbers.
(a) 101 x **1 1**
(b) 1101 x 10
(c) 1010 x **l 0 1**
(d) 1 0 1 l x 1 0 1 0

Solution: (a)

```
  101
x 1 1
  101
 101
 1111
```

(b)

```
 11101
  x1 0
 0000
1101
11010
```

(c)

```
 1010
 x101
 1010
 0000
1010
110010
```

(d)

```
 1011
x101
0
  0000
 1011
 0000
1011
1101110
```

Multiplication of fractional number is performed in the same way as with fractional numbers in the decimal numbers.

Example 16: Perform the binary multiplication 0.01 x **11.**
Solution:

```
  0.01
 1 1 x
  01
 01x
 0.11
```

BCA Guides Nepal

Binary Division:

Division in the binary number system employees the same procedure asdivision in the decimal system, as will be seen in the following examples.

Example 17: Perform the following binary division.

(a) 110 ÷ 11

1100÷1 1

Solution:(a)

```
    10
 11 110
    11
    0 0
    0 0
    0 0
```

(b)

```
    100
 11 11000
    11
    0 0
    0 0
      0 0
      0 0
        0 0
```

Binary division problems with remainders are also treated the same as in thedecimal system, as illustrates the following example.

Example 18: Perform the following binary division:

(a) 1111 ÷ 110 (b) 1100 ÷ 101

Solution: (a)

```
     10. 1
110 1 111.00
    1 10
      1 10
      1 10
      0 0 0
```

(b)

```
     10. 011
110 100.00
    1 01
    100
    000
     1000
      101
       110
       101
         1
```

(reminder)

BCA Guides Nepal

Error Detection & Correction Codes

We know that the bits 0 and 1 corresponding to two different range of analog voltages. So, during transmission of binary data from one system to the other, the noise may also be added. Due to this, there may be errors in the received data at other system.

That means a bit 0 may change to 1 or a bit 1 may change to 0. We can't avoid the interference of noise. But, we can get back the original data first by detecting whether any errorss present and then correcting those errors. For this purpose, we can use the following codes.

- Error detection codes
- Error correction codes

Error detection codes − are used to detect the errorss present in the received data bitstreambitstream. These codes contain some bitss, which are included appendedappended to the original bit stream. These codes detect the error, if it is occurred during transmission of the original data bitstreambitstream.**Example** − Parity code, Hamming code.

Error correction codes − are used to correct the errorss present in the received data bitstreambitstream so that, we will get the original data. Error correction codes also use the similar strategy of error detection codes.**Example** − Hamming code.

Therefore, to detect and correct the errors, additional bitss are appended to the data bits at the time of transmission.

Parity Code

It is easy to include appendappend one parity bit either to the left of MSB or to the right of LSB of original bit stream. There are two types of parity codes, namely even parity code and odd parity code based on the type of parity being chosen.

Even Parity Code

The value of even parity bit should be zero, if even number of ones present in the binary code. Otherwise, it should be one. So that, even number of ones present in **even parity code**. Even parity code contains the data bits and even parity bit.

The following table shows the **even parity codes** corresponding to each 3-bit binary code. Here, the even parity bit is included to the right of LSB of binary code.

Binary Code	Even Parity bit	Even Parity Code
000	0	0000

001	1	0011
010	1	0101
011	0	0110
100	1	1001
101	0	1010
110	0	1100
111	1	1111

Here, the number of bits present in the even parity codes is 4. So, the possible even number of ones in these even parity codes are 0, 2 & 4.

- If the other system receives one of these even parity codes, then there is no error in the received data. The bits other than even parity bit are same as that of binary code.
- If the other system receives other than even parity codes, then there will be an errorss in the received data. In this case, we can't predict the original binary code because we don't know the bit positionss of error.

Therefore, even parity bit is useful only for detection of error in the received parity code. But, it is not sufficient to correct the error.

Odd Parity Code

The value of odd parity bit should be zero, if odd number of ones present in the binary code. Otherwise, it should be one. So that, odd number of ones present in **odd parity code**. Odd parity code contains the data bits and odd parity bit.

The following table shows the **odd parity codes** corresponding to each 3-bit binary code. Here, the odd parity bit is included to the right of LSB of binary code.

Binary Code	Odd Parity bit	Odd Parity Code

BCA Guides Nepal

000	1	0001
001	0	0010
010	0	0100
011	1	0111
100	0	1000
101	1	1011
110	1	1101
111	0	1110

Here, the number of bits present in the odd parity codes is 4. So, the possible odd number of ones in these odd parity codes are 1 & 3.

- If the other system receives one of these odd parity codes, then there is no error in the received data. The bits other than odd parity bit are same as that of binary code.
- If the other system receives other than odd parity codes, then there is an error*ss* in the received data. In this case, we can't predict the original binary code because we don't know the bit position*ss* of error.

Therefore, odd parity bit is useful only for detection of error in the received parity code. But, it is not sufficient to correct the error.

BCA Guides Nepal

What is Excess-3 Code?

The excess-3 code (or XS3) is a non-weighted code used to express code used to express decimal numbers. It is a self-complementary binary coded decimal (BCD) code and numerical system which has biased representation. It is particularly significant for arithmetic operations as it overcomes shortcoming encountered while using 8421 BCD code to add two decimal digits whose sum exceeds 9. Excess-3 arithmetic uses different algorithm than normal non-biased BCD or binary positional number system.

Representation of Excess-3 Code

Excess-3 codes are unweighted and can be obtained by adding 3 to each decimal digit then it can be represented by using 4 bit binary number for each digit. An Excess-3 equivalent of a given binary binary number is obtained using the following steps:

- Find the decimal equivalent of the given binary number.
- Add +3 to each digit of decimal number.
- Convert the newly obtained decimal number back to binary number to get required excess-3 equivalent.

You can add 0011 to each four-bit group in binary coded decimal number (BCD) to get desired excess-3 equivalent.

These are following excess-3 codes for decimal digits −

Decimal Digit	BCD Code	Excess-3 Code
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111

BCA Guides Nepal

Decimal Digit	BCD Code	Excess-3 Code
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

The codes 0000 and 1111 are not used for any digit.

Example-1 −Convert decimal number 23 to Excess-3 code.

So, according to excess-3 code we need to add 3 to both digit in the decimal number then convert into 4-bit binary number for result of each digit. Therefore,

= 23+33=56 =0101 0110 which is required excess-3 code for given decimal number 23.

ASCII

ASCII stands for the "American Standard Code for Information Interchange".

It was designed in the early 60's, as a standard character set for computers and electronic devices.

ASCII is a 7-bit character set containing 128 characters.

It contains the numbers from 0-9, the upper and lower case English letters from A to Z, and some special characters.

The character sets used in modern computers, in HTML, and on the Internet, are all based on ASCII.

BCA Guides Nepal

Unit 3 Combinational Logic Design

3.1 Basic Logic Gates NOT, OR and AND 3.2 Universal Logic Gates NOR and NAND 3.3 EX-OR and EX-NOR Gates 3.4 Boolean Algebra: 3.3.1 Postulates & Theorems 3.3.2 Canonical Forms - Simplification of Logic Functions 3.5 Simplification of Logic Functions Using Karnaugh Map. 3.5.1 Analysis of SOP And POS 'ExRession 3.6 Implementation of Combinational Logic Functions 3.6.1 Encoders & Decoders 16 Hrs. 3.6,2 Half Adder, & Full Adder 3.7 Implementation of Data Processing Circuits 3.7.1 Multiplexers and De-Multiplexers 3.7.2 Parallel Adder -Binary Adder-Parity Generator /Checker-Implementation of Logical Functions Using Multiplexers. 3.8 Basic Concepts of Programmable Logic 3.8.1 PROM 3.8.2 EPROM 3.8.3 PAL 3.8.4 PLA

3.1 Basic Logic Gates NOT, OR and AND

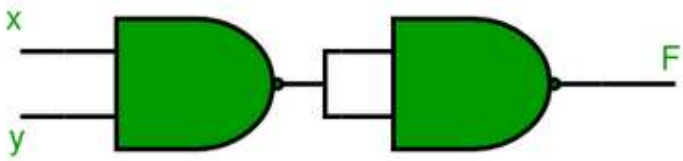
3.2 Universal Logic Gates NOR and NAND 3.3 EX-OR and EX-NOR Gates

In Boolean Algebra, the **NAND** and **NOR** gates are called **universal gates** because any digital circuit can be implemented by using any one of these two i.e. any logic gate can be created using NAND or NOR gates only.

1. Implementation of AND Gate using Universal gates.

a) Using NAND Gates

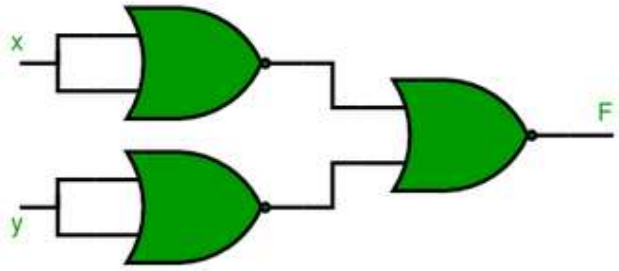
The AND gate can be implemented by using two NAND gates in the below fashion:



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

b) Using NOR Gates

Implementation of AND gate using only NOR gates as shown below:

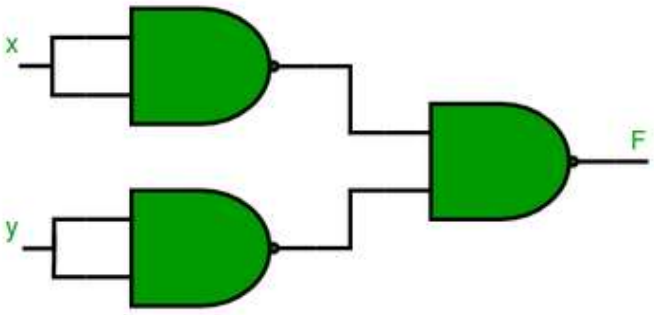


x	y	F
0	0	0
1	0	0
1	1	1
0	1	0

2. Implementation of OR Gate using Universal gates.

a) Using NAND Gates

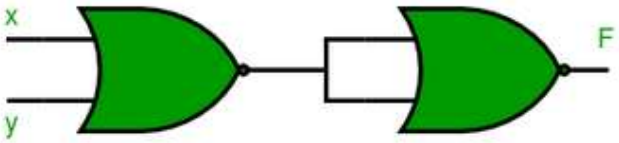
The OR gate can be implemented using the NAND gate as below:



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

b) Using NOR Gates

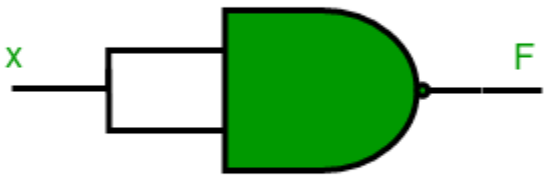
Implementation of OR gate using two NOR gates as shown in the picture below:



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

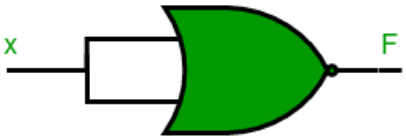
3. Implementation of NOT Gate using Universal gates.

a) Using NAND Gates



x	F
0	1
1	0

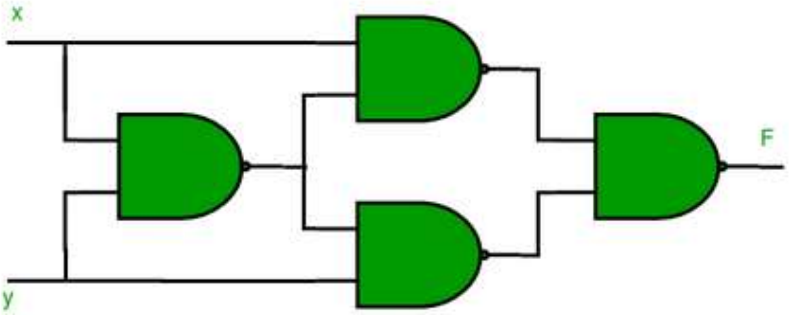
b) Using NOR Gates



x	F
0	1
1	0

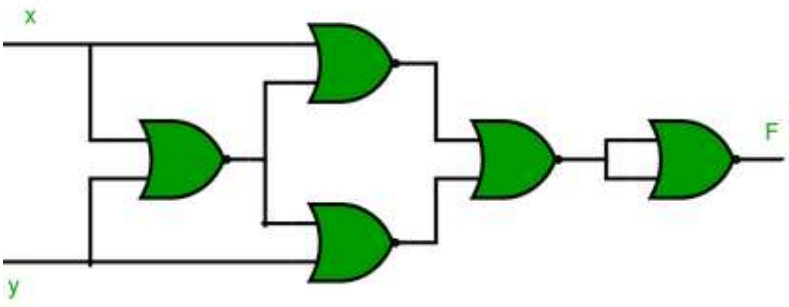
4. Implementation of XOR Gate using Universal gates.

a) Using NAND Gates



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

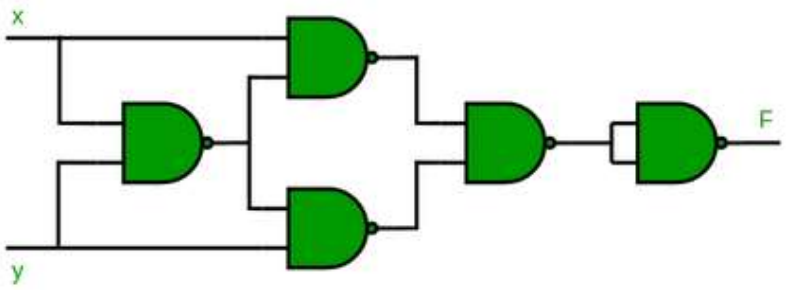
b) Using NOR Gates



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

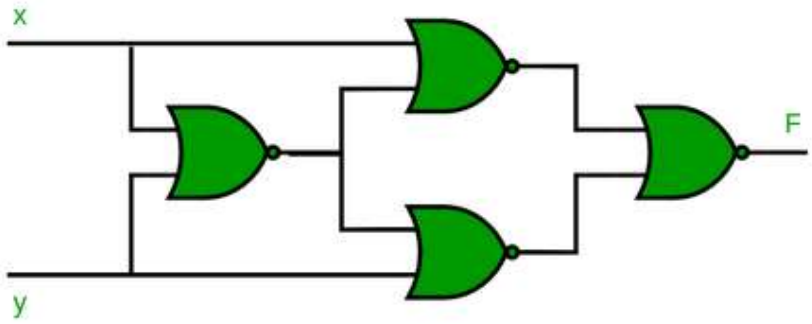
5. Implementation of XNOR Gate using Universal gates.

a) Using NAND Gate



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

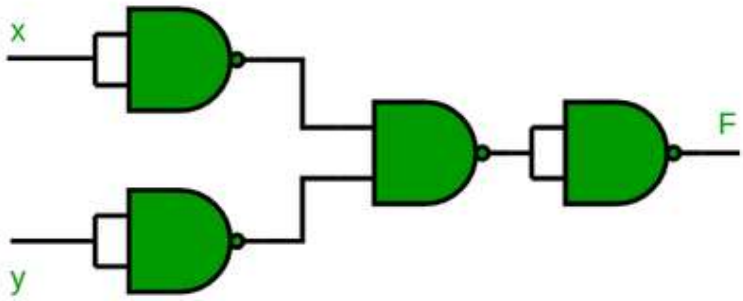
b) Using NOR Gate



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

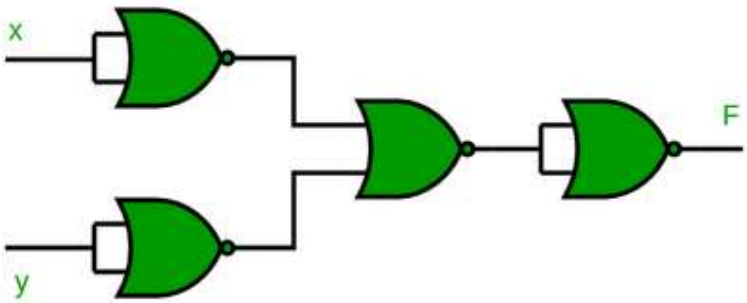
BCA

6. Implementation of NOR Gate using NAND Gates



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

7. Implementation of NAND Gate using NOR Gates



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

3.4 Boolean Algebra

Switching algebra is also known as **Boolean Algebra**. It is used to analyze digital gates and circuits It is logic to perform mathematical operation on binary numbers i.e., on '0' and '1'. Boolean Algebra contains basic operators like AND, OR and NOT etc. Operations are represented by '.' for AND , '+' for OR . Operations can be performed on variables which are represented using capital letter eg 'A' , 'B' etc.

Properties of switching algebra –

- **Annulment law –** a variable ANDed with 0 gives 0, while a variable ORed with 1 gives 1, i.e.,

A.0 = 0
A + 1 = 1

- **Identity law –** in this law variable remain unchanged it is ORed with '0' or ANDed with '1', i.e.,
  A.1 = A
  A + 0 = A
- **Idempotent law –** a variable remain unchanged when it is ORed or ANDed with itself, i.e.,
  A + A = A
  A.A = A
- **Complement law –** in this Law if a complement is added to a variable it gives one, if a variable is multiplied with its complement it results in '0', i.e.,
  A + A' = 1
  A.A' = 0
- **Double negation law –** a variable with two negation its symbol gets cancelled out and original variable is obtained, i.e.,
  ((A)')'=A
- **Commutative law –** a variable order does not matter in this law, i.e.,
  A + B = B + A
  A.B = B.A
- **Associative law –** the order of operation does not matter if the priority of variables are same like '*' and '/', i.e.,
  A+(B+C) = (A+B)+C
  A.(B.C) = (A.B).C
- **Distributive law –** this law governs opening up of brackets, i.e.,
  A.(B+C) = (A.B)+(A.C)
  A+(B.C) = (A+B).(A+C)
- **Absorption law –**:-This law involved absorbing the similar variables, i.e.,
  A.(A+B) = A
  A + AB = A
- **De Morgan law –** the operation of an AND or OR logic circuit is unchanged if all inputs are inverted, the operator is changed from AND to OR, the output is inverted, i.e.,
  (A.B)' = A' + B'
  (A+B)' = A'.B'

3.3.1 Postulates & Theorems

Postulates of Boolean Algebra

S.No.	Name of the Postulates	Postulate Equation
1	Law of Identity	A + 0 = 0 + A = A A . 1 = 1 . A = A
2	Commutative Law	(A + B) = (B + A) (A . B) = (B . A)
3	Distributive Law	A . (B + C) = (A . B) + (B . C) A + (B . C) = (A + B) . (B + C)
4	Associative Law	A + (B + C) = (A + B) + C (A . B) . C = A . (B . C)
5	Complement Law	A + A' = 1 A . A' = 0

Theorems of Boolean Algebra

S.No	Theorem	Statement	Equations
1	Duality Theorem	A boolean relation can be derived from another boolean relation by changing OR sign to AND sign and vice versa and complementing the 0s and 1s.	A + A' = 1 and A . A' = 0 are the dual relations.
2	DeMorgan's Theorem 1	Complement of a product is equal to the sum of its complement.	(A . B)' = A' + B'
3	DeMorgan's Theorem 2	Complement of a sum is equal to the product of the complement.	(A + B)' = A' . B'
4	Idempotency Theorem	–	A + A = A A . A = A

BCA Guides Nepal

S.No	Theorem	Statement	Equations
5	Involution Theorem	–	A” = A
6	Absorption Theorem	–	A + (A . B) = A A . (A + B) = A
7	Associative Theorem	–	A + (B + C) = (A + B) + C A . (B . C) = (A . B) . C
8	Consensus Theorem	–	AB + A’C + BC = AB + A’C (A + B) + (A’ + C) + (B + C) = (A + B) + (A’ + C)

3.3.2 Canonical Forms - Simplification of Logic Functions

Canonical Form – In Boolean algebra,Boolean function can be expressed as Canonical Disjunctive Normal Form known as **minterm** and some are expressed as Canonical Conjunctive Normal Form known as **maxterm** .
In Minterm, we look for the functions where the output results in “1” while in Maxterm we look for function where the output results in “0”.
We perform **Sum of minterm** also known as Sum of products (SOP) .
We perform **Product of Maxterm** also known as Product of sum (POS).
Boolean functions expressed as a sum of minterms or product of maxterms are said to be in canonical form.
Standard Form – A Boolean variable can be expressed in either true form or complemented form. In standard form Boolean function will contain all the variables in either true form or complemented form while in canonical number of variables depends on the output of SOP or POS.
A Boolean function can be expressed algebraically from a given truth table by forming a :

- minterm for each combination of the variables that produces a 1 in the function and then taking the OR of all those terms.
- maxterm for each combination of the variables that produces a 0 in the function and then taking the AND of all those terms.

Truth table representing minterm and maxterm –

				Minterms		*Maxterms*
X	Y	Z		*Product Terms*		*Sum Terms*
0	0	0		$m_0 = \overline{X} \cdot \overline{Y} \cdot \overline{Z} = \min(\overline{X}, \overline{Y}, \overline{Z})$		$M_0 = X + Y + Z = \max(X, Y, Z)$
0	0	1		$m_1 = \overline{X} \cdot \overline{Y} \cdot Z = \min(\overline{X}, \overline{Y}, Z)$		$M_1 = X + Y + \overline{Z} = \max(X, Y, \overline{Z})$
0	1	0		$m_2 = \overline{X} \cdot Y \cdot \overline{Z} = \min(\overline{X}, Y, \overline{Z})$		$M_2 = X + \overline{Y} + Z = \max(X, \overline{Y}, Z)$
0	1	1		$m_3 = \overline{X} \cdot Y \cdot Z = \min(\overline{X}, Y, Z)$		$M_3 = X + \overline{Y} + \overline{Z} = \max(X, \overline{Y}, \overline{Z})$
1	0	0		$m_4 = X \cdot \overline{Y} \cdot \overline{Z} = \min(X, \overline{Y}, \overline{Z})$		$M_4 = \overline{X} + Y + Z = \max(\overline{X}, Y, Z)$
1	0	1		$m_5 = X \cdot \overline{Y} \cdot Z = \min(X, \overline{Y}, Z)$		$M_5 = \overline{X} + Y + \overline{Z} = \max(\overline{X}, Y, \overline{Z})$
1	1	0		$m_6 = X \cdot Y \cdot \overline{Z} = \min(X, Y, \overline{Z})$		$M_6 = \overline{X} + \overline{Y} + Z = \max(\overline{X}, \overline{Y}, Z)$
1	1	1		$m_7 = X \cdot Y \cdot Z = \min(X, Y, Z)$		$M_7 = \overline{X} + \overline{Y} + \overline{Z} = \max(\overline{X}, \overline{Y}, \overline{Z})$

From the above table it is clear that minterm is expressed in product format and maxterm is expressed in sum format.

Sum of minterms –
The minterms whose sum defines the Boolean function are those which give the 1's of the function in a truth table. Since the function can be either 1 or 0 for each minterm, and since there are 2^n minterms, one can calculate all the functions that can be formed with n variables to be (2^(2^n)). It is sometimes convenient to express a Boolean function in its sum of minterm form.

- **Example –** Express the Boolean function F = A + B'C as standard sum of minterms.
- **Solution –**
  A = A(B + B') = AB + AB'
  This function is still missing one variable, so
  A = AB(C + C') + AB'(C + C') = ABC + ABC'+ AB'C + AB'C'
  The second term B'C is missing one variable; hence,
  B'C = B'C(A + A') = AB'C + A'B'C
  Combining all terms, we have
  F = A + B'C = ABC + ABC' + AB'C + AB'C' + AB'C + A'B'C
  But AB'C appears twice, and
  according to theorem 1 (x + x = x), it is possible to remove one of those occurrences. Rearranging the minterms in ascending order, we finally obtain

F = A’B’C + AB’C’ + AB’C + ABC’ + ABC
= m1 + m4 + m5 + m6 + m7
SOP is represented as Sigma(1, 4, 5, 6, 7)

- **Example –** Express the Boolean function F = xy + x’z as a product of maxterms
- **Solution –**
  F = xy + x’z
  = (xy + x’)(xy + z)
  = (x + x’)(y + x’)(x + z)(y + z)
  = (x’ + y)(x + z)(y + z)
  x’ + y = x’ + y + zz’
  = (x’+ y + z)(x’ + y + z’) x + z
  = x + z + yy’
  = (x + y + z)(x + y’ + z) y + z
  = y + z + xx’
  = (x + y + z)(x’ + y + z)
  F = (x + y + z)(x + y’ + z)(x’ + y + z)(x’ + y + z’)
  = M0*M2*M4*M5
  POS is represented as Pi(0, 2, 4, 5)
    - **Example –**
      F(A, B, C) = Sigma(1, 4, 5, 6, 7)
      F'(A, B, C) = Sigma(0, 2, 3) = m0 + m2 + m3
      Now, if we take the complement of F’ by DeMorgan’s theorem, we obtain F in a different form:
      F = (m0 + m2 + m3)’
      = m0’m2’m3′
      = M0*M2*M3
      = PI(0, 2, 3)

- **Example –** Convert Boolean expression in standard form F=y’+xz’+xyz
- **Solution –** F = (x+x’)y'(z+z’)+x(y+y’)z’ +xyz
  F = xy’z+ xy’z’+x’y’z+x’y’z’+ xyz’+xy’z’+xyz

5 Simplification of Logic Functions Using Karnaugh Map

In many digital circuits and practical problems we need to find expression with minimum variables. We can minimize Boolean expressions of 3, 4 variables very easily using K-map without using any Boolean algebra theorems. K-map can take two forms Sum of Product (SOP) and Product of Sum (POS) according to the need of problem. K-map is table like representation but it gives more

information than TRUTH TABLE. We fill grid of K-map with 0's and 1's then solve it by making groups.

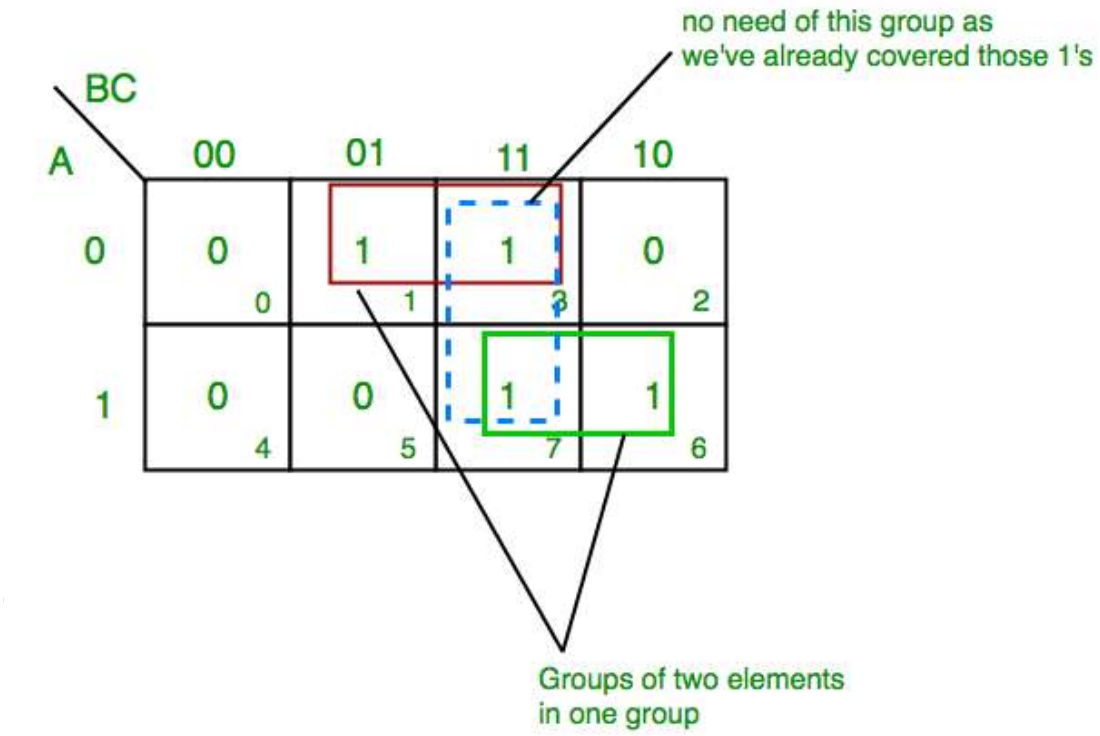
Steps to solve expression using K-map-

1. Select K-map according to the number of variables.
2. Identify minterms or maxterms as given in problem.
3. For SOP put 1's in blocks of K-map respective to the minterms (0's elsewhere).
4. For POS put 0's in blocks of K-map respective to the maxterms(1's elsewhere).
5. Make rectangular groups containing total terms in power of two like 2,4,8 ..(except 1) and try to cover as many elements as you can in one group.
6. From the groups made in step 5 find the product terms and sum them up for SOP form.

SOP FORM :

1. K-map of 3 variables –

Z= ∑A,B,C(1,3,6,7)



From **red** group we get product term—

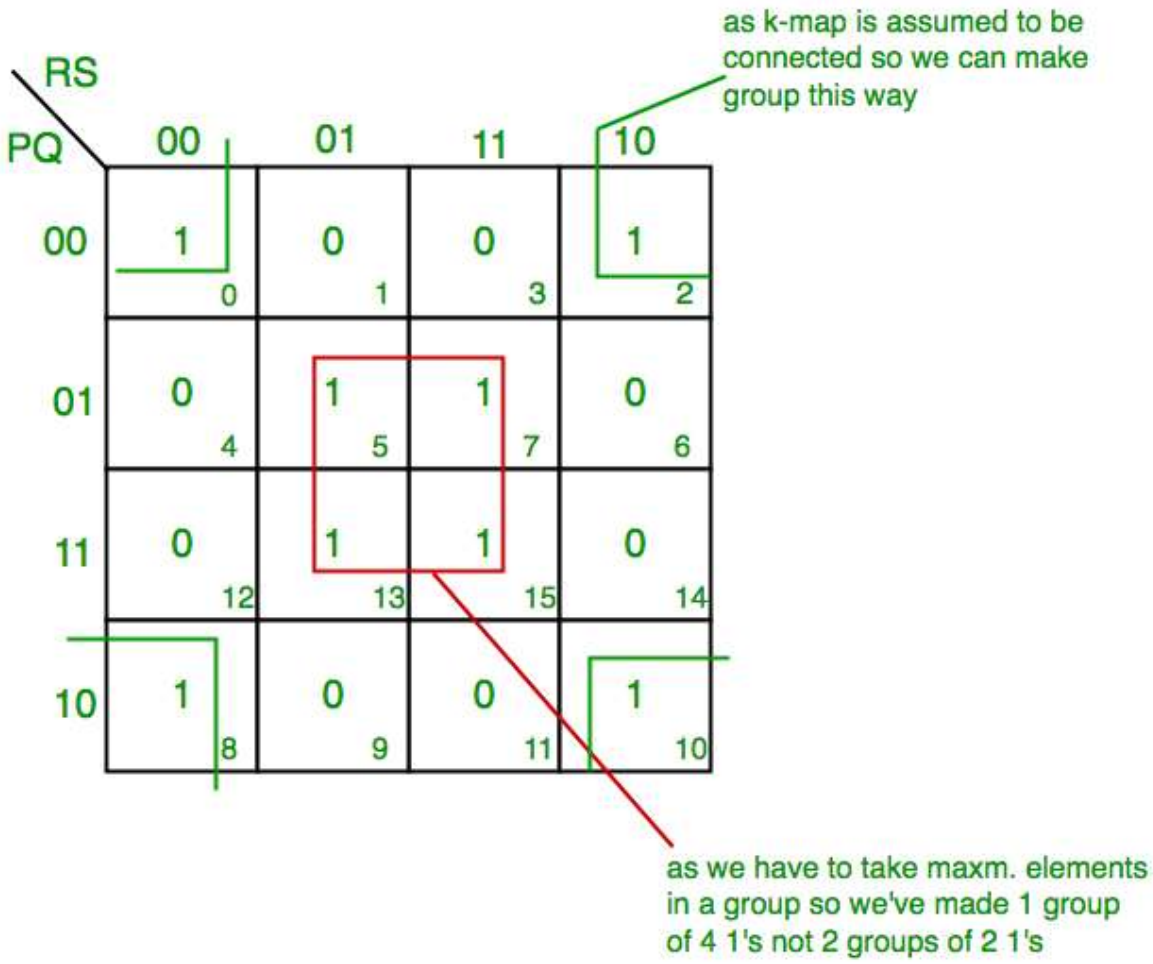
A'C

From **green** group we get product term—
AB

Summing these product terms we get- **Final expression (A'C+AB)**

2. K-map for 4 variables –

F(P,Q,R,S)=∑(0,2,5,7,8,10,13,15)



From **red** group we get product term—
QS

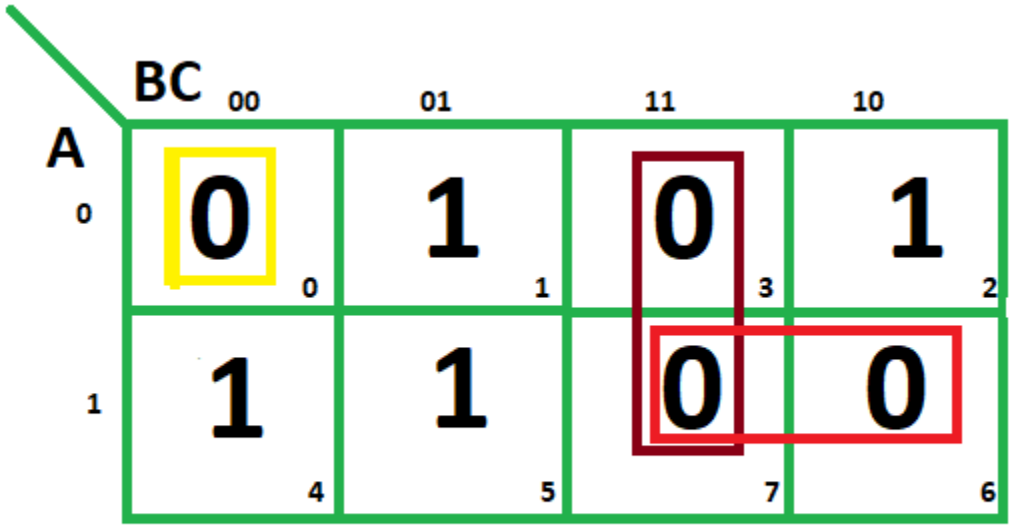
From **green** group we get product term—
Q'S'

Summing these product terms we get- **Final expression (QS+Q'S')**

POS FORM :

1. K-map of 3 variables –

F(A,B,C)=π(0,3,6,7)



From **red** group we find terms

A B

Taking complement of these two

A' B'

Now **sum** up them

(A' + B')

From **brown** group we find terms

B C

Taking complement of these two terms

B’ C’

Now sum up them

(B’+C’)

From **yellow** group we find terms

A' B' C’

Taking complement of these two

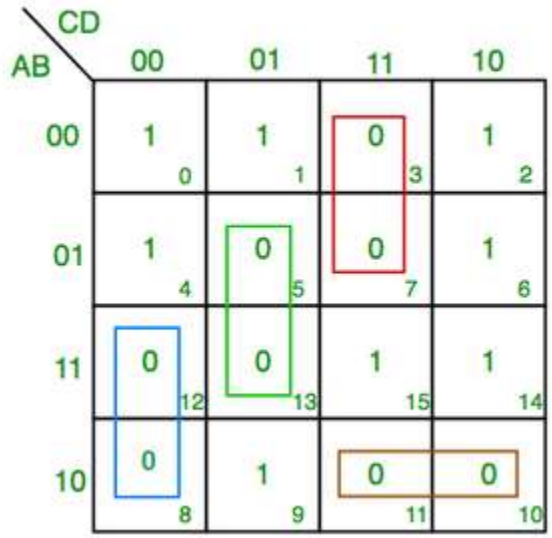
A B C

Now **sum** up them
(A + B + C)

We will take product of these three terms : **Final expression –**
(A' + B') (B' + C') (A + B + C)

2. K-map of 4 variables –

F(A,B,C,D)=π(3,5,7,8,10,11,12,13)



From **green** group we find terms
C' D B

Taking their complement and summing them

(C+D'+B')

From **red** group we find terms
C D A'

Taking their complement and summing them

(C'+D'+A)

From **blue** group we find terms
A C' D'

Taking their complement and summing them

(A'+C+D)

From **brown** group we find terms
A B’ C

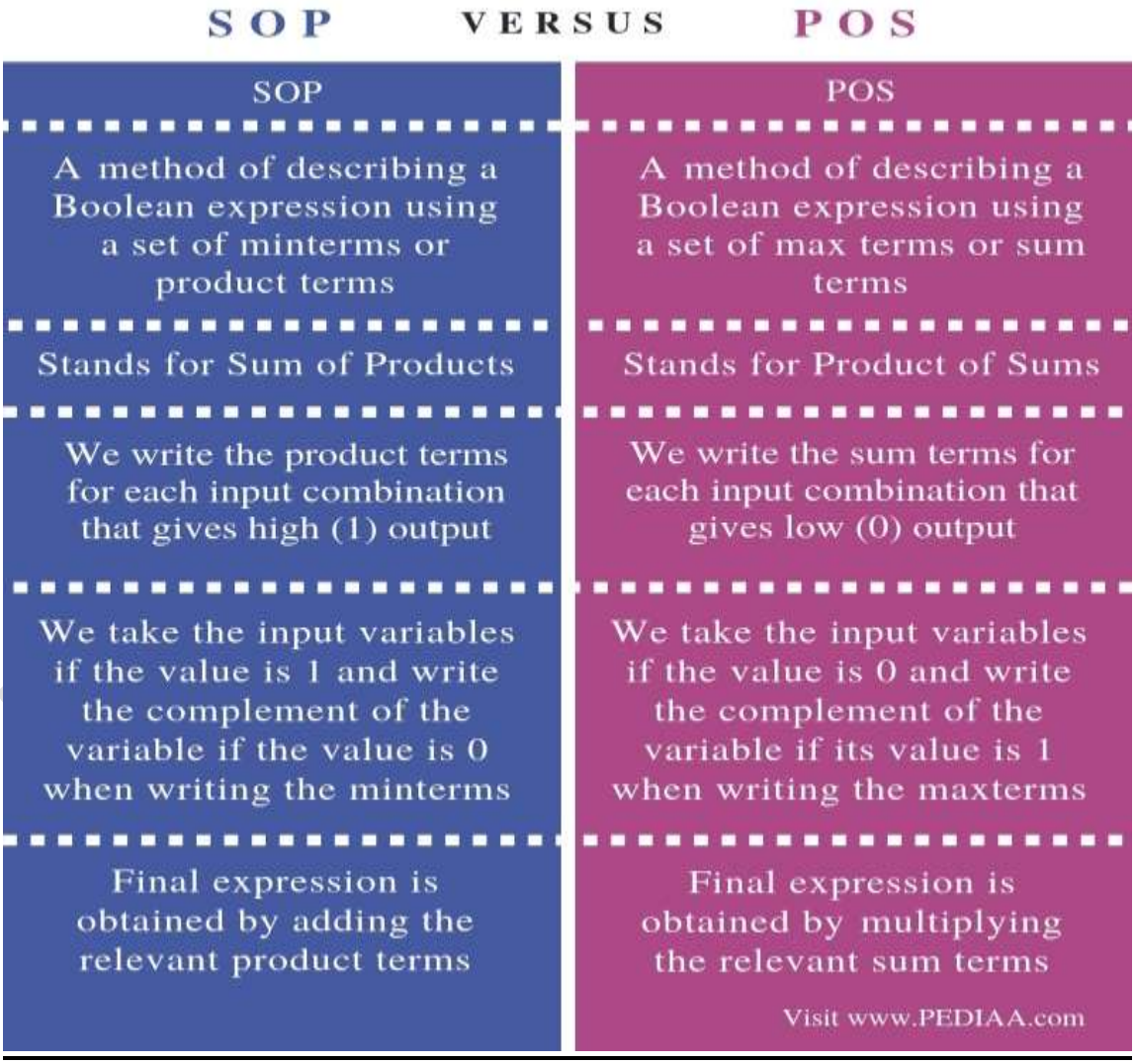
Taking their complement and summing them

(A’+B+C’)

Finally we express these as product –

(C+D’+B’).(C’+D’+A).(A’+C+D).(A’+B+C’)
PITFALL– *Always remember ***POS ≠ (SOP)’***
*The correct form is (**POS of F)=(SOP of F’)’**

3.5.1 Analysis of SOP And POS 'Expression



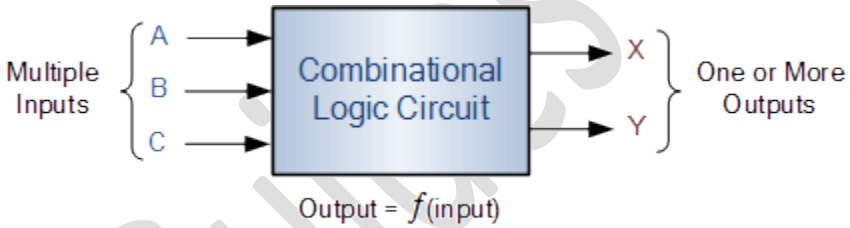
3.6 Implementation of Combinational Logic Functions

Unlike Sequential Logic Circuits whose outputs are dependant on both their present inputs and their previous output state giving them some form of *Memory*. The outputs of **Combinational Logic Circuits** are only determined by the logical function of their current input state, logic "0" or logic "1", at any given instant in time.

The result is that combinational logic circuits have no feedback, and any changes to the signals being applied to their inputs will immediately have an effect at the output. In other words, in a **Combinational Logic Circuit**, the output is dependant at all times on the combination of its inputs. Thus a combinational circuit is *memoryless*.

So if one of its inputs condition changes state, from 0-1 or 1-0, so too will the resulting output as by default combinational logic circuits have "no memory", "timing" or "feedback loops" within their design.

Combinational Logic



Combinational Logic Circuits are made up from basic logic NAND, NOR or NOT gates that are "combined" or connected together to produce more complicated switching circuits. These logic gates are the building blocks of combinational logic circuits. An example of a combinational circuit is a decoder, which converts the binary code data present at its input into a number of different output lines, one at a time producing an equivalent decimal code at its output.

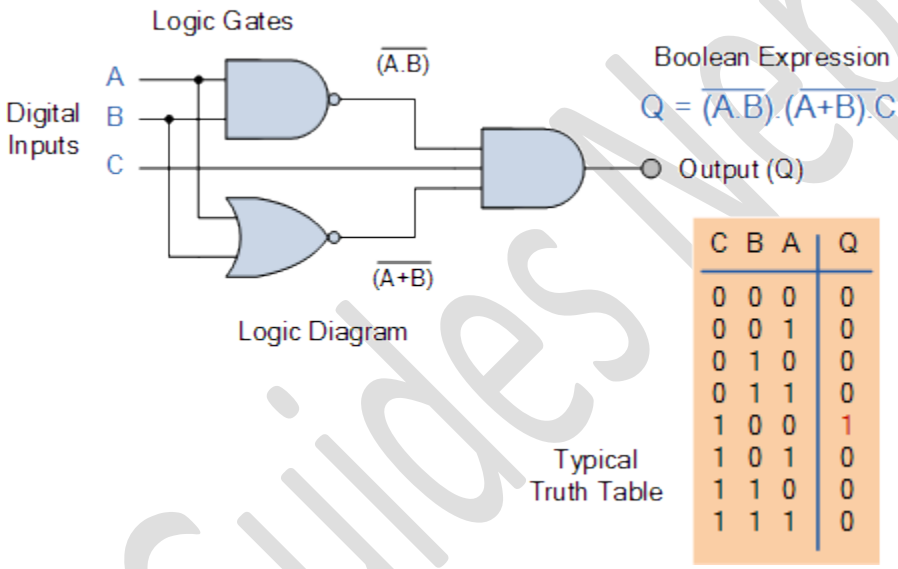
Combinational logic circuits can be very simple or very complicated and any combinational circuit can be implemented with only NAND and NOR gates as these are classed as "universal" gates.

The three main ways of specifying the function of a combinational logic circuit are:

BCA Guides Nepal

- 1. Boolean Algebra – This forms the algebraic expression showing the operation of the logic circuit for each input variable either True or False that results in a logic “1” output.
- 2. Truth Table – A truth table defines the function of a logic gate by providing a concise list that shows all the output states in tabular form for each possible combination of input variable that the gate could encounter.
- 3. Logic Diagram – This is a graphical representation of a logic circuit that shows the wiring and connections of each individual logic gate, represented by a specific graphical symbol, that implements the logic circuit.

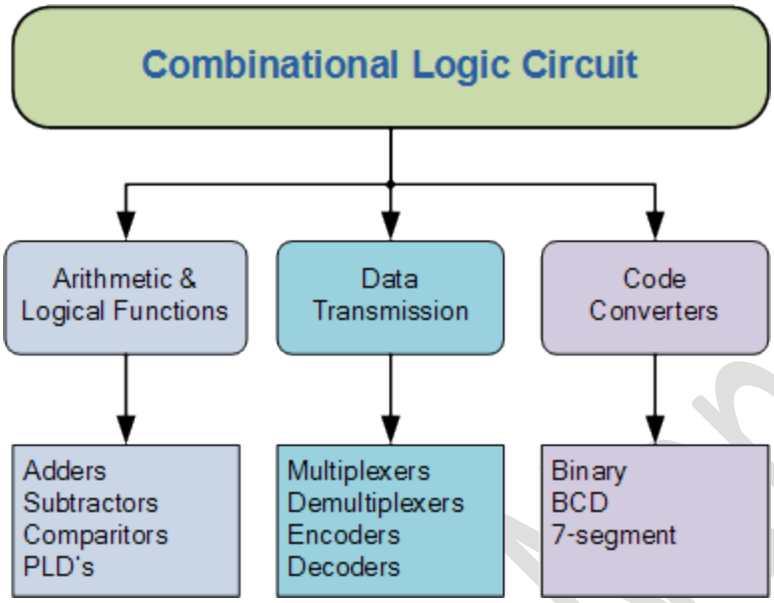
and all three of these logic circuit representations are shown below.



As combinational logic circuits are made up from individual logic gates only, they can also be considered as “decision making circuits” and combinational logic is about combining logic gates together to process two or more signals in order to produce at least one output signal according to the logical function of each logic gate. Common combinational circuits made up from individual logic gates that carry out a desired application include *Multiplexers*, *De-multiplexers*, *Encoders*, *Decoders*, *Full* and *Half Adders* etc.

BCA Guides Nepal

Classification of Combinational Logic



3.7 Implementation of Data Processing Circuits

What are data processing circuits?

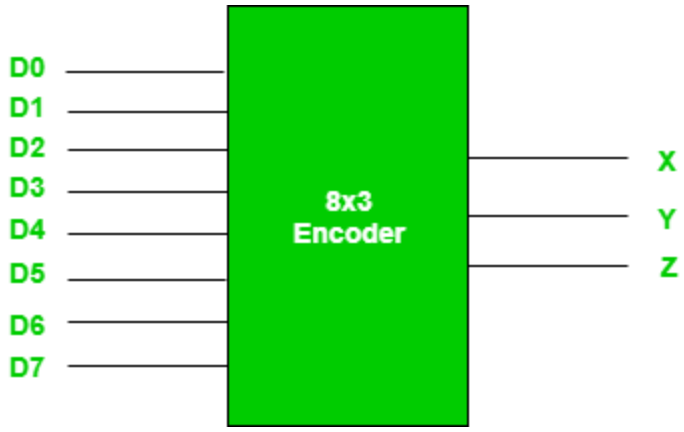
Data-processing circuits are **logic circuits that process binary data**. Such circuits may be multiplexers, demultiplexers, encoder, decoder, EX-OR gates. Implementations are below:

3.6.1 Encoders & Decoders

1. Encoders –

An encoder is a combinational circuit that converts binary information in the form of a 2^N input lines into N output lines, which represent N bit code for the input. For simple encoders, it is assumed that only one input line is active at a time.

As an example, let's consider **Octal to Binary** encoder. As shown in the following figure, an octal-to-binary encoder takes 8 input lines and generates 3 output lines.



Truth Table –

D7	D6	D5	D4	D3	D2	D1	D0	X	Y	Z
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

As seen from the truth table, the output is 000 when D0 is active; 001 when D1 is active; 010 when D2 is active and so on.

Implementation –

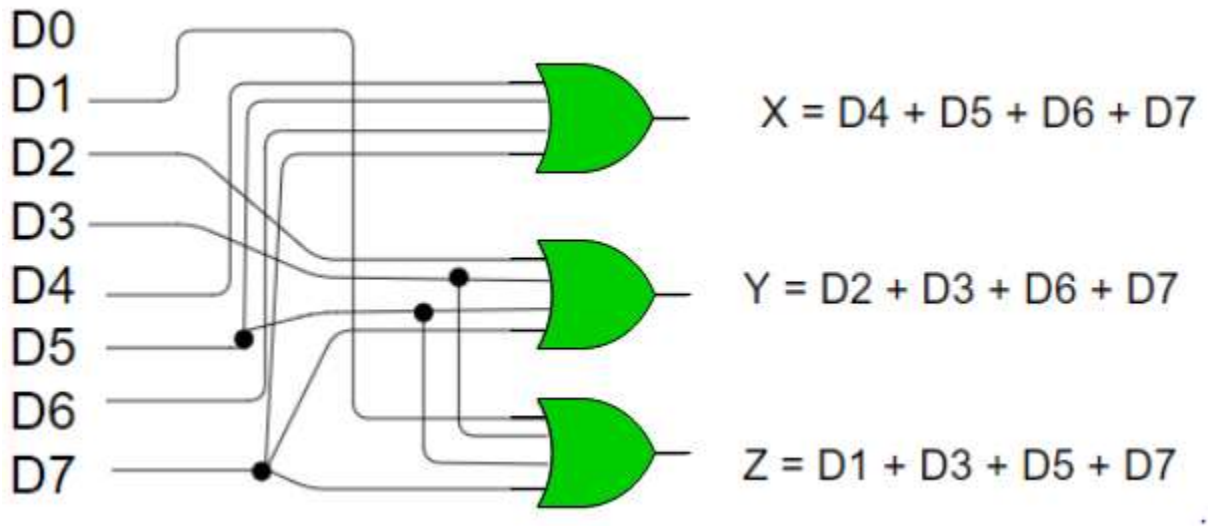
From the truth table, the output line Z is active when the input octal digit is 1, 3, 5 or 7. Similarly, Y is 1 when input octal digit is 2, 3, 6 or 7 and X is 1 for input octal digits 4, 5, 6 or 7. Hence, the Boolean functions would be:

```
X = D4 + D5 + D6 + D7
```

Y = D2 +D3 + D6 + D7

Z = D1 + D3 + D5 + D7

Hence, the encoder can be realised with OR gates as follows:



One limitation of this encoder is that only one input can be active at any given time. If more than one inputs are active, then the output is undefined. For example, if D6 and D3 are both active, then, our output would be 111 which is the output for D7. To overcome this, we use Priority Encoders.

Another ambiguity arises when all inputs are 0. In this case, encoder outputs 000 which actually is the output for D0 active. In order to avoid this, an extra bit can be added to the output, called the valid bit which is 0 when all inputs are 0 and 1 otherwise.

Priority Encoder –
A priority encoder is an encoder circuit in which inputs are given priorities. When more than one inputs are active at the same time, the input with higher priority takes precedence and the output corresponding to that is generated.

Let us consider the 4 to 2 priority encoder as an example.
From the truth table, we see that when all inputs are 0, our V bit or the valid bit is zero and outputs are not used. The x's in the table show the don't care condition, i.e, it may either be 0 or 1. Here, D3 has highest priority, therefore, whatever be the other inputs, when D3 is high, output has to be 11. And D0 has the lowest priority, therefore the output would be 00 only when D0 is high and the other input lines are low. Similarly, D2 has higher priority over D1 and D0 but lower than D3 therefore the output would be 010 only when D2 is high and D3 are low (D0 & D1 are don't care).

Truth Table –

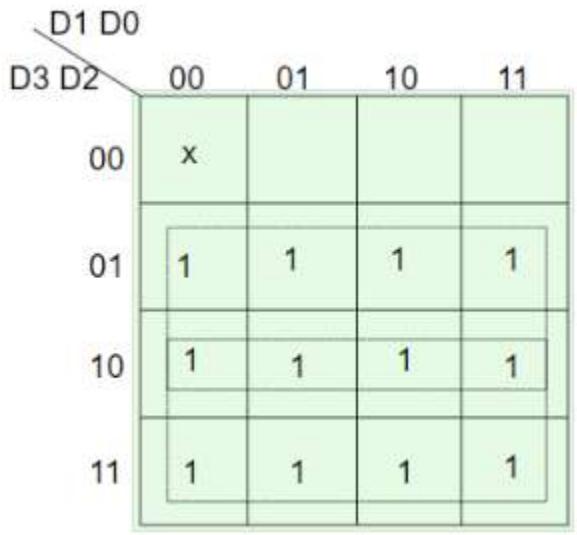
D3	D2	D1	D0	X	Y	V
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

Implementation –

It can clearly be seen that the condition for valid bit to be 1 is that at least any one of the inputs should be high. Hence,

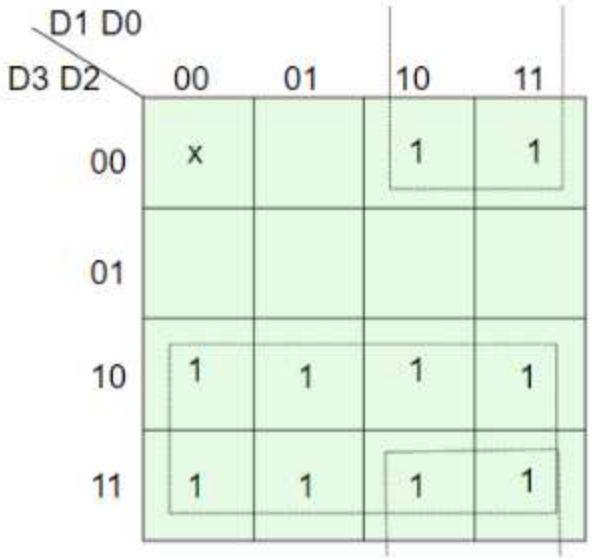
```
V = D0 + D1 + D2 + D3
```

For X:



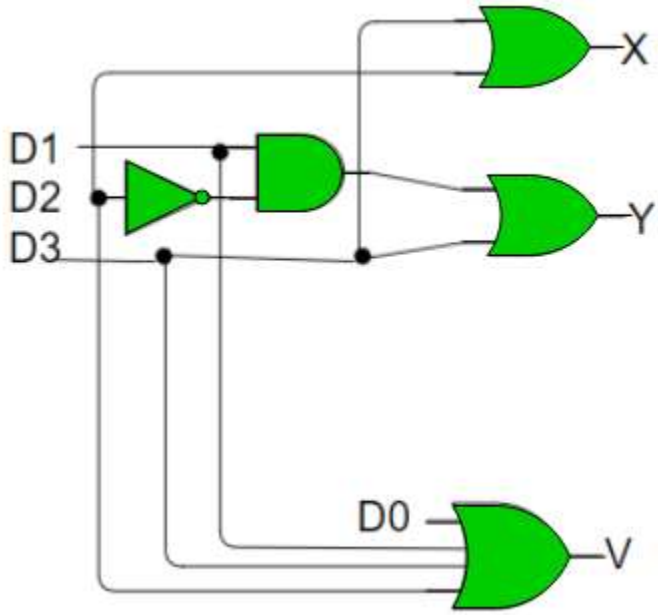
=> X = D2 + D3

For Y:



=> Y = D1 D2' + D3

Hence, the priority 4-to-2 encoder can be realized as follows:



2. Decoders –

A decoder does the opposite job of an encoder. It is a combinational circuit that converts n lines of input into 2^n lines of output.

Let's take an example of 3-to-8 line decoder.

Truth Table –

X	Y	Z	D0	D1	D2	D3	D4	D5	D6	D7
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

Implementation –

D0 is high when X = 0, Y = 0 and Z = 0. Hence,

D0 = X’ Y’ Z’

Similarly,

D1 = X’ Y’ Z

D2 = X’ Y Z’

D3 = X’ Y Z

D4 = X Y’ Z’

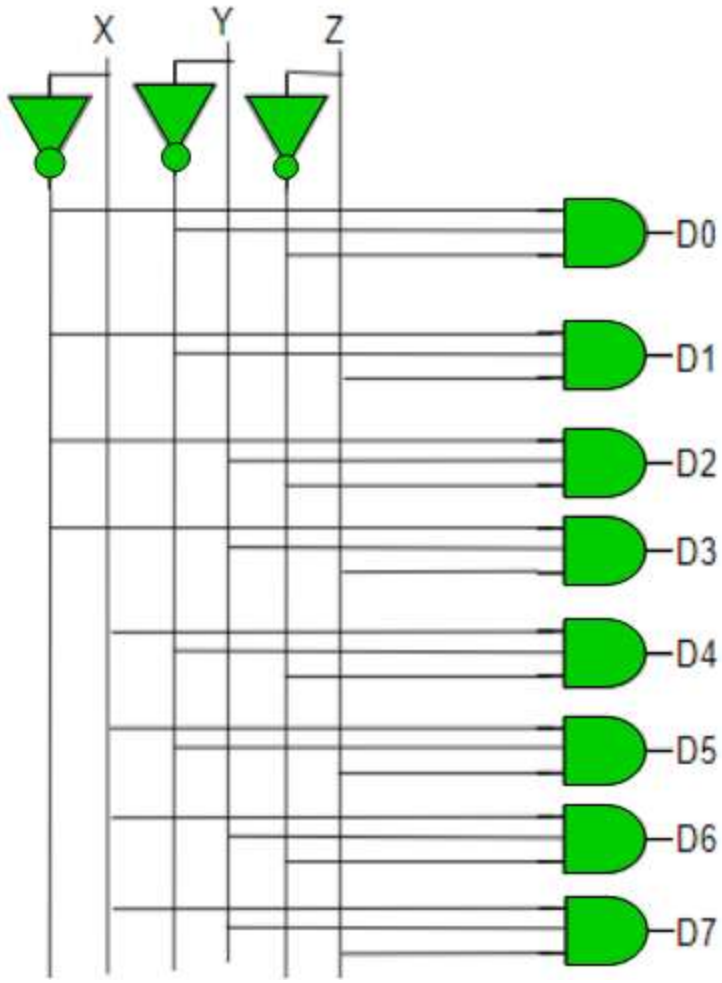
D5 = X Y’ Z

D6 = X Y Z’

D7 = X Y Z

Hence,

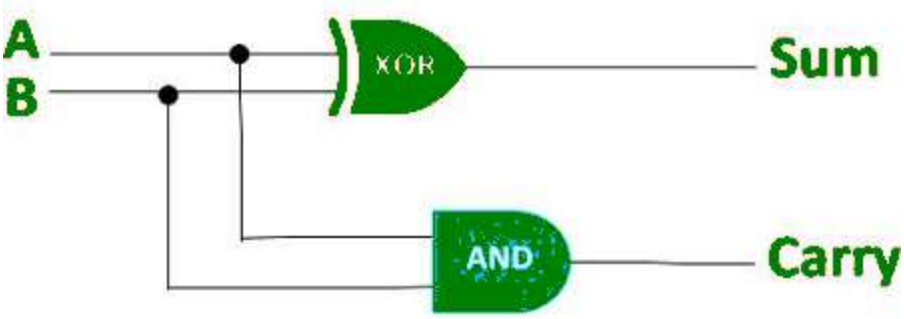
BCA Guides



3.6,2 Half Adder, & Full Adder

1. Half Adder **:**

Half Adder is a combinational logic circuit which is designed by connecting one EX-OR gate and one AND gate. The half adder circuit has two inputs: A and B, which add two input digits and generates a carry and a sum.



Half Adder

The output obtained from the EX-OR gate is the sum of the two numbers while that obtained by AND gate is the carry. There will be no forwarding of carry addition because there is no logic gate to process that. Thus, this is called Half Adder circuit.

Logical Expression :

```
Sum = A XOR B

Carry = A AND B
```

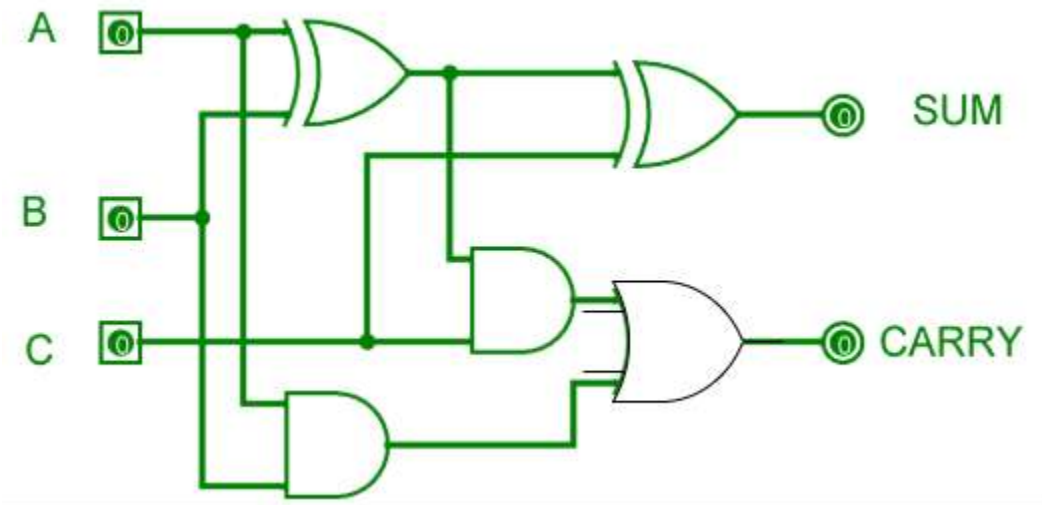
Truth Table :

Truth Table			
Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

2. Full Adder **:**

Full Adder is the circuit which consists of two EX-OR gates, two AND gates and one OR gate. Full Adder is the adder which adds three inputs and produces two outputs which consists of two EX-OR gates, two AND gates and one OR gate.

The first two inputs are A and B and the third input is an input carry as C-IN. The output carry is designated as C-OUT and the normal output is designated as S which is SUM.



Equation obtained by EX-OR gate is the sum of the binary digits. While the output obtained by AND gate is the carry obtained by addition.

Truth Table :

Input			Output	
A	B	Cin	Sum	Carry
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

Logical Expression :

SUM = (A XOR B) XOR Cin = (A $\oplus$ B) $\oplus$ Cin

CARRY-OUT = A AND B OR Cin(A XOR B) = A.B + Cin(A $\oplus$ B)

Difference between Half adder and full adder :

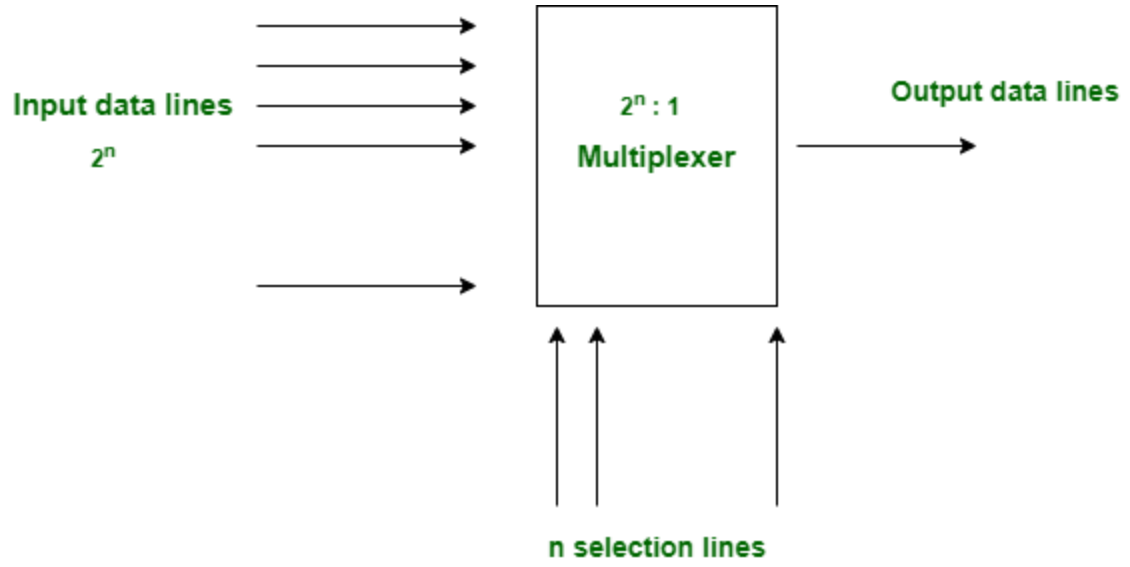
S.No.	Half Adder	Full Adder
1	Half Adder is combinational logic circuit which adds two 1-bit digits. The half adder produces a sum of the two inputs.	Full adder is combinational logical circuit that performs an addition operation on three one-bit binary numbers. The full adder produces a sum of the three inputs and carry value.
2	Previous carry is not used.	Previous carry is used.
3	In Half adder there are two input bits (A, B).	In full adder there are three input bits (A, B, C-in).
4	Logical Expression for half adder is : S=a⊕b ; C=a*b.	Logical Expression for Full adder is : S=a⊕b⊕Cin; Cout=(a*b)+(Cin*(a⊕b)).
5	It consists of one EX-OR gate and one AND gate.	It consists of two EX-OR, two AND gate and one OR gate.
6	It is used in Calculators, computers, digital measuring devices etc.	It is used in Multiple bit addition, digital processors etc.

3.7.1 Multiplexers and De-Multiplexers

1. Multiplexer **:**

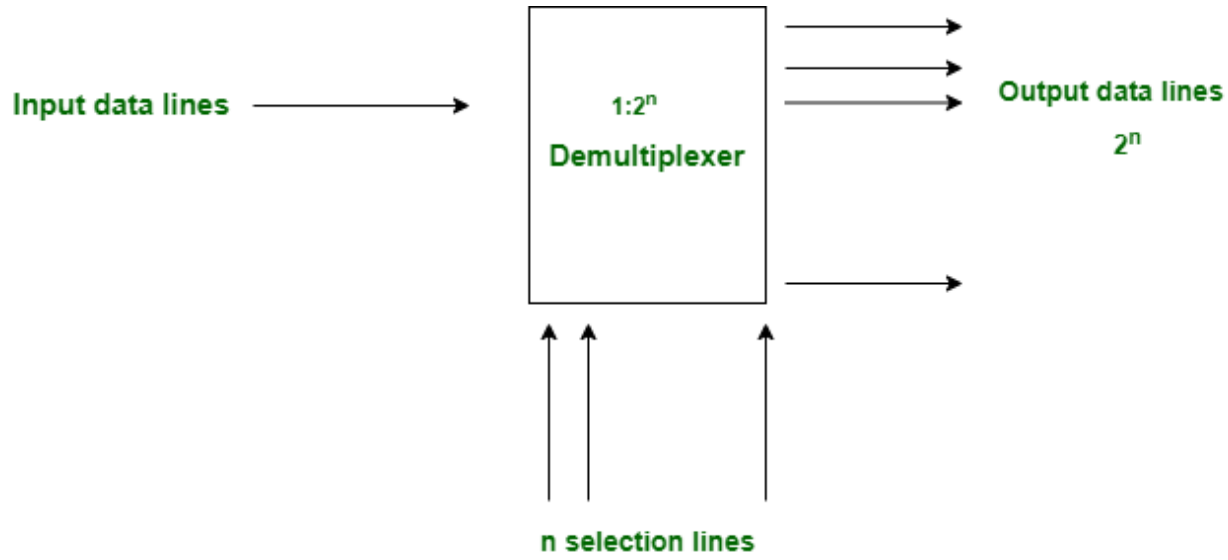
Multiplexer is a data selector which takes several inputs and gives a single output.In multiplexer we have 2^n input lines and 1 output lines where n is the number of selection lines.

BCA Guides Nepal



2. De-multiplexer :

Demultiplexer is a data distributor which takes a single input and gives several outputs.In demultiplexer we have 1 input and 2^n output lines where n is the selection line.



Difference between of Multiplexer and Demultiplexer :

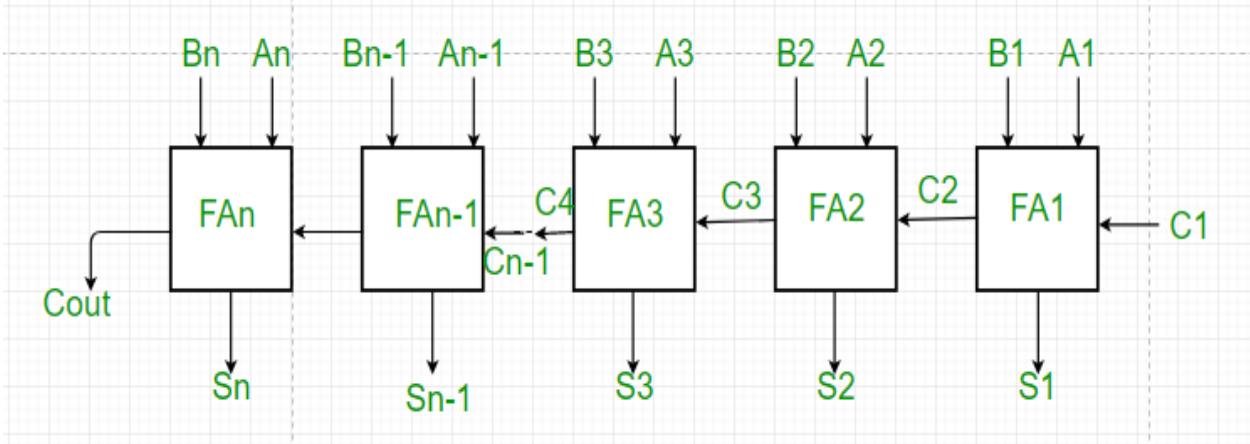
Multiplexer	Demultiplexer
Multiplexer processes the digital information from various sources into a single source.	Demultiplexer receives digital information from a single source and converts it into several sources

Multiplexer	Demultiplexer
It is known as Data Selector	It is known as Data Distributor
Multiplexer is a digital switch	Demultiplexer is a digital circuit
It follows combinational logic type	It also follows combinational logic type
It has n data input	It has single data input
It has a single data output	It has n data outputs
It works on many to one operational principle	It works on one to many operational principle
In time division Multiplexing, multiplexer is used at the transmitter end	In time division Multiplexing, demultiplexer is used at the receiver end

3.7.2 Parallel Adder -Binary Adder-Parity Generator /Checker-

Parallel Adder –

A single full adder performs the addition of two one bit numbers and an input carry. But a **Parallel Adder** is a digital circuit capable of finding the arithmetic **sum** of two binary numbers that is **greater than one bit** in length by operating on corresponding pairs of bits in parallel. It consists of **full adders connected in a chain** where the output carry from each full adder is connected to the carry input of the next higher order full adder in the chain. **A n bit parallel adder requires n full adders to perform the operation.** So for the two-bit number, two adders are needed while for four bit number, four adders are needed and so on. Parallel adders normally incorporate carry lookahead logic to ensure that carry propagation between subsequent stages of addition does not limit addition speed.

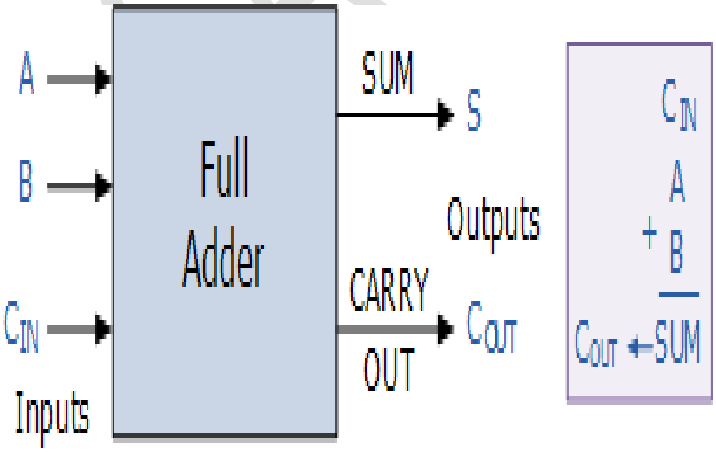


Working of parallel Adder –

1. As shown in the figure, firstly the full adder FA1 adds A1 and B1 along with the carry C1 to generate the sum S1 (the first bit of the output sum) and the carry C2 which is connected to the next adder in chain.
2. Next, the full adder FA2 uses this carry bit C2 to add with the input bits A2 and B2 to generate the sum S2(the second bit of the output sum) and the carry C3 which is again further connected to the next adder in chain and so on.
3. The process continues till the last full adder FAn uses the carry bit Cn to add with its input An and Bn to generate the last bit of the output along last carry bit Cout.

Binary Adder

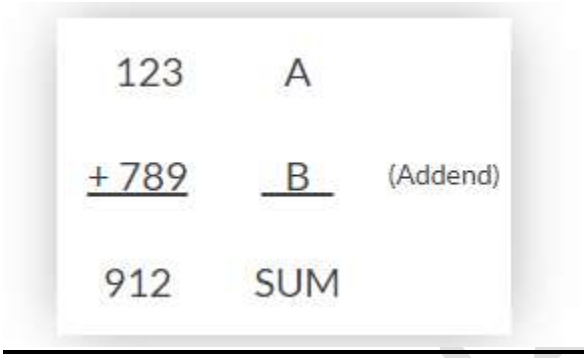
Binary Adders are arithmetic circuits in the form of half-adders and full-adders used to add together two binary digits



Another common and very useful combinational logic circuit which can be constructed using just a few basic logic gates allowing it to add together two or more binary numbers is the **Binary Adder**.

A basic Binary Adder circuit can be made from standard AND and Ex-OR gates allowing us to "add" together two single bit binary numbers, A and B.

The addition of these two digits produces an output called the SUM of the addition and a second output called the CARRY or Carry-out, (C_{OUT}) bit according to the rules for binary addition. One of the main uses for the *Binary Adder* is in arithmetic and counting circuits. Consider the simple addition of the two denary (base 10) numbers below.



Parity Bit Generator

There are two types of parity bit generators based on the type of parity bit being generated. **Even parity generator** generates an even parity bit. Similarly, **odd parity generator** generates an odd parity bit.

Even Parity Generator

Now, let us implement an even parity generator for a 3-bit binary input, WXY. It generates an even parity bit, P. If odd number of ones present in the input, then even parity bit, P should be '1' so that the resultant word contains even number of ones. For other combinations of input, even parity bit, P should be '0'. The following table shows the **Truth table** of even parity generator.

Binary Input WXY	Even Parity bit P
000	0
001	1

BCA Guides Nepal

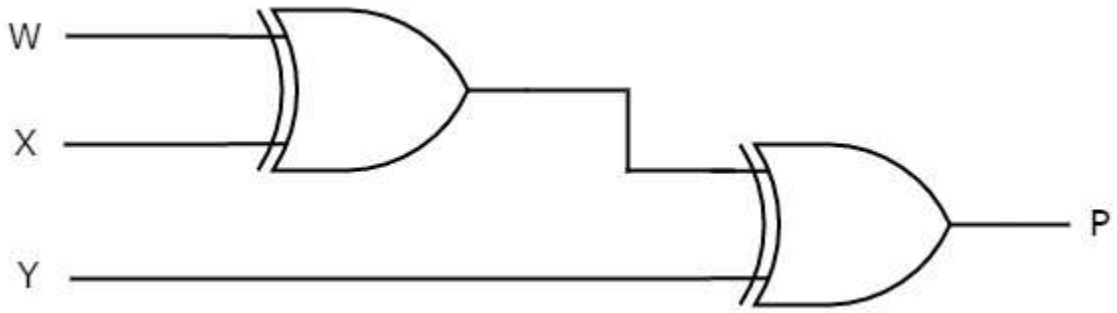
010	1
011	0
100	1
101	0
110	0
111	1

From the above Truth table, we can write the **Boolean function** for even parity bit as

$$P=W'X'Y+W'XY'+WX'Y'+WXY$$

$$\Rightarrow P=W'(X'Y+XY')+W(X'Y'+XY) \Rightarrow P=W'(X'Y+XY')+W(X'Y'+XY)$$
$$\Rightarrow P=W'(X\oplus Y)+W(X\oplus Y)'=W\oplus X\oplus Y \Rightarrow P=W'(X\oplus Y)+W(X\oplus Y)'=W\oplus X\oplus Y$$

The following figure shows the **circuit diagram** of even parity generator.

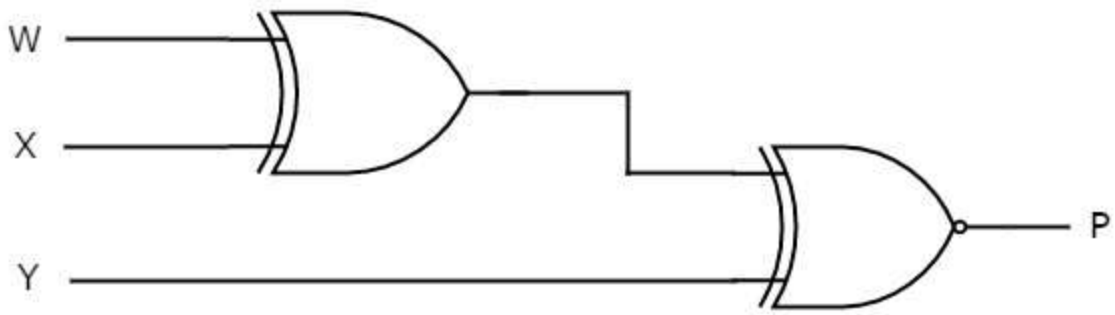


This circuit consists of two **Exclusive-OR gates** having two inputs each. First ExclusiveOR gate having two inputs W & X and produces an output W ⊕ X. This output is given as one input of second Exclusive-OR gate. The other input of this second Exclusive-OR gate is Y and produces an output of W ⊕ X ⊕ Y.

Odd Parity Generator

If even number of ones present in the input, then odd parity bit, P should be '1' so that the resultant word contains odd number of ones. For other combinations of input, odd parity bit, P should be '0'.

Follow the same procedure of even parity generator for implementing odd parity generator. The **circuit diagram** of odd parity generator is shown in the following figure.



The above circuit diagram consists of Ex-OR gate in first level and Ex-NOR gate in second level. Since the odd parity is just opposite to even parity, we can place an inverter at the output of even parity generator. In that case, the first and second levels contain an ExOR gate in each level and third level consist of an inverter.

Parity Checker

There are two types of parity checkers based on the type of parity has to be checked. **Even parity checker** checks error in the transmitted data, which contains message bits along with even parity. Similarly, **odd parity checker** checks error in the transmitted data, which contains message bits along with odd parity.

Even parity checker

Now, let us implement an even parity checker circuit. Assume a 3-bit binary input, WXY is transmitted along with an even parity bit, P. So, the resultant word datadata contains 4 bits, which will be received as the input of even parity checker.

It generates an **even parity check bit, E**. This bit will be zero, if the received data contains an even number of ones. That means, there is no error in the received data. This even parity check bit will be one, if the received data contains an odd number of ones. That means, there is an error in the received data.

The following table shows the **Truth table** of an even parity checker.

4-bit Received Data WXYP	Even Parity Check bit E
0000	0
0001	1
0010	1
0011	0
0100	1

BCA Guides Nepal

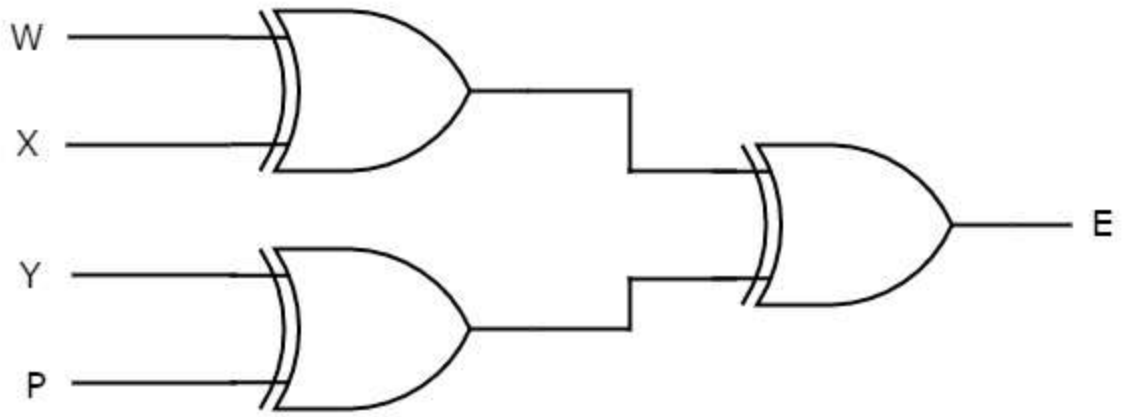
0101	0
0110	0
0111	1
1000	1
1001	0
1010	0
1011	1
1100	0
1101	1
1110	1
1111	0

From the above Truth table, we can observe that the even parity check bit value is '1', when odd number of ones present in the received data. That means the Boolean function of even parity check bit is an **odd function**. Exclusive-OR function satisfies this condition. Hence, we can directly write the **Boolean function** of even parity check bit as

$$E=W \oplus X \oplus Y \oplus P$$

The following figure shows the **circuit diagram** of even parity checker.

BCA Guides Nepal



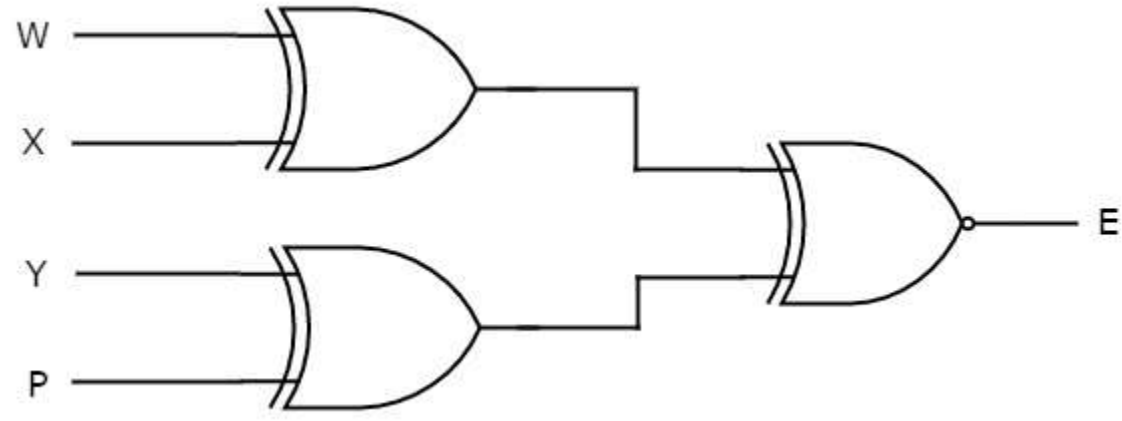
This circuit consists of three **Exclusive-OR gates** having two inputs each. The first level gates produce outputs of $W \oplus X$ W⊕X & $Y \oplus P$ Y⊕P. The Exclusive-OR gate, which is in second level produces an output of $W \oplus X \oplus Y \oplus P$ W⊕X⊕Y⊕P

Odd Parity Checker

Assume a 3-bit binary input, WXY is transmitted along with odd parity bit, P. So, the resultant word datadata contains 4 bits, which will be received as the input of odd parity checker.

It generates an **odd parity check bit, E**. This bit will be zero, if the received data contains an odd number of ones. That means, there is no error in the received data. This odd parity check bit will be one, if the received data contains even number of ones. That means, there is an error in the received data.

Follow the same procedure of an even parity checker for implementing an odd parity checker. The **circuit diagram** of odd parity checker is shown in the following figure.

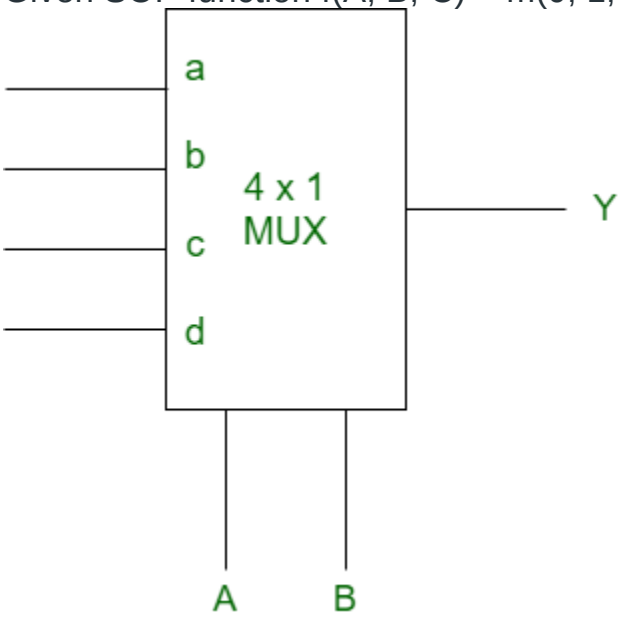


The above circuit diagram consists of Ex-OR gates in first level and Ex-NOR gate in second level. Since the odd parity is just opposite to even parity, we can place an inverter at the output of even parity checker. In that case, the first, second and third levels contain two Ex-OR gates, one Ex-OR gate and one inverter respectively.

3.7.3 Implementation of Logical Functions Using Multiplexers.

Given a SOP function and a multiplexer is also given. We will need to implement the given SOP function using the given MUX.
There are certain steps involved in it:
Step 1: Draw the truth table for the given number of variable function.
Step 2: Consider one variable as input and remaining variables as select lines.
Step 3: Form a matrix where input lines of MUX are columns and input variable and its compliment are rows.
Step 4: Find AND between both rows on the basis of the truth table.
Step 5: Hence whatever is found is considered as input of MUX.
We will illustrate it with an example:

Example:
Given SOP function f(A, B, C) = m(0, 1, 4, 6, 7) and MUX is



For 3 variable function, the truth table is

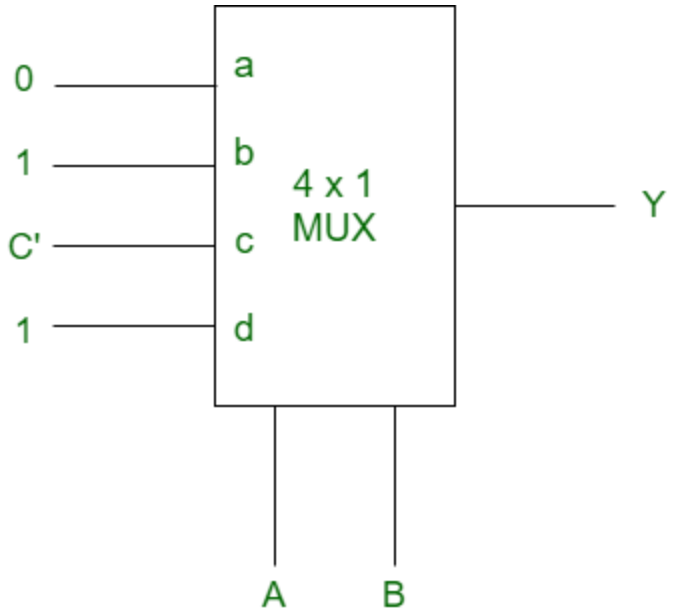
Truth Table

	A	B	C	Y
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

Let A and B are the select lines and C be the input,

	a	b	c	d
C'	0	2	4	6
C	1	3	5	7
	1	0	C'	1

Thus, for the implementation of given logical function, required is one 4×1 MUX and and inverter.



3.8 Basic Concepts of Programmable Logic

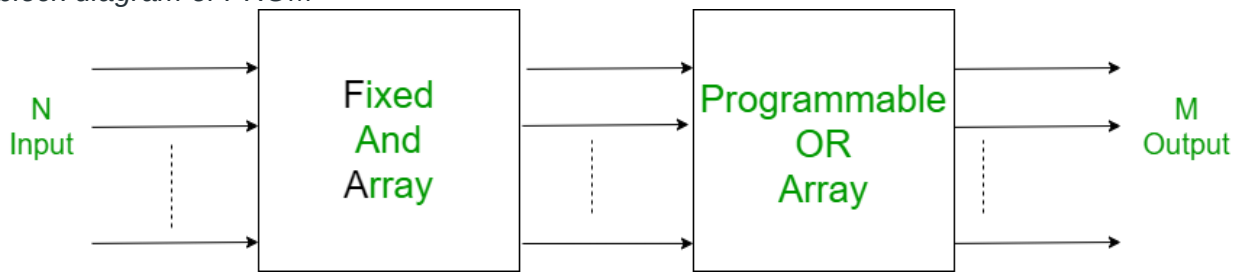
A programmable logic device is an electronic component used to build reconfigurable digital circuits. Unlike integrated circuits which consist of logic gates and have a fixed function, a PLD has an undefined function at the time of manufacture.

3.8.1 PROM

PROM stands for **Programmable Read-Only Memory**
It is a memory chip on which information can be composed as it were once. Once a program has been composed onto a PROM, it remains there until the end of time. Not at all like Ram, PROMs hold their substance when the computer is turned off. A PROM is fabricated as blank memory, and after that, an uncommon burner is utilized to compose to it once. PROMS ordinarily store low-level gadget drivers for particular equipment and are burned once.

A programmable read-only memory may be a frame of advanced memory where the setting of each bit is bolted by a meld or anti-fuse. It is one sort of ROM. The information in them is changeless and cannot be changed. PROM could be a sort of ROM that's modified after the memory is built. PROM chips have a few diverse applications, counting cell phones, video amusement supports, RFID labels, restorative gadgets, and other gadgets. They give a straightforward implies of programming electronic gadgets

block diagram of PROM



Applications of PROM

- Mobile Phones for giving Client Particular Selections.
- Implantable Restorative devices.
- Radio-Frequency Identification (RFID)tags.
- High definition Multimedia Interfaces(HDMI)

Characteristics of PROM

- It has programmable random access memory
- In prom And gates are fixed and or gates are programmable
- Prom Works as a memory
- It is not reusable.
- The storage endurance of PROM is high
- few functions can be actualized(it is effective when the work to be implemented(i.e the min terms) must have all the variables.

Advantages of PROM

- The programming can be done utilizing numerous sorts of program and does not depend on difficult wiring of the program to the chip.
- Since it isn't conceivable to un-blow the intertwine, so the realness of the information remains intaglio and it is outlandish to evacuate or modify the substance.

Disadvantages of PROM

- PROM is that the data once burnt cannot be deleted or changed when recognized with errors.

3.8.2 EPROM

EPROM stands for **Erasable Programmable Read-Only Memory**. It is a memory chip that is non-volatile in nature that is it can hold the data even after the power supply is stopped. It can be reused again and again as it is easily programmable and erasable.

In 1967, Dawon Kahng and Simon Sze at Bell Labs proposed that the floating gate of a MOSFET(metal-oxide-semiconductor field-effect transistor) could be used for the cell of a reprogrammable ROM (read-only memory). With this concept in mind, Dov Frohman of Intel invented EPROM in 1971. Frohman

designed the Intel 1702, a 2048-bit EPROM, which was announced by Intel in 1971.

Characteristics

- Each and every EPROM is programmed by electronic devices.
- The data contained in EPROM is erased by exposing it to ultraviolet light.
- EPROM can store data minimum for 10-20 years.
- Erasing window is kept covered to avoid unwanted exposure to UV light to avoid accidental loss of data.

Advantages

- Easily erasable and programmable.
- Quite effective – As memory data can be erased again and again for use, therefore it eliminates the need of other external memory.<="" li="" style="box-sizing: border-box;">

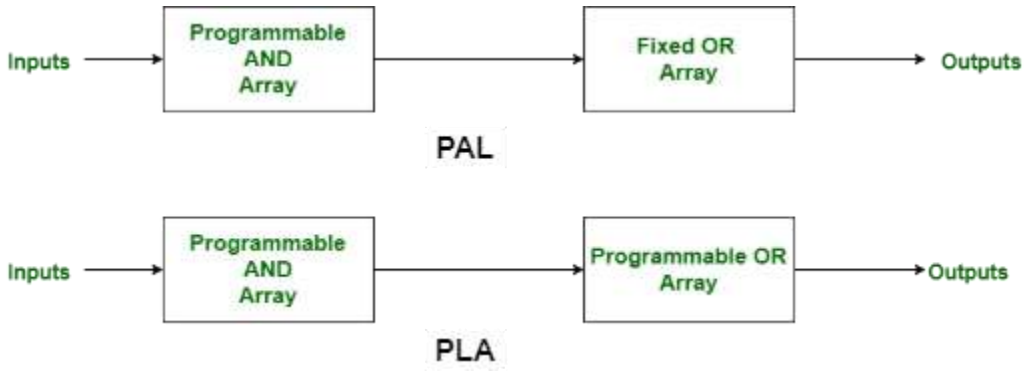
Disadvantages

- A particular selected data is not deleted instead whole data gets erased which is the cause of worry for the user.
- User needs to keep backup as whole data gets erased.
- It needs UV light to erase the data which is very rare.
- Process of erasing data is quite complex.

3.8.3 PAL

3.8.4 PLA

Programmable Logic Array (PLA) and Programming Array Logic (PAL) are the categories of programming logic device.
In PLA or Programmable Logic Array, there are massive functions can be implemented. Whereas in PAL or Programmable Array Logic, there is finite functions can be implemented.
The distinction between PLA and PAL is that, PAL have programmable AND array and fixed or array. On the other hand, PLA have programmable AND array and programming OR array.



Let’s see that the difference between PLA and PAL:

S.NO	PLA	PAL
1.	PLA stands for Programmable Logic Array.	While PAL stands for Programmable Array Logic.
2.	PLA speed is lower than PAL.	While PAL’s speed is higher than PLA.
3.	The complexity of PLA is high.	While PAL’s complexity is less.
4.	The cost of PLA is also high.	While the cost of PAL is low.
5.	Programmable Logic Array is less available.	While Programmable Array Logic is more available than Programmable Logic Array.
6.	It is less used than PAL.	While it is more used than PLA.

BCA Guides Nepal

Unit 4 Counters & Registers 16 Hrs.

4.1 RS, JK, JK Master - Slave. D & T Flip flops 4.1.1 Level Triggering and Edge Triggering 4.1.2 Excitation Tables 4.2 Asynchronous and Synchronous Counters 4.2.1 Ripple Counter: Circuit and State Diagram and Timing Waveforms 4.2.2 Ring Counter: Circuit and State Diagram and Timing Waveforms 4.2.3 Modulus 10 Counter: Circuit and State Diagram and Timing Waveforms 4.2.4 Modulus Counters (5, 7, 11) and Design Principle, Circuit and State Diagram 4.2.5 Synchronous Design of Above Counters, Circuit Diagrams and State Diagrams 4.3 Application of Counters 4.3.1 Digital Watch 4.3.2 Frequency Counter 4.4 Registers 4.4.1 Serial in Parallel out Register 4.4.2 Serial in Serial out Register 4.4.3 Parallel in Serial out Register 4.4.4 Parallel in Parallel out Register 4.4.5 Right Shift, Left Shift Register

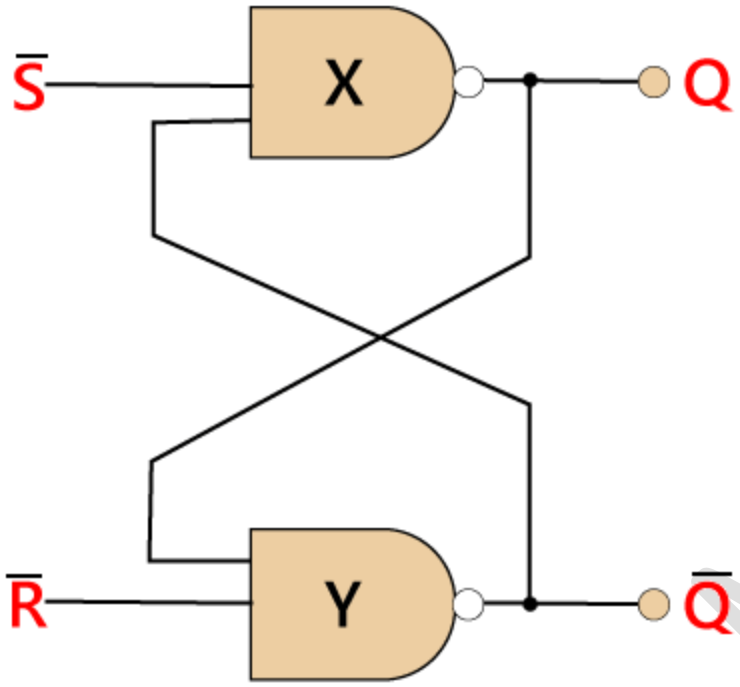
4.1 SR, JK, JK Master - Slave. D & T Flip flops

Basics of Flip Flop

A circuit that has two stable states is treated as a **flip flop**. These stable states are used to store binary data that can be changed by applying varying inputs. The flip flops are the fundamental building blocks of the digital system. Flip flops and latches are examples of data storage elements. In the sequential logical circuit, the flip flop is the basic storage element. The latches and flip flops are the basic storage elements but different in working. There are the following types of flip flops:

SR Flip Flop

The S-R flip flop is the most common flip flop used in the digital system. In SR flip flop, when the set input "S" is true, the output Y will be high, and Y' will be low. It is required that the wiring of the circuit is maintained when the outputs are established. We maintain the wiring until set or reset input goes high, or power is shutdown.



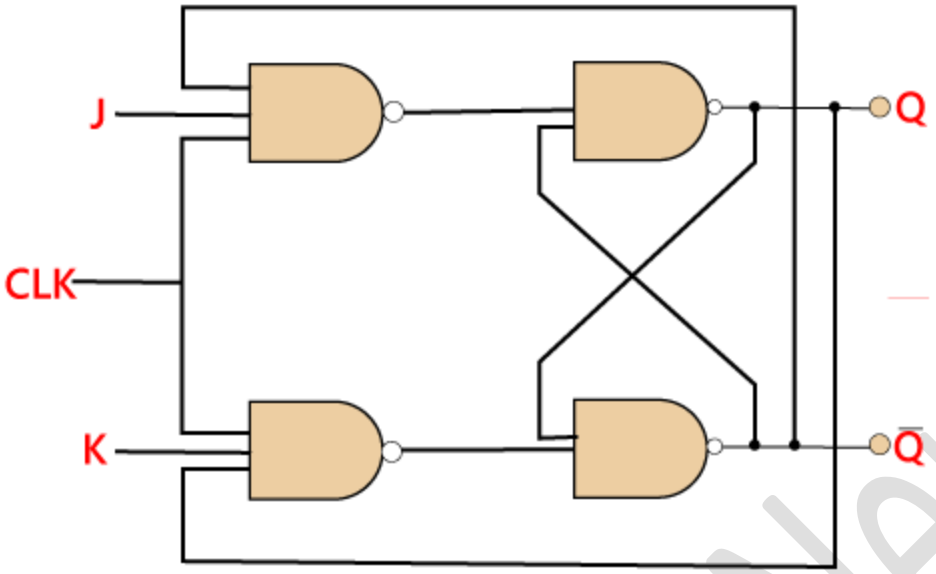
The S-R flip flop is the simplest and easiest circuit to understand.

Truth Table:

S	R	Y	Y'
0	0	0	1
0	1	0	1
1	0	1	0
1	1	∞	∞

J-K Flip-flop

The JK flip flop is used to remove the drawback of the S-R flip flop, i.e., undefined states. The JK flip flop is formed by doing modification in the SR flip flop. The S-R flip flop is improved in order to construct the J-K flip flop. When S and R input is set to true, the SR flip flop gives an inaccurate result. But in the case of JK flip flop, it gives the correct output.



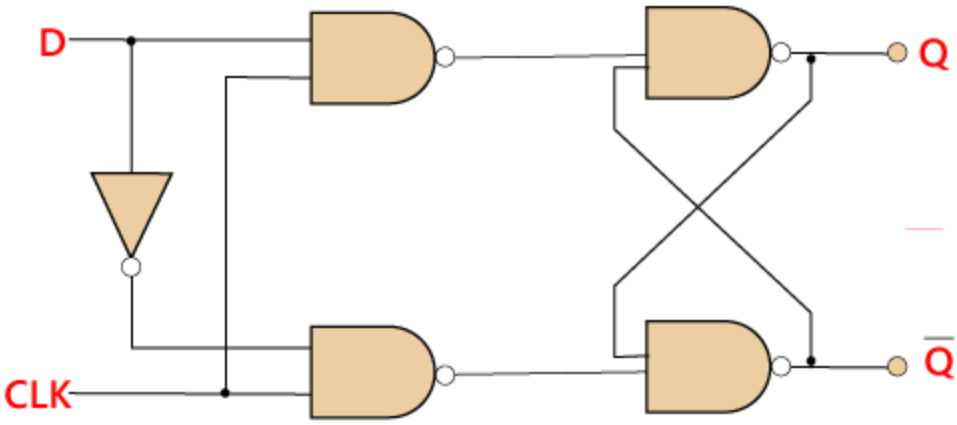
In J-K flip flop, if both of its inputs are different, the value of J at the next clock edge is taken by the output Y. If both of its input is low, then no change occurs, and if high at the clock edge, then from one state to the other, the output will be toggled. The JK Flip Flop is a Set or Reset Flip flop in the digital system.

Truth Table:

J	K	Y	Y'
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	1
0	1	1	0
1	0	1	1
1	1	1	0

D Flip Flop

D flip flop is a widely used flip flop in digital systems. The D flip flop is mostly used in shift-registers, counters, and input synchronization.

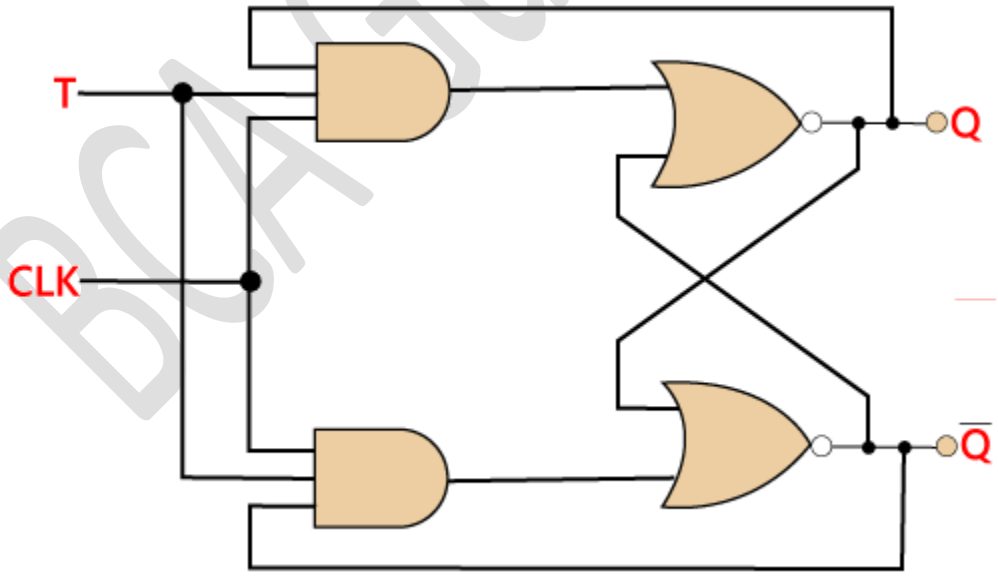


Truth Table:

Clock	D	Y	Y'
↓ » 0	0	0	1
↑ » 1	0	0	1
↓ » 0	1	0	1
↑ » 1	1	1	0

T Flip Flop

Just like JK flip-flop, T flip flop is used. Unlike JK flip flop, in T flip flop, there is only single input with the clock input. The T flip flop is constructed by connecting both of the inputs of JK flip flop together as a single input.



BCA Guides Nepal

The T flip flop is also known as **Toggle flip-flop**. These T flip-flops are able to find the complement of its state.

Truth Table:

T	Y	Y (t+1)
0	0	0
1	0	1
0	1	1
1	1	0

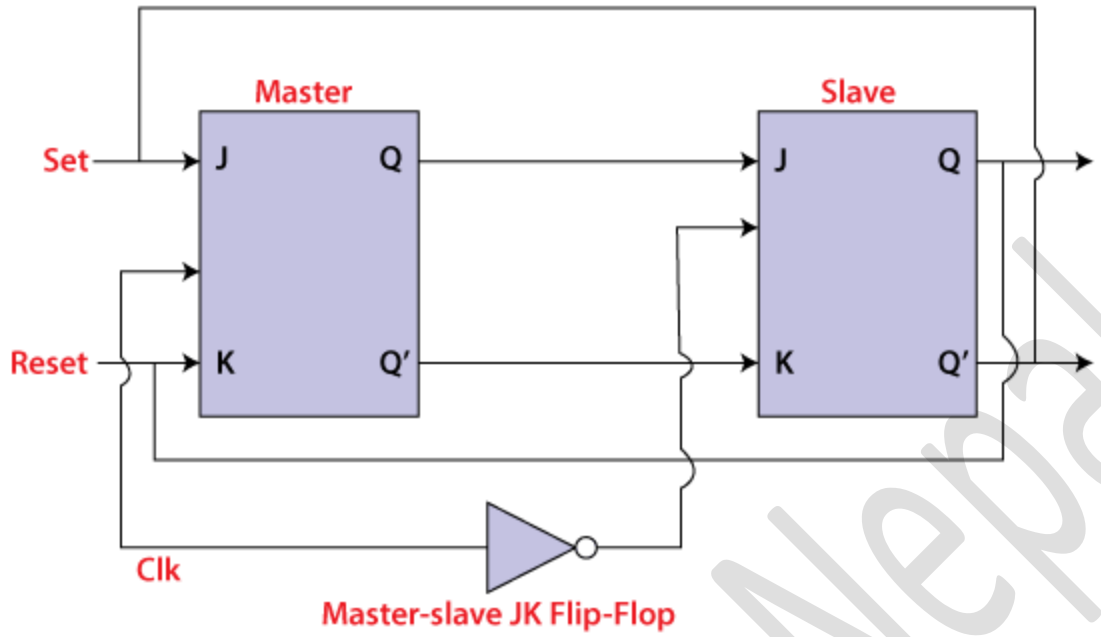
Master-Slave JK Flip Flop

In "JK Flip Flop", when both the inputs and CLK set to 1 for a long time, then Q output toggle until the CLK is 1. Thus, the uncertain or unreliable output produces. This problem is referred to as a race-round condition in JK flip-flop and avoided by ensuring that the CLK set to 1 only for a very short time.

Explanation

The master-slave flip flop is constructed by combining two JK flip flops. These flip flops are connected in a series configuration. In these two flip flops, the 1st flip flop work as "master", called the master flip flop, and the 2nd work as a "slave", called slave flip flop. The master-slave flip flop is designed in such a way that the output of the "master" flip flop is passed to both the inputs of the "slave" flip flop. The output of the "slave" flip flop is passed to inputs of the master flip flop.

In "master-slave flip flop", apart from these two flip flops, an inverter or NOT gate is also used. For passing the inverted clock pulse to the "slave" flip flop, the inverter is connected to the clock's pulse. In simple words, when CP set to false for "master", then CP is set to true for "slave", and when CP set to true for "master", then CP is set to false for "slave".

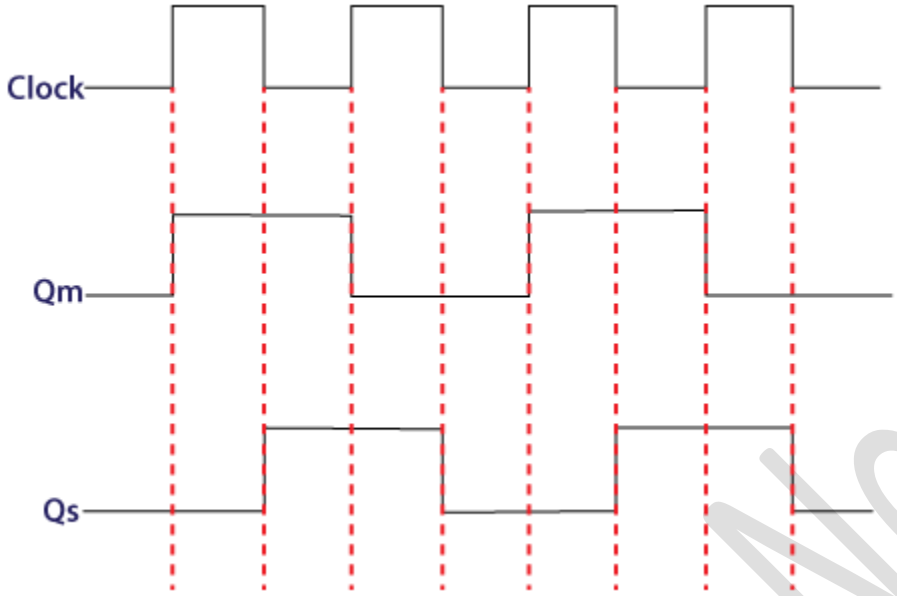


Master-slave JK Flip-Flop

Working:

- When the clock pulse is true, the slave flip flop will be in the isolated state, and the system's state may be affected by the J and K inputs. The "slave" remains isolated until the CP is 1. When the CP set to 0, the master flip-flop passes the information to the slave flip flop to obtain the output.
- The master flip flop responds first from the slave because the master flip flop is the positive level trigger, and the slave flip flop is the negative level trigger.
- The output Q'=1 of the master flip flop is passed to the slave flip flop as an input K when the input J set to 0 and K set to 1. The clock forces the slave flip flop to work as reset, and then the slave copies the master flip flop.
- When J=1, and K=0, the output Q=1 is passed to the J input of the slave. The clock's negative transition sets the slave and copies the master.
- The master flip flop toggles on the clock's positive transition when the inputs J and K set to 1. At that time, the slave flip flop toggles on the clock's negative transition.
- The flip flop will be disabled, and Q remains unchanged when both the inputs of the JK flip flop set to 0.

Timing Diagram of a Master Flip Flop:

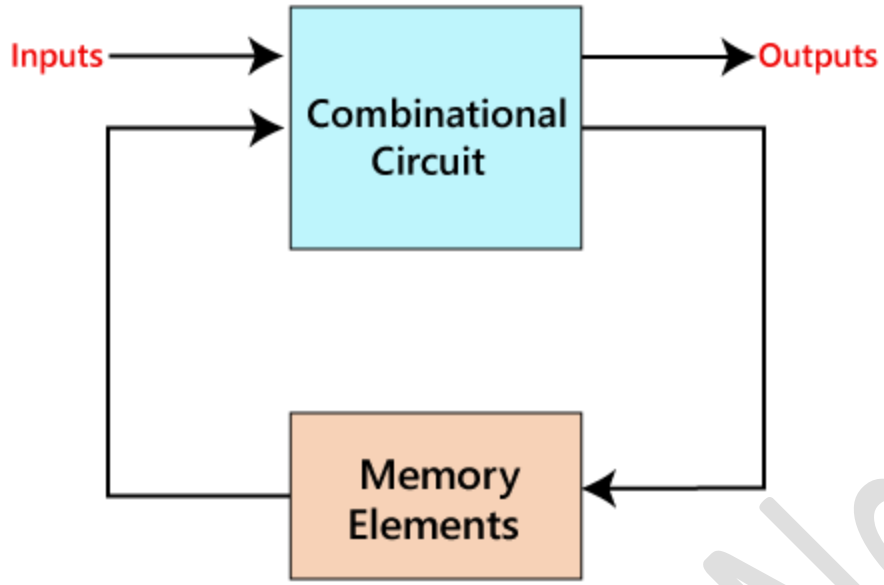


- When the clock pulse set to 1, the output of the master flip flop will be one until the clock input remains 0.
- When the clock pulse becomes high again, then the master's output is 0, which will be set to 1 when the clock becomes one again.
- The master flip flop is operational when the clock pulse is 1. The slave's output remains 0 until the clock is not set to 0 because the slave flip flop is not operational.
- The slave flip flop is operational when the clock pulse is 0. The output of the master remains one until the clock is not set to 0 again.
- Toggling occurs during the entire process because the output changes once in the cycle.

4.1.1 Level Triggering and Edge Triggering

Introduction

In our previous sections, we learned about combinational circuit and their working. The combinational circuits have set of outputs, which depends only on the present combination of inputs. Below is the block diagram of the synchronous logic circuit.



The sequential circuit is a special type of circuit that has a series of inputs and outputs. The outputs of the sequential circuits depend on both the combination of present inputs and previous outputs. The previous output is treated as the present state. So, the sequential circuit contains the combinational circuit and its memory storage elements. A sequential circuit doesn't need to always contain a combinational circuit. So, the sequential circuit can contain only the memory element.

Difference between the combinational circuits and sequential circuits are given below:

	Combinational Circuits	Sequential Circuits
1)	The outputs of the combinational circuit depend only on the present inputs.	The outputs of the sequential circuits depend on bo present inputs and present state(previous output).
2)	The feedback path is not present in the combinational circuit.	The feedback path is present in the sequential circuits.
3)	In combinational circuits, memory elements are not required.	In the sequential circuit, memory elements play a important role and require.
4)	The clock signal is not required for combinational circuits.	The clock signal is required for sequential circuits.

5)	The combinational circuit is simple to design.	It is not simple to design a sequential circuit.

Types of Sequential Circuits

Asynchronous sequential circuits

The clock signals are not used by the **Asynchronous sequential circuits**. The asynchronous circuit is operated through the pulses. So, the changes in the input can change the state of the circuit. The asynchronous circuits do not use clock pulses. The internal state is changed when the input variable is changed. The un-clocked flip-flops or time-delayed are the memory elements of asynchronous sequential circuits. The asynchronous sequential circuit is similar to the combinational circuits with feedback.

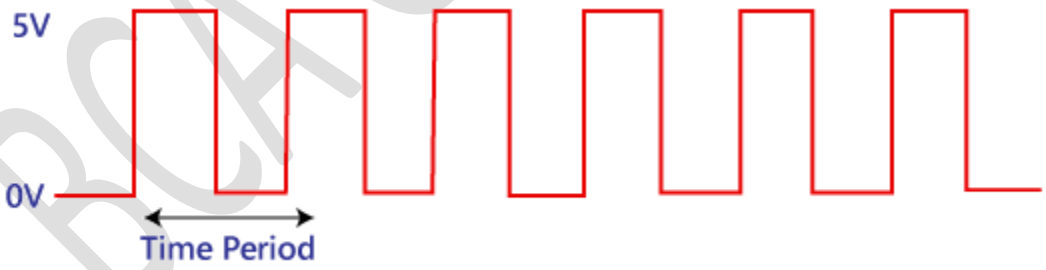
Synchronous sequential circuits

In synchronous sequential circuits, synchronization of the memory element's state is done by the clock signal. The output is stored in either flip-flops or latches(memory devices). The synchronization of the outputs is done with either only negative edges of the clock signal or only positive edges.

Clock Signal and Triggering

Clock signal

A clock signal is a periodic signal in which ON time and OFF time need not be the same. When ON time and OFF time of the clock signal are the same, a square wave is used to represent the clock signal. Below is a diagram which represents the clock signal:



A clock signal is considered as the square wave. Sometimes, the signal stays at logic, either high 5V or low 0V, to an equal amount of time. It repeats with a certain time period, which will be equal to twice the 'ON time' or 'OFF time'.

Types of Triggering

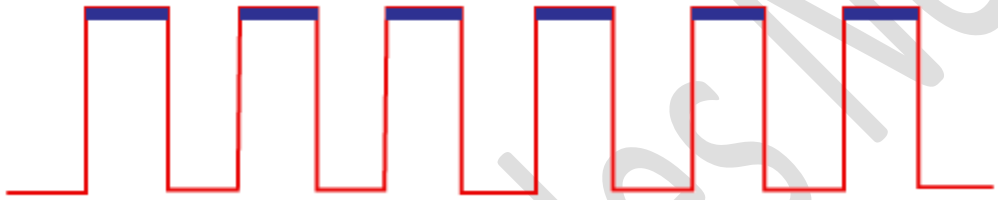
These are two types of triggering in sequential circuits:

Level triggering

The logic High and logic Low are the two levels in the clock signal. In level triggering, when the clock pulse is at a particular level, only then the circuit is activated. There are the following types of level triggering:

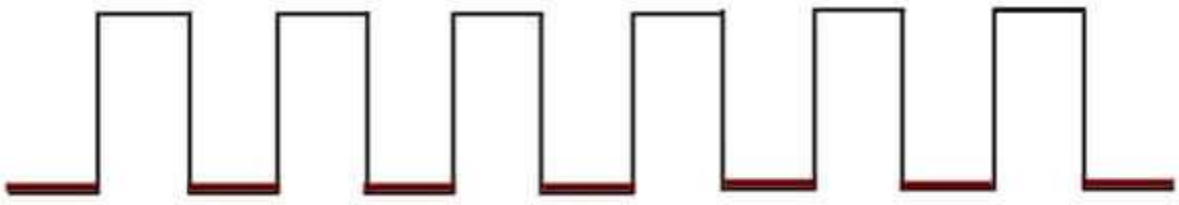
Positive level triggering

In a positive level triggering, the signal with Logic High occurs. So, in this triggering, the circuit is operated with such type of clock signal. Below is the diagram of positive level triggering:



Negative level triggering

In negative level triggering, the signal with Logic Low occurs. So, in this triggering, the circuit is operated with such type of clock signal. Below is the diagram of Negative level triggering:



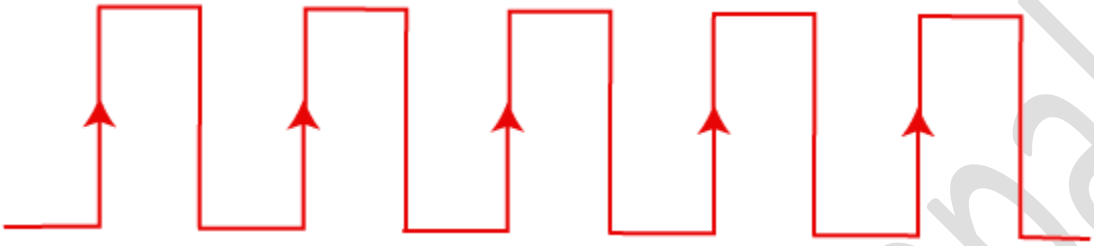
Edge triggering

In clock signal of edge triggering, two types of transitions occur, i.e., transition either from Logic Low to Logic High or Logic High to Logic Low.

Based on the transitions of the clock signal, there are the following types of edge triggering:

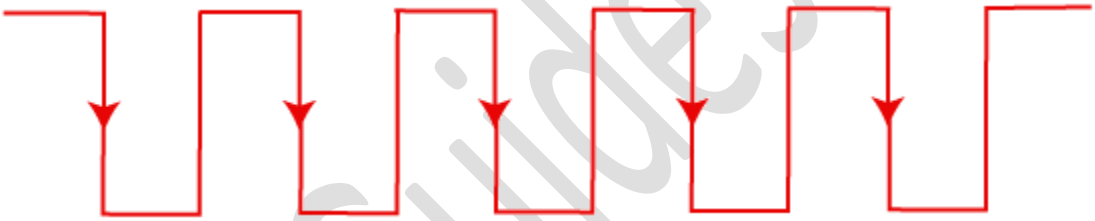
Positive edge triggering

The transition from Logic Low to Logic High occurs in the clock signal of positive edge triggering. So, in positive edge triggering, the circuit is operated with such type of clock signal. The diagram of positive edge triggering is given below.



Negative edge triggering

The transition from Logic High to Logic low occurs in the clock signal of negative edge triggering. So, in negative edge triggering, the circuit is operated with such type of clock signal. The diagram of negative edge triggering is given below.



4.1.2 Excitation Tables

4.2 Asynchronous and Synchronous Counters

Counters

A special type of sequential circuit used to count the pulse is known as a counter, or a collection of flip flops where the clock signal is applied is known as counters.

The counter is one of the widest applications of the flip flop. Based on the clock pulse, the output of the counter contains a predefined state. The number of the pulse can be counted using the output of the counter.

Truth Table

Clock	Counter output		State number	Decimal counter output
	Q_B	Q_A		
Initially	0	0	-	0
1st	0	1	1	1
2nd	1	0	2	2
3rd	1	1	3	3
4th	0	0	4	0

There are the following types of counters:

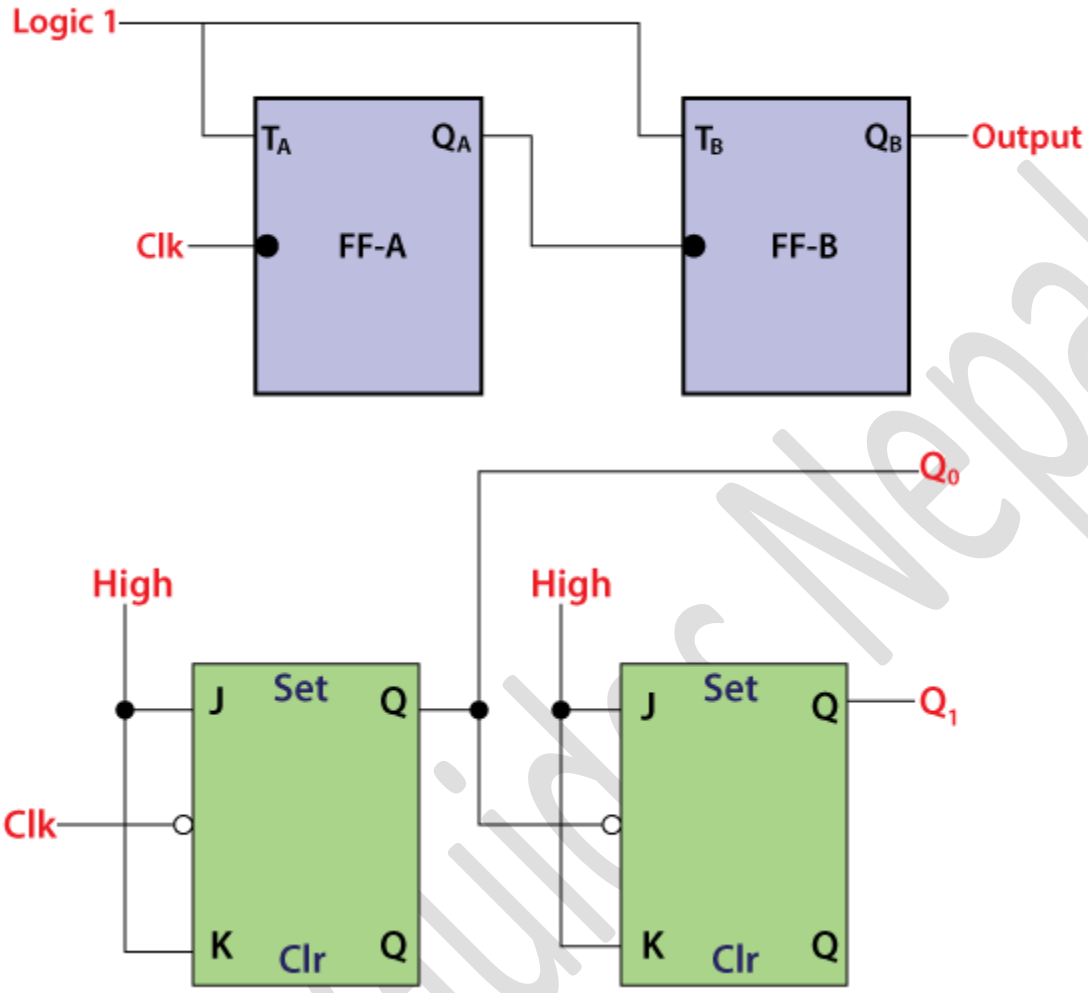
- Asynchronous Counters
- Synchronous Counters

Asynchronous or ripple counters

The **Asynchronous counter** is also known as the **ripple counter**. Below is a diagram of the 2-bit **Asynchronous counter** in which we used two T flip-flops. Apart from the T flip flop, we can also use the JK flip flop by setting both of the inputs to 1 permanently. The external clock pass to the clock input of the first flip flop, i.e., FF-A and its output, i.e., is passed to clock input of the next flip flop, i.e., FF-B.

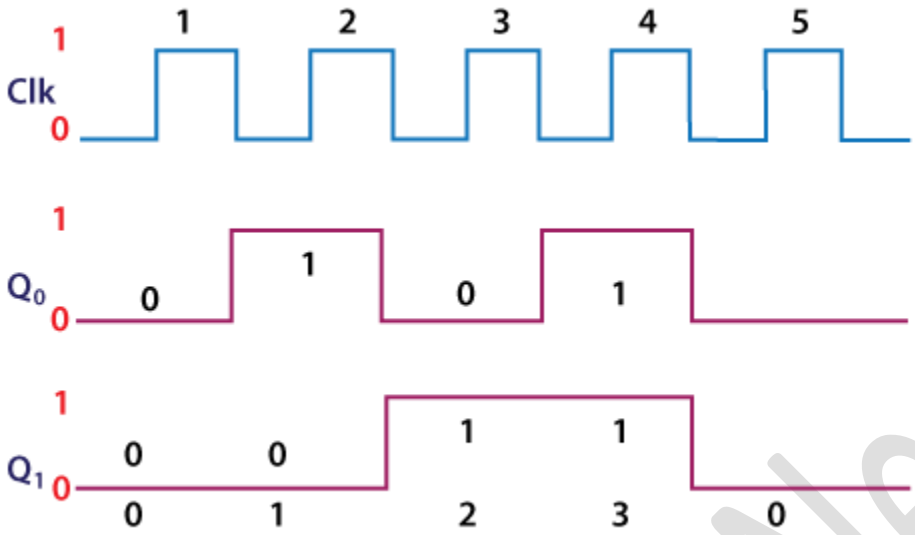
BCA Guides Nepal

Block Diagram



BCA Guides Nepal

Signal Diagram



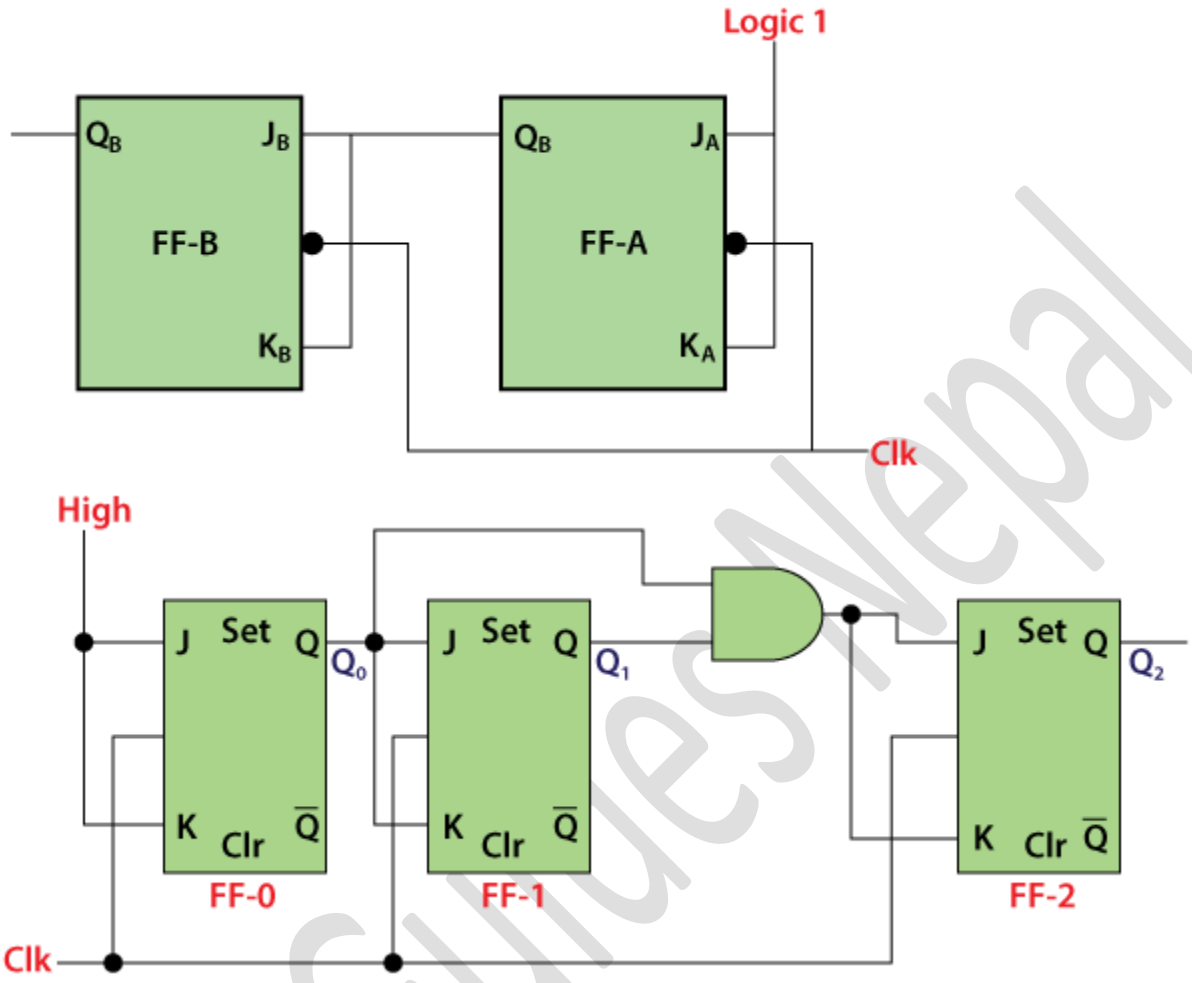
Synchronous counters

In the **Asynchronous counter**, the present counter's output passes to the input of the next counter. So, the counters are connected like a chain. The drawback of this system is that it creates the counting delay, and the propagation delay also occurs during the counting stage. The **synchronous counter** is designed to remove this drawback.

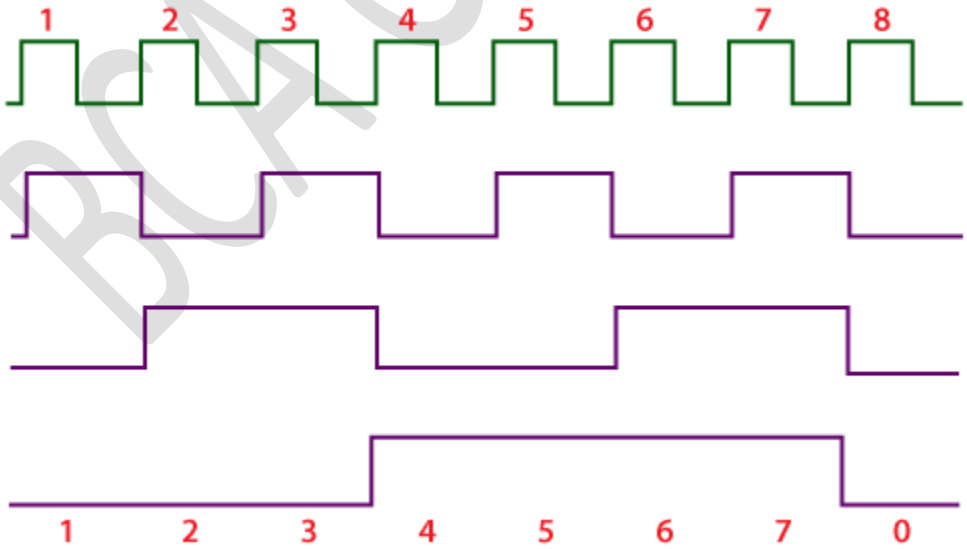
In the **synchronous counter**, the same clock pulse is passed to the clock input of all the flip flops. The clock signals produced by all the flip flops are the same as each other. Below is the diagram of a 2-bit synchronous counter in which the inputs of the first flip flop, i.e., FF-A, are set to 1. So, the first flip flop will work as a toggle flip-flop. The output of the first flip flop is passed to both the inputs of the next JK flip flop.

BCA Guides Nepal

Logical Diagram



Signal Diagram



BCA Guides Nepal

4.2.1 Ripple Counter: Circuit and State Diagram and Timing Waveforms

Ripple Counter

Ripple counter is a special type of **Asynchronous** counter in which the clock pulse ripples through the circuit. The n-MOD ripple counter forms by combining n number of flip-flops. The n-MOD ripple counter can count 2n states, and then the counter resets to its initial value.

Features of the Ripple Counter:

- Different types of flip flops with different clock pulse are used.
- It is an example of an asynchronous counter.
- The flip flops are used in toggle mode.
- The external clock pulse is applied to only one flip flop. The output of this flip flop is treated as a clock pulse for the next flip flop.
- In counting sequence, the flip flop in which external clock pulse is passed, act as LSB.

Based on their circuitry design, the counters are classified into the following types:

Up Counter

The up-counter counts the states in ascending order.

Down Counter

The down counter counts the states in descending order.

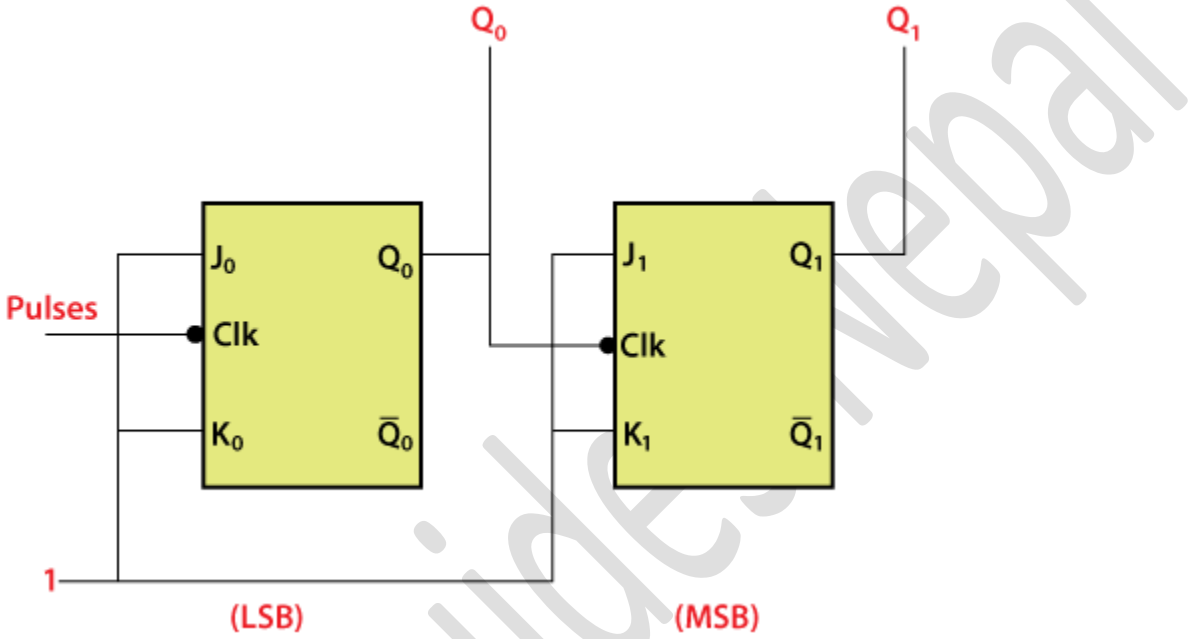
Up-Down Counter

The up and down counter is a special type of bi-directional counter which counts the states either in the forward direction or reverse direction. It also refers to a reversible counter.

Binary Ripple Counter

A **Binary counter** is a **2-Mod counter** which counts up to 2-bit state values, i.e., 22 = 4 values. The flip flops having similar conditions for toggling like T and JK are used to construct the **Ripple counter**. Below is a circuit diagram of **a binary ripple counter**.

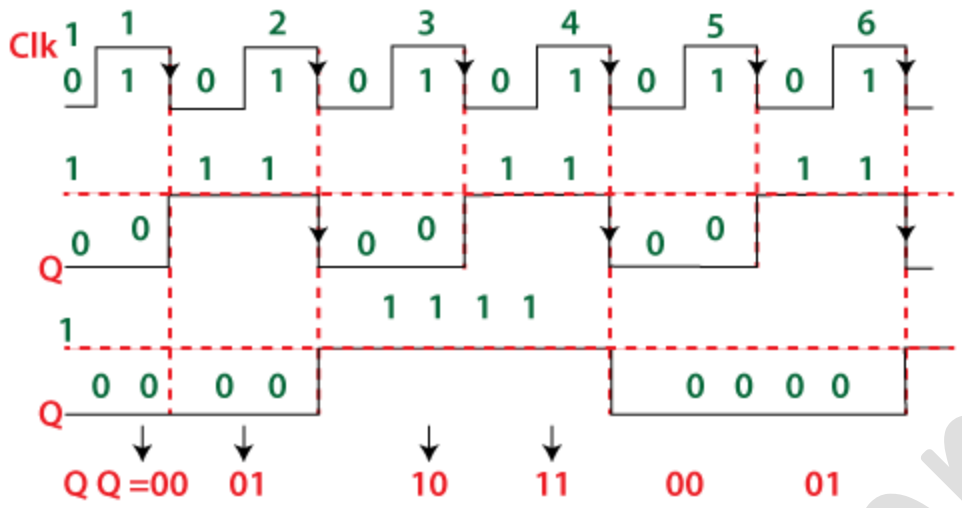
In the circuit design of the binary ripple counter, two JK flip flops are used. The high voltage signal is passed to the inputs of both flip flops. This high voltage input maintains the flip flops at a state 1. In JK flip flops, the negative triggered clock pulse use.



The outputs Q_0 and Q_1 are the LSB and MSB bits, respectively. The truth table of JK flip flop helps us to understand the functioning of the counter.

J_n	K_n	Qn+1
0	0	Q_n
1	0	1
0	1	0
1	1	Q_n

When the high voltage to the inputs of the flip flops, the fourth condition is of the JK flip flop occurs. The flip flops will be at the state 1 when we apply high voltage to the input of the flip-flop. So, the states of the flip flops passes are toggled at the negative going end of the clock pulse. In simple words, the flip flop toggle when the clock pulse transition takes place from 1 to 0.



The state of the output Q_0 change when the negative clock edge passes to the flip flop. Initially, all the flip flops are set to 0. These flip flop changes their states when the passed clock goes from 1 to 0. The JK flip flop toggles when the inputs of the flip flops are one, and then the flip flop changes its state from 0 to 1. For all the clock pulse, the process remains the same.

Number of input pulses	Q_1	Q_0
0	–	-
1	0	0
2	0	1
3	1	0
4	1	1

The output of the first flip flop passes to the second flip flop as a clock pulse. From the above timing diagram, it is clear that the state of the second flip flop is changed when the output Q_0 goes transition from 1 to 0. The outputs Q_0 and Q_1 treat as LSB and MSB. The counter counts the values 00, 01, 10, 11. After counting these values, the counter resets itself and starts counting again from 00, 01, 10, and 1. The count values until the clock pulses are passed to J_0K_0 flip flop.

4.2.2 Ring Counter: Circuit and State Diagram and Timing Waveforms

Ring Counter

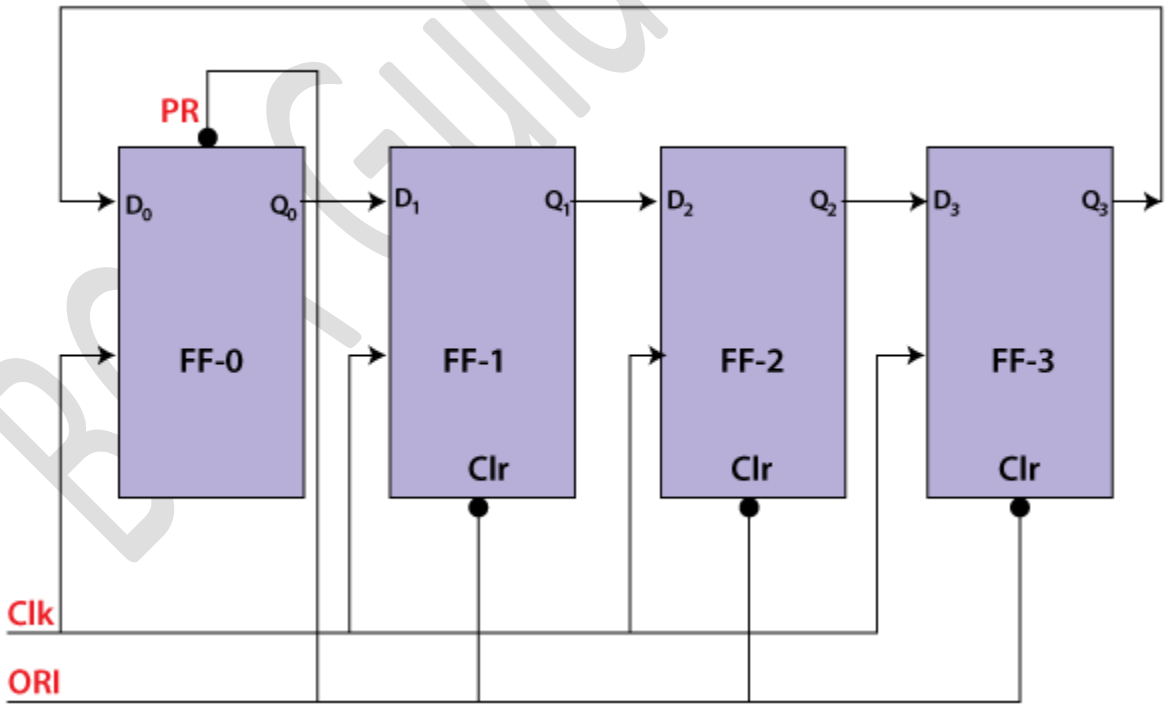
A ring counter is a special type of application of the **Serial IN Serial OUT** Shift register. The only difference between the shift register and the ring counter is that the last flip flop outcome is taken as the output in the shift register. But in the ring counter, this outcome is passed to the first flip flop as an input. All of the remaining things in the ring counter are the same as the shift register.

In **the Ring counter**

No. of states in Ring counter = No. of flip-flop used

Below is the block diagram of the 4-bit ring counter. Here, we use 4 **D flip flops**. The same clock pulse is passed to the clock input of all the flip flops as a synchronous counter. The **Overriding input(ORI)** is used to design this circuit.

The Overriding input is used as **clear** and **pre-set**.



BCA Guides

The output is 1 when the pre-set set to 0. The output is 0 when the clear set to 0. Both PR and CLR always work in value 0 because they are active low signals.

1. PR = 0, Q = 1
2. CLR = 0, Q = 0

These two values(always fixed) are independent with the input D and the Clock pulse (CLK).

Working

The ORI input is passed to the PR input of the first flip flop, i.e., FF-0, and it is also passed to the clear input of the remaining three flip flops, i.e., FF-1, FF-2, and FF-3. The pre-set input set to 0 for the first flip flop. So, the output of the first flip flop is one, and the outputs of the remaining flip flops are 0. The output of the first flip flop is used to form the ring in the **ring counter** and referred to as **Pre-set 1**.

ORI	Clk	Q_0	Q_1	Q_2	Q_3
low	X	1	0	0	0
high	low	0	1	0	0
high	low	0	0	1	0
high	low	0	0	0	1
high	low	1	0	0	0

In the above table, the highlighted 1's are **pre-set 1**.

The **Pre-set 1** is generated when

- ORI input set to low, and that time the Clk doesn't care.
-
- When the ORI input set to high, and the low clock pulse signal is passed as the negative clock edge triggered.

A ring forms when the **pre-set 1** is shifted to the next flip-flop at each clock pulse.

So, 4-bit counter, 4 states are possible which are as follows:

1. 1 0 0 0

2. 0 1 0 0
3. 0 0 1 0
4. 0 0 0 1

Types of Ring Counter

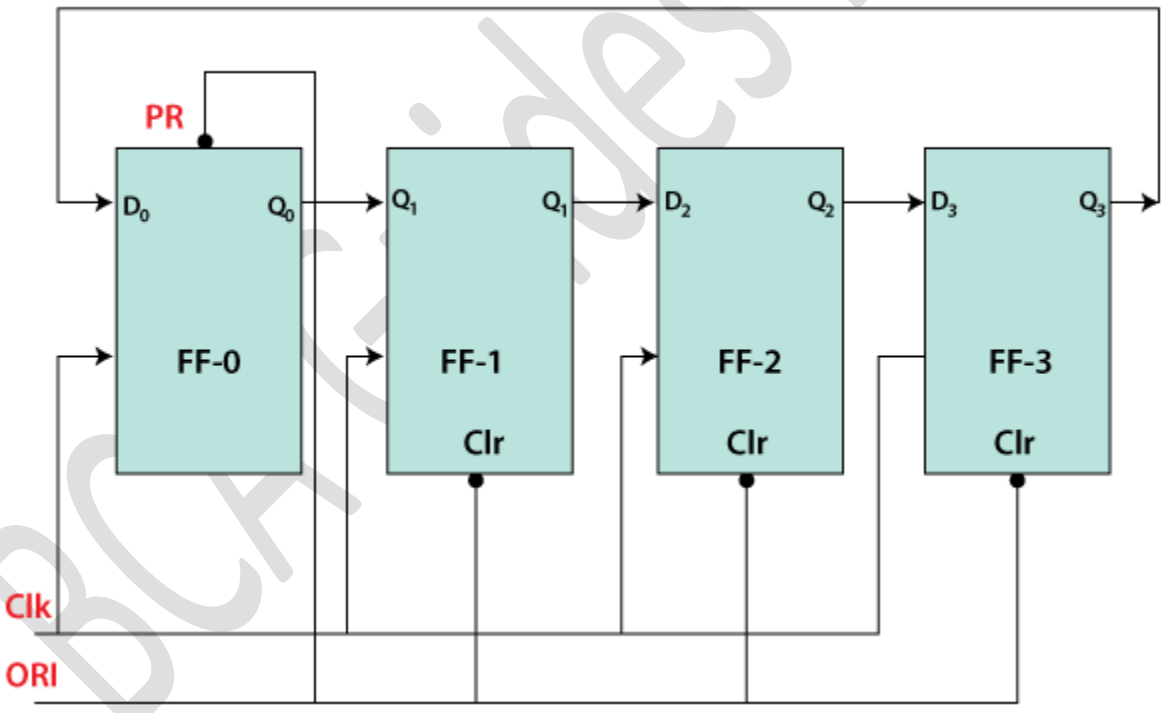
The ring counter is classified into two parts which are as follows:

Straight Ring Counter

The **Straight Ring Counter** refers to as **One hot Counter**. The outcome of the last flip-flop is passed to the first flip-flop as an input. In the ring counter, the ORI input is passed to the PR input for the first flip flop and to the clear input of the remaining flip flops.

Note: The straight ring counter circulates the single 1 (or 0) bit around the ring.

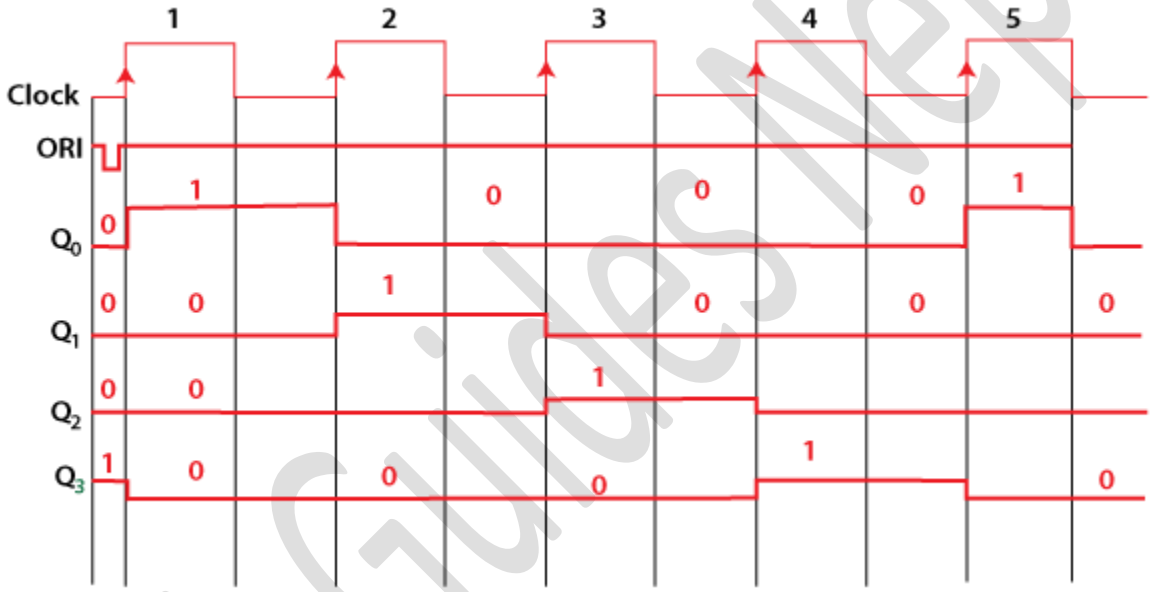
Logic Diagram



Truth Table

ORI	Clk	Q_0	Q_1	Q_2	Q_3
Low	X	1	0	0	0
High	Low	0	1	0	0
High	Low	0	0	1	0
High	Low	0	0	0	1
High	Low	1	0	0	0

Signal Diagram



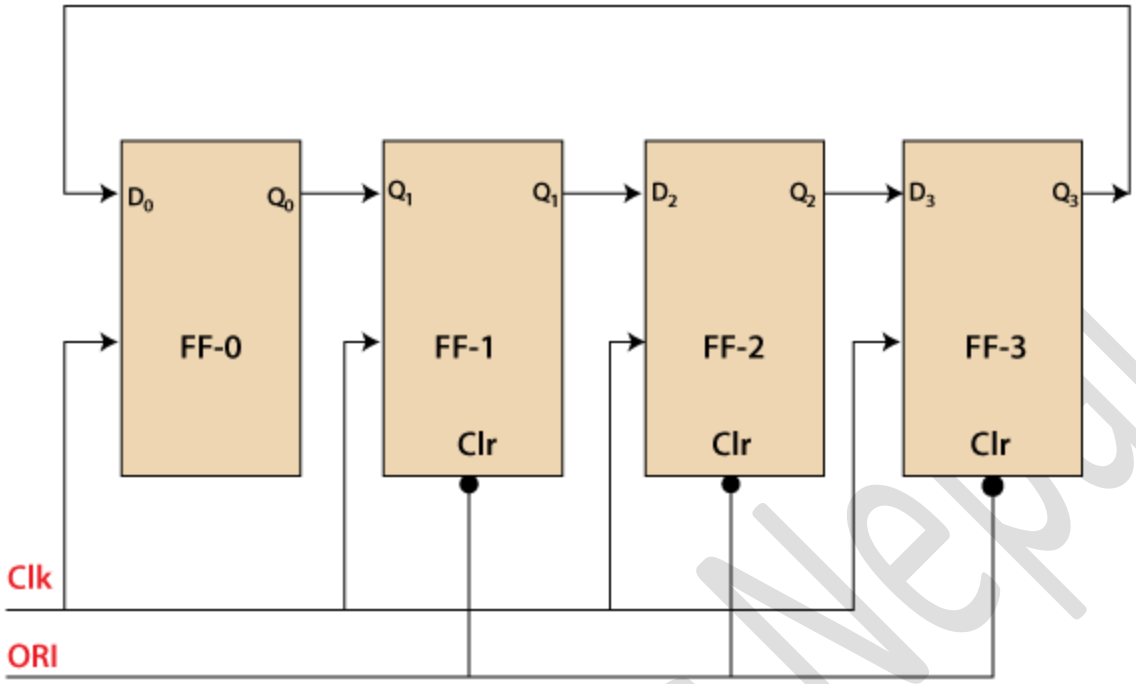
Twisted Ring Counter

The **Twisted Ring Counter** refers to as a **switch-tail ring Counter**. Like the **straight ring counter**, the outcome of the last flip-flop is passed to the first flip-flop as an input. In the twisted ring counter, the ORI input is passed to all the flip flops as **clear** input.

Note: The twisted ring counter circulates a stream of 1's followed by 0 around the ring.

Logic Diagram

BCA Guides Nepal

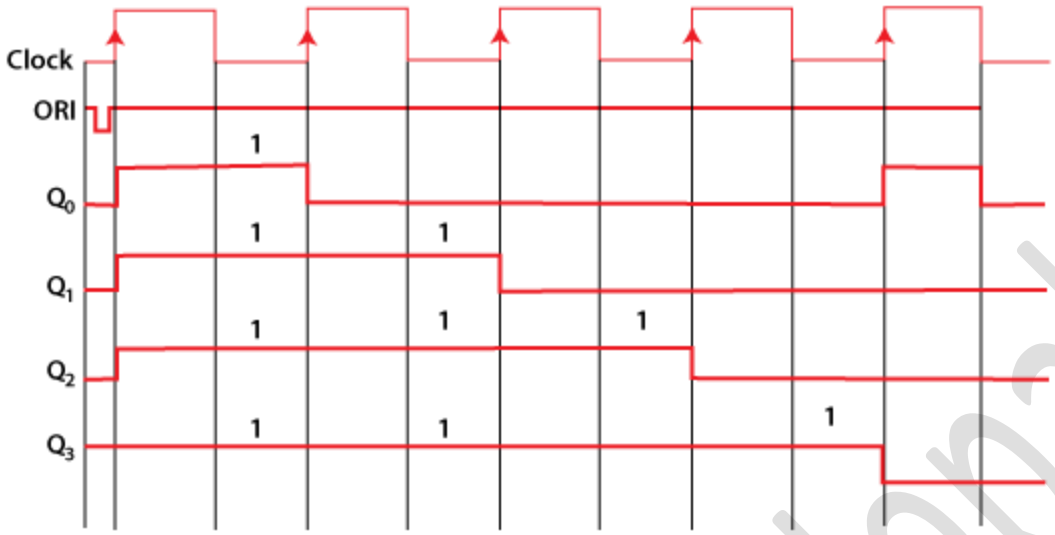


Truth Table

ORI	Clk	Q_0	Q_1	Q_2	Q_3
low	X	0	0	0	0
high	high	1	0	0	0
high	high	1	1	0	0
high	high	1	1	1	0
high	high	1	1	1	1
high	high	0	1	1	1
high	high	0	0	1	1
high	high	0	0	0	1

Signal Diagram

BCA Guides Nepal



4.2.3 Modulus 10 Counter: Circuit and State Diagram and Timing Waveforms

4.2.4 Modulus Counters (5, 7, 11) and Design Principle, Circuit and State Diagram

Modulus Counters, or simply *MOD counters*, are defined based on the number of states that the counter will sequence through before returning back to its original value. For example, a 2-bit counter that counts from 00_2 to 11_2 in binary, that is 0 to 3 in decimal, has a modulus value of 4 (00 → 1 → 10 → 11, and return back to 00) so would therefore be called a modulo-4, or mod-4, counter. Note also that it has taken four clock pulses to get from 00 to 11.

As in this simple example there are only two bits, (n = 2) then the maximum number of possible output states (maximum modulus) for the counter is: $2^n = 2^2$ or 4. However, counters can be designed to count to any number of 2^n states in their sequence by cascading together multiple counting stages to produce a single modulus or MOD-N counter.

Therefore, a “Mod-N” counter will require “N” number of flip-flops connected together to count a single data bit while providing 2^n different output states, (n is the number of bits). Note that N is always a whole integer value.

The we can see that MOD counters have a modulus value that is an integral power of 2, that is, 2, 4, 8, 16 and so on to produce an n-bit counter

BCA Guides Nepal

depending on the number of flip-flops used, and how they are connected, determining the type and modulus of the counter.

4.2.5 Synchronous Design of Above Counters, Circuit Diagrams and State Diagrams

4.3 Application of Counters

Applications of counters

- Frequency counters.
- Digital clocks.
- Analog to digital convertors.
- With some changes in their design, counters can be used as frequency divider circuits. ...
- In time measurement. ...
- We can design digital triangular wave generator by using counters.

4.3.1 Digital Watch

What is the role of clock in digital circuits?

Digital circuits rely on clock signals to know when and how to execute the functions that are programmed. If the clock in a design is like the heart of an animal, then clock signals are **the heartbeats that keep the system in motion**.J

4.3.2 Frequency Counter

A **frequency counter** is an electronic instrument, or component of one, that is used for measuring frequency. Frequency counters usually measure the number of cycles of oscillation, or pulses per second in a periodic electronic signal. Such an instrument is sometimes referred to as a cymometer, particularly one of Chinese manufacture.

4.4 Registers

Registers

A **Register** is a collection of flip flops. A flip flop is used to store single bit digital data. For storing a large number of bits, the storage capacity is increased by grouping more than one flip flops. If we want to store an n-bit word, we have to use an n-bit register containing n number of flip flops.

The register is used to perform different types of operations. For performing the operations, the CPU use these registers. The faded inputs to the system will store into the registers. The result returned by the system will store in the registers. There are the following operations which are performed by the registers:

Fetch:

It is used

- To take the instructions given by the users.
- To fetch the instruction stored into the main memory.

Decode:

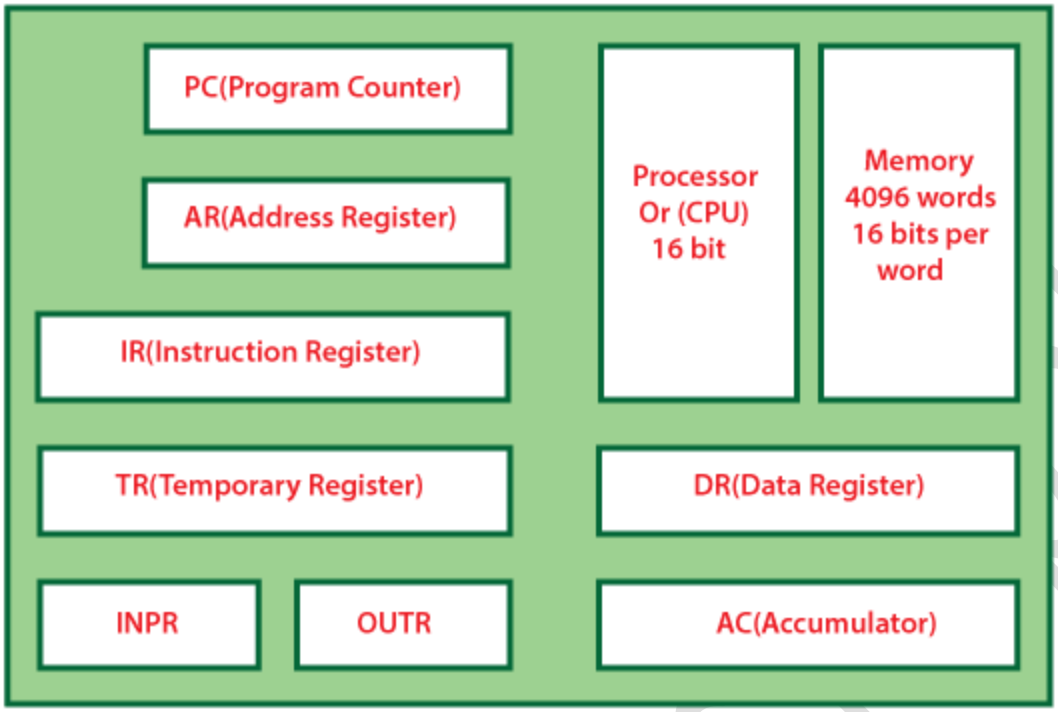
The decode operation is used to interpret the instructions. In decode, the operation performed on the instructions is identified by the CPU. In simple words, the decode operation is used to decode the instructions.

Execute:

The execution operation is used to store the result produced by the CPU into the memory. After storing this result, it is displayed on the user screen.

Types of Registers

There are various types of registers which are as follows:



MAR or Memory Address Register

The MAR is a special type of register that contains the memory address of the data and instruction. The main task of the MAR is to access instruction and data from memory in the execution phase. The MAR stores the address of the memory location where the data is to be read or to be stored by the CPU.

Program Counter

The program counter is also called an instruction address register or instruction pointer. The next memory address of the instruction, which is going to be executed after completing the execution of current instruction is contained in the program counter. In simple words, the program counter contains the memory address of the location of the next instruction.

Accumulator Register

The CPU mostly uses an accumulator register. The accumulator register is used to store the system result. All the results will be stored in the accumulator register when the CPU produces some results after processing.

MDR or Memory Data Register

Memory Data Register is a part of the computer's control unit. It contains the data that we want to store in the computer storage or the data fetched from the computer storage. The MDR works as a buffer that contains anything for which the processor is ready to use it. The MDR contains the copied data of the memory for the processor. Firstly the MDR holds the information, and then it goes to the decoder.

The data which is to be read out or written into the address location is contained in the **Memory Data Register**.

The data is written in one direction when it is fetched from memory and placed into the MDR. In write instruction, the data place into the MDR from another CPU register. This CPU register writes the data into the memory. Half of the minimal interface between the computer storage and the microprogram is the memory data address register, and the other half is the memory data register.

Index Register

The **Index Register** is the hardware element that holds the number. The number adds to the computer instruction's address to create an effective address. In CPU, the index register is a processor register used to modify the operand address during the running program.

Memory Buffer Register

Memory Buffer Register is mostly called MBR. The MBR contains the Metadata of the data and instruction written in or read from memory. In simple words, it adds is used to store the upcoming data/instruction from the memory and going to memory.

Data Register

The data register is used to temporarily store the data. This data transmits to or from a peripheral device.

4.4.1 Serial in Parallel out Register

4.4.2 Serial in Serial out Register

4.4.3 Parallel in Serial out Register

4.4.4 Parallel in Parallel out Register

4.4.5 Right Shift, Left Shift Register

Shift Register

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

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

The shift register is classified into the following types:

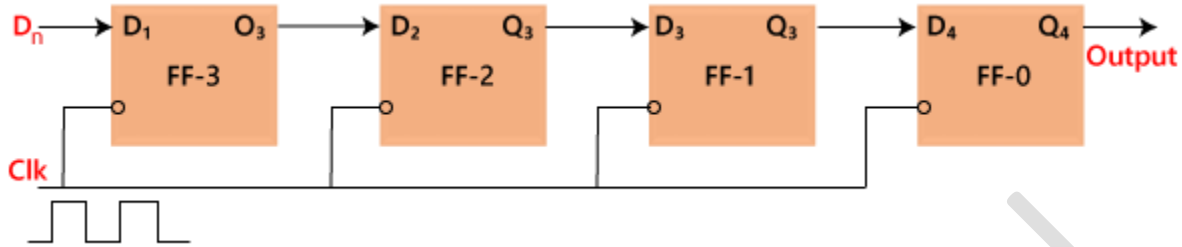
- Serial In Serial Out
- Serial In Parallel Out
- Parallel In Serial Out
- Parallel In Parallel Out
- Bi-directional Shift Register
- Universal Shift Register

Serial IN Serial OUT

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

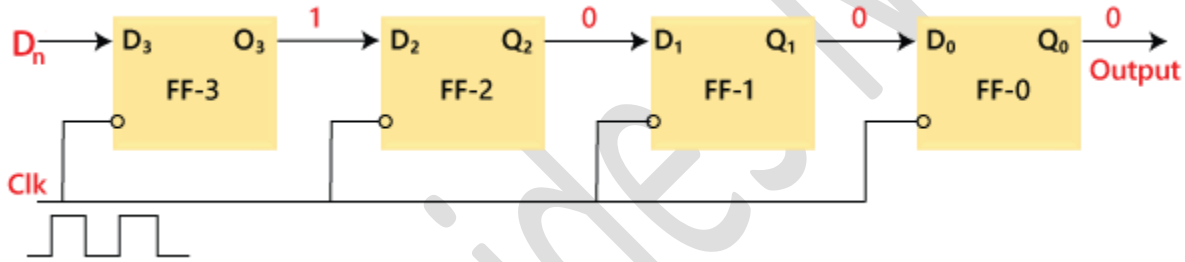
Initially, all the flip-flops are set in "reset" condition i.e. $Y_3 = Y_2 = Y_1 = Y_0 = 0$. If we pass the binary number 1111, the LSB bit of the number is applied first to the Din bit. The D3 input of the third flip flop, i.e., FF-3, is directly connected to the serial data input D3. The output Y_3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

Block Diagram:

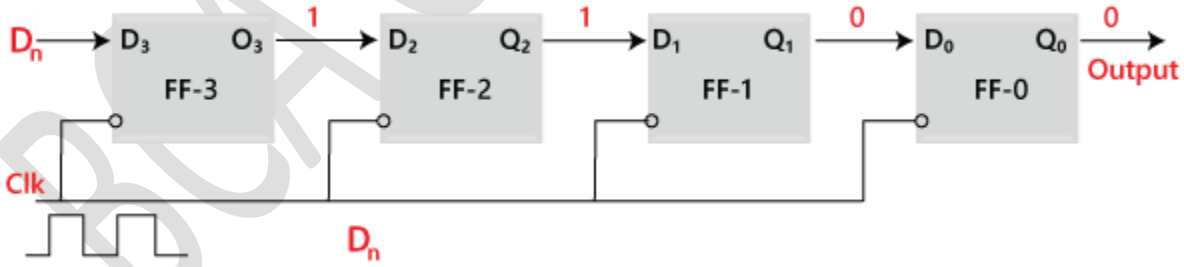


Operation

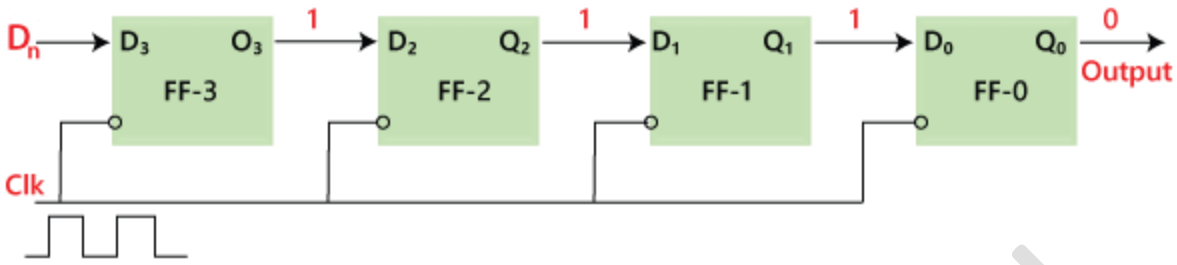
When the clock signal application is disabled, the outputs Y_3 Y_2 Y_1 Y_0 = 0000. The LSB bit of the number is passed to the data input D_{in}, i.e., D_3. We will apply the clock, and this time the value of D_3 is 1. The first flip flop, i.e., FF-3, is set, and the word is stored in the register at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



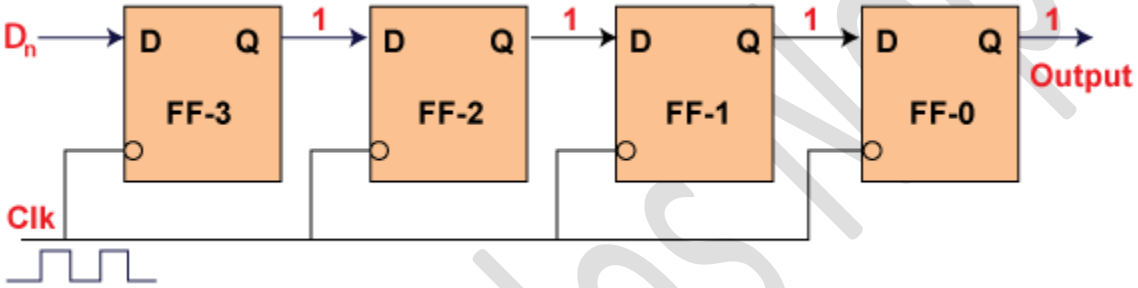
The next bit of the binary number, i.e., 1, is passed to the data input D_2. The second flip flop, i.e., FF-2, is set, and the word is stored when the next negative edge of the clock hits. The stored word is changed to 1100.



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



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



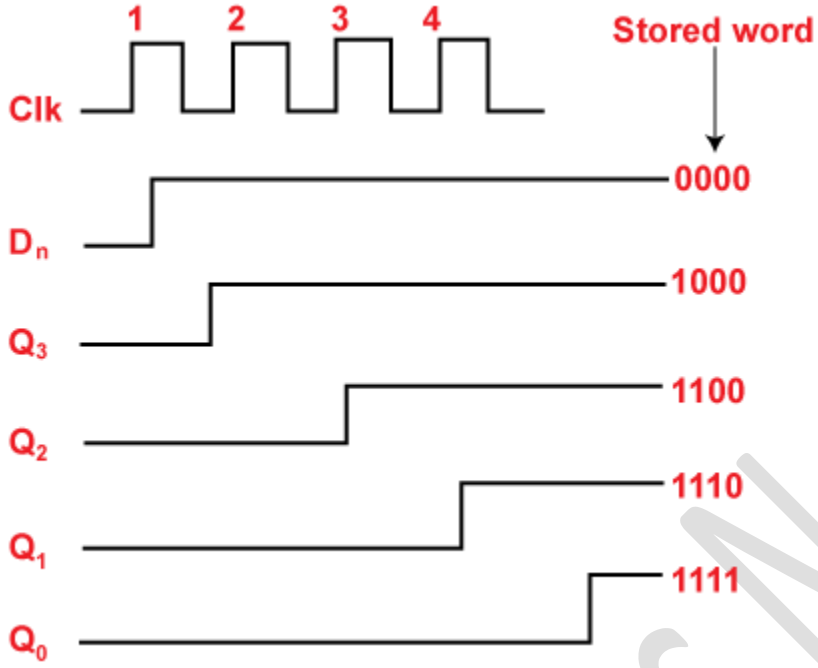
Truth Table

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

→ Direction of data travel

BCA Guides Nepal

Waveforms

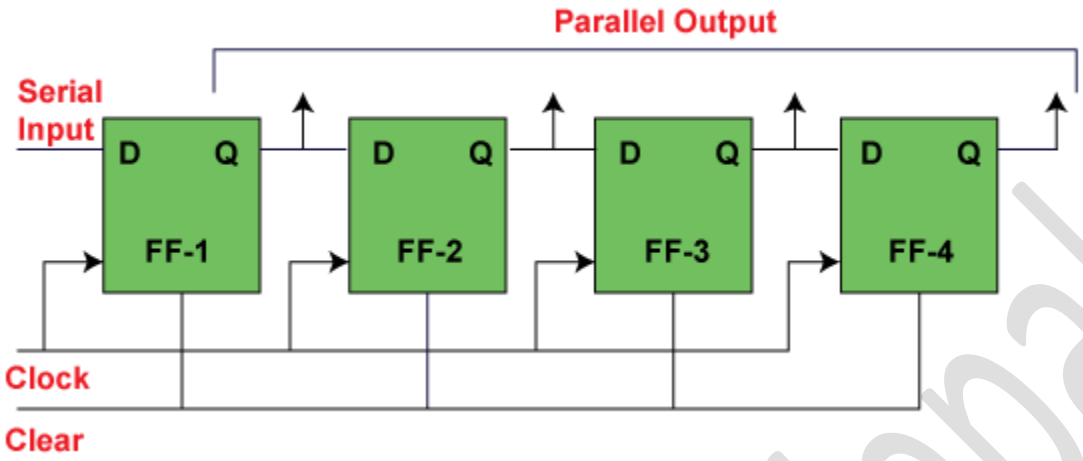


Serial IN Parallel OUT

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

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

Block Diagram



Parallel IN Serial OUT

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

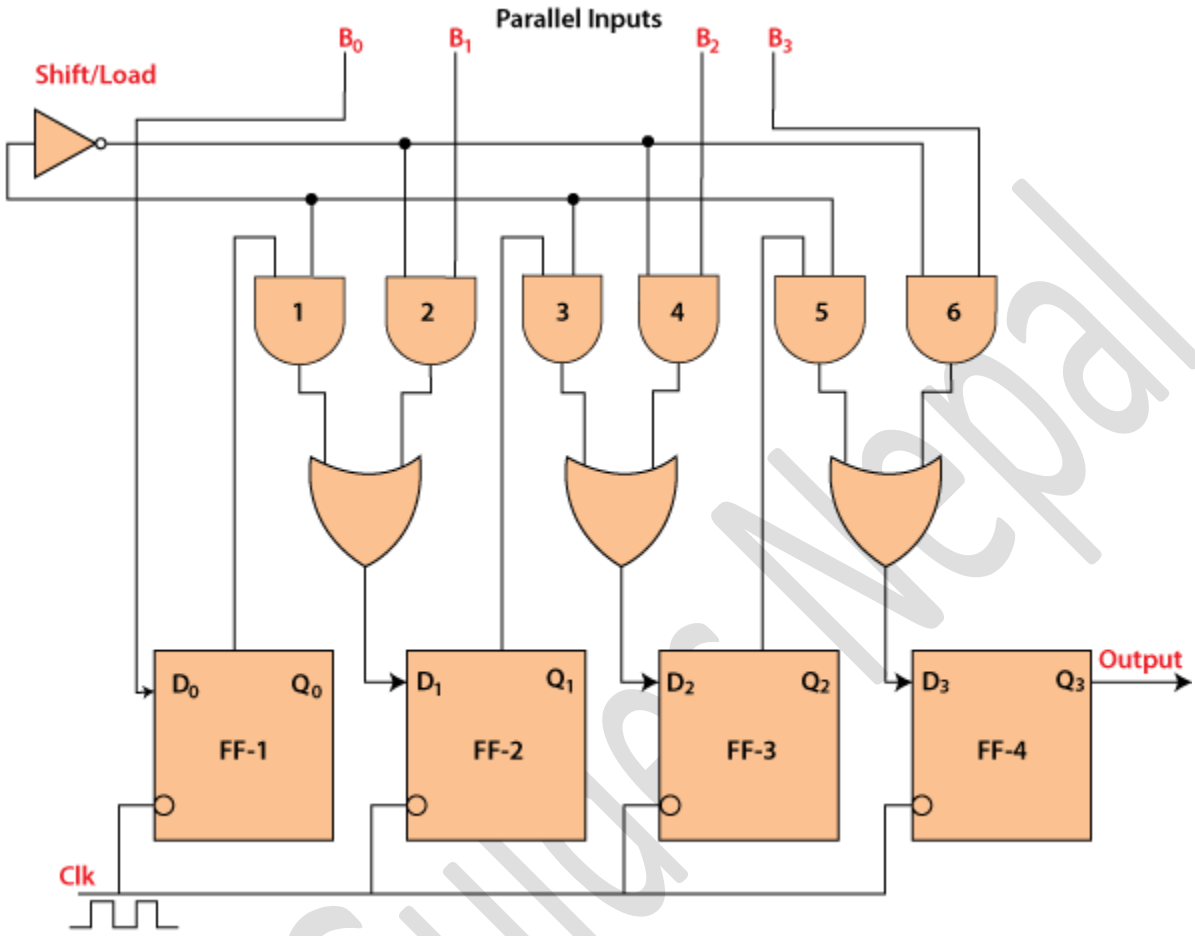
Load mode

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

Shift mode

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

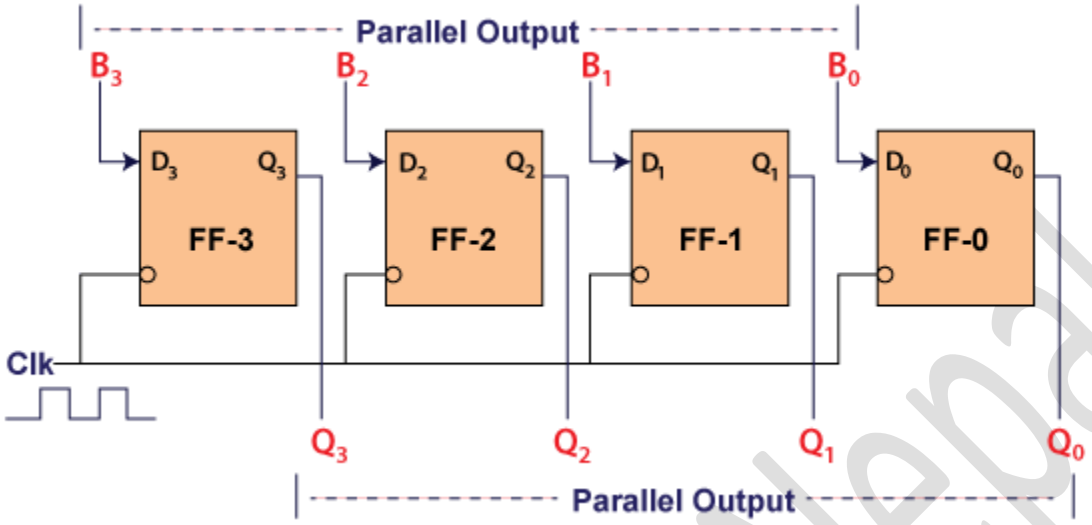
Block Diagram



Parallel IN Parallel OUT

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

Block Diagram



Bidirectional Shift Register

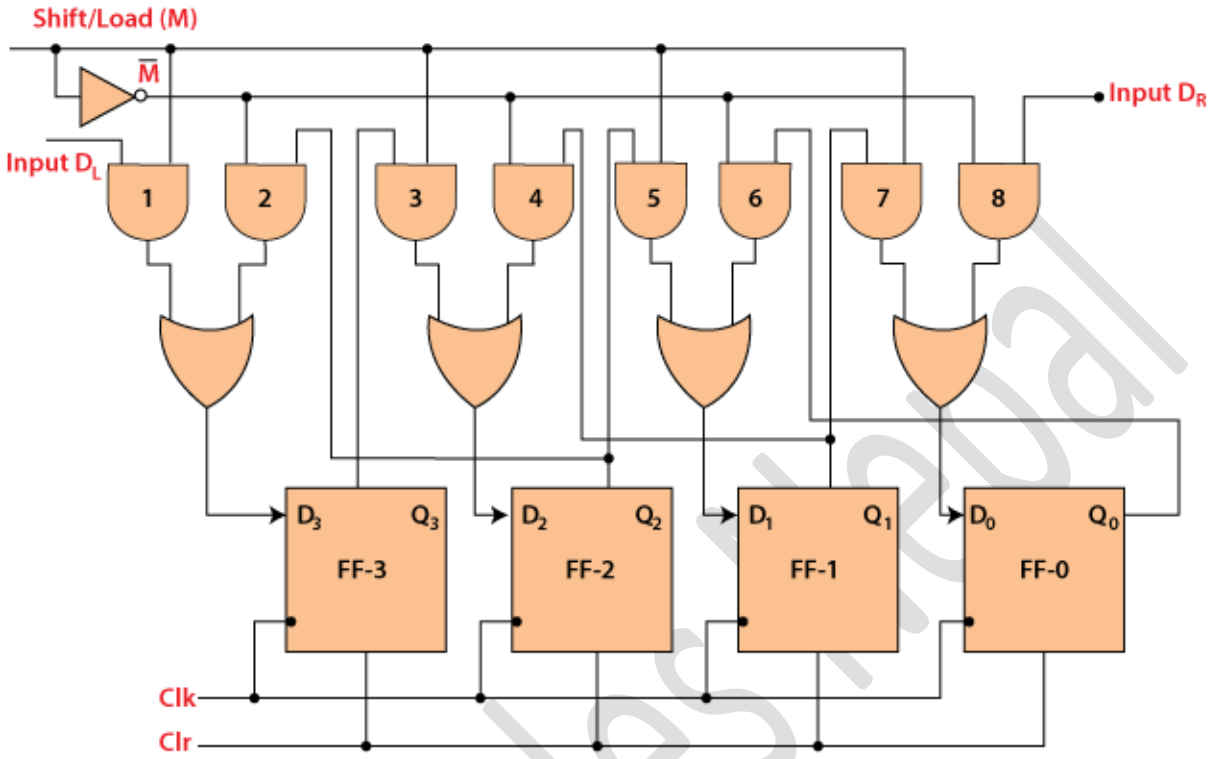
The binary number after shifting each bit of the number to the left by one position will be equivalent to the number produced by multiplying the original number by 2. In the same way, the binary number after shifting each bit of the number to the right by one position will be equivalent to the number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

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

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

BCA Guides Nepal

Block Diagram



Operations

1) Shift right operation(M=1)

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

2) Shift left operation(M=0)

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

Latches

A **Latch** is a special type of logical circuit. The latches have **low** and **high** two stable states. Due to these states, latches also refer to as **bistable-multivibrators**. A latch is a storage device that holds the data using the feedback lane. The latch stores 1 -bit until the device set to 1. The latch changes the stored data and constantly trials the inputs when the enable input set to 1.

Based on the enable signal, the circuit works in two states. When the enable input is high, then both the inputs are low, and when the enable input is low, both the inputs are high.

Types of Latches

There are various types of latches used in digital circuits which are as follows:

- **SR Latch**
- **Gated S-R Latch**
- **D latch**
- **Gated D Latch**
- **JK Latch**
- **T Latch.**

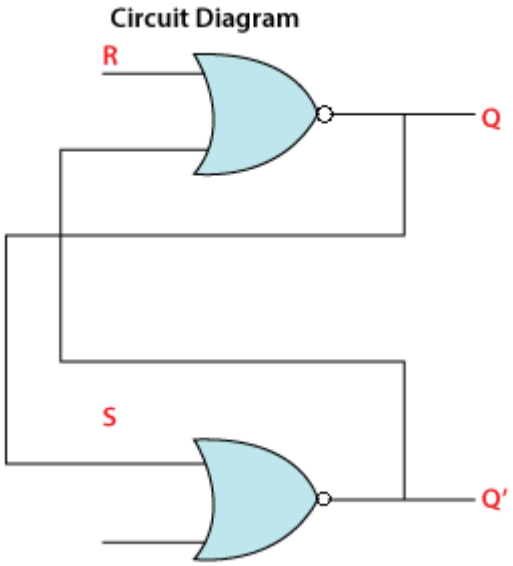
SR Latch

The SR latch is a special type of asynchronous device which works separately for control signals. It depends on the S-states and R-inputs. The SR latch design by connecting two **NOR** gates with a cross loop connection. The SR latch can also be designed using the NAND gate. Below are the circuit diagram and the truth table of the SR latch.

Truth Table

S	R	Q	Q'
0	0	latch	Latch
0	1	0	1

1	0	1	0
1	1	0	0



Circuit Diagram

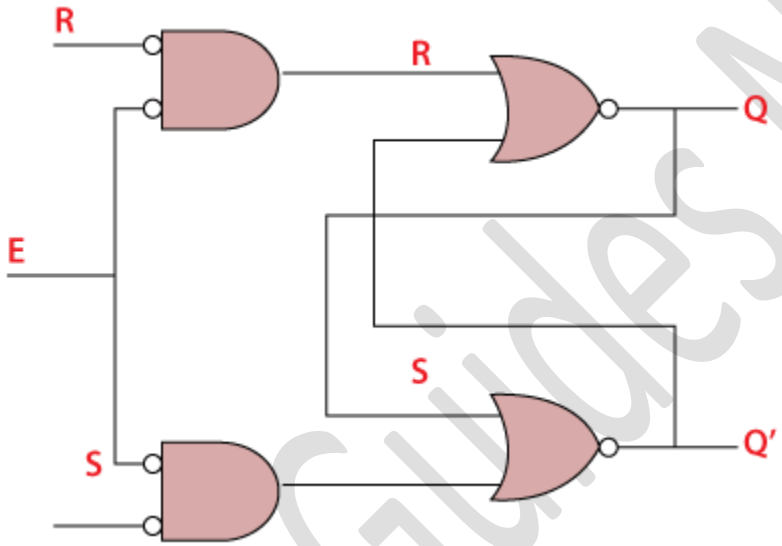
Gated SR Latch

A **Gated SR Latch** is a special type of **SR Latch** having three inputs, i.e., Set, Reset, and Enable. The enable input must be active for the SET and RESET inputs to be effective.The **ENABLE** input of gated SR Latch enables the operation of the SET and RESET inputs.This **ENABLE** input connects with a switch. The Set-Reset inputs are enabled when this switch is on. Otherwise, all the changes are ignored in the set and reset inputs. Below are the circuit diagram and the truth table of the Gated SR latch.

Truth Table

E	S	R	Q	Q'
0	0	0	Latch	Latch
0	0	1	Latch	Latch
0	1	0	Latch	Latch
0	1	1	Latch	Latch
1	0	0	Latch	Latch
1	0	1	0	1
1	1	0	1	0
1	1	1	0	0

Circuit Diagram



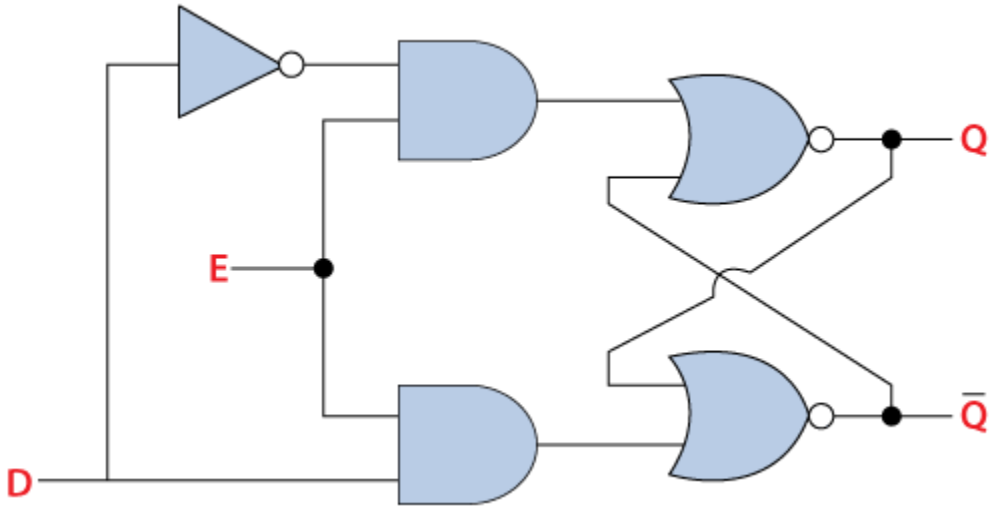
D Latch

The **D latch** is the same as **D flip flop**. The only difference between these two is **the ENABLE** input. The output of the latch is the same as the input passed to the **Data** input when the **ENABLE** input set to 1. At that time, the latch is open, and the path is transparent from input to output. If the **ENABLE** input is set to 0, the D latch's output is the last value of the latch, i.e., independent from the input D, and the latch is closed. Below are the circuit diagram and the truth table of the D latch.

Truth Table

E	D	Q	Q'
0	0	latch	latch
0	1	latch	latch
1	0	0	1
1	1	1	0

Circuit Diagram

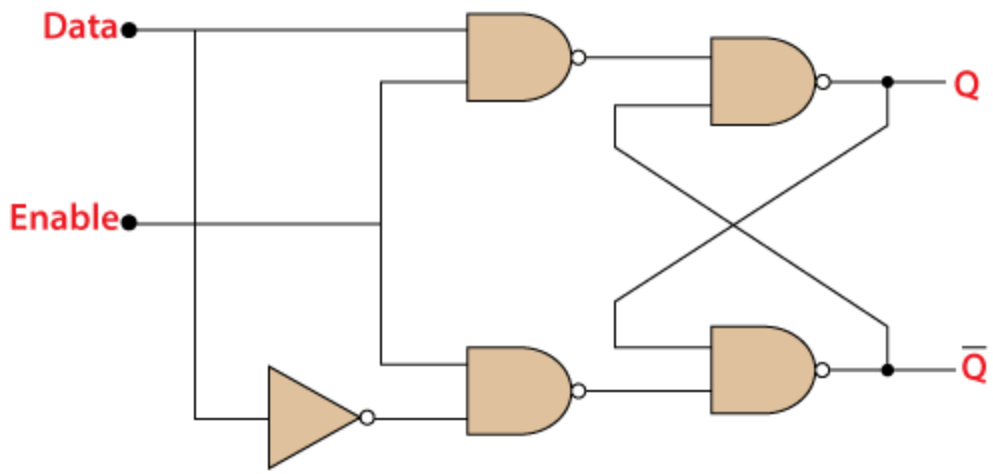


Gated D Latch

The Gated D Latch is another special type of gated latch having two inputs, i.e., DATA and ENABLE. When the enable input set to 1, the input is the same as the Data input. Otherwise, there is no change in output.

We can design the gated D latch by using gated SR latch. The set and reset inputs are connected together using an inverter. By doing this, the outputs will be opposite to each other. Below is the circuit diagram of the Gated D latch.

Circuit Diagram



JK Latch

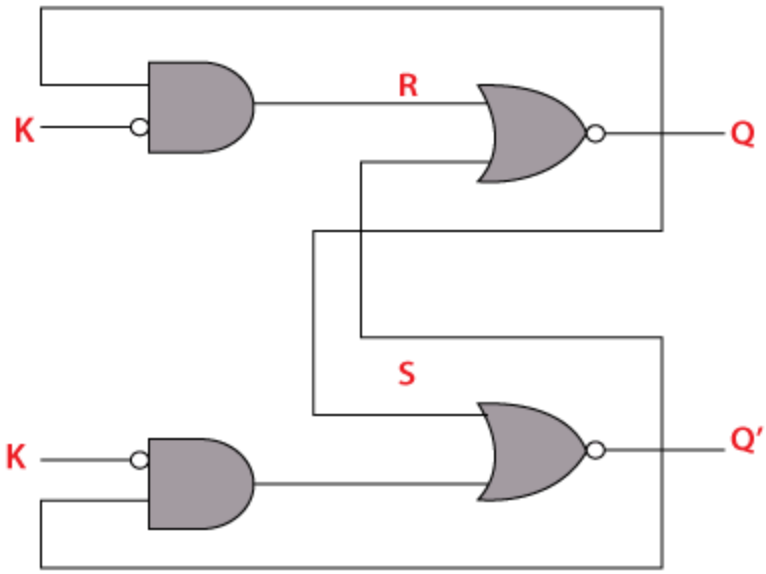
The JK Latch is the same as the SR Latch. In JK latch, the unclear states are removed, and the output is toggled when the JK inputs are high. The only difference between SR latch JK latches is that there is no output feedback towards the inputs in the SR latch, but it is present in the JK latch. The circuit diagram and truth table of the JK latch are as follows:

Truth Table

J	K	Q_{next}	Comment
0	0	Q	No change
0	1	0	Reset
1	0	1	Set
1	1	Q'	Toggle

Circuit Diagram

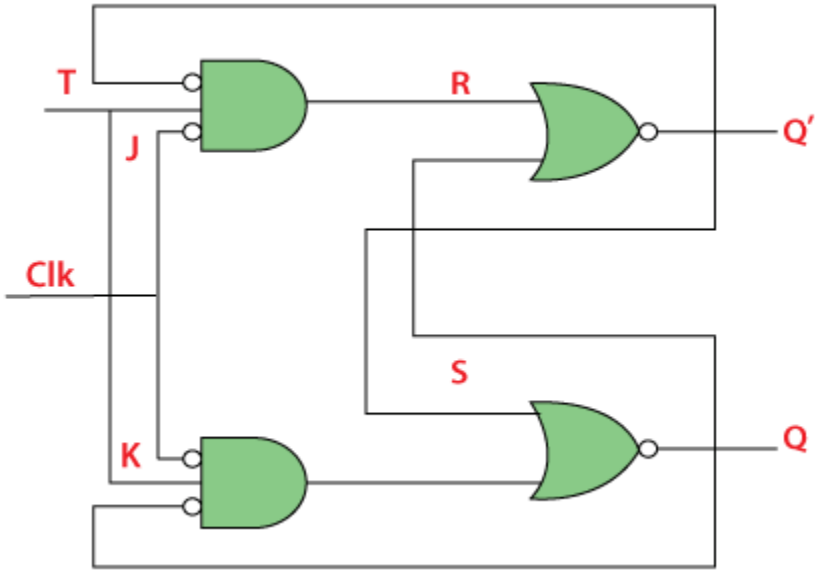
BCA Guides Nepal



T Latch

The T latch forms by shorting the JK latch inputs. The output of the T latch toggle when the input set to 1 or high. Below is the circuit diagram of the T latch.

Circuit Diagram



BCA Guides Nepal

Unit 5 Sequential Logic Design

5.1 Basic Models of Sequential Machines • Concept of State • State Diagram 5.2 State Reduction through Partitioning and Implementation of Synchronous Sequential Circuits 5.3 Use of Flip-Flops it Realizing the Models 5.4 Counter Design

BCA Guides Nepal