BASICS OF DIGITAL ELECTRONICS

Binary Numbers system:-

The binary numbers system is a radix-2 no. system with '0' and '1' as the two independent digits. All large binary numbers are represented in terms of '0' and '1'. The procedure for writing higher order binary numbers after '1' is similar to the one explained in the case of decimal number system. For example, the first 16 numbers in the binary number system would be 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101, 1110 and 1111. The next number after 1111 is 10000, which is the lowest binary number with five digits. This also proves the point made earlier that a maximum of only 16 (=2^4) numbers could be written with four digits. Starting from the binary point, the place values of different digits in a mixed binary number are 2^0, 2^1, 2^2 and so on (for the integer part) and 2^{-1}, 2^{-2}, 2^{-3} and so on (for the fractional part).

Octal Numbers system:-

Octal number system has a base of eight and uses the numbers from 0 to 7. The octal numbers, in the number system, are usually represented by binary numbers when they are grouped in pairs of three. For example, 128 is expressed as 001010_2, where 1 is equivalent to 001 and 2 is equivalent to 010.

Octal symbol - 0, 1, 2, 3, 4, 5, 6 and 7

Apart from octal numbers system, there are other number systems in Maths, such as:

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

Octal Numbers system Table

We use only 3 bits to represent octal no.s Each group will have a distinct value bet^n 000 and 111.

Octal Digital Value	Binary Equivalent
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Octal Number	Equivalent Binary Number
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111

3] Hexadecimal number system :-

The hexadecimal number system is a type of number system that has a base value equal to 16. It is also pronounced sometimes as 'hex'. Hexadecimal numbers are represented by only 16 symbols. These symbols or values are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. Each digit represents a decimal value. For example, D is equal to base - 10 13.

Hexadecimal number systems can be converted to other number systems such as binary number (base-2), octal number (base-8) and decimal number systems (base-10). ~~The concept of the number system is widely explained in the~~

Decimal Numbers	4 bit Binary no.
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101

1.2 Binary addition

Binary addition is one of the binary operation. To recall, the term 'Binary operation" represent the basic operations of mathematics that are performed on two operands. Basic arithmetic operations like addition, subtraction, multiplication, and division, play an important role in mathematics.

Rules of Binary Addition:

Binary addition is much easier than the decimal addition when you remember the following tricks or rules. Using these rules, any binary number can be easily added. The four rules of binary addition are:

- 0+0=0
- 0+1=1
- 1+0=1
- 1+1=10

Ex: (i)

$$\begin{array}{r} {\scriptstyle 1\ \ \ \ } \\ 101 \\ +\ 101 \\ \hline 1010 \end{array}$$

So, the resultant of the addition operation is 1010.

(ii)

$$\begin{array}{r} {\scriptstyle 1\ \ \ \ \ } \\ 10011 \\ (+)\ 11101 \\ \hline 101110 \end{array}$$

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	0 (where 1 is carried over)

Binary Subtraction

Binary subtraction is one of the four binary operations, where we perform the subtraction method for two binary numbers (comprising only two digits, 0 and 1). This operation is similar to the basic arithmetic subtraction performed on decimal numbers in Maths. Hence, when we subtract 1 from 0, we need to borrow 1 from the next higher order digit, to reduce the digit by 1 and the remainder left here is also 1.

Binary subtraction :-

$0 - 0 = 0$

$1 - 0 = 1$

$1 - 1 = 0$

$0 - 1 = 1$ (Borrow 1)

(i) Ex:-

$$\begin{array}{r} 1010 \\ (-)\ 101 \\ \hline 0101 \end{array}$$

(ii)

$$\begin{array}{r} 00011010 \\ (-)\ 00011100 \\ \hline 00001110 \end{array}$$

Binary Multiplication :-

Binary multiplication is one of the four binary arithmetic. The other three fundamental operations are addition, subtraction and division. In the case of a binary operation, we deal with only two digits, i.e. 0 and 1. The operation performed while finding the binary product is similar to the conventional multiplication method. The four major steps in binary digit multiplication are:

$0 \times 0 = 0$ $\quad$ $1 \times 0 = 0$

$0 \times 1 = 0$ $\quad$ $1 \times 1 = 1$

Example:- ~~1010 x 101~~

1011.01 x 110.1

```
  1011.01
    110.1
  -------
  101101
 00000000
 -------
 0101101          ..... first intermediate sum
1011001
-------
1110000 1 ..... Second intermediate sum
101101
-------
1001001.001 ..... final sum
```

Binary Division RULE

The binary division is much easier than the decimal division when you remember the following division rules. The main rules of the binary division include:

1 ÷ 1 = 1

1 ÷ 0 = Meaning less

0 ÷ 1 = 0

0 ÷ 0 = Meaning less

Similar to the decimal numbers system, the binary division is similar, which follows the four-step process.

- Divide
- Subtract
- Multiply
- Bring down

Binary division follows the long division method to find the resultant in an easy way.

Comparison with Decimal value

$(01111100)_2 = (1111100)_2 = 124_{10}$

$(0010)_2 = (10)_2 = 2_{10}$

You will get the resultant value as 62 when you divide 124 by 2.

So the binary equivalent of 62 is $(111110)_2$

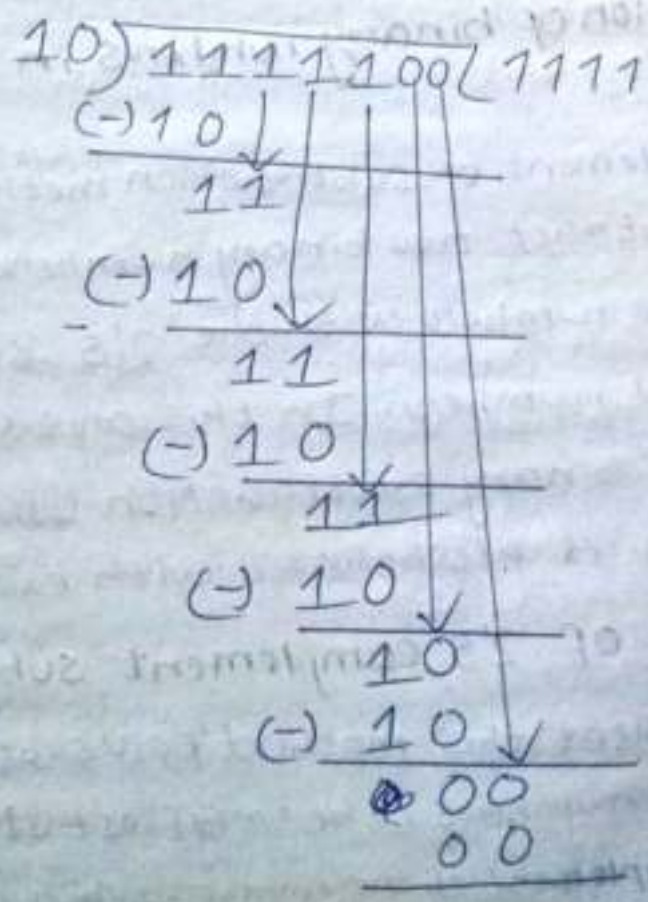
$= 62_{10}$

Both the binary and the decimal system produce the same result.

Example : 1 (Binary Division Examples)

Solve 01111100 ÷ 0010

A: Here the dividend is 01111100, and the divison is 0010



1's complement and 2's complement numbers for a binary number.

1's complement of a binary number is another binary number obtained by toggling all bits in it, i.e., transforming the 0 bit to 1 and 1 bit to 0.

Examples:-

1's complement of "0111" is "1000"

1's complement of "1100" is "0011"

2's complement

2's complement of a binary number is 1, added to the 1's complement of the binary number.

Ex:- 2's complement of "0111" is "1001"

2's complement of "1100" is "0100"

1.4 Subtraction of binary numbers in 2's complement method.

A:- 2's complement ~~of~~ subtraction method is a way to subtract two binary numbers by actually adding one number with the 2's complement of another number. In this article, the method of binary subtraction using 2's complement is elaborated with examples

Method of 2's complement subtraction

To implement this method for subtracting two binary numbers, the very first step is find the 2's complement of the number which is to be subtracted from another number. TO get the 2's

complement, first of all 1's complement is find and then 1 is added to this. The addition is the required 2's complement.

Suppose, we need to find the 2's complement of binary number 10010. First, find 1's complement To find this, replace all 1 to 0 and all 0 to 1. Therefore, 1's complement of 10010 will be 01101. Now, add 1 to this as shown below

```
  01101
+     1
-------
  01110  <-- 2's complement
```

Ex:-1

Subtract $(1010)_2$ from $(1111)_2$ using 2's complement method.

A:- 2's complement of $(1010)_2$ is $(0110)_2$.

Step:-2:- Add $(0110)_2$ to $(1111)_2$. This is shown below.

```
    1 1 1 1
+   0 1 1 0
-----------
1 0 1 0 1   <-- Omit this carry
```

0 1 0 1 Answer

1.6 Importance of parity Bit.

A parity bit is a bit, with a value of 0 or 1 that is added to a block of data for error detection purposes. It gives the data either an odd or even parity, which is used to validate the integrity of the data.

parity bits are often used in data transmission to ensure that data is not corrupted during the transfer process. For example, every 7 bits of data may include a parity bit (for a total of 8 bits), or one byte). If the data transmission protocol is set to an odd parity, each data packet must have an odd parity. If it is set to even, each packet must have an even parity. If a packet is received with the wrong parity, an error will be produced and the data will need to be retransmitted.

The parity bit for each data packet is computed before the data is transmitted. Below are examples of how a parity bit would be computed using both odd & even parity settings.

Odd parity:	Even parity
• Initial value 1010101 (four 1s)	• Initial value 1010101 (four 1s)
• parity bit added: 1	• parity bit added: 0
• Transmitted value: 10101011	• Transmitted value: 10101010
• Result: odd parity (five 1s)	• Result: Even parity (four 1s)

The value of the parity bit depends on the initial parity of the data. For example, the binary value 10000000 has an odd parity. Therefore, a '0' would be added to keep the parity odd and a '1' would be added to give the value an even parity.

While parity checking is a useful way validating data, it is not a foolproof method. For instance, the values 1010 and 1001 have the same parity. Therefore, if the value 1010 is transmitted and 1001 is received, no error will be detected. This means parity checks are not 100% reliable when validating data. Still, it is unlikely that more than one bit will be incorrect in a small packet of data. Therefore, parity checks are most reliable when using small packet sizes.

1.6 Digital Circuit :-

Introduction to logic :-

A logic circuit is one the behaves like a voltage controlled switch i.e. two position devices with ON and OFF status.

(ii) This is turned a binary device in which ON state is represented by one and off by zero.

LOGIC GATES :-

(i) Logic gates are the basic logic circuit.

(ii) These are the fundametal building blocks from which all other logic circuits and digital systems are concentred.

(iii) Logic gates are basically three types.

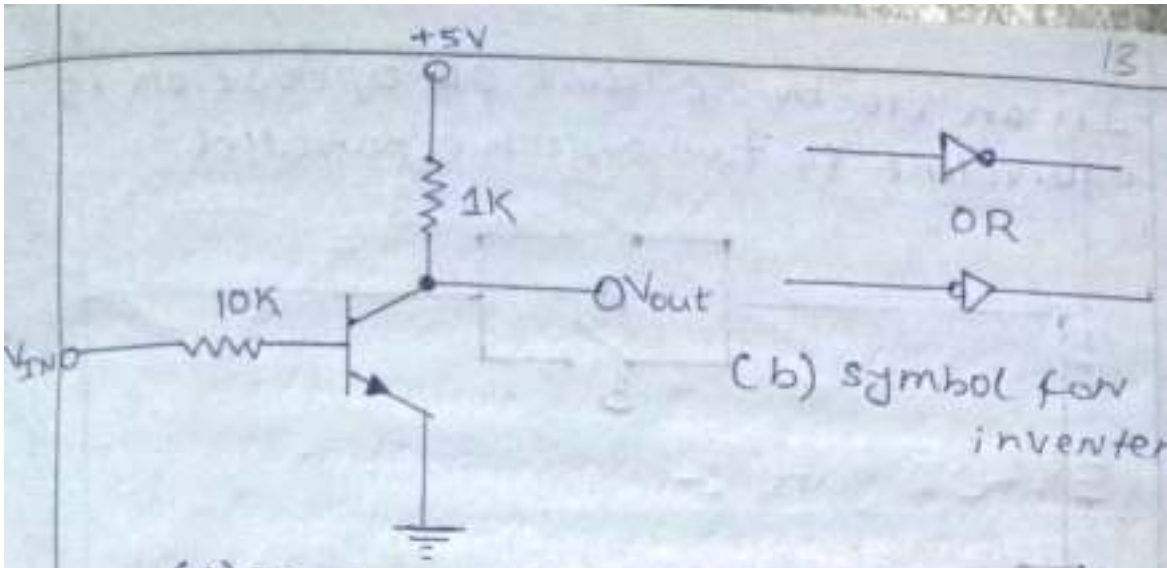
* Inverter gate
* AND gate
* OR gate

Inverter gate :-

(i) An inverter is a logic gate with single input and single output.

(ii) The output is always the opposite of the input state.

(iii) An ~~inter~~ inverter is also called a NOT gate because the output is not the same as the input. The output is also sometimes called the complement (opposite) of the input



(a) Transistor Inverter

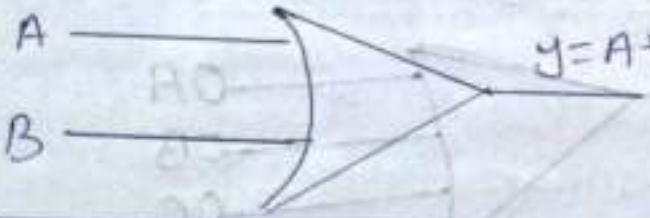
(b) symbol for inverter

Vin	Vout
Low	High
High	Low

Vin	Vout
0	1
1	0

OR GATE

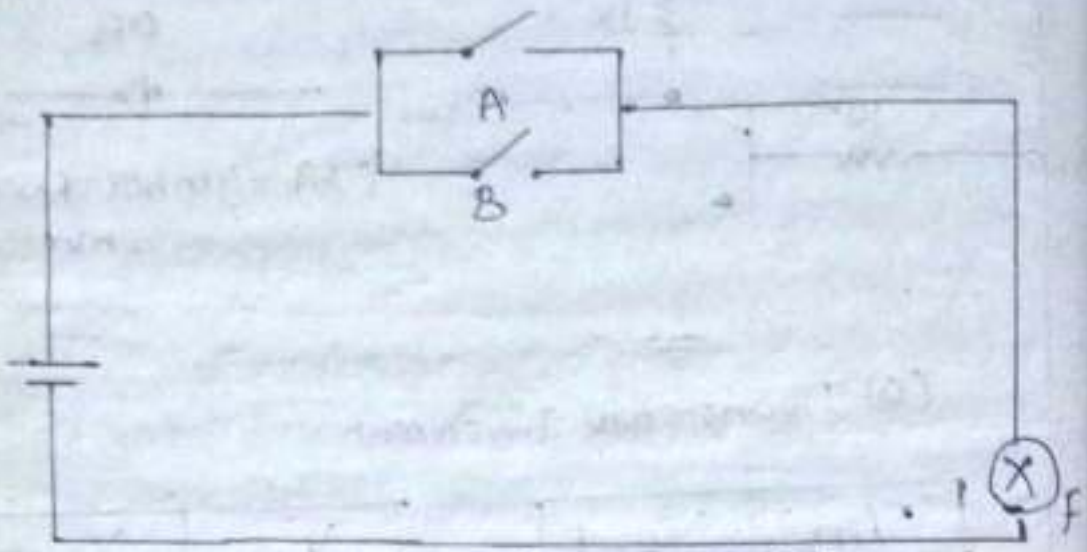
The logic inputs A and B are combined using OR operation (+) to produced the output.



A	B	Y=A+B
Low	Low	High
Low	High	High
High	Low	High
High	High	High

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

In an electric circuit OR operation is equivalent to two switch a parallel.



$F = A + B$

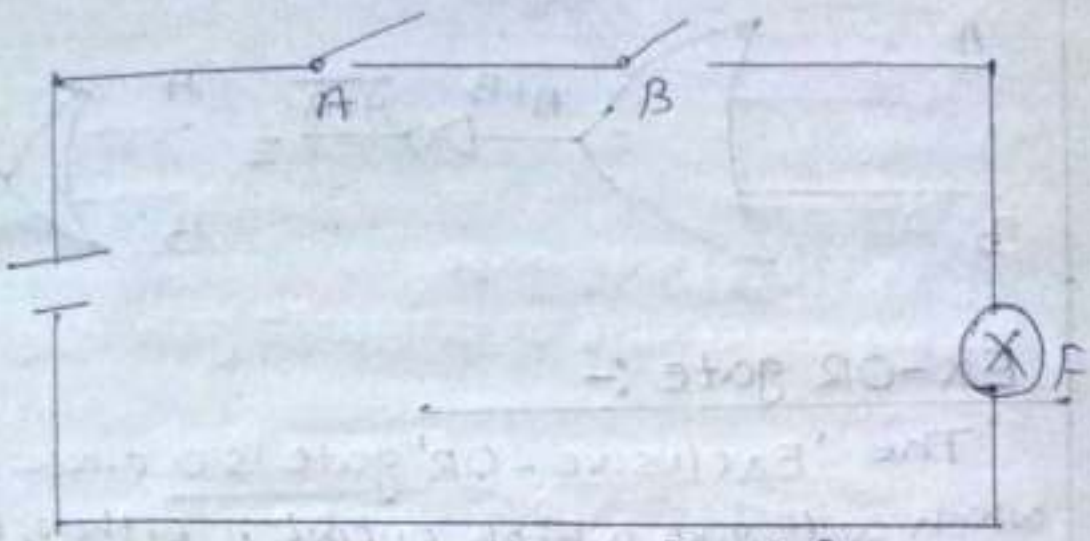
$F = A \text{ OR } B$

AND GATE :-

Two logics inputs A and B are combined using the AND operation (·) to produced the output Y.

A	B	Y
LOW	LOW	LOW
LOW	High	LOW
High	LOW	LOW
High	High	High

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



$F = A \cdot B$

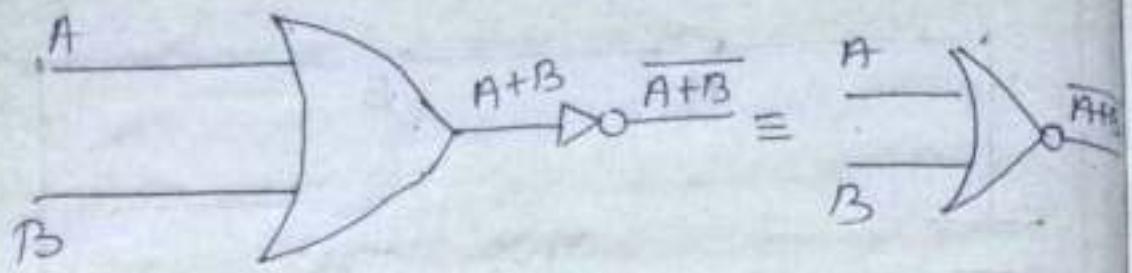
F = A AND B

NOR GATE

The NOR gate ("not OR gate") is a logic gate that produces a high output (1) only if all its inputs are false, and low output (0) otherwise. Hence the NOR gate is the inverse of an OR gate, and its circuit is produced by connecting an OR gate to a NOT gate. Just like an OR gate, a NOR gate may have any number of input probes but only one output probe.

NOR gate means "not an OR gate" hence the output of this logic gate is just the reverse of that of an OR gate.

Inputs		Output
A	B	$X = \overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0



EX-OR gate :-

The 'Exclusive-OR' gate is a circuit which will give a high output if either, but not both of its two inputs are high. An encircled plus sign (⊕) is used to show the EX-OR operation.

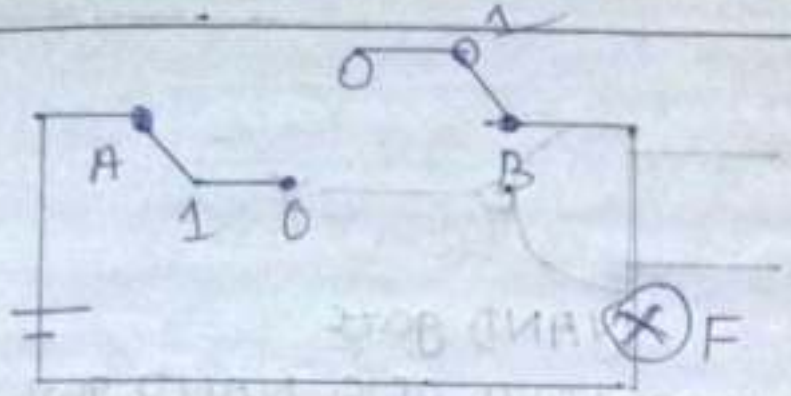
$Y = A \oplus B$

$(A \cdot \overline{B}) + (\overline{A} \cdot B)$

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

EX-OR gate is created from AND, NAND and OR gates. The output is high only when both the inputs are different.

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



$$F = A \oplus B \text{ OR}$$
$$F = (A \cdot \overline{B}) + (\overline{A} \cdot B)$$

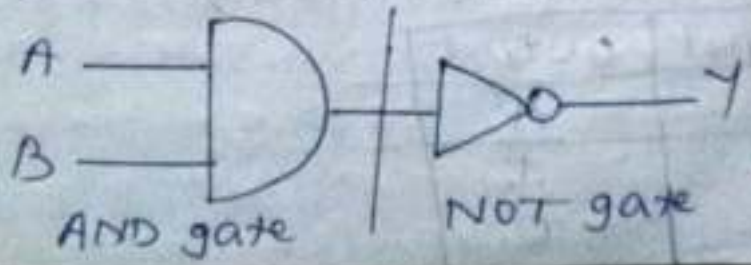
NAND GATE

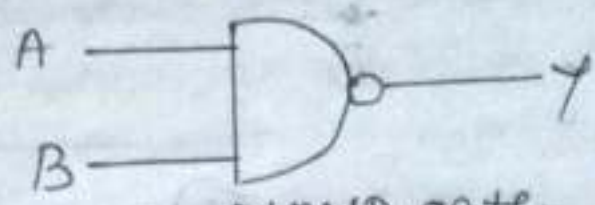
~~NAND gate~~

The NAND gate ON "NOT AND" gate is the combination of two basic logic gates, the AND gate and the NOT gate connected in series. The NAND gate and NOR gate can be called the universal gates since the combination of these gates can be used to accomplish any of the basic operations. Hence, NAND gate and NOR gate combination can produce an inventen, an OR gate ON an AND gate.

Symbol and truth table of NAND gate

The symbol of the NAND gate is represented as a combination of AND gate and NOT gate. The Boolean expression is given as $Y = \overline{A \cdot B}$.





NAND gate

The truthtable of a NAND gate is given below

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

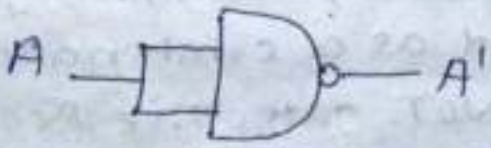
1.2 Realize AND, OR, NOT operations using NAND, NOR gates.

1.1 NAND gates as NOT gate

A NOT produces complement of the input. It can have only one input, tie the inputs of a NAND gate together. Now it will work as a NOT gate. Its output is

$$Y = (A \cdot A)'$$

$$Y = (A)'$$



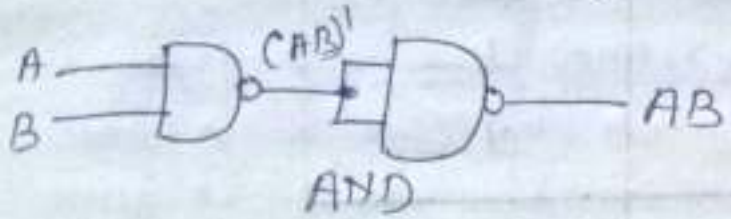
NOT (inverter)

Input	output
A	A'
0	1
1	0

1.2 NAND Gates as AND gate

A NAND produces complement of AND gate. so, if the output of a NAND gate is inverted, overall output will be that of an AND gate.

$$Y = ((A \cdot B)')'$$



INPUT		OUTPUT
A	B	F = A·B
0	0	0
0	1	0
1	0	0
1	1	1

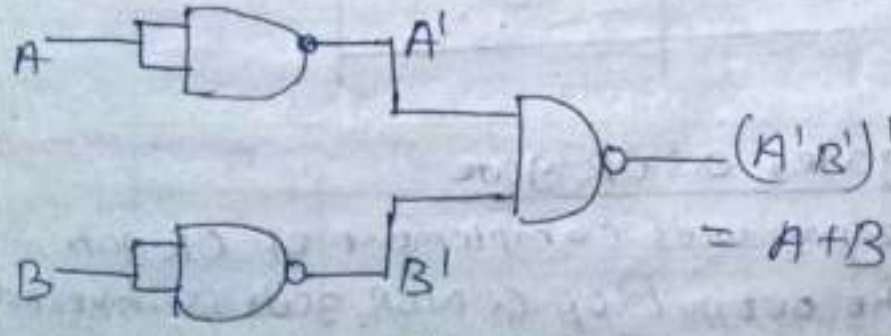
1.3 NAND gates as OR gate

From Demorgan's theorems:

$(A \cdot B)' = A' + B'$

$(A' \cdot B')' = A'' + B'' = A + B$

So, give the inverted inputs to a NAND gate, obtain OR operation at output.



(OR)

A	B	X = A+B
0	0	0
0	1	1
1	0	1
1	1	1

(2) NOR gates as NOT gate

A NOT produces complement of the input. It can have only one input, tie the inputs of a NOR gate together. Now it will work as a NOT gate. Its output is

$$Y = (A+A)'$$
$$Y = (A)'$$

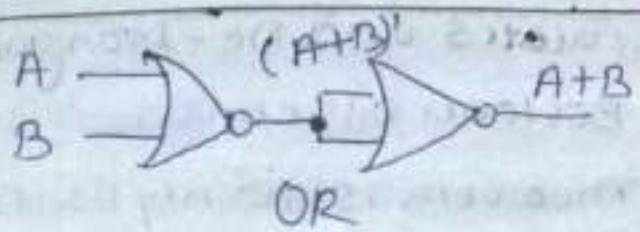
NOT

INPUT	OUTPUT
A	A'
0	1
1	0

22 NOR gates as OR gate

A NOR produces complement of OR gate. So, if the output of a NOR gate is inverted, overall output will be that of an OR gate.

$$Y = ((A+B))'$$
$$Y = (A+B)$$



OR

A	B	X = A+B
0	0	0
0	1	1
1	0	1
1	1	1

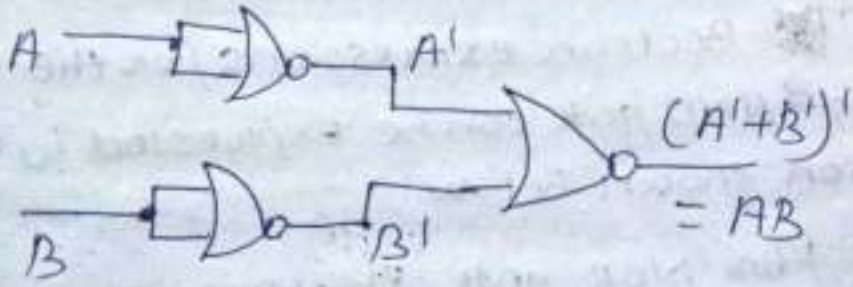
2.3 NOR gates as AND gate

From DeMorgan's theorems:

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

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

So, give the inverted inputs to a NOR gate, Obtain AND operation at output.



AND

Input		output
A	B	F = A·B
0	0	0
0	1	0
1	0	0
1	1	1

Different postulates and De-Morgan's theorems in Boolean algebra.

Ans Demorgan's Theorem is mainly used to solve the various Boolean algebra expressions. The Demorgan's theorem defines the Uniformity between the gate with the same inverted input and output. It is used for implementing the basic gate operation likes NAND gate and NOR gate.

The Demorgan's theorem mostly used in digital programming and for making digital circuit diagrams.

There are two De-Morgan's Theorems. They are described below in detail.

De-Morgan's First Theorem:-

According to De-Morgan's first theorem, a NOR gate is equivalent to a bubbled AND gate. The Boolean expressions for the bubbled AND gate can be expressed by the equation shown below.

For NOR gate, the equation is;

$$Z = \overline{A + B}$$

For the bubbled AND gate the eqn is.

$$Z = \overline{A} \cdot \overline{B}$$

As the NOR and bubbled gates are interchangeable, i.e. both gates have exactly identical outputs for the same set of inputs.

Therefore, the equation can be written as shown below:

$$\overline{A+B} = \overline{A} \cdot \overline{B} \quad \ldots\ldots (1)$$

This equation (1) on identity shown above is known as De-Morgan's Theorem. The symbolic representation of the theorem is shown in the figure below:

NOR gate

Bubbled AND gate

De-Morgan's second Theorem:-

De-Morgan's second Theorem states that the NAND gate is equivalent to a bubbled OR gate.

The Boolean expression for the NAND gate is given by the equation shown below:

$$Z = \overline{A \cdot B}$$

The Boolean expression for the bubbled OR gate is given by the equation shown below:

$$Z = \overline{A} + \overline{B}$$

Since NAND and bubbled OR gates are interchangeable, i.e., both gates have identical outputs for the same set of inputs. Therefore, the equations become as given below:

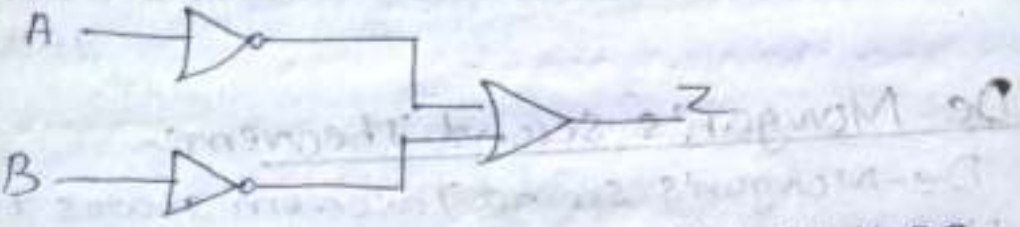
$$\overline{A \cdot B} = \overline{A} + \overline{B} \quad \ldots (2)$$

This identity ON equation(2) shown above is known as DeMorgan's second theorem. The symbolic representation of the theorem is shown in the figure below:

NAND Gate Bubbled OR gate.

The Bubbled OR Gate:-

The logic circuit of the bubbled OR gate is shown below:



The truth table for the bubbled OR gate is shown below:

A	B	Z
0	0	1
0	1	1
1	0	1
1	1	0

In this, both the inputs are inverted before they are applied to an OR gate. The output of a bubbled OR gate can be derived from its logic circuit and can be expressed by the eqn shown below

$$Z = \overline{A} + \overline{B}$$

Here are the results when the logic circuit of bubbled OR gate when all

the possible sets of inputs are applied such as 00, 01, 10 or 11.

For AB: 00

$$Z = \overline{0} + \overline{0} = 1 + 1 = 1$$

For AB: 01

$$Z = \overline{1} + \overline{0} = 0 + 1 = 1$$

For AB: 10

$$Z = \overline{1} + \overline{0} = 0 + 1 = 1$$

For AB: 11

$$Z = \overline{1} + \overline{1} = 0 + 0 = 0$$

The truth table for the bubbled AND gate is exactly identical to the truth table of a NAND gate. Hence, NAND and bubbled OR gate is interchangeable.

Postulates and Basic Laws of Boolean Algebra:-

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

Boolean postulates:

Consider the binary numbers 0 and 1, Boolean variable X and its complement or. Either the Boolean variable or Complement of it is known as Literal. The four possible logical OR operation among these literals and binary numbers

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

$$x+0=x$$
$$x+1=1$$
$$x+x=x$$
$$x+x'=1$$

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

$$x\cdot 1=x$$
$$x\cdot 0=0$$
$$x\cdot x=x$$
$$x\cdot x'=0$$

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

Note - The complement of complement of any Boolean variable is equal to the variable it self. i.e, $x''=x$.

Basic Laws of Boolean Algebra

Following are the three basic laws of Boolean Algebra.

- Commutative law
- Associative law
- Distributive law

Commutative law :-

If any logical operation of two Boolean variables give the same result irrespective of the order of these two variables, then that logical operation is said to be commutative. The logical OR & logical AND operations of two Boolean variables X & Y are shown below.

$$X+Y = Y+X$$

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

Associative law :-

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

$$X+Y+Z = X+Y+Z$$

$$X.Y.Z = X.Y.Z$$

Associative law obeys for logical OR & logical AND operations.

Distributive Law :-

If any logical operation can be distributed to all the terms present in the Boolean function, then that logical operation is said to be Distributive. The distribution

of logical OR & logical AND operations of three Boolean variables X, Y & Z are shown below.

$$X.Y+Z = X.Y + X.Z$$

$$X+Y.Z = X+Y . X+Z$$

Distributive law obeys for logical OR and logical AND operations.

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

4 Complement law $A+A' = 1$

$A \cdot A' = 0$

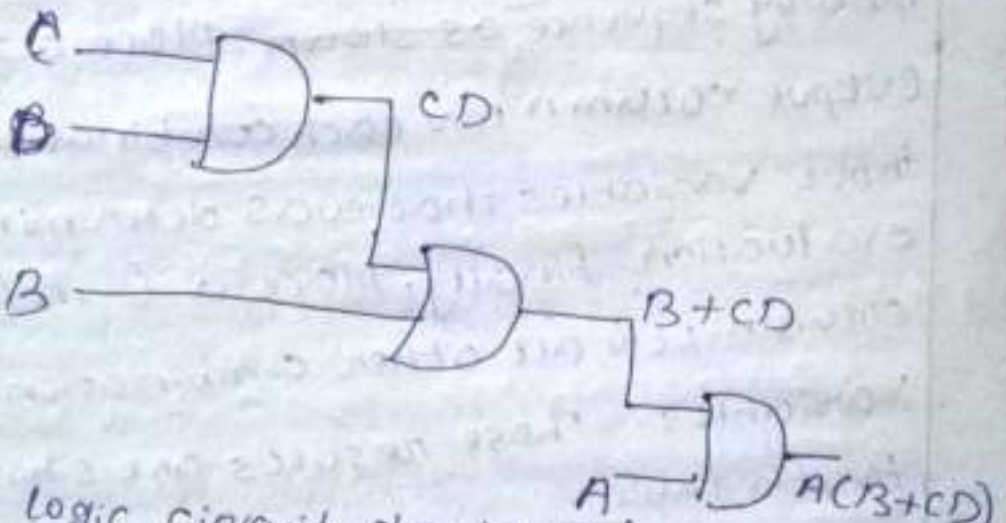
11. Use of Boolean Algebra for simplification of ~~step~~ Logic expression.

A: Boolean algebra provides a concise way to express the operation of a logic circuit formed by a combination of logic gates so that the output can be determined for various combinations of input values.

Boolean Expression for a logic circuit.

To device the Boolean expression for a given logic circuit, begin at the left-most inputs and work towards the final output, writing the expression for each gate for the example circuit in the Boolean expression is determined as follows

- The expression for the left-most AND gate with inputs C and D is CD.
- The output of the left-most AND gate is one of the inputs to the OR gate and B is the other input. Therefore, the expression for the OR gate is B+CD.
- The output of the OR gate is one of the inputs to the right-most AND gate and A is the other input. Therefore, the expression for this AND gate is A(B+CD), which is the final output expression for the entire circuit.



A logic circuit showing the development of the Boolean expression for the output.

Constructing a Truth Table for a Logic circuit:-

Once the boolean expression for a given logic circuit has been determined, a truth table that shows the output for all possible values of the input variables can be developed.

The procedure requires that you evaluate the Boolean expression for all possible combinations of values for the input variables. In the case of the circuit, there are four input variables (A, B, C & D) and therefore sixteen ($2^4 = 16$) combinations of values are possible.

Putting the results in Truth Table format

The first step is to list the sixteen input variable combinations of 1's and 0's in a binary sequence ~~as shown~~. Place a 1 in the output column for each combination of input variables that was determined in the evaluation. Finally, place a 0 in the output column for all other combinations of input variables. These results are shown in the truth table

A	B	C	D	OUTPUT A(B+CD)
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1

Inputs				Output
A	B	C	D	A(B+CD)
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Simplification using Boolean Algebra:-

A simplified Boolean expression uses the fewest gates possible to implement a given expression

Example:-

Using Boolean algebra techniques, simplify this expression:

$$AB+A(B+C)+B(B+C)$$

Solution:-

Step:-1

Apply the distributive law to the second and third terms in the expression, as follows.

$$AB+AB+AC+BB+BC$$

Step-2

Apply rule 7 ($BB=B$) to the fourth term

$$AB+AB+AC+B+BC$$

Step-3

Apply rule 5 ($AB+AB=AB$) to the first two terms.

$$AB+AC+B+BC$$

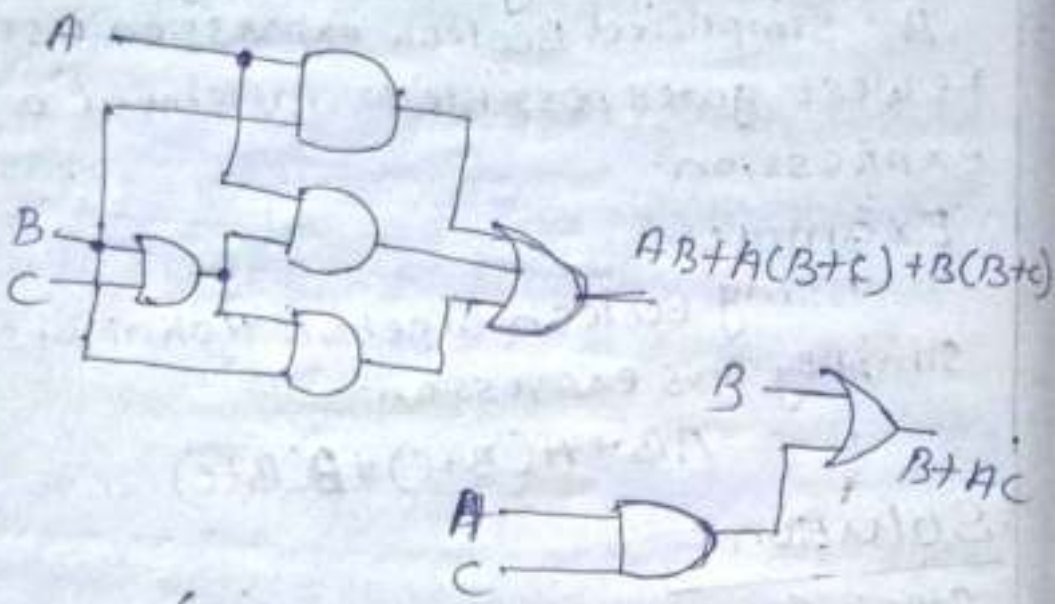
Step-4 Apply rule 10 ($B+BC=B$) to the last two terms.

AB+AC+B

step-5: Apply rule 1 (AB+B = B) to the first and third terms.

B+AC

At this point the expression is simplified as much as possible.



GATE circuits for example above.

1.11

Karnaugh Map for 2, 3, 4 Variable, simplification of SOP And POS logic expression Using K-map.

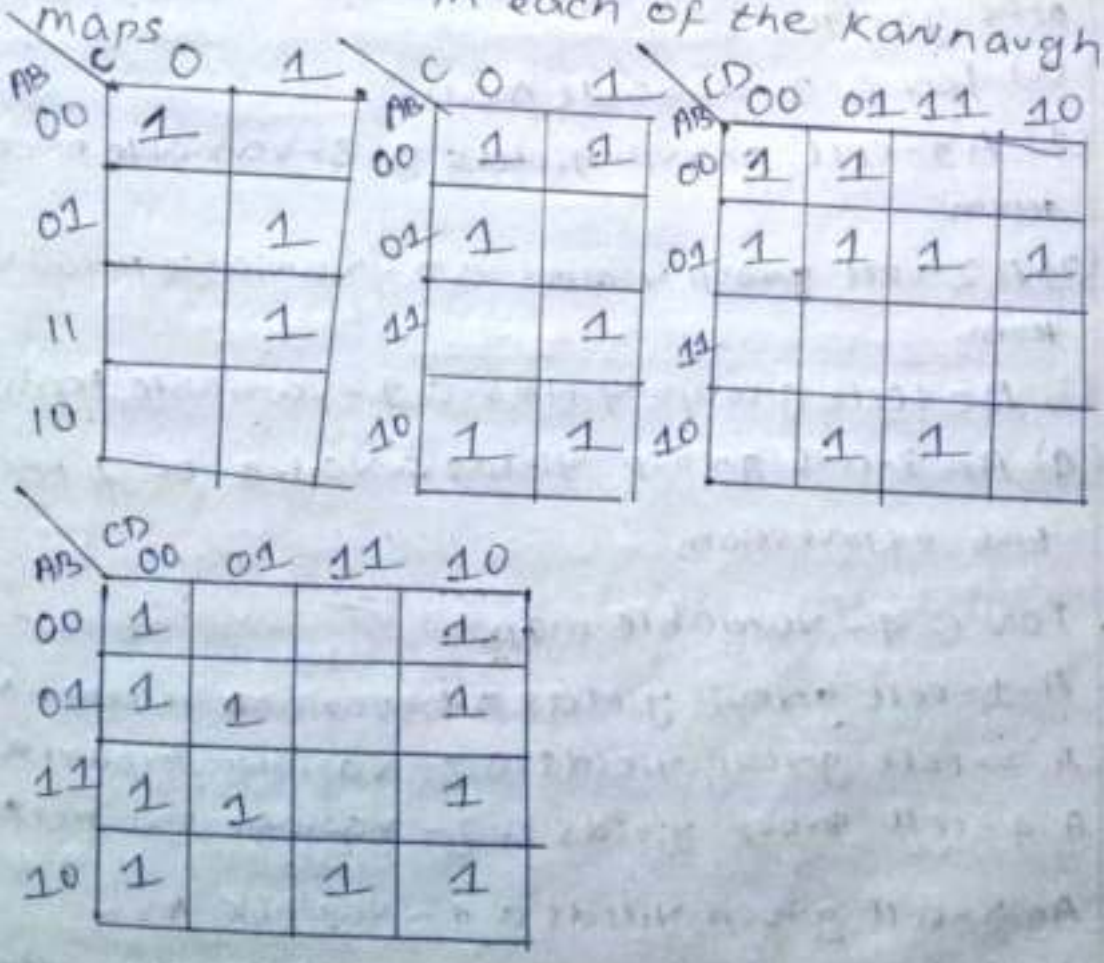
Karnaugh Map Simplification of SOP Expression.

Grouping the 1s, you can group 1s on the karnaugh map according to the following rules by enclosing those adjacent cells containing 1s. The goal is to maximize the size of the groups and to minimize the number of groups.

- A group must contain either 1, 2, 4, 8 or 16 cells, which are all powers of two. In the case of a 3-variable map, $2^3 = 8$ cells is the maximum group.
- Each cell in a group must be adjacent to one or more cells in that same group.
- Always include the largest possible number of 1s in a group in accordance with rule 1.
- Each 1 on the map must be included in at least one group. The 1s already in a group can be included in another group as long as the overlapping groups include non-common 1s.

Example :-

Group the 1s in each of the Karnaugh maps



Solution:-

The groupings are shown in fig (4.7). In some cases, there may be more than one way to group the 1s to form maximum groupings.

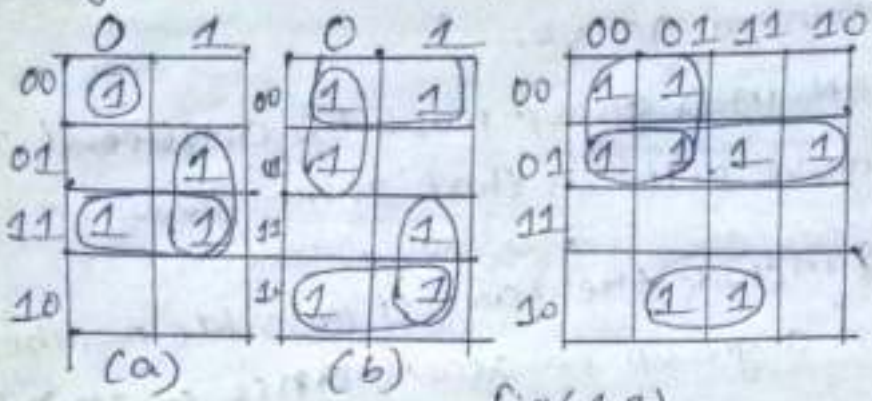
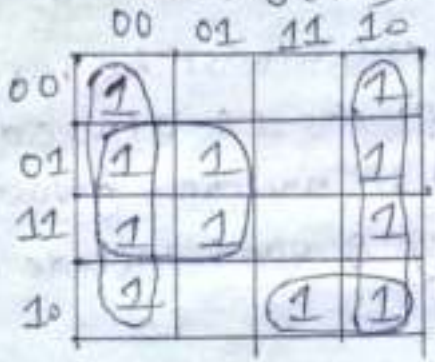


fig (4-7)



Determine the minimum product term for each ~~of the~~ group.

(a) for a 3-variable map:

(1) A 1-cell group yields a 3-variable product term

(2) A 2-cell group yields a 2-variable product term.

(3) A 4-cell group yields a 1-variable term

(4) An 8-cell group yields a value of 1 for the expression.

b. for a 4-variable map:

(1) A 1-cell group yields a 4-variable product term

(2) A 2-cell group yields a 3-variable product term.

(3) A 4-cell group yields a 2-variable product term

4) An 8-cell group yields a 1-variable term.

(5) 1 16-cell group yields a value of 1 for the expression

Karnaugh Map Simplification of POS Expression:-

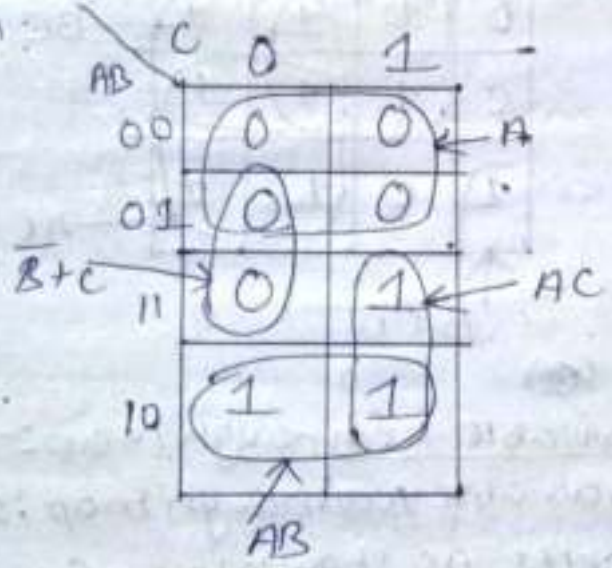
The process for minimizing a POS expression is basically the same as for an SOP expression except that you group 0s to produce minimum sum terms instead of grouping 1s to produce minimum product terms. The rules for grouping the 0s are the same as those for grouping the 1s that you learned before.

Example - 1

Use a Karnaugh map to minimize the following standard POS expression Also, derive the equivalent SOP expression.

$$(A+B+C)(A+B+\bar{C})(A+\bar{B}+C)(A+\bar{B}+\bar{C})(\bar{A}+\bar{B}+C)$$

Solution:

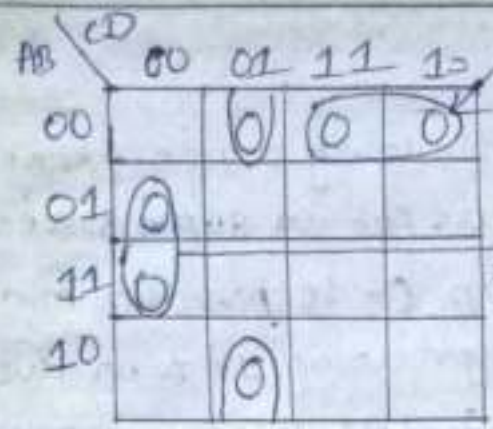


Example - 2

Using a Karnaugh map, convent the follow standard POS expression into a minimum POS expression, a standard SOP expression, and a minimum SOP expression.

$$(\bar{A}+\bar{B}+C+D)(A+\bar{B}+C+D)(A+B+C+\bar{D})$$

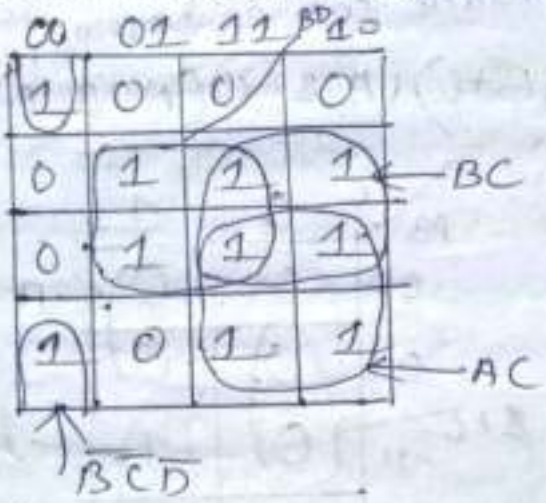
$$(A+B+\bar{C}+\bar{D})(\bar{A}+B+C+\bar{D})(A+B+\bar{C}+D)$$



Minimum POS: $(A+B+C)(\bar{B}+\bar{C}+D)$
$(B+C+\bar{D})$

(b) Standard SOP:-

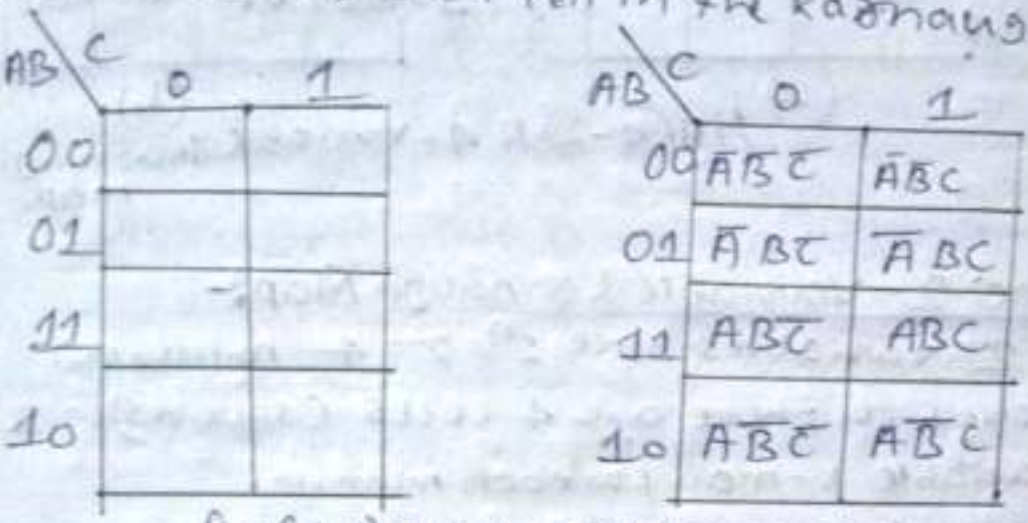
$\bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}BCD + \bar{A}B\bar{C}\bar{D} + AB\bar{C}\bar{D} +$
$A\bar{B}C\bar{D} + A\bar{B}\bar{C}\bar{D} + AB\bar{C}D + A\bar{B}CD + ABCD$



The 3-variable Karnaugh Map:-

The 3-variable Karnaugh map is an array of eight cells. as shown in fig (4-1)(a). In this case, A, B & C are used for the variables although other letters could be used. Binary values of A & B are along the left side (notice the sequence) and the values of 'C' are across the top. The value of a given cell is the binary values of A and B at the left in the same row combined with the value o

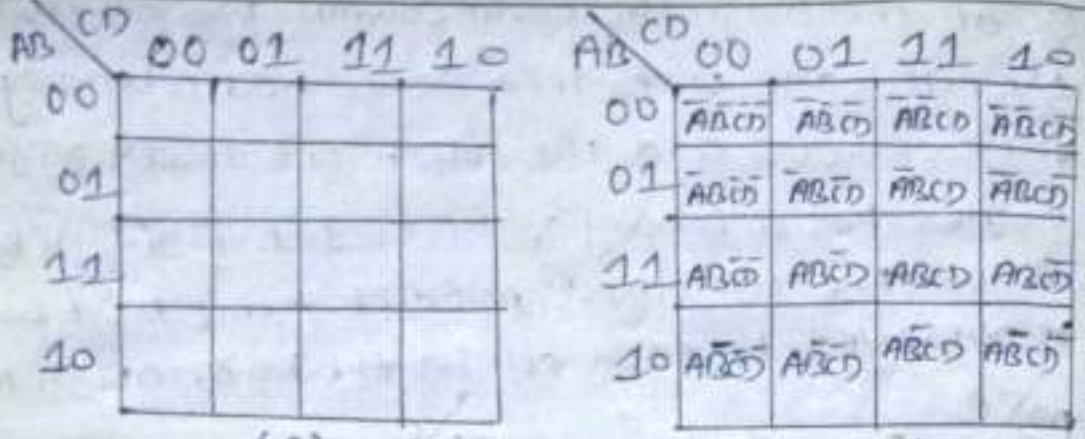
'C' at the top in the same column. For example, the cell in the upper left corner has a binary value of 000 and the cell in the lower right corner has a binary value of 101. Fig(4-1)(b) shows the standard product terms that are represented by each cell in the Karnaugh map



fig(4-1)A 3-variable Karnaugh map showing product terms.

The 4-Variable Karnaugh Map:-

The 4-variable Karnaugh map is an array of sixteen cells, as shown in fig(4-2)(a). Binary values of A and B are along the left side and the values of C and D are across the top. The value of a given cell is the binary values of 'C' & 'D' at the top in the same column. For example, the cell in the upper right corner has a binary value of 0010 and the cell in the lower right corner has a binary value of 1010. Fig(4-2)(b) shows the standard product terms that are represented by each cell in the 4-variable Karnaugh map.



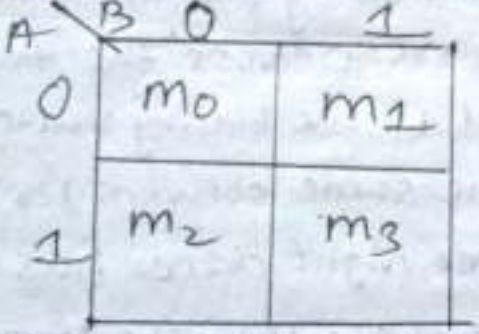
fig(4-2) A 4-variable Karnaugh map.

The 2-Variable Karnaugh Map:-

2 variables have $2^n = 2^2 = 4$ minterms. Therefore there are 4 cells (squares) in 2 variable k-map for each minterm.

Consider variable A & B as two variables. The rows of the columns will be represented by variable B.

The square facing the combination of the variable represents that min term as shown in fig below.



Grouping in 2 variables k-map is easy as there are few squares.

Examples of 2 variable K-Map:-

Function F(A,B)

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

$F = \Sigma(m_0, m_1, m_2) = \bar{A}\bar{B} + A\bar{B} + A\bar{B}$

1.5 Use of weighted and un-weighted codes & write Binary equivalent number for a number in 8421, Excess-3 and Gray code and vice - versa

Weighted codes:

The weighted codes are those that obey the position weighting principle, which states that the position of each number represent a specific weight.

In these codes each decimal digit is represented by a group of four bits.

In weighted codes, each digit is assigned a specific weight according to its position.

For example, in 8421/BCD code, 1001 the weights of 1,1,0,1 (from left to right) are 8,4,2 and 1 respectively.

Examples: 8421, 2421 and 84-2-1.

Non-weighted codes:

The non-weighted codes are not positionally weighted. In other words codes that are not assigned with any weighted to each digit position.

Example: Excess-3 (XS-3), Gray code.

Decimal	BCD 8 4 2 1	Excess-3 BCD+0011
0	0 0 0 0	0011
1	0 0 0 1	0100
2	0 0 1 0	0101
3	0 0 1 1	0110
4	0 1 0 0	0111
5	0 1 0 1	1000
6	0 1 1 0	1001
7	0 1 1 1	1010
8	1 0 0 0	1011
9	1 0 0 1	1100

* Use of weighted and un-weighted codes & Write Binary equivalent number for a number 8421, Excess-3 and Gray code

Weighted codes :-

The weighted codes are those that obey the position weighting principle, which states that the position of each number represent a specific weight.

In these codes each decimal digit is represented by a group of four bits.

In weight codes, each digit is assigned a specific weight according to its position.

For example, in 8421/BCD code, 1001 the weights of 1, 1, 0, 1 (from left to right) are 8, 4, 2 & 1 respectively.

Examples : 8421, 2421 & 84-2-1

Non-weighted codes:

The non-weighted codes are not positionally weighted. In other words codes that are not assigned with any weighted to each digit position.

Example :- Excess-3 (XS-3), Gray code.

Decimal	BCD 8 4 2 1	Excess-3 BCD +0011
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 1 0
8	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 0 0

Gray Code :-

It is a binary numbers system in which every successive pairs of numbers differs in only one bit.

→ It is used in applications in which the normal sequence of binary numbers generated by the hardware may produce an error during the transition from one number to the next.

→ For example, the states of a system may change from 3(011) to 4(100) as 011-001-101-100.

→ Therefore there is a high chance of wrong state being read while the system changes from the initial state to the final state.

→ This could have serious consequences for the machine using the information. The Gray code elimenates this problem since only one bit changes its value during any transition between two numbers.

Converting Binary to Gray code :-

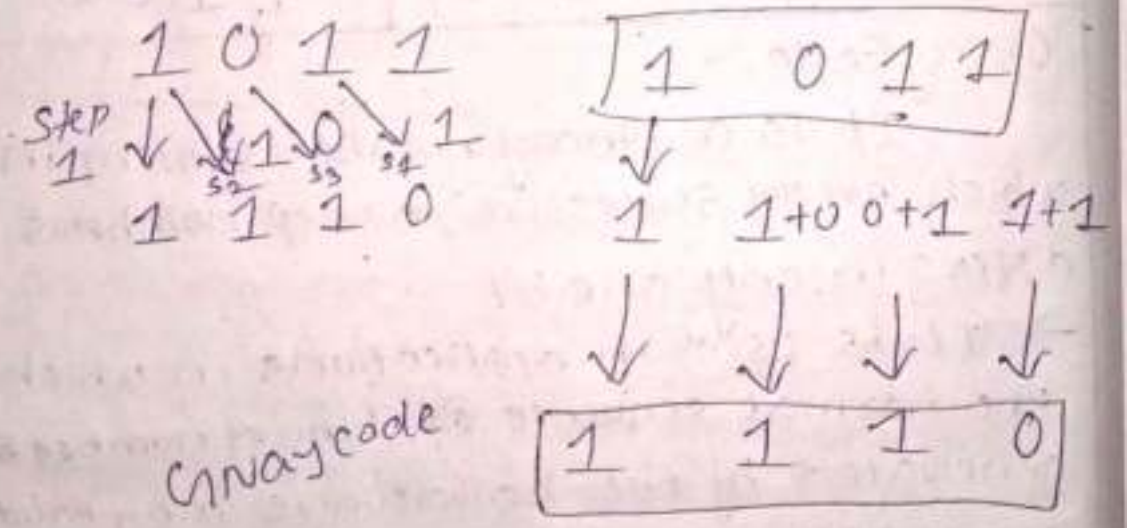
Step-1: Record the MSB as it is.

Step-2: Add the MSB to the next bit, record the sum and neglect the carry.

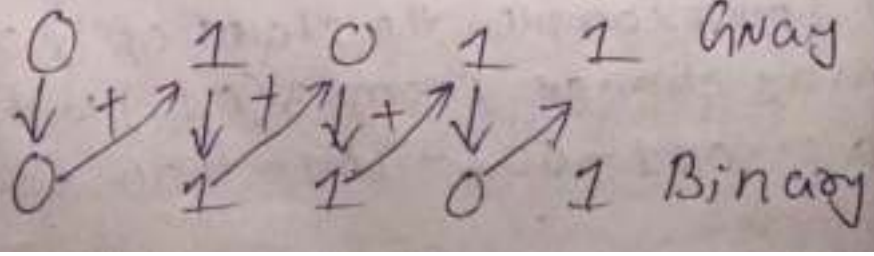
Step-3: Repeat the process.

E.g: convert 1011 to Gray code.

MSB ⟶ 1011 → LSB



Converting Gray to Binary



Importance Of Parity Bit

1. A parity bit is a check bit, which is added to a block of data for error detection purposes.
2. It is used to validate the integrity of the data.
3. The value of the parity bit is assigned either 0 & 1 that makes the number of 1s in the message block either even or odd depending upon the type of parity.
4. parity check is suitable for single bit error detection only.

Types of parity ~~bit's~~ checking →

(1) Even parity:- Here the total number of bits in the message is made even.

(2) Odd parity:- Here the total number of bits in the message is made odd.

Error detection by Adding parity bit

In case of even parity:-

If number of 1s is even, parity value is 0, If number of 1s is odd, parity bit value is 1.

In case of odd parity:

If number of 1s is odd, parity value is 0. If number of 1s is even parity bit value is 1.

e.g.

case 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |

No error
Number of 1 bits is 4 which is even

case 2 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |

Error detected
No of 1 bits=5 which is no even

case 3 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |

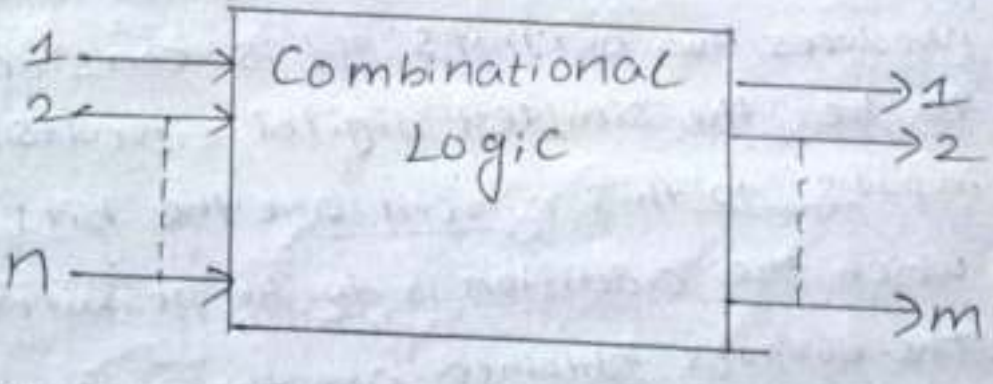
failure to detect error,
No. of 1 bit=4 which is even.

— o —

2 COMBINATIONAL LOGIC CIRCUITS

2.1 Give the concept of combinational logic circuit

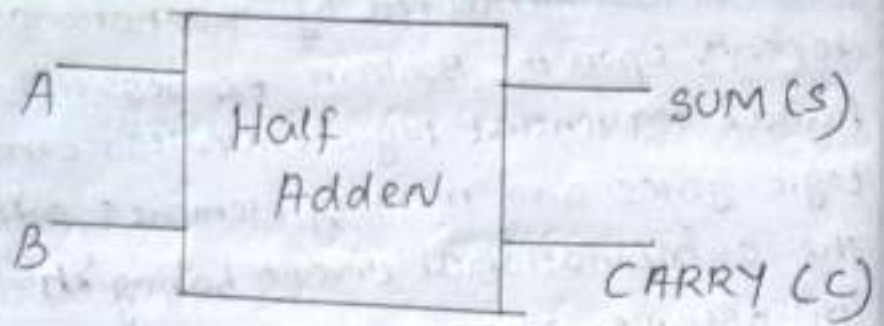
A combinational circuit is the digital logic circuit in which the output depends on the combination of inputs at that point of time with total disregard to the past state of the inputs. The digital logic gate is the building block of combinational circuits. The function implemented by combinational circuit is depend upon the Boolean expressions. On the other hand, sequential logic circuits, consists of both logic gates and memory elements such as flip-flops. The combinational circuit having 'n' inputs and 'm' outputs. The n number of inputs shows that there are 2^n possible combinations of bits at the input. Therefore, the output is expressed in terms m boolean expressions.



2.2 Half adders circuit and verify its functionality using truth table.

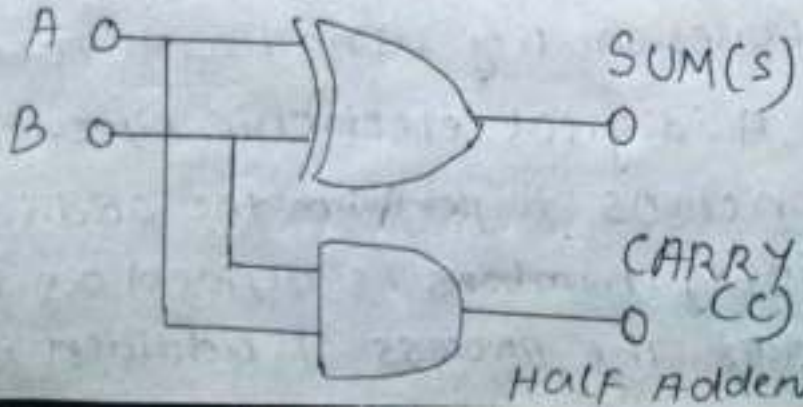
A:- A digital electronic circuit that functions to perform the addition on the binary numbers is defined as Half Adder. The process of addition is denary

the sole difference is the number system chosen. There exists only 0 & 1 in the binary numbering system. The weight of the number is completely based on the positions of the binary digits. Among those 1 & 0, 1 is treated as the largest digit and 0 as the smaller one. The Block diagram of this adder is



Half Adder circuit Diagram

A half adder consists of two inputs and produces two outputs. It is considered to be the simplest digital circuits. The inputs to this circuit are the bits on which the addition is to be performed. The outputs obtained are the sum and carry.



Half Adder

The circuit of this adder comprises of two gates. They are AND & XOR gates. The applied inputs are the same for both the gates present in the circuit. But the ~~input~~ output is taken from each gate. The output of the XOR gate is referred to as SUM and the output of AND is known CARRY.

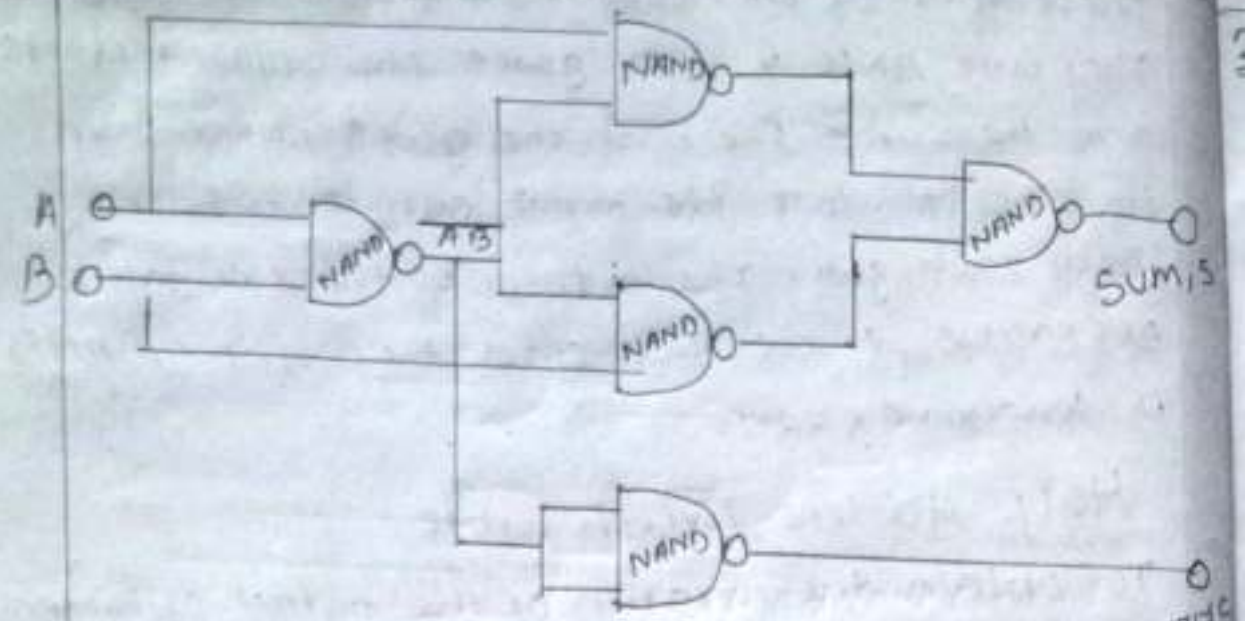
Half Adder Truth table:

To obtain the relation of the output obtained to the applied input and can be analyzed using a table known as TRUTH table.

INPUT		OUTPUT	
A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

2.3 Realize a Half-adder using NAND gates only and NOR gates only.

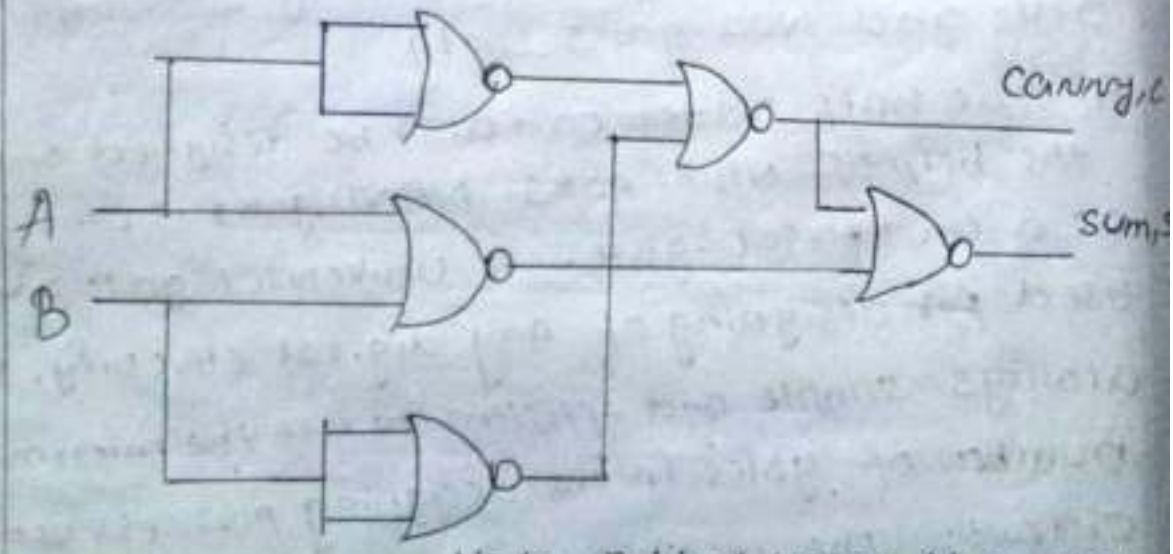
A:- The half adder can also be designed with the help of NAND gates. NAND gate is considered as a universal gate. A universal gate can be used for designing of any digital circuity. It is always simple and efficient to use the minimum number of gates in the designing process of our circuit. The minimum number of NAND gates required to design half adder is 5.



Half Adden Using NAND Gates

Half Adden Using NOR Gates:-

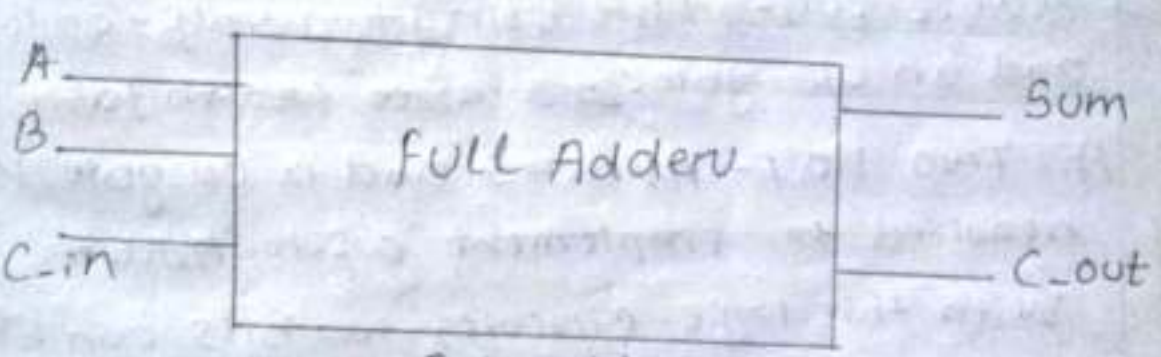
The NOR gate is also a universal gate. Thus, it can also be used for designing of any digital circuit. The Half adden can be designed using 5 NOR gates. This is the minimum number of NOR gates to design half adden.



Half Adden Using Minimum NOR Gates

2.4 Full adder circuit and explain its operation with truth table.

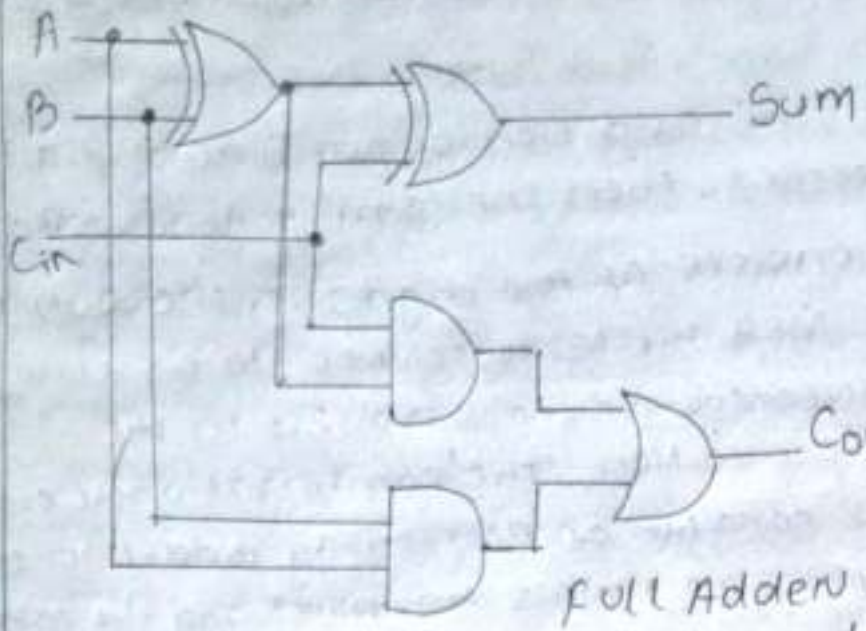
A:- When the addition of two binary digits is performed, then the sum is generated. If it consists of two digits in the output then the MSB bit is referred to as carry. This is treated as the third bit in the process of addition. The combinational circuit which is capable of performing addition on three input bits that is two inputs and the carry-in from the previous operation is known as full Adder.



Full Adder Block Diagram

Full Adder circuit Diagram, Truth table & Equation

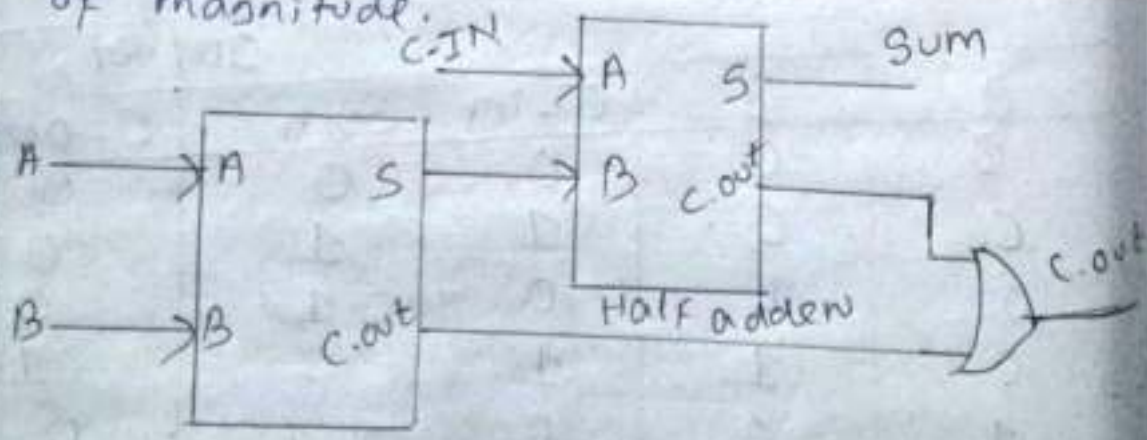
			Output	
A	B	C-in	Sum	C-out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Full Adden cincuit diagnam.

2.5 Realize full-adden using two Half-addens and an OR-gate. ~~and write the truth table~~.

A: Two Half-Addens and a OR gate is nequired to implement a full Adden. With this logic cincuit, two bits can be added togethen, taking a canny fnom the next lowen onden of magnitude, and sending a canny to the next higher ondes of magnitude.



2.6 Full subtractor circuit and explain its operation with truth table.

A:- A full subtractor is a combinational circuit that performs subtraction of two bits, one is minuend and other is subtrahend, taking into account borrow of the previous adjacent lower minuend bit. This circuit has three inputs and two outputs. The three inputs A, B and Bin, denote the minuend, subtrahend, and previous borrow, respectively. The two outputs, D and Bout represent the difference and output borrow, respectively.

Truth Table:-

INPUT			OUTPUT	
A	B	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

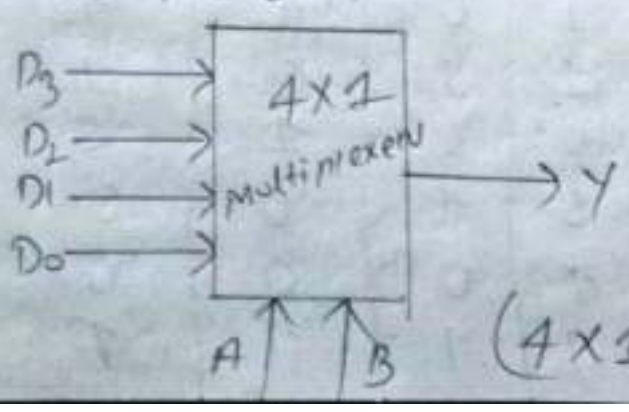
21. Operation of 4x1 Multiplexers & 1x demultiplexers

A:- A multiplexer is a circuit with many inputs but only one output. Multiplex means many into one. Multiplexer is also called as MUX. By applying control signals, we can steer any input to the output. Thus it is also called a data selector and control inputs are termed select inputs.

The circuit has n input signals, m control signals and 1 output signal. Note that, m control signals can select at the most 2^m input signals thus $n<2^m$.

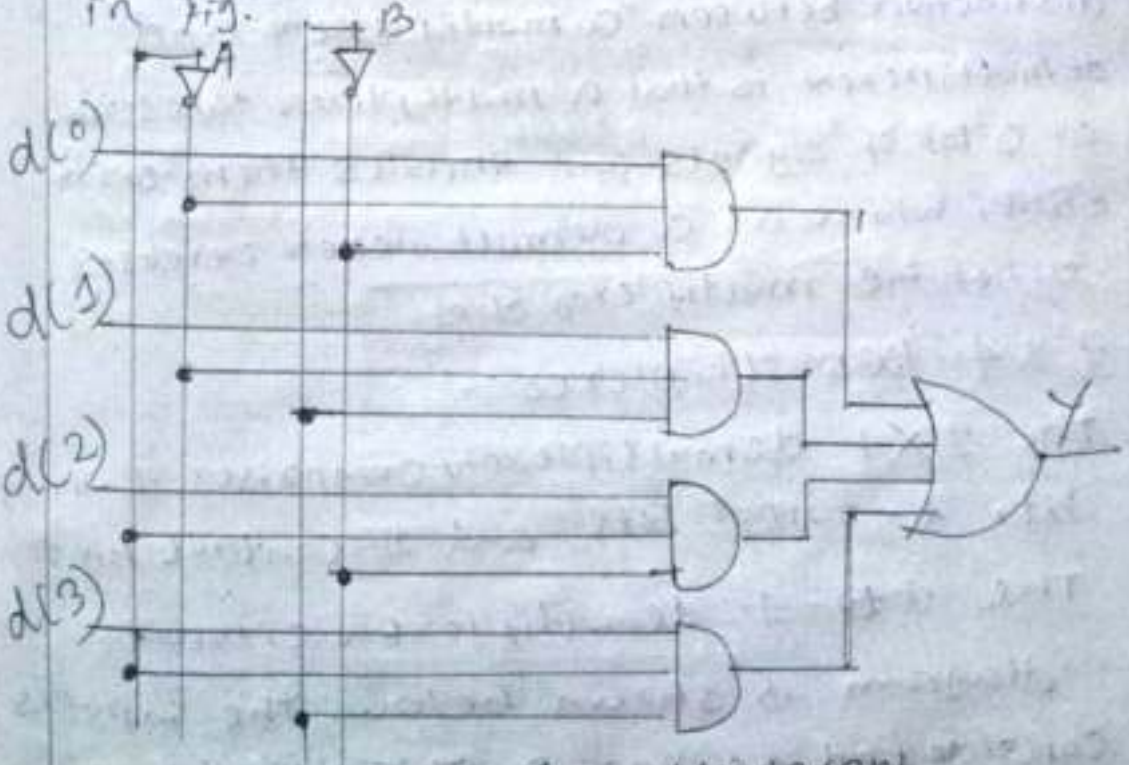
4x1 multiplexer :-

The 4-to-1 multiplexer comprises 4-input bits, 1-output bit, and 2-control bits. The four input bits are namely D_0, D_1, D_2, & D_3, respectively; only one of the input bit is transmitted to the output. The output 'y' depends on the value of control input AB. The control bit AB decides which of the input data bit should transmit the output.



(4x1 Multiplexer)

If the control input is changed to 11, then all gates are restricted except the bottom AND gate. In this case, D_3 is transmitted to the output and $Y = D_0$. If the control input is changed to AB = 11, all gates are disabled except the bottom AND gate. In this case, D_3 is transmitted to the output and $Y = D_3$ the best example of 4X1 multiplexer is IC 74153. In this IC, the output is same as the input. Another example of 4X1 multiplexer is IC 75352. In this IC, the output is the compliment of the input. The circuit diagram of a 4X1 multiplexer is shown in fig.



4-to-1 multiplexer

A	B	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

(Truth table)

Demultiplexer :-

A demultiplexer is a device, that has one input and multiple output lines which is used to send a signal to one of the various devices. Demultiplexer means one into many. By applying control signals, we can steer the input signal to one of the output lines. The most prominent distinction between a multiplexer and demultiplexer is that a multiplexer takes two or a lot of signals and encodes them on a wire, whereas a demultiplexer reverses what the multiplexer does.

1 x 4 Demultiplexer :-

The 1x4 demultiplexer comprises 1-input bit, 4-output bits and 2-control bits. The 1-to-4 demultiplexer circuit diagram is shown below. The input is considered as Data D. This data bit is transmitted to the Data bit of the output lines, which depends on the AB value and the

control input. When the control input AB=01, the upper second AND gate is permitted while the remaining AND gates are restricted. Thus, only data bit D is transmitted to the output and Y_1 = Data. If the data bit D is low, the output Y_1 is low. If data bit D is high, the output Y_1 is high. The value of the output Y_1 depends upon the value of data bit D, the remaining outputs are in a low state. If the control input changes to AB = 10, then all the gates are restricted except the third AND gate from the top. Then, data bit D is transmitted only to the output Y_2; and Y_2 = Data. The best example of 1-to-4 demultiplexer is IC 74 155.

2.8 Working of two bit magnitude comparator.

A: A comparator used to compare two binary numbers each of two bits is called a 2-bit Magnitude comparator. It consists of four inputs and three outputs to generate less than, equal to and greater than between two binary numbers. The truth table for a 2-bit comparator is given below:

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

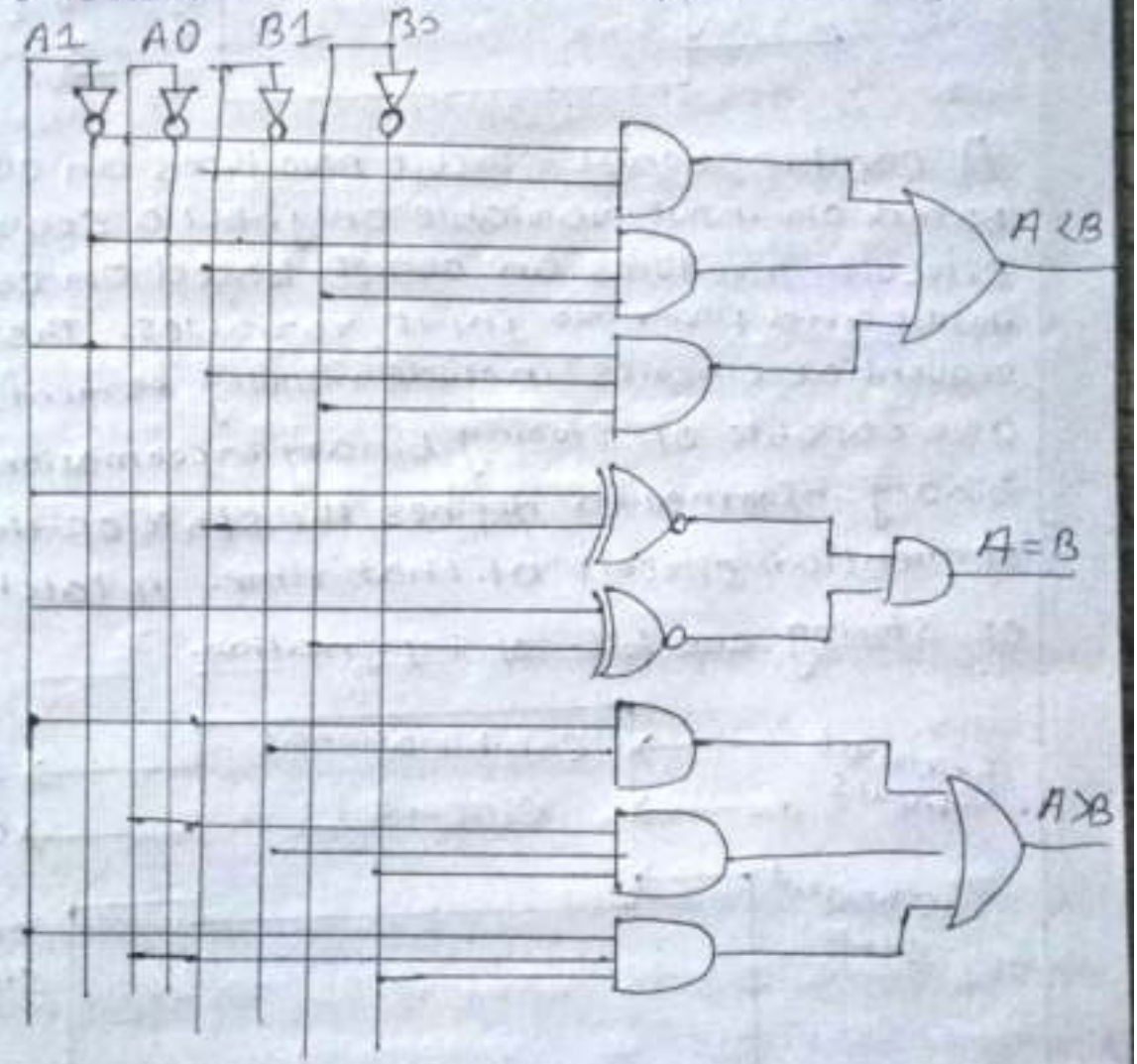
from the above truth table K-map for each output can be drawn as follows:

A>B

A1 A0 \ B1 B0	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

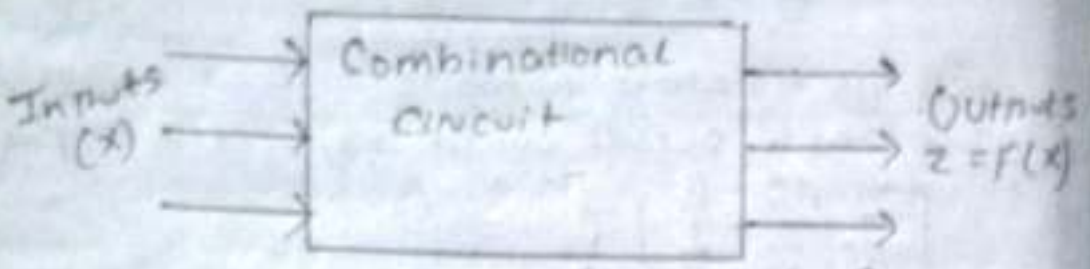
A1A0 \ B1B0	00	01	11	10
00	(1)	0	0	0
01	0	(1)	0	0
11	0	0	(1)	0
10	0	0	0	(1)

From the above K-maps logical expressions for each output can be expressed as follows.

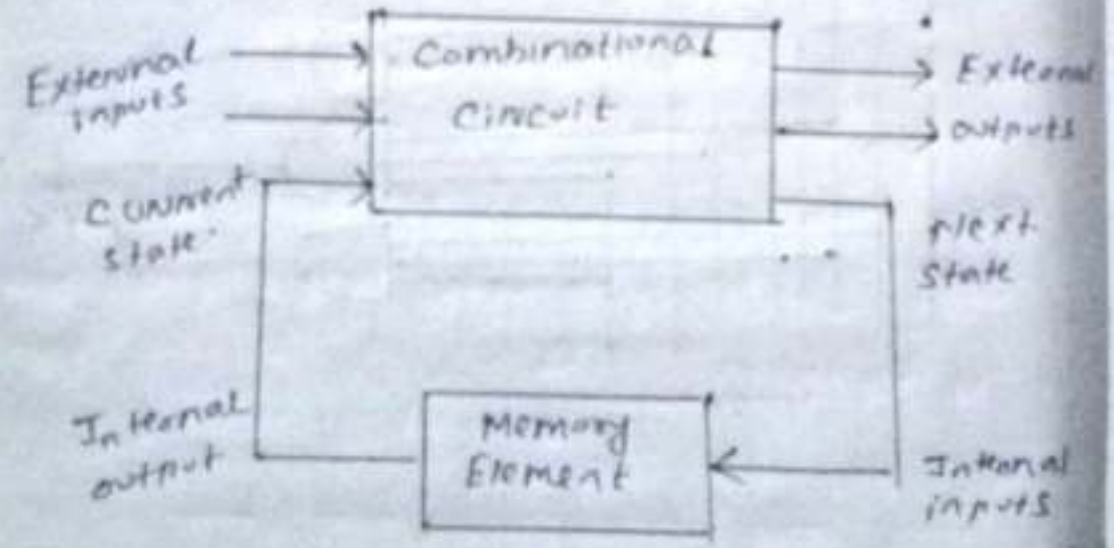


SEQUENTIAL LOGIC CIRCUITS

A sequential circuit is a combinational logic circuit that consists of inputs variable (x), logic gates (computational circuit), and output variable (z).



A combinational circuit produces an output based on input variable only, but a sequential circuit produces an output based on current input and previous input variables. That means sequential circuits include memory elements that are capable of storing binary information. That binary information defines the state of the sequential circuit at that time. A latch capable of storing one bit of information.



Sequential circuit

Types of sequential circuits - There are two types of sequential circuits:

1. Asynchronous sequential circuits
2. Synchronous sequential circuits.

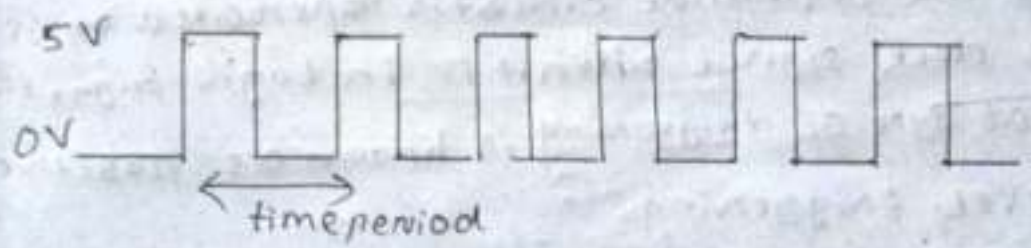
2. State the necessity of clock and give the concept of level clocking and edge triggering.

Clock signal & Triggering :-

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

Clock signal :-

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



In the above figure, square wave is considered as clock signal. This signal stays at logic High 5 V for some time and stays at Logic Low 0V for equal amount of time. This pattern repeats with some time period. In this case, the time period will be equal to either twice of ON time or twice of Off time. The reciprocal of the time period of clock signal is known as the frequency of the clock signal. All sequential circuits are operated with clock signal. So, the frequency at which the sequential circuits can be operated accordingly the clock signal frequency has to be chosen.

Types Of Triggering:-

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

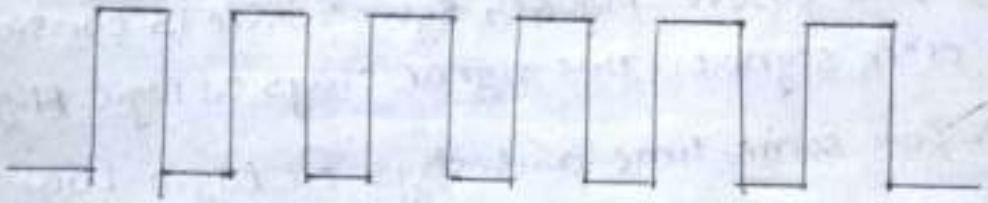
- Level triggering.
- Edge triggering.

Level triggering:-

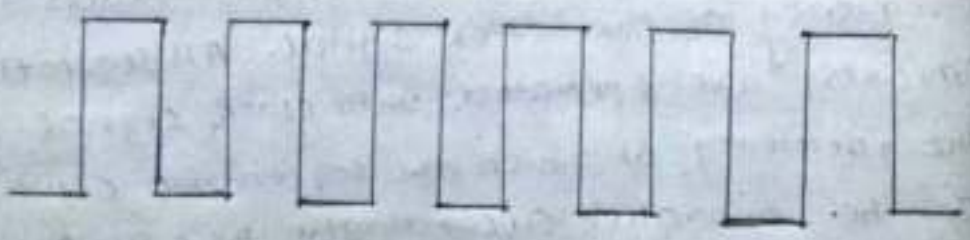
There are two levels, namely logic high and logic Low in clock signal. following are the two types of level triggering.

- positive level triggering
- Negative level triggering.

If the sequential circuit is operated with the clock signal when it is in Logic High, then that type of triggering is known as positive level triggering.



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



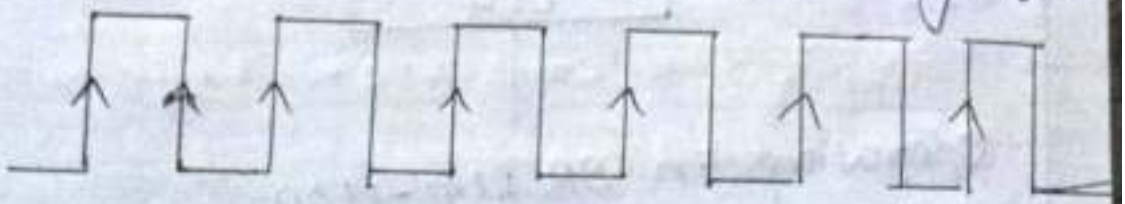
Edge triggering :-

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

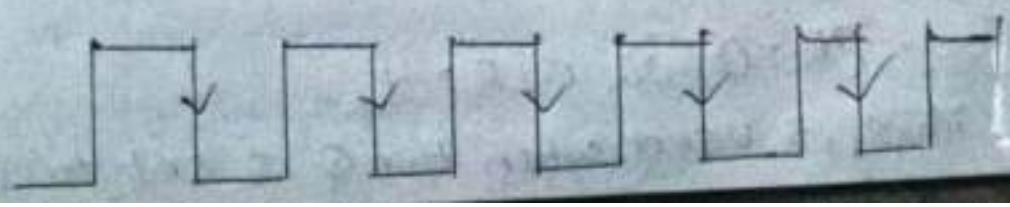
following are the two types of edge triggering based on the transitions of clock signal.

- positive edge triggering
- Negative edge triggering.

- If the sequential circuit is operated with the clock signal that is transitioning from logic low to logic High, then that type of triggering is known as positive edge triggering. It is also called as rising edge triggering. It is shown in the following figure.



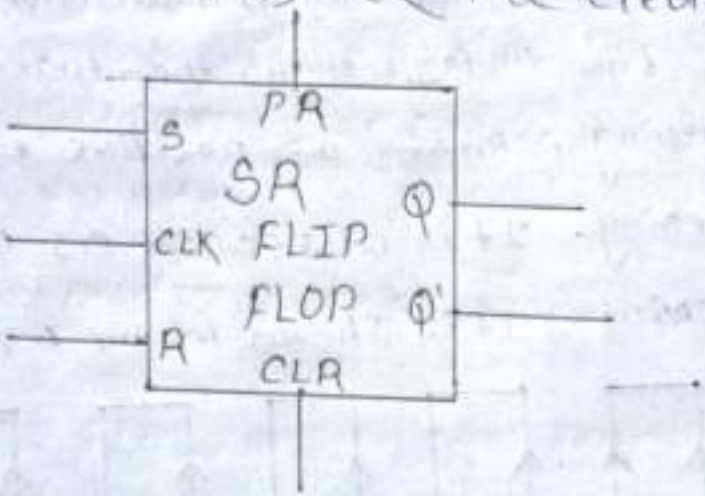
- If the sequential circuit is operated with the clock signal that is transitioning from Logic High to logic Low, then that type of triggering is known as Negative edge triggering. It also called as falling edge triggering.



3.3 Clocked SR flip flop with present and clear inputs.

SR flip-flop:-

In SR flip-flop, with the help of present and clear, when the power is switched ON, the state of the circuit keeps on changing, i.e. it is uncertain. It may come to set ($Q=1$) or Reset ($Q'=0$) state. In many applications, it is desired to initially set and Reset the flip flop. This thing is accomplished by the present (PR) & the clear (CLR).



Operations in SR flip-flop-

- Case-1

$PR = CLR = 1$

The asynchronous inputs are inactive and the flip flop responds freely to the S, R and the CLK inputs in the normal way.

- Case-2

$PR = 0$ & $CLR = 1$

This is used when the Q is set to 1.

Case-3:

PR = 1 & CLR = 0

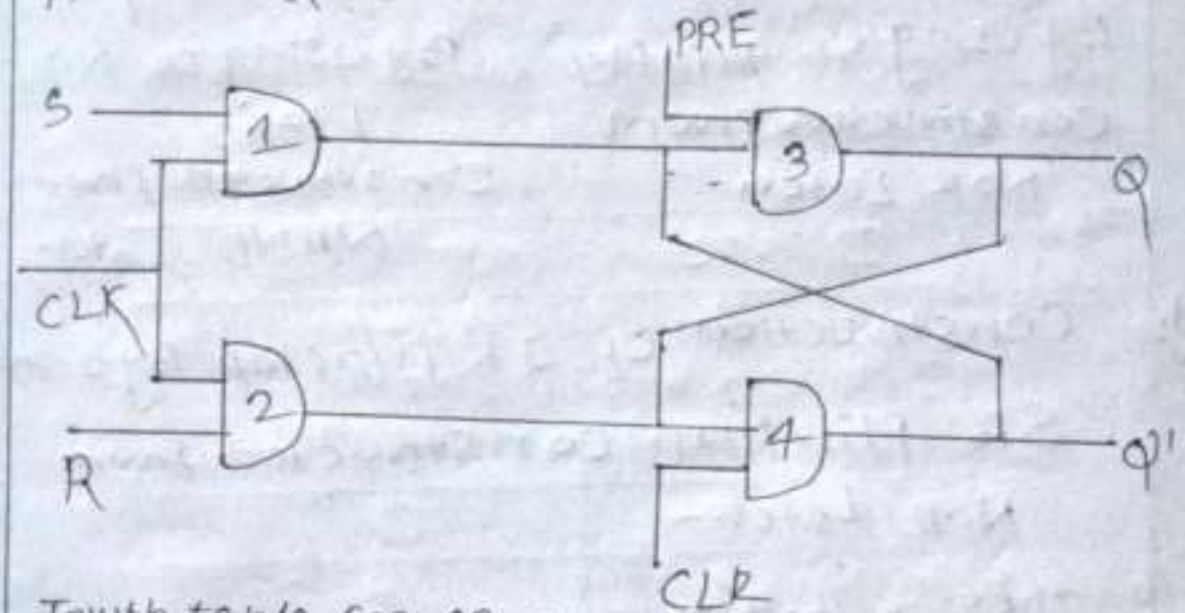
This is used when the Q' is set to 1.

case-4:

PR = CLR = 0

This is an invalid state.

SR Flip-flop with the representation of Preset & Clear.



Truth table for SR flip-flop-

Inputs					Outputs		
PR	CLA	CLK	S	R	Q(n+1)	Q'(n+1)	Comments
0	1	NA	NA	NA	1	0	set
1	0	NA	NA	NA	0	1	Reset
1	1	0	NA	NA	Q(n)	Q'(n)	No change
1	1	1	0	0	Q(n)	Q'(n)	No change
1	1	1	1	0	1	0	set
1	1	1	0	1	0	1	Reset
1	1	1	1	1	NA	NA	Not allowed

3. Construct level clocked JK flipflop using S-R flipflop and explain with truth table.

A: There are following two methods for constructing a JK flip-flop

Methods for constructing SR flip-flop

- By using SR flipflop constructed from NOR Latch
- By using SR flip flop constructed from NAND Latch.

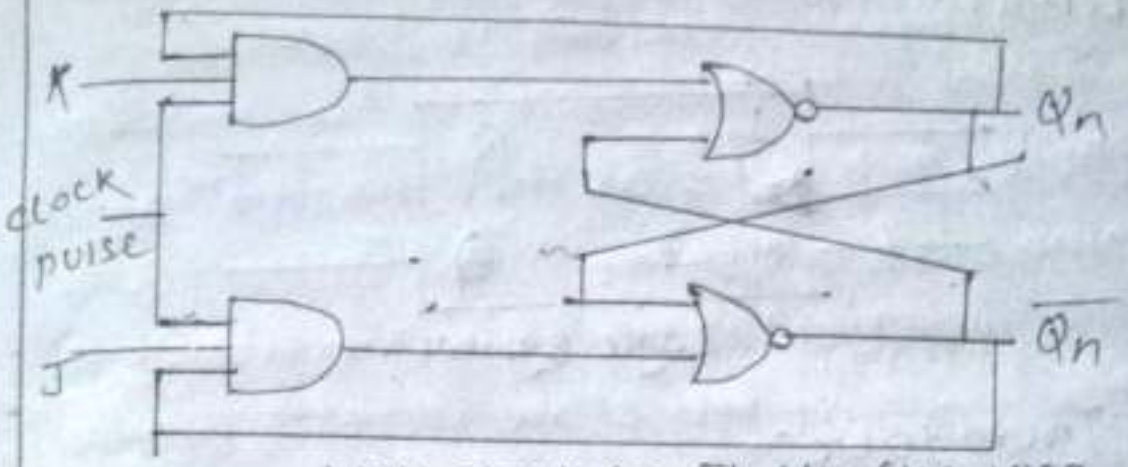
1. Construction of JK flipflop By using SR flipflop constructed from NOR Latch-

This method of constructing JK flipflop uses.

- SR flipflop constructed from NOR latch
- Two other connections

Logic circuit -

The logic circuit for JK flipflop constructed using SR flipflop constructed from NOR latch is as shown below.



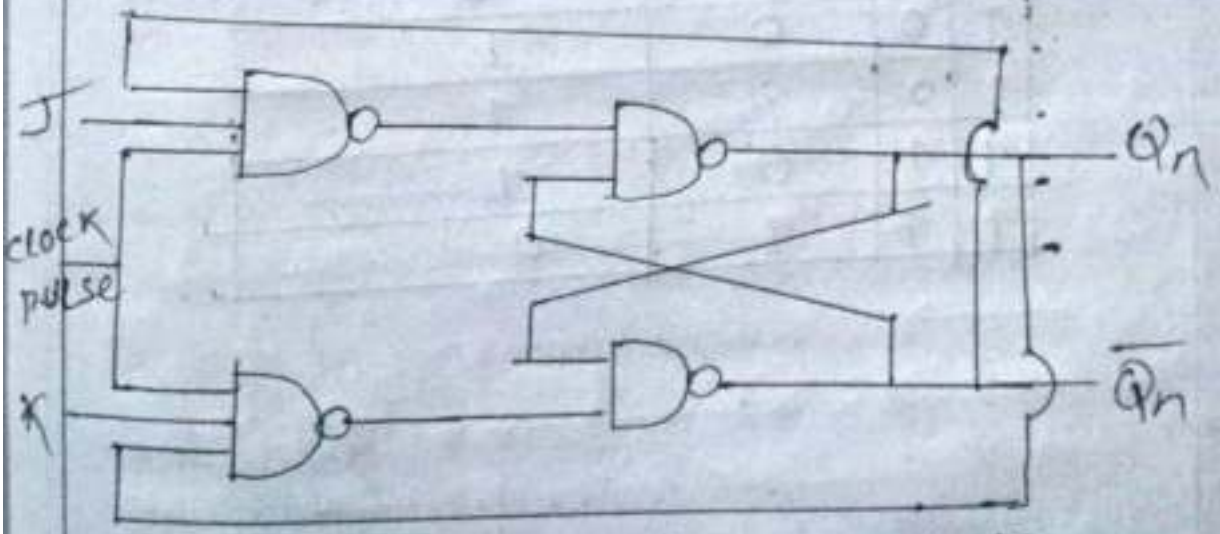
Logic circuit for JK flip flop using SR flipflop (constructed from NOR Latch)

2. Construction of JK flip flop by using SR flip flop constructed from NAND Latch-

- SR flip flop constructed from NAND Latch.
- TWO other connections.

Logic circuit:-

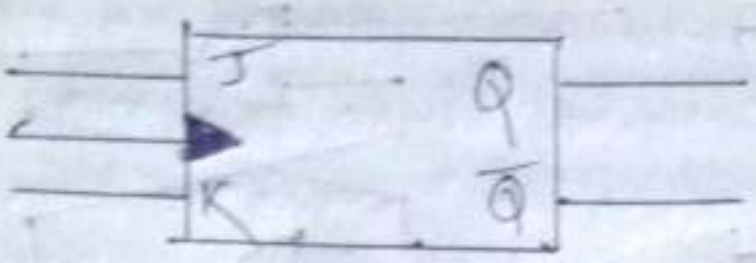
The logic circuit for Jk flipflop constructed using SR flip flop constructed from NAND latch is as shown below.



(Constructed using NAND Latch)

Logic Symbol:

The logic symbol for JK flip flop is as shown below.



Logic Symbol

Truth table:-

The truth table for JK flip flop is as shown below.

INPUTS			OUTPUTS
J	K	Qn (present state)	Qn+1 (Next state)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

INPUTS			OUTPUTS	REMARKS
J	K	Q_n (Present state)	Q_{n+1} (Next state)	States and Conditions
0	0	X	Q_n	Hold state condition ⇒ J = K = 0
0	1	X	0	Reset state condition J = 0, K = 1
1	0	X	1	set state condition J = 1, K = 0
1	1	X	Q_n'	Toggle state condition J = K = 1

3.5 Concept of race around condition and study of master slave JK flipflop.

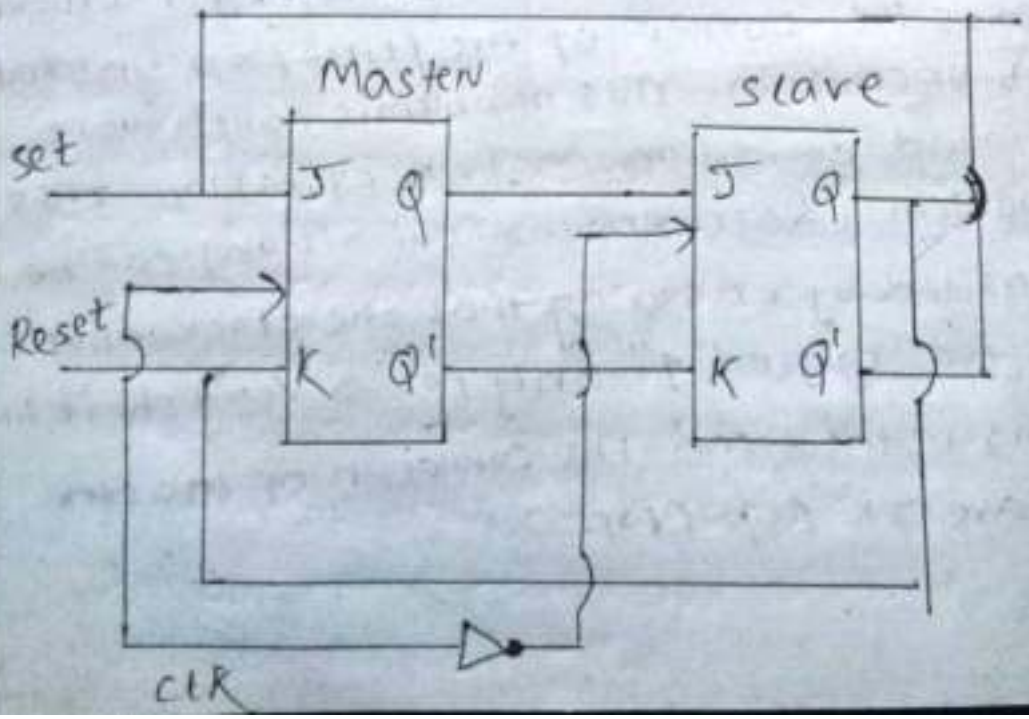
Race Around condition in JK flip-flop:-

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

Master Slave JK flip flop :-

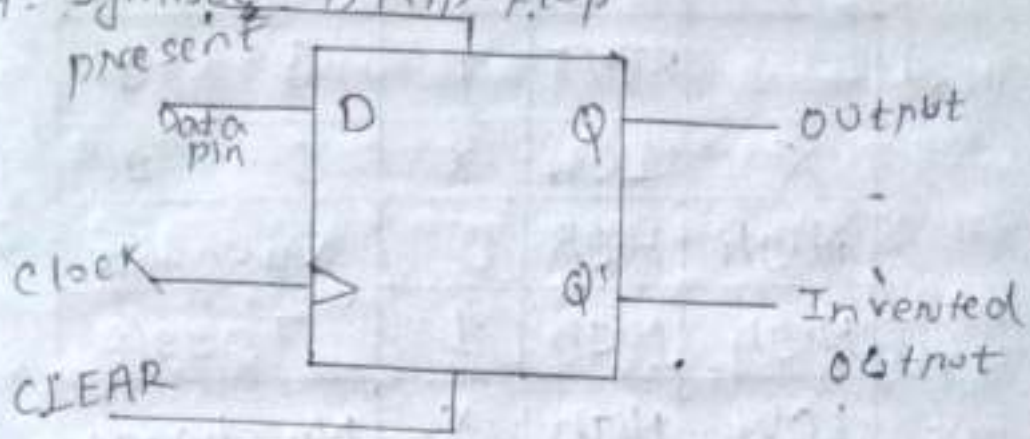
The master slave flip flop is basically a combination of two JK flipflops connected together in a series configuration. Out of these, one acts as the "master" and the other as a "salve". The output from the master flipflop is connected to the two inputs of the slave flipflop whose ~~input~~ output is fed back to inputs of the master flipflop.

In addition to these two flip-flops, the circuit also includes an inverter. The inverter is connected to clock pulse in such a way that the inverted clock pulse is given to the slave flip-flop. In other words if CP = 0 for a master flip-flop, then CP = 1 for a salve flip-flop and if CP = 1 for master flip flop then it becomes 0 for slave flip flop.



2.6 Give the truth tables of edge triggered D and T flip flops and drawing their symbols.

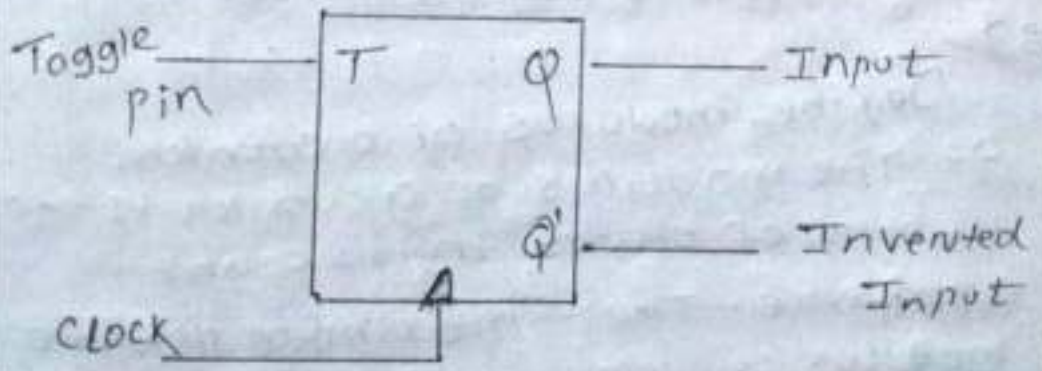
A: Symbol: D flip-flop



Truth table of D flip-flop:

CLOCK	INPUT	OUTPUT	
	D	Q	Q'
LOW	X	0	1
HIGH	0	0	1
HIGH	1	1	0

* Symbol: T flip-flop



Truth Table of T flip flop:

Clock	INPUT		OUTPUT	
	RESET	T	Q	Q'
X	LOW	X	0	1
High	High	0	No change	
High	High	1	Toggle	
LOW	High	X	No change	

3.7 Applications of flip flop:

A:-
- Counters.
- frequency Dividers.
- Shift-Registers.
- storage Registers.
- Bounce elimination switch.
- Data storage
- Data transfer.
- Latch.

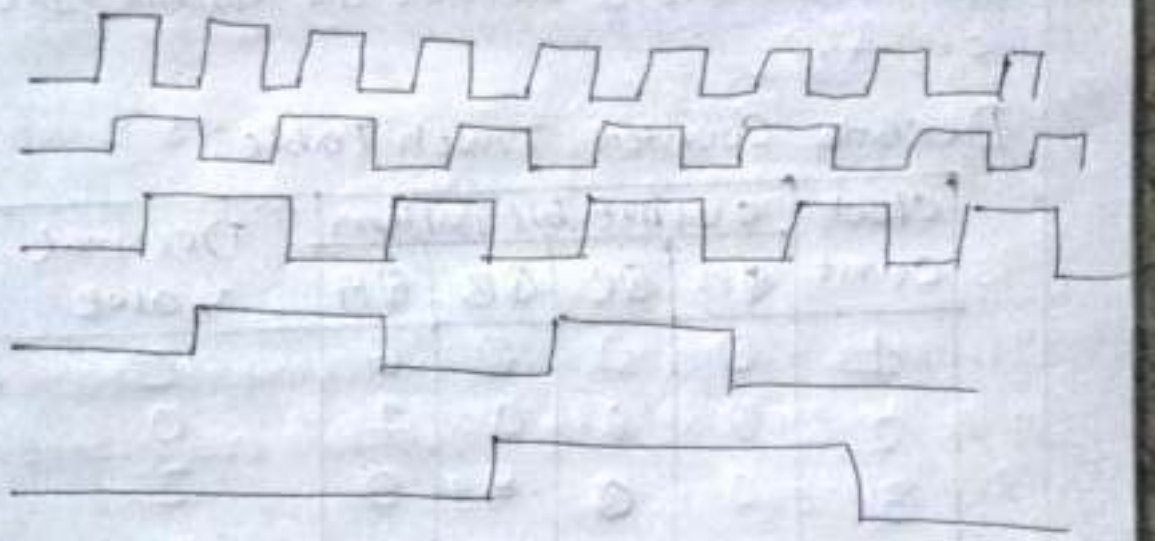
3.8 Define modulus of a counter.

A:- The modulus of a counter is the number of states in its count sequence. The maximum possible modulus is determined by the number of flip-flops. For example, a four-bit counter can have a modulus of up to 16 (2^4).

3.9 4-bit asynchronous counter and its timing diagram.

Asynchronous counter :-

In asynchronous counter we don't use universal clock, only first flip flop is driven by main clock and the clock input of rest of the following counters is driven by output of previous flip flops.



(b) Timing diagram.

3.10 Asynchronous decade counter.

A:- This type of asynchronous counter counts upwards on each trailing edge of the input clock signal starting from 0000 until it reaches an output 1001 (decimal 9). Both outputs QA and QD are now equal to logic "1". On the application of the next clock pulse, the output from the 74LS10 NAND gate changes state from logic "1" to a logic "0" level.

As the output of the NAND gate is connected to the ~~CLAE~~ CLEAR ($\overline{CLR}$) inputs of all the 74LS73 J-K flip-flops, this signal causes all of the Q outputs to be reset back to binary 0000 on the ~~output~~ count of 10. As outputs QA & QD are now both equal to logic "0" as the flip-flop's have just been reset, the output of the NAND gate returns back to a logic level "1" and the counter restarts again from 0000. We now have a decade or Modulo-10 up-counter.

Decade Counter Truth Table:

Clock count	output bit pattern				Decimal value
	QD	QC	QB	QA	
1	0	0	0	0	0
2	0	0	0	1	0
3	0	0	1	0	2
4	0	0	1	1	3
5	0	1	0	0	4
6	0	1	0	1	5
7	0	1	1	0	6
8	0	1	1	1	7
9	1	0	0	0	8
10	1	0	0	1	9

3.17 4-bit Synchronous counter.

4-Bit Synchronous Up counter:-

The designing of a 4-Bit synchronous up counter can be done like a 3-bit synchronous up-counter but the difference is in the number of flip-flops used. In this counter, four JK flip-flops are used to design. The main reason to use this flip flop is, it toggles its condition if both the inputs are high based on the CLK signal. An external CLK signal is given to all four flip-flops in parallel. This counter includes 16 output states where it counts from 0000 to 1111. As compared to 3-bit, the timing diagram of this counter & its operation is also the same.

4-Bit Synchronous Down counter:-

The main function of this counter is to count the numbers in decreasing order. As compared to the up counter, the down counter is also the same but it must reduce its count. Thus, JK flipflop inputs are connected towards the Q' and the same external CLK signal is connected to four flipflops within the circuit. Whenever this counter counts down the series, at first all the inputs of the FF will be in high condition because they have to count down the series. So, it will begin with 1111 & stop with 0000 like an up counter. In this type of counter, it should be noted that, if the front flip flop generates low logic at its output, then the previous flip flop will toggle simply.

3.P Distinguish between synchronous and asynchronous counters.

A: Synchronous counter:

1. In Synchronous counter, all flip flop are triggered with same clock simultaneously.
2. Synchronous counter is faster than asynchronous counter in operation.
3. Synchronous counter does not produce any decoding ~~errors~~.
4. Synchronous counter is also called parallel counter.
5. Synchronous counter examples are: Ring counter, Johnson Counter.

Asynchronous Counter:-

(1) In asynchronous counter, different flip flops are triggered with different clock, not simultaneously.
(2) Asynchronous Counter is slower than Synchronous counter in operation.
(3) Asynchronous counter produces decoding errors.
(4) Counter is also called serial counter.
(5) Asynchronous counter examples are: Ripple up counter, Ripple Down Counter.

5.13 State the need for a Register and list the four types of register.

A: Sequential logic circuits can be constructed to produce either simple edge-triggered flip-flops or more complex sequential circuits such as storage registers, shift registers, memory devices or counters.

(1) Serial In - serial out (SISO) shift register.

In the serial in-serial out shift register, data is input ~~signal~~ serially until it reaches the output. At this time, it exists in a serial manner as well. The shifting (movement) of the data flows from left to right in the register. Each shift is initiated with a clock cycle. The figure below shows a 4-bit siso register in operation.

(2) Serial In - parallel out (SIPO) shift register.

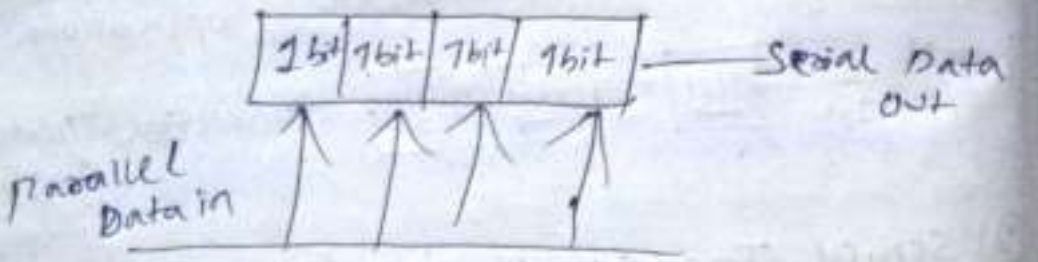
In a serial in-parallel out shift register, the data is input serially one bit at a time and output in a parallel form. Each bit is shifted on its own clock pulse. The table below shows an illustration of a 4-bit SIPO shift register with each clock pulse.

Clear	CLK 0	CLK 1	CLK 2	CLK 3
1001	0	0	0	0
	1	0	0	0
	0	1	0	0
	0	0	1	0
	1	0	0	1

On the first clock pulse (CLK 0), digit 1 on the right is loaded. On the second clock pulse (CLK 1), digit 1, which is already loaded, is shifted right and digit 0 is loaded behind it. This shifting and loading continues on subsequent clock pulses until all inputs are loaded and output.

3. Parallel In-Serial out (PISO) shift Register:

A:- The parallel in-serial out shift register receives the data input in parallel batches on every clock pulse and the data is shifted and output serially. This can be seen in the figure here:



4. Parallel-In Parallel-out shift register (PIPO):-

A:- The shift register, which allows parallel input (data is given separately to each flip flop and in a simultaneous manner) and also produces a parallel output is known as parallel-In parallel-out shift register.