

UNIT I	BASIC CONCEPTS	9
Review of number systems-representation-conversions, Review of Boolean algebra- theorems, sum of product and product of sum simplification, canonical forms min term and max term, Simplification of Boolean expressions-Karnaugh map, completely and incompletely specified functions, Implementation of Boolean expressions using universal gates ,Tabulation methods.		
UNIT II	COMBINATIONAL LOGIC CIRCUITS	9
Problem formulation and design of combinational circuits - Code-Converters, Half and Full Adders, Binary Parallel Adder – Carry look ahead Adder, BCD Adder, Magnitude Comparator, Decoder, Encoder, Priority Encoder, Mux/Demux, Case study: Digital trans-receiver / 8 bit Arithmetic and logic unit, Parity Generator/Checker, Seven Segment display decoder		
UNIT III	SYNCHRONOUS SEQUENTIAL CIRCUITS	9
Latches, Flip flops – SR, JK, T, D, Master/Slave FF, Triggering of FF, Analysis and design of clocked sequential circuits – Design - Moore/Mealy models, state minimization, state assignment,lock - out condition circuit implementation - Counters, Ripple Counters, Ring Counters, Shift registers, Universal Shift Register. Model Development: Designing of rolling display/real time clock		
UNIT IV	ASYNCHRONOUS SEQUENTIAL CIRCUITS	9
Stable and Unstable states, output specifications, cycles and races, state reduction, race free assignments, Hazards, Essential Hazards, Fundamental and Pulse mode sequential circuits, Design of Hazard free circuits.		
UNIT V	LOGIC FAMILIES AND PROGRAMMABLE LOGIC DEVICES	9
Logic families- Propagation Delay, Fan - In and Fan - Out - Noise Margin - RTL ,TTL,ECL, CMOS - Comparison of Logic families - Implementation of combinational logic/sequential logic design using standard ICs, PROM, PLA and PAL, basic memory, static ROM,PROM,EPROM,EEPROM EAPROM.		

45 PERIODS
30 PERIODS

PRACTICAL EXERCISES :

1. Design of adders and subtractors & code converters.
2. Design of Multiplexers & Demultiplexers.
3. Design of Encoders and Decoders.
4. Design of Magnitude Comparators
5. Design and implementation of counters using flip-flops
6. Design and implementation of shift registers.

COURSE OUTCOMES :

At the end of the course the students will be able to

CO1: Use Boolean algebra and simplification procedures relevant to digital logic.

CO2: Design various combinational digital circuits using logic gates.

CO3:Analyse and design synchronous sequential circuits.

CO4: Analyse and design asynchronous sequential circuits. .

CO5: Build logic gates and use programmable devices

TOTAL:75 PERIODS

TEXTBOOKS :

1. M. Morris Mano and Michael D. Ciletti, 'Digital Design', Pearson, 5th Edition, 2013.(Unit - I - V)

Review of Number system:-

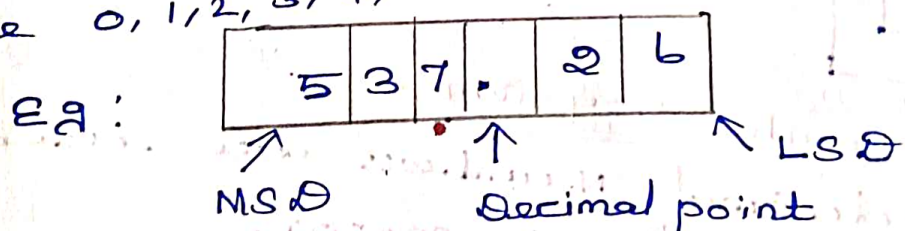
* Number system is a basis for counting items. we human use decimal number system 0, 1, 2, 3, 4, 5, 6, 7, 8 & 9. Modern computer uses binary number system 0 and 1

* There are four different types of Number system that is

- ↳ Decimal Number system
- ↳ Binary Number system
- ↳ Octal Number system
- ↳ Hexa decimal Number system

Decimal Number system:-

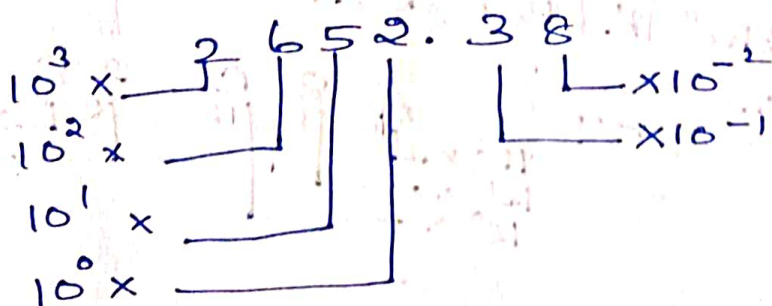
* The base or radix of the decimal Number system is 10. The decimal number are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.



↳ MSD → Most significant digit

↳ LSD → Least significant digit

Eg : (2652.38)₁₀



Binary Number system :-

* A Number system that uses only two digits 0 and 1 is called a binary Number system

* The binary system is also called a base two system

* The symbol 0 and 1 are known as bits

* The weight or place value of each position can be expressed in terms of power of 2 and as $2^0, 2^1, 2^2$... etc.

Eg $(11010.011)_2$

Octal Number system :-

* The Base (Radix) of octal number system is 8. The octal Numbers are 0, 1, 2, 3, 4, 5, 6, and 7

Eg $(765.42)_8$

$8^2 \times$ $8^1 \times$ $8^0 \times$ $8^{-1} \times$ $8^{-2} \times$

Hexadecimal Number system :-

* Hexadecimal Numbers are used extensively in Microprocessor.

* The hexadecimal Number system has a base of 16.

* It uses the digit 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 plus the A, B, C, D, E and F as 16 digit symbols.

Eg $(A8C.28)_{16}$

$16^2 \times$ $16^1 \times$ $16^0 \times$ $16^{-1} \times$ $16^{-2} \times$

Convert the following decimal numbers to binary, octal and Hexadecimal

i) $(455)_{10}$ ii) $(256.22)_{10}$

Solution

Decimal to Binary

i) $(455)_{10}$

$$\begin{array}{r}
 2 \overline{) 455} \\
 \underline{2 \ 27} \quad -1 \\
 2 \overline{) 113} \\
 \underline{2 \ 56} \quad -1 \\
 2 \overline{) 28} \\
 \underline{2 \ 14} \quad -0 \\
 2 \overline{) 7} \\
 \underline{2 \ 3} \quad -1 \\
 1 \quad -1
 \end{array}$$

Binary Number

$(455)_{10} = (111000111)_2$

ii) $(256.22)_{10}$

$$\begin{array}{r}
 2 \overline{) 256} \\
 \underline{2 \ 128} \quad -0 \\
 2 \overline{) 64} \quad -0 \\
 2 \overline{) 32} \quad -0 \\
 2 \overline{) 16} \quad -0 \\
 2 \overline{) 8} \quad -0 \\
 2 \overline{) 4} \quad -0 \\
 2 \overline{) 2} \quad -0 \\
 1 \quad -0
 \end{array}$$

$(256)_{10} = 100000000$

Integers

$$\begin{array}{l}
 2 \times 2 = 0.44 - 0 \\
 \downarrow \\
 0.44 \times 2 = 0.88 - 0 \\
 \downarrow \\
 0.88 \times 2 = 1.76 - 1 \\
 \downarrow \\
 0.76 \times 2 = 1.52 - 1
 \end{array}$$

$(0.22)_{10} = 0.0011..._2$

$(256.22)_{10} = (100000000.0011)_2$

Decimal to octal

$$8 \overline{) 455}$$
$$\underline{56} \quad -7$$

$$\underline{7} \quad -0 \uparrow$$

$$(455)_{10} = (707)_8$$

$$(256.22)_{10}$$

$$8 \overline{) 256}$$
$$8 \overline{) 32} \quad -0$$

$$\underline{4} \quad -0 \uparrow$$

$$\therefore (256)_{10} = (400)_8$$

$$0.22 \times 8 = 1.76 \quad -1$$

$$0.76 \times 8 = 6.08 \quad -6$$

$$0.08 \times 8 = 0.64 \quad -0$$

$$0.64 \times 8 = 5.12 \quad -5$$

$$(0.22)_{10} = (0.1605)_8$$

$$(256.22)_{10} = (400.1605)_8$$

Decimal to Hexadecimal

$$16 \overline{) 455}$$

$$16 \overline{) 28} \quad -7$$

$$\underline{1} \quad -12 (C) \uparrow$$

$$(455)_{10} = (1C7)_{16}$$

$$(256.22)_{10}$$

$$16 \overline{) 256}$$

$$16 \overline{) 16} \quad -0$$

$$\underline{1} \quad -0 \uparrow$$

$$(256)_{10} = (100)_{16}$$

$$0.22 \times 16 = 3.52 \quad -3$$

$$0.52 \times 16 = 8.32 \quad -8$$

$$0.32 \times 16 = 5.12 \quad -5$$

$$0.12 \times 16 = 1.92 \quad -1$$

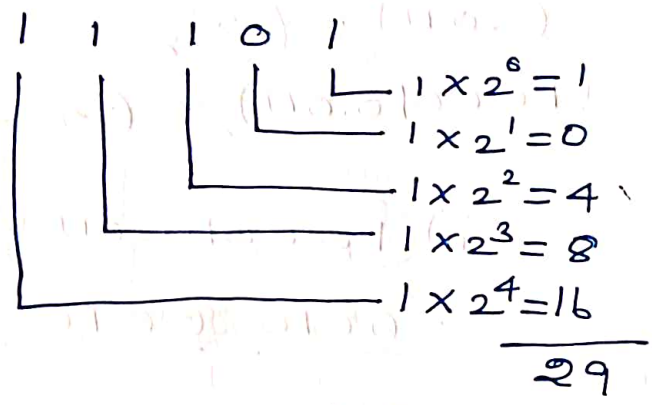
$$(0.22)_{10} = (0.3851)_{16}$$

$$(256.22)_{10} = (100.3851)_{16}$$

Convert the following binary number to decimal, octal and hexadecimal solution

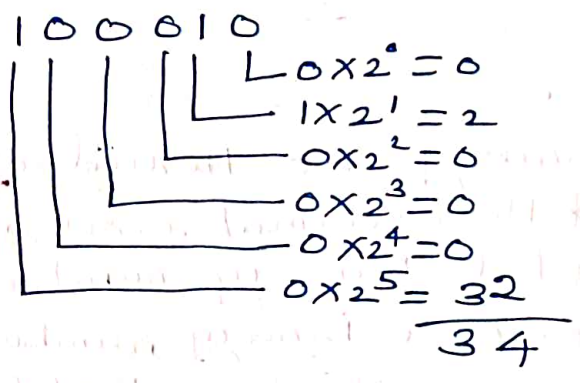
Binary to Decimal

$(11101)_2$

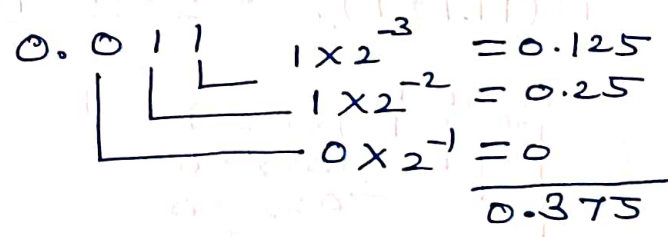


$(11101)_2 = (29)_{10}$

$(100010.011)_2$



$(100010) = (34)_{10}$



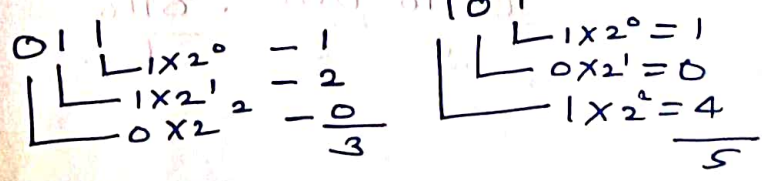
$(100010.011) = (34.375)_{10}$

Binary to Octal

i) $(11101)_2$

3 bit binary number split into 3-bits from right side

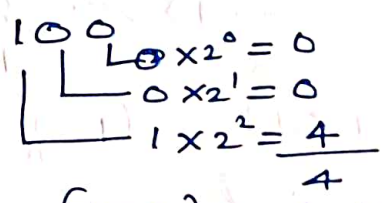
$11101 \rightarrow \underline{011} \ \underline{101}$



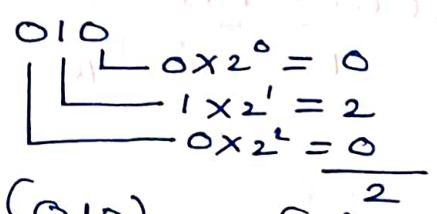
$(\underline{011} \ \underline{101})_2 = (35)_8$

ii) $(100010.011)_2$

$(100010) = \frac{100}{1} \ \frac{010}{2}$



$(100)_2 = (4)_8$



$(010)_2 = (2)_8$

fractional value
split the 3 bit from
left side

$$\begin{array}{r} \cdot 011 \\ \begin{array}{l} \text{---} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \text{---} \end{array} \\ \begin{array}{l} 1 \times 2^0 = 1 \\ 1 \times 2^1 = 2 \\ 0 \times 2^2 = 0 \\ \hline 3 \end{array} \end{array}$$

$$(.011)_2 = (.3)_8$$

$$(100010.011)_2 = (42.3)_8$$

Binary to Hexadecimal

* Hexadecimal is set of
4 bit binary number
* split binary number
into 4 bit from right side

$$i) (11101)_2 = \underline{0001} \quad \underline{1101}$$

$$\begin{array}{r} 0001 \\ \begin{array}{l} \text{---} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \text{---} \end{array} \\ \begin{array}{l} 1 \times 2^0 = 1 \\ 0 \times 2^1 = 0 \\ 0 \times 2^2 = 0 \\ 0 \times 2^3 = 0 \\ \hline 1 \end{array} \end{array}$$

$$(0001)_2 = (1)_{16}$$

$$\begin{array}{r} 1101 \\ \begin{array}{l} \text{---} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \text{---} \end{array} \\ \begin{array}{l} 1 \times 2^0 = 1 \\ 0 \times 2^1 = 0 \\ 1 \times 2^2 = 4 \\ 1 \times 2^3 = 8 \\ \hline 13 \end{array} \end{array}$$

$$(1101)_2 = (13)_{16} = (D)_{16}$$

$$\therefore \frac{(0001)}{(1)} \quad \frac{(1101)}{(D)}_2 = (1D)_{16}$$

$$ii) (100010.011)_2$$

$$= \underline{0010} \quad \underline{0010}$$

$$\begin{array}{r} 0010 \\ \begin{array}{l} \text{---} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \text{---} \end{array} \\ \begin{array}{l} 0 \times 2^0 = 0 \\ 1 \times 2^1 = 2 \\ 0 \times 2^2 = 0 \\ 0 \times 2^3 = 0 \\ \hline 2 \end{array} \end{array}$$

$$(0010)_2 = (2)_{16}$$

$$0.1110$$

$$\begin{array}{r} 0.1110 \\ \begin{array}{l} \text{---} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \text{---} \end{array} \\ \begin{array}{l} 0 \times 2^0 = 0 \\ 1 \times 2^1 = 2 \\ 1 \times 2^2 = 4 \\ 0 \times 2^3 = 0 \\ \hline 6 \end{array} \end{array}$$

$$(.1110)_2 = (.6)_{16}$$

$$(100010.011)_2 = (22.6)_{16}$$

$$\frac{0010}{2} \quad \frac{0010}{2} \quad \cdot \quad \frac{0110}{6}$$

Convert the following octal number to decimal binary and hexadecimal.

- i) $(35)_8$ ii) $(42.3)_8$

- ii) $(42.3)_8$

Octal to Decimal.

$$\begin{array}{r} 35 \\ \left\{ \begin{array}{l} 5 \times 8^0 = 5 \\ 3 \times 8^1 = 24 \end{array} \right. \\ \hline 29 \end{array}$$

$(35)_8 = (29)_{10}$

$$\begin{array}{r} 42 \\ \left\{ \begin{array}{l} 2 \times 8^0 = 2 \\ 4 \times 8^1 = 32 \end{array} \right. \\ \hline 34 \end{array}$$

$(42)_8 = (34)_{10}$

$\cdot 3 \left\{ \begin{array}{l} 3 \times 8^{-1} = 0.375 \end{array} \right.$

$(42.3)_8 = (34.375)_{10}$

Octal to Binary

Octal number is set of 3-bit binary number

$$\begin{array}{r} 3 \quad 5 \\ \downarrow \quad \downarrow \\ 2 \left| \begin{array}{r} 3 \\ 1 \end{array} \right. - 1 \\ \hline 3 = 011 \end{array} \quad \begin{array}{r} 2 \left| \begin{array}{r} 5 \\ 2 \\ 1 \end{array} \right. - 1 \\ \hline 5 = 101 \end{array}$$

$3 = 011$ $5 = 101$

$\therefore (35)_8 = (011101)_2$

- ii) $(42.3)_8$

$$\begin{array}{r} 4 \quad 2 \\ \downarrow \quad \downarrow \\ 2 \left| \begin{array}{r} 4 \\ 2 \\ 1 \end{array} \right. - 0 \\ \hline 4 = 100 \end{array} \quad \begin{array}{r} 2 \left| \begin{array}{r} 2 \\ 1 \end{array} \right. - 0 \\ \hline 2 = 010 \end{array}$$

$4 = 100$

fractional value
 0.3

$$2 \left| \begin{array}{r} 3 \\ 1 \end{array} \right. - 1$$

$3 = 011$

$(42.3)_8 = (100010.011)_2$

Octal to Hexa decimal

* There are no direct octal to hexadecimal
* convert octal to binary and binary to hexadecimal

$(35)_8 = (11101)_2$

$= \frac{0001}{1} \quad \frac{1101}{13(10)}$

$(35)_8 = (1B)_{16}$

convert between

binary and binary

$(42)_8 = (100010)_2$

$= \frac{0010}{2} \quad \frac{0010}{2}$

fractional value

$(.3)_8 = 0.011$

$0.011 = 0.0110$

$(42.3)_8 = (22.4)_{16}$

convert the following hexadecimal number to decimal binary and octal

i) $(10)_{16}$ ii) $(22.6)_{16}$

sol

Hexadecimal to Decimal

i) 10_{16}

$$\begin{array}{l} 1 \quad 0 \quad 13 \\ \left[\begin{array}{l} 13 \times 16^0 = 13 \\ 1 \times 16^1 = 16 \\ \hline 29 \end{array} \right. \end{array}$$

$(10)_{16} = (29)_{10}$

ii) 22.6_{16}

$$\begin{array}{l} 2 \quad 2 \\ \left[\begin{array}{l} 2 \times 16^0 = 2 \\ 2 \times 16^1 = 32 \\ \hline 34 \end{array} \right. \end{array}$$

$(22)_{16} = (34)_{10}$

fractional number

$.6_{16} \quad \left[\begin{array}{l} 6 \times 16^{-1} = 0.375 \end{array} \right.$

$(22.6)_{16} = (34.375)_{10}$

Hexadecimal to Binary

* hexadecimal is a set of 4 bit binary number

$10 = \frac{1}{0001} \frac{13}{1101}$

$(10)_{16} = (00011101)_2$

22.6

$= \frac{2}{0010} \frac{2}{0010} \frac{6}{0110}$

$\therefore (22.6)_{16} = (00100010.0110)_2$

Hexadecimal to octal

* hexadecimal to octal

No conversion

* convert hexadecimal to binary & binary to octal

$(10)_{16} = (00011101)_2$

$\frac{000}{0} \quad \frac{011}{3} \quad \frac{101}{5}$

$(10)_{16} = (35)_8$

ii) $(22.6)_{16} = (00100010.0110)_2$

$\frac{100}{4} \quad \frac{010}{2} \quad \frac{011}{3}$

$\therefore (22.6)_{16} = (42.3)_8$

Number with different bases

Decimal Base 10	Binary Base 2	Octal Base 8	Hexa decimal Base 16
00	0000	00	00
01	0001	01	01
02	0010	02	02
03	0011	03	03
04	0100	04	04
05	0101	05	05
06	0110	06	06
07	0111	07	07
08	1000	10	08
09	1001	11	09
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11

Arithmetic operations:-

* Arithmetic operations perform Addition, subtraction, multiplication and division of two or more numbers.

* In computers and general purpose processors the arithmetic operations are done by using binary numbers not by using decimal numbers.

Binary Additions:-

Rules for binary Addition

Augend (A)	+ Addend (B)	Carry (C)	Sum (S)	Result
0	+ 0	0	0	0
0	+ 1	0	1	1
1	+ 0	0	1	1
1	+ 1	1	0	10

Example

Add the binary numbers

i) 1101 and 0011

ii) 1100 and 1111

iii) 101.01 and 010.11

iv) 1110.011 and 0101.010

Sol:-

i) Binary Decimal

$$\begin{array}{r}
 \text{Carry} \rightarrow \\
 \begin{array}{r}
 111 \\
 1101 \\
 0011 \\
 \hline
 100000
 \end{array}
 \end{array}$$

13
03

16

Binary Decimal

$$\begin{array}{r}
 \text{Carry} \rightarrow \\
 \begin{array}{r}
 1100 \\
 1111 \\
 \hline
 11011
 \end{array}
 \end{array}$$

12
15

27

iii) 101.01 and 010.11

iv) 1110.011 and 0101.010

Binary	Decimal
1111	
101.01	5.25
010.11	2.75
<hr/>	<hr/>
1000.00	8.00

1110.011	14.375
0101.010	5.25
<hr/>	<hr/>
10011.101	19.625

Binary subtraction:-

* subtraction is the inverse operation of addition.

* To subtract two numbers it is necessary to establish a procedure for subtracting a larger from a small digit subtract the binary numbers

i) 101 from 111

ii) 1100 and 1101

sol:-

Binary	Decimal
111	7
- 101	- 5
<hr/>	<hr/>
010	2

Binary	Decimal
1101	13
- 1100	- 12
<hr/>	<hr/>
0001	1

Binary Multiplication:-

* The Multiplication of binary numbers is done in the same manner as the Multiplication of decimal numbers

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

Multiply the binary numbers 10110 and 1100

Sol

$$\begin{array}{r} 10110 \\ 1100 \\ \hline 00000 \\ 00000 \\ 10110 \\ 10110 \\ \hline 100001000 \end{array}$$

Binary Division:-

* The process for dividing one binary number by another is the same that which is followed for decimal numbers
Divide the binary number: 11001 and 101

Sol

$$\begin{array}{r} 101 \overline{) 11001} \\ \underline{101} \\ 0101 \\ \underline{101} \\ 000 \end{array}$$

Error detection and correction

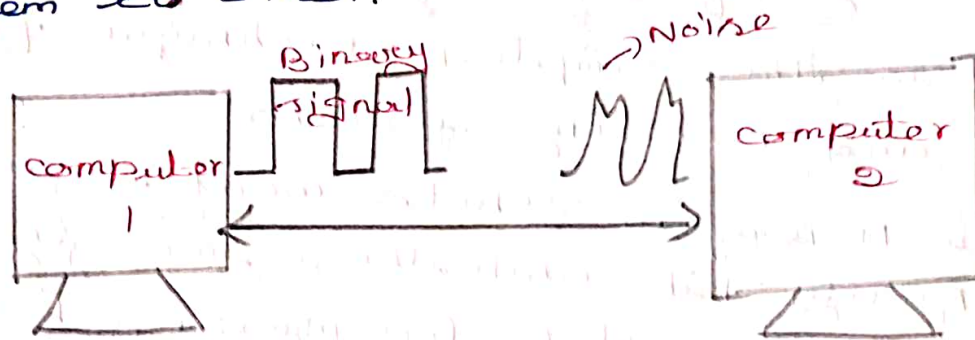
Error:-

* Error is condition when the output information does not match with the input information.

* During the transmission digital signal

7

suffer from noise that can introduce error in binary bits travelling from one system to other.



Error detecting codes:-

* whenever a message is transmitted it may get by noise or data may get corrupted.

* To avoid this error detecting codes which are additional data added to given digital message to help detect if an error occurred during transmission of message

* A simple example of error detecting code is parity check.

Error correcting codes:-

* Along with error - detecting code we can also pass some data to figure out the original message from corrupt message received

* This type of code is called Error correcting codes

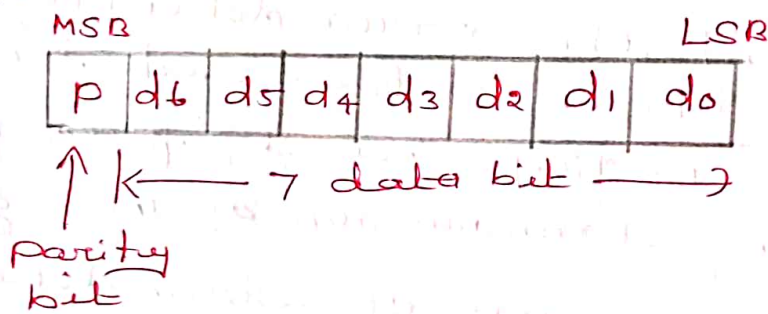
* such codes also detect the exact location of the corrupt bit

* In error correcting codes parity check has simple way to detect errors

and to determine the corrupt bit location.

Parity checking of Error Detection:-

- * It is the simplest technique for detecting and correcting errors.
- * The MSB of an 8 bits word is used as the parity bit and the remaining 7 bits are used as data (or) message bit.
- * The parity of 8 bits transmitted word can be either even parity or odd parity.



Even parity:-

- * Even parity means the number of 1's in the given word including the parity bit should be even (2, 4, 6, ...).

Odd parity:-

- * Odd parity means the number of 1's in the given word including the parity bit should be odd (1, 3, 5, ...).

Hamming Code :-

- * Hamming code is useful for both detection and correction of errors present in the received data.

* This code uses multiple parity bits and we have to place these parity bits in the positions of powers of 2

* The minimum value of k for which the following relation is correct valid is nothing but the required number of parity bits

$$2^k \geq n + k + 1$$

n → number of binary codes

k → number of parity bit

$b_{n+k}, b_{n+k-1}, \dots, b_3, b_2, b_1$, & parity bits

P_k, P_{k-1}, \dots, P_1

* k parity bit positions we can place the n bits of binary code

* we can use either even parity or odd parity while forming a Hamming code.

* The same parity technique should be used in order to find any error present in the received data

Procedure for finding parity bits:-

* Find the value of P_1 based on the number of ones present in bit position b_3, b_5, b_7 and so on. (2^0)

* Find the value of P_2 based on the number of ones present in bit position b_3, b_6, b_7 and so on. (2^1)

* Find the value of P_3 based on the number of ones present in bit position b_5, b_6, b_7 so on (2^2)

* similarly find other values of parity bit

Example:-

* let us find the Hamming code of binary code $d_4, d_3, d_2, d_1 = 1000$, consider even parity bits.

Given binary code is $n = 4$

$$2^k \geq n + k + 1$$

$$2^k \geq 4 + k + 1$$

$$2^k \geq 5 + k$$

* minimum value of k that satisfied the above relation is 3.

Hence require 3 parity bits P_1, P_2 and P_3 . Hamming code will be 7

7 bit Hamming code is

$$b_7 b_6 b_5 b_4 b_3 b_2 b_1 = d_4 d_3 d_2 P_3 d_1 P_2 b_1$$

$$P_1 = b_7 \oplus b_5 \oplus b_3 = 1 \oplus 0 \oplus 0 = 1$$

$$P_2 = b_7 \oplus b_6 \oplus b_3 = 1 \oplus 0 \oplus 0 = 1$$

$$P_3 = b_7 \oplus b_6 \oplus b_5 = 1 \oplus 0 \oplus 0 = 1$$

substituting these parity bits the

$$b_7 b_6 b_5 b_4 b_3 b_2 b_1 = 1001011$$

Boolean Algebra:-

* Boolean Algebra is a branch of mathematics that deals with operations on logical values with binary variables

* variables are represented as binary numbers. true = 1 false = 0

Laws of boolean Algebra:-

* It is most common law used to formulate various algebraic structures

* Three basic laws of Boolean algebra

↳ * Commutative Laws

↳ * Associative Laws

↳ * Distributive Laws

Commutative Law:-

* Law 1 $\rightarrow A \cdot B = B \cdot A$

* Law 2 $\rightarrow A + B = B + A$

Law 1 $A \cdot B = B \cdot A$

represents the Multiplication (AND) operation

operation

A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

LHS

A	B	$B \cdot A$
0	0	0
0	1	0
1	0	0
1	1	1

RHS

LHS = RHS

Hence proved

Law 2 $A+B = B+A$

(+) represent

OR operation

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

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

Associative Laws:-

* Law 1 $\rightarrow A+(B+C) = (A+B)+C$

* Law 2 $\rightarrow (A \cdot B) \cdot C = A \cdot (B \cdot C)$

Law 1 $A+(B+C) = (A+B)+C$

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

=

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

Law 2 $(A \cdot B) \cdot C = A \cdot (B \cdot C)$

A	B	C	A.B	(A.B).C
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

=

A	B	C	B.C	A.(B.C)
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Distributive Law:-

$$A \cdot (B + C) = AB + AC$$

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

=

A	B	C	A.B	A.C	A.B+AC
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

De Morgan's Theorems

* These are two ^{the important} important parts of Boolean Algebra

of Boolean Algebra

↳ theorem 1 $\overline{AB} = \overline{A} + \overline{B}$

↳ theorem 2 $\overline{A+B} = \overline{A} \cdot \overline{B}$

Theorem 1 : $\overline{AB} = \overline{A} + \overline{B}$

* The complement of a product is equal to the sum of the complement.

* that is AND operation is equal to the OR operation of the complements

A	B	AB	\overline{AB}
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

=

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

Theorem 2 $\overline{A+B} = \overline{A} \cdot \overline{B}$

* complement of sum is equal to the product of the complements

* OR operation is equal to the AND operation

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

=

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

Boolean Relation	Dual theorem
$A + 0 = A$	$A \cdot 1 = A$
$A + 1 = 1$	$A \cdot 0 = 0$
$A + A = A$	$A \cdot A = A$
$A + \bar{A} = 1$	$A \cdot \bar{A} = 0$
$A + AB = A$	$A \cdot (A+B) = A$
$A + \bar{A}B = A+B$	$A(\bar{A}+B) = AB$
$(A+B)(A+C) = A+BC$	$AB+AC = A(B+C)$
$AB + \bar{A}C + BC = AB + \bar{A}C$	$(A+B)(\bar{A}+C)(B+C) = (A+B)(\bar{A}C)$

Example

simplify the following boolean expression

a) $ABC\bar{D} + A\bar{B}CD$

b) $AB + ABC + ABC(D+E)$

c) $\overline{AB + \bar{A} + AB}$

sol

a) $ABC\bar{D} + A\bar{B}CD$

$$= AC\bar{D}(B + \bar{B})$$

$$= AC\bar{D} \cdot 1$$

$$= AC\bar{D}$$

b) $AB + ABC + A\overbrace{B(C+D+E)}$

$$= AB + ABC + ABD + ABE$$

$$= ABC(1+C) + ABD + ABE$$

$$= AB + ABD + ABE$$

$$\therefore B + \bar{B} = 1$$

$$\therefore 1 + C = 1$$

$$= AB(1 + D) + ABC$$

$$\therefore 1 + D = 1$$

$$= AB + ABC$$

$$\therefore 1 + C = 1$$

$$= AB(1 + C)$$

$$= AB$$

$$c) \overline{AB + \bar{A} + AB}$$

4

Apply De Morgan theorem

$$= \overline{\bar{A} + \bar{B} + \bar{A} + AB}$$

$$\therefore A \cdot \bar{A} = \bar{A}$$

$$= \overline{\bar{A} + \bar{B} + \bar{A} + AB}$$

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

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

$$A \cdot \bar{A} = 0$$

$$= A\bar{A}B + AB\bar{B}$$

$$= 0 + 0$$

$$= 0$$

Karnaugh map minimization (K-Map)

* A Karnaugh map (K-Map) is a method of simplifying Boolean algebra expressions.

* It is a graphical representation of a Boolean function.

* It is used in digital electronics and computer science.

* Simplify Boolean expressions and minimize the number of gates required to implement a logic circuit.

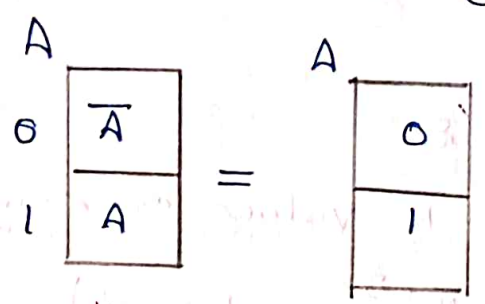
One variable K-map :-

* K-Map is made up of squares.

* Each square represent the 2^n possible value.

$n=1$

$2^n = 2^1 = 2$ (two values of square)
(that is 0 & 1)



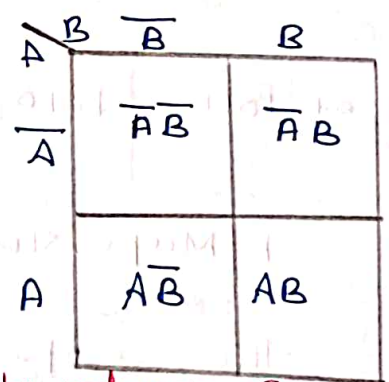
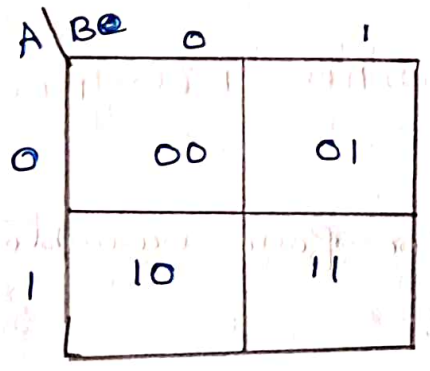
K-map structure for one variable

Two variable K-Map

* Two variables A, B and $n=2$

$2^n = 2^2 = 2 \times 2 = 4$

It has two variables and 4 squares of the values 00, 01, 10, 11



K-Map structure for two variable

Three variable K-Map

* Three variable A, B, C and $n=3$

$2^n = 2^3 = 2 \times 2 \times 2 = 8$

* It has three variables and 8-square

A \ BC	00	01	11	10
0	000	001	011	010
1	100	101	111	110

=

A \ BC	$\overline{B}\overline{C}$	$\overline{B}C$	BC	$B\overline{C}$
\overline{A}	$\overline{A}\overline{B}\overline{C}$	$\overline{A}\overline{B}C$	$\overline{A}BC$	$\overline{A}B\overline{C}$
A	$A\overline{B}\overline{C}$	$A\overline{B}C$	ABC	$AB\overline{C}$

K-Map structure for three variable

Four variable K-Map:-

*Variables A B C and D

$$2^n = 2^4 = 2 \times 2 \times 2 \times 2 = 16 \text{ values or squares}$$

AB \ CD	00	01	11	10
00	0000	0001	0011	0010
01	0100	0101	0111	0110
11	1100	1101	1111	1110
10	1000	1001	1011	1010

=

AB \ CD	00	01	11	10
00	$\overline{A}\overline{B}\overline{C}\overline{D}$	$\overline{A}\overline{B}\overline{C}D$	$\overline{A}\overline{B}C\overline{D}$	$\overline{A}\overline{B}CD$
01	$\overline{A}B\overline{C}\overline{D}$	$\overline{A}B\overline{C}D$	$\overline{A}BC\overline{D}$	$\overline{A}BCD$
11	$AB\overline{C}\overline{D}$	$AB\overline{C}D$	$ABC\overline{D}$	$ABCD$
10	$A\overline{B}\overline{C}\overline{D}$	$A\overline{B}\overline{C}D$	$A\overline{B}C\overline{D}$	$A\overline{B}CD$

K-Map structure for four variable

Example

Minimize the expression

$$Y = \overline{A}\overline{B}C + \overline{A}BC + \overline{A}B\overline{C} + ABC + AB\overline{C}$$

sol

$$y = 001 + 010 + 010 + 111 + 110$$

	BC	00	01	11	10
A	0	000	001	011	010
	1	100	101	111	110

$\bar{A}C$ (points to the group of 1s in the first row)
 B (points to the group of 1s in the second column)

$$Y = \bar{A}C + B$$

Minimize the expression using k-map

$$Y = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + \bar{A}BCD + A\bar{B}\bar{C}\bar{D}$$

$$Y = 0000 + 011\bar{D} + 011D + 1111 + 1110 + 1011$$

	CD	00	01	11	10
AB	00	0000	0001	0011	0010
	01	0100	0101	0111	0110
	11	1100	1101	1111	1110
	10	1000	1001	1011	1010

$\bar{A}\bar{B}\bar{C}\bar{D}$ (points to the circled 1 in the top-left cell)
 BC (points to the group of 1s in the second column)
 ACD (points to the group of 1s in the third column)

$$Y = \bar{A}\bar{B}\bar{C}\bar{D} + BC + ACD$$

Simplify the Boolean expression

$$Y = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + \bar{A}BCD + A\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D + A\bar{B}C\bar{D} + A\bar{B}CD$$

$$y = 0001 + 0101 + 0111 + 0110 + 1100 + 1101 + 1111 + 1011$$

AB \ CD	00	01	11	10
00	0000	0001	0011	0010
01	0100	0101	0111	0110
11	1100	1101	1111	1110
10	1000	1001	1011	1010

Annotations in the K-map:
 - $\bar{A}\bar{C}\bar{D}$ (grouped in cell 0001)
 - $\bar{A}BC$ (grouped in cells 0101, 0111, 0110)
 - BD (grouped in cells 0110, 1110)
 - ABC (grouped in cells 1101, 1111)
 - ACD (grouped in cells 1011, 1111)

$$y = \bar{A}\bar{C}\bar{D} + ABC\bar{C} + ACD + \bar{A}BC + BD$$

Quine McCluskey method (or) tabular method

* Quine and E.J McCluskey developed an tabular method to simplify the Boolean expression.

* This method is called Quine McCluskey or tabular method.

Example

simplify the given expression using Quine McCluskey method.

$$f(a, b, c, d) = \sum m(0, 1, 2, 3, 8, 9)$$

$$\bar{a}\bar{b} + \bar{a}b + a\bar{b}$$

sol

sol
step 1: List all the minterms in the binary form

Minterm	Binary representation			
	8	4	2	1
m_0	0	0	0	0
m_1	0	0	0	1
m_2	0	0	1	0
m_3	0	0	1	1
m_8	1	0	0	0
m_9	1	0	0	1

step 2: Arrange the minterms according to the Number of 1s and also split them

Minterm	Binary Representation	Group
m_0	0000	Number of 1s zero
m_1	0001	Number of 1s zero
m_2	0010	
m_8	1000	
m_3	0011	Number of 1s two
m_9	1001	

step 3: Compare each binary number with next higher group.

* If they differ by only one position and put check mark

step 4:- Repeat step 3 for the resultant column and continue these cycles until no further elimination of variables takes place

Minterm	Binary representation				Minterms	Binary representation			
	a	b	c	d		a	b	c	d
0, 1	0	0	0	-	0, 1, 8, 9	-	0	0	-
0, 2	0	0	-	0	0, 1, 2, 3	0	0	-	-
0, 8	-	0	0	0					
1, 3	0	0	-	1					
1, 9	-	0	0	1					
2, 3	0	0	1	-					
8, 9	1	0	0	-					

step 5:- List the prime Implicants

Prime Implicants	Binary representation			
	a	b	c	d
0, 1, 8, 9	-	0	0	-
0, 1, 2, 3	0	0	-	-

step 6:- Form prime Implicant chart

Prime Implicants	Minterm					
	0	1	2	3	8	9
0, 1, 8, 9 ✓	X	X			X	X
0, 1, 2, 3 ✓	X	X	X	X		

step 7: Select the minimum number of primes that must cover all the minterms

$$f(a, b, c, d) = \bar{b}\bar{c} + \bar{a}\bar{b}$$

Minimize the given ~~map~~ expression using tabulation method

$$F(x_1, x_2, x_3, x_4) = \sum(0, 5, 7, 8, 9, 10, 11, 14, 15)$$

sol

step 1 :- List all minterms in the binary form

Minterm	Binary representation			
	8	4	2	1
m ₀	0	0	0	0
m ₅	0	1	0	1
m ₇	0	1	1	1
m ₈	1	0	0	0
m ₉	1	0	0	1
m ₁₀	1	0	1	0
m ₁₁	1	0	1	1
m ₁₄	1	1	1	0
m ₁₅	1	1	1	1

step 2 :- Arrange the minterms according to the number of 1's

Minterm	Binary Representation	Group
m ₀	0 0 0 0	Number of 1's zero
m ₈	1 0 0 0	
m ₅	0 1 0 1	Number of 1's two
m ₉	1 0 0 1	
m ₁₀	1 0 1 0	
m ₇	0 1 1 1	Number of 1's three
m ₁₁	1 0 1 1	
m ₁₄	1 1 1 0	
m ₁₅	1 1 1 1	Number of 1's four

step 3:- Compare each binary number with higher group

step 4:- Repeat step 3

Min terms	Binary representation	Minterms	Binary Representation
0, 8	- 0 0 0	8, 9, 10, 11	1 0 - -
8, 9	1 0 0 - ✓	8, 10, 9, 11	1 0 - -
8, 10	1 0 - 0 ✓		
5, 7	0 1 - 1	10, 11, 14, 15	1 - 1 -
9, 11	1 0 - 1 ✓	10, 14, 11, 15	1 - 1 -
10, 11	1 0 1 - ✓		
10, 14	1 - 1 0 ✓		
7, 15	- 1 1 1		
11, 15	1 - 1 1 ✓		
14, 15	1 1 1 - ✓		

} Both are similar

} Both are similar

step 5:- List the prime Implicant

Prime Implicant	Binary Representation			
	π_1	π_2	π_3	π_4
0, 8 ✓	-	0	0	0
5, 7 ✓	0	1	-	1
7, 5	-	1	1	1
8, 9, 10, 11 ✓	1	0	-	-
10, 11, 14, 15 ✓	1	-	1	-

$\overline{\pi_2} \overline{\pi_3} \overline{\pi_4}$

$\overline{\pi_1} \pi_2 \pi_4$

$\pi_1, \overline{\pi_2}$

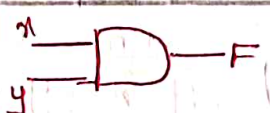

π_1, π_3


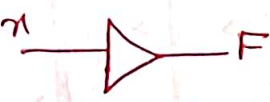



step 6 :- Prime Implicant Table


Prime Implicant	Min terms									
	0	5	7	8	9	10	11	14	15	
0, 8 ✓	x			x						
5, 7 ✓		x	x							
7, 15			x						x	
8, 9, 10, 11 ✓				x	x	x	x			
10, 11, 14, 15 ✓						x	x	x	x	
	✓	✓			✓			✓		

$$f(x_1, x_2, x_3, x_4) = \overline{x_2} \overline{x_3} x_4 + \overline{x_1} x_2 x_4 + x_1 \overline{x_2} + x_1 x_3$$

Digital Logic Gates

Name	Graphic symbol	Algebraic function	Truth Table																			
<p>AND</p> <p>* AND gate has two or more inputs</p>		$F = xy$ <p>* It is also called as product gate</p>	<table border="1"> <thead> <tr> <th colspan="2">Input</th> <th>output</th> </tr> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	Input		output	x	y	F	0	0	0	0	1	0	1	0	0	1	1	1	
Input		output																				
x	y	F																				
0	0	0																				
0	1	0																				
1	0	0																				
1	1	1																				
<p>OR</p> <p>* OR gates two or more inputs.</p>		$F = x + y$ <p>* It is also called as sum gate</p>	<table border="1"> <thead> <tr> <th colspan="2">Input</th> <th>output</th> </tr> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	Input		output	x	y	F	0	0	0	0	1	1	1	0	1	1	1	1	
Input		output																				
x	y	F																				
0	0	0																				
0	1	1																				
1	0	1																				
1	1	1																				

Name	Graphic symbol	Algebraic function	Truth table																			
Inverter (or) NOT * only one input.	 <p>* Input is 0, then the output is 1</p>	$F = \bar{x}$ * It is called Inverter gate (or) Complementary gate	<table border="1"> <thead> <tr> <th>Input x</th> <th>output F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </tbody> </table>	Input x	output F	0	1	1	0													
Input x	output F																					
0	1																					
1	0																					
Buffer * only one input.	 <p>* Input is 0, then output is 0</p>	$F = x$	<table border="1"> <thead> <tr> <th>Input x</th> <th>output F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> </tr> </tbody> </table>	Input x	output F	0	0	1	1													
Input x	output F																					
0	0																					
1	1																					
NAND * Two or more input more gate * combination of NOT + AND gate	 <p>* The output of NAND gate high only when one of the input is low</p>	$F = \overline{xy}$ * It is also called universal gate.	<table border="1"> <thead> <tr> <th colspan="2">Input</th> <th>output F</th> </tr> <tr> <th>x</th> <th>y</th> <th></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	Input		output F	x	y		0	0	1	0	1	1	1	0	1	1	1	0	
Input		output F																				
x	y																					
0	0	1																				
0	1	1																				
1	0	1																				
1	1	0																				
NOR * Two or more input * combination of NOT + OR gate	 <p>* output is high when all inputs are low</p>	$F = \overline{x+y}$ * It is also called universal gate	<table border="1"> <thead> <tr> <th colspan="2">Input</th> <th>output F</th> </tr> <tr> <th>x</th> <th>y</th> <th></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	Input		output F	x	y		0	0	1	0	1	0	1	0	0	1	1	0	
Input		output F																				
x	y																					
0	0	1																				
0	1	0																				
1	0	0																				
1	1	0																				
Exclusive-OR (XOR)	 <p>* The output is high only when odd number of input are high</p>	$F = x\bar{y} + \bar{x}y$ $= x \oplus y$	<table border="1"> <thead> <tr> <th colspan="2">Input</th> <th>output F</th> </tr> <tr> <th>x</th> <th>y</th> <th></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	Input		output F	x	y		0	0	0	0	1	1	1	0	1	1	1	0	
Input		output F																				
x	y																					
0	0	0																				
0	1	1																				
1	0	1																				
1	1	0																				

Name	Graphic symbol	Algebraic function	Truth Table		
			Input	output	
Exclusive - NOR (or) Equivalence		$F = xy + \bar{x}\bar{y}$ $= x \odot y$	x	y	
			0	0	1
			0	1	0
			1	0	0
			1	1	1

Digital logic families and their characteristics:-

Logic families

Bipolar Logic families

- * Resistor-Transistor Logic (RTL)
- * Diode Transistor Logic (DTL)
- * Transistor-Transistor Logic (TTL)
- * Emitter Coupled Logic (ECL)

Mos Logic families

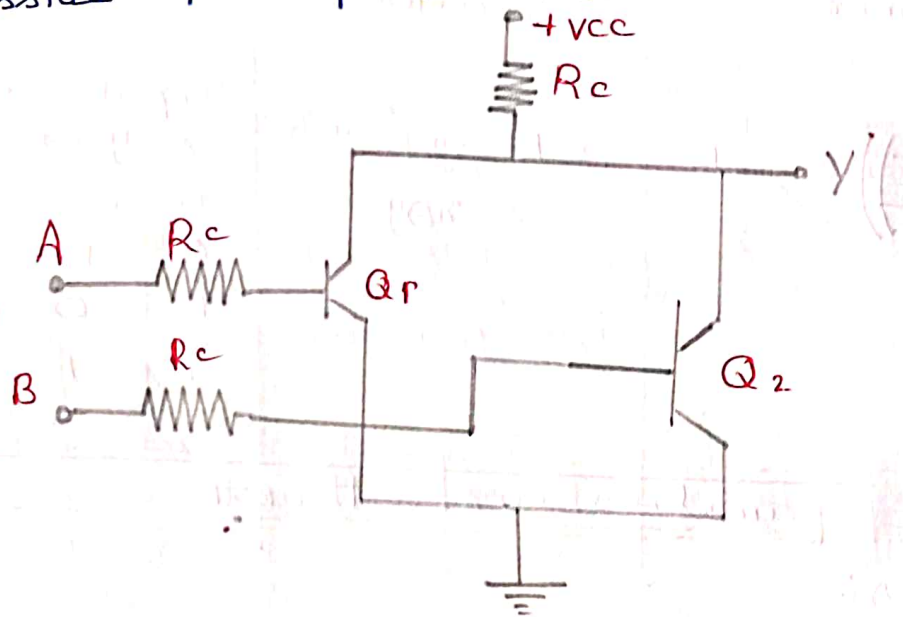
- * p-channel MOSFET (P-MOS)
- * N-channel MOSFET (N-MOS)
- * Complementary MOSFET (CMOS)

Resistor Transistor Logic (RTL)

* RTL circuit consist of resistors and transistors of both input and output stage circuits in a NOR logic gate

* The emitters of both the transistors are connected to a common ground and collectors of both transistors are connected through a common collector Resistor R_c

* The resistor R_c is commonly known as passive pullup resistor



2 input RTL NOR gate circuit diagram

* RTL gate input voltage corresponding to low level is required to be low enough for the corresponding transistor to be cut off

* When the input voltage corresponding to high level should be high enough to drive the corresponding transistor to saturation

* When the both the inputs are low transistor Q_1 and Q_2 are cut-off and the output is high

* When the both inputs are high transistors Q_1 and Q_2 are saturation and the output is low

* The saturation voltage, $V_{CE(sat)}$ for transistor is approximately 0.2V, so RTL gates low level output voltage is 0.2

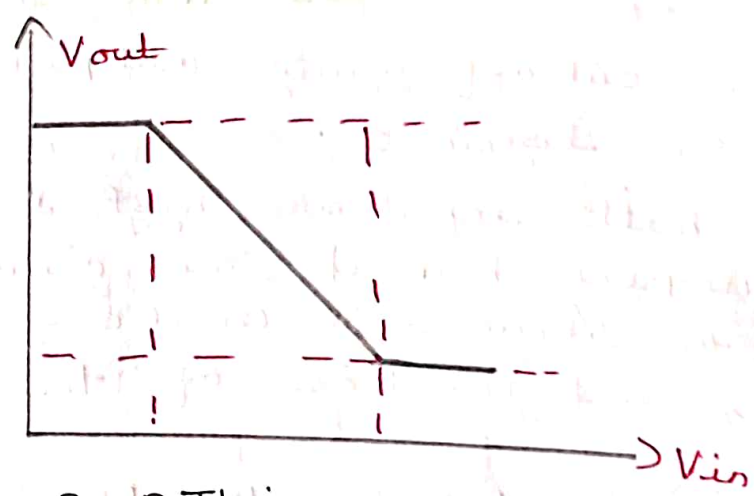
* A high level output voltage depends on the number of gates connected to the output.

* The Number of gates connected to the output increases output voltage decreases

V_A	V_B	Transistor Q_1	Transistor Q_2	V_y
Logic 0	Logic 0	cut-off	cut-off	Logic 1
Logic 0	Logic 1	cut-off	saturation	Logic 0
Logic 1	Logic 0	saturation	cut-off	Logic 0
Logic 1	Logic 1	saturation	saturation	Logic 0

Characteristics of RTL

- * The speed of operation is low
- * Fan out is 4 or 5 with a switching delay of 50ns and fan in 4
- * Poor noise immunity
- * Elimination of base resistors in RTL will reduce the power dissipation
- * sensitive to temperature
- * The noise margin from zero to the threshold voltage is about 0.5V and from one to the threshold voltage is only 0.2V



Advantages of RTL:-

- * Most simple digital circuit
- * Minimum number of transistor are required

* Operation of this circuit very simple

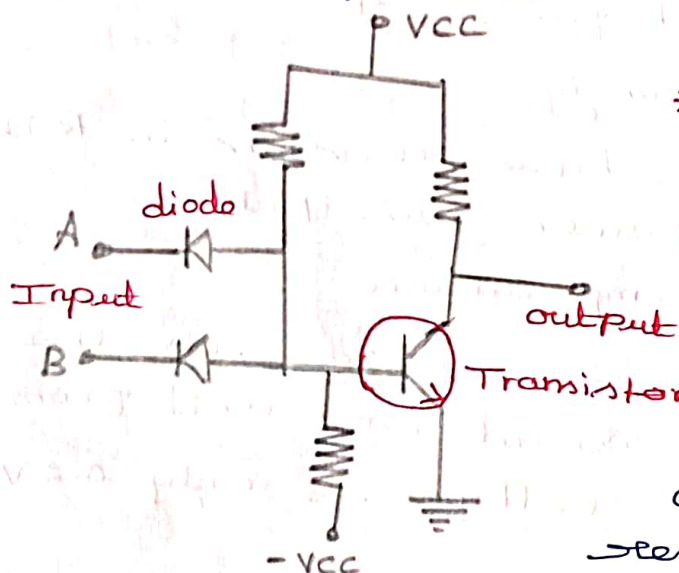
Disadvantage of RTL

- * Very poor Response time
- * High propagation delay
- * Power dissipation is high

Diode - Transistor Logic (DTL)

* The Logic gates are built with PN junction diode and transistors.

* Here diodes are used as input components and transistors are used as output components



* The circuit consist of two inputs diode transistor Logic NAND gate

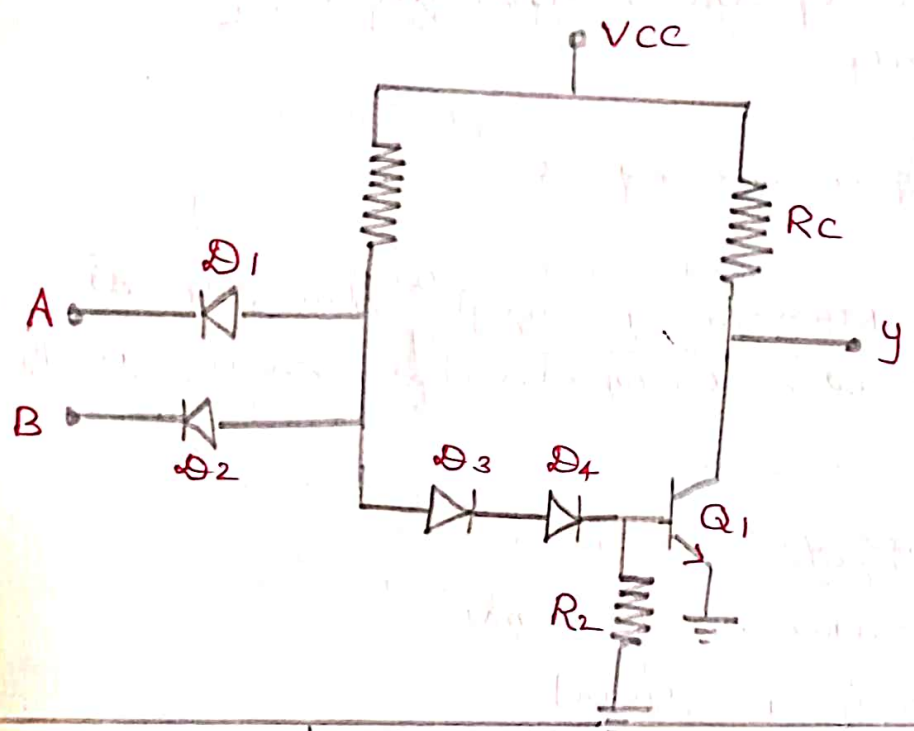
* when both inputs are low diode D_1 and D_2 conduct resulting 0.7V at point X

* Q_1 is cut off giving output voltage $V_o = V_{CC}$ so logic 1.

* when both input are high D_1 and D_2 are reversed biased. This causes the base current of transistor Q_1 to flow through R_1, D_3, D_4 and the base of the transistor Q_1 .

* The diode D_3, D_4 we need increased voltage level to drive transistor in saturation this improve the noise margin for DTL Gate

* when any one input is high (or) Low, the transistor Q_1 is cut off giving output voltage $V_o = V_{CC}$ and logic 1



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

Inputs		Diodes		Transistor output	
A	B	D_1	D_2	T	Y
0	0	Forward biased	Forward biased	cut-off	1
0	1	Forward biased	Reverse Reverse biased	cut-off	1
1	0	Reverse biased	Forward biased	cut-off	1
1	1	Reverse biased	Reverse biased	Saturation	0

Characteristics:

Propagation delay:-

* The turn off delay is considerably larger than the turn on delay often by a factor of 2 (or) 3

* The propagation delay of TTL is 25ns

Fanout :-

* A fanout as high as 8 is possible with the TTL family because of the high input impedance of the subsequent gates in the Logic 1 state

Fan in :-

* It has a fan in of 8

Noise Immunity :-

* The Noise Margin is high due to the additional diode D_4 connected in series with D_1

Advantages :-

- * fan out is high
- * power dissipation is 8-12mw
- * Noise immunity is good

Disadvantages :-

- * More elements are required
- * Propagation delay is more
- * speed of operation is less

Transistor Transistor Logic (TTL)

* Transistor Transistor Logic TTL is named for its dependence on transistor alone to perform basic logic operation

* There are many versions or families of TTL

- ↳ standard TTL
- ↳ High speed TTL
- ↳ Low power TTL
- ↳ schottky TTL

TTL have three configuration for outputs

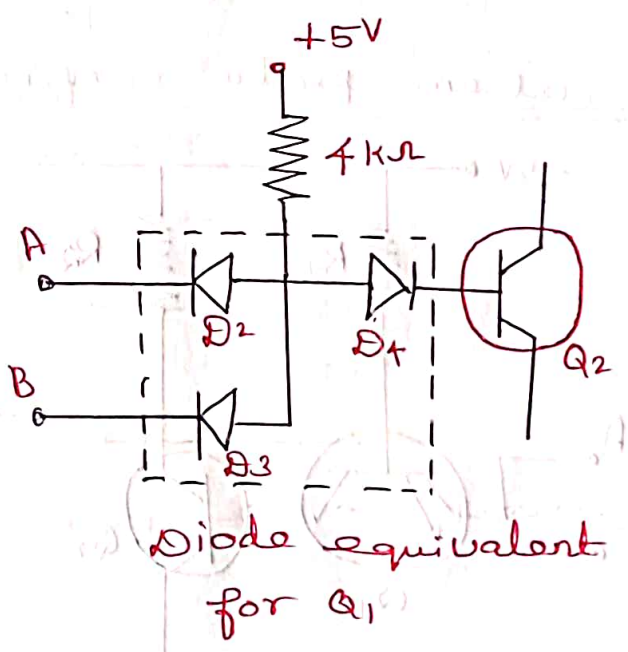
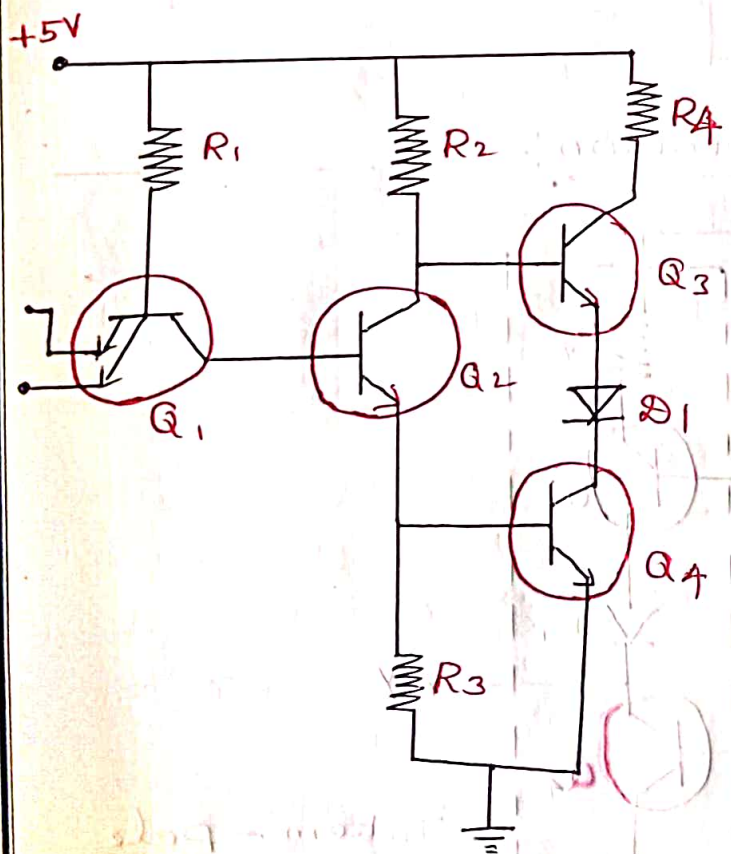
- * Multiple emitter transistor
- * Totem-pole output
- * Tristate output

Multiple emitter transistor

* The two input TTL NAND gate input structure consists of multiple emitter transistor and output structure consist totem pole output

* The transistor Q_1 having two emitters one for each input to the gate

* Diode D_2 and D_3 represent the two emitter base junction of Q_1 and D_4 is collection base junction



Diode equivalent for Q_1

Two input TTL NAND gate

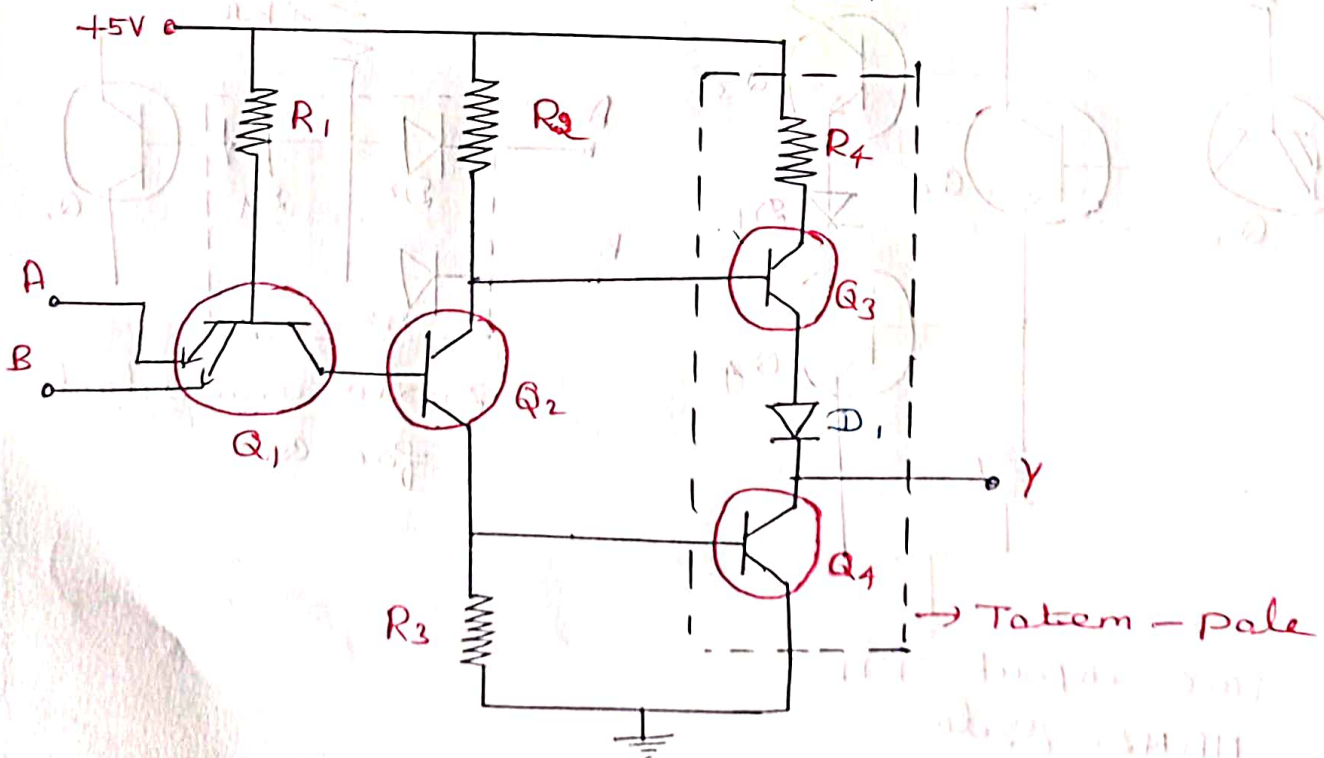
* The input voltage A and B are either Low or high (+5V)

* If A and B are logic Low ($A=0, B=0$) diode d_2 and d_3 are forward biased hence Q_2 and Q_3 conducts and base voltage of Q_1 is pulled down

* If A and B are logic ($A=1$ and $B=1$) diode Q_2 and Q_3 are reverse bias making them off

Input		output Y
A	B	
0	0	1
0	1	1
1	0	1
1	1	0

Totem pole configuration:-



* Transistor Q3 and Q4 form a totem-pole such configuration is known as active pull-up (or) totem pole output

* It is produce low output impedance

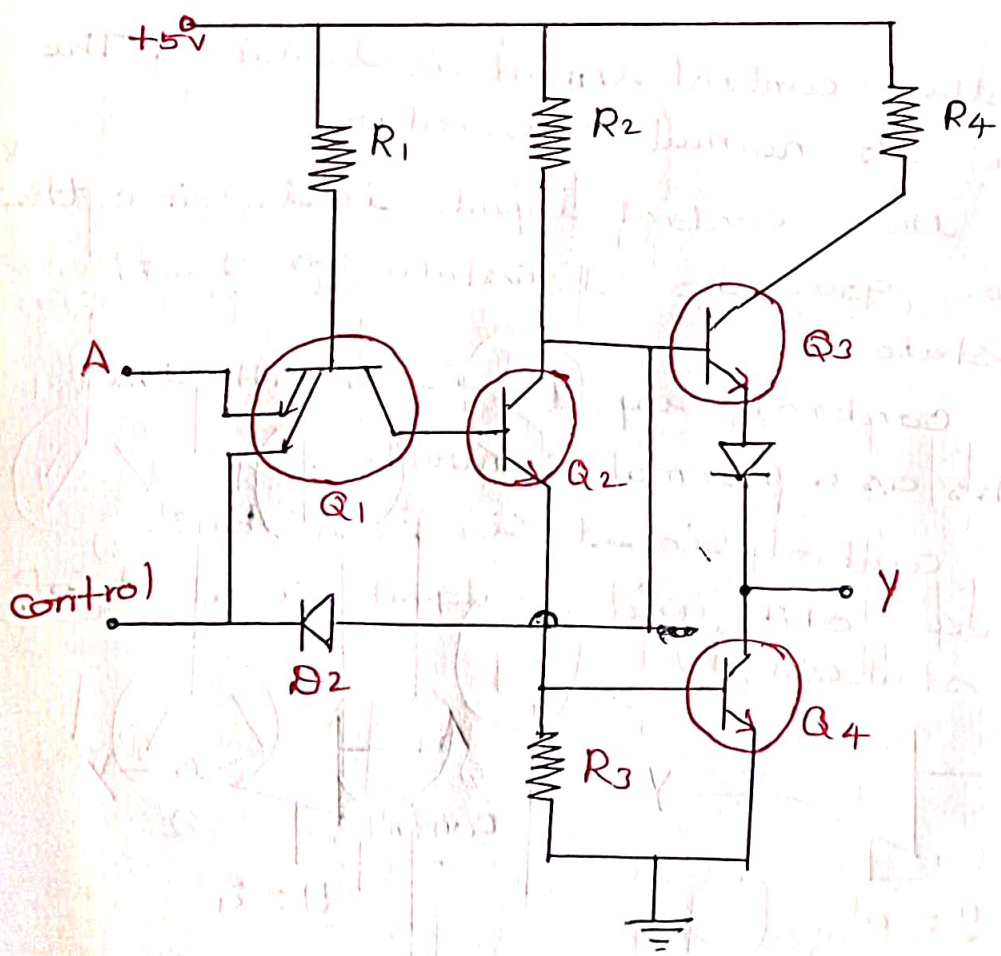
* Q3 acts as an emitter follower higher output or Q4 is saturated (Low output)

* Q3 is conducting the output impedance is only 70Ω

* Q4 is saturated the output impedance is only 12Ω the output impedance value is low

* The propagation delay is low

Tri state output configuration :-



* Logic gates have two output states logic 0 and logic 1

* But the tristate or three state gate as three output states.

* A Low level state or logic 0 state, when the lower transistor in the totem pole is ON and the upper transistor is OFF

* A high level state or logic 1, when the lower transistor in the totem pole is OFF and upper transistor is ON

* A third state when both transistors in totem pole are off. This provides an open circuit or high impedance state

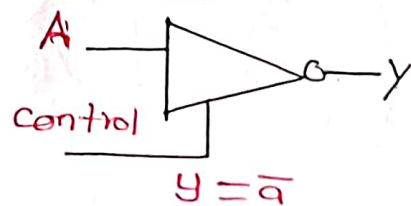
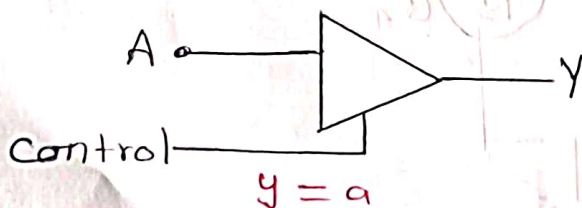
* The tristate gate consists of an extra input called a control input or enable input.

* when the control input is logic 1 The gate perform its normal operation.

* when the control input is logic 0 the output gate goes to tristate or high impedance state

* when control input is HIGH the circuit works as a normal Inverter.

* when control input is Low both transistors are OFF and output is at high impedance states.



TTL characteristics

Power dissipation:-

* A standard TTL gate has an

average power dissipation of about 10mW

Propagation delay:-

* Propagation delay is the time it takes for the output of a gate to change after the input have changed.

Fanout:-

* A standard output can typically drive 10 standard TTL inputs.

Operating speed:-

* TTL is faster than CMOS

Advantage :-

- * High speed
- * propagation delay 10ns
- * Moderate power dissipation
- * Low cost
- * Moderate packaging density

Disadvantages:-

- * Highest power dissipation
- * Lower noise immunity
- * Less fan out.

Emitter coupled logic (ECL)

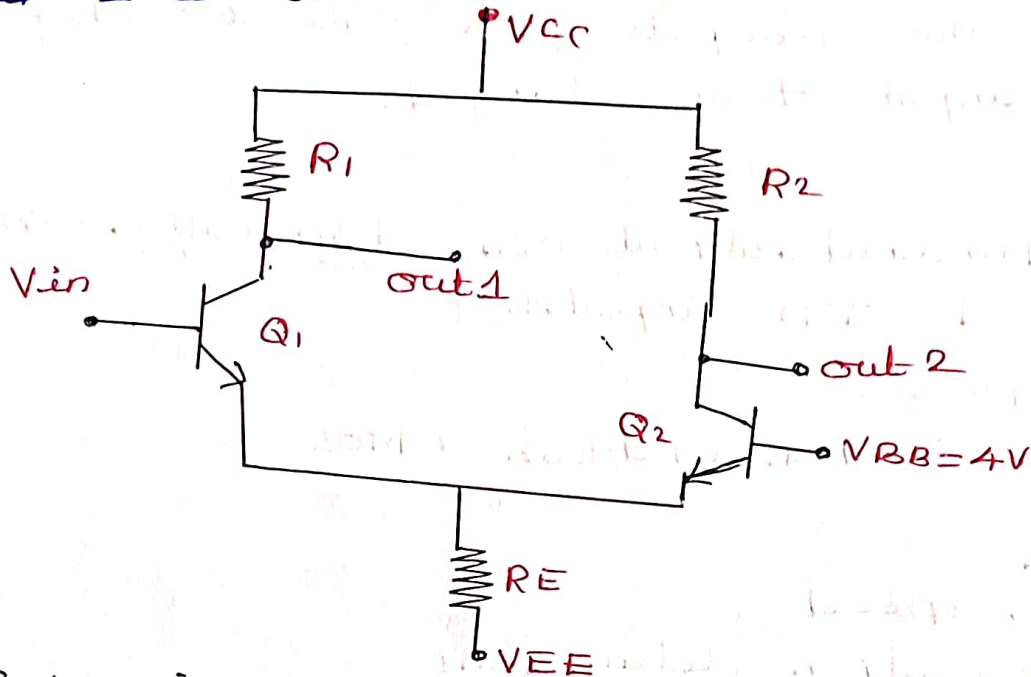
* Emitter coupled logic (ECL) is a non saturated digital logic family.

* ECL logic family has the lowest propagation delay of any family and is used to very high speed operation.

* The output provide both OR and NOR functions.

* Each input is connected to the base of transistor

* Voltage level for high state is 0.8V and Low state is 1.8V



If V_{in} is High:

* The transistor is turned ON and Q_1 is turned OFF

* The current in R_1 flows into the base of Q_2

* when input voltage V_{in} is High transistor Q_1 is ON but not saturated and transistor Q_2 is OFF

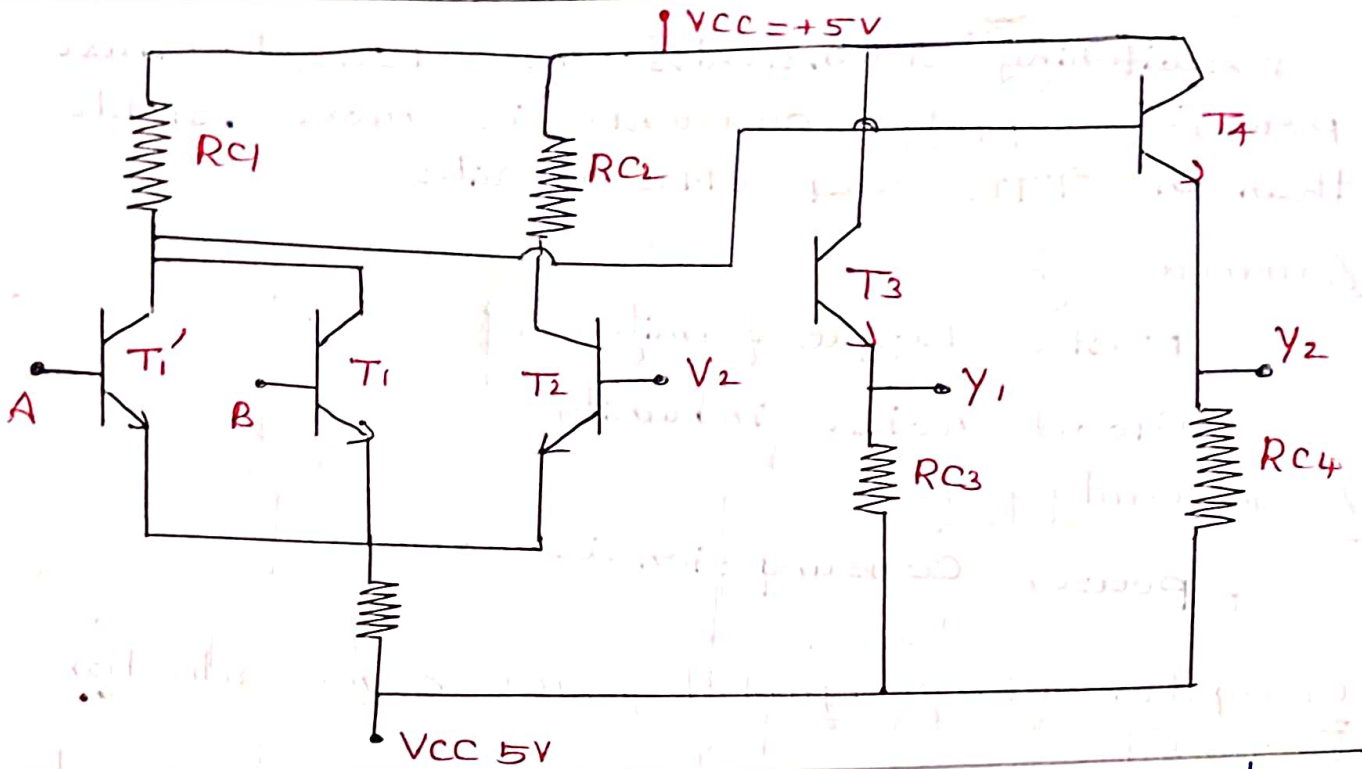
If V_{in} is Low:-

* when the input voltage is Low transistor Q_2 is ON but not saturated and transistor Q_1 is OFF.

Emitter coupled logic OR/NOR gate:

* The circuit has two output Y_1 and Y_2 . Y_1 is OR logic Y_2 is NOR Logic

* emitter followers used at output of difference amplifier to shift the DC level.



Inputs		Transistors			Transistor		output	
A	B	T ₁	T ₁ '	T ₂	T ₃	T ₄	Y ₁	Y ₂
0	0	cut off	cut off	Active	Active	cut off	0	1
0	1	cut off	Active	cut off	cut off	Active	1	0
1	0	Active	Active	cut off	cut off	Active	1	0
1	1	Active	Active	cut off	cut off	Active	1	0

Characteristics:-

- * It is fastest of Logic families
- * Transistors are not allowed to complete saturation and thus eliminating storage delay
- * To prevent transistors from going into complete saturation logic levels
- * Noise margin is reduced and it is difficult to achieve good noise immunity

* switching transients are less because power supply current is more stable than in TTL and CMOS circuits

Advantages:-

* Faster Logic family

* Good noise immunity

Disadvantages:-

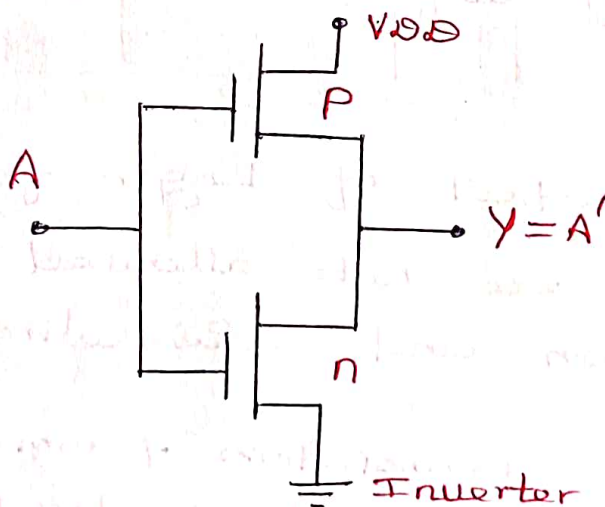
* power consumption is more

Complementary metal oxide semiconductor (CMOS)

* CMOS circuit contain both NMOS and PMOS devices to speed the switching of capacitive loads

* It consumes low power and can be operated at high voltage resulting in improved noise immunity

* It consists of p-channel transistor and n-channel transistor



CMOS Logic circuit

* p channel device is V_{DD} connected to source terminal

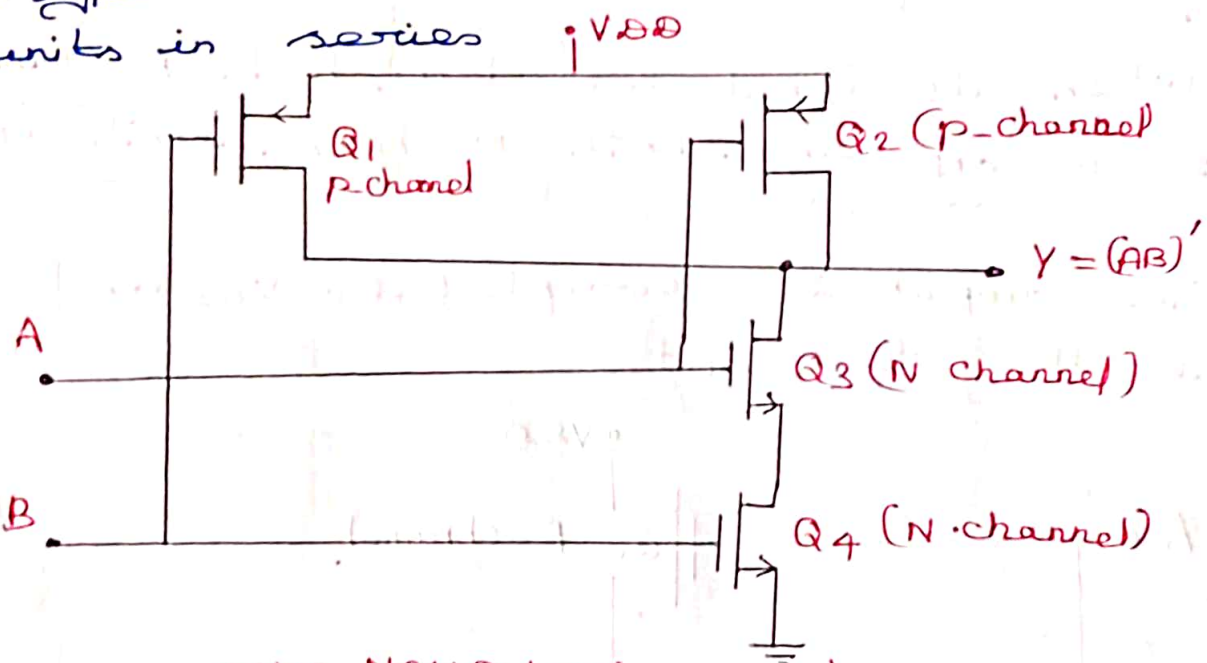
* And n-channel device is connected to the ground

* when the input is Low, Q₁ is ON and Q₂ is OFF, output is high

* when the input is high Q₁ is OFF and Q₂ is Low, output is Low

CMOS NAND gate:-

* A two input NAND gate consists of two p-type units in parallel and two n-type units in series



CMOS NAND Logic circuit

* when the input are low Q₁ and Q₂ are ON, Q₃ Q₄ OFF and output is high

* when the any one of the input is Low, the corresponding MOSFET Q₁ or Q₂ is ON, Q₃ (or) Q₄ is ON and output is high

* when the input are high Q₁ and Q₂ OFF, and Q₃ Q₄ are ON and output is Low

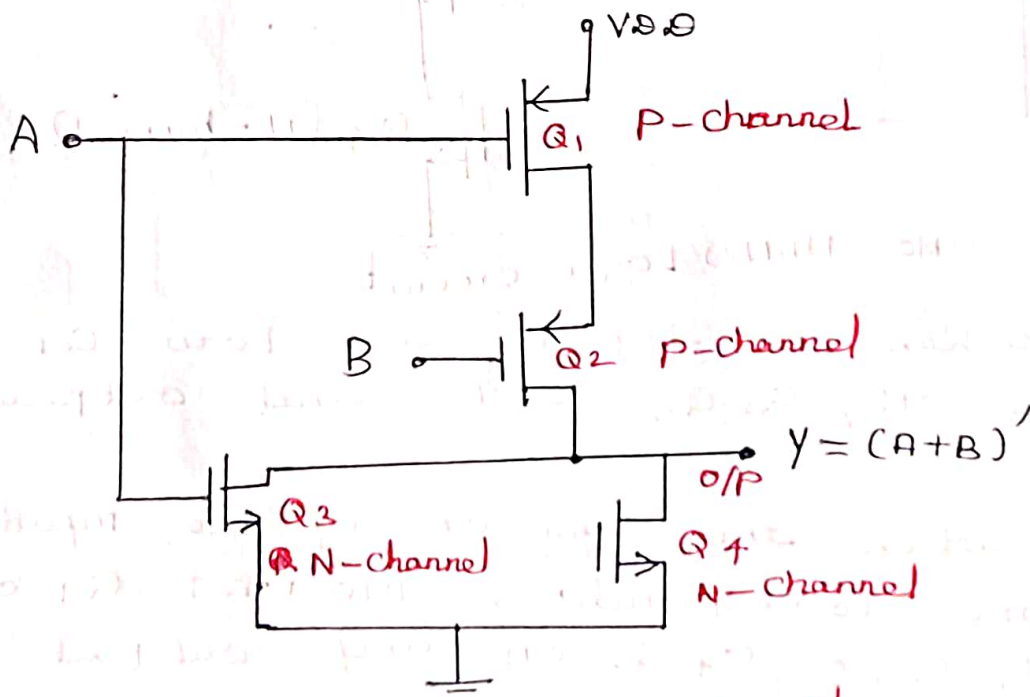
A	B	Q ₁	Q ₂	Q ₃	Q ₄	V _o (Output)
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	1
1	0	OFF	ON	ON	OFF	1
1	1	OFF	OFF	ON	ON	0

CMOS - NOR Gate :-

* A two input NOR gate consists of two N-type units parallel and P-type units in series

* when all input are low, both p-channel units are ON and both N-channel units are OFF.

* The output is coupled to V_{DD} and goes to the high state



CMOS NOR logic circuit

Advantages :-

- * consume less power
- * operate high voltage
- * Fan out more
- * Better Noise Margins
- * Improve Noise Immunity

Disadvantages :-

- * switching speed Low
- * greater propagation delay

Combinational Logic circuit :-

* The combinational Logic circuits are the circuits that contain different types of Logic gates.

* A circuit in which different types of logic gates are combined is known as combinational logic circuit.

* The output of the combinational circuit is determined from the present combination of inputs, and previous input

* There are different types of combinational logic circuits such as

↳ Adder

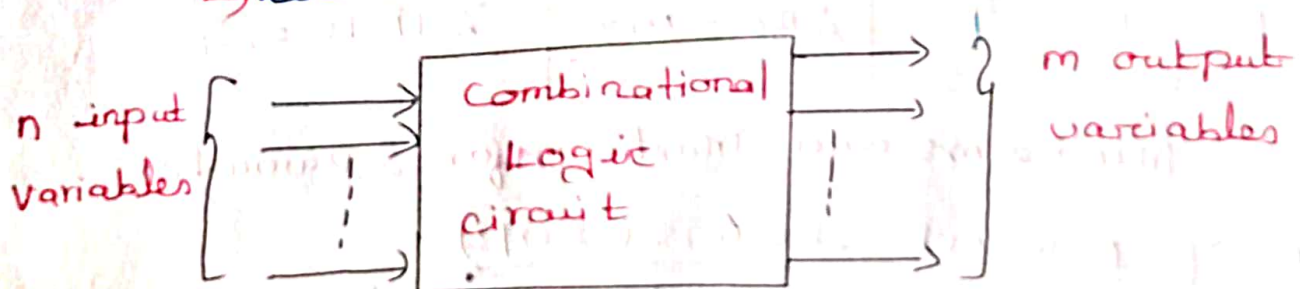
↳ Subtractor

↳ Decoder

↳ Encoder

↳ Multiplexer

↳ De-multiplexer



block diagram of combinational Logic circuit

* The combinational circuit doesn't have any backup or previous memory

* The present state of the circuit is not affected by the previous state of the input

Representation of Logic functions:-

* Each individual term in standard SOP form is called minterm. and each individual term in standard POS form is called maxterm.

* In n variable logical function there are 2^n minterms and 2^n maxterms.

Variable			Minterms	Max terms
A	B	C	m_i	M_i
0	0	0	$\bar{A}\bar{B}\bar{C} = m_0$	$A+B+C = M_0$
0	0	1	$\bar{A}\bar{B}C = m_1$	$A+B+\bar{C} = M_1$
0	1	0	$\bar{A}B\bar{C} = m_2$	$A+\bar{B}+C = M_2$
0	1	1	$\bar{A}BC = m_3$	$A+\bar{B}+\bar{C} = M_3$
1	0	0	$A\bar{B}\bar{C} = m_4$	$\bar{A}+B+C = M_4$
1	0	1	$A\bar{B}C = m_5$	$\bar{A}+B+\bar{C} = M_5$
1	1	0	$AB\bar{C} = m_6$	$\bar{A}+\bar{B}+C = M_6$
1	1	1	$ABC = m_7$	$\bar{A}+\bar{B}+\bar{C} = M_7$

Minterms and Maxterms for 3 variables

Sum of product terms (SOP)

* Sum of product term is also called as sum of minterms

* If two or more minterms are combined with an OR Logic is said to be sum of product (SOP)

* A minterm is a term of two or more variables which are combined with AND Logic.

$$f(A, B, C) = AB + BC + AC$$

Sum
↓
Product terms

Product of sum terms (POS)

* Product of sum term is also called as product of maxterms.

* If two or more maxterms are combined with an AND Logic is said to be a product of sum (POS)

* A maxterm is a term of two or more variables which are combined with OR Logic

$$f(A, B, C) = (A+B) \cdot (B+\bar{C}) \cdot (A+\bar{C})$$

Product
↓
sum terms

Standard SOP form

* standard SOP form means the each of the product term consists of all variable of the function in either complemented form or uncomplemented form

$$f(A, B, C) = \bar{A}\bar{B}C + A\bar{B}C + A\bar{B}\bar{C}$$

product term
Sum

* Here the function has three variables these three variables are present in each of the product terms either in complemented or uncomplemented

Converting SOP to standard SOP form:-

Step 1:- * Find the missing variable in each product

Step 2 :-

* ANDing each product term by ORing the missing variable

Step 3 :-

* Expand the terms by applying distributive Law

Step 4 :-

* Reduce the expression by omitting repeated product terms if any

convert the given expression in standard SOP form

$$f(A, B, C) = AC + A\bar{B}$$

sol

Step 1 :- Find missing variables

$$* f(A, B, C) = AC + A\bar{B}$$

↳ c missing
↳ B missing

Step 2 :- * AND product term with ORing the missing variables

$$f(A, B, C) = AC(B + \bar{B}) + A\bar{B}(C + \bar{C})$$

Step 3 :- * Expand the terms and rearrange it

$$f(A, B, C) = ACB + AC\bar{B} + A\bar{B}C + A\bar{B}\bar{C}$$
$$= ABC + A\bar{B}C + A\bar{B}\bar{C} + A\bar{B}C$$

Step 4 * omit repeated product terms

$$\therefore [A\bar{B}C + A\bar{B}C] = A\bar{B}C$$

$$f(A, B, C) = ABC + A\bar{B}\bar{C} + A\bar{B}C$$

\therefore standard SOP form is

$$f(A, B, C) = ABC + A\bar{B}\bar{C} + A\bar{B}C$$

Express the Boolean function $F(A,B,C) = A + \bar{B}C$ in a standard sum of minterms

sol

step 1: Find the missing variables

$$f(A,B,C) = A + \bar{B}C$$

↳ A is missing

↳ B and C missing

step 2:- AND product term with ORing missing variables

$$F(A,B,C) = A \cdot (B + \bar{B}) \cdot (C + \bar{C}) + \bar{B}C \cdot (A + \bar{A})$$

step 3: Expand the terms and reorder it

$$= (AB + A\bar{B}) \cdot (C + \bar{C}) + A\bar{B}C + \bar{A}\bar{B}C$$

$$= ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + A\bar{B}C + \bar{A}\bar{B}C$$

step 4:- omit repeated product terms

$$F(A,B,C) = ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + \bar{A}\bar{B}C$$

$\therefore A\bar{B}C + A\bar{B}C = A\bar{B}C$

* The standard SOP form

$$F(A,B,C) = ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + \bar{A}\bar{B}C$$

Standard POS form:-

* Similarly to standard SOP form, the standard POS form means the each of sum term consists of all variable of the function in either complemented or uncomplemented form

$$f(A,B,C) = (A + B + C) \cdot (A + \bar{B} + \bar{C}) \cdot (\bar{A} + B + \bar{C})$$

sum term

product

Converting pos to standard pos form:-

Step 1:-

* find the missing variable in each sum term

Step 2:-

* ORing each sum terms by ANDing the missing variables and its complement

Step 3:-

* Expand the terms by applying distributive Law and reorder the variables in the sum terms.

Step 4:-

* Reduce the expression by omitting repeated sum terms

convert the expression in standard pos form

$$f(A, B, C) = (A + C) \cdot B$$

sol

Step 1:- Find the missing variables

$$f(A, B, C) = (A + C) \cdot B$$

└───┬───> A and C missing
└───┬───> B is missing

Step 2:- ORing sum terms by AND the missing variable

$$f(A, B, C) = (A + C) + (B \cdot \bar{B}) B + (A \cdot \bar{A}) + (C \cdot \bar{C})$$

Step 3:- Expand the term and reorder it

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

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

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

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

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

step 4:- omit repeated term.

$$f(A,B,C) = (A+B+C) \cdot (A+\bar{B}+C) \cdot (A+B+\bar{C}) \cdot (\bar{A}+B+C) \cdot (\bar{A}+B+\bar{C})$$

the standard pos form is

$$f(A,B,C) = (A+B+C) \cdot (A+\bar{B}+C) \cdot (A+B+\bar{C}) \cdot (\bar{A}+B+C) \cdot (\bar{A}+B+\bar{C})$$

Reduce the following function using k maps

$$f(w,x,y,z) = \sum m(0,1,8,9,10)$$

Sol

	yz	00	01	11	10	
wx	00	1	1			
	01	1				
	11					
	10	1	1		1	

Groupings shown in the map:
 - A group of four 1s at (00,00), (01,00), (00,01), (01,01) is circled with a dashed line and labeled $\bar{w}\bar{y}\bar{z}$.
 - A group of two 1s at (00,01) and (01,01) is circled with a dashed line and labeled $\bar{x}\bar{y}$.
 - A group of four 1s at (00,10), (01,10), (10,10), (11,10) is circled with a dashed line and labeled $w\bar{x}\bar{z}$.

simplified Boolean expression is

$$y = \bar{w}\bar{y}\bar{z} + \bar{x}\bar{y} + w\bar{x}\bar{z}$$

Reduce the following four variable function to its minimum form.

$$y = \bar{A}\bar{B}C\bar{D} + \bar{A}BC\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}BC\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}BC\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}BC\bar{D}$$

Sol

$$y = 0010 + 1110 + 1010 + 1011 + 1000 + 1100 + 0011 + 0000$$

$$y = \sum (0, 2, 3, 8, 10, 11, 12, 13)$$

	00	01	11	10
00	1		1	
01				
11				
10	1		1	1

$\overline{B}\overline{C}\overline{D}$ (grouping cells 1, 2, 8, 9)
 $\overline{B}C$ (grouping cells 3, 11)
 $A\overline{D}$ (grouping cells 1, 5, 9, 13)

The simplified Boolean expression is

$$Y = \overline{B}\overline{C}\overline{D} + \overline{B}C + A\overline{D}$$

Minimize the expression.

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

Sol

$$(A+B+\overline{C}) = 001 = M_1$$

$$(A+\overline{B}+\overline{C}) = 011 = M_3$$

$$(\overline{A}+\overline{B}+\overline{C}) = 000 = M_0$$

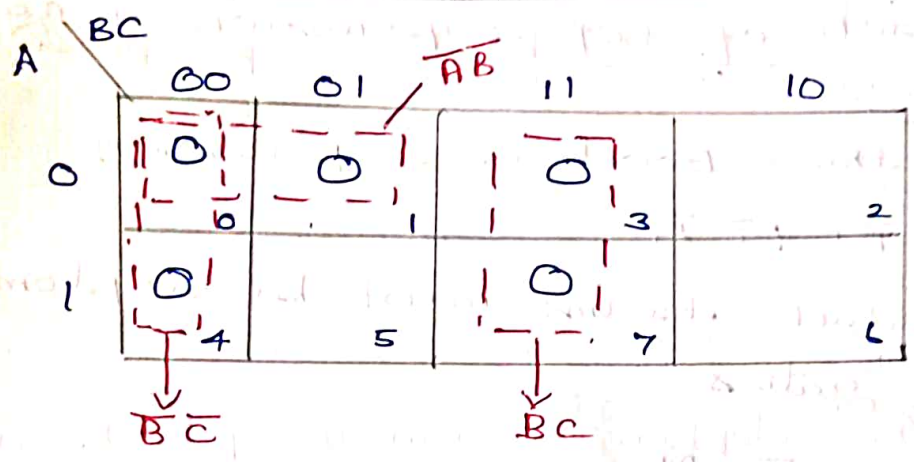
$$(\overline{A}+B+C) = 100 = M_4$$

$$(A+B+C) = 000 = M_0$$

$$Y = \prod M (0, 1, 3, 4, 7)$$

$$11100 +$$

$$00000 +$$



$$Y = \bar{B}\bar{C} + \bar{A}\bar{B} + BC$$

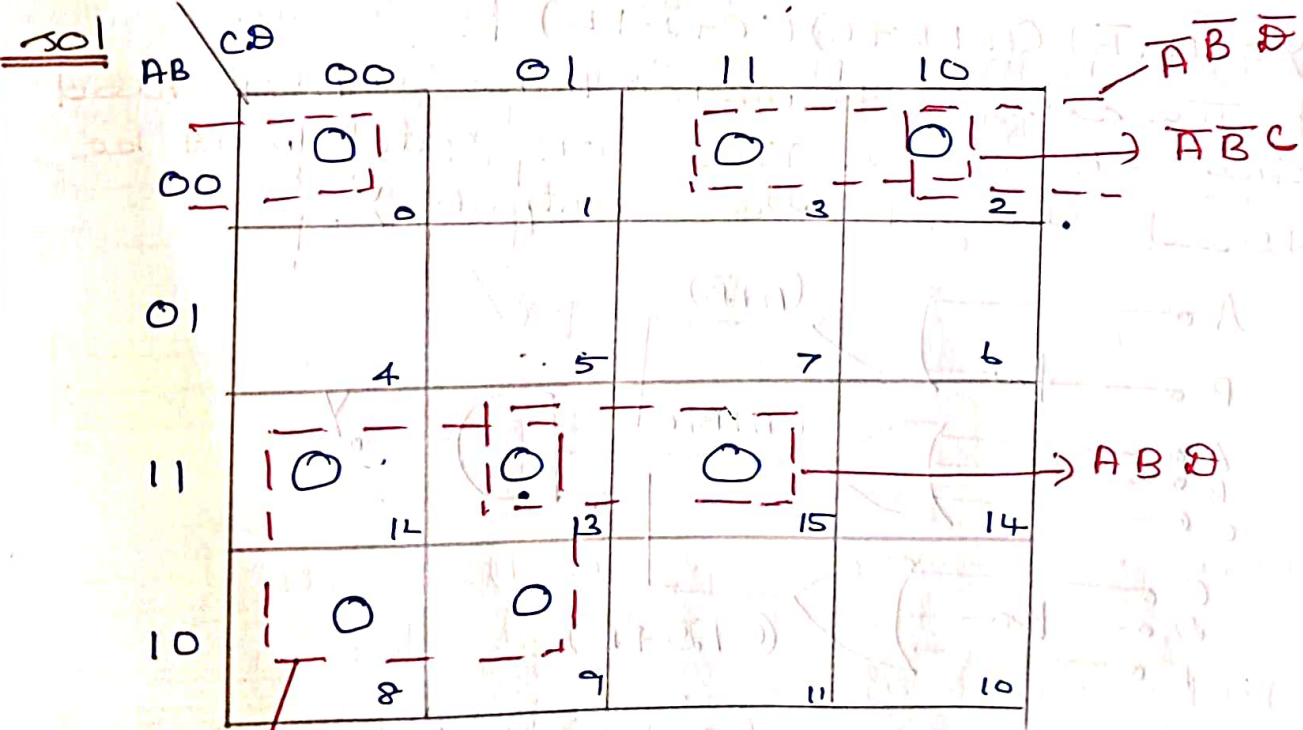
demorgan theorem simplified Boolean expression

$$Y = \bar{B}\bar{C} + \bar{A}\bar{B} + BC$$

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

Reduce the following function using K-Map

$$F(A, B, C, D) = \Pi_M(0, 2, 3, 8, 9, 12, 13, 15)$$



$$Y = \bar{A}\bar{B}\bar{D} + \bar{A}\bar{B}C + \bar{A}C + ABD$$

Demorgan Law

$$Y = \bar{A}\bar{B}\bar{D} + \bar{A}\bar{B}C + \bar{A}C + ABD$$

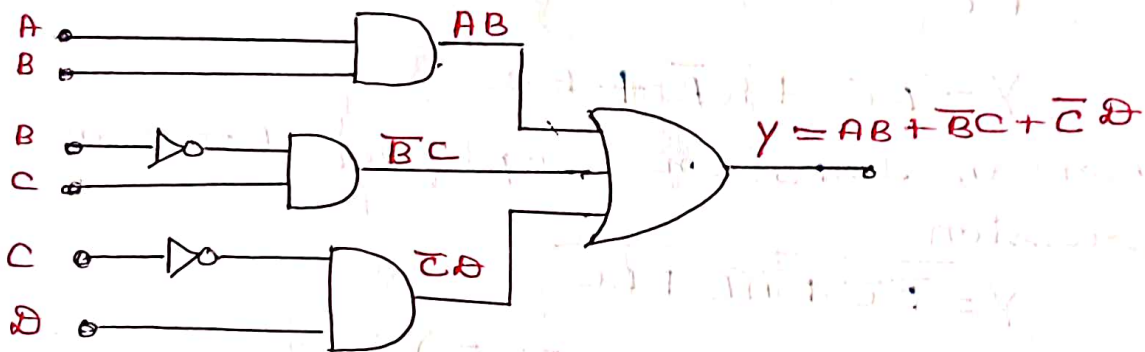
$$Y = (A+B+D) \cdot (A+B+\bar{C}) \cdot (\bar{A}+C) \cdot (\bar{A}+B+\bar{D})$$

Implementation of SOP expression using Logic gates :-

* Consider the Boolean expression

$$Y = AB + \bar{B}C + \bar{C}\bar{D}$$

* The product terms must be implemented using AND gates

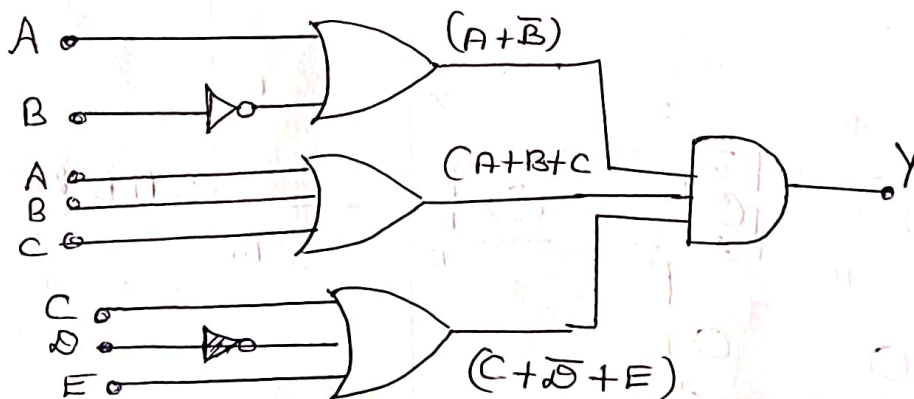


Implementation of POS expression using Logic gates :-

* Consider POS Boolean expression

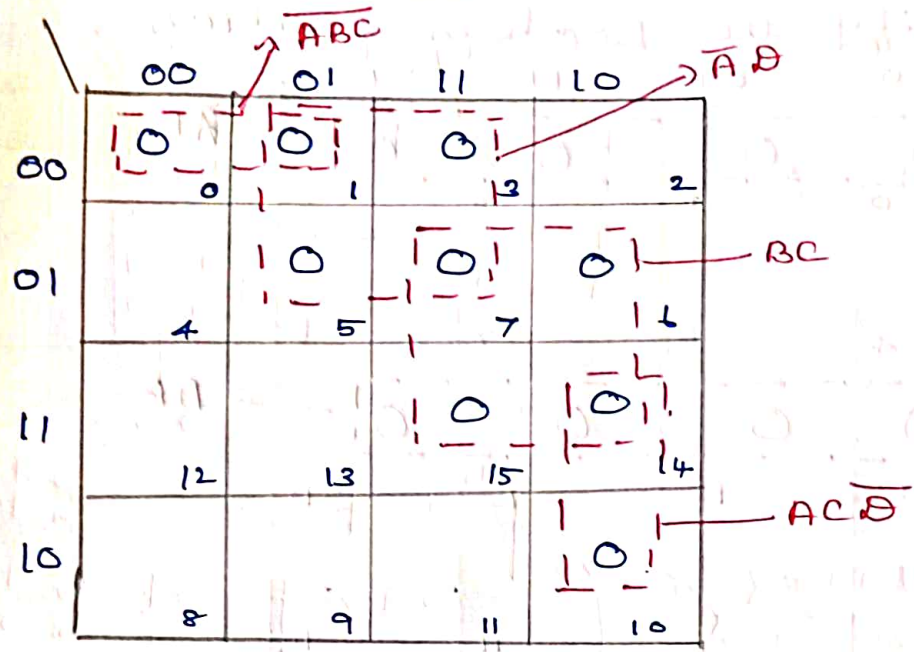
$$Y = (A + \bar{B})(A + B + C)(C + \bar{D} + E)$$

* The sum of terms must be implemented using OR gates. These OR gates must be ANDed to get the output Y.



Given $Y(A, B, C, D) = \Pi(0, 1, 3, 5, 6, 7, 10, 14, 15)$
draw the k-map and obtain the expression. Realize the minimum expression using basic gates.

Sol

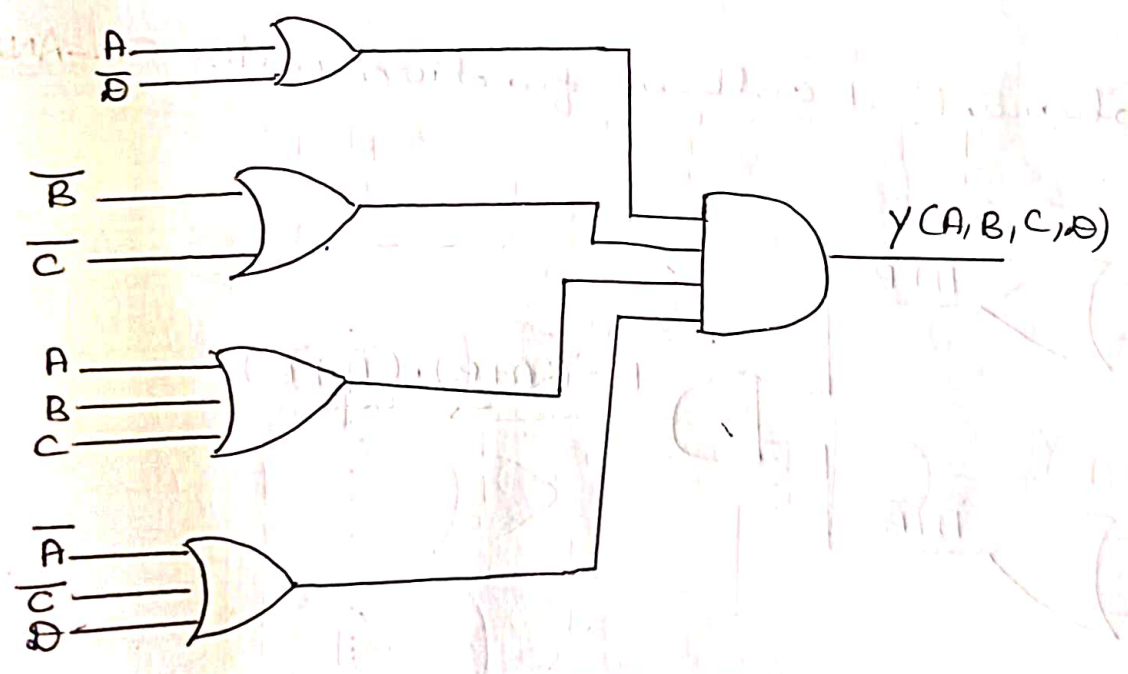


$$Y = \overline{A}B + BC + \overline{A}B\overline{C} + AC\overline{D}$$

demorgan's Law

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

Logic diagram



simplify and implement the following pos function using NOR gates

$$f(A, B, C, D) = \prod M(0, 1, 2, 3, 12, 13, 14, 15)$$

Sol

step 1:- simplify the Boolean function,

AB \ CD	00	01	11	10
00	0	0	0	0
01				
11	0	0	0	0
10				

The Karnaugh map shows two groups of four 0s:

- A group of four 0s in the top row (minterms 0, 1, 3, 2) is circled with a dashed red line and labeled $\overline{A}\overline{B}$.
- A group of four 0s in the third row (minterms 12, 13, 15, 14) is circled with a dashed red line and labeled AB .

$$f(A, B, C, D) = \overline{A}\overline{B} + AB$$

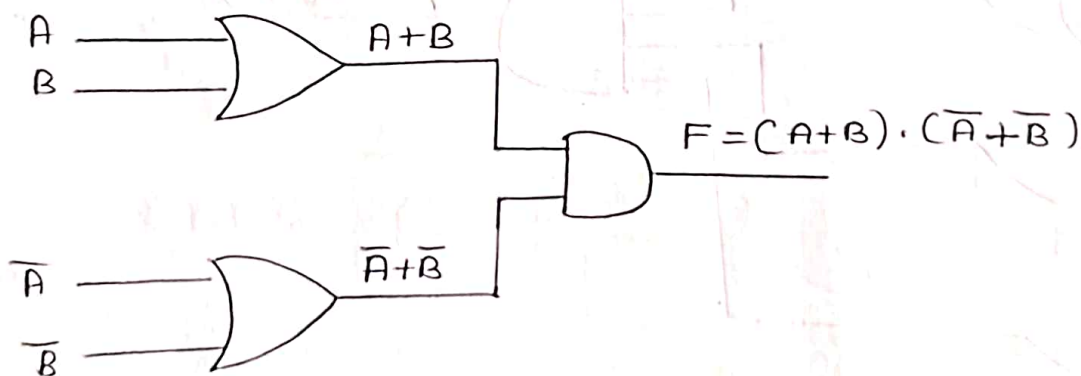
Demorgan Law

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

$$f(A, B, C, D) = (A+B) \cdot (\overline{A} + \overline{B})$$

step 2:-

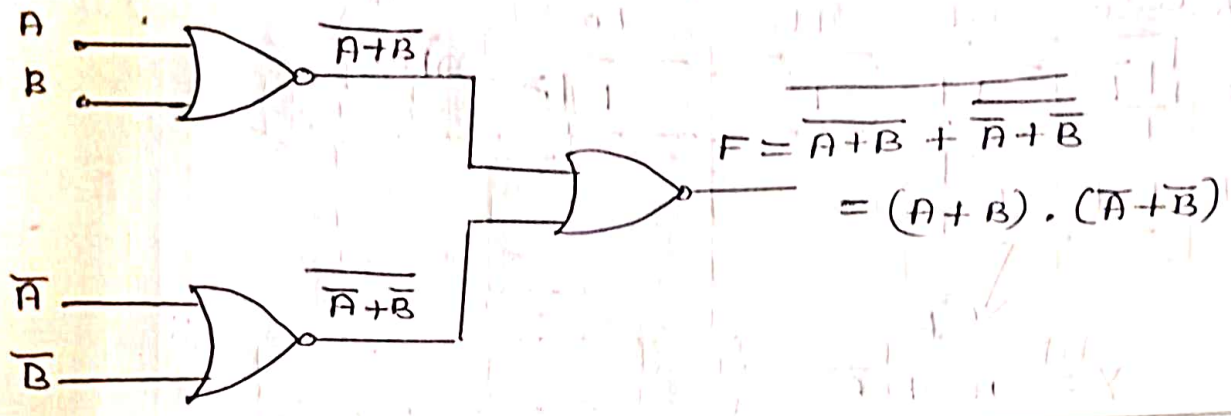
Implement Boolean function with OR-AND gate



step 3:-

Convert OR-AND Logic to NOR-NOR

Logic



Design a combinational circuit with three inputs and one output

- i) The output is 1 when the binary value of the input is less than 6 otherwise output is 0
- ii) The output is 1 when the binary value of the input is an even number
- iii) The output is 1 when the binary value of the input is an odd number

sol

i) The output is 1 when binary value of the input is less than 6, otherwise output is 0

equivalent decimal value	Input			output Y
	A	B	C	
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

output will be 1 for the binary value less than 6

simplified the expression

A	BC	00	01	11	10
0		1	1	1	1
1		1	1		

\bar{A} (indicated by a red dashed box around the top row)

 \bar{B} (indicated by a red dashed box around the first two columns)

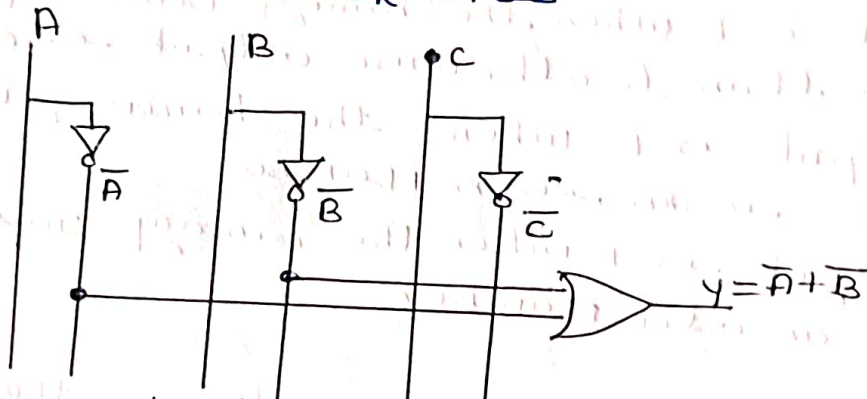
$$Y = \bar{A} + \bar{B}$$

Draw the Logic diagram

$$Y = \bar{A} + \bar{B}$$

Not Gate

OR Gate



ii) The output 1 when the binary value of inputs is an even number

equivalent decimal value	Input			output
	A	B	C	
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0

The even numbers 0, 2, 4, 6, produce the output as 1

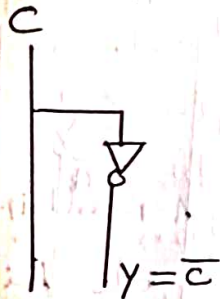
simplified the expression

	BC	00	01	11	10
A					
0		1	0	0	1
1		1	0	0	1

$$y = \bar{c}$$

Draw the Logic diagram

$$y = \bar{c} \text{ Not gate}$$



iii) when the output, binary value of the input is odd Number

Equivalent decimal value	Input			output Y
	A	B	C	
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

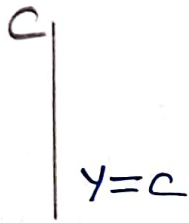
The odd Number 1, 3, 5, 7 produce the output 1

simplified the expression

A \ BC	00	01	11	10
0	0	1	1	0
1	0	1	1	0

c

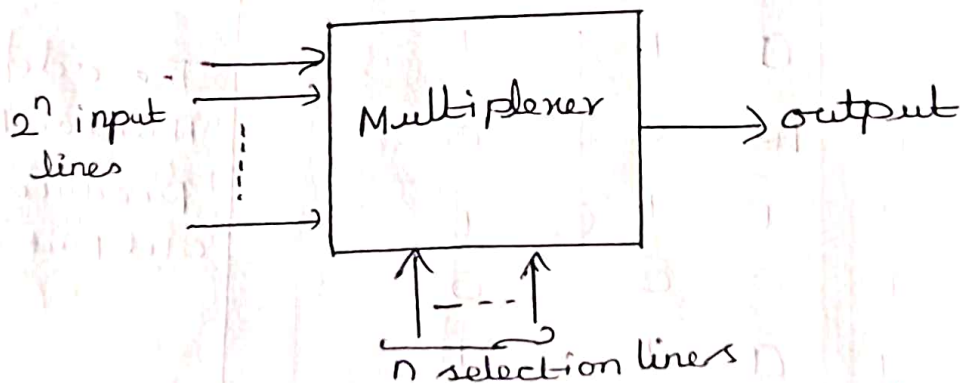
Draw the logic diagram



Multiplexers

* Multiplex means Many to one. A Multiplexer is a combinational circuit that selects binary information from one of many input lines into single output line.

* Multiplex consists of 2^n number of input lines and only one output line.



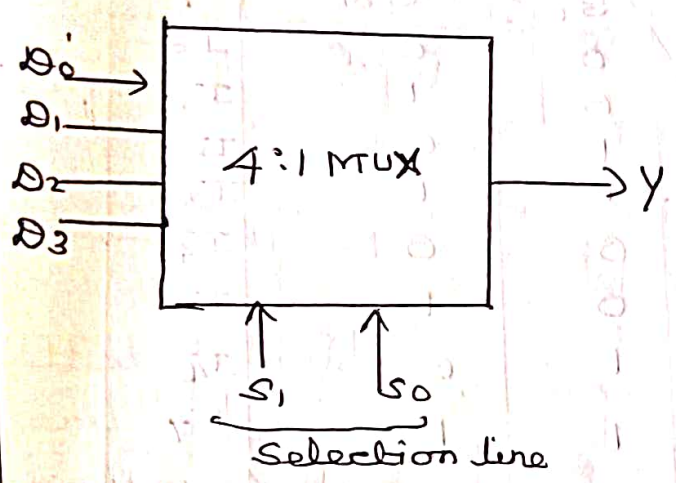
Block diagram of Multiplexer

* It has two different types

- ↳ 4:1 Multiplexer
- ↳ 8:1 Multiplexer

4:1 Multiplexer:-

* A 4:1 multiplexer has 4 inputs and two selection lines and single output line Y



selection line		Y	
S ₁	S ₀		
0	0	D ₀	I ₀
0	1	D ₁	I ₁
1	0	D ₂	I ₂
1	1	D ₃	I ₃

truth table

Block diagram

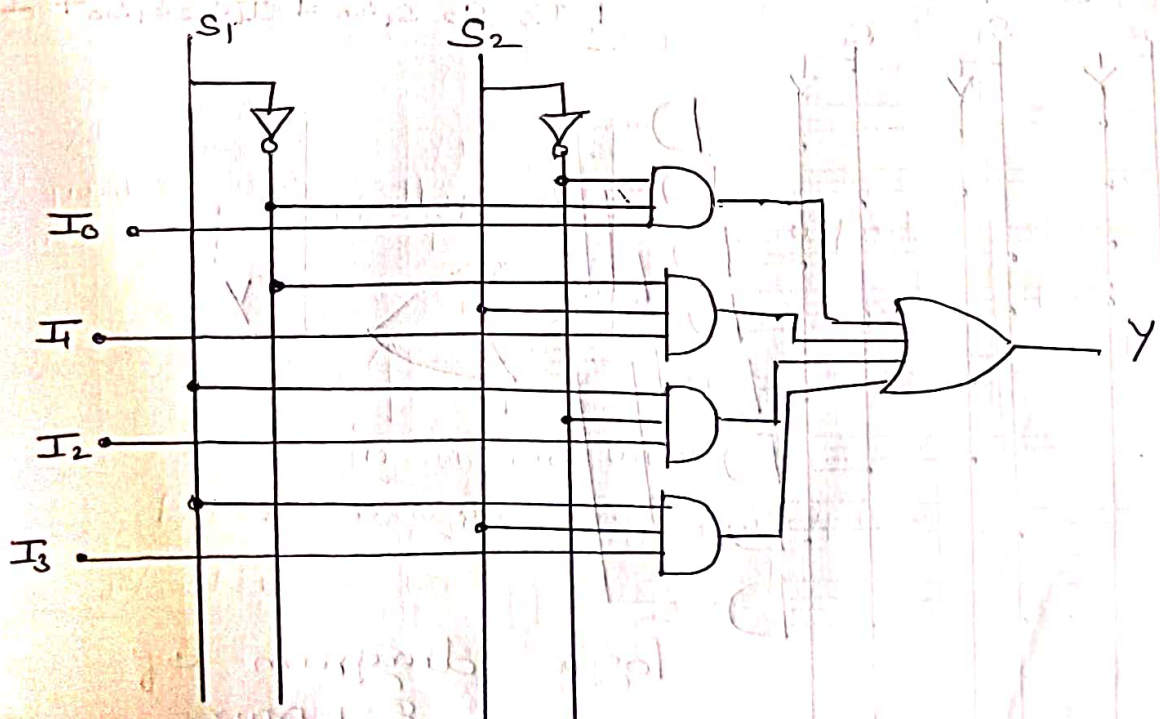
$$y = I_0 \bar{S}_1 \bar{S}_0$$

$$y = I_1 \bar{S}_1 S_0$$

$$y = I_2 S_1 \bar{S}_0$$

$$y = I_3 S_1 S_0$$

Hence output $y = I_0 \bar{S}_1 \bar{S}_0 + I_1 \bar{S}_1 S_0 + I_2 S_1 \bar{S}_0 + I_3 S_1 S_0$

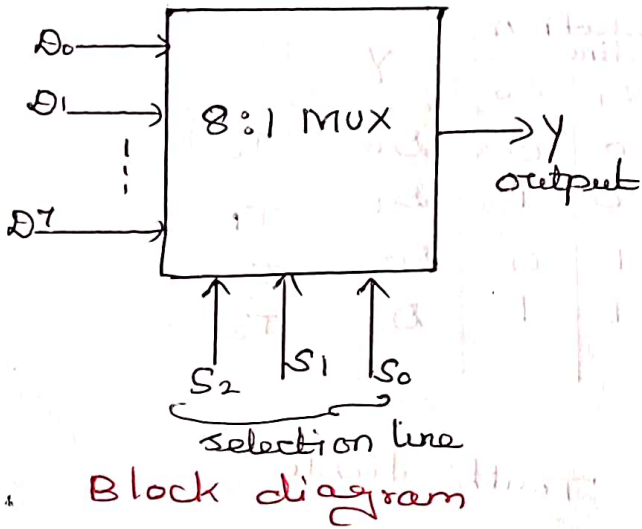


Logic diagram of 4:1 MUX

8:1 Multiplexer:-

* A 8:1 Multiplexer has 8 input and three selection lines and single output line

* $2^3 = 2 \times 2 \times 2 = 8$ (Inputs are $(D_0 - D_7)$)



selection Lines			output
S ₂	S ₁	S ₀	Y
0	0	0	I ₀
0	0	1	I ₁
0	1	0	I ₂
0	1	1	I ₃
1	0	0	I ₄
1	0	1	I ₅
1	1	0	I ₆
1	1	1	I ₇

Truth Table

$$Y = I_0 \cdot \bar{S}_2 \bar{S}_1 S_0$$

$$Y = I_1 \bar{S}_2 \bar{S}_1 S_0$$

$$Y = I_2 \bar{S}_2 S_1 \bar{S}_0$$

$$Y = I_3 \bar{S}_2 S_1 S_0$$

$$Y = I_4 S_2 \bar{S}_1 S_0$$

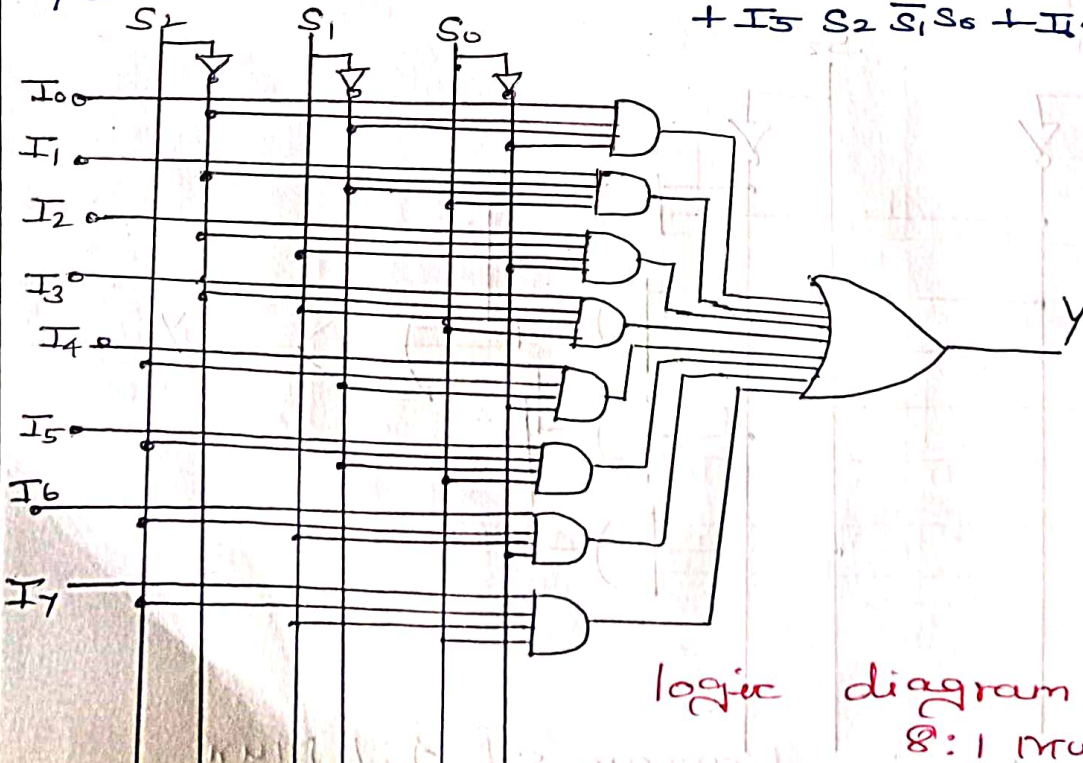
$$Y = I_5 S_2 \bar{S}_1 S_0$$

$$Y = I_6 S_2 S_1 \bar{S}_0$$

$$Y = I_7 S_2 S_1 S_0$$

Hence overall output is

$$Y = I_0 \bar{S}_2 \bar{S}_1 \bar{S}_0 + I_1 \bar{S}_2 \bar{S}_1 S_0 + I_2 \bar{S}_2 S_1 \bar{S}_0 + I_3 \bar{S}_2 S_1 S_0 + I_4 S_2 \bar{S}_1 \bar{S}_0 + I_5 S_2 \bar{S}_1 S_0 + I_6 S_2 S_1 \bar{S}_0 + I_7 S_2 S_1 S_0$$



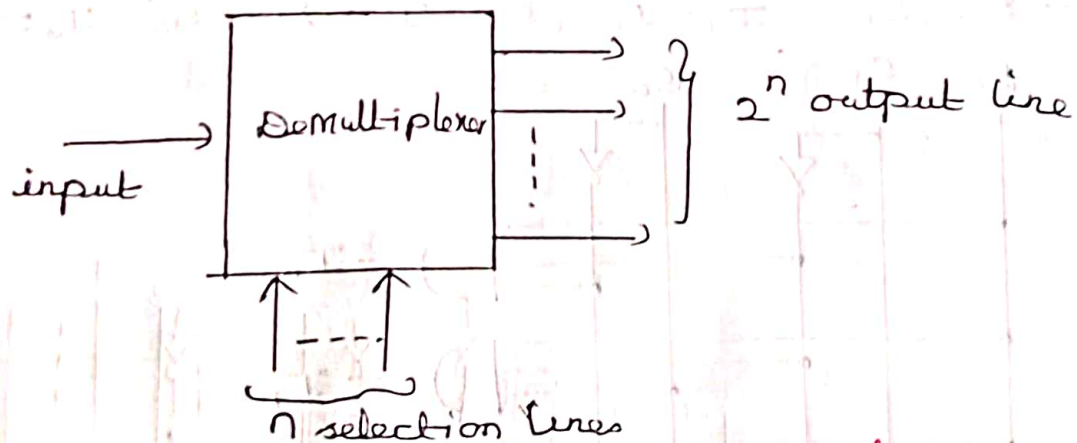
Application of Multiplexer:-

- * used in data acquisition system
 - * It is used in frequency multiplexing system
 - * It is used A/D and D/A converter
 - * It is used in time multiplexing system
 - * It is used in Implement combinational logic circuit
- It is used in data selector

Demultiplexer:-

* A Demultiplexer is a digital circuit that receives information on a single line and transmits the information on one of the several output lines

* It consists of one input line and n selection lines and 2^n output lines



Block diagram of Demultiplexer

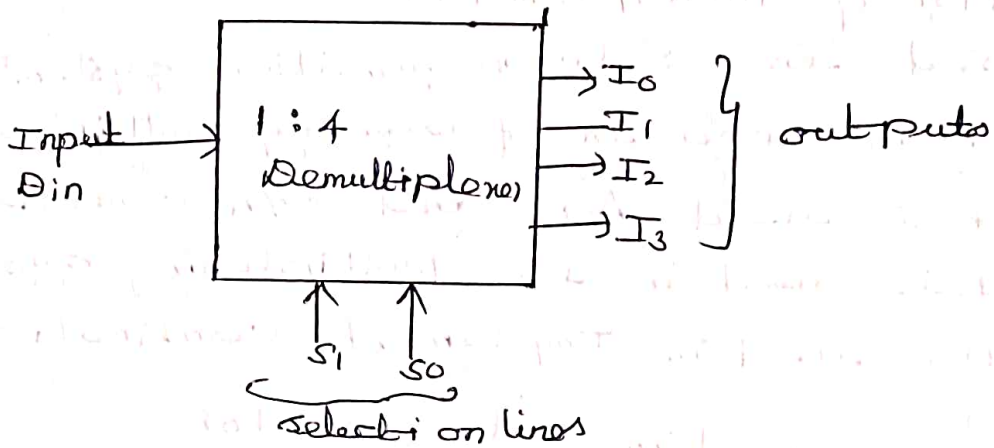
* It has two different types

↳ 1:4 Demultiplexer

↳ 1:8 Demultiplexer

1:4 Demultiplexer:-

* Consider a 1 to 4 demultiplexer which has a single input (A) and four outputs (Y_0, Y_1, Y_2, Y_3) and two selection lines (S_1 and S_0)

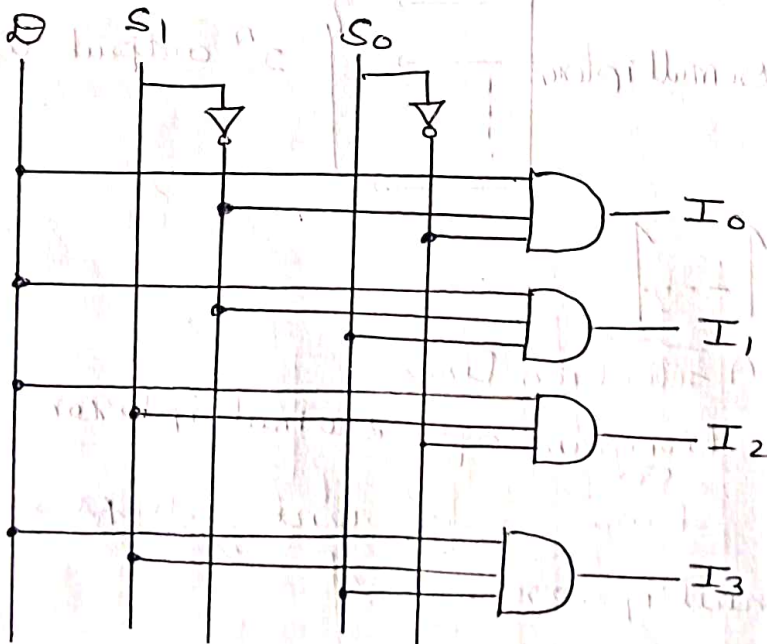


1:4 Demultiplexer

Inputs			Outputs			
S ₁	S ₀	Data Input	I ₀	I ₁	I ₂	I ₃
0	0	D	D	0	0	0
0	1	D	0	D	0	0
1	0	D	0	0	D	0
1	1	D	0	0	0	D

1:4 Demultiplexer Truth Table

$$I_0 = D \bar{S}_1 \bar{S}_0 \quad I_1 = D \bar{S}_1 S_0 \quad I_2 = D S_1 \bar{S}_0 \quad I_3 = D S_1 S_0$$



logic diagram of 1:4 Demux

1:8 Demultiplexer:-

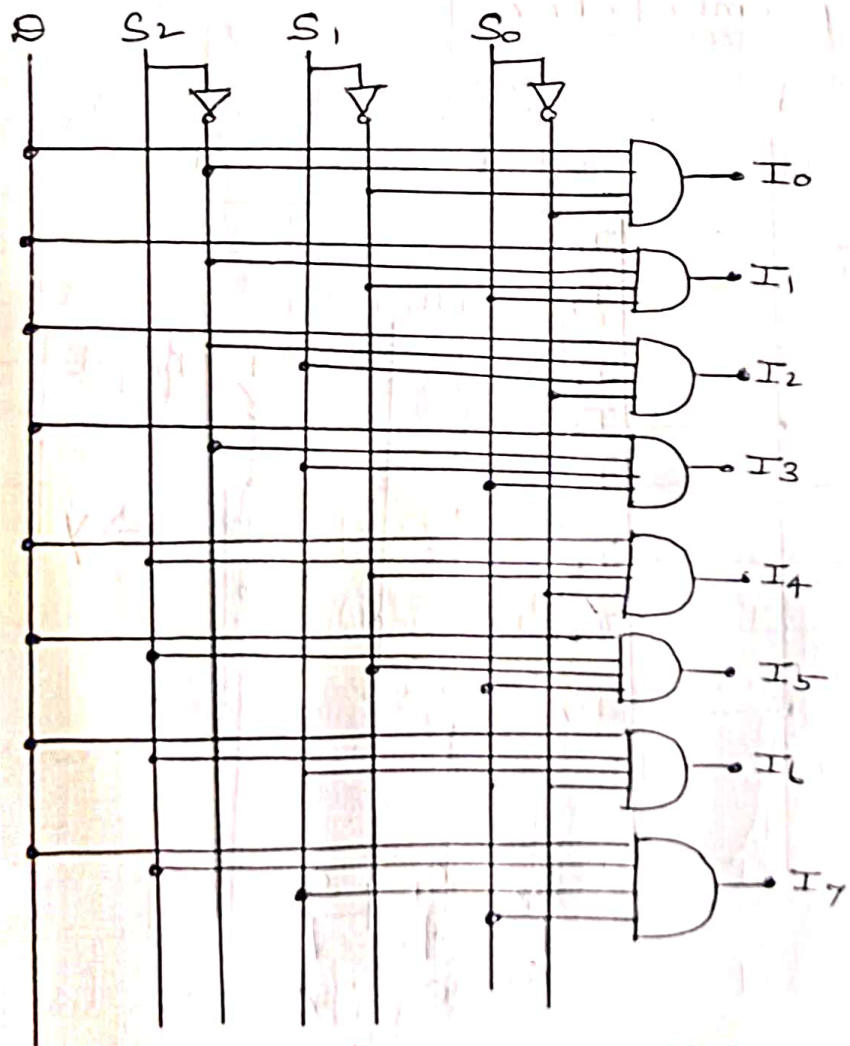
*consider a 1 to 8 demultiplexer which has a single input (D) and eight outputs

($I_0 - I_7$) and three selection inputs (S_2, S_1 & S_0)

Inputs				output							
S_2	S_1	S_0	Data Input	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0

$I_0 = \overline{S_2} \overline{S_1} \overline{S_0}$ $I_1 = \overline{S_2} \overline{S_1} S_0$ $I_2 = \overline{S_2} S_1 \overline{S_0}$ $I_3 = \overline{S_2} S_1 S_0$

$I_4 = S_2 \overline{S_1} \overline{S_0}$ $I_5 = S_2 \overline{S_1} S_0$ $I_6 = S_2 S_1 \overline{S_0}$ $I_7 = S_2 S_1 S_0$



Logic diagram of 1:8 De-Mux

Implement the following Boolean expression using a suitable multiplexer.

$$F(A, B, C, D) = \sum(0, 1, 3, 5, 7, 9, 11, 14, 15)$$

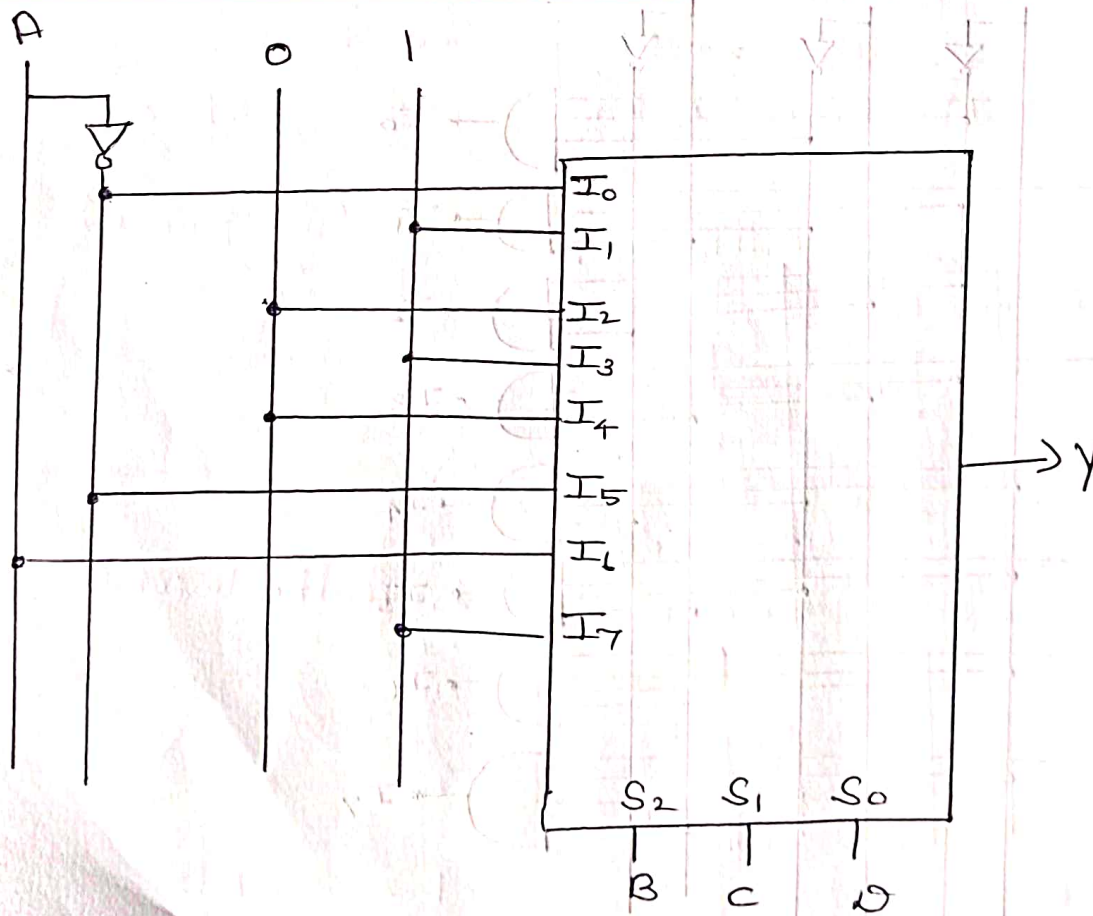
sol

* It has 4 variables, we need a multiplexer with 3 selection lines and 8 inputs

Implementation table

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
\bar{A}	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
	A	1	0	1	0	\bar{A}	A	1

Multiplexer implementation



Decoder :-

* A decoder is a combinational circuit that converts binary information from n input lines to maximum 2^n unique output lines.

* Each output line will be activated for only one of the possible combination of inputs.

* Based on the number of output and input decoder can be classified as

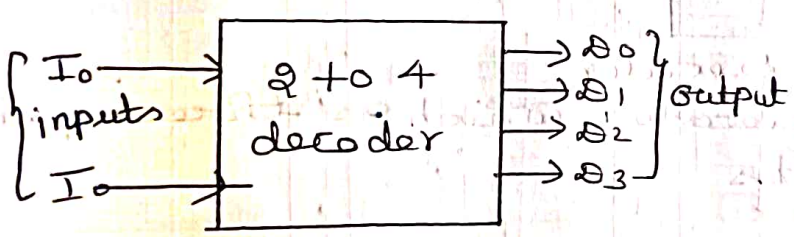
↳ 2 to 4 line decoder

↳ 3 to 8 line decoder

↳ 4 to 16 line decoder

2 to 4 line Decoder :-

* A 2 to 4 decoder consists of 2 inputs lines I_1 and I_0



Input		Outputs			
I_1	I_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

2 to 4 line decoder

from

$D_0 = \overline{I_1} \overline{I_0}$

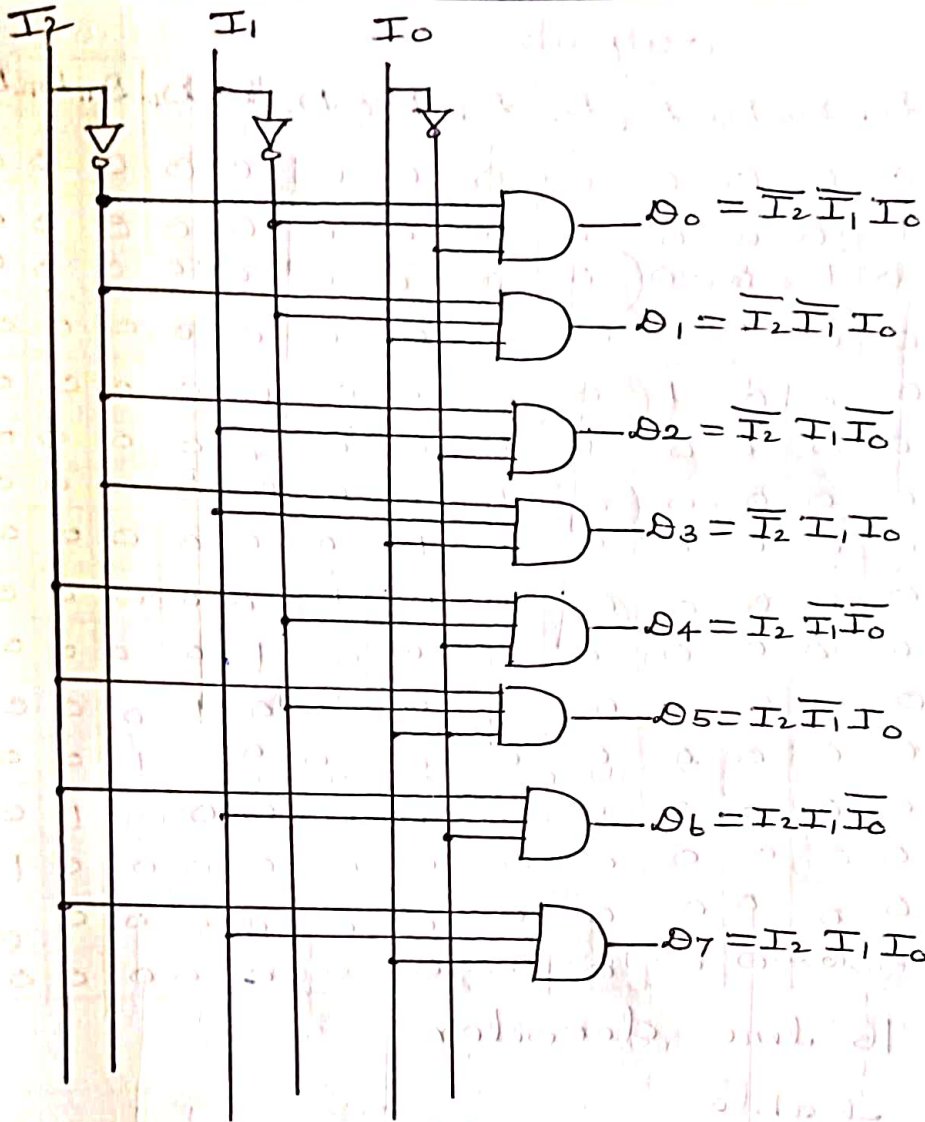
$D_1 = \overline{I_1} I_0$

$D_2 = I_1 \overline{I_0}$

$D_3 = I_1 I_0$

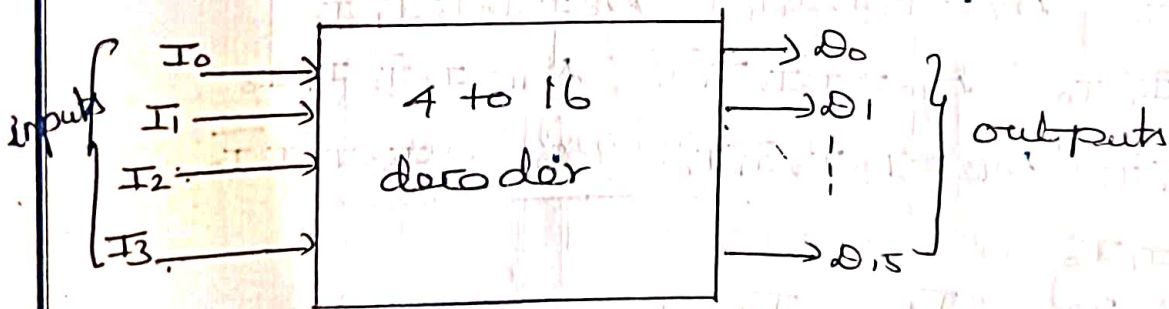
- * D_0 is active when the inputs $I_1 = I_0 = 0$.
- * D_1 is active when the inputs $I_1 = 0$ & $I_0 = 1$
- * D_2 is active when the inputs $I_1 = 1$ and $I_0 = 0$
- * D_3 is active when the inputs $I_1 = 1$ and $I_0 = 1$

Truth table 2 to 4 line decoder



Logic diagram

4 to 16 line decoder:-



4 to 16 line decoder

Inputs				Outputs															
I_3	I_2	I_1	I_0	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	D_{12}	D_{13}	D_{14}	D_{15}
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

4 to 16 line decoder

From truth table

$$D_0 = \overline{I_3} \overline{I_2} \overline{I_1} \overline{I_0}$$

$$D_1 = \overline{I_3} \overline{I_2} I_1 \overline{I_0}$$

$$D_2 = \overline{I_3} \overline{I_2} I_1 I_0$$

$$D_3 = \overline{I_3} I_2 \overline{I_1} \overline{I_0}$$

$$D_4 = \overline{I_3} I_2 I_1 \overline{I_0}$$

$$D_5 = \overline{I_3} I_2 I_1 I_0$$

$$D_6 = I_3 \overline{I_2} \overline{I_1} \overline{I_0}$$

$$D_7 = I_3 \overline{I_2} I_1 \overline{I_0}$$

$$D_8 = I_3 \overline{I_2} I_1 I_0$$

$$D_9 = \overline{I_3} \overline{I_2} \overline{I_1} I_0$$

$$D_{10} = \overline{I_3} \overline{I_2} I_1 \overline{I_0}$$

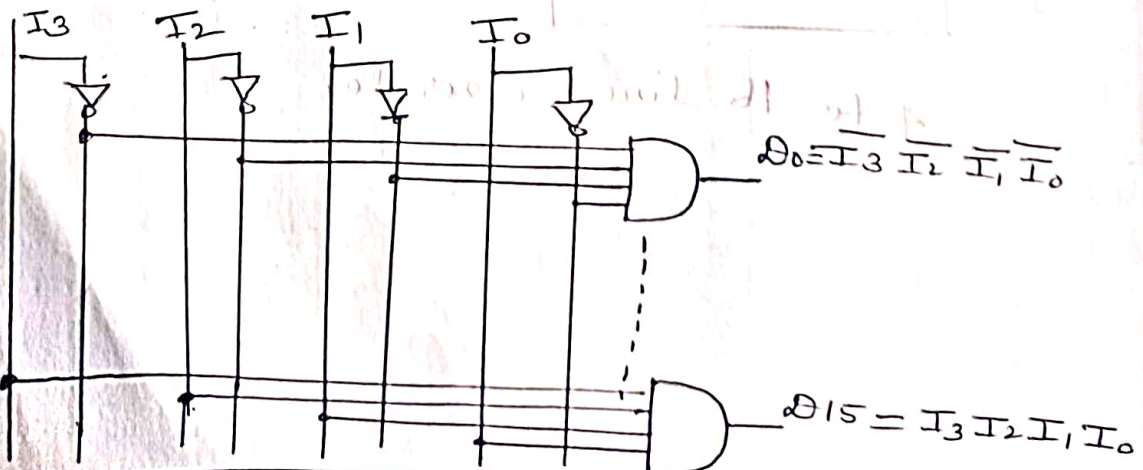
$$D_{11} = \overline{I_3} \overline{I_2} I_1 I_0$$

$$D_{12} = I_3 I_2 \overline{I_1} \overline{I_0}$$

$$D_{13} = I_3 I_2 I_1 \overline{I_0}$$

$$D_{14} = I_3 I_2 I_1 I_0$$

$$D_{15} = I_3 I_2 I_1 I_0$$

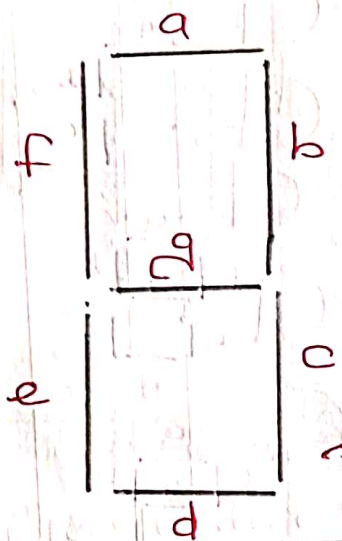


BCD to seven segment Display decoder

* The seven segment display is normally used for displaying any one of decimal digits 0 to 9.

* It has four inputs and (A, B, C & D) and seven output (a, b, c, d, e, f, g)

* The seven segment display as segments a, b, c, d, e, f & g which is constructed as



seven segment display

Digit.	Segment Activated	Display segment
0	a, b, c, d, e, f	<pre> a ----- f b e c d ----- </pre>
1	b, c	<pre> b c </pre>
2	a, b, d, e, g	<pre> a ----- g b e d ----- </pre>
3	a, b, c, d, g	<pre> a ----- g b d ----- </pre>

4	b, c, f, g	
5	a, c, d, f, g	
6	a, c, d, e, f, g	
7	a, b, c	
8	a, b, c, d, e, f, g	
9	a, b, c, d, f, g	

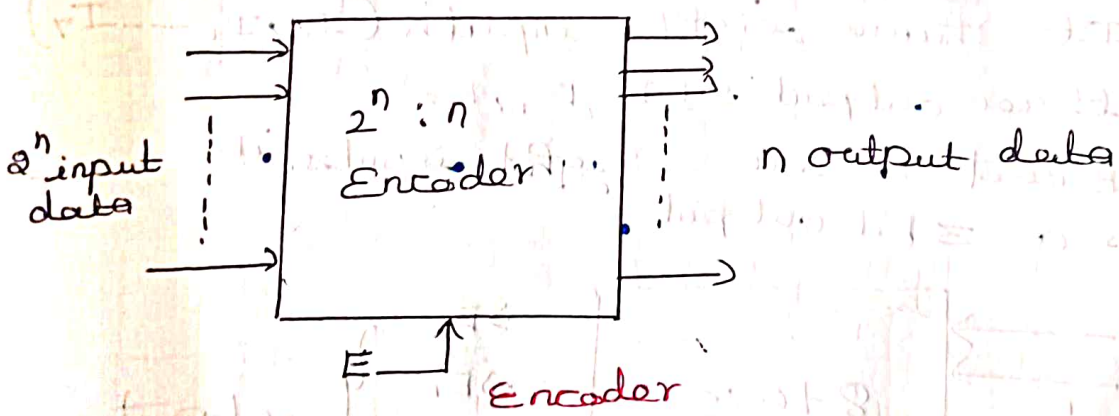
digit	BCD Input				segment output						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

Applications :-

- * counter system
- * analog to digital converter
- * seven segment display system

Encoder :-

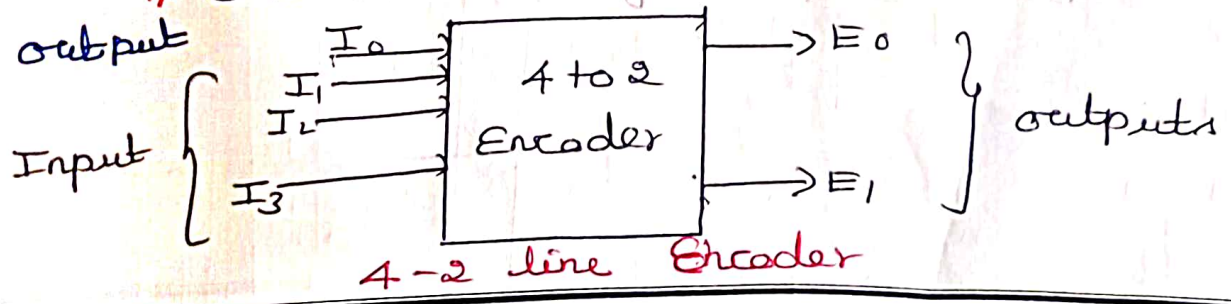
- * Encoder performs the inverse operation of decoder.
- * Encoder is a combinational circuit that converts 2^n number of input lines to n number of output line
- * Based on the number of output and inputs it is classified to the
 - ↳ 4 to 2 line encoder
 - ↳ 8 to 3 line encoder
 - ↳ priority encoder



4 to 2 line Encoder

* 4 to 2 line encoder has 4 input lines and 2 output line

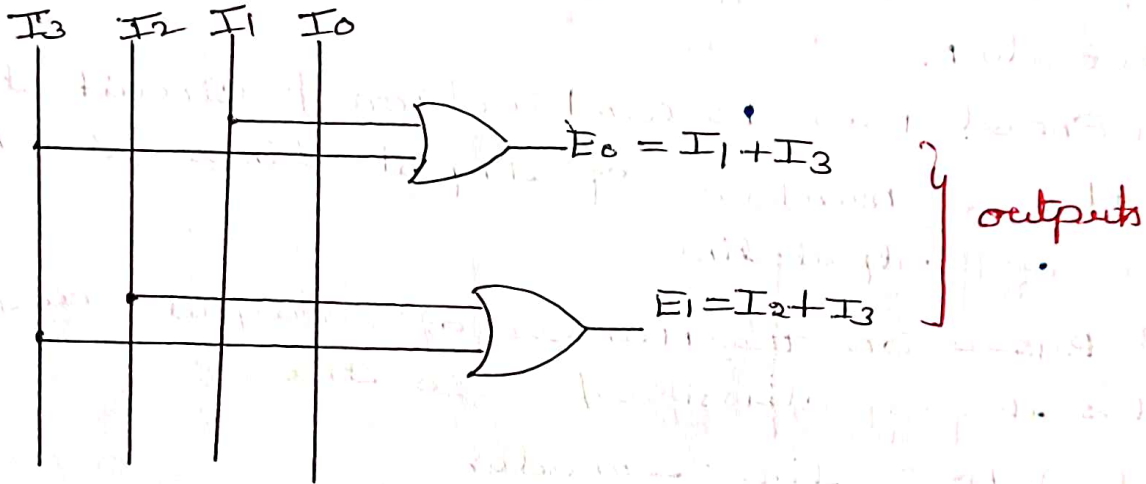
* Based on the 4 input it produces 2 bit



Inputs				Outputs	
I_3	I_2	I_1	I_0	E_1	E_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

$$E_0 = I_1 + I_3$$

$$E_1 = I_2 + I_3$$



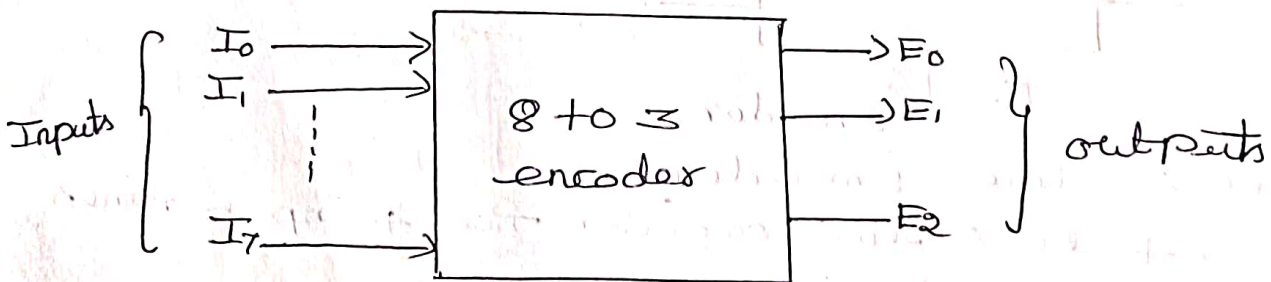
Inputs

Logic diagram 4-2 line encoder

8 to 3 line encoder

* It has eight inputs (I_0, I_1, \dots, I_7) and three outputs (E_0, E_1, E_2)

* Based on the eight inputs it produces a 3 bit output.



Block diagram of 8-3 line Encoder

Inputs								Outputs		
I ₀	I ₁	I ₂	I ₃	I ₄	I ₅	I ₆	I ₇	E ₂	E ₁	E ₀
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

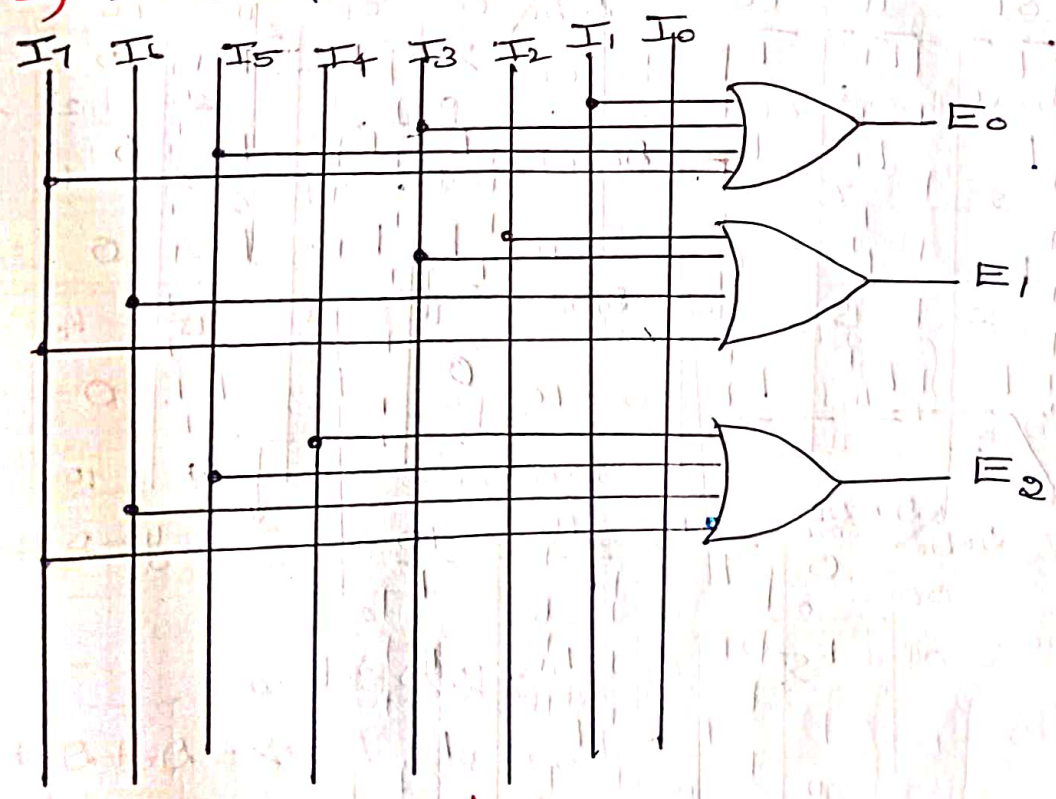
Truth table

* The logic expression for E₀, E₁, and E₂ is

↳ $E_0 = I_1 + I_3 + I_5 + I_7$

↳ $E_1 = I_2 + I_3 + I_6 + I_7$

↳ $E_2 = I_4 + I_5 + I_6 + I_7$



Logical diagram

Priority Encoder:-

* It is an encoder that includes the priority function.

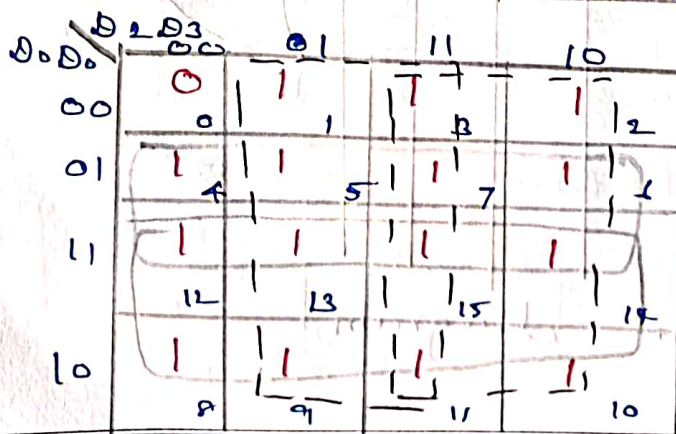
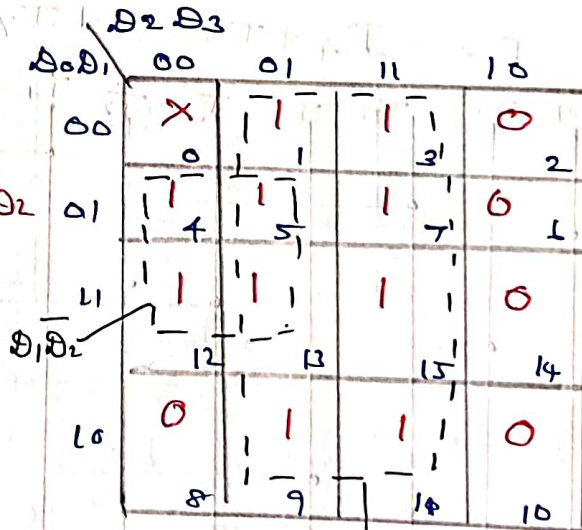
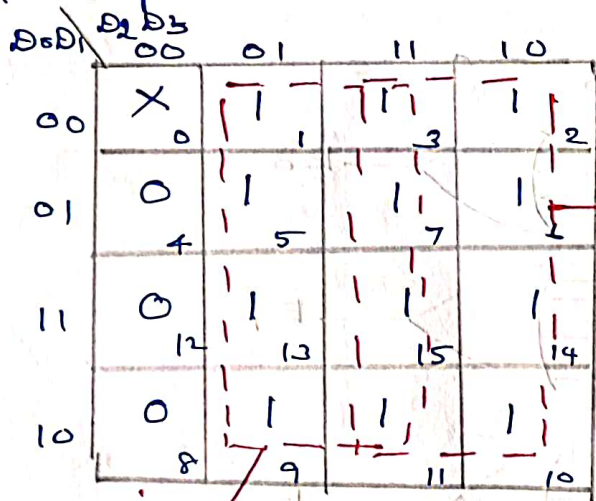
* If two or more inputs are equal to 1 at the same time having highest priority.

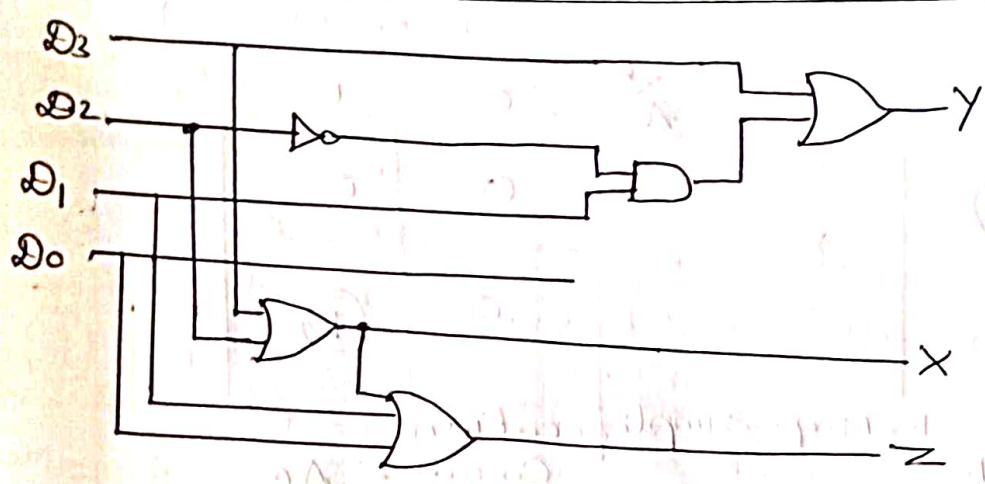
Inputs				Output		
D ₀	D ₁	D ₂	D ₃	X	Y	Z
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Truth table

* The simplified Boolean expression can be

K-Map





Logic diagram

Binary Adder - subtractor

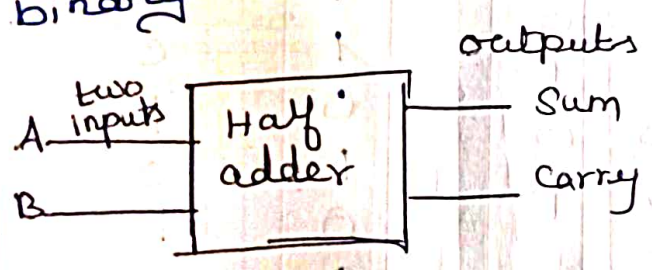
* Digital computers perform a variety of information processing tasks. among that most common functions are various arithmetic operations.

* The most basic arithmetic operation is addition of two binary digits

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

Half adder :-

* The half adder perform addition operation on two binary inputs and produce two binary output as a sum and carry bit



Input		output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

A \ B	0	1
0	0	1
1	1	0

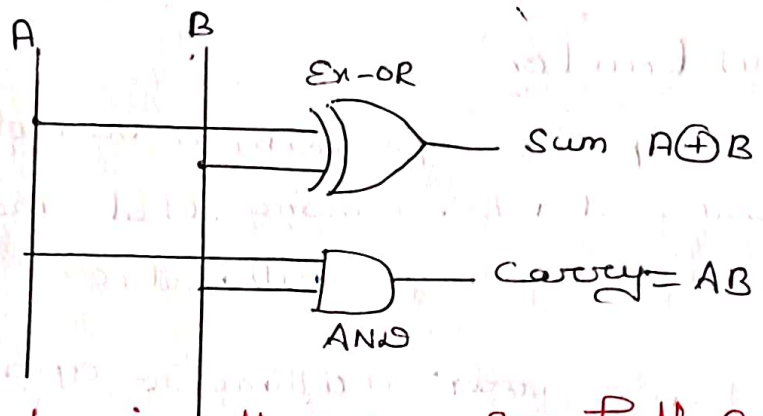
A \ B	0	1
0	0	0
1	0	1

K-Map simplification

$$\text{Sum} = A\bar{B} + \bar{A}B$$

$$\text{Carry} = AB$$

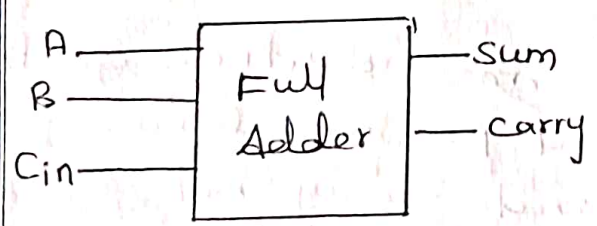
$$= A \oplus B$$



Logic diagram for half adder

Full Adder:-

*The full Adder performs addition operation on three binary inputs and produce output as sum and carry bit



Logic symbol

Inputs			Outputs	
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

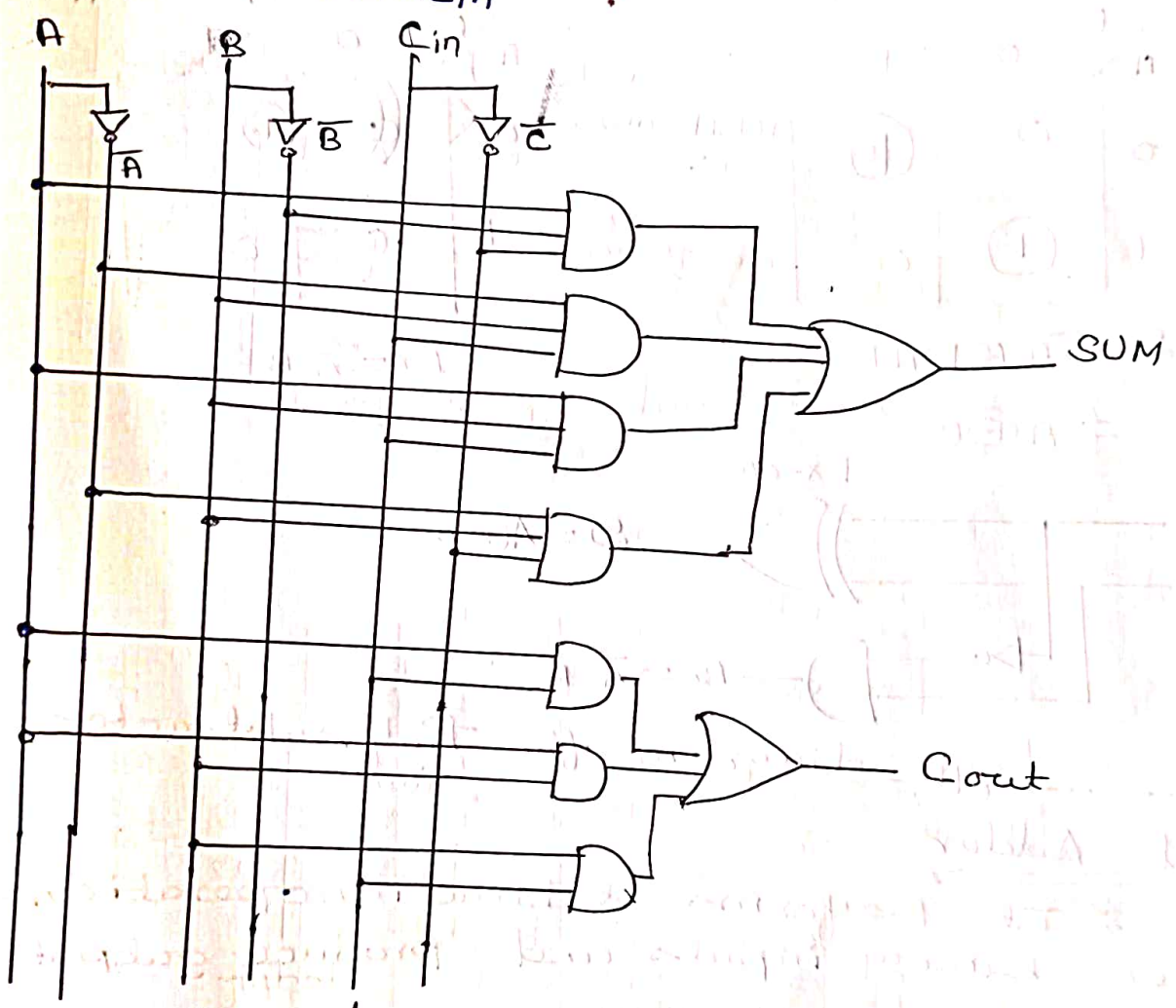
K-Map simplification

BCin	00	01	11	00
A	0	1	0	1
1	1	0	1	0
	0	1	3	2
	4	5	7	6

BCin	00	01	11	10
A	0	0	1	0
1	0	1	1	1
	0	1	3	2
	4	5	7	6

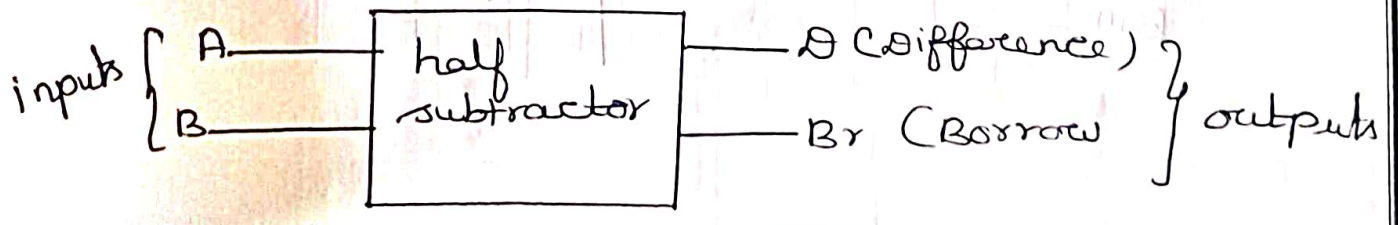
Sum = $A\bar{B}\bar{C}_{in} + \bar{A}\bar{B}C_{in} + ABC_{in} + \bar{A}BC_{in}$

Carry = $AC_{in} + AB + BC_{in}$



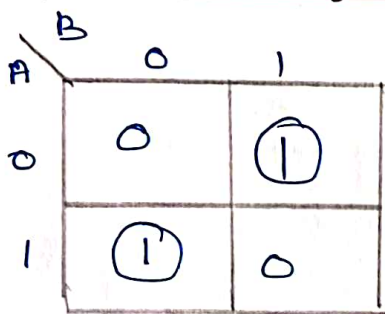
Half subtractor:-

* A half subtractor perform subtraction on two binary inputs and produce two binary output as difference and Borrow



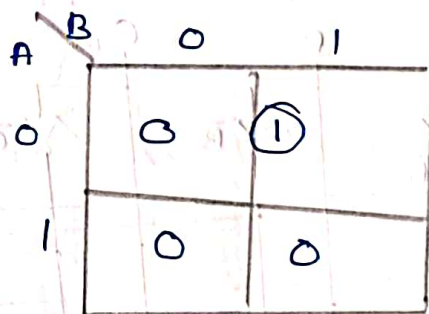
Input		output	
A	B	D	Br
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

K-Map simplification

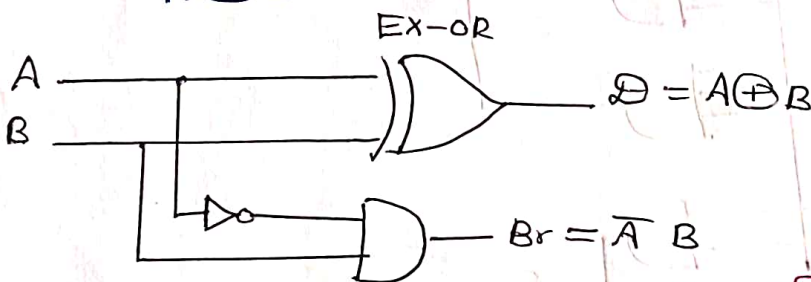


$$D = \bar{A}B + A\bar{B}$$

$$= A \oplus B$$



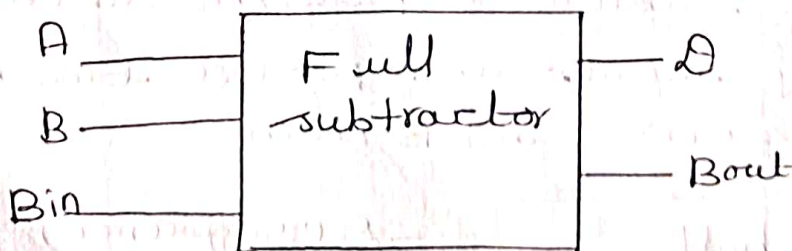
$$Br = \bar{A}B$$



Logic diagram for half subtractor

Full Adder

* It performs subtraction operation on three binary inputs and produce output as difference and carry bit



Input			output	
A	B	B _{in}	D	B _{out}
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

K-Map simplification

		B _{in}			
		00	01	11	10
A	0	0	1	0	1
	1	1	0	1	0
		0	1	3	2
		4	5	7	6

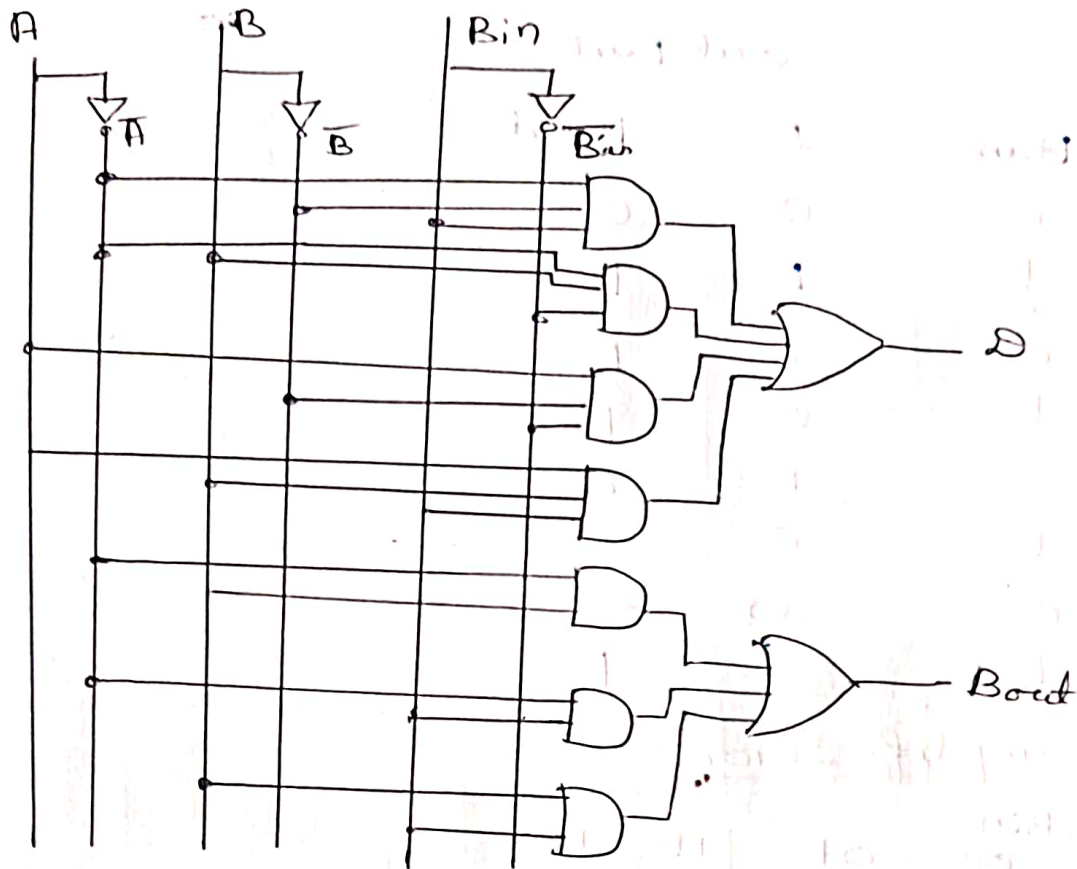
for D

$$D = \bar{A}\bar{B}B_{in} + \bar{A}B\bar{B}_{in} + A\bar{B}\bar{B}_{in} + AB B_{in}$$

		B _{in}			
		00	01	11	10
A	0	0	1	1	1
	1	0	0	1	0
		0	1	3	2
		4	5	7	6

for B_{out}

$$B_{out} = \bar{A}B_{in} + \bar{A}B + B B_{in}$$



logic diagram

UNIT-III

Synchronous sequential circuits

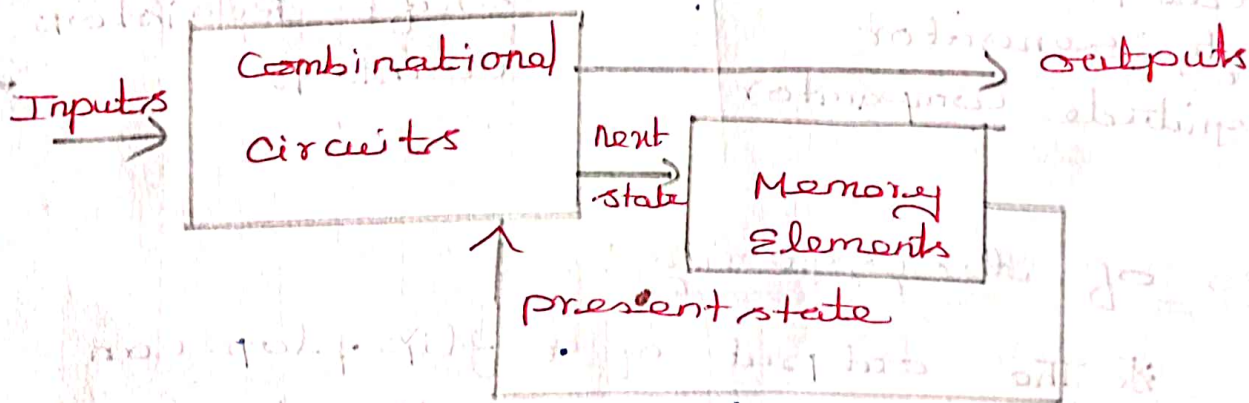
sequential circuit

* In sequential logic circuit it consists of combinational circuit to which storage elements are connected to form a feedback path

* The storage elements are capable of storing binary information either 1 or 0

* The information stored in the memory elements at any given time is the present state

* The present state and the external circuit determine the output and the next state of sequential circuits



block diagram

* The output variable depends not only on the present input variables but also on the past history of input variables

Comparison between combinational and sequential circuits

combinational logic	Sequential Logic
<ul style="list-style-type: none">* The output variable at all times depends on the combination of input variables	<ul style="list-style-type: none">* output variable depends not only on the present input but also depend upon the past history of input
<ul style="list-style-type: none">* Memory unit is not required	<ul style="list-style-type: none">* Memory unit is required to store the past history of input variables
<ul style="list-style-type: none">* Faster in speed	<ul style="list-style-type: none">* slower than combinational circuit
<ul style="list-style-type: none">* Easy to design	<ul style="list-style-type: none">* Hard to design
Ex:- <ul style="list-style-type: none">• AdderParity generatorMagnitude comparator	Ex:- <ul style="list-style-type: none">Shift RegistersCounters

Types of triggering:-

- * The output of a flip flop can be changed by bring a small change in the input signal
- * The small change can be brought with the help of a clock pulse. or commonly known as trigger pulse.

2

* When such trigger pulse is applied to the input the output changes and thus the flip-flop is said to be triggered.

* Flip flops are applicable in designing counters or registers which store data in the form of multibit numbers.

* Such registers need a group of flip-flops connected to each other as sequential circuits. These sequential circuits require trigger pulses.

* The number of trigger pulses that is applied to the input of the circuit determines the number in a counter.

* A single pulse makes the bit move one position when it is applied onto registers that store multibit data.

* There are four different types of pulse triggering methods.

1) High Level Triggering

2) Low Level Triggering

3) Positive Triggering

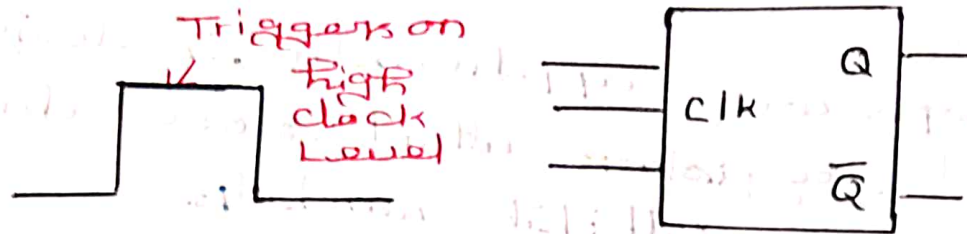
4) Negative Triggering

High Level Triggering:-

* When a flip-flop is required to respond at its high state a high level

triggering methods is used.

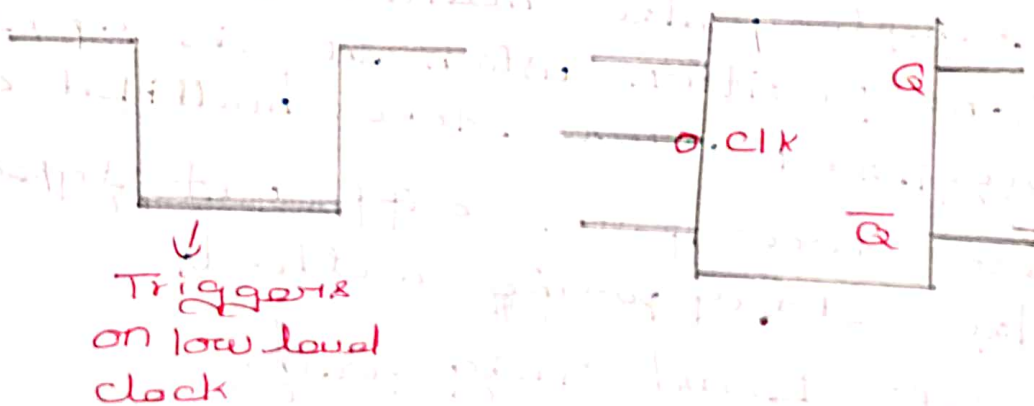
* It is mainly identified from the straight lead from the clock input



Low level triggering:-

* when a flip flop is required to respond at its Low state a Low level triggering method is used

* It is mainly identified from the clock input lead along with low state Indicator bubble.

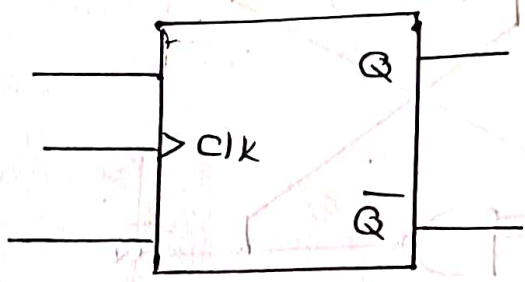
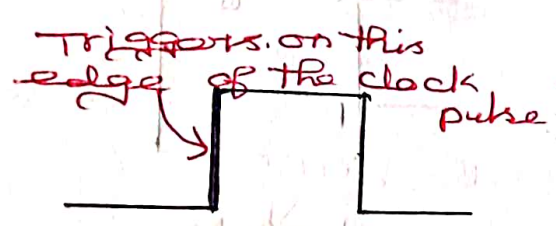


positive edge Triggering:-

* when a flip flop is required to respond at a Low to high transition state positive edge triggering method is used

* It is mainly identified from the

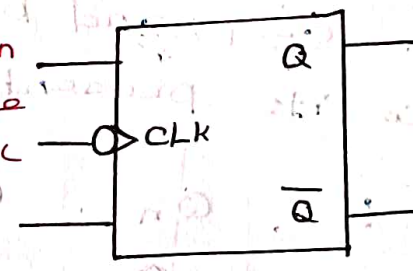
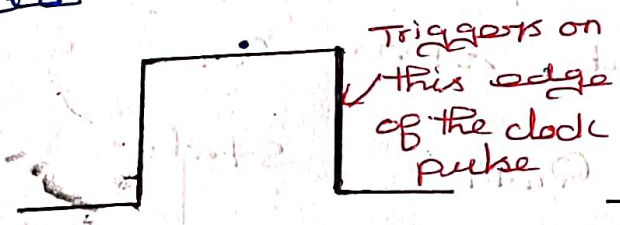
clock input lead along with a triangle



Negative edge Triggering :-

* When a flip flop is required to respond during the high to low transition state a Negative edge triggering method is use

* It is mainly identified from the clock input lead along with low state indicator and triangle

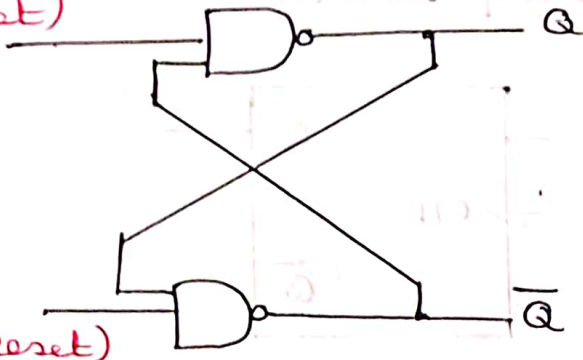


SR Latch using NAND Gates :-

* The SR Latch can also be implemented using NAND gates

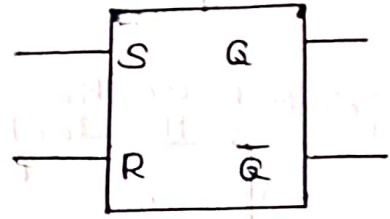
* The input of this Latch is S and R To understand how this circuit function recall that a low on any input to a NAND gates forces its output high

S (Set)



R (Reset)

SR Latch using NAND gates



Logic symbol

* when $S=0$ and $R=0$ the output of both gates will produce 0

* when $S=0$ and $R=1$ the latch is reset to 0

* when $S=1$ and $R=0$ the latch is set to 1

* when $S=1$ and $R=1$ the output Q_{n+1} remain in its present state Q_n

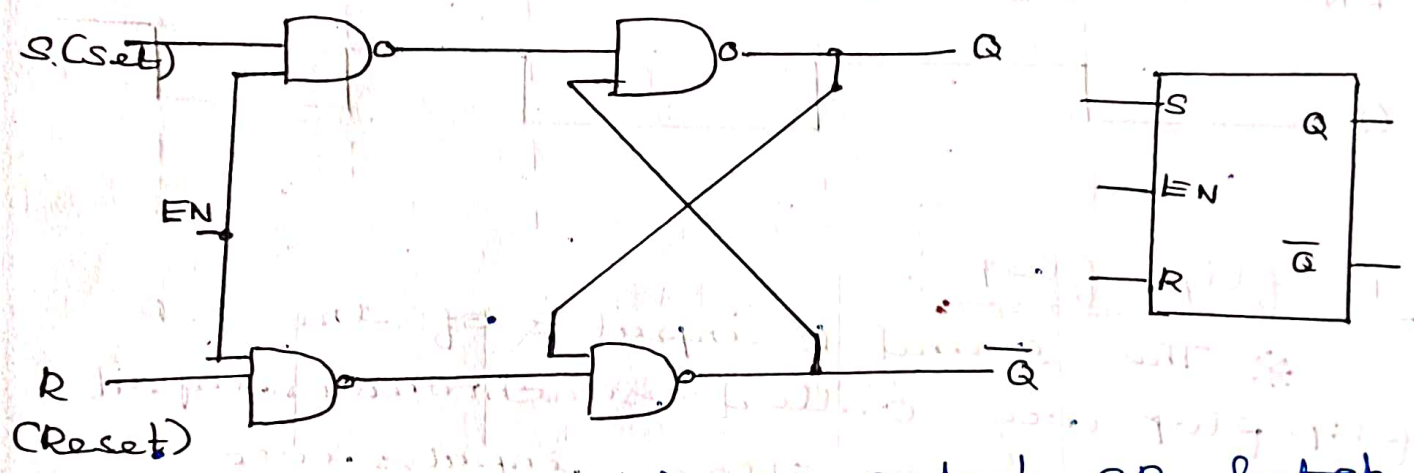
S	R	Q_n	Q_{n+1}	State
0	0	0	X	Indeterminate (*)
0	0	1	X	
0	1	0	1	Set
0	1	1	1	
1	0	0	0	Reset
1	0	1	0	
1	1	0	0	No change
1	1	1	1	

gated SR Latch :-

* In the SR Latch the output changes occur immediately after the input

Input changes

- * A Latch that is sensitive to the input only when an enable input is active
- * Such a latch with enable input is known as gated SR Latch
- * The circuit behaves like SR Latch when $EN=1$
- * It retains its previous state when $EN=0$

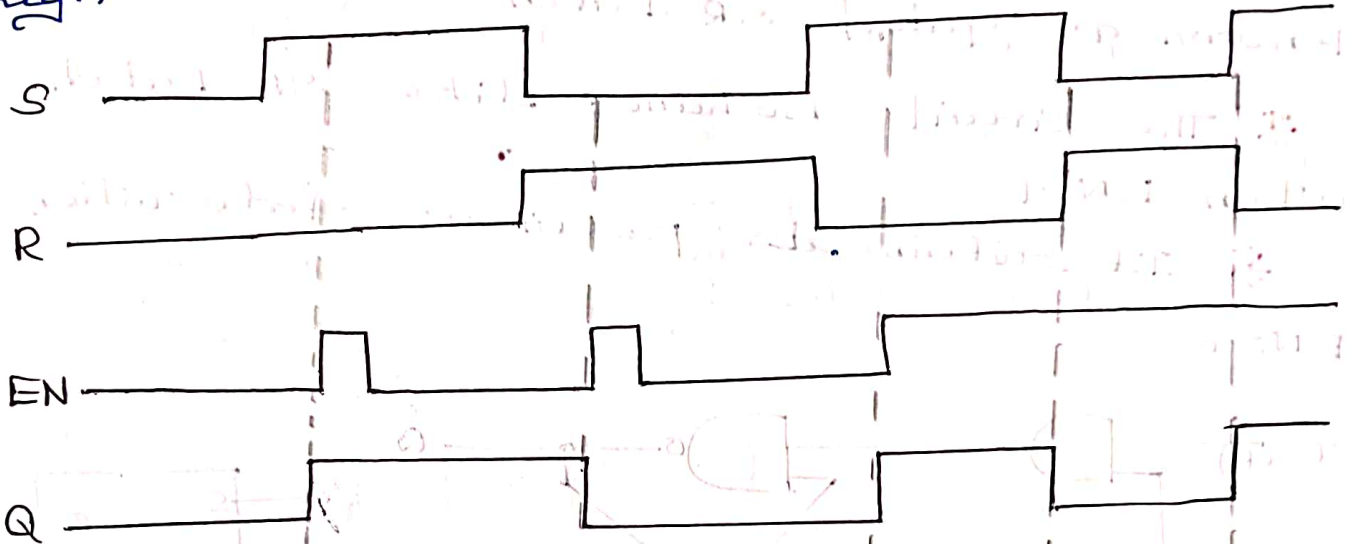


* The truth table of gated SR latch is shown below

EN	S	R	Q_n	Q_{n+1}	state
1	0	0	0	0	No change (NC)
1	0	0	1	1	
1	0	1	0	0	Reset
1	0	1	1	0	
1	1	0	0	1	Set
1	1	0	1	1	
1	1	1	0	X	Indeterminate *
1	1	1	1	X	
0	X	X	0	0	NO change (NC)
0	X	X	1	1	

* when S is high and R is Low, a high on the EN input set the Latch

* when S is Low and R is high, a high on the EN input resets the Latch

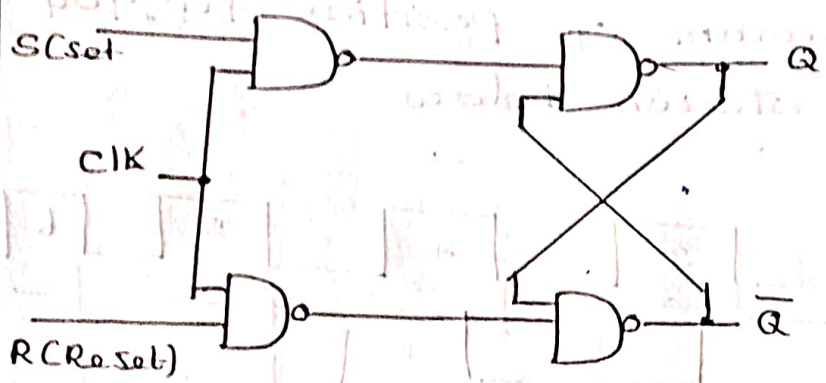


SR flip flop :-

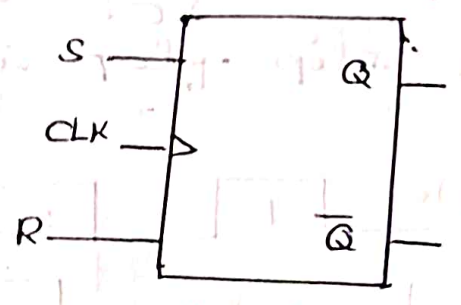
* The S and R inputs of the SR flip flop are called synchronous input because data on these inputs are transferred to the flip flop output only on the triggering edge of the clock pulse

* The SR latch circuit similar to SR flip flop but enable signal is replaced by clock pulse

* The positive edge of the clock pulse, the circuit responds to the S and R inputs



SR flip flop



Logic symbol

* when S is high and R is Low the Q output goes high on triggering edge of the clock pulse and flip flop set

* when S is Low and R is high, the Q output goes Low on the triggering edge of clock pulse and flip flop is Reset

* when S and R is Low output does not change from its prior state
An invalid condition exists when both S and R are high

CLK	S	R	Q _n	Q _{n+1}	state
1	0	0	0	0	No change CNC
1	0	0	1	1	
1	0	1	0	0	Reset
1	0	1	1	0	
1	1	0	0	1	Set
1	1	0	1	1	
1	1	1	0	X	Indeterminate *
1	1	1	1	X	

Truth table SR flip flop

* The timing diagram of positive triggered SR flip flop is shown below



Input output wave form of SR flip flop

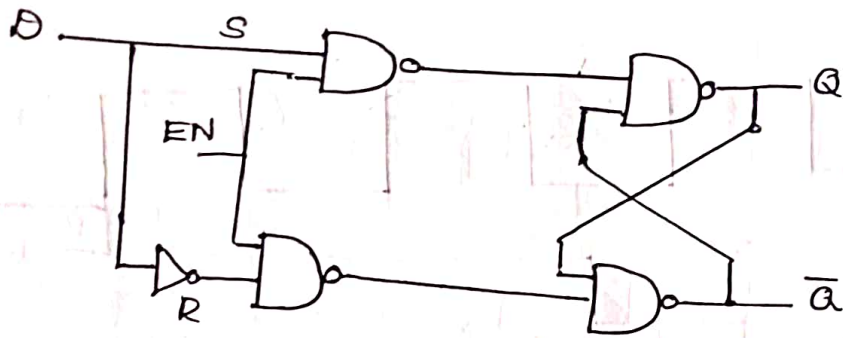
D-Latch:-

* In SR Latch, when both inputs are same (00 or 11) the output either does not change or it is invalid

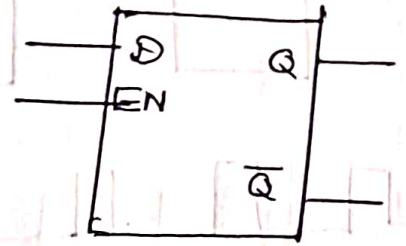
* In many practical applications these input conditions are not required

* These input conditions can be avoided by making them complement of each other

* This modified SR Latch is known as D Latch or Delay Latch



D-latch



Logic symbol

* D input goes directly to the S input and its complement is applied to the R input

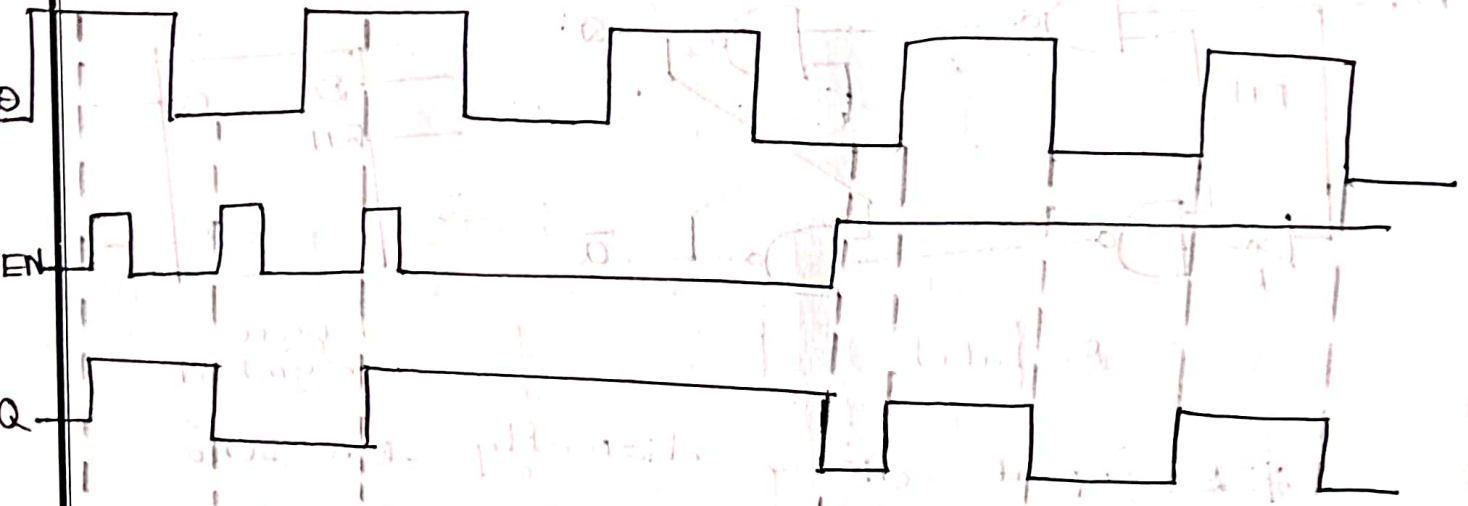
* Therefore only two input conditions exist either $S=0$ and $R=1$ or $S=1$ and $R=0$.

EN	D	Q_n	Q_{n+1}	State
1	0	X	0	Reset
1	1	X	1	Set
0	X	X	Q_n	No change (N.C.)

* Q output follows the D input for this reason this latch is called transparent latch

* when D is high and EN is high Q goes high when D is low and EN is high Q goes low

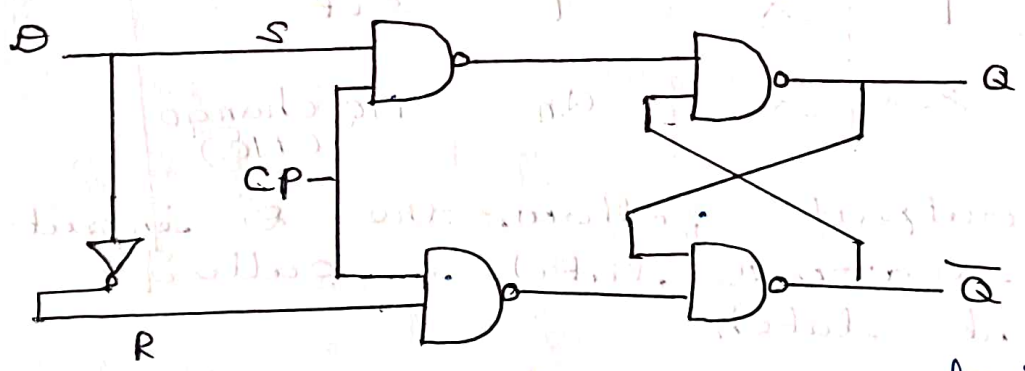
* when EN is low the state of this latch is not affected by the D-input



D flip flop using NAND gates:-

* Like in D Latch, in D flip flop the basic SR flip flop is used with complemented inputs

* The D flip flop is similar to D Latch except clock pulse is used instead of enable input

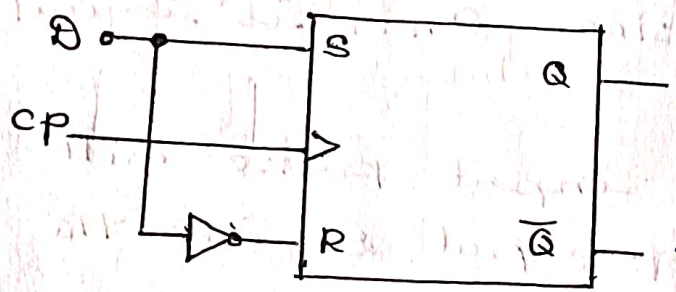


* To eliminate the undesirable condition of indeterminate state in the RS flip flop is to ensure that input S and R are never equal to 1 at the same time

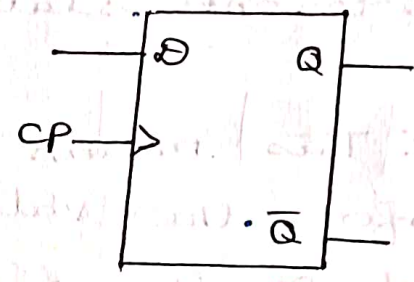
* This is done by D flip flop. The D flip flop has one input is called

delay input and clock pulse input

* The D flip flop using SR flip flop is shown below



using SR flip flop



graphic symbol

clock	D	Q_{n+1}	state
1	0	0	Reset
1	1	1	Set
0	x	Q_n	No change

* The timing diagram of positive edge triggered D flip flop is shown below



* Looking at the truth table for D flip flop we can realize that Q_{n+1} function follows the D input at the positive going edges of the clock pulse

Characteristic table & characteristic equation

* The D flip flop shows the next state of the flip flop is independent of present state since Q_{n+1} is equal to D

* This means that input pulse will transfer the value of input D into the output of the flip flop independent of the output before pulse was applied

Q_n	D	Q_{n+1}
0	0	0
0	1	1
1	0	0
1	1	1

$Q_n \backslash D$	0	1
0	0	1
1	0	1

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

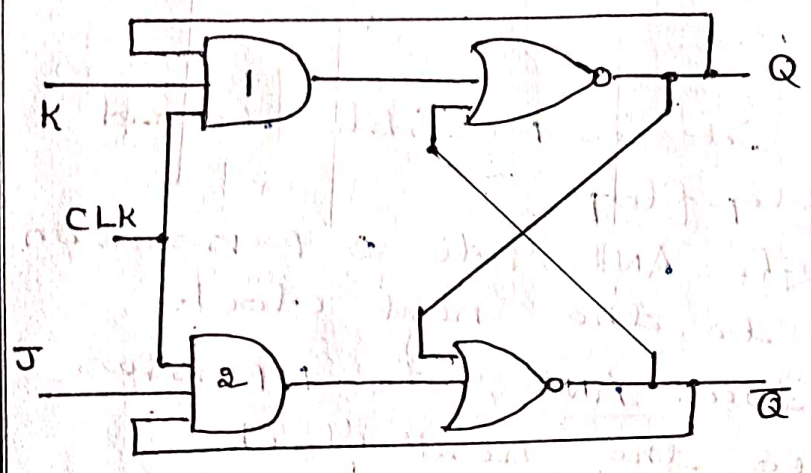
JK flip flop:-

* J.K flip flop has two inputs J (set) and K (Reset)

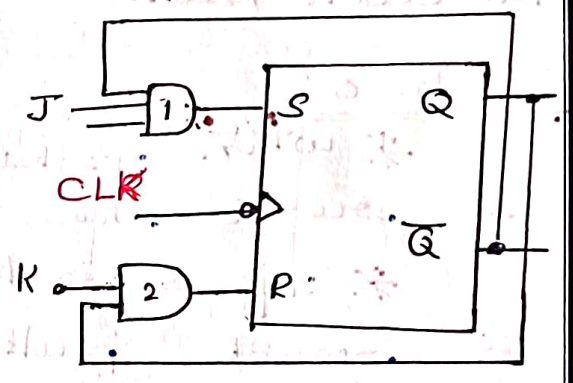
* A JK flip flop can be obtained from the clocked S-R flip flop by augmenting two AND gates

* The data input J and output Q are applied to the first AND gate and its output (JQ) is applied to the S input of SR flip flop.

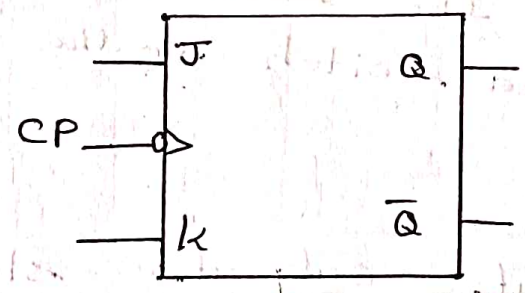
* Similarly the data input K and the output Q are applied to the second AND gate and its output (KQ) is applied to the R input SR-flip flop



JK flip flop



using SR flip flop



Graphic symbol

$J = K = 0$:-

* when $J = K = 0$ both AND gates are disabled. Then clock pulse have no effect hence the flip flop output is previous output

$J = 0, K = 1$:-

* when $J = 0$, and $K = 1$ AND gate is disabled $S = 0, R = 1$ This condition will Reset the flip flop to zero

$J=1, K=0$:-

* when $J=1$ and $K=0$ AND gate 2 is disabled $\therefore S=1, R=0$ therefore the flip flop will set on the application of the clock pulse

$J=K=0$:-

* when $J=K=1$ it is possible to set or reset the flip flop

* If Q is high AND gate 2 passes on a reset pulse to the next clock

* when Q is low AND gate 1 passes on a set pulse to the next clock

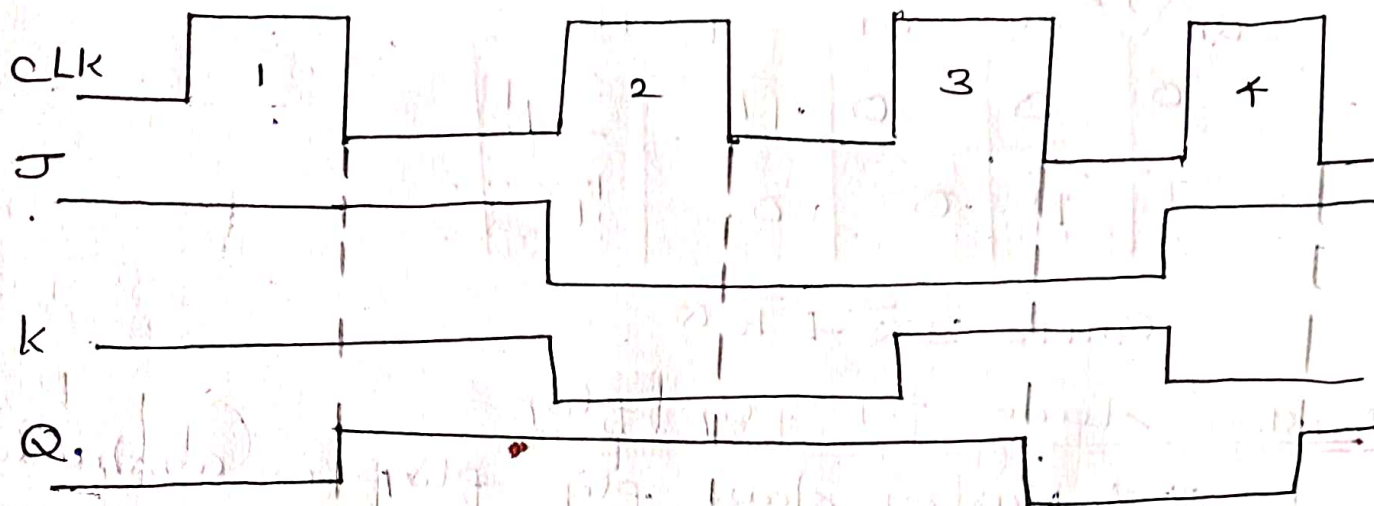
* Either way Q changes to the complement of the last state i.e. toggle. Toggle means to switch to the opposite state.

Truth Table :-

CLK	Inputs J, K	Outputs Q_{n+1}	state
1	0 0	Q_n	No change
1	0 1	0	Reset
1	1 0	1	Set
1	1 1	Q_n	Toggle

* The timing diagram of negative edge triggered JK flip flop is shown in below

over



Input output wave form of JK flip flop.

Characteristics table :-

* The characteristics table for JK flip flop is shown in the below.

* From the table K map for the next state transition Q_{n+1} can be drawn and simplified.

* Logic expression which represents the characteristics equation of JK flip flop

Q_n	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Characteristics table

Q_n \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

$$Q_{n+1} = J\bar{Q} + \bar{K}Q$$

Master slave JK flip flop

* A Master slave flip flop is constructed using two separate JK flip flops

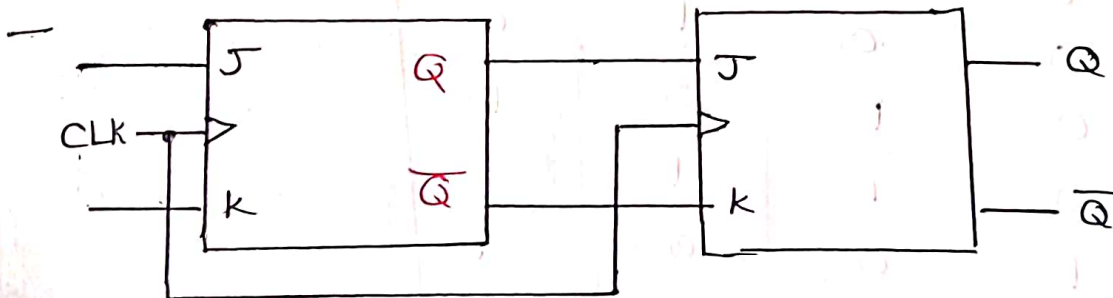
* It is, the first flip flop is called the master

* It is driven by positive edge of the clock pulse

* The second flip flop is called slave

* It is driven by the negative edge of the clock pulse

* The logic diagram of master slave JK flip flop is shown below



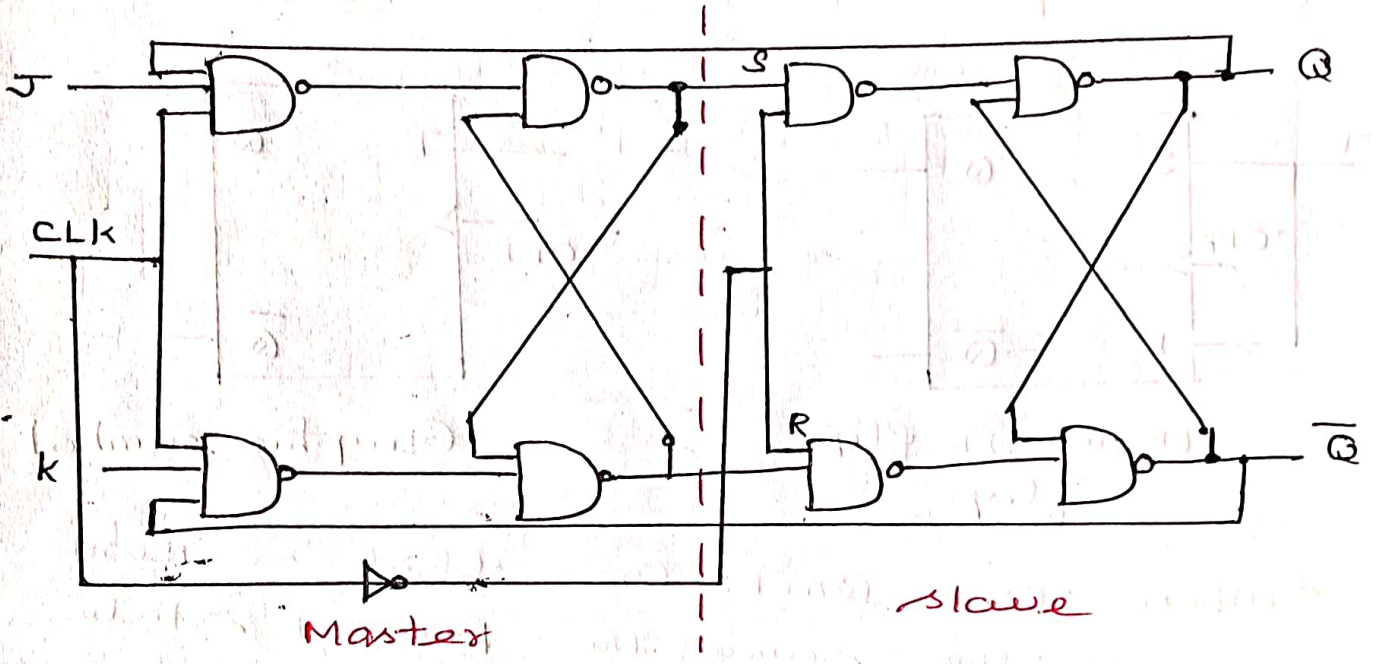
Logic diagram

* When the clock pulse has a +ve edge the master acts according to its JK inputs but the slave does not

does not respond. since it requires a negative edge at the clock input

* when the clock input has negative edge the slave flip flop copies master outputs

* But the master does not respond since it requires a positive edge at its clock input



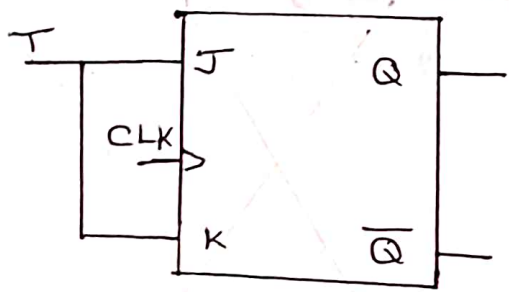
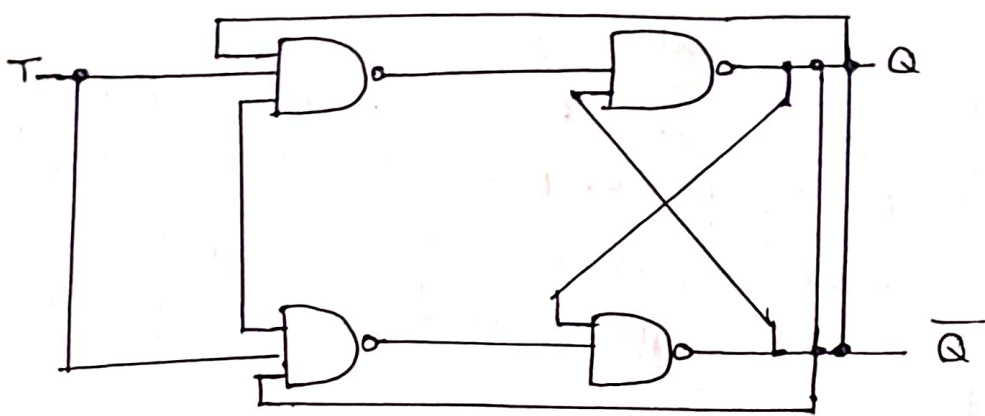
T- flip flop:-

* The T (Toggle) flip flop is a modification of the JK flip flop.

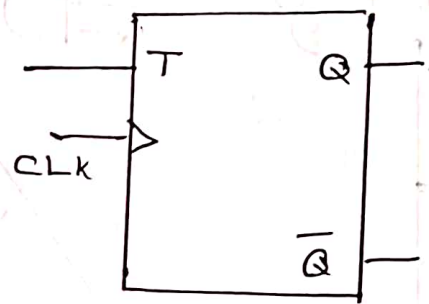
* It is obtained from JK flip flop by connecting both inputs J and K together i.e. single input

* Regardless of the present state the flip flop complements its output when the clock pulse occurs while input T=1

(4)



Using JK flip flop



Graphic symbol

* when $T=0$ $Q_{n+1} = Q_n$ that is next state is the same the present state and no change occurs

* when $T=1$ $Q_{n+1} = \overline{Q_n}$ that is next state is the complement of the present state

T	Q_{n+1}	state
0	Q_n	No change
1	$\overline{Q_n}$	Toggle

* The characteristics table for T flip flop is shown below and characteristics equation is derived by using K-Map

Q_n	T	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

$Q_n \backslash T$	0	1
0	0	1
1	1	0

$$Q_{n+1} = T\overline{Q_n} + \overline{T}Q_n$$

Conversion of flip flop:-

SR flip flop to D flip flop:-

step 1

* Draw the characteristics table of D flip flop and excitation table of SR flip flop as shown below

Characteristic table			Excitation table	
Q_n	D	Q_{n+1}	S	R
0	0	0	0	X
0	1	1	1	0
1	0	0	0	1
1	1	1	X	0

step 2:- obtain simplified expression for SR in terms of D and Q_n using

K-Map

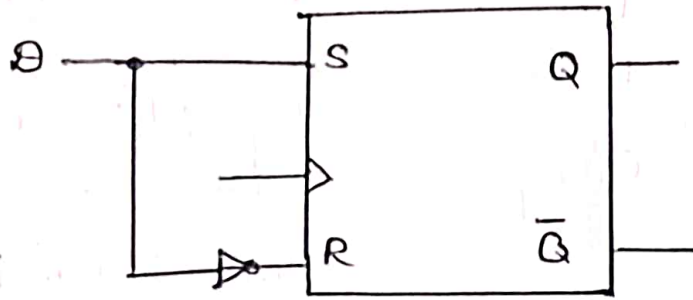
$Q_n \backslash D$	0	1
0	0	1
1	0	X

$$S = D$$

$Q_n \backslash D$	0	1
0	X	0
1	1	0

$$R = \overline{D}$$

step 3:- Draw the circuit of D Flip flop using SR flip flop



D flip flop using SR flip flop

SR flip flop to JK flip flop:-

step 1 : Draw the characteristics table of JK flip flop and excitation table of SR flip flop

Characteristics table				Excitation table	
Q_n	J	K	Q_{n+1}	S	R
0	0	0	0	0	X
0	0	1	0	0	X
0	1	0	1	1	0
0	1	1	1	1	0
1	0	0	1	X	0
1	0	1	0	0	1
1	1	0	1	X	0
1	1	1	0	0	1

step 2:-

* obtain simplified expression for S and R in terms of JK and Q_n using K-map

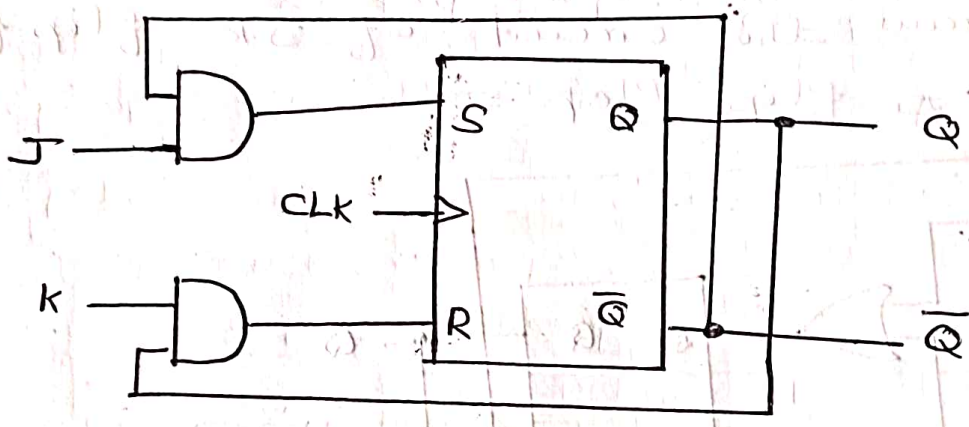
	J \ k	00	01	11	10
Q _n	0	0	0	1	1
	1	X	0	0	X

$S = \overline{Q_n} J$

	J \ k	00	01	11	10
Q _n	0	X	X	0	0
	1	0	1	1	0

$R = Q_n k$

step 3: Draw the circuit of JK flip flop



Convert a D flip flop into a JK flip flop:-

step 1: Draw the characteristic table of D flip flop and excitation table of JK flip flop as shown in the table

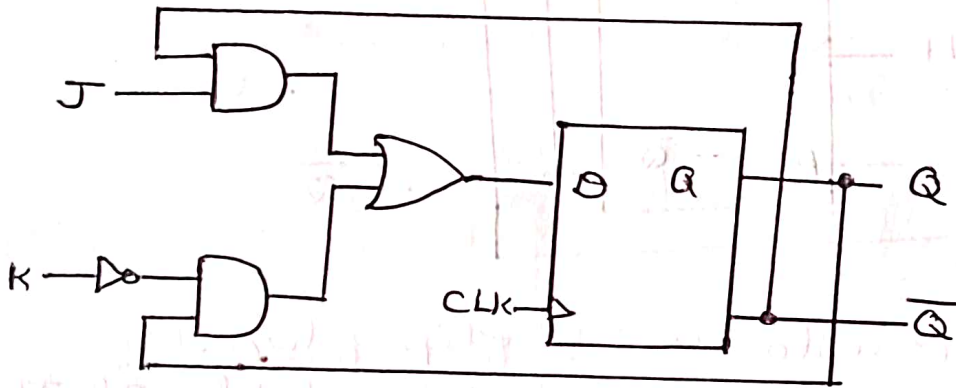
characteristics table				Excitation table	
Q _n	J	K	Q _{n+1}	D	
0	0	0	0	0	
0	0	1	0	0	
0	1	0	1	1	
0	1	1	1	1	
1	0	0	1	1	
1	0	1	0	0	
1	1	0	1	1	
1	1	1	0	0	

step 2:- obtain simplified expression for Δ in terms of Q_n and using K-Map

	JK	00	01	11	10
Q_n	0	0	0	1	1
	1	1	0	0	1

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

step 3:- Draw the circuit of JK flip flop using Δ flip flop



JK flip flop using Δ flip flop

JK flip flop to SR flip flop:-

step 1 : Draw the characteristics table of SR flip flop and excitation table JK flip flop

Characteristic table			Excitation table		
Q_n	S	R	Q_{n+1}	J	K
0	0	0	0	0	X
0	0	1	0	0	X
0	1	0	1	1	X
0	1	1	X	X	X
1	0	0	1	X	0
1	0	1	0	X	1
1	1	0	1	X	0
1	1	1	X	X	X

step 2: obtain the simplified expression for JK in terms of Qn and SR using K-Map

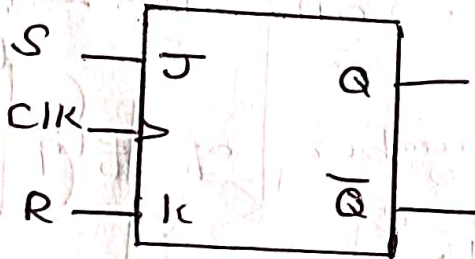
SR \ Qn	00	01	11	10
0	0	0	X	1
1	X	X	X	X

$$J = S$$

SR \ Qn	00	01	11	10
0	X	X	X	X
1	0	1	X	0

$$K = R$$

step 3: Draw the circuit of SR flip flop using JK



Module - N Counter:

Step 1: Find the Number of flip flops required

$$2^N \geq n$$

N → Number of flip flops

n → Mod Number

Step 2: write Excitation table respective flip flop

Step 3: Determine the transition table using the given condition and excitation table of the flip flop

Step 4: find the simplified expression for flip flop input using K-Map

step 5: Implement the logic diagram for Expression

Design a synchronous mod 6 counter using clocked JK flip flop

Mod-6 counter means it has 6 states of counting sequence that is the counter has count the value from 000 to 101 in decimal 0 to 5

count = 000 \rightarrow 001 \rightarrow 010 \rightarrow 011 \rightarrow 100 \rightarrow 101 \rightarrow 000
state = 1 2 3 4 5 6 1

step 1 :- Find the number of flip flop required

$$\text{condition } 2^N \geq n$$

$$2^N \geq 6$$

$$2^3 = 6$$

$$8 \geq 6$$

$N = 3$ (Three JK flip flops are required)

step 2 :- write an excitation table for JK flip flop

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Step 3: determine the transition table

Present state			Next state			Flip flop Input					
Q _A	Q _B	Q _C	Q _{A+1}	Q _{B+1}	Q _{C+1}	J _A	K _A	J _B	K _B	J _C	K _C
0	0	0	0	0	1	0	x	0	x	1	x
0	0	1	0	1	0	0	x	1	x	x	1
0	1	0	0	1	1	0	x	x	0	1	x
0	1	1	1	0	0	1	x	x	1	x	1
1	0	0	1	0	1	x	0	0	x	1	x
1	0	1	0	0	0	x	1	0	x	x	1
1	1	0	x	x	x	x	x	x	x	x	x
1	1	1	x	x	x	x	x	x	x	x	x

Step 4: find the expression for flip flop Inputs

For J_A

Q _A \ Q _B Q _C	00	01	11	10
0	0	0	1	0
1	x	x	x	x

$J_A = Q_B Q_C$

For K_A

Q _A \ Q _B Q _C	00	01	11	10
0	x	x	x	x
1	0	1	x	x

$K_A = Q_C$

For J_B

Q _A \ Q _B Q _C	00	01	11	10
0	0	1	x	x
1	0	0	x	x

$J_B = \overline{Q_A} Q_C$

For K_B

Q _A \ Q _B Q _C	00	01	11	10
0	x	x	1	0
1	x	x	x	x

$K_B = Q_C$

FOR Jc

	$Q_B Q_C$	00	01	11	10
Q_A	0	1	X	X	1
	1	1	X	X	X

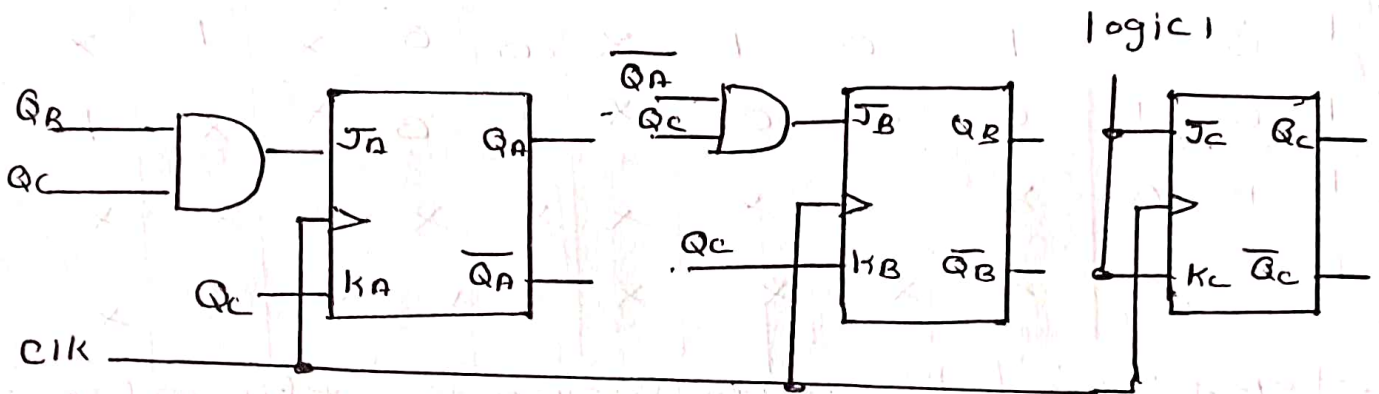
$J_c = 1$

FOR Kc

	$Q_B Q_C$	00	01	11	10
Q_A	0	X	1	1	X
	1	X	1	X	X

$K_c = 1$

step 5: - Implement the logic diagram of the counter



Logic diagram of Mod 6 synchronous counter

Design a synchronous Mod 5 counter using T flip flop:

* design a counter using T flip flop which count from 000 to 100 in decimal 0 to 4

000 → 001 → 010 → 011 → 100 → 000

step 1: Find the number of flip flops required

$$\text{condition } 2^n \geq n$$

$$2^3 \geq 5$$

∴ $N = 3$ (Three T flip flops required)

step 2 :- write an excitation table for T flip flop

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

step 3 :- determine the transition table

present state			Next state			Flip flop Inputs		
Q_A	Q_B	Q_C	Q_{A+1}	Q_{B+1}	Q_{C+1}	T_A	T_B	T_C
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	0	0	0	0	0	0
1	0	1	X	X	X	X	X	X
1	1	0	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X

step 4 :- find the expression for flip flop inputs

FOR T_A

Q_A	Q_B	Q_C	00	01	11	10
0	0	0	0	0	1	0
1	0	0	1	1	1	1
0	1	0	0	1	1	1
1	1	0	1	1	1	1

$T_A = Q_A + Q_B Q_C$

FOR T_B

Q_A	Q_B	Q_C	00	01	11	10
0	0	0	0	1	1	0
1	0	0	1	1	1	1
0	1	0	0	1	1	1
1	1	0	1	1	1	1

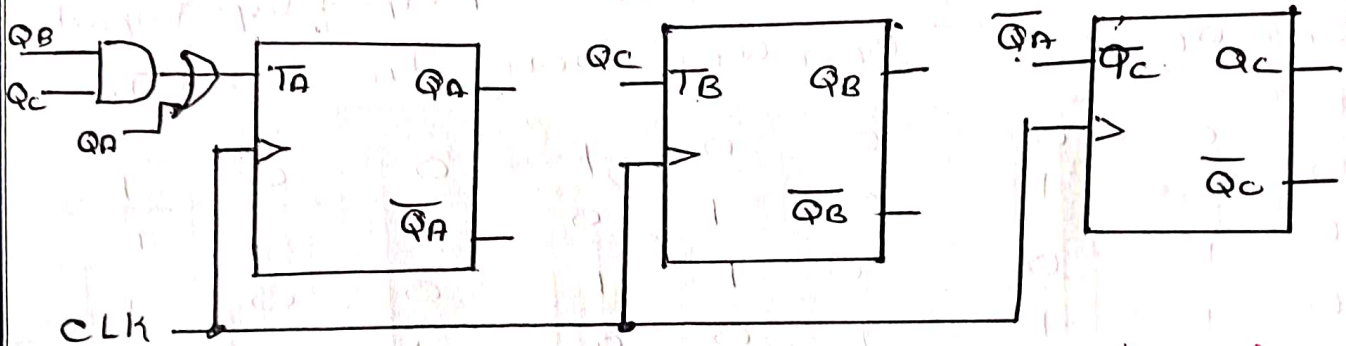
$T_B = Q_C$

For T_c

Q_A	Q_B	Q_C	00	01	11	10
0	0	0	1	1	1	1
1	0	0	0	X	X	X

$$T_c = \overline{Q_A}$$

step 5 Implement the Logic diagram



Logic diagram of Mod 5 counter using T flip flop

Design of a synchronous Mod 6 counter using clocked SR flip flop

* Design counter using SR flip flop which count from 000 to 101 in decimal 0 to 5

000 → 001 → 010 → 011 → 100 → 101 → 000

step 1: Find the number of flip flops required

$$2^N \geq n$$

$$2^3 \geq 6$$

Three SR flip flop used

step 2:- write an excitation table for SR flip flop

Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

step 3: determine the Excitation table

present state			Next state			Flip flop Inputs					
Q_A	Q_B	Q_C	Q_{A+1}	Q_{B+1}	Q_{C+1}	S_A	R_A	S_B	R_B	S_C	R_C
0	0	0	0	0	1	0	X	0	X	1	0
0	0	1	0	1	0	0	X	1	0	0	1
0	1	0	0	1	1	0	X	X	0	1	0
0	1	1	1	0	0	1	0	0	1	0	1
1	0	0	1	0	1	X	0	0	X	1	0
1	0	1	0	0	0	0	1	0	X	0	1
1	1	0	X	X	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X	X	X

step 4:- Find the expression for flip flop Inputs

For S_A

Q_A \ $Q_B Q_C$	00	01	11	10
0	0	0	1	0
1	X	0	X	X

For R_A

Q_A \ $Q_B Q_C$	00	01	11	10
0	X	X	0	X
1	0	1	X	X

$S_A = Q_B Q_C$

$R_A = \overline{Q_B} Q_C$

FOR S_B

$Q_B Q_C$	00	01	11	10
0	0	1	X	0
1	0	0	X	X

$$S_B = \overline{Q_A} Q_C$$

FOR R_B

$Q_B Q_C$	00	01	11	10
0	X	0	1	0
1	X	X	X	X

$$R_B = Q_B Q_C$$

FOR S_C

$Q_B Q_C$	00	01	11	10
0	1	0	0	1
1	1	0	X	X

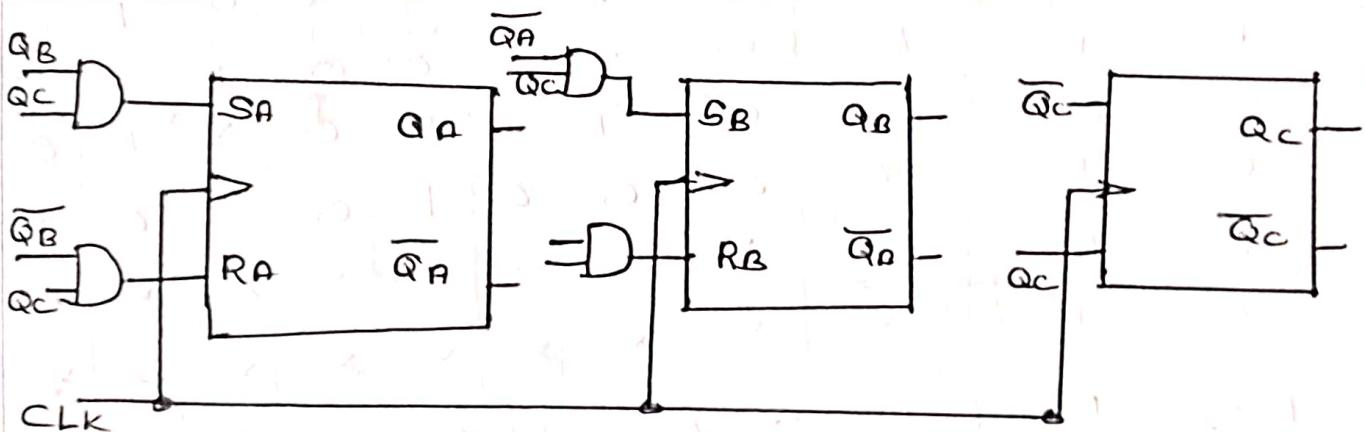
$$S_C = \overline{Q_C}$$

FOR R_C

$Q_B Q_C$	00	01	11	10
0	0	1	1	0
1	0	1	X	X

$$R_C = Q_C$$

Step 5 : Implement Logic diagram



Logic diagram of mod 6 counter using SR flip flop

Shift Registers:-

* A register capable of shifting the binary information held in each cell to its neighboring cell, in a selected direction is called shift register

* The logical configuration of a shift register consists of a chain of flip flops in cascade

* In shift register the output of one flip flop connected to the input of next flip flop

* All flip flop receive common clock pulses which activate the shift of data from one stage to the next stage

Types:-

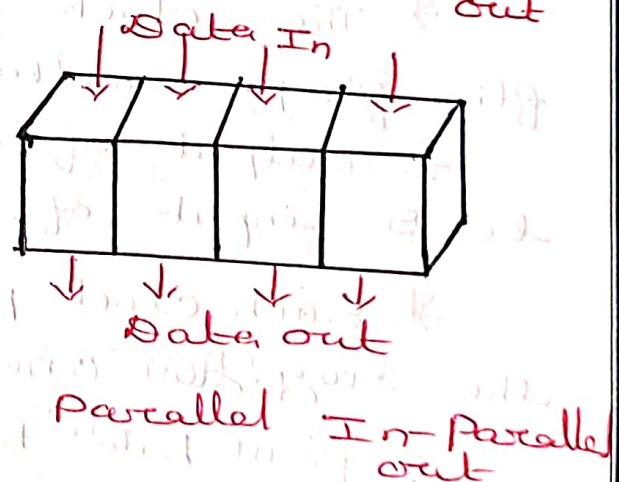
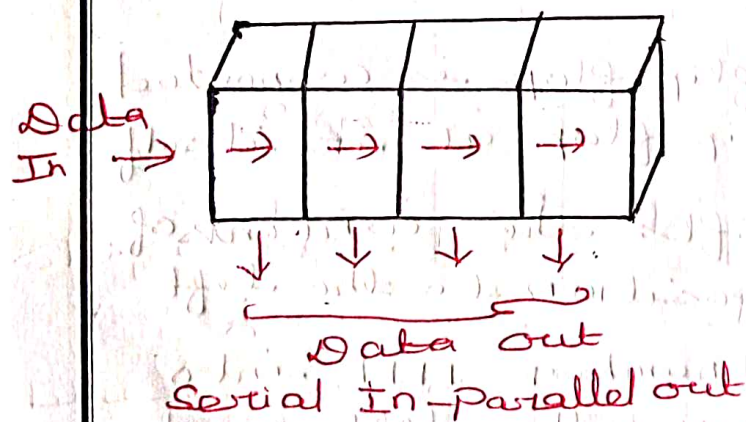
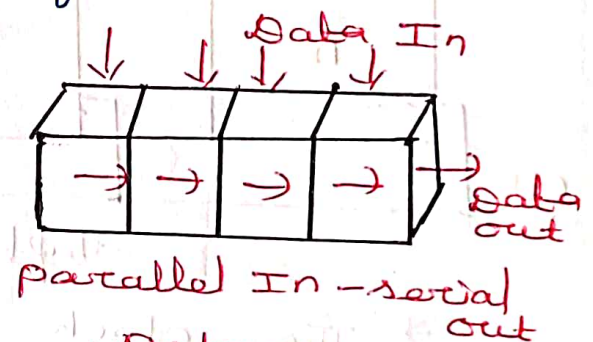
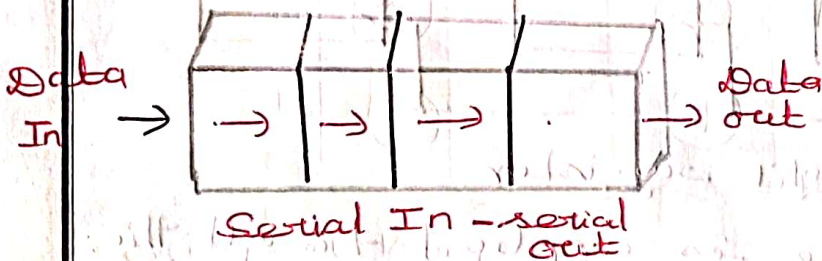
* There are four types of shift registers namely

↳ serial In serial out shift Register (SISO)

↳ Serial In parallel out shift Register (SIPO)

↳ parallel In serial out shift Register (PISO)

↳ ~~parallel~~ parallel In parallel out shift Register (PIPO)

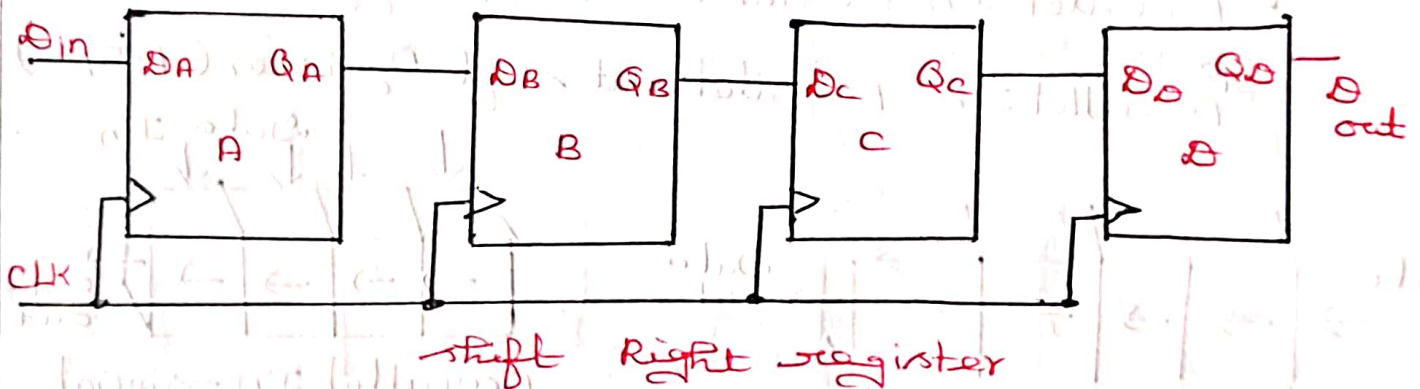
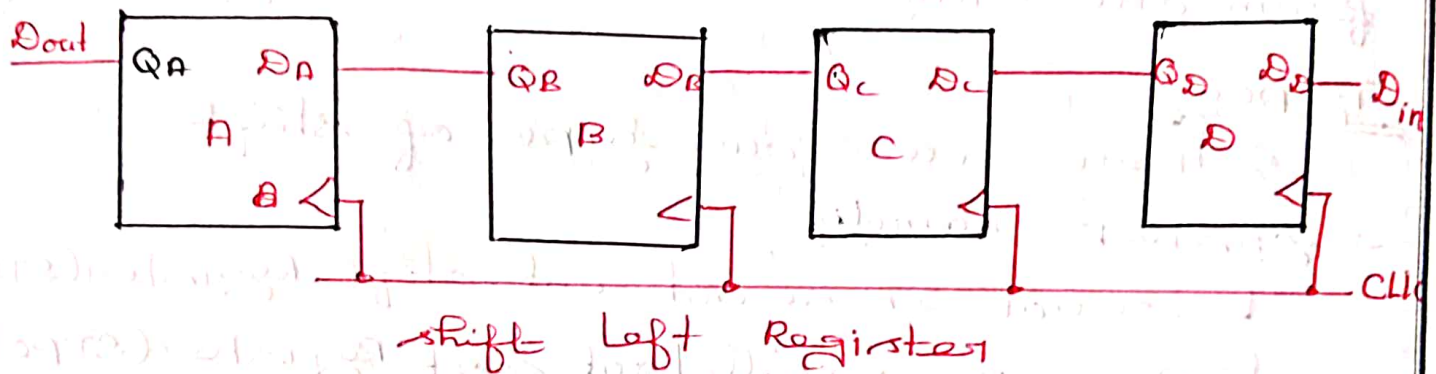


Serial In serial out shift Register (SISO)

* SISO Receives input data as single bit in serial and produces the stored information on its single output also in serial form

* Data may be shifted in right side as well as left side

* Data may be shifted left using shift left register and shifted right using shift right register.



* The clock pulse is applied to all the flip flop simultaneously

* output of each flip flop is connected to D input of the flip flop at its itself

* Each clock pulse shifts the contents of the register one bit position to the left

* four bit binary number 1111 into the Register, beginning with the left most bit

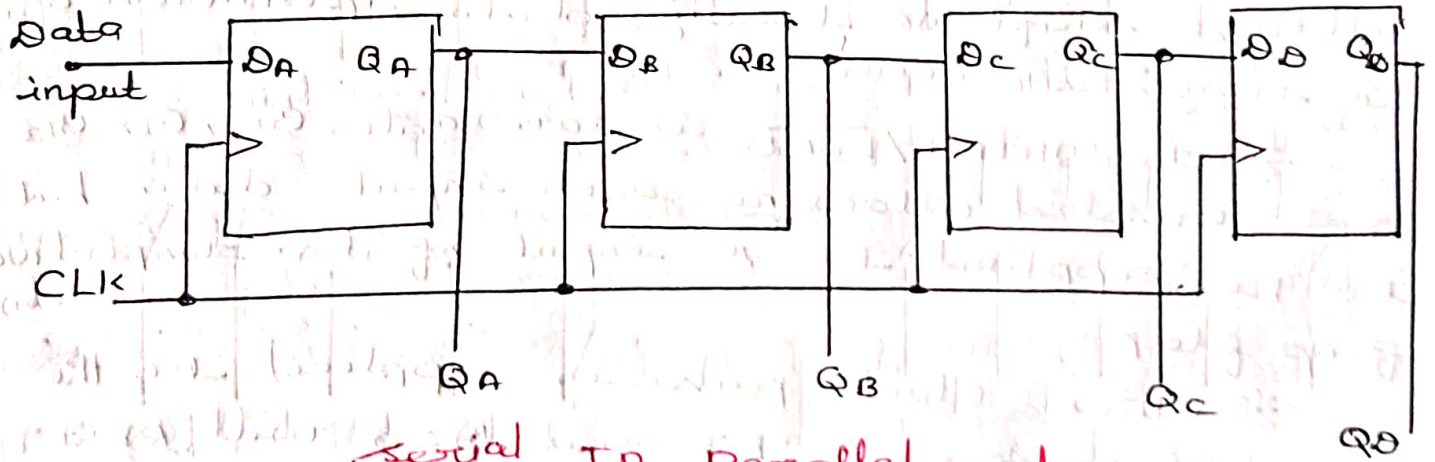
* Initially Register is cleared so $QAQBQCQD = 0000$

Q _A	Q _B	Q _C	Q _D	Serial I/P data	Clock pulse
0	0	0	0	1	0
0	0	0	1	1	1
0	0	1	1	1	2
0	1	1	1	1	3
1	1	1	1	1	4

Serial In parallel out shift Register :-

* In this case the data bits are entered into Register in the same manner as serially but output is taken in parallel

* once data are stored each bit appears on its respective output line and all bits are available simultaneously



Serial In parallel out shift Register

* The data in each stage after each clock pulse is shown in Table below

Shift Pulse	Serial Data Input	Parallel Inputs			
		QA	QB	QC	QD
0	1	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

Parallel In serial out shift Register:-

* Fast a register with parallel data input Register bits are entered simultaneously into their respective stages on parallel lines

* There are four input lines XA, XB, XC, XD for entering data in parallel into the Register

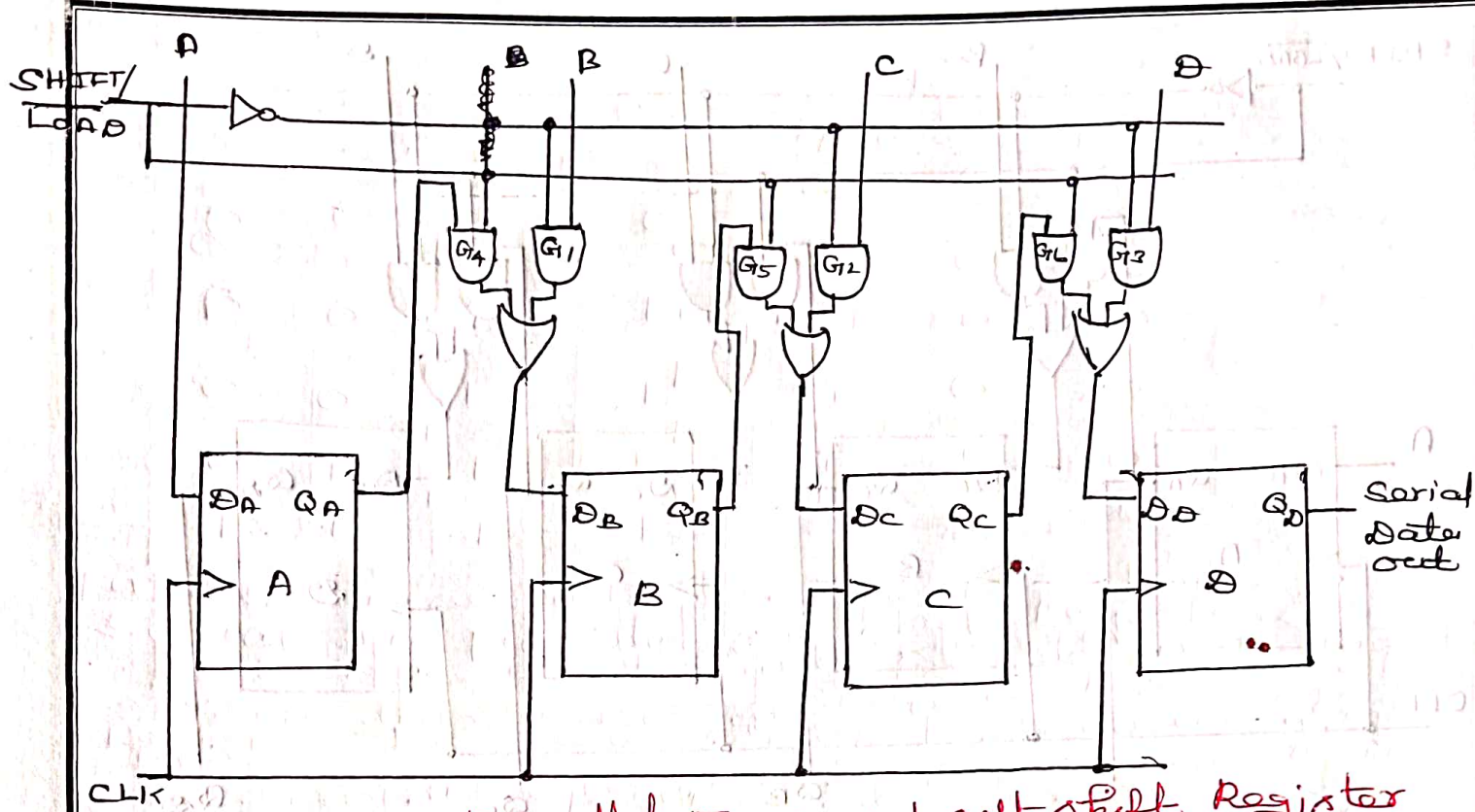
* $\overline{\text{SHIFT/LOAD}}$ is the control input which allows shift or loading data operation of the register

* When $\overline{\text{SHIFT/LOAD}}$ is low gates G1, G2, G3 are enabled allowing each input data bit to be applied to D input of its respective flip flop

* When clock pulse is applied to the flip flop $\overline{D}=1$ SET and $\overline{D}=0$ will RESET

* When $\overline{\text{SHIFT/LOAD}}$ is high, gates G1, G2, G3 are disabled and gates G4, G5, G6 are enabled

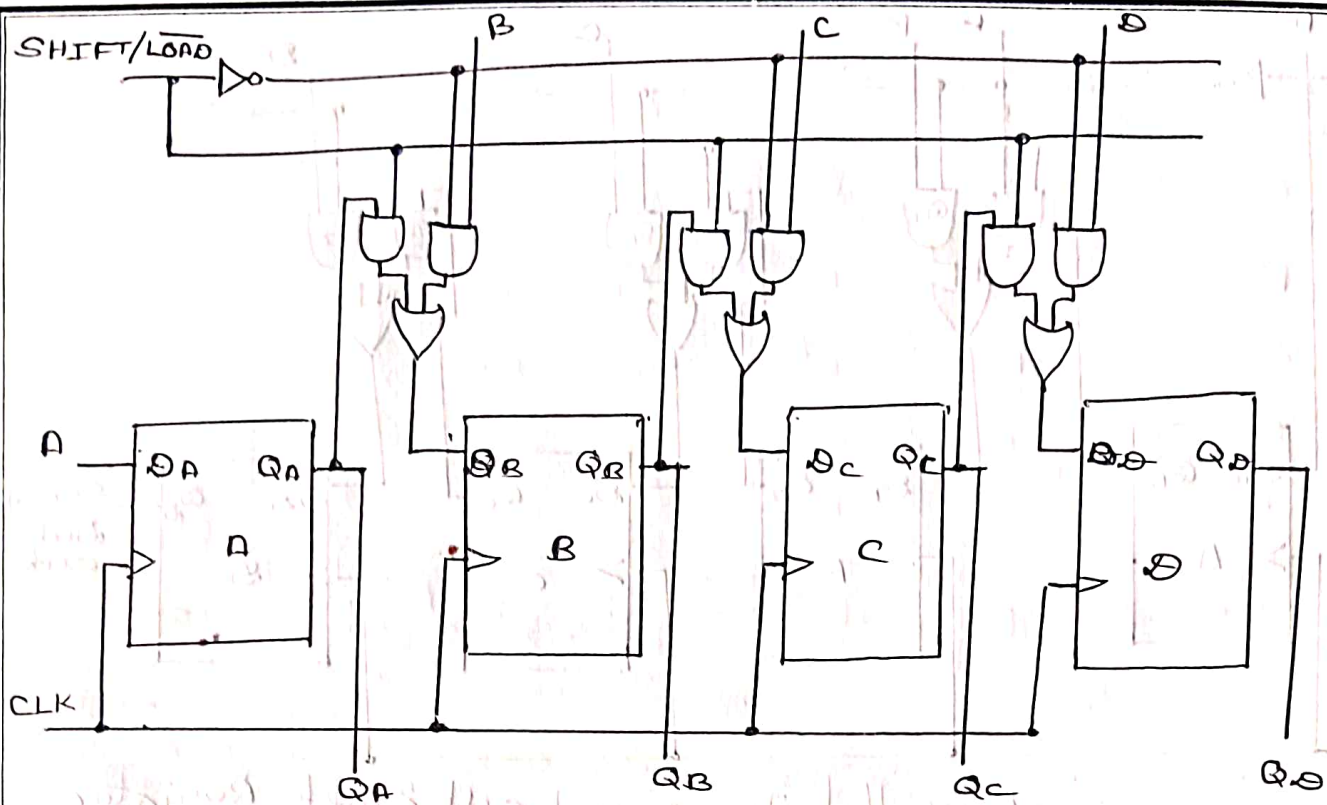
* This allows the data bit to shift Right from one stage to next



Parallel In serial out shift Register

Parallel IN parallel out shift Register:-

- * Data Inputs can be shifted either In or out of register in parallel form.
- * A, B, C and D are the four parallel data input lines and QA, QB, QC and QD are four parallel data output lines.
- * The SHIFT/LOAD is the control input that allows the four bit data to enter in parallel or shift the data serially.
- * When SHIFT/LOAD is low gates G11 through G13 are enabled, allowing the data at parallel input to the D input.
- * When clock pulse is applied the flip flop with D=1 will SET and D=0 will RESET.



Parallel In parallel out shift

Register:

Application of shift Register:

- ↳ serial and parallel converter
- ↳ parallel to serial converter
- ↳ Delay line
- ↳ shift Register counters
- ↳ sequence generator
- ↳ sequence detector
- ↳ serial adder/subtractor

Compare Moore and Mealy circuits

* In synchronous or clocked sequential circuits clocked flip flops are used as memory elements, which change their individual states in synchronism with the periodic clock signal

* The change in states of flip flops and change in state of entire circuit occur on the transition clock signal

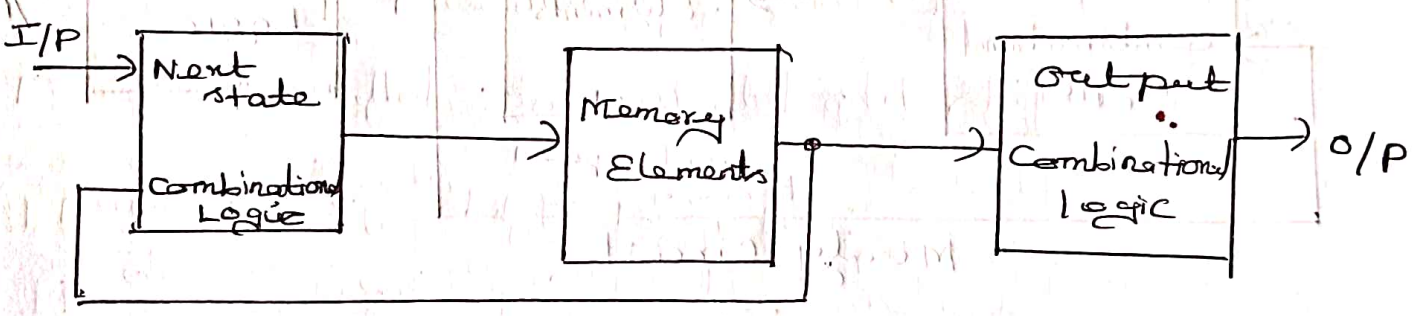
* It is classified to the two model

L) Moore circuit [o/p depends on the present state]

L) Mealy circuit [o/p depends on both present of the flip flop & input]

Moore circuit

* when the output of the sequential circuit depends only on the present state of the flip flop.



Moore Model

* It appears only after the clock pulse is applied

* It varies in synchronism with the clock input

Mealy circuit :-

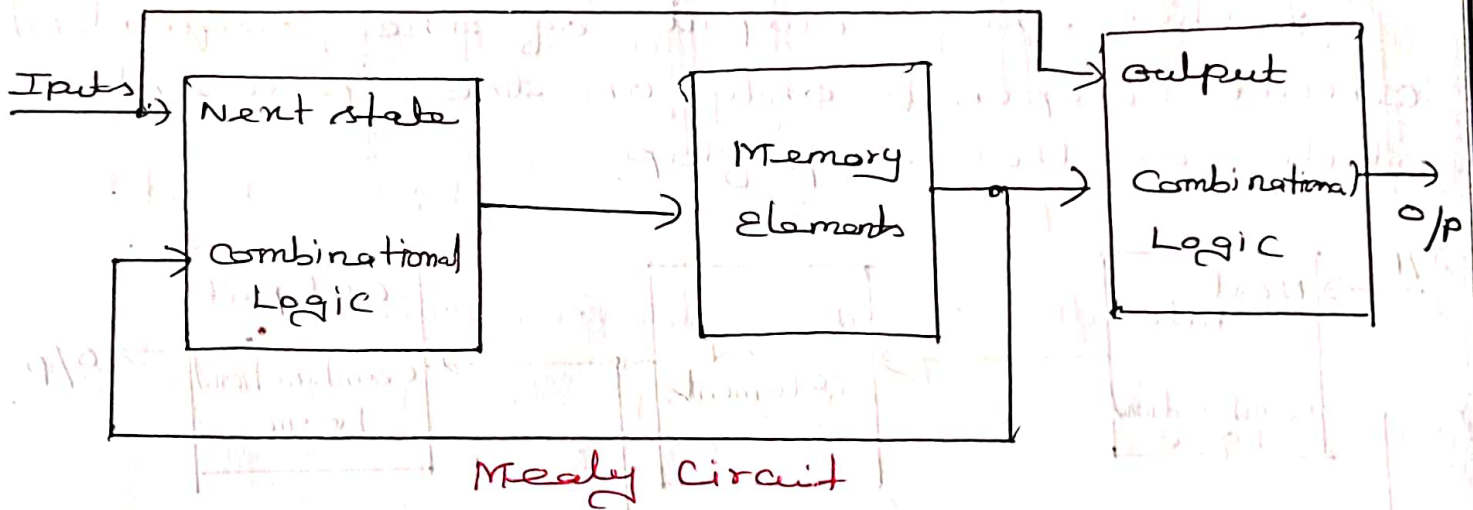
* when output of the sequential circuit depends on both present state of flip flops and inputs of sequential circuits is referred to Mealy circuit

* changes in the input within the clock pulses cannot affect the state of the flip flop.

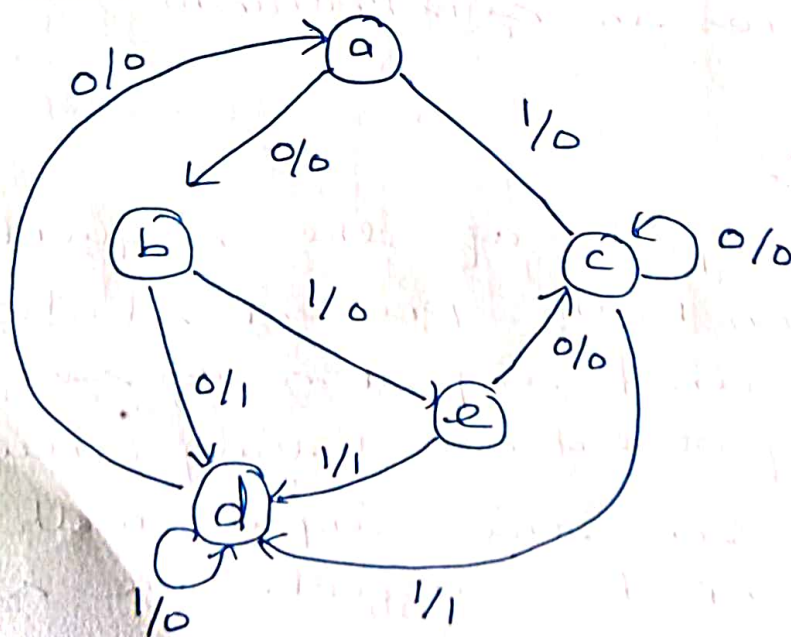
* However they can affect the output of the circuit

* Due to this input variations are not synchronized with the clock the derived output will also not be synchronized with the clock and we get false output

* This false output can be eliminated by allowing input change only at the active transition of the clock



To reduce the number of states in following state diagram



solution

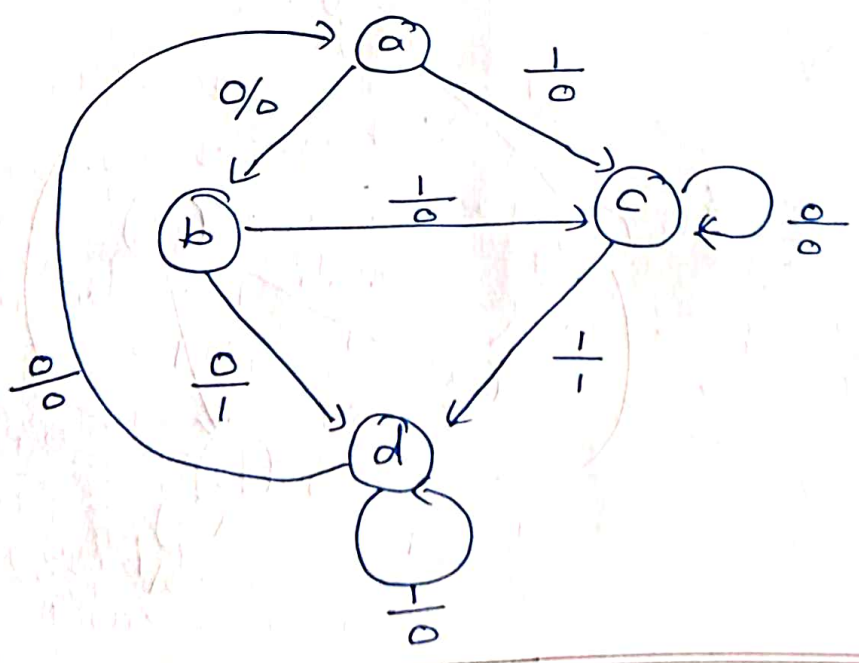
step 1: state table

Present state	Next state		output	
	x=0	x=1	x=0	x=1
a	b	c	0	0
b	d	e	1	0
<u>c</u>	<u>c</u>	<u>d</u>	<u>0</u>	<u>1</u>
d	a	d	0	0
<u>e</u>	<u>c</u>	<u>d</u>	<u>0</u>	<u>1</u>

step 2: Reduced state table e=c

Present state	Next state		output	
	x=0	x=1	x=0	x=1
a	b	c	0	0
b	d	c	1	0
c	c	d	0	1
d	a	d	0	0

step 3: Reduced state diagram



UNIT-4

An Asynchronous sequential circuit is described by the following excitation and output function

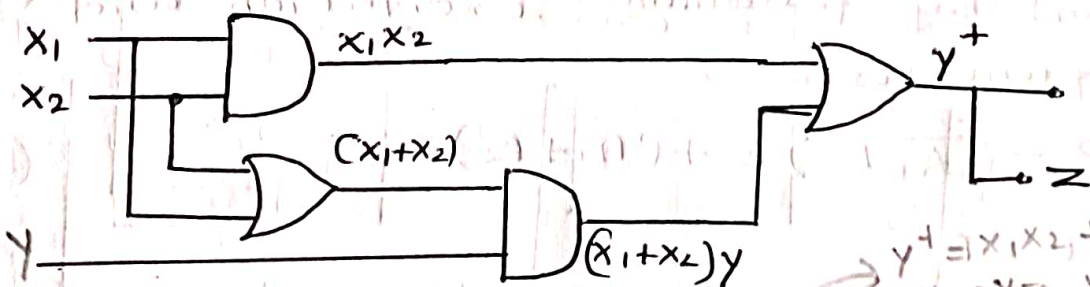
$$y^+ = x_1 x_2 + (x_1 + x_2) y$$

$$z = y^+$$

- a) draw the logic diagram of the circuit
- b) derive the transition table and output map

sol

a) logic diagram



State table

present state			next state	Stable/ Unstable	o/p
x ₁	x ₂	y	y ⁺	y ⁺ = y = stable y ⁺ ≠ y = unstable	
0	0	0	0	S	0
0	0	1	0	US	-
0	1	0	0	S	0
0	1	1	1	S	1
1	0	0	0	S	0
1	0	1	1	S	1
1	1	0	1	US	-
1	1	1	1	S	1

$y^+ = x_1 x_2 + (x_1 + x_2) y$
 $x_1 = 0, x_2 = 0, y = 0 \Rightarrow y^+ = 0$

2^2
 $2 \times 2 \times 2 = 8$

Transition Map

	$x_1 x_2$			
y	00	01	11	10
0	0	0	1	0
1	0	1	1	1
	y^+			

Output Map

	$x_1 x_2$			
y	00	01	11	10
0	0	0	-	0
1	-	1	1	1
	O/P			

An Asynchronous sequential circuit is described by following excitation and output function

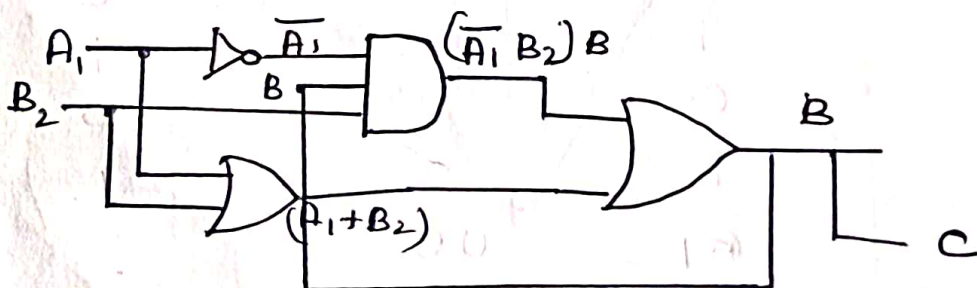
$$B = (\bar{A}_1 B_2) B + (A_1 + B_2)$$

$$C = B$$

- 1) draw the logic diagram
- 2) draw the transition table and output map

solution

1) Logical diagram



2) Transition table

present state			Next state	Stable/ unstable $y = y^+ \Rightarrow$ stable $y \neq y^+ \Rightarrow$ unstable	O/P
A ₁	B ₂	B	B		
0	0	0	0	S	0
0	0	1	0	US	1
0	1	0	1	US	1
0	1	1	1	S	1
1	0	0	0	S	0
1	0	1	0	US	1
1	1	0	1	US	1
1	1	1	1	S	1

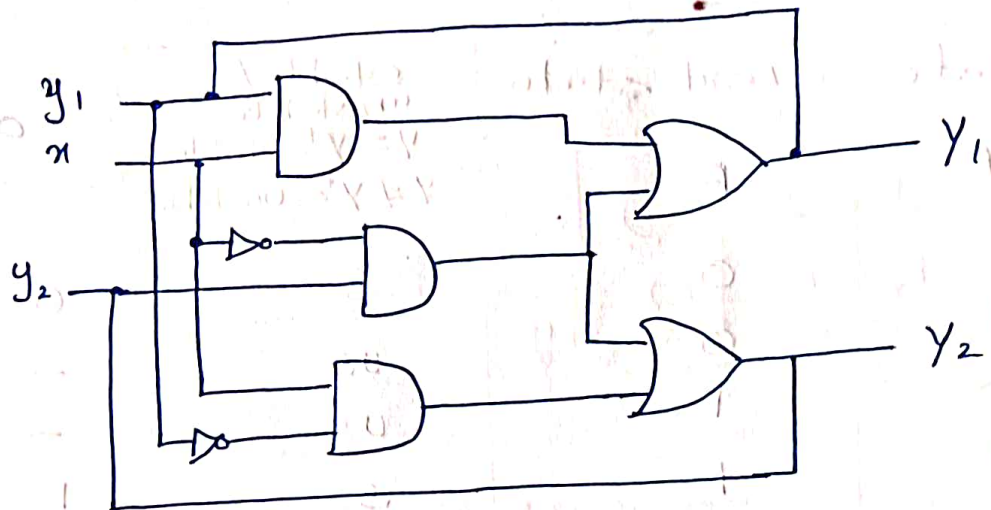
Transition table

		A ₁ B ₂	00	01	11	10
B	0	0	0	1	1	
	1	0	0	1	1	

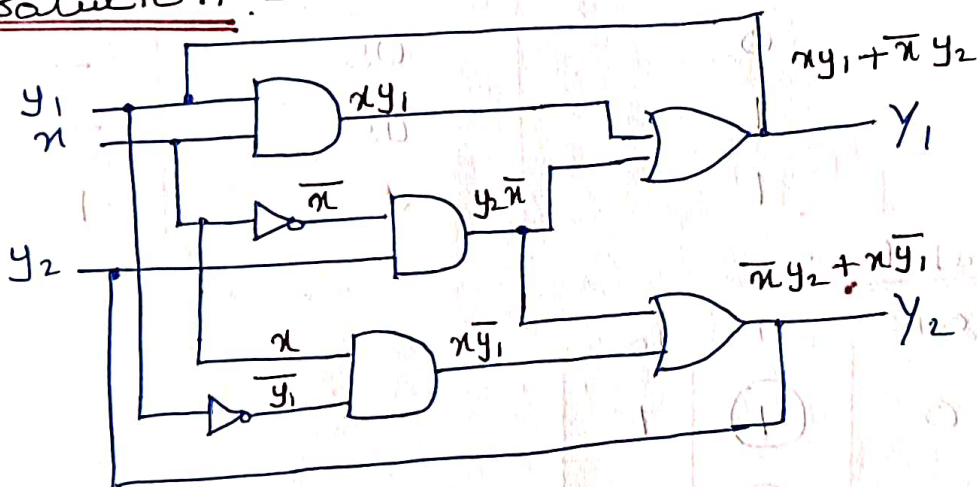
Output Map

		A ₁ B ₂	00	01	11	10
B	0	0	-	1	-	
	1	0	-	1	1	

Consider the following asynchronous sequential circuit and draw maps transition table and state table



solution :-



$$Y_1 = \pi y_1 + \bar{\pi} y_2$$

$$Y_2 = \bar{\pi} y_2 + \pi \bar{y}_1$$

(Logical equation)

$$Y_1 = \pi y_1 + \bar{\pi} y_2 \quad \left(\begin{array}{l} y_1=0, y_2=0, \pi=0 \\ y_1=0, y_2=1, \pi=0 \\ y_1=1, y_2=0, \pi=1 \\ y_1=1, y_2=1, \pi=1 \end{array} \right)$$

$$Y_1 = 0 \cdot 0 + 1 \cdot 0 = 0 \quad \left| \quad Y_2 = \frac{\pi y_2 + \pi \bar{y}_1}{1 \cdot 0 + 0 \cdot 0} = 0$$

state table

present state			Next state		stable / unstable
y ₁	y ₂	π	Y ₁	Y ₂	
0	0	0	0	0	S
0	0	1	0	1	US
0	1	0	1	1	US
0	1	1	0	1	S
1	0	0	0	0	US
1	0	1	1	0	S
1	1	0	1	1	S
1	1	1	1	0	US

Map for y_1

$y_2 \backslash y_1$	00	01	11	10
0	0	0	0	1
1	0	1	1	1

Map for y_2

$y_2 \backslash y_1$	00	01	11	10
0	0	1	1	1
1	0	0	0	1

Transition map

$y_2 \backslash y_1$	00	01	11	10
0	00	01	01	11
1	00	10	10	11

○ - stable

An asynchronous sequential circuit has two internal states and one output function describing the circuit are

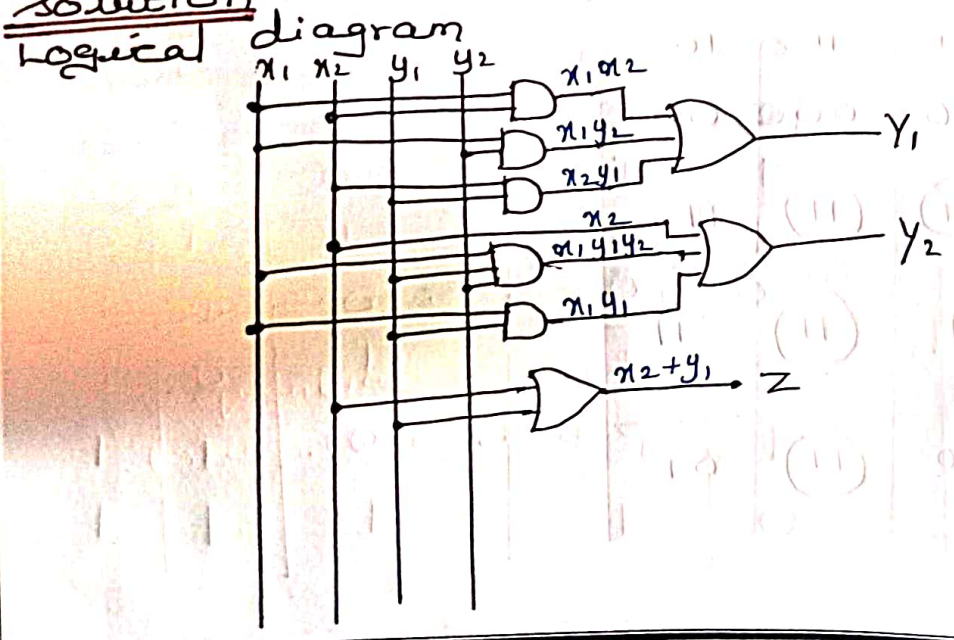
$$y_1 = x_1 x_2 + x_1 y_2 + x_2 y_1$$

$$y_2 = x_2 + x_1 y_1 y_2 + x_1 y_1$$

$$z = x_2 + y_1$$

- a) Draw the logical diagram of the circuit
- b) derive the transition table and output map

solution



state table

Present state				Next state		Stable/ Unstable	output Z
x_1	x_2	y_1	y_2	Y_1	Y_2	S	0
0	0	0	0	0	0	US	-
0	0	0	1	0	0	US	-
0	0	1	0	0	0	US	-
0	0	1	1	0	0	US	-
0	1	0	0	0	1	US	-
0	1	0	1	0	1	S	1
0	1	1	0	1	1	US	-
0	1	1	1	1	1	S	-
1	0	0	0	0	0	S	0
1	0	0	1	1	0	US	-
1	0	1	0	0	1	US	-
1	0	1	1	1	1	S	1
1	1	0	0	1	1	US	-
1	1	0	1	1	1	US	-
1	1	1	0	1	1	US	-
1	1	1	1	1	1	S	1

Transition Map

$x_1 x_2$	$y_1 y_2$	00	01	11	10
00		00	00	00	00
01		01	01	11	11
11		11	11	11	11
10		00	10	11	01

output Map

x_1, x_2 \ y_1, y_2	00	01	11	10
00	0	-	-	-
01	-	1	1	-
11	-	-	1	-
10	0	-	1	-

Hazards of sequential circuit:-

* A hazard of potential or actual malfunction of a logic network during the transition between two input states as a result of a single variable change

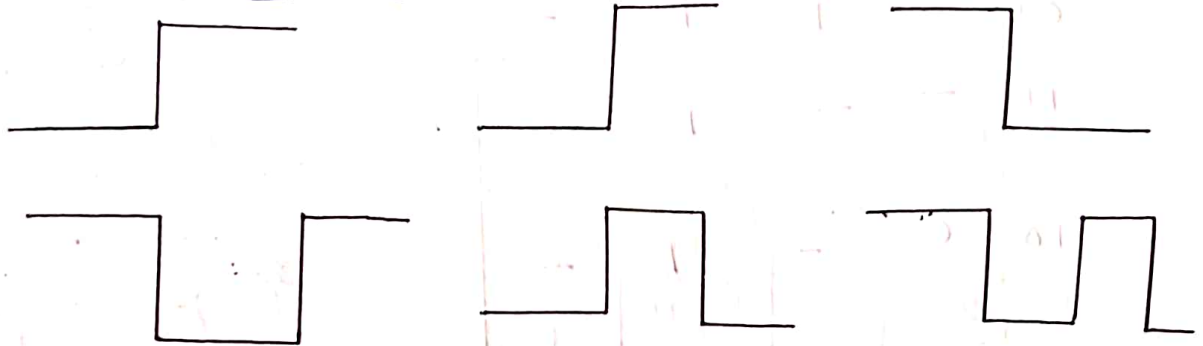
* Hazards occur in the combinational circuit where they may cause a temporary false output value!

* when such combinational circuits are used in asynchronous sequential circuits, they may result in a transition to form a wrong stable state

* There are two types of hazards

- ↳ static hazards
- ↳ dynamic hazards
- ↳ essential hazards

* similar to static and dynamic hazards caused by delay in combinational circuit essential hazards occurs in sequential circuits



Static 1 hazard

Static 0 hazard

Dynamic hazard

static hazards:-

* static hazard is a condition which result in single ~~momentary~~ momentary incorrect output

* due to change in single input variable when the output is expected to remain in the same state

* Two types of static hazards

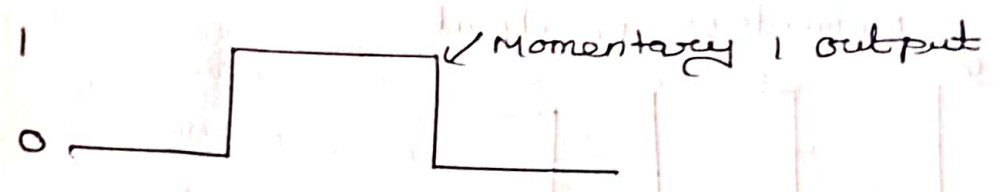
↳ static 0 hazard

↳ static 1 hazard

static 0 Hazards:-

* static 0 hazard the output is to remain the value 0 and momentary 1 output is possible during the transition between the two input states

* Then the hazard is called static 0 hazard



static 1 hazards:-

* when the output is remain at the value 1 and a momentary 0 output is possible during the transition between the two input states then the hazard is called a static 1 hazard

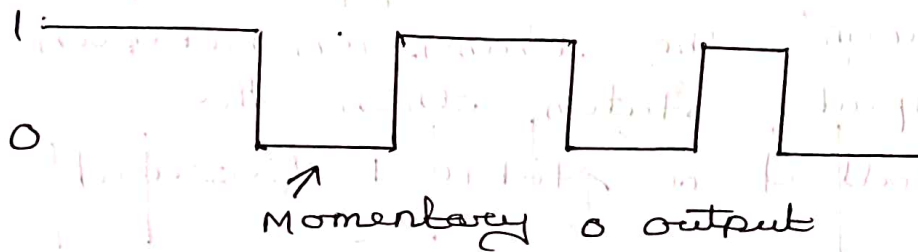
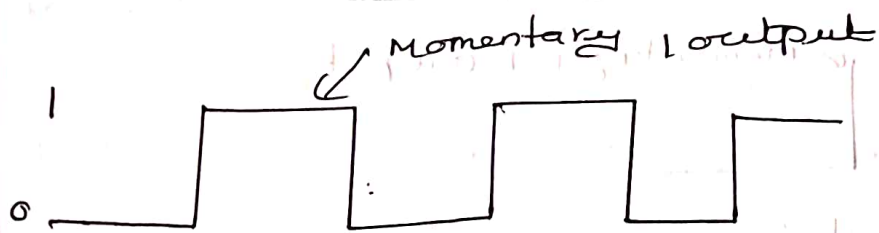


Dynamic hazard

* Dynamic hazards occur when the output of a network is to change between its two logic states but a momentary false output signal occurs during the transient behavior

* A dynamic 1 hazard is defined as a transient change occurring three or more times at an output terminal of a logic network

* when the output is supposed to change only once during a transition between two inputs states differing in the value of one variable.



Dynamic Hazards

Essential Hazards:-

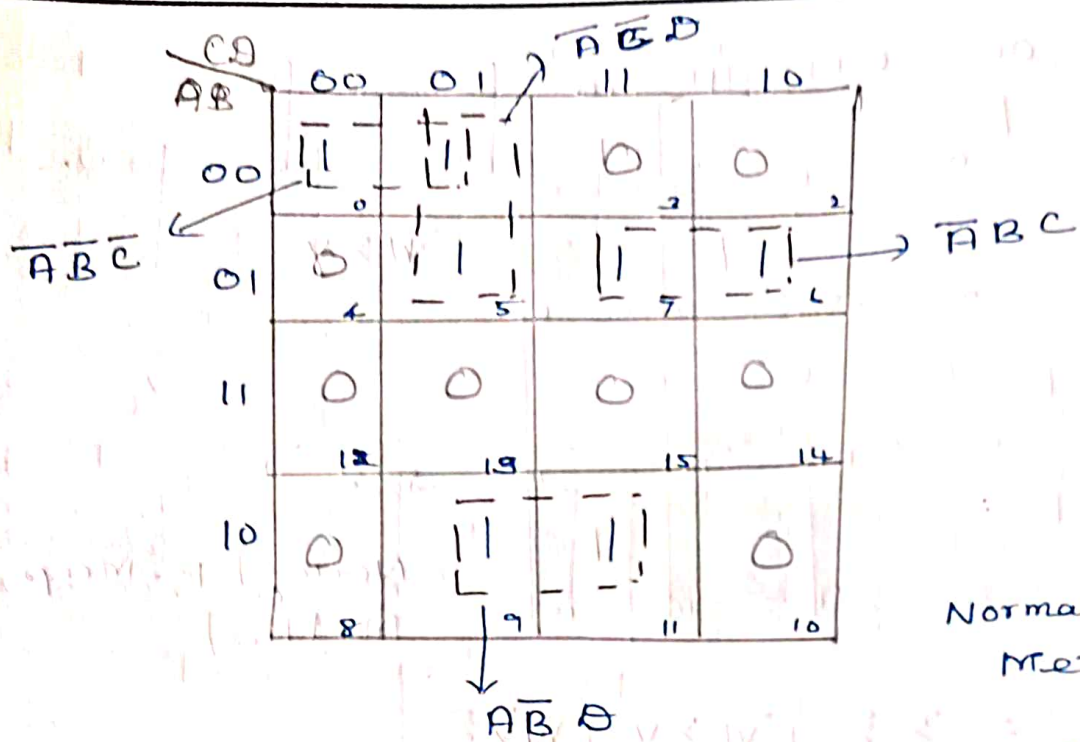
* Essential Hazards is type of that exists only in asynchronous sequential circuits with a two or more feedback

* It is caused by unequal delays along two or more paths that originate from the same input

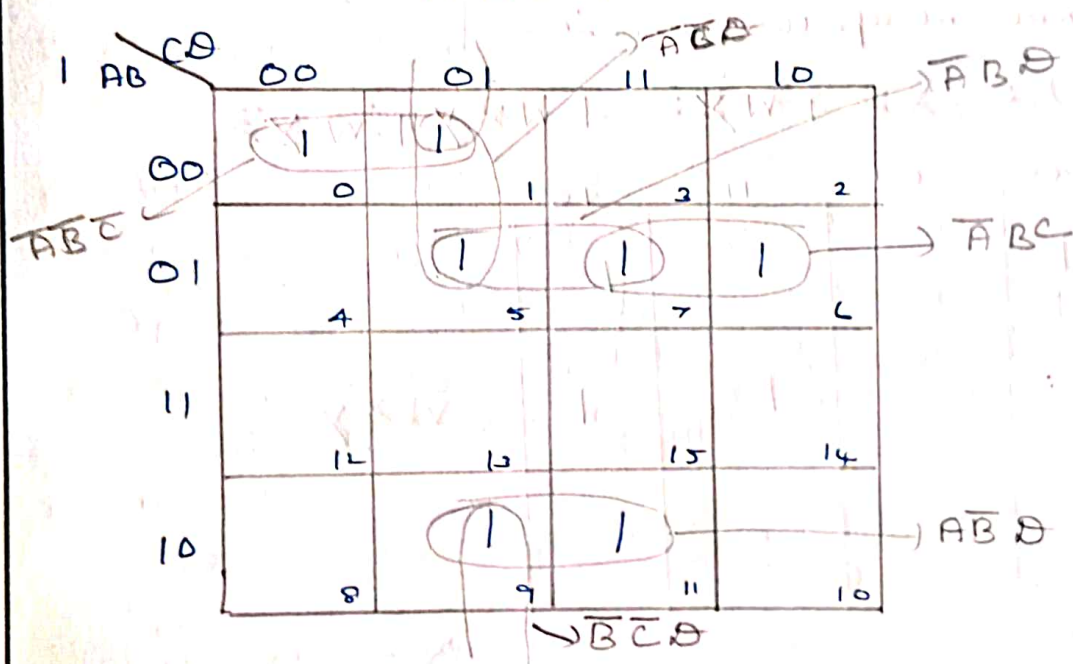
* An excessive delay through an inverter circuits in comparison to the delay associated with the feedback path may cause essential hazard

Give the hazard free realization for the following Boolean function

$$f(A, B, C, D) = \sum m (0, 1, 5, 6, 7, 9, 11)$$



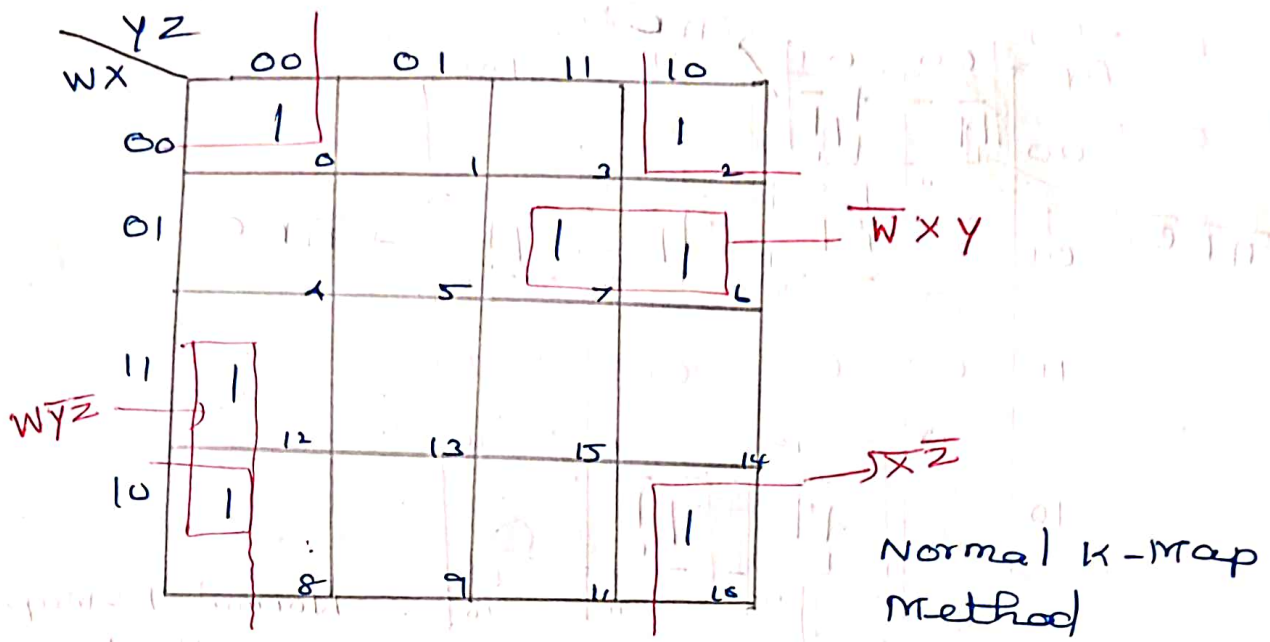
$$F = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{C}D + \bar{A}BC + \bar{A}\bar{B}D$$



$$F = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}D + \bar{A}BC + \bar{A}\bar{B}D + \bar{A}\bar{C}D + \bar{B}\bar{C}D$$

Give the hazard free realization for the following Boolean function

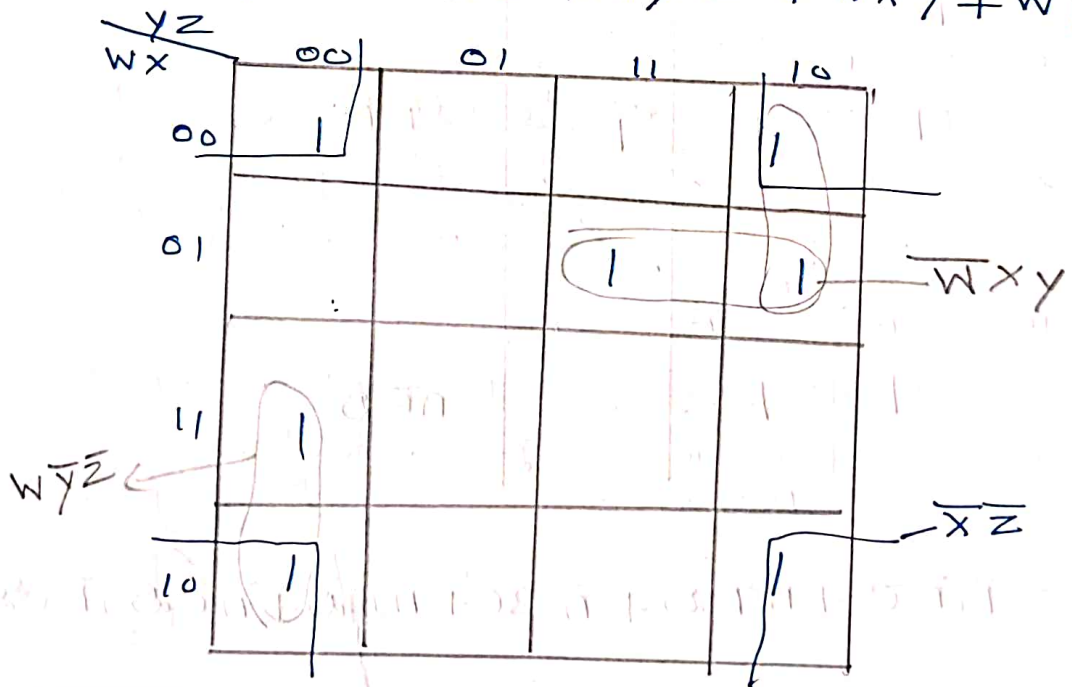
$$F(W, X, Y, Z) = \sum m(0, 2, 6, 7, 8, 10, 12)$$



$$F(WX, YZ) = \overline{X}Z + \overline{W}XY + W\overline{Y}\overline{Z}$$

Hazard free expression is

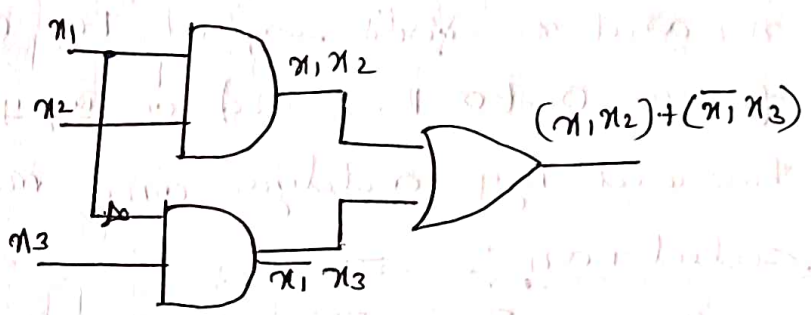
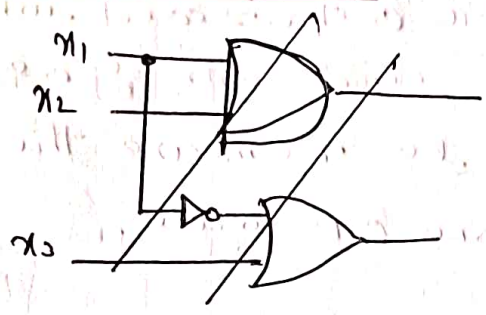
$$F(WX, YZ) = \overline{X}Z + \overline{W}X\overline{Y}\overline{Z} + \overline{W}XY + \overline{W}Y\overline{Z}$$



Implement the following logic and analyse for the presence of any hazard.
 $F = \pi_1 \pi_2 + \pi_1 \pi_3$. If hazard is present briefly explain the type of hazard and design a hazard free circuit.

Solution:-

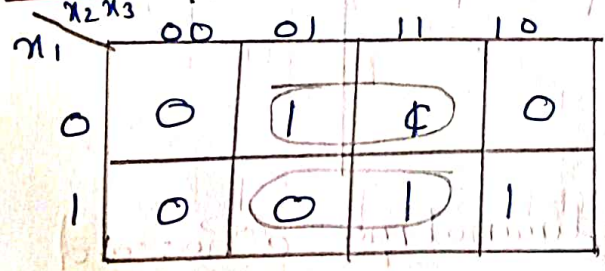
Logical diagram:-



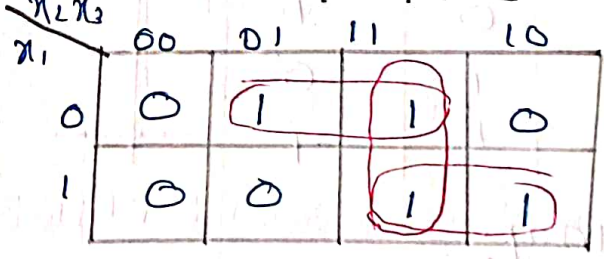
state table:-

present state			output $F = (x_1x_2) + (x_1x_3)$
x_1	x_2	x_3	
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

K-Map:

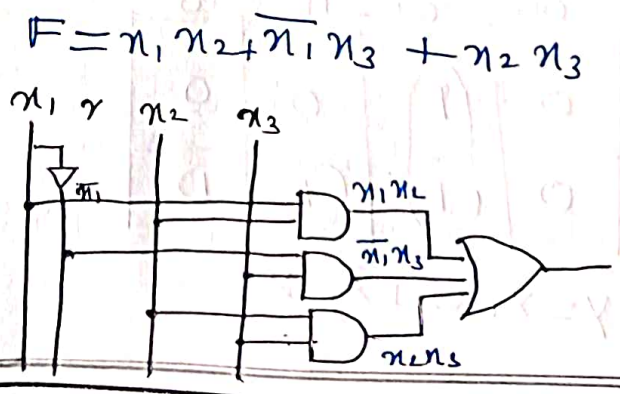


Hazard-free



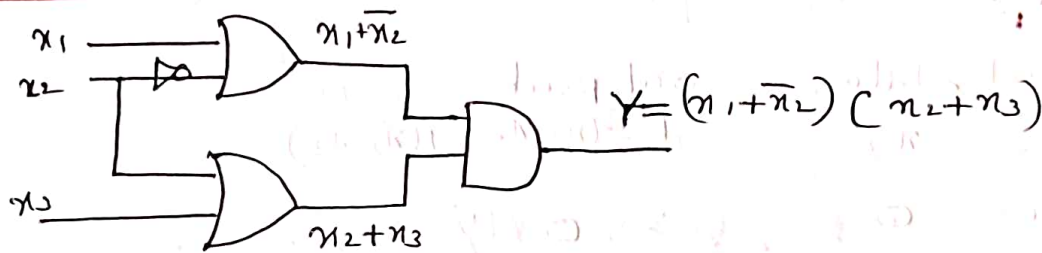
$F = x_1x_2 + \bar{x}_1x_3$

Hazard free circuit:-



Draw the logic diagram for the product of sum expression given by $Y = (x_1 + \bar{x}_2)(x_2 + x_3)$
 show that there is a static 0 hazard when x_1 and x_3 are equal to 0 and x_2 goes from 0 to 1. Find a way to remove the hazard by adding one more OR gate.

solution :-



state table

present state			output
x_1	x_2	x_3	$Y = (x_1 + \bar{x}_2)(x_2 + x_3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

K-map

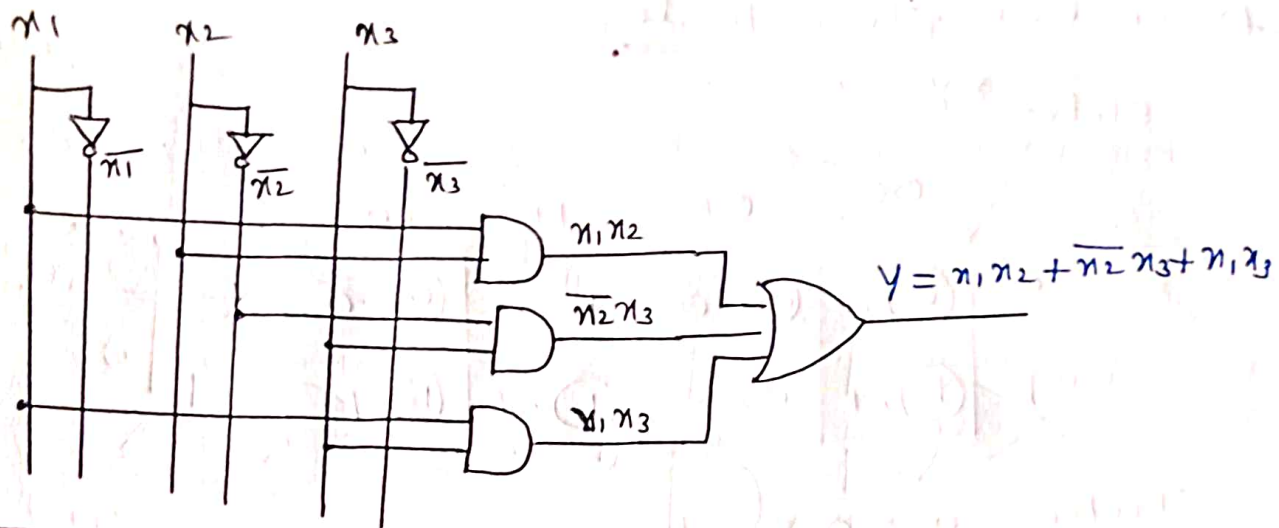
$x_2 x_3$	00	01	11	10
x_1				
0	0	1	0	0
1	0	1	1	1

$$Y = x_1 x_2 + \bar{x}_2 x_3$$

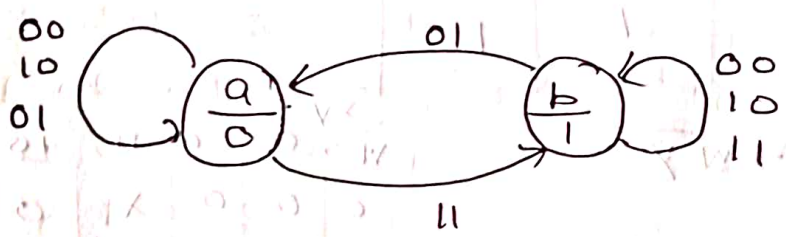
Eliminating a hazard

$x_2 x_3$	00	01	11	10
x_1				
0	0	1	0	0
1	0	1	1	1

$$Y = x_1 x_2 + \bar{x}_2 x_3 + x_1 x_3$$



Design a fundamental mode asynchronous sequential circuit with the following state diagram



solution:-

There are two states a and b then one output

input $\rightarrow xy$

output Z

state table:-

present state	Next state				output Z
	00	01	11	10	
a	a	a	b	a	0
b	b	a	b	b	1

Primitive flow table:

States	xy			
	00	01	11	10
a	(a, 0)	(a, 0)	b, -	(a, 0)
b	(b, 1)	a, -	(b, 1)	(b, 1)

Take $a = 0$, $b = 1$

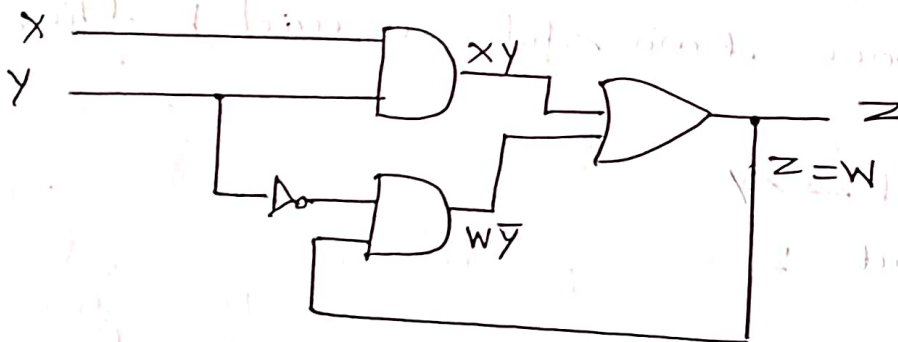
	xy	00	01	11	10
w	0	0	0	1	0
	1	1	0	1	1

$$Z = xy + w\bar{y}$$

	xy	00	01	11	10
w	0	0	0	X	0
	1	1	X	1	1

$$Z = W$$

Logic diagram



Design an asynchronous sequential circuit with two input x & y and with one output z . whenever y is 1 input x is transferred to z . when y is 0. the output does not change for. an change in x

Given:-

$xy \rightarrow$ input

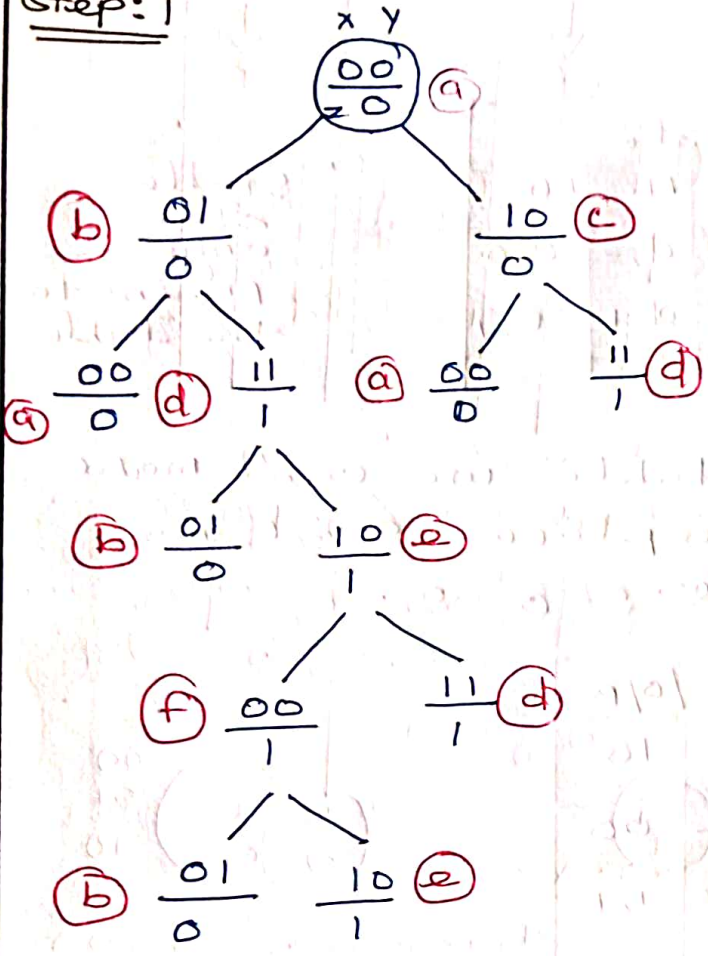
$z \rightarrow$ output

$y=1 \rightarrow z=x$

$y=0 \rightarrow z = \text{previous o/p}$

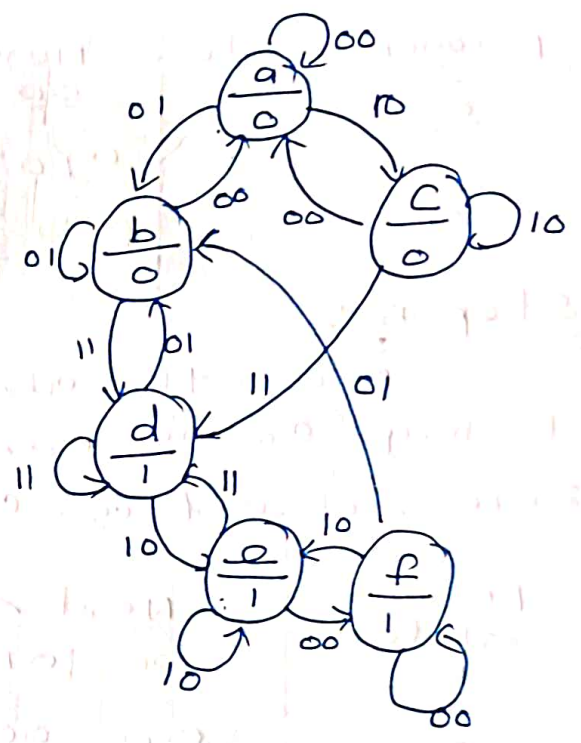
Flow table :-

Step: 1



Step 2

state diagram



Step 3: primitive flow table :-

present state	Next state			
	00	01	11	10
a	(a,0)	b,-	--	c,-
b	a,-	(b,0)	d,-	--
c	a,-	--	d,-	(c,0)
d	--	b,-	(d,1)	e,-
e	f,-	--	d,-	(e,1)
f	(f,1)	b,-	--	e,-

Step 4

a, b, c → S₀

d, e, f → S₁

Present state	Next state			
	00	01	11	10
S ₀	a, 0	b, 0	d, -	c, 0
S ₁	f, 1	b, -	d, 1	e, 1

State state only present to the table

Step 5 :-

From the above table we can't make K-map so we can replace a, b, c, as zero, d, e, f as one & S₀=0, S₁=1

Present state F	Next state / o/p			
	00	01	11	10
0	00	00	1, -	00
1	11	0, -	1, 1	1, 1

Step 6: (K-map)

K-Map Next state

F \ xy	00	01	11	10
0	0	0	1	0
1	1	0	1	1

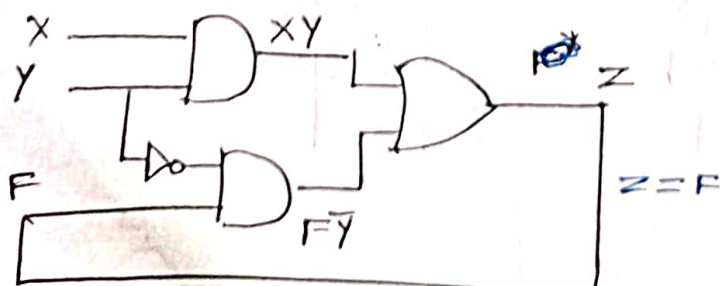
$$Z = XY + F\bar{Y}$$

K-Map for o/p

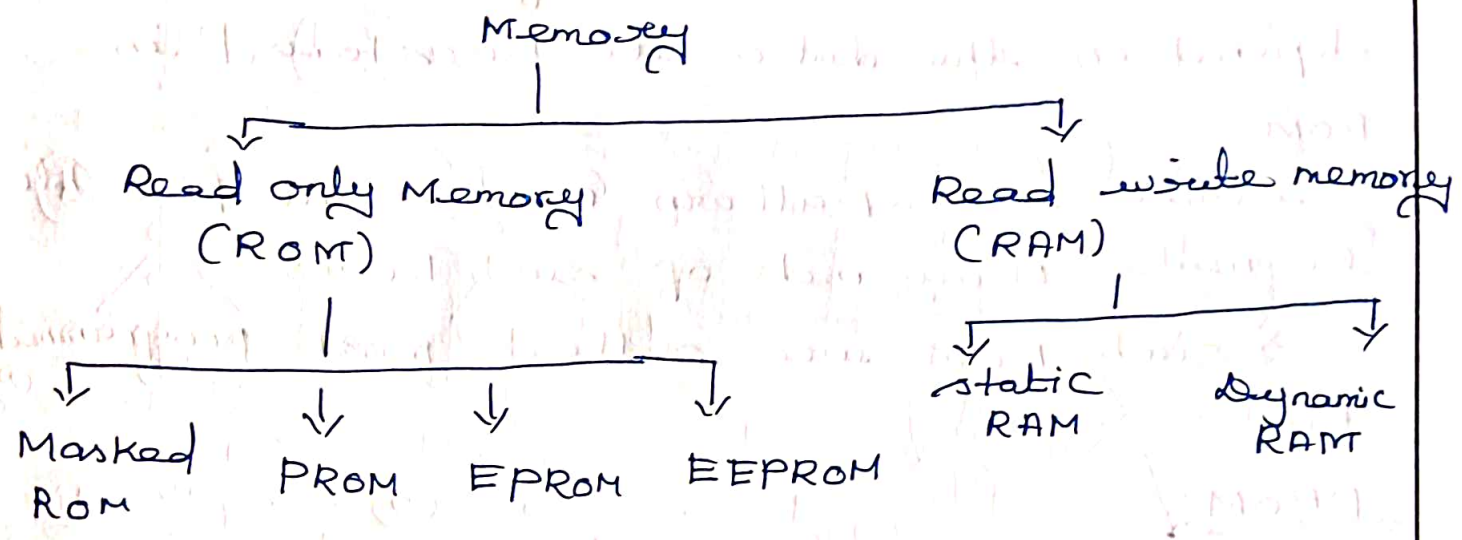
F \ xy	00	01	11	10
0	0	0	X	0
1	1	X	1	1

$$Z = F$$

Step 7 :- (Logic diagram)



Types of Memory :-



Read only Memory (ROM)

- * We cannot read from or write into memory
- * It is read only memory we cannot write data in this memory
- * It is non volatile memory that can hold data even if power is turned off

Read write memory (RAM)

- * Random access memory is a volatile memory it cannot hold data when the power is turned off
- * we can read from or write into the RAM so it is called read/write memory

Masked ROM :-

- * In integrated circuit a thin metallized layer connects the gates of

some transistors to the row select lines
the gate connections of MOS transistors
depend on the data to be stored in
ROM

* once the pattern mask is it possible
to make thousands of such ROMs

* such ROM are called mask programmed
ROM

PROM:-

* It is programmable read only
Memory

* PROM are programmed by used to
provide the programming facility each
address select and data line intersection
has its own fused MOSFET transistor

* The PROM are one time programmable
once programmed the information stored
is permanent

EPROM:-

* Erasable programmable read only
memory EPROM used MAS circuits.

* The store 1 and zero as a pocket of
charge in a buried layer of the IC chip

* EPROM can be programmed by the
user with a special EPROM programmer
we can erase the stored data

Static RAM:-

* Memories that consists of circuit capable of retaining their state as long as power is applied are known as static memory

* These are random access memory (RAM) and hence combine called static RAM memories

Dynamic RAM:-

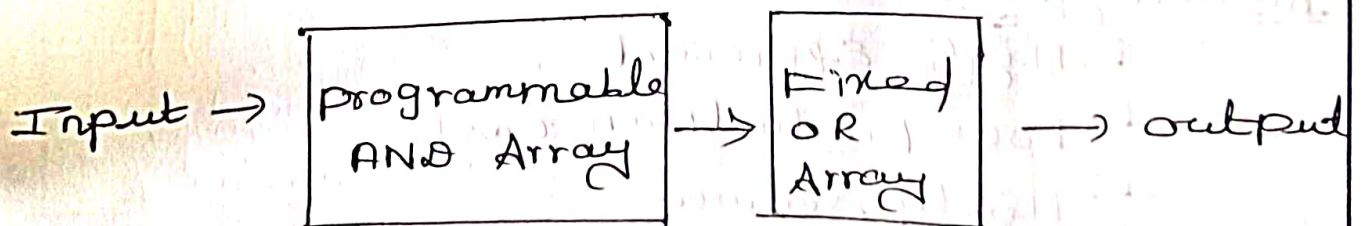
* Dynamic RAM stores the data charge on capacitor dynamic RAM contains more memory cell as compared to static RAM per unit area

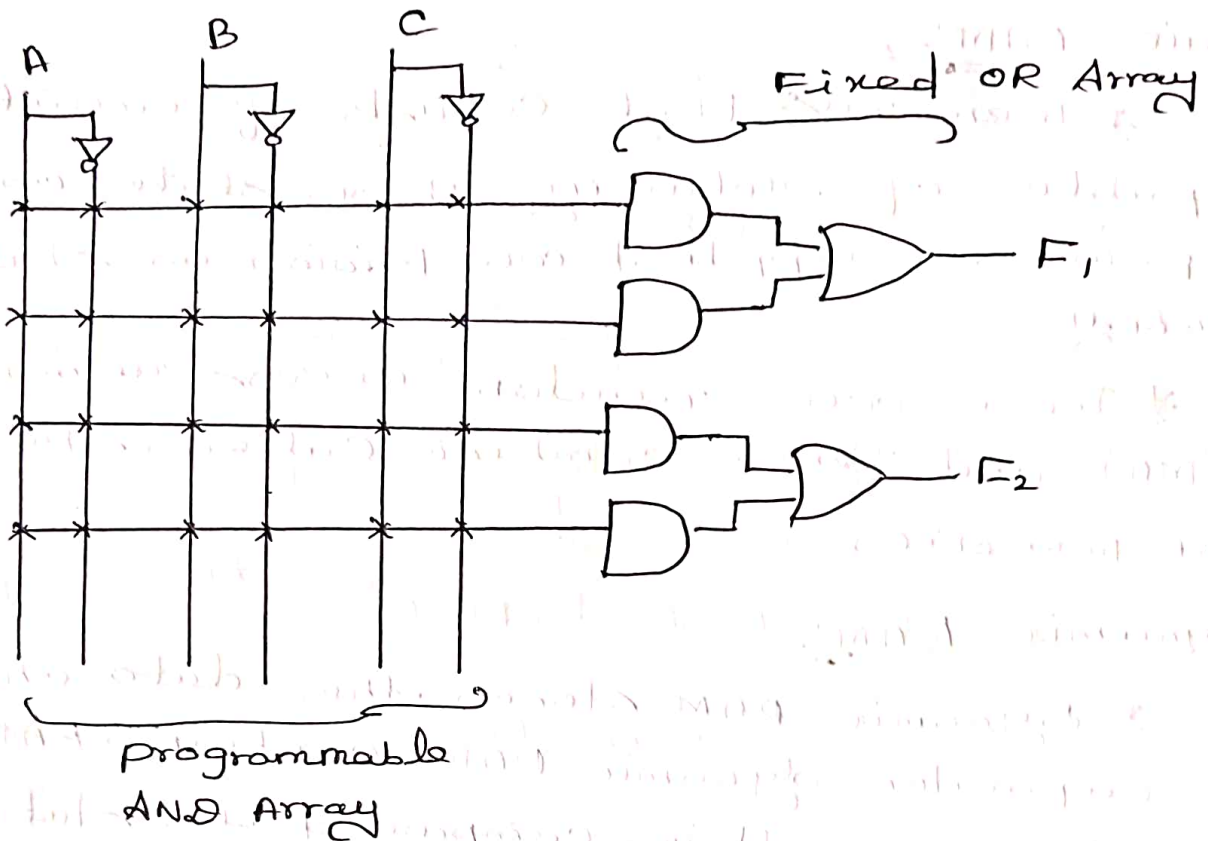
Programmable Array Logic (PAL)

* PLA is a device with a programmable AND array and programmable OR array

* PAL programmable logic device is fixed OR array and a programmable AND array

* The PAL is easier to program, but it is not flexible as the PLA.





* The input to the AND gates are programmable while the inputs to the OR gates are hard wired.

* This means every AND gates can be programmed to generate any ~~data~~ desired product of the input variables and their complements.

* Each OR gate is hard-wired to only selected AND gates outputs.

* In designing with PAL, the Boolean functions must be simplified to fit into each section.

Advantages :-

- ↳ High Efficient
- ↳ Low production cost
- ↳ Highly secure

- ↳ High Reliability
- ↳ Low power required for working
- ↳ More flexible to design

Design using PAL the following boolean functions

$$W(A, B, C, D) = \sum(2, 12, 13)$$

$$X(A, B, C, D) = \sum(7, 8, 9, 10, 11, 12, 13, 14, 15)$$

$$Y(A, B, C, D) = \sum(0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15)$$

$$Z(A, B, C, D) = \sum(1, 2, 8, 12, 13)$$

Solution:-

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

$$W = ABC + \bar{A}BC\bar{D}$$

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

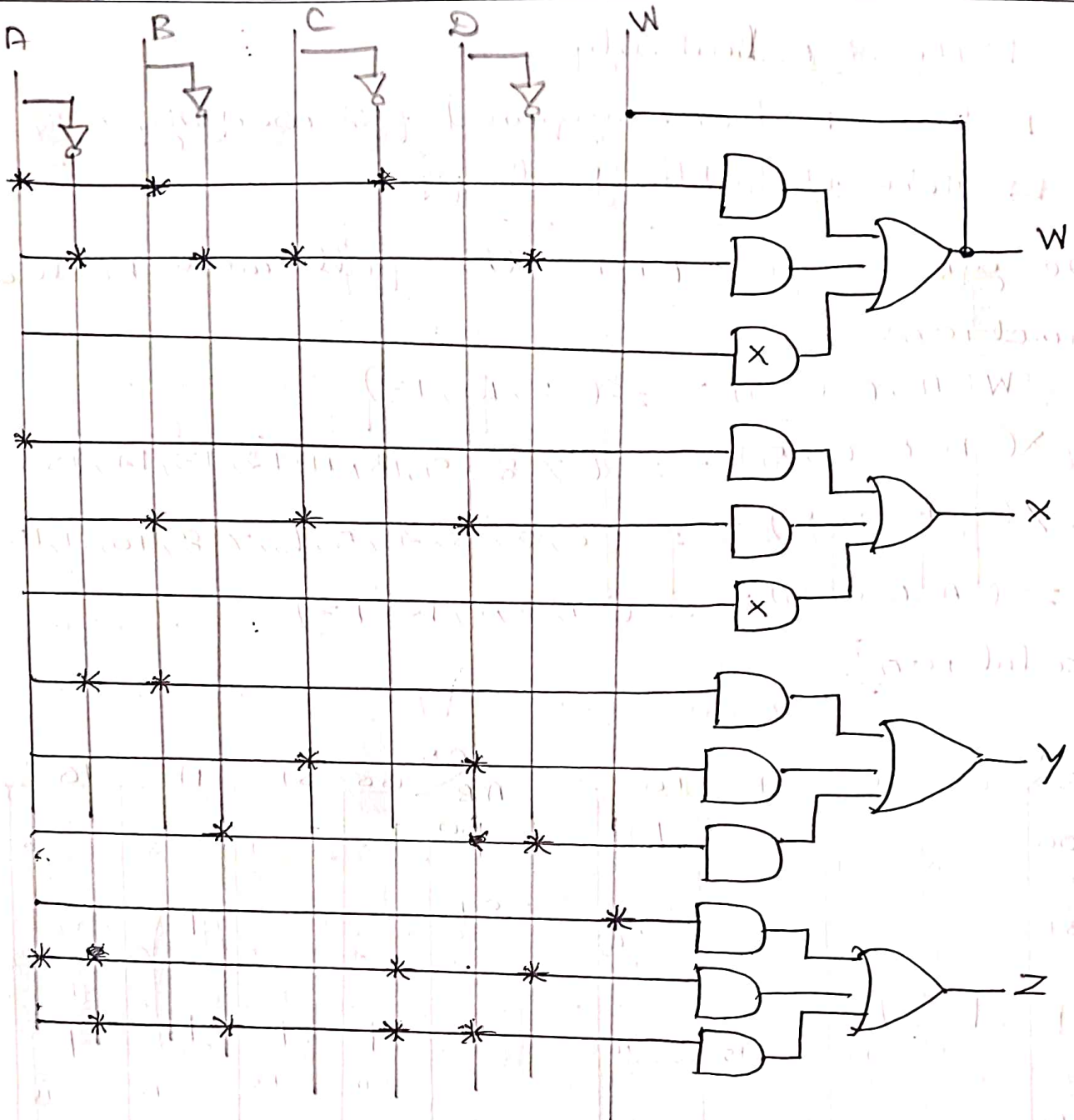
$$X = A + BC\bar{D}$$

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

$$Y = \bar{B}\bar{D} + \bar{A}B + C\bar{D}$$

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

$$Z = \underbrace{ABC + \bar{A}BC\bar{D}}_W + A\bar{C}\bar{D} + ABC\bar{D}$$



PAL Diagram

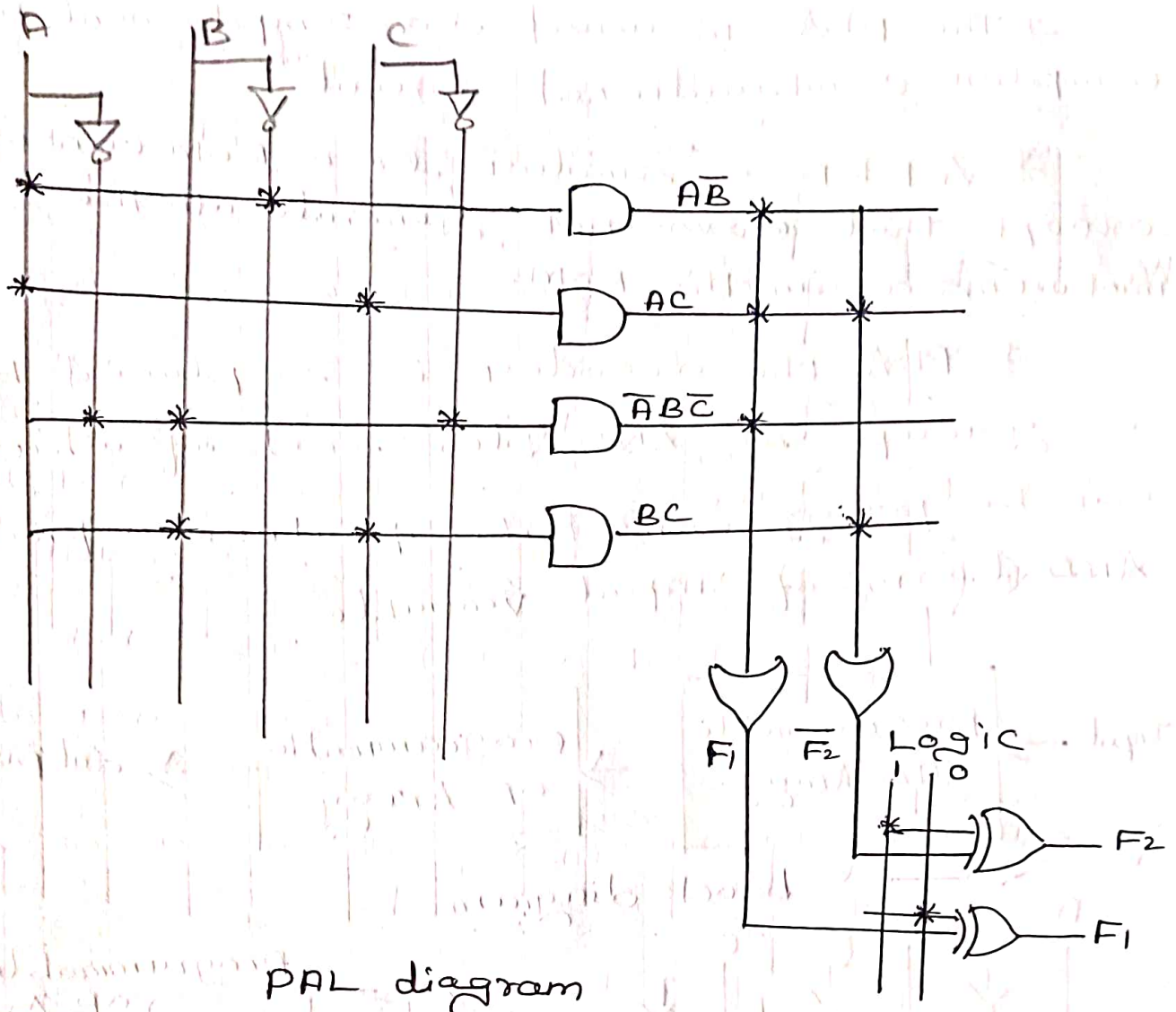
Implement the following function using 3 input 4 product term and 2 output PLA

$$F_1 = A\bar{B} + AC + \bar{A}BC$$

$$F_2 = \overline{AC + BC}$$

solution

The product term AC is common to F_1 and F_2 so we need only 4 product term



Programmable Logic array (PLA)

- * PLA is fixed architecture logic device with programmable AND gates followed by programmable OR gates.
- * PLA is basically a type of programmable logic device used to build a reconfigurable digital circuit
- * programmable logic device have an undefined function at the time of manufacturing, but they are programmed before being made into use.
- * PLA is a combination of memory logic

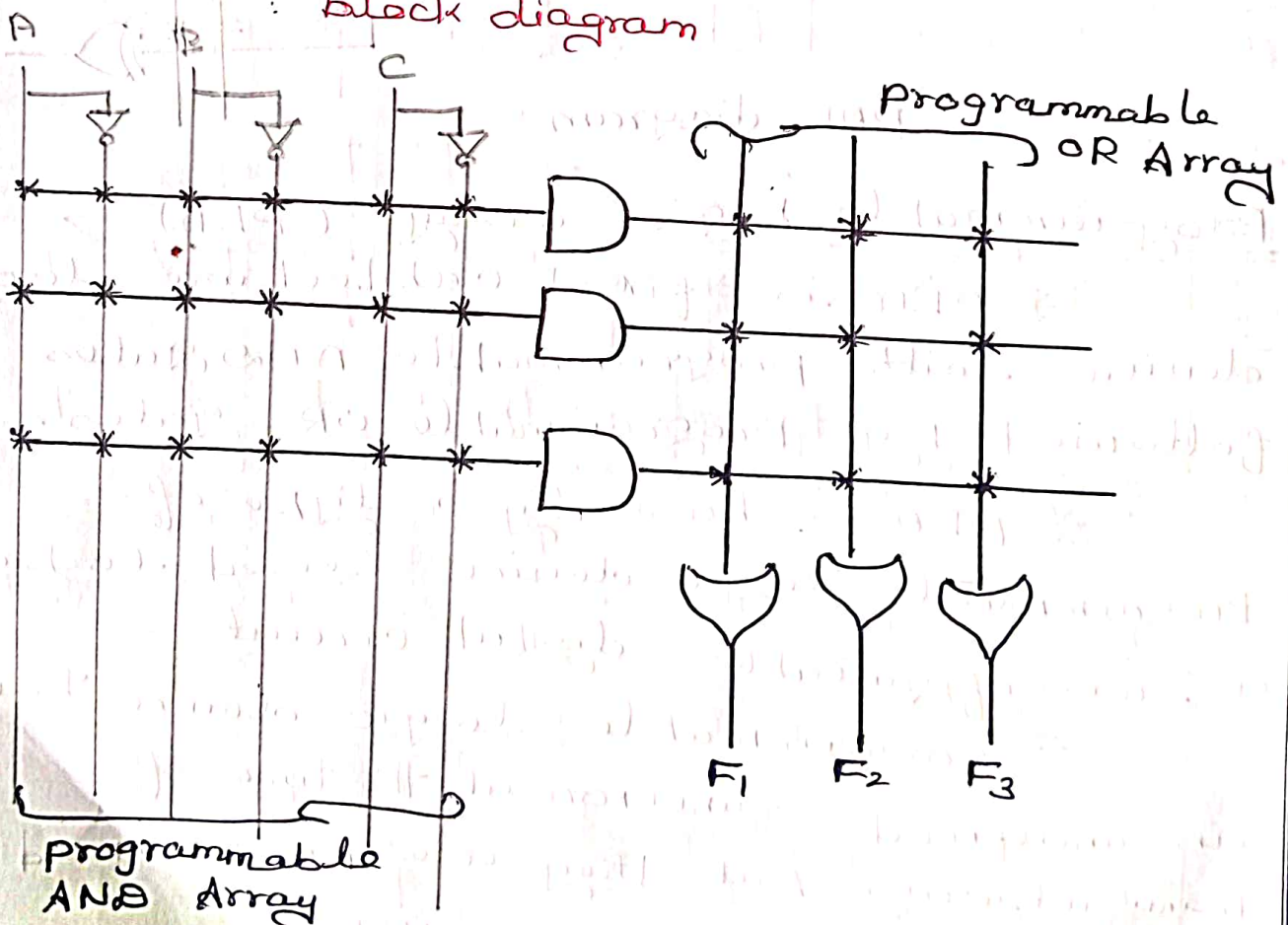
* The PLA is used to Implement a complex combinational circuit

* A PLA is similar to a ROM concept except that does not generate all the minterms as in the ROM.

* PLA the decoder is replaced by a group of AND gates each of which can be programmed to produce a product AND terms of input variable



block diagram



Basic configuration of PLA

Application:-

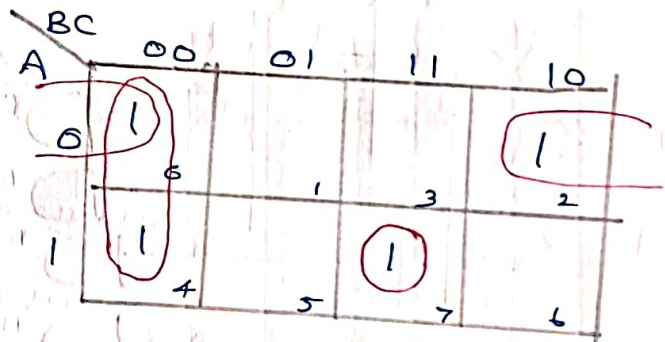
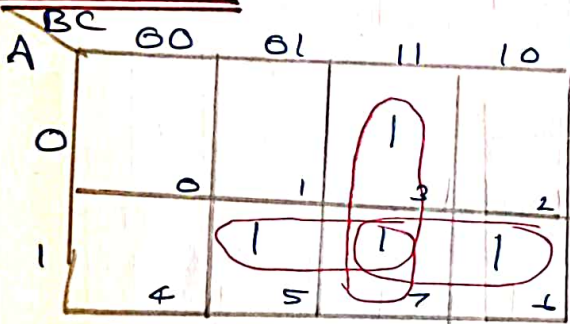
- * It is used for provide control over datapath
- * It is used for counter
- * It is used for decoder
- * It is used for Bus interface in programmed I/O

Implement the following functions with PLA

$$F_1(A, B, C) = \sum_m(3, 5, 6, 7)$$

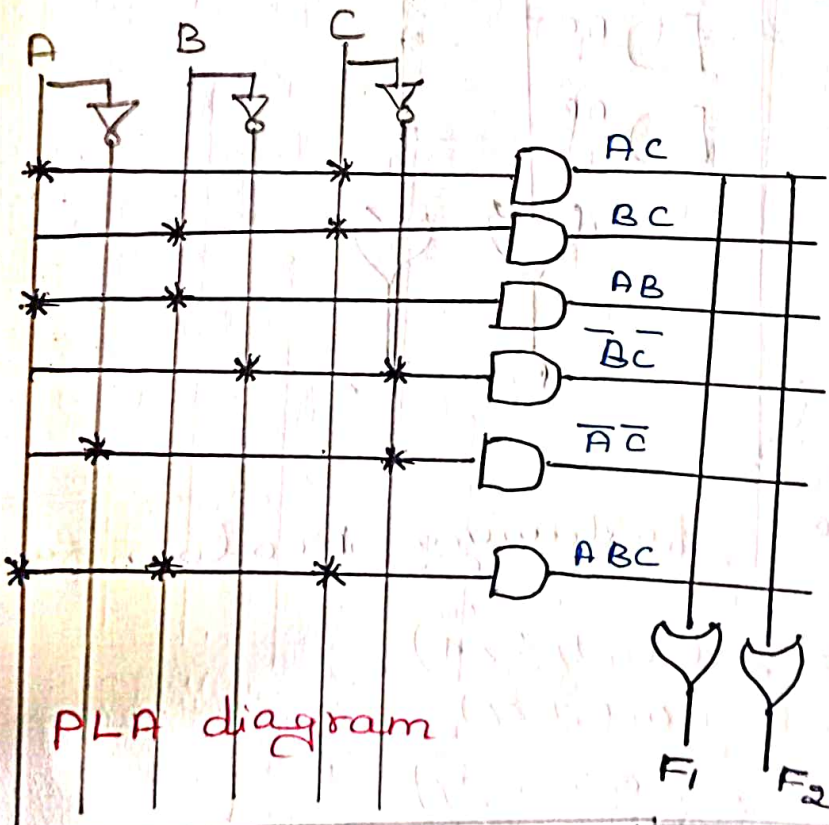
$$F_2(A, B, C) = \sum_m(0, 2, 4, 7)$$

Solution:-



$$F_1 = AC + BC + AB$$

$$F_2 = \bar{B}\bar{C} + \bar{A}\bar{C} + ABC$$



PLA diagram

Implement the following two Boolean function with a PLA

$$F_1(A, B, C) = \sum m(0, 1, 2, 4)$$

$$F_2(A, B, C) = \sum m(0, 5, 6, 7)$$

Solutions :-

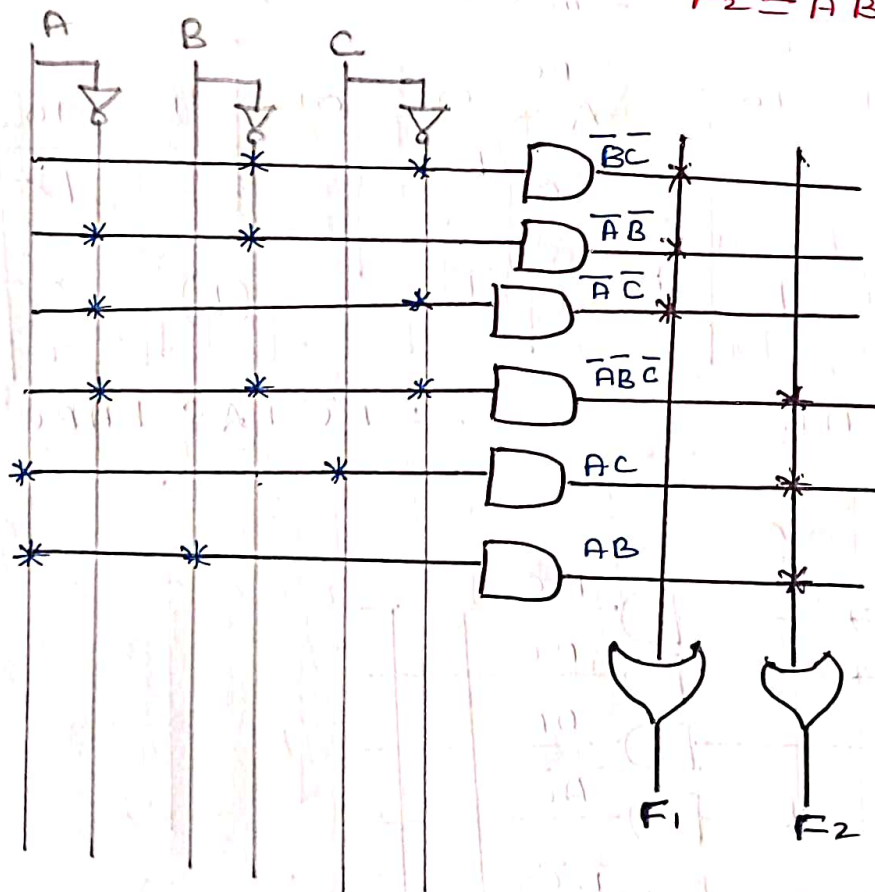
For F_1

BC \ A	00	01	11	10
0	1	1		1
1	1			
	0	1	3	2
	4	5	7	6

$$F_1 = \overline{B}\overline{C} + \overline{A}\overline{B} + \overline{A}\overline{C}$$

BC \ A	00	01	11	10
0	1			
1		1	3	2
	4	5	7	6

$$F_2 = \overline{A}\overline{B}\overline{C} + AC + AB$$



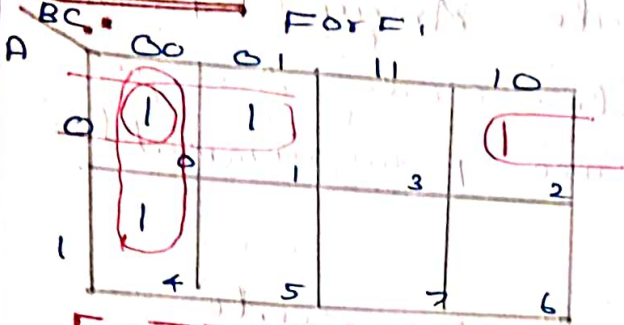
Implement the following Boolean function with a PLA

$$F_1(A, B, C) = \sum m(0, 1, 2, 4)$$

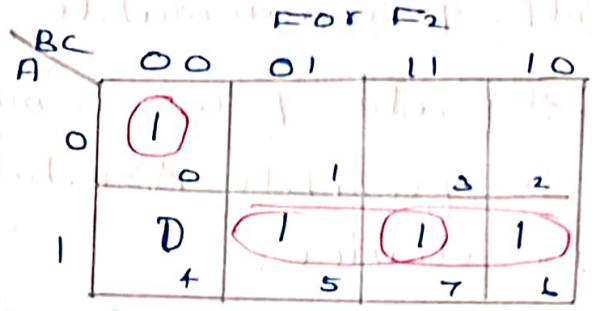
$$F_2(A, B, C) = \sum m(0, 5, 6, 7)$$

$$F_3(A, B, C) = \sum m(0, 3, 5, 7)$$

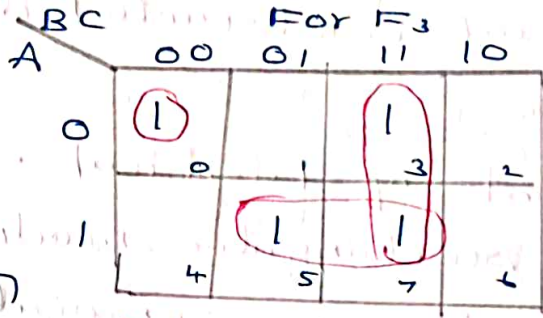
solution



$F_1 = \overline{B}C + A\overline{B} + \overline{A}C$

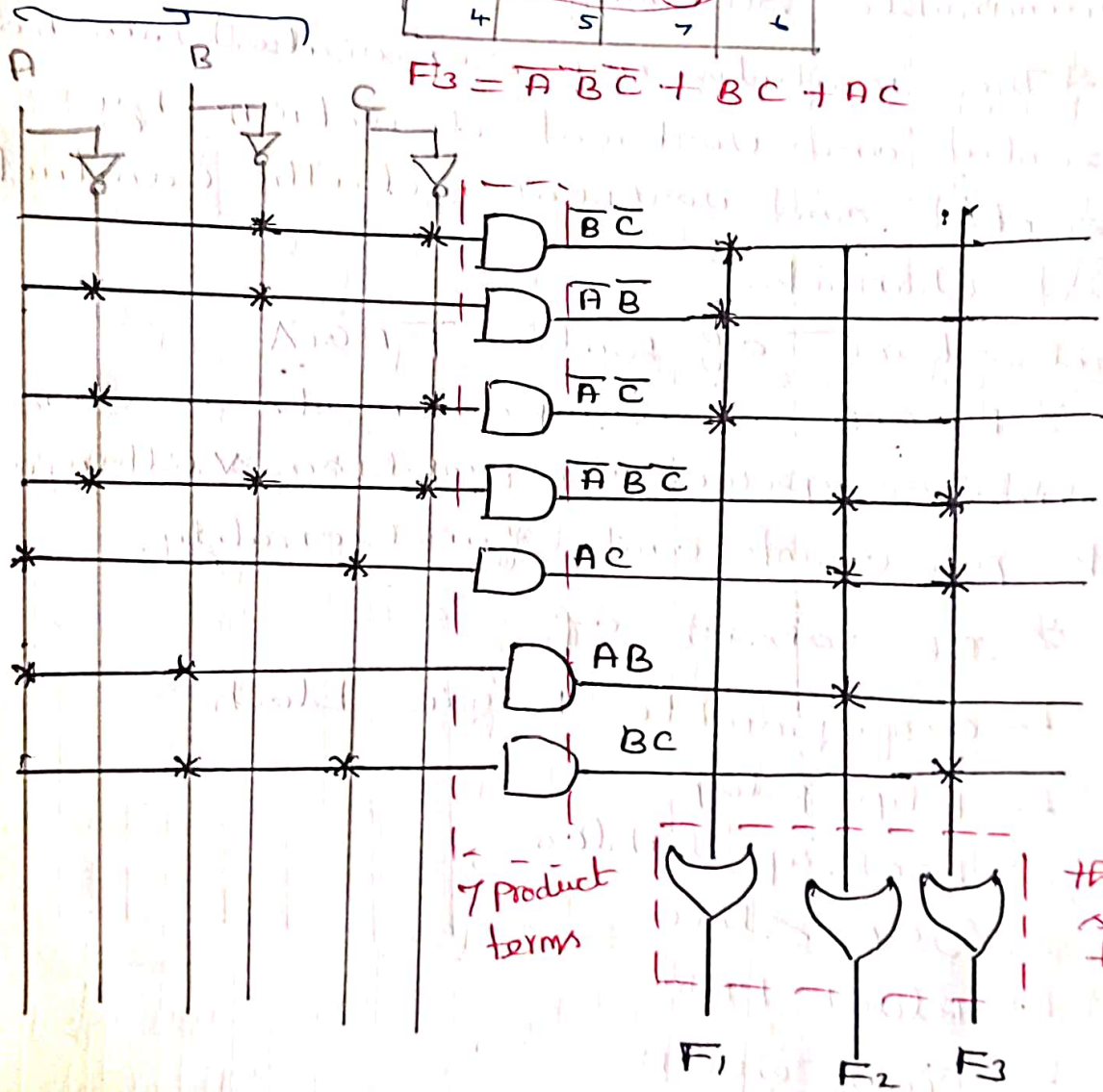


$F_2 = \overline{A}BC + AC + AB$



$F_3 = \overline{A}BC + BC + AC$

3 input



7 Product terms

three sum terms

Field Programmable Gate Array (FPGA)

* Field Programmable Gate Array is a flexible Architecture programmable logic device.

* It is single very large scale Integrated circuit constructed on a single piece of silicon.

* It consists of identical individually programmable rectangular modules

* The modules are separated in both horizontal and vertical directions by horizontal and vertical metallic conductors called channels.

Architecture of xilinx FPGA

* FPGA is improvements in density performance, power consumption, voltage levels pin counts and functionality.

* It consists of

↳ configurable Logic block

↳ Flip Flop

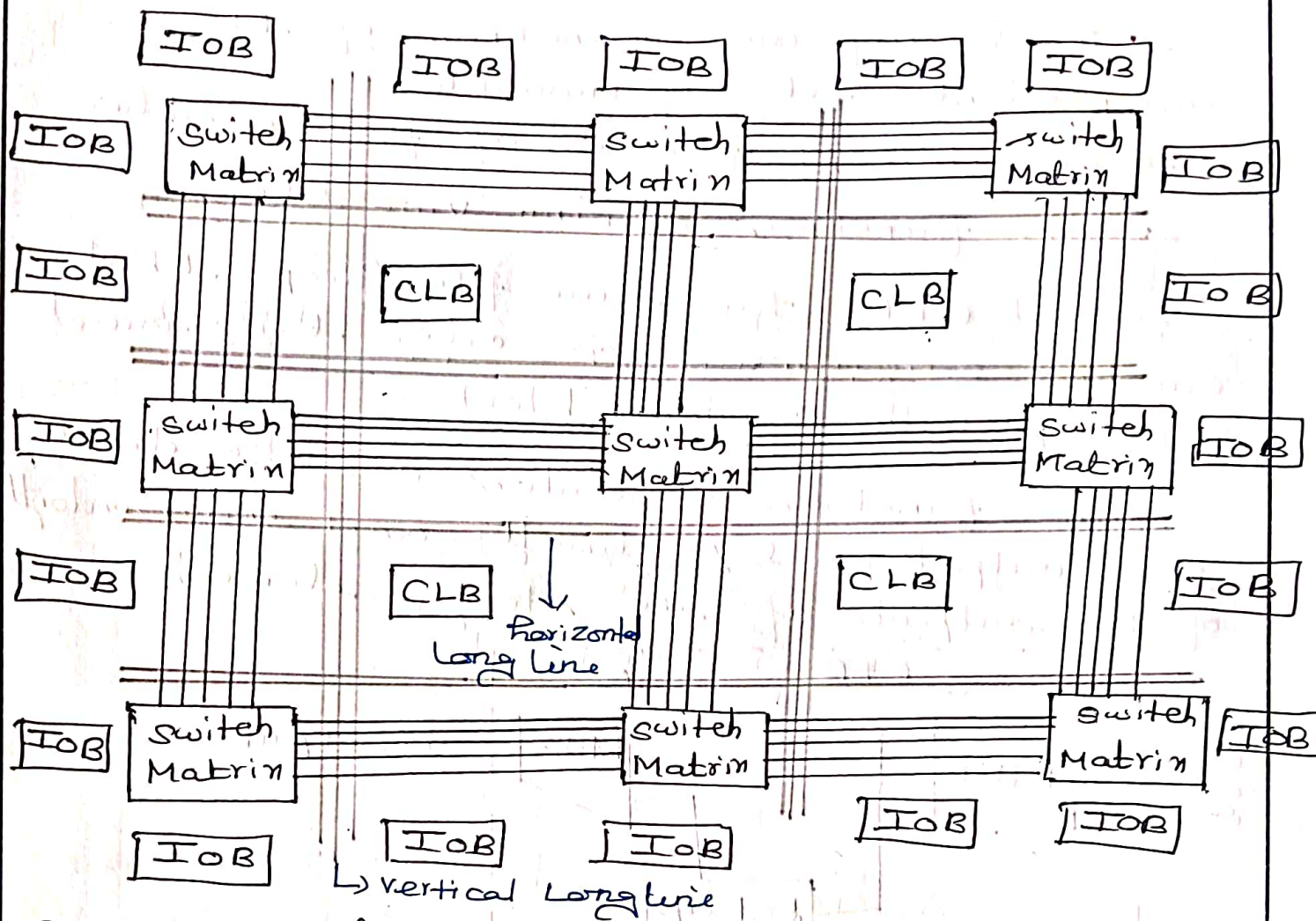
↳ Look up tables

↳ DSP slices

↳ Block RAM

↳ Transceivers

↳ Input / Output Blocks



Configurable logic block (CLB)

- * Configurable Logic blocks contain logic for the FPGA. Each CLB consists of
 - * programmable look up table
 - * Multiplexer
 - * Registers
 - * Path for control signals
- * Two of the function generators (F and G) of the look up table can generate any arbitrary function of four inputs and the third function generator (H)
- * The three function generators can be programmed to generate

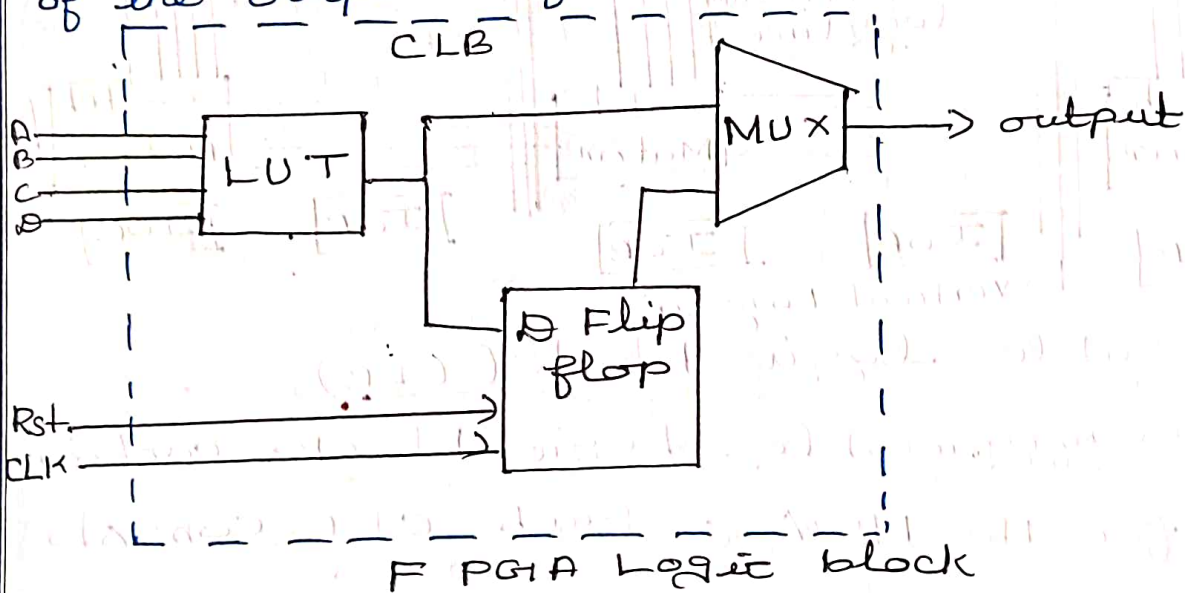
→ Three different functions of three independent sets of variables

→ An arbitrary function of five variables

→ Some functions of nine variables

* Each CLB has two flip flops that can be configured as edge triggered flip flop with common block

* The function generators can also drive two outputs (x & y) directly and independently of the outputs of the storage elements

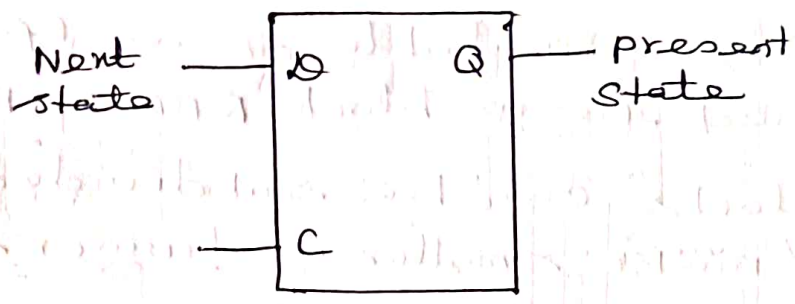


Flip Flops:-

* A flip flop is a circuit that has two stable states and can be used to store state information.

* Flip flops are binary shift registers that synchronize logic and save logical states between clock cycles within an FPGA circuit

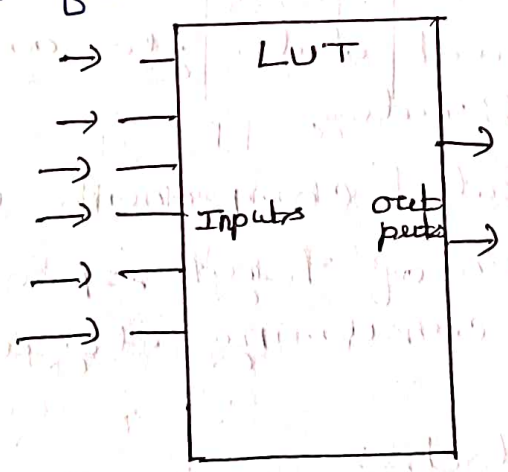
* A flip flop stores a single bit of data



- ↳ Data input of a memory device is called the Next state
- ↳ output from a memory device is called the present state

Lookup Tables: (LUT)

- * A Look up table (LUT) determines what the output is for any given inputs.
- * In the context of combinational logic it is the truth table and defines how combinatorial logic behaves.
- * A truth table is predefined list of output for each combination of input



DSP slices:-

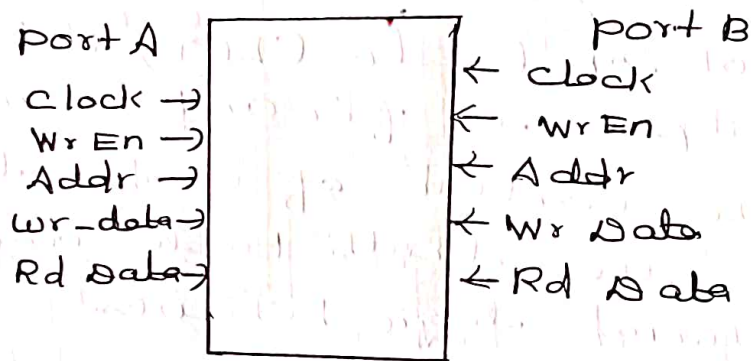
- * DSP slice or sometimes referred to as a block or cell, is one of the specialized components in a FPGA
- * It carries out digital signal processing functions like filtering or multiplying more efficiently than using many CLB

Block RAM :-

* The memory available on an FPGA chip is referred to as block RAM or BRAM.

* These blocks can be subdivided or cascaded to make smaller or larger sizes BRAM available.

* Digital signal processing algorithms frequently need to keep track of an entire block of data and without onboard memory



Transceivers :-

* Transceivers are made to transmit and receive serial data to and from one FPGA at high rates.

* This dedicated component allows for implementation of high speed data transfer without consuming logic resources of the FPGA.

Input / output blocks :-

* Input output (IO) blocks are the components through which data transfer is and out of the FPGA.

* The IO blocks are configurable in several ways depending on the type of data

∴ * They operate at lower speeds. than transceivers but maintain more functional flexibility.

Advantages :-

- ↳ Ability to reprogram in the field to fix bugs
- ↳ very less time to configure
- ↳ Lower non-recurring Engineering costs

Applications :-

- * Consumer electronics
 - ↳ digital flat panel displays
 - ↳ Information applications
 - ↳ home networking
- * Data Center
- * Industrial
 - ↳ Industrial Automation
 - ↳ Medical Imaging
- * Medical
 - ↳ diagnostic monitoring
 - ↳ security applications
- * Wireless communications
 - ↳ RF
 - ↳ baseband
 - ↳ connectivity
 - ↳ WCDMA
 - ↳ HSPA
 - ↳ video and Image processing

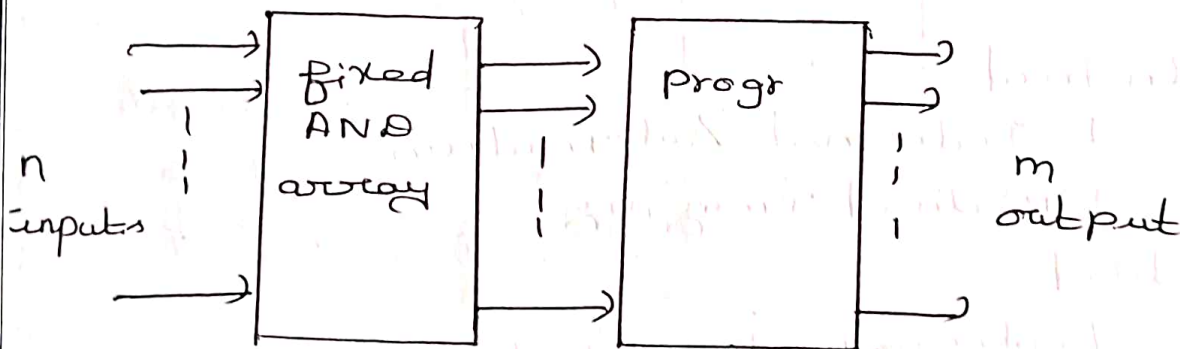
programmable Read only Memory (PROM)

* Read only Memory ROM is a memory device which stores the binary information permanently

* If the ROM has programmable feature, then it is called as programmable ROM PROM.

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

* PROM is a programmable logic device that has fixed AND Array & programmable OR Array.



* Here the inputs of AND gates are not of Programmable type.

* we have to generate 2^n product terms by using 2^n AND gates having n input each.

* we can implement these product terms by using $n \times 2^n$ decoder so this decoder generates n minterms

* Here the inputs OR gates are programmable.

* That means we can program any number of required product terms, since all the outputs of AND gates are applied as inputs to each OR gate.

* The outputs of PROM will be in the form of sum of minterms.

CPLD - complex programmable Logic device

* Complex programmable Logic device (CPLD) is one of the PLD. It is used for the implementation of logical circuits.

* The programmable logic devices such as PLA and PAL have limited number of inputs, product terms and outputs.

* This device can support up to 32 total number of inputs and outputs only.

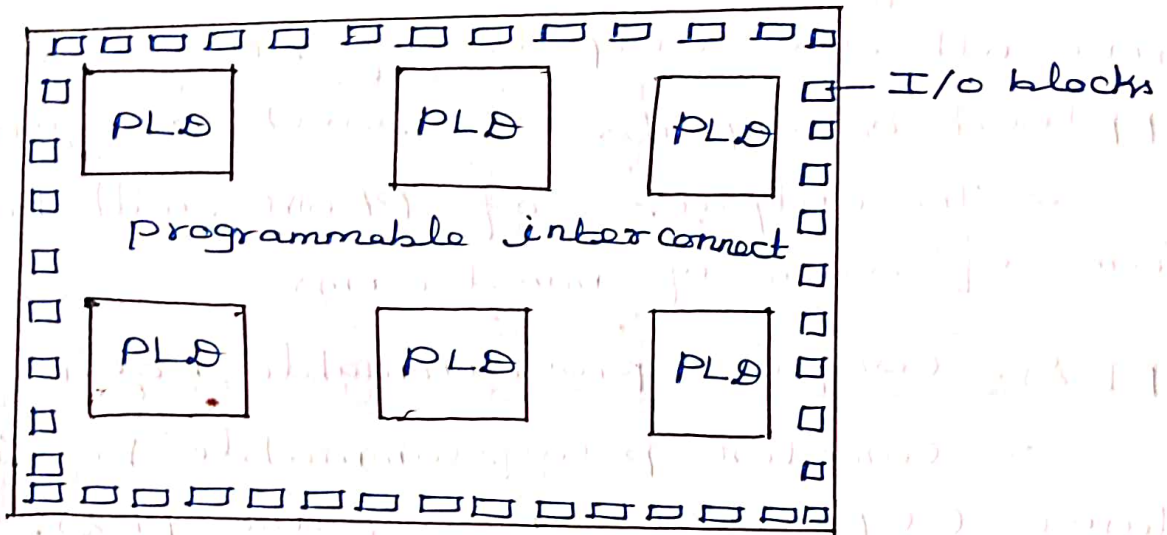
* The complexity of any digital IC chip can be specified in terms of number of equivalent 2-input NAND gates.

* A typical PAL has 8 macro cells. If each macro cell represents about 20 equivalent gates.

* For circuit requiring very large number of gates, CPLD having large

Number of macrocells can implement circuits of up to about 10,000 equivalent gates.

* They are as fast as PLAs but more



General structure of a CPLD

* They digital IC that are just like a large number of PAL in single silicon chip connected to each other through a cross point switch.

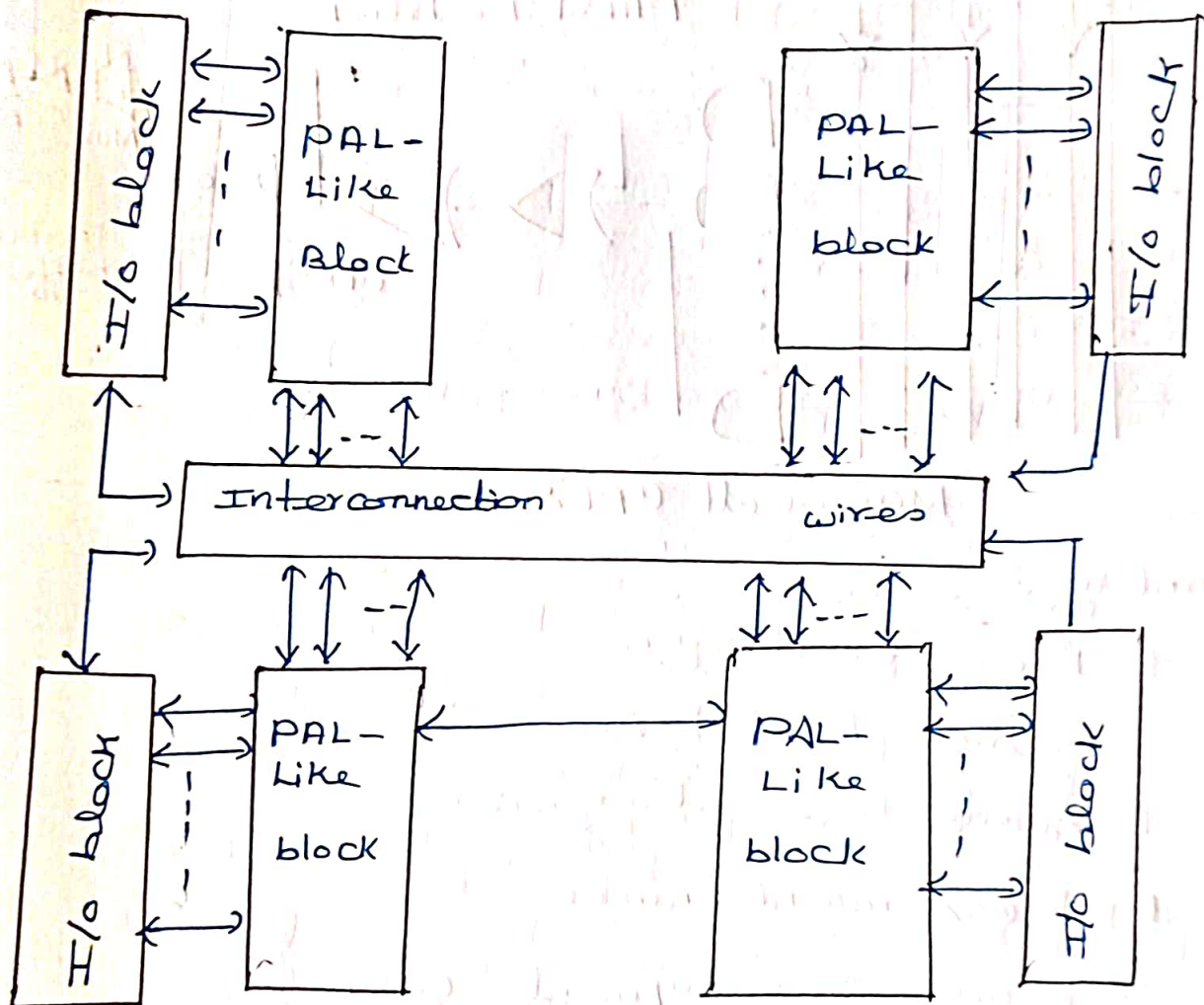
Architecture of CPLD

* It consists of a number of PAL like blocks, input/output blocks and a set of interconnection wires.

* The PAL like blocks are connected to set of interconnection wires and to an input/output block.

* The input block is used to drive signals to the pins of the CPLD at the appropriate voltage levels with appropriate current.

- * A PAL like block (also called functional block) usually consists of 16 macro cells
- * Each macro cell consists of an AND-OR configuration, an EX-OR gate a flip flop, a multiplexer and a tristate buffer

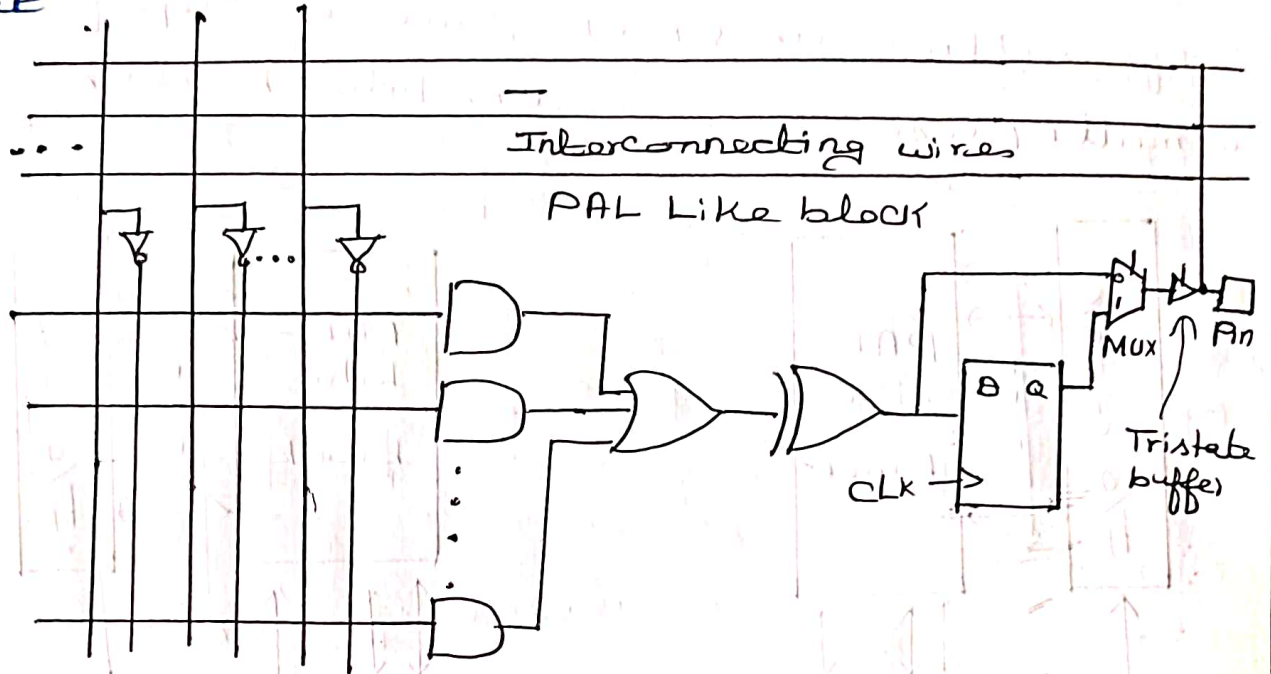


Architecture of CPLD

- * Each AND-OR configuration usually consists of 5 to 20 AND gates and an OR gate with 5 to 20 inputs
- * An EX-OR gate is used to obtain the output of OR gate in an inverted or non inverted form depending upon its other

input being 1 or 0 respectively

* A buffer acts as a switch, which enables the chip pin to be used output or as an input



Macrocell CPLD

Advantages:-

- ↳ Easy to design
- ↳ Reduce board Area
- ↳ Available in tiny sizes
- ↳ Less maintenance
- ↳ Lower development cost

UNIT-5

VHDL

* VHDL is one of the commonly used Hardware Description Languages (HDL) in digital circuit design.

* VHDL stands for VHSIC Hardware Description Language.

* VHSIC \rightarrow Very High speed integrated circuit.

* It is a programming language that describes a logic circuit by function, data flow behavior, and/or structure.

* It is used for many purposes.

\rightarrow For describing hardware.

\rightarrow As a Modeling Language.

\rightarrow For simulation of hardware.

\rightarrow For early performance estimation.

\rightarrow For synthesis of hardware.

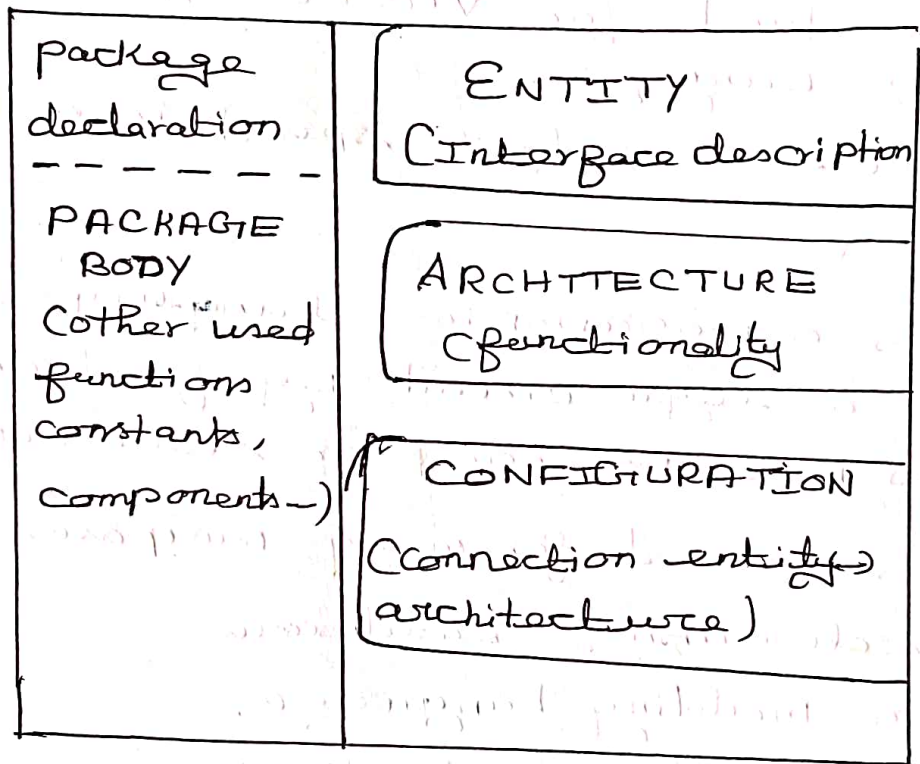
\rightarrow For fault simulation, test & verification of designs.

* A VHDL design begins with an ENTITY block describes the interface for the design.

* The interface defines the input and output logic signals of the circuit being designed.

* The architecture block describes the internal operation of the design.

* The blocks are numerous other functional blocks used to build the design elements of the logic circuit being created



VHDL - program structure

* entity entity name is

[port Cinterface - signal - declaration];

* end [entity] [entity-name];

* architecture architecture - name of entity name is [declarations]

* begin
architecture body

* end
[architecture] [architecture-name];

RTL Design (Register Transfer Level design)

* Register transfer level design lies between a purely behavioral description of the desired circuit and a purely structural one

* RTL description a circuit register and the sequence of transfer between these register but does not describe the hardware used to carry out these operations.

* Determine the number and sizes of registers needed to hold the data used by the device

* Determine the logic and arithmetic operations that need to be performed on these register contents. and

* design a sequential circuit whose output control how the register contents are updated in order to obtain the desired results.

* An RTL design is similar to writing a computer program in a conventional programming language.

* choosing registers is the same as choosing variables analogous to writing expressions involving the variables and operations

* Designing the controller sequential circuit is similar to deciding on the flow

flow control within the program.

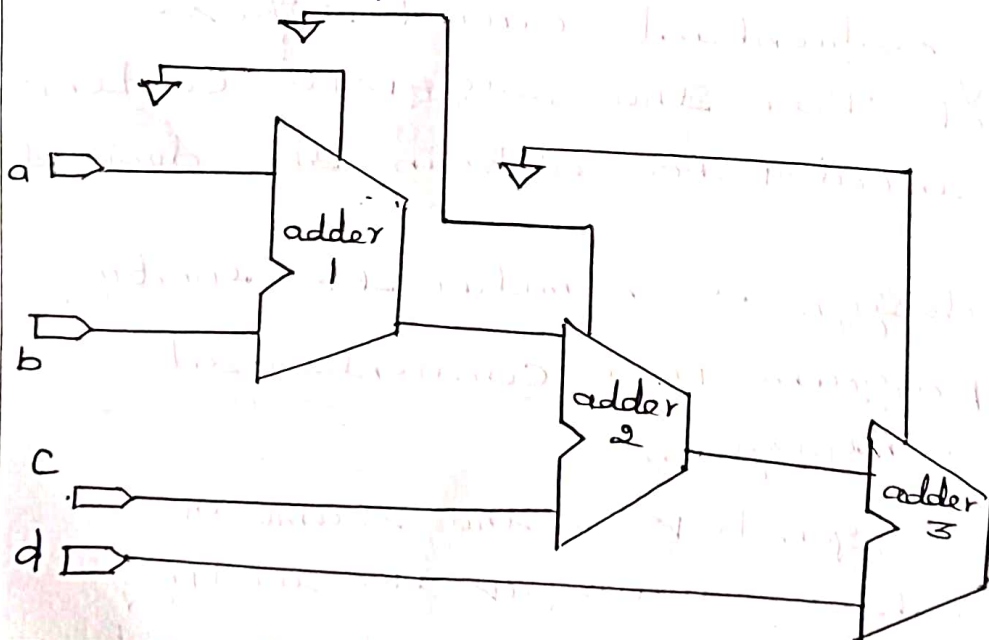
$$s \leftarrow (C + b) + c + d;$$

* This particular description is similar enough that it can be synthesized.

* The resulting circuit will be a fairly large combinational circuit comprising three adder circuit

* A behavioral description not being concerned with implementation details would be complete at this point

```
s = 0;  
s = s + a;  
s = s + b;  
s = s + c;  
s = s + d;
```



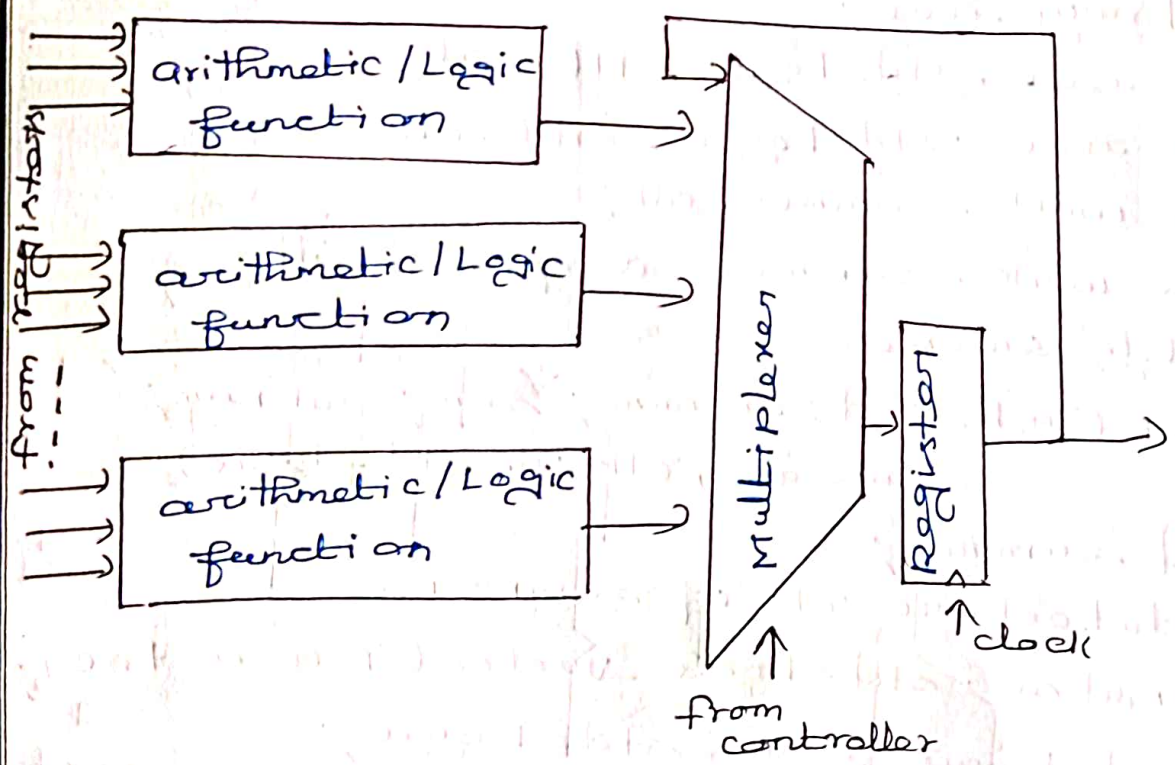
* where each operation is executed sequentially

* The logic required is now one adder, a register to hold the value of s is

between operations, a multiplexer to select the input to be added and a circuit to clear 0 at the start of the computation

* The process requires more steps and will take longer time. circuits that divide up a computation into a sequence of arithmetic and logic operations. are quite common and this type of design is called RTL or data flow design.

* Register and combinational function blocks called data path the controller that controls the transfer of data through the function blocks between the registers.



* RTL design the gate level design and optimization of the data path (register, multiplexer, and combinational functions) is done by the synthesizer

* The designer must design the sequential circuit and decide which register transfers are performed in which state.

* RTL designer can trade off datapath complexity against speed.

* RTL design is well suited for the design of CPUs and special purpose processors such as disk drive controller, video display cards, network adapter, etc.

* The width of register types of combinational functions and their input will be determined by the application.

Examples:-

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.summer.all;
use work.summer_components.all;
entity summer is
port (a, b, c, d; in num; sum; out num;
      update, clk : in std_logic);
end summer;
architecture rtl of summer is
signal sel : std_logic_vector (1 down to 0);
signal Load, clear : std_logic;
begin
  d1 : data_path port map (a, b, c, d, sum, sel,
                          Load, clear, clk);
  C1 : controller port map (update, sel, Load, clear,
                           CLK);
end rtl;
```

Package declaration:-

- * A package is a convenient mechanism to store and share declarations
- * It is optional design unit
- * A set of declarations contained in a package declaration may be shared by many design units
- * It defines items that can be made visible to other design units
- * A package is represented by
 - ↳ * package declaration
 - ↳ * package body

Package declaration

* It defines the interface to the package

Syntax:

package declarations
 subtype declarations
 constant declarations
 signal declarations
 variable declarations
 sub program declarations
 file declarations
 alias declarations
 component declarations
 attribute declarations
 disconnection specifications
 use clauses
 END package - name;

* The items declared in a package declaration can be accessed by other design units by using the library and use clauses.

package body :-

* It contains the details of a package that is the behavior of the subprograms and the values of the deferred constants which are declared in a package declaration.

* The package body may contain other declarations.

Syntax;

package body package - name IS

subprogram bodies

complete constant declarations

subprogram declarations.

⊕ type and subtype declarations

file and alias declarations

use clauses

END package - name;

* The name of the package must be same as the name of its corresponding package declarations.

* The package declaration does not have any subprogram (or) deferred constant declarations. a package body is not necessary.

Example :-

The 4 bit full adder circuit includes package declarations.


```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY full adder IS
PORT (x, y, cin : in bit, cout, sum : out bit);
END full adder;

```

ARCHITECTURE equation of full adder IS

```

begin
  sum <= x XOR y XOR cin;
  cout <= (x and y) or (x and cin) or (y and cin);
END equation;

```

ENTITY adder 4 IS

```

PORT (a, b : in bit_vector (3 down to 0); ci : in bit;
      s : out bit_vector (3 down to 0); co : out bit);
END adder 4;

```

ARCHITECTURE structure of adder 4 IS

Component full adder

```

PORT (x, y, cin, in bit; sum, cout, out bit);
END component
signal C : bit_vector (3 down to 1);

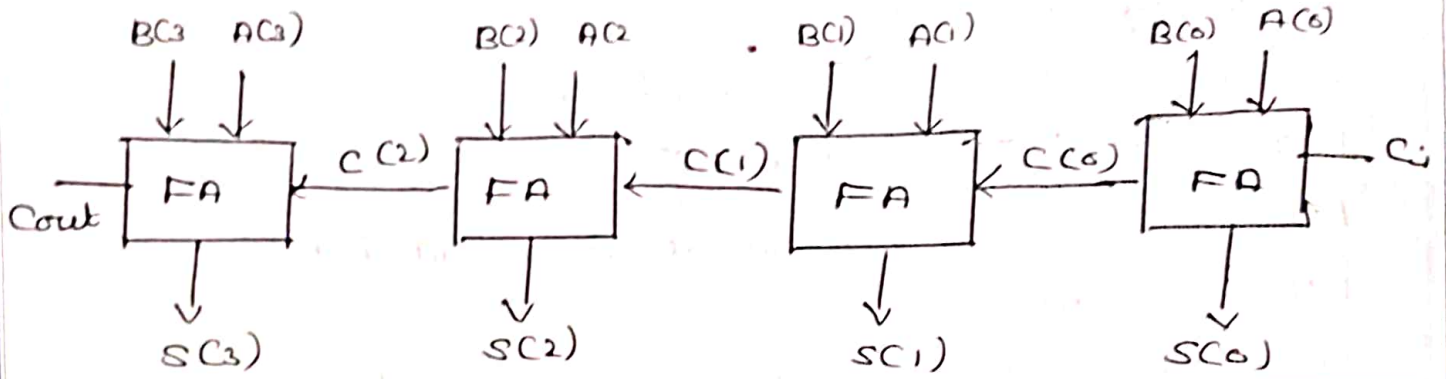
```

begin

```

FA0 : full adder port map (a(0), b(0); ci, c(0)
                           sc(0));
FA1 : full adder port map (a(1), b(1) c(1) c(2), s(1));
FA2 : full adder port map (a(2), b(2), c(2), c(3), s(2));
FA3 : full adder port map (a(3), b(3), c(3), c(0), s(3));
END structure.

```



write HDL for half adder



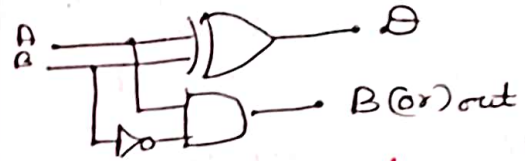
Half adder

```

Library IEEE;
use IEEE.STD_Logic_1164.all;
entity half_adder is
port (a, b: in STD_Logic;
      s, c: out STD_Logic);
end half_adder;
architecture half_add_arc of
half_adder is
begin
  s <= a xor b;
  c <= a and b;
end half_add_arc

```

write HDL for half subtractor



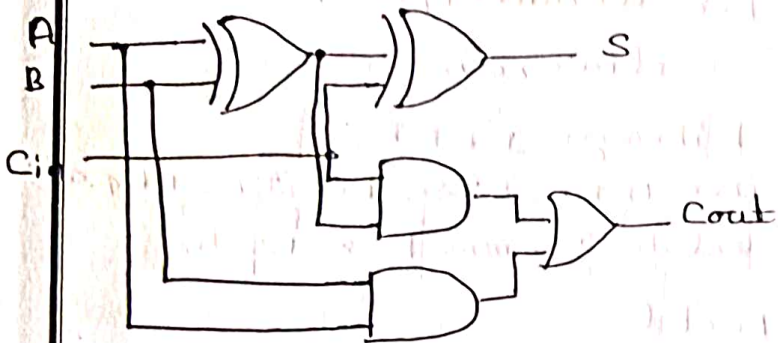
Half subtractor

```

Library IEEE;
use IEEE.STD_Logic_1164.all;
entity half_subtractor is
port (a, b: in STD_Logic;
      diff, borrow: out STD_Logic);
end half_subtractor;
architecture half_subtractor_arc
of half_subtractor is
begin
  diff <= a xor b;
  borrow <= (not a) and b;
end half_subtractor_arc

```

Write HDL for full adder



Library IEEE;

use IEEE.STD-LOGIC-1164.ALL;

entity full add is

port (A: in STD-LOGIC;

B: STD-LOGIC;

C in: STD-LOGIC;

S: out STD-LOGIC;

Cout: out STD-LOGIC);

end full add;

architecture behavioral of
full add is begin

S <= A XOR B XOR C in;

Cout <= (A AND B)

OR

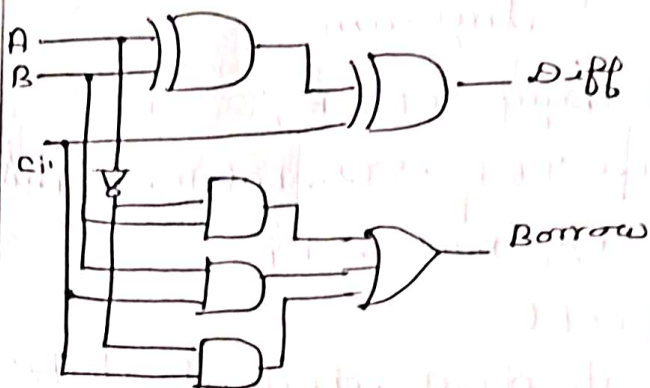
(C in AND B A)

OR

(C in AND B B);

end behavioral

Write HDL for full subtractor



Library IEEE;

use IEEE.STD-LOGIC-1164.all;

entity full subtractor is

port (a, b, c: in STD-LOGIC;

difference, borrow: out

STD-LOGIC);

end full - subtractor

architecture full sub of
full - subtractor is
begin

difference <= a XOR b XOR c;

borrow <= ((not a) and b)

OR

(b and c)

OR

(c and (not a));

end full sub;

Write HDL program for
8:1 Multiplexer
diagram

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity mux 8-1 is
port (
    d0, d1, d2, d3, d4, d5, d6, d7:
        in bit;
    S0, S1, S2: in bit;
    F: out bit);
end mux 8-1;
architecture mux 8-1 arc of
mux 8-1 is signal X0, X1, X2,
X3, X4, X5, X6, X7: bit
begin
X0 <= d0 and (not S0) and (not
S1) and (not S2);
X1 <= d1 and (not S0) and (not S1)
and S2;
X2 <= d2 and (not S0) and S1 and S2;
X3 <= d3 and (not S0) and S1 and S2;
X4 <= d4 and S0 and (not S1) and
not S2;
X5 <= d5 and S0 and (not S1) and
S2;
X6 <= d6 and S0 and S1 and (not
S2);
X7 <= d7 and S0 and S1 and S2;
F <= X0 or X1 or X2 or X3 or X4 or
X5 or X6 or X7;
end mux 8-1 arc;
```

Write HDL program for
1:8 demultiplexer
diagram

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity demux 1-8 is
port (
    din: in bit;
    S0, S1, S2: in bit;
    d0, d1, d2, d3, d4, d5, d6, d7: out
        bit;
);
end demux 1-8;
architecture demux 1-8 arc of
demux 1-8 is begin
d0 <= din and (not S0) and (not S1)
and (not S2);
d1 <= din and (not S0) and (not S1) and
S2;
d2 <= din and (not S0) and S1 and
not S2;
d3 <= din and (not S0) and S1 and S2;
d4 <= din and S0 and (not S1) and not S2;
d5 <= din and S0 and (not S1) and S2;
d6 <= din and S0 and S1 and (not S2);
d7 <= din and S0 and S1 and S2;
end demux 1-8 arc;
```

write VHDL program for 4 bit up counter

diagram

```

Library IEEE;
use IEEE.std.LogiC_1164.all;
use IEEE.std.LogiC_unsigned.all;
entity counter is
port C, CLR: in std-Logic;
Q: out std-logic-vector(3 downto 0);
end counter;
architecture bhv-counter is
signal tmp: std-logic-vector(3 downto 0);
begin
process (C, CLR)
begin
if (CLR = '1') then
tmp <= "0000";
else if (C'even and C = '1') then
tmp <= tmp + 1;
end if;
end process
Q <= tmp;
end bhv-counter

```

write VHDL program for Mod-b synchronous counter

diagram

```

Library IEEE;
use IEEE.STD-LOGIC_1164.all;
use IEEE.std-Logic_arith.all;
use IEEE.std-Logic_unsigned.all;
entity mod b counter is
port CLK, reset: in STD-LOGIC;
dout: out
STD-LOGIC-VECTOR(2 downto 0);
end mod b-counter
architecture mod b of mod b-counter is begin
counter: process (CLK, reset) is
variable m: integer range 0 to 7
:= 0;
begin
if (reset = '1') then
m := 0;
else if (rising-edge (CLK)) then
m := m + 1;
end if
if (m = b) then
m := 0;
end if
dout <= conv_std-Logic-vector(m, 3);
end process counter;
end mod b;

```

Write VHDL program for flip flops:-

SR flip flop

```
Library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_arith.all;  
use ieee.std_logic_unsigned.all;  
entity SR_FF is  
port (S, R, CLOCK: in std_logic;  
Q, Qbar: out std_logic);  
end SR_FF;
```

architecture behavioral of

SR_FF is begin

process (CLOCK)

variable tmp: std_logic;

begin

if (CLOCK = '1' and CLOCK event)

if (S = '0' and R = '0') then

tmp = tmp

else if (S = '1' and R = '0') then

tmp = 'z'

else if (S = '0' and R = '1') then

tmp = '0';

else

tmp = '1';

end if;

end if;

Q <= tmp;

QBAR <= not tmp;

end process;

end behavioral

JK flip flop

```
Library ieee;  
ieee std_logic_1164.all;  
use ieee.std_logic_arith.all;  
use ieee.std_logic_unsigned.all;  
entity JK_FF is  
port (J, K, CLOCK: in std_logic;  
Q, QB: out std_logic);  
end JK_FF;
```

architecture behavioral of

JK_FF is begin

process (CLOCK)

variable tmp: std_logic;

begin

if (CLOCK = '1' and CLOCK event)

if (J = '0' and K = '0') then

tmp = tmp

else if (J = '1' and K = '1') then

tmp := not tmp;

else if (J = '0' and K = '1') then

tmp = '0';

else

tmp = '1';

end if;

end if;

Q <= tmp;

QB <= not tmp;

end process;

end behavioral;

D-Flip flop

```
Library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_arith.all;  
use ieee.std_logic_unsigned.all;  
entity D_FF is  
Port (D, CLock : in std_logic;  
Q : out std_logic);  
end D_FF;  
architecture behavioral of  
D_FF is begin  
process (CLock)  
begin  
if (CLock = '1' and clock'event) then  
Q <= D;  
end if;  
end process;  
end behavioral;
```

T Flip flop

```
library IEEE;  
use IEEE.STD_Logic_1164.all;  
entity T_FF is  
port CT : in std_logic;  
clock : in std_logic;  
Q : out std_logic;  
end T_FF;  
architecture behavioral of  
T_FF is  
signal tmp : std_logic;  
begin  
process (CLock)  
begin  
if clock'event and clock = '1' then  
if T = '0' then  
tmp <= tmp;  
else if T = '1' then  
tmp <= not (tmp);  
end if;  
end if;  
end process;  
Q <= tmp;  
end behavioral;
```