MAYUR
Date
Page

* Flip-Flop Defination
* Latches Defination
(1) SR Latches using NOR Gate
(2) SR Latches using NAND Gate
(3) SR or RS Flip-Flop.
(4) JK Flip-Flop.
(5) JK Master-Slave Flip-Flop.
(6) D Flip-Flop
(7) T Flip-Flop
(8) Register Defination
(9) Types of Register (a) SISO, (b) SIPO, (c) PISO, (d) PIPO
(10) Right Shift Register, Left Shift Register.
(11) Counter Defination
(12) Difference between Synchronous and ripple counter.
(13) 2 bit Synchronous up counter.
(14) 3 bit Synchronous up counter.
(15) 3 bit Synchronous up/down counter.
(16) 3 bit Asynchronous Counter (Ripple Counter)
(17) MOD-7 Ripple Counter (Asynchronous Counter)
(18) MOD-7 Synchronous Counter.
(19) Basic Logic Gates AND, OR, NOT.
(20) Universal Gates NAND and NOR.
(21) Ex-OR and Ex-NOR Gates.
(22) Implement AND, OR, NOT using NAND.
(23) Implement AND, OR, NOT using NOR.
(24) Define combinational circuit. Write combination circuit design procedure. Design Half Adder and Full Adder with its truth table, Logic Diagram and Boolean Expression.

NAMAYUR 780X420 SINGLE LINE 2001 205X256 667 OK UPt 30-03-2021

MAYUR
Date
Page

(25) Half Subtractor and Full Subtractor.
(26) Define Encoder. Design 4X2 Encoder.
(27) Design Decimal to BCD Encoder.
(28) Octal to Binary Encoder
(29) Define Decoder. Design 2X4 Decoder.
(30) Design BCD to Decimal Decoder.
(31) Define Priority Encoder. Design 4:2 priority encoder with its block diagram, truth table, circuit diagram and mathematical expression.
(32) What is race-around condition? Explain how JK flip-flop is used to eliminate race around condition.
(33) Multiplexer. Design 2X1 Multiplexer.
(34) Design 4X1 multiplexer.
(35) Design 8X1 multiplexer.
(36) Implement Full adder using IC 4X1 multiplexer.
(38) Define PLA with its block diagram. Realize BCD to gray code converter using PLA.
(39) Implement $F = \Sigma(0,1,4,5,7)$ using (a) Multiplexer (b) Decoder (c) PLA.
(40) Define Decoder. Draw logic diagram and truth table for 3 to 8 Decoder.
(42) De-Morgan's Theorem.

Flip-Flops

MAYUR
Date
Page

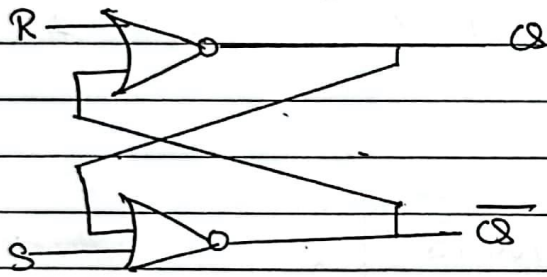
⇒ Flip flop is a sequential logic circuit which is capable of storing one bit. The output of flip may be 0 or 1.

Latches

→ Latches is one bit storage at a time
→ Output is connected to input.
→ It is a basic storage element.

S = Set
R = Reset

(1) SR Latches using NOR Gate.



Case-I
S=0, R=1, Q=0, $\overline{Q}$=1
S=0, R=0, Q=0, $\overline{Q}$=1 } Memory

Case-II
S=1, R=0, Q=1, $\overline{Q}$=0
S=0, R=0, Q=1, $\overline{Q}$=0 } Memory

Truth Table for NOR Gate

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

Case-III
S=1, R=1, Q=0, $\overline{Q}$=0
S=0, R=0, Q=0, $\overline{Q}$=1 } Not used

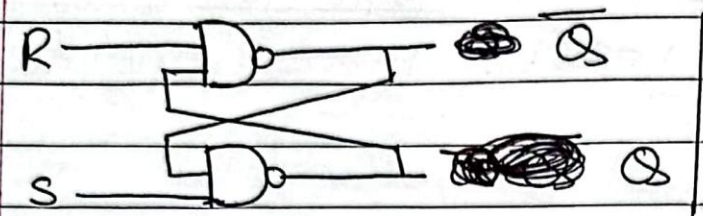
Final Truth Table

S	R	Q	$\overline{Q}$
0	0	Memory	
0	1	0	1
1	0	1	0
1	1	Not used	

MAYUR 780X420 SINGLE LINE 2COL 205X256 667 OK

MAYUR
Date
Page

(2) SR Latch using NAND Gate.



Cas-I

S=0, R=1, Q=1, $\bar{Q}$=0

S=0, R=0, Q=1, $\bar{Q}$=1

Truth Table

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

Case-II

S=1, R=0, Q=0, $\bar{Q}$=1

S=0, R=0, Q=1, $\bar{Q}$=1

case-III

S=1, R=1, Q=1, $\bar{Q}$=0

S=0, R=0, Q=1, $\bar{Q}$=1

Final Truth Table

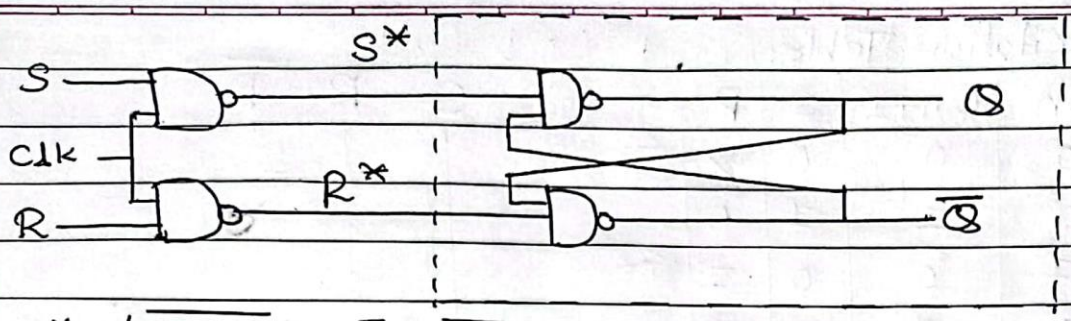
S	R	Q	$\bar{Q}$
0	0	Invalid	
0	1	1	0
1	0	0	1
1	1	Memory	

(2) SR or RS Flip-Flop

⇒ Truth Table for SR latch with NAND

S	R	Q	$\bar{Q}$
0	0	Invalid	
0	1	1	0
1	0	0	1
1	1	Memory	

MAYUR
Date
Page



$S^* = (\overline{clk \cdot S}) = \overline{S} + \overline{clk}$

$R^* = (\overline{clk \cdot R}) = \overline{R} + \overline{clk}$

Truth Table For SR Flip-Flop

clk	S	R	Q $\overline{Q}$
0	X	X	Memory
1	0	0	Memory
1	0	1	0 1
1	1	0	1 0
1	1	1	Invalid

When,

S=0 R=1 S^*=1 R^*= 0 → 0, 1

S=1 R=0 S^*=0 R^*= 1 → 1, 0

S=1 R=1 S^*=0 R^*=0 → Invalid

S=0 R=0 S^*=1 R^*=1 → Memory

Final Truth Table → Characteristics table

clk	S	R	Qn+1
0	X	X	Qn
1	0	0	Qn
1	0	1	0
1	1	0	1
1	1	1	Invalid

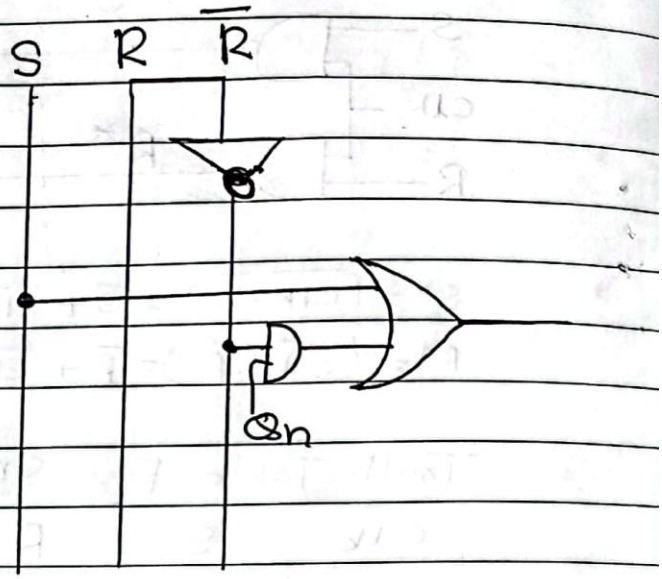
Qn	S	R	Qn+1
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	X

NAMAYUR 780X420 SINGLE LINE 200L 205X256 667 OK LDt 30-03-2

MAYUR
Date
Page

Excitation Table

Qn	Qn+1	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0



k-Map for Qn+1

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

$Q_{n+1} = S + Q_n \overline{R}$

(4) J k Flip-Flop

⇒ It is also called toggle flip-flop.

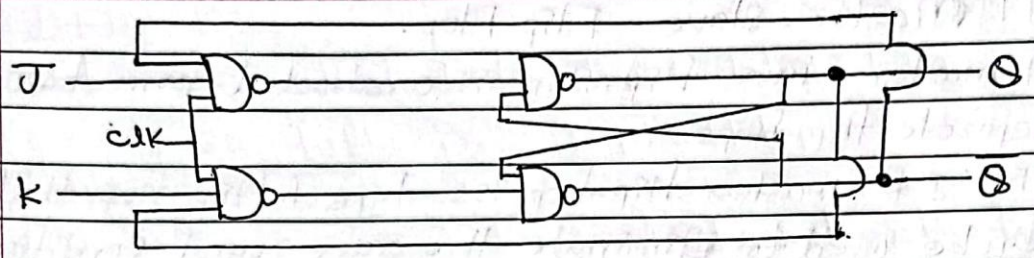
⇒ To overcome the invalid state of SR flip-flop JK flip-flop is used.

⇒ The JK flip flop is simplar to the SR flip-flop but there is no change in state when the J and k inputs are both low.

Truth table of SR Flip-Flop

Clk	S	R	Qn+1
0	X	X	Qn
1	0	0	Qn
1	0	1	0
1	1	0	1
1	1	1	Invalid → यो state माई valid बनाउन JK flipflop use गरिन्छ।

MAYUR
Date
Page



Assume, $Q=0$, $\overline{Q}=1$

When, $J=1$, $k=1$ & $clk=1$

$Q = 0 \; 1 \; 0$

$\overline{Q} = 1 \; 0 \; 1$ (Racing Condition / Toggle Condition)

Truth Table for J k Flip-Flop

clk	J	k	Q_{n+1}
0	X	X	Q_n
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	$\overline{Q_n}$

k-Map

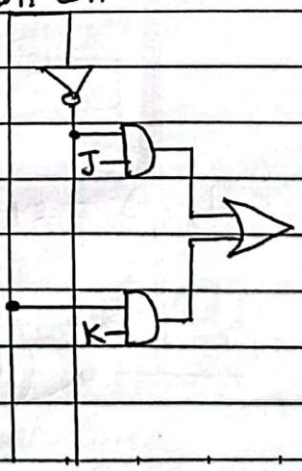
Q_n \ JK	00	01	11	10
0			1	1
1	1			1

$\therefore Q_{n+1} = \overline{Q_n} J + Q_n \overline{K}$

Characteristics Table → Excitation Table

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

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



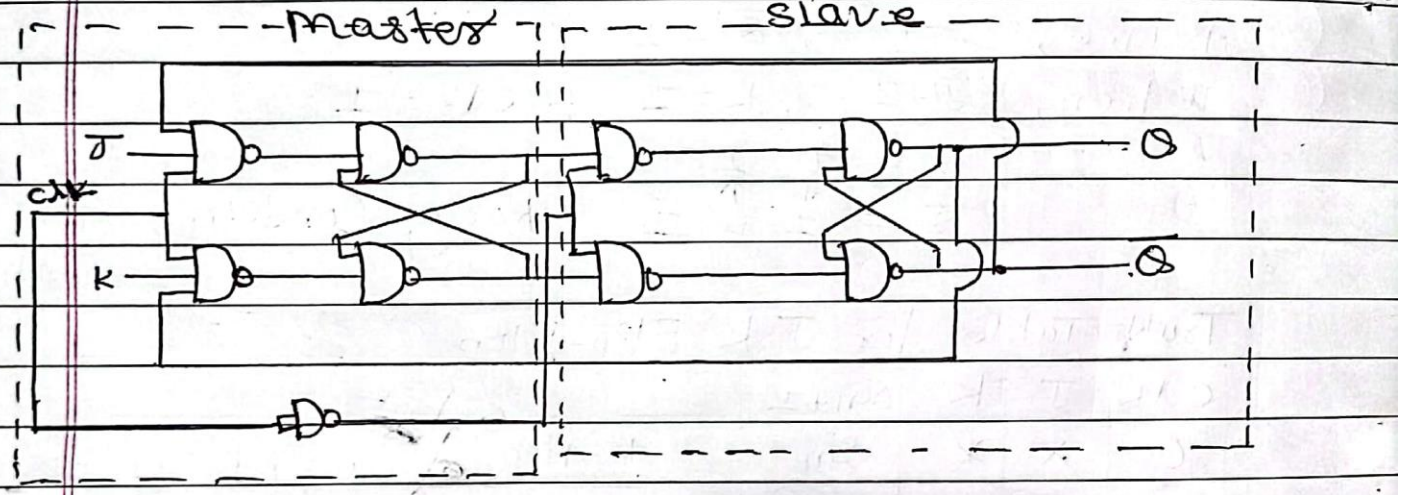
MAYUR 780X420 SINGLE LINE 2COL 205X256.667 OK / Dt. 30-03-2021

MAYUR
Date
Page

(5) JK Master-Slave Flip-Flop.

⇒ A master-slave flip-flop is constructed from two separate flip-flops.

⇒ The JK master flip-flop is a type of flip-flop that can be used to eliminate the race round condition.



MAYUR
Date
Page

(6) D Flip-Flop

⇒ It is also called Data Input Flip-Flop.

Truth table for SR Flip-Flop.

Clk	S	R	Qn+1
0	X	X	Qn
1	0	0	Qn
1	0	1	0
1	1	0	1
1	1	1	Invalid

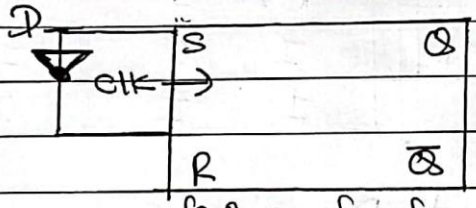


fig:- D-flip-flop

D=0, S=0, R=1

D=1, S=1, R=0

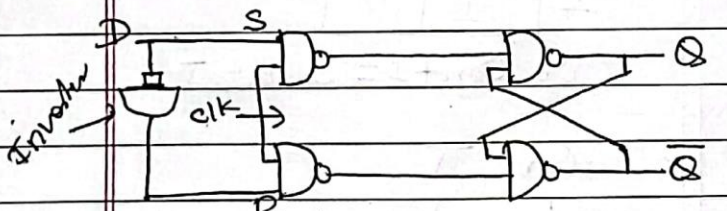


fig:- Circuit Diagram

Truth Table for D flip-flop

Clk	D	Qn+1
0	X	Qn
1	0	0
1	1	1

Characteristic table

Qn	D	Qn+1
0	0	0
0	1	1
1	0	0
1	1	1

Excitation Table

Qn	Qn+1	D
0	0	0
0	1	1
1	0	0
1	1	1

K-map for Qn+1

Qn \ D	0	1
0		1
1	1	

∴ $Q_{n+1} = Q_n \oplus D$

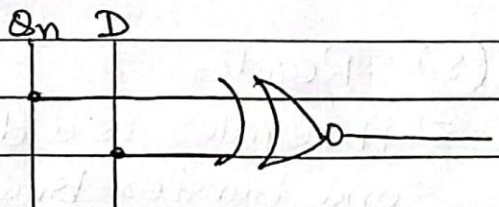


fig:- Circuit Diagram

MAYUR 780X420 SINGLE LINE 2COL 205X256.667 OK / Dt. 30-03-2021

(7) T- Flip-Flop

⇒ Also called Toggle flip-flop

⇒ A T flip-flop is a digital circuit that can store one bit of data and toggle its state between 0 and 1 based on the application of toggling input.

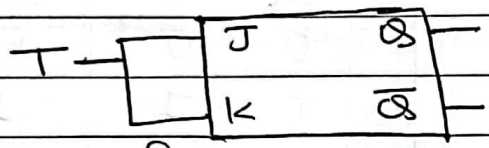


fig:- Block Diagram

Excitation Table

Qn	Qn+1	T
0	0	0
0	1	1
1	0	1
1	1	0

Truth Table for T flip-flop

clk	T	Qn+1
0	X	Qn
1	0	Qn
1	1	$\overline{Qn}$

Characteristic Table

Qn	T	Qn+1
0	0	0
0	1	1
1	0	1
1	1	0

K-Map for Qn+1

Qn \ T	0	1
0		1
1	1	

∴ $Qn+1 = Qn \oplus T$

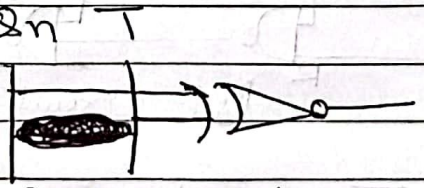


fig:- Circuit diagram.

(8) Register

⇒ A register is a digital circuit that is used to store and transfer binary data. It is made up of a group of flip-flops that are interconnected to form a storage element.

MAYUR
Date
Page

(9) Types of Register

(a) Serial In / ~~Parallel~~ Serial Out. (SISO)

⇒ A SISO register has one data input and one dat output. It stores one bit of data at a time and shifts it out serially.

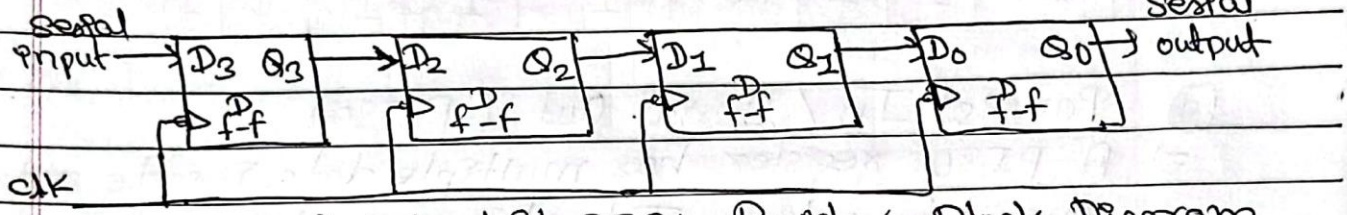


fig:- 4 bit SISO Register Block Diagram.

Truth table

Clk	D_3 D_2 D_1 D_0	Q_3	Q_2	Q_1	Q_0
Initially		0	0	0	0
1		1	0	0	0
1		1	1	0	0
1		1	1	1	0
1		1	1	1	1

(b) Serial In/Parallel out. (SIPO)

⇒ A SIPO register has one data input and multiple data outpus. It stores one bit of data at a time and shifts it out serially to all the output line in parallel.

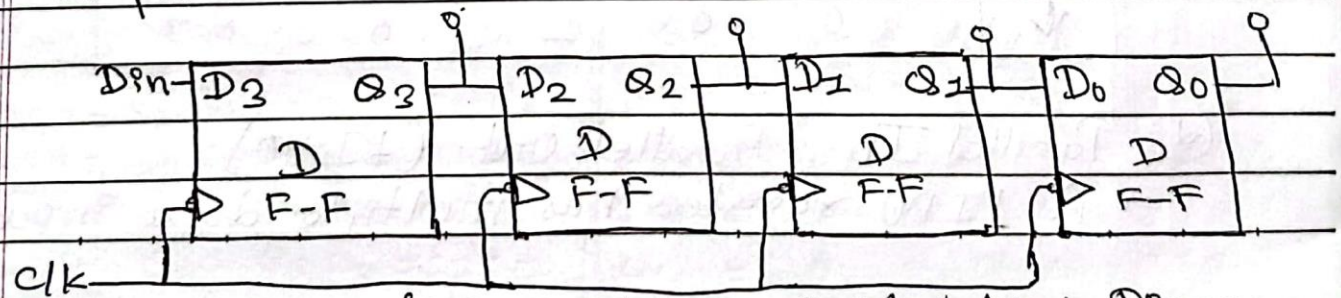


fig:- 4 bit SIPO Register Block Diagram.

MAYUR 780X420_SINGLE LINE_2COL_205X256.667_OK/ Dt. 30-03-2021

MAYUR
Date
Page

Truth Table

clk	D_3	D_2	D_1	D_0	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	0	0
↓	1	0	0	0	1	0	0	0
↓	0	1	0	0	0	1	0	0
↓	1	0	1	0	1	0	1	0
↓	1	1	0	1	1	1	0	1

(c) Parallel In / Serial Out (PISO)

⇒ A PISO register has multiple data inputs and one data output. It stores one bit of data at a time and shifts it out serially.

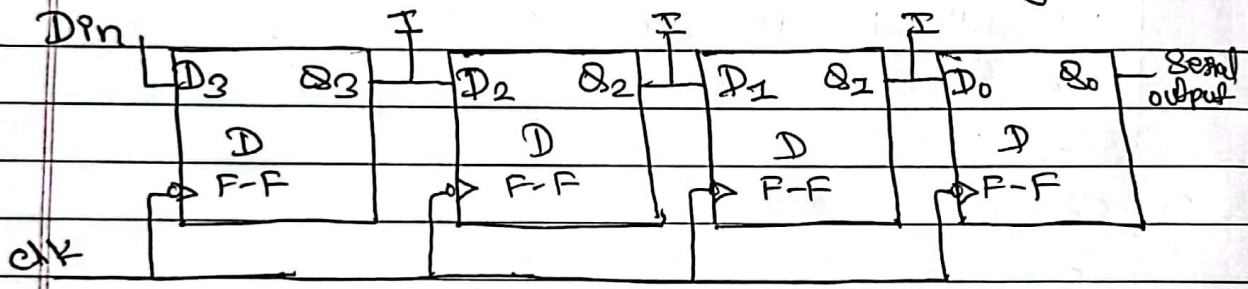


fig: Block Diagram

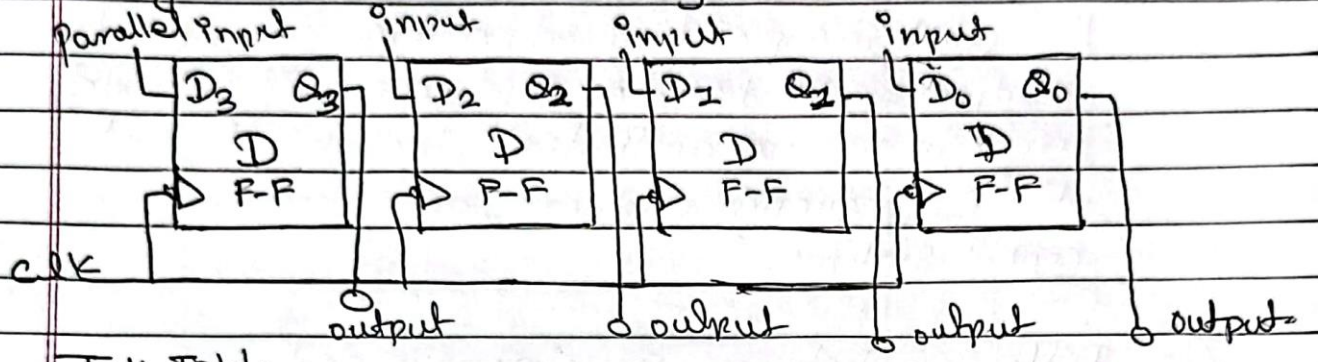
Truth Table

clk	D_3	D_2	D_1	D_0	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	0	0
↓	1	1	0	1	1	1	0	1
↓	0	1	1	0	0	1	1	0
↓	0	0	1	1	0	0	1	1
↓	0	0	0	1	0	0	0	1

(d) Parallel In / Parallel Out (PIPO)

⇒ A PIPO register has multiple data input

and multiple data output. PIPO registers are comm. -only used for data storage and transfer.



Truth Table

clk	D_3	D_2	D_1	D_0	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	0	0
1	1	1	0	1	1	1	0	1

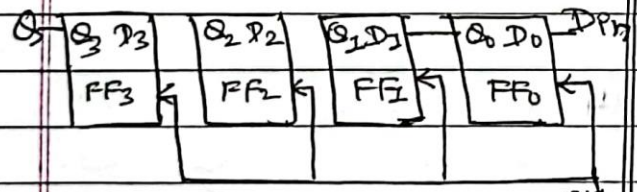
(10)

Shift Left Register

→ In a shift left Register, the bits stored in the register are shifted to the left.

→ Uses D flip-flop

→ 4 bit register uses 4 ff.



initially $Q_3Q_2Q_1Q_0 = 0000$

Let $D_{in} = 1$

1 cp = $Q_3Q_2Q_1Q_0 = 0001$

2 cp = $Q_3Q_2Q_1Q_0 = 0011$

3 cp = $Q_3Q_2Q_1Q_0 = 0111$

4 cp = $Q_3Q_2Q_1Q_0 = 1111$

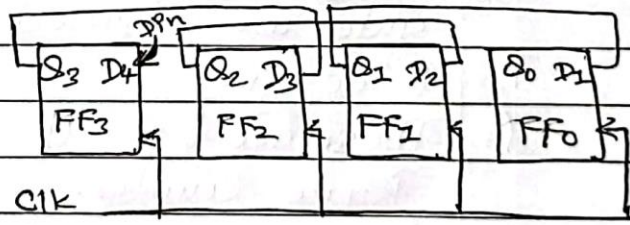
→ Stored word wouldn't change as long as $D_{in} = 1$

Shift Right Register

→ In a shift right register, the bits stored in the register are shifted to the right.

→ Uses D flip-flop.

→ 4 bit register use 4 ff.



$Q_3Q_2Q_1Q_0 = 0000$

1 CP = $Q_3Q_2Q_1Q_0 = 1000$

2 CP = $Q_3Q_2Q_1Q_0 = 1100$

3 cp = $Q_3Q_2Q_1Q_0 = 1110$

4 cp = $Q_3Q_2Q_1Q_0 = 1111$

→ as long as $D_{in} = 1$, Register will store same data

(11) Counter Defination

⇒ In a digital logic, a counter is a sequential circuit that counts in a predetermined sequence and generates an output signal based on that count. It is widely used in various applications, including digital clocks, frequency dividers, event counting and control systems.

(12) Difference between Synchronous and ripple Counter

	Synchronous Counter		Ripple Counter.
(1)	There is no connection between output of the first flip-flop and clock of next flip-flop.	(1)	Flip-Flop are connected in such a way that the output of first flip-flop driver the clock of next flip-flop.
(2)	All flip flop are clocked simultaneously.	(2)	Flip-flop are not clocked simultaneously.
(3)	Large number of logic gates are required to design.	(3)	Less number of Logic gates are required to design.
(4)	These are faster than Ripple Counter.	(4)	These are slower than Synchronous counter.
(5)	High Cost	(5)	Low cost.
(6)	Speed is high	(6)	Speed is slow.

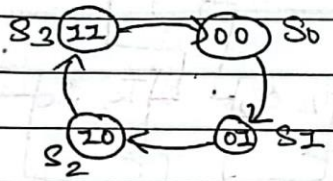
(13) 2 bit Synchronous up counter.

⇒ J K flip-flop

Excitation table for JK flipflop

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

State Diagram



Circuit Excitation table

Q_1	Q_2	Q_1	Q_2	J_1	K_1	J_2	K_2
0	0	0	1	0	X	1	X
0	1	1	0	1	X	X	1
1	0	1	1	X	0	1	X
1	1	0	0	X	1	X	1

k-Map for J_1

Q_1 \ Q_2	0	1
0		1
1	X	X

$\therefore J_1 = Q_2$

k-Map for K_1

Q_1 \ Q_2	0	1
0	X	X
1		1

$\therefore K_1 = Q_2$

k-map for J_2

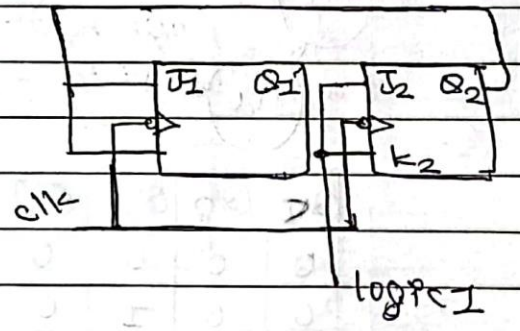
Q_1 \ Q_2	0	1
0		X
1	1	X

$\therefore J_2 = 1$

k-map for K_2

Q_1 \ Q_2	0	1
0	X	1
1	X	1

$\therefore K_2 = 1$



MAYUR 780X420 SINGLE LINE 2COL 205X256 667 (OK/LDL 30-03-2021

(14) 3 bit synchronous up Counter.

⇒ Excitation Table for T-flip-flop

Qn	Q(n+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

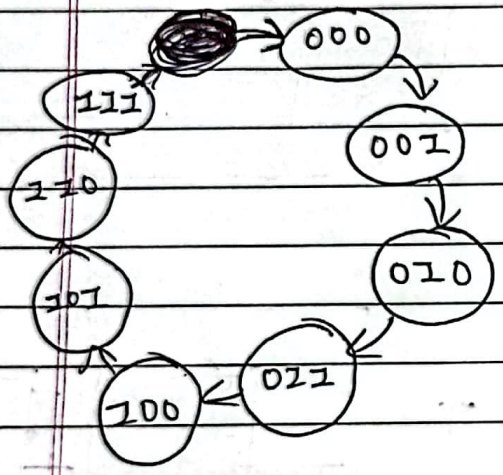
K-map for T_A

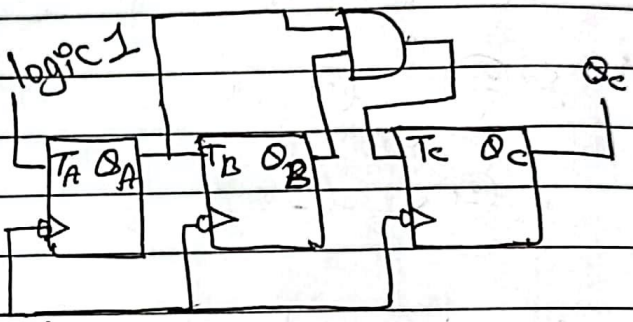
Q_C \ $Q_B Q_A$	00	01	11	10
0	1	1	1	1
1	1	1	1	1

∴ $T_A = 1$

∴ $T_B = Q_A$

∴ $T_C = Q_B Q_A$





Q_C	Q_B	Q_A	Q_C^T	Q_B^T	Q_A^T	T_C	T_B	T_A
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

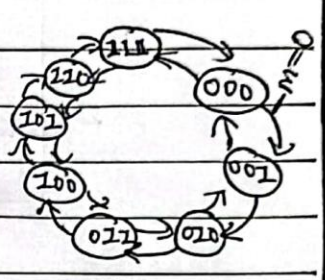
MAYUR
Date
Page

(15) 3 bit Synchronous up/down Counter.

⇒

M	Q_C	Q_B	Q_A	Q_C^*	Q_B^*	Q_A^*	T_C	T_B	T_A
0	0	0	0	0	0	1	0	0	1
0	0	0	1	0	1	0	0	1	1
0	0	1	0	0	1	1	0	0	1
0	0	1	1	1	0	0	1	1	1
0	1	0	0	1	0	1	0	0	1
0	1	0	1	1	1	0	0	1	1
0	1	1	0	1	1	1	0	0	1
0	1	1	1	0	0	0	1	1	1
1	0	0	0	1	1	1	1	1	1
1	0	0	1	0	0	0	0	0	1
1	0	1	0	0	0	1	0	1	1
1	0	1	1	0	1	0	0	0	1
1	1	0	0	0	1	1	1	1	1
1	1	0	1	1	0	0	0	0	1
1	1	1	0	1	0	1	0	1	1
1	1	1	1	1	1	0	0	0	1

M = 0 → up Counter
M = 1 → down Counter



excitation table of T-flipflop

Q_n	$Q_{(n+1)}$	T
0	0	0
0	1	1
1	0	1
1	1	0

K-map for T_C

M Q_C \ $Q_B Q_A$	00	01	11	10
00			1	
01			1	
11	1			
10	1			

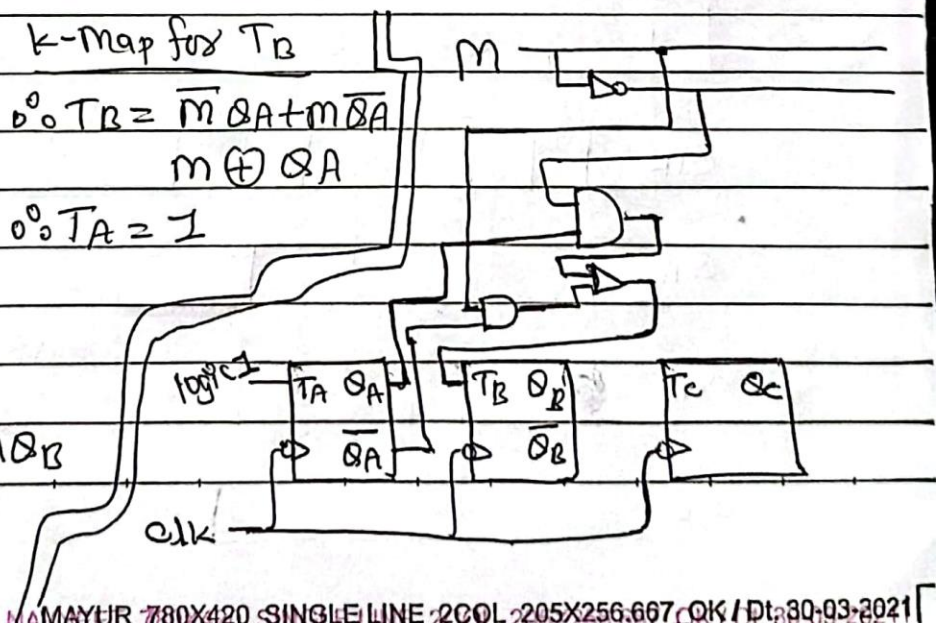
$T_C = M\overline{Q_B}Q_A + M Q_A Q_B$

K-map for T_B

$T_B = \overline{M} Q_A + M\overline{Q_A}$

$M \oplus Q_A$

$T_A = 1$

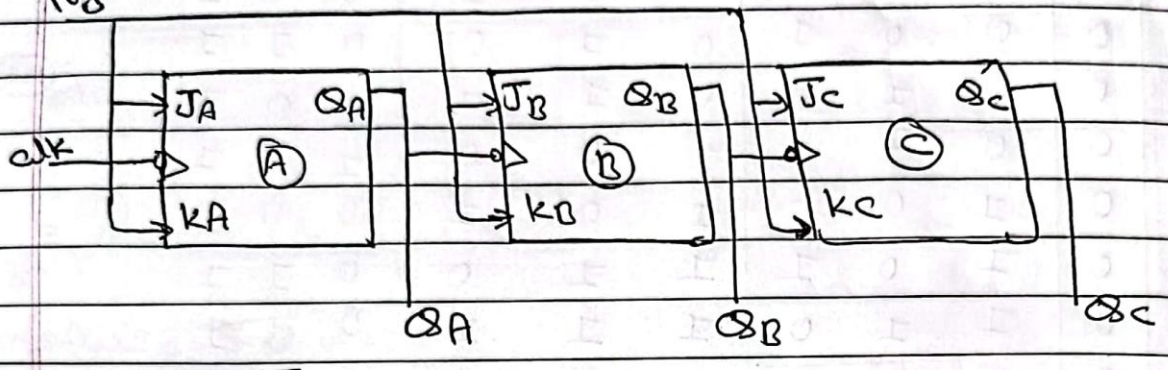


MAYUR 780X420 SINGLE LINE 2COL 205X256.667 OK / Dt: 30-03-2021

MAYUR
Date
Page

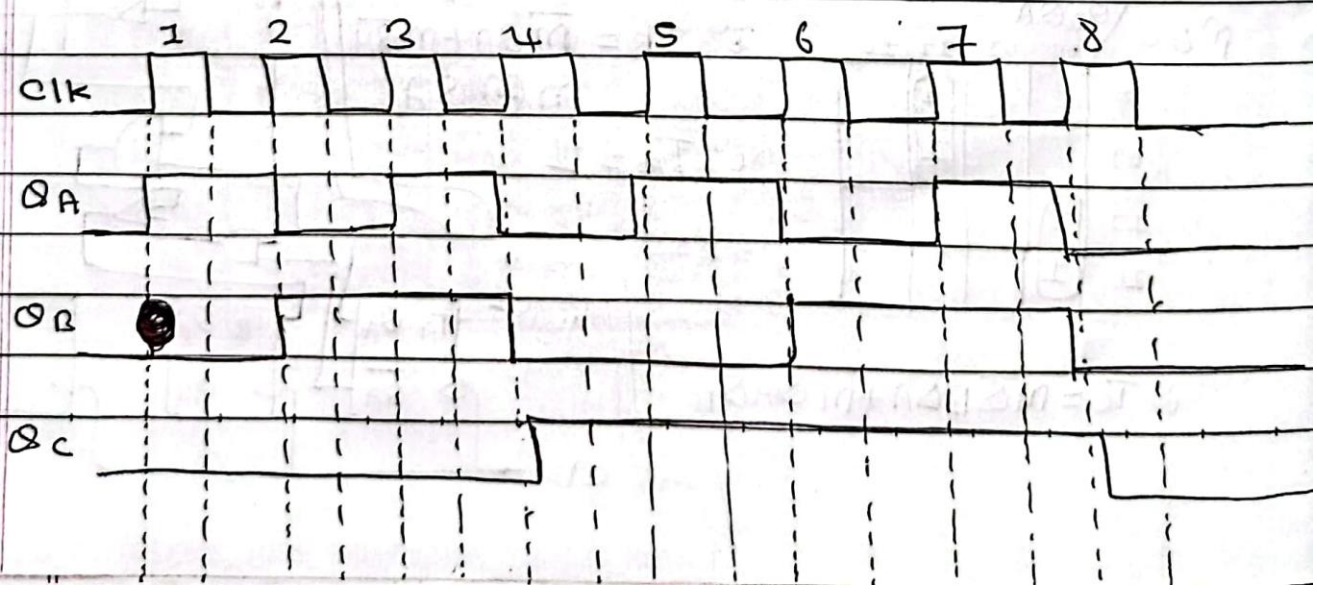
(16) 3 bit Asynchronous Counter (Ripple Counter)

⇒ logic



Truth Table

clk	Qc	QB	QA	Decimal
Initially	0	0	0	0
↓	0	0	1	1
↓	0	1	0	2
↓	0	1	1	3
↓	1	0	0	4
↓	1	0	1	5
↓	1	1	0	6
↓	1	1	1	7
↓	0	0	0	0



(17) MOD-7 Ripple Counter (Asynchronous Counter)

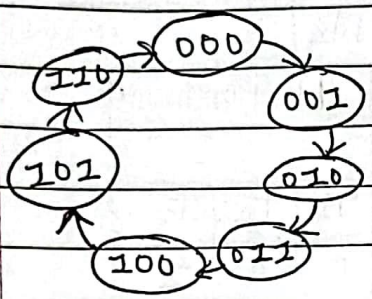
⇒ MOD-7 Ripple Counter

No. of clock = 7

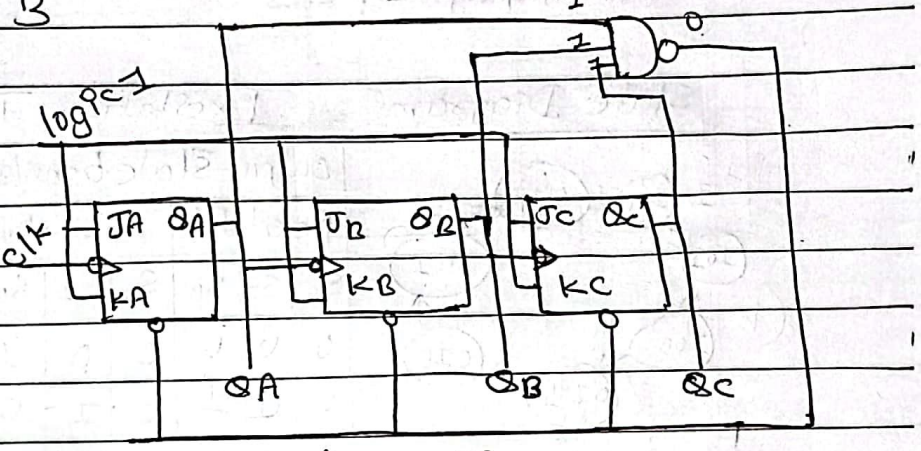
Range of Counting = 0 - 6

No. of flip flop = 3

1 1 1
Q_A Q_B Q_C



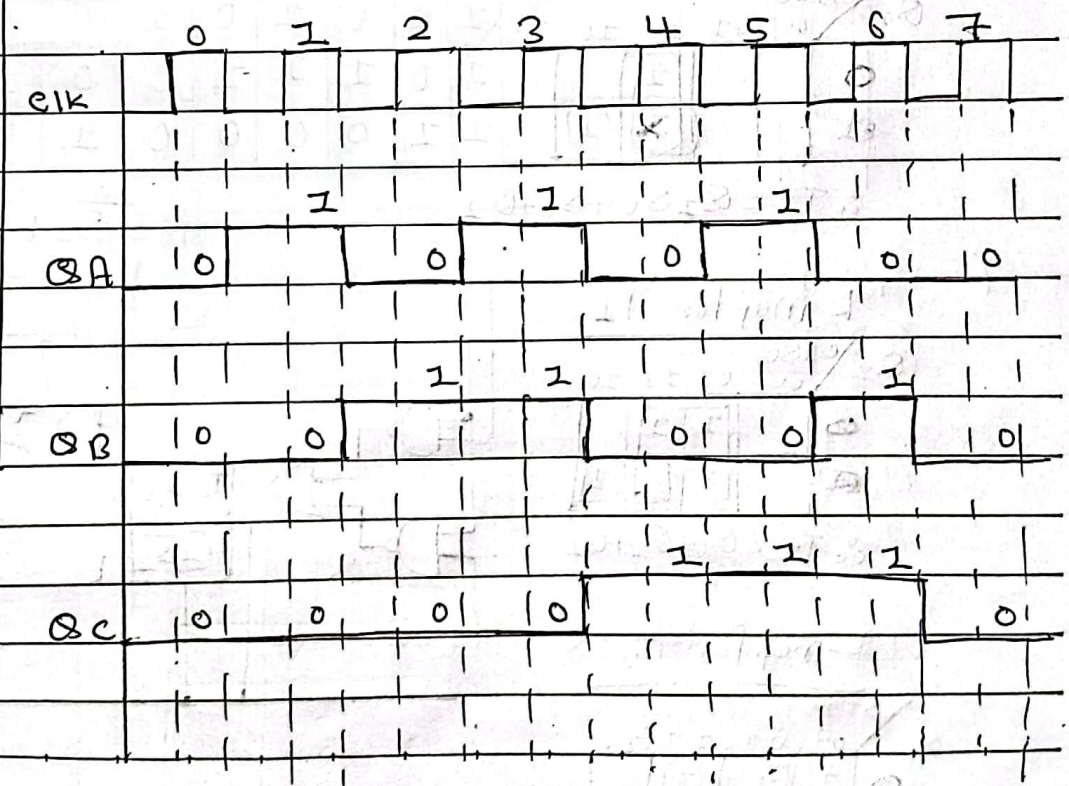
State Diagram



Logic Diagram

clk	Q_C	Q_B	Q_A
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	0	0	0

Truth table



Timing diagram

MAYUR 780X420 SINGLE LINE 2COL 205X256 687 OK / Dt 30-03-2021

(18) MOD-7 Synchronous Counter.

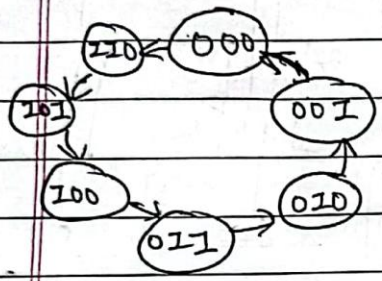
⇒ MOD-7 counter

Count. N=7

Range of Counting = 0 to N-1 = 0 to 6

No. of flipflop = 3

State Diagram



Excitation Table

output state transitions						flipflop inputs		
Present state			Next state					
S_2	S_1	S_0	S_2	S_1	S_0	T_1	T_2	T_3
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	0	0	0	1	1	0

k-map for T_2

Q_2 \ Q_1Q_0	00	01	11	10
0			1	
1			X	1

$\therefore T_2 = Q_1Q_0 + Q_2 + Q_1$

k-map for T_1

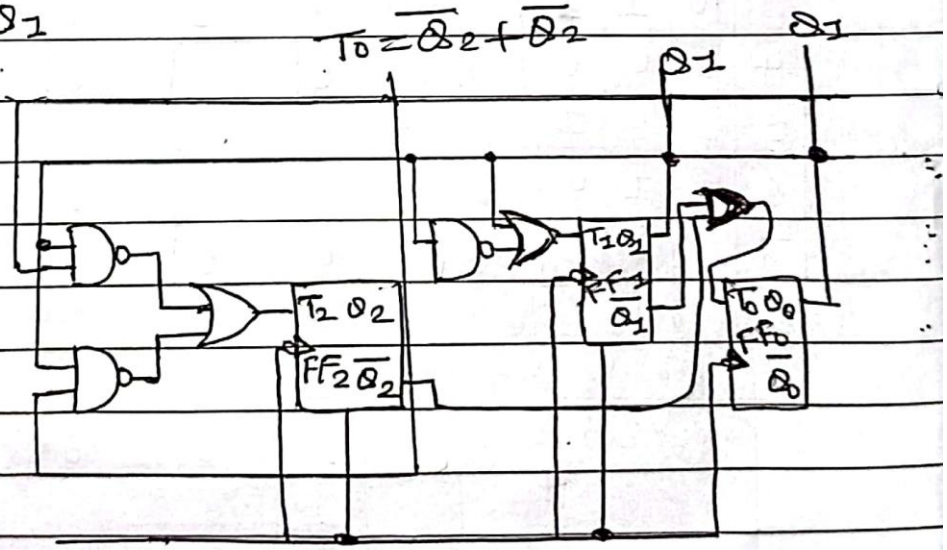
Q_2 \ Q_1Q_0	00	01	11	10
0		1	1	
1		X	X	1

$\therefore T_1 = Q_0 + Q_2 + Q_1$

k-map for T_0

Q_2 \ Q_1Q_0	00	01	11	10
0	1	1	1	
1	1	1	X	0

$T_0 = \overline{Q_2} + \overline{Q_2}$



MAYUR
Date
Page

(19) Basic Logic Gates AND, OR, NOT.

⇒ (a) AND Gates.

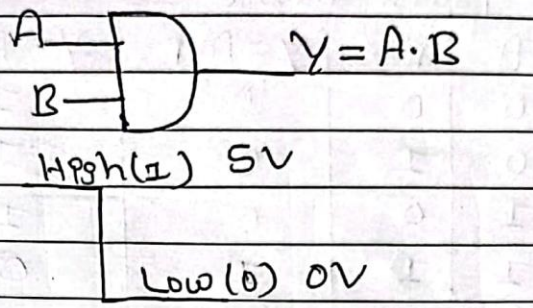
⇒ $Y = A \cdot B$

$2^2 = 4$

Truth Table

A	B	$Y = A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

Circuit Diagram



(b) OR Gate

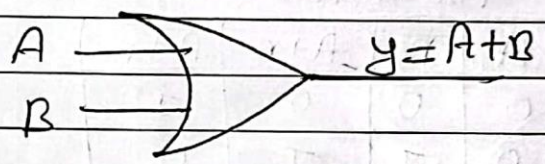
⇒ $Y = A + B$

$2^2 = 4$

Truth Table

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

Circuit Diagram



(c) OR Gate

⇒ Converts high input (1) to low (0) and vice-versa.

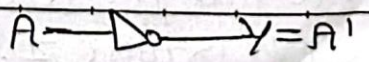
$Y = A'$

$2^1 = 2$

Truth table

A	$Y = A'$
0	1
1	0

Circuit Diagram



MAYUR
Date
Page

(20) Universal Logic gates NOR and NAND.

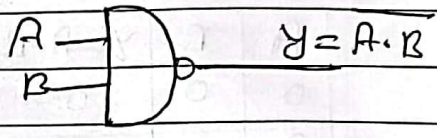
(a) NAND. Gates. (NOT+AND)

⇒ $Y = \overline{A \cdot B}$

Circuit Diagram

Truth Table

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



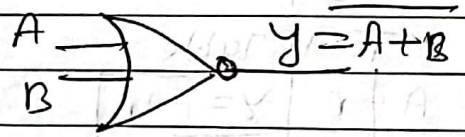
(b) NOR Gates. (NOT+OR)

⇒ $Y = \overline{A+B}$

Truth Table

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

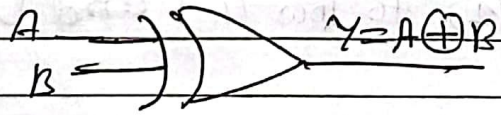
Circuit Diagram



(21) Ex-OR and Ex- NOR Gates.

(a) Exclusive OR (X-OR) ($A \oplus B$)

⇒ Circuit Diagram



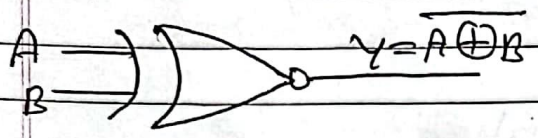
If input same then output is always 0. otherwise 1.

Truth Table

A	B	$Y = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

(b) Exclusive (NOR = X-NOR) $A \oplus B$

⇒ Circuit Diagram



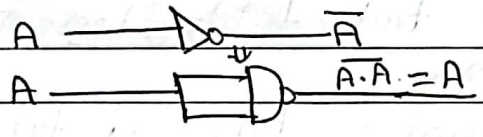
If input is same then output is always 1 Otherwise 0.

Truth Table

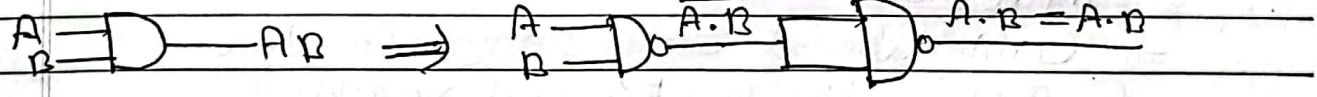
A	B	$Y = \overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

(22) Implement AND, OR and NOT gates using NAND gates.

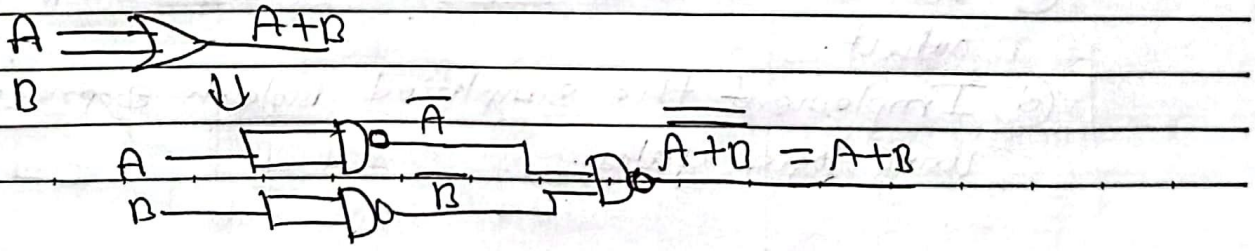
(a) NAND Gates as NOT Gate.



(b) NAND gates as AND.



(c) NAND Gates as OR.



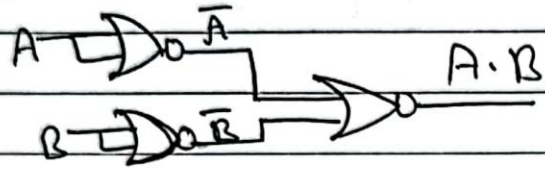
MAYUR 780X420 SINGLE LINE 2COL 205X256:667 OK / Dt: 30-03-2021

MAYUR
Date
Page

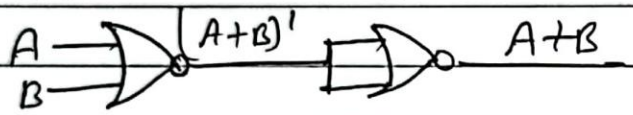
(23) Implement AND, OR, NOT using NOR.

(a) NOT using NOR gates.

(b) AND using NOR gates.



(c) OR gates using NOR gates.



(24) Define Combinational Circuit. Write combination circuit design procedure. Design Half Adder and Full Adder with its truth table, Logic Diagram and Boolean Expression.

⇒ A combinational circuit is a type of digital circuit where the output is only determined by the current input state..

⇒ Combinational circuit Design Procedure.

(a) Define the input/output variables.
(b) Create a truth table for the desired function.
(c) Derive the simplified Boolean expression for the output.
(d) Implement the simplified Boolean expression using logic gates.

MAYUR
Date
Page

⇒ Half Adder :- A half adder is a combinational circuit that can add two binary digits (bits) and produce the sum and carry output.

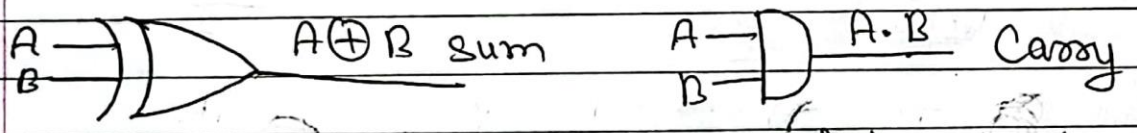
Truth Table

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

Based on truth table Boolean Expression for sum and carry are:- Sum = $A \oplus B$

Carry = $A \cdot B$

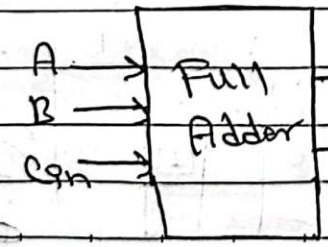
Logical Diagram



So, Half adder can be implemented using the following Boolean Expression :- Sum = $A \oplus B$ = A XOR B.

Carry = $A \cdot B$ = A AND B.

⇒ Full Adder :- A full adder is a combinational Logic circuit that adds three inputs and two outputs. The first two input is A and B and another input is Cin (Carry In) and the two output is sum and carry out (C-out).



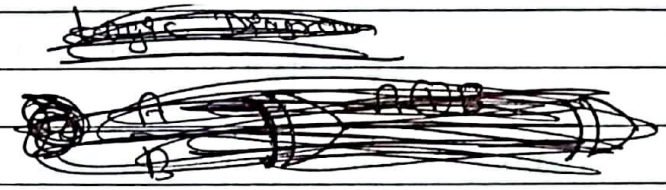
Expression

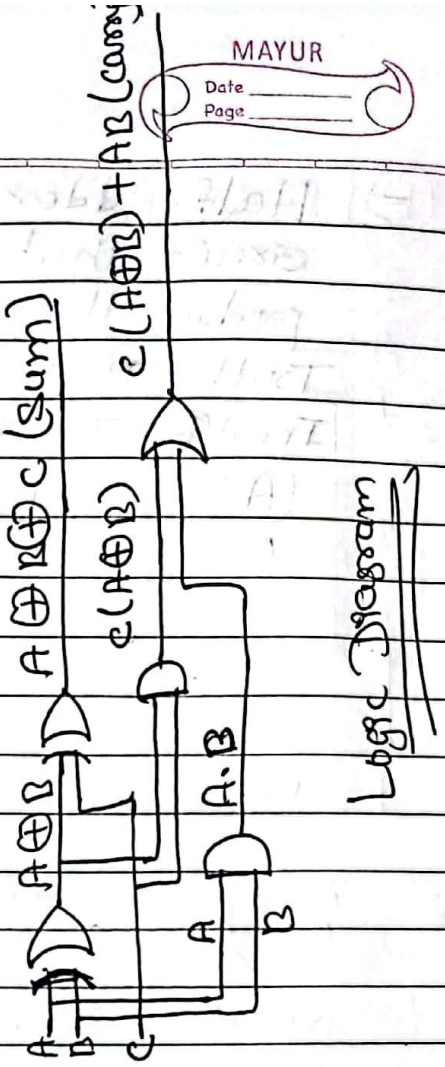
Sum = $A \oplus B \oplus C$

Carry = $C(A \oplus B) + AB$

Truth Table.

Inputs			Outputs	
A	B	C-in	Sum	C-out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

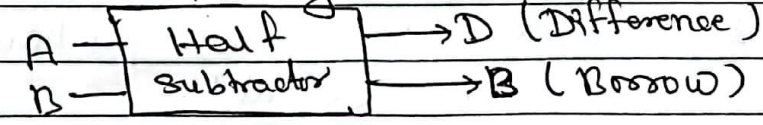




(25) Half Subtracter and Full Subtractor.

(a) Half Subtractor

⇒ It is a combinational circuit used to subtract two bit binary no.



Block Diagram

Truth table

A	B	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1

K-map for D

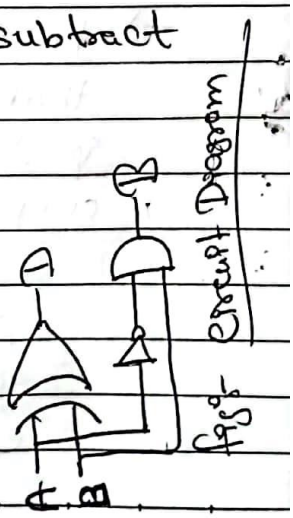
A \ B	0	1
0		1
1	1	

$\therefore D = A \oplus B$

K-map for B

A \ B	0	1
0		1
1		

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



(b) Full Subtractor

⇒ It is a combinational circuit used to subtract three bit binary no.

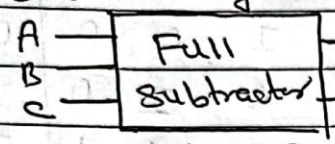


fig:- Block Diagram

Truth table

A	B	C	D	B
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-map for D

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

$D = A \oplus B \oplus C$

K-map for B

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

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

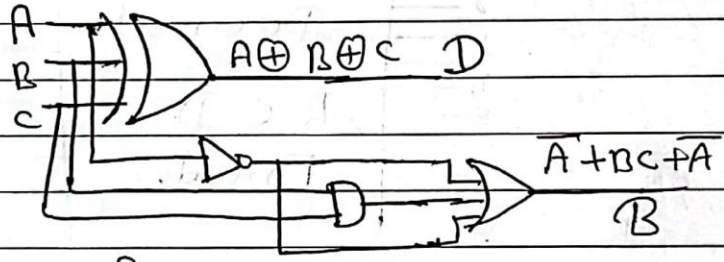


fig:- Circuit Diagram

(26) Define Encoder. Design 4X2 Encoder.

⇒ It is a combinational circuit that perform reverse operation of decoder.

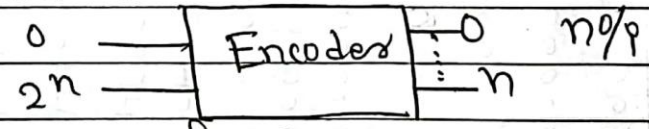
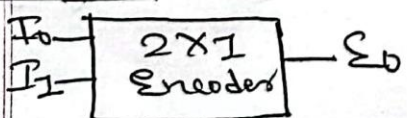


fig:- Encoder.

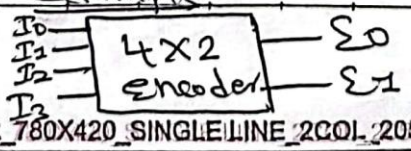
→ only one input active at a time.

→ for any 2^n inputs there are 'n' output line.

Example.



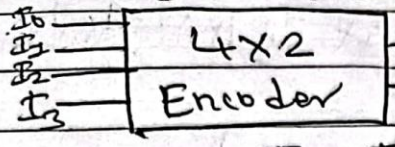
Example



MAYUR_780X420_SINGLELINE_2COL_205X256.667 OK / Dt: 30-03-2021

MAYUR
Date
Page

⇒ Design 4X2 Encoder.



Truth Table

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

$E_0 = I_2 + I_3$

$E_1 = I_1 + I_3$

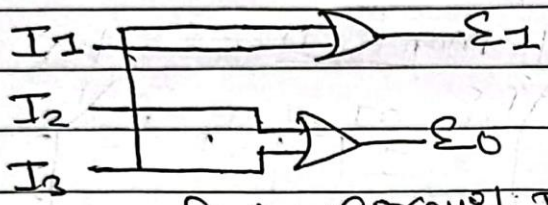


Fig:- Circuit Diagram

(27) Design Decimal to BCD Encoder.

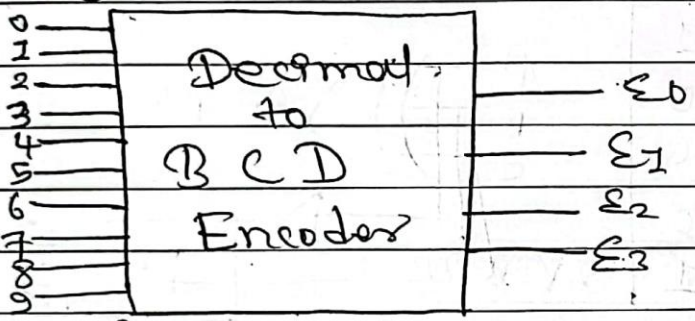


Fig:- Decimal to BCD Encoder.

I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	E_0	E_1	E_2	E_3
1	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	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1
x	x	x	x	x	x	x	x	x	x	1	0	1	0
x	x	x	x	x	x	x	x	x	x	1	0	1	1
x	x	x	x	x	x	x	x	x	x	1	1	0	0
x	x	x	x	x	x	x	x	x	x	1	1	0	1
x	x	x	x	x	x	x	x	x	x	1	1	1	0
x	x	x	x	x	x	x	x	x	x	1	1	1	1

$E_0 = I_1 + I_3 + I_5 + I_7 + I_9$

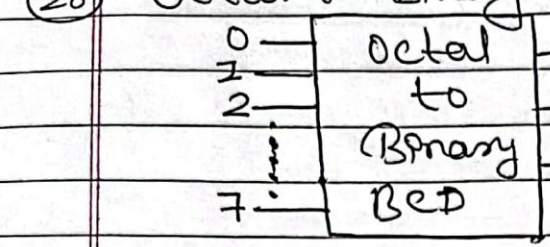
$E_1 = I_2 + I_3 + I_6 + I_7$

$E_2 = I_4 + I_5 + I_6 + I_7$

$E_3 = I_8 + I_9$

MAYUR
Date
Page

(28) Octal to Binary Encoder.



I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	E_0	E_1	E_2
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

$\therefore E_0 = I_4 + I_5 + I_6 + I_7$

$\therefore E_1 = I_2 + I_3 + I_6 + I_7$

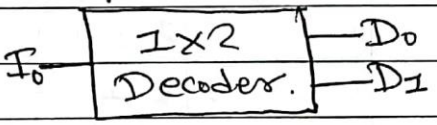
$\therefore E_2 = I_1 + I_3 + I_5 + I_7$

(29) Define Decoder. Design 2×4 Decoder.

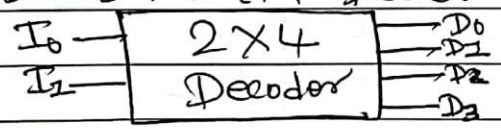
⇒ A decoder is a combinational circuit that detects the presence of particular input.

In Decoder, there is 'n' no. of input lines and at most "2^n" no. of output line.

Example:



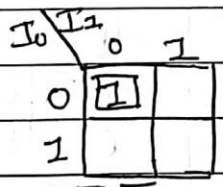
⇒ Design 2×4 Decoder.



Truth table

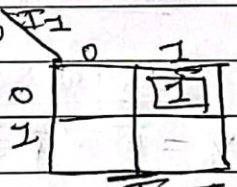
I_0	I_1	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

k-map for D_0



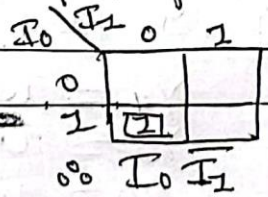
$\therefore \bar{I_0}\bar{I_1}$

k-map for D_1



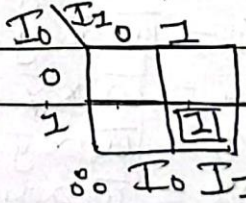
$\therefore \bar{I_0} I_1$

k-map for D_2

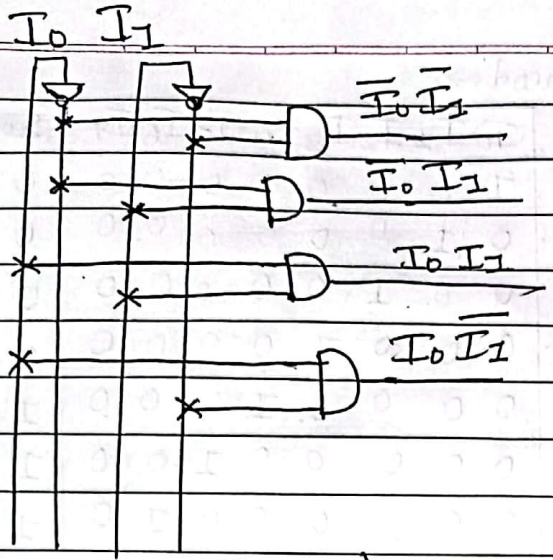


$\therefore I_0 \bar{I_1}$

k-map for D_3

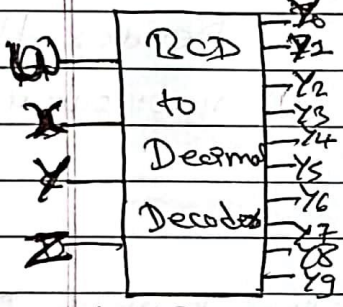


$\therefore I_0 I_1$



(20) Design BCD to Decimal Decoder.

⇒ A decoder that takes a 4-bit BCD as input code and produces 10 outputs corresponding to decimal digit is called a BCD to Decimal Decoder.



Block Diagram

Decimal	Inputs				Outputs									
	W	X	Y	Z	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	1	0	0	0	0	0	0	0
3	0	0	1	1	0	0	0	1	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0	1	0	0	0	0	0
5	0	1	0	1	0	0	0	0	0	1	0	0	0	0
6	0	1	1	0	0	0	0	0	0	0	1	0	0	0
7	0	1	1	1	0	0	0	0	0	0	0	1	0	0
8	1	0	0	0	0	0	0	0	0	0	0	0	1	0
9	1	0	0	1	0	0	0	0	0	0	0	0	0	1

Logical Expression

$Y_0 = \bar{w}\bar{x}\bar{y}\bar{z}$

$Y_1 = \bar{w}\bar{x}\bar{y}z$

$Y_2 = \bar{w}\bar{x}y\bar{z}$

$Y_3 = \bar{w}\bar{x}yz$

$Y_4 = \bar{w}x\bar{y}\bar{z}$

$Y_5 = \bar{w}x\bar{y}z$

$Y_6 = \bar{w}xy\bar{z}$

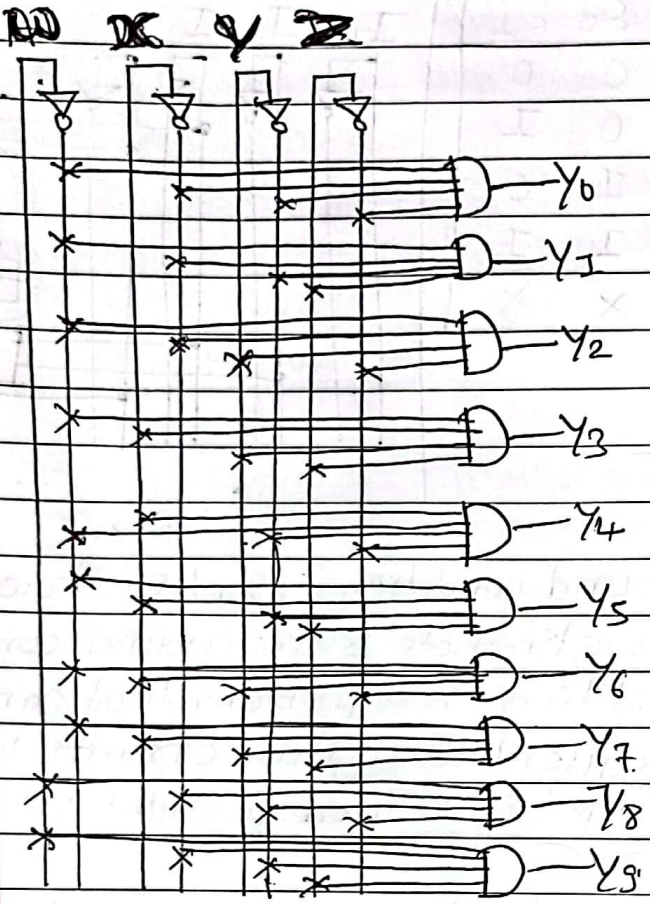
$Y_7 = \bar{w}xyz$

$Y_8 = w\bar{x}\bar{y}\bar{z}$

$Y_9 = w\bar{x}\bar{y}z$

MAYUR
Date
Page

Logical Diagram :-



(31) Define Priority Encoder. Design 4:2 priority encoder with its block diagram, truth table, circuit diagram and mathematical expression.

⇒ A priority encoder is a digital circuit that ~~encodes~~ encoder multiple binary inputs into a smaller number of output bits, where the output represents the highest priority input.

⇒ 4:2 priority encoder.

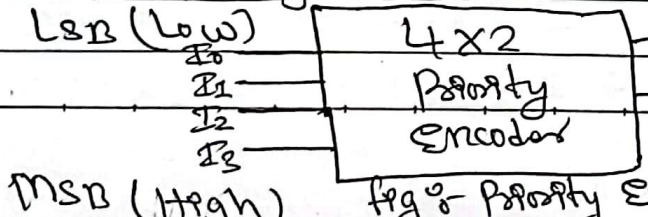


fig :- Priority Encoder

MAYUR 780X420 SINGLE LINE 2COL 205X256

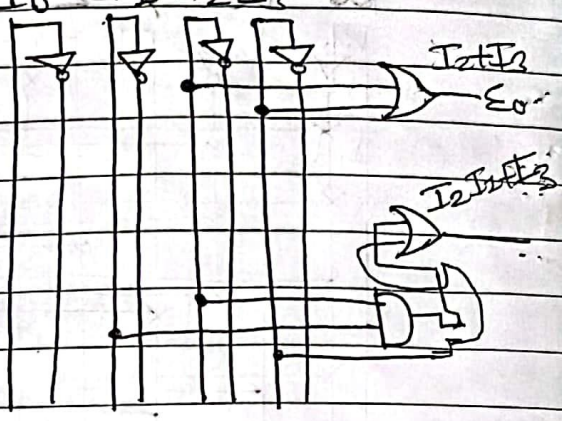
⇒ Truth Table

I_3	I_2	I_1	I_0	E_0	E_1
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1
0	X	X	X	X	X

$E_0 = I_2 + I_3$

$E_1 = I_2 I_1 + I_3$

⇒ Circuit Diagram



(32) What is race around condition? Explain how JK flip-flop is used to eliminate race around condition.

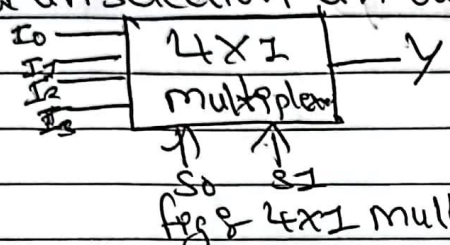
⇒ Race-around condition is a problem that can occur in synchronous sequential digital circuits when the output of a flip-flop is used to control its input.

One way to eliminate the race around condition is by using a J.K. master flip-flop, which has two input (J and k) and two output (Q and $\bar{Q}$). The JK flip-flop has a toggle function, meaning that it will change its output state whenever its inputs are different.

(33) Multiplexer. Design 2X1 Mux.

⇒ Multiplexer is a combinational circuit that select the binary information from one input line and direct it to output line.

⇒ It is a combinational circuit which has 2^n input line and an selection an output lines.



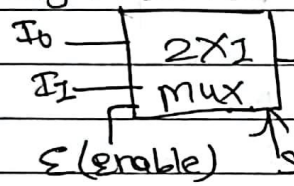
S → Selection line.

fig: 4X1 multiplexer

⇒ Advantages of multiplexer are:

(a) It reduce no. of wire.
(b) It reduce no. of gate.
(c) It reduce cost and circuit complexity.

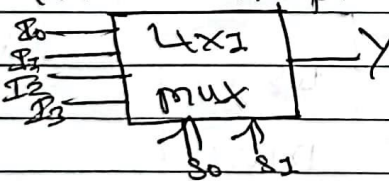
⇒ Design 2X1 multiplexer.



E	S	Y
0	X	0
1	0	I_0
1	1	I_1

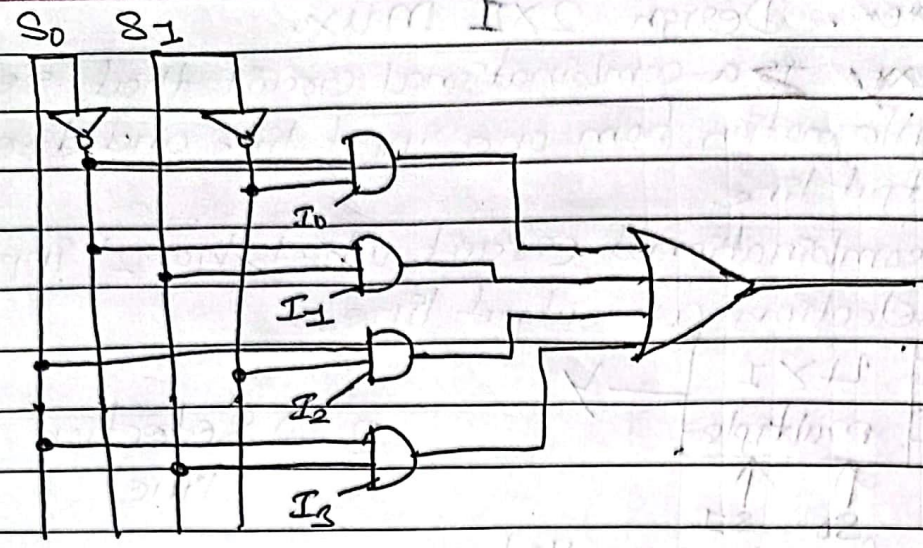
$y = E(I_0\bar{S} + I_1 S)$

(34) 4X1 Multiplexer.



S_0	S_1	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

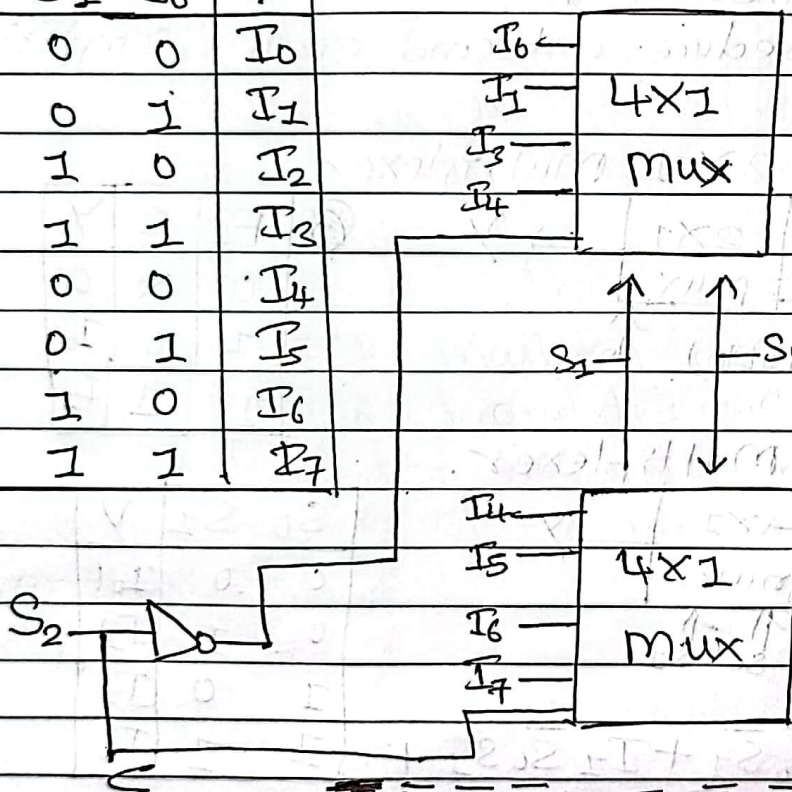
$\therefore y = I_0\bar{S}_0\bar{S}_1 + I_1\bar{S}_0 S_1 + I_2 S_0\bar{S}_1 + I_3 S_0 S_1$



(Q5) Design 8X1 Multiplexer using 4X1 multiplexer.

Truth Table of 8X1 mux

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

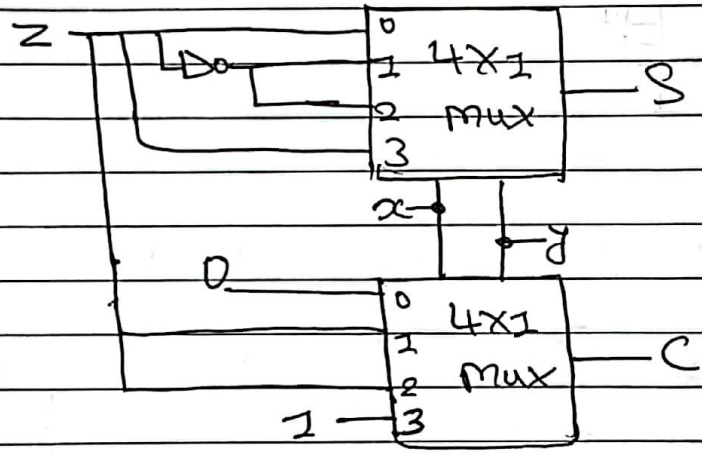


$$y = I_0 S_2 \bar{S}_1 \bar{S}_0 + I_1 \bar{S}_2 \bar{S}_1 S_0 + I_2 \bar{S}_0 S_1 \bar{S}_0 + I_3 S_1 \bar{S}_1 S_0 + I_4 S_2 \bar{S}_1 \bar{S}_0 + I_5 S_2 \bar{S}_1 S_0 + I_6 S_2 S_1 \bar{S}_0 + I_7 S_2 S_1 S_0$$

MAYUR
Date
Page

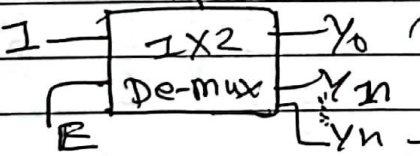
(36) Implement full adder using two 4x1 multiplexer.

x	y	z	S	C	
0	0	0	0	0	} S=z, C=0 (0)
0	0	1	1	0	
0	1	0	1	0	} S=$\bar{z}$, C=z (1)
0	1	1	0	1	
1	0	0	1	0	} S=$\bar{z}$, C=z (2)
1	0	1	0	1	
1	1	0	0	1	} S=z, C=1 (3)
1	1	1	1	1	



Block Diagram.

(37) De-Multiplexer. Design 1x2 De-multiplexer.



→ Reverse of Multiplexer
→ Always on I/p and many O/ps.

Fig:- De-multiplexer.

⇒ Design 1x2 De-mux.

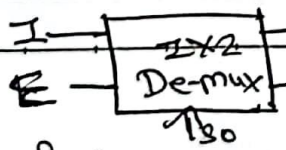
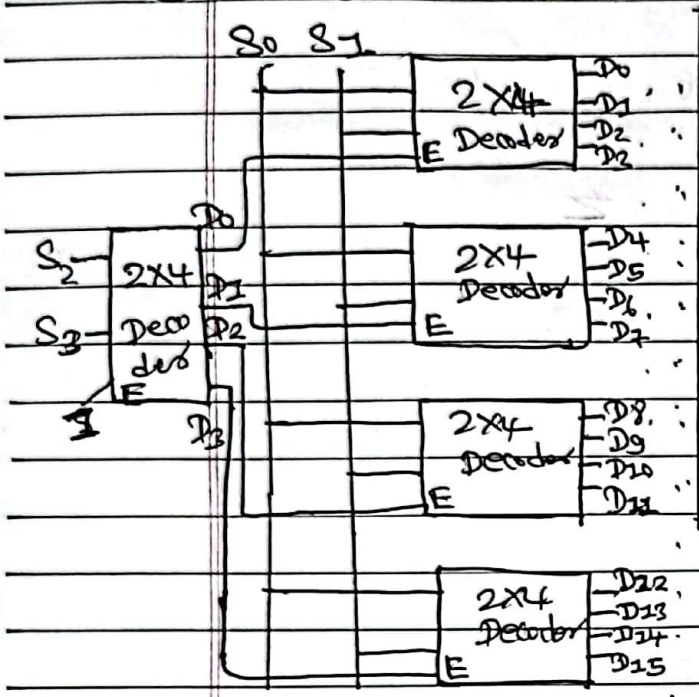


Fig:- 1x2 De-mux

E	S	Y_0	Y_1	
0	X	0	0	
1	0	I	0	∴ $Y_0 = E\bar{S}I$
1	1	0	I	∴ $Y_1 = E.SI$

MAYUR_780X420_SINGLE LINE_2COL_205X256.667 OK/DL 30-03-2021

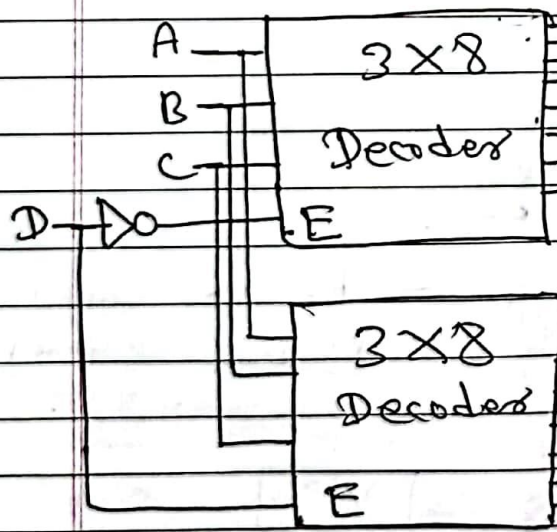
(37) Design 4X16 Decoder using 2X4 Decoder.



A	B	C	D	output
0	0	0	0	D0 = 1
0	0	0	1	D1 = 1
0	0	1	0	D2 = 1
0	0	1	1	D3 = 1
0	1	0	0	D4 = 1
0	1	0	1	D5 = 1
0	1	1	0	D6 = 1
0	1	1	1	D7 = 1
1	0	0	0	D8 = 1
1	0	0	1	D9 = 1
1	0	1	0	D10 = 1
1	0	1	1	D11 = 1
1	1	0	0	D12 = 1
1	1	0	1	D13 = 1
1	1	1	0	D14 = 1
1	1	1	1	D15 = 1

E - Enable.

(38) Design 4X16 Decoder using 3X8 Decoder.



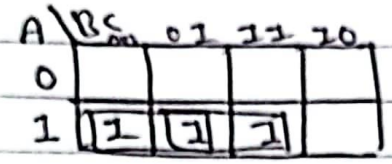
D	A	B	C	output
0	0	0	0	D0 = 1
0	0	0	1	D1 = 1
0	0	1	0	D2 = 1
0	0	1	1	D3 = 1
0	1	0	0	D4 = 1
0	1	0	1	D5 = 1
0	1	1	0	D6 = 1
0	1	1	1	D7 = 1
1	0	0	0	D8 = 1
1	0	0	1	D9 = 1
1	0	1	0	D10 = 1
1	0	1	1	D11 = 1
1	1	0	0	D12 = 1
1	1	0	1	D13 = 1
1	1	1	0	D14 = 1
1	1	1	1	D15 = 1

(38) Define PLA with its block diagram. Realize BCD to gray code converter using PLA.

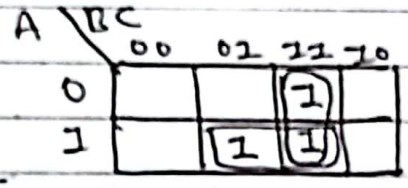
⇒ PLA stands for Programmable Logic Array, which is a type of digital circuit that can be programmed to implement any combinational logic function.

It consists of a programmable AND array, a programmable OR array, and a set of output gates.

A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

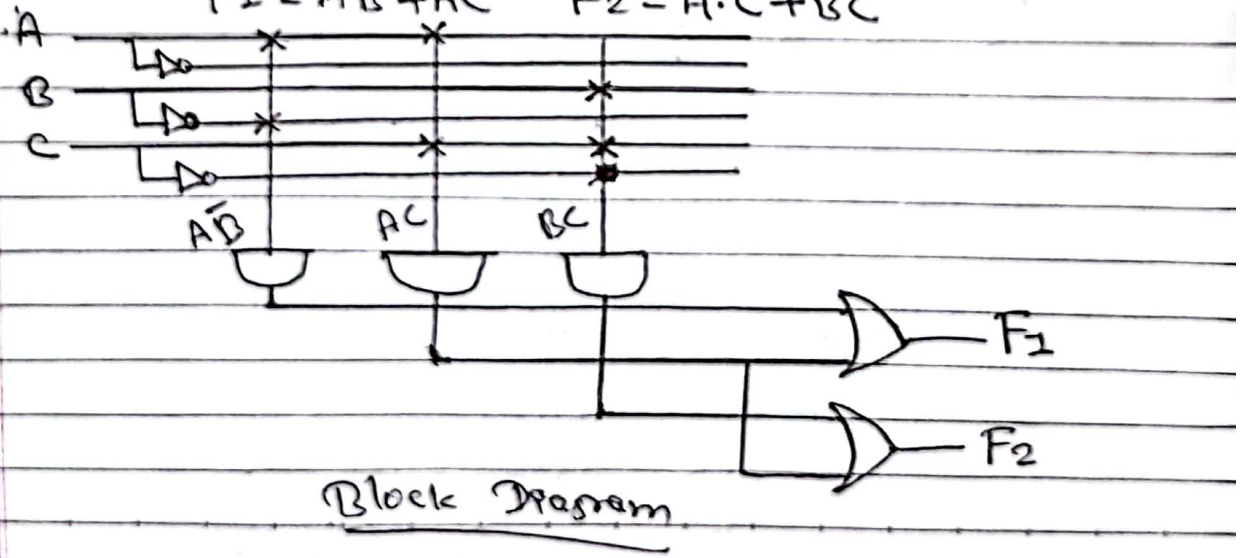


$F_2 = (A\bar{B} + AC$



$F_2 = (A \cdot C) + B \cdot C$

$F_1 = A\bar{B} + AC$ $\quad F_2 = A \cdot C + BC$



Block Diagram

MAYUR
Date
Page

(38) Define PLA with its block diagram. Realize BCD to gray code converter using PLA.

⇒ PLA stands for Programmable Logic Array, which is a type of digital circuit that can be programmed to implement any combinational logic function.

It consists of a programmable AND array, a programmable OR array, and a set of output gates.

A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

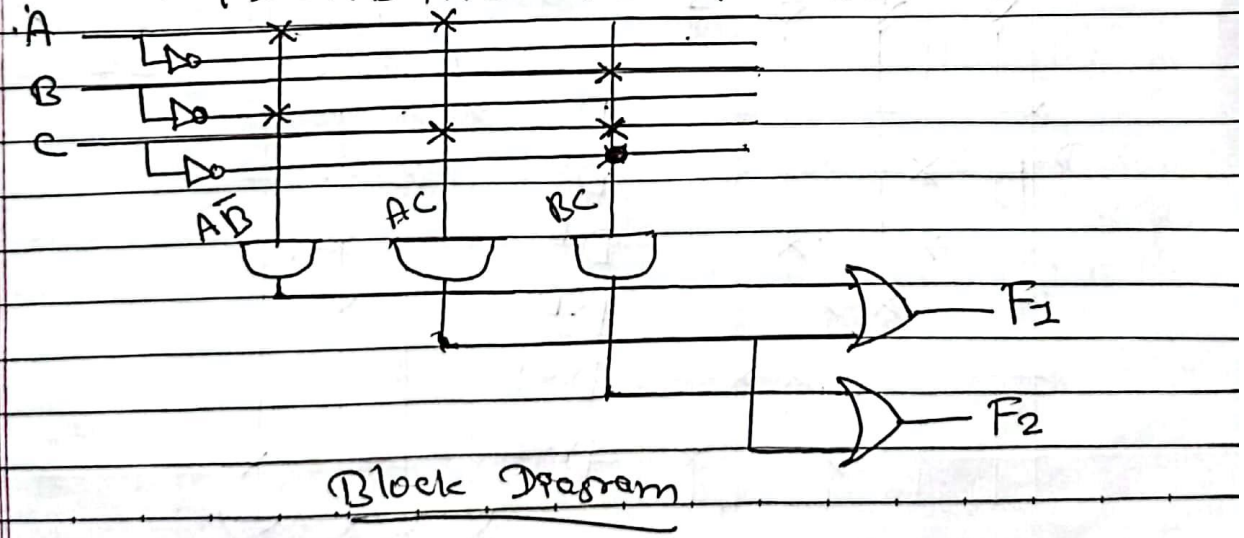
A \ BC	00	01	11	10
0				
1	1	1	1	

$F_1 = (A\bar{B}) + AC$

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

$F_2 = (A \cdot C) + B \cdot C$

$F_1 = A\bar{B} + AC \quad F_2 = A \cdot C + BC$



Block Diagram

MAYUR 780X420 SINGLE LINE 2COL 205X256.667 OK / Dt. 30-03-2021

⇒ BCD to Gray Code using PLA.

Input				output			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

don't care outputs

$2^4 = 16$

0 to 15

don't care → 10, 11, 12, 13, 14, 15

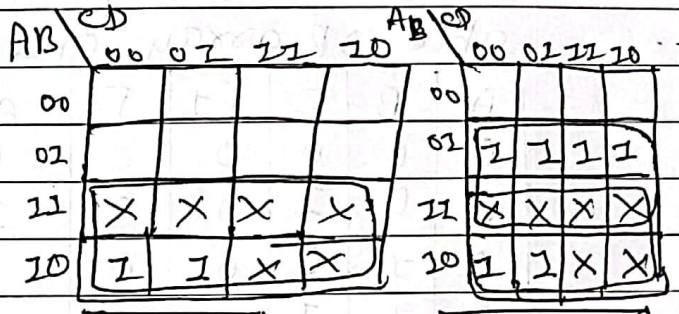
BCD → 0 to 9.

$w = \Sigma m(8,9) + d(10,11,12,13,14,15)$

$x = \Sigma m(4,5,6,7,8,9) + d(10,11,12,13,14,15)$

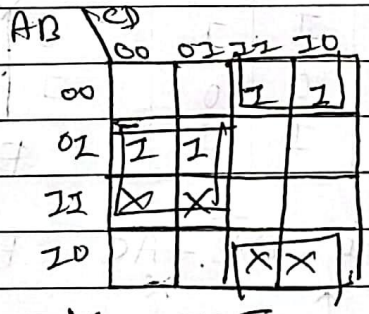
$y = \Sigma m(2,3,4,5) + d(10,11,12,13,14,15)$

$z = \Sigma m(1,2,5,6,9) + d(10,11,12,13,14,15)$

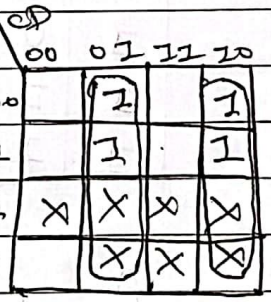


$w = A$

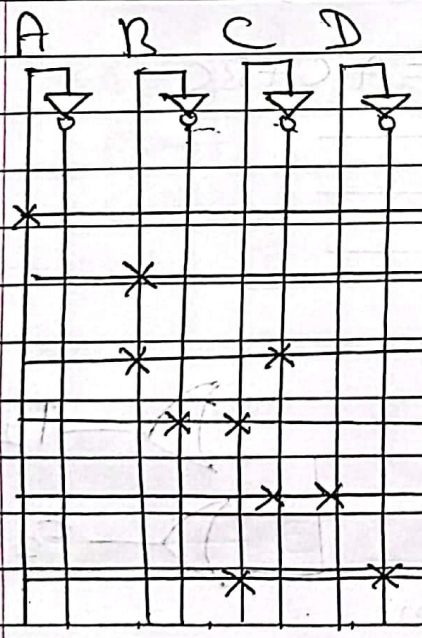
$X = A + B$

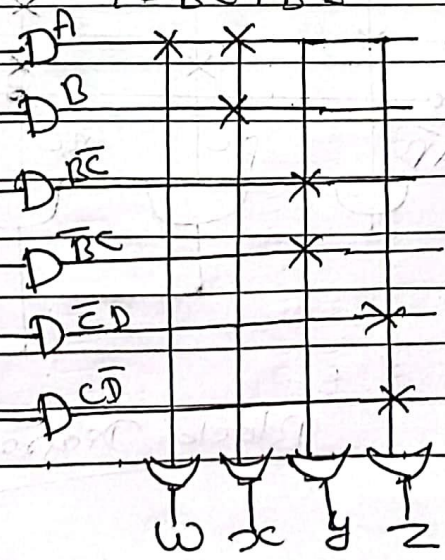


$Y = B\bar{C} + \bar{B}C$



$Z = \bar{C}D + C\bar{D}$

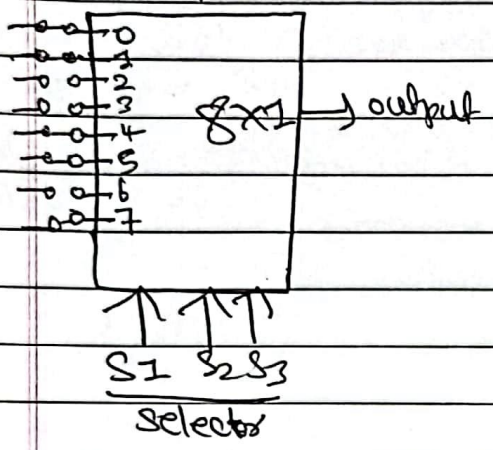




MAYUR
Date
Page

(39) Implement ~~Ex-OR~~ $F = \Sigma(0,1,4,5,7)$ using.
(a) Multiplexer (b) Decoder (c) PLA

(a) Multiplexer.



(b) Decoder

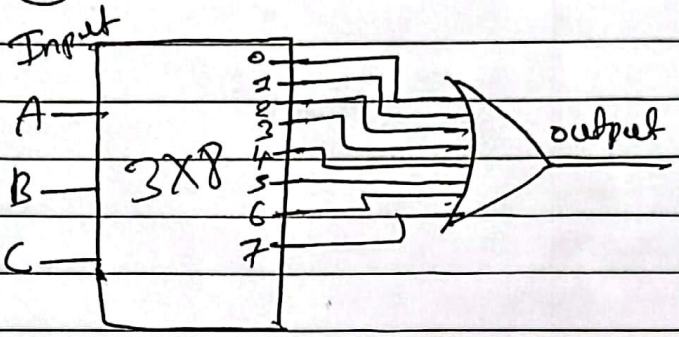


fig: Decoder

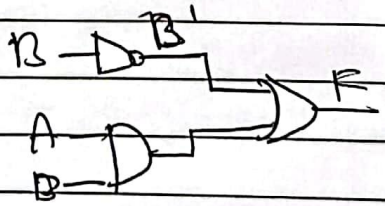
(c) PLA $\Sigma(0,1,4,5,7)$

Input			Output
A	B	C	$F = \Sigma(0,1,4,5,7)$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

K-Map

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

$\Rightarrow B' + AC$



circuit Diagram

MAYUR
Date
Page

(23) Define Decoder. Draw Logic diagram and truth table of 3 to 8 decoder.

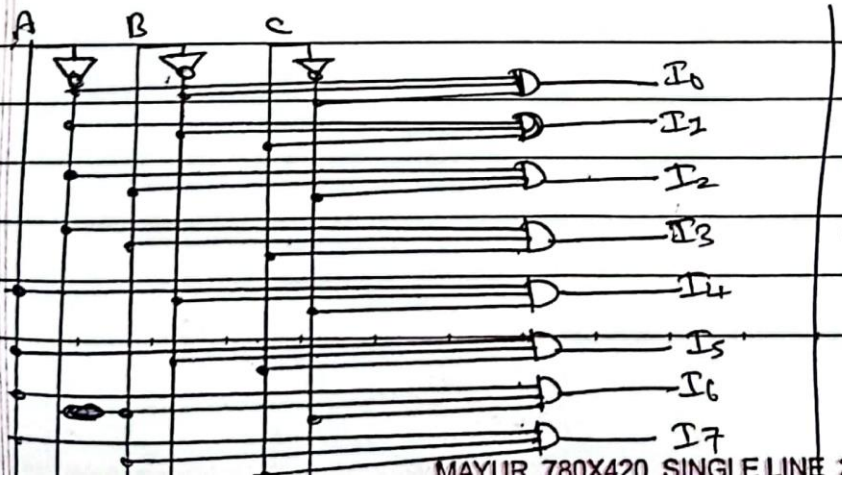
⇒ A decoder is a digital circuit that converts a binary code into a specific output signal. It has one or more input lines and several output lines and is used in digital systems to decode address and command signals.

3 to 8 Decoder

→ It has 3 input and 8 output lines.
→ It is also called binary to octal decoder.
→ AND gate is used in this type of decoder.

Truth Table for 3 to 8 Decoder

A	B	C	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0



Boolean function

$I_0 = A' \cdot B' \cdot C'$
$I_1 = A' \cdot B' \cdot C$
$I_2 = A' \cdot B \cdot C'$
$I_3 = A' \cdot B \cdot C$
$I_4 = A \cdot B' \cdot C'$
$I_5 = A \cdot B' \cdot C$
$I_6 = A \cdot B \cdot C'$
$I_7 = A \cdot B \cdot C$

MAYUR 780X420 SINGLE LINE 2COL 205X256.667 OK/Dt.30-03-2021

MAYUR
Date
Page

Basic Theorems

1) Commutative Law
   (i) $(A+B) = (B+A)$
   (ii) $(A \cdot B) = (B \cdot A)$

2) Associative Law
   (i) $(A+B)+C = A+(B+C)$
   (ii) $(A \cdot B) \cdot C = A \cdot (B \cdot C)$

3) Distributive Law
   (i) $A \cdot (B+C) = A \cdot B + B \cdot C$
   (ii) $A+(B+C) = (A+B) \cdot (A+C)$

4) Identity Law
   (i) $A+0 = A$
   (ii) $A \cdot 1 = A$

5) Complement Law
   (i) $A+A' = 1$
   (ii) $A \cdot A' = 0$

De-Morgan's Theorem

Rules → (1) Statement
(2) Graphical Symbol (Figure)
(3) Truth table
(4) Conclusion.

First Theorem :-

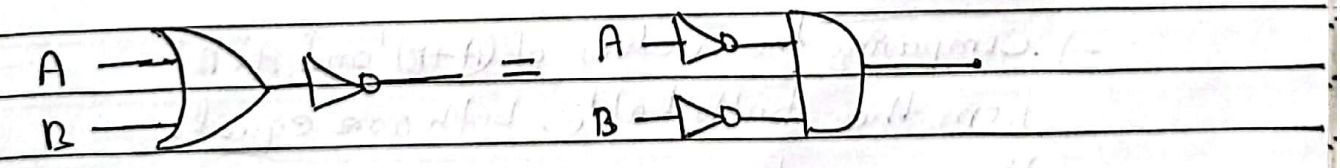
⇒ The DeMorgan's first theorem states that, "The complement of a sum equals to the product of the complements".

i.e. $(A+B)' = A' \cdot B'$

Proof :-

Graphical Symbol :-

$\overline{(A+B)}$ or $(A+B)'$ $\quad$ $A' \cdot B'$



MAYUR 780X420 SINGLE LINE 2COL 205X256.667 OK / Dt. 30-03-2021

MAYUR
Date
Page

Truth table

Input		Output	
A	B	A+B	(A+B)'
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Input		Output
A	B'	A'.B'
0	0	
0	1	
1	0	
1	1	

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

Input				Output
A	B	A'	B'	(A'.B')
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

Conclusion

⇒ Comparing the values of $(A+B)'$ and $A' \cdot B'$ from the truth table, both are equal.

Hence, proved.

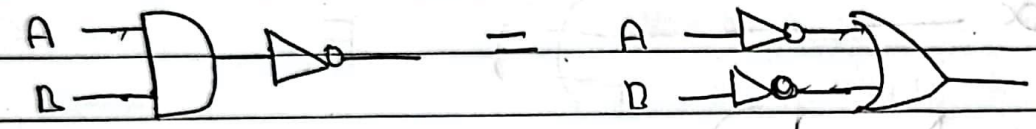
Second Theorem:

Statement: It states that, "The complement of a product is equal to the sum of the complements."

i.e. $(A \cdot B)' = A' + B'$

Proof Graphical Symbols

$(A.B)'$ or $\overline{A.B}$ $\qquad$ $A' + B'$



Truth table

Input		output		Input		Output
A	B	A.B	$\overline{A.B}$	A'	B'	A'+B'
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

Conclusion

⇒ Comparing the values of $(A.B)'$ and $A'+B'$ from the truth table, both are equal. Hence, proved.

MAYUR 780X420 SINGLE LINE 2COL 205X256 667 OK / Dt: 30-03-2021

2. Error Detection Codes.

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

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

odd parity Message	P
0000	1
0001	0
0010	0
0011	1
0100	0
0101	1
0110	1
0111	0
1000	0

even parity message	P
0000	0
0001	1
0010	1
0011	0
0100	1
0101	0
0110	0
0111	1
1000	1

1001	1
1010	1
1011	0
1100	1
1101	0
1110	0
1111	1

1001	0
1010	0
1011	1
1100	0
1101	1
1110	1
1111	0

3. Gray Code. (Reflected code)

It is a binary coding scheme used to represent digits generated from a mechanical sensor that may be prone to error. Used in telegraphy in the 1800s, and also known as "reflected binary code". Gray code was patented by Bell labs researcher Frank Gray in 1947. In Gray Code, there is only one bit location different bet^n 2 successive values, which makes mechanical transitions from one digit to the next less error prone. The following chart shows normally binary representations from 0 to 15 and the corresponding Gray Code.

Decimal digit	Binary code	Gray Code
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1

Features	PLA	PAL
Full Forms	PLA is an abbreviation for Programmable Logic Array.	DHCP is an abbreviation for Programmable Array Logic.
Cost	Its cost is high.	Its cost is low.
Speed	Its speed is low.	Its speed is high.
Usable	It is less usable.	It is more usable.
Availability	It is less available.	It is easier to produce and more easily.
Function Implementation	It has a limited amount of functions implemented.	It has a huge number of functions implemented.
Complexity	Its complexity is high than PAL.	Its complexity is less than PLA.
Design	It may be built utilizing a programmable set of AND gates and a fixed set of OR gates.	A programmable set of AND and OR gates may be utilized to build PAL.
Flexibility	It is more flexible as compared to PAL.	It is less flexible than PLA.

S.NO	PROM	EPROM
1.	PROM is not reusable.	EPROM is reusable multiple times.
2.	PROM is inexpensive.	EPROM is costlier than PROM.
3.	Writing to PROMS is irreversible, which means its memory is permanent.	EPROM's processes can be reversed.
4.	The storage endurance of PROM is high.	EPROM's storage endurance of PROM is less than PROM.
5.	PROM is totally sheathed during a plastic cowl.	EPROM is boxed in during a rock crystal window so the ultraviolet radiation rays will transfer through it.
6.	PROM is a type of ROM that is written only.	EPROM is a type of ROM that is read and written optically.
7.	If there's a miscalculation or error or bug while writing on PROM, it becomes unusable.	If there's a miscalculation or error or bug while writing on EPROM, it will still be used once more.
8.	PROM is the older version of EPROM.	EPROM is the modern version of PROM.