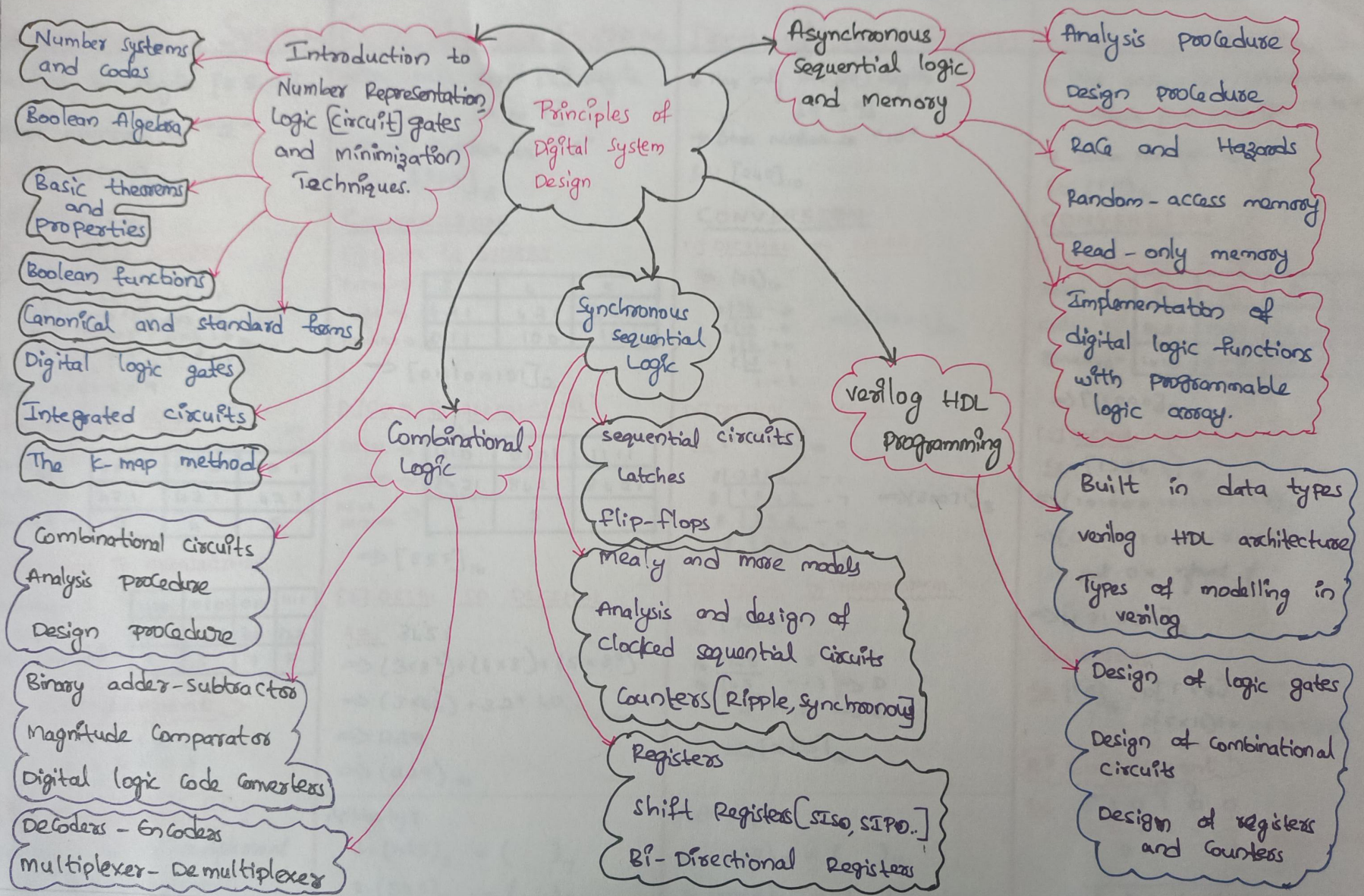
ECA47


Principles of Digital system Design
Introduction to Number Representation Logic [circuit] gates and minimization Techniques.
Number systems and codes
Boolean Algebra
Basic theorems and properties
Boolean functions
Canonical and standard forms
Digital logic gates
Integrated circuits
The K-map method
Combinational Logic
Combinational circuits
Analysis procedure
Design procedure
Binary adder-subtractor
Magnitude Comparator
Digital logic code Converters
Decoders - Encoders
Multiplexer - Demultiplexer
Synchronous sequential Logic
sequential circuits
Latches
flip-flops
Mealy and more models
Analysis and design of clocked sequential circuits
Counters [Ripple, synchronous]
Registers
Shift Registers [SISO, SIPO..]
Bi-Directional Registers
Asynchronous Sequential logic and memory
Analysis procedure
Design procedure
Race and Hazards
Random-access memory
Read-only memory
Implementation of digital logic functions with programmable logic arrays.
verilog HDL programming
Built in data types
verilog HDL architecture
Types of modelling in verilog
Design of logic gates
Design of combinational circuits
Design of registers and Counters

NUMBER SYSTEM

BINARY NUMBER SYSTEM

→ Has only two digits [0 & 1]

→ Base number is "2"

Ex: $(1111000)_2$

CONVERSION

[1] BINARY TO DECIMAL

Ex: 1001

$1 \times 2^0 = 1 \times 1 = 1$

$0 \times 2^1 = 0 \times 2 = 0$

$0 \times 2^2 = 0 \times 4 = 0$

$1 \times 2^3 = 1 \times 8 = 8$

$\Rightarrow 1+0+0+8 = 9$.

[2] BINARY TO OCTAL

Binary	011	100	101
Code	421	421	421
Octal	3	4	5

[3] BINARY TO HEXADECIMAL

Binary	1110	0101	0111	1111
code	8421	8421	8421	8421
Hexadecimal	E	5	7	F

1's Complement

0 1 0 1 0 0

↓ ↓ ↓ ↓ ↓ ↓

1 0 1 0 1 1

2's Complement

Adding 1 to 1's complement is 2's complement.

OCTAL NUMBER SYSTEM

→ Has only eight [8] digits [0 to 7]

→ Base number is "8"

Ex: $[360]_8$

CONVERSION

[1] OCTAL TO BINARY

OCTAL →	3	4	5
CODE →	421	421	421
BINARY →	011	100	101

$\Rightarrow [011100101]_2$

[2] OCTAL TO HEXADECIMAL

OCTAL →	1110	0101	1111
CODE →	8421	8421	8421
HEXA DECIMAL →	E	5	F

$\Rightarrow [E5F]_{16}$

[3] OCTAL TO DECIMAL

Ex: 345

$\Rightarrow (3 \times 8^2) + (4 \times 8^1) + (5 \times 8^0)$

$\Rightarrow (3 \times 64) + 32 + 40$

$\Rightarrow 229$

$\Rightarrow (229)_{10}$

Activity:

1. $(423)_5 = (\quad)_7$
2. $(323)_4 = (\quad)_9$

DECIMAL NUMBER SYSTEM

→ Has only ten [10] digits [0 to 9]

→ Base number is "10"

Ex: $[240]_{10}$

CONVERSION

[1] DECIMAL TO BINARY

Ex: $(24)_{10}$

2 | 24 - 0
2 | 12 - 0
2 | 6 - 0
2 | 3 - 1
1 - 1

$\Rightarrow [11000]_2$

[2] DECIMAL TO OCTAL

Ex: $[12345]_{10}$

8 | 12345 - 1
8 | 1543 - 7
8 | 192 - 0
8 | 24 - 0
3 - 3

$\Rightarrow (30071)_8$

[3] DECIMAL TO HEXADECIMAL

Ex: (725)

16 | 725 - 5 → 5
16 | 45 - 13 → D
2 - 2 → 2

$\Rightarrow [2D5]_{16}$

Activity:

1. $(1024)_{10} = (\quad)_{16}$
2. $(ABCD)_{16} = (\quad)_{10}$

HEXADECIMAL NUMBER SYSTEM

→ Has sixteen [16] Alphanumeric values from (0 to 9)& (A to F)

→ Base number is "16"

Ex: $(FO)_{16}$

CONVERSION

[1] HEXADECIMAL TO BINARY

Hexa -	A	2	D	E
Code -	8421	8421	8421	8421
Binary -	1010	0010	1101	1110

$\Rightarrow [11000]_2$

[2] DECIMAL TO OCTAL

Ex: $[12345]_{10}$

$\Rightarrow (1010000101101111 0)_2$

$\rightarrow (001010000101101111 0)_2$

∴ put 0's infront of 1.

$\Rightarrow [121336]_8$

TO DECIMAL

Ex: $[FO]_{16} = O[F \times 16^1) + (O \times 16^0)$

$= (15 \times 16) + 0 = (240)_{10}$

2's Complement

Ex:

0 0

0 1 0 1 0 0

↓ ↓ ↓ ↓ ↓ ↓

1 0 1 0 1 1 → 1's complement

1 +

1 0 1 1 0 0

BOOLEAN FUNCTIONS SIMPLIFICATION AND IMPLEMENTATION

Boolean algebra :-

deals with binary Variable and Logic operation

Boolean function:- Consists of binary variables, Constants (0/1) and Logic operators

Ex :- $Y = ab' + b'c$

Simplications

→ Using boolean axioms and Laws
→ K-map
→ Tabulation Method

Different form of Boolean functions :-

1. Canonical form
   - L Sum of min terms
   - L product of max terms
2. Standard form
   - L SOP
   - L POS

Postulates and theorms of BA

1. $x + 0 = x$; $x.1 = x$
2. $x + x' = 1$; $x.x' = 0$
3. $x + x = x$; $x.x = x$
4. $x + 1 = 1$; $x.0 = 0$
5. Involution $(x')' = x$
6. Commutative $x + y = y + x$
7. Associative $x+(y+z) = (x+y)+z$
8. Distributive $x(y+z) = xy + xz$; $x+(yz) = (x+y)(x+z)$
9. Demorgans $(x+y)' = x'.y'$; $(x.y)' = x'+y'$
10. absorption $x + xy = x$; $x(x+y) = x$

Conversion of Standard form into Canonical form.

Ex:-1 $Y = a'b + bc$ (Stand-form)

Ans:- $Y = a'b(c+c') + (a+a')bc$
$= a'bc + a'bc' + abc + a'bc$
$= a'bc + a'bc' + abc$ (Canon)
011 010 111
m_3 m_2 m_7

$Y = \Sigma_m(2,3,7) \rightarrow$ (CF)

Truth table for Boolean function

Ex:- $Y = a'b + bc$

A:- Convert it into CF

$Y = \Sigma_m(2,3,7)$

→ place 1 against this minterm in O/P

a	b	c	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

POS form of given Boolean Exp

$F = \Sigma_m(0,3,5,7,9,11)$

Ans:- Identify no. of Variables place other no. other than min term given

Ans:- $f = \Pi_m(1,2,4,6,8,10,12,13,14,15)$

SOP and POS form of BF

$F = \Sigma_m(1,3)$
$= a'b + ab$ - SOP
$b(a+a') = b$ (or)

$f = \Pi_m(0,2)$
$= (a+b).(a'+b)$ - POS
$= aa' + ab + a'b + b.b$
$= a'b + ab + b = b(1 + a + a')$
$= b$

Boolean for Simplication using BF

1. Given in SOP form

$f_1 = A'BC + A'B'C + C'$

A:- $f_1 = A'C(B+B') + C'$
$= A'C + C' = C' + CA'$
$= C' + A'$

2. Given in POS form

$f_2 = (A'+B+C')(A'+B')(C+A)$

A:- $f_2 = (A'A' + A'B' + A'B + BB' + A'+A'C' + B'C')(C+A)$
$= (A'(A'B' + A'B + A'C' + B'C'))(C+A)$
$= A'B'C + A'BC + A'CC' + B'CC' + AA'B' + AA'B + AA'C' + AB'C' + A'C + AA'$
$= A'B'C + A'BC + AB'C' + A'C$
$= A'C(B'+B+1) + AB'C'$
$= A'C + AB'C'$

Simplification Using K-map

Step 1 :- Identify no. of i/p(n) Variable in given function

Step 2 :- Bring given function in Canonical form

Step 3 :- Draw Suitable n-varia K-map for Simplication

Ex :- find minimal SOP and POS form of given function

1. $Y = a'bc + bc' + ac$

A:- Bring it to Canonical form

$Y = a'bc + (a+a')bc' + a(b+b')c$
$= a'bc + abc' + a'bc' + a'b'c' + ab'c$
011 110 010 111 101
$= \Sigma_m(2,3,5,6,7)$

→ Given func is 3 i/p variable func Hence draw 3 variable K-map.

a \ bc	00	01	11	10
0			1	1
1		1	1	1

$Y = b + ac \rightarrow$ SOP

→ To find POS form

a \ bc	00	01	11	10
0	0	0		
1	0			

$Y = (a+b).(b+c) \rightarrow$ POS

Ex:-2 :- find the minimal SOP and POS form of given fun

$Y_2 = \Pi m(1,3,5,7,8,9,12,13)$

A :- Given fn. is in Comonical form and 4 Variable i/p fn. Hence use 4 Variable K-map

ab \ cd	00	01	11	10
00		0	0	
01		0	0	
11	0	0		
10	0	0		

$Y_2 = (a+d')(a'+c)$

min POS

ab \ cd	00	01	11	10
00	1			1
01	1			1
11			1	1
10			1	1

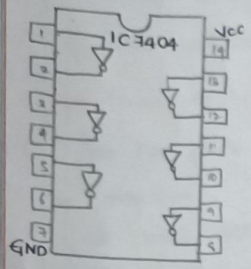
$Y_2 = a'd' + ac$

min SOP

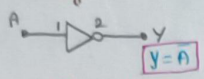
Logic Gates

NOT Gate

PIN DIAGRAM - IC 7404



Symbol



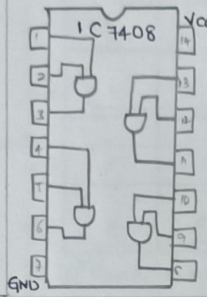
$Y = \bar{A}$

Truth table

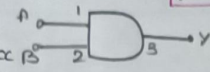
A	$Y = \bar{A}$
0	1
1	0

AND GATE

PIN DIAGRAM - IC 7408



Symbol



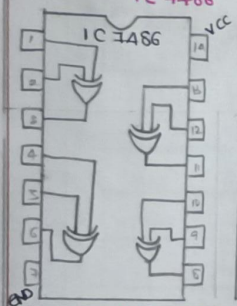
$Y = AB$

Truth table

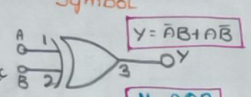
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

X-OR GATE

PIN DIAGRAM - IC 7486



Symbol



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

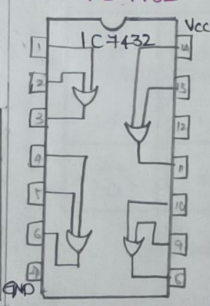
$Y = A \oplus B$

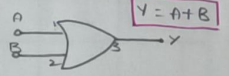
Truth Table

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

OR GATE

PIN DIAGRAM - IC 7432





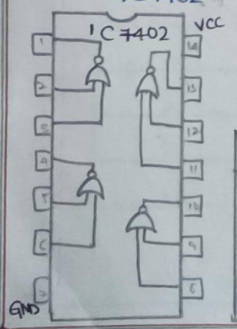
$Y = A + B$

Truth table

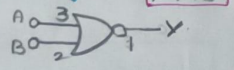
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

NOR Gate

PIN DIAGRAM - IC 7402



Symbol



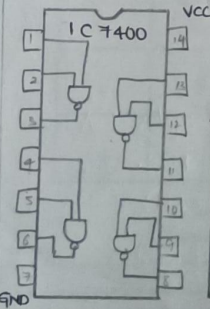
$Y = \overline{A + B}$

Truth table

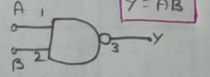
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

NAND GATE

PIN DIAGRAM - IC 7400



Symbol

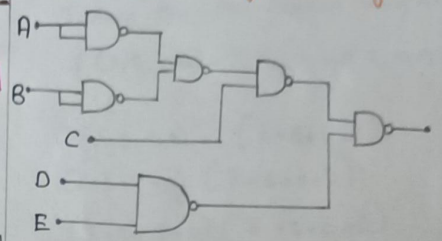


$Y = \overline{AB}$

Truth table

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

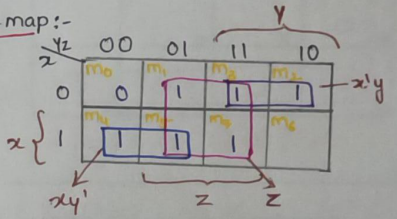
* Implement boolean function $f = (A+B)C + DE$ using NAND gates.



* Implement the following Boolean function with NAND gates:

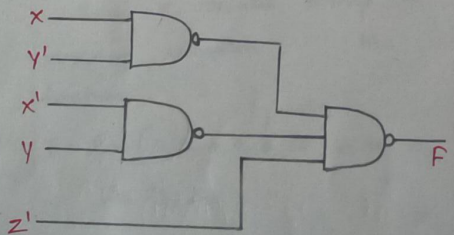
$F(x,y,z) = (1,2,3,4,5,7)$

k-map:-



$f = xy' + x'y + z$

Logic Diagram:-



* Implement the following Boolean function with NOR-NOR logic

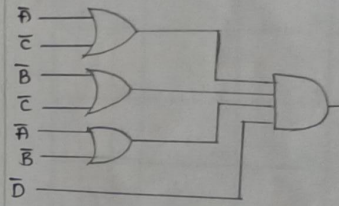
$Y = AC + BC + AB + D$

Sol:-

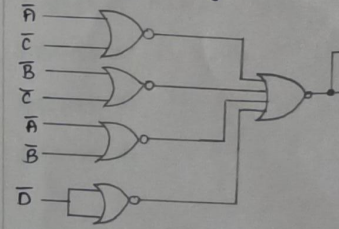
Step 1:- Express Boolean function in POS form

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

Step 2:- Implement Boolean function with OR-NAND logic.



Step 3:- Convert OR-AND logic to NOR-NOR logic.



Activity:-

① Implement the following Boolean function using NAND-NAND

$F(x,y,z) = (0,2,3,4,5,7)$

② NOR-NOR Implementation

$F(A,B,C) = \Pi M(0,2,4,5,6)$

K-Map SIMPLIFICATION

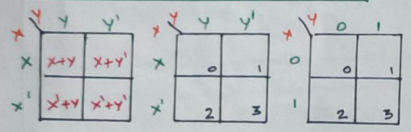
PRODUCT OF SUMS (POS) REDUCTION USING K-MAP

K-map is a table consisting of cells which represents a minterm or maxterm.

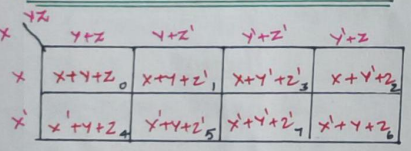
For POS each cell in k-map represents a maxterm

We will fill the cells with '0' whose maxterm output is '0'

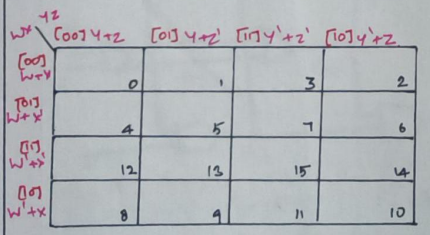
TWO VARIABLE K-MAP IN POS



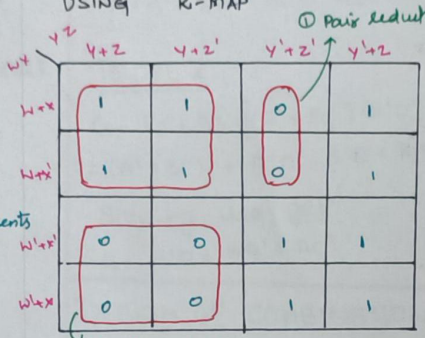
THREE VARIABLE K-MAP IN POS

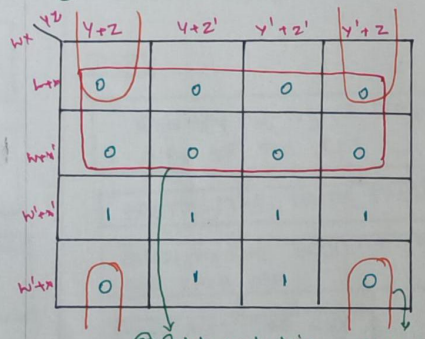


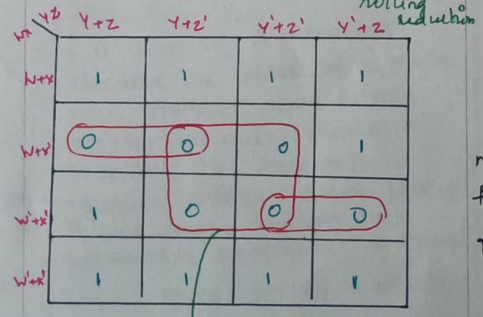
FOUR VARIABLE K-MAP IN POS



REDUCING RULES FOR POS USING K-MAP





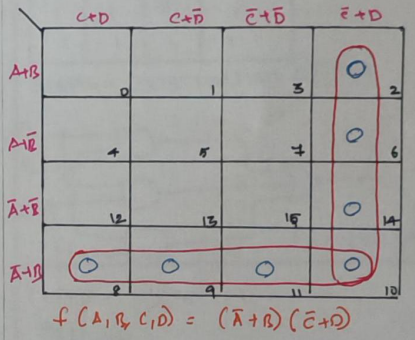


PROBLEMS

Reduce the POS function using k-map

$f(A,B,C,D) = \Pi M(2,6,8,9,10,11,14)$

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



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

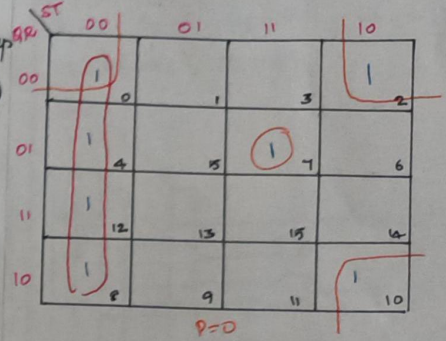
5 VARIABLE K-MAP OF SOP

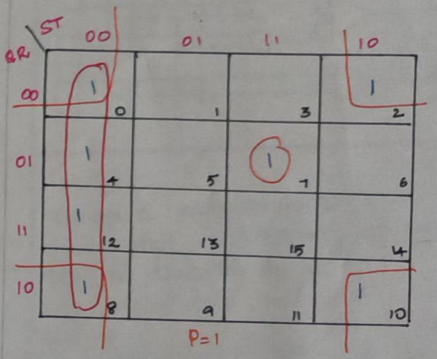
$2^5 = 32$ cells to fill each minterm

PROBLEM:

Solve 5-variable k-map for maximum reduction, the function

$F = \Sigma m(0,2,4,7,8,10,12,14,18,20,22,24)$





SUBCUBE 1: 0, 4, 8, 12, 16, 20, 28, 24

SUBCUBE 2: 0, 2, 8, 10

SUBCUBE 3: 18, 24

SUBCUBE 4: 7, 23

$\bar{S}\bar{T} + \bar{Q}RST + QR\bar{S}\bar{T} + \bar{Q}R\bar{S}T$

PROBLEMS

$f(A,B,C,D,E) = \Sigma m(0,2,4,6,9,11,13,15,17,21,25,27,29,31)$

→ The circuit whose output depend upon the present state inputs alone are called "Combinational Circuits".

Analysis of Combinational Logic Circuit

- To obtain the Boolean functions from the logic circuit

STEPS

Label the O/Ps of All the Gates

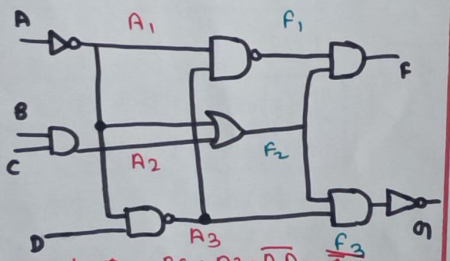
↓

Determine the Boolean FN of each O/P

↓

Continue till the final O/P is reached

PROBLEM - Mano 4.2



$A_1 = A'$; $A_2 = BC$; $A_3 = \overline{A_1 D} = \overline{A'D}$

$F_1 = (A_1, A_3)' = [A', (\bar{A}D)']'$; $F_2 = A_2 + A_1$

$F_2 = A' + BC$; $F_3 = F_2 \cdot A_3$

$F = F_1 . F_2 = [A'(\bar{A}D)]' . (A' + BC)$

$= [A + (A + D')'] (A' + BC)$

COMBINATIONAL LOGICS

Analysis
Design

$= (A + A'D)(A' + BC) = ABC + A'D$

To Get G

$G = F_2 + A_3 = \{(A' + B()(A'D)'\}'$

$= (A' + BC)' + A'D = A'D + A(BC)'$

Solving, we get

$G = A'D + AB' + AC'$

DESIGN OF COMBINATIONAL LOGIC CKTS

STEPS

IDENTIFY THE NO. OF I/Ps & O/Ps OF THE CKT

↓

CREATE THE TRUTH TABLE

↓

SIMPLIFY THE BOOLEAN FN. FOR EACH O/P

↓

CONSTRUCT THE CKT

PROBLEMS

1. A team of 3 members decides the issues of a class. Each member votes either Yes / NO for every issue. A proposal is passed it it receives atleast 2 Yes. Design a CKT which determines whether a proposal is paned.

Step 1 :- No of i/ps - 3 ; No of O/Ps - 1

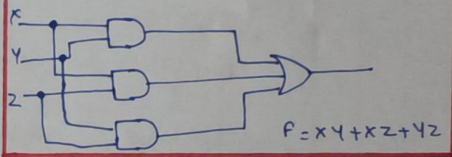
Step 2 :- Draw the TT

Step 3 :- Simplify

M_1 x	M_2 y	M_3 z	Proposal F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$F = xy + xz + yz$

Step 4 :- Construction of CKT



Problem - Mano 4.6

Activity

PROBLEM

Design a CKT which squares a 3-bit binary no and gives it as the O/P

Step 1 : No of I/Ps - 3
No of O/P - 6

Step 2 : TT

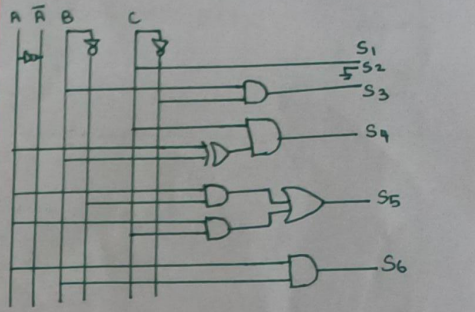
I/P			O/P					
A	B	C	S_6	S_5	S_4	S_3	S_2	S_1
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1
0	1	0	0	0	0	1	0	0
0	1	1	0	0	1	0	0	1
1	0	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0	1
1	1	0	1	0	0	1	0	0
1	1	1	1	0	1	0	0	1

Step 3 :- Simplification

$S_1 = C$; $S_2 = 0$; $S_3 = B\bar{C}$; $S_4 = C(A \oplus B)$

$S_5 = A\bar{B} + AC$ $S_6 = AB$

Step 4 :-



PROBLEM

Design a combinational logic circuit that can detect a no 8421 BCD no divisible by 2

Step 1: No of i/p - 4 No of o/Ps - 1

Step 2: Step 3: simplify

I/P				O/P
D_8	D_4	D_2	D_1	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

$Y = D_1'$

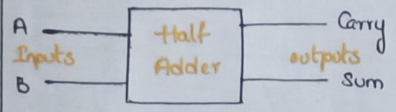
BINARY ADDER & SUBTRACTOR

ADDERS

HALF ADDER [Adding of 2-bit]

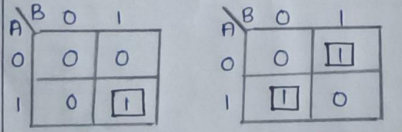
FULL ADDER [Adding of 3-bit]

Half - Adder :-



Truth Table for Half - Adder

Inputs		outputs	
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

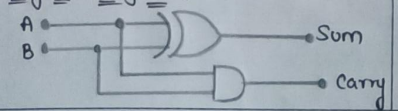


Carry = AB

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

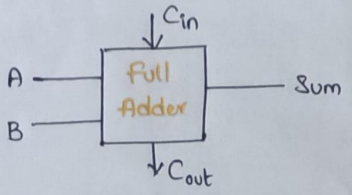
$= A \oplus B$

Logic Diagram :-



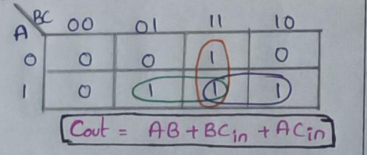
Full - Adder :-

- A full adder performs the additions of 3-bit

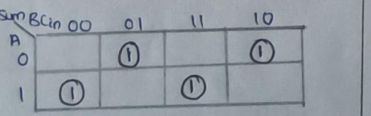


Truth Table

Inputs			outputs	
A	B	Cin	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



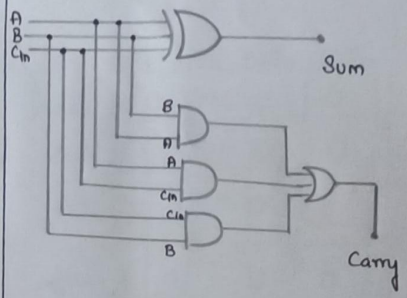
Cout = $AB + BC_{in} + AC_{in}$



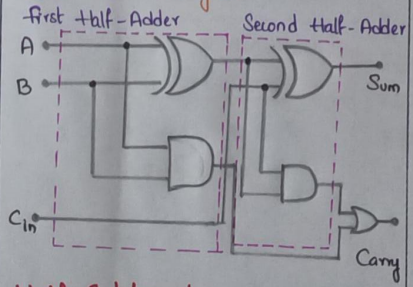
Sum = $\bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$

Sum = $C_{in} \oplus (A \oplus B)$

Logic diagram for Full - Adder :-



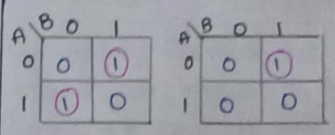
* Implementation of full adder with two Half-adder and an OR gate



Half Subtractor :-

- It is a combinational circuit which performs subtraction of 2-bits

Inputs		Outputs	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

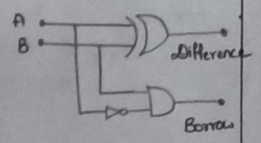


Difference = $\bar{A}B + A\bar{B}$

$= A \oplus B$

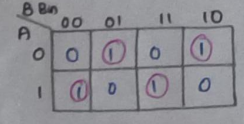
Borrow = $\bar{A}B$

Logic Diagram

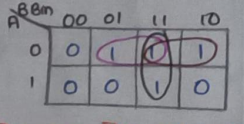


Full Subractor :-

Inputs			Outputs	
A	B	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

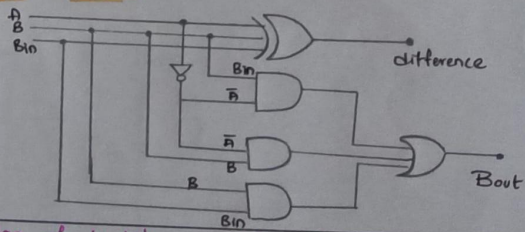


Difference = $B_{in} \oplus (A \oplus B)$

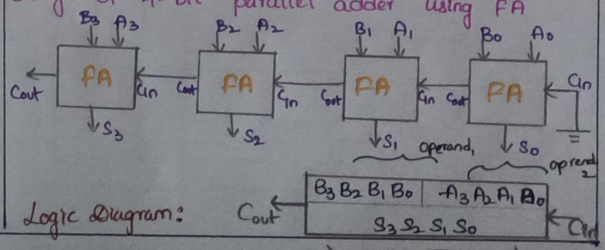


Borrow = $\bar{A}B_{in} + BB_{in} + \bar{A}B$

Logic Diagram

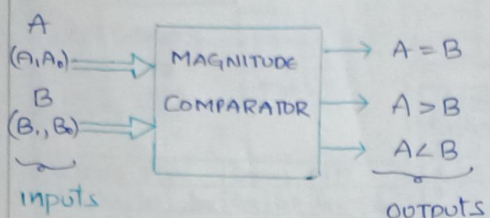


Design of 4-bit parallel adder using FA



MAGNITUDE COMPARATOR

2-BIT COMPARATOR



TRUTH TABLE

INPUT				OUTPUT		
A		B				
A_1	A_0	B_1	B_0	A=B	A>B	A<B
0	0	0	0	1	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	0	1	0
0	1	0	1	1	0	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	0	1	0
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	0	1	1	0	0	1
1	1	0	0	0	1	0
1	1	0	1	0	1	0
1	1	1	0	0	1	0
1	1	1	1	1	0	0

K-MAP

A=B

	$\bar{B}_1\bar{B}_0$	$\bar{B}_1B_0$	B_1B_0	$B_1\bar{B}_0$
$\bar{A}_1\bar{A}_0$	1	0	0	0
$\bar{A}_1A_0$	0	1	0	0
A_1A_0	0	0	1	0
$A_1\bar{A}_0$	0	0	0	1

A>B

	$\bar{B}_1\bar{B}_0$	$\bar{B}_1B_0$	B_1B_0	$B_1\bar{B}_0$
$\bar{A}_1\bar{A}_0$	0	0	0	0
$\bar{A}_1A_0$	1	0	0	0
A_1A_0	1	1	0	1
$A_1\bar{A}_0$	1	1	0	0

A<B

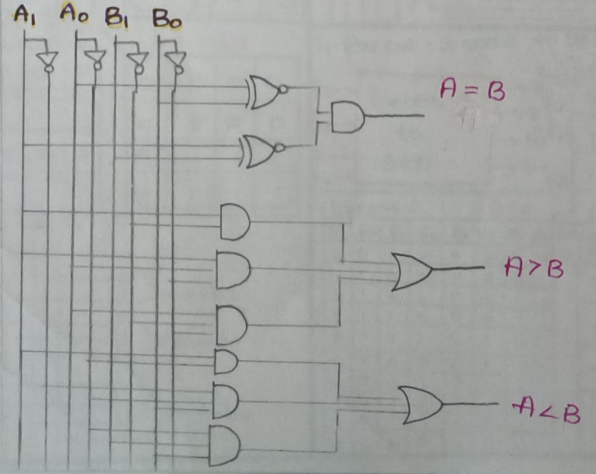
	$\bar{B}_1\bar{B}_0$	$\bar{B}_1B_0$	B_1B_0	$B_1\bar{B}_0$
$\bar{A}_1\bar{A}_0$	0	1	1	1
$\bar{A}_1A_0$	0	0	1	1
A_1A_0	0	0	0	0
$A_1\bar{A}_0$	0	0	1	0

$(A=B) = (A_0 \odot B_0)(A_1 \odot B_1)$

$(A>B) = A_1\bar{B}_1 + A_1A_0\bar{B}_0 + A_0\bar{B}_1B_0$

$(A<B) = \bar{A}_1B_1 + \bar{A}_1\bar{A}_0B_0 + \bar{A}_0B_1B_0$

Logic Diagram:-



PRIORITY ENCODER

→ In priority Encoder, if Two or More inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

TRUTH TABLE

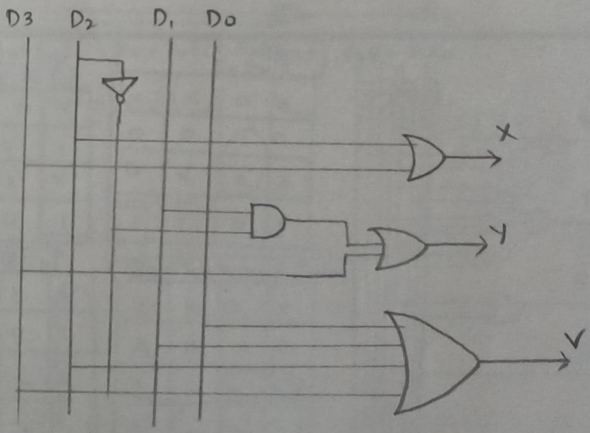
inputs				OUTPUTS		
D_0	D_1	D_2	D_3	X	Y	V
0	0	0	0	x	x	0
1	0	0	0	0	0	1
x	1	0	0	0	1	1
x	x	1	0	1	0	1
x	x	x	1	1	1	1

K-MAP SIMPLIFICATION EQUATIONS

For X => $X = D_2 + D_3$

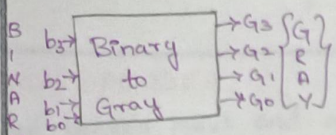
For Y => $Y = D_3 + D_1D_2$

For V => $V = D_0 + D_1 + D_2 + D_3$



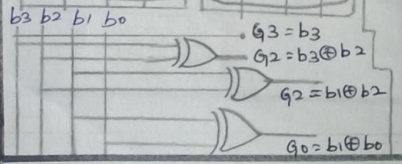
CODE CONVERTERS

1. Binary to gray :-

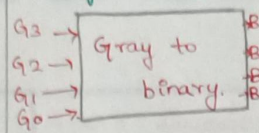


Binary I/P				Gray O/P			
b_3	b_2	b_1	b_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

$G_0 = b_1 \oplus b_0$ | $G_1 = b_1 \oplus b_2$ | $G_2 = b_3 \oplus b_2$ | $G_3 = b_3$

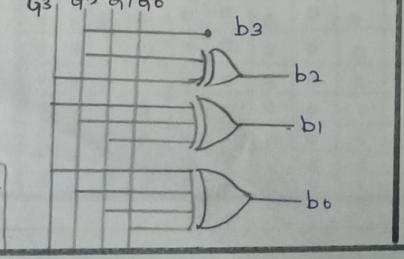


2. Gray to Binary :-

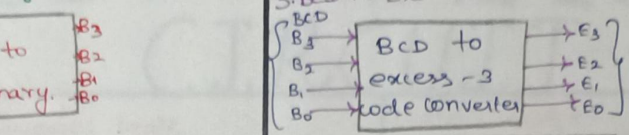


Gray I/P				Binary O/P			
G_3	G_2	G_1	G_0	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

$B_0 = G_3 \oplus G_2 \oplus G_1 \oplus G_0$ | $b_1 = G_3 \oplus G_2 \oplus G_1$ | $b_2 = G_3 G_2 + G_3 \bar{G}_2 + \bar{G}_3 \oplus G_2$ | $b_2 = G_3$



3. BCD To Excess 3 code converter :-



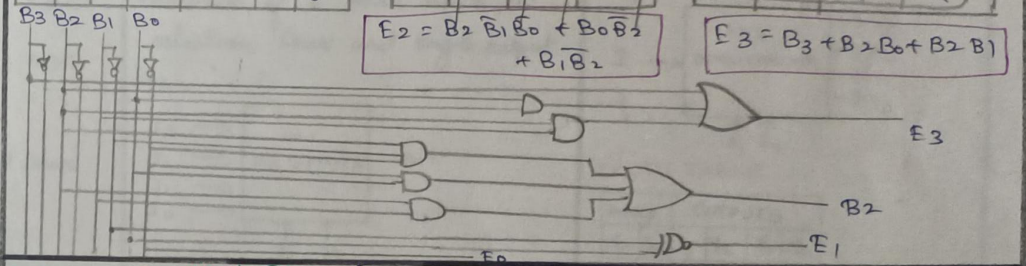
BCD				Excess 3 code			
B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0				
1	0	1	1				
1	1	0	0				
1	1	0	1				
1	1	1	0				
1	1	1	1				

$E_0 = B_0$

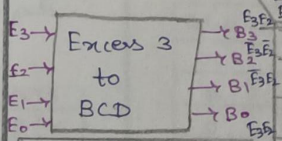
$E_1 = B_1 \odot B_0$

$E_2 = B_2 \bar{B}_1 \bar{B}_0 + B_0 \bar{B}_2 + B_1 \bar{B}_2$

$E_3 = B_3 + B_2 B_0 + B_2 B_1$



4. Excess - 3 code to BCD :-



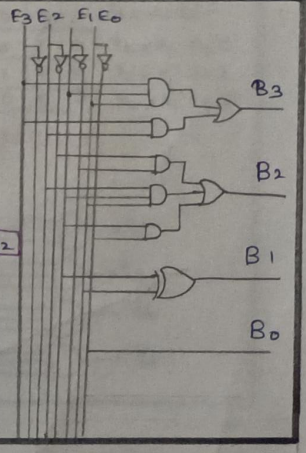
Excess 3				BCD			
E_3	E_2	E_1	E_0	B_3	B_2	B_1	B_0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	0
1	0	0	0	0	1	0	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	1	1
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	1

$B_0 = \bar{E}_0$

$B_1 = E_1 \oplus E_0$

$B_2 = E_2 E_1 + E_2 \bar{E}_1 \bar{E}_0 + \bar{E}_0 E_2$

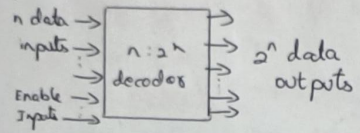
$B_3 = G_3 G_1 G_0 + \bar{E}_3 G_2$



COMBINATIONAL LOGIC CIRCUIT

BINARY DECODER

n bit binary input and a one activated output out of 2^n outputs.

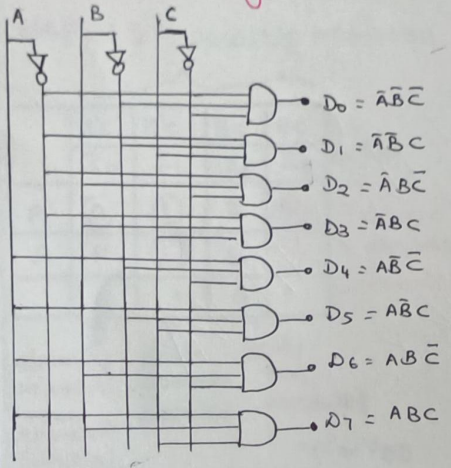


3 to 8 Decoder – Truth Table

A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

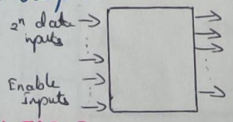
A 3 to 8 Decoder has
→ 3 Inputs
→ 8 Outputs.

3 to 8 Decoder Logic



BINARY ENCODER

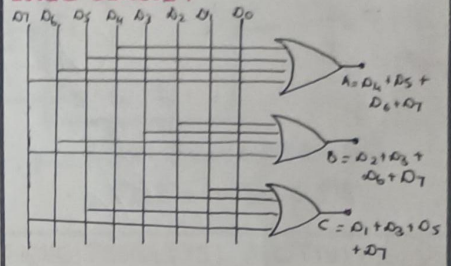
A binary encoder has 2^n input lines and n output lines.



TRUTH TABLE

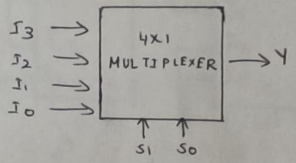
INPUTS								OUTPUTS		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	A	B	C
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

LOGIC CIRCUIT



MULTIPLEXER

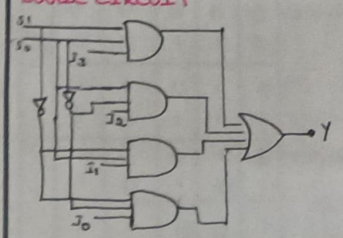
A combinational circuit that has maximum of 2^n data inputs, 'n' selection lines and single output line.



Selection lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

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

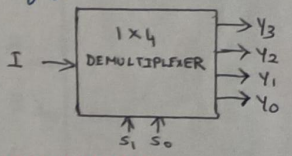
LOGIC CIRCUIT



DEMULTIPLEXER

1 – input, 2^n – output

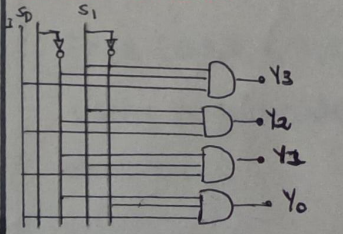
'N' selection lines, 2^n combination of zeros and ones.



TRUTH TABLE

INPUTS		OUTPUTS				
S_1	S_0	Y_3	Y_2	Y_1	Y_0	
0	0	0	0	0	1	$Y_3 = S_1S_0'I$
0	1	0	0	1	0	$Y_2 = S_1S_0'I$
1	0	0	1	0	0	$Y_1 = S_1'S_0I$
1	1	1	0	0	0	$Y_0 = S_1'S_0'I$

LOGIC CIRCUIT



IMPLEMENTATION OF BOOLEAN FUNTIONS USING DECODER : MUX

IMPLEMENTATION USING MUX

Step 1: Find the no. of select lines & i/p lines of the Mux

Eg: In 8:1 Mux No. of select lines : 3 if no. of i/p = 4 (ie n)
No. of select lines = 3(i-1)

Step 2: Formation of implentation table

1 Variable written in the First column of the table
All the combinations of n-1 variables are written column wise.

Step 3: write all the nos in the cells of the Imp table Encircle the given minterm

Step 4: Draw the circuit

PROBLEM: Implement the function
$f(A,B,C) = \Sigma(0,1,3,5,7)$

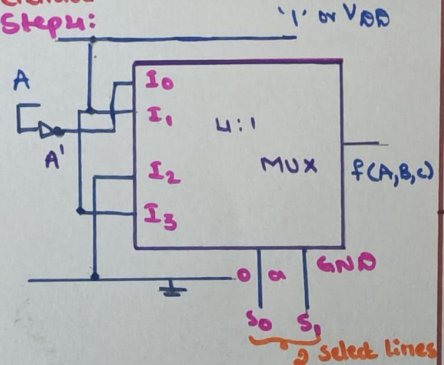
Step 1:
No. of i/p : 3;
No. of Select lines 2
(ie 3-1)

Step 2; 3 Possible Minterm

	B'C	B'C	BC'	BC	← I/p
	I_0	I_1	I_2	I_3	
A'	(0)	(1)	2	(3)	Given minterm encircled
A	4	(5)	6	(7)	
	A'	1	0	1	

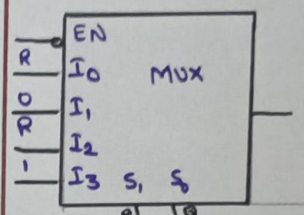
Element in upper row encircled — Both elements encircled — Both not encircled

Step 4:



GATE PROBLEM
A Mux is shown where s, & s_0 → select lines, I_0 to I_5 - data lines, F(P,Q,R) is the o/p

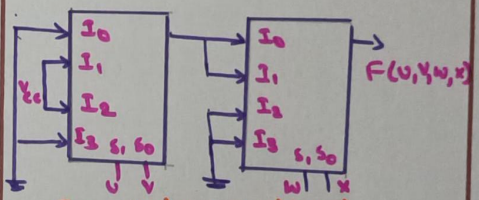
Find F



Sol:
$I_0 = \bar{P}\bar{Q}$
$I_1 = \bar{P}Q$
$I_2 = P\bar{Q}$
$I_3 = PQ$

$\therefore F = \bar{P}\bar{Q}R + \bar{P}Q \times 0 + P\bar{Q}R + PQ$
$= \bar{P}\bar{Q}R + P\bar{Q}R + PQ$

GATE PROBLEM: ACTIVITY
A 4-variables Boolean fn is realised using 4:1 Muxes. Find F



$F = (u'v + uv')w'$

IMPLEMENTATION USING DECODER

Steps
No. of i/p - n
No. of o/p from decodes - 2^n
↑
Act as minterm
Necessary minterm combined using OR gate.

PROBLEM

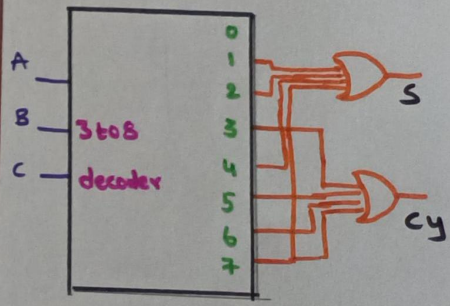
Implement Full addr using decoder.

I Draw the truth table

A	B	C	S	Cy
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$S = \Sigma(1,2,4,7)$
$Cy = \Sigma(3,5,6,7)$
S → Sum
Cy → Carry
1,2,3 etc - Minterm nos

II 3 i/ps, so a 3-to-8 decoder needed

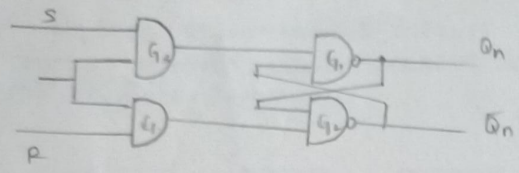


PROBLEM:
Implement the functions
$F = \Sigma(0,3,5,7,9,11,13)$
using a suitable decoder.

FLIP FLOPS

SR FLIPFLOP

LOGIC DIAGRAM



S	R	Q_{n+1}	state
0	0	Q_n	No change
0	1	0	reset
1	0	1	set
1	1	X	undefined

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

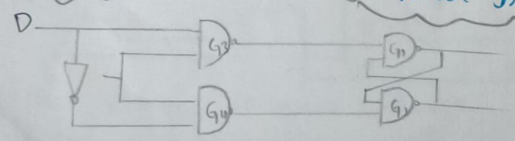
	$R\bar{Q}_n$	$\bar{R}Q_n$	RQ_n	$R\bar{Q}_n$
$\bar{S}$	0	1	0	0
S	1	1	X	X

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

Flip-Flop Applications

- Counters
- Shift Registers
- Data storage
- Latch
- Data Transfer

D FLIPFLOP



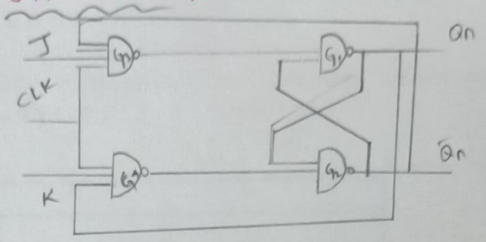
D	Q_{n+1}	state
0	0	reset
1	1	set

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

	$\bar{Q}_n$	Q_n
D	0	0
D	1	1

$Q_{n+1} = D$

JK FLIPFLOP



J	K	Q_{n+1}	state
0	0	Q_n	No change
0	1	0	Reset
1	0	1	set
1	1	$\bar{Q}_n$	Toggles

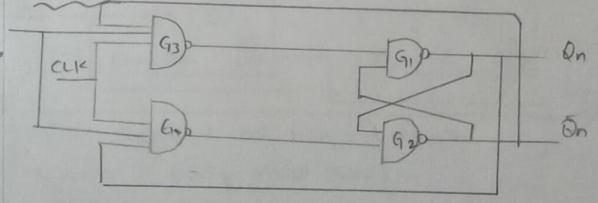
Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

	$\bar{K}\bar{Q}_n$	$\bar{K}Q_n$	$K Q_n$	$K\bar{Q}_n$
$\bar{J}$	0	1	0	0
J	1	1	0	1

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

Characteristic Equation

T FLIPFLOP

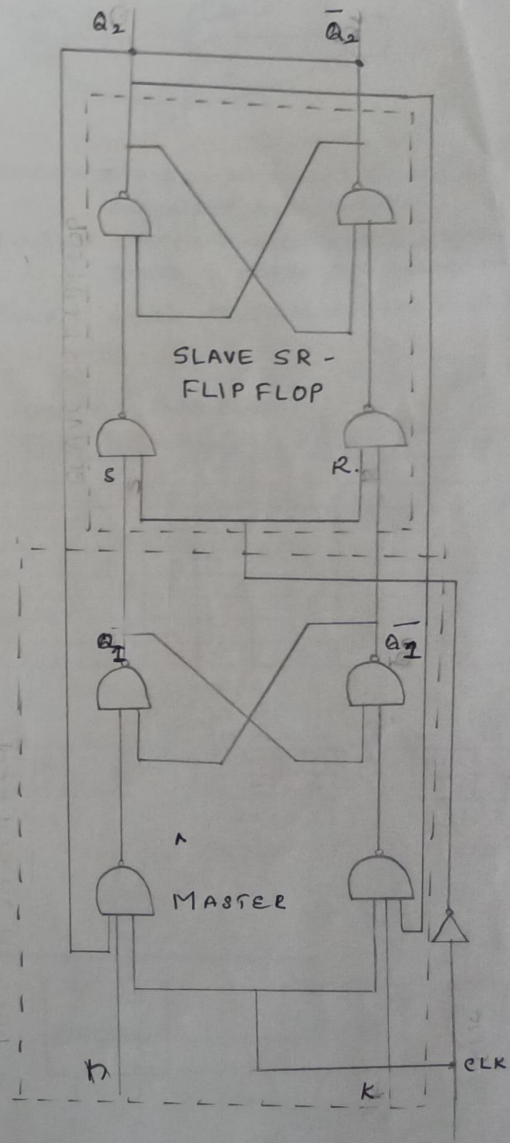


T	Q_{n+1}	state
0	Q_n	No change
1	$\bar{Q}_n$	Toggles

	$\bar{Q}_n$	Q_n
$\bar{T}$	0	1
T	1	0

Characteristic Equation $\Rightarrow$ $Q_{n+1} = T\bar{Q}_n + \bar{T}Q_n$

MASTER SLAVE JK FLIPFLOP



MOORE & MEALY MODEL

MOORE CIRCUIT

* CLOCKED SYNCHRONOUS CIRCUIT
* OUTPUT DEPENDS ONLY ON PRESENT STATE OF THE FLIP-FLOPS

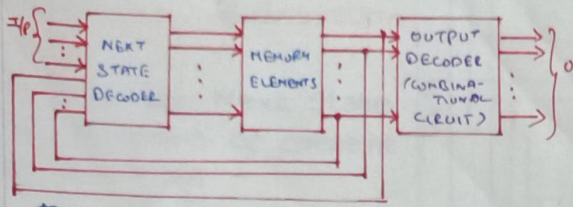


FIG: MOORE CIRCUIT WITH AN OUTPUT DECODER

NEXT STATE = f{ PRESENT STATE, INPUTS }

OUTPUT = f{ PRESENT STATE }

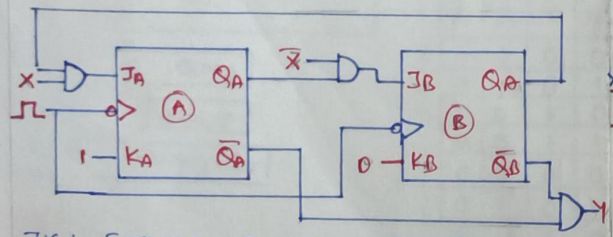


FIG: EXAMPLE OF MOORE CIRCUIT.

* BOTH FLIP FLOPS GETS CLOCK.
* CLOCKED CIRCUIT
* OUTPUT $Y = Q_B \cdot Q_A$ - PRESENT STATE ONLY
* INPUT CHANGES DOES NOT AFFECT THE OUTPUT
* REQUIRES MORE NUMBER OF STATES FOR IMPLEMENTING SAME FUNCTION.
* IT VARIES IN SYNCHRONISM WITH THE CLOCK INPUT.

MEALY CIRCUIT

* CLOCKED SEQUENTIAL CIRCUIT
* OUTPUT DEPENDS ON PRESENT STATE OF FLIP FLOP(S) & INPUT(S).

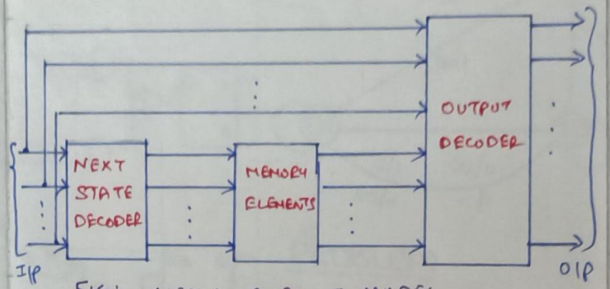


FIG: MEALY CIRCUIT MODEL

NEXT STATE = f{ PRESENT STATE, INPUTS }

OUTPUT = f{ PRESENT STATE, INPUTS }

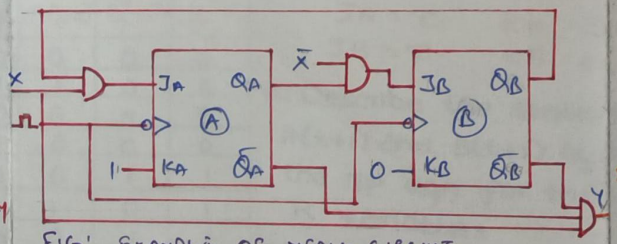


FIG: EXAMPLE OF MEALY CIRCUIT.

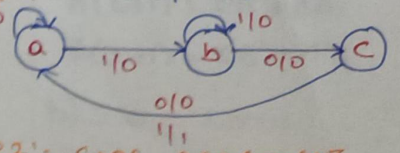
* CLOCK - BOTH FLIP FLOPS
* CLOCKED CIRCUIT
* OUTPUT $Y = Q_B \cdot \bar{Q}_A$ → PRESENT STATE & INPUTS
* INPUT CHANGES AFFECT THE OUTPUT OF THE CIRCUIT
* IT REQUIRES LESS NUMBER OF STATE FOR IMPLEMENTING SAME FUNCTION.

DESIGN 101 SEQUENCE DETECTORS

MEALY MACHINE

INPUT:- 0 1 1 0 1 0 1 0 1 1 0 0 1

STEP 1:- DEVELOP THE STATE DIAGRAM.



STEP 2:- CODE ASSIGNMENT

RULE 1:- STATES - SAME NEXT STATES - ADJACENT ASSIGNMENTS.

RULE 2:- STATES - NEXT STATE - SINGLE STATE - ADJACENT ASSIGNMENT.

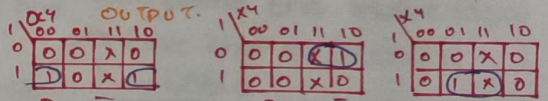
RULE 1 GIVEN PREFFERENCE OVER RULE 2

STEP 3:- MAKE PRESENT STATE / NEXT STATE TABLE.

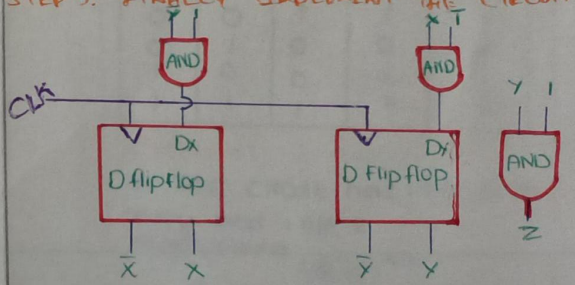
USE D - FLIP FLOP

PRESENT STATE		I/P	NEXT STATE		FLIP FLOP		O/P
X	Y		X'	Y'	D_X	D_Y	
0	0	0	0	0	0	0	0
0	0	1	1	0	1	0	0
0	1	0	0	0	0	0	0
0	1	1	0	0	0	0	1
1	0	0	0	1	0	1	0
1	0	1	1	0	1	0	0
1	1	0	X	X	X	X	X
1	1	1	X	X	X	X	X

STEP 4:- DRAW K MAPS FOR D_X, D_Y & Z



STEP 5: FINALLY IMPLEMENT THE CIRCUIT



ANALYSIS OF SEQUENTIAL CIRCUITS

Analysis - What a given Circuit will do under Certain operating Conductors

State equation or Transition equation

→ Specifies the Next State as a function of present State as input

Steps to Analyse a clocked Sequential Circuit

Sequential Circuit if NOT given obtain if from state & QNs

↓

State & QNs

↓

State Table

↓

State Diagram

PROBLEM

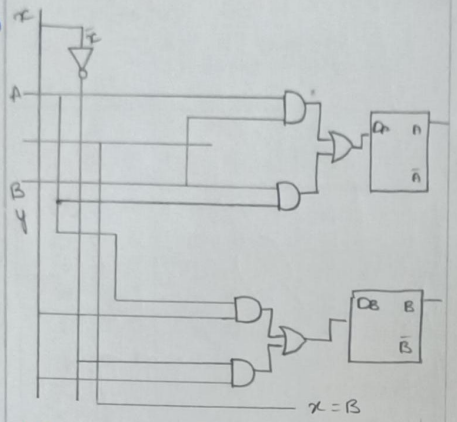
A Sequential Circuit with 2D F/F A and B 2 inputs x,y, 1o/p Z is specified by its for Next State O/P equations.

$A(t+1) = x'y + xA$

$B(t+1) = xA + x'B$; $Z = B$

a) Draw the logic diagram
b) List the State table
c) Draw the State diagram.

a) Logic Diagram

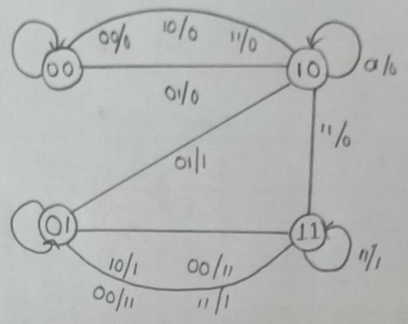


b) State table :-

A	B	x	y	A(t+1)	B(t+1)	Z
0	0	0	0	0	0	0
0	0	0	1	1	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	0
0	0	0	0	0	1	1
0	1	0	1	1	0	1
0	1	1	0	0	1	1
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	1	0
1	1	0	0	0	1	1
1	1	0	1	0	1	1
1	1	1	0	0	1	1
1	1	1	1	1	1	1

A,B - States

x & y - I/Ps



PROBLEM

A Sequential Circuit has 2 Jk F/F, A,B are input x Jk Circuit is described by the for F/F input eqn

$JA = x$, $KA = B$

$JB = x$, $KB = A$

a) Describe the State eqn A(t+1) and B(t+1) by Substitut the I/P eqn for the J and K variables

b) Draw the State table State diagram.

a) State eq'n for Jk F/F

$A(t+1) = JA' + K'A$

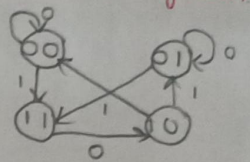
Sub JA, KA F/F

$A(t+1) = xA' + BA$

$B(t+1) = xB' + A'B$

b) State table

c) State diagram



Jk F/F is used

a) Either State eqn can be used

b) Executation tables can be used to find Next State

PROBLEM

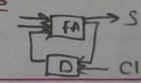
Complete the State table

→ Use the excitation table

A	x	A	JA	KA
0	0	1	1	X
0	1	0	0	X
1	0	0	X	1
1	1	1	X	0

Problem :

A Seq Circuit has 1 F/F 2 I/P x & y and 1 O/P Z State table



SYNCHRONOUS COUNTER

COUNTER
- ★ Counts clock pulses.
- ★ Group of flip flops.

TYPES
- ★ Synchronous
- ★ Asynchronous

DESIGN OF COUNTERS
- ★ Identify the number of variables for the given problem
- ★ Construct state table using flip flop excitation table
- ★ Get state equation for flipflop i/p variable using k-map
- ★ Construct sequential diagram using flip flops.

EXCITATION TABLE OF ALL FLIP FLOPS

Q(t)	Q(t+1)	J K	S R	T	D
0	0	0 X	0 X	0	0
0	1	1 X	1 0	1	1
1	0	X 1	0 1	1	0
1	1	X 0	X 0	0	1

EXAMPLE:1
→ Design a 3 bit Synchronous counter using T-Flip Flop

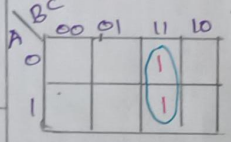
ANS:- No. of variable is 3
Hence use 3 FF

2) State Table

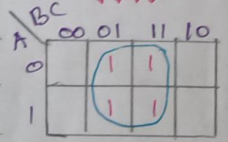
PS A B C	NS A B C	T_A T_B T_C
0 0 0	0 0 1	0 0 1
0 0 1	0 1 0	0 1 1
0 1 0	0 1 1	0 0 1
0 1 1	1 0 0	1 1 1
1 0 0	1 0 1	0 0 1
1 0 1	1 1 0	0 1 1
1 1 0	1 1 1	0 0 1
1 1 1	0 0 0	1 1 1

3) State eqn.

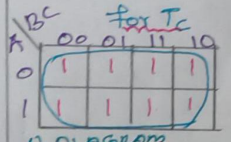
for T_A



for T_B

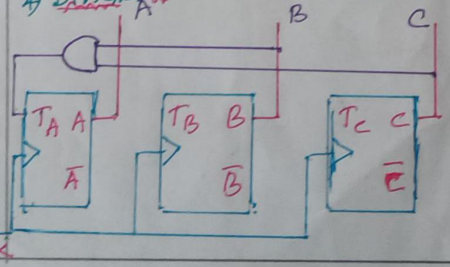


for T_C



$\therefore T_A = BC$
$T_B = C$
$T_C = 1$

4) DIAGRAM



DESIGN OF COUNTER FOR RANDOM SEQUENCE
→ Design a syn. counter for the following sequence 0, 3, 2, 7, 6 ... using D FF

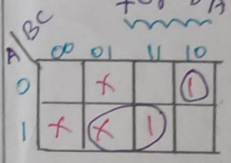
ANS:

State Table

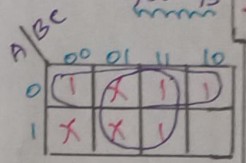
Seq.	PS A B C	NS A B C	D_A D_B D_C
0	0 0 0	0 1 1	0 1 1
3	0 1 1	0 1 0	0 1 0
2	0 1 0	1 1 1	1 1 1
7	1 1 1	1 1 0	1 1 0
6	1 1 0	0 0 0	0 0 0

→ 1, 4, 5 :→ don't cares

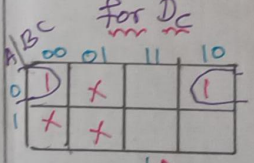
for D_A



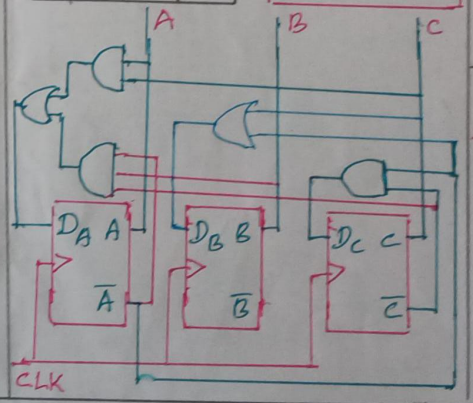
for D_B



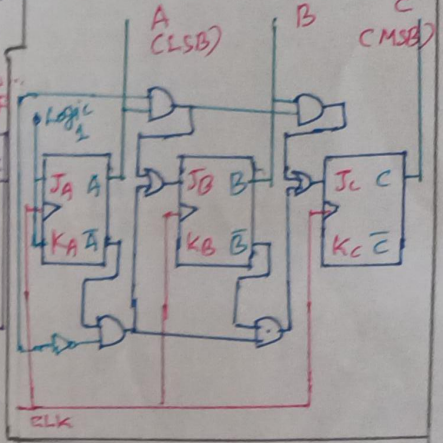
for D_C



$\therefore D_A = A'BC' + AC$
$D_B = A' + C$
$D_C = A'C'$



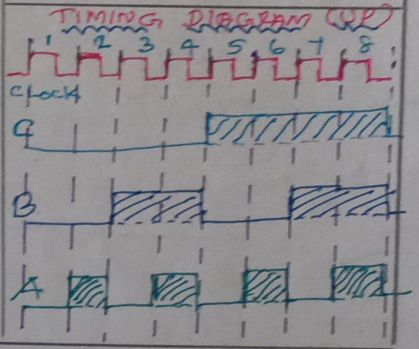
BINARY UP-DOWN COUNTER



STATE TABLE

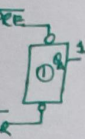
UP COUNTER C B A	DOWN COUNTER C B A
0 0 0	1 1 1
0 0 1	1 1 0
0 1 0	1 0 1
0 1 1	1 0 0
1 0 0	0 1 1
1 0 1	0 1 0
1 1 0	0 0 1
1 1 1	0 0 0

TIMING DIAGRAM (UP)



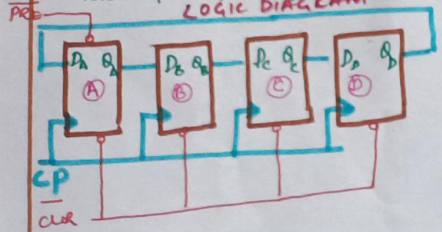
RING COUNTER

* O/p of each stage - connected to Yp of next



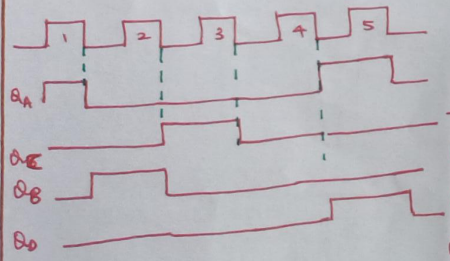
* $\overline{CLR}$ / $\overline{PRE}$ | ⇒ $Q_A = 1$, $Q_B = Q_C = Q_D = 0$
* FIRST CLOCK PULSE $Q_B = 1$ other 0
* Next CP - '1' shifted around

LOGIC DIAGRAM



CP	Q_A	Q_B	Q_C	Q_D
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1

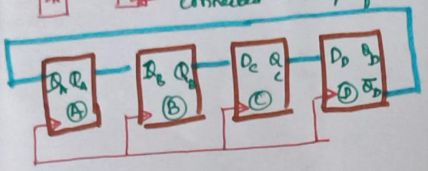
TIMING SEQUENCE



JOHNSON COUNTER

⇓

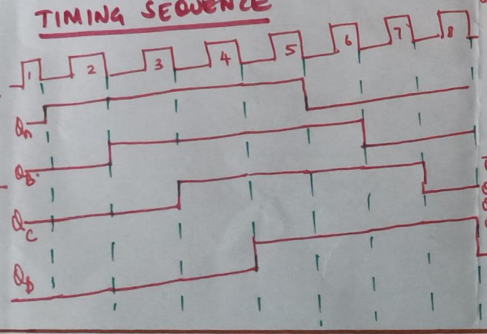
* K Bit RING | Shift | Twisted ring | Switch Tail counter
* O/p(Q) is connected to D Yp of next stage
* O/p($\overline{Q}$) of last FF - connected to D I/p of first FF



CP	Q_A	Q_B	Q_C	Q_D
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

* Initially all FF=0 $Q_A = Q_B = Q_C = Q_D = 0$
* Next CP: $Q_A = \overline{Q_D}$
* CLR1 - $Q_A = 1$ $Q_B = Q_C = Q_D = 0$
  CLR2 - $Q_A = 1$ $Q_B = 1$ $Q_C = Q_D = 0$
* After 8 states - sequence repeats

TIMING SEQUENCE



MOD 10 COUNTER

- Divide by 10
- Decade counter - 10 states
- DECIMAL COUNTER / BCD SYN COUNTER

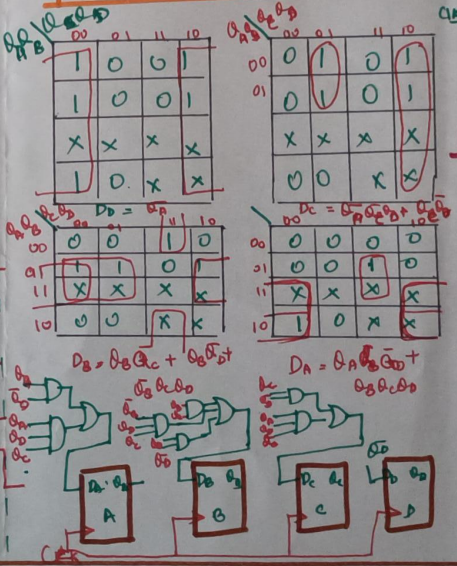
Step 1: FIND NO: FF

$2^n > N$ $N = 10 \Rightarrow n = 4$

Step 2: Type of FF D

Step 3: Excitation table Yp of FF

PRESENT STATE $Q_A Q_B Q_C Q_D$	NEXT STATE $Q_{A+1} Q_{B+1} Q_{C+1} Q_{D+1}$
0 0 0 0	0 0 0 1
0 0 0 1	0 0 1 0
0 0 1 0	0 0 1 1
0 0 1 1	0 1 0 0
0 1 0 0	0 1 0 1
0 1 0 1	0 1 1 0
0 1 1 0	0 1 1 1
0 1 1 1	1 0 0 0
1 0 0 0	1 0 0 1
1 0 0 1	0 0 0 0



RIPPLE COUNTER

* Asynchronous Counter - Clock ripples
* O/p of stage 1 is the CLK for next stage
  Yp of 1 : O/p Y - Reset for all FF

BCD RIPPLE : BCD - (0-9)

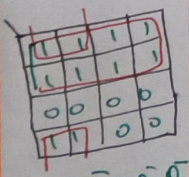
Step 1: NO: FF $N = 10$ $2^n \geq 10$ $n = 4$

Step 2: Type of FF - JK

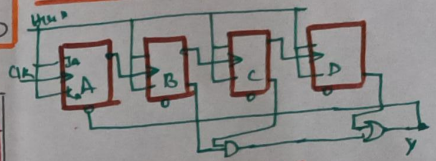
Step 3: Truth Table

CLK	$Q_D Q_C Q_B Q_A$	Y
0	0 0 0 0	1
1	0 0 0 1	1
2	0 0 1 0	1
3	0 0 1 1	1
4	0 1 0 0	1
5	0 1 0 1	1
6	0 1 1 0	1
7	0 1 1 1	1
8	1 0 0 0	1
9	1 0 0 1	1
Other possibilities - 0		Y = 0

K map

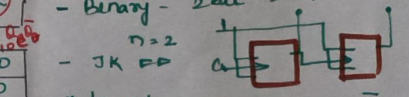


$Y = \overline{Q_D} + \overline{Q_C} \overline{Q_B}$

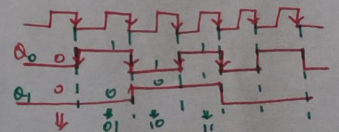


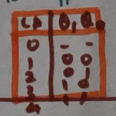
BINARY RIPPLE COUNTER

- Binary - 2 bit ⇒ 4 states $2^n \geq N$ $n = 2$
- JK FF



* for JK ⇒ J=1 K=1 $Q_{n+1} = \overline{Q_n}$ toggled at negative going CP. so Q_0 toggles at CP & Q_1 toggles at Q_0





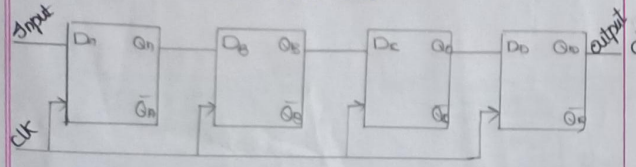
Counter counts 00, 01, 10, 11 & restarts to 00.

SHIFT REGISTERS

Shift Registers

* Flip Flop can be used to store a single bit
* N Flip flops are to be connected
* It is a group of flip-flops used to store multiple bits of data.

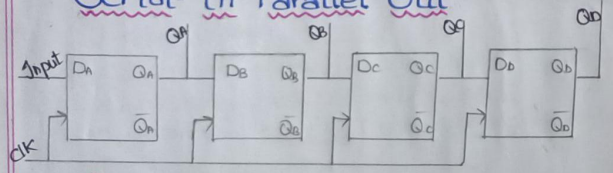
Serial In Serial Out



Truth table

I/P Din	CLK	QA	QB	QC	QD	O/P Dout
1	1	1	0	0	0	0
1	2	1	1	0	0	0
1	3	1	1	1	0	0
1	4	1	1	1	1	1

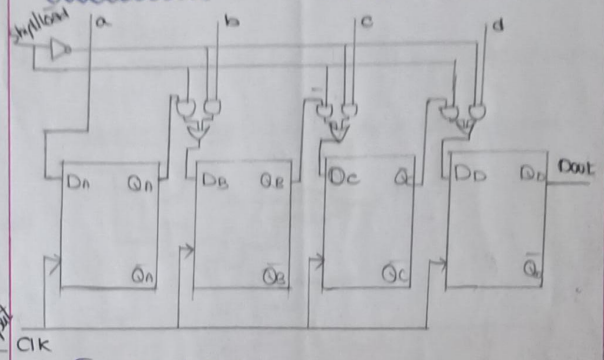
Serial In Parallel Out



Truth table

Input Din	CLK	Output QA	QB	QC	QD
1	1	1	0	0	0
1	2	1	1	0	0
1	3	1	1	1	0
1	4	1	1	1	1

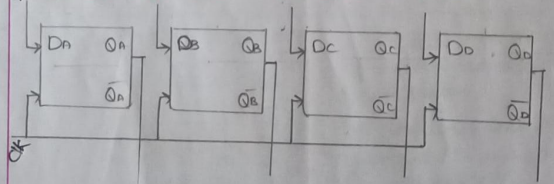
Parallel In - Serial Out



Truth table

CLK Pulse	QA	QB	QC	QD
0	1	0	0	1
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1

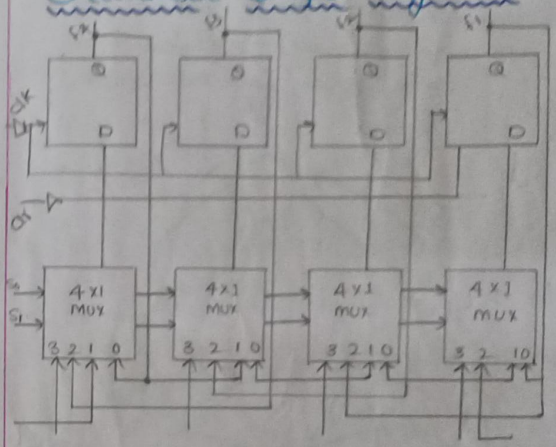
Parallel In - Parallel Out



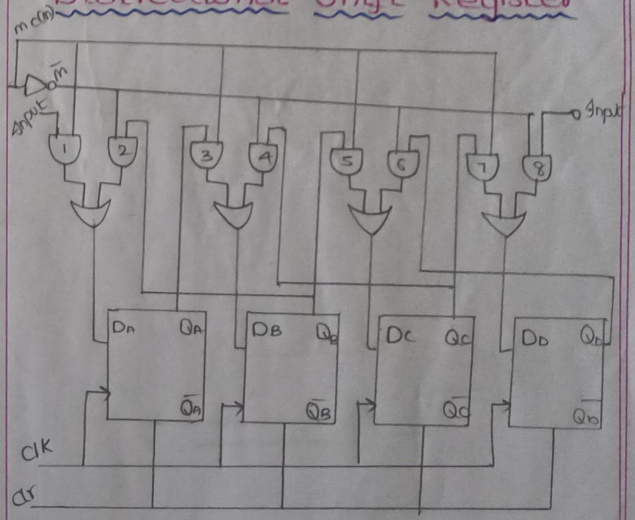
Truth table

CLK Pulse	QA	QB	QC	QD
0	0	0	0	0
1	1	1	0	1

Universal Shift Registers



Bidirectional Shift Register



Asynchronous Sequential Circuit

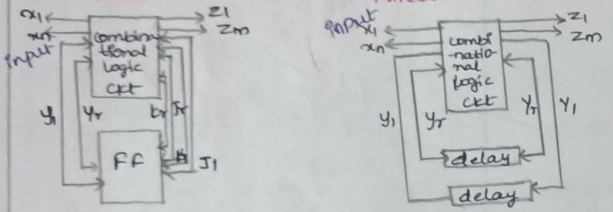
Asynchronous Sequential circuit:

Asynchronous Seq. circuit :-

1. Memory element - unclocked flipflop or delay elements. 2. Input changes - memory element changes
3. state change - delay time

Types of Synchronous circuits
- combinational circuit and delay element.

Asynchronous ckt
- Pulse mode ckt
- Fundamental mode ckt



Excitation variables : J_1-J_r, t_1-t_r, Y_1-Y_r

Secondary variables : y_1-y_r

Input variables x_1-x_n ; output variable Z_1-Z_m.

* Input variable changes, Secondary variable does not change - delay.
* Excitation variable - next state of the ckt.
* at steady state (stable) $Y_i=y_i$
* unstable $Y_i \neq y_i$ (continuous transition)

For proper operation:-
1. only one I/P variable change at a time.
2. Input - wait to react stable state
3. time between two input - longer than to react stable state

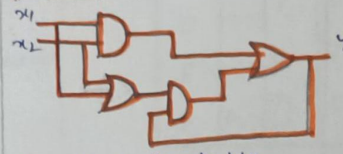
Analysis:-

1. Determine next secondary state and output equations.
2. State table 3. transition table 4. output map

Problems:-

An asynchronous ckt given by $Y=x_1x_2+(x_1+x_2)y$, $Z=Y$.

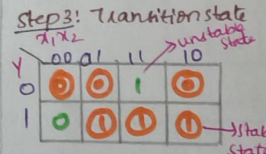
Solution:- Step1: Draw the logic diagram



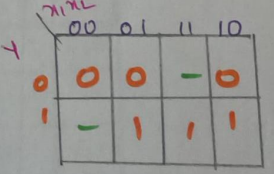
Step2: State table

Present state			Next state	Stable state	Output z
y	x_1	x_2	Y	Yes/NO	z
0	0	0	0	Yes	0
0	0	1	0	Yes	0
0	1	1	1	NO	1
0	1	0	0	Yes	0
1	0	0	0	NO	0
1	0	1	1	Yes	1
1	1	1	1	Yes	1
1	1	0	1	Yes	1

Step3: Transition state

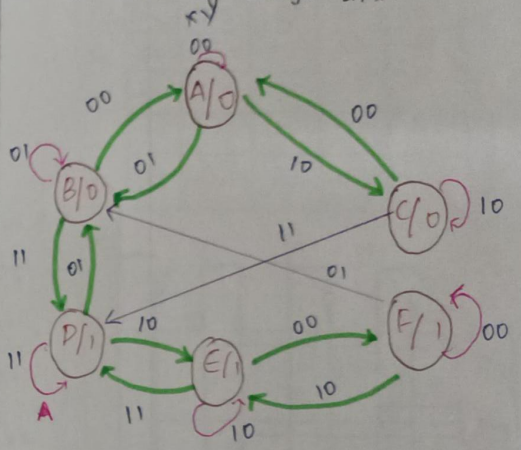


Step4: output map
* output mapped for all stable states
* For unstable states output is mapped unspecified.

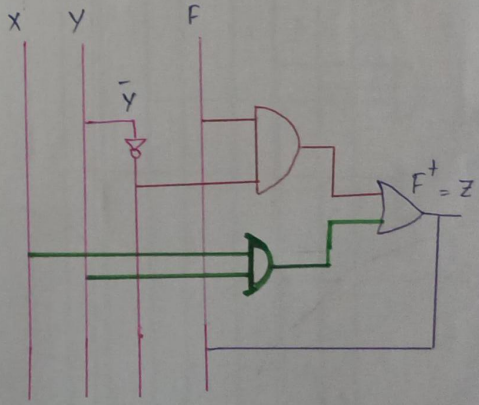


Steps:- behaviour of the ckt.
* The circuit gives carry output of the full circuit.

Draw the state diagram for the logic assume a asynchrone Sequential ckt with two inputs x and 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 any change in x.



LOGIC DIAGRAM :-



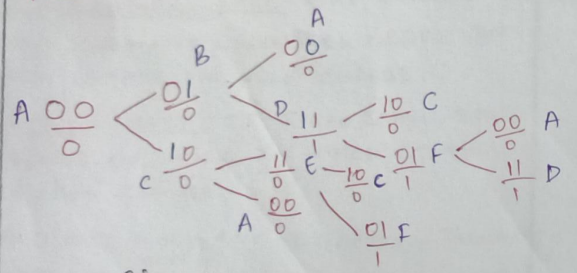
DESIGN OF ASYNCHRONOUS SEQUENTIAL CIRCUIT

Design of asynchronous sequential circuit

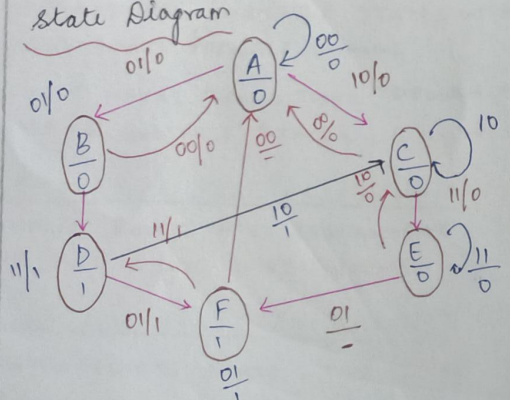
1. State diagram 2. Primitive table
3. state assignment 4. Analyse primitive table

Problems :- Design an asynchronous sequence circuit has two inputs X_2 and X_1 and one of p Z. when $X_1 = 0$ the o/p Z is 0 The first change in X_2 occurs while X_1 is 1 will cause output Z to be 1. The output Z will remain 1 until X_1 returns to 0

Step-1 State diagram Input $X_2 X_1$
Flow Table



State Diagram



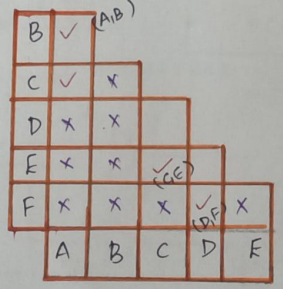
Step-2 :- Primitive flow table

(Circle) stable state

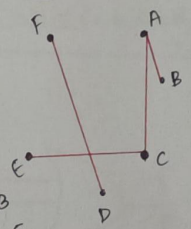
two bit change don't care

Present state	00	01	11	10
A	(A) 0	B, 0	–, –	C, 0
B	A, 0	(B) 0	D, –	–, –
C	A, 0	–, –	E, 0	(C) 0
D	–, –	F, 1	(D) 1	C, –
E	–, –	F, –	(E) 0	C, 0
F	A, –	(F) 1	D, 1	–, –

Implicit table Compare the two rows A, B of Primitive flow table and put ✓ if same x if different.



Merger graph



Reduce Primitive Flow Table :-

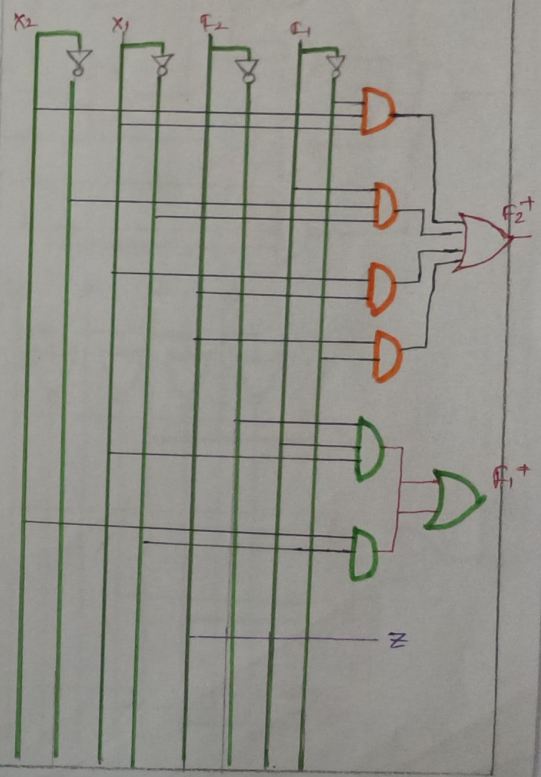
	00	01	11	10
A = B	A, 0	B, 0	D, –	C, 0
C = E	A, 0	F, –	E, 0	C, 0
D = F	A, –	F, 1	D, 1	C, –

Step 3 : State assignment

	00	01	11	10
00	S_0	S_0, 0	S_0, 0	S_2, –	S_1, 0
01	S_1	S_0, 0	S_2, –	S_1, 0	S_1, 0
10	S_2	S_0, –	S_2, 1	S_2, 1	S_1, –

Assume
A = B ⇒ S_0
C = E ⇒ S_1
D = F ⇒ S_2

$F_2 F_1$	00	01	11	10
S_0	00	00, 0	00, 0	10, –	01, 0
S_1	01	00, 0	11, –	01, 0	01, 0
	11	–, –	10, –	–1–	01, –
S_2	10	00, –	10, 1	10, 1	–1–



RACES AND HAZARDS

RACES

* TWO OR MORE STATE VARIABLES CHANGE
* CHANGE IN INPUT VARIABLES.
* STATE VARIABLE CHANGES - UNPREDICABLE MANNER

Eg: IF STATE VARIABLE CHANGES FROM 00 → 11 FOR A GIVEN INPUT - DELAY.

1st / 2nd VARIABLE CHANGE FASTER

00 → 01 → 11

00 → 10 → 11

* NON CRITICAL RACE - IF FINAL STABLE STATE DOES NOT DEPEND ON ORDER IN WHICH STATE VARIABLE CHANGE
* CRITICAL RACE: IF FINAL STABLE STATE DEPENDS ON ORDER IN WHICH STATE VARIABLE CHANGE.
* AVOID RACE - CIRCUIT DIRECTED - THROUGH INTERMEDIATE UNSTABLE STATES WITH UNIQUE STATE VARIABLE CHANGE.
* CIRCUIT MAKES TRANSITIONS THROUGH A SERIES OF UNSTABLE STATES - CYCLE.

CRITICAL RACE FREE ASSIGNMENT

- SHARED ROW STATE ASSIGNMENT
- ONE HOT STATE ASSIGNMENT

SHARED ROW STATE ASSIGNMENT

INTERMEDIATE STATE - ONE STATE VARIABLE CHANGE - ONE TIME WHEN STATE TRANSITIONS.

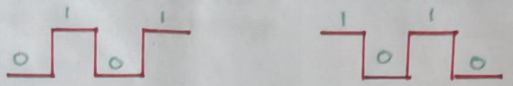
HAZARDS

STATIC
- STATIC 0
- STATIC 1

DYNAMIC

- UNWANTED SWITCHING TRANSIENT / SPIKE / GLITCH
- APPEAR AT THE OUTPUT OF THE CIRCUIT.
- CAUSE - DIFFERENCE IN PROP DELAY AT DIFFERENT PATH.

DYNAMIC HAZARD



CIRCUIT GOES THROUGH THREE / MORE TRANSIENTS

ELIMINATING HAZARDS

CONSIDER $F(A,B,C) = \Sigma_m(1,5,6,7)$

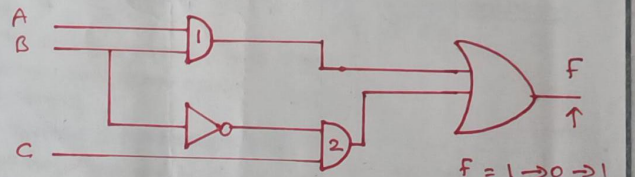
A \ BC	00	01	11	10
0	0	1	0	0
1	1	1	1	1

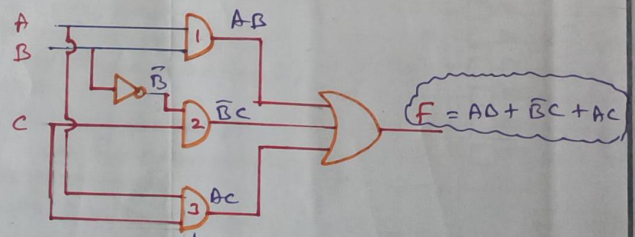
→ HAZARD FREE

ACTIVITY

FIND A CIRCUIT THAT HAS NO STATIC HAZARDS & IMPLEMENTS THE BOOLEAN FUNCTION

$F(A,B,C,D) = \Sigma(0,2,6,7,8,10,12)$





↳ INDEPENDENT OF B INPUT.

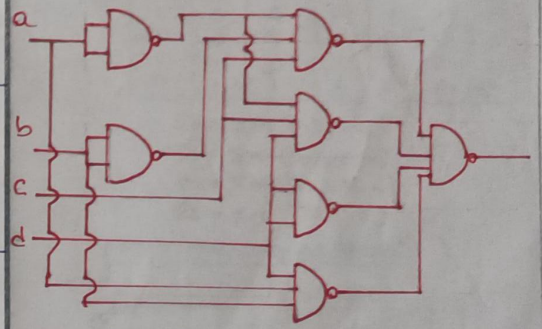
FIND A STATIC AND DYNAMIC HAZARD FREE REALIZATION FOR THE FOLLOWING FUNCTION USING (i) NAND, (ii) NOR

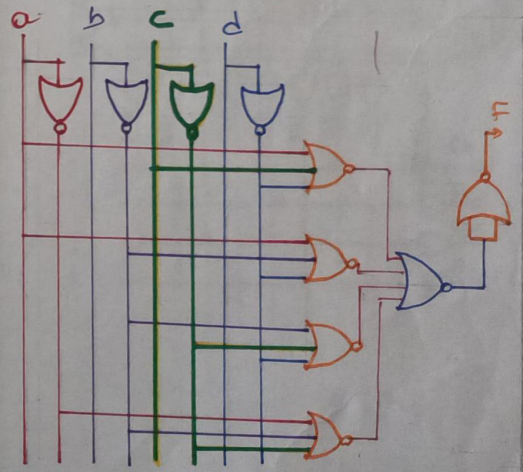
$F(a,b,c,d) = \Sigma_m(1,5,7,14,15)$

$F = \bar{a}\bar{c}d + \bar{a}bd + bcd + abc$

ab \ cd	00	01	11	10
00		1		
01		1	1	
11		1	1	
10				

1. COMPLEMENT INPUT
2. COMPLEMENT OUTPUT
3. REPLACE NAND GATE BY NOR GATE.



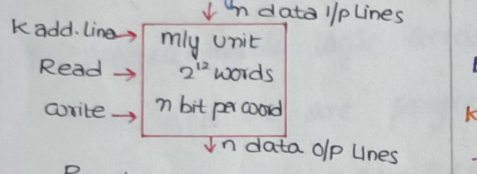


Memory Devices

Memory Devices < ROM (Read only m/y) / RAM (Random access m/y)

Memory: 1 byte = 8 bits
1 word = 2 bytes.

RAM Block Diagram



Read & Write operation

* Store word into m/y:
  - Apply Binary add (Desired word) ⇒ Add lns
  - Data bits Stored m/y to the data I/p lines
  - Activate the write I/p

* Store word out of m/y:
  - Apply Binary Add (DW) ⇒ Add. lines
  - Activate the read I/p

RAM memory:-
- Each word location is seperated with space
- Each word occupying one particular location.

Types of RAM < Dynamic RAM / Static RAM

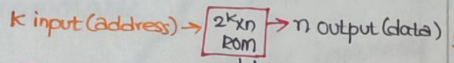
- DRAM loses its data in a very short time (milli sec)

ROM memory:-
- Read Only Memory
- Non volatile memory
- Used for permanent storage
- Random Access Property

ROM

Binary information must be specified by the designer
↓
Embedded in the unit of from the required interconnection pattern
↓
Data stays with in the unit even when power is turned off and on again

Block diagram



Types of ROM < MROM / PROM / EPROM / EEPROM

MROM (Hashed ROM)
- Hardwired device
- Pre programmed set of data as instruction.

PROM (programmable ROM)
- Data can be modified only once by a user.
- Inside the prom chip these are small fuses which are burnt open during programming.

EPROM (Erasable programmable ROM)
- Data can be erased by exposing it to ultra violet light for a duration of upto 40 min
- During programming an electric change is trapped in an insulated gate region

EEPROM (Electrically Erasable PROM)
- EEPROM Can be programmed & erased electrically
- 10,000 times can be reprogram ed
- Erasing required 4 to 10 ms

Applications of Static ROM

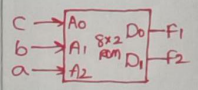
→ Uses transistors, No Capacitors.
→ Transisters do not require power to prevent leakage
→ SRAM not a required refresh on regular basis.

Q1 ROM Problem:

Using ROM realize the following expression $f_1(a,b,c) = \Sigma m(0,1,3,5,7)$
$f_2(a,b,c) = \Sigma m(1,2,5,6)$

Sol:- $2^3 = 8$ minterms

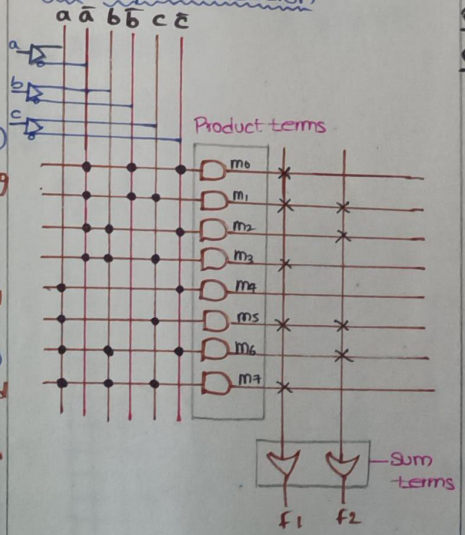
Block diagram



Truth table

A_2	A_1	A_0	F_1	F_2
0	0	0	1	0
0	0	1	1	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0

ROM Implementation



Q2 ROM Problem:

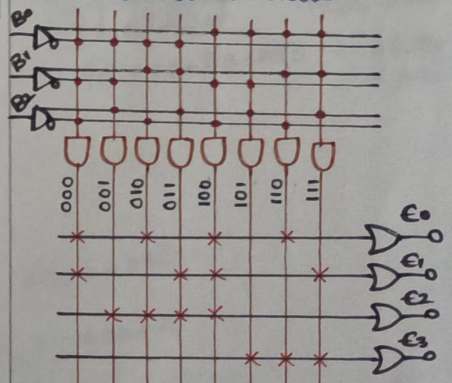
Design a Combinational using a PROM. the circuit accepts 3 bit binary number and generates its equivalent excess-3 Code.

Sol:- $2^3 = 8$ (0-7)

Input			Output			
B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	1	1
0	0	1	0	1	0	0
0	1	0	0	1	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	1
1	0	1	1	0	0	0
1	1	0	1	0	0	1
1	1	1	1	0	1	0

$E_0 = \Sigma m(0,2,4,6)$
$E_1 = \Sigma m(0,3,4,7)$
$E_2 = \Sigma m(1,2,3,4)$
$E_3 = \Sigma m(5,6,7)$

PROM Implementation



Memory devices

Memory devices

* PLA
* PAL
* EPROM
* PROM
* ROM
* RAM

Programmable logic array (PLA)

* Programmable logic array (PLA)

In programmable logic array "AND and OR" are programmable

Example:- $F_1 = \Sigma m(3,5,7) = F_1(A,B,C)$

$F_2 = \Sigma m(4,5,7) = F_2(A,B,C)$

given functions

$F_1(A,B,C) = \Sigma m(3,5,7)$

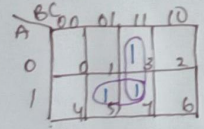
$F_2(A,B,C) = \Sigma m(4,5,7)$

PLA → Will be implemented using AOI logic

A - AND, O - OR

I - Invert

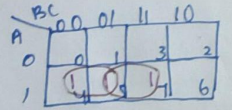
K-map for F_1



$F_1 = BC + AC$

$F_1 = C(A+B)$

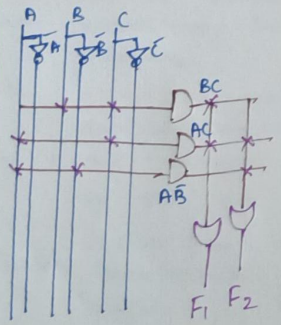
K-map for F_2



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

$F_2 = A(C+\bar{B})$

Logic daigram



$F_1 = BC + AC$

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

This is the programmable logic array for given functions

Programmable array logic (PAL)

* programmable array logic (PAL)

In progammable array logic "AND is programmable" but OR is fixed

PAL → Will be implemented using AOI logic

A-AND, O-OR, I-Invert

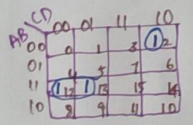
Example:- $W(A,B,C,D) = \Sigma m(2,12,13)$

$X(A,B,C,D) = \Sigma m(7,8,9,10,11,12,13,14,15)$

$Y(A,B,C,D) = \Sigma m(0,2,3,4,5,6,7,8,10,11,15)$

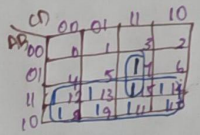
$Z(A,B,C,D) = \Sigma m(1,2,8,12,13)$

K-map for W



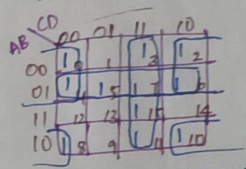
$W = \bar{A}B\bar{C}\bar{D} + AB\bar{C}$

K-map for X



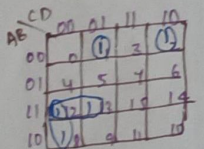
$X = BCD + A$

K-map for Y



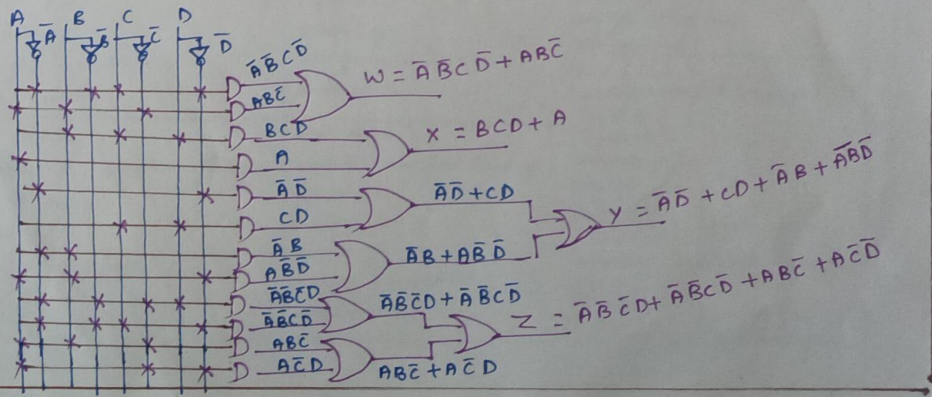
$Y = \bar{A}\bar{D} + CD + \bar{A}B + A\bar{B}\bar{D}$

K-map for Z



$Z = \bar{A}\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D} + AB\bar{C} + A\bar{C}\bar{D}$

Logic daigram



Verilog HDL

VERILOG HDL :

★ Verification logic HDL

★ allows different levels of abstraction to be mixed in the same model.

Built in data types :

NETS : represents structural connectivity

Registers : abstract storage elements

wire : connectivity with no logical behaviour.

Variable type	input	output	in out
Net	Yes	Yes	Yes
Register	No	Yes	No

Memory declaration : (2D array)

★ provides an extention to the register variable declaration

Eg : - reg [3:0]

strings : sized by procedural assignment statement.

constant : Declared by the keyword parameter.

Eg : parameter byte - size = 8

operators :

⇒ Arithmetic operators :
+, −, *, / (Divide), % (modulus)

⇒ Logical operators :
&& (AND), || (OR), ! (negation)

⇒ Bitwise operators :
^ (EX-OR), ~^ (EX-NOR), & (AND)

⇒ Relational operators :
>, <, >=, <=

⇒ Equality operators :
== (logic equality), === (case equality)

⇒ shift operators :
<< (left shift), >> (Right shift)

Module name
port list, port declaration (if port present)
parameters (optional)

Declaration of wire register and variables

Data flow statements (assign)

Instantiation of lower level models

always & initial blocks. All Block → behavioural statements

Tasks & Functions.

end module statements.

Modelling styles :

1) Behavioural model level - describe how the hardware should behave without any interface to digital hardware.

2) Data flow : provides the means of describing combinational circuits by their functional feature rather than by their gate structures.

3) Gate level : each gate and its interconnections are explicitly specified.

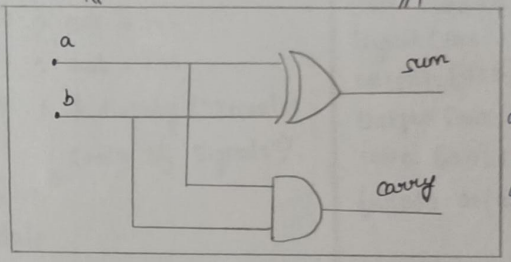
★ Registers normally represents storage elements, they need clock signals to update their output value.

★ Data flow modelling :

```
module half_adder (sum, carry, a, b);
input a, b;
output sum, carry;
assign s = a ^ b;
assign c = a & b;
end module
```

★ Gate level modelling :

```
Module half_adder (sum, Carry, a, b);
input a, b;
output sum, carry;
xor sum, (sum, a, b);
and carry, (carry, a, b);
end module
```



★ Initial statements are executed one and always statements are executed repetitively.

★ Procedural and continuous assignments can co-exist with in a module.

★ Behavioural level modelling :

```
module Half_adder (sum, carry, a, b)
input a, b;
output sum, carry;
always @ (a or b)
assign {carry, sum} = a+b;
end module
```

VERILOG PROGRAM FOR LOGIC GATES.

AND GATE :

```
Module and_dataflow (y, a, b);
input a, b;
output y;
assign y = a & b;
end module
```

NAND GATE :

```
module nand gate (a, b, y);
input a;
input b;
output y;
assign y = ~ (a & b);
end module
```

OR GATE :

```
Module or_dataflow (a, b, y);
input a, b;
output y;
assign y = (a | b);
end module
```

NOR GATE :

```
Module nor gate (a, b, y);
input a;
input b;
output y;
assign y = ~ (a | b);
end module
```

NOT GATE :

```
module not_dataflow (a, y);
input a;
output y;
assign y = ~ a;
end module
```

XOR GATE :

```
module xorgate (a, b, y);
input a;
input b;
output y;
assign y = a ^ b;
end module
```

XNOR GATE :

```
module xnorgate (a, b, y);
input a;
input b;
output y;
assign y = ~ (a ^ b);
end module
```

VERILOG PROGRAM FOR COMBINATIONAL CIRCUITS

Half Adder

```
module halfadder(a,b,s,c);
input a,b;
output s,c;
assign s=a^b;
assign c=a&b;
endmodule
```

Full Adder

```
module fulladder(a,b,c,sum,carry);
input a,b,c;
output sum,carry;
assign sum = a^b^c;
assign carry = (a&b)|(b&c)|(c&a);
endmodule
```

Half subtractor

```
module halfsub (a,b,d,br);
input a,b;
output d,br;
wire d,br;
assign d= a^b;
assign br= (~a)&b;
endmodule
```

Full subtractor

```
module fullsub(a,b,bin, diff, bout);
input a,b, bin;
output diff, bout;
wire diff, bout;
assign diff = bin^a^b;
assign bout = (~a&bin)|(~a&b)|(b&bin);
endmodule
```

2 to 4 Decoder

```
module decoder24_assign (en,a,b,y);
input en,a,b;
output [3:0] y;
wire enb, na, nb;
assign enb = ~en;
assign na = ~a;
assign nb = ~b;
assign y[0] = ~(enb&na&nb);
assign y[1] = ~(enb&na&b);
assign y[2] = ~(enb&a&nb);
assign y[3] = ~(enb&a&b);
endmodule
```

Priority Encoder

```
module priorityencoder (en,i,y);
input en;
input [7:0] i;
output [2:0] y;
assign y[2] = i[4]|i[5]|i[6]|i[7]&en;
assign y[1] = i[2]|i[3]|i[6]|i[7]&en;
assign y[0] = i[1]|i[3]|i[5]|i[7]&en;
endmodule
```

4:1 Multiplexer

```
module mux4to1(out, i0, i1, i2, i3,
               s1, s0);
output out;
input i0, i1, i2, i3;
input s1, s0;
reg out;
always @(s1 or s0 or i0 or i1 or i2
         or i3)
case ({s1, s0})
2'b00 : out = i0;
2'b01 : out = i1;
2'b10 : out = i2;
2'b11 : out = i3;
default : $display("Invalid
          Contn of Signals");
endcase
endmodule
```

1:4 Demultiplexer

```
module demux41(d0,d1,d2,d3,
               s0,s1,y);
input s0,s1,y;
output d0,d1,d2,d3;
reg d0,d1,d2,d3;
always @(s0,s1,y)
begin
case ({s1,s0})
2'b00: begin d0<=y; d1<=0; d2<=0;
       d3<=0; end
2'b01: begin d0<=0; d1<=y; d2<=0;
       d3<=0; end
2'b10: begin d0<=0; d1<=0; d2<=y;
       d3<=y; end
2'b11: begin d0<=0; d1<=0; d2<=0;
       d3<=y; end
endcase
end module
```

8 Bit Ripple Carry adder

```
module ripple8bitadder(a,b,
            Cin, Sum, Cout);
input [07:0]a;
input [07:0]b;
input Cin;
output [7:0]sum;
output Cout;
wire [8:0]c;
fulladd a1(a[0], b[0], Cin,
            sum[0], c[0]);
fulladd a2(a[1], b[1], c[0],
            Sum[1], c[1]);
fulladd a3(a[2], b[2], c[1],
            sum[2], c[2]);
fulladd a4(a[3], b[3], c[2],
            sum[3], c[3]);
fulladd a5(a[4], b[4], c[3],
            sum[4], c[4]);
fulladd a6(a[5], b[5], c[4]
            sum[5], c[5]);
fulladd a7(a[6], b[6], c[5],
            sum[6], c[6]);
fulladd a8(a[7], b[7], c[6],
            Sum[7], COut);
endmodule

modulefulladd (a,b,Cin,sum,Cout);
input a;
input b;
input Cin;
output sum, Cout;
assign sum = (a^b^Cin);
assign Cout = ((a&b)|(b&Cin)|(a&Cin));
```

VERILOG PROGRAM FOR SEQUENTIAL CIRCUITS

Verilog HDL Design of Sequential circuits:

UP counter:

```
module counter (clk, clrx, q);
input clk, clrx;
output [3:0] q;
reg [3:0] tmp;
always @ (posedge clk or posedge clrx)
begin
if (clrx)
temp <= 4'b0000;
else
tmp <= tmp + 1'b1;
end
assign q = tmp;
endmodule
```

Down Counter:

```
module counter (clk, s, q);
input clk, s;
output [3:0] q;
reg [3:0] tmp;
always @ (posedge clk)
begin
if (s)
tmp <= 4'b1111;
else
tmp <= tmp 1'b1;
end
assign q = tmp;
endmodule
```

Serial in Serial out shift Register:

```
module SISO (clk, rst, qin, qout, qoutbar);
input clk;
input rst;
input qin;
output qout;
output qoutbar;
wire qout1, qout2, qout3, qoutb1, qoutb2,
                                qoutb3;

dflipflop1 a_dflipflop_1 (clk, qin, rst, qout1, qoutb1);
dflipflop2 a_dfilpflop_2 (clk, qout1, rst, qout2,
                                qoutb2),
dflipflop3 a_dflipflop_3 (clk, qout2, rst, qout3,
                                qoutb3);
dflipflop4 a_dfilpflop_4 (clk, qout3, rst,
                                qout, qoutbar);
endmodule

module dflipflop1 (clk, Din, rst, Dout, Doutbar);
input clk;
input Din;
input rst;
output Dout;
output Doutbar;
reg Dout;
reg Doutbar;
always @ (posedge clk)
if (rst == 0)
begin
Dout <= Din;
Doutbar <= ~Din;
end
else
begin
Dout <= 1'b0;
Doutbar <= 1'b1;
end
endmodule
```

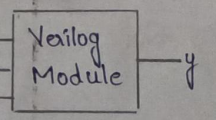
Parallel in serial out shift Register

```
module PISO (clk, rst, key, qin1, qin2,
      qin3, qin4, qout);
input clk;
input rst;
input key;
input qin1;
input qin2;
input qin3;
input qin4;
input qout;
reg qout1, qout2, qout3, qout;
always @ (posedge clk)
if (rst == 0)
begin
if (key)
begin
qout1 <= qin1;
qout2 <= qin2;
qout3 <= qin3;
qout <= qin4;
end
else
begin
qout1 <= qout;
qout2 <= qout1;
qout3 <= qout2;
qout <= qout3;
end
end
else
qout <= 1'b0
endmodule
//testbench

module PISO_tst ();
reg clk, key, rst, qin1, qin2, qin3,
                        qin4, qout;

PISO P1 (clk, rst, key, qin1, qin2, qin3,
                        qin4, qout);
initial
begin
clk = 1'b0;
rst = 1'b1;
key = 1'b1;
end
initial
begin
# 100 rst = 1'b0;
qin1 = 1'b0;
qin2 = 1'b1;
qin3 = 1'b0;
qin4 = 1'b1;
# 100 key = 1'b0;
#100
key = 1'b1;
qin1 = 1'b1;
qin2 = 1'b0;
qin3 = 1'b1;
qin4 = 1'b1;
# 100 key = 1'b0;
# 100 key = 1'b1;
qin1 = 1'b1;
qin2 = 1'b0;
qin3 = 1'b1;
qin4 = 1'b1;
# 100 key = 1'b0;
# 10000 $ stop;
end
always
#5 clk = ~clk;
endmodule
```

⇒ Module is main building Block in verilog

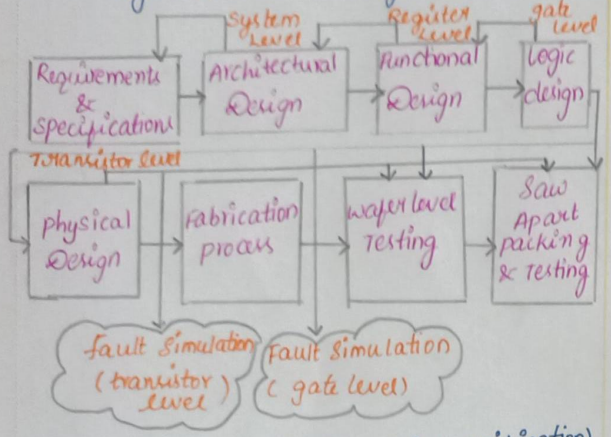


inputs - a, b, c output - y

Application And Simulation Tools

Example of digital System design

The integrated circuit design process



* Note all the simulation (design verification) -helps to ensure the design works and assists in debugging design errors.
* In order to simulate a circuit, we must describe it in a manner that can be interpreted and understood by the simulator.

Applications and advantages of digital Systems

→ A digital System is a combination of devices designed to manipulate logical information or physical quantities.

→ These devices are most often electronic, but they can also be mechanical, magnetic or pneumatic.

The chief reasons for the shift to digital technology are :

→ Digital systems are generally easier to design The circuits used in digital systems are switching circuits, where exact values of voltage or current are not important, only the range (High or Low) in which they fall.

→ Information Storage is easy This is accomplished by special devices and circuits that can latch onto digital information and hold it for as long as necessary, and mass Storage techniques that can store billions of bits of information in a relatively small physical Space.

→ Accuracy and precision are easier to main -tain throughout the System.

→ operations can be programmed

→ Digital circuits are less affected by noise

→ more digital circuitry can be fabricated on IC chips.

Design capture with Hardware Description Language

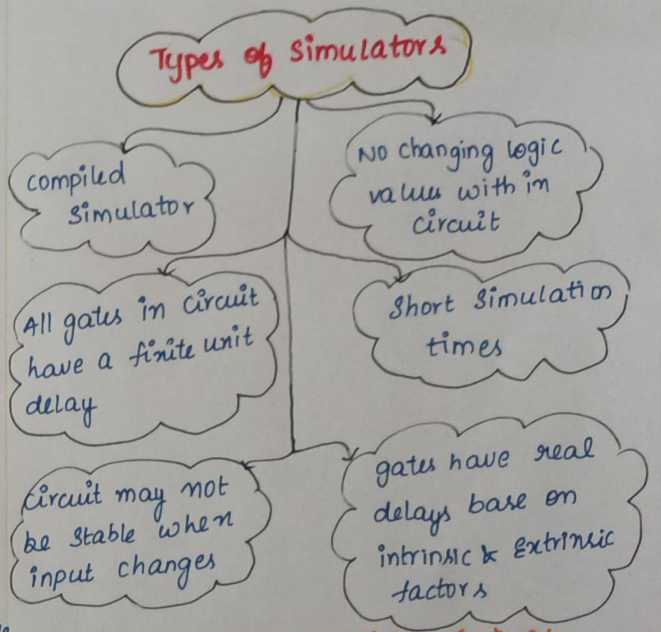
→ net list
- connections via signal name

→ Schematic
- connections Either Explicit or via signal name
- produces a netlist for simulation

→ Higher level language (VHDL or verilog)
- Synthesis to gate level netlist

→ All of these design descriptions can go to simulation for design verification

Types of Simulators



ongoing Research in digital Systems

→ Quantum computing
→ Quantum technology
→ composite Semiconductors
→ Devices Such as FINFET, High power MOSFETs
→ modelling of devices
→ Nano technology