

UNIT-I - Boolean Algebra and logic gates

Review of Number systems - Arithmetic operations - Binary Codes - Boolean Algebra and Theorems - Boolean Functions - Simplification of Boolean Functions using Karnaugh map and Tabulation method - Logic gates - NAND and NOR Implementations

Introduction

Digital signals:

Why are Binary Numbers used?

Almost all digital computers and systems are based on 2 state operations.

Some examples of 2 state devices are:

Switches (open or closed), magnetic cores

Punch cards (hole or no hole), Tape

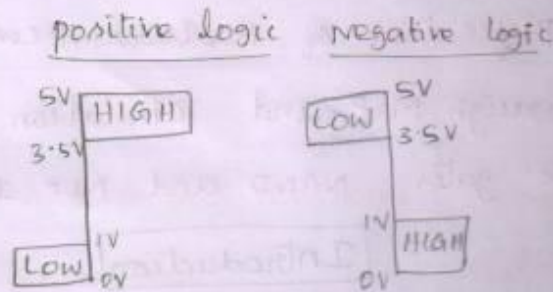
recorders (magnetize or nonmagnetize),

Transistor circuits (cutoff or saturation).

Hence it is convenient to use binary number when analyzing or designing digital circuits.

What is a Digital signal?

A digital signal has 2 discrete levels or values. These levels can be represented using the terms Low and HIGH.



Number Systems

The study of number s/m will be required to understand the digital computers. Various number s/m such as binary, octal, hexadecimal and decimal. All number s/m are known as positional number s/m, because each digit position is assigned with a particular weight.

Base Conversions:-

Binary to Decimal conversion:-

Any binary number can be converted into its equivalent decimal number using the weights assigned to each bit position.

Example (1): Find the eqt of binary

number $(11111)_2$

solution:

$$\begin{aligned}(x)_{10} &= 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ &= 16 + 8 + 4 + 2 + 1\end{aligned}$$

$$(x)_{10} = (31)_{10}$$

$$(11111)_2 = (31)_{10}$$

Example (2): $(0.101)_2$

solution:

$$\begin{aligned}(0.101)_2 &= 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\ &= 0 + \frac{1}{2} + 0 + \frac{1}{8} = 0.5 + 0 + 0.125 \\ &= (0.625)_{10}\end{aligned}$$

$$(0.101)_2 = (0.625)_{10}$$

4 bit binary numbers and corresponding decimal

Binary number			
B ₃	B ₂	B ₁	B ₀
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0

Decimal Number	
D ₁	D ₀
0	0
0	1
0	2
0	3
0	4
0	5
0	6
0	7
0	8

Example (2):- $(0.6875)_{10}$

$$\begin{array}{lcl} 0.6875 \times 2 & = & 1.3750 \\ 0.3750 \times 2 & = & 0.7500 \\ 0.7500 \times 2 & = & 1.5000 \\ 0.5000 \times 2 & = & 1.0000 \end{array}$$

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

Example (3):- $(0.85)_{10}$

$$\begin{array}{lcl} 0.85 \times 2 & = & 1.70 \\ 0.70 \times 2 & = & 1.4 \\ 0.40 \times 2 & = & 0.8 \\ 0.80 \times 2 & = & 1.6 \\ 0.60 \times 2 & = & 1.2 \\ 0.20 \times 2 & = & 0.4 \end{array}$$

$$(0.85)_{10} = (0.110110)_2$$

Decimal to Base-r:-

Decimal to octal (Base-8)

Decimal to hexadecimal (base-16)

Example (1):- Convert decimal $(153)_{10}$ to octal and hexadecimal.

Solution:- Base-8:-

$$\begin{array}{r} 8 \overline{) 153} \\ 8 \overline{) 19} - 1 \\ 8 \overline{) 3} - 3 \\ 0 - 2 \end{array}$$

Binary Number			
B ₃	B ₂	B ₁	B ₀
1	1	0	1
1	1	1	0
1	1	1	1

Decimal	
D ₁	D ₀
1	3
1	4
1	5

Decimal to Binary Conversion:-

Any decimal number can be converted into its eqt binary number. For integers the conversion is obtained by continuous division by 2 and keeping track of the remainder, while for fractional parts, the conversion is affected by continuous multiplication by 2 and keeping track of the integer generated.

Example 10: Convert $(41)_{10}$ to an eqt base-2 number.

$$\begin{array}{r}
 2 \overline{) 41} \\
 \underline{20 - 1} \\
 2 \overline{) 10 - 0} \\
 \underline{5 - 0} \\
 2 \overline{) 5 - 1} \\
 \underline{2 - 1} \\
 1 - 0
 \end{array}
 \quad \uparrow
 \quad \text{Answer: } (101001)_2$$

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

Base - 16:-

$$\begin{array}{r} 16 \overline{) 153} \\ 16 \overline{) 9} - 9 \uparrow \\ 0 - 9 \end{array} \quad (99)_{16}$$

$$\therefore (53)_{10} = (99)_{16}$$

Example (2):- Convert $(0.6875)_{10}$ to octal and hexadecimal.

Solution:-

Octal:-

$$0.6875 \times 8 = 5.5000$$

$$0.5000 \times 8 = 4.0000$$

$$\therefore (0.6875)_{10} = (0.54)_8$$

hexadecimal:-

$$0.6875 \times 16 = 11$$

$$\therefore (0.6875)_{10} = (11)_{16}$$

Example (3):- Convert $(0.153)_{10}$ to octal and hexadecimal.

Solution:-

Octal:-

$$0.153 \times 8 = 1.224$$

$$0.224 \times 8 = 1.792$$

$$0.792 \times 8 = 6.336$$

$$0.336 \times 8 = 2.688$$

$$0.688 \times 8 = 5.504$$

Hexa decimal:-

$$\begin{array}{r|l} 0.153 \times 16 = & 8.208 \\ 0.208 \times 16 = & 3.328 \\ 0.328 \times 16 = & 5.248 \end{array}$$

$$\therefore (0.153)_{10} = (0.835)_{16}$$

Base-r to Decimal:-

Example (1):- Convert $(6327.4051)_8$ into its equivalent decimal number.

Solution:- here $n=8$

$$(6327.4051)_8 = 6 \times 8^3 + 3 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 4 \times 8^{-1} + 0 \times 8^{-2} + 5 \times 8^{-3} + 1 \times 8^{-4}$$

$$= 3072 + 192 + 16 + 7 + \frac{4}{8} + 0 + \frac{5}{512} + \frac{1}{4096}$$

$$= (3287.510098)_{10}$$

$$(6327.4051)_{10} = (3287.510098)_{10}$$

Example (2):- Obtain decimal eqt of hexadecimal number $(3A.2F)_{16}$

$$(3A.2F)_{16} = 3 \times 16^1 + A \times 16^0 + 2 \times 16^{-1} + F \times 16^{-2}$$

$$= 3 \times 16^1 + 10 \times 16^0 + 2 \times 16^{-1} + 15 \times 16^{-2}$$

$$= 48 + 10 + \frac{2}{16} + \frac{15}{162}$$

$$= (58.1836)_{10}$$

Binary to Octal and Binary to Hexadecimal Conversion:

Example (1): Convert $(1001110)_2$ to its octal eqt.

Solution: $(1001110)_2 = (\underline{001}, \underline{001}, \underline{110})$
 $= (1\ 1\ 6)_8$

$$\boxed{(1001110)_2 = (116)_8}$$

Example (2): Convert $(0.10100110)_2$ to its eqt octal number.

$$(0.10100110)_2 = (0.\underline{101}, \underline{001}, \underline{100})$$
$$= (0.514)_8$$

$$\boxed{(0.10100110)_2 = (0.514)_8}$$

Example (3): Convert $(11001110001.0001111001)_2$ to octal number.

$$\underline{1100} \underline{1110} \underline{001} . \underline{0001} \underline{0111} \underline{1001}$$

$$(6341.0571)_8$$

Example (4): $(1011011110.1100100011)_2$ to hexadecimal numbers.

$$\underline{0011} \underline{0110} \underline{1111} \underline{1011}$$

$$\underline{0010} \underline{1101} \underline{1110} . \underline{1100} \underline{1000} \underline{1100}$$

Complements:-

The direct method of subtraction requires a complicated logic circuit because to represent negative integer we need a notation for negative sign. but the computer must represent everything with binary digits. To overcome this problem, complements are used in computer for simplifying the subtraction operation and for logical multiplication.

Two Types of Complements:-

- (i) Radix complement or r 's complement
- (ii) Diminished radix complement or $(r-1)$'s complement.

r 's complement or Radix complement:-

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

where, N - number

n - number of digits in integer part

r - radix

$r=10$; 10's complement for decimal number

Examples

10's complement of $2389 = 10^4 - (2389)_{10}$
 $= (7611)_{10}$

10's complement of $(75.25)_{10} = 10^2 - (75.25)_{10}$
 $= (24.75)_{10}$

(r-1)'s complement:-

The number N in base r can be represented in $(r-1)$'s complement as

$$(r-1)^2 N = (r^n - r^{-m} - N)$$

$m \rightarrow$ the number of digits in fractional part

$n \rightarrow$ no. of digits in the int. part

$r \rightarrow$ radix

$N \rightarrow$ Number.

Example) If $r = 10$, then $(r-1) = 9$'s Complement

The 9's complement of $(19.23)_{10}$
 $= (10^2 - 10^{-2} - 19.23)$
 $= (80.76)_{10}$

Example) The 1's complement of

$$(101.001)_2 = 2^3 - 2^{-3} - (101.001)$$

$$= 8 - 0.125 - (101.001)$$

$$= 7.875 - (101.001)$$

$$= 111.111 - 101.001$$

Signed Binary numbers:-

In sign-magnitude representation, an additional bit, known as sign bit is used to indicate the sign of numbers. For

+ve number \rightarrow sign bit 0

-ve number \rightarrow sign bit 1

The remaining bits in the number indicate the magnitude (absolute number)

Example 01000100

MSB

0

Magnitude bits

1000100 \Rightarrow 1000100 = $(68)_{10}$

sign bit

0 \rightarrow +ve number

Binary codes

1. Binary 2. BCD 3. Excess-3, Gray and ASCII codes

Gray code:- It is arranged in such a way that every transition from one value to next value involves only one bit change

Conversion of Binary code to Gray code:-

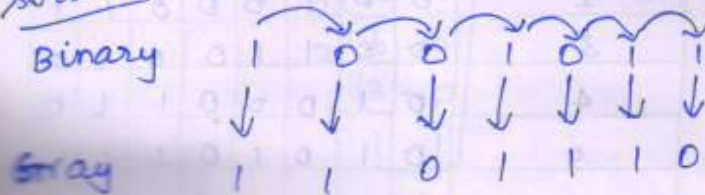
Step (1):- Record the MSB

Step (2):- Add this bit to the next position

Step (3):- Record successive sums until completed.

(Example) Convert the binary 1001011 to Gray code.

Solution:-



Conversion of Gray to Binary code:-

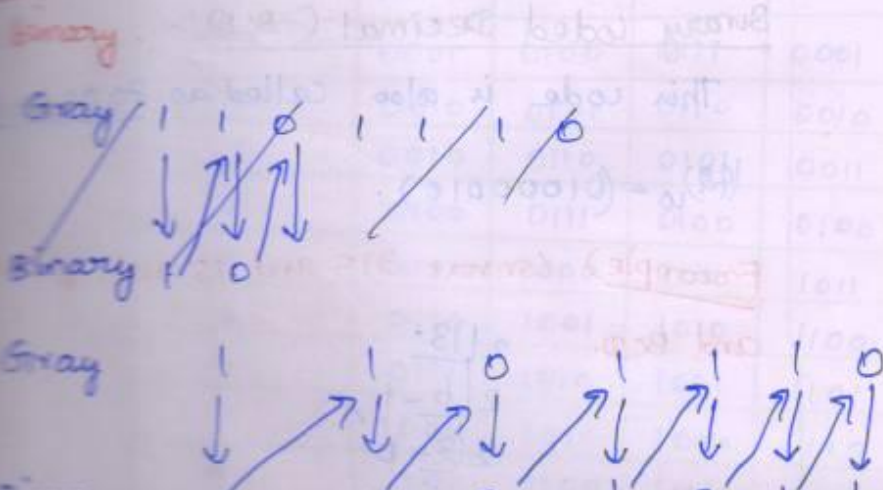
Step (1):- Record the MSB

Step (2):- Add the binary MSB to the next significant bit of the Gray code.

Step (3):- Record the result, ignoring carries

Step (4):- Continue the process until the LSB is reached.

(Example) Convert the Gray code 1101110 to binary



Decimal number	Binary				Gray			
	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
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
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

Binary coded Decimal (BCD):-

This code is also called as 8421 Code

$$(42)_{10} = (01000010)$$

Example) Convert 395 and 13 into binary

and BCD.

$$\begin{array}{r} 2 \overline{) 13} \\ 2 \overline{) 6} - 1 \\ 2 \overline{) 3} - 0 \end{array}$$

$$\begin{array}{r}
 2 \overline{) 395} \\
 2 \overline{) 192} - 1 \uparrow \\
 2 \overline{) 96} - 0 \\
 2 \overline{) 48} - 0 \\
 2 \overline{) 24} - 0 \\
 2 \overline{) 12} - 0 \\
 2 \overline{) 6} - 0 \\
 2 \overline{) 3} - 0 \\
 1 - 1
 \end{array}
 \quad (110000001)_2$$

↳ 9 bits

BCD of $(13) = 1101$

BCD of $395 = 0011\ 1001\ 0101 \Rightarrow 12\text{ bits}$

Excess-3 code: This is another form of BCD code but unweighted code. For example, decimal 5 coded as $0101 + 0011 = 1000$ in excess-3 code.

Example

Decimal Digits	BCD	Excess-3	8-4-2-1	2-4-2-1
0	0000	0011	0000	0000
1	0001	0100	0111	0001
2	0010	0101	0110	0010
3	0011	0110	0101	0011
4	0100	0111	0100	0100
5	0101	1000	1011	1011
6	0110	1001	1010	1100
7	0111	1010	1001	1101
8	1000	1011	1000	1110
9	1001	1100	1111	1111

Hamming code:- One of the most common error correcting code is the hamming code. The hamming code is constructed by adding a number of parity bits to each groups of n -bit information in such a way that able to locate the bit position in which error occurs.

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

k parity bits are added to an n-bit data word, forming a new word of n+k bits. k must satisfy the equality

$$2^{k-1} \geq n+k.$$

Alpha numeric codes:- consists of letters & numbers. ASCII (American standard code for Information Interchange) code is a standard 7-bit, most widely used binary code for alphanumeric characters. It consists of total 128 characters, 10 numbers (0 to 9), 26 uppercase letters (A to Z), 26 lowercase letters (a to z), 32 special characters (., /, & etc) and 34 non printing control function characters.

Binary Arithmetic

Binary Addition:-

Rules of Binary Addition

Augend	Addend	sum	Carry	Result
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	0	1	10

Example 1) 1011 add with 1100

$$\begin{array}{r}
 1011 \\
 + 1100 \\
 \hline
 11011
 \end{array}$$

Carry

$$1011 + 1100 = (0111)_2$$

2. 01010111 and 00110101

$$\begin{array}{r}
 01010111 \\
 + 00110101 \\
 \hline
 10001100
 \end{array}$$

Binary Subtraction:-

Rules for subtraction

Minuend	Subtrahend	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Example: 100 - 011

$$\begin{array}{r}
 0100 \\
 - 0011 \\
 \hline
 0111
 \end{array}$$

Binary multiplication:-

Example:- multiply 1011 by 101.

$$\begin{array}{r} 1011 \leftarrow \text{Multiplicand} \\ 101 \leftarrow \text{Multiplier} \\ \hline 0000 \\ 1011 \\ \hline 11011 \end{array} \quad \left. \begin{array}{l} 0000 \\ 1011 \end{array} \right\} \text{partial product}$$

$$\boxed{1011 \times 101 = 11011}$$

Binary Division:-

Example) Divide 110110 by 101.

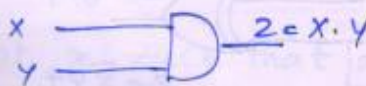
$$\begin{array}{r} 10 \leftarrow \text{Quotient} \\ 101 \overline{) 110110} \leftarrow \text{Dividend} \\ \underline{101} \\ 00111 \\ \underline{101} \\ 100 \rightarrow \text{Remainder} \end{array}$$

Boolean Algebra

Basic Digital circuits:-

AND operation:-

x	y	z
0	0	0

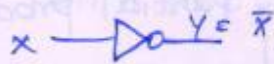


OR operation:-



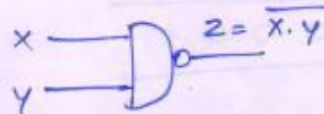
x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

NOT operation:-



x	y
0	1
1	0

NAND operation:-



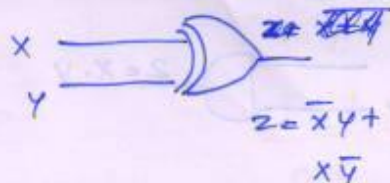
x	y	z
0	0	1
0	1	1
1	0	1
1	1	0

NOR operation:-

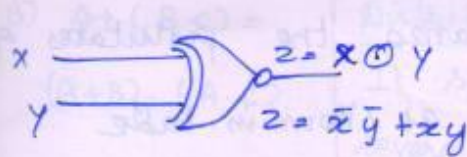


x	y	z
0	0	1
0	1	0
1	0	0
1	1	0

EX-OR & EX-NOR:-



x	y	z
0	0	0
0	1	1
1	0	1
1	1	0



x	y	z
0	0	1
0	1	0
1	0	0
1	1	1

Basic Definitions:-

Postulate 1:- closure with respect to the operator $(+)$ and (\cdot) , $(+)$ \rightarrow addition (\cdot) \rightarrow multiplication

Postulate 2:- The additive identity is zero which means $x + 0 = x$ or $0 + x = x$

The multiplicative identity is 1 which means $x \cdot 1 = x$ or $1 \cdot x = x$

Postulate 3:- The multiplication (\cdot) is distributive over addition $(+)$ and addition is distributive over multiplication (\cdot)

Ex):- $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$ and $x + (y \cdot z) = (x + y) \cdot (x + z)$

Postulate 4:- Commutative with respect to both $(+)$ and (\cdot) . For example

$x + y = y + x$ and $x \cdot y = y \cdot x$

Postulate 5:- For every element x there exists an element x' such that $x + x' = 1$ and $x \cdot x' = 0$

we can summarize the postulates of Boolean algebra as shown in table

S.no	Postulate	comment
1.	Result of each operation is either 0 or 1.	$1, 0 \in B$
2.	a) $\begin{cases} 0+0=0 \\ 0+1=1 \\ 1+0=1 \end{cases}$ b) $\begin{cases} 1 \cdot 1 = 1 \\ 0 \cdot 1 = 1 \cdot 0 = 0 \end{cases}$	Identify elements
3.	a) $(A+B) = (B+A)$ b) $(A \cdot B) = (B \cdot A)$	<u>commutative law I:-</u> The order in which the variables are ORed makes no difference in the output. <u>commutative law II:-</u> The order in which the variables are ANDed makes no difference in the output
4.	a) $A \cdot (B+C) = (A \cdot B) + (A \cdot C)$	<u>Distributive law I:-</u> It states that ORing several variables and ANDing results with a single variable

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

Distributive law-II:-

It states that ANDing several variables and ORing the result with a single variable is eqt ORing the result with a single variable with each of the several variables the ANDing the sums.

5- a) $A + \bar{A} = 1$ since
 If $A = 0$, then $\bar{A} = 1$
 $0 + 1 = 1$, vice versa
 and

Complement

b) $A \cdot \bar{A} = 0$ since
 If $A = 1$ then $\bar{A} = 0$
 $1 \cdot 0 = 0$

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

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

a) $A + (B + C) = (A + B) + C$

Associative law-I:-

It states that in the ORing of several variables the result is the same regardless of the grouping of the variables

b) $(A \cdot B) \cdot C = A \cdot (B \cdot C)$

Associative law-II:-

It states that in the ANDing of several variables the result is the same regardless of the grouping

* De Morgan's theorem:-

De Morgan suggested two theorems that form an important part of Boolean Algebra. They are,

$$1. \overline{AB} = \bar{A} + \bar{B}$$

$$2. \overline{A+B} = \bar{A} \cdot \bar{B}$$

De Morgan's first theorem:-

$\overline{AB} = \bar{A} + \bar{B} \rightarrow$ The complement of a product is equal to the sum of the complements.

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

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

De Morgan's second theorem:-

$\overline{A+B} = \bar{A} \cdot \bar{B} \rightarrow$ The complement of a sum is equal to the product of the complements.

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

Boolean Functions

Principle of duality:-

The principle of duality theorem says that, starting with a Boolean relation, you can derive another boolean relation by:

1. changing OR sign to an AND sign
2. changing AND sign to an OR sign
3. complementing any 0 or 1 appearing in the expression.

(Example 1):- If A and B are Boolean variables and if $A=1$ and $\overline{A+B}=0$ find B.

sol Given $\overline{A+B}=0$

$\therefore A+B=1$, Now if $A=1$, we have

$$1+B=1$$

Therefore B can be either 0 or 1

$$B = 0 \text{ or } 1$$

Example 5 :- Prove the following using De Morgan's theorem:

$$AB + CD = \overline{\overline{AB} \cdot \overline{CD}} \text{ and } (A+B) \cdot (C+D) = \overline{\overline{A+B} + \overline{C+D}}$$

sol

$$(i) AB + CD = \overline{\overline{AB + CD}}$$

$$AB + CD = \overline{\overline{AB} \cdot \overline{CD}}$$

$$\therefore \overline{\overline{y}} = y$$

$$\therefore \overline{A+B} = \overline{A} \cdot \overline{B}$$

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

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

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

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

Tutorial samples on Boolean Function

simplify $x + x'y$

sol

$$\text{Let } Z = x + x'y = x + xy + x'y$$

$$= x + y(x + \overline{x})$$

$$\boxed{x + x'y = x + y}$$

Apply De Morgan's theorem to simplify

$$\overline{A + B\overline{C}}$$

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

3) Prove that $(x_1 + x_2)(\bar{x}_1 \bar{x}_3 + x_3)(\bar{x}_2 + x_1 x_3) = \bar{x}_1 x_2$

so)

Apply De Morgan's Law

$$(x_1 + x_2)(\bar{x}_1 \bar{x}_3 + x_3)(\bar{x}_2 \cdot \overline{x_1 x_3})$$

$$(x_1 + x_2)(\bar{x}_1 \bar{x}_3 + x_3)(x_2 \cdot (\bar{x}_1 + \bar{x}_3))$$

$$(x_1 + x_2)(\bar{x}_1 \bar{x}_3 + x_3)(\bar{x}_1 x_2 + x_2 \bar{x}_3)$$

$$x_1 \bar{x}_1 (x_1 + x_2)(\bar{x}_1 \bar{x}_3 + x_3)(\bar{x}_1 x_2 + x_2 \bar{x}_3)$$

$$(x_1 \bar{x}_1 + x_1 x_3 + \bar{x}_1 x_2)(\bar{x}_1 x_2 + x_2 \bar{x}_3)$$

$$(0 + x_1 x_3 + \bar{x}_1 x_2 + x_2 x_3)(\bar{x}_1 x_2 + x_2 \bar{x}_3)$$

$$0 + 0 + \bar{x}_1 x_2 + \bar{x}_1 x_2 \bar{x}_3 + x_2 x_3 + 0$$

$$\bar{x}_1 x_2 + \bar{x}_1 x_2 \bar{x}_3 + \bar{x}_1 x_2 x_3$$

$$\bar{x}_1 x_2 (1 + \bar{x}_3 + x_3)$$

$$\bar{x}_1 x_2, \text{ hence proved.}$$

4) $Y = ABC\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}BC\bar{D} + ABC\bar{D}$

so)

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

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

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

$$\boxed{Y = B\bar{D}}$$

$$5) Y = AB + A(B+C) + B(B+C)$$

10)

$$Y = AB + AB + AC + BB + BC$$

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

$$= AB + AC + B(1+C)$$

$$= AB + AC + B$$

$$= B(1+A) + AC$$

$$\boxed{Y = B + AC}$$

$$6) Y = A\bar{B}D + A\bar{B}\bar{D}$$

10)

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

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

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

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

10)

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

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

$$= 0 + \bar{A}B + AB + B$$

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

$$= B + B$$

$$\boxed{Y = B}$$

$$8) Y = (A+B+C) \cdot (A+B)$$

sol)

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

$$= A \cdot A + A \cdot B + B \cdot A + B \cdot B + C \cdot A + C \cdot B$$

$$= A + AB + AB + B + AC + BC$$

$$= A + AB + B + AC + BC$$

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

$$= A + B + AC$$

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

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

$$9) (A+B)(\bar{A}+C)(B+C) = (A+B) \cdot (\bar{A}+C)$$

LHS:-

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

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

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

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

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

30) Simply Boolean exp $Y = (A+B)(\bar{A}\bar{C}+C)(\bar{B}+AC)$

sol)

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

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

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

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

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

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

$$= AC + B\bar{A} + ACB\bar{C} + \bar{A}\bar{C}B \cdot B \cdot \bar{A} + \bar{A}\bar{C}B \cdot B \cdot C + BC B\bar{A} + BC B\bar{C}$$

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

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

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

NAND and NOR Implementation

* Digital circuits are constructed using NAND or NOR gates only rather than AND, OR and NOT gates

* NAND and NOR gates are easier to fabricate

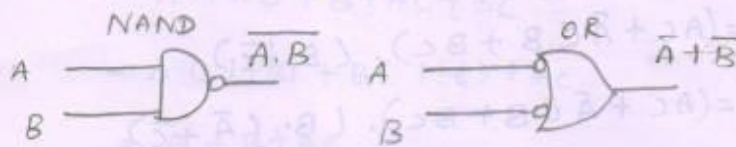
* Hence NAND and NOR gates are

Two level NAND-NAND Implementation:

* To facilitate the conversion to NAND logic, it is convenient to define an alternative graphic symbol for the gate.

According to De Morgan's Theorem

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



steps to be followed

1. Simplify the given logic expression and convert it in the SOP form
2. Draw logic circuit using AND, OR and NOT gate
3. Replace every AND gate by a NAND gate, every OR gate by a bubbled OR gate and NOT gate by a NAND inverter
4. Replace bubbled-OR gate by NAND gate.

Example Implement the foll. Boolean eqn using only NAND gates.

$$Y = AB + CDE + F$$

Multi level NAND circuits:-

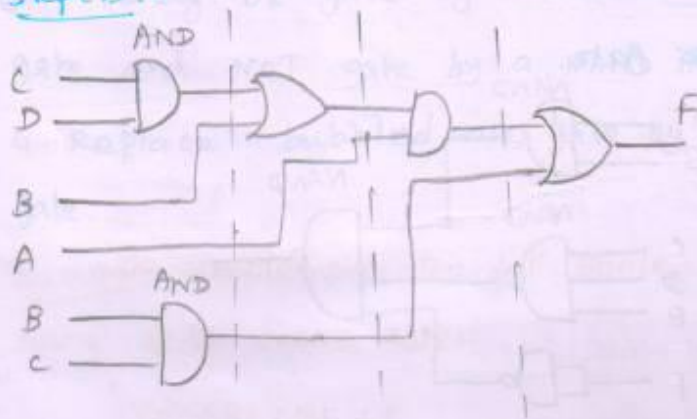
Procedure:-

1. Convert all AND gates to NAND gates with AND-invert graphic symbols
2. Convert all OR gates to NAND with invert-OR graphic symbol
3. Check all the bubbles in the diagram. For every bubble that is not compensated by the other small circle along the same line, insert an inverter or complement the input literal.

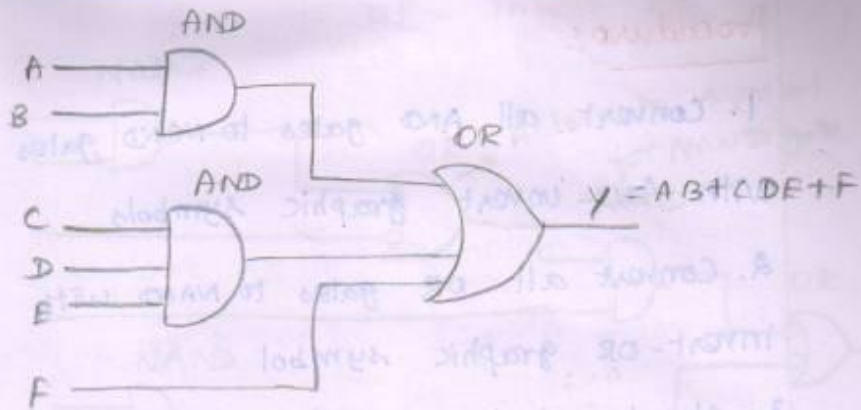
(Example) Implement the foll. Boolean exp using NAND gates only,

$$F = A(CD + B) + Bc$$

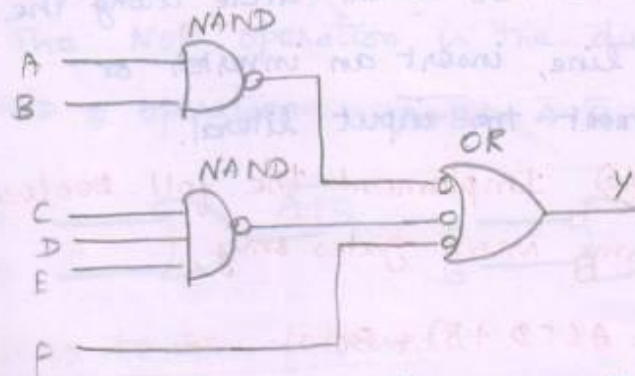
Step 1:-



Step (1):- Realisation using basic gates

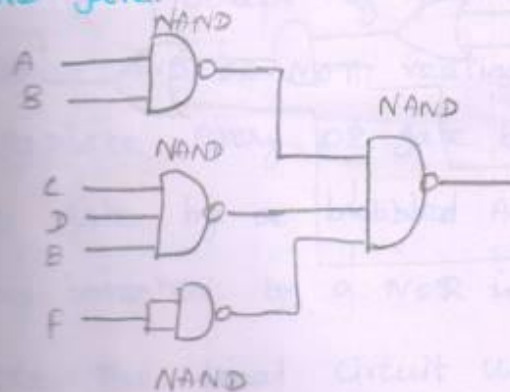


Step (2):- Replace AND by NAND and OR by bubbled OR, NOT \Rightarrow Inverter.

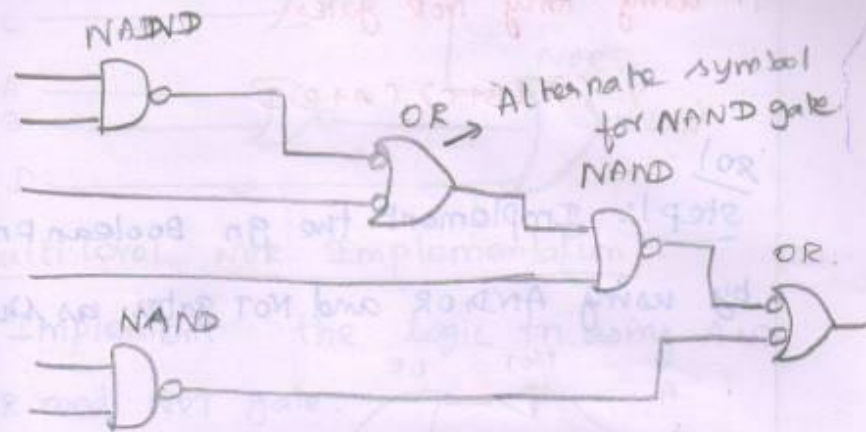


Step (3):- Draw the logic circuit using only

NAND gate.

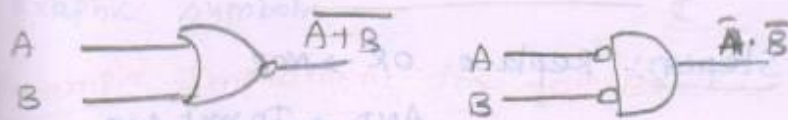


Step (8):- Replace AND - AND invert (NAND)
OR - invert OR.



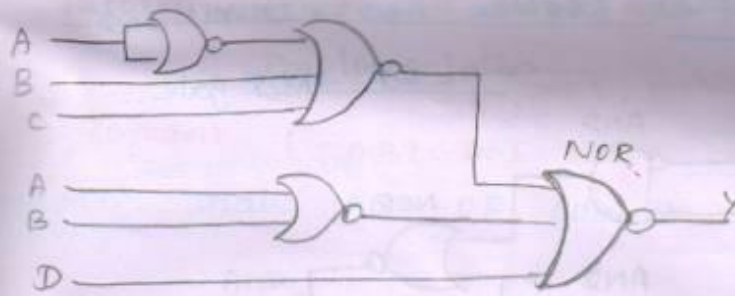
NOR Implementation:-

The NOR operation is the dual of the AND operation. $\overline{A+B} = \bar{A} \cdot \bar{B}$



Steps to be followed:-

1. Simplify the given logic exp and convert it into product of sum (POS) form
2. Draw AND-OR-NOT realisation
3. Replace every OR gate by NOR, every AND gate by a bubbled AND gate and every inverter by a NOR inverter
4. Draw the final circuit using only NOR gate.

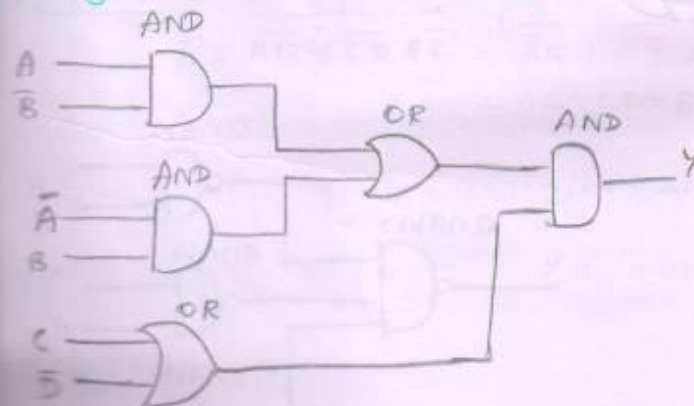


Multi level NOR Implementation :-

- 1) Implement the logic fn using AND, OR and NOT gate.
- 2) Convert all AND gates to NOR gates with invert-AND graphic symbol
- 3) Convert all OR gates with OR invert graphic symbols

Example Implement the foll. Boolean fn using NOR gates. $Y = (A\bar{B} + \bar{A}B)(C+D)$

Step 1:- Implement the Boolean fn using AND, OR and NOT gates.

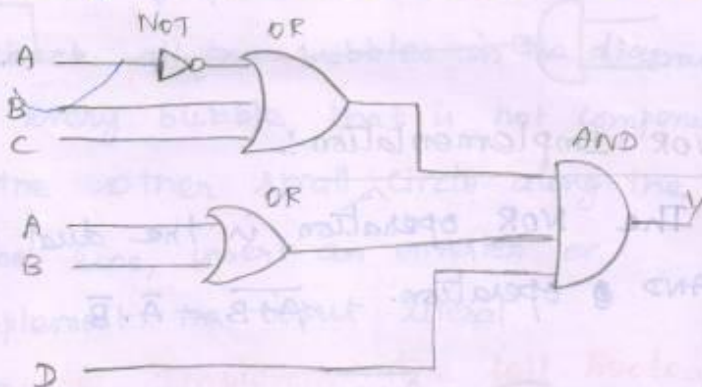


Example Implement the foll. Boolean fn using only NOR gates.

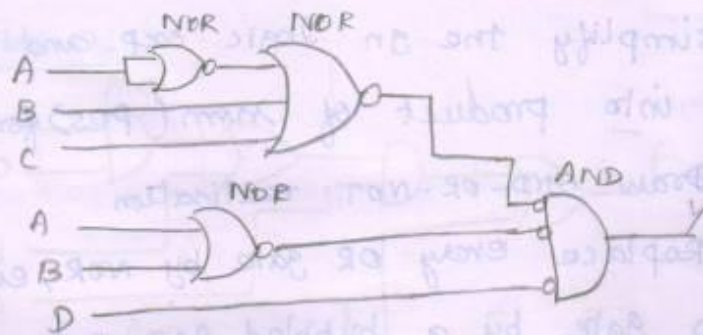
$$Y = (\bar{A} + B + C)(A + B)D$$

sol

Step 1:- Implement the gn Boolean fn by using AND, OR and NOT gates as shown

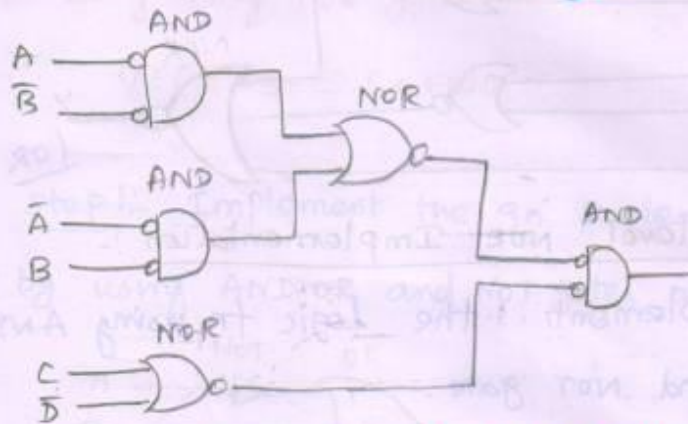


Step 2:- Replace OR \rightarrow NOR
AND \rightarrow Invert AND
~~NOT~~ NOT \rightarrow NOR invert

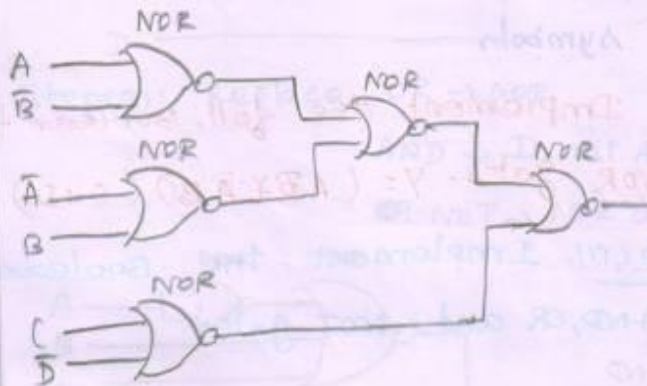


Step 3:- Replace invert AND by NOR gate

Step(2):- Replace AND \rightarrow Invert AND
OR \rightarrow NOR gate.



Step(3):- Check each line has even no. of bubbles. If any line does not have even no. of bubbles then insert bubble.



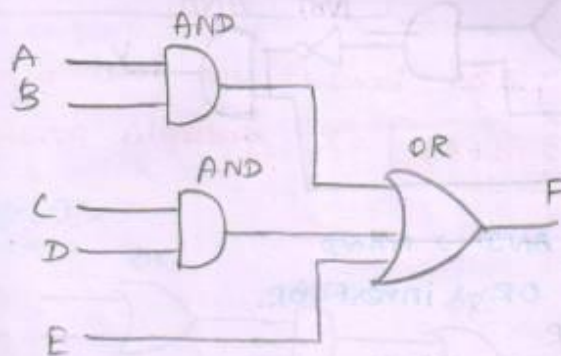
Tutorial samples in NAND, NOR

39

Implementation

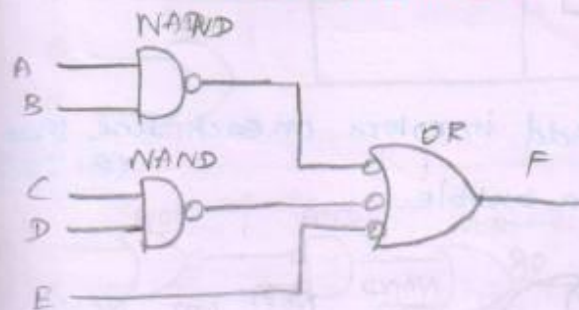
Q Implement $F = AB + CD + E$ using only NAND.

Step 1:- Draw AND-OR circuit



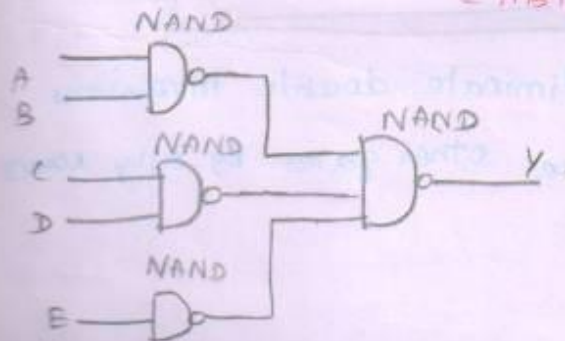
Step 2:- AND \rightarrow NAND

OR \rightarrow invert OR.



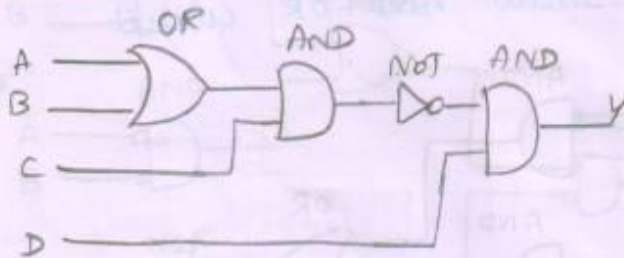
Step 3:- replace other gates by NAND gates.

$$F = \overline{AB} + \overline{CD} + \overline{E} = \overline{\overline{AB}} + \overline{\overline{CD}} + \overline{\overline{E}} \\ = AB + CD + E$$



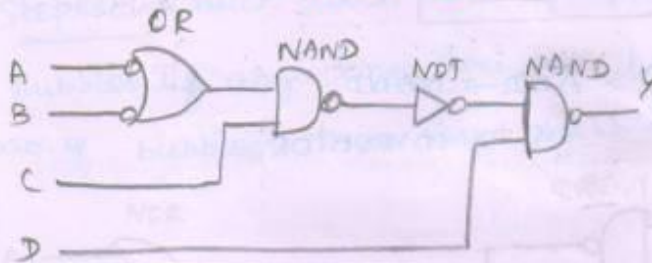
$$Y = (A + B)C'D$$

Step (1):- Draw the logic diagram

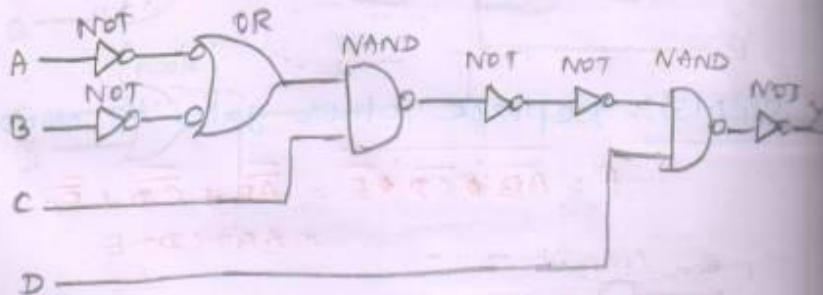


Step (2):- AND \rightarrow NAND

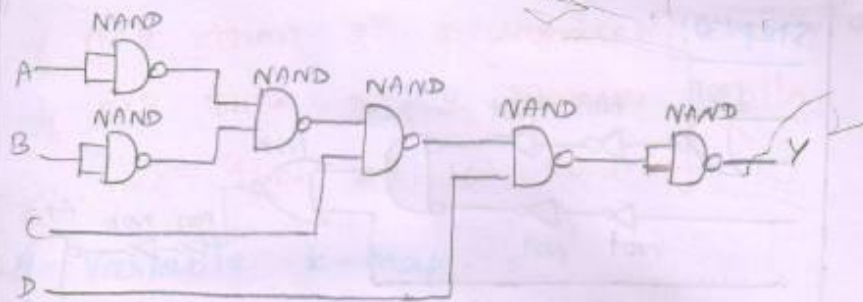
OR \rightarrow invert-OR.



Step (3):- Add inverters on each line that received a bubble.

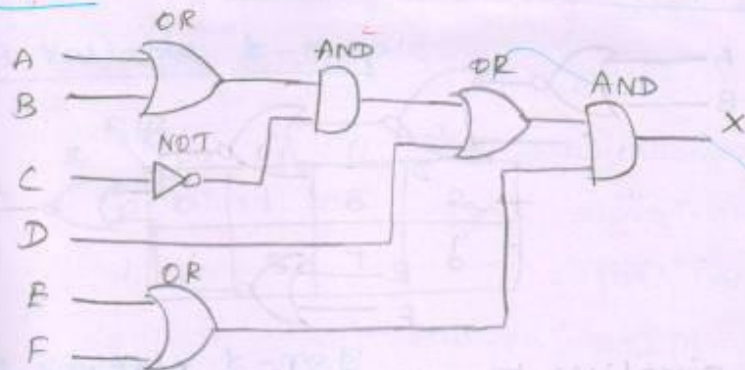


Step (4):- Eliminate double inversions and replace other gates by only NAND gates.

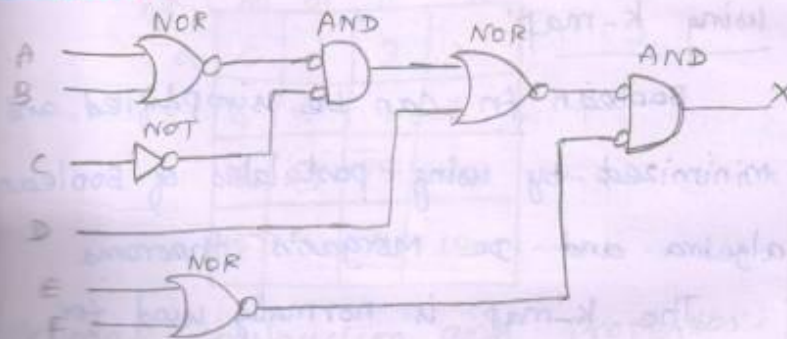


③ Draw the multilevel NOR circuit for Boolean Algebra $X = [(A+B)\bar{C}+D](E+F)$

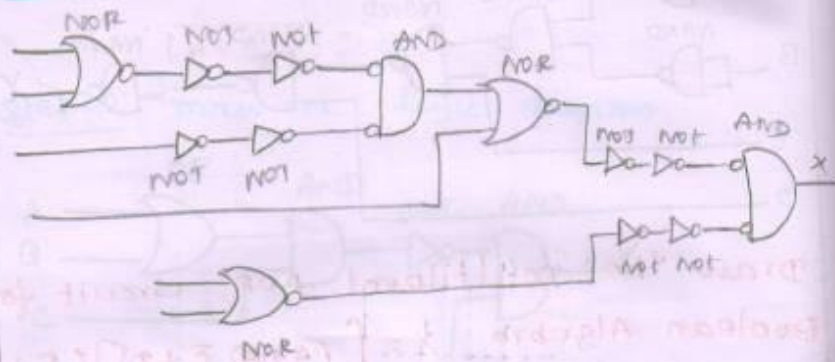
Step (1):-



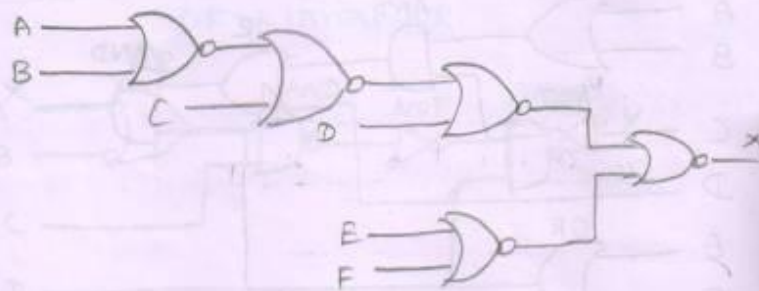
Step (2):-



Step (3):-



Step (4):-



Simplification of Boolean Functions

using k-map:-

Boolean fn can be simplified and minimized by using postulates of Boolean algebra and De Morgan's theorems.

The k-map is normally used for representing the minimizing three-variable or four-variable fn.

The k-map is made up of squares. Each square represents one term.

if $n=2$ then $2^2 = 4$ squares
 if $n=3$ then $2^3 = 8$ squares
 if $n=4$ then $2^4 = 16$ squares

43

cells.

2 variable k-map:-

x \ y	0	1
0	0	1
1	2	3

3 variable k-map:-

x \ yz	00	01	11	10
0	0	1	3	2
1	4	5	7	6

4 variable k-map:-

wx \ yz	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

k-map construction and properties:-

Grouping of 2 adjacent Ones:-

\Rightarrow there are 2 adjacent ones on the map, they can be combined together to form a pair. A pair results in a term of 2

(Example) simplify the Boolean fn

$$f(a, b, c) = \sum (0, 2, 5, 6, 7)$$

Sol

1. Mark 1 in each which is gn in fn.

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

2. Identify the possible adjacent cells

(i) There are 2 pairs of adjacent squares indicated in the map-one in rectangle shape and another is in oval shape.

(ii) One square (cell 0) is single

3. Observe the value of variables associated with these squares and determine the terms which appear in ANDed form.

(i) The lower rectangle represents 'ac'.

One variable 'b' is different, hence gets eliminated.

(ii) Right most oval represents 'bc'.

(iii) single cell represents a term.

4. Finally write the simplified em

$$f = ac + b\bar{c} + \bar{a}\bar{b}\bar{c}$$

45

* Single cell represents one minterm and gives a term of three literals

* Grouping of two adjacent squares, named as 'pair', eliminate one variable and represents a term with 2 literals

* There are some cases where 2 cells are not touching each other but considered as adjacent cells.

Example) simplify the fn $f(a,b,c) = \sum (0,1,2,9)$

		bc			
a	0	00	01	11	10
	1	01	11	10	00
		0	1	2	3
		4	5	6	7

* Two adjacent cells are grouped together to give the literal term ' $\bar{b}c$ '

* cells 0 and 2 are also adjacent combined to give a term $\bar{a}\bar{c}$.

* The simplified function $f = \bar{b}c + \bar{a}\bar{c}$.

Grouping of 4 and 8 adjacent ones:-

Example) $f = \bar{a}c + ac + a\bar{b} + a\bar{b}\bar{c}$

Sol

1. Expand each 2 literals product term

$$\bar{a}c = \bar{a}c(b + \bar{b}) = \bar{a}bc + \bar{a}\bar{b}c = 001 + 011 =$$

minterms

2) The 3 literals term $a\bar{b}\bar{c}$, total 5 minterms, m_1, m_3, m_4, m_5 and m_7 .

Thus $f(a,b,c) = \Sigma(1,3,4,5,7)$

3) Draw three variables map

a \ bc				
	00	01	11	10
0	0	1	3	2
1	4	5	7	6

4) Group cell 1, 3, 5, 7 to give only one literal ' c '.

5) Group the 2 adjacent squares, cell 4 and cell 5. literal = $a\bar{b}$.

$$f = a\bar{b} + c$$

Example:- $f(a,b,c) = \Sigma(0,1,2,3,4,5,6,7)$

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

The simplified function $f = 1$

Grouping of 8 cells - Octet: An octet

removes 3 variables and gives a product term with single literal.

ab \ cd	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$$f = d$$

ab \ cd	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$$f = \bar{d}$$

(Example) Simplify the Boolean fn

$$f(a, b, c, d) = \sum(0, 1, 5, 7, 8, 9, 11, 14, 15)$$

sol

ab \ cd	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$$f = \bar{b}\bar{c} + abc + \bar{a}bd + a\bar{b}$$

$$f = \bar{b}\bar{c} + abc + \bar{a}bd + cd$$

Tabulation method (or) Quine-McCluskey ⁴⁹ Method

(Example) simplify the foll. Boolean fn by using the tabulation method

$$f = \sum (0, 1, 2, 8, 10, 11, 14, 15)$$

Solution:- solution of Boolean fn using Quine-McCluskey approach.

* Construct a prime implicant table and determine prime implicants by searching and combining minterms

* List all the Prime Implicants

* Evaluate prime Implicants and essential prime implicants, if any, by forming PI table.

* Get the simplified soln by ORing all the prime implicants in SOP form.

Step 1:- Arrange all the minterms of the fn in binary form according to the number of 1's contained and group them into sections separated by horizontal lines, as shown below.

a		Grouping into sections according to no. of 1's
m	wxyz	
0	0000	Containing no 1's
1	0001	
2	0010	Containing single 1's
8	1000	
10	1010	Containing two 1's
11	1011	
14	1110	Containing three 1's
15	1111	Containing four 1's

Step 1:- Any 2 minterms that differ from each other by only one variable can be combined, and the unmatched variable is removed. For this search compare each binary no in one section with those of next section down only, and if they differ only by one position, put a check mark and copy the new term in the next column dash (-) in the position that they differ. For example, the $m_0(0000)$ combines with $m_1(0001)$ to form $(000-)$ which is copied in column (b) as shown below,

a		b
wxyz		wxyz
0	0000✓	000-
1	0001✓	

This combination is eqt to the algebraic operation

$$m_0 + m_1 = W'x'y'z' + W'x'y'z$$

$$= W'x'y'(z' + z)$$

$$m_0 + m_1 = W'x'y'$$

Similarly m_0 also combines with m_2 and m_8 to form $(00-0)$ and (-000) respectively. All these new terms are entered into first section of column 'b' is illustrated below,

a		b
wxyz		wxyz
0	0000✓	000-
1	0001✓	00-0
2	0010✓	-000
8	1000✓	

Step 13:- Apply the same process as discussed in step 8. The terms of column (b) have only 3 variables. The resultant terms of column (b) are entered in column (c)

This searching and comparing cycle is continued until a single pass through cycle yields no further elimination of literals. In this example, the operation stops at the column (c).

a		b		c	
Wxyz		Wxyz		Wxyz	
0	0000✓	0,1	000-	0,2,8,10	-0-0
1	0001✓	0,2	00-0✓	0,8,2,10	-0-0
2	0000✓	0,8	-000✓	10,11,14,15	1-1-
8	1000✓	2,10	-010✓	10,14,11,15	1-1-
10	1010✓	8,10	10-0✓		
11	1011✓	10,11	101-✓		
14	1110✓	10,14	1-10✓		
15	1111✓	11,15	1-11✓		
		14,15	111-✓		

Step (4): List all the unchecked terms which constitute prime implicants

PI	Binary value
0,1	000-
0,2,8,10	-0-0
0,8,2,10	-0-0
10,11,14,15	1-1-
10,14,11,15	1-1-

Step (5): select The minimum no. of prime implicants which must cover all the minterms. In the present example step 5 is not needed. The above example is purposely chosen for simplicity. Finally the sum of prime implicants gives the minimized fn in sum of products form.

$$f = W'x'y' + x'z' + wy$$

(Example) simplify the boolean fn by using a Quine-McCluskey method.

$$f(W, x, y, z) = \sum (1, 4, 6, 7, 8, 9, 10, 11, 15)$$

sol:-

a	b	c
0001 1✓	1, 9 (8)	8, 9, 10, 11 (1, 2)
0100 4✓	4, 6 (2)	8, 10, 9, 11 (1, 2)
1000 8✓	8, 9 (1)	
	8, 10 (2)	
0110 6✓	6, 7 (1)	
1001 9✓	9, 11 (2)	
1010 10✓	10, 11 (1)	
0111 7✓	7, 15 (8)	
1011 11✓	11, 15 (4)	
1111 15✓		

Step(1):- All the minterms are listed in the table. column (a) along with their binary representation for the purpose of counting the number of 1's.

Step(2):- The minterms are grouped in sections according to their number of 1's.

Step(3):- The minterms are compared by the decimal method and a match is found if the no. in the section below.

Step(4):- There are only 2 matches of terms in column (b).

Step(5):- Prime Implicants are listed

PI		wxyz	Term
1, 9	(8)	-001	$x'y'z$
4, 6	(2)	01-0	$w'xz'$
6, 7	(1)	011-	$w'xy$
7, 15	(8)	-111	xyz
11, 15	(4)	1-11	wyz
8, 9, 10, 11	(1, 2)	10--	wx'

Step(6):- selection of prime implicants for final solution,

PI	Decimal	Minterms									
		1	4	6	7	8	9	10	11	15	
$x'y'z$	1, 9*	x					x				
$w'xz'$	4, 6*		x	x							
$w'xy$	6, 7			x	x						
xyz	7, 15				x					x	
wyz	11, 15								x	x	
wx'	8, 9, 10, 11*	✓	✓	✓		✓	✓	✓	✓		

Therefore, the minimum set of prime implicants was found and their sum gives the required minimized fn.

$$f = x'y'z + w'xz' + wx' + xyz$$

Simplification using 4-variable k-map:-

Isolated cell and pair of cells:-

A cell which is not adjacent to any other cells can be referred to as 'isolated cell'. It consists of 4 literals. whereas pair removes one variable and gives a product.

cd \ ab	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$$f = \bar{b}c\bar{d} + \bar{a}bd + ab\bar{a} + a\bar{b}c\bar{d}$$

Grouping of 4 cells - Quad:- A quad removes 2 variables and gives a product term with 2 literals.

cd \ ab	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$$f = bd$$

cd \ ab	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$$f = \bar{b}d$$

Unit - II - Combinational Logic

57

Combinational circuits - Analysis and Design procedures - Circuits for Arithmetic operations, Code conversions - Decoders and Encoders - Multiplexers and Demultiplexers - Introduction to HDL - HDL models of combinational Circuits.

Combinational Circuits

Digital circuits are classified into 2 categories:-

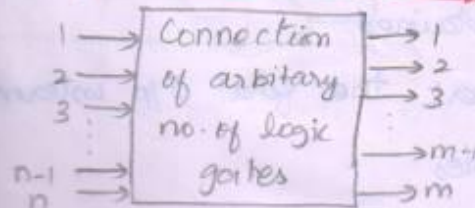
* Combinational Circuits

* Sequential Circuits

Combinational Circuits :- A combinational circuit consists of an arbitrary number of logic gates which are connected together to form an output for a certain combination of input variables without any feedback.

The o/p at any instant of time depends on the i/p present at that instant of time only.

Block diagram of combinational ckt



Analysis and Design Procedure

The purpose of analysis of any device is to know the function of that device. Basically analysis of a combinational circuit is to obtain the op of that circuit whereas circuit design involves obtaining a digital logic circuit from the specified design objective (o/p).

Analysis procedure:- Analysis of a digital ckt results a Boolean fn, a truth table and sometimes an explanation of ckt also.

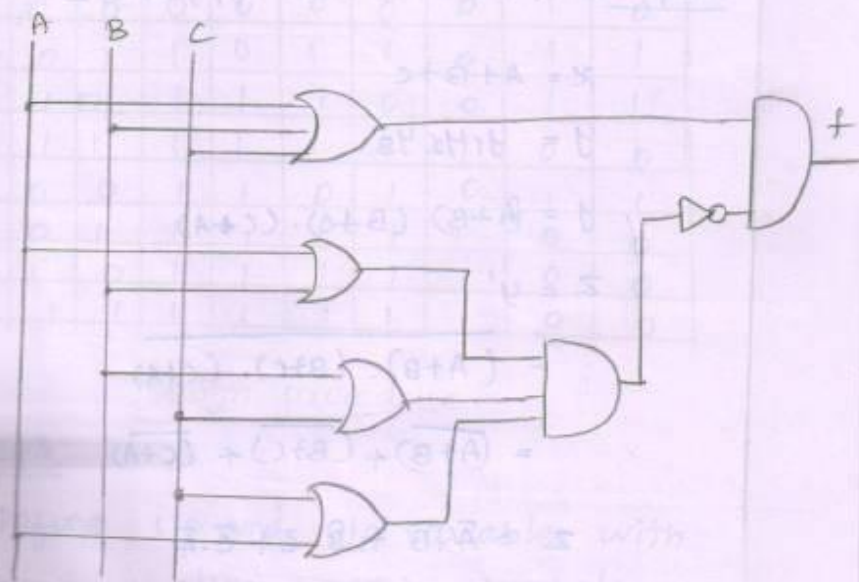
To obtain the Boolean fn of the gn ckt:-

- * Designate the o/p of selected gates with suitable alphanumeric symbols.
- * Find the o/p expression for each selected gate which is connected to i/p.
- * Find the o/p expression for the gates which is connected to previous gates.
- * Repeat this process until the final o/p is obtained.
- * Convert the final o/p in terms of i/p variables.

To obtain the truth table directly from the digital circuit :-

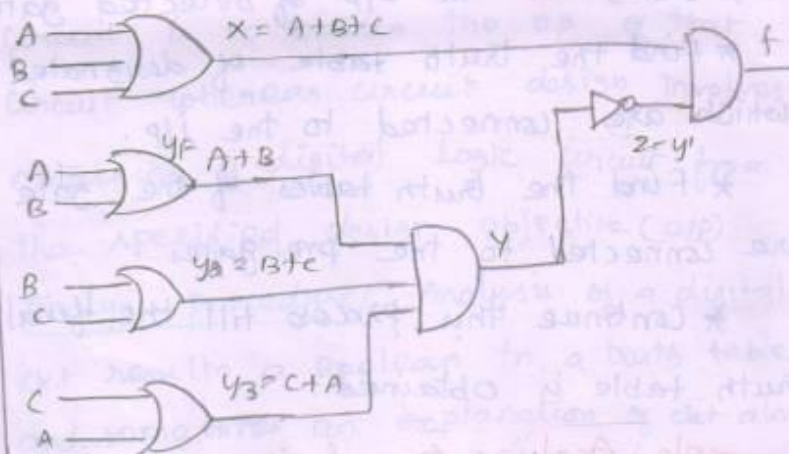
- * List all the binary combinations of i/p variables
- * Designate the o/p of selected gates
- * Find the truth table of designated gates which are connected to the i/p.
- * Find the truth tables of the gate which are connected to the pre. gates
- * Continue this process till the final truth table is obtained.

(Example) Analyze the logic circuit.



Solution:-

Step(1):- Determine the Boolean fn of the
gn circuit. Designate all intermediate
gates as 'x', 'y' and 'z'.



step(2):- Find the o/p x, y, and z.

$$x = A + B + C$$

$$y = y_1 y_2 y_3$$

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

$$z = y'$$

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

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

$$z = \bar{A} \cdot \bar{B} + \bar{B} \cdot \bar{C} + \bar{C} \cdot \bar{A}$$

Step(3):- Find the final o/p f

$$f = xz$$

Step(4):- x and z are replaced with

$$f = xz$$

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

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

Obtain the truth table of the logic:-

Step (1):- List all 8 combinations of 3 i/p

Step (2):- Individual truth tables for all designated gates.

Step (3):- Truth table for z is determined

Step (4):- Finally truth table for f is determined as shown

A	B	C	x	y ₁	y ₂	y ₃	y	z	f
0	0	0	0	0	0	0	0	1	0
0	0	1	1	0	1	1	0	1	1
0	1	0	1	1	1	0	0	1	1
0	1	1	1	1	1	1	1	0	0
1	0	0	1	1	0	1	0	1	1
1	0	1	1	1	1	1	1	0	0
1	1	0	1	1	1	1	1	0	0
1	1	1	1	1	1	1	1	0	0

Design procedure

Steps:-

* Define i/p and o/p variables with meaningful alpha-numeric symbols

* Specify required circuit's fundamental behaviour by means of truth table and

Boolean expression

* simplify ~~required~~ Boolean exp or truth table with the help of any standard technique such as k-map, tabulation or Boolean theorem.

* K-map, tabulation realize the required logic diagram according to the simplified exp.

Example) Design a combinational circuit with 3 i/p's A, B, C and one o/p Y. The o/p should be logic 1 when atleast a i/p is logic 1.

Step (1): No of i/p - 3 (A, B, C)

No. of o/p - 1 (Y)

Step (2): No. of possible combination - 2^n

$$2^3 = 8$$

Inputs			output
A	B	C	Y
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

From the truth table,

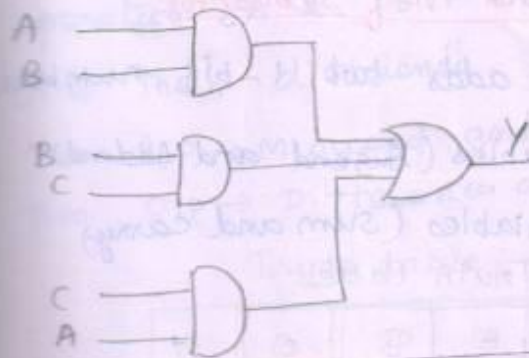
$$Y = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

Step (3): Derived Boolean fn is simplified using k-map.

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

$$Y = AC + AB + BC$$

Step (4): Logic diagram is realized as shown,



Circuit for Arithmetic operations

Adders/Subtractors.-

Adders/subtractors perform the basic arithmetic operation of addition and subtraction of 2 binary digits respectively.

Addition	Subtraction
$0+0 = 0$ } don't	$1-0 = 0$
$0+1 = 1$ } produce	$0-1 = 1$ ← with a borrow of 1
$1+0 = 1$ } any	$1-0 = 1$
$1+1 = 10$ } carry	$1-1 = 0$
<p>* If minuend bit < Subtrahend bit then 1 is borrowed from next higher significant bit</p>	

Half adder and Half Subtractor:

Half adder: It adds two 1-bit numbers.

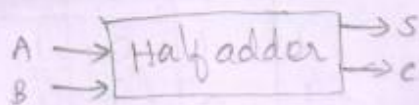
Two I/P Variables (Augend and Addend),

two O/P Variables (Sum and carry)

Truth table

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

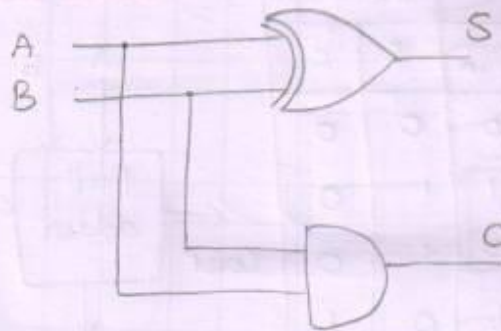
Block Diagram



$$S = \bar{A}B + A\bar{B} = A \oplus B$$

$$C = AB$$

Half adder circuit



Half subtractor:- It performs subtraction operation on 2-bit numbers and gives their difference.

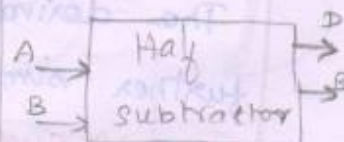
Two i/p → Minuend and subtrahend

Two o/p → Difference and borrow.

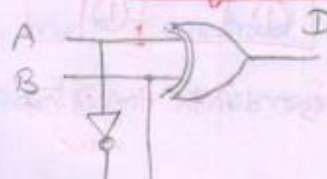
Truth table

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

Block diagram



Half adder circuit

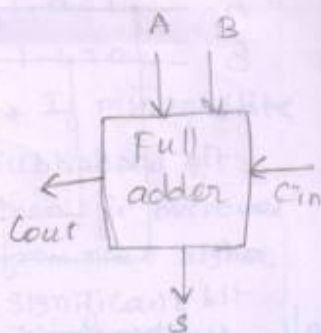


Full adder: It is a combinational circuit that adds three binary bits, and 2 O/P bits.

Truth table

Inputs			Outputs	
A	B	C _{in}	S	C
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

Block diagram

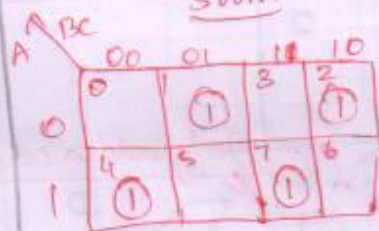


$$S = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$

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

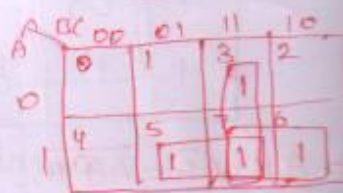
The derived Boolean expression can be further simplified using k-map

Sum



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

Carry



$$C = AB + BC + CA$$

Circuit

sum

in

out

out

out

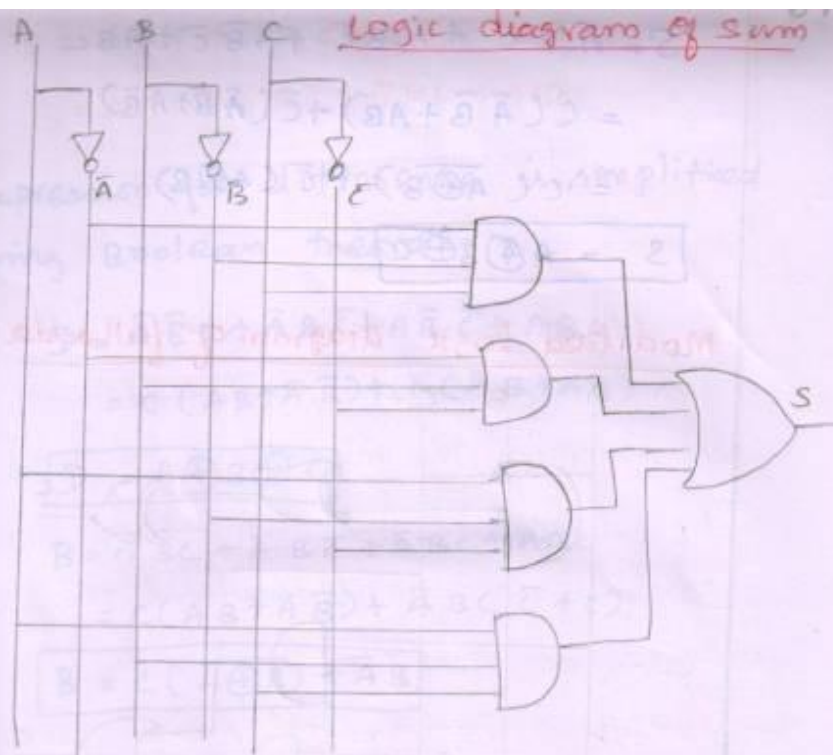
out

out

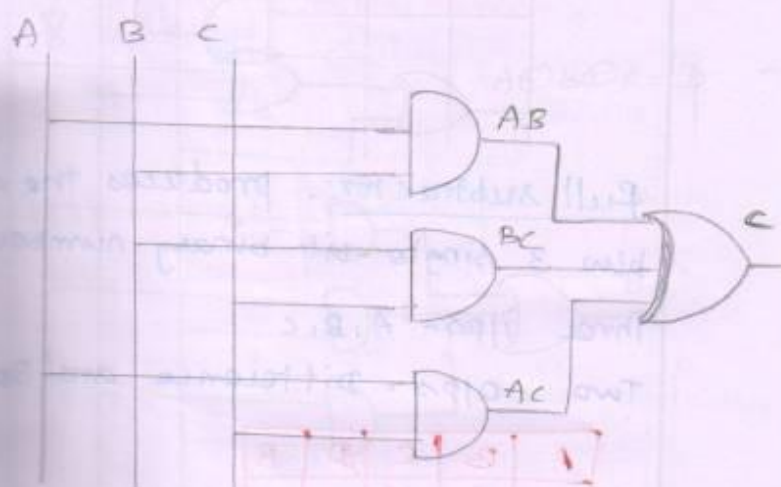
can be

10
2
1

+CA



Logic diagram of carry



The expression for sum could not be simplified using k-map method. It is simplified using Boolean Theorems,

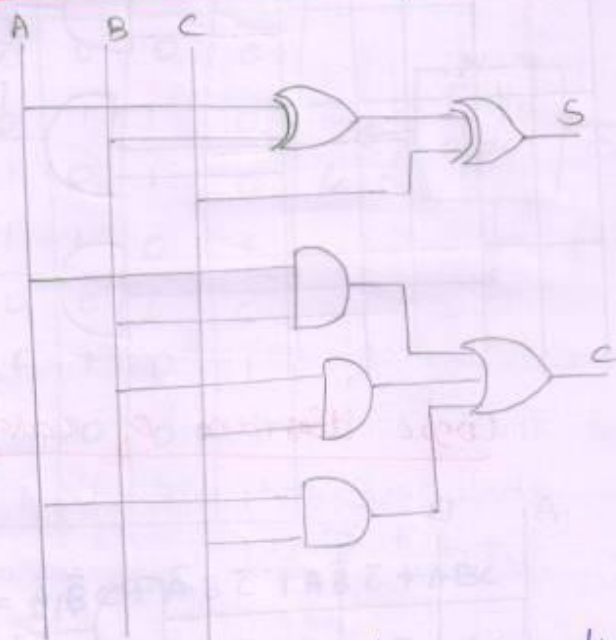
$$S = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$

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

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

$$\boxed{S = A \oplus B \oplus C}$$

Modified logic diagram of full adder:



Full subtractor:- produces the difference
blw 3 single-bit binary numbers.

Three i/p's - A, B, C

Two o/p's - Difference and Borrow

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

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

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

Expression for difference is simplified using Boolean theorems.

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

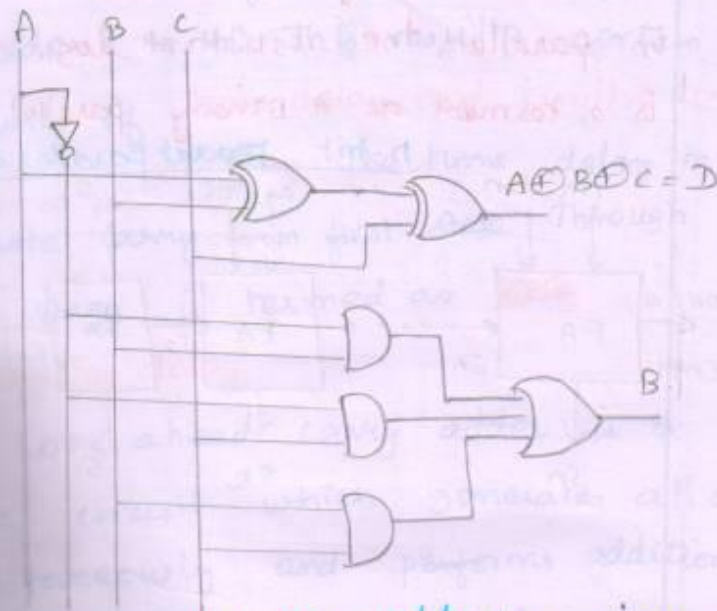
$$= C(\bar{A}\bar{B} + \bar{A}B + A\bar{B} + AB)$$

$$D = A \oplus B \oplus C$$

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

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

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



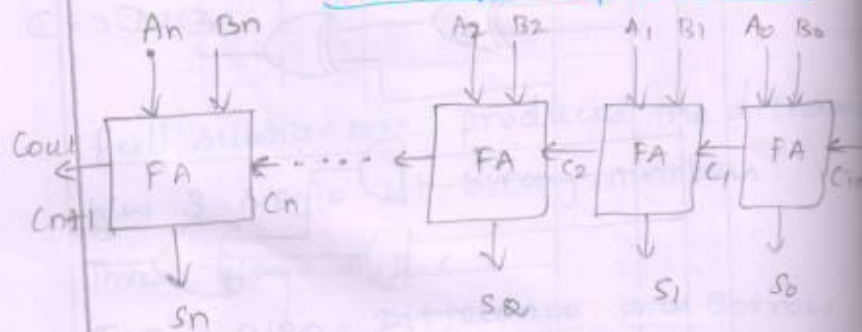
Binary parallel bit adder:- A binary parallel adder is a combinational circuit which produces the sum of 2 binary numbers.

Consider 2 binary numbers.

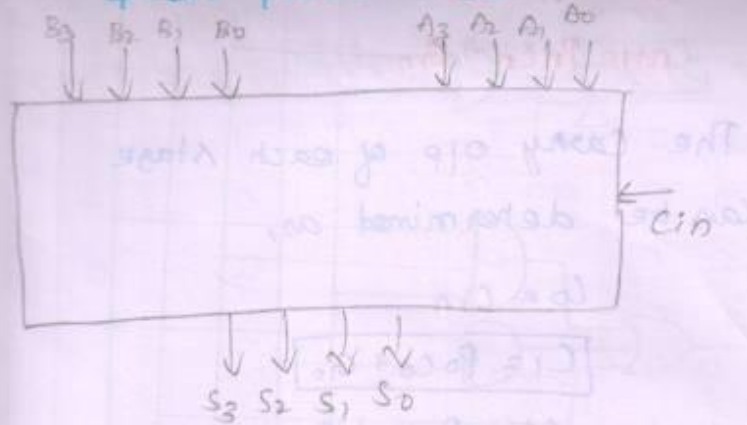
$$\begin{array}{r} 111 \\ 111 \\ \hline 1001 \\ 1000 \\ \hline \end{array}$$

For each column, a full adder is needed for summing the bits. This is achieved by cascading the full adders. The o/p carry of a full-adder forms the i/p carry for the next full adder.

When the full adders are connected in parallel, the resultant logic circuit is termed as a Binary parallel adder.
n-bit parallel adder



4-bit parallel adder.



Carry-Look ahead adder:-

Carry propagation:- In parallel adder, each full adder had to wait for the carry for the next full adder except first full adder (A_0, B_0). This results certain amount of time delay and limits the speed of addition. The time delay to propagate carry from first stage through last stage is termed as ~~carry~~ propagation delay.

Look ahead carry adder is a logic circuit which generates all carry simultaneously and performs addition with increased speed.

Boolean exp for carry propagate P_n ,
Carry generate G_n , sum S_n , Carry C_{n+1} .

$$P_n = A_n \oplus B_n$$

$$S_n = P_n \oplus C_n$$

$$C_{n+1} = P_n C_n + G_n$$

The carry out of each stage can be determined as,

$$C_0 = C_{in}$$

$$C_1 = P_0 C_0 + G_0$$

$$C_2 = P_1 C_1 + G_1$$

sub C_1 in C_2 eqn

$$C_2 = P_1 (P_0 C_0 + G_0) + G_1$$

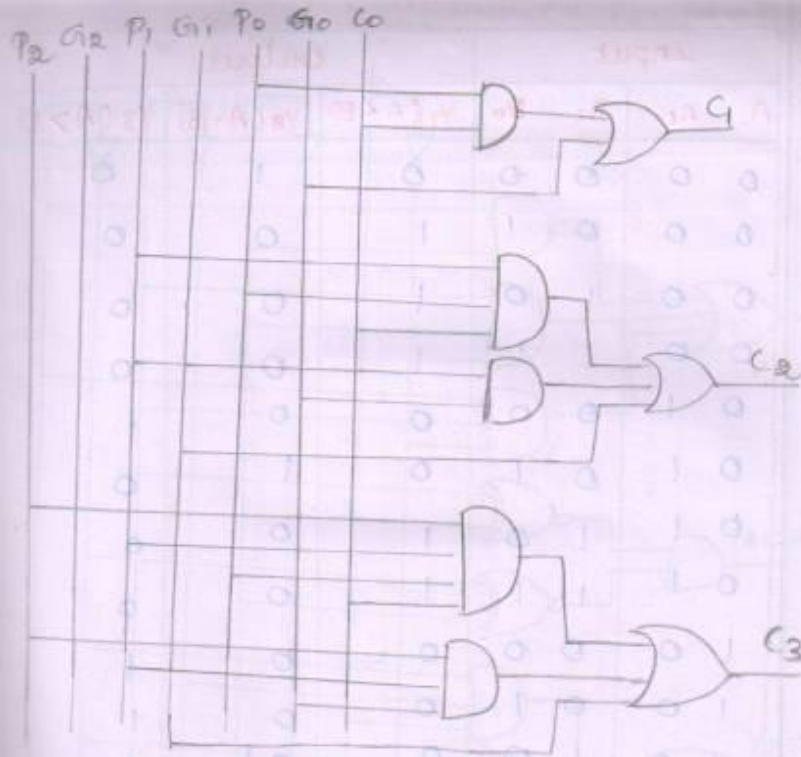
$$C_2 = P_1 P_0 C_0 + P_1 G_0 + G_1$$

$$C_3 = P_2 C_2 + G_2$$

sub C_2 in C_3 eqn.

$$C_3 = P_2 (P_1 P_0 C_0 + P_1 G_0 + G_1) + G_2$$

$$C_3 = P_2 P_1 P_0 C_0 + P_2 P_1 G_0 + P_2 G_1 + G_2$$

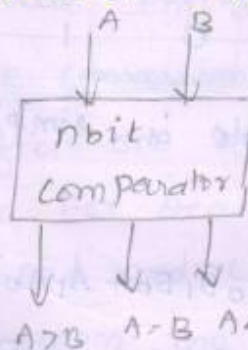


Magnitude Comparator: It compares the magnitudes of a binary numbers.

Two (ip) $\rightarrow A, B$

Three O/p $\rightarrow A > B, A = B, A < B$.

Block diagram



2-bit magnitude comparator:-

Two 2-bit numbers $A = A_1 A_0, B = B_1 B_0$.

produce Three O/p, Y_1, Y_2 , and Y_3 .

$Y_1 \Rightarrow A_1 A_0 < B_1 B_0$

Input				Output		
A ₁	A ₀	B ₁	B ₀	Y ₁ (A < B)	Y ₂ (A = B)	Y ₃ (A > B)
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

The truth table is simplified using the k-map.

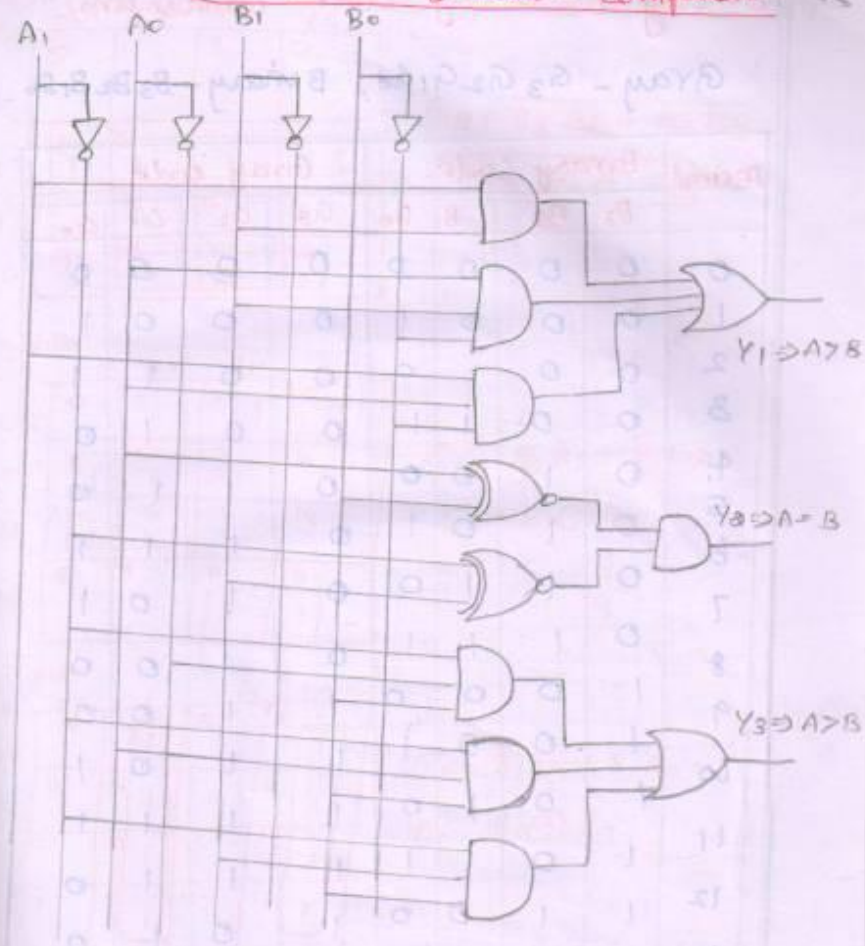
$$Y_1 = A_1 \bar{B}_1 + A_0 \bar{B}_1 B_0 + A_1 A_0 \bar{B}_0$$

$$Y_2 = (A_0 B_0 + \bar{A}_0 \bar{B}_0) (A_1 B_1 + \bar{A}_1 \bar{B}_1)$$

$$= (\overline{A_0 \oplus B_0}) (\overline{A_1 \oplus B_1})$$

$$Y_3 = \bar{A}_1 B_1 + \bar{A}_0 \bar{A}_0 B_0 + \bar{A}_0 B_1 B_0$$

Two-bit magnitude comparator



Code Conversions

Need for code conversion: It is necessary to convert from one binary code to another binary code. For example, consider a simple key-pad and LED display sm. This sm accepts the i/p in BCD number and displays the number on 7-segment LED. But the sm is capable of processing the data in binary form only. In this

Binary to Gray code conversions:

Gray - $G_3 G_2 G_1 G_0$, Binary - $B_3 B_2 B_1 B_0$.

Decimal	Binary Code				Gray code			
	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
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
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

k-map simplification

$B_3 B_2$	$B_1 B_0$			
	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$$G_3 = B_3$$

$B_3 B_2$

	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$$G_3 = B_3 B_2 + B_3 B_2$$

$$G_3 = B_3 \oplus B_2$$

$B_2 B_1$

	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$$G_1 = B_2 \bar{B}_1 + \bar{B}_1 B_2$$

$$G_1 = B_1 \oplus B_2$$

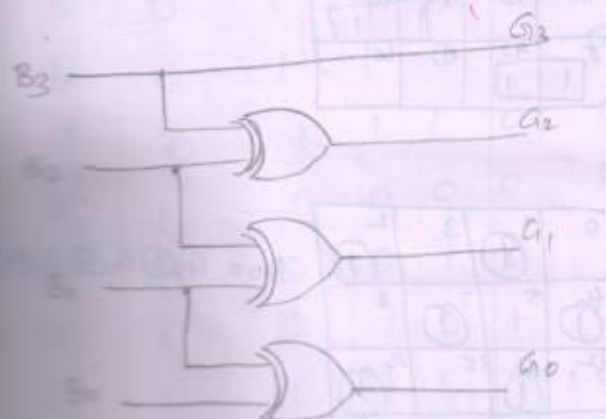
$B_1 B_0$

	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$$G_0 = \bar{B}_1 B_0 + B_1 \bar{B}_0$$

$$G_0 = B_0 \oplus B_1$$

Logic diagram:- $G_3 = B_3, G_2 = B_3 \oplus B_2$
 $G_1 = B_2 \oplus B_1, G_0 = B_1 \oplus B_0.$



Gray to Binary :-

k-map simplification:-

	$G_1 G_0$ 00 01 11 10			
$G_3 G_2$	0	1	3	2
00				
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$$B_3 = G_3$$

	$G_1 G_0$ 00 01 11 10			
$G_3 G_2$	0	1	3	2
00				
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

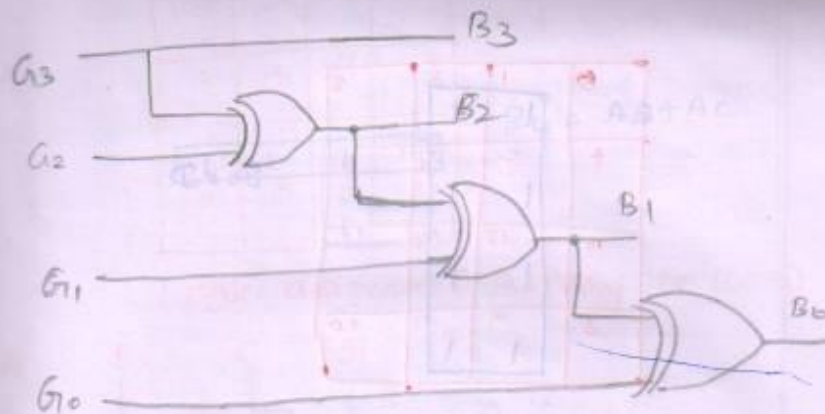
$$B_2 = G_3 \oplus G_2$$

	$G_1 G_0$ 00 01 11 10			
$G_3 G_2$	0	1	3	2
00				
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$$B_1 = G_3 \oplus G_2 \oplus G_1$$

0	1	3	2
4	5	7	6
12	13	15	14
8	9	11	10

$$B_0 = G_3 \oplus G_2 \oplus G_1 \oplus G_0$$



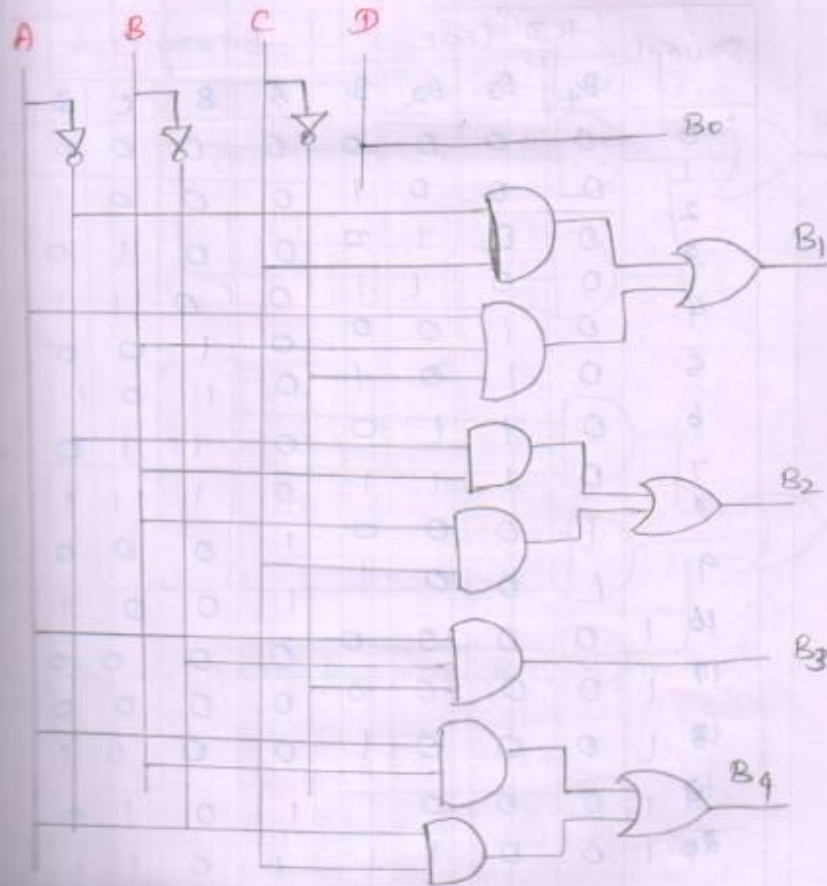
Binary to BCD:- Converts Binary number into its equivalent BCD Code.

Decimal	Binary code				BCD Code				
	A	B	C	D	B ₄	B ₃	B ₂	B ₁	B ₀
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1	0
3	0	0	1	1	0	0	0	1	1
4	0	1	0	0	0	0	1	0	0
5	0	1	0	1	0	0	1	0	1
6	0	1	1	0	0	0	1	1	0
7	0	1	1	1	0	0	1	1	1
8	1	0	0	0	0	1	0	0	0
9	1	0	0	1	0	1	0	0	1
10	1	0	1	0	1	0	0	0	0
11	1	0	1	1	1	0	0	0	1
12	1	1	0	0	1	0	0	1	0
13	1	1	0	1	1	0	0	1	1

0	1	3	2
4	5	7	6
12	13	15	14
8	9	11	10

$$B_4 = AB + AC$$

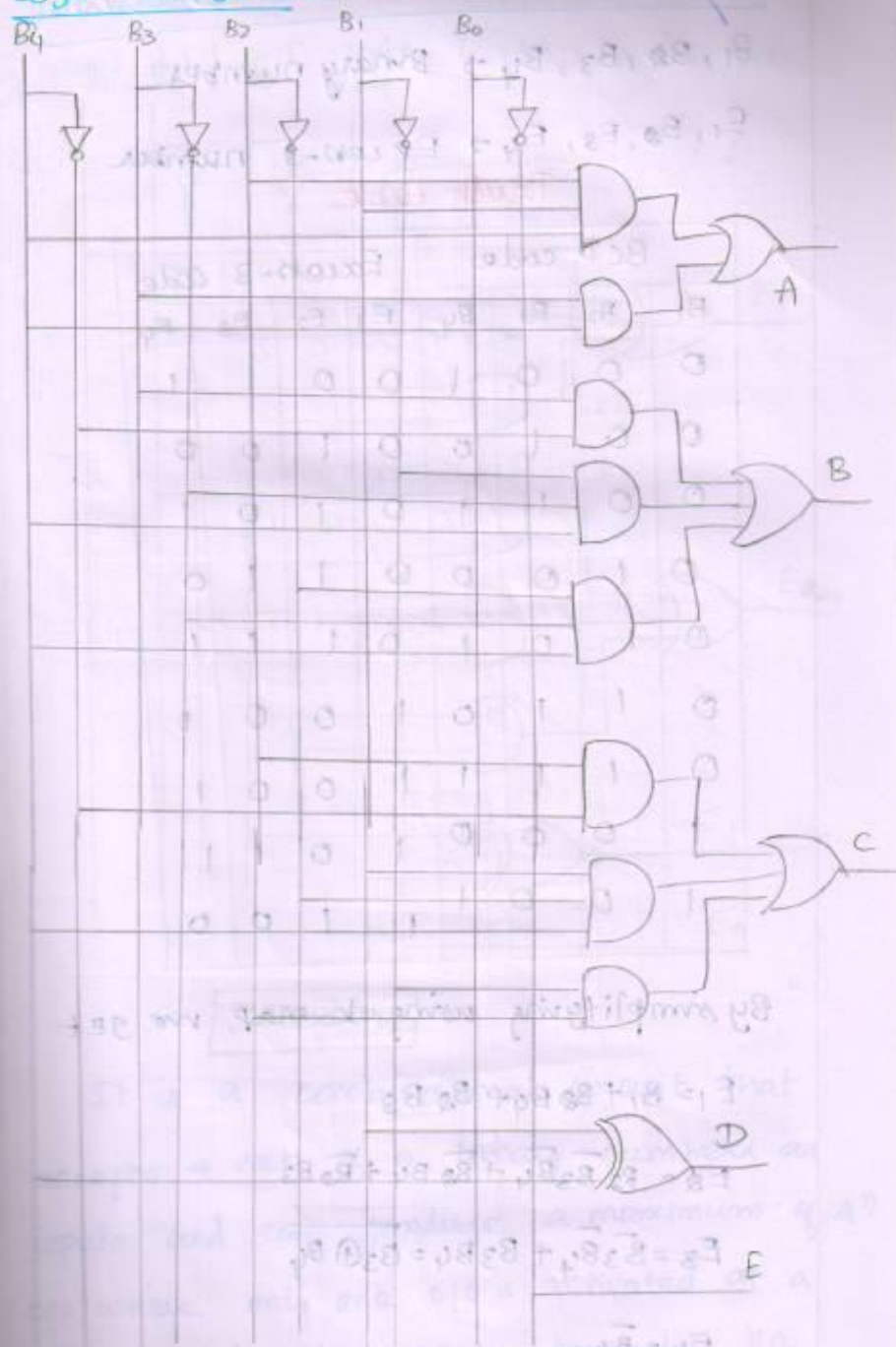
Logic diagram (Binary to BCD)



BCD to Binary Conversion:- It is known that the binary and BCD are the same from decimal numbers 0 through 9. Hence, a minimum of five I/P Variables are required to design BCD to Binary Converter.

Decimal	BCD Code				Binary Code			
	B ₄	B ₃	B ₂	B ₁	A	B	C	D
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	0
3	0	0	1	1	0	0	1	1
4	0	1	0	0	0	1	0	0
5	0	1	0	1	0	1	0	1
6	0	1	1	0	0	1	1	0
7	0	1	1	1	0	1	1	1
8	1	0	0	0	1	0	0	0
9	1	0	0	1	1	0	0	1
10	1	0	0	0	0	0	0	0
11	1	0	0	0	0	0	0	0
12	1	0	0	1	0	0	0	1
13	1	0	0	1	1	0	1	0
14	1	0	1	1	1	0	1	1
15	1	0	1	1	1	1	0	0

Logic diagram:-



BCD to Excess-3 code conversions:

$B_1, B_2, B_3, B_4 \rightarrow$ Binary number

$E_1, E_2, E_3, E_4 \rightarrow$ Excess-3 number

Truth table

BCD code				Excess-3 code			
B_1	B_2	B_3	B_4	E_1	E_2	E_3	E_4
0	0	0	1	0	0	1	1
0	0	1	0	0	1	0	0
0	0	1	1	0	1	0	1
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	1	0	0	0
0	1	1	1	1	0	0	1
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

By simplifying using k-map, we get

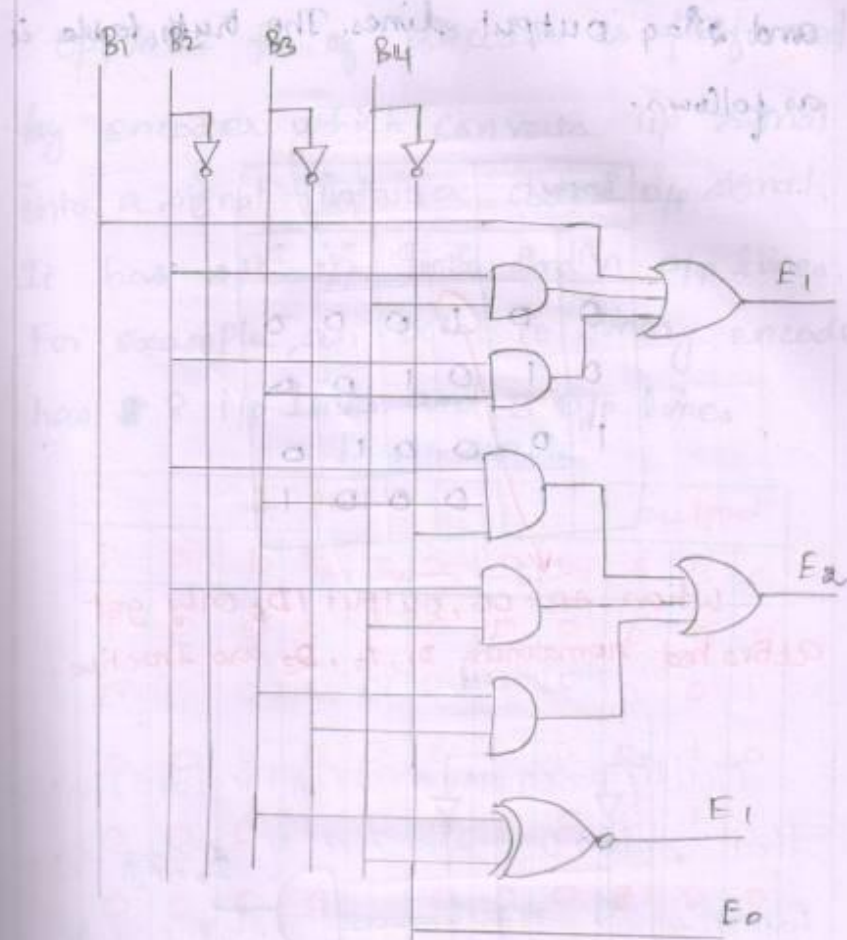
$$E_1 = B_1 + B_2 B_4 + B_2 B_3$$

$$E_2 = B_2 \bar{B}_3 \bar{B}_4 + \bar{B}_2 B_4 + \bar{B}_2 B_3$$

$$E_3 = \bar{B}_3 \bar{B}_4 + B_3 B_4 = B_3 \oplus B_4$$

$$E_4 = \bar{B}_4$$

Logic diagram for BCD to Excess-3



Decoders

It is a combinational circuit that accepts a set of n binary numbers as inputs and can produce a maximum of 2^n outputs where only one output is activated at a time corresponding to a particular input.

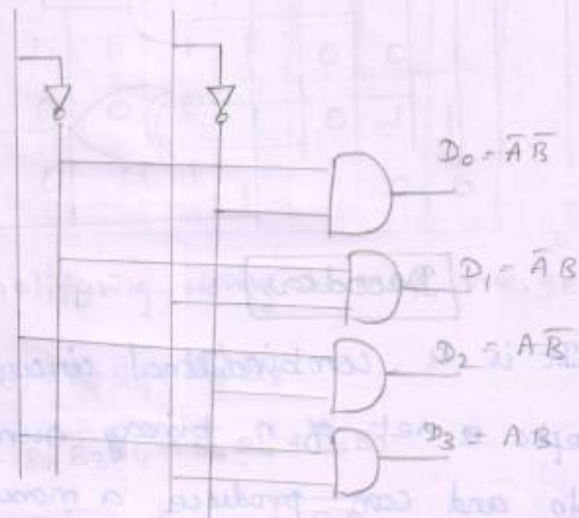
If a decoder has m output lines then $m \leq 2^n$.

Input \rightarrow Decoder \rightarrow $m \leq 2^n$ outputs

2 to 4 line decoder has 2 input lines and $2^2 = 4$ output lines. The truth table is as follows.

Input		output			
A	B	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

When $AB = 00$, output D_0 only get activated. remaining D_1, D_2, D_3 are inactive.



Logic diagram of 2 to 4 decoder

Encoder

opposite fn of decoder is performed by encoder which converts i/p signal into a signal into a coded o/p signal. It has 2^n i/p lines and n o/p lines. For example, an octal to binary encoder has 8 i/p lines and 3 o/p lines.

Truth table

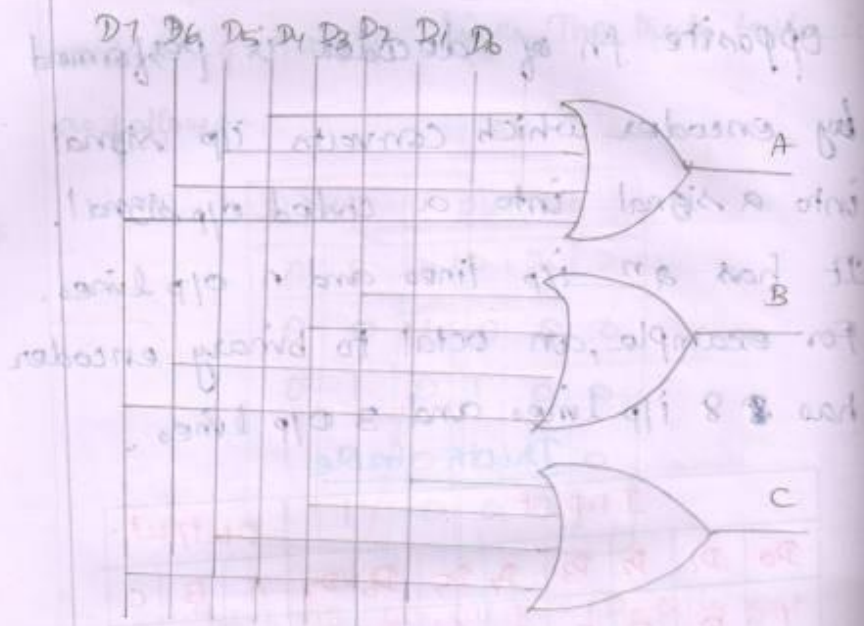
Input								Output		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	A	B	C
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Logical expressions:-

$$A = D_4 + D_5 + D_6 + D_7 \text{ (MSD)}$$

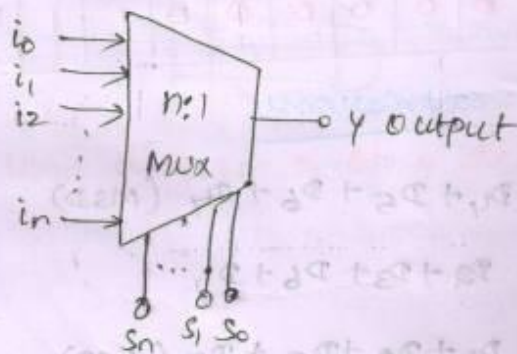
$$B = D_2 + D_3 + D_6 + D_7$$

$$C = D_1 + D_3 + D_5 + D_7 \text{ (LSD)}.$$



Multiplexer

Multiplexer is a Combinational Circuit that selects one out of many i/p's. normally 2^n inputs and only one o/p. The selection of i/p is done by n select lines.



4:1 mux:- four i/p lines - i_0, i_1, i_2, i_3 .

two select lines - s_0, s_1 .

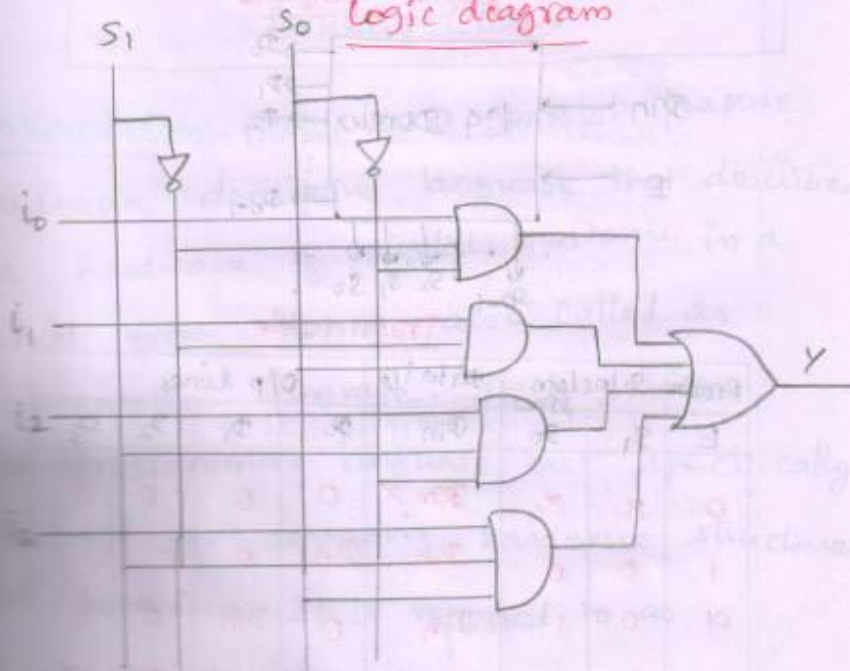
Based on selection information, the mux selects one i/p and sends it to o/p y .

Truth table

select I/P		output
s_1	s_0	y
0	0	i_0
0	1	i_1
1	0	i_2
1	1	i_3

Boolean expression $y = \bar{s}_0 \bar{s}_1 i_0 + \bar{s}_1 s_0 i_1 + s_1 \bar{s}_0 i_2 + s_1 s_0 i_3$

Logic diagram

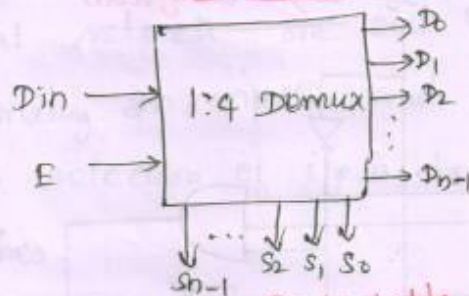


Demultiplexers

A demultiplexer is a combinational circuit that performs the reverse operation of mux. It receives the information on a single i/p and distributes it over many o/p lines. The selection lines specify one of several o/p lines where the information has to be transmitted.

4:1 Demux: 1:4 demux accepts data on a single i/p variable D_{in} (Data i/p) and distributes over four o/p's, D_0, D_1, D_2 & D_3 .
Two selection lines: S_1 and S_0 .

Block diagram



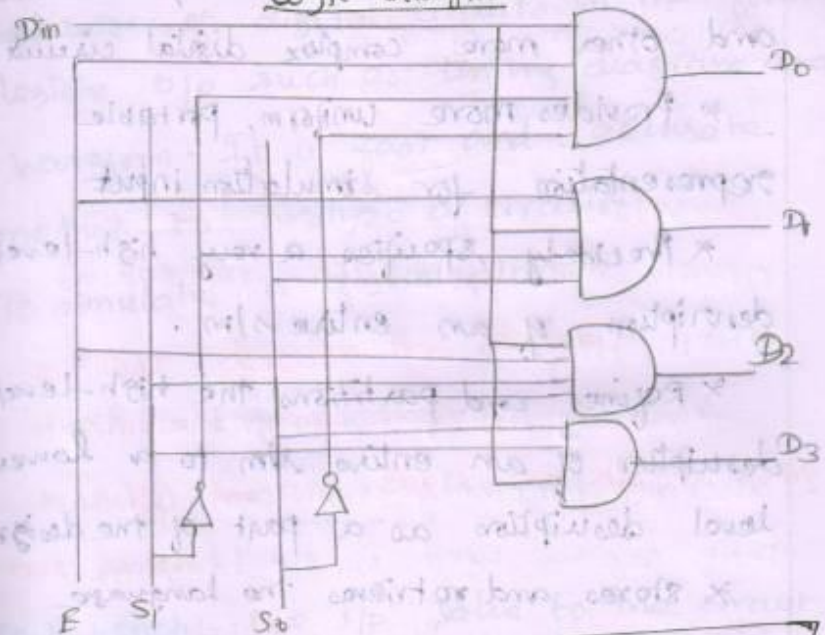
Truth table

Enable E	Selection lines		Data i/p D_{in}	o/p lines			
	S_1	S_0		D_0	D_1	D_2	D_3
0	x	x	D_{in} X	0	0	0	0
1	0	0	D_{in}	D_{in}	0	0	0
1	0	1	D_{in}	0	D_{in}	0	0
1	1	0	D_{in}	0	0	D_{in}	0
1	1	1	D_{in}	0	0	0	D_{in}

$$D_0 = \bar{S}_1 \bar{S}_0 D_{in}, D_1 = \bar{S}_1 S_0 D_{in}, D_2 = S_1 \bar{S}_0 D_{in}, D_3 = S_1 S_0 D_{in}$$

$$D_3 = S_1 S_0 D_{in}$$

logic diagram



Introduction to Hardware Description Language (HDL)

Introduction: HDL is a general purpose hardware descriptive language that describes the hardware of digital systems in a textual form. hence also called as documentation language. It is similar to a programming language, but specifically developed for describing hardware structures and behaviours. It is referred to as a structural description for describing an

Features:-

- * Represents logic diagrams, Boolean exp and other more complex digital circuits
- * Provides more uniform, portable representation for simulation input
- * Precisely specifies a very high-level description of an entire sm.
- * Refines and partitions the high-level description of an entire sm to a lower-level description as a part of the design.
- * Stores and retrieves the language

Content.

Major design steps:-

- * Design entry
- * Simulation
- * Synthesis and timing verification
- * Fault simulation

Design entry:-

It writes description of a hardware functionality. The description could be

* Boolean expression

* Truth table

* Interconnection of gates

Logic simulation:- is the application of simulation software that displays the behavior of digital circuits in the form of legible o/p such as timing diagram and waveform. It is fast and accurate method to analyze a circuit.

To simulate a digital circuit

* First describe the design

* Simulate the design

* Verify and check the design using test bench

Test bench:- the i/p value to the circuit that tests operation of a design is known as test bench.

Logic synthesis & timing verification:-

Logic synthesis is a process by which an abstract form of desired circuit (netlist in terms of logic gates) is described in HDL. It generates database that describes the elements and structure of a circuit which in turn helps to fabricate IC.

Net list:- Physical components and their interconnections

Timing verification checks each signal path for any propagation delay and confirms the operation of fabricated IC.

Types of HDL:-

xVHDL (Very High speed Integrated Circuits HDL), sometimes referred as VHSIC HDL.

A verilog HDL has a syntax that describes precisely the legal constructs that can be used in the language.

Starting with verilog - module and some important keywords:-

module:- is the basic building block in verilog. It is denoted by 'module' and 'endmodule' that contains HDL text.

Identifier-list of variables.

Input and output indicates whether the variable is i/p or O/p.

Important data type

Wire - declares interconnection, if any it

Integer - stores the value used to manipulate quantities

reg - declares variable in memory

Predefined logic values:

* 0, 1, X and Z

* X \rightarrow uninitialized or unknown value

* Z \rightarrow high impedance value

Bitwise operators:

* Bitwise NOT

\sim

* AND

$\&$

* OR

$|$

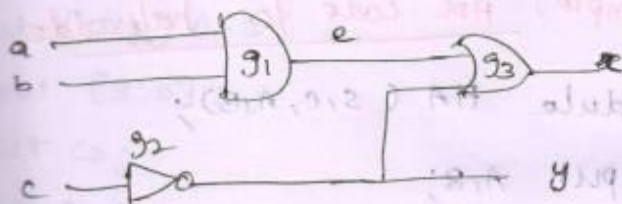
* XOR

\wedge

* XNOR

$\sim \wedge$ or $\wedge \sim$

Example HDL model for simple circuit



module simple_circuit (a,b,c, x,y);

input a,b,c;

output x,y;

wire e;

and g1 (e,a,b) // first letter- o/p

not g2 (y,c) and then i/p

or g3 (x,e).

endmodule.

HDL for combinational circuits

* Gate level or structural modeling

(Circuit is constructed in terms of gates)

* Data flow modeling

(It uses 'assign' keyword followed by predictive o/p and equal sign)

* Behavioral modeling

(Describes the behaviour of the circuit, Mostly used for sequential circuits, It uses the key word 'always' followed by expression).

(Example) HDL code for half adder

```
module HA (S, C, A, B);
```

```
input A, B;
```

```
output S, C;
```

```
Xor (S, A, B);
```

```
and (C, A, B);
```

```
endmodule
```


HDL code for full adder

```
module FA (S, Cout, A, B, Cin);  
  input A, B, Cin;  
  output S, Cout;  
  wire S1, C1, C2;  
  halfadder HA1 (S1, C1, A, B);  
  halfadder HA2 (S, C2, S1, C1);  
  or G1 (Cout, C2, C1);  
endmodule
```

HDL code for 4 bit adder

```
module 4bit-adder (S, C4, A, B, Co);  
  input [3:0] A, B;  
  output [3:0] S;  
  input Co;  
  output C4;  
  wire C1, C2, C3;  
  fulladder FA0 (S[0], C1, A[0], B[0], C[0]);  
  fulladder FA1 (S[1], C2, A[1], B[1], C[1]);  
  fulladder FA2 (S[2], C3, A[2], B[2], C[2]);  
  fulladder FA3 (S[3], C4, A[3], B[3], C[3]);  
endmodule
```

Unit-3 Synchronous Sequential Circuits

Sequential Circuits, latches and flipflops -

Analysis and design procedures - state

Reduction and state Assignment -

Shift Registers - Counters - HDL for sequential logic circuits.

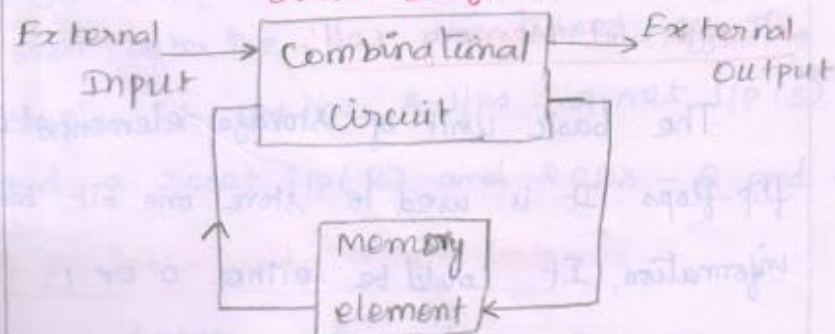
Sequential circuits - In sequential circuits

the outputs at any instant of time are dependent not only on the present inputs but also on past inputs. In these

circuits the o/p signals are feedback nature

The memory elements are capable of storing binary information. A memory element is used to store one bit binary information.

Block Diagram



sequential circuits are classified into 2 types

(i) Synchronous sequential circuit

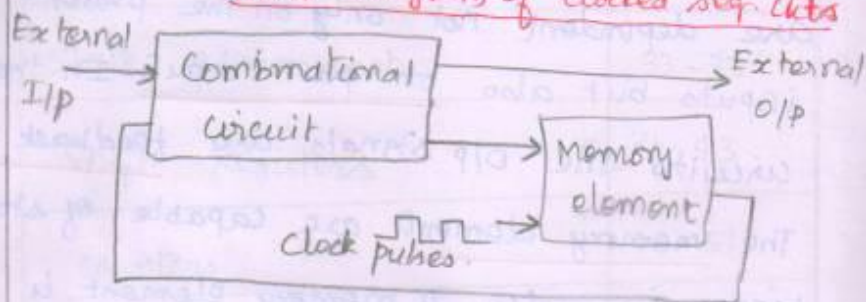
* Asynchronous sequential circuit.

Synchronous sequential circuits:-

→ is a circuit whose o/p depends on present i/p and present state at discrete instants of time. The instances of time are defined by the clock i/p and memory elements to give the next state.

* It uses clock pulses in the i/p of memory elements are clocked sequential circuits

Block diagram of clocked seq. ckt's



Latches and FlipFlops

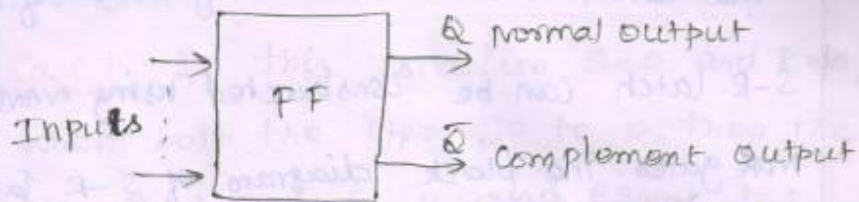
A one-bit memory cell:-

The basic unit of storage elements is the flip-flops. It is used to store one bit binary information. It could be either 0 or 1.

It is obtained by using NAND or NOR gates.

The FF has 2 O/p's, one → normal o/p Q .

another → complement of Q .



FF has 2 stable state, set and reset.
If the normal output $Q=1$, the ckt is referred as set state. If $Q=0$, then it is referred as reset state.

The basic flip-flop types are,

(i) S-R flip-flop

(ii) J-K "

(iii) Delay FF (DFF)

(iv) Toggle FF (TFF)

S-R-FF: An S-R FF is an arrangement of logic gates that maintains a stable o/p even after the i/p's are turned off. This simple FF ckt has 2 i/p's - a set i/p (S) and a reset i/p (R) and 2 o/p's - Q and \bar{Q}

S-R-latch with NAND Gate:

Latch	FF
It is a level sensitive memory element	It is a edge sensitive memory element

operation :

Case 1:- In this condition $S=0$ and $R=0$, when both the i/p's go to 0 then the o/p $Q=1$ and \bar{Q} is also equal to 1, which should be avoided. The o/p Q & \bar{Q} are always complementary. They should not be at same states - either 0 or 1.

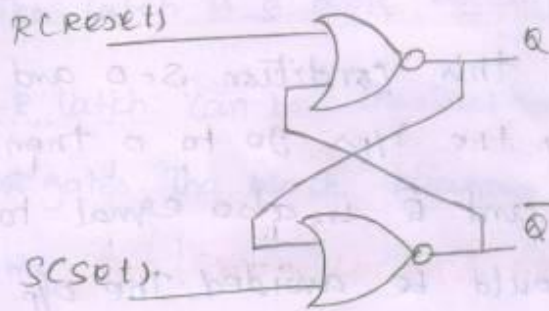
Case 2:- When $S=1$ and $R=0$ then the o/p $Q=0$. The condition clears the latch.

Case 3:- $S=0$, $R=1$ then the o/p $Q=1$ & $\bar{Q}=0$. The $S=0$ and $R=1$ is said to be set the state of the latch.

Case 4:- $S=1$, $R=1$ does not affect the state of the latch. It remains in the previous state.

NOR S-R Latch:- The i/p signals of NOR latch are complement - are those values used for the NAND latch. Another name of NAND latch is sometimes $\bar{S}\bar{R}$ latch also, since the NAND latch needs 0 at its i/p to change its state.

RC Reset)



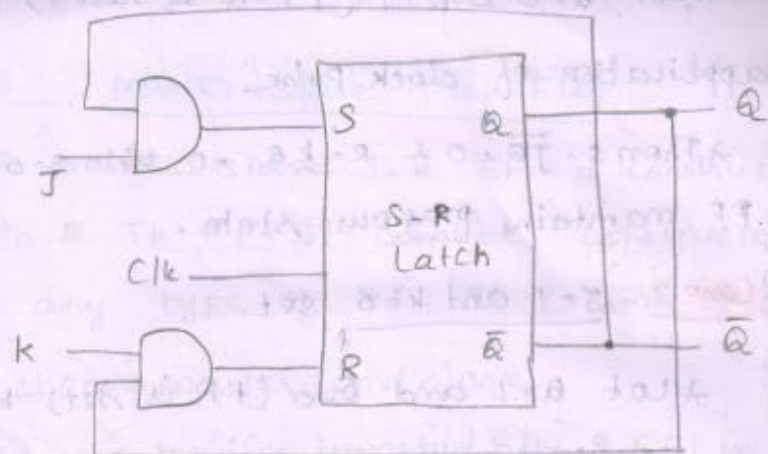
SC Set)

Truth table of S-R latch (NOR).

Inputs		Outputs		Comments
S	R	Q	\bar{Q}	
0	0	1	0	No change
0	1	0	1	Reset
1	0	1	0	Set
1	1	0	0	Forbidden

J-k-Flipflop

The indeterminate condition of S-R FF, when both i/p's are equal to one, could be eliminated by the J-k-FF. The S-R latch can be modified with 2 additional AND gates. The o/p of S-R latch are connected to the one i/p of each AND gate. The other i/p's of AND gates are connected to i/p signals J and k. Therefore, $S = J\bar{Q}$ and



Truth table of J-k FF

Clock i/p	Inputs		Outputs		Comments
	J	k	Q	\bar{Q}	
0	x	x	1	0	No change
1	0	0	1	0	No change
1	0	1	0	1	Reset
1	1	0	1	0	Set
1	1	1	0	1	Toggle

Case 1:- $J=0$ and $k=0$ No change

* Both the AND gates gives 0 o/p's.

* The i/p's to the basic FF are $S=0$ & $R=0$ and FF maintains the present state.

Case 2:- $J=0$ & $k=1$ Reset

* Let $Q=1$ & $\bar{Q}=0$ (FF is set) before application of clock pulse.

* Then $S = J\bar{Q} = 0$ and $R = kQ = 1$, with

* Let $A=0$ & $\bar{A}=1$ (FF is reset) before application of clock pulse.

* Then $S = \bar{J}A = 0$ & $R = KA = 0$ with $S=0$, $R=0$, FF maintains previous state.

Case 3:- $J=1$ and $K=0$ set.

* Let $A=1$ and $\bar{A}=0$ (FF is set) before application of clock pulse.

* Then $S = \bar{J}A = 0$ and $R = KA = 0$ with $S=0$ & $R=0$, FF will be at previous state.

* Let $A=0$, $\bar{A}=1$ (FF is reset), before application of clock pulse.

* Then $S = \bar{J}A = 1$, $R = KA = 0$, with $S=1$ & $R=0$ FF will be set.

Case 4:- $J=1$ and $K=1$, Toggle

* Let $A=1$ and $\bar{A}=0$ (FF is set) before the application of clk pulse.

* Then $S = \bar{J}A = 0$ and $R = KA = 1$, with $S=0$, $R=1$, FF will be reset, that is complement of previous state.

* Let $A=0$, $\bar{A}=1$ (FF is reset) before application of clk pulse.

* With $J = K = 1$, FF will toggle.

Master-Slave Flip-Flop

A master-slave J-K FF is constructed with 2 JK FF. It can be constructed for any type of FF. It consists of 2 sections - master and slave.

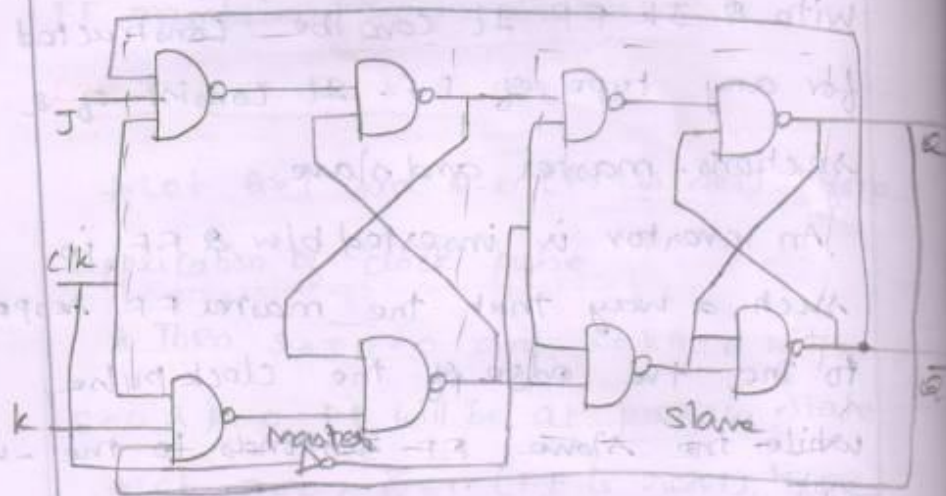
An inverter is inserted b/w 2 FF in such a way that the master FF responds to the +ve edge of the clock pulse, while the slave FF responds to the -ve edge of the clock pulse.

Master-Slave J-K FF



Operation:- When a clk pulse is 1 the master FF is enabled. Since the same clk pulse is applied to slave FF through the inverter, the slave FF receives 0 clk pulse and it is disabled. When pulse becomes 0, the master is disabled and the slave is

is made isolated from slave FF and thus prevented from external inputs affecting it.



* If $J=1$ and $K=0$, the master FF sets on the \uparrow ve clk edge. The o/p of the master, drives the i/p of the slave FF. When the \downarrow ve going edge of the clk sigl is reached, the slave FF also sets.

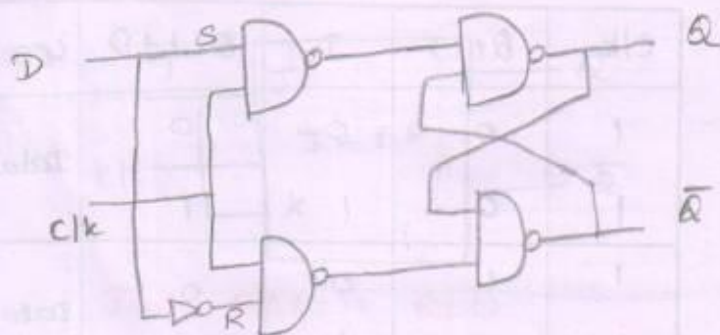
* If $J=0$ and $K=1$, the master FF resets on the \uparrow ve edge clk pulse, when the \downarrow ve clk edge reached the slave FF is also reset.

Delay Flip Flop

(11)

The delay FF is a modification of S-R FF. Just by inserting an inverter in the circuit b/w S and R i/p terminals S-R FF, a D FF can be formed. The inverter in the ckt never allows both the i/p to be 1 and hence it eliminates the indeterminate condition of S-R FF.

Delay FF



Case 1:- $clk = 0$

*The o/p of first 2 gates (from left) are 1 and FF maintains the previous state

Case 2:- $clk = 1$

*Let $D = 1$, the o/p of gate 1 is 0 which drives gate 3 to produce 1 and hence Q o/p goes to 1.

Let $D=0$, i/p to the gate 2 is 1.

Gate 2 gives 0 o/p, which makes gate 4 to give 1, in turn the o/p Q goes to 0.

Delay FF (Symbolic)



Characteristic Table

clk	$Q(t)$	D	$Q(t+D)$	Comments
1	0	0	0	Data input
1	0	1	1	
1	1	0	0	Data input
1	1	1	1	

Characteristic equation of DFF

$$Q(t+D) = D$$

T Flip-Flop

(13)

Toggle FF is a modified version of J-k FF. To obtain TFF, the i/p's J and K are tied together. When TFF has the single i/p T and a clk i/p.

* When the T i/p is 1 and clock pulse is not applied, the o/p does not change. The o/p maintains the present state.

* When the T i/p is 1, and $clk = 1$, The o/p Q is the complement of present state.



$$T=0, Q(t+1) = Q(t)$$

$$T=1, Q(t+1) = \bar{Q}(t)$$

Characteristic equation:

clk	Q(t)	T	Q(t+1)	Comments
1	0	0	0	No change
1	0	1	1	Toggle
1	1	0	1	No change
1	1	1	0	Toggle

Application of Flip-Flop

Applications of FF.

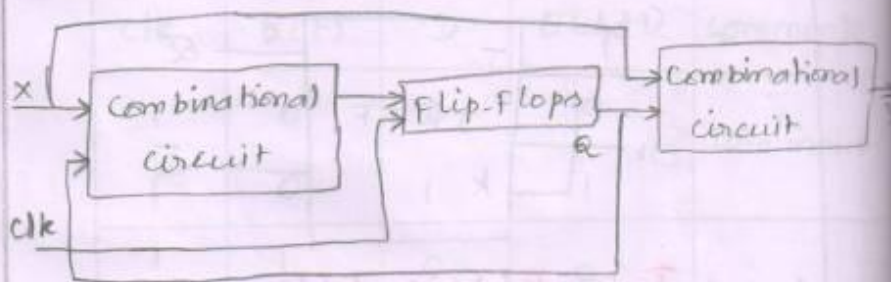
(i) Counters

(ii) Shift Registers

(iii) Memory.

Synchronous Sequential Circuits:-

Output of ckt always depend on present state, however this connection exist when o/p depends on primary i/p's also along with present state.



Terms and Definitions used:-

Input and output Variables:-

The variables that enter the machine is termed as i/p variables and all out going variables are termed as o/p variables.

Excitation Variables:- Input variables to memory are defined as excitation variables. they excite memory change of states.

State and State variables: The content⁽¹⁵⁾

of memory is termed as state. The o/p of FF is termed as state variables.

Both the terms and state variable are closely related.

Present state and next state: The present

state indicates the state of machine before occurrence of a clk pulse whereas next state is the memory status of machine after a clk pulse applied.

State table and Transition table:

The state information of the ckt along with its i/p-o/p behaviour can be represented in the form of table is known as state table.

Another form of state table is known as transition table where states are represented with binary values instead of symbols.

State Diagram: It is a graphical

representation of ckt that shows the progression of states when machine is clocked. It is somewhat similar to flow chart. It consists of no. of circles

(2) The circle represent states and the directed lines indicate the transition b/w states.

State equation: The relation of next state to present state and ip can be shown in the form of algebraic eqn is termed as state eqn.

State Reduction: It is a process of removing one of 2 equivalent states without altering the i/p-o/p condition of machine. With state reduction technique It is possible to reduce a considerable no. of states from state table which may result less requirement of no. of FF or no. of gates while designing a seq. ckt.

State Assignment: It is a procedure of assigning binary values to the state may be arbitrary or based on certain guidelines.

Symbols, Characteristic Tables and Excitation tables of all FF

Type	Symbol	Characteristic table	Excitation Table																																			
SR FF		<table border="1"> <thead> <tr> <th>S</th> <th>R</th> <th>$Q(t+1)$</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>$Q(t)$</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>?</td> </tr> </tbody> </table>	S	R	$Q(t+1)$	0	0	$Q(t)$	0	1	0	1	0	1	1	1	?	<table border="1"> <thead> <tr> <th>$Q(t)$</th> <th>$Q(t+1)$</th> <th>S</th> <th>R</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>x</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>x</td> <td>1</td> </tr> </tbody> </table>	$Q(t)$	$Q(t+1)$	S	R	0	0	0	x	0	1	0	1	1	0	1	0	1	1	x	1
S	R	$Q(t+1)$																																				
0	0	$Q(t)$																																				
0	1	0																																				
1	0	1																																				
1	1	?																																				
$Q(t)$	$Q(t+1)$	S	R																																			
0	0	0	x																																			
0	1	0	1																																			
1	0	1	0																																			
1	1	x	1																																			
JK FF		<table border="1"> <thead> <tr> <th>J</th> <th>K</th> <th>$Q(t+1)$</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>$Q(t)$</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>$\bar{Q}(t)$</td> </tr> </tbody> </table>	J	K	$Q(t+1)$	0	0	$Q(t)$	0	1	0	1	0	1	1	1	$\bar{Q}(t)$	<table border="1"> <thead> <tr> <th>$Q(t)$</th> <th>$Q(t+1)$</th> <th>J</th> <th>K</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>x</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>x</td> </tr> <tr> <td>1</td> <td>0</td> <td>x</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>x</td> <td>1</td> </tr> </tbody> </table>	$Q(t)$	$Q(t+1)$	J	K	0	0	0	x	0	1	0	x	1	0	x	0	1	1	x	1
J	K	$Q(t+1)$																																				
0	0	$Q(t)$																																				
0	1	0																																				
1	0	1																																				
1	1	$\bar{Q}(t)$																																				
$Q(t)$	$Q(t+1)$	J	K																																			
0	0	0	x																																			
0	1	0	x																																			
1	0	x	0																																			
1	1	x	1																																			
D FF		<table border="1"> <thead> <tr> <th>D</th> <th>$Q(t+1)$</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> </tr> </tbody> </table>	D	$Q(t+1)$	0	0	1	1	<table border="1"> <thead> <tr> <th>$Q(t)$</th> <th>$Q(t+1)$</th> <th>D</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	$Q(t)$	$Q(t+1)$	D	0	0	0	0	1	1	1	0	0	1	1	1														
D	$Q(t+1)$																																					
0	0																																					
1	1																																					
$Q(t)$	$Q(t+1)$	D																																				
0	0	0																																				
0	1	1																																				
1	0	0																																				
1	1	1																																				
T FF		<table border="1"> <thead> <tr> <th>T</th> <th>$Q(t+1)$</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>$Q(t)$</td> </tr> <tr> <td>1</td> <td>$\bar{Q}(t)$</td> </tr> </tbody> </table>	T	$Q(t+1)$	0	$Q(t)$	1	$\bar{Q}(t)$	<table border="1"> <thead> <tr> <th>$Q(t)$</th> <th>$Q(t+1)$</th> <th>T</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	$Q(t)$	$Q(t+1)$	T	0	0	0	0	1	1	1	0	1	1	1	0														
T	$Q(t+1)$																																					
0	$Q(t)$																																					
1	$\bar{Q}(t)$																																					
$Q(t)$	$Q(t+1)$	T																																				
0	0	0																																				
0	1	1																																				
1	0	1																																				
1	1	0																																				

Characteristic Equation of FF

Type	characteristic Eqn
SR FF	$Q(t+1) = S + \bar{R}Q(t)$
JK FF	$Q(t+1) = J\bar{Q}(t) + \bar{k}Q(t)$
D FF	$Q(t+1) = D$
T FF	$Q(t+1) = T \oplus Q(t)$

State Diagrams and state Tables of FF

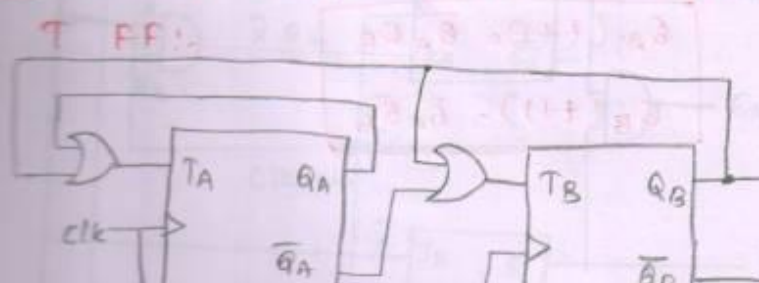
Type	state Table	State Diagram																				
SR FF	<table border="1"> <thead> <tr> <th>$Q(t)$</th><th>S</th><th>R</th><th>$Q(t+1)$</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td><td>0</td></tr> </tbody> </table>	$Q(t)$	S	R	$Q(t+1)$	0	0	0	0	0	1	0	1	1	0	0	1	1	0	1	0	
$Q(t)$	S	R	$Q(t+1)$																			
0	0	0	0																			
0	1	0	1																			
1	0	0	1																			
1	0	1	0																			
JK FF	<table border="1"> <thead> <tr> <th>$Q(t)$</th><th>J</th><th>k</th><th>$Q(t+1)$</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0/1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>0/1</td><td>1</td><td>0</td></tr> </tbody> </table>	$Q(t)$	J	k	$Q(t+1)$	0	0	0	0	0	1	0/1	1	1	0	0	1	1	0/1	1	0	
$Q(t)$	J	k	$Q(t+1)$																			
0	0	0	0																			
0	1	0/1	1																			
1	0	0	1																			
1	0/1	1	0																			
D FF	<table border="1"> <thead> <tr> <th>$Q(t)$</th><th>D</th><th>$Q(t+1)$</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> </tbody> </table>	$Q(t)$	D	$Q(t+1)$	0	0	0	0	1	1	1	1	1	1	0	0						
$Q(t)$	D	$Q(t+1)$																				
0	0	0																				
0	1	1																				
1	1	1																				
1	0	0																				
T FF	<table border="1"> <thead> <tr> <th>$Q(t)$</th><th>T</th><th>$Q(t+1)$</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> </tbody> </table>	$Q(t)$	T	$Q(t+1)$	0	0	0	0	1	1	1	1	0	1	0	1						
$Q(t)$	T	$Q(t+1)$																				
0	0	0																				
0	1	1																				
1	1	0																				
1	0	1																				

Analysis Procedure

(17)

- * Determine the s/m Variables: I/p variables, state variables & O/p variables.
- * Assign names to the variables if they are not clear from the logic diagram
- * Identify the type of FF used and write their characteristic eqn.
- * Derive state eqns for next states for each state variables
- * Identify the model of circuit whether the ckt is a Mealy or a Moore and derive the o/p eqn
- * Construct a transition table.
- * Assign symbols to the states and construct a state table or state diagram.
- * Develop timing diagram (optional)

Example) A simple sequential ckt using



Step(1)! Characteristic eqn of TFF

$$Q(t+1) = T \oplus Q = \bar{T}Q + T\bar{Q}$$

Step(2)! Excitation eqns (FF i/p eqn)

$$T_A = Q_A + Q_B$$

$$T_B = \bar{Q}_A + Q_B$$

Step(3)! Develop state eqns with the help of excitation eqn.

$$Q_A(t+1) = T_A \oplus Q_A$$

$$= \bar{T}_A Q_A + T_A \bar{Q}_A$$

$$= (\bar{Q}_A + Q_B) Q_A + (Q_A + Q_B) \bar{Q}_A$$

$$\boxed{Q_A(t+1) = \bar{Q}_A Q_B}$$

$$Q_B(t+1) = T_B \oplus Q_B$$

$$= \bar{T}_B Q_B + T_B \bar{Q}_B$$

$$= (\bar{Q}_A + Q_B) Q_B + (Q_A + Q_B) \bar{Q}_B$$

$$\boxed{Q_B(t+1) = \bar{Q}_A \bar{Q}_B}$$

The state eqns,

$$\boxed{Q_A(t+1) = \bar{Q}_A Q_B}$$

$$\boxed{Q_B(t+1) = \bar{Q}_A \bar{Q}_B}$$

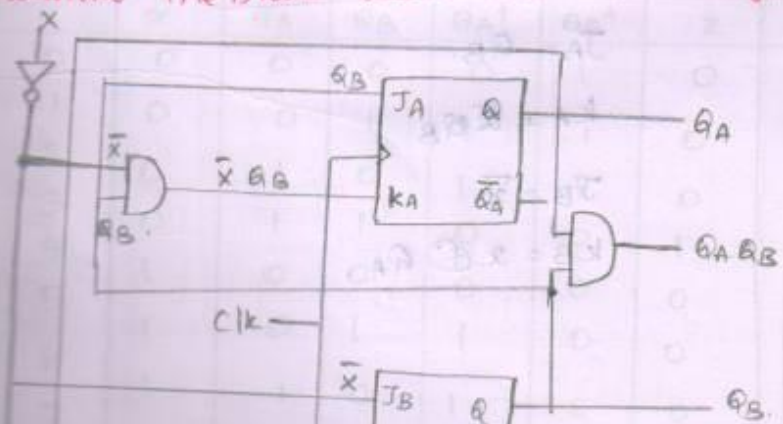
STEP(4):- state table according to state eqn (1) and state diagram.

Present state		Next state	
Q_A	Q_B	$Q_A(t+1)$	$Q_B(t+1)$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	0

State diagram:



Example 8) Consider the seq ckt with 2 J-K FF A and B and one i/p x with and one o/p z. Determine the state table and state diagram.



Solution:-
(1) List all the required data from diagram

Input Variable - x

O/p Variable - z

State variable - Q_A & Q_B

Type of FF - JK

Circuit model - Moore since.

(2) Characteristic eqn of JK FF:-

$$Q(t+1) = J\bar{Q}(t) + \bar{K}Q(t)$$

According to gn logic diagram

$$Q_A(t+1) = J_A \bar{Q}_A(t) + \bar{K}_A Q_A(t)$$

$$Q_B(t+1) = J_B \bar{Q}_B(t) + \bar{K}_B Q_B(t)$$

The simplified characteristic eqn of JK FF

$$Q_A^+ = J_A \bar{Q}_A + \bar{K}_A Q_A$$

$$Q_B^+ = J_B \bar{Q}_B + \bar{K}_B Q_B$$

(3) Excitation eqns:-

$$J_A = Q_B$$

$$K_A = \bar{Q}_B$$

$$J_B = \bar{x}$$

$$K_B = x \oplus Q_A$$

(4) state eqns and o/p eqn:

WRT, $Q_A^+ = J_A \bar{Q}_A + \bar{K}_A Q_A$

$Q_B^+ = J_B \bar{Q}_B + \bar{K}_B Q_B$

Substituting the value of J_A, K_A, J_B and K_B from i/p eqns to the above eqn,

$$\begin{aligned} Q_A^+ &= Q_B \bar{Q}_A + \bar{x} \bar{Q}_B Q_A \\ &= Q_B \bar{Q}_A + (x + \bar{Q}_B) Q_A \end{aligned}$$

$Q_A^+ = Q_B \bar{Q}_A + x Q_A + Q_A \bar{Q}_B$

$$\begin{aligned} Q_B^+ &= \bar{x} \bar{Q}_B + (\bar{x} \oplus Q_A) Q_B \\ &= \bar{x} \bar{Q}_B + (\bar{x} \bar{Q}_A + x Q_A) Q_B \end{aligned}$$

$Q_B^+ = \bar{x} \bar{Q}_B + \bar{x} \bar{Q}_A Q_B + x Q_A Q_B$

$Z = Q_A Q_B$

(5) State Table:-

Decimal	Input	Present state		Next state		Output
	x	Q_A	Q_B	Q_A^+	Q_B^+	Z
0	0	0	0	0	1	0
1	0	0	1	1	1	0
2	0	1	0	1	1	0
3	0	1	1	0	0	1
4	1	0	0	0	0	0
5	1	0	1	1	0	0
6	1	1	0	1	0	0
7	1	1	1	0	0	1

K-map simplification:-

Q_A^+ $Q_A Q_B$

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

$Q_A^+ = \Sigma(1, 2, 5, 6, 7)$

Q_B^+ $Q_A Q_B$

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

$Q_B^+ = \Sigma(0, 1, 2, 7)$

Z $Q_A Q_B$

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

$Z = \Sigma(3, 7)$

Modified state table:-

Present state		Next state and o/p		
Q_A	Q_B	$Q_A^+ \quad x=0 \quad z$	$Q_B^+ \quad x=0 \quad z$	$Q_A^+ \quad x=1 \quad z$
0	0	0	1	0
0	1	1	1	0
1	0	1	1	0
1	1	0	0	1

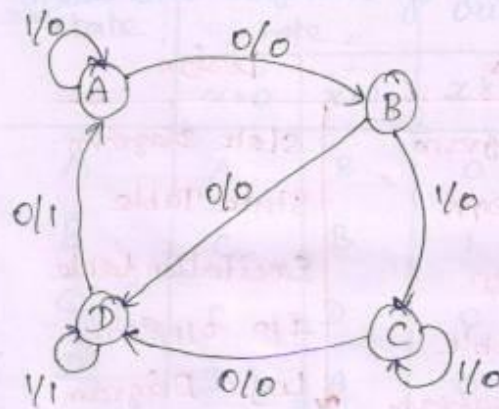
(6) Assign the state variable:-

(25)

Let $00 \rightarrow A$, $01 \rightarrow B$, $10 \rightarrow C$ and $11 \rightarrow D$

Present state	Next state		output	
	$x=0$	$x=1$	$x=0$	$x=1$
A	B	A	0	0
B	D	C	0	0
C	D	C	0	0
D	A	D	1	1

(7) State Diagram:-



Design Procedure

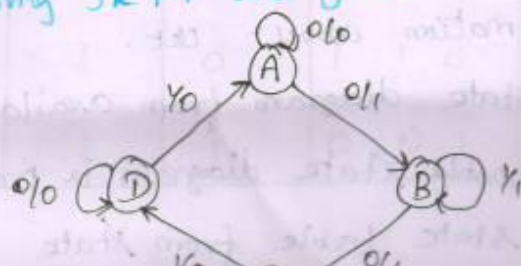
- * Read circuit description carefully and list all information about ckt.
- * Develop state diagram from available information. Mostly state diagram is provided.
- * Develop state table from state

- * Reduce the no of state if any, using state reduction techniques.
- * Assign binary values of each state if the state table contains letter symbols.
- * Select the type of FF to be used
- * Determine the no. of FF $\rightarrow n$ FF for 2^n states
- * Develop excitation table of selected FF.
- * Derive excitation eqn and o/p eqn.
- * Draw the logic diagram.

The summary of analysis and design.

Analysis	Design
Logic diagram	State Diagram
I/P, O/P eqn	State Table
State Eqn	Excitation table
State table	I/P, O/P Eqn
State diagram	Logic Diagram

(Example) Design a clock sequential ckt where state diagram is given in fig. using JK FF and gates



solution: In this ~~the~~ example, state diagram⁽²⁷⁾ is ~~gn~~, therefore it is ~~is~~ easy to obtain the final logic diagram. Also the FF should be JK FF. There are four states in this example, hence 2 FF are needed to realize the ckt along with no. of gates.

(1) State table:

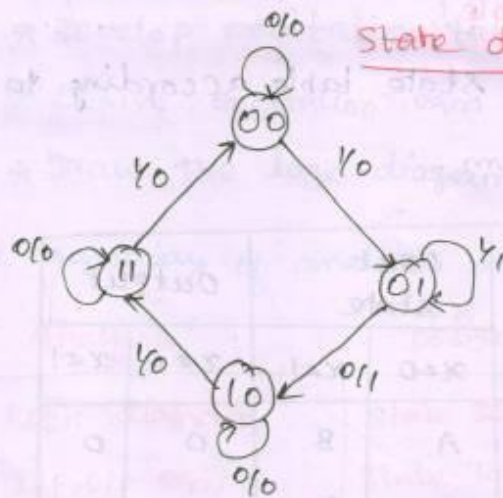
First draw state table according to the state diagram.

Present state	Next state		Output	
	$x=0$	$x=1$	$x=0$	$x=1$
A	A	B	0	0
B	C	B	1	1
C	C	D	0	0
D	D	A	0	0

(2) State Reduction and State Assignment:

Based on the assigned value of states, once again the state diagrams along with state assignment is shown below. This state diagram, in fact not needed, it is shown here only for the sake of illustration.

State Assignment	
states	Binary Value
A	0 0
B	0 1
C	1 0
D	1 1



(3) New table or Transition table

This table can be obtained directly from the state diagram.

Present state		Next state and output					
		x = 0			x = 1		
Q _A	Q _B	Q _A ⁺	Q _B ⁺	z	Q _A ⁺	Q _B ⁺	z
0	0	0	0	0	0	1	0
0	1	1	0	1	0	1	1
1	0						
1	1						

4) Modified state table

29

Input x	Present state		Next state		Output z
	Q_A	Q_B	Q_A^+	Q_B^+	
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	1	1	0
1	1	1	0	0	0

5) Excitation table

* when present state is 0, k is don't care

* when present state is 1, j is don't care

* If 'no change' in next state, respective j or k is 0 otherwise 1.

directly

	Flip-Flop A				Flip-Flop B.				Output
Input	JS	KS	Flip-Flop J/P		present state	Next state	FF J/P		
	QA	QA ⁺	JA	KA	QB	QB ⁺	JB	KB	
Z	QA	QA ⁺	JA	KA	QB	QB ⁺	JB	KB	Z
0	0	0	0	X	0	0	0	X	0
0	0	1	1	X	1	0	X	1	1
0	1	0	X	0	0	0	0	X	0
0	1	1	X	0	1	1	X	0	0
1	0	0	0	X	0	1	1	X	0
1	0	0	0	X	1	1	X	0	1

(b) Flip-Flop I/P eqns and Ckt o/p eqns:

J_A

$\bar{A} \bar{B}$	00	01	11	10
0	0	1	x	x
1	x	x	x	x

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

K_A

$\bar{A} \bar{B}$	00	01	11	10
0	x	x	3	2
1	x	x	1	6

$$K_A = x \bar{B}$$

J_B

$\bar{A} \bar{B}$	00	01	11	10
0	0	x	x	2
1	1	x	x	1

$$J_B = x$$

K_B

$\bar{A} \bar{B}$	00	01	11	10
0	x	1	3	x
1	x	5	1	x

$$K_B = \bar{x} \bar{A} + x \bar{A}$$

Z

$\bar{A} \bar{B}$	00	01	11	10
0	0	1	3	2
1	4	1	7	6

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

(c) Input - Output Equation:-

$J_A = \bar{x} \bar{B}$	$K_A = x \bar{B}$	} FF I/P Eqn.
$J_B = x$	$K_B = \bar{x} \oplus \bar{A}$	

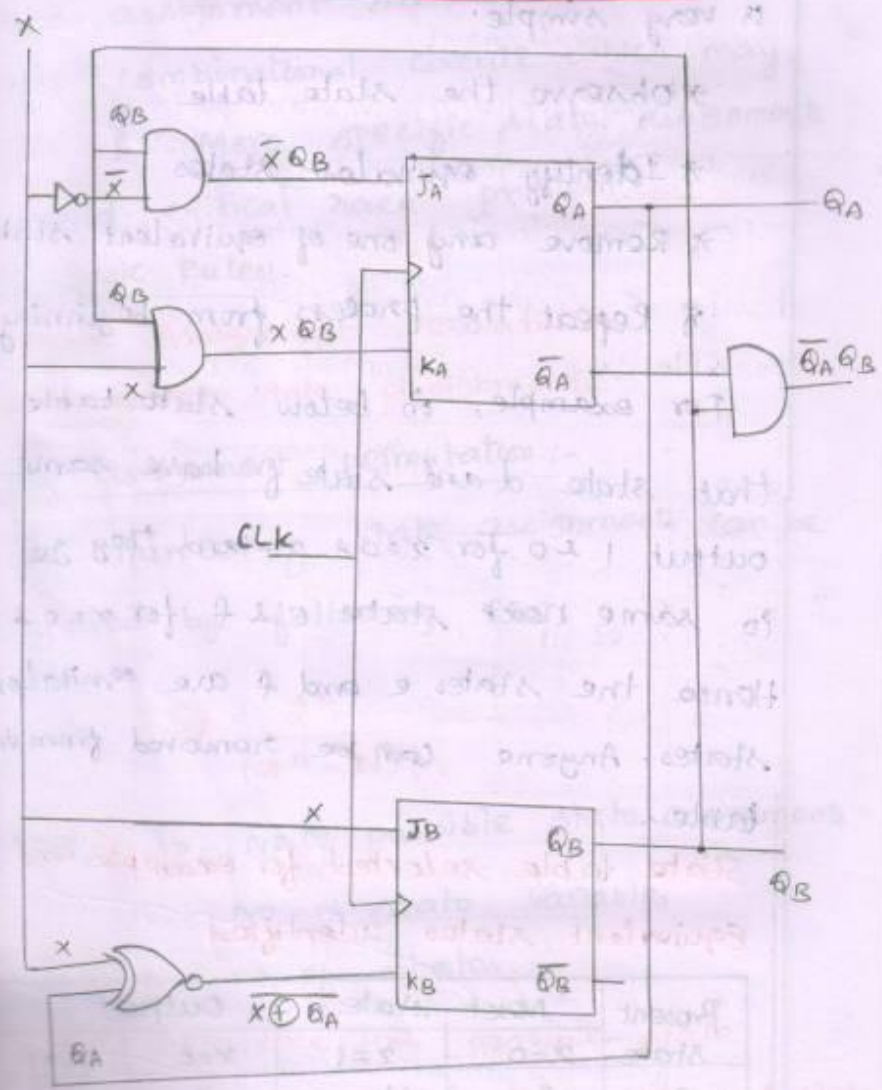
ip eqm.

(8) Logic Diagram:-

(31)

Finally logic diagram is drawn according to i/p-o/p fns obtained using k-map.

Logic diagram of given design



F I/p Eqm.

State Reduction

It is a way of reducing the no. of states in tcom, reducing the no. of FF and no. of gates in seq. ckt. This technique is very simple:

- * Observe the state table
- * Identify equivalent states
- * Remove any one of equivalent state
- * Repeat the process from beginning

For example, in below state table that state d and state f have same output 1 & 0 for $x=0$ & $x=1$ and they go to same next state e & f for $x=0$ & $x=1$. Hence the states e and f are equivalent states. Anyone can be removed from the table.

State table selected for example and
Equivalent states identified

Present state	Next state		Output	
	$x=0$	$x=1$	$x=0$	$x=1$
a	c	b	0	0
b	d	a	0	1
c	g	d	1	1
d	e	f d	1	0
e	f d	a	0	1

State Assignment

(33)

Representation of state variables with unique binary codes is termed as state Assignment. The most common criterion is that the chosen assignment should result in a simple combinational circuit which may excite FF. More specific state assignments to avoid critical race problem.

Two Basic Rules:-

* State Assignment permutation

* Equivalent state assignments.

State assignment permutation :-

The number of state assignments can be calculated by following eqn,

$$S_p = \frac{2^n}{(2^n - N)!}$$

where, S_p - no of possible state assignments

n - no. of state variables

N - no. of states.

Example Determine the possible state assignments of sequential ckt with (i) 4 state and 2 state variables (ii) five state and three state variables.

$$(i) S_p = \frac{(2^2)!}{4!} = \frac{4!}{4!} = 1$$

Equivalent state Assignment:

Two state assignments are said to be equivalent, if they contribute same combinational circuit to drive the FF.

* when one assignment is complement of another assignment

* when columns of two sets of state assignments are interchanged.

For example, let a state machine is having 4 states and 2 state variables. The 4 possible state assignments are shown below.

States	State Assignments			
	Assignment 1	Assignment 2	Assignment 3	Assignment 4
	b ₁ b ₀	b ₁ b ₀	b ₁ b ₀	b ₁ b ₀
A	0 0	1 1	0 0	0 0
B	0 1	1 0	1 0	0 1
C	1 0	0 1	0 1	1 1
D	1 1	0 0	1 1	1 0

↑ ↑ ↑
Complement each other
column Interchanged.

The non-equivalent state assignments should be considered when assigning the binary values to the states. The no. of nonequivalent state assignments can be found from,

$$S_{NE} = (2^n - 1)!$$

Example Determine the non equivalent (5)

state assignments of sequential machine with

(i) 4 state and 2 state variable

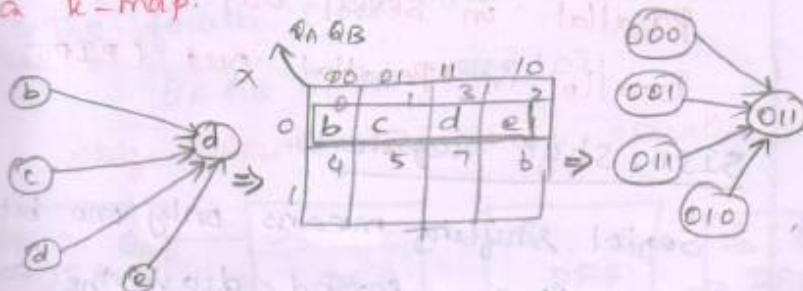
(ii) 5 state and 3 state variable.

$$(i) S_{NE} = \frac{(2^2 - 1)!}{(2^2 - 4)!(2)!} = 3.$$

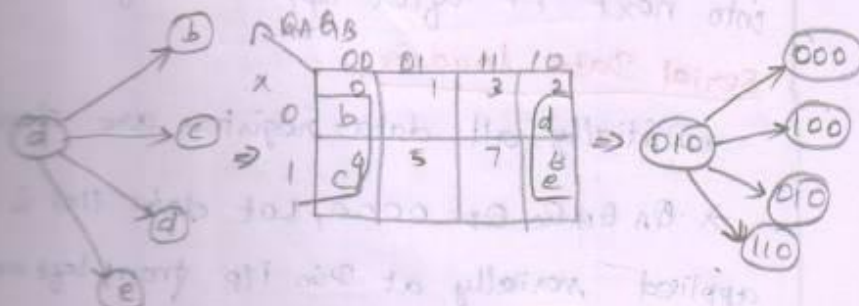
$$(ii) S_{NE} = \frac{(2^3 - 1)!}{(2^3 - 5)!(3)!} = 140.$$

Basic Rules of State Assignment

Rule 1:- states having the next state for a given i/p condition should be made adjacent in a k-map.



Rule 2:- Two or more next states having same present state can be made adjacent below.



Shift Registers

It is a register that moves the binary information from one FF to its neighbouring FF upon the occurrence of a clock pulse. is called Shift Register. Based on the manner in which data is entered into the register and come out from the register, the registers are broadly categorized into 4 types.

Serial In serial out (SISO)

Serial in parallel out (SIPO)

Parallel in serial out (PISO)

Parallel in parallel out (PIPO)

SISO Shift Register:-

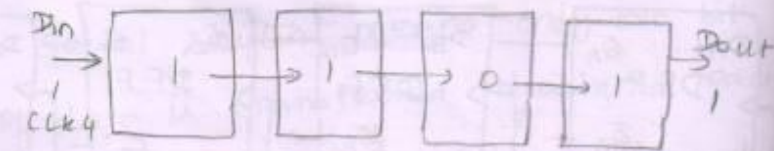
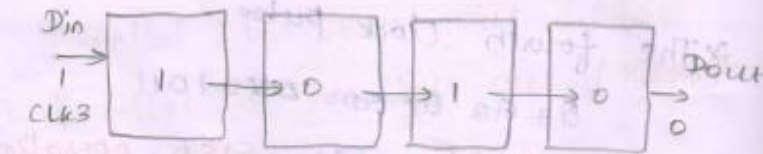
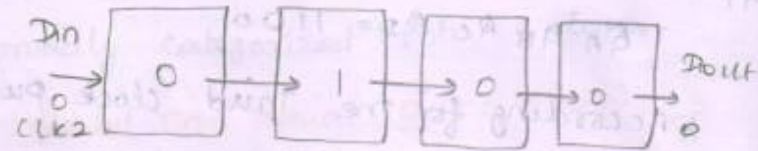
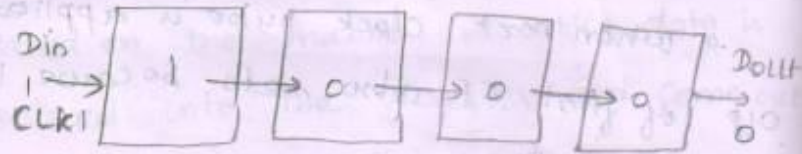
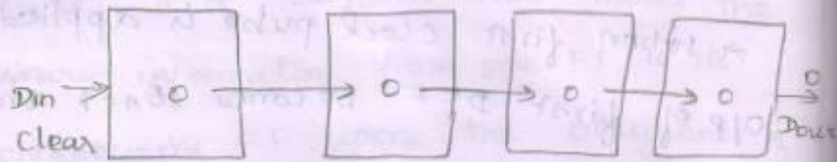
Serial shifting means only one bit of data is transferred during the each clock pulse. In SISO shift register data enters into the left most FF and shifts into next FF after application of clk pulse.

Serial Data Loading:-

*Initially all data registers are cleared.

*QA QB QC QD = 0000, Let data 1101 is applied serially at Din i/p from left most

Serial Data loading



serial loading example

DATA	1 0 1 1			
clk pulse	Din	QA	QB	QC
CLEAR	X	0	0	0
CLK1	1	1	0	0
CLK2	0	0	1	0
CLK3	1	1	0	1
CLK4	1	1	1	0

Serial Data out:-

(39)

After 4 clock pulse the data (1011) is completely stored in the register. To get the data out from the register, the bits must be shifted right serially and taken off from the output terminal Dout.

* After fourth clock pulse the left-most bit, 1 is available on Dout.

* When fifth clock pulse is applied, the second MSB appears on Dout.

* Sixth clock pulse shifts next bit to the output.

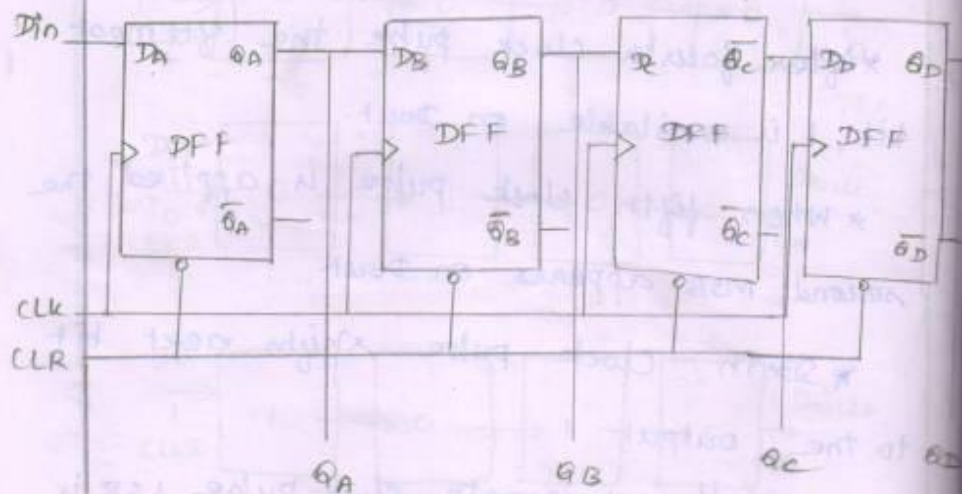
* Finally at seventh clock pulse LSB is taken off from register.

Data at 4th clock pulse	1011				
	Din	A _A	A _B	A _C	A _D
clock pulse	Din	A _A	A _B	A _C	A _D
CLK 4	0	1	1	0	1
CLK 5	0	0	1	1	0
CLK 6	0	0	0	1	1
CLK 7	0	0	0	0	1

Serial In Parallel out (SIPO) Shift Register:-

Data bits are entered serially into the register similar to SISO. After loading all bits, data can be taken out simultaneously at fourth clock pulse itself.

SIPO Shift Register

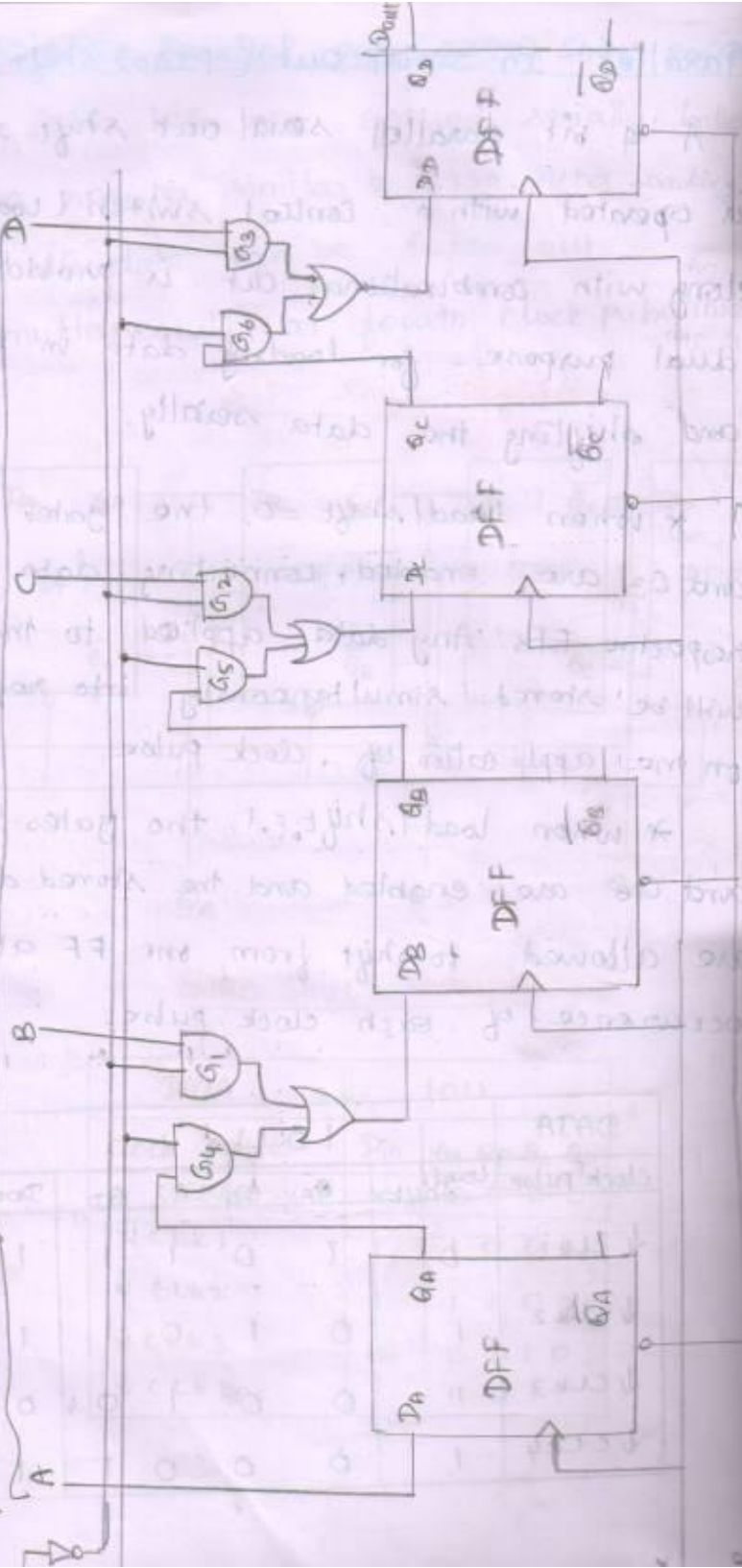


SIPO Shift example

DATA	1011
Clock pulse	Din QA QB QC QD
CLEAR	X 0 0 0 0
↓ CLK1	1 1 0 0 0
↓ CLK2	1 1 1 0 0
↓ CLK3	0 0 1 1 0
↓ CLK4	1 1 0 1 1

PIPO Shift Register

Shift Co Load 1 Shift



(43)

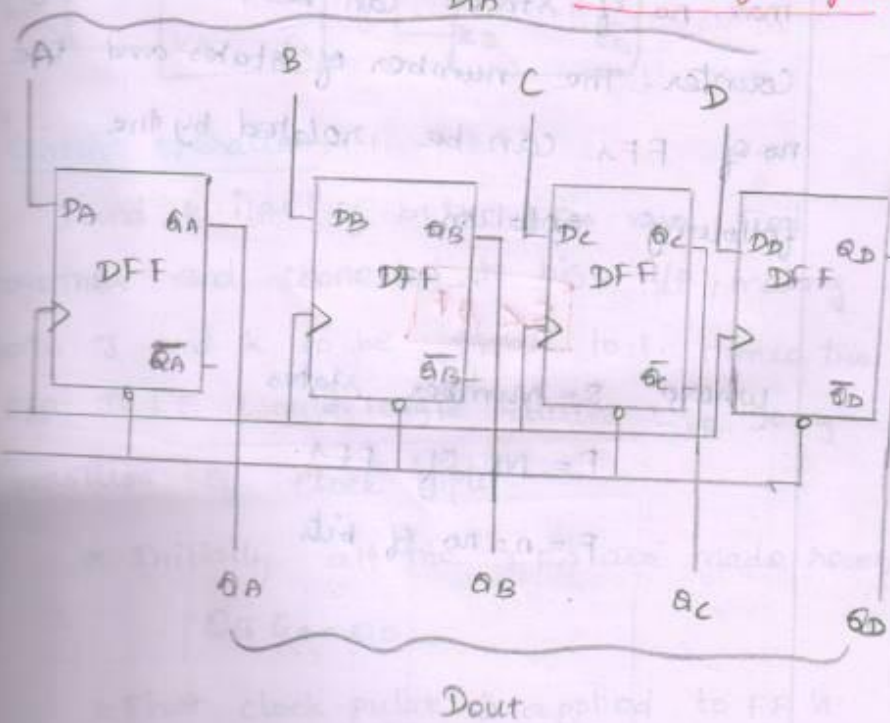
Parallel In Parallel Out (PIPO) Shift Register

Arrangement of FFs in this register is such that it takes only one clock pulse to enter the data into register. Immediately after the entry of data bits, are available on the parallel O/P as shown.

PIPO Shift example

DATA	1011			
CLK pulse	Din		Dout	
	A	B	C	D
	A ₀ A ₁ A ₂ A ₃		A ₀ A ₁ A ₂ A ₃	
↓ CLK 1	1	0	1	1
	1011		1011	

PIPO Shift Register



Counters

A group of complementary FFs connected in a specified manner to perform counting operations is called counters. A counter actually counts no. of clock pulses and exhibits equal no. of states at its o/p terminal. For example, a 2-bit counter counts four clock pulses and produces four ($2^2 = 4$) states. At fourth clock pulse it returns to initial state. The no. of FFs used in a counter depends on the max. no. of states can be produced by the counter. The number of states and the no. of FFs can be related by the following relation,

$$S < 2^F$$

where, S = Number states

F = No. of FFs.

$F = n$ = no. of bits.

of FF 'B' also. Since the FF responds only to a -ve going transition of their clk. i/p, the state of FF 'B' does not change.

$$Q_B Q_A = 01$$

* During the second clk pulse, the o/p of the first J-K FF becomes 0. Now the clock pulse is a -ve going pulse. The second FF, responds to this -ve going pulse and changes its state from 0 to 1.

$$Q_B Q_A = 10$$

* When ~~third~~ third clk pulse is applied as usual FF A toggles its state while the state of FF B remains same.

$$Q_B Q_A = 11$$

* At fourth clock pulse, the o/p of both the FF becomes 0.

$$Q_B Q_A = 00.$$

State Sequence Table

clock pulse	$Q_B Q_A$
Initially	00
CLK1	01
CLK2	10
CLK3	11

UP/DOWN Counter:-

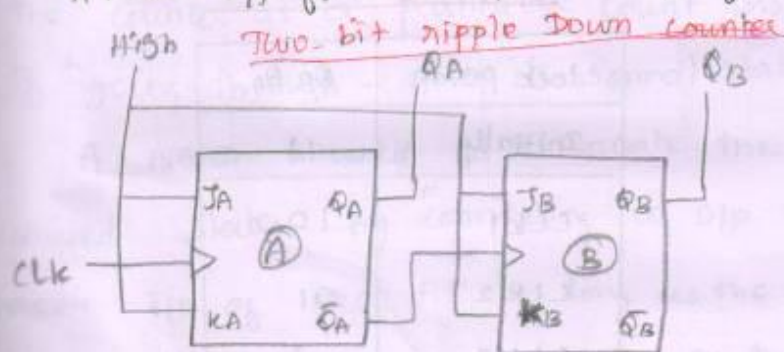
1st method:-

* clock pulse is applied at clock i/p of first FF

* Remaining FFs are triggered by complementary o/p of previous FF.

2nd method:-

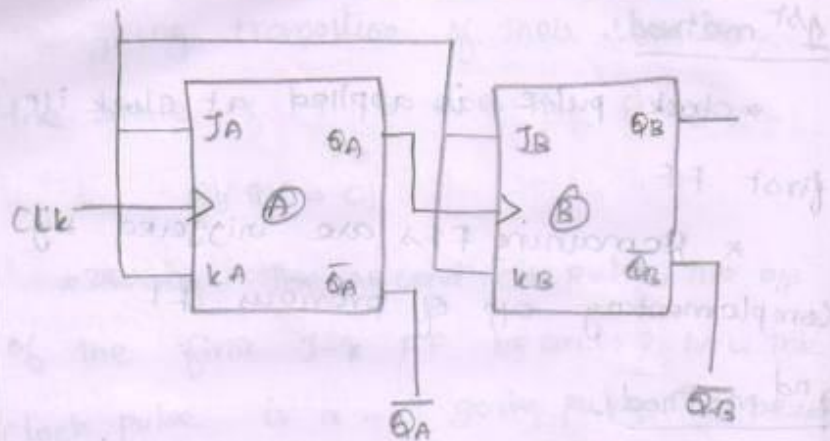
* Take o/p from complementary of FF.



State sequence of Down counter

clock pulse	QB QA
Initially	0 0
CLK 1	1 1
CLK 2	1 0
CLK 3	0 1
CLK 4	0 0
	Initial state

High. 2-bit ripple down counter



State Sequence Table

Clock pulse	$\overline{Q_B} \overline{Q_A}$
Initially	11
CLK 1	10
CLK 2	01
CLK 3	00
CLK 4	11 Initial state

Modulus of counter

For example, a 3-bit counter can be modified to count only 5 states by truncating remaining 3 states. that can be given a name as mod-5 counter. Thus the modulus of counter can be defined as the desired no. of states

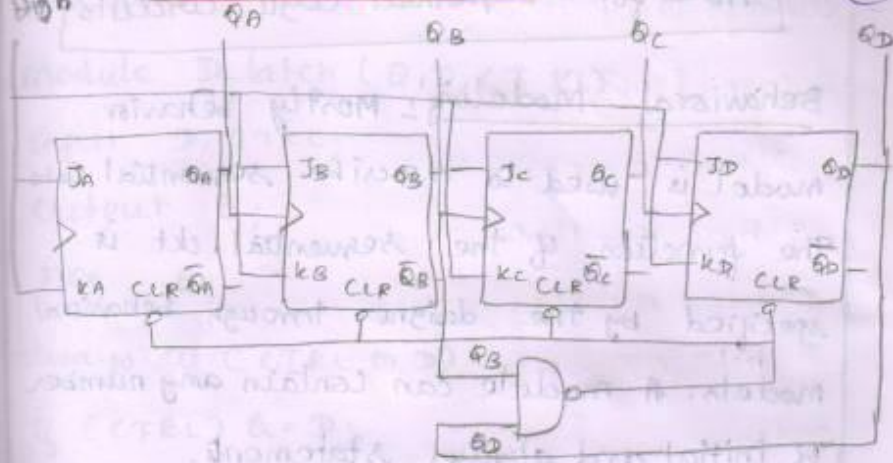
The desired no. of states may be in binary sequence or could be unique states.

Mod-5 Counter: It is actually a 3-bit ripple counter where last 3 states are forcibly truncated. The truncation in the count sequence is obtained by resetting the counter at a particular count instead of going through all of its normal states.

A NAND is used to truncate the unused states by connecting its o/p to the RESET i/p of each FF. As long as the NAND o/p is HIGH, there is no effect on the counter. When it goes low, it resets all the FFs so that the counter immediately goes to the 000 state.

The o/p's of the FF A and FF C are connected to the i/p's of NAND gate so that the NAND o/p will go low whenever $E_A \cdot E_C = 1$. This condition occurs when the counter goes from the 100 state to the 001 state. The low at the NAND o/p resets the counter to the 000 state.

Mod-10 Ripple Counter



State Table of mod-5 counter

clock pulse	QC	QB	QA
Initially	0	0	0
CLK1	0	0	1
CLK2	0	1	0
CLK3	0	1	1
CLK4	1	0	0
CLK5	1	0	1

HDL for Sequential Logic Circuits

Behavioral Modeling: Mostly behavior model is used to describe sequential ckt. The function of the sequential ckt is specified by the designer through behavioral models. A module can contain any number of initial and always statements.

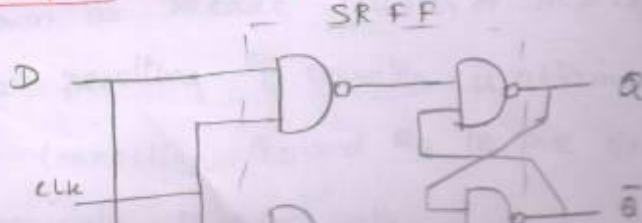
Example 1:- 2-bit up-down counter

```
initial
begin
    clock = 1'b0;
end,
```

Example 2:-

```
initial
begin
    clock = 1'b0;
    repeat (30)
        #10 clock = ~clock;
    end.
```

Example 3:- Describe 2-latch and 2-FF.



// D Latch

module D-latch (Q, D, CTRL);
input D, CTRL;
output Q;
reg Q;
always @ (CTRL or D)
if (CTRL) Q = D;
endmodule.

// D Flip Flop

module D-FF (Q, D, CLK);
output Q;
input D, CLK;
reg Q;
always @ (posedge CLK)
Q = D;
endmodule.

Example 4:- Describe JK flip flop using
Characteristic table rather than Char. eqn.

module JK-FF (J, K, CLK, Q, Qnot);
input J, K, CLK;
output Q, Qnot;
reg Q;
always @ (posedge CLK)
Q = (J & ~Q) | (~K & Q);
Qnot = ~Q;

always @ (posedge clk)

case ({j,k})

2'b00: a = a;

2'b01: a = 1'b0;

2'b10: a = 1'b1;

2'b11: a = ~a;

endcase

endmodule

P. Balakrishna
23/11/21

initial

begin

clock = 1'b0;

end

always @ (posedge clk)

initial

begin

clock = 1'b0;

repeat (10)

begin

clock = ~clock;

end

end



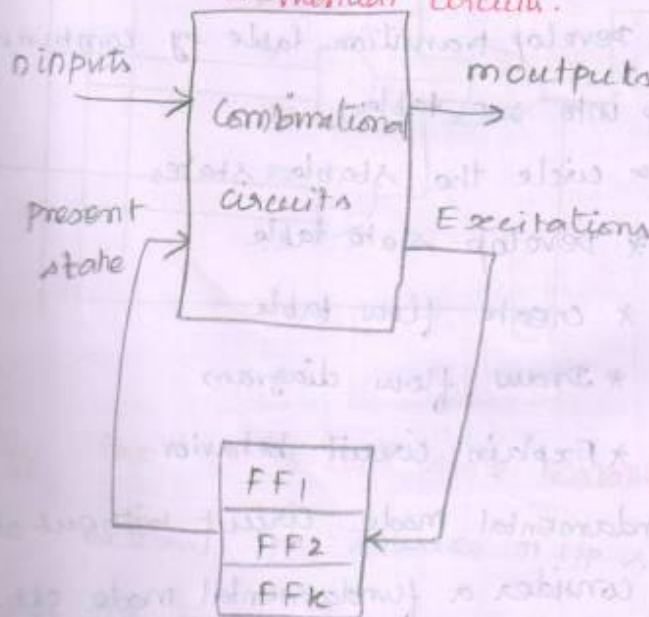
UNIT-IV Asynchronous sequential circuits (51)

Analysis and design of Asynchronous sequential circuits - Reduction of state and Flow tables - Race-free state Assignment - Hazards.

Asynchronous sequential circuit:

It is a circuit whose behaviour is independent of clock pulse but controlled by its input signal. The speed of operation of this circuit is faster than the synchronous circuit.

Block diagram of Asynchronous sequential circuit.



Analysis of Asynchronous sequential circuit.

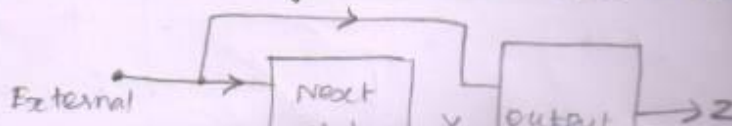
It is the process of obtaining the table or a diagram for a given logic diagram that describes the sequence of internal states and o/p's as a function of changes in the i/p variables.

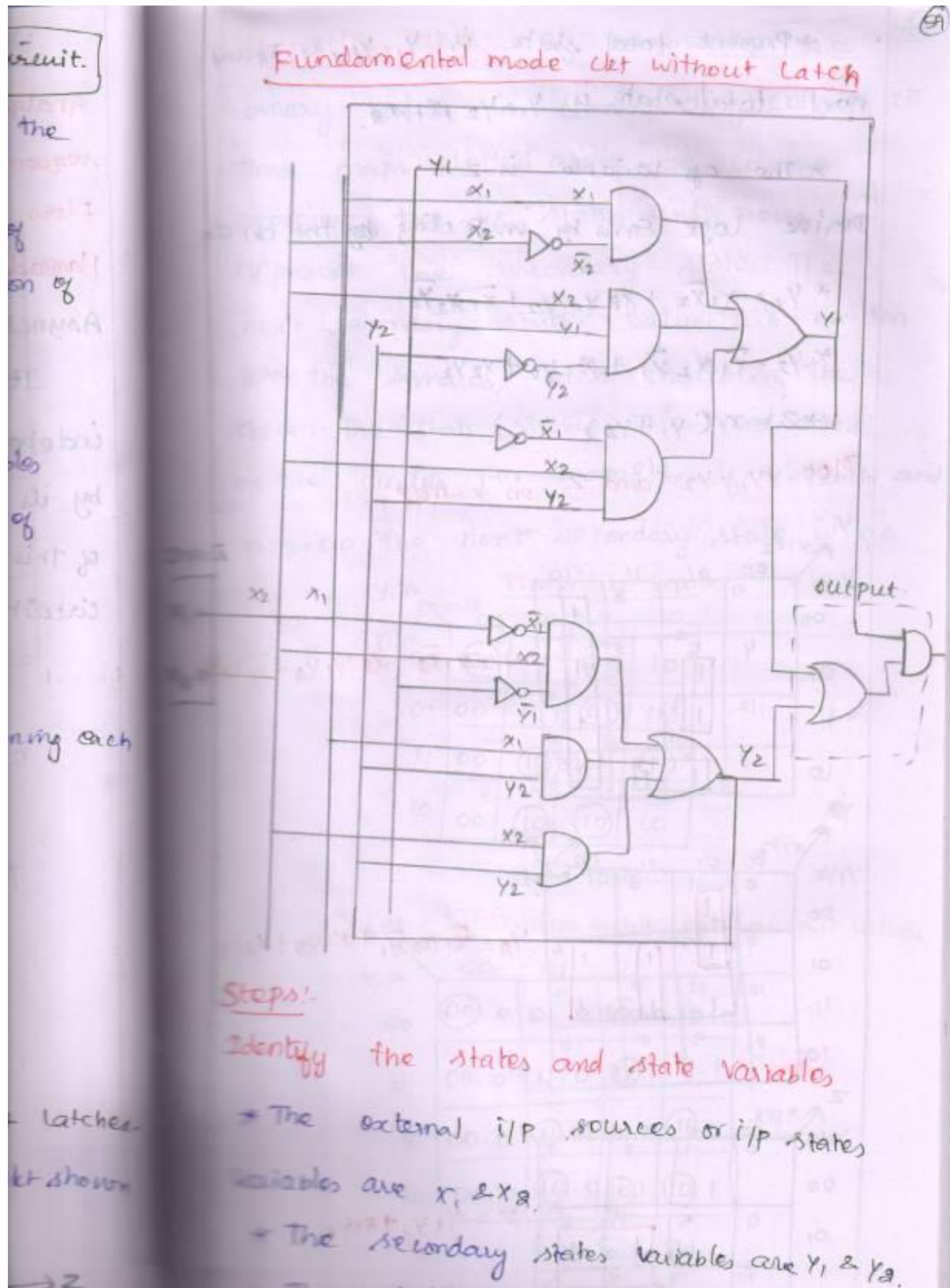
Steps involved:-

- * Identify states and state variables
- * check whether $SR=0$ in the case of circuits with latches
- * Derive logic equations
- * Plot each eqn in k-map
- * Develop transition table by combining k-map into one table
- * circle the stable states
- * Develop state table
- * Create flow table
- * Draw flow diagram
- * Explain circuit behavior.

Fundamental mode circuit without latches

consider a fundamental mode ckt shown





* Present total state is Y_1, Y_2, X_1, X_2 and
next total state is Y_1, Y_2, X_1, X_2 .

* The op variable is Z .

Derive Logic Eqs by inspecting the dtdn.

$$* Y_1 = X_1 \bar{X}_2 + X_2 Y_1 + \bar{X}_1 X_2 \bar{Y}_2$$

$$* Y_2 = \bar{X}_1 X_2 \bar{Y}_1 + X_1 Y_2 + X_2 Y_2$$

$$* Z = X_1 (Y_1 + Y_2)$$

Plot Y_1, Y_2 and Z in k-map

Y_1

$X_1 X_2$	00	01	11	10
$Y_1 Y_2$	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$$Y_1 = X_1 \bar{X}_2 + X_2 Y_1 + \bar{X}_1 X_2 \bar{Y}_2$$

Y_2

$X_1 X_2$	00	01	11	10
$Y_1 Y_2$	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$$Y_2 = \bar{X}_1 X_2 \bar{Y}_1 + X_1 Y_2 + X_2 Y_2$$

Z

$X_1 X_2$	00	01	11	10
$Y_1 Y_2$	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$$Z = X_1 Y_1 + X_1 Y_2$$

Transition Table:- By combining the binary values in corresponding cells of all three maps plotted above. The columns represent the $Y_1 Y_2$ states and rows represent the secondary states. The next-secondary states values are written into the squares, each indicating the state. The state (stable) are indicated by the circles. For example, if $Y_1 Y_2 = 11$ and $X_1 X_2 = 00$, the next secondary state is 00

Transition table

	$X_1 X_2$	00	01	11	10
$Y_1 Y_2$	00	00 ⁰	01 ¹	00 ³	10 ²
	01	00 ⁴	11 ⁵	01 ⁷	11 ⁶
	11	00 ¹²	11 ¹³	01 ¹⁵	11 ¹⁴
	10	00 ⁸	10 ⁹	10 ¹¹	10 ¹⁰

State Table.

Transition table with output values

	$X_1 X_2$	00	01	11	10
$Y_1 Y_2$	00	00.0	01.0	00.0	10.0
	01	00.0	11.0	01.1	11.1
	11	00.0	11.0	01.1	11.1
	10	00.0	10.0	10.1	10.1

State table

State Table:

Similar to sequential synchronous circuit, the state table for asynchronous ckt can be formed with the help of logic eqns. Since the state table contains the same information as transition table, it may not be the part of the analysis of asynchronous ckt. In state table the secondary variables and excitation variables are named as present states and next states respectively.

Present state				Next state				Stable state	Output Z
Y_1	Y_2	X_1	X_2	Y_1	Y_2	X_1	X_2		
0	0	0	0	0	0	0	0	✓	0
0	0	0	1	0	1	0	1	X	0
0	0	1	0	1	0	1	0	X	0
0	0	1	1	0	0	1	1	✓	0
0	1	0	0	0	0	0	0	X	0
0	1	0	1	1	1	0	1	X	0
0	1	1	0	1	1	1	0	X	1
0	1	1	1	0	1	1	1	✓	1
1	0	0	0	0	0	0	0	X	0
1	0	0	1	1	0	0	1	✓	0
1	0	1	0	1	0	1	0	✓	1
1	0	1	1	1	0	1	1	✓	1
1	1	0	0	0	0	0	0	X	0
1	1	0	0	0	0	0	0	✓	0

It is worth noting ~~the~~ in the case of (b3) asynchronous ckt that, at least one next state value is same as present state in each row.

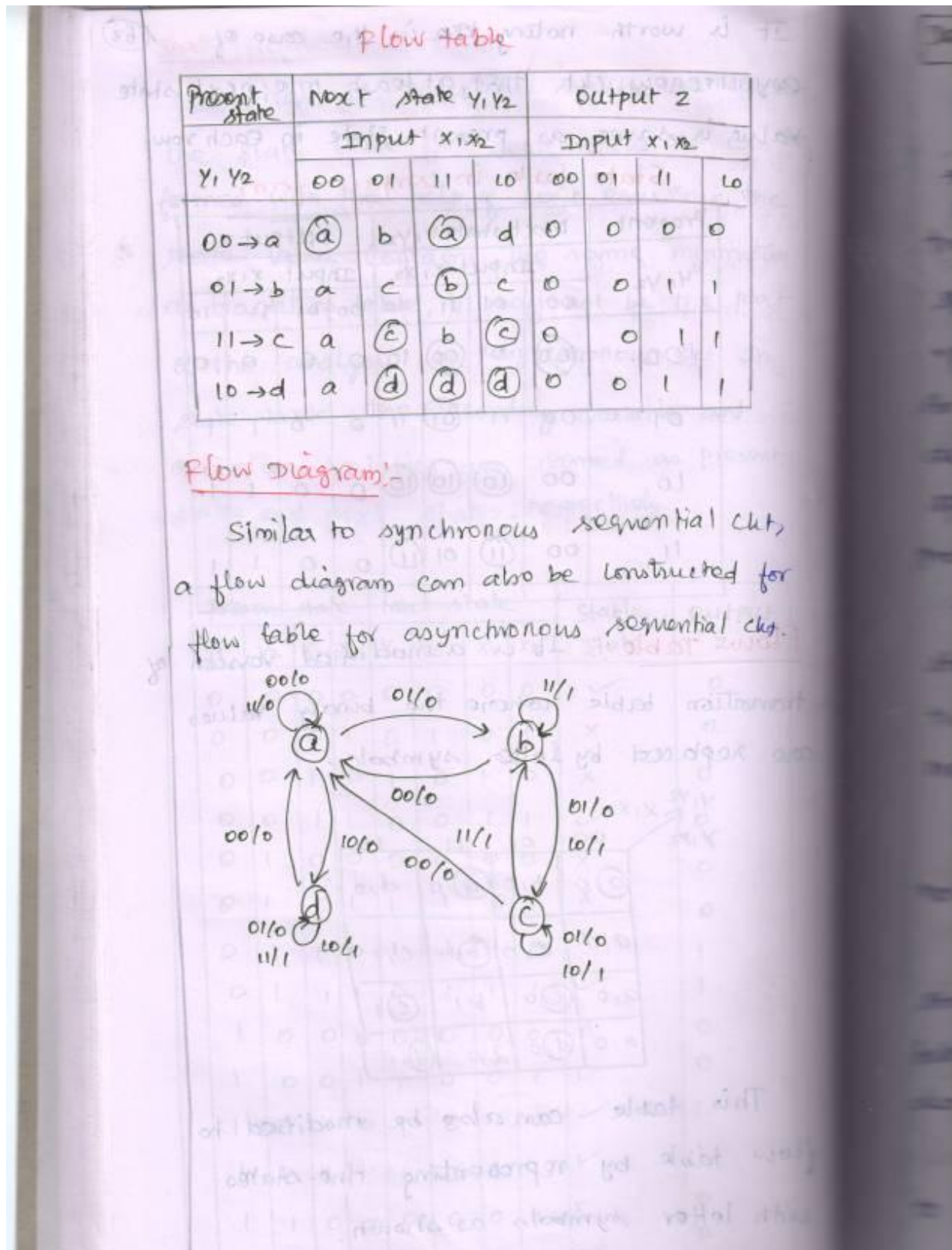
State table in compact form

Present y_1, y_2	Next state y_1, y_2				Output z			
	Input x_1, x_2				Input x_1, x_2			
	00	01	11	10	00	01	11	10
00	00	01	00	10	0	0	0	0
01	00	11	01	11	0	0	1	1
10	00	10	10	10	0	0	1	1
11	00	11	01	11	0	0	1	1

Flow Table:- It is a modified version of transition table where the binary values are replaced by letter symbols.

y_1, y_2 x_1, x_2	00	01	11	10
a, 0	b, 0	a, 0	d, 0	
a, 1	a, 0	b, 1	c, 1	
a, 0	c, 0	b, 1	c, 1	
a, 1	d, 0	d, 1	d, 1	

This table can also be modified to flow table by representing the states with letter symbols as shown.



Design of Asynchronous sequential circuits ⁽⁶⁵⁾

It is the reverse process of analysis.

- * Construct a primitive flow table from the given design specifications. The design specification may be in the form of "problem statement" or "state diagram". An intermediate step could include the development of a state diagram if the specification is written in the form of problem statement.

- * Eliminate the redundant states by state reduction techniques. It may include the information of implication table - merger diagram - covering table - closure table.

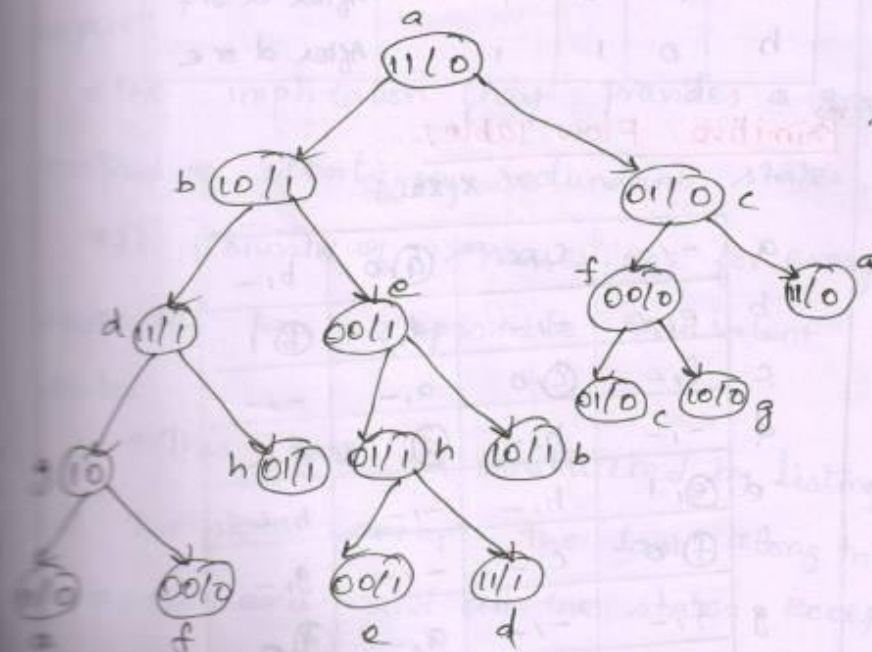
- * Reduce the flow table by merging rows in the primitive flow table.

- * Assign the binary values to each state of the reduced flow table. Care must be taken to eliminate the circle race, if any during the state assignment.

- * Develop o/p map by assigning o/p's to the unstable state, if any.

Solution* Analysis of the problem.

- (1) It has 2 i/p's x_1, x_2 and o/p z
- (2) Both i/p's never change simultaneously
- (3) O/P toggles as long as $x_1 = 1$ and x_2 changes from 1 to 0
- (4) O/P remains unchanged when $x_1 = 0$ irrespective of the status of x_2 .
- (5) When $x_1 = 1$ and there is no change in x_2 or x_2 changes from 0 to 1, the o/p holds the previous status.

* Rough state diagram:-

Function Table:-

With the help of rough state diagram, the function table for the given example is formed and shown in table.

states	Inputs		outputs	Comments
	x_1	x_2	z	
a	1	1	0	Initial condition set
b	1	0	1	After c or g
c	0	1	0	After a or f
d	1	1	1	After b or h
e	0	0	1	After b or h
f	0	0	0	After c or g
g	1	0	1	After d or f
h	0	1	1	After d or e

Primitive Flow Table:- $x_1 x_2$

a	-,-	c,-	(a) 0	b,-
b	e,-	-,-	d,-	(b) 1
c	f,-	(c) 0	a,-	-,-
d	-,-	h,-	(d) 1	g,-
e	(e) 1	h,-	-,-	b,-
f	(f) 0	c,-	-,-	g,-
g	f,-	-,-	a,-	(g) 0
h	e,-	(h) 1	d,-	-,-

State Reduction Techniques

(69)

It can be divided into 2 parts.

(1) Completely specified states

(2) Incompletely " "

State Reduction of completely specified series:-

Implication Table (or implication chart)
and Implied states:-

* The equivalent states can be defined as "Two states are said to be equivalent if they produce same output and go to the same next state for every possible input".

* The implication chart provides a graphic method of identifying redundant states.

* It consists of squares, one for every suspected pair of possible equivalent states.

* This chart is constructed by listing all the states except the last along the horizontal axis and all the states except the first along the vertical axes.

equivalency.

Let the states be a, b, c, d, e and f

Implication Table

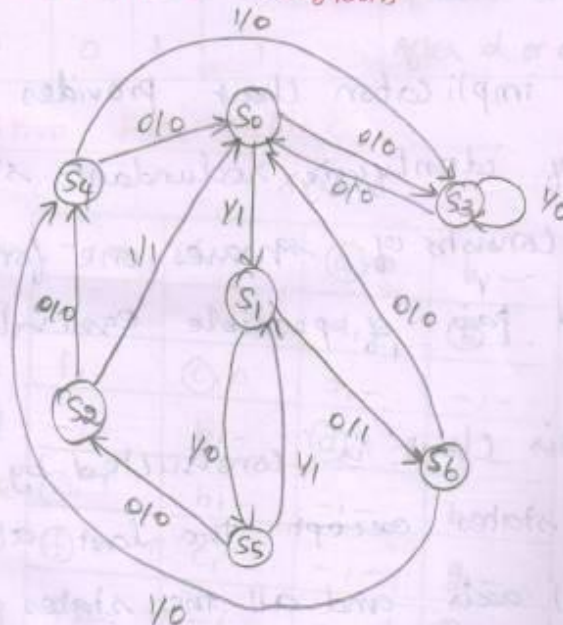
Implies the possibility that $a=b$

b					
c					
d					
e					
f					
	a	b	c	d	e

Except
1st state

except last state

(Example) Determine the redundant states for the state diagram, and obtain reduced state diagram



solution:

(11)

* Derive a state table according to gn state diagram.

Present state	Next state		output	
	x=0	x=1	x=0	x=1
S ₀	S ₃	S ₁	0	1
S ₁	S ₆	S ₅	1	0
S ₂	S ₄	S ₀	0	1
S ₃	S ₀	S ₃	0	0
S ₄	S ₀	S ₃	0	0
S ₅	S ₂	S ₁	0	1
S ₆	S ₀	S ₄	0	0

* Form an implication table and fill the squares of the table according to the method explained in above example

S ₁	x					
S ₂	S ₂ , S ₄	x				
S ₃	x	x	x			
S ₄	x	x	x	✓		
S ₅	S ₂ , S ₃	x	S ₂ , S ₄	x	x	
	x		S ₀ , S ₁			
S ₆	x	x	x	S ₃ , S ₄	S ₅ , S ₆	x
				✓	✓	
	S ₀	S ₁	S ₂	S ₃	S ₄	S ₅

* List all the equivalent states from the implication chart,

$(s_0, s_2) (s_3, s_4) (s_3, s_6) (s_4, s_6)$

* Form a larger group of eqt chart

The state pairs $(s_3, s_4) (s_3, s_6) (s_4, s_6)$ can be merged into a larger group of three eqt states (s_3, s_4, s_6) since each one is eqt to the other two states.

The eqt states are,

$(s_0, s_2) (s_3, s_4, s_6)$

* List all the states together

$(s_0, s_2) (s_1) (s_3, s_4, s_6) (s_5)$

Now the 7 states can be produced to 4 states by replacing s_2 and s_0 and states s_4 and s_6 by s_3 as shown below,

$(s_0, \overset{s_0}{s_2}) (s_1) (s_3, \overset{s_3}{s_4}, \overset{s_3}{s_6}) (s_5)$

* Develop Reduced state table

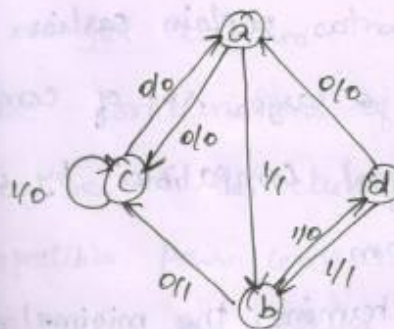
Present state	Next state		output	
	$x=0$	$x=1$	$z=0$	$z=1$
s_0	s_3	s_1	0	1
s_1	s_3	s_5	1	0
s_3	s_0	s_3	0	0

* Redraw the state table by assigning new variables to the states and obtain Reduced State Diagram.

Reduced Flow Table

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

Reduced state diagram



State Reduction of Incompletely specified states

In some examples, some states and ops are not specified in it and are shown by dashes, indicate don't-care conditions. Therefore the concept of eqv states cannot work here but the concept of 'compatible' is used for reducing the states of a primitive flow table.

Steps to obtain primitive flow table:-

- * First find all compatible pairs using implication table as explain earlier
- * Next find a larger set of compatibles known as maximal compatibles, by using a merger diagram.
- * Finally determine the minimal set of maximal compatibles that satisfies the close-covering conditions.

Compatible states:- Two states are said to be compatible if for every i/p state there exist the same o/p whenever they are specified and their next state are also

The unspecified state or o/p is denoted by '-' which could be assigned any desired state or o/p and stable states are circled.

Stable state

	00	01	11	10
a	c, -	a , 0	b, -	- , -
b	- , -	a, -	b , 0	e, -

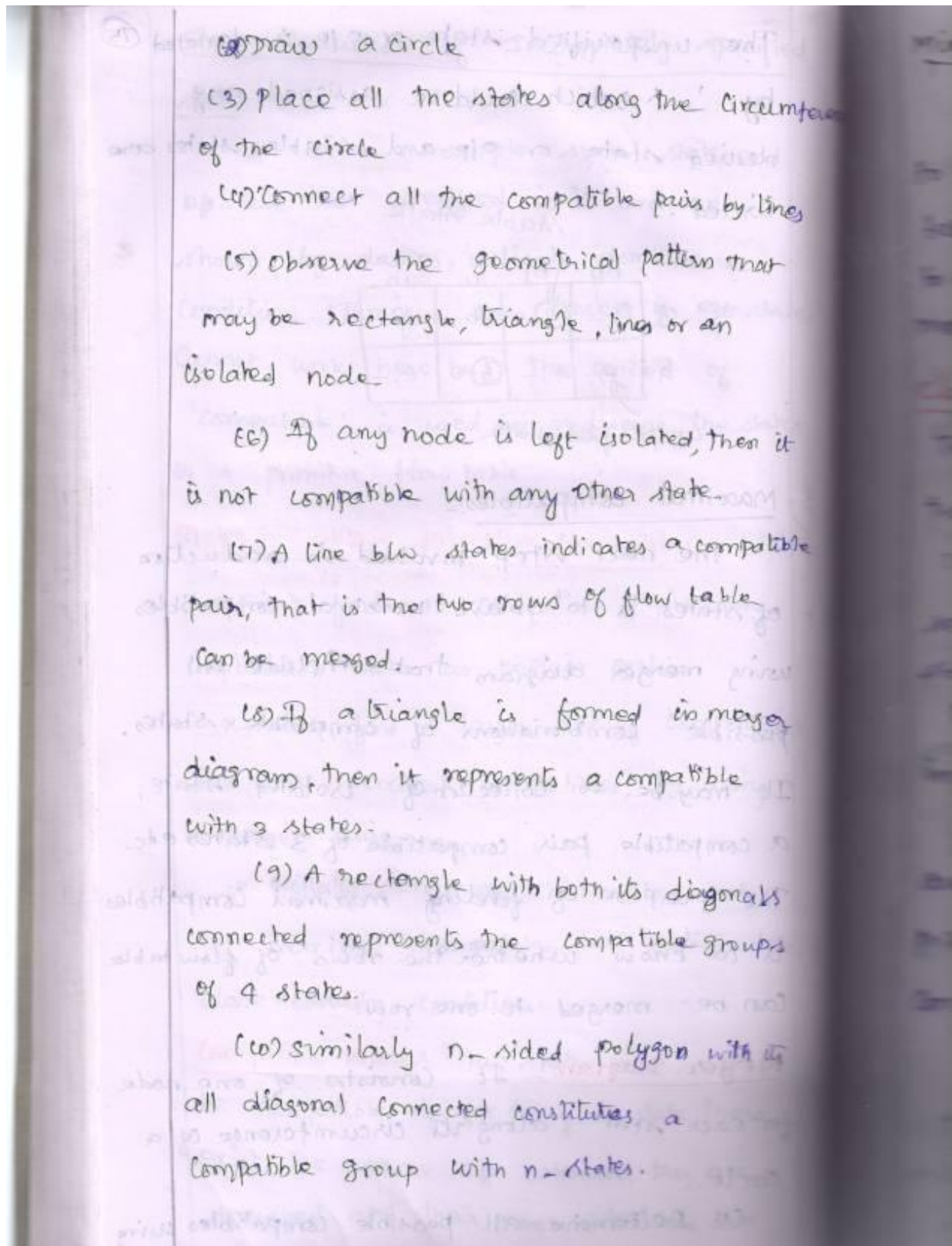
unspecified state

Maximal Compatibles:-

The next step involved in reduction of states is to obtain maximal compatibles using merger diagram that include all possible combinations of compatible states. It may be a collection of isolated state, a compatible pair compatible of 3 states etc. The purpose of finding maximal compatibles is to know whether the rows of flow table can be merged to one row.

Merger Diagram:- It consists of one node for each state along its circumference of a circle.

(i) Determine all possible compatibles using



Minimal set of maximal compatibles

(47)

The maximal compatible can be used to merge 2 or more than 2 rows of flow table into one row. However it is not necessary to use entire set of maximal compatibles for merging rows.

Close - Covering Condition:

The conditions to be satisfied for merging the rows of flow table are:

(1) Covering condition: The selected minimal set must include all the states of the state table.

(2) Close condition: The set must include implied states if any.

In the modified table the ~~non~~ minimal states are replaced by internal states and prime implicants are replaced by maximal compatibles.

Race-Free state Assignments

Techniques used:

(1) Shared row state assignment

(2) Multiple row " "

(3) One hot " "

(1) Shared row state Assignments

This method introduces an extra row in a flow table that is "shared" b/w the 2 stable states. Adding a new row to the flow table permits an unstable state transition b/w the 2 stable states.

Two row flow table:

For 2 rows flow table, only one binary variable is required to assign two states. Critical race may occur only when two if variables change at the same time. For single variable there will be no critical race.

Three row Flow table:

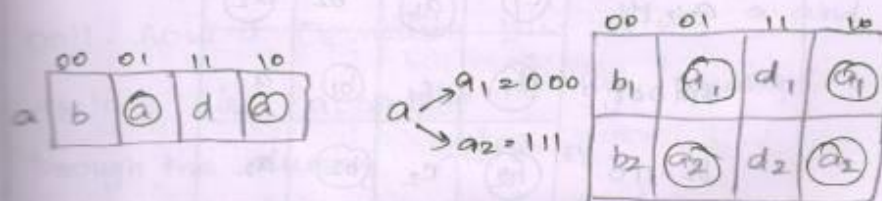
For 3 row flow table, a minimum of 2 binary variables are needed to assign 3 states. In this case there may be a situation of

may yield a critical race condition if assignment is not done properly. (19)

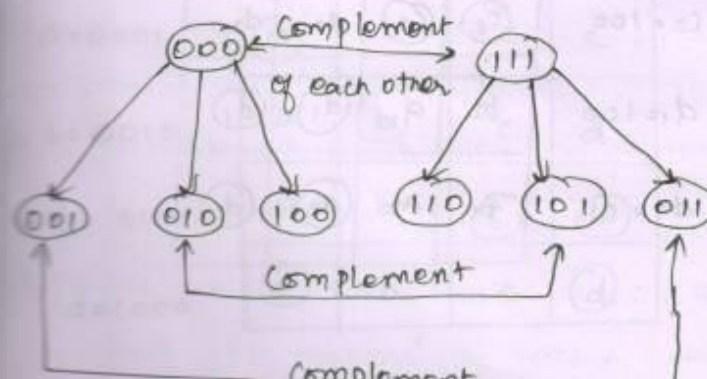
(a) Multiple Row State Assignment:-

In this method, each row of flow table is split into 2 or more rows in such a way that the binary value of both the split rows should be complement of each other and the next state should be adjacent to present state corresponding to that row.

For example, a row of a part of flow table, say 'a' is split into 2 or more states a_1 and a_2 with binary assignment 000 and 111.



Universal State Assignment Map



Example Assign the binary variable to the reduced flow table as shown in fig using multiple row assignment technique.

Four row

	00	01	11	10
a	c	(a)	b	(a)
b	(b)	c	(b)	a
c	(c)	(c)	d	d
d	b	a	(d)	(d)

	00	01	10	11
$q_1 = 000$	(c ₂)	(a ₁)	b ₁	(a ₁)
$q_2 = 111$	(c ₁)	(a ₂)	b ₂	(a ₂)
$b_1 = 001$	(b ₁)	c ₁	(b ₁)	a ₁
$b_2 = 110$	(b ₂)	c ₂	(b ₂)	a ₂
$c_1 = 011$	(c ₁)	(c ₁)	d ₁	d ₁
$c_2 = 100$	(c ₂)	(c ₂)	d ₂	d ₂
$d_1 = 100$	b ₂	a ₁	(d ₁)	(d ₁)
$d_2 = 011$	b ₁	a ₂	(d ₂)	(d ₂)

3) one hot state assignment..

In this method, only one variable is active or 'hot' for each row in the original flow table. For example, a 4 row flow table requires 4 binary values to assign the states. Many extra rows are needed to provide single variable changes b/w the state transitions.

(Example) The state assignment for a is 0001 and the state assignment for b is 0010. There are 2 variables change from a to b when transition occurs. A new row e is introduced whose state assignment is 0011. Now a transition b/w state a and state b is accomplished by passing through the dummy state e.

Four Row Flow table

	00	01	10	11
a=0001	(a)	b	c	c
b=0010	a	(b)	c	d
c=0100	a	b	(c)	(c)
d=1000	(d)	b	c	(d)

One Hot State Assignment

a=0001	(a)	e	f	f
b=0010	e	(b)	g	h
c=0100	f	g	(c)	(c)
d=1000	d	h	i	(d)
e=0011	a	b	-	-
f=0101	a	-	c	c
g=0110	-	b	c	-
h=1010	-	b	-	d
i=1100	-	-	c	-

Unspecified output Assignments:

The unspecified o/p's must be assigned with binary values so that the circuit does not produce any momentary false o/p during transitions b/w 2 stable states.

Rules:-

* If the 2 stable states have same o/p, assign same o/p value to the unstable state.

* If the 2 stable states have different o/p, assign don't care to the unstable state which is transient b/w them.

Hazards

(83)

Apart from the critical race problem, there is one more problem called Hazards, which may cause circuit malfunction.

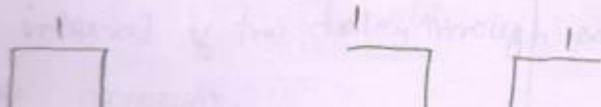
Hazards is actually an unwanted switching transient that occurs at the o/p, in the form of glitches or spikes, of a circuit because of the unequal propagation delays through different path of the combinational circuit or the asynchronous seq. circuit.

Two types of hazards:-

- (1) Static hazard
 - Static 0
 - Static 1
- (2) dynamic hazard

Static 0 hazard:- When a circuit was expected to produce '0' o/p but it produced momentary '1' o/p, it is referred to as static 0 hazard.

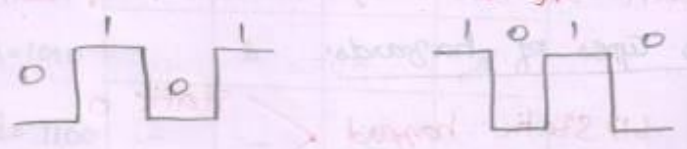
Static 1 hazard:- When a circuit was expected to produce '1' o/p but it produced momentary '0' o/p, it is referred to as static 1 hazard.



② Dynamic hazard:- Another class of hazards exists where the o/p changes more than once as a result of a single i/p variable change, which is referred to as dynamic hazard.

A 0-1-0 or 1-0-1-0 change is called a dynamic hazard. This occurs when at least 3 levels of logic are present and can be eliminated by using only 2 levels of logic.

changing pattern ~~1-0-1-0~~

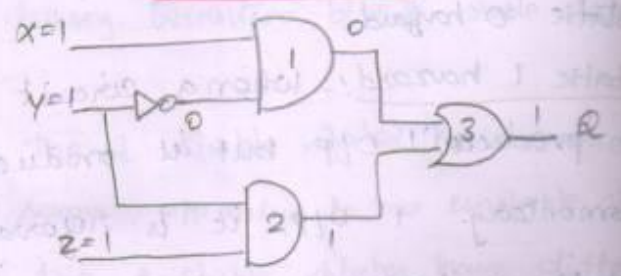


changing pattern 0-1-0-1

Circuits with Hazard:-

AND-OR circuit:-

Initially all 3 i/p are 1 and ckt gives the o/p 1. Initial condition



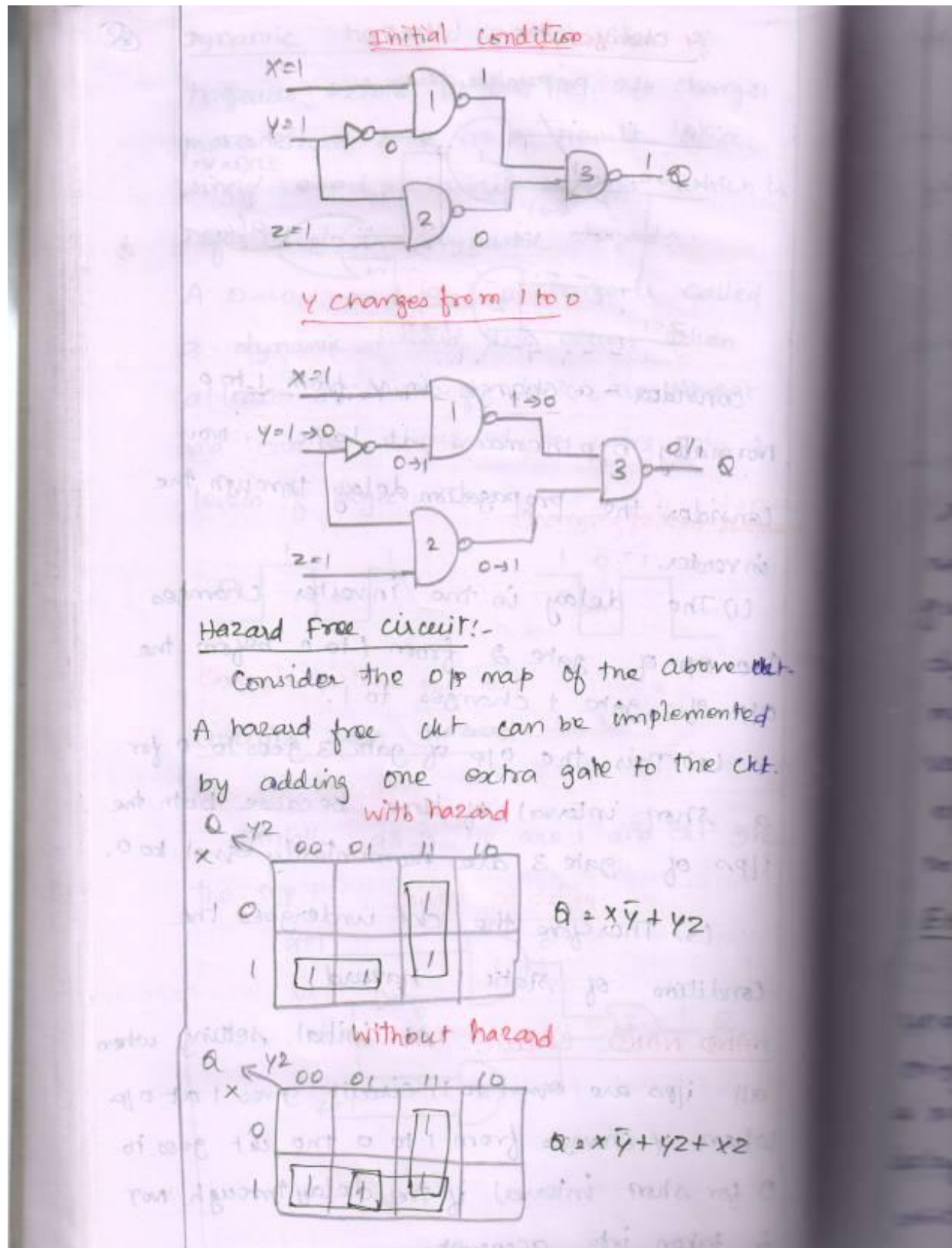
y changes from 1 to 0

Propagation delay

consider a change in y from 1 to 0.
 Normally o/p remains at logic 1. Now
 consider the propagation delay through the
 inverter.

- (1) The delay in the inverter changes
 the o/p of gate 2 from 1 to 0 before the
 o/p of gate 1 changes to 1.
- (2) Thus the o/p of gate 3 goes to 0 for
 a short interval of time because both the
 i/p's of gate 3 are momentarily equal to 0.
- (3) Therefore the ckt undergoes the
 condition of static 1 hazard.

NAND-NAND circuit:- At initial setting when
 all i/p's are equal to 1, circuit gives 1 at o/p.
 When y changes from 1 to 0 the ckt goes to
 0 for short interval if the delay through not
 is taken into account.



② This type of hazard cannot be eliminated by adding an extra gate. It can be eliminated only by adjusting the propagation delay path of a signal. Because the delay paths depend on the final logic design and the ckt layout, the removal of essential hazards is an application dependent problem.

Detection of Essential Hazard

After 1 transition After 3 transition

I/P $\Rightarrow 0 \rightarrow 1$ $0 \rightarrow 1 \rightarrow 0 \rightarrow 1$

	0	1
a	(a)	b
b	c	(b)
c	(c)	d
d	(d)	(d)

$a \rightarrow b$ $a \rightarrow b \rightarrow c \rightarrow d$
 Different stable state hazard exists

$c \rightarrow d$ $c \rightarrow d \rightarrow d \rightarrow d$
 Same state no hazard exists

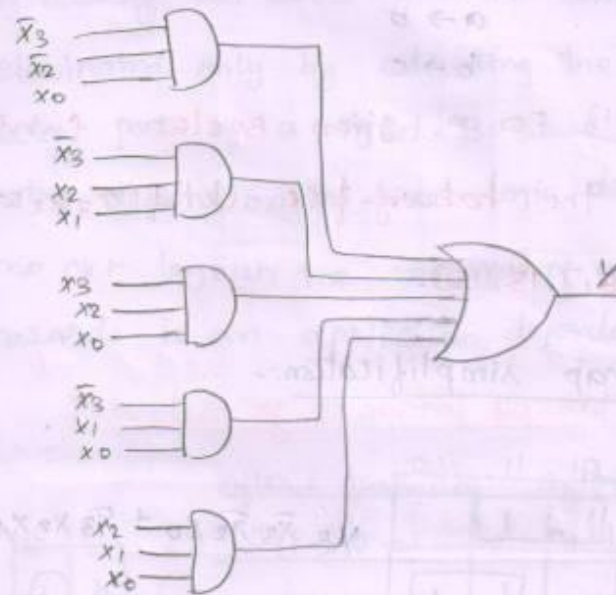
$d \rightarrow d$ $d \rightarrow d \rightarrow d \rightarrow d$
 Same state no hazard exists

I/P $\Rightarrow 1 \rightarrow 0$ $1 \rightarrow 0 \rightarrow 1 \rightarrow 0$

$b \rightarrow c$ $b \rightarrow c \rightarrow d \rightarrow d$
 Different stable state hazard exists

$d \rightarrow d$ $d \rightarrow d \rightarrow d \rightarrow d$
 Same state no hazard exists

Logic diagram:



Example 2.2 Show the following flow table, shown in fig, has essential hazards

	0	1
a	a	b
b	b	c
c	c	d
d	d	a

Solution: Essential hazard exists when the present state has a different next total state after one transition of an ilp variable and after 3 transitions of an ilp variable. Evaluate each stable state

After 1 transition After 3 transition⁽¹¹⁾

Flip $\rightarrow 0 \rightarrow 1$ $0 \rightarrow 1 \rightarrow 0 \rightarrow 1$

States

$a \rightarrow b$ Different $a \rightarrow b \rightarrow b \rightarrow c$
 $b \rightarrow c$ Different $b \rightarrow c \rightarrow c \rightarrow d$
 $c \rightarrow d$ Different $c \rightarrow d \rightarrow d \rightarrow a$
 $d \rightarrow a$ Different $d \rightarrow a \rightarrow a \rightarrow b$

*The essential hazards are,

$a \rightarrow b$
 $b \rightarrow c$
 $c \rightarrow d$
 $d \rightarrow a$

Example 3 Obtain the Boolean form for the hazard free realisation of the Boolean fn

$f = \Sigma(0, 2, 6, 7, 8, 10, 12)$

sol:- Implement using k-map overlap the 2 groups

AB \ CD	00	01	11	10
00	1			1
01			1	1
10	1			
11	1			1

10

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

Hence the hazard free Boolean eqn is,

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

P. Balaji

Index - Unit V

Topic Name	Page No
RAM	1-12
Memory decoding	12-17
ROM	20-24
Error Detection & correction	17-20
PAL	29-33
PLA	25-28
Sequential programmable devices	33-37
ASIC	37-42

P. Balakrishna

UNIT-5 MEMORY AND PROGRAMMABLE LOGIC

DEVICES

RAM and ROM - memory decoding - Error detection and correction - Programmable Logic array - Programmable Array Logic - Sequential programmable Devices - Application Specific Integrated Circuits

Introduction :- Memory is an important unit in any data processing system where information, after processing is stored in memory and retrieved from memory whenever needed. It is basically a collection of cells used to store binary information such as RAM and ROM.

RANDOM ACCESS MEMORY (RAM) RAM is a processing memory unit. Unlike storage devices like hard disc, magnetic tapes where Read and write operations are carried out in a predetermined manner, in RAM these operations occur in random fashion. It is a **volatile type of memory**, where the stored information will be lost if the power is turned off.

Basic Terms and Definitions.

* Memory stores programs and data.

* 1 byte = 8 bits

* 1 word: Group of bits arranged in multiple of bytes, a unit of transfer b/w main memory and registers, usually size of register.

* $1 \text{ KB} = 2^{10} \text{ bytes}$; $1 \text{ MB} = 2^{20} \text{ bytes}$; $1 \text{ GB} = 2^{30} \text{ bytes}$

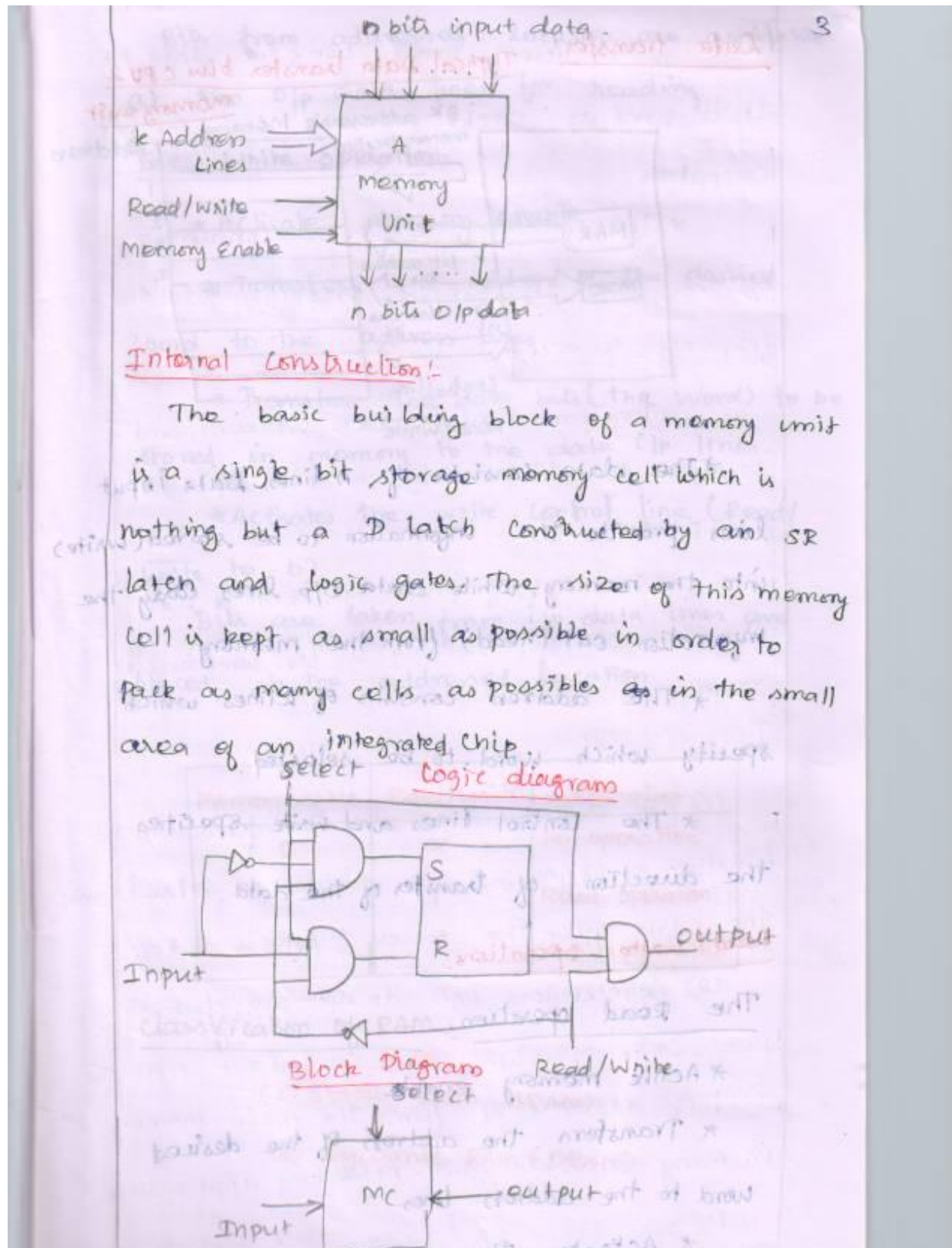
$1 \text{ TB} = 2^{40} \text{ bytes}$.

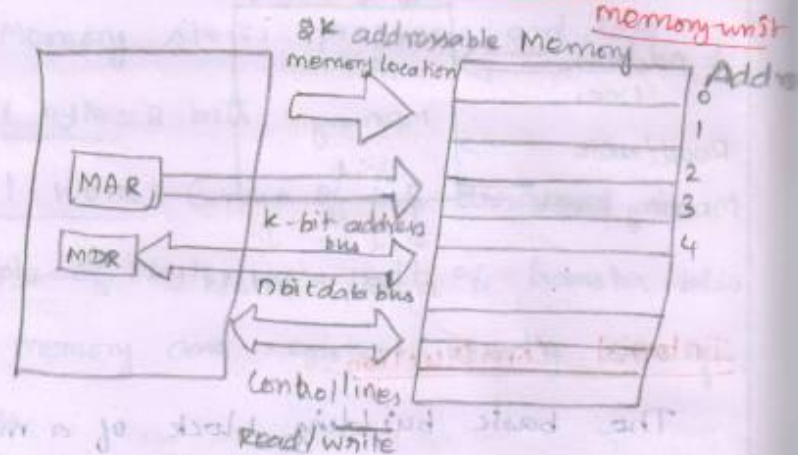
* Access time: The time required for a memory device ~~do not~~ ~~posses~~ all these properties to select an addressable memory location and read the content (word).

* Cycle time: The time required for a memory to complete a write operation.

Memory unit:

Stores binary data in the form of words which are multiple of bytes. For example, a 16-bit word is a group of two bytes, and a 64-bit word is formed by 8 bytes.



Data transfer:-Typical Data transfer b/w CPU & memory-unit

* The data consists of n lines. Data input lines provide the information to be stored (write) into the memory, while data o/p lines carry the information out (read) from the memory.

* The address consists of k lines which specify which word to be selected.

* The control lines and write specifies the direction of transfer of the data.

Read / write operations:-The Read operation:-

- * Active memory enable
- * Transfers the address of the desired word to the address lines.

* Activates the

Bits from addressed location are available⁵
at the o/p data lines for reading.

The write operation:

- * Activate memory enable
- * Transfers the address of the desired word to the address lines
- * Transfers the data bits (the word) to be stored in memory to the data i/p lines.
- * Activates the write control line (Read/Write to 0)

Bits are taken from i/p data lines and placed in the addressed location.

Memory enable	Read/write	Operation
0	X	No operation
1	1	Read operation
1	0	Write operation

Classification of RAM:

(1) Static RAM (SRAM)

(2) Dynamic RAM (DRAM)

SRAM: * Bits are stored using the data FF.

It is used as cache memory for CPU in computers.

* The amount of power consumed by the SRAM is directly proportional to the frequency of access.

* They are used in LCD screens, and printers to hold the image to be displayed and printed.

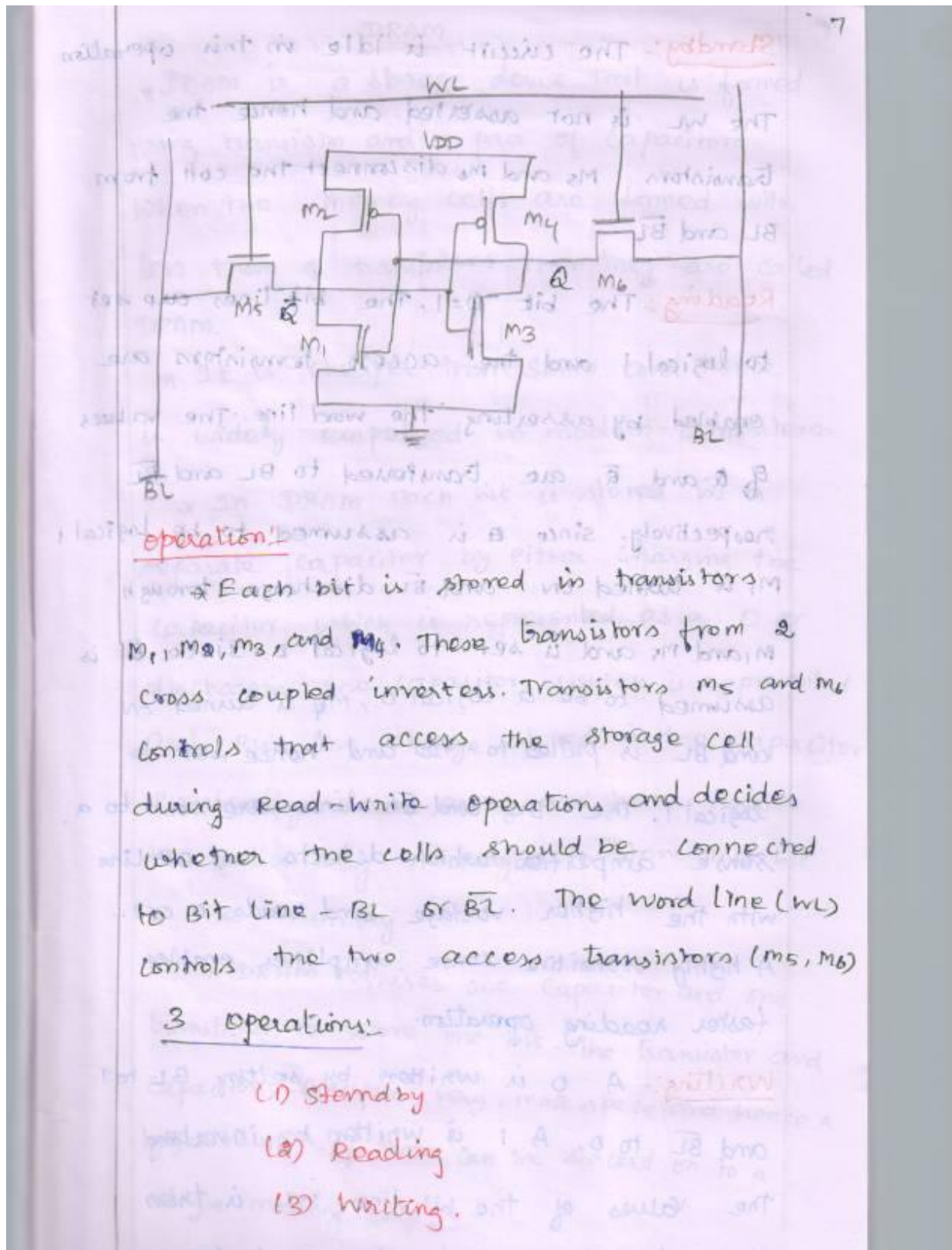
* The cost of producing SRAM is high. Also the storage capacity is less compared to DRAM.

Types of SRAM:

(1) Non-volatile SRAM has the fundamental function of SRAM but in addition it also save the data when the power supply is cutoff.

(2) Asynchronous RAM do not use clock signals. Read-write operations are carried out with the help of addresses. It forms the main memory in many industrial and networking applications.

(3) Synchronous RAM use clock signals.



Standby:- The circuit is idle in this operation.

The W_L is not asserted and hence the transistors M_5 and M_6 disconnect the cell from BL and \overline{BL} .

Reading:- The bit $Q=1$. The bit lines are set to logical 1 and the access transistors are enabled by asserting the word line. The values of A and \overline{A} are transferred to BL and \overline{BL} respectively. Since A is assumed to be logical 1, M_1 is turned on and \overline{BL} discharges through M_1 and M_5 and is set to logical 0. Since \overline{A} is assumed to be a logical 0, M_4 is turned on and BL is pulled to V_{DD} and hence set to logical 1. The BL and \overline{BL} lines are sent to sense amplifier which detects the bit line with the higher voltage and reads 1 or 0. A highly sensitive sense amplifier enables faster reading operation.

Writing:- A 0 is written by setting BL to 1 and \overline{BL} to 0. A 1 is written by inverting the values of the bit lines. W_L is then

DRAM

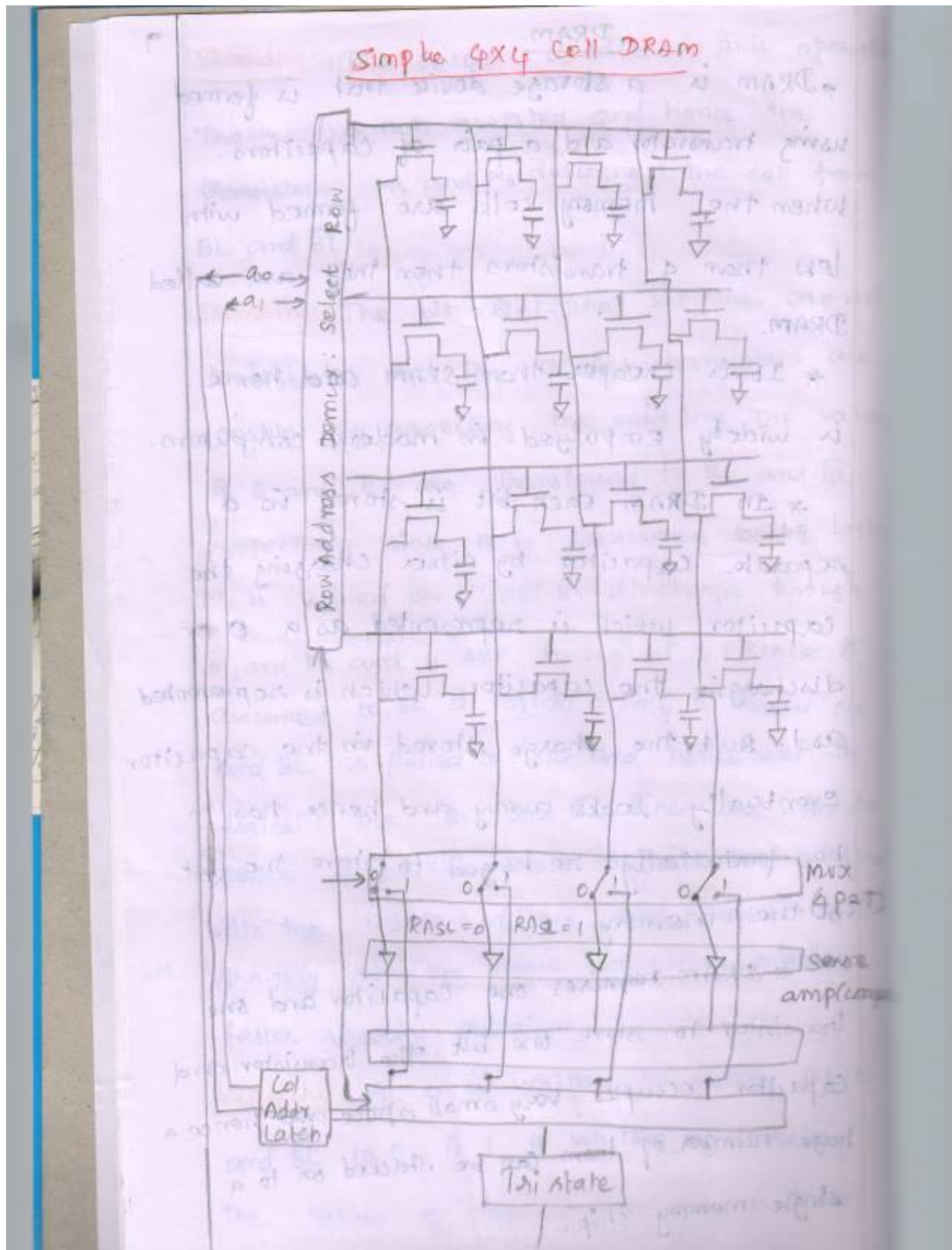
* DRAM is a storage device that is formed using transistor and a pair of capacitors.

When the memory cells are formed with less than 4 transistors then they are called DRAM.

* It is cheaper than SRAM and hence is widely employed in modern computers.

* In DRAM each bit is stored in a separate capacitor by either charging the capacitor which is represented as a 0 or discharging the capacitor which is represented as 1. But the charge stored in the capacitor eventually leaks away and hence has to be periodically recharged to store the bit in the memory.

* DRAM requires one capacitor and one transistor to store one bit. The transistor and capacitor occupies very small space and hence a huge number of them can be stacked on to a single memory chip.



Reading:

* Initially the sense amplifiers are disconnected.

* The bit lines are pre-charged to a voltage. After this the pre-charging circuit is switched off. All the charge leaks away from the capacitor.

* To connect a cell's storage capacitor to the bit line, the word line of the desired row is then driven high and then the transistor starts conducting.

* If the stored value is a logical 1, then the charge is transferred from the storage cell to the bit line.

* If the stored value is logical 0, then the charge is transferred from the ^{bit line} storage cell to the storage cell.

* This process creates a small voltage difference b/w the two bit lines.

* The sense amplifiers are now connected to bit lines.

* This process opens up the row and allows the data in the desired cell

Writing: To write data into a particular storage cell a row is opened and the specific column's sense amplifier is forced to a high or a low state. This causes the Bit line to charge and discharge the capacitor of the particular storage cell.

Refresh Rate: The capacitor in each storage cell has to be refreshed periodically. Refresh logic is provided in a DRAM which performs a periodic refresh.

Memory Decoding

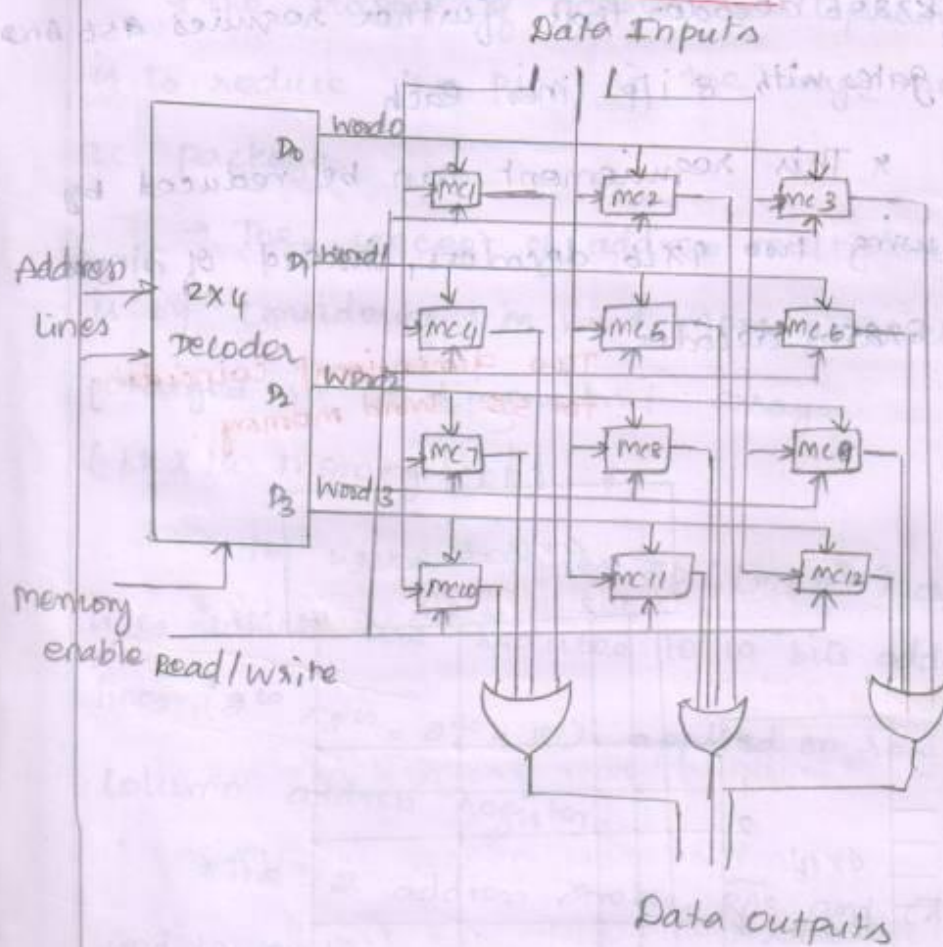
* In order to splice a memory device into address space of the processor, decoding is necessary. In 4×3 RAM, it consists of 4 words of 3 bits each, having total 12 memory cells.

* To address four words there is a need of 2 address lines; hence a 2×4 decoder can be used to select any one location.

The read/write i/p determine operation. 13

*when read/write = 1, the word from selected location is available at o/p terminal for reading and when read/write = 0, data from i/p lines written into the selected lines.

A 4X3 RAM Cell



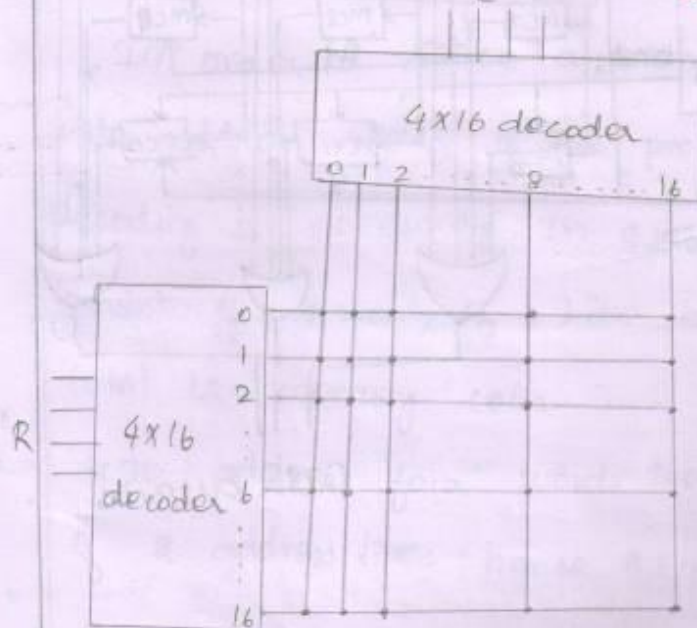
Coinciding decoding:-

* As the memory space in a memory is large, the requirement of decoder is large, in turn the number of AND increases in more numbers.

* For example, a 256 word memory requires 8x256 decoder that further requires 256 gates with 8 i/p lines each.

* This requirement can be reduced by using two 4x16 decoders, instead of single 8x256 decoder.

Two dimensional coinciding for 512 word memory.



* In this arrangement one decoder select row address which constitute four most significant bits of addresses and the other select column address that contains four least significant bits of addresses.

Address Multiplexing:

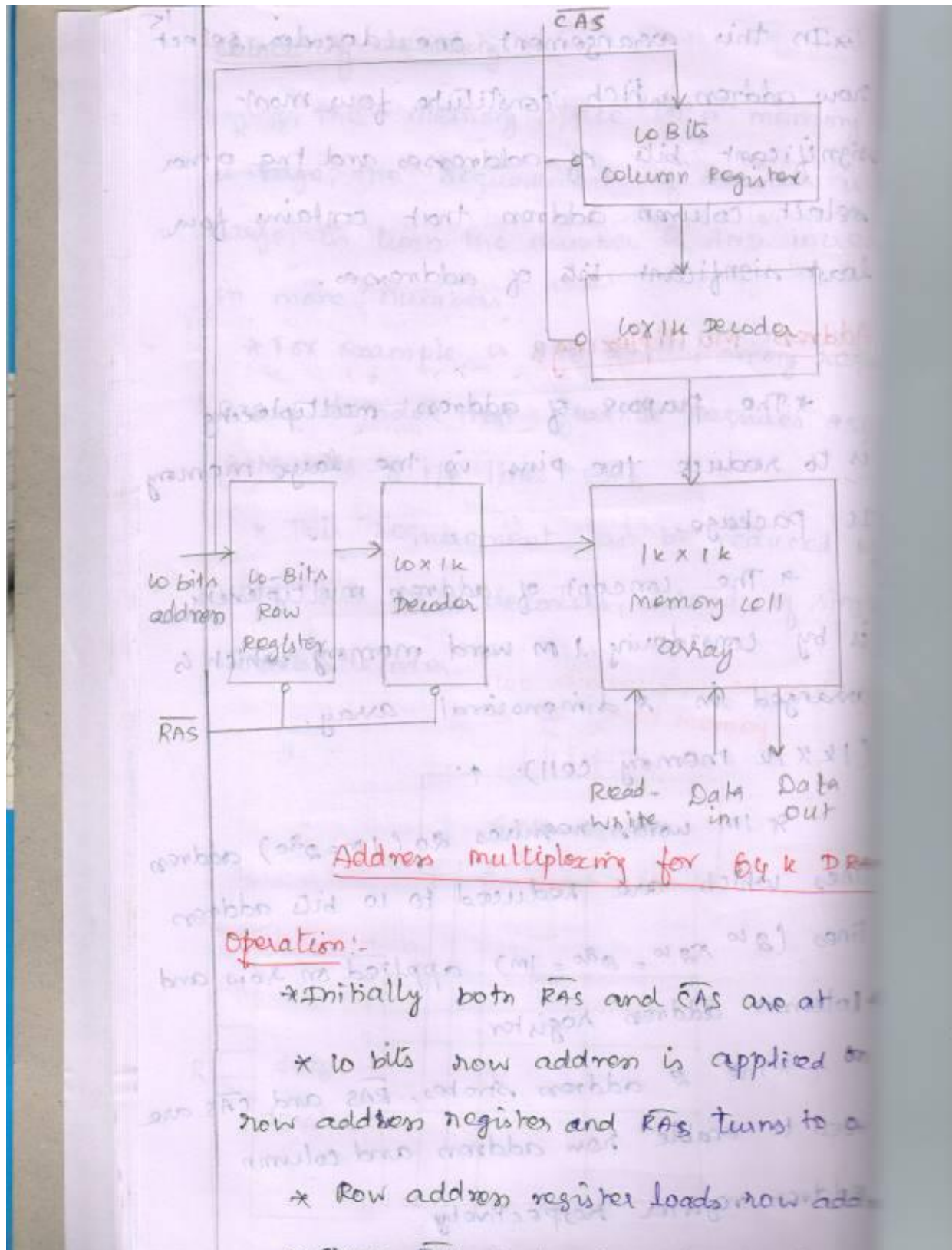
* The purpose of address multiplexing is to reduce the pins in the large memory IC package.

* The concept of address multiplexing is by considering 1M word memory which is arranged in 2 dimensional array.

(1k x 1k memory cell). *

* 1M words requires 20 ($1M = 2^{20}$) address lines which are reduced to 10 bits address lines ($2^{10} \times 2^{10} = 2^{20} = 1M$) applied on row and column address register.

* The 2 address strobes, \overline{RAS} and \overline{CAS} are used to enable row address and column address register respectively.



one row of array.

* \overline{RAS} returns to 1

* 10 bits column address is applied to column register.

* \overline{CAS} becomes 0.

* Column address register is enabled and it stores column address.

* Column decoder is enabled and it selects one column of array.

* Thus one cell corresponding to row and column address gets selected.

* Read/write operation is performed.

* \overline{CAS} goes back to 1 before starting next location selection.

Error Detection and Correction

* It is important to have some error detection and error correction mechanism for reliable memory interaction.

* The common mechanism for detection and correction of error is parity bit.

* While reading the parity bit is checked.

Error Correction:

* For correcting the error, multiple parity check bits are stored with the data.

* These parity bits are read along with data and compared with new generated check bits.

* If the check bits are matched with stored data, it refers no error has occurred otherwise a syndrome is generated which is used to identify the erroneous bit.

Hamming Code:

If more error-correcting bits are included with a message, and if those bits can be arranged such that different incorrect bits produce different error results then bad bits could be identified.

Determination of the length of the data bits and check bits:

It is a class of binary linear code where for each integer $i \geq 2$ there is a code of total length $L = 2^i - 1$ bits which

$$\boxed{d = 2^i - i - 1} \rightarrow \textcircled{1}$$

The syndrome S contains P bits of the range 0 to $2^P - 1$ where 0 indicates no error and $2^P - 1$ indicate which of the L bits in error. Therefore the range of $P \geq d + P$ that gives the following relationship

$$\boxed{d + P \leq 2^P - 1} \rightarrow \textcircled{2}$$

This eqn. can be used to determine the length of data bits.

Example 1) Find the length of message bits and check bits for $i=1$ and $i=4$. Verify the result.

Sol

For $i=3$,

$$L = 2^i - 1 = 2^3 - 1 = 7.$$

$$d = 2^3 - 3 - 1 = 4.$$

Therefore

$$P = L - d = 7 - 4 = 3$$

Using eqn (2)

$$d \leq 2^P - 1 - P = 2^3 - 1 - 3 = 4.$$

For $i=4$,

$$L = 2^i - 1 = 2^4 - 1 = 15.$$

$$d = 2^4 - 4 - 1 = 11.$$

Therefore

$$P = L - d = 15 - 11 = 4.$$

Using eqn (2)

$$d \leq 2^4 - 1 - 4 = 11.$$

Single - Error Correction and Double Error

Detection:

With small modification, hamming code can be used to correct single error and detect double errors. Thus the 13bit code word will be 0011100101001. This code word gives even parity when it evaluated if $p=0$ the parity is correct else parity wrong. Hence we can conclude.

(1) If $s=0$ and $p=0$, no error

(2) If $s \neq 0$ and $p=1$, single error occurs and can be corrected.

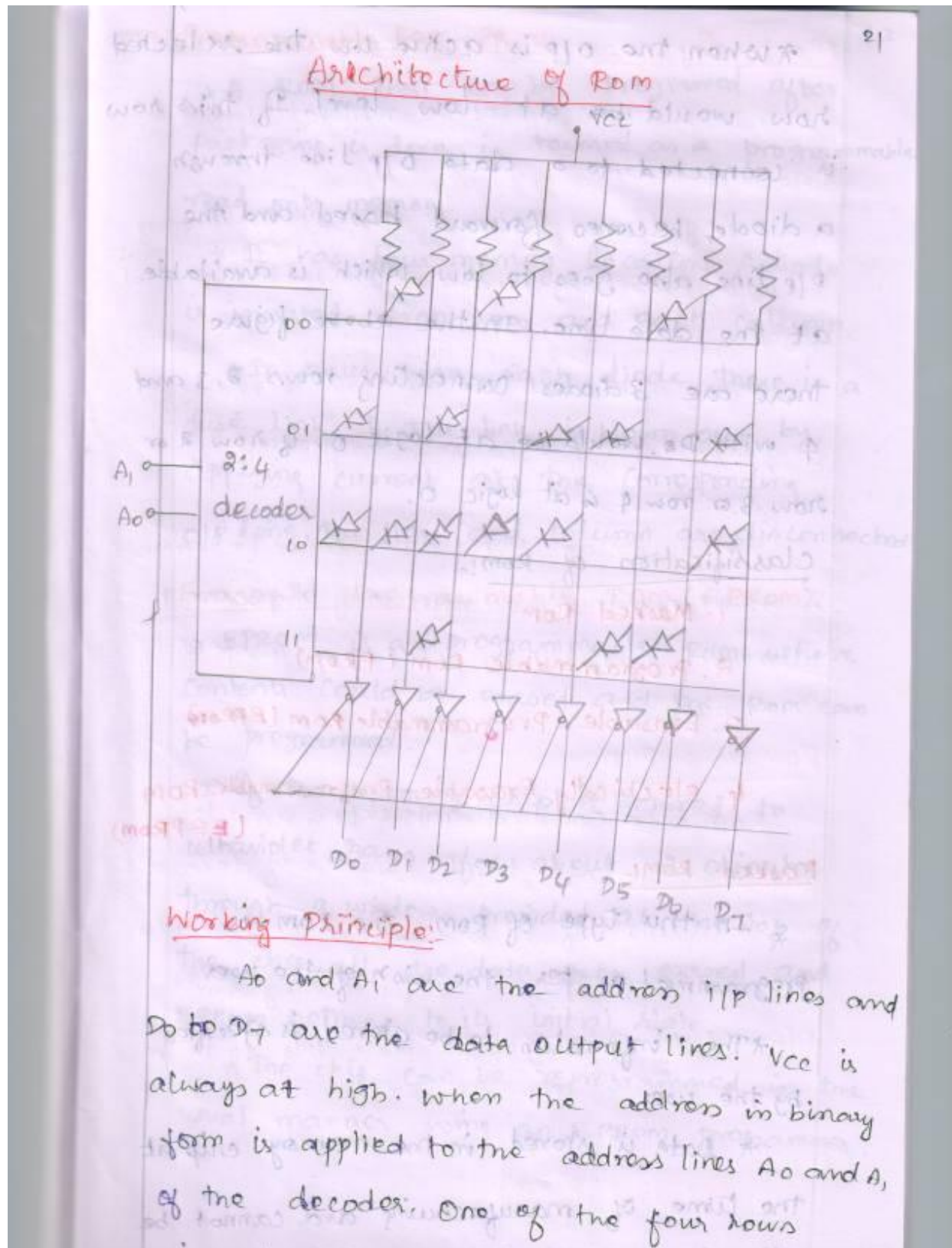
(3) If $s \neq 0$ and $p=0$, double error occurs and cannot be corrected.

(4) If $s=0$ and $p=1$, error occurred in

Read Only Memory (Rom)

Rom is a memory device which stores binary information permanently. Data stored in Rom can be read, but cannot be written to it.

Architecture of Rom: Basically Rom consists of a number of diodes and a decoder.



13 * When the o/p is active low, the select row would be at low level. If this is connected to a data o/p line through a diode, becomes forward biased and the o/p line also goes to low which is available at the data line. In the above figure there are 3 diodes connecting rows 2, 3 & 4 with D₈, would be at logic 0, if row 2, row 3 or row 4 is at logic 0.

Classification of Rom:-

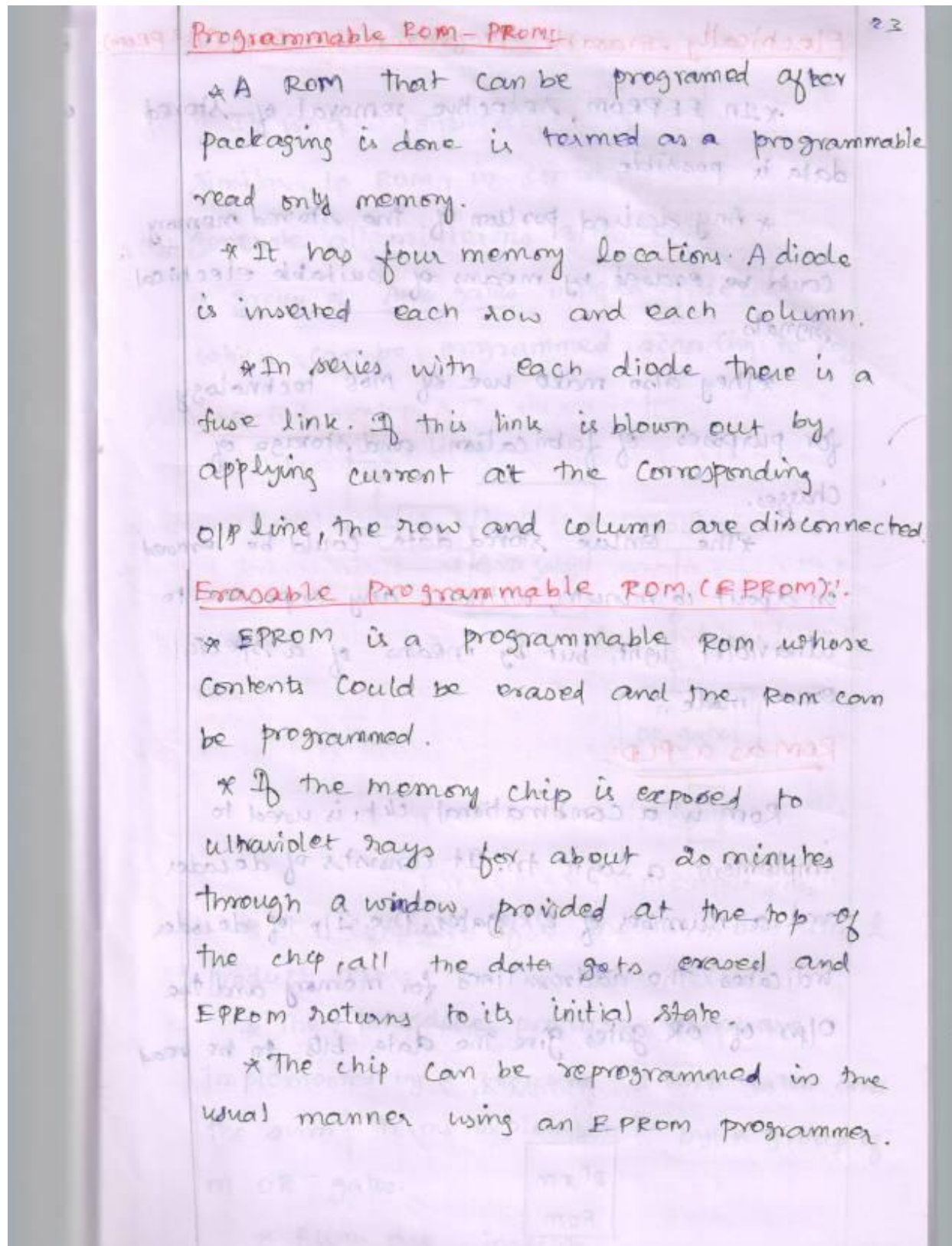
1. Masked Rom
2. Programmable Rom (PROM)
3. Erasable - Programmable Rom (EPROM)
4. Electrically Erasable - Programmable Rom (EEPROM)

Masked Rom:-

* In this type of Rom, the Rom is programmed as per the wish of the user.

* The information to be stored is specified by the user.

* Data is stored in the memory chip. The time of manufacturing and cannot



Electrically Erasable Programmable Rom (EEPROM)

* In EEPROM, selective removal of data is possible.

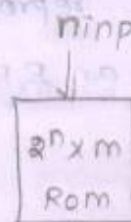
* Any desired portion of the stored memory could be erased by means of suitable electrical signals.

* They also make use of MOS technology for purposes of fabrication and storage of charges.

* The entire stored data could be erased in about 10 minutes, without any exposure to ultraviolet light, but by means of a special erase mode.

Rom as a PLD:-

Rom is a combinational ckt. is used to implement a logic fn. It consists of decoders and a number of OR gates. The n inputs of decoders indicates the address lines for memory and outputs of OR gates give the data bits to be

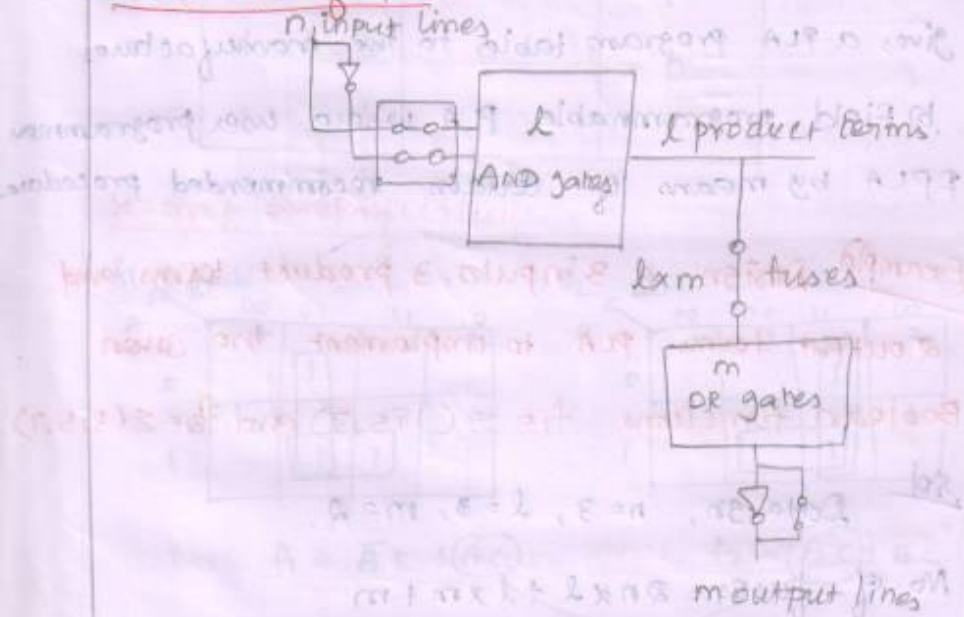


Programmable Logic Array (PLA)

25

It is a combinational circuits that is similar to ROM in concept but does not generate all minterms as in ROM. PLA uses a group of AND gates instead of decoder, which can be programmed according to logic fn.

Structure of PLA:



* It consists of n inputs, m outputs, l product terms and m sum terms.

* The ~~procedure~~ product terms are implemented by a group of l AND gates and the sum terms implemented by a group of m OR gates.

A fuses are inserted.

c) b/w OR gates and o/p lines.

* The number of fuses depends on no. of i/p lines, no. of o/p lines, no. of sum and no. of product terms. Total no. of fuses = ~~2n x l~~

$$2n \times l + l \times m + m$$

* There may be two types of PLA.

a) mask-programmable PLA where the consumer gives a PLA program table to the manufacturer.

b) field-programmable PLA where user programs FPLA by means of certain recommended procedure.

Example Design a 3 inputs, 3 product terms and 2 output terms PLA to implement the given Boolean functions $f_1 = \sum(1, 5, 7)$ and $f_2 = \sum(3, 6)$.

Sol Data given, $n = 3$, $l = 3$, $m = 2$.

$$\text{No. of fuses} = 2n \times l + l \times m + m$$

$$= 2 \times 3 \times 3 + 3 \times 2 + 2$$

$$= 26$$

B/w inputs to AND gates 18 fuses.

B/w AND and OR 6 fuses.

at o/p 2 fuses.

27

Inputs			Outputs	
A	B	C	f ₁	f ₂
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

k-map simplification:-

f₁

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

f₂

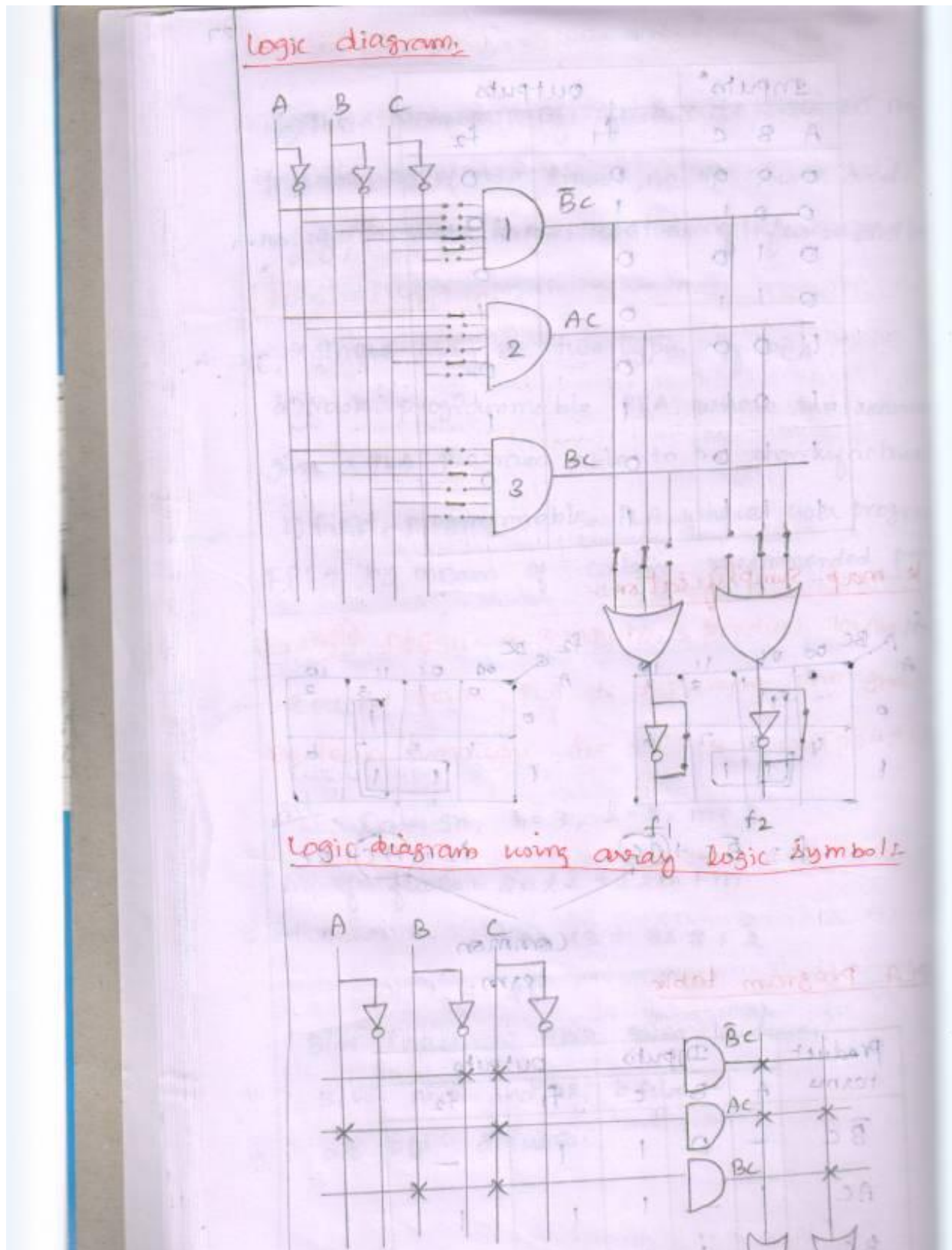
	BC	00	01	11	10
A	0	0	1	3	2
	1	4	5	7	6
			1	1	

$f_1 = \bar{B}C + \textcircled{AC}$
 $f_2 = \textcircled{AC} + BC$

Common term.

PLA program table

Product terms	Inputs			Outputs	
	A	B	C	f ₁	f ₂
$\bar{B}C$	-	0	1	1	-
AC	1	-	1	1	1



Programmable Array Logic (PAL)

PAL is another class of PLD where hundreds of gates are interconnected through hundreds of fuses. Designing concept of PAL is same as that of PLA. In PAL OR array are **fixed** and only **AND array** are programmable.

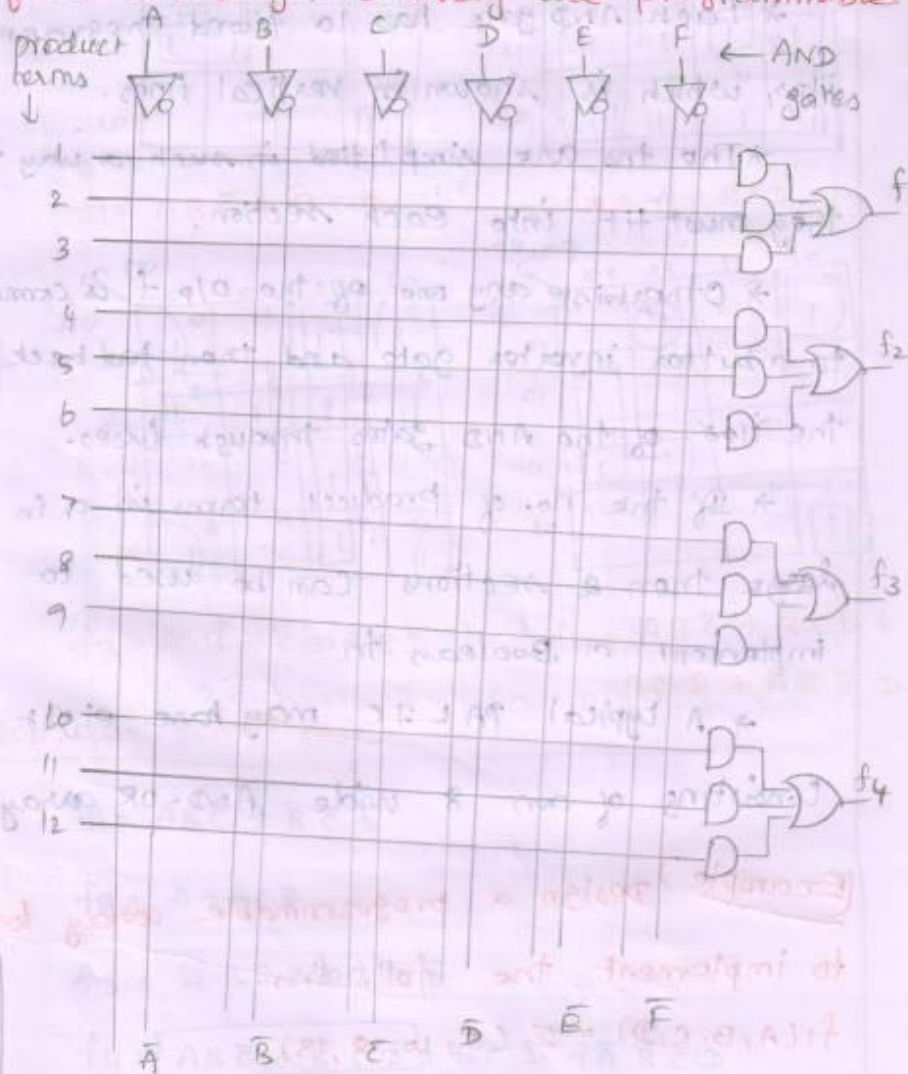
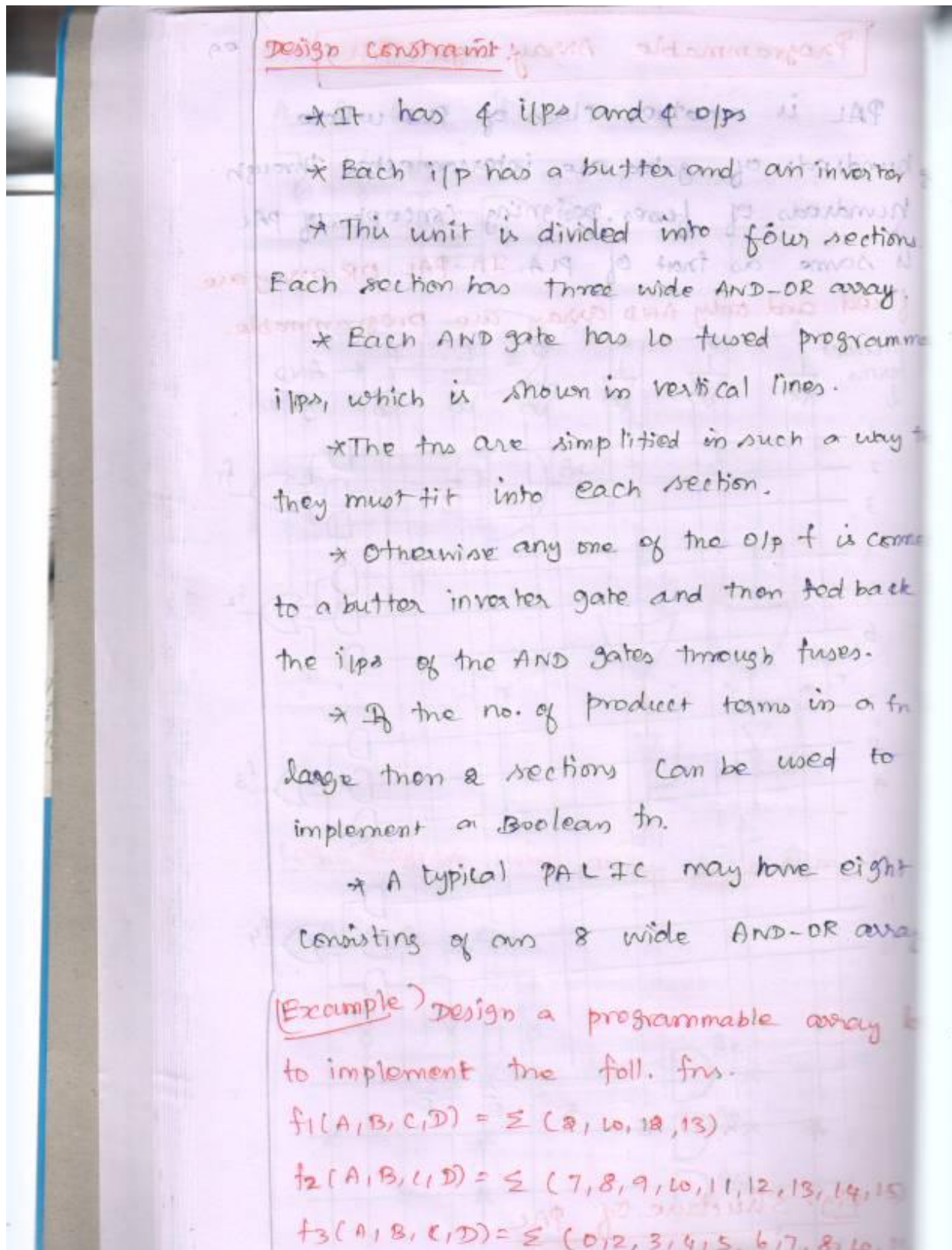
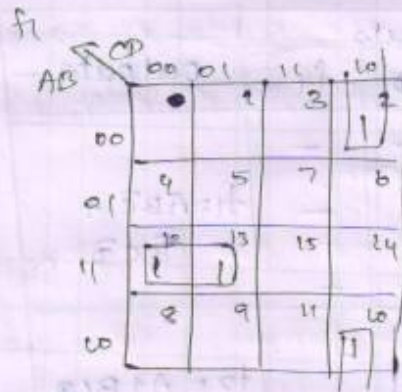


Fig. Structure of PAL

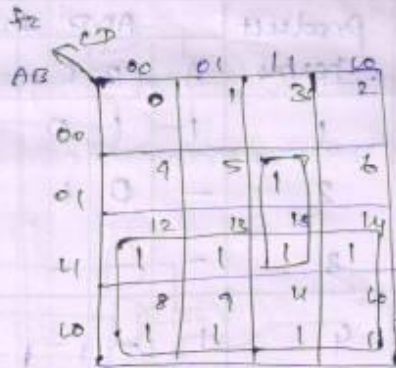


solution:

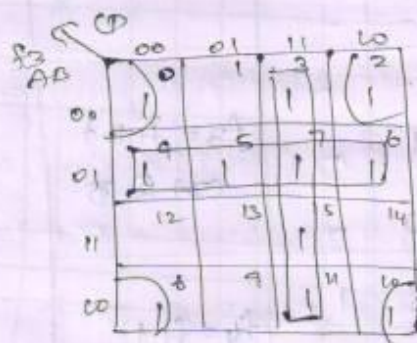
31

K-map simplification:

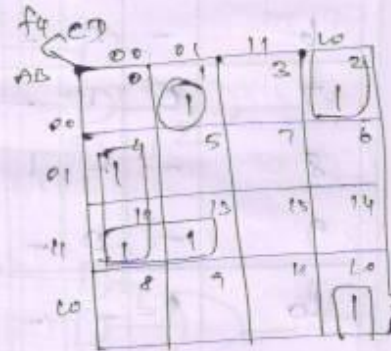
$$f_1 = AB\bar{C} + \bar{B}C\bar{D}$$



$$f_2 = A + BCD$$



$$f_3 = \bar{A}\bar{B} + CD + \bar{B}\bar{D}$$



$$f_4 = AB\bar{C} + \bar{B}C\bar{D} + B\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D$$

Boolean functions:-

$$f_1 = AB\bar{C} + \bar{B}C\bar{D}$$

$$f_2 = A + BCD$$

$$f_3 = \bar{A}\bar{B} + CD + \bar{B}\bar{D}$$

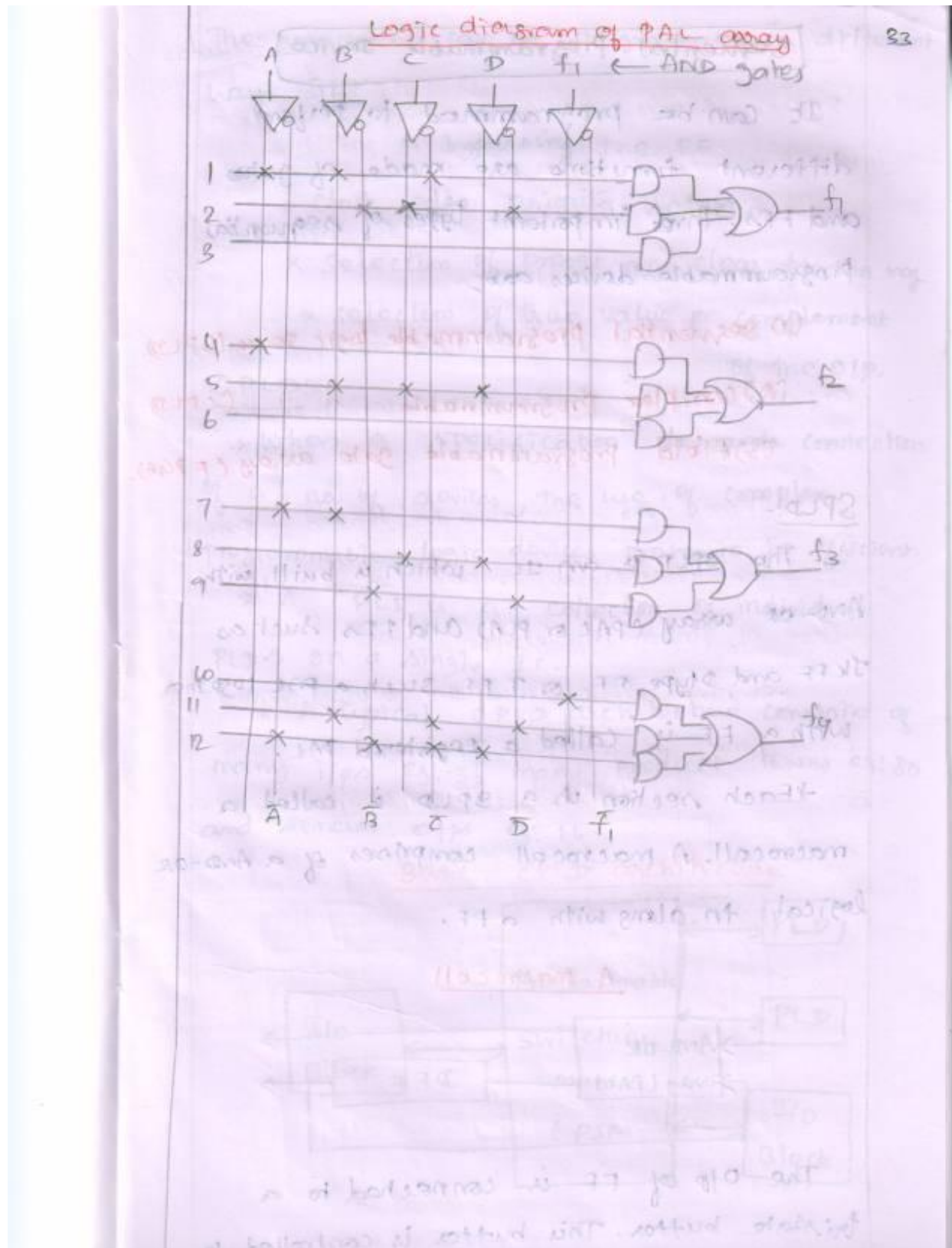
$$f_4 = AB\bar{C} + \bar{B}C\bar{D} + B\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D$$

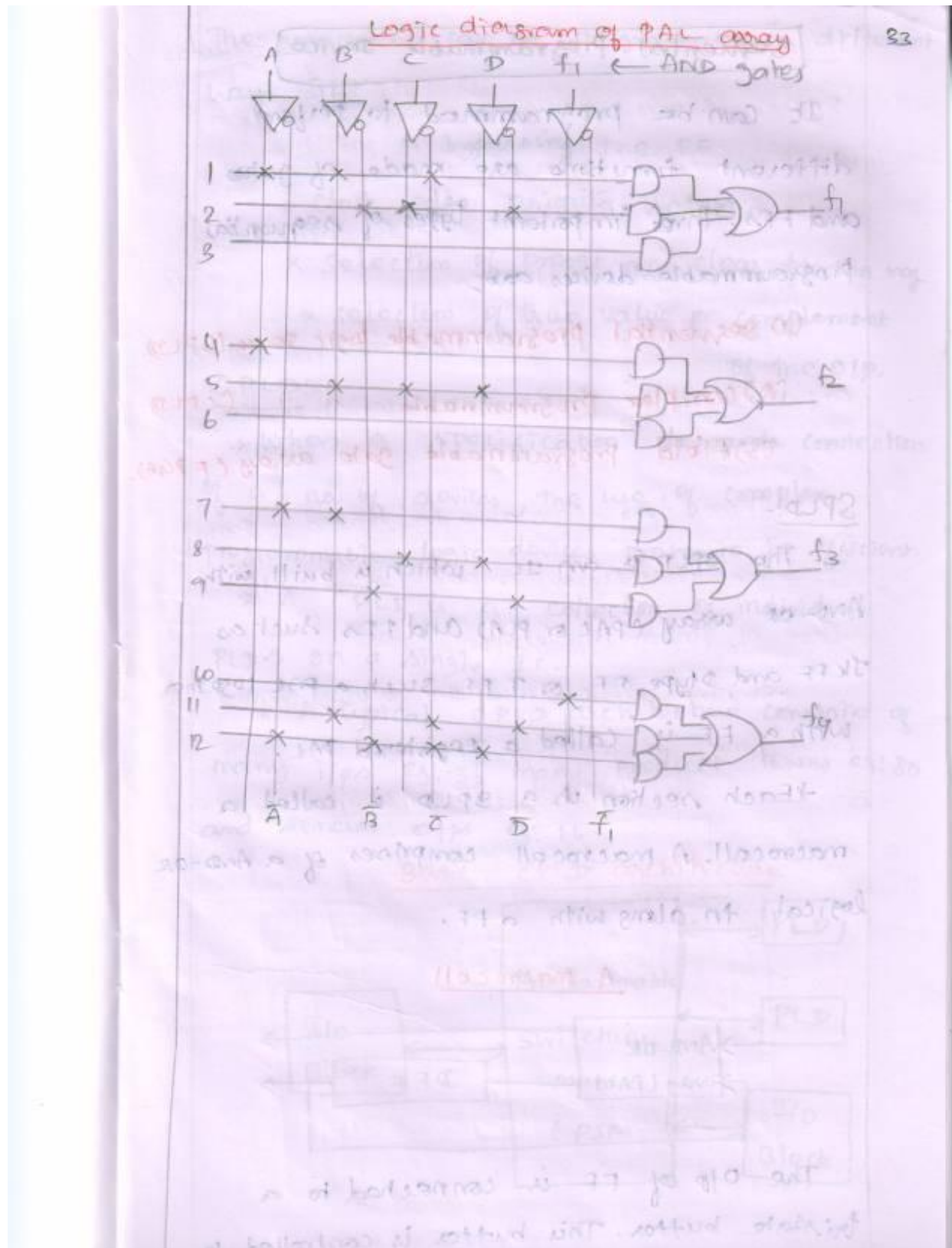
↓
 f_1

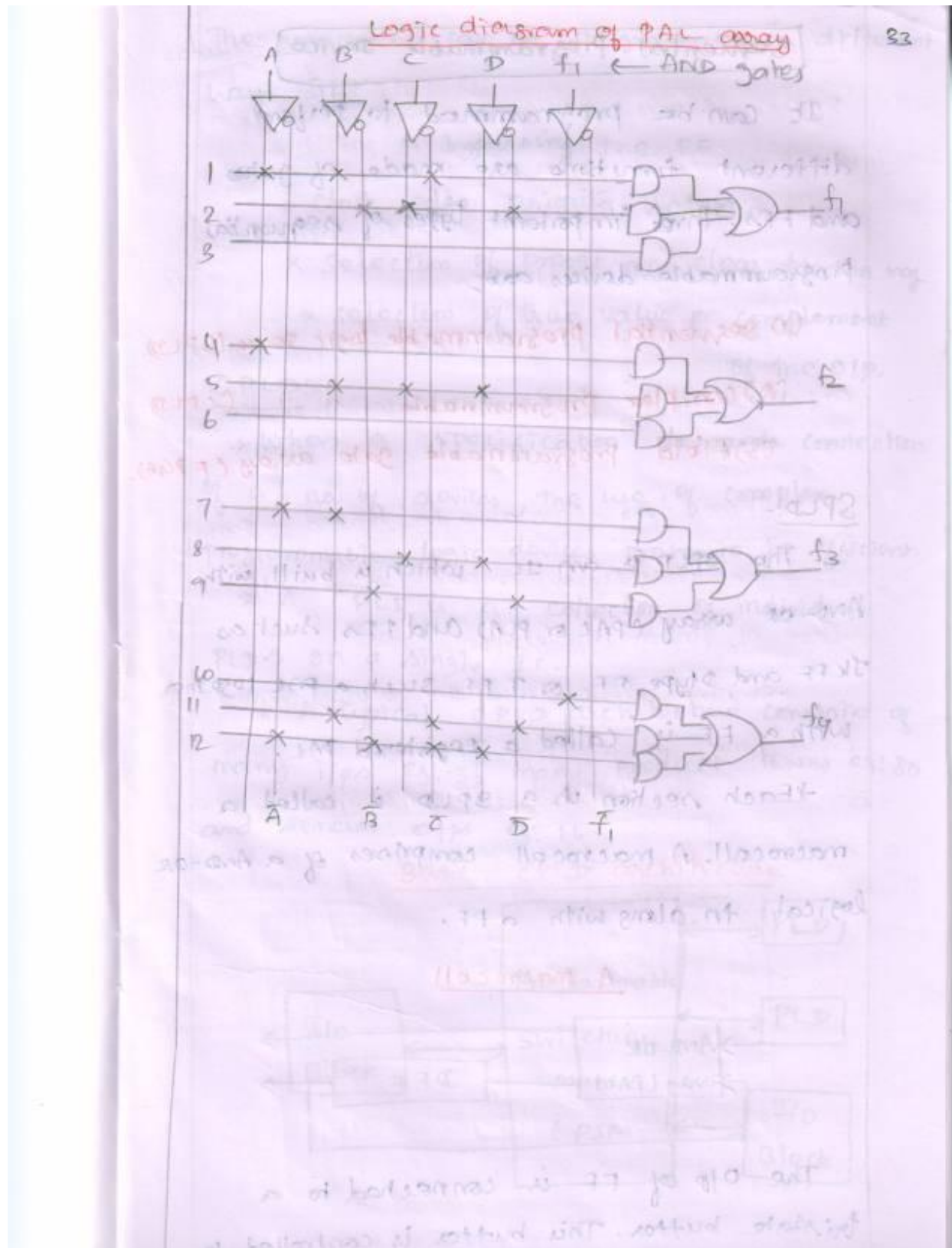
$$f_4 = f_1 + B\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D$$

PAL program table

Product terms.	AND Inputs				f ₁	Outputs
	A	B	C	D		
1	1	1	0	—	—	
2	—	0	1	0	—	$f_1 = AB\bar{C} +$
3	—	—	—	—	—	$\bar{B}C\bar{D}$
4	1	—	—	—	—	
5	—	1	1	1	—	$f_2 = A + BCD$
6	—	—	—	—	—	
7	0	1	—	—	—	
8	—	—	1	1	—	$f_3 = \bar{A}B +$
9	—	0	—	0	—	$CA\bar{B}\bar{D}$
10	—	—	—	—	1	$f_4 = f_1 +$
11	—	1	0	0	—	$B\bar{C}\bar{D} +$
12	0	0	0	1	—	$A\bar{B}C\bar{D}$







Sequential Programmable Device

It can be programmed to perform different functions are made of gates and FFs. Three important types of sequential programmable devices are,

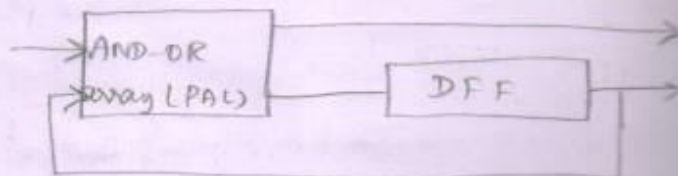
- (1) sequential programmable logic device
- (2) complex programmable
- (3) field programmable gate array

SPLD:-

The SPLD is an IC which is built with AND-OR array (PAL or PLA) and FFs. Such as JK FF and D-type FF or T FF. Such a PAL with a FF is called a registered PAL.

*Each section in a SPLD is called a macrocell. A macrocell comprises of a logical fn along with a FF.

A macrocell



The O/p of FF is connected to a tristate buffer. This buffer is controlled

Sequential Programmable Device

It can be programmed to perform different functions are made of gates and FFs. Three important types of sequential programmable devices are,

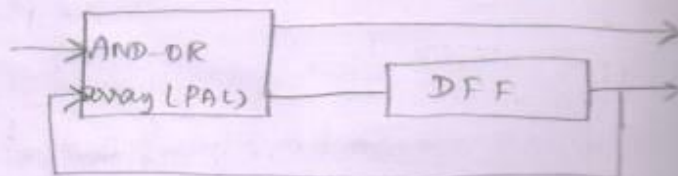
- (1) sequential programmable logic device
- (2) complex programmable
- (3) field programmable gate array

SPLD:-

The SPLD is an IC which is built with AND-OR array (PAL or PLA) and FFs. Such as JK FF and D-type FF or T FF. Such a PAL with a FF is called a registered PAL.

*Each section in a SPLD is called a macrocell. A macrocell comprises of a logical fn along with a FF.

A macrocell



The O/p of FF is connected to a tristate buffer. This buffer is controlled

The macrocell can be programmed in different ways like

- * Using or bypassing the FF
- * Clock edge priority selection
- * Selection of preset and clear for the reg.
- * selection of true value or complement of the o/p.

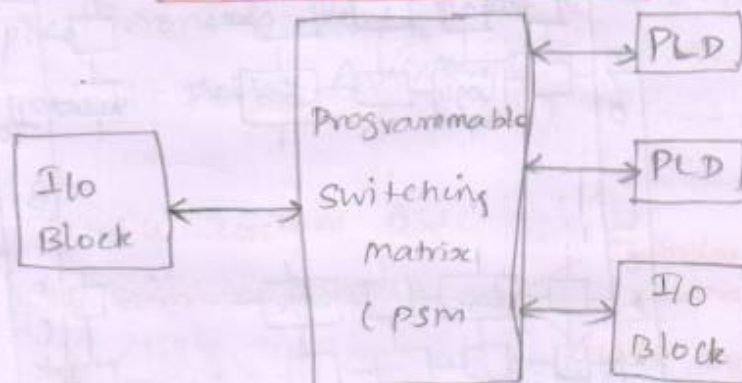
CPLD:-

* When a specification demands connection of a no. of devices, the use of complex programmable logic devices proves to be efficient.

* A CPLD is a collection of individual PLDs on a single IC.

* A typical CPLD architecture consists of many i/p's eg: 36 many product terms eg: 80 and several o/p's eg: 16

Basic CPLD architecture



The macrocell can be programmed in different ways like

- * Using or bypassing the FF
- * Clock edge priority selection
- * Selection of preset and clear for the reg.
- * selection of true value or complement of the o/p.

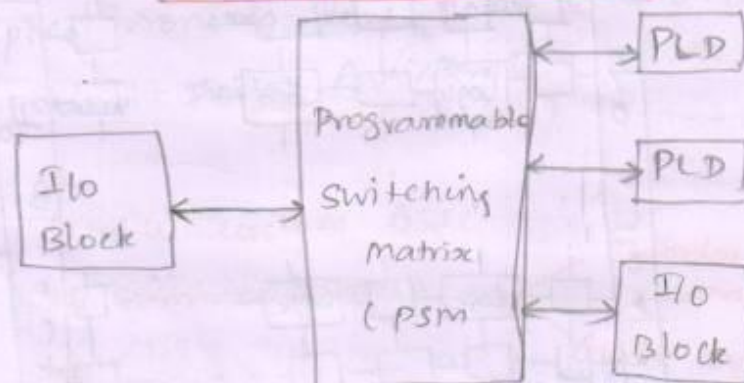
CPLD:-

* When a specification demands connection of a no. of devices, the use of complex programmable logic devices proves to be efficient.

* A CPLD is a collection of individual PLDs on a single IC.

* A typical CPLD architecture consists of many i/p's eg: 36 many product terms eg: 80 and several o/p's eg: 16

Basic CPLD architecture



FPGA: It consists of an array of hundreds of logic blocks and I/O blocks which are programmed to be interconnected.

* A FPGA block consists of truth table multiplexers, gates and FFs.

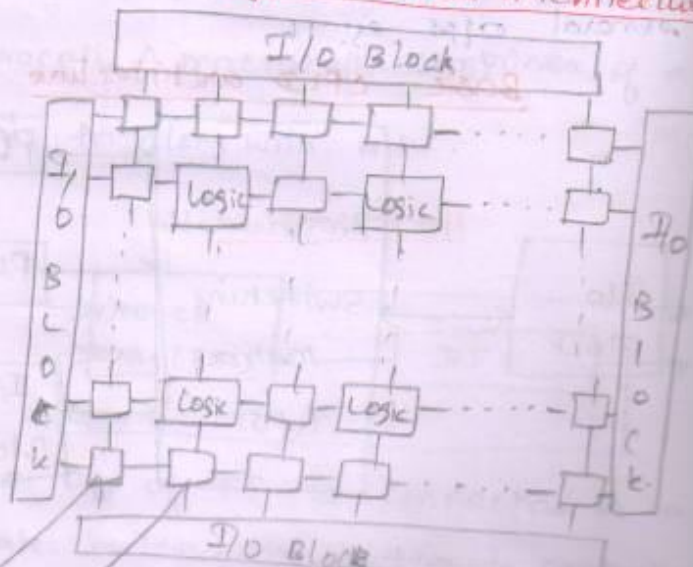
* The function of the combinational circuit explained in the truth table which is stored in the SRAM.

* These functions can also be stored in ROM.

* The memory is temporary and is lost in case of power interruption.

* So, the device must be reprogrammed every time it is turned on.

Typical FPGA Architecture



Xilinx FPGA:

FPGA was built with XC3000 and XC4000 following which Spartan and Virtex families came into market with improved density, performance, power consumption, voltage levels, pin counts and functionality. The Xilinx Spartan is made up of an array of configurable logic blocks (CLB), I/O blocks (IOB) and a memory.

Application specific Integrated Circuits

- * It is an integrated circuit that is developed to satisfy a particular function
- * ASIC finds application in auto emission control, battery management for household appliances, sensitive photo transistors, and optical sensors, low noise audio circuit and personal digital Assistants.

Types:-

- (1) Full custom ASIC
 - (2) Semi custom ASIC
 - (3) Gate array based.
- Standard cell based

Full-custom ASIC:-

* By specifying the layout of the and the interconnections between them.

* In a full custom ASIC all the layers are predefined and customized for a specific application.

* This type of design is used only if ASIC technology is new and there are existing cell libraries available.

* microprocessors, sensors are some of the applications where full custom ASICs are used.

Standard Cell based ASIC:-

* Predesigned, pretested logic cells are used.

* These logic cells which are made of AND gates, OR gates, multiplexers and FFs are known as standard cells.

* The rows of standard cells are known as a standard-cell area or a flexible block.

* It is known as megacells.

* The standard cells can be placed anywhere on the silicon wafer and it is enough if the position and the

39

A cell based ASIC die

The diagram illustrates a cell-based ASIC die layout. It consists of a grid of blocks. Block 1 is a long horizontal rectangle at the top, labeled 'Flexible Blocks'. Below it are blocks 2 and 3, which are small squares labeled 'Fixed blocks'. Further down are blocks 4 and 5, which are larger squares. The entire grid is surrounded by a border of small circles, labeled 'Bonding pads'.

- * A RTL design which provides the description of the ASIC is developed using an sign is then transformed into a large collection of standard cells.
- * Finally, the design rule check is performed to ensure that the device performs well under extreme voltage and temperature.
- * After the checking is done a photo mask is generated which is used for physical fabrication of ASIC.

Gate - Array Based ASIC

- * In this method the transistors and other devices are predefined.
- * The predefined pattern of transistors on a gate array is called the base array and the element which is used in masking, the base array is called

The designer only defines the interconnection between transistors using masks.

This type of gate array is called **mask programmable gate array (MPGA)**.

The wafers on which the transistors are etched are stockpiled.

Since only the interconnections are unique in a MPGA, the stockpiled wafers can be used for different users, as needed.

Three different types of MPGA are,

(1) **channelled Gate Array**

(2) **channelless gate array**

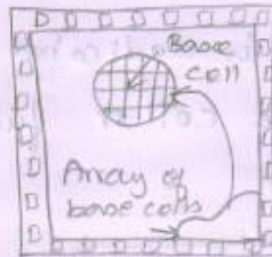
(3) **structured gate array**

channelled Gate array:-



In channelled gate array predefined spaces are used in between the rows of transistors for interconnection.

*It takes couple of weeks to manufacture. as manufacturing is done in a channelless Gate array.



*This is also known as channel-free gate array or sea-of-gates (SoG) array.

*There are no predefined spaces b/w the rows for interconnection or wiring.

*Wiring is done using rows of unused transistors.

*The amount of logic that can be implemented in a given silicon wafer is high in a channelless gate array compared to the channelled gate array. It takes a

couple of days to a couple of weeks

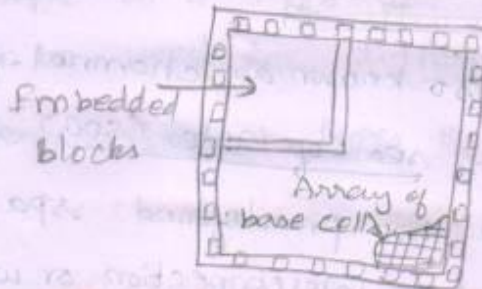
to manufacture a channelless gate array

ASIC.

Structured Gate Array:-

* It is also known as embedded array (or) master slice (or) master image

* In a structured gate array, some of the IC area is dedicated to either a different base cell or full custom IC



* A couple of days to a couple of weeks are needed to manufacture a structured gate array-ASIC.

* Improved area efficiency and increased performance as seen in a CBIC and lower cost and faster turnaround as seen in MGA are offered in a structured gate array ASIC.

P. Balaji