Prashanta Khanal

3 copy.

Chapter 4

Prepared by
Buddha Laxmi Malu

Simplification of Boolean Functions

Boolean Function

Boolean function is an expression formed with binary variables, the two operators OR and AND, the unary operator Not, parentheses, and equal sign for a given value of the variable. The function can be either 0 or 1.

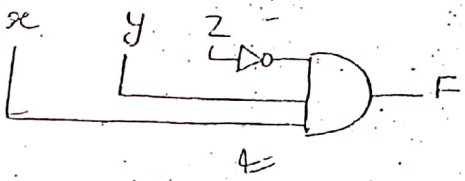
Examples

$F_1 = xyz'$. The function is equal to 1 if $x = 1$ and $y = 1$ and $z' = 1$, otherwise $F_1 = 0$.

Truth table

x	y	z	z'	$F_1 = xyz'$
0	0	0	1	0
0	0	1	0	0
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	0	0
1	1	0	1	1
1	1	1	0	0

Gate Implementation



$F = xy + x'z + yz = xy + x'z + yz(x + x')$

$= xy + x'z + xyz + x'yz$

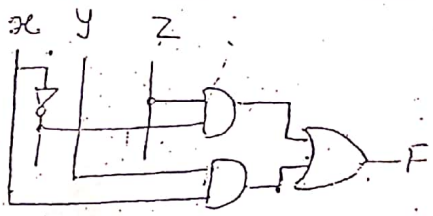
$= xy(1 + z) + x'z(1 + y)$

$= xy + x'z$

Truth table

x	y	z	xy	x'	x'z	xy + x'z
0	0	0	0	1	0	0
0	0	1	0	1	1	1
0	1	0	0	1	0	0
0	1	1	0	1	1	1
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	1	0	0	1
1	1	1	1	0	0	1

Gate Implementation



Complement of a Function

The complement of a function F is F' or $\overline{F}$ and is obt
from an interchange of 0's for 1's and 1's for 0's in
value of F. The complement of a function may be de
algebraically through De Morgan's Theorem.

Examples

$F_1 = x'yz' + x'y'z'$

Complement of $F_1 = \overline{F_1}$

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

$= (x'yz')' \cdot (x'y'z')'$

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

(42)

$F_2 = x(y'z' + yz)$

$F_2' = [x(y'z' + yz)]'$

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

$= x' + (y'z')' \cdot (yz)'$

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

minterm का 0 का (') लेते हैं
maxterm का 1 का (') लेते हैं

Minterms and Maxterms :-

			AND Minterm (Σ)		(OR) Maxterm (π)	
x	y	z	term	Designation	term	Designation
0	0	0	$x'y'z'$	m_0	$x+y+z$	M_0
0	0	1	$x'y'z$	m_1	$x+y+z'$	M_1
0	1	0	$x'yz'$	m_2	$x+y'+z$	M_2
0	1	1	$x'yz$	m_3	$x+y'+z'$	M_3
1	0	0	$xy'z'$	m_4	$x'+y+z$	M_4
1	0	1	$xy'z$	m_5	$x'+y+z'$	M_5
1	1	0	xyz'	m_6	$x'+y'+z$	M_6
1	1	1	xyz	m_7	$x'+y'+z'$	M_7

$F = xy + x'z$

$= (xy + x')(xy + z)$

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

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

$(x' + y) = [(x' + y) + zz'] = \{(x' + y) + z\}(x' + y + z')$

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

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

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

$\therefore F = (x' + y + z)(x' + y + z')(x + y + z)(x + y' + z)$

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

$F = M_0 \cdot M_2 \cdot M_4 \cdot M_5$

$F(x, y, z) = \Pi\,(0, 2, 4, 5)$

3 variables $\therefore$ total combination $2^n = 2^3 = 8$

Among 8 (0, 2, 4, 5) outputs are low and rest all are high, hence rest (1, 3, 6, 7) can be presented as sum of products.

$\therefore F(x, y, z) = \Sigma\,(1, 3, 6, 7)$

Standard form :-

In standard form, the terms that form the function may contain one, two or any number of literals. There are two types of standard forms.

1) Sum of Products (SOP) (Minterm) (1 को 0)

$F_1 = y' + xy + x'y'z'$

$0 \rightarrow A$

$1 \rightarrow A$

2) Product of Sums (POS) (Maxterm) (0 को 1)

$F_2 = x(y' + z)(x' + y + z' + w)$

Non Standard Form:

A Boolean function may be expressed in a non standard form.

$F_3 = (AB+CD)(A'B'+C'D)$.

This function is neither in SOP nor in POS form. It may be changed to a standard form by using distribution law to remove the parenthesis.

$F_3 = ABA'B' + ABC'D + A'B'CD + CDC'D$

$= ABC'D + A'B'CD$

Simplification of Boolean function:

The map method / Karnaugh method / (K-map):

→ The map method is regarded as a pictorial form a truth table.

→ The map is a diagram made up of squares. Each represents one minterms. The map presents a visual diagram of all possible ways that a function may be expressed in a standard form.

Two variable K-map

→ There are four minterms for two variables; hence it consists of four squares, one for each minterm.

x \ y	0	1
0	$x'y'$ m_0	$x'y$ m_1
1	xy' m_2	xy m_3

Three Variable K-map

x \ yz	00	01	11	10
0	000 m_0	001 m_1	011 m_3	010 m_2
1	100 m_4	101 m_5	111 m_7	110 m_6

Four variable K-map

wx \ yz	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

z (01, 11); x (01, 11); w (11, 10); y (11, 10)

Five variable K-map

AB \ CDE	000	001	011	010	110	111	101	100
00	0	1	3	2	6	7	5	4
01	8	9	11	10	14	15	13	12
11	24	25	27	26	30	31	29	28
10	16	17	19	18	22	23	21	20

C (110, 111, 101, 100); A (11, 10); E (001, 011); D (010); E (110, 111)

Example :- $F(x,y,z) = \Sigma(1,3,6)$

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

x \ yz	00	01	11	10
x 0	0	1	1	0
x 1	0	0	0	1

$F = xy'z + x'yz + xyz'$

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

$= x'z + xyz'$ (2)

Simplifying by K-map

$F(x,y,z) = x'z + xyz'$

(2) $F(x,y,z) = \Sigma(1,3,7)$ $\quad 2^3 = 8$

x \ yz	00	01	11	10
x 0	0	1	1	0
x 1	0	0	1	0

$F(x,y,z) = x'z + yz$

(3) $F(x,y,z) = \Sigma(1,3,5,7)$

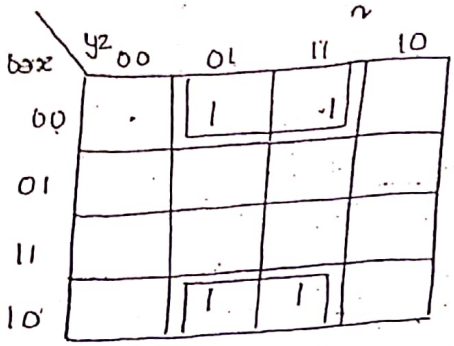
x \ yz	00	01	11	10
x 0	0	1	1	0
x 1	0	1	1	0

$F(x,y,z) = z$

Note :- which variable is constant, take that one only.

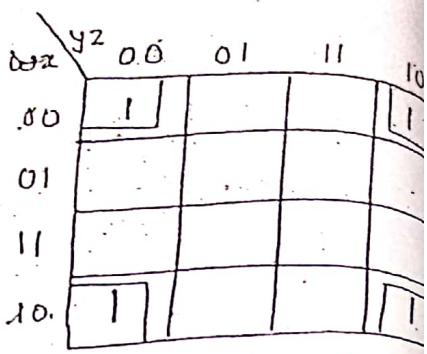
Example: Simplify the Boolean expression using K-

$F(w,x,y,z) = \Sigma(1,3,9,11)$



$\therefore F(w,x,y,z) = x'z'$

$F(w,x,y,z) = \Sigma(0$

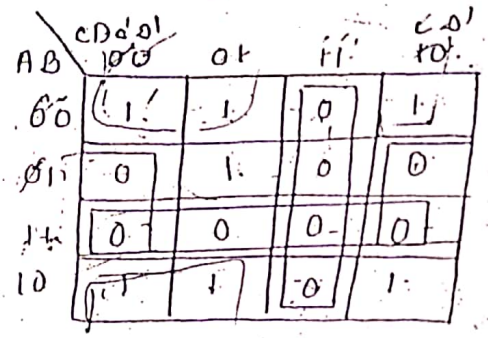


$F(w,x,y,z) = x'z'$

Product of sums simplification

Example 1: simplify the following Boolean function in of sums:

$F(A,B,C,D) = \Sigma(0,1,2,5,8,9,10)$



$F' = CD + AB + BD'$ $F = (C'+D')\cdot(A'+B')\cdot(B'+D)$

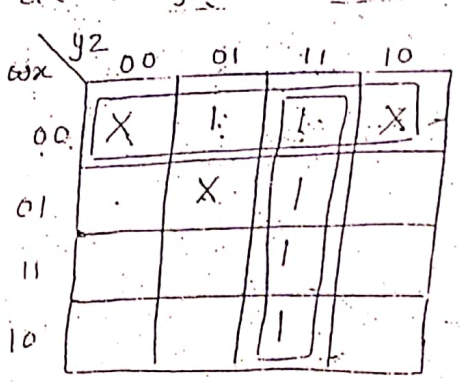
→ take duals.

Don't Care Condition

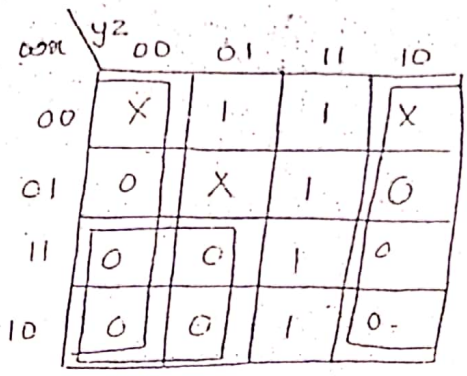
Example 2: Simplify the Boolean function

$F(w,x,y,z) = \Sigma(1,3,7,11,15)$ and dont care condi

$d(w,x,y,z) = \Sigma(0,2,5)$



$F = w'x' + yz$

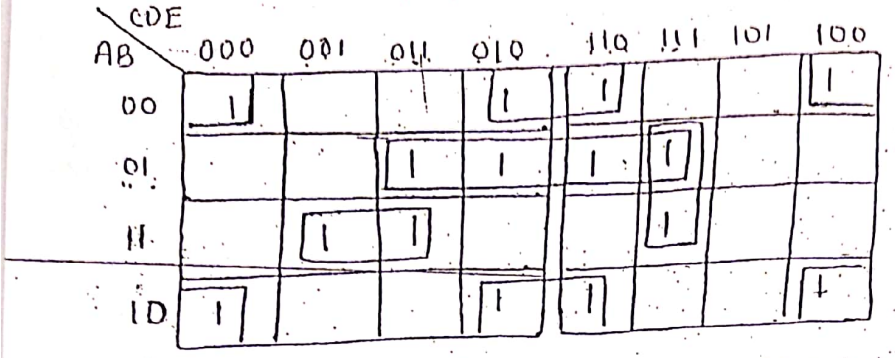


$F = z' + wy'$

→ Don't care condition is represented by a symbol 'X'.
→ It can take either value of 0 or 1 as the need arises.

Example 3: Simplify the Boolean expression

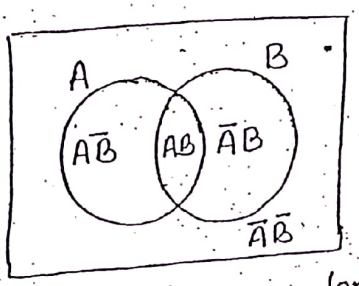
$f(A,B,C,D,E) = \Sigma(0,2,4,6,10,11,14,15,16,18,20,22,25,27,3$



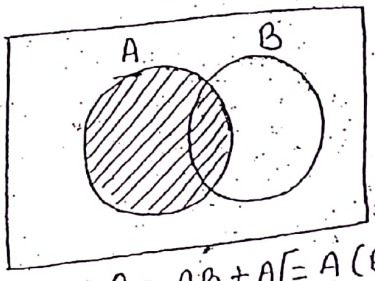
$F = A'BD + B'D'E' + ABC'E + B'DE' + BCDE$

8421

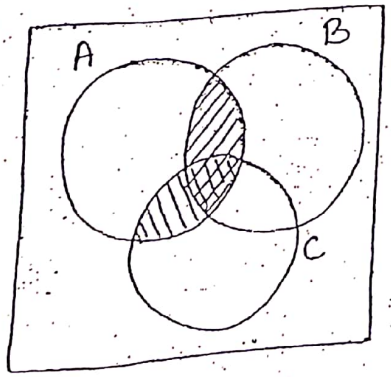
VENN DIAGRAM



Venn diagram for two variables



$A = AB + A[= A(B+1)]$
A



A (B+C)
AB+AC

8421

Prepared by
Buddha Laxmi

Chapter 5
Combinational Logic

Combinational logic

A combinational ckt consists of logic gates w
outputs at any time are determined directly f
the present combination of inputs without rega
to previous inputs. Eg. Adders, Subtractors, Multiple

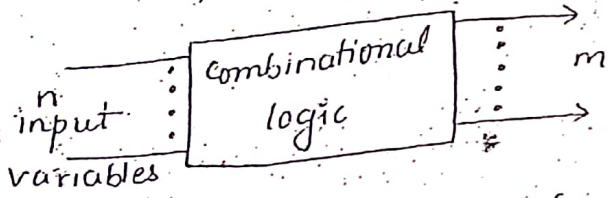


Fig: Block Diagram of combinational ckt.

Sequential logic

Sequential ckt consists of memory element in ad
to logic gates. The outputs of a sequential ckt dep
on present inputs and past inputs. Eg Flip-flop, re
counters. etc.

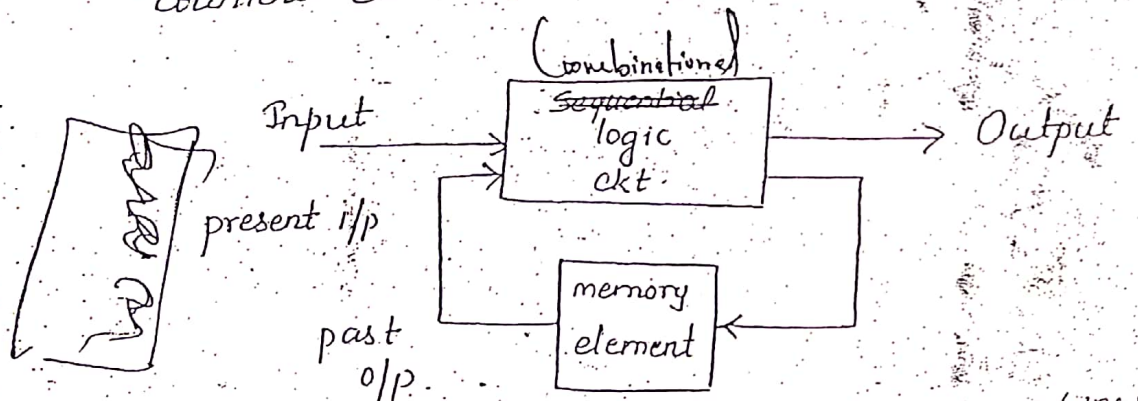


Fig: Block Diagram of a ~~sequential~~ combinational ckt.

Design Procedure :- Combinational ckt

Steps:

i) The problem is stated.
ii) The numbers of available input variables and req
output variables is determined.
iii) The input and output variables are assigned letter sy

(iv) The truth table that defines the required relationship between inputs and outputs is determined.
(v) The simplified Boolean function for each output is obtained
(vi) The logic diagrams is drawn

A practical design method would have to consider such constrains as

1) Minimum numbers of gates.
2) Minimum numbers of inputs to gate.
3) Minimum propagation time of the signal through the ckt.
4) Minimum number of interconnections.
5) Limitations of the driving capability of each gate.

Half Adder

A combinational ckt that performs the addition of two bits is called a half-adder.

Step 1

0+0 = 00
0+1 = 01
1+0 = 01
1+1 = 10

Step 2:

We find this ckt needs two binary inputs and two binary outputs.

Step 3

We are obtaining assign symbols x & y to the two inputs and S (for sum) & C (for carry) to outputs.

Step 4

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The s output represents the least significant bit of the sum.

step 5

K-map for s

x \ y	0	1
0	0	1
1	1	0

$S = xy' + x'y$

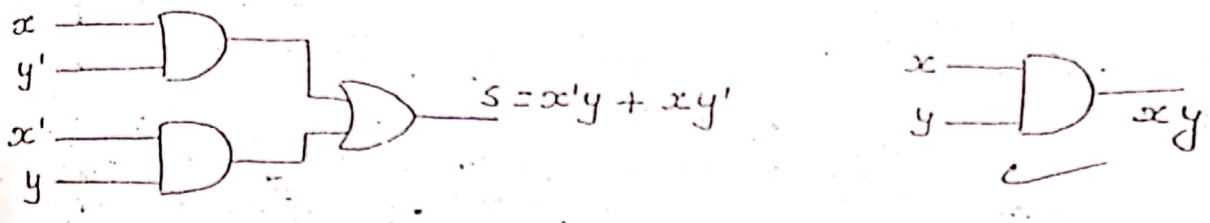
$= x \oplus y$

K-map for c

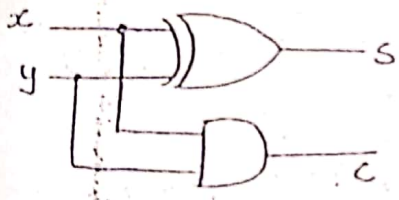
x \ y	0	1
0	0	0
1	0	1

$C = xy$

step 6



step : 7



Here, logic diagram is further simplified by using XOR gate.

Full Adder

A full-adder is a combinational ckt that forms the arithmetic sum of three inputs. It consists of 3 inputs and two outputs.

truth table

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Step 5

k-map for S

x \ yz	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$S = x'y'z + x'yz' + xy'z' + xyz$

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

$= z(x \odot y) + z'(x \oplus y) = z(x \oplus y)' + z'(x \oplus y)$

$= x \oplus y \oplus z$

k-map for C

x \ yz	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$C = xz + xy + y'z$ (not necessary)

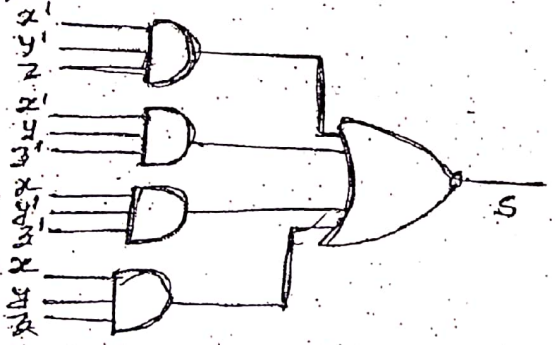
$= xyz + xy'z + xyz + xyz' + xyz + x'yz$

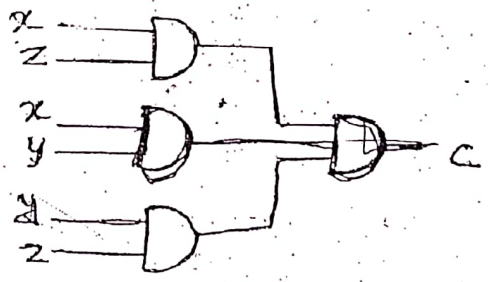
$= xy + z(x \oplus y)$

$= xyz + xyz' + xy'z + x'yz$

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

Step 6 →

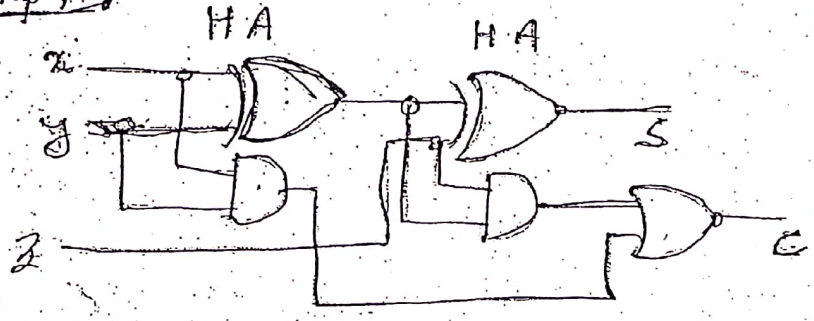




$xz + xy + yz$

$= xy + z(x + z)$

Step 7 →



Half Subtractor

A half-subtractor is a combinational ckt that subtracts two bits and produces their difference. The half subtractor needs two inputs and two outputs.

Truth table

inputs		outputs	
x	y	B	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

D → difference
B → Borrow

K-map for D

x \ y	0	1
0	0	1
1	1	0

$D = xy' + x'y$

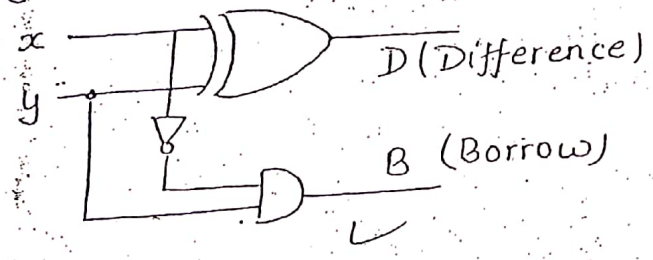
$= x \oplus y$

K-map for B

x \ y	0	1
0	0	1
1	0	0

$B = x'y$

Logic implementation



Full Subtractor

A full subtractor is a combinational ckt that performs subtraction between two bits, taking into account that a 1 may have been borrowed by a lower significant stage.

This ckt has three inputs and two outputs.

13 (x-y)-z

step 4

Truth table

x	y	z	B	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

step 5

K-map for D

$D = \Sigma(m_1, m_2, m_4, m_7)$

x \ yz	00	01	11	10
0		1	1	1
1	1		1	

$\therefore D = x'y'z + x'yz' + xy'z' + xyz$

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

$= x'(y \oplus z) + x(y \odot z)$

$= x'(y \oplus z) + x(y \oplus z)'$

$= x \oplus y \oplus z$

$(y \odot z) \Rightarrow (y \oplus z)'$

K-map for B

$B = \Sigma(m_1, m_2, m_3, m_7)$

x \ yz	00	01	11	10
0		1	1	1
1			1	

$B = x'z + x'y + yz$

$= x'yz + x'y'z + x'yz + x'yz' + yz$

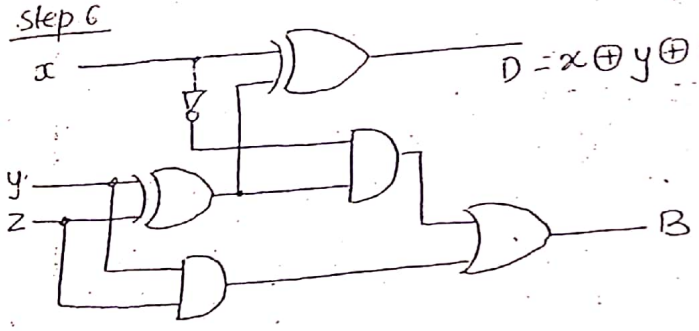
$= x'yz + x'y'z + x'yz' + yz$

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

$= x' \cdot (y \oplus z) + yz(x'+1)$

$= x' \cdot (y \oplus z) + yz$

step 6



$x - y - z$

$= x - (y+z)$ — HA

$= x - (\omega)$ — HS

Ques

BCD To Excess-3 Code Conversion:

Design a ckt to convert BCD code to excess-3 output.

step: 4

Truth table

	Input BCD				Output Excess 3 code			
	A	B	C	D	ω	x	y	z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

$d = \Sigma(10, 11, 12, 13, 14, 15)$

K-map for ω

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

$\omega = \Sigma(5, 6, 7, 8, 9)$

$\omega = A + BD + BC$

K-map for x

$x = \Sigma(1, 2, 3, 4, 9)$

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

$x = BC'D' + B'D + B'C$

K-map for y

$y = \Sigma(0, 3, 4, 7, 8)$

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

$y = C'D' + CD$

K-map for z

$z = \Sigma(0, 2, 4, 6, 8)$

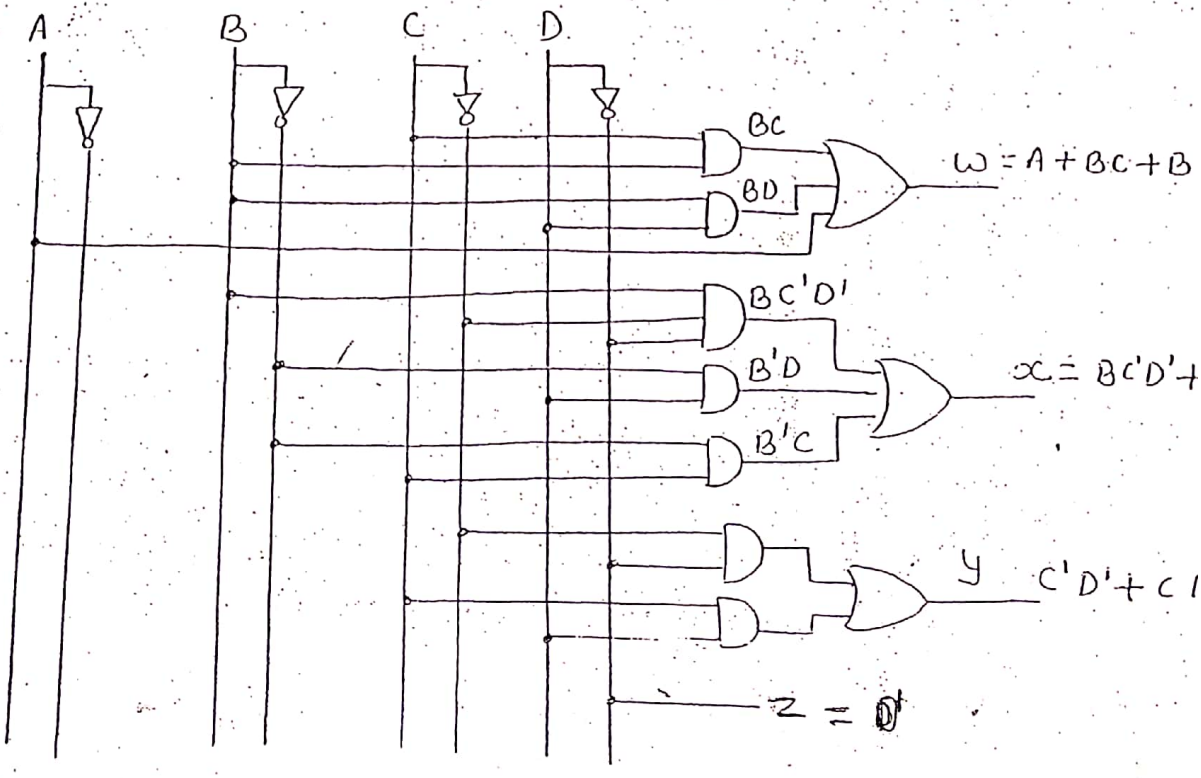
AB \ CD	00	01	11	10
00	1			1
01	1			1
11	X	X	X	X
10	1		X	X

$z = C'D' + CD'$

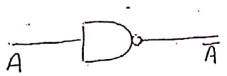
$= (C' + C)\, D'$

$= D'$

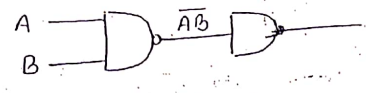
Logic Implementation



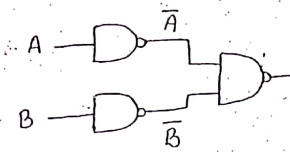
Multilevel NAND Circuit



NOT (inverter)



AND



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

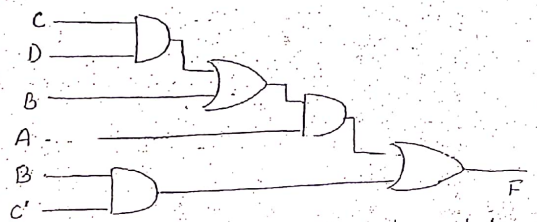
Boolean Function Implementation

Steps

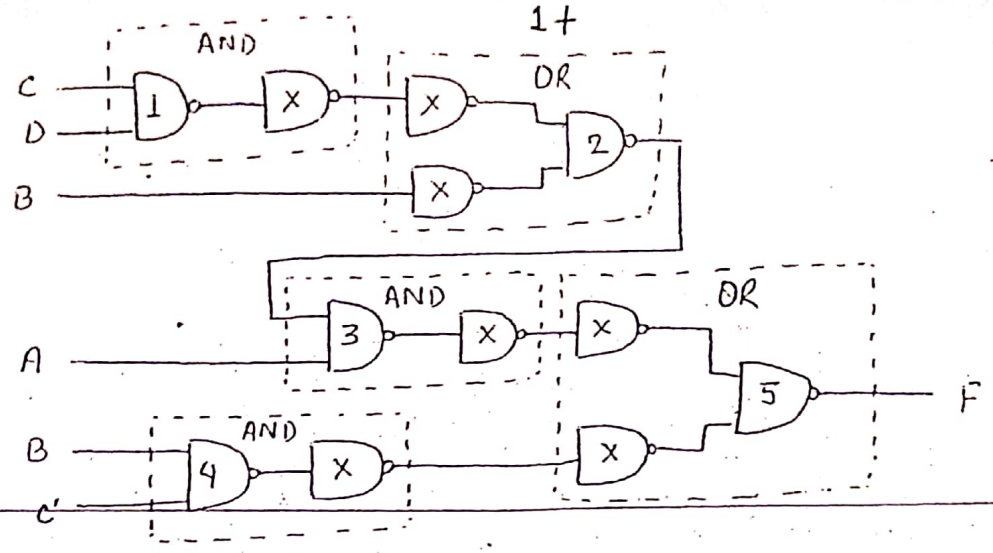
1. For the given algebraic expression, draw the lo diagram with AND, OR, and NOT gates. Assume the normal and complement i/ps are available.
2. Draw a second logic diagram with the equi NAND logic, as given in above figures.
3. Remove any two cascaded inverters, from the since double inversion does not perform a logic fu Remove inverters connected to single external i/ps and the corresponding i/p variable. The new logic diagram required NAND gate implementation.

Example

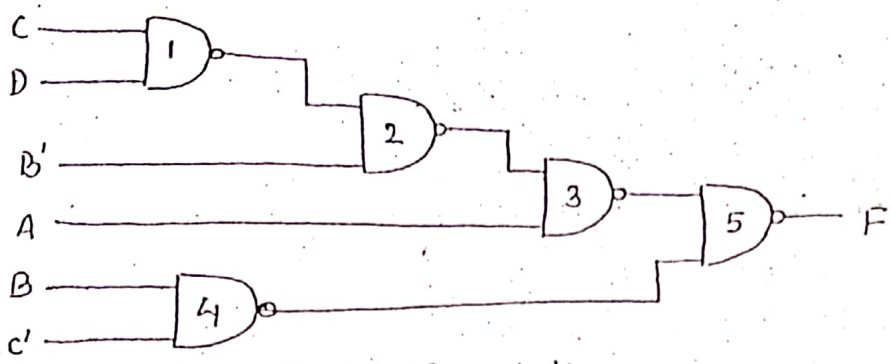
1) $F = A(B + CD) + BC'$



a) AND/OR implementation

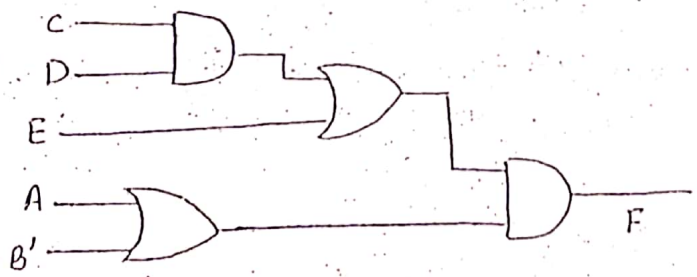


(b) substituting eq. NAND functions form

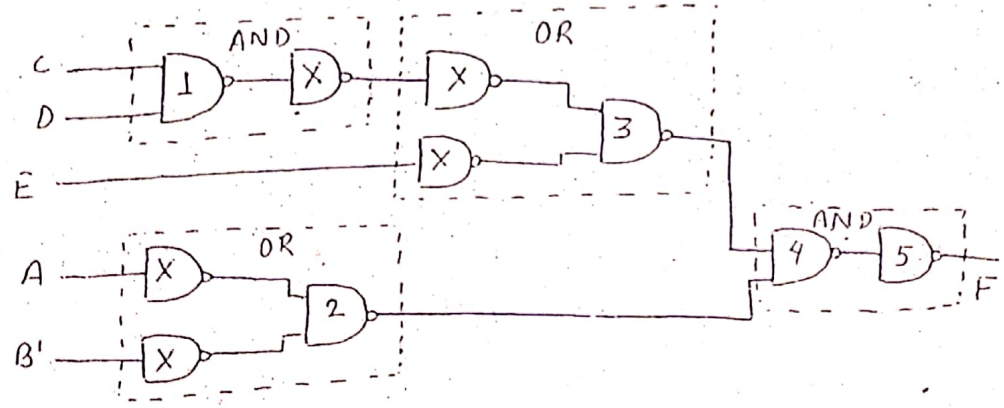


(c) NAND implementation

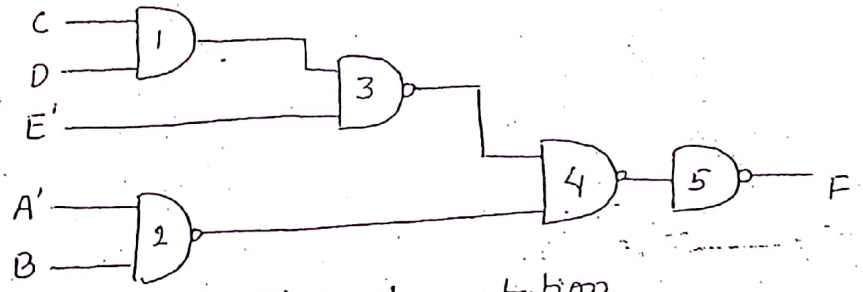
$F = (A+B')(CD+E)$



a) AND/OR implementation

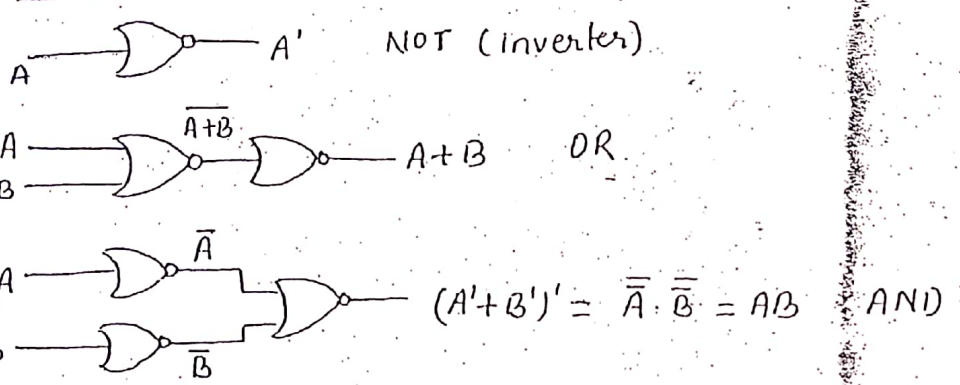


b) Substituting eq NAND functions



c) NAND implementation

Multilevel NOR Circuit

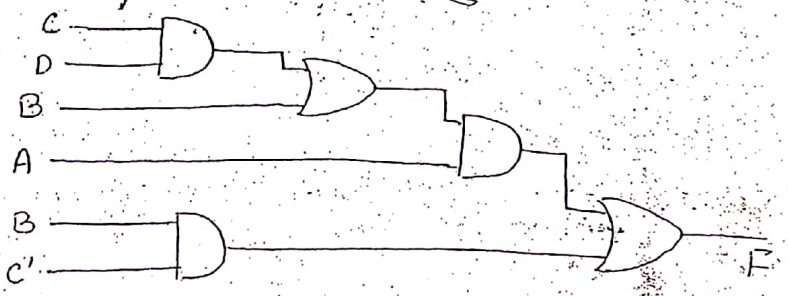


Boolean Function Implementation

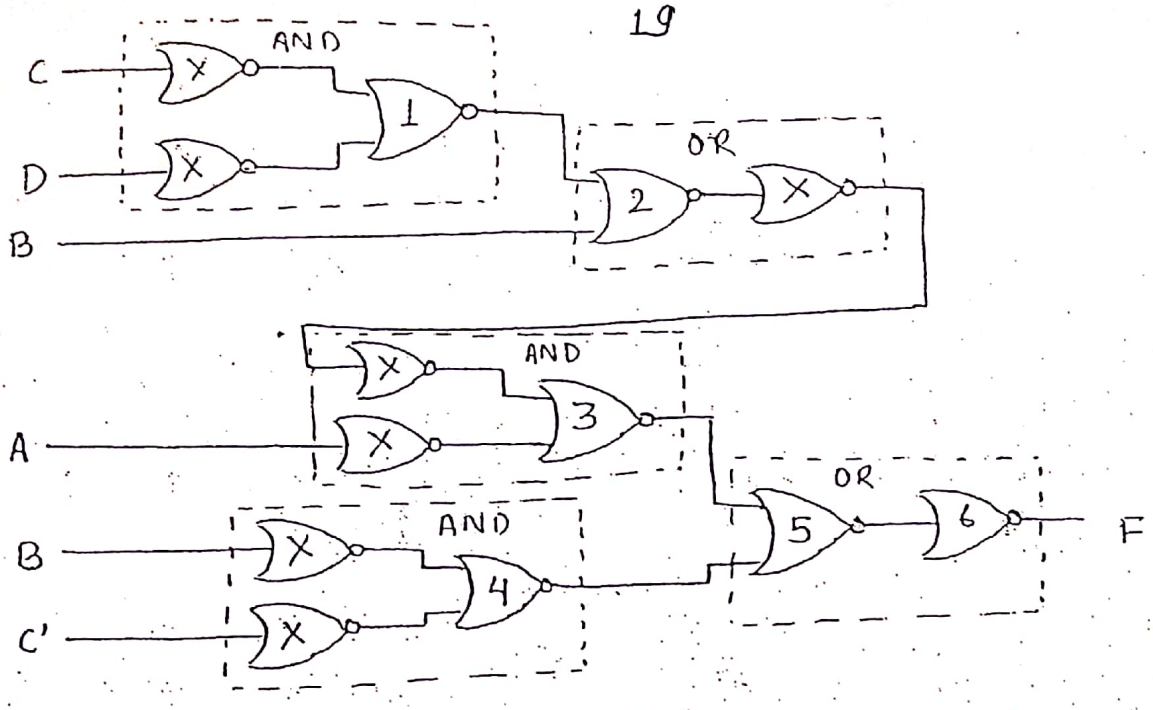
Steps

1. Draw the AND-OR logic diagram from the given alg expression. Assume that both the normal and the complement I/ps are available.
2. Draw a second logic diagram with q. NOR logic, given in Figure, substituted for each AND, OR and gate.
3. Remove pairs of cascaded inverters from the diagram. Remove inverters connected to single external i/ps and complement the corresponding i/p variable.

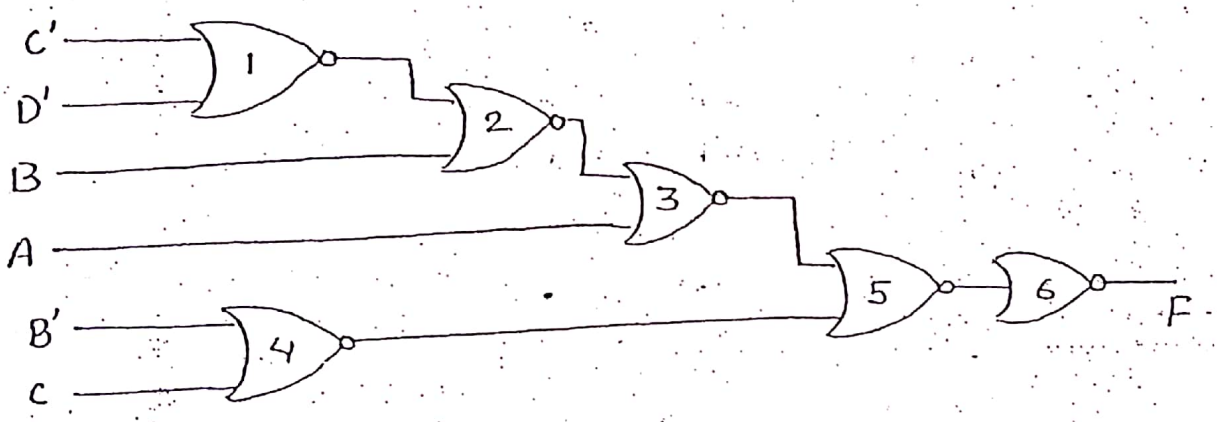
Example $F = A(B+CD) + BC'$



d) AND/OR implementation



(b) substituting eq. NOR functions.



c) NOR Implementation

Analysis Procedure

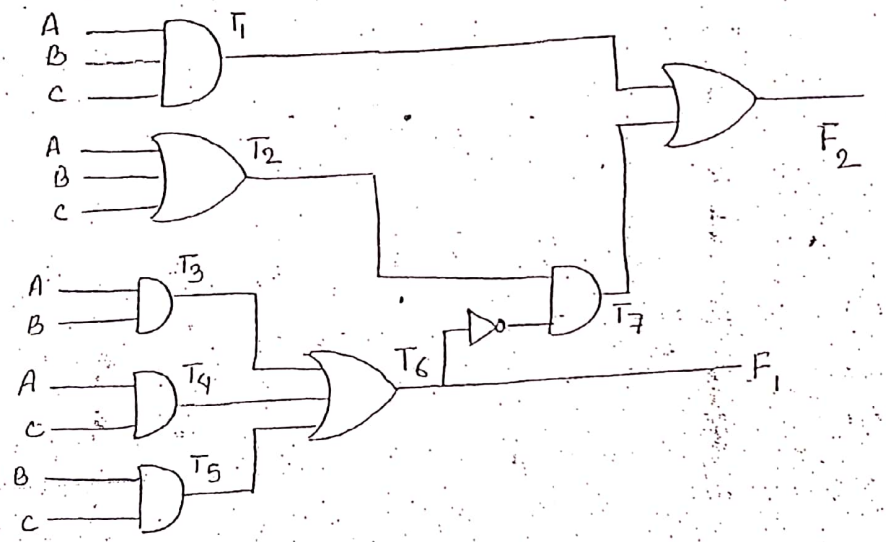
→ Deriving a Boolean function or a truth table fro logic ckt diagram is an analysis procedure.

Procedure

1. Label with arbitrary symbols for all gate outputs are a function of the input variables. obtain b function for each gate.
2. Label with other arbitrary symbols. Those gates wh a function of i/p variables and previously labelled
3. Obtain o/p function in terms of i/p variables on

Example

Analyse the ckt given below



$T_1 = ABC$ $T_2 = (A+B+C)$ $T_3 = (AB)$

$T_4 = AC$ $T_5 = BC$

$T_6 = T_3 + T_4 + T_5 = AB + AC + BC$

$\boxed{F_1 = T_6 = AB+AC+BC}$

$T_7 = T_2 \overline{T_6}' = (A+B+C)(AB+AC+BC)'$

$F_2 = T_1 + T_7 = ABC + (A+B+C) \cdot (AB+AC+BC)'$

$= ABC + (A+B+C)\ (\overline{AB} \cdot \overline{AC} \cdot \overline{BC})$

$= ABC + (A+B+C)\ (A'+B')\ (A'+C')(B'+C')$

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

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

$F_2 = ABC + AB'C' + A'BC' + A'B'C$

Truth table

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

Parity generation and checking

Parity bit:

A parity bit is an extra bit included with a binary message to make the number of 1's either odd or even so we have two types of parity.

i) odd parity ⇒

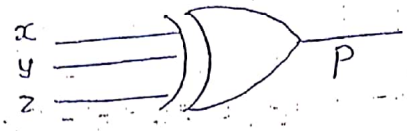
ii) Even parity ⇒

Even-parity Generator

Three-bit message			Parity bit
x	y	z	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

For even-parity generator, $P = 1$ when no. of 1's
$P = 0$ " " " 1's

Hence it is an Exclusive-OR Gate (XOR Ga



OR

3-bit even parity generator

It can be verified by K-map as well

x \ yz	00	01	11	10
0		1		1
1	1		1	

$P = x'y'z + x'yz' + xy'z' + xyz$

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

$= x'(y \oplus z) + x(y \odot z)$

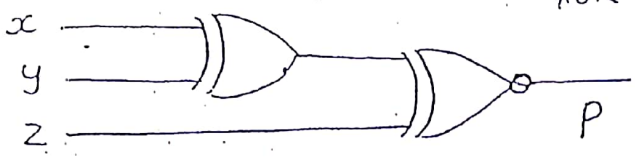
$= x \oplus y \oplus z$

Odd Parity Generator:

Three-bit message			Parity bit
x	y	z	P
0	0	0	1
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	0

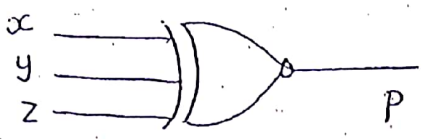
For odd parity generator $P = 1$ when no. of 1's is even
$P = 0$ " " " 1's " odd

Hence odd parity is obtained through complement of either XOR or an equivalence expression



$x \oplus y \odot z$

OR

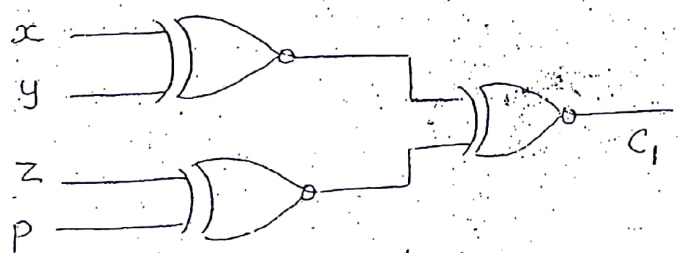


$(x \oplus y \oplus z)' = A \oplus B \odot C$

3-bit odd parity generator

Parity Checker

Four bit received				odd parity checker	even parity checker
x	y	z	P	C_1	C_2
0	0	0	0	1	0
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	0	1
1	0	0	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	0	1
1	1	1	0	0	1
1	1	1	1	1	0



odd parity checker

K- map for odd parity generator

x \ yz	00	01	11	10
0	1		1	
1		1		1

$P = x'y'z' + x'yz + xy'z + xyz'$

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

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

$= x \oplus (y \odot z)$

$= (x \oplus y) \odot z$

K- map for odd parity checker

xy \ zp	00	01	11	10
00	1		1	
01		1		1
11	1		1	
10		1		1

$P = x'y'z'p' + x'y'zp + x'yz'p + x'yzp' + xyz'p' + xyzp +$
$xy'z'p + xy'zp'$

$= x'y'(z'p' + zp) + x'y(z'p + zp') + xy(z'p' + zp) + xy'(z$

$= x'y'(z \odot p) + x'y(z \oplus p) + xy(z \odot p) + xy'(z \oplus p)$

$= x'[y'(z \odot p) + y(z \oplus p)] + x[y(z \odot p) + y'(z \oplus p)]$
$\bar{a}\ \bar{b}\qquad a\ b\qquad\qquad a\ b\qquad \bar{a}\ \bar{b}$

$= x'[y \odot (z \oplus p)] + x[y \odot (z \odot p)]$
$\bar{a}\ \bar{b}\qquad\qquad a\qquad b$

$= x \odot y \odot z \odot p$

$= (x \odot y) \odot (z \odot p)$

K-map for even parity checker

xy \ zp	00	01	11	10
00		1		1
01	1		1	
11		1		1
10	1		1	

$P = x'y'z'p + x'y'zp' + x'yz'p' + x'yzp + xyz'p + xyzp'$
$+ xy'z'p' + xy'zp$

$= x'y'(z'p + zp') + x'y(z'p' + zp) + xy(z'p + zp')$
$+ xy'(z'p' + zp)$

$= x'y'(z \oplus p) + x'y(z \odot p) + xy(z \oplus p) + xy'(z \odot p)$

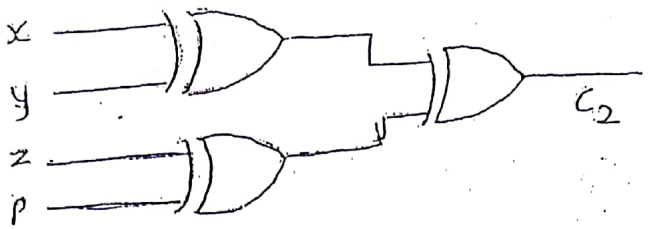
$= x'[y'(z \oplus p) + y(z \odot p)] + x[y(z \oplus p) + y'(z \odot p)]$
$\quad \bar{a}\ b \qquad\qquad a\ \bar{b} \qquad\qquad a\ \bar{b} \qquad\qquad \bar{a}\ b$

$= \underset{\bar{a}}{x'}\underset{b}{[y \oplus (z \oplus p)]} + \underset{a}{x}\underset{\bar{b}}{[y \oplus (z \odot p)]}$

$= x \oplus y \oplus z \oplus p$

$= (x \oplus y) \oplus (z \oplus p)$

$(x \oplus y) \oplus (z \oplus p)$ even
$(x \oplus y) \oplus (z \odot p)$ odd



4-bit even parity checker

Some questions

1. Design a ckt to convert 3 bit binary to gray

Let x, y, z be three input variables for binary
and A, B, C be three o/p variables for gray

x	y	z	A	B	C
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

8421

K-map for A

x \ yz	00	01	11	10
0				
1	1	1	1	1

$A = x$

K-map for B

x \ yz	00	01	11	10
0			1	1
1	1	1		

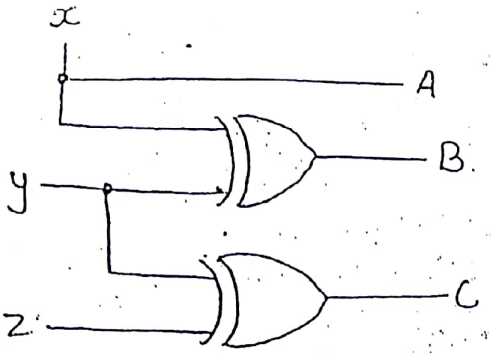
$B = x'y + xy' = x \oplus y$

K-map for C

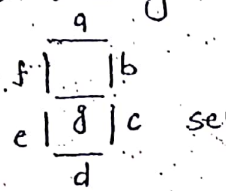
x \ yz	00	01	11	10
0		1		1
1		1		1

$C = y'z + yz' = y \oplus z$

Logic implementation



2) Design the BCD-to-seven-segment decoder ckt.



seven segment display

BCD inputs

	w	x	y	z	a	b	c	d	e	f	g	
0	0	0	0	0	1	1	1	1	1	1	0	0
1	0	0	0	1	0	1	1	0	0	0	0	1
2	0	0	1	0	1	1	0	1	1	0	1	2
3	0	0	1	1	1	1	1	1	0	0	1	3
4	0	1	0	0	0	1	1	0	0	1	1	4
5	0	1	0	1	1	0	1	1	0	1	1	5
6	0	1	1	0	1	0	1	1	1	1	1	6
7	0	1	1	1	1	1	1	0	0	0	0	7
8	1	0	0	0	1	1	1	1	1	1	1	8
9	1	0	0	1	1	1	1	1	0	1	1	9

K map for a

wx \ yz	00	01	11	10
00	1		1	1
01		1	1	1
11	X	X	X	X
10	1	1	X	X

$a = y + w + xz + x'z'$

$= w + y + x \odot z$

K-map for b

wx \ yz	00	01	11	10
00	1	1	1	1
01	1		1	
11	X	X	X	X
10	1	1	X	X

$b = w + x' + y'z' + yz$

$= w + x' + y \odot z$

K-map for c

wx \ yz	00	01	11	10
00	1	1	1	
01	1	1	1	1
11	X	X	X	X
10	1	1	X	X

$c = w + x + y + z$

K-map for d

wx \ yz	00	01	11	10
00	1		1	1
01		1		1
11	X	X	X	X
10	1	1	X	X

$d = w + yz' + w'x'z' + x'y'z$

$= w + yz' + w'x'(z' + y) + x$

K-map for e

wx \ yz	00	01	11	10
00	1			1
01				1
11	X	X	X	X
10	1		X	X

$e = wz' + yz' + w'x'z'$

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

K-map for f

wx \ yz	00	01	11	10
00	1			
01	1	1		1
11	X	X	X	X
10	1	1	X	X

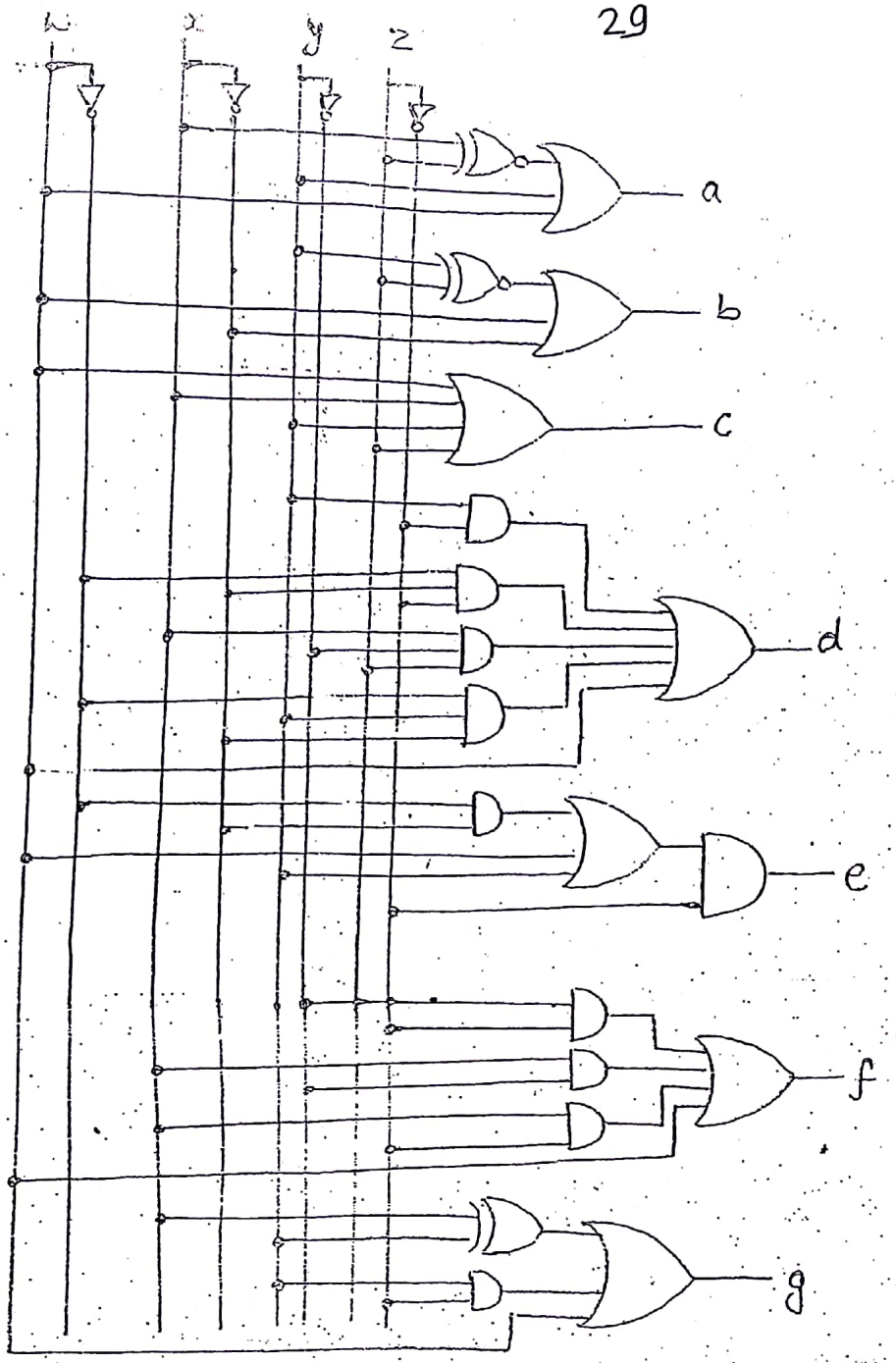
$f = w + y'z' + xy' + xz'$

K-map for g

wx \ yz	00	01	11	10
00			1	1
01	1	1		1
11	X	X	X	X
10	1	1	X	X

$g = w + yz' + x'y + xy'$

$= w + yz' + x \oplus y$



logic implementation

MSI & LSI Components in Combinational logic

Prepared by:
Buddha Laxmi Ma

Combinational logic with MSI & LSI.

MSI → Medium scale Integration
LSI → Large scale Integration

→ The classical procedure assumes that; given two ckts perform the same function, the one that requires fe gates is perferable because of its low cost, is not nec true when it is used.

→ With integrated ckts, it is not the count of gates th dekermines the cost but the number & type of ICs employed & the no. of external interconnections needed to implement the given fn.

→ Before applying the classical method it is always to investigate the possibility of an alternate method may be more efficient for the particular problem.

Binary Parallel Adder:

→ The full adder is the sum of two bits and a previous carry.

→ Two binary numbers of n bits each can be added by the parallel adder.

→ when a pair of bits are added through a full ad the ckt produces a carry that is added to one significant position higher.

Input carry		1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
output carry	0	0	1	1	C_{i+1}

Full adder

Z
X
Y
S
C

→ The bits are added with full adders, starting from the least significant position, to form sum bit & carry bit.

→ The i/p carry C_1 in the least significant position must be 0. The value of C_{i+1} in a given significant position is the o/p carry of the full adder.

→ The value is transferred into the i/p carry of full adder i.e. one significant position higher to the left.

→ Thus the sum of bits are generated from right to left as soon as previous carry bit is generated.

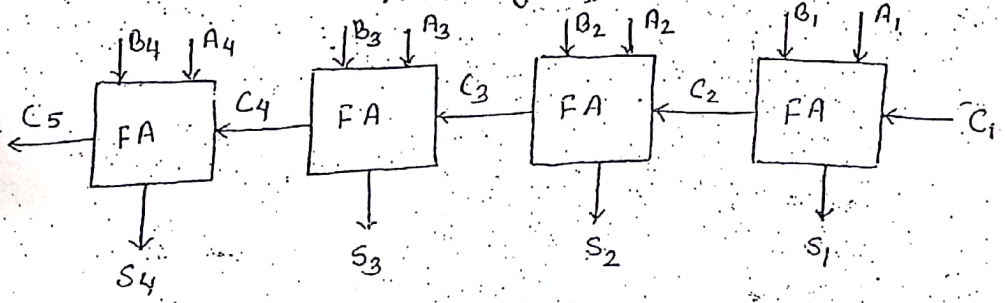
→ The sum of bits are generated in two ways.

{ → serial
→ Parallel }

→ The serial addition method uses only one full adder ckt and a storage device to hold the generated output carry. The pair of bits in A & B are transferred serially one at a time, through the single full adder to produce a string of output bits.

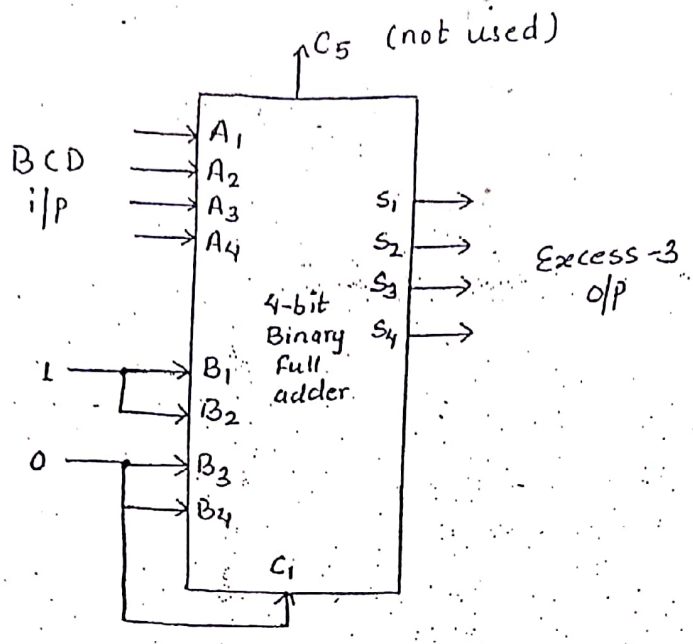
→ The parallel method uses n full adder ckts & all bits of A & B are applied simultaneously. The o/p carry of one full adder is connected to the input carry of next full adder at one significant position higher.

→ A binary parallel adder is a digital function that produces the arithmetic sum of two binary numbers in parallel. It consists of full adder connected in cascade. With the output carry from one full adder connected to the i/p carry of the next full adder.



4-bit binary Parallel adder

Design a BCD to excess-3 code converter



BCD-to-excess-3 code converter

→ The BCD digit is applied to input A. Input B are set to a constant 0011. This is done by applying logic 1 to B_1 & B_2 & logic 0 to B_3 & B_4 and also in C_1.

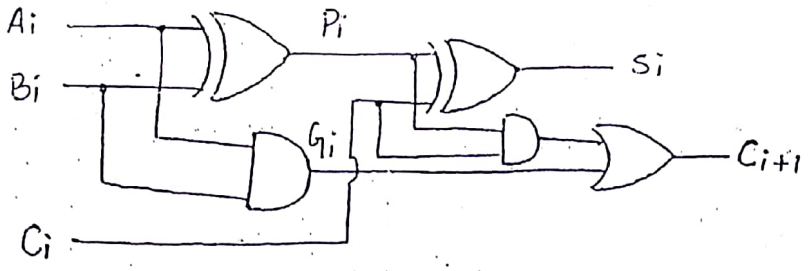
→ The S outputs from the ckt give the excess-3 equivalent code of the input BCD digit.

Carry Propagation

→ The signal must propagate through the gates before the correct output sum is available in the o/p terminals.

→ The total propagation time is equal to the propagation delay of a typical gate times the no. of gate levels in the ckt.

→ The longest propagation delay time in a parallel adder is the time it takes the carry to propagate through the full adders.



Full Adder Circuit

→ If there are four full adders in the parallel adder, the output carry C_5 would have $2 \times 4 = 8$ gate levels (AND & OR) from C_1 to C_5.

→ The total propagation time would be the propagation time of one half adder + 8 gate level.

→ There are several technique to reduce the carry propagation time in a parallel adder. The most widely used technique employs the principle of look-ahead carry.

$$P_i = A_i \oplus B_i$$

$$G_i = A_i B_i$$

The output sum & carry

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_i C_i$$

→ Here G_i is called a carry generate & it produces an output carry when both A_i & B_i are one, regardless of the output carry.

→ P_i is called carry propagate because it is the term associated with the propagation of the carry from C_i to C_{i+1}.

Now, Boolean function for the carry output of each stage for 4 bit adder,

$$C_2 = G_1 + P_1 C_1$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 (G_1 + P_1 C_1) = G_2 + G_1 P_2 + P_1 P_2 C_1$$

$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_1$$

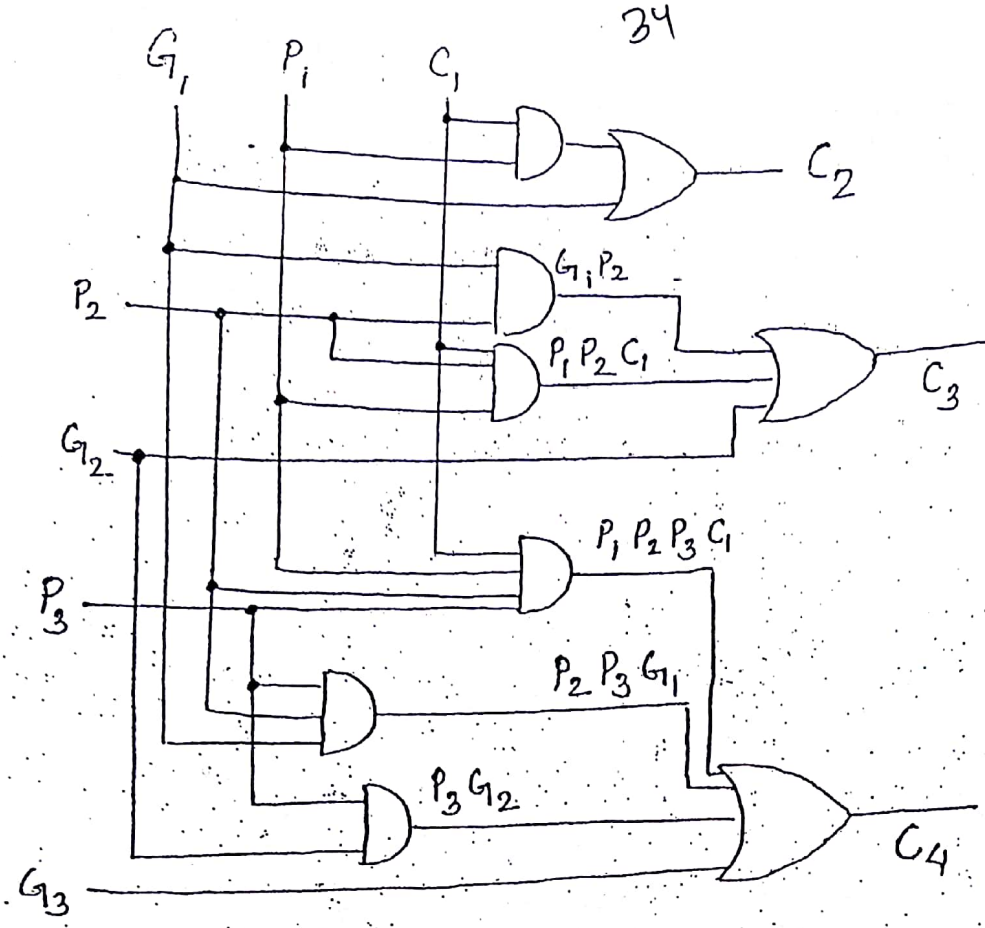
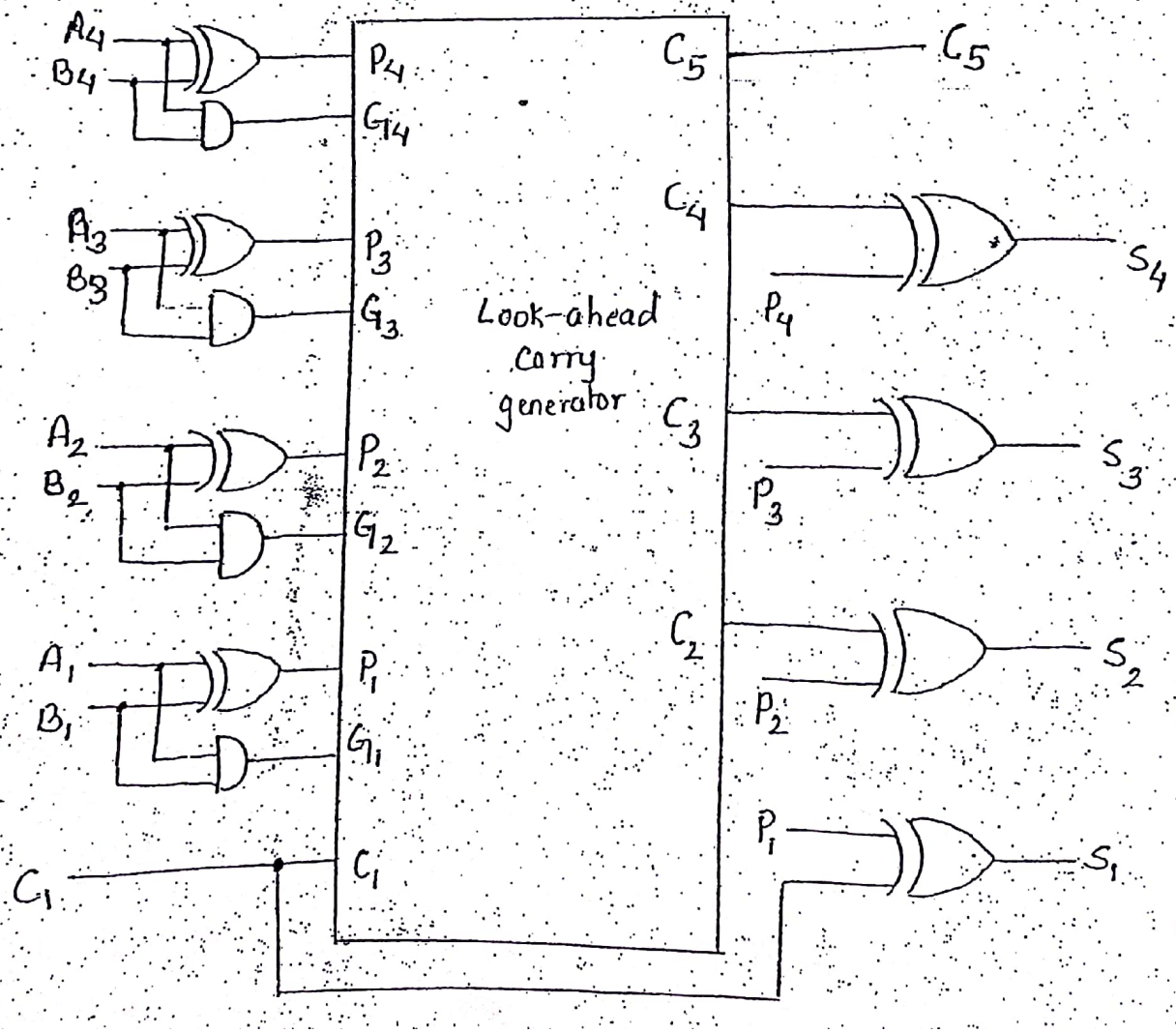


Fig: logic diagram of a look-ahead carry Gene



Binary Subtractor:

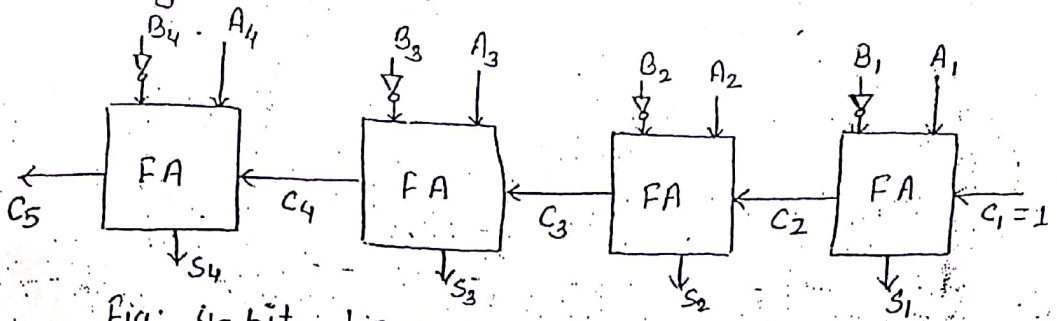


Fig: 4-bit binary parallel subtractor

$$\begin{array}{llllll} & A_4 & A_3 & A_2 & A_1 & \leftarrow \text{Carry in } [C_1] \\ + & \overline{B}_4 & \overline{B}_3 & \overline{B}_2 & \overline{B}_1 & \\ \hline & S_4 & S_3 & S_2 & S_1 & \end{array}$$

→ The circuit consists of a parallel adder with inverter placed at each data input of B & the initial input carry C_1 is set to 1. For subtracting (A−B).

→ The inverter ckt produces 1's complement of B & carry in $C_1 = 1$ is added to the 1's complement, so the effect is equivalent to adding A & 2's complement of B. The final carry is ignored.

Binary Adder – Subtractor

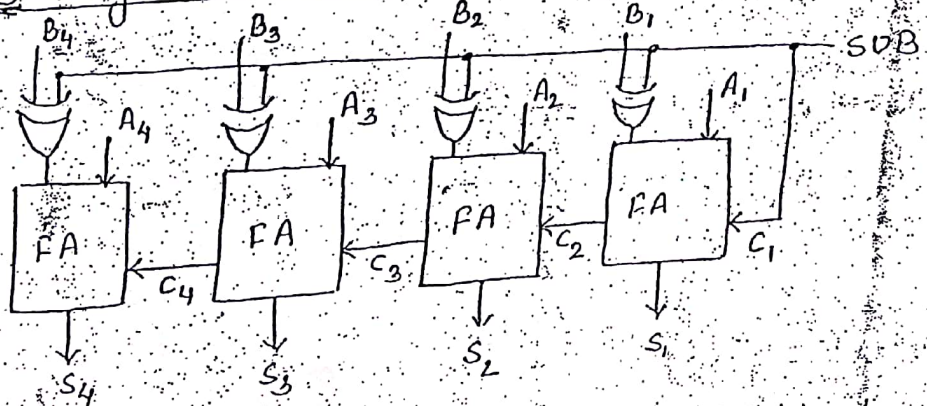


Fig: 4-bit binary adder-subtractor

→ The addition & subtraction can be combined into one circuit with one common adder with an exclusive-OR gate at each input of B to the full adder. The SUB input controls the operation when

SUB = 0, the ckt perform as binary adder.

As SUB = 0

$B \oplus 0 = B0' + B'0 = B$

with initial carry input $C_1 = 0$

When SUB = 1, the ckt perform as binary subtractor.

As SUB = 1, $B \oplus 1 = B1' + B'1 = B'$

with initial carry input $C_1 = 1$

i.e 2's complement of B is added to A.

BCD Adder

→ Two decimal digits in BCD are applied to a 4-bit binary adder. The adder will perform the sum and produces output which may range from 0 to 19.

Sum cannot be greater than $1 + 9 + 9 = 19$ (↓ possible carry from a previous)

→ The binary sum are listed below in the table and are labelled by symbols K, Z_8, Z_4, Z_2 & Z_1. Here K is the carry and the subscripts under the letter Z represents the weight 8, 4, 2 & 1 that can be assigned to four bits in the BCD code.

Derivation of a BCD adder

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	0	1	1	0	0	1	0	0	4
0	0	1	0	0	0	0	1	0	1	5
0	0	1	0	1	0	0	1	1	0	6
0	0	1	1	0	0	0	1	1	1	7
0	0	1	1	1						

K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	Decimal
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

id

→ First 5 columns list the binary sum & the second 5 columns list the BCD sum.

→ The sum of binary and BCD are same till 1001 (9).

→ When binary sum is greater than 1001 (9), BCD invalid codes are generated.

→ The addition of binary 0110 (6) converts the invalid into valid BCD & also produces an output carry.

→ It is obvious from the table that correction is needed when binary sum has output carry i.e $K=1$ and whenever $Z_8=1$ and either Z_4 or Z_2 or both are 1.

→ Thus we can derive output carry as

$$C = K + Z_8 Z_4 + Z_8 Z_2$$

When $C=1$, it is necessary to add 0110 (6) to binary sum.

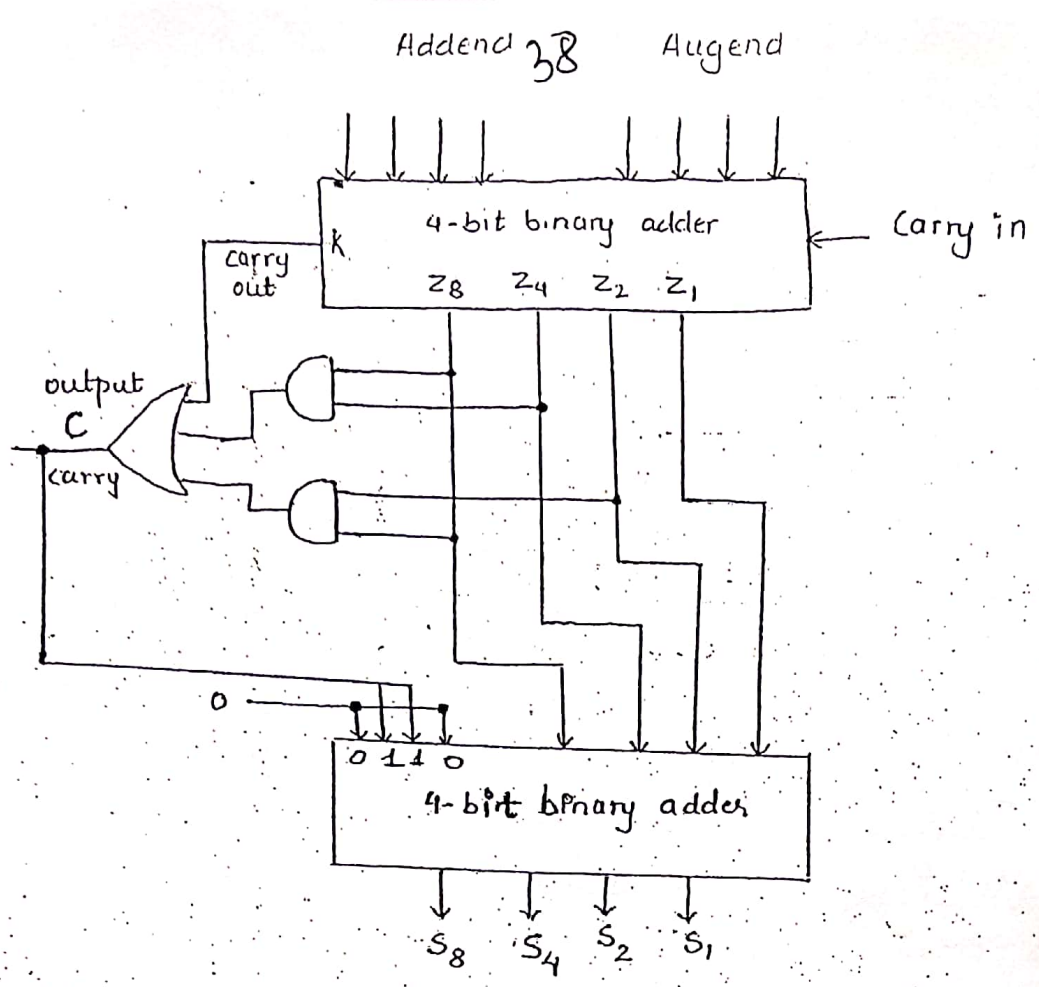


Fig: Block diagram of a BCD adder

Magnitude Comparator :

→ A magnitude comparator is a combinational circuit th compares two numbers A & B and determines their relat magnitude.

→ The outcome of the comparision is specified by three binary variables that indicate whether $A > B$, $A = B$ or $A < B$.

→ Consider two Numbers A & B with four digits eac write the coefficient of the numbers with descendi significance position.

$$A = A_3 A_2 A_1 A_0$$
$$B = B_3 B_2 B_1 B_0$$

For A=B

→ The two numbers are equal if all pairs of significant digits are equal i.e if $A_3 = B_3$ & $A_2 = B_2$ & $A_1 = B_1$ & $A_0 = B_0$.

→ When the numbers are binarey the digits are either 0 or 1.

→ If both the inputs are same the output must be high i.e the equivalence X-NOR function.

$$x_i = A_i' B_i + A_i B_i' \quad i = 0, 1, 2, 3$$

A⊕B $A_i' B_i + A_i B_i'$

→ If all the significant digits are equal then

$$(A = B) = x_3 \cdot x_2 \cdot x_1 \cdot x_0$$

$x_i = A_i B_i + \bar{A_i}\,\bar{B_i}$

$A = B = x_3 x_2 x_1 x_0$

$A > B = A_3 B_3' + x_3 A_2 B_2' + x_3 x_2 A_1 B_1' + x_3 x_2 x_1 A_0 B_0'$

For A>B

→ If A is greater than or less than B, we inspect the relative magnitudes of pairs of significant digits starting from the most significant position. If the two digits are equal, we compare the next lower significant pair of digits. This comparision continues until a pair of unequal digits is reached.

→ If the corresponding digit of A = 1 & that of B = 0 then A>B.

$$\therefore (A > B) = A_3 B_3' + x_3 A_2 B_2' + x_3 x_2 A_1 B_1' + x_3 x_2 x_1 A_0 B_0'$$

For A<B

→ If the corresponding digit of A is 0 & that of B is 1, we have A<B

$$\therefore (A < B) = A_3' B_3 + x_3 A_2' B_2 + x_3 x_2 A_1' B_1 + x_3 x_2 x_1 A_0' B_0$$

(TTL type 7485 is a 4-bit magnitude comparator)

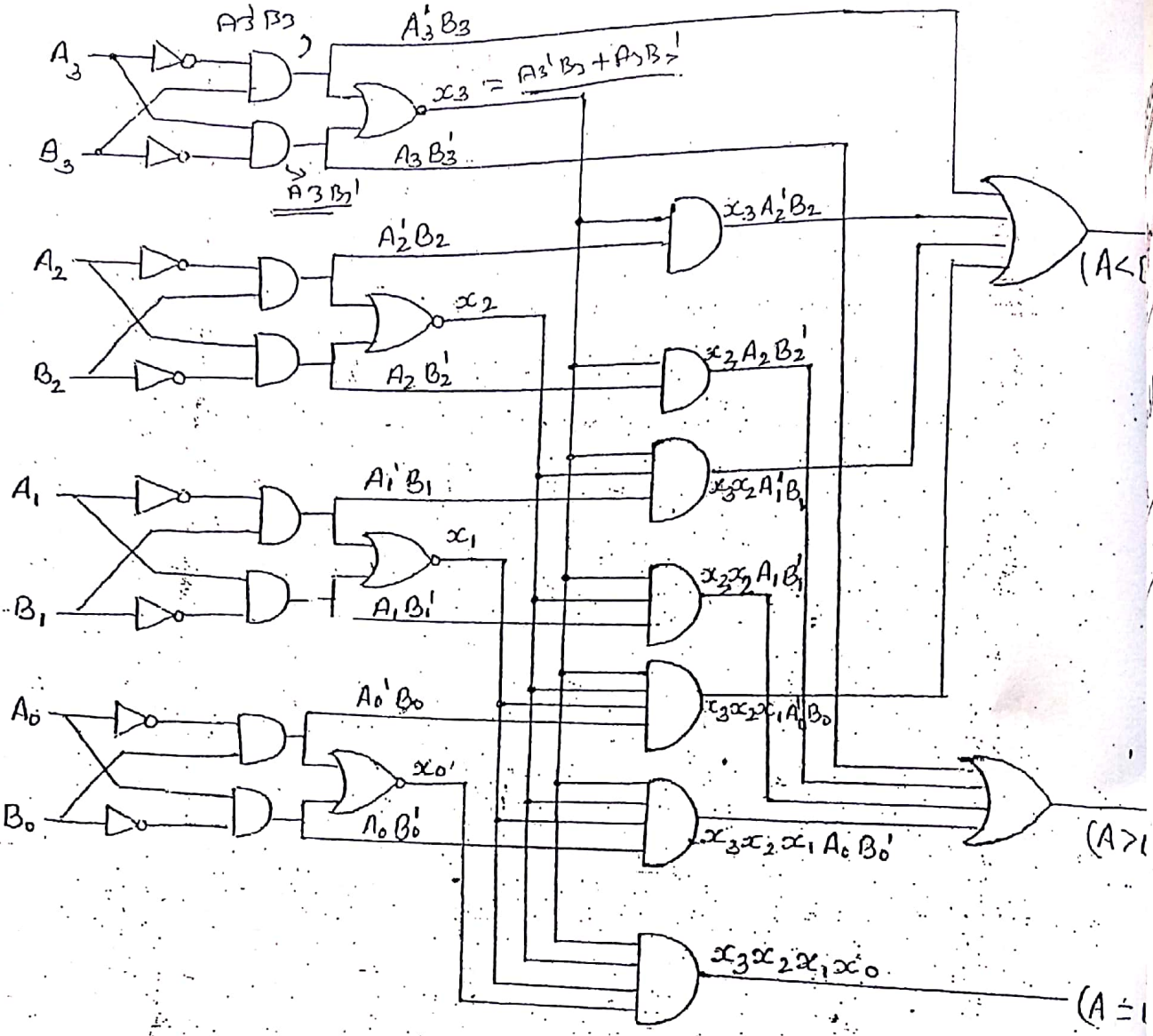


Fig: 4-bit magnitude comparator

Decoder:-

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

→ The basic function of a decoder is to detect the presence of a specified combination of bits (code) on its inputs and to indicate the presence of that code by a specified output level.

→ If the n bit information from n input lines has unused or don't care combinations, the decoder output will have less than 2^n outputs.

→ n are the input lines and m are the output lines the in decoder.

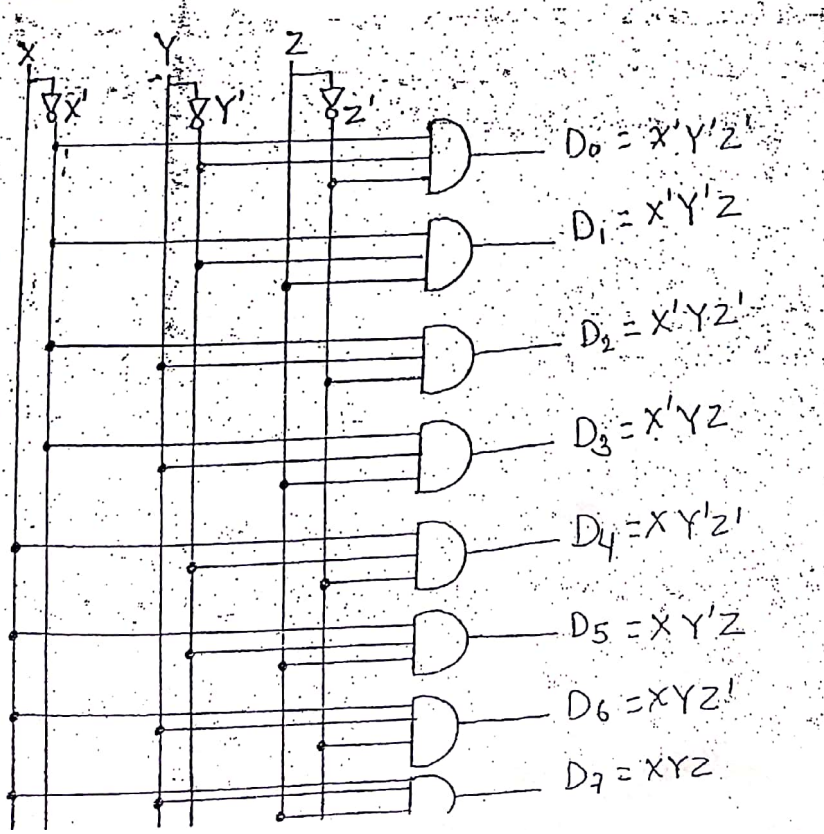
$$m \leq 2^n$$

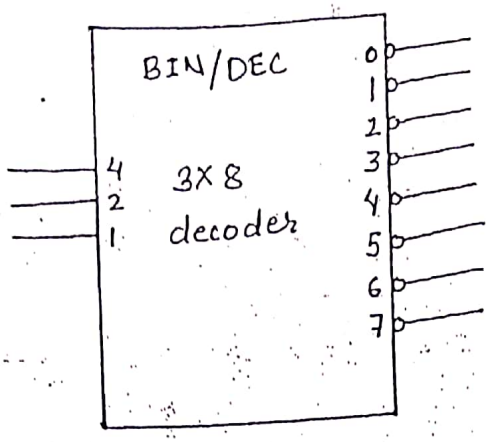
3-bit decoder

→ The 3 input lines are decoded into eight outputs, each output representing one of the minterms of the 3-input variables. Also called 3 to 8 line decoder.

Truth table of 3 to 8 line decoder

Inputs			outputs							
X	Y	Z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1





→ A logic symbol for 3 to 8 lines decoders with active low outputs is shown above.

→ The BIN/DEC label indicates that a binary input ma the corresponding decimal output. The input label 4, 2, & represents the binary input positional weight respectivly

Design a BCD to Decimal decoder

→ It has 4 inputs & 10 utputs, one for each decimal digi

→ 4-10 line BCD to decimal decoder.

→ Since the ckt has 10 outputs 10 K-maps are required simplyfy each one output function.

→ 6 don't care cond^n at input can be used to simplify each of the output function.

→ Instead of drawing 10-kmap, all output variables D_0 to D_9 is drawn in 1 K-map. To simplyfy each output seperately with don't care conditions.

wx \ yz	00	01	11	10
00	D_0	D_1	D_3	D_2
01	D_4	D_5	D_7	D_6
11	X	X	X	X
10	D_8	D_9	X	X

so, $D_0 = w'x'y'z'$

$D_1 = w'x'y'z$

$D_2 = x'yz'$

$D_3 = x'yz$

$D_4 = xy'z'$

$D_5 = xy'z$

$D_6 = xyz'$

$D_7 = xyz$

$D_8 = wz'$

$D_9 = wz$

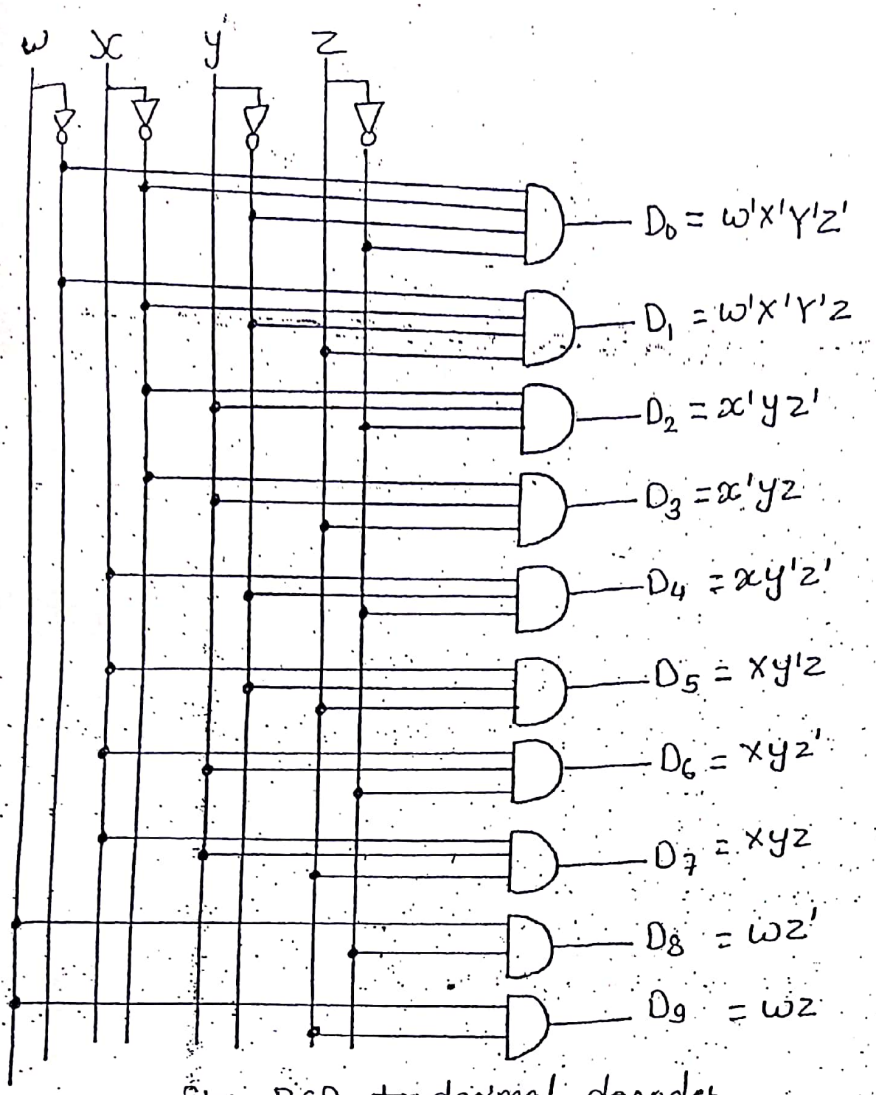


Fig: BCD to decimal decoder

Example question

Implement a full adder circuit with a decoder and two OR gates (S) (C)

→ A decoder provides the 2^n minterm of n input variables since any Boolean function can be expressed in sum of minterms canonical form, one can use a decoder to generate the minterms and a external OR gate to form the sum.

$S(x,y,z) = \Sigma(1,2,4,7)$

$C(x,y,z) = \Sigma(3,5,6,7)$

Since there are three inputs and a total of eight minterms, we need 3 to 8 line decoder

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

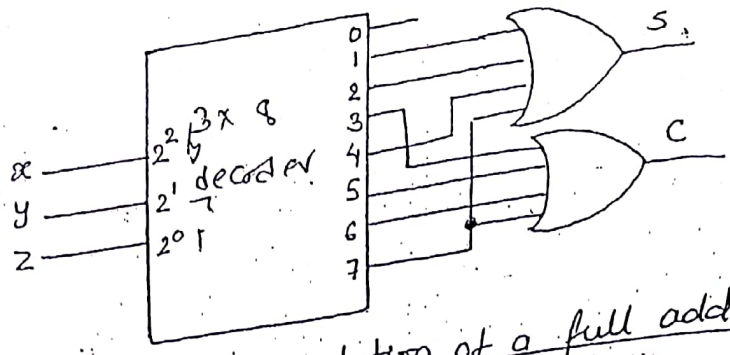


Fig: Implementation of a full adder with a deco

Encoder

→ An encoder produces reverse operation from that decoder.

→ It has 2^n (or less) input lines & n output lines.

→ The output line generates the binary code for the 2^n input lines.

→ The process of converting from familier symbols or numbers to a coded format is called encoding.

Octal to Binary encoder

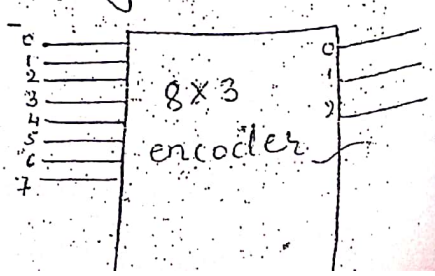
Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	X	Y	Z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Fig: Truth table of octal to binary encoder

$$X = D_4 + D_5 + D_6 + D_7$$

$$Y = D_2 + D_3 + D_6 + D_7$$

$$Z = D_1 + D_3 + D_5 + D_7$$



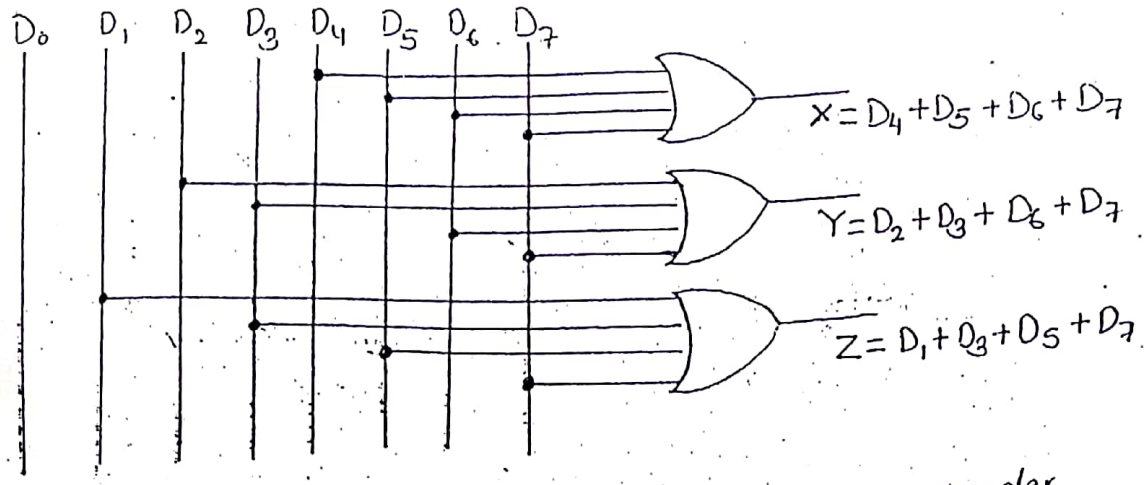


Fig: logic implementation of octal to Binary encoder

Here, D_0 is not connected to any OR gate, the binary output must be all 0 in this case. An all 0's output is also obtained when all inputs are 0.

→ Priority Encoder: These encoder establish an input priority to ensure that the highest-priority input line is encoded. eg. If both D_2 & D_5 are logic 1 simultaneously, the output will be 101 because D_5 has a highest priority over D_2.

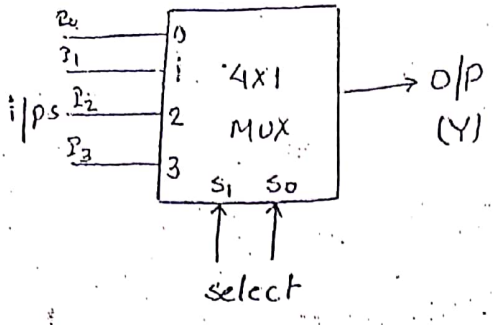
Multiplexer (Many to one) (many input भएपनि एउटै output)

→ A digital multiplexer is a combinational circuit that selects binary information from one of many input lines and directs into a single output line.

→ The selection of a particular input line is controlled by a set of selection lines.

→ Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

→ A multiplexer is also called a data selector since it selects one of many inputs & gives the binary information to the output line.



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

Fig: Block diagram

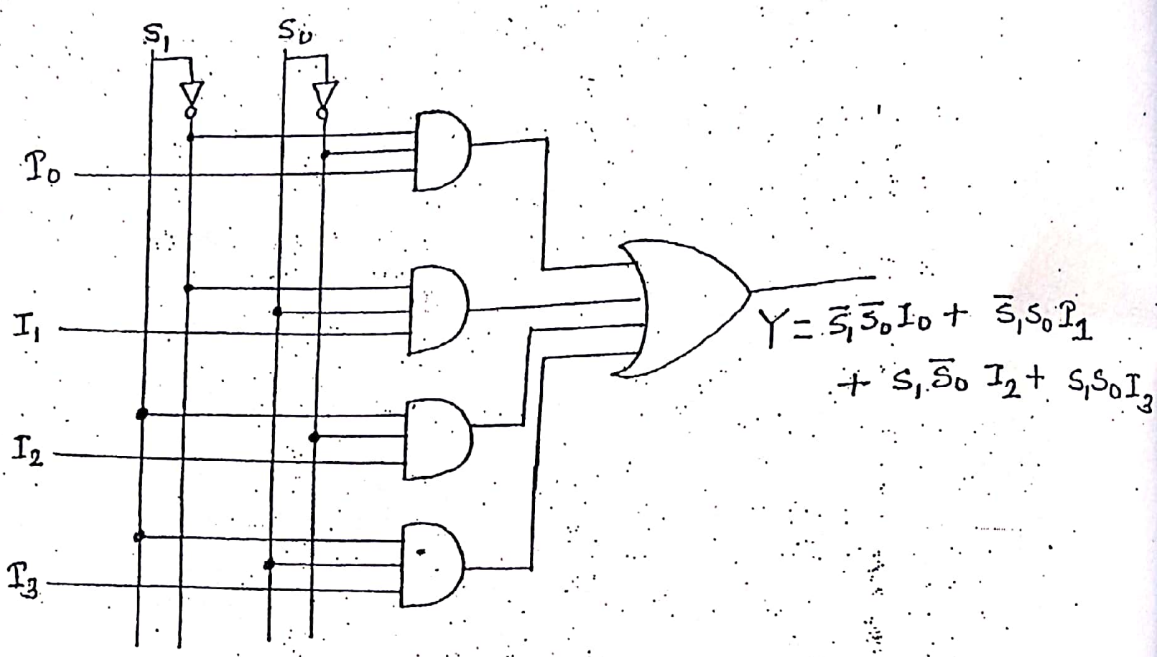


Fig: 4x1 multiplexer

→ It is used for connecting two or more sources to a single destination among computer units & it is useful for constructing a common bus system. It can be used to implement any boolean function.

Boolean function implementation of Multiplexer

$F(A,B,C) = \Sigma(1,3,5,6)$

→ If we have a Boolean function of n+1 variables, we take n of these variables and connect them to the selection lines of a multiplexer. The remaining single variable of a function is used for the inputs of the multiplexer.

→ If A is this single variable, the inputs of the multiplexer are chosen to be either A or A' or 1 or 0. By using these four values for the inputs and by connecting the other variables to the selection lines, one can implement any Boolean function with a MUX.

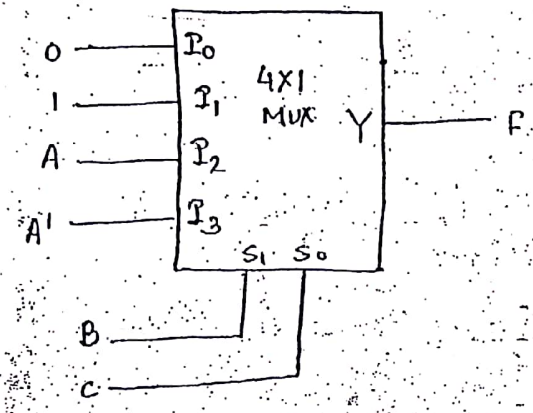
Minterm	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

a) Truth table

	I_0	I_1	I_2	I_3
A'	0	(1)	2	(3)
A	4	(5)	(6)	7
	0	1	A	A'

A = 1 for 4, 5, 6 & 7
A = 0 for 0, 1, 2 & 3

b) Implementation table



c) MUX implementation

For implementation table

→ If the two minterms in a column are not circled, apply 0 to the corresponding multiplexer input.

→ If the two minterms are circled, apply 1 to the corresponding multiplexer input.

→ If the bottom minterm is circled and the top is not circled, apply A to the corresponding multiplexer i/p.

→ If the top minterm is circled and the bottom is not circled, apply A' to the corresponding multiplexer i/p.

4B

Implement the Boolean function with a MUX

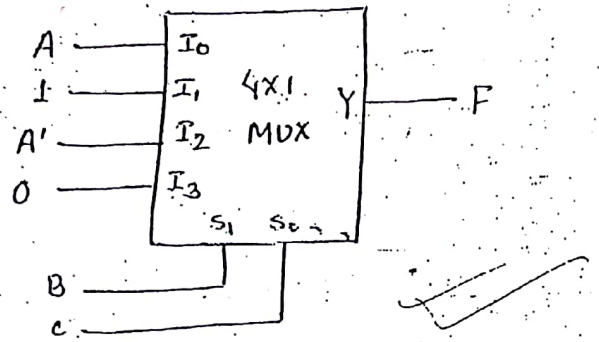
$F(A, B, C) = \Sigma(1, 2, 4, 5)$

minterm	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

a) Truth table

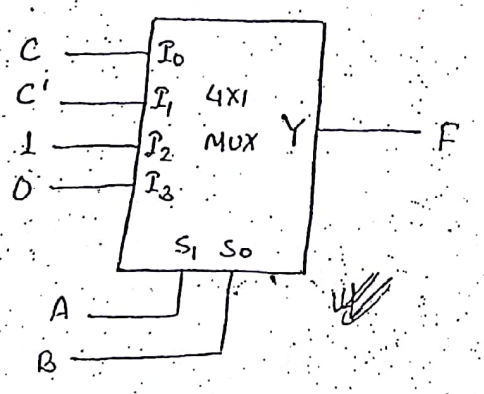
	I_0	I_1	I_2	I_3
A'	0	(1)	(2)	3
A	(4)	(5)	6	7
	A	1	A'	0

b) Implementation table



If we take A, B as selection inputs then,

	I_0	I_1	I_2	I_3
C'	0	(2)	(4)	6
C	(1)	3	(5)	7
	C	C'	1	0



Multiplexer implementation

Boolean function $F(W,X,Y,Z) = \Sigma(1,2,5,7,11,15)$ Implement using 8 to 1 MUX.

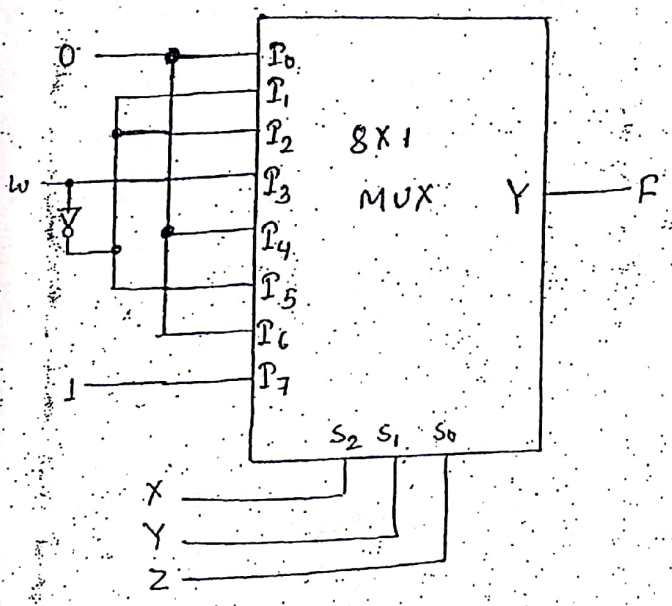
Soln:

minterm	W	X	Y	Z	F
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	1

a) Truth table

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
W'	0	(1)	(2)	3	4	(5)	6	(7)
W	8	9	10	(11)	12	13	14	(15)
	0	W'	W'	W	0	W'	0	1

b) Implementation table



Multiplexer implementation

Demultiplexer

→ A demultiplexer is a circuit that receives information on a single line & transmits this information on one of 2^n possible output lines.

→ The selection of a specific output line is controlled by the bit values of n selection lines.

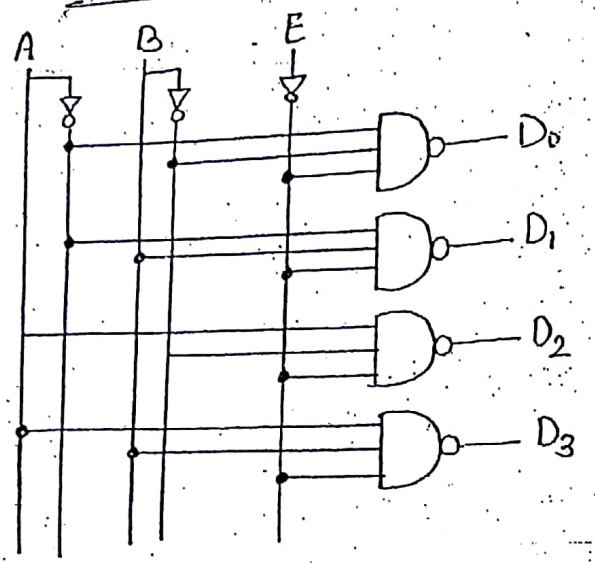
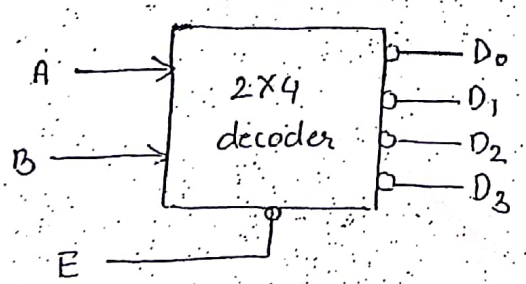


Fig: Logic Diagram of a 2 to 4 line decoder with enable (E) input.

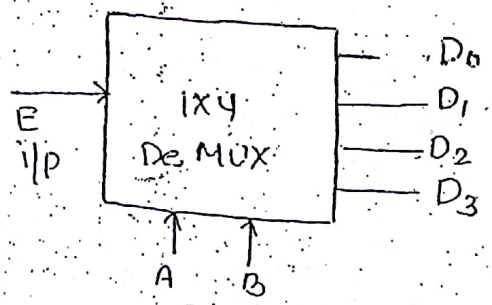
E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

Truth table

* it is also called distribu
* it is similar to dec
Demux perform one to
many output.. MUX M to
device but demux it
devile.



Decoder with enable



Demultiplexer

i/p

S_1	S_0
0	0
0	1
1	0
1	1

o/p

D_0	D_1	D_2	D_3
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

→ A decoder can function as demultiplexer if the E(enable) line is taken as a data input & lines A & B are taken as the selection lines.

→ The single input variable E has a path to all four outputs, but the input is directed to only one of the o/p lines at a time, which is selected by selection line.

for eg.

→ If selection, AB = 00, o/p D_0 will have the same data as the i/p value E has.

→ If AB = 10, o/p D_2 has the same data as the i/p value E has.

→ A decoder with an enable i/p is referred to as a decoder / demultiplexer. It is the enable input that makes the ckt a demultiplexer.

→ Decoder / demultiplexer ckt can be connected to form a larger decoder ckt.

→ two 3X8 decoder with enable inputs connected to form a 4X16 decoder

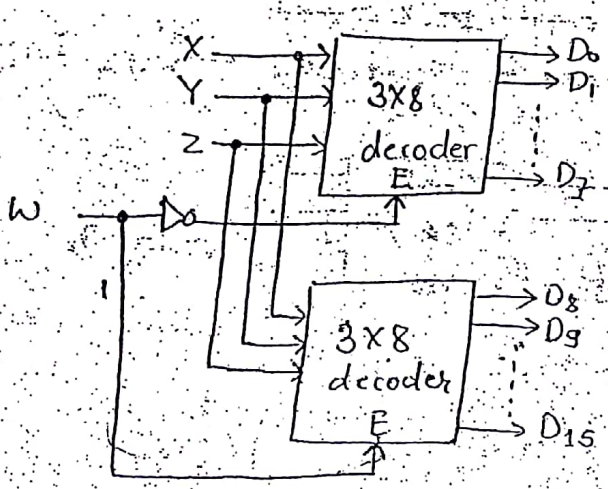


Fig: A 4X16 decoder constructed with two 3X8 decoders

→ When w=0, the top decoder is enabled and the bottom is disabled.

→ when w=1, the bottom one is enabled & the top is disabled.

Hence, the enable lines are a convenient feature fo
two or more IC package for the purpose of expan
digital function into a similar function with mo
and outputs.

Programmable Logic Device (PLD)

→ A programmable logic device (PLD) is an integr
circuit with internal logic gates that are connecte
electronic fuses.

→ Programming the device involves the blowing of
along the paths that must be disconnected so as to
a particular configuration.

→ The initial state of a PLD has all the fuses int

→ Programming the device involves blowing of inter
fuses to achieve desired logic function.

* Simple Programmable logic Devices (SPLDs)

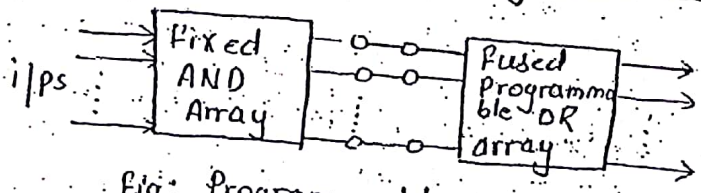


Fig: Programmable Read Only Memory (PROM)

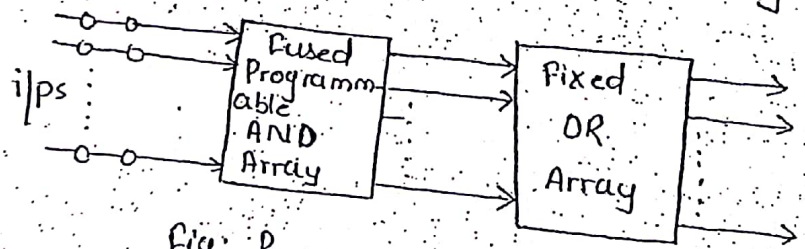


Fig: Programmable Array Logic (PAL)

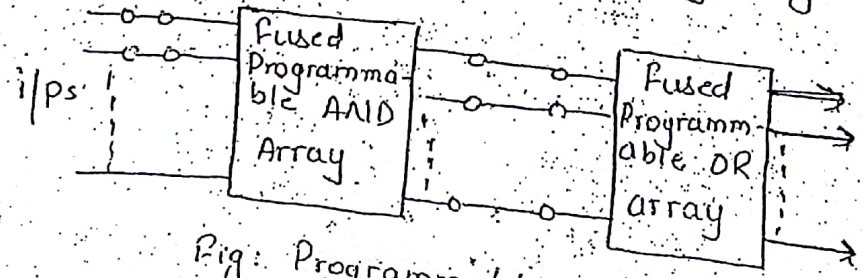


Fig: Programmable Logic Array (PLA)

Read Only Memory (ROM)

→ It consists of n inputs lines & m output lines.
→ Each bit combination of the i/p variable is called an address.
→ Each bit combination that comes out of the o/p lines is called a word.
→ An address is essentially a binary numbers that denotes one of the minterms of n variable.
→ The numbers of distinct address possible with n i/p variables as 2^n.
→ An o/p word can be selected by a unique address and since there are 2^n distinct addresses in a ROM.
→ There are 2^n distinct words which are said to store in the unit.
→ A ROM is characterised by the number of word 2^n and the number of bits per word m.
→ Internally, the ROM is a combinational logic ckt with AND gates connected as a decoder and a number of OR gates equal to the number of output in the unit.

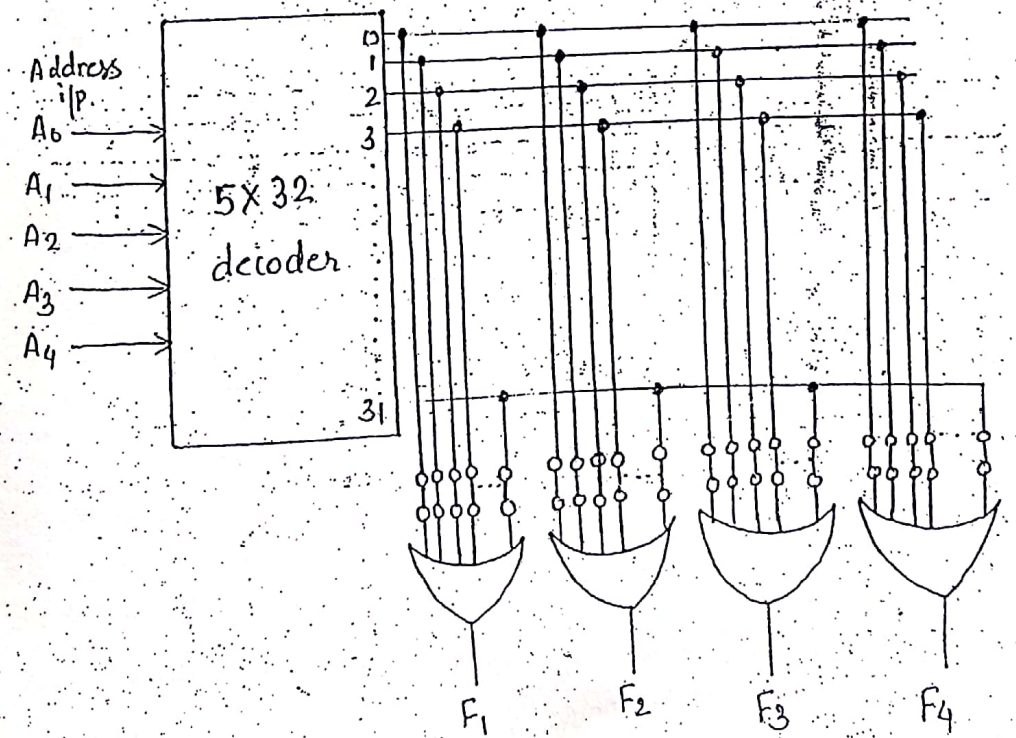


Fig: Logic Construction of a 32X4 ROM
128 bit

Combinational logic Implementation

Example:

1) Implement $F_1(A_1, A_0) = \Sigma(1, 2, 3)$

$F_2(A_1, A_0) = \Sigma(0, 2)$ with a ROM

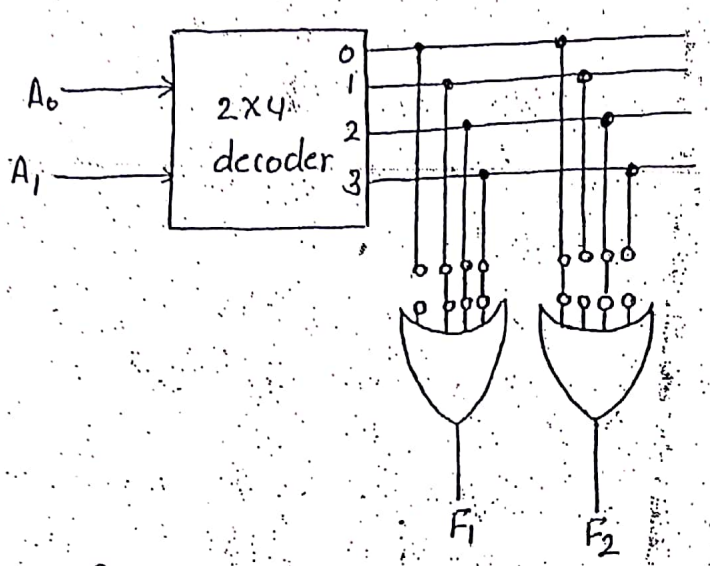


Fig: Combinational-ckt implementation with a 4X2 ROM with AND-OR gates.

2) Implement $F_1(A, B, C) = \Sigma(0, 1, 5, 7)$

$F_2(A, B, C) = \Sigma(2, 3, 4)$

$F_3(A, B, C) = \Sigma(0, 1, 6)$

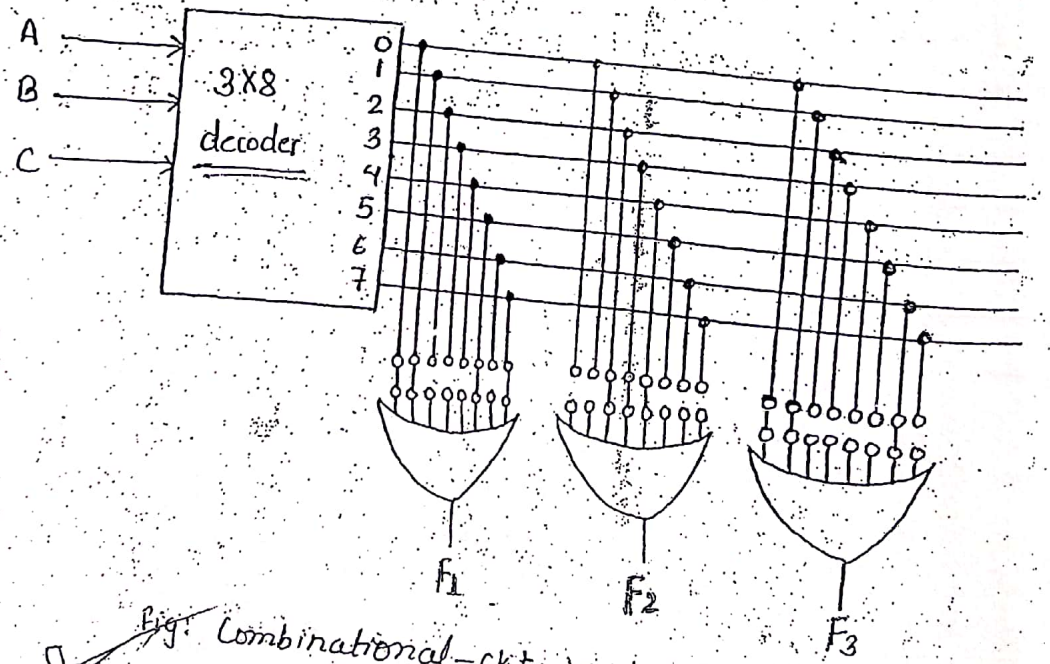


Fig: Combinational-ckt implementation with a 8X3 ROM with AND-OR gates.

Programmable Logic Array (PLA)

→ A PLA is similar to a ROM in concept but the PLA does not provide full decoding of the variables and does not generate all the minterms as in the ROM.

→ In PLA, the decoder is replaced by a group of AND gates, each of which can be programmed to generate a product term of the input variables.

→ The AND and OR gates inside the PLA are initially fabricated with fuses among them.

→ The specific Boolean functions are implemented in sum of products form by blowing appropriate fuses and leaving the desired connection.

→ The size of the PLA is specified by the number of inputs, the number of product terms and the number of outputs.

→ A typical PLA has 16 inputs, 48 product terms & 8 outputs. Then the no. of programmed fuses is

$[2n \times K + K \times m + m]$,

$= [2 \times 16 \times 48 + 48 \times 8 + 8]$

where as that of ROM is $2^n \times m$.

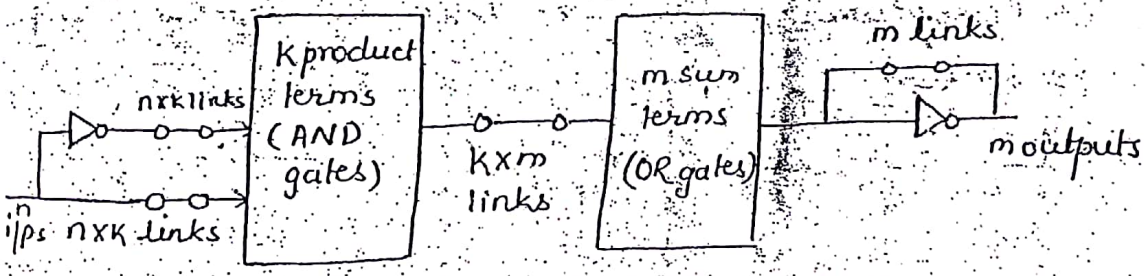


Fig: PLA block diagram

Example Implement $F_1 = \Sigma(4,5,7)$, $F_2 = \Sigma(3,5,7)$

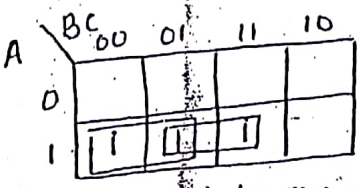
$F_1 = AB' + AC$

$F_2 = AC + BC$

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

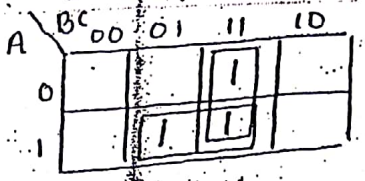
Truth table

K-map for F_1



$F_1 = AB' + AC$

K-map for F_2



$F_2 = BC + AC$

Product terms	Inputs			Outputs		
	A	B	C	F_1	F_2	
AB' (1)	1	0	–	1	–	
AC (2)	1	–	1	1	1	
BC (3)	–	1	1	–	1	
				T	T	T/C

T- Implies output dictates that the link across the o/p inverters remains in place.

C- (Complement) - specifies that the corresponding link be broken.

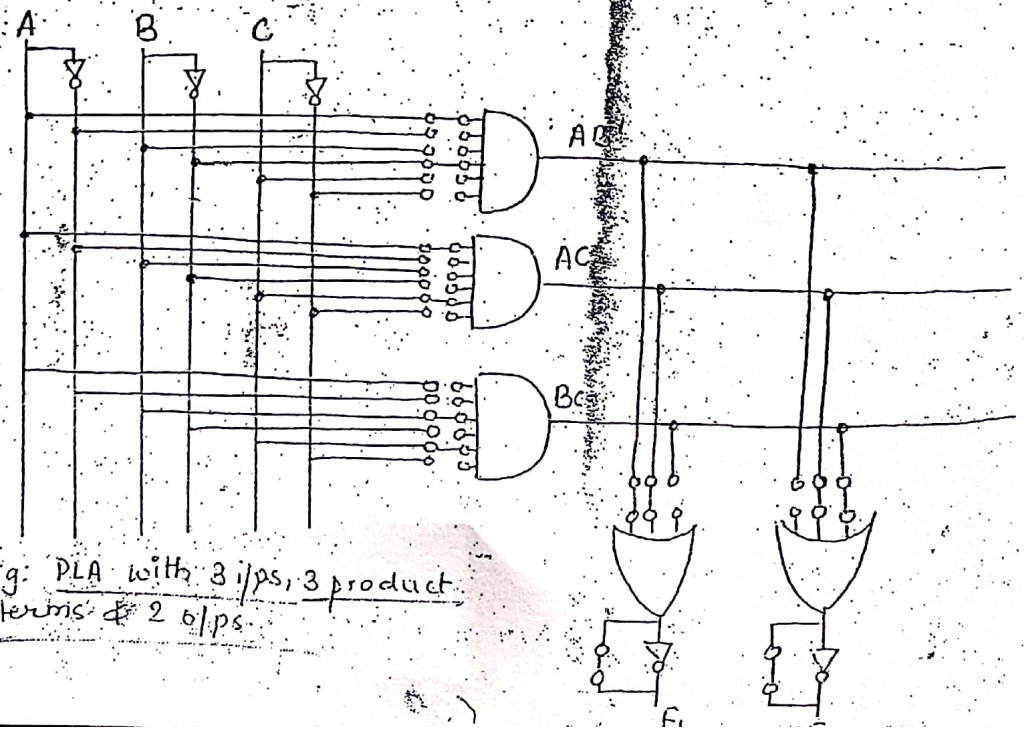


Fig: PLA with 3 i/ps, 3 product terms & 2 o/ps.

Programmable Logic Array (PLA)

→ A PLA is similar to a ROM in concept but the PLA does not provide full decoding of the variables and does not generate all the minterms as in the ROM.

→ In PLA, the decoder is replaced by a group of AND gates, each of which can be programmed to generate a product term of the input variables.

→ The AND and OR gates inside the PLA are initially fabricated with fuses among them.

→ The specific Boolean functions are implemented in sum of products form by blowing appropriate fuses and leaving the desired connection.

→ The size of the PLA is specified by the number of inputs, the number of product terms and the number of outputs.

→ A typical PLA has 16 inputs, 48 product terms & 8 outputs. Then the no. of programmed fuses is

$[2n \times K + K \times m + m]$,

$= [2 \times 16 \times 48 + 48 \times 8 + 8]$

where as that of ROM is $2^n \times m$.

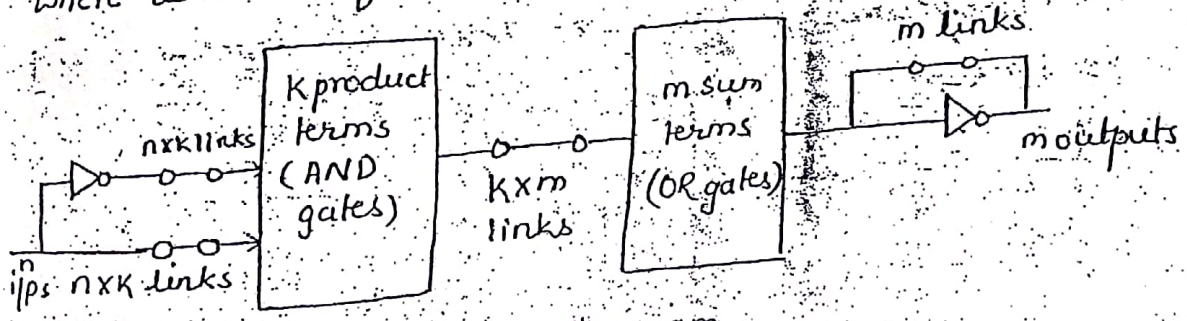


Fig: PLA block diagram

Example Implement

$F_1 = AB' + AC$

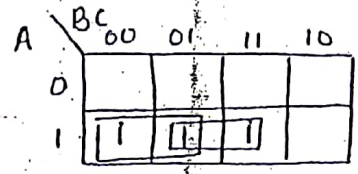
$F_2 = AC + BC$

$F_1 = \Sigma(4,5,7)$, $F_2 = \Sigma(3,5,7)$

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

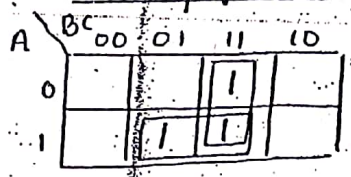
Truth table

K-map for F_1



$F_1 = AB' + AC$

K-map for F_2



$F_2 = BC + AC$

Product terms		Inputs			Outputs		
		A	B	C	F_1	F_2	
AB'	(1)	1	0	–	1	–	
AC	(2)	1	–	1	1	1	
BC	(3)	–	1	1	–	1	
					T	T	T/C

T- implies output dictates that the link across the o/p inverters remains in place.

C- (complement) - specifies that the corresponding link be broken.

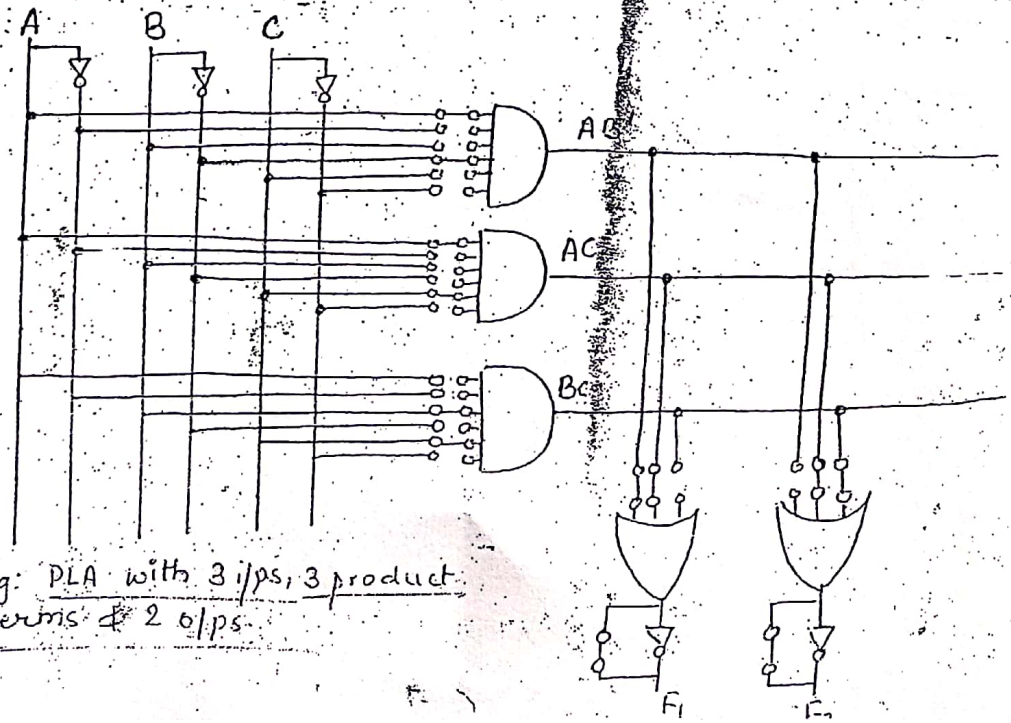


Fig: PLA with 3 i/ps, 3 product terms & 2 o/ps.

6. A combinational circuit is defined by the functions

$F_1(A,B,C) = \Sigma(3,5,6,7)$

$F_2(A,B,C) = \Sigma(0,2,4,7)$

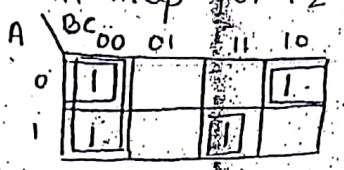
Implement the circuit with a PLA having three inputs, four product terms and two outputs.

Soln

K-map for F_1

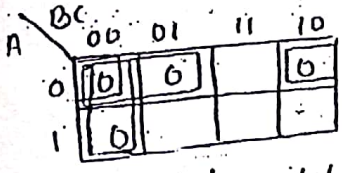
$F_1 = BC + AC + AB$

K-map for F_2

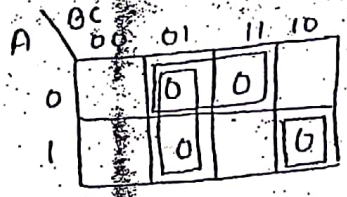


$F_2 = A'C' + B'C' + ABC$

→ Here total no. of product terms are 6 but we should use only four. Therefor let us see from pos soln.



$F_1' = A'B' + A'C' + B'C'$



$F_2' = A'C + B'C + ABC'$

→ For four product terms we can use F_1 & F_2

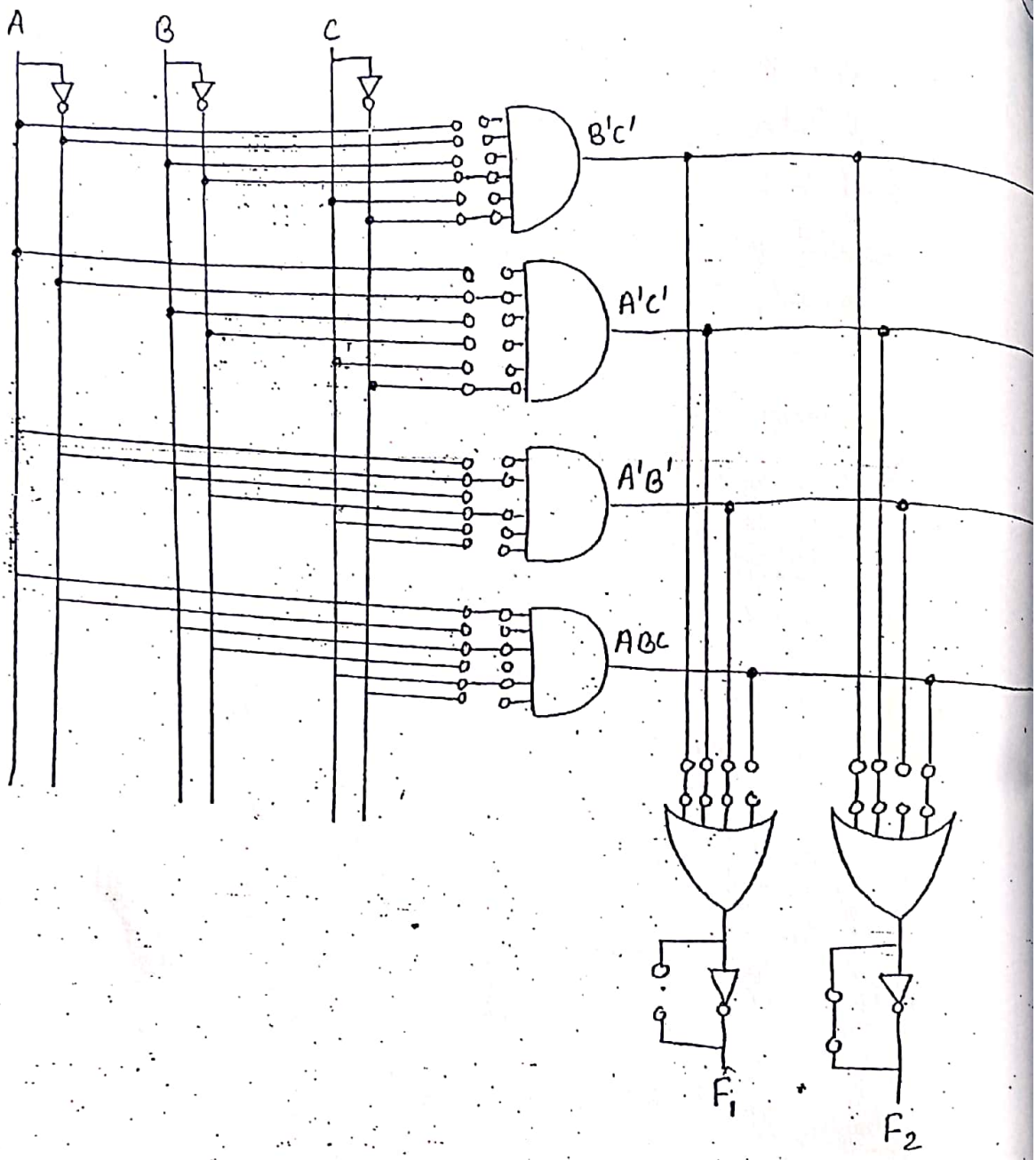
PLA program table

Product terms	Inputs			Outputs	
	A	B	C	F_1	F_2
1 $B'C'$	—	0	0	1	1
2 $A'C'$	0	—	0	1	1
3 $A'B'$	0	0	—	1	0
4 ABC	1	1	1	0	1
				C	T

T/C

$F_1 = (B'C' + A'C' + A'B')'$

$F_2 = B'C' + A'C' + ABC$



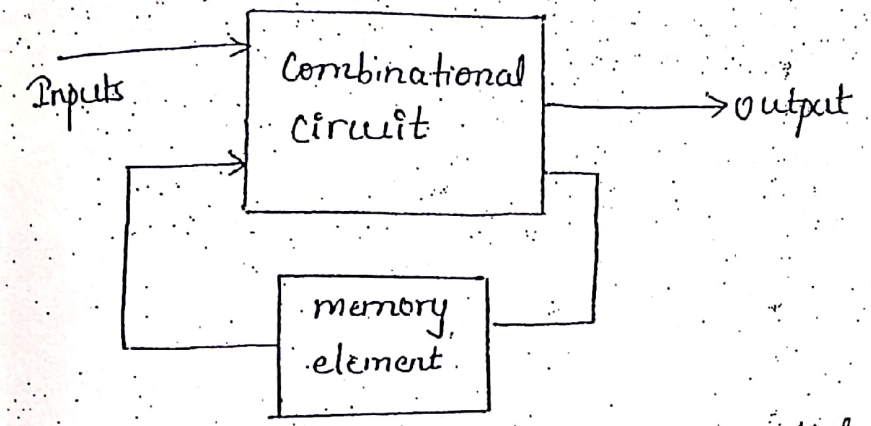
Chapter-7

Sequential logic

Prepared By:
Buddha Laxmi Mahajan

Sequential logic

→ A sequential logic circuit consists of a combinational circuit to which memory elements are connected to form a feedback path.



[in chapter 5 photo cop
it is writen that
sequential ckt ins
the box but that
is wrong. It must
be combinational
ckt.]

Fig: Block diagram of a sequential circuit

→ The outputs of a sequential circuit depends on present inputs and past or previous state. Eg. flipflop, registers, counters.

Types of sequential logic:-

1) Synchronous:

→ In this system signals that affect the memory elements only at discrete instants of time.

→ Synchronous is achieved by a time device called a master-clock generator which generates a periodic train of clock pulses.

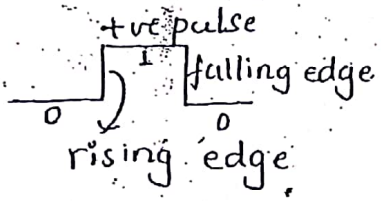
→ Synchronous sequential circuits that use clock pulses in the input of memory elements are called clocked sequential circuit.

2) Asynchronous:

→ The behaviour of an asynchronous sequential logic system depends upon the order in which its input signals change and can be affected at any instant of time.

Clock :-

→ A clock is a control signal that periodically makes a transition from a 0 to 1 & then back to 0 again. we usually denote the clock by the symbol clk or cp.



Flipflop

→ Flipflops are binary cells capable of storing one bit information.

→ A flipflop circuit has two outputs one for the normal value & one for the complement value of the bit stored in it.

→ Binary information can enter in a flipflop in a variety of ways, a fact which give rise to different types of flipflops.

Basic flip-flop circuit

SR flipflop or RS flipflop

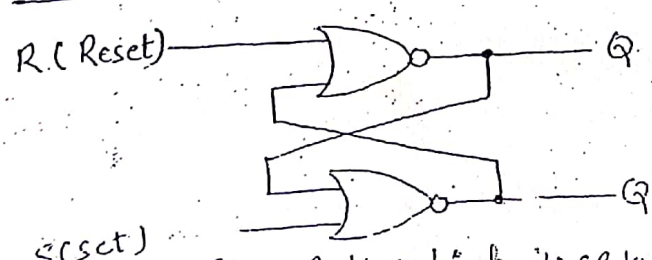


Fig: Active high i/p SR latch

→ An active high i/p SR (set-Reset) latch is formed with crossed coupled

Note

* if S = 1 → set
  i.e Q = 1 ; $\bar{Q}$ = 0
* if R = 1 → Reset
  i.e Q = 0 , $\bar{Q}$ = 1
* if S = 0 & R = 0 No change

Inputs S	R	output Q	$\bar{Q}$	Comment
0	0	NC	NC	No change
0	1	0	1	latch reset
1	0	1	0	latch set
1	1	0	0	invalid condition

Truth table for active-high input SR latch

$\bar{S}\bar{R}$ flipflop

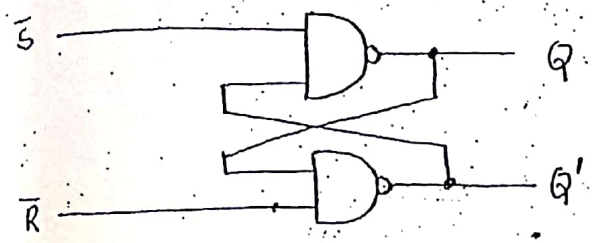


Fig: Active low input $\bar{S}\bar{R}$ latch

→ An active-low input S-R latch is formed with cross-coupled NAND gates.

Input $\bar{S}$	$\bar{R}$	output Q	$\bar{Q}$	comment
0	0	1	1	invalid condition
0	1	1	0	set
1	0	0	1	Reset
1	1	NC	NC	No change

Truth table for active-low input $\bar{S}\bar{R}$ latch

if $\bar{S}=0$ & $\bar{R}=0$ invalid
$\bar{S}=1$ & $\bar{R}=1$ NC
$\bar{S}=1$ Reset
$\bar{R}=1$ set

Clocked SR flipflop

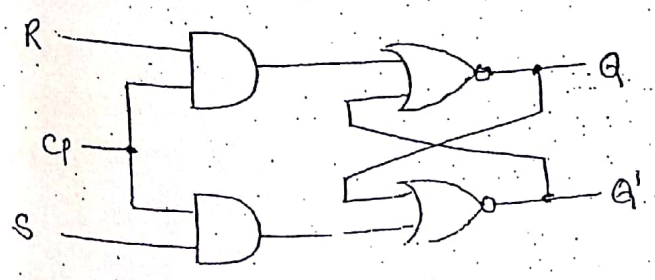
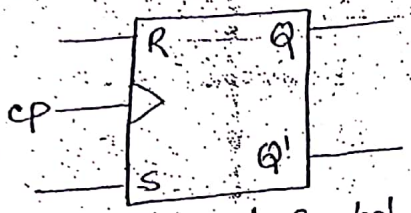


Fig: logic Diagram

Inputs			outputs		Comment
S	R	CP	Q	Q'	
0	0	X	NC	NC	No change
0	1	↑	0	1	Reset
1	0	↑	1	0	set
1	1	↑	?	?	Invalid

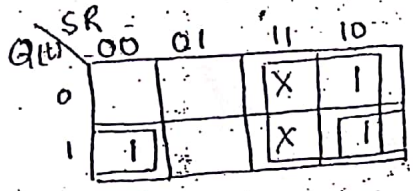
Truth table



Graphical Symbol

Q(t)	S	R	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	invalid X
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	invalid X

characteristic table

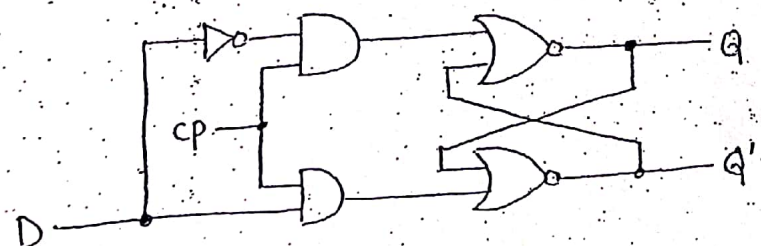


$Q(t+1) = S + QR'$

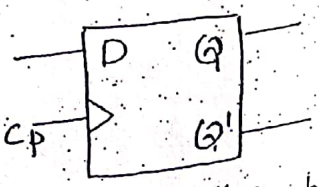
Characteristic Equation

D flipflop : (Disability to transfer 'Data')

→ It is an RS flipflop with a inverter in the R input



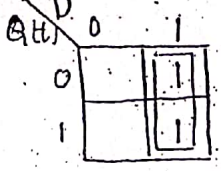
logic Diagram



Graphical Symbol

Q(t)	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

characteristic table



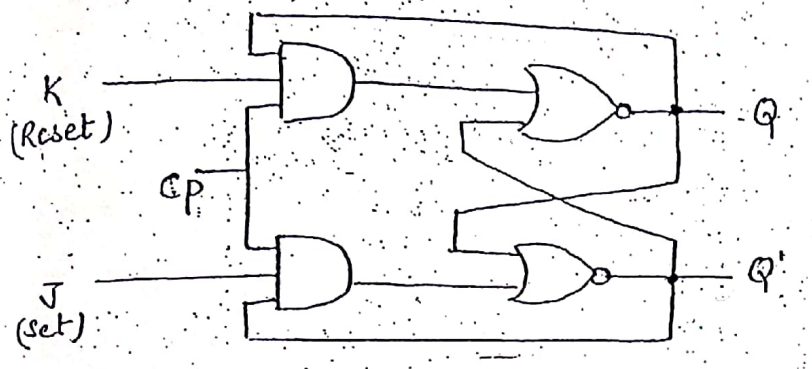
$Q(t+1) = D$

Characteristic Equation

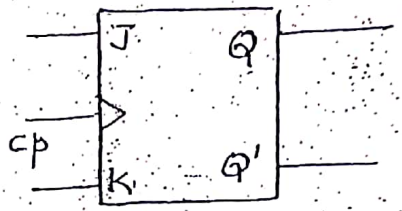
JK flipflop

→ It is a refinement of RS flipflop. The indeterminate(invalid) state of RS flipflop is defined in JK flipflop.

→ Input J & K behaves like S & R i.e set & Reset respectively.
J → set K → reset



logic diagram



Graphical Symbol

S J	R K	Q(t+1)	comment
0	0	0	No change
0	1	0	Reset
1	0	1	set
1	1	1	Complement o toggle
0	0	1	No change
0	1	0	Reset
1	0	1	set
1	1	0	Toggle

Characteristic table

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

$Q(t+1) = Q'J + QK'$ characteristic Equation.

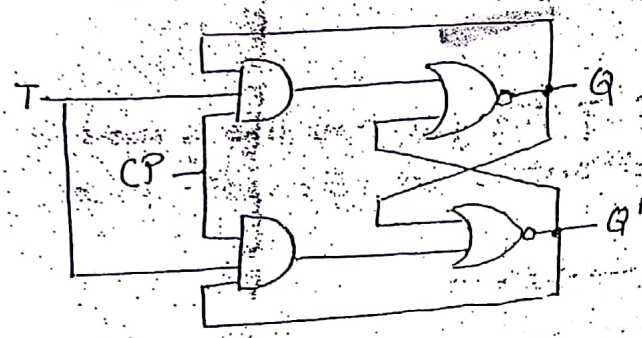
when J=0, K=0 flipflop retains its previous state.
when J=0, K=1 flipflop goes to reset state $Q(t+1)=0$
when J=1, K=0 flipflop goes to set state $Q(t+1)=1$
when J=1, K=1 flipflop complement its previous state. $Q(t+1)=Q'$

→ The feedback is taken from Q' to J so that the flipflop goes to set

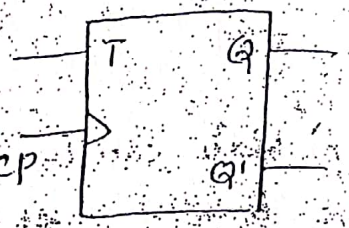
→ Similarly the feedback is taken from Q to K so that the flipflop goes to reset only when Q=1.

T-flipflop

→ T is the same input version of JK flipflop. The name T comes from the ability to toggle or complement the state when T=1.



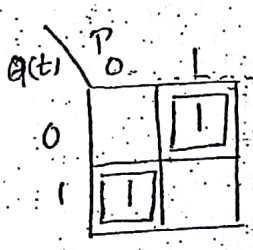
logic diagram



Graphic symbol

Q(t)	T	Q(t+1)	Comment
0	0	0	No change
0	1	1	Toggle
1	0	1	No change
1	1	0	Toggle

characteristic table



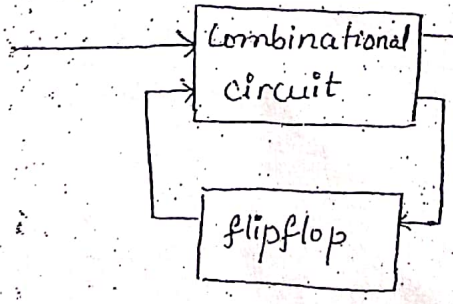
$Q(t+1) = Q'T + QT'$

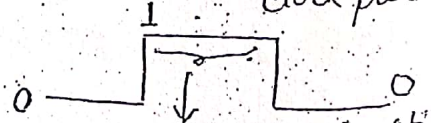
Characteristic eqn

Triggering of flipflops

→ Clocked flipflops are triggered by pulses.

→ A pulse starts from an initial value of 0 goes momentarily to 1 & after a short time returns to its initial value.





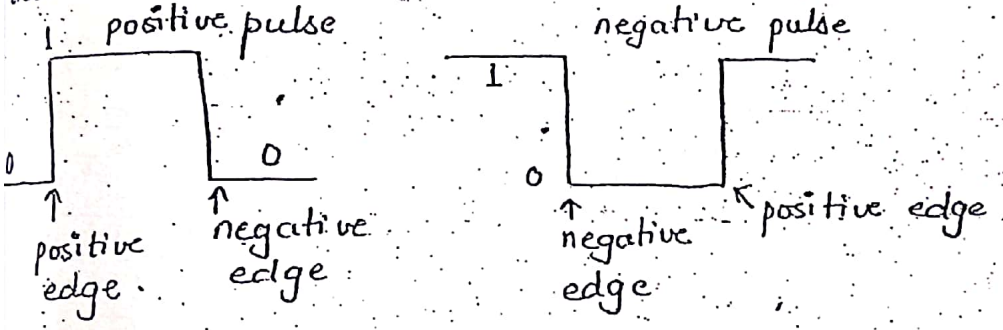
flipflop is sensitive for time duration equal

Note:- The path can produce instability if the outputs of flipflops are changing while the outputs of the combinational circuit that goes to flip-flop inputs are being sampled by the clock pulse.

→ This timing problem can be prevented if the outputs of the flipflop do not start changing until the pulse input has returned to 0.

Solution:

This problem can be solved if the flipflops are made sensitive to the pulse transition rather than the pulse duration.



→ A clock pulse may be either positive or negative.
→ The pulse goes through two signal transition from 0 to 1 and the return from 1 to 0.
→ The positive transition is defined as positive edge & the negative transition is defined as the -ve edge This definition applies also to negative pulse.

Master - slave flipflop

(Pulse - triggered master - slave)

→ Data are entered into the flipflop at the leading edge of the clock pulse but the output does not reflect the input state until the trailing edge.

→ The pulse triggered Master - slave flip-flop does not allow data to change while the clock pulse is active.

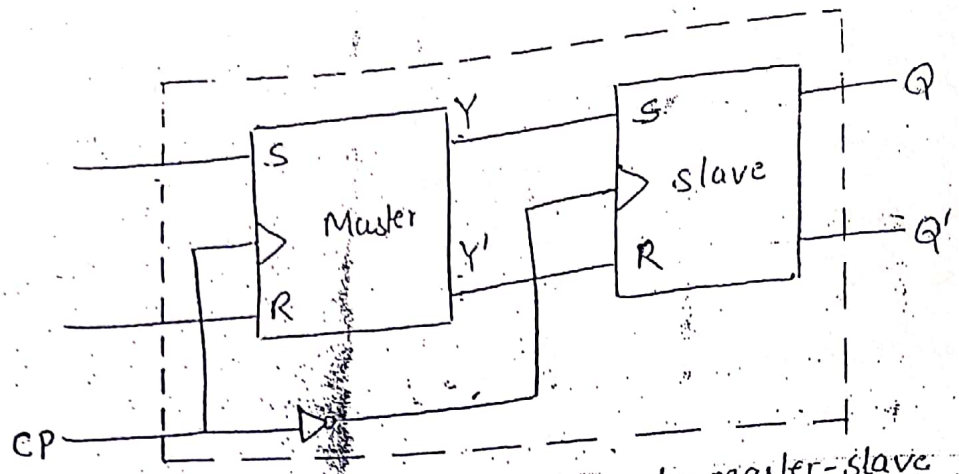


Fig: logic diagram of master-slave flip-flop

→ A master-slave flip-flop is constructed from two separate flip-flops, one circuit serves as a master and the another circuit as slave. The overall circuit is referred to as a Master-slave flipflop.

When CP = 0

→ The o/p of the inverter is 1, the slave flip-flop is enabled & Q = Y, while Q' is Y'.

→ The master is disabled because CP = 0.

When CP = 1

→ The information at the external R & S inputs is transmitted to the master flipflop. The slave flip-flop however is isolated as long as the pulse is at its 1 level.

→ When the pulse returns to 0, the master flip-flop is isolated. The slave flip-flop then goes to the same state as the master flipflop.

→ It shows that the state changes in all flip-flops coincide with the negative edge transition of the pulse. If an additional inverters is added betn the CP terminal & the input of the master flipflops Master-slave flip-flop will be sensitive to the positive edge of the clock pulse. It is called positive edge triggering.

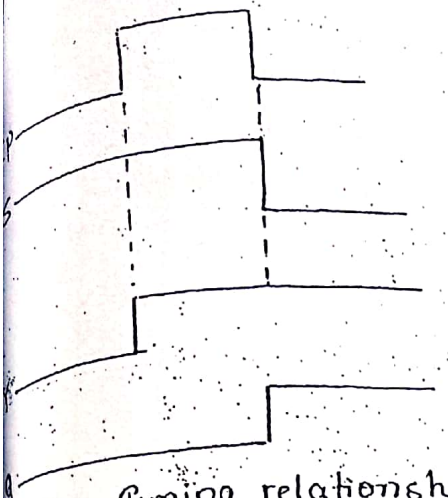
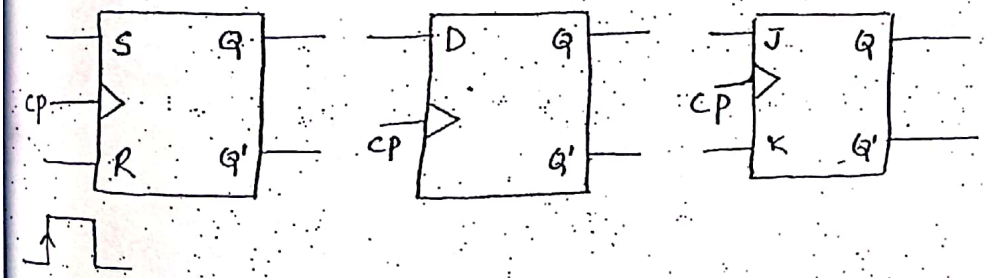
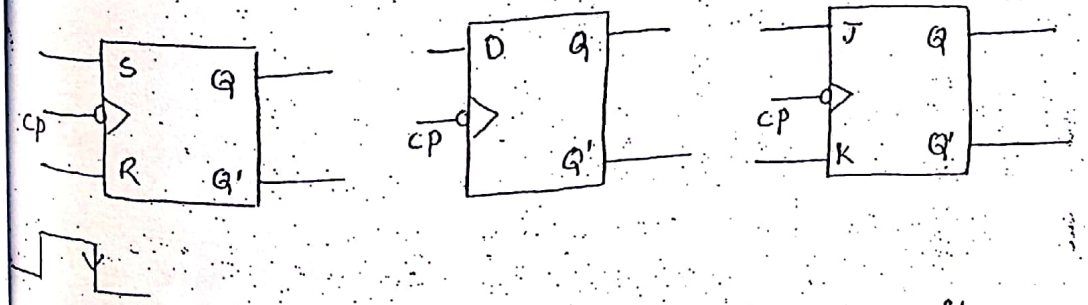


fig: Timing relationships in a master-slave flipflop

Edge triggered flip-flop

→ An edge triggered flip-flop changes states either at the positive edge (rising edge) or at the negative edge (falling edge) of the clockpulse & is sensitive to its inputs only at the clock (CP) input. This triangle is called the clynamic input indicator "▷".





Truth table for a positive edge-triggered SR flipflop

Input			output		comment
S	R	CP	Q	Q'	
0	0	X	NC	NC	No change
0	1	↑	0	1	Reset
1	0	↑	1	0	set
1	1	↑	?	?	Invalid condition

Analysis of clocked sequential circuits

(i) Analysis → logic diagram
↓
Characteristic equation
↓
State table
↓
State diagram

(ii) Design → Verbal statement
↓
State diagram
↓
State table
↓
Characteristic equation
↓
logic diagram

→ The analysis of sequential circuit consists of obtaining a table & a state diagram for the time sequence of input, output & internal states. It is also possible to write Boolean expression that describe the behaviour sequential circuit called characteristic eqn or state eqn.

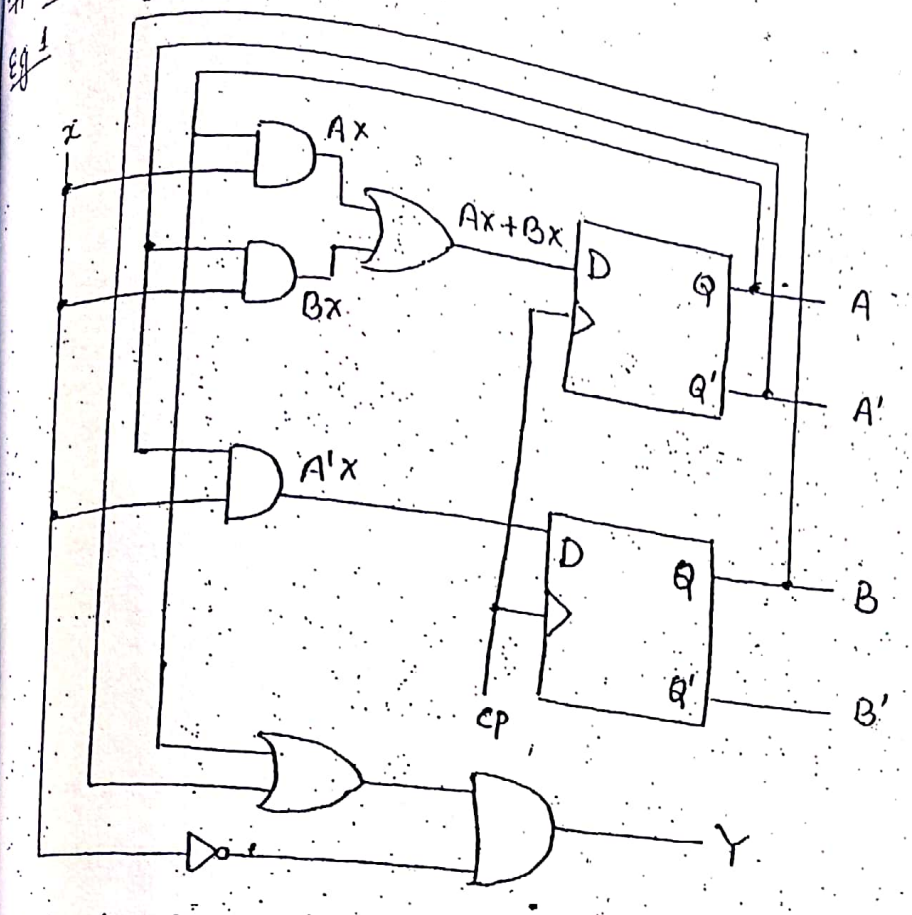
State table :-
The time sequence of inputs, outputs & flipflops states are listed in state table.

State diagram :-
→ The information available in the state table may be represented graphically in a state diagram.
→ If there are 'n' number of flipflop then there will be 2^n number of states.

State eqn or characteristic eqn :-
→ A state equation is an algebraic expression that specifies the conditions for a flipflop state transition.

Analysis

Eg 1



Characteristic eqn

$$D_A(t+1) = A(t)\,x(t) + B(t)\,x(t)$$

$$D_B(t+1) = A'(t)\,x(t)$$

Output $Y(t+1) = [A(t) + B(t)]\,x'(t)$

$\Rightarrow$ $D_A = Ax + Bx$

$D_B = A'x$

$Y = (A+B)x'$

State table: 2 flipflops so $2^2 = 4$ states

Present state A	Present state B	Next state x=0 A	Next state x=0 B	Next state x=1 A	Next state x=1 B	Output x=0 Y	Output x=1 Y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

State diagram

Mealy Model :- output are the function of the present state & inputs.

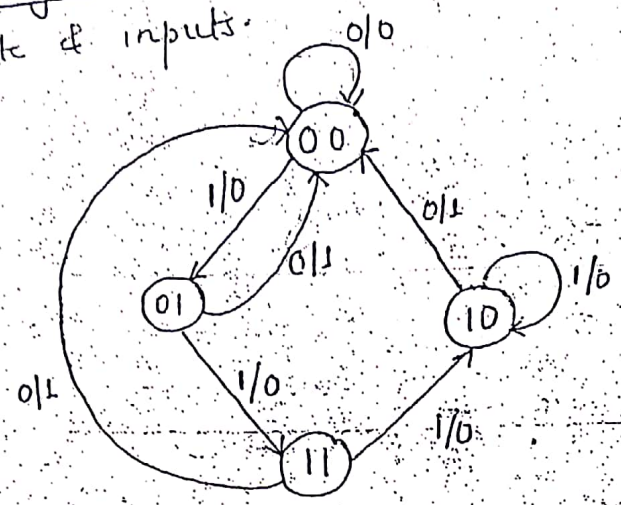
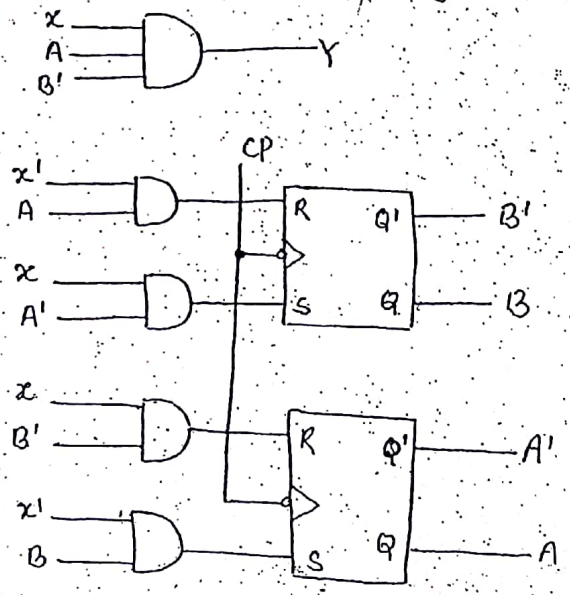


Fig: Mealy Model State diagram

In Moore Model :- Outputs are the function of the present state only [nothing is written along the direction line]

Eg 2:- Analysis the following clocked sequential logic



State Equations

Characteristic eqn of SR flipflop is

$$Q(t+1) = S + QR'$$

Here,

$R = Ax'$
$S = xA'$ } for B

$R = xB'$
$S = x'B$ } for A'

$A(t+1) = Bx' + (B'x)'A = AB + Ax' + Bx'$

$B(t+1) = A'x + (Ax')'B = A'x + A'B + Bx$

} These can also be identified by state table using K-map.

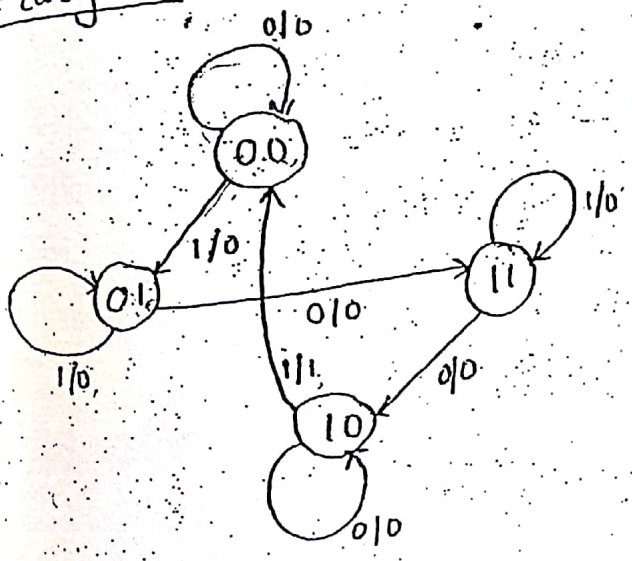
$Y = xAB'$

State table

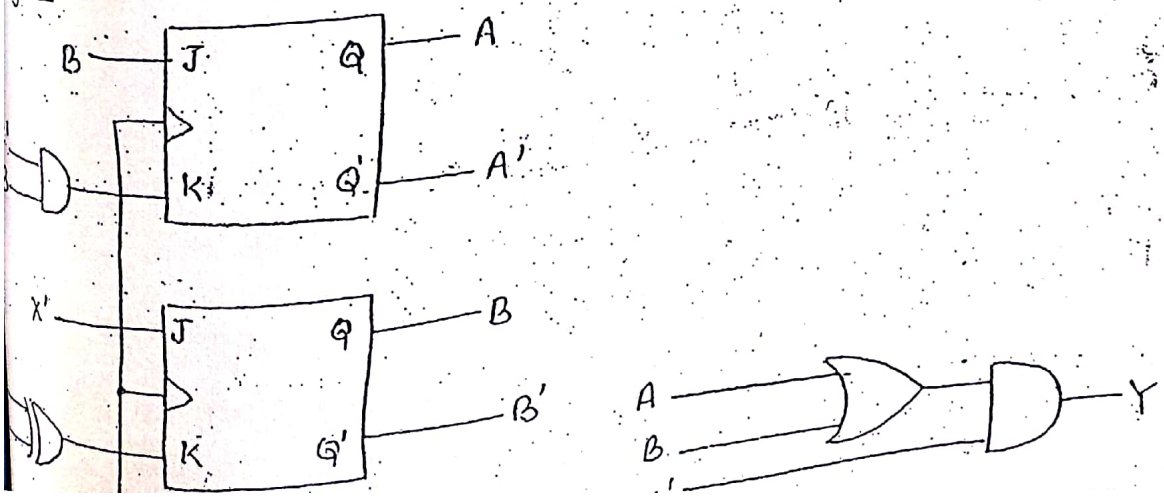
2 flipflops so $2^2 = 4$ States

Present state	Next state x=0		Next state x=1		Output x=0	Output x=1
B	A	B	A	B		
0	0	0	0	1	0	0
1	1	1	0	1	0	1
0	1	0	0	0	0	0
1	1	0	1	1		

State diagram



Eg:3



For JK flip flop

$Q(t+1) = JQ' + K'Q$

$A(t+1) = BA' + (x'B)'A$

$= A'B + Ax + AB'$

$B(t+1) = x'B' + (x \oplus A)'B$

$= x'B' + (x \odot A)B$

$= x'B' + xAB + x'A'B$

for A

$J = B$

$K = x'B$

for B

$J = x'$

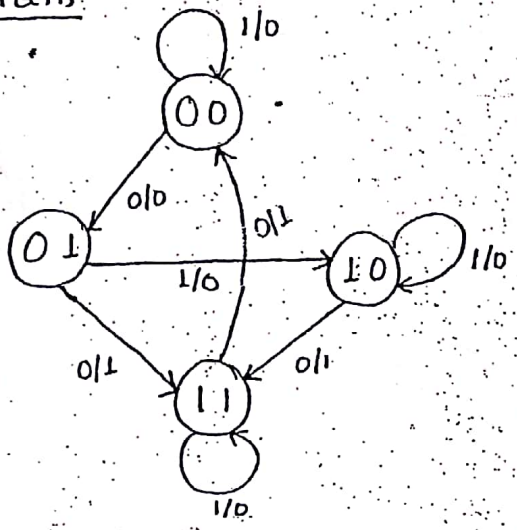
$K = X \oplus A$

$Y = (A+B)x'$

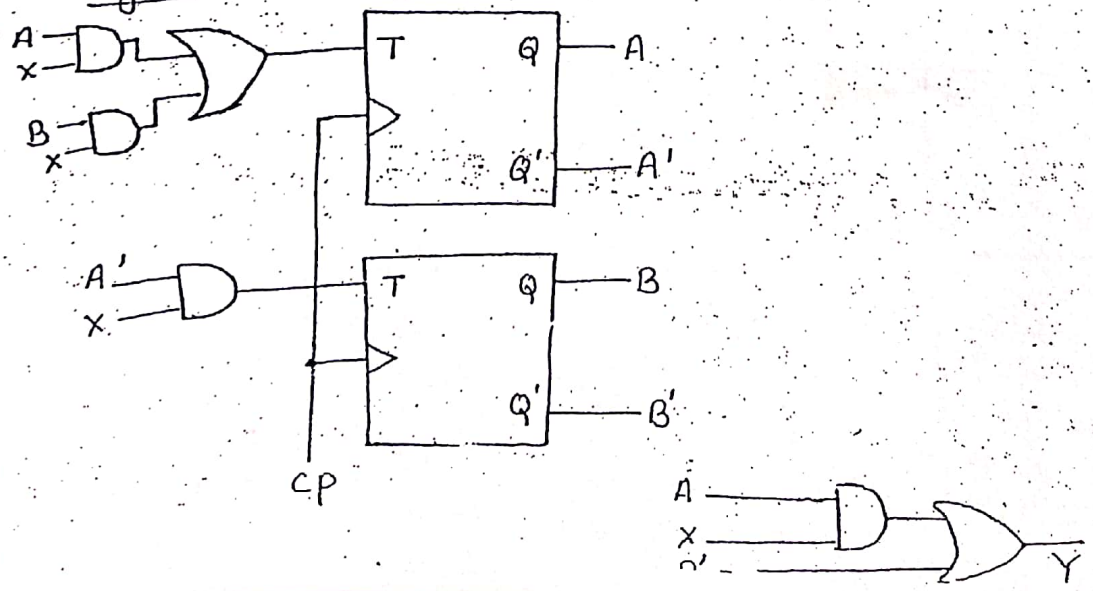
State table

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

State diagram



Eg: 4



for A

$T = Ax + Bx = x(A+B)$

$Y = Ax + B'$

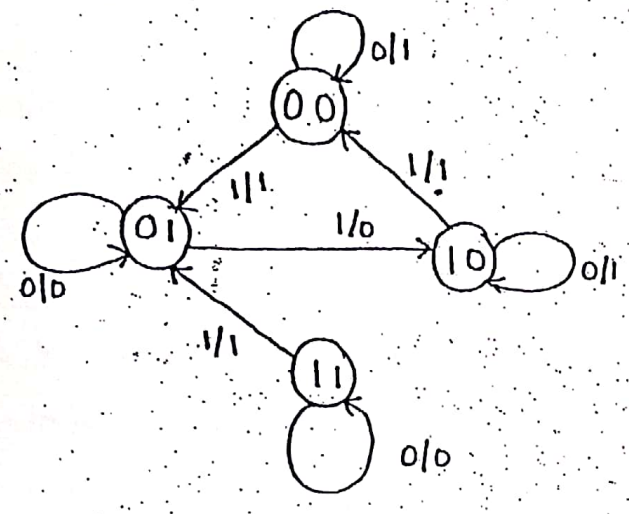
for B

$T = A'x$

State table

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

State diagram



State equation

For T flipflop

$T(t+1) = TQ' + T'Q$

Here for A flipflop

$A(t+1) = TA' + TA$

$= (Ax+Bx)A' + (Ax+Bx)'A$

$= A'Bx + \overline{Ax}\cdot\overline{Bx}\,A$

$= A'Bx + (\bar{A}+\bar{x})(\bar{B}+\bar{x})A = A'Bx + A\bar{x}\bar{B} + A\bar{x}$

$= A'Bx + AB'x' + Ax'$

for B flipflop

$B(t+1) = TB' + T'B = A'xB' + (A'x)'B$

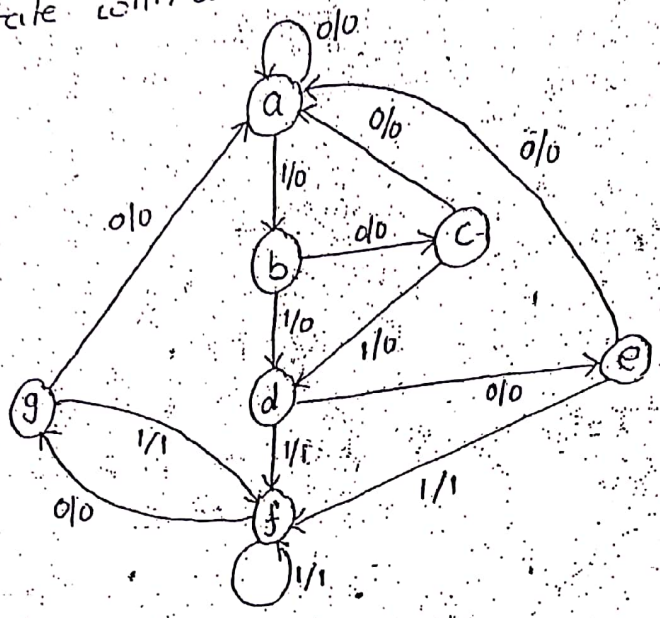
$= A'B'x + (A+x')B$

$= A'B'x + AB + Bx'$

Scanned by CamScanner

State Reduction & Assignment

→ Two states are said to be equal if they give the same output and goes to the same next state upon the application of same input. Hence we can replace one state with another.



State diagram

In the above state diagram, minimize the number of states. Consider the input sequence 0 1 0 1 0 0 1 0 1 0 starting from the initial state a.

State:	a	a	b	c	d	e	f	f	g	f	g
input:	0	1	0	1	0	1	1	0	1	0	→ (Keep it as your wish you can keep any number)
output:	0	0	0	0	0	1	1	0	1	0	

initial state = a
input applied = 0
output = 0
so next state = a

2nd
initial state = a
input = 1
output = 0
so next state = b & so on

Step 1

Find out the two present states that go to the same next state & hence the same output for the both states of input.

State table

Present state	Next state x=0	Next state x=1	Output x=0	Output x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	~~f~~ d	0	1
e	a	~~f~~ d	0	1
~~f~~ (cancelled)	~~g~~ e	f	0	1
~~g~~ (cancelled)	a	f	0	1

→ Here stage e & g both go to a & f & have same o/p 0 & 1 for the input x=0 & x=1 respectively.

→ These states g & e are equivalent so one can be cancelled.

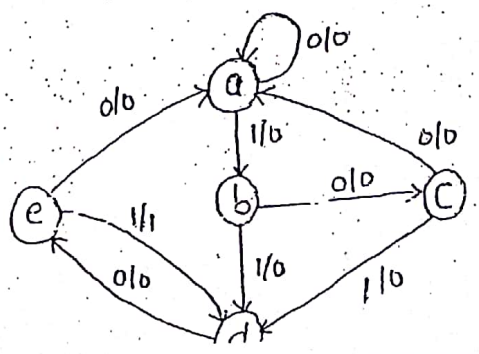
Step 2:

The row with state g at present state is crossed out & state g at all other place is replaced by e each time it occurs, in the next state column.

Reduced state table

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

Step 3



Step 4

State:	a	a	b	c	d	e	d	d	e	d	e
Input:	0	1	0	1	0	1	1	0	1	0	
Output:	0	0	0	0	0	1	1	0	1	0	

Conclusion :→ For 7 states → 3 flipflops are required. 7 states are assigned binary values & the last state can be taken for don't care condition.

→ Reduced to 5 states → also required 3 flipflop → 5 states are only assigned binary values & the last 3 states can be taken for the don't care condition.

→ In this case as the reduced state state from 7 to 5 state does not reduce the number of flipflop but the circuit will be more simple than before.

State Assignment :

State assignment procedure is the method of assigning binary values to state in such a way to reduce the cost of the combinational circuit that drives the flipflop.

State	Assignment 1	Assignment 2	Assignment 3
a	000	000	000
b	001	010	100
c	010	011	010
d	011	101	101
e	100	111	011

Three possible binary state assignment

Present state	Next state x=0	Next state x=1	output x=0	output x=1
a=000	000	001	0	0
b=001	010	011	0	0
c=010	000	011	0	0
d=011	100	011	0	1
e=100	000	011	0	1

Flip-flop Excitation table

characteristic

S	R	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	?

Excitation

Q(t)	Q(t+1)	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

SR flipflop

JK characteristic

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q'(t)

Excitation

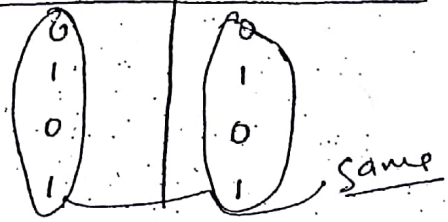
Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

D characteristic

D	Q(t+1)
0	0
1	1

Excitation

Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1



T characteristic

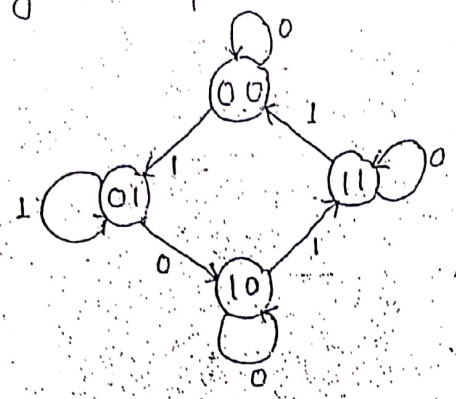
T	Q(t+1)
0	Q(t)
1	Q'(t)

Excitation

Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

Imp

Design a sequential ckt using JK flipflop.



Characteristic table for JK

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q'(t)

Excitation

Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Draw the Excitation Table
Since there are 4 states so 2 flipflops are used

Present state A	Present state B	Input x	Next state A	Next state B	flipflop inputs J_A	K_A	J_B	K_B
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	1	0	1	X	X	1
0	1	1	0	1	0	X	X	0
1	0	0	1	0	X	0	0	X
1	0	1	1	1	X	0	1	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1

K-map for J_A

A \ Bx	00	01	11	10
0				1
1	X	X	X	X

$J_A = Bx'$

for J_B

A \ Bx	00	01	11	10
0	1	X	X	
1	1	X	X	

$J_B = x$

for K_A

A \ Bx	00	01	11	10
0	X	X	X	X
1			1	

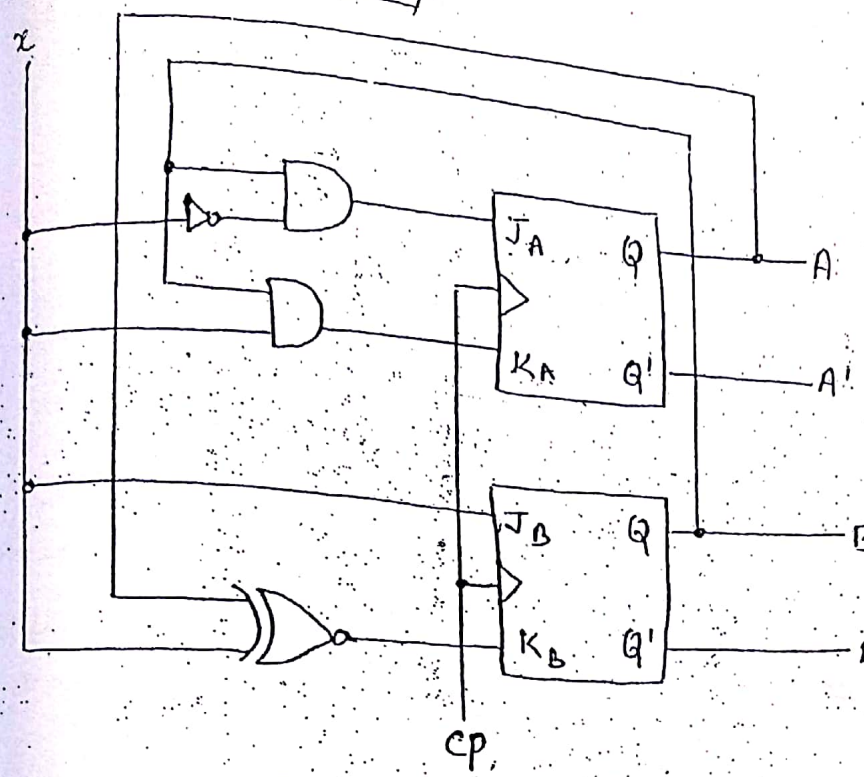
$K_A = Bx$

for K_B

A \ Bx	00	01	11	10
0	X	X		1
1	X	X	1	

$K_B = A'x' + Ax = A \odot x$

Imp



Imp

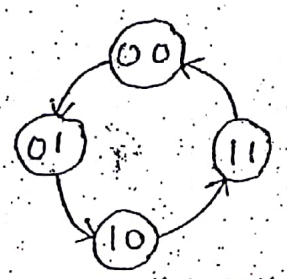
Fig: logic diagram of sequential ckt.

Design of Counter

Counter is a circuit which does not have any input and it counts repeatedly the respective sequence or states upon the application of pulse.

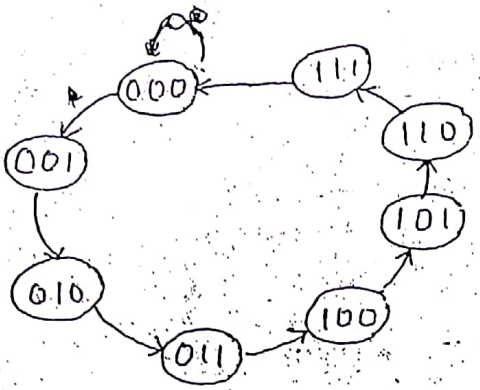
In a counter, the sequence of states may follow a binary count or any other sequence of states.

→ A counter that follows binary sequence is called binary counter. A n bit counter requires n flipflops and can count maxm of 0 to 2^n-1.



2-bit binary counter

Design a 3 bit binary counter using JK flipflop.



characteristic for JK

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q'(t)

Excitation fo JK

Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Excitation table

Present state			Next state			flipflop Inputs					
A	B	C	A	B	C	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	0	0	0	X	1	X	1	X	1

K-map for J_A

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

$J_A = BC$

for J_B

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

$J_B = C$

for J_C

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

$J_C = 1$

for K_A

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

$K_A = BC$

for K_B

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

$K_B = C$

for K_C

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

$K_C = 1$

Chapter 8
Register, Counter & Memory Unit

Prepared by
Budeltha Laxmi Mahar

Sequential circuit are classified into three categories
Registers
Counters
Memory Unit

Register :-

→ A register is a group of binary storage cells suitabe for holding binary information.
→ It is a group of flipflops & gates.
→ An n bit register has a group of n flipflops & is capable of storing any binary information counting n bits.

Counters :-

→ A counter is essentially a register that goes through a predetermined sequence of states upon the application of input pulses.

Memory Unit :-

→ It a collection of storage cells together with associated circuits needed to transfer information in and out of storage.

Register

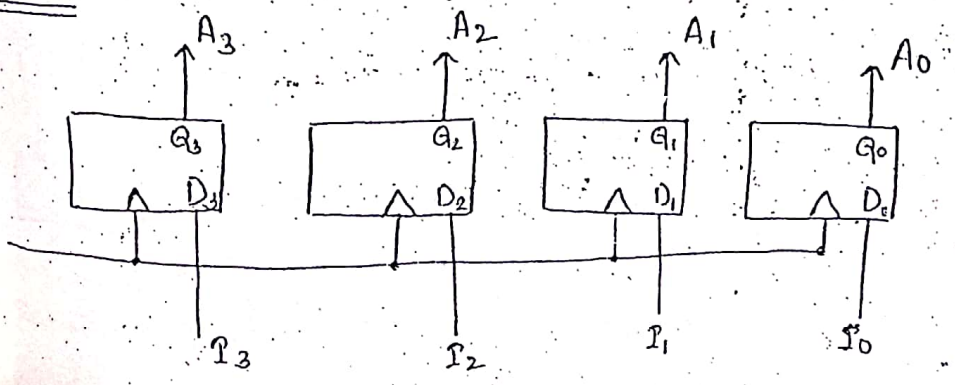


Fig: 4 bit register

→ A group of flipflops sensitive to pulse duration is usually called as Gated latch.

→ A group of flipflops sensitive to pulse transition is called Register.

→ Gated latch is used for temporary storage.

Register with Parallel load

→ The transfer of new information into a register is referred to a loading the register.

→ If all bits of the register are loaded simultaneously with a single clock pulse we say that the loading is done in parallel.

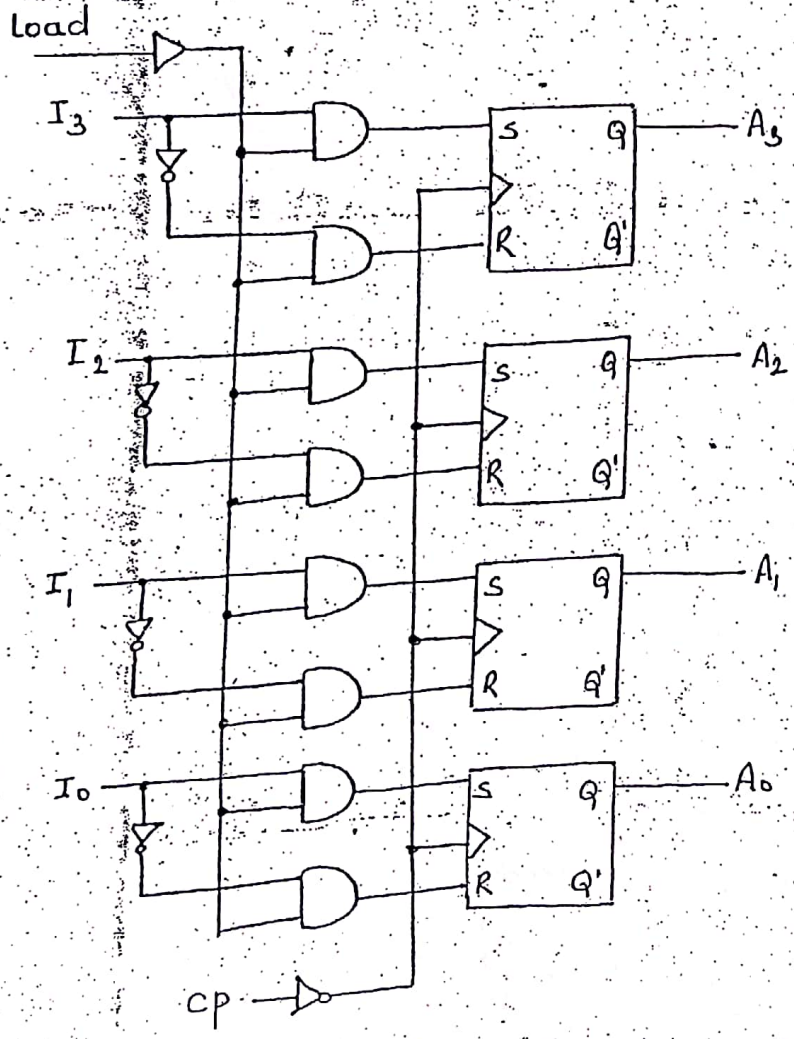
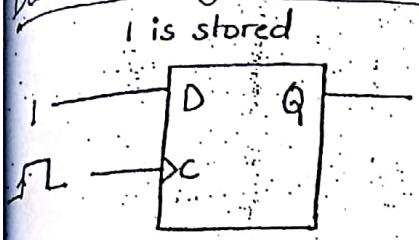


Fig: 4-bit register with parallel load

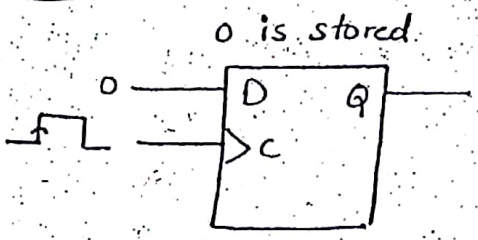
When load = 1, I_0, I_1, I_2, I_3 is loaded with falling edge of the clock pulse. Clear is maintained at high level for the synchronous operation.

Shift Register

A register is a digital circuit with two basic functions, Data storage & data movement.

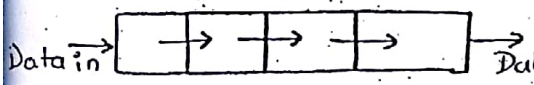


When a 1 is on D Q becomes a 1 at the triggering edge of clock or remains 1 if already in a SET stage.

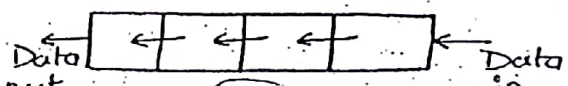


When a 0 is on D, Q becomes a 0 at the triggering edge of clock or remains a 0 if already in the RESET state.

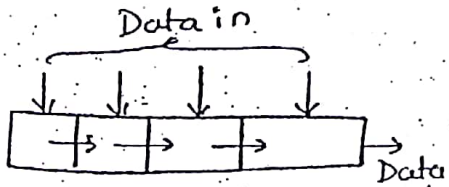
→ The shifting capability of a register permits the movement of data from stage to stage within the register or in to or out of the register upon application of clock pulses.



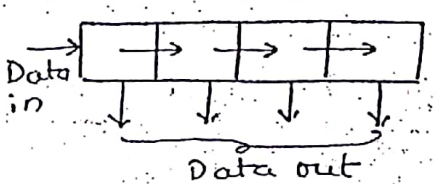
a) Serial in/shift right/serial out



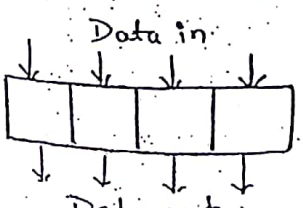
b) Serial in/shift left/serial out



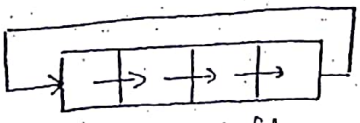
c) Parallel in | serial out



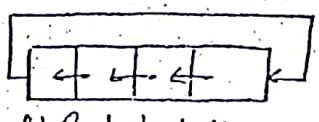
d) Serial in / parallel out



e) Parallel in | parallel out



f) Rotate right



f) Rotate left

Fig: Basic data movement in shift registers

i) Serial In/ Serial Out shift Register

→ A register capable of shifting its binary information either to the right or to the left is called a shift register.

→ Serial In / serial Out shift register accepts data serially that is one bit at a time on a single line. It produces the stored information on its output also in a serial form.

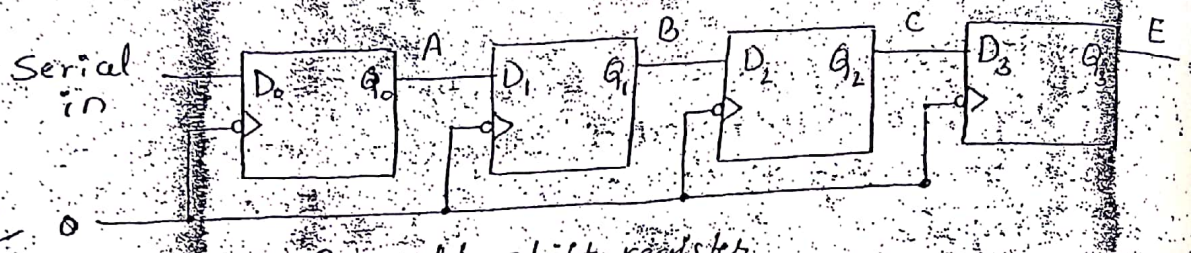


Fig: The right shift register

→ The serial Input determines what goes into the left-most flipflop during the shift. The serial output is taken from the output of the right-most flipflop prior to the application of a pulse.

→ Let us assume the input sequence is 1010. Then the shifting sequence table follows as below.

	A	B	C	E
Initial content	0	0	0	0
CLK ↑	1	0	0	0
CLK ↑	0	1	0	0
CLK ↑	1	0	1	0
CLK ↑	0	1	0	1

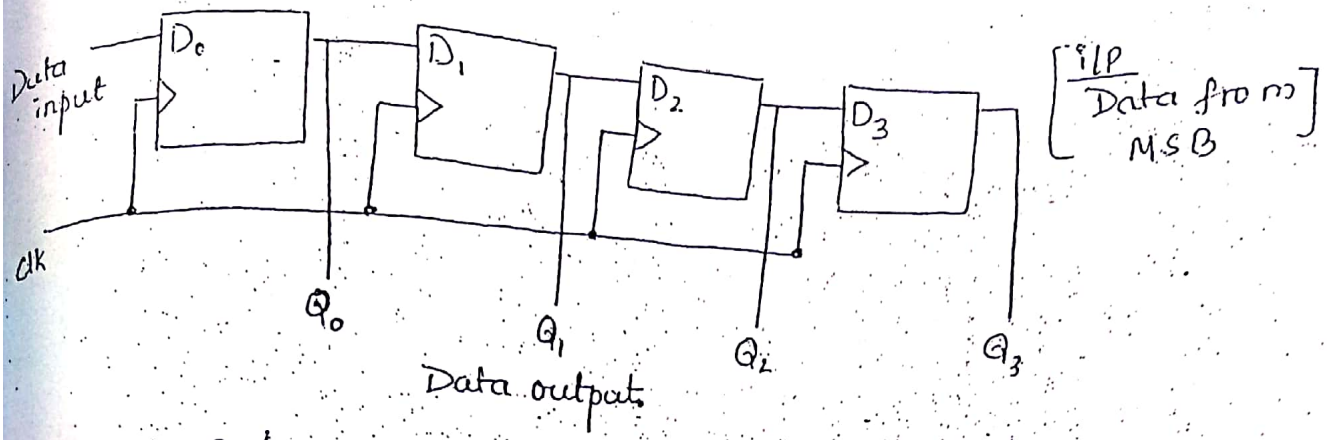
(Data from MSB)

right shifting the data

ii) Serial In/ Parallel Out Shift Register

→ Data bits are entered serially (right-most bit first) into this type of register in the same manner. The difference is the way in which the data bits are taken out of the register; in the parallel out register, the output of each stage is available.

87



Parallel In/ serial Out shift Register :-

→ For a register with parallel data inputs, the bit are entered simultaneously into their respective stages on parallel lines rather than on a bit by bit basis on one line as with serial data inputs.

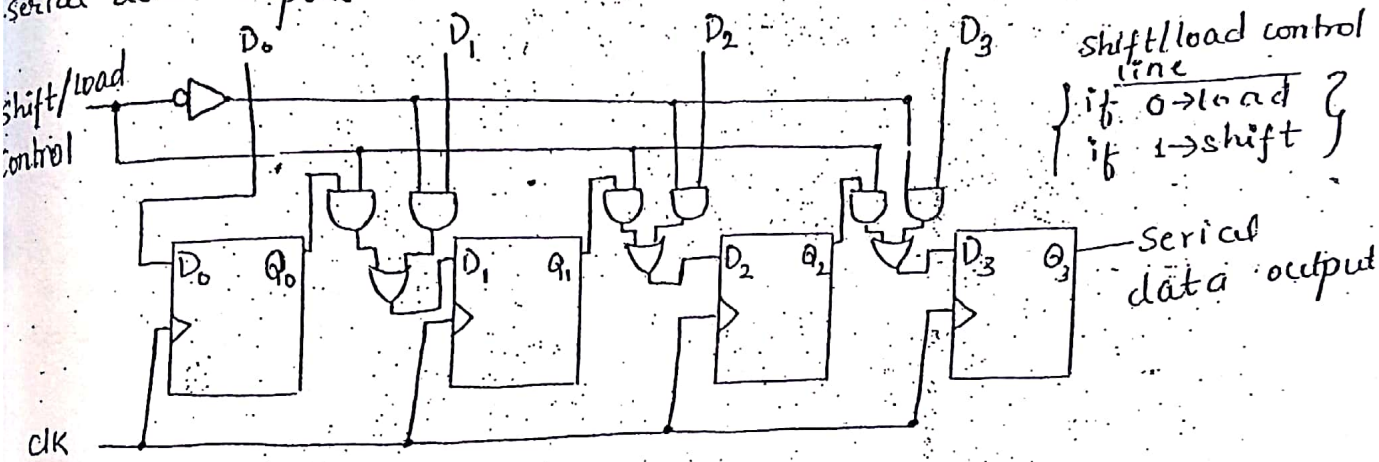


Fig: logic diagram

Parallel In/ Parallel Out shift Register

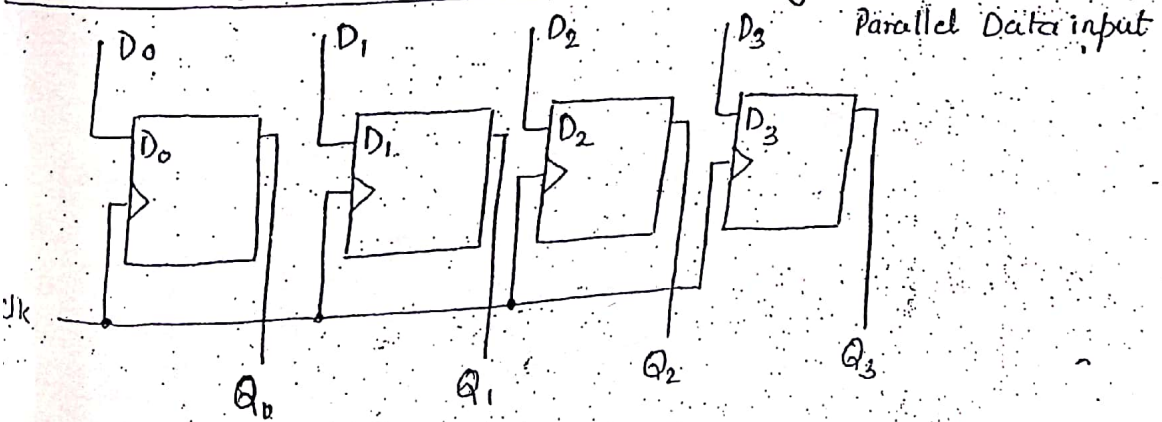
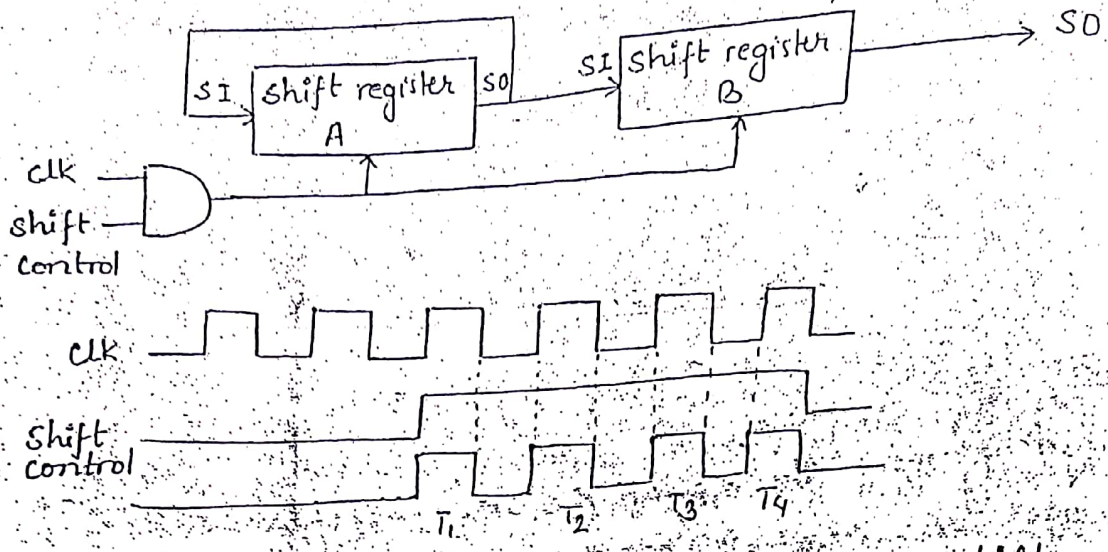


Fig: logic diagram

Serial Transfer



Let us assume that the initial content of shift register A = 1011 & that of B = 0010

Time pulse	shift register A	shift register B	serial o/p at B
Initial value	1 0 1 1	0 0 1 0	0
After T_1	1 1 0 1	1 0 0 1	1
After T_2	1 1 1 0	1 1 0 0	0
After T_3	0 1 1 1	0 1 1 0	0
After T_4	1 0 1 1	1 0 1 1	1

Serial Adder (Assignment, refer book)

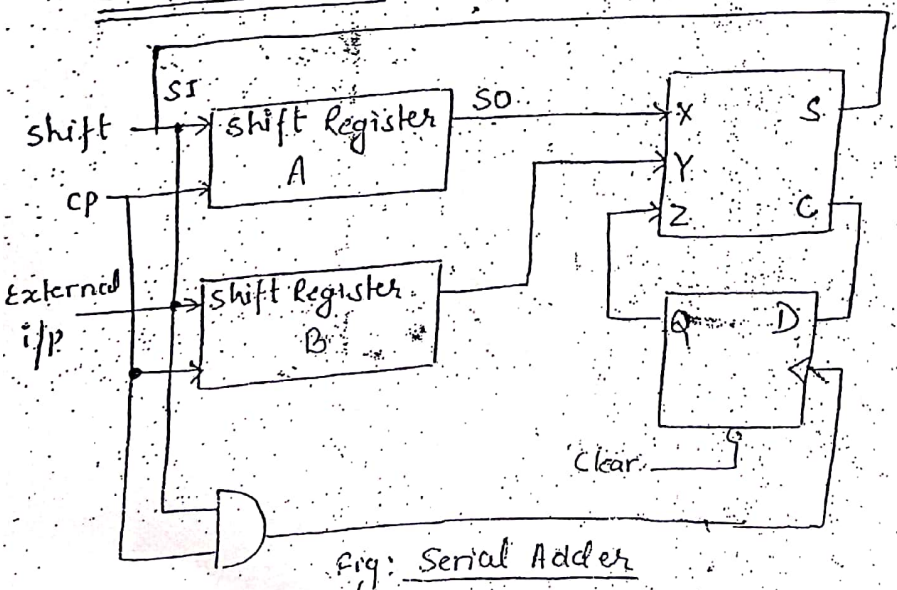


fig: Serial Adder

Assignment

* Design a serial adder using sequential logic procedure by using JK flipflop. (refer book)
* Bidirectional shift Register with parallel load. (Refer Book)

Counter

- Synchronous: All flipflops are triggered by the same clock pulse
- Asynchronous: The o/p of the one flipflop is used to trigger the another flipflop.

Binary Ripple Counter

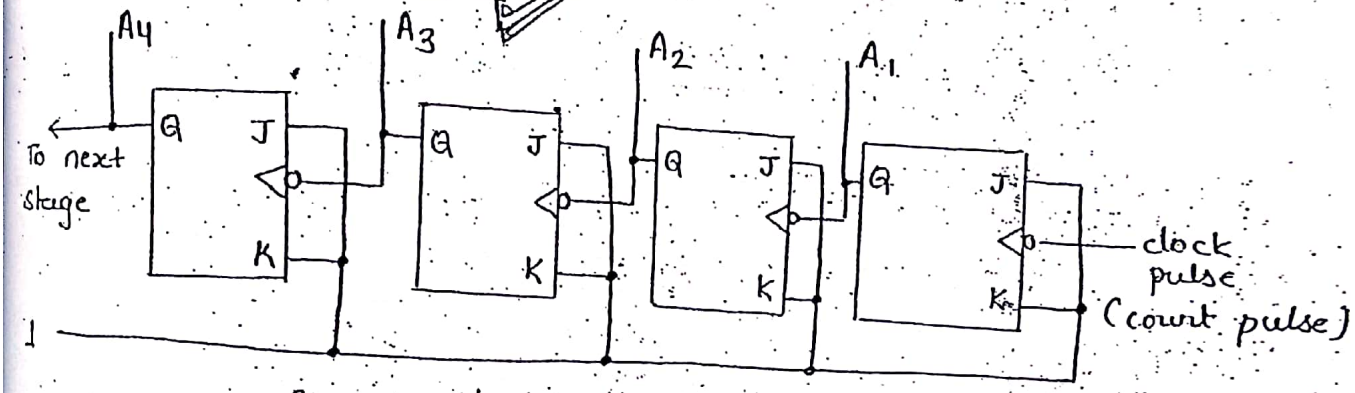


Fig: 4-bit binary ripple counter

Whenever the first flipflop get the count pulse it complements the previous state i.e It is always in the complementing.

Count sequence				conditions for complementing flipflops
A_4	A_3	A_2	A_1	
0	0	0	0	complement A_1
0	0	0	1	complement A_1, A_1 will go from 1 to 0 & complement A_2
0	0	1	0	complement A_1
0	0	1	1	complement A_1, A_1 will go from 1 to 0 & complement A_2; A_2 " " " 1 to 0 & " A_3
0	1	0	0	complement A_1
0	1	0	1	complement A_1, A_1 will go from 1 to 0 & complement A_2
0	1	1	0	complement A_1
0	1	1	1	complement A_1, A_1 will go from 1 to 0 & complement A_2; A_2 " " " 1 to 0 & " A_3
1				A_1 " " " 1 to 0 & " A_2

→ Binary ripple counter consists of a series connection of complementing flipflops, with the output of each flipflop connected to the CP input of the next higher-order flipflop.
→ The flipflop holding least significant bit receives the incoming count pulse.
→ The LSB A_1 must be complemented with each count pulse.
→ Every time A_1 goes from 1 to 0, it complements A_2. Every time A_2 goes from 1 to 0, it complements A_3, and so on.
→ Ripple counters are sometime called asynchronous counters.
→ A binary counter with a reverse count is called a binary down-counter. The above circuit functions as a binary down counter if the output are taken from the complement terminals Q' of all flipflops.

BCD Ripple Counter

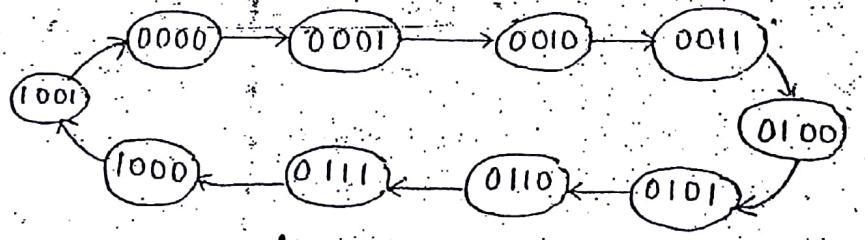


fig: state diagram of a decimal BCD counter.

→ Counters can also be designed to have a number of states in their sequence that is less than the maximum of 2^n states. Counter with ten states in their sequence are called decade counter.
→ To obtain a truncated sequence, it is necessary to force the counter to recycle before going through all of its possible states.
→ For example: the BCD decade counter must recycle back to the 0000 state after the 1001 state. NAND gate is used to decade the next count (1010) and the output of the NAND gate is applied to the clear (CLR) inputs of the flipflops.

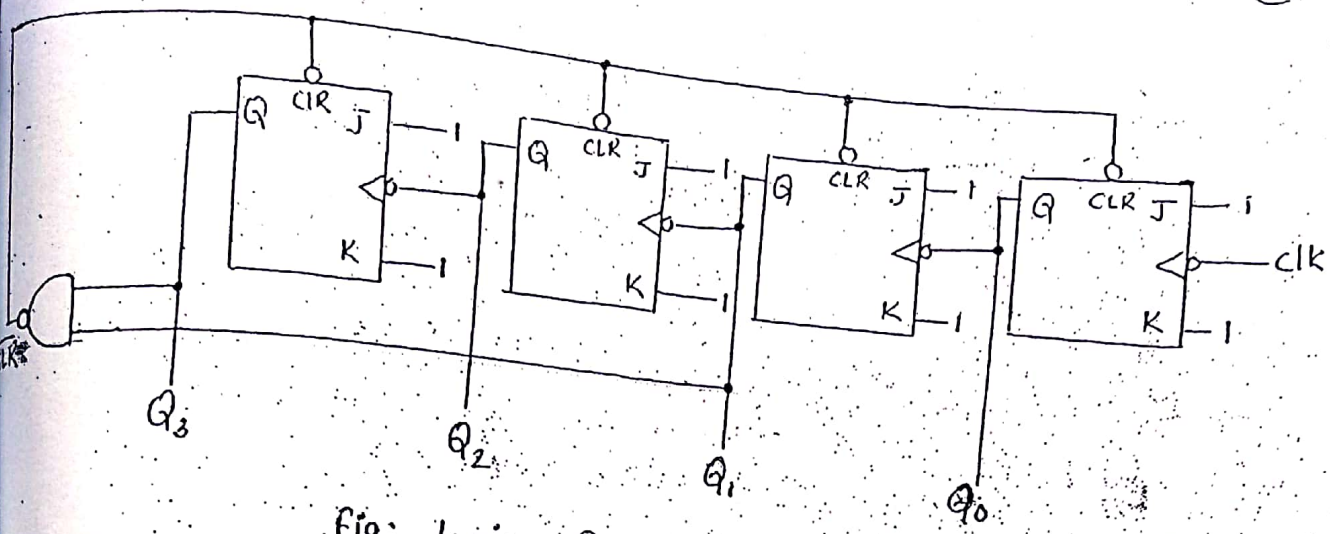


Fig: logic Implementation of asynchronous decade counter (BCD Ripple counter)

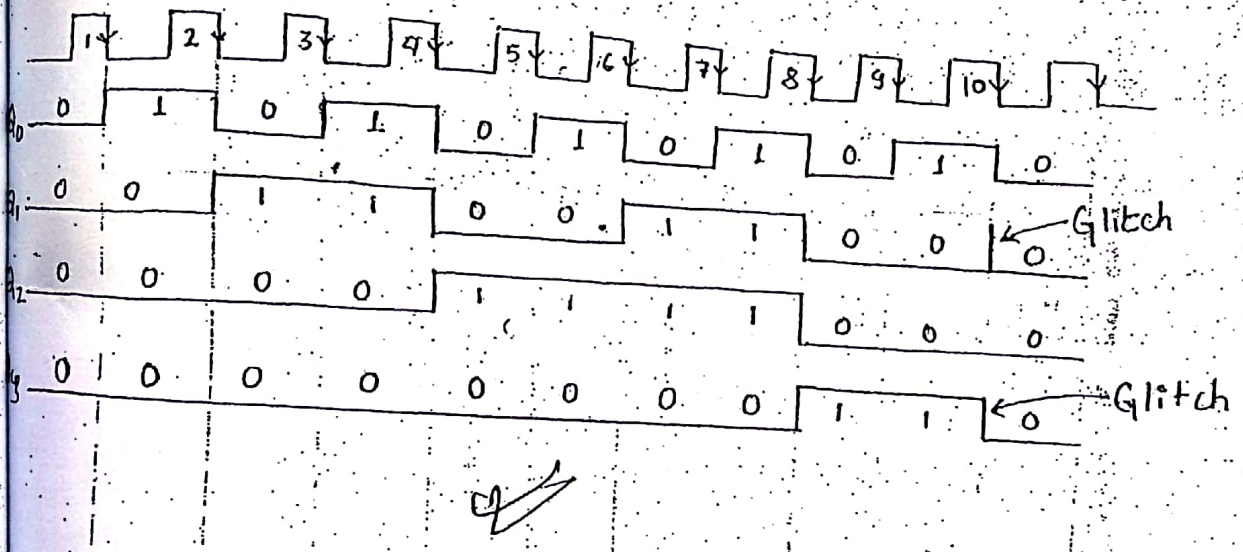
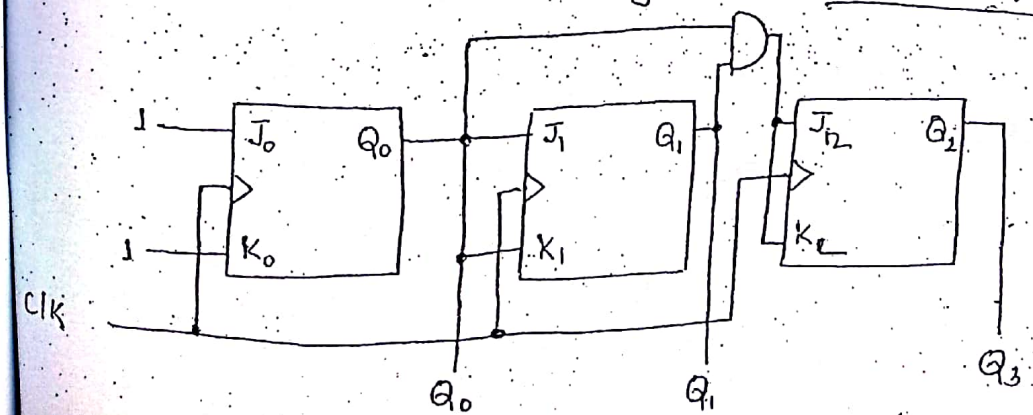


Fig: Timing diagram of a synchronous BCD ripple counter

Synchronous Counter

→ Synchronous means that all the flipflop in the counter are clocked at the same time by a common clock pulse.



3-bit synchronous binary counter

Timing diagram

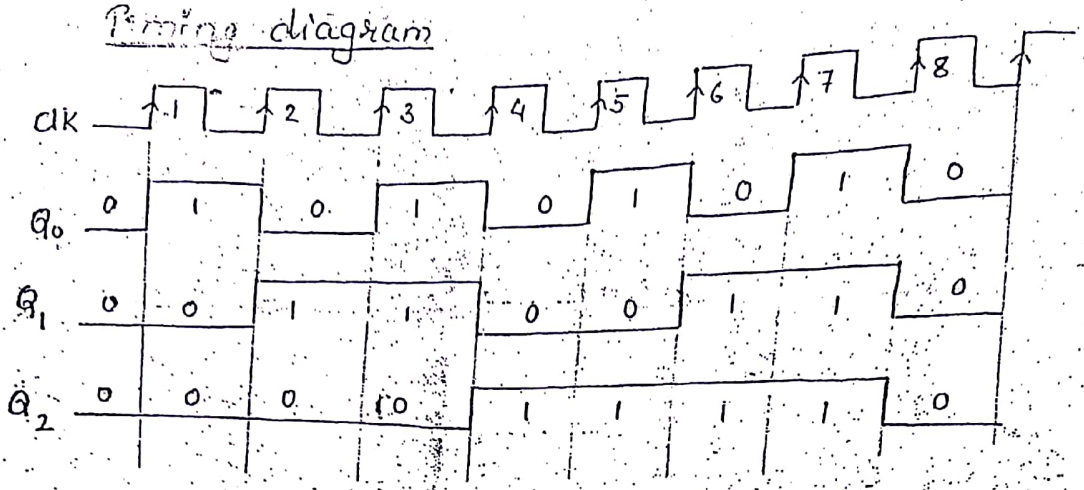


Fig: Timing diagram of 3 bit synchronous binary counter

* Note

Synchronous counter can be designed by following the designing procedure as in the previous chapter.

Design of synchronous counter include following steps.

1) state diagram
2) Excitation table of the flipflop being used
3) k-map & simplified Boolean expression
4) logic implementation.

Ring Counter

→ A ring counter is a type of counter composed of circular shift register.

→ The output of last shift register is feed back to the input of first register.

→ In ring counter circulation is done by shifting with only one flipflop being at a time & all other are cleared.

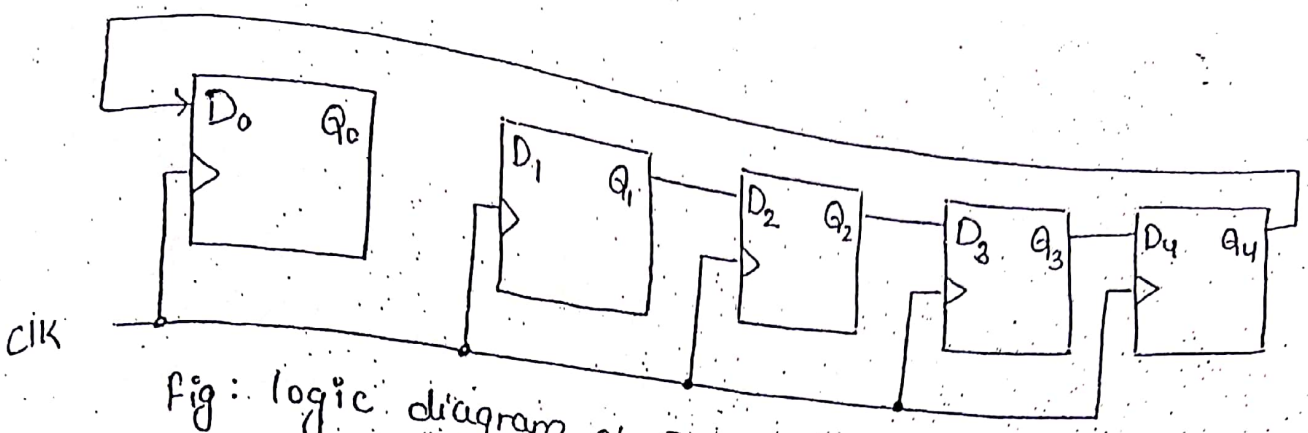


Fig: logic diagram of 5 bit ring counter

CLK	Q_0	Q_1	Q_2	Q_3	Q_4
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	0
4	0	0	0	0	1
5	1	0	0	0	0
6	0	1	0	0	0

Fig: Sequence table

Johnson Counter :- D→

→ In a Johnson counter the complement of the output of the last flip-flop is connected back to the D input of the first flipflop.

→ This feedback arrangement produces a unique sequence of states so called twisted ring counter.

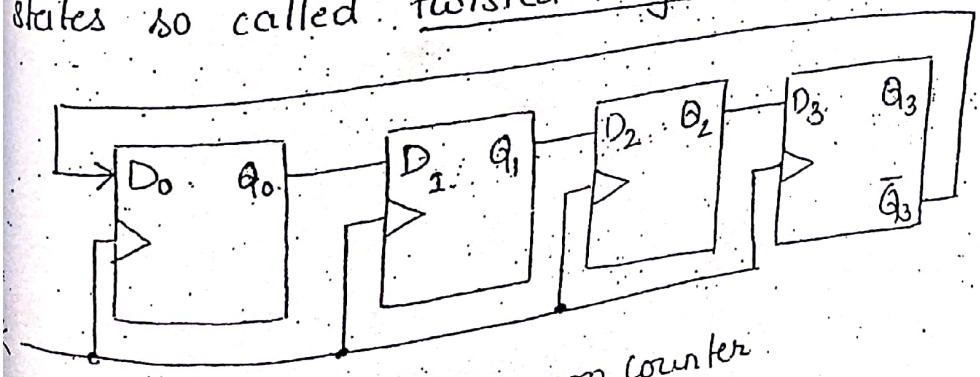


Fig: 4-bit Johnson counter

Clock pulse	Q_0	Q_1	Q_2	Q_3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

Fig: Sequence table of 4-bit Johnson counter

→ The 4-bit sequence has a total of 8 states or bit pattern. Similarly the 5-bit sequence has a total of 10 states.

Assignment

Design a 4-bit binary synchronous up-down counter.

Design a 4-bit binary counter with parallel load.

The Memory Unit

The register in a digital computers may be classified as either operational or storage type.

Operational register

An operational register is capable of storing binary information in its flip-flops and in addition has combinational gate capable of data processing tasks.

Storage register

→ A storage register is used solely for temporary storage of binary information. This information can not be altered when transferred in and out of the register.

→ A memory unit stores binary information in groups called words. each word being stored in a memory register.

Memory Unit

→ A memory unit is a collection of storage registers together with the associated circuit needed to transfer information in & out of the registers.

→ The storage registers in a memory unit are called memory registers.

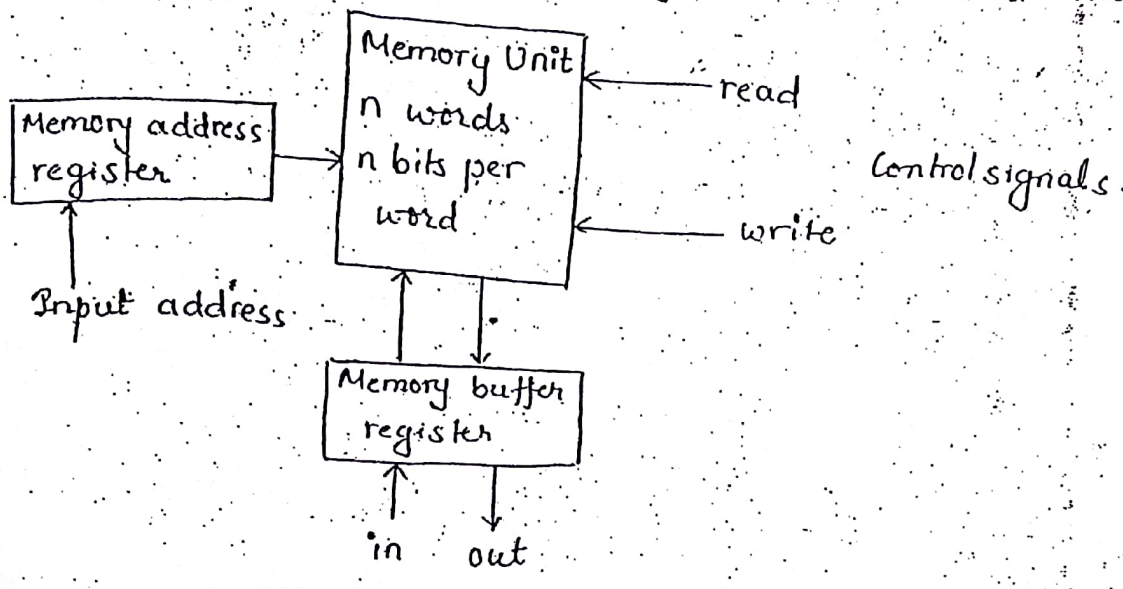


Fig: Block diagram of a memory unit showing communication with environment.

→ The communication between a memory unit and its environment is achieved through two control signals & two external registers.

Memory address Register (MAR) :-

→ It specifies the memory word selected.

Read control signal

→ A read signal specifies a transfer out function.

Write control signal

→ A write signal specifies transfer in function.

Memory Buffer Register (MBR) :-

→ The information transfer to and from registers in memory & the external environment is communicated through the common

Integrated - Circuit Memory

RAM
↓ Bg.

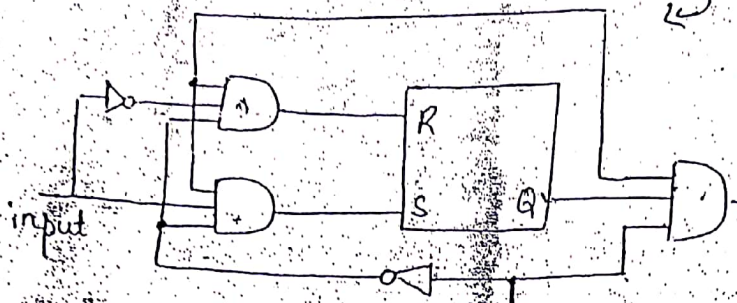


Fig: logic diagram

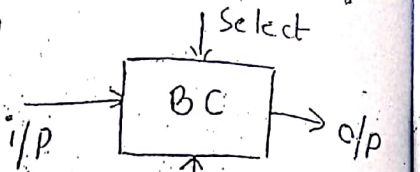
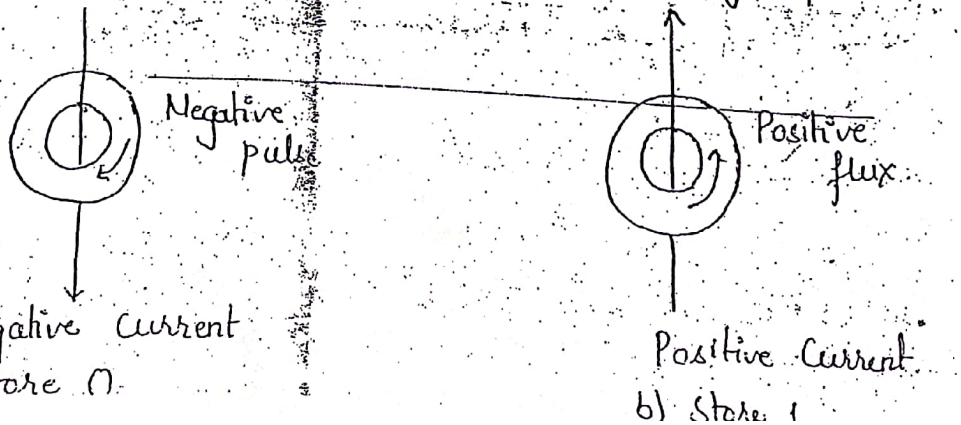


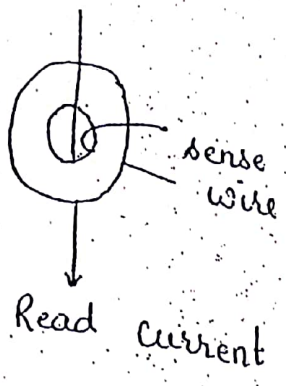
Fig: Block diagram of Memory cell

→ The binary cell has three inputs & one output.
→ The select input enables the cell for reading or writing.
→ The read/write input determines the cell operation when it is selected.
→ A 1 in read/write input forms a path from the flipflop to the output terminal. The information in the input terminal is transferred into the flipflop when the read/write control is 0.

Magnetic - core memory :-

→ A magnetic core memory was magnetic core to store binary information.
→ A magnetic core employs three physical quantities, current, magnetic flux & voltage for its operation.
→ The signal that excites the core is a current pulse in a wire passing through the core.
→ The binary information stored is represented by the direction of magnetic flux within the core.
→ The output binary information is extracted from a wire linking the core in the form of a voltage pulse.





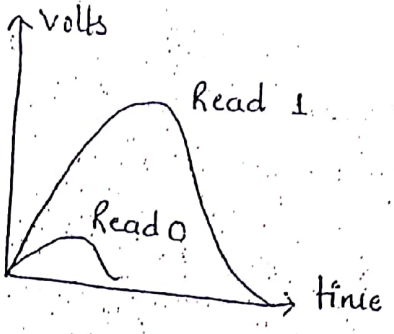


Fig:- Reading a bit from a magnetic core. Output of sense wire

→ Read out is accomplised by applying a current in the negative direction. If the core is in the 1 state, the current reverse the direction of magnetigation & the resulting change of the flux produces a voltage pulse in the sense wire.

→ If the core is already in the 0 state, the negative current leaves the core magnetized in the same direction causing a very slight disturbance of magnetic flux which results in a very small output voltage in the sense wire.

→ Note that this is a destructive read out, since the read current always returns the core to the 0 state. The previously stored value is lost

RAM (Random Access Memory):-

→ A memory unit is a collection of storage cell together with associated circuit needed to transfer information in and out of the device.

→ Memory cells can be accessed for information transfer to or from any desired random location & hence the name Random access memory.

→ A memory unit stores binary information in group of bits called words. A word in memory is an ability of bits that moves in and out of storage as a unit.

→ The capacity of a memory unit is usually stated as the total

→ The communication between a memory & its environment is achieved through data input & output lines, address selection lines, & control lines that specify the direction of transfer.

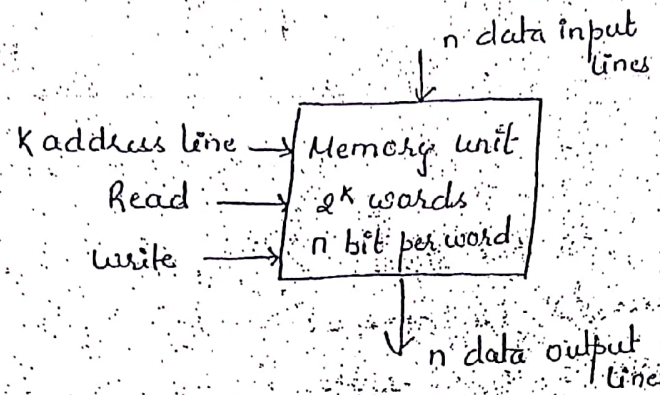


fig:- Block diagram of a memory unit

The memory unit is specified by the numbers of words it contains & the number of bits in each word.

Memory address		
Binary	Decimal	
00000 00000	0	0001 1111 0011 1110
00000 00001	1	0010 1010 0101 0111
00000 00010	2	
⋮	⋮	
11111 11101	1021	
11111 11110	1022	
11111 11111	1023	0000 1111 0011 0011

Fig: Content of a 1024 × 16 memory

Control Inputs to Memory Chip

Memory Enable	Read/write	Memory Operation
0	X	None
1	0	Write to selected word
1	1	Read from selected word

Access time:

→ The access time of a memory is the time required to select a word and either read or write it.

→ In RAM, the access time is always the same regardless of the particular location of the word.

→ In a sequential access memory, the time it makes to access

a word depends on the position of the word with respect to the reading-head position & therefore the access time is variable.

Types of Memories

→ Integrated-circuit RAM units are available in two possible operating modes.

i) Static (use flipflop)

ii) dynamic (MOS transistors)

→ Memory units that lose the stored information when the power is turned off are said to be volatile.

→ Integrated-circuit RAMs, both static & dynamic are of this category since the binary cells need external power to maintain the stored information.

→ In contrast, nonvolatile memory, such as magnetic disk, retains its stored information after removal of power. This is because the data stored on magnetic compounds is manifested by the direction of magnetization, which is retained after power is turn off. Another non volatile memory is the read only memory (ROM).

Hazards: (2012, 2017, 2009)

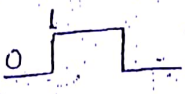
→ Hazards are unwanted switching transients that may appear at the output of a circuit because different paths exhibit different propagation delays.

→ Hazards occur in combinational circuits where they may cause a temporary false-output value. When this condition occurs in asynchronous sequential circuits it may result in a transition of a wrong stable stat.

→ A hazard is a condition where a single variable change produces a momentary output change when no output change should occur.

Types of hazard

1) Static 1 - hazard 

2) Static 0 - hazard 

3) Dynamic hazard

Dynamic hazard causes the output to change three or more times it should change from 1 to 0 or vice versa.

combinational ckt	sequential ckt
output variable at any instant of time are depend only on the present input variable.	① The output variable at any time instant are depend on the present & past o/p variable.
① It does not have a memory	② It has memory
It does not have feed back	③ It has a feed back
② They are fast	④ They slow.
③ Easy to Design	⑤ comparatively hard to design
Building block are basic gate	⑥ Building block basic gate (combinational)

Synchronous	Asynchronous sequential ckt
They are external clock pulse for transition from one to next state	① They do not need external clock pulse for transition from one to next.
② slower than Asynchronous sequential ckt	② faster than synchronous ckt
③ More reliable.	③ less Reliable.
④ Required more hardware so more cost.	④ Require less Hardware so cheap.
⑤ common clock pulse may be use.	⑤ common clock pulse can be use.

a word depends on the position of the word with respect to the reading-head position & therefore, the access time is variable.

Types of Memories

→ Integrated-circuit RAM units are available in two possible operating modes.
  i) Static (use flipflop)
  ii) dynamic (MOS transistors)

→ Memory units that lose the stored information when the power is turned off are said to be volatile.

→ Integrated-circuit RAMs, both static & dynamic are of this category since the binary cells need external power to maintain the stored information.

→ In contrast, nonvolatile memory, such as magnetic disk, retains its stored information after removal of power. This is because the data stored on magnetic compounds is manifested by the direction of magnetization, which is retained after power is turn off. Another non volatile memory is the read only memory (ROM).

Hazards: (2012, 2017, 2009)

→ Hazards are unwanted switching transients that may appear at the output of a circuit because different paths exhibit different propagation delays.

→ Hazards occur in combinational circuits where they may cause a temporary false-output value. When this condition occurs in asynchronous sequential circuits it may result in a transition of a wrong stable stat.

→ A hazard is a condition where a single variable change produces a momentary output change when no output change should occur.

Arithmetic Logic Unit (ALU)

prepared by
Buddha Laxmi Ma

Arithmetic logic Unit

→ An arithmetic logic Unit is a multioperation, combinational-logic digital function.

→ It can perform a set of basic arithmetic operations and a set of logic operations.

→ The ALU has a number of selection lines to select a particular operation in the unit.

→ The selection lines are decoded within the ALU so that K selection variables can specify upto 2^k distinct operations.

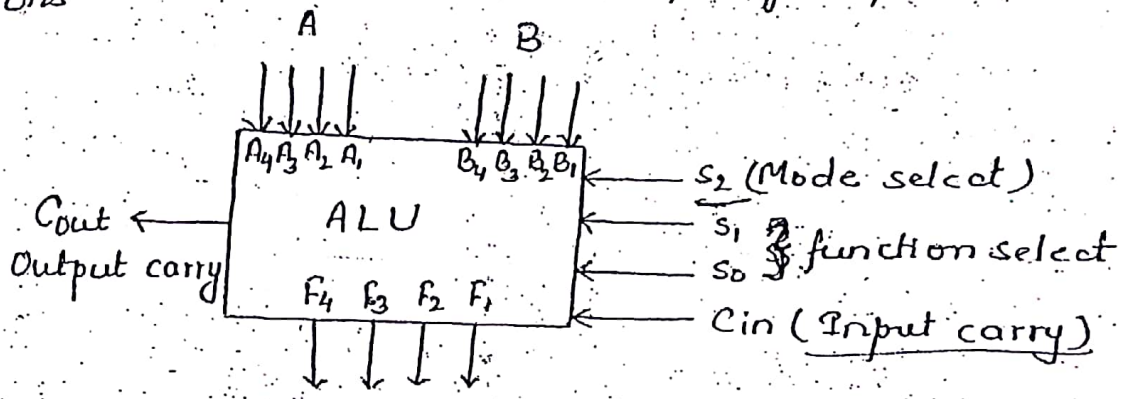


Fig: Block diagram of a 4-bit ALU

→ The four data inputs from A are combined with the four inputs from B to generate an operation at the F output.

→ The mode-select input S_2 distinguishes between arithmetic & logic operations. The two function-select inputs S_1 & S_0 specify the particular arithmetic or logic operation to be generated.

→ With three selection variables, it is possible to specify four arithmetic operations & four logic operations.

→ The i/p & o/p carry has meaning only during arithmetic operation.

The input carry in the least significant position of an ALU is quite often used as a fourth selection variable that can double the number of arithmetic operations.

#Design of Arithmetic Circuit

→ The basic component of the arithmetic section of ALU is a parallel adder. By controlling the data i/p to the parallel adder, it is possible to obtain different types of arithmetic operations.

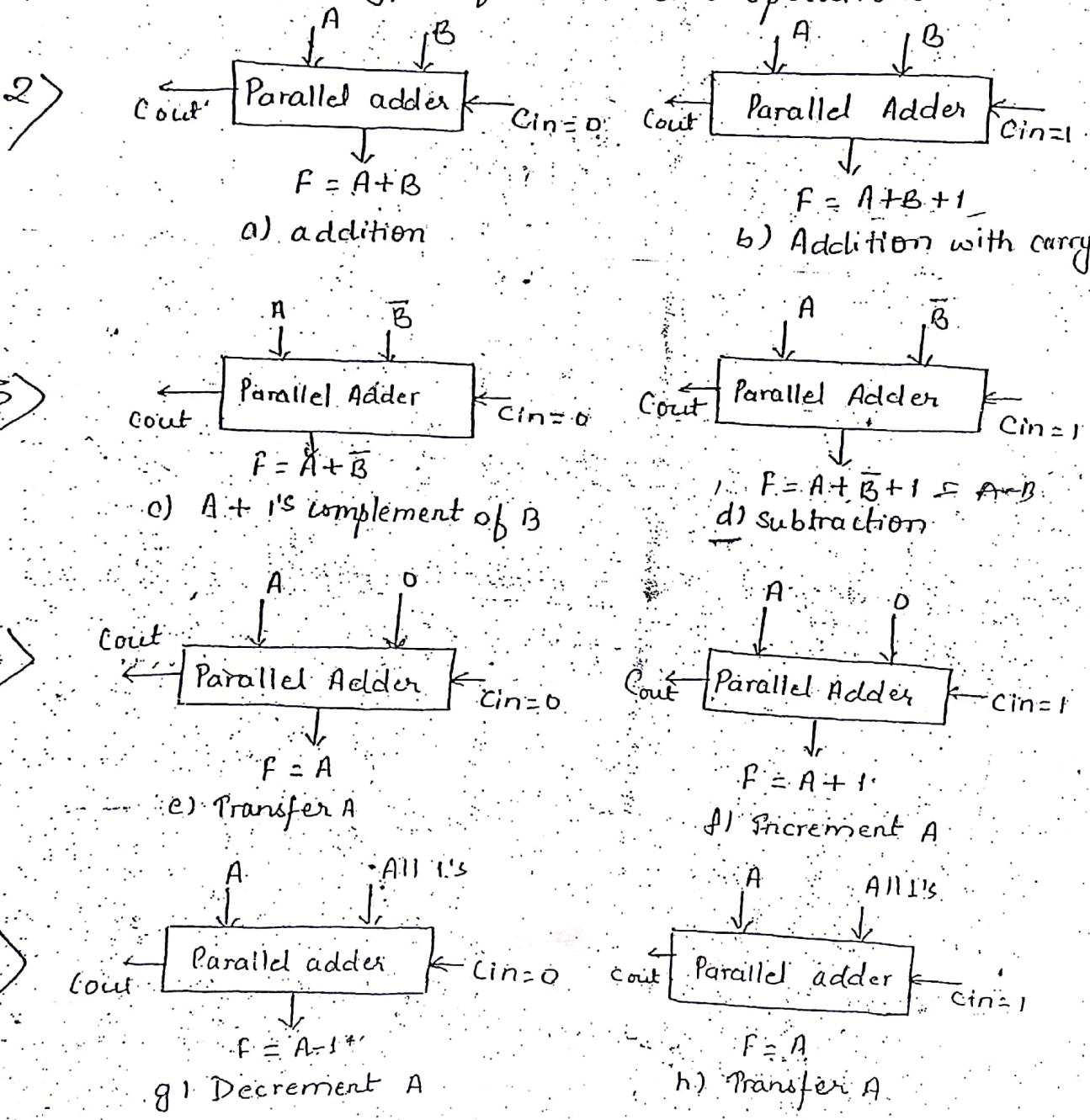


Fig: Operations obtained by controlling one set of i/ps to a parallel adder.

Case (8) Decrement of A

→ Consider a parallel adder with n full-adder circuits

→ When the output carry is 1, represents the number 2^n because in binary consists of a 1 followed by n 0's.

→ Subtracting 1 from 2^n, we obtain 2^n-1, which in binar is a number of n 1's.

→ Adding 2^n-1 to A, we obtain $F = A+2^n-1 = 2^n+A-1$. If the output carry 2^n is removed, we obtain $F = A-1$.

Example

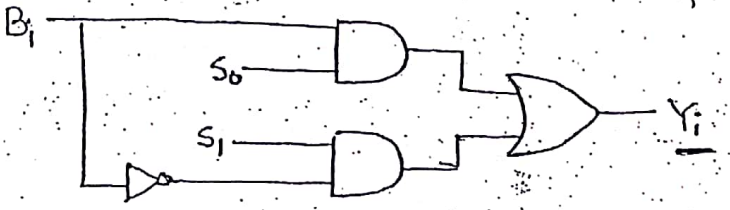
$A = 0000\ 1001 = (9)_{10}$

$2^n = 10000\ 0000 = (256)_{10}$ (2^8)

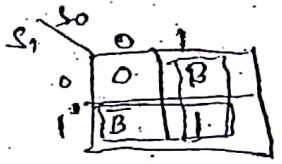
$2^n-1 = 1111\ 1111 = (255)_{10}$

$A+2^n-1 = 1\ 0000\ 1000 = (256+8)_{10}$

Removing the output carry $2^n = 256$, we obtain $8 = 9-1$.



S_1	S_0	Y_i
0	0	B
0	1	B_i
1	0	B_i'
1	1	1



$Y = S_1 \cdot \bar{B} \cdot 1 + S_0 B$

$= S_1 \bar{B} + S_0 B$

Fig: True / complement, one/zero circuit

2010, 2011, 2011, 2009.

Function table for the arithmetic circuit given above

Function select			Y equals	O/p equals	Function
S_1	S_0	Cin			
0	0	0	0	F = A	Transfer A
0	0	1	0	F = A+1	Increment A
0	1	0	B	F = A+B	Add B to A
0	1	1	B	F = A+B+1	Add B to A plus 1
1	0	0	$\bar{B}$	F = A+$\bar{B}$	Add 1's complement of B to A
1	0	1	$\bar{B}$	F = A+$\bar{B}$+1	Add 2's complement of B to A
1	1	0	All 1's	F = A-1	Decrement A
1	1	1	All 1's	F = A	Transfer A

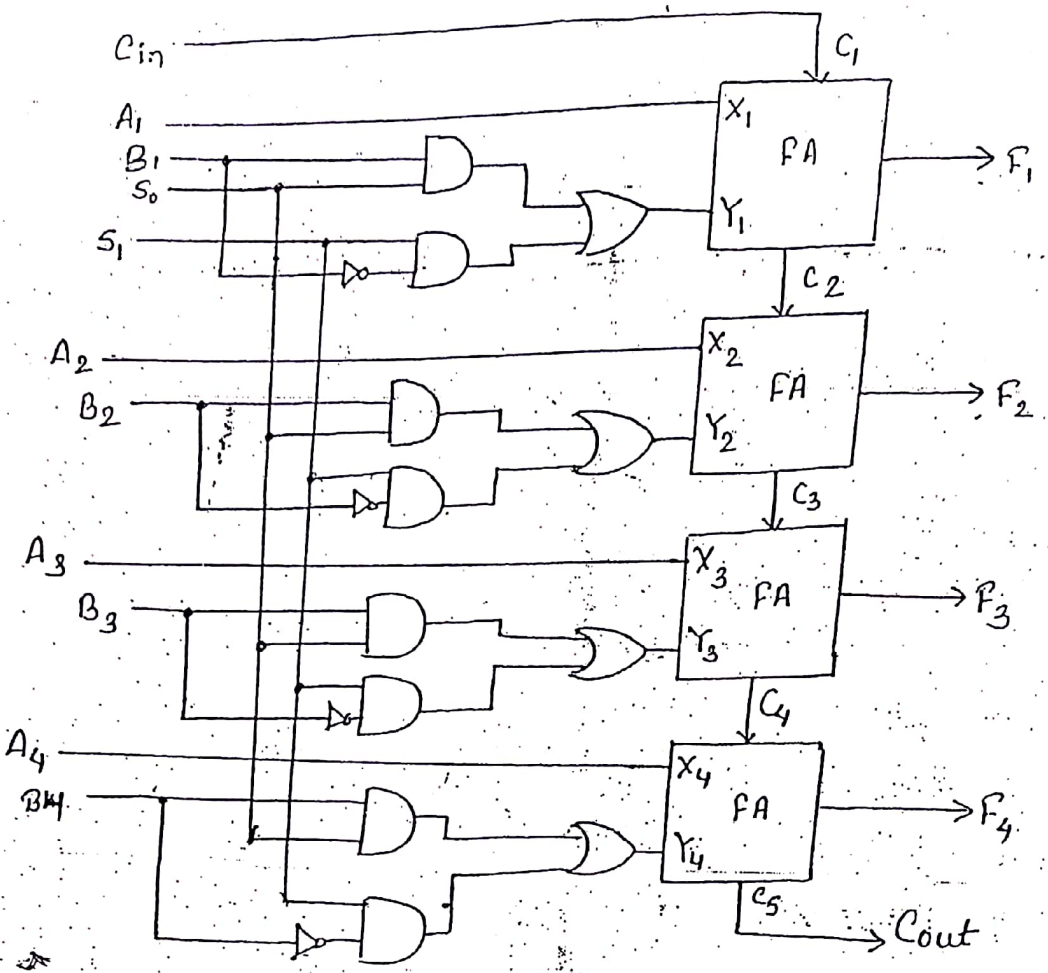


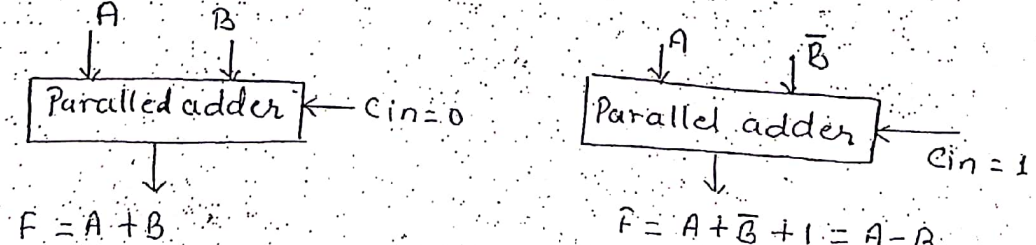
Fig: logic diagram of arithmetic circuit (4-bit)

$$X_i = A_i$$

$$Y_i = B_i S_0 + B_i' S_1 \qquad i = 1, 2, \ldots, n$$

Example

* Design an adder / subtractor circuit with one selection variable S and two inputs A and B. when S = 0 the circuit performs A + B. When S = 1 the circuit performs A − B by taking the 2's complement of B.



a) function specification

Design of logic Circuit

→ The logic microoperations manipulate the bits of the operands separately and treat each bit as a binary variable

→Consider different four logic operations AND, OR, NOT & XOR, for these operations we need two selection variables.

→Let us design a logic circuit

S_1	S_0	output	Operation
0	0	$F_i = A_i + B_i$	OR
0	1	$F_i = A_i \oplus B_i$	XOR
1	0	$F_i = A_i B_i$	AND
1	1	$F_i = A_i'$	NOT

Fig: Function table

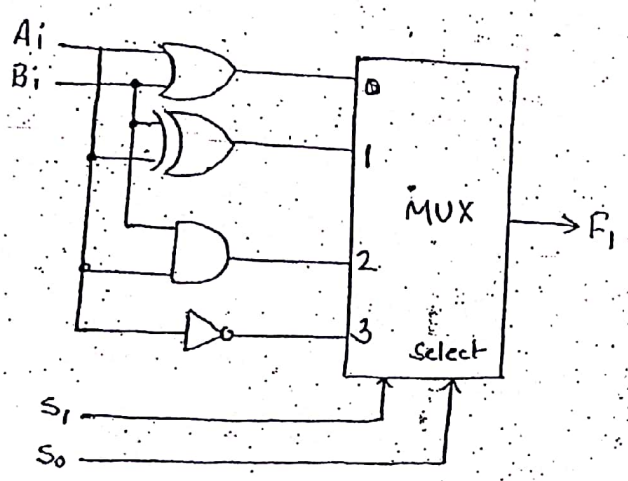


Fig: logic diagram

→ The simplest and most straightforward way to design a logic circuit is shown in above figure.

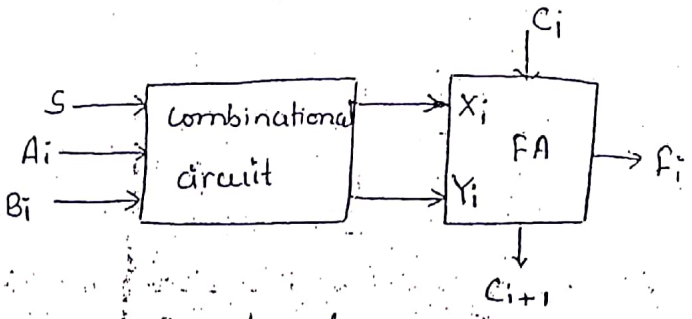
→ The diagram shows one typical stage designated by subscript i. The ckt must be repeated n times for an n-bit logic circuit.

→ The four gates generated the four logic operations OR, XOR, AND, and NOT.

→ The two selection variables in the multiplexer select one of the gates for the output.

→ The function table lists the output logic generated as a function of the two selection variables.

S	Xi	Yi	Cin
0	Ai	Bi	0
1	Ai	B_i'	1



b) Specifying combinational circuit:

S	Ai	Bi	Xi	Yi
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	0
1	1	0	1	1
1	1	1	1	0

for Xi

S \ AiBi	00	01	11	10
0	0	0	1	1
1	0	0	1	1

$X_i = A_i$

for Yi

S \ AiBi	00	01	11	10
0	0	1	1	0
1	1	0	0	1

$Y_i = S'B_i + SB_i'$
$= S \oplus B_i$

$C_{in} = S$

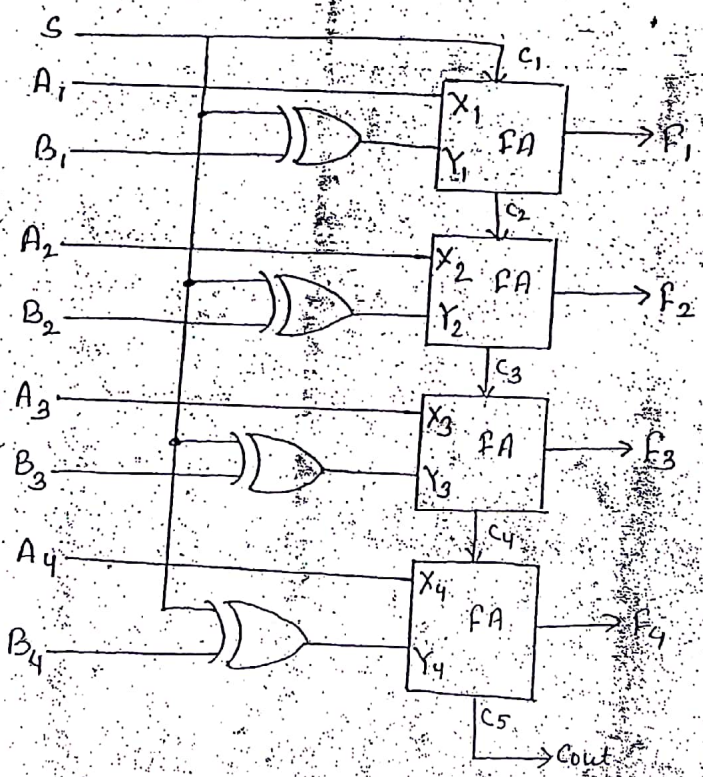


Fig: 4-bit adder/ subtractor circuit

→ The logic circuit can be combined with the arithmetic circuit to produce one arithmetic logic unit.

→ selection variables S_1 & S_0 can be made common to both and S_2 is used to differentiate betn the two.

when $S_2 = 0$, the arithmetic o/p is selected, but when $S_2 = 1$ the logic o/p is selected.

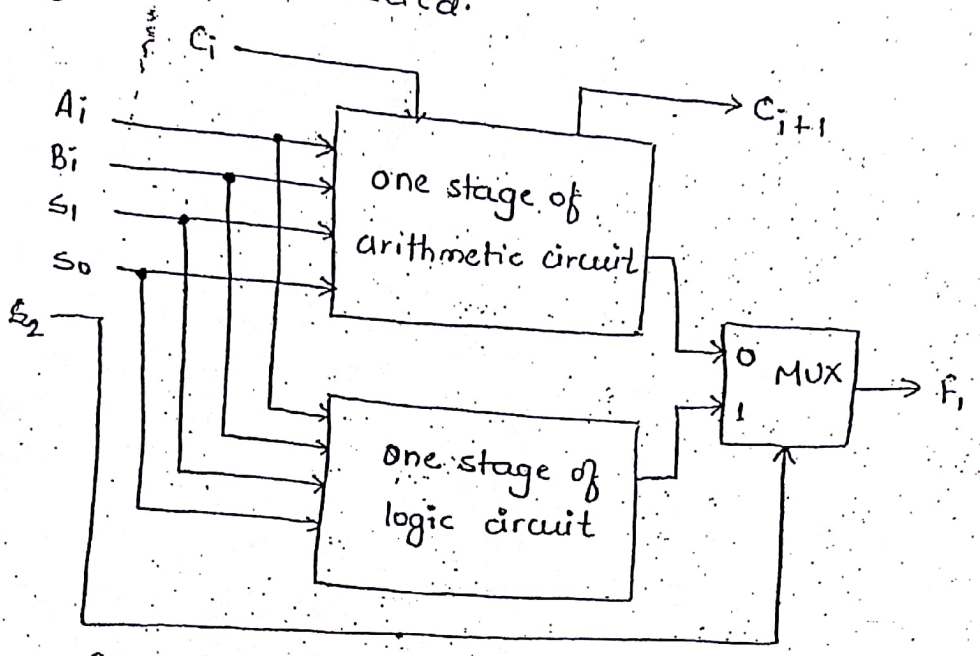


Fig: Combining ALU

Note

* The more efficient way to design ALU is if we investigate the posibility of generating logic operations in an already available arithmetic circuit.

→ The full adder output is

$$F_i = X_i \oplus Y_i \oplus C_i$$

when $S_2 = 1$ the logic operation is selected, $C_i = 0$

$$F_i = X_i \oplus Y_i \quad [\because x \oplus 0 = x$$

Thus, with input carry to each stage equal to 0, the full adder circuit generates XOR operation.

S_2	S_1	S_0	X_i	Y_i	C_i	$F_i = X_i \oplus Y_i$	Operation	Required Operation
1	0	0	A_i	0	0	$F_i = A_i$	Transfer A	OR
1	0	1	A_i	B_i	0	$F_i = A_i \oplus B_i$	XOR	XOR
1	1	0	A_i	B_i'	0	$F_i = A_i \odot B_i$	Equivalence	AND
1	1	1	A_i	1	0	$F_i = A_i'$	NOT	NOT

Output F_i is then equal to $X_i \oplus Y_i$ and produces the XO and NOT operations. when $S_2S_1S_0 = 100$, each A_i is OR with B_i to provide OR operation. when $S_2S_1S_0 = 110$, each A_i is ORed with B_i' to provide the AND operati

Selection				Output	Function
S_2	S_1	S_0	Cin		
0	0	0	0	F = A	Transfer A
0	0	0	1	F = A+1	Increment A
0	0	1	0	F = A+B	Addition
0	0	1	1	F = A+B+1	Add with carry
0	1	0	0	$F = A+\bar{B}$ (A-B-1)	Subtract with borrow
0	1	0	1	F = A-B (A+$\bar{B}$+1)	Subtraction
0	1	1	0	F = A-1	Decrement A
0	1	1	1	F = A	Transfer A
1	0	0	X	F = A ∨ B	OR
1	0	1	X	F = A ⊕ B	XOR
1	1	0	X	F = A ∧ B	AND
1	1	1	X	F = $\bar{A}$	Complement A

Function table for the ALU

Status Register

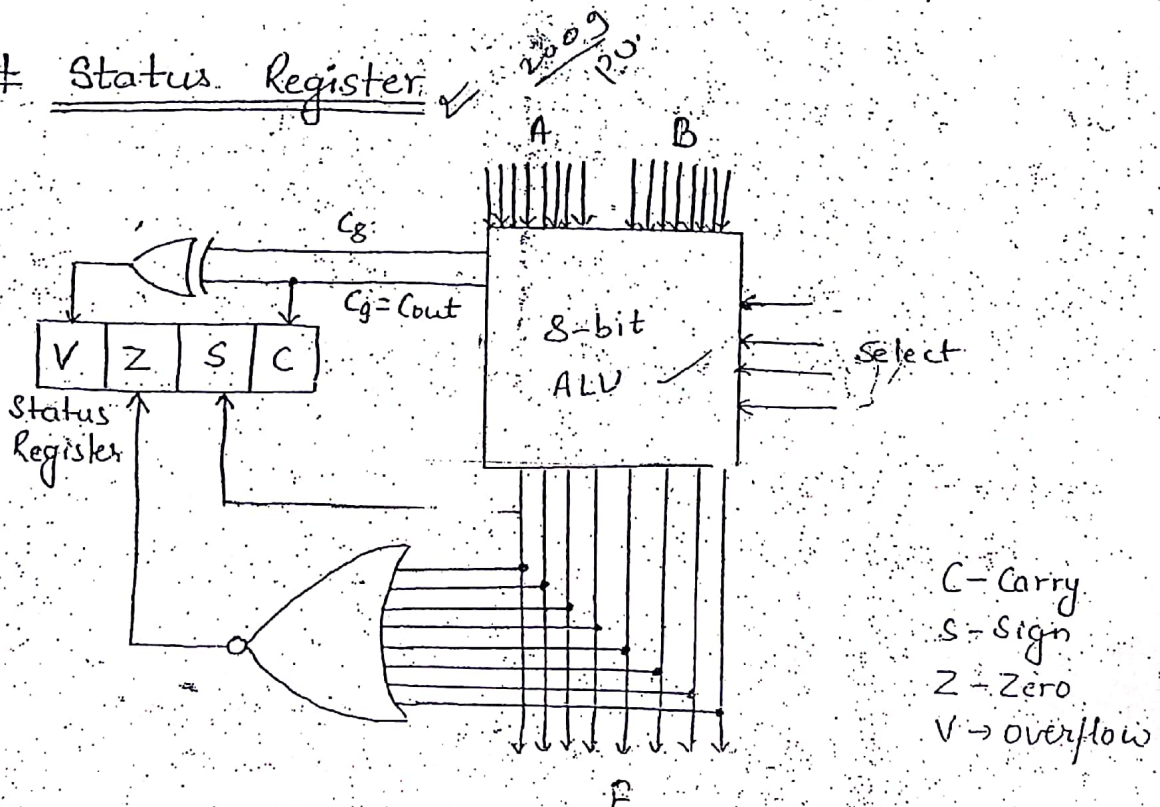


Fig: Setting bits in a status register

→ status register is an additional register with ALU which gives the status of the final output of ALU.
→ status bits of status register are used for further and bits or flag bits.
→ status-bit conditions are sometime called condition-co
→ Figure shows the block diagram of an 8-bit ALU with 4-bit status register.
→ The four status bits are symbolized by C, S, Z & V. The bits are set or cleared as a result of an operation performe in the ALU:

1. Bit C is set if the output carry of the ALU is 1. It is cleared if the output carry is 0.
2. Bit S is set if the highest-order bit of the result in the output of the ALU (the sign bit) is 1. It is cleared if the highest-order bit is 0.
3. Bit Z is set if the output of the ALU contains all 0's and cleased otherwise. Z=1 if the result is zero, and Z=0 if the result is nonzero.
4. Bit V is set if the exclusive-OR of carries C_8 & C_9 is 1, and cleared otherwise. This is the condition for overflow when the numbers are in sign-2's-complement representation. For the 8-bit ALU, V is set if the result is greater than 127 or less than −128.

Application

A = 101x1100 when x is the bit to be checked. The AND operation of A with
B = 00010000 produces result
i) if x=0 then Z status bit is set
ii) if x=1, then Z bit is cleared since the result is not zero.

2.01 Rail

Design of Shifter :-

→ The shift unit attached to a processor transfers h output of the ALU onto the output bus. The shifter may transfer the information directly wi a shift or it may shift the information to the r or left.

→ Provision is sometimes made for no transfer fro the ALU to the output bus. The shifter provides the shift operations commonly not available in A

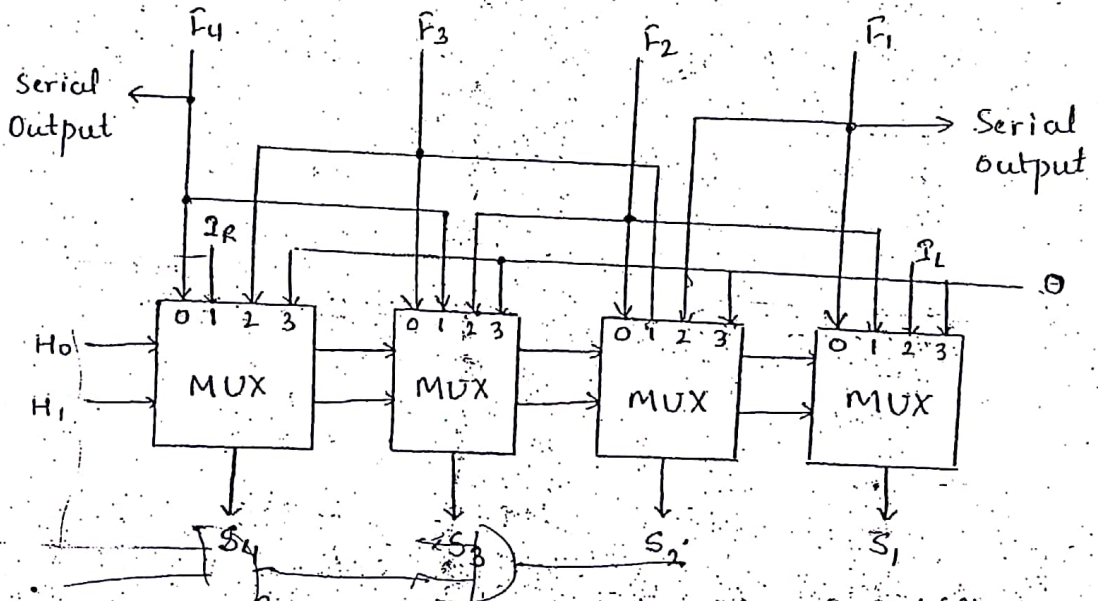


Fig: 4-bit combinational-logic shifter

H_1	H_0	operation	Function
0	0	$S \leftarrow F$	Transfer F to S (no shift)
0	1	$S \leftarrow shr\ F$	shift-right F into S
1	0	$S \leftarrow shl\ F$	shift-left F into S
1	1	$S \leftarrow 0$	Transfer 0's into S

Fig: Function table for shifter

Processor Unit

The selection variables in a processor unit control the micro-operations executed within the processor during any given clock pulse. The selection variable control the buses, the ALU, the shifter and the destination register.

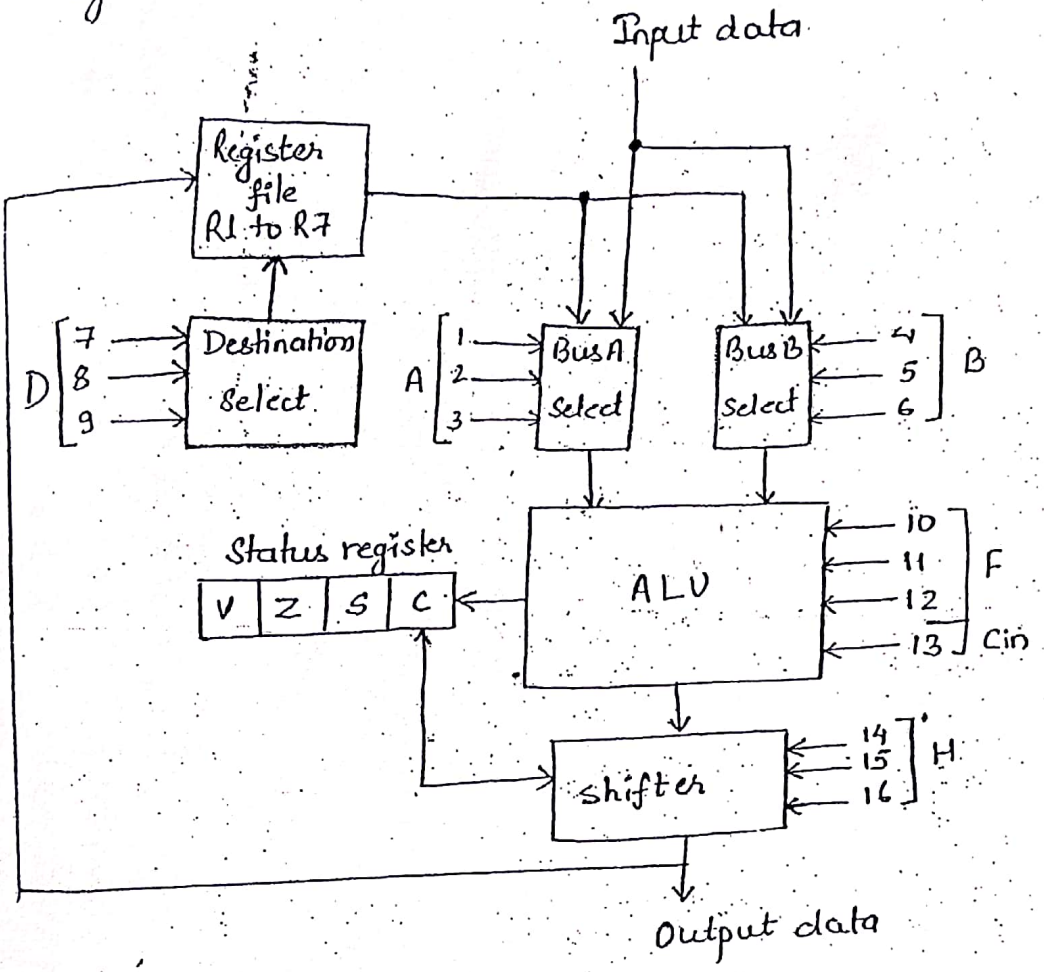


Fig: Block diagram

1 2 3	4 5 6	7 8 9	10 11 12	13	14 15 16
A	B	D	F	Cin	H

Control word

→ Processor unit consists of seven registers R_1 through R_7 and status register. The output of the seven registers go through two multiplexer to select the i/p to the ALU.

→ Input data from an external source are also selected by the same multiplexers.

→ The o/p of ALU goes through a shifter and then to a set of external o/p terminal. The o/p from the shifter can be transfered to any one of the registers or to an external destination.

→ There are 16 selection variables in the unit and their function is

specified by a control word. The 16 bit control word, when applied to the selection variables in the processor, specifies a given micro operation.

→ All fields, except Cin, have a code of three bits. The three bits of A select a source register for the input to left side of the ALU. The B field is the same, but it selects the source information for the right input of the ALU. D field selects the destination register. The field, together with the bin in Cin select a function for the ALU.

→ The H field select the type of shift in the shift unit.

→ A control word of 16 bits is needed to specify a micro-operation for the processor unit. The most efficient way to generate control words with so many bits is to store them in a memory unit which functions as a control memory where all control words are stored.

→ The sequence of control words is then read from the control memory, one word at a time, to initiate the desired sequence of micro-operations. This type of control operation is called microprogramming.

Design of Accumulator: (2012 spring)

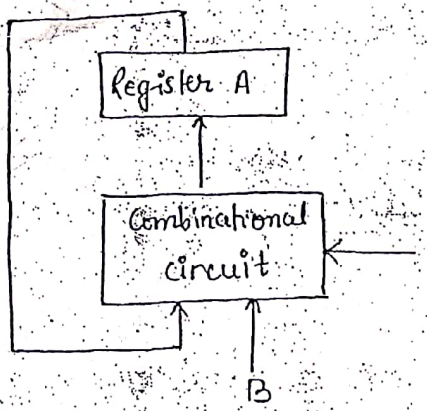


Fig: Block diagram of accumulator

→ Some processor units distinguish one register from all others and call it an accumulator register.

→ The accumulator register is essentially a bidirectional shift register with parallel load which is connected to an ALU