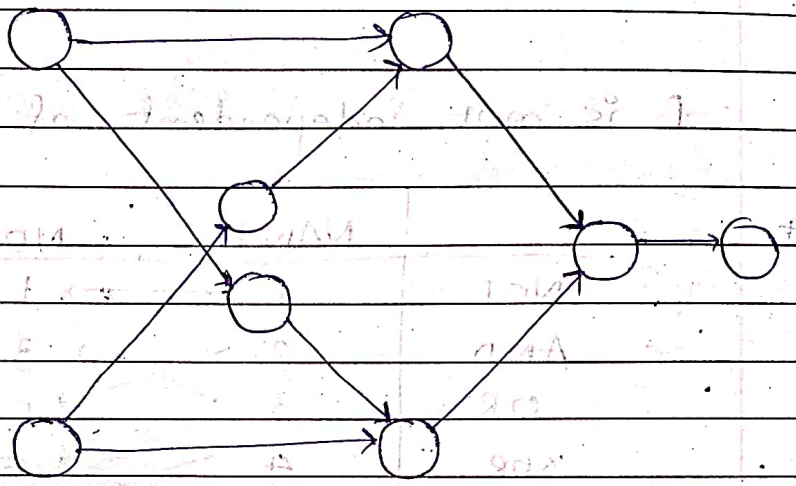
NF

DATE:
PAGE:

Design Methodology

→ Abstraction : Hiding the data
- Gate level (least form of abstraction)
- Register level
- Processor level

→ Sytem Representation:

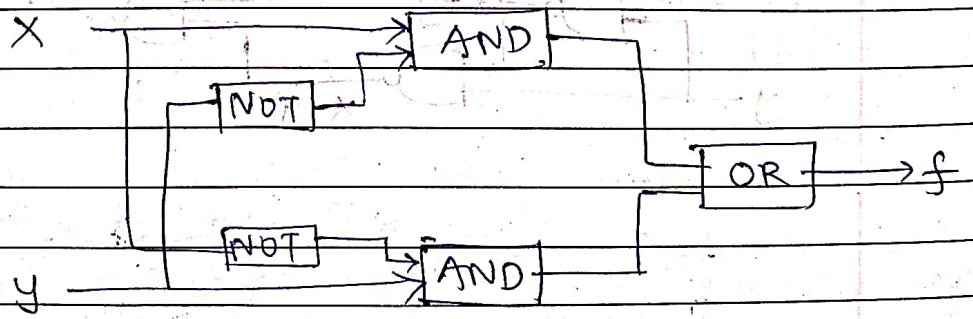


$V = \{V_1, V_2, \ldots\}$

$E = \{e_1, e_2, \ldots\}$

$e_1 = \{V_1, V_2\}$

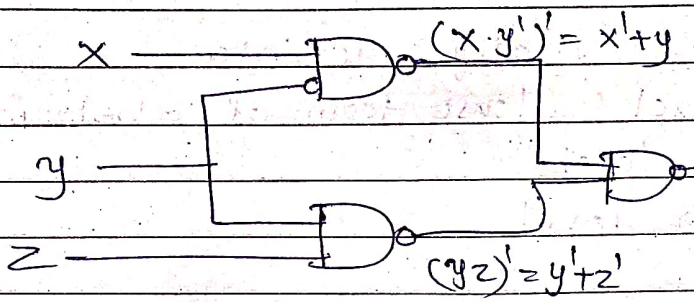
→ Schemetic diagram:



Structure vs Behaviour

↙ mathematical eqn ↘ truth table

two ways of representation



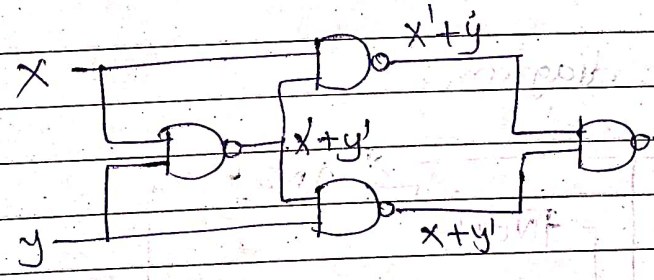
$f = [(x'+y)\cdot(y'+z')]'$

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

$x\cdot y' + y\cdot z$

f is not independent of any variable.

	NAND	NOR
NOT	1	1
AND	2	3
OR	3	2
XOR	4	5
XNOR	5	4
NAND	1	4
NOR	4	1



$F = xy' + x'y$

DATE:
PAGE:

→ Hardware Description language (HDL)
VHDL, VERILOG

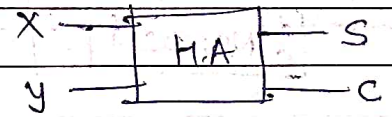
Advantage :- They are very precise and technology indep^{n}
- Used to draw prototype (Simulation)

Disad. - They are very complex.
- They are not insightful (not able to know internal thing).

① Entity ② Architecture

Structure Behaviour

ex.



→ Behavioural VHDL Description :

Entity is half-adder
port (x, y : in bit; Sum, carry : out bit);
end half adder

architecture behaviour of half-adder is
begin
sum ⇐ x XOR y
carry ⇐ x AND y after 5ns
end behaviour

Assumption : XOR and AND are already in VHDL

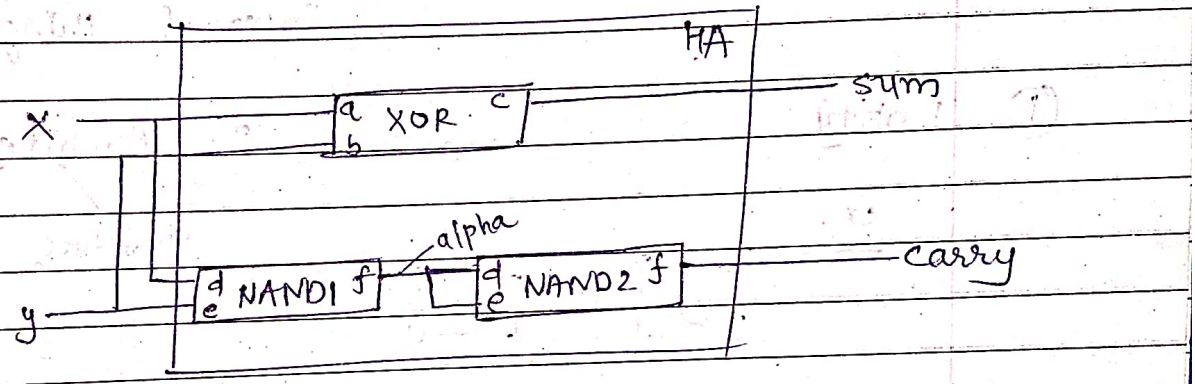
IMP (If que. is of 5 or 6 marks draw circuit, find eq^{n} using kmap and write above)

DATE:
PAGE:

Structural VHDL description :

⇐ : signal assignment

[In half adder we can write,
half adder is
if 'c' XY = 11' then carry ⇐ 1 else carry ⇐ 0]



Define → Architecture structure of ~~entity~~ half adder is
component XOR-circuit port (a,b : in bit, c : out bit)
end component

component NAND-gate port (d,e : in bit, f : out bit)
end component

signal alpha : bit
end half adder

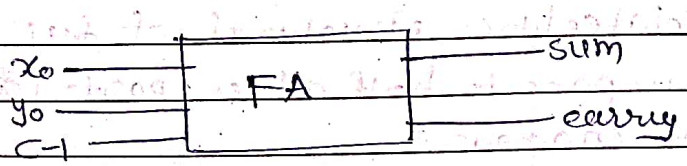
Network list (Netlist)

begin
XOR : XOR-circuit portmap (a ⇒ x, b ⇒ y, c ⇒ sum)
NAND1 : NAND-gate portmap (d ⇒ x, e ⇒ y, f ⇒ alpha)
NAND2 : NAND-gate portmap (d ⇒ alpha, e ⇒ alpha, f ⇒ carry)
end
end structure

DATE:
PAGE:

→ VHDL Description of Full adder :

1. Behavioural VHDL Desc. :



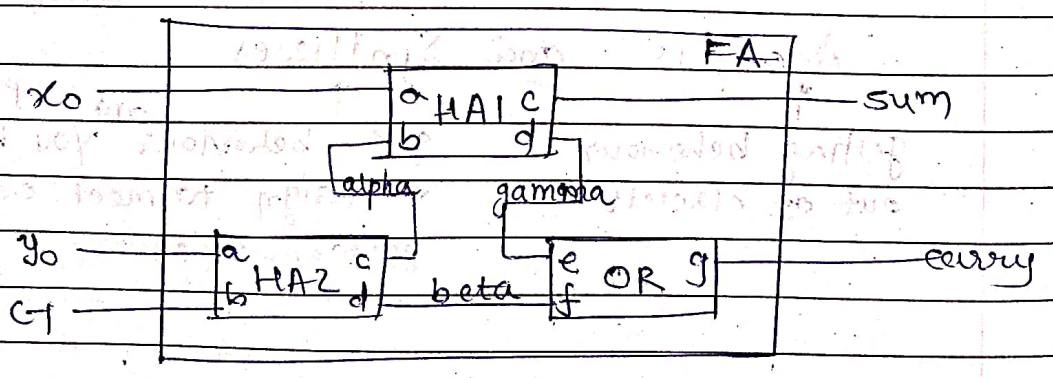
$$sum = x_0 \oplus y_0 \oplus C_{-1}$$

$$Carry = x_0y_0 + x_0C_{-1} + y_0C_{-1}$$

```
entity is full-adder
port (x0, y0, C-1 : in bit ; sum, carry : out bit)
end full adder

Architecture behaviour of full adder is
begin
   sum <= x0 XOR y0 XOR C-1 ;
   carry <= (x0 AND y0) OR (x0 AND C-1) OR
                           (y0 AND C-1)
end behaviour
```

2. Structural VHDL Desc :



DATE:
PAGE:

```
entity is full adder
port (xo, yo, C-1 : in bit ; sum, carry : out bit);
end full adder

Architecture structural of full adder is
  component half adder port (a, b : in bit, c, d : out bit)
end component
  component OR gate port (e, f : in bit, g : out bit);
end component

  signal α, β, γ : bit)
end full adder
begin
  HA1 : half adder portmap (a ⇒ xo, b ⇒ alpha,
                            c ⇒ sum, d ⇒ gamma);
  HA2 : half adder portmap (a ⇒ yo , b ⇒ C-1 ,
                            c ⇒ alpha, d ⇒ beta);
  OR : OR gate portmap (e ⇒ gamma , f ⇒ beta,
                        g ⇒ carry)

end
end structure
```

→ Design Process

Analysis and Syntheses

↑ getting behaviours out of circuit

↑ given behaviour and component you have to design to meet cost & performance

DATE:
PAGE:

Goals: Design should be correct
Performance maximize
cost minimize
power consumption
reliability
compatibility

→ CAD ⟹ synthesizes a design
prototype of the system either
given out actual hardware implementation

→ Design levels:

	level	Component	IC	Info. unit	Time unit
(1)	Gate (logic)	logic gates ff	SSI	bits	10^{-12} to 10^{-9} s
(2)	Register (Reg. transfer-RTL)	Reg, counter comb. circuit seq. circuit	MSI	word	10^{-9} to 10^{-6} s
(3)	Processor (architecture behaviour system)	CPU, Memories, I/O devices	VLSI	block of words	10^{-3} to 10^{3} s

DATE:
PAGE:

(1) Gate level / hardware design

Full adder

$$S_0 = x_0 y_0 C_{-1} + x_0' y_0 C_{-1}' + x_0' y_0' C_{-1} + x_0 y_0' C_{-1}'$$

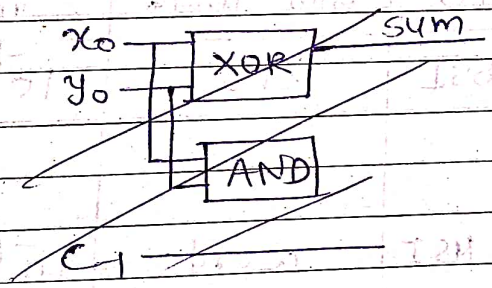
$$C_0 = x_0 y_0 + x_0 C_{-1} + y_0 C_{-1}$$

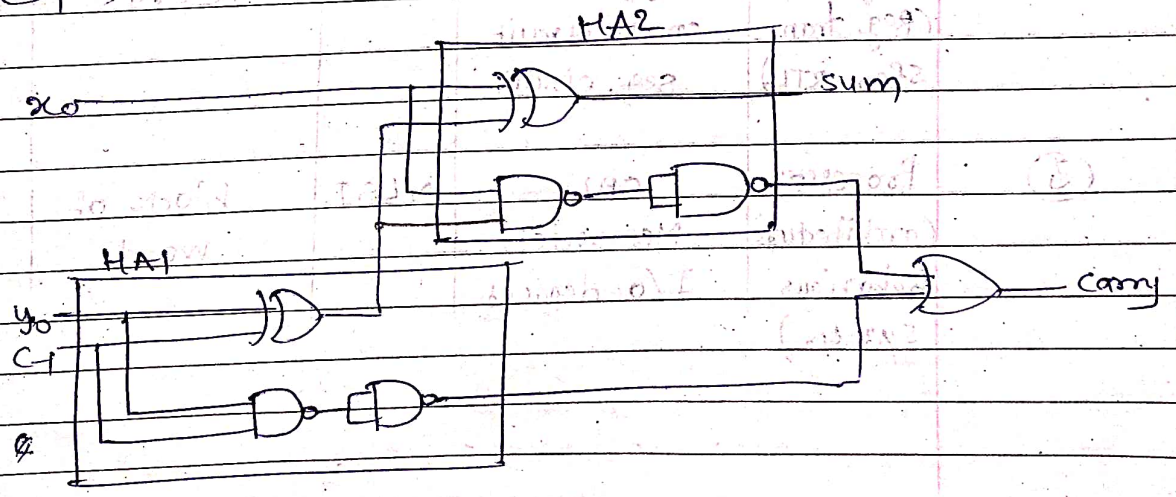
$$= (x_0 + y_0) \cdot (x_0 + C_{-1}) \cdot (y_0 + C_{-1})$$

Half adder

$$S_0 = x_0 y_0'' + x_0' y_0$$

$$C_0 = x_0 y_0$$



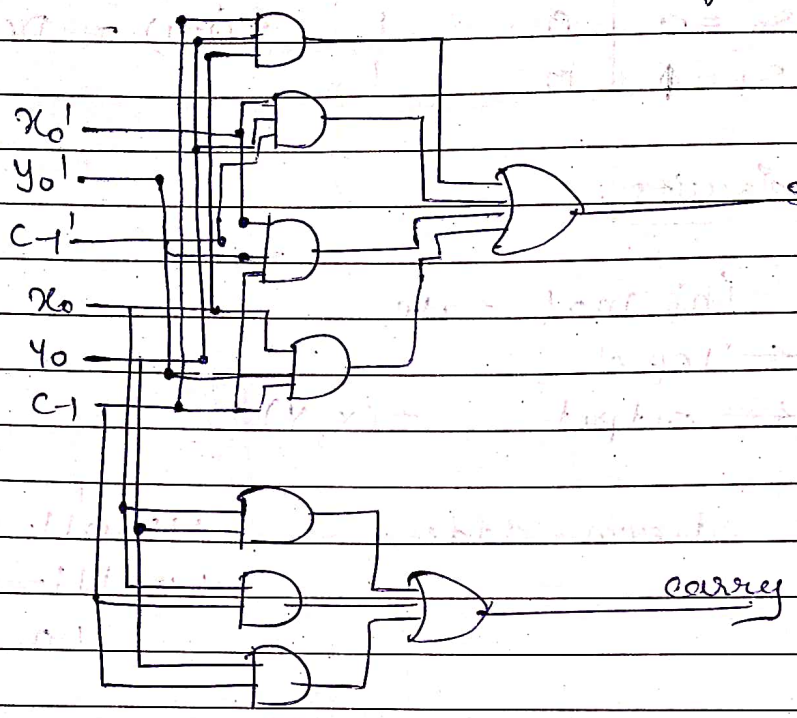


4 gate level

DATE:
PAGE:

Using AND-OR

& gate level

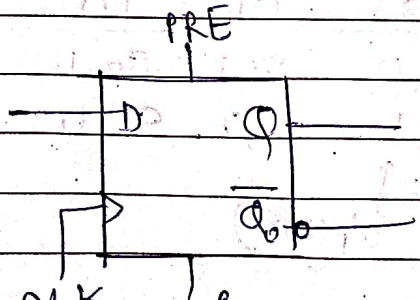


Fan-in ⇒ How many input lines is given
Fan-out ⇒ How many output lines are there

→ convert into NAND-NOR

1. take bar of each term
2. convert AND → OR or OR → AND
3. take whole bar

→ flipflops



level trigger ⇒ Problem
transient noise
it will change the o/p

CLR, PRE → asynchronous signal

DATE:
PAGE:

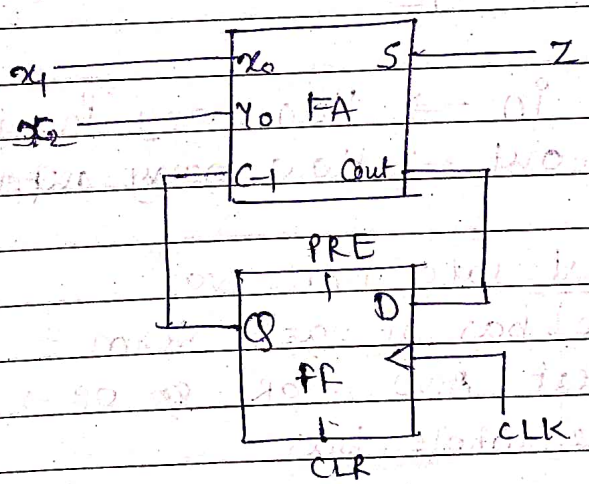
		Di	
		0	1 ← input
S(i)	$S_0 = 0$	0	1 S(i+1) = D(i)
PS	$S_1 = \phi$	0	1

→ Seq. circuits:

Y ← Internal state
x ← input
z ← output , z(x, Y)

(1) 2 bit stream adder

x_1: 1 1 0 1 0 1 1 →
x_2: 1 0 0 1 1 →
1 0

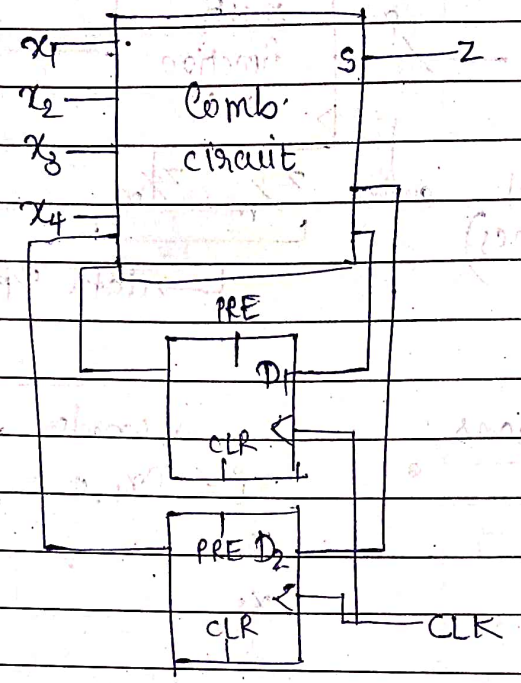


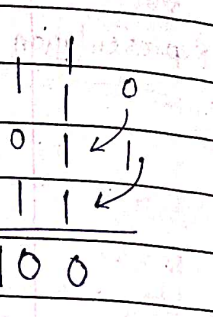
truth table:

Present state		x_1x_2 = 00	01	10	11
	$S_0 = 0$	$S_0, 0$	$S_0, 1$	$S_0, 1$	$S_1, 0$
	$S_1 = 1$ (carry)	$S_0, 1$	$S_1, 0$	$S_1, 0$	$S_1, 1$

↑ O/P (sum)

(2) 4 bit stream adder



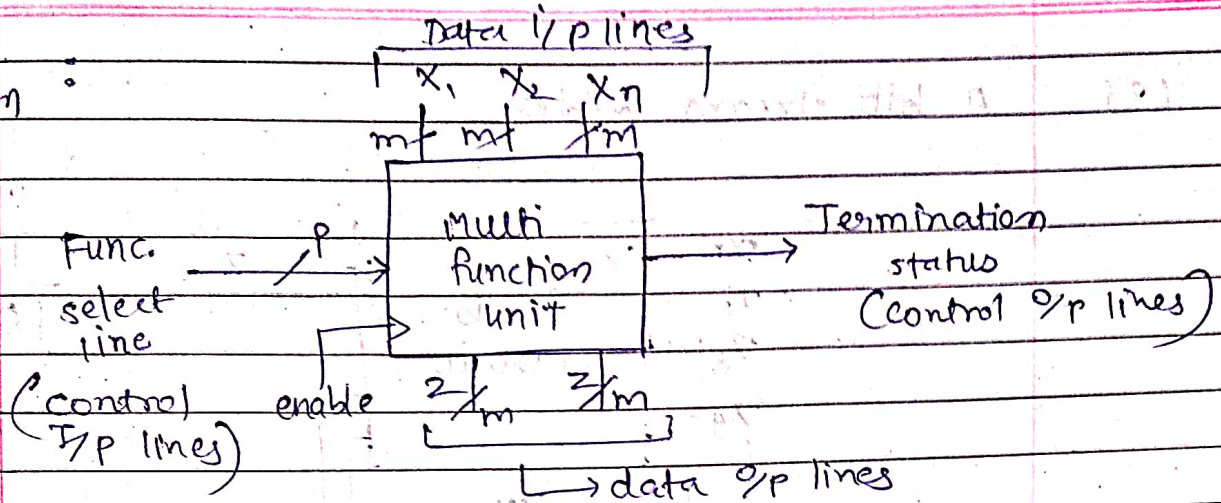


	0 1 2 3 4 $x_1 x_2 x_3 x_4$ 5 12 15
$S_0 = 00$	
$S_1 = 01$	
$S_2 = 10$	$S_2, 0$ $S_2, 1$
$S_3 = 11$	

(2) Register level → Inf. unit = words

components : word gates
Mux / Demux
Decoder / encoders
adders
ALU, PLD
shift reg, counter

Block representation :



→ Operations

n words m bits

$X_i = (x_{i,0}, x_{i,1}, x_{i,2}, \ldots x_{i,m-1})$

x_1

x_2

$x_n = (\quad x_{n,m-1})$

$z = (\quad)$

$z(X_1, X_2, \ldots, X_n) =$

$[z(x_{1,0}, x_{2,0}, \ldots x_{n,0}),$ } bit by bit operation

$z(x_{1,1}, \ldots x_{n,1}),$

$z(x_{1,m-1}, \ldots x_{n,m-1})]$

— $n = 2$ variables $2^{2^n} = 16$

— n bit variables and m bit $= 2^{2^{nm}}$

Coord gate $X = x_1 x_2 \ldots x_m$

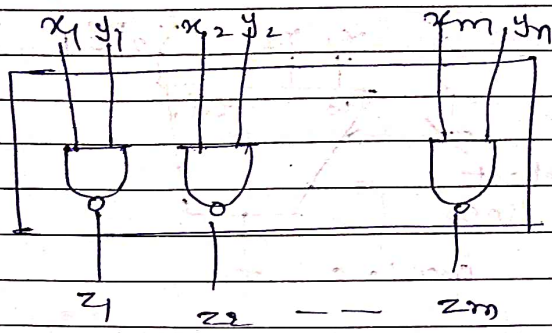
$Y = y_1, y_2, \ldots y_m$

$f = \overline{xy}$

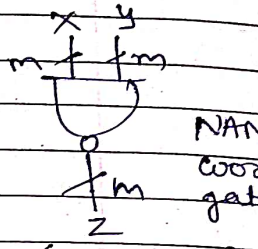
$z(\overline{xy}) = z_1(\overline{x_1 y_1}), z_2(\overline{x_2 y_2}), z_3(\overline{x_3 y_3}) \ldots z_m(\overline{x_m y_m})$

$z = (z_1, z_2, \ldots, z_m) = (\overline{x_1 y_1}, \overline{x_2 y_2}, \ldots, \overline{x_m y_m})$

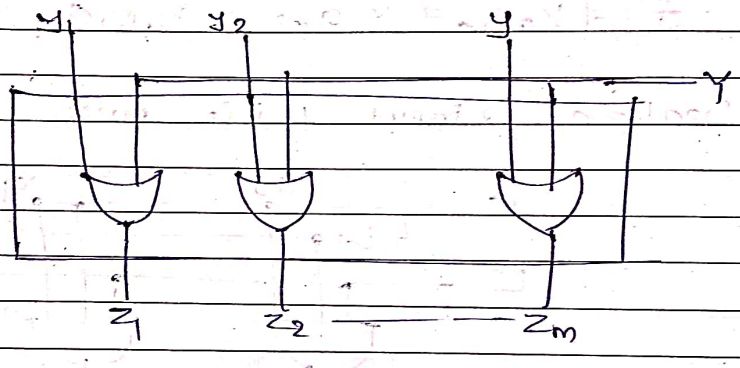
Logic diagram

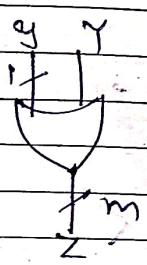


Block diagram:



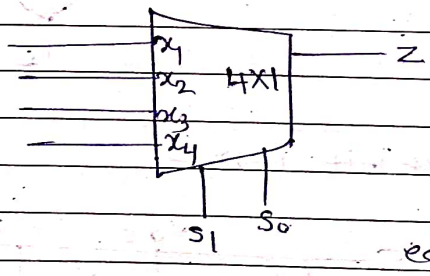
$f = y + Y$ (word) (1 bit)





→ Multiplexer

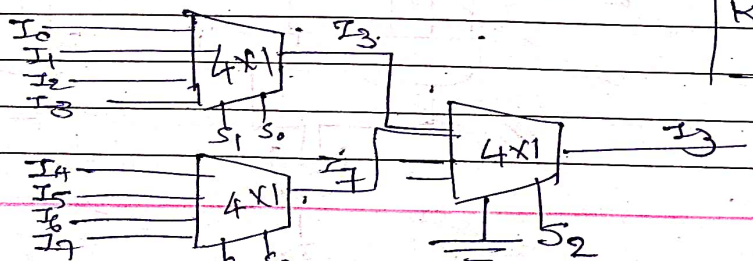
In OS
- data selecting
- routing



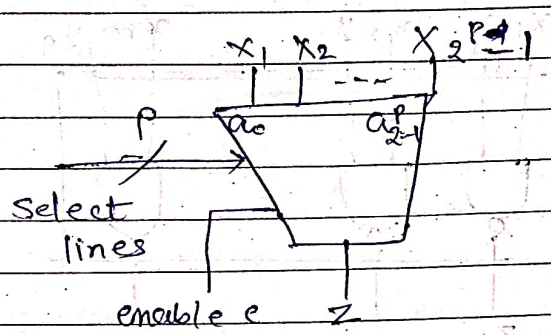
max. no. of data source = k and each IO data line carries m bits

k input (k way) m bit MUX

8X1 using only 4X1 MUX

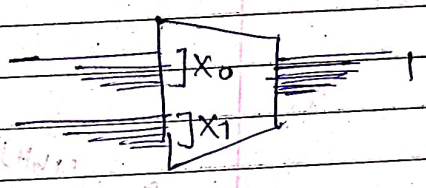


DATE:
PAGE:



k way m bit registers

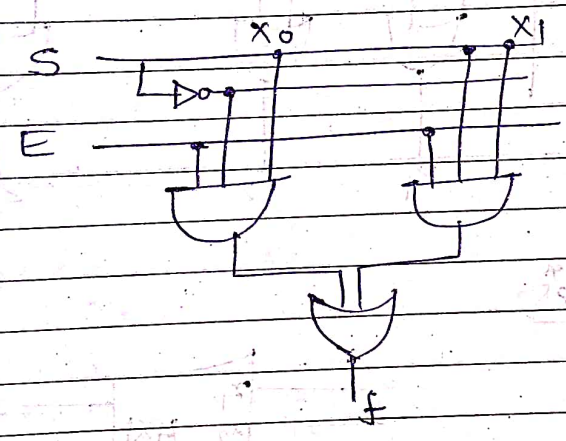
2 i/p 4 bit MUX



$$f = (\bar{S}x_0 + Sx_1)E$$

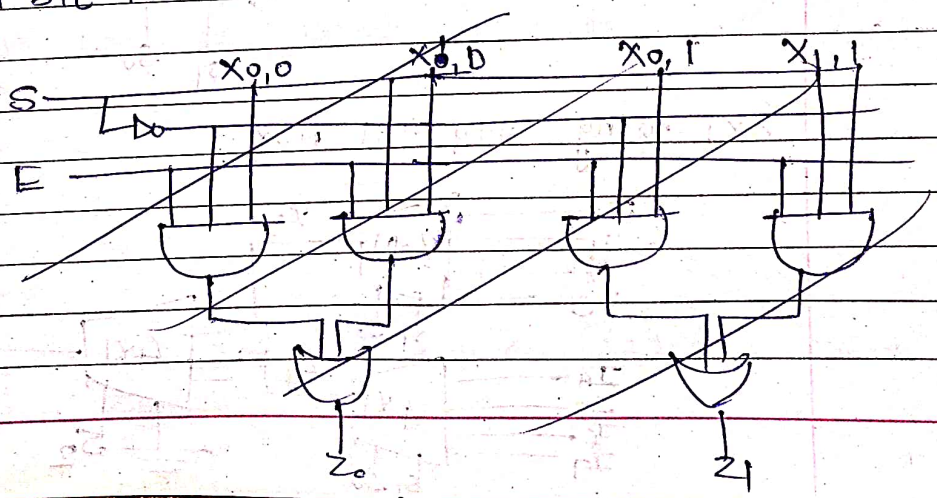
$$f = (x_0a_0 + x_1a_1 + \dots x_{2^p-1}a_{2^p-1})E$$

eg Realize 2 input 1 bit MUX



$$f = (\bar{S}x_0 + Sx_1)E$$

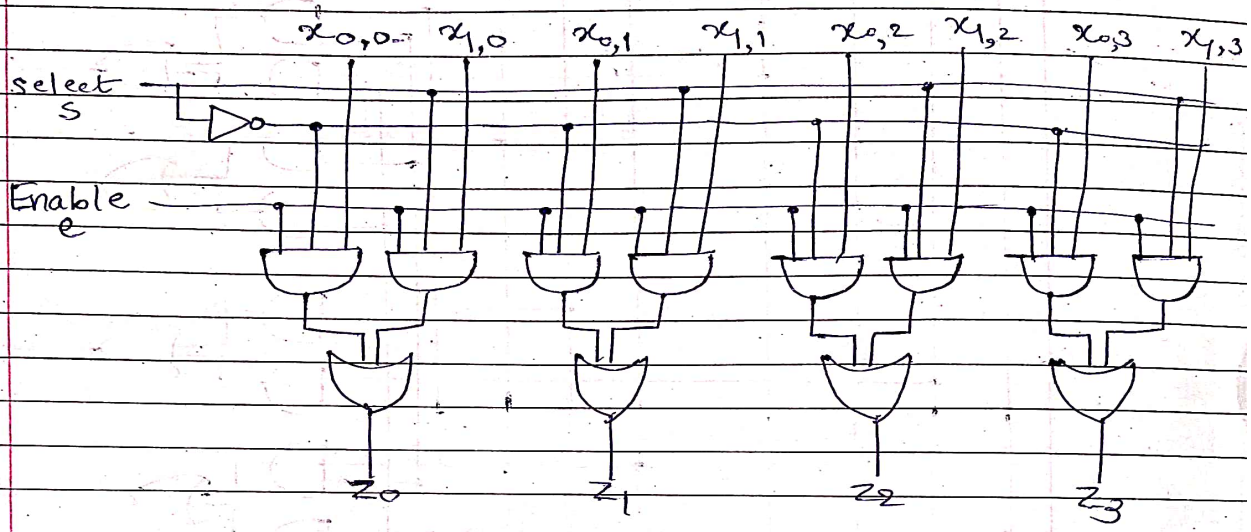
~~2 i/p 4 bit MUX~~



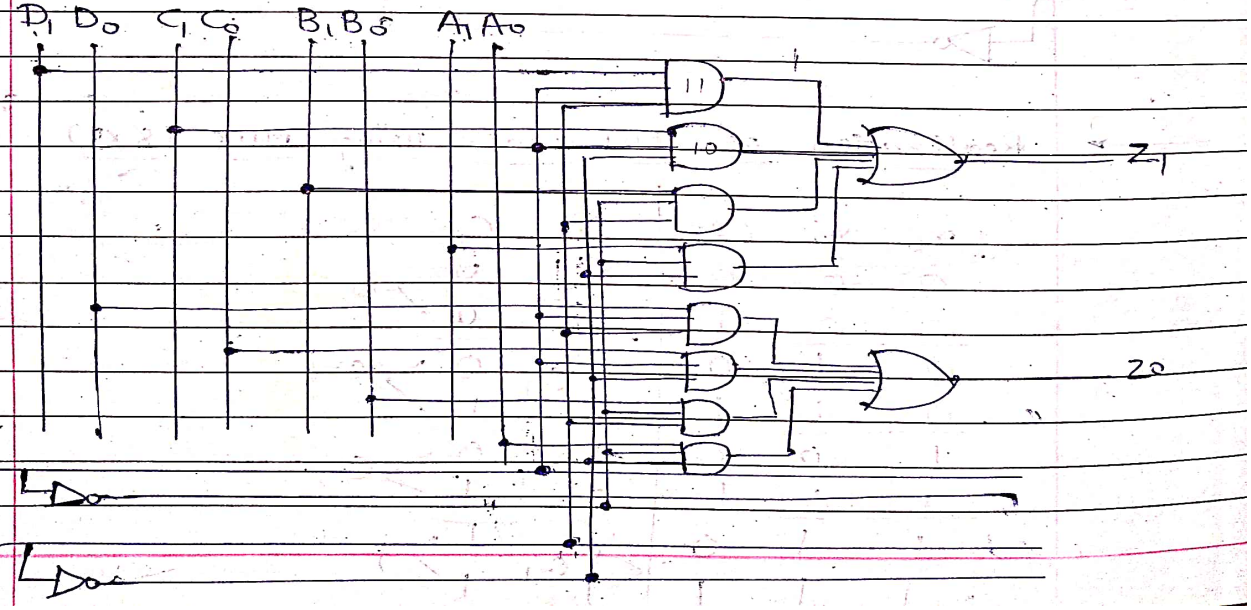
DATE:
PAGE:

~~4 bit 2 bit MUX~~

* 2 Input 4 bit MUX

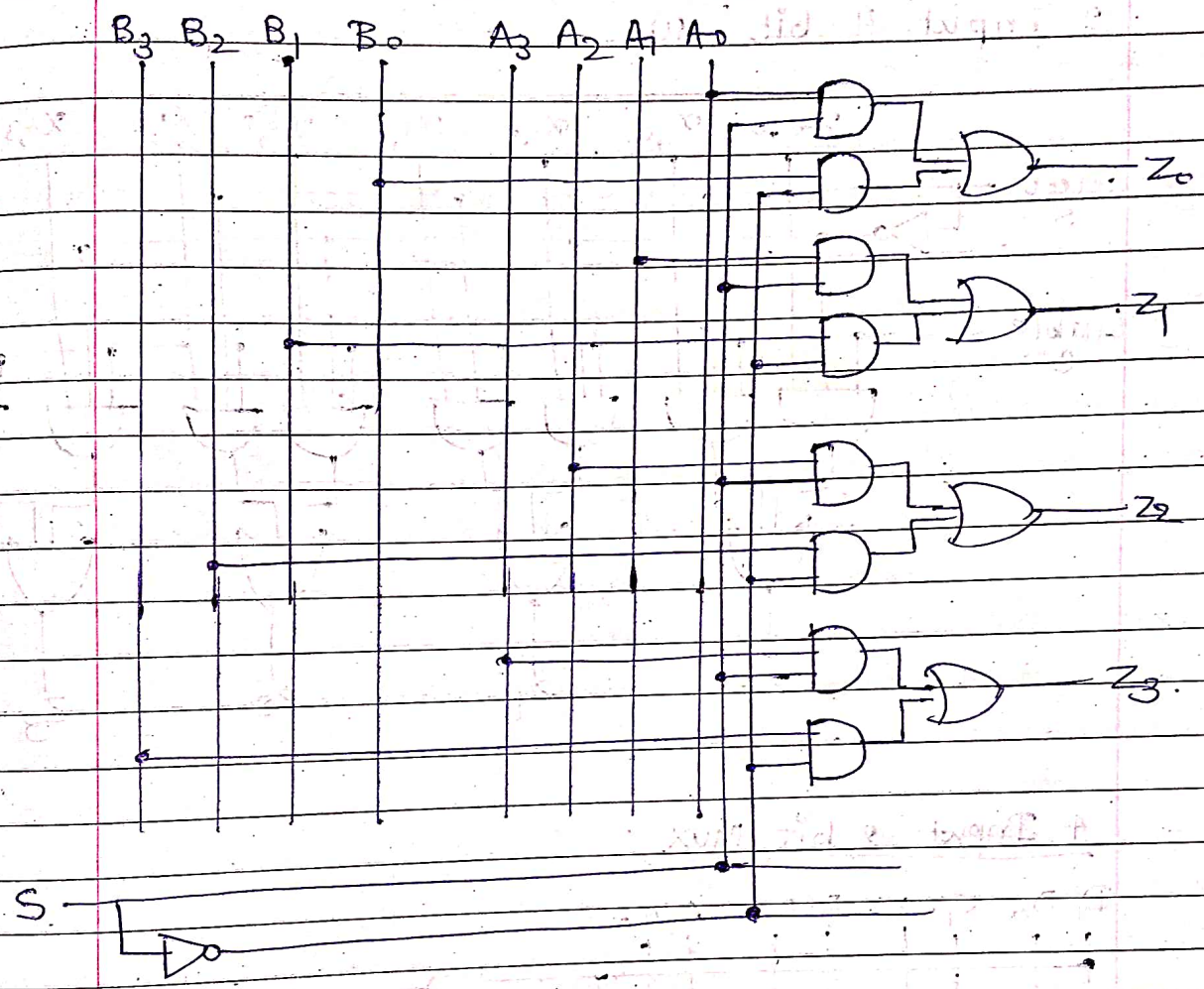


* 4 Input 2 bit MUX



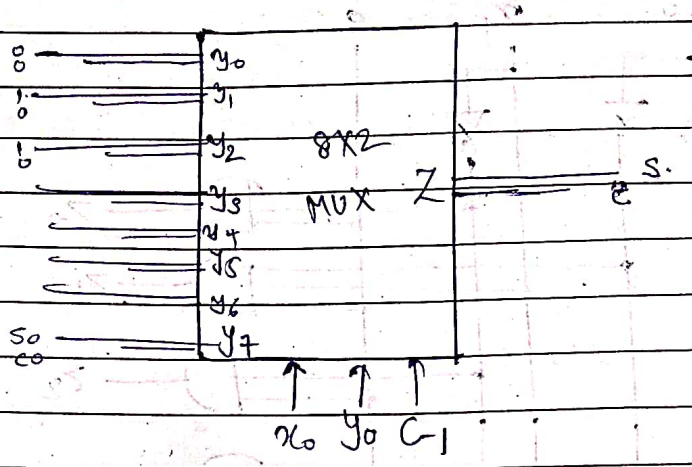
DATE:
PAGE:

Another Representation of 2 Input 4 bit MUX

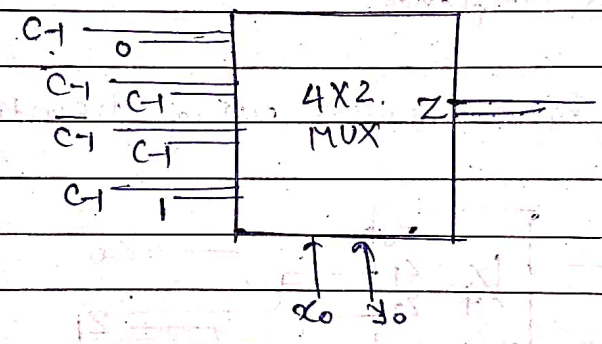


→ Realization of full adder using MUX (8×1) 8 way 8 i/p 2 bit (must be 2 bit)

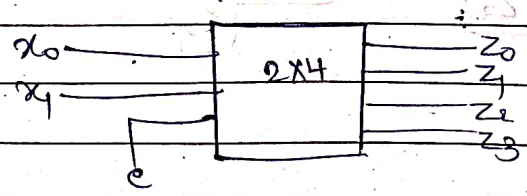
x_0	y_0	C_{-1}	S_0	C_0
0	0	0	$\overline{C_{-1}}$	0
0	0	1	$\overline{C_{-1}}$	0
0	1	0	$\overline{C_{-1}}$	C_{-1}
0	1	1	$\overline{C_{-1}}$	C_{-1}
1	0	0	$\overline{C_{-1}}$	C_{-1}
1	0	1	$\overline{C_{-1}}$	C_{-1}
1	1	0	C_{-1}	1
1	1	1	C_{-1}	1



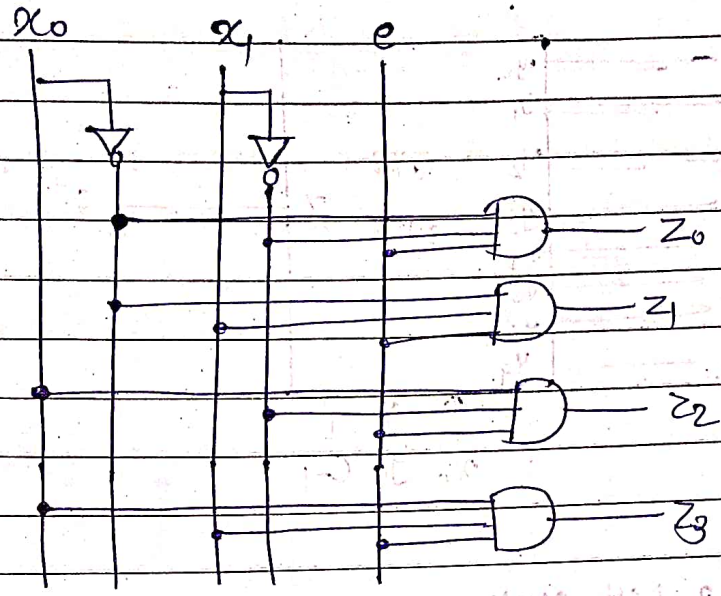
4 i/p 2 bit MUX



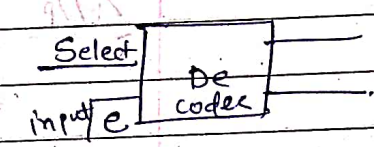
→ Decoder / Encoder :



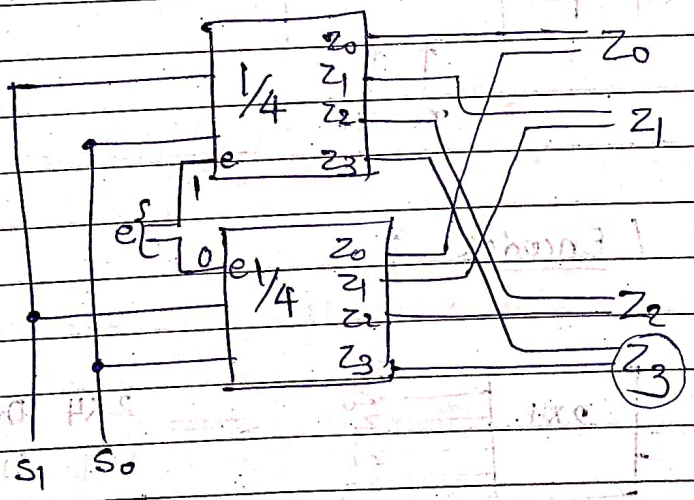
⇐ 2X4 Decoder
1/4 Decoder
$1/2^n$ decoder, n = i/ps.



Use : p-ROM (select the address)
In Demultiplexer



ex. 4 o/p 2 bit DeMUX using Decoder



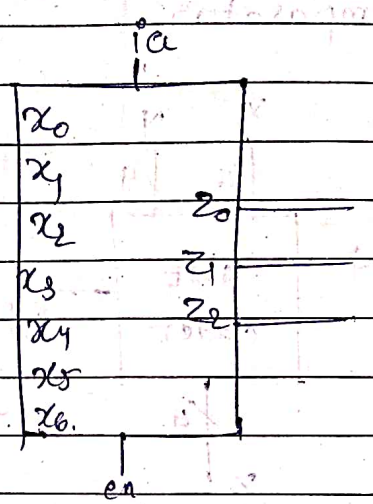
ex. 8 o/p 3 bit DeMUX.

DATE:
PAGE:

→ Priority Encoder

ex. 8x3 Priority Encoder

Block diagram



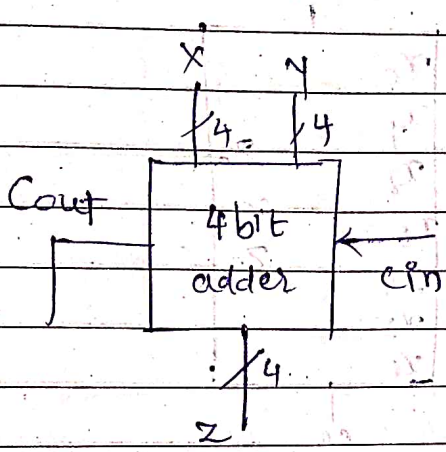
e	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	z_2	z_1	z_0	ia (active)
0	x	x	x	x	x	x	x	x	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	0	0	1	x	0	0	1	1
1	0	0	0	0	0	1	x	x	0	1	0	1
1	0	0	0	0	1	x	x	x	0	1	1	1
1	0	0	0	1	x	x	x	x	1	0	0	1
1	0	0	1	x	x	x	x	x	1	0	1	1
1	0	1	x	x	x	x	x	x	1	1	0	1
1	1	x	x	x	x	x	x	x	1	1	1	1

$$z_2 = \overline{x_7}\,\overline{x_6}\,\overline{x_5}\,x_4 + \overline{x_7}\,\overline{x_6}\,x_5 + \overline{x_7}\,x_6 + x_7$$
$$= x_7 + x_6 + x_5 + x_4$$
$$z_1 = x_7 + x_6 + x_3 + x_2$$
$$z_0 = x_7 + x_5 + x_2 + x_0$$

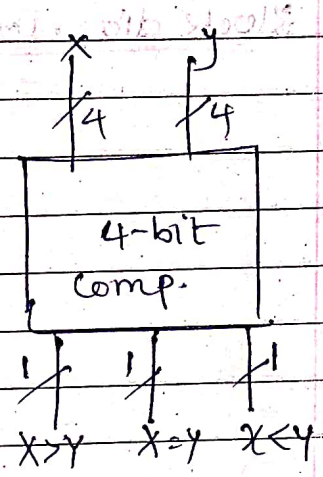
DATE:
PAGE:

Arithmetic elements :

1. Adder
2. Comparator



(1)



(2)

Case 1 $X > Y \quad X - Y > 0 \quad \text{---(1)}$

$Y = (2^n - 1) - \bar{Y} \quad \text{---(2)}$

From (1) & (2),

$X - [(2^n - 1) - \bar{Y}] > 0$

$X + \bar{Y} \geq 2^n - 1$

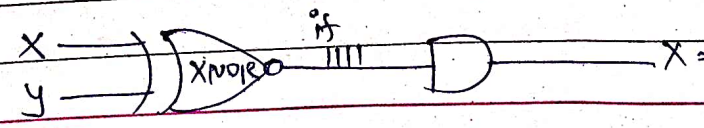
↳ if cout is 1, then $X > Y$

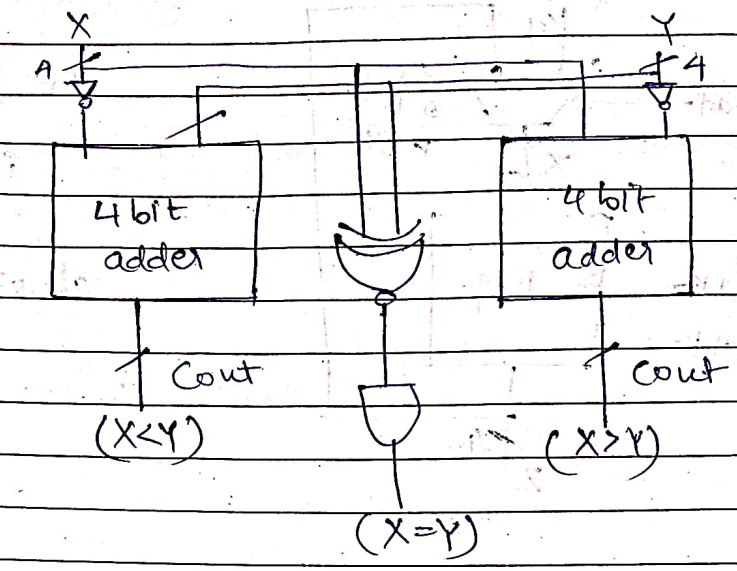
Case 3 $X < Y \implies Y > X$

$\therefore Y + \bar{X} > 2^n - 1$

↳ if carry generated,

Case 2

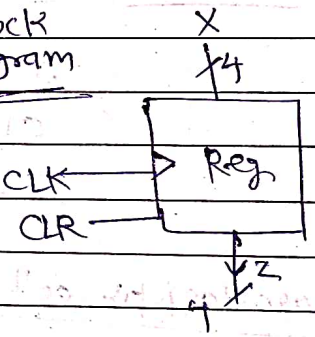




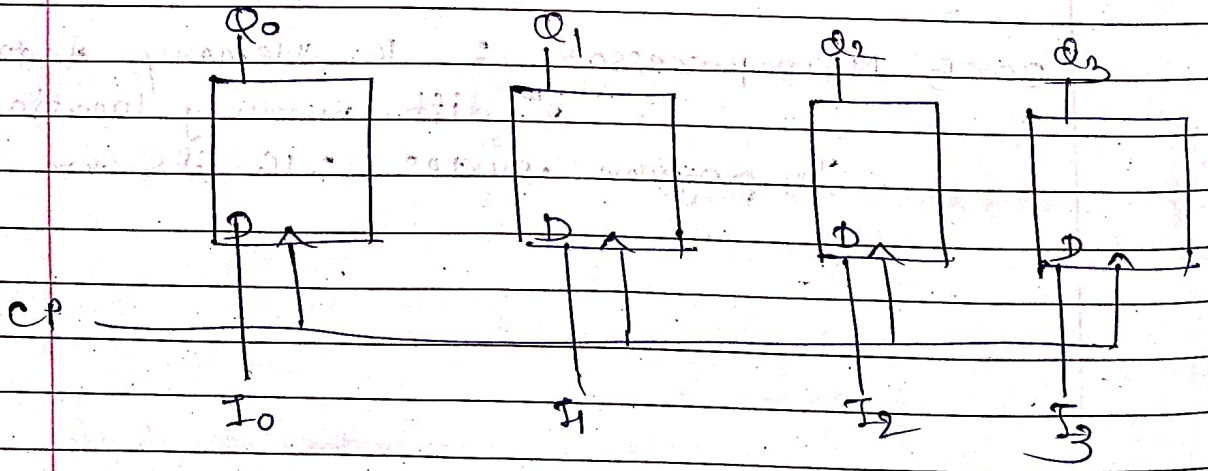
Sequential Elements

Registers
→ parallel
→ shift → left
→ Right

Block diagram

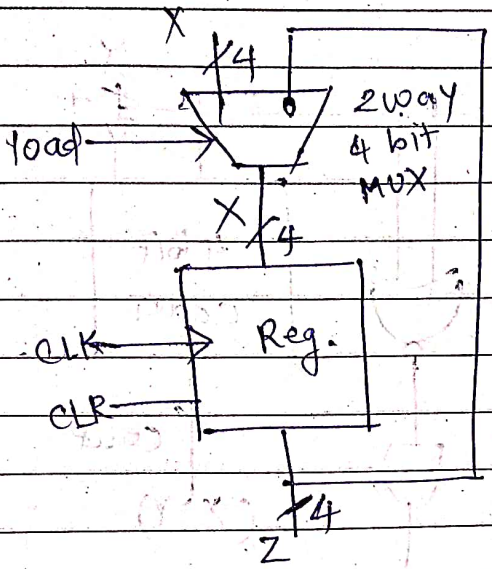


Ex. 4-bit parallel register



left = $x, Z_{m-1}, Z_{m-2}, \ldots Z_0$

right = $Z_{m-1}, Z_{m-2}, \ldots, Z_0, x$

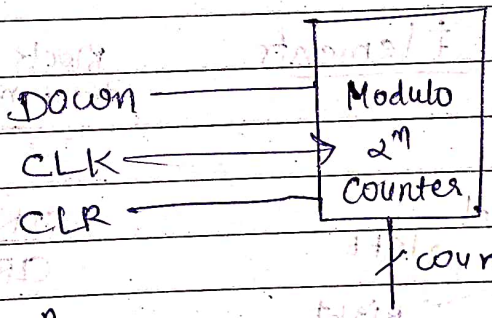


IF i/p is 0 ⇒ no change

i/p is 1 ⇒ i/p (new)

Characteristic eqn. = ?

Counter



Characteristic eqn

up counter $S_{i+1} = S_i$ plus 1

down counter $S_{i+1} = S_i$ minus 1

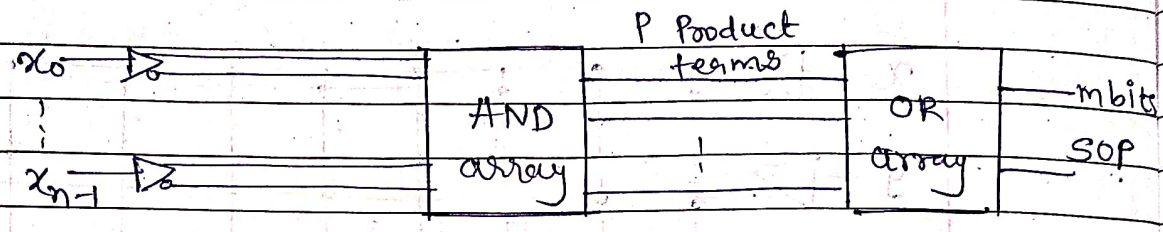
8085 Microprocessor : 16 Memory Address

2^{16} diff. memory locations

In program counter : 16 ffs are required

DATE:
PAGE:

PLD



n inputs X p product terms X m outputs

- divided into three types.

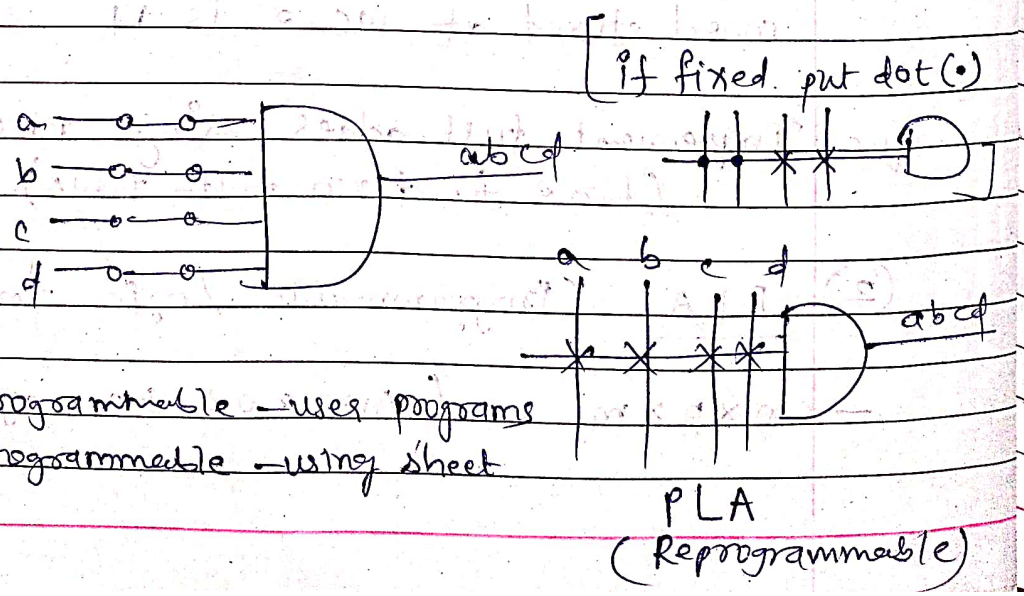
	AND	OR
PROM	fixed	programmable
PLA	prog.	prog
PAL	prog.	fixed

Inflexable

(1) PROM (Programmable Read only memory)

- n to 2^n decoder.
- n i/p max. product terms 2^n AND gates (fixed)

8X2 → bit



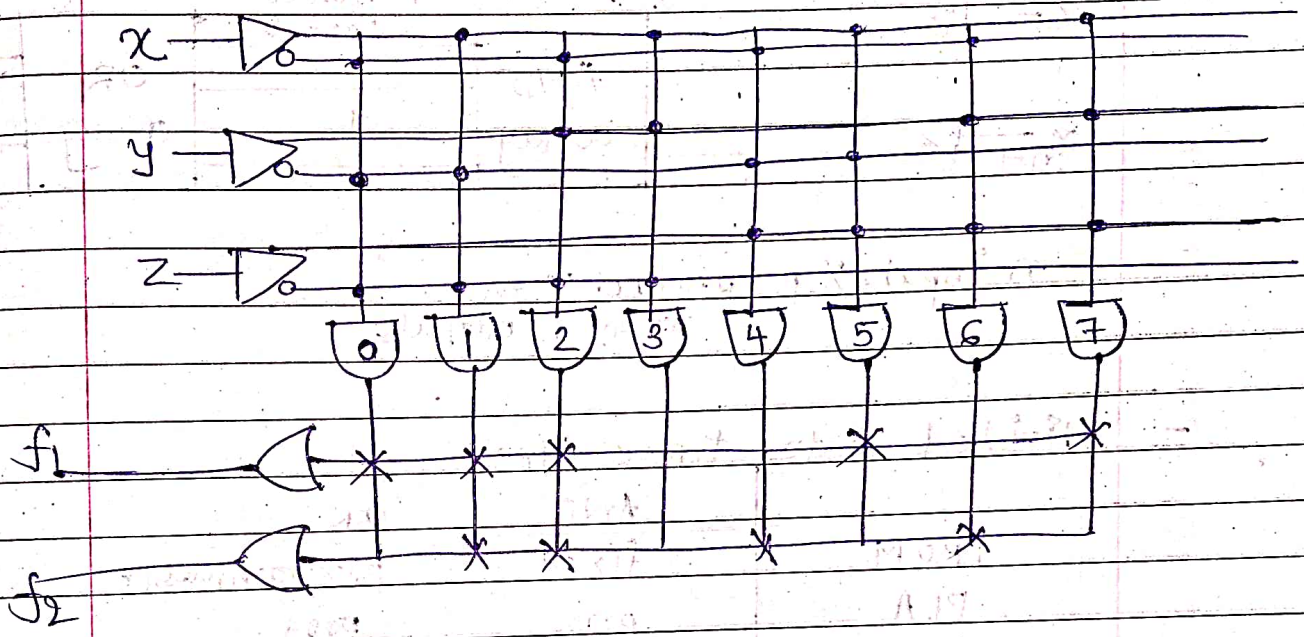
field programmable – user programs

Mask programmable – using sheet

PLA (Reprogrammable)

DATE:
PAGE:

eg 8×2 (8 i/p s and 2 bit) (PROM)



que Implement $f_1(x,y,z) = \Sigma(0,1,2,5,7)$
$f_2(x,y,z) = \Sigma(1,2,4,6)$

⟹ f1 & f2 = 2 OR gates
x, y, z = 3 i/ps and 8 AND gates.

– coord stored at loc. 6 is 01
coord stored at loc. 2 is 11

que Implement full adder using PROM.
(have to draw truth table)

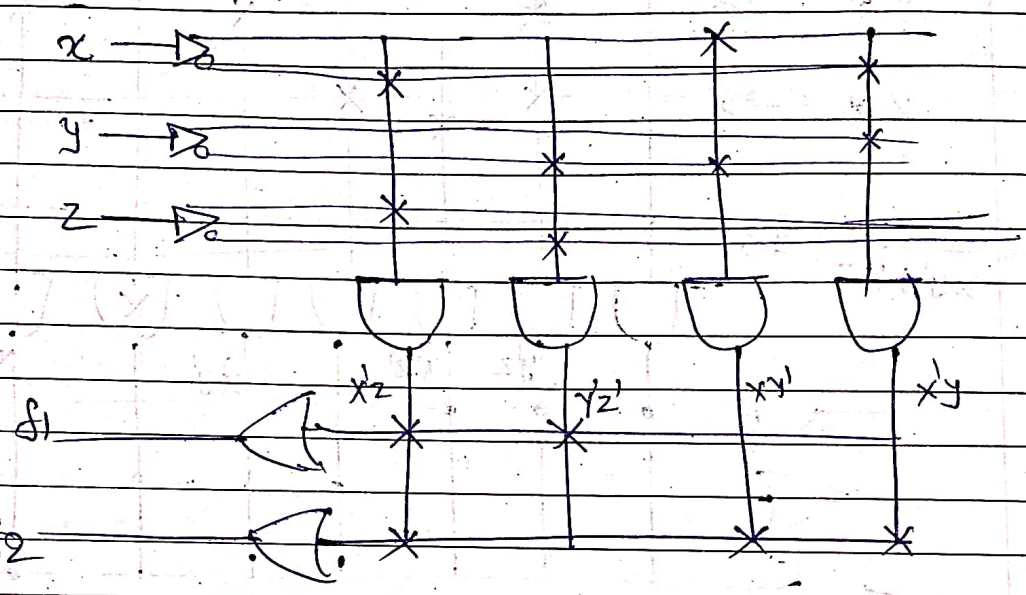
(2) PLA (Programmable Logic Array)

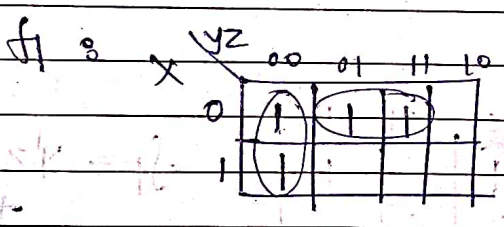
→ n×p×m (AND & OR are prog.)

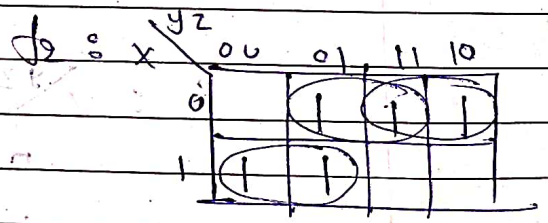
DATE:
PAGE:

Que $f_1 = \Sigma(0,1,3,4)$
$f_2 = \Sigma(1,2,3,4,5)$
Implement above using 3X4X2 PLA.

→







$f_1 = y'z' + \underline{x'z}$
common

$f_2 = xy' + \underline{x'z} + x'y$
common

Minterms = $\underset{3}{xy'} + \underset{2}{y'z'} + \underset{1}{x'z} + \underset{4}{x'y}$

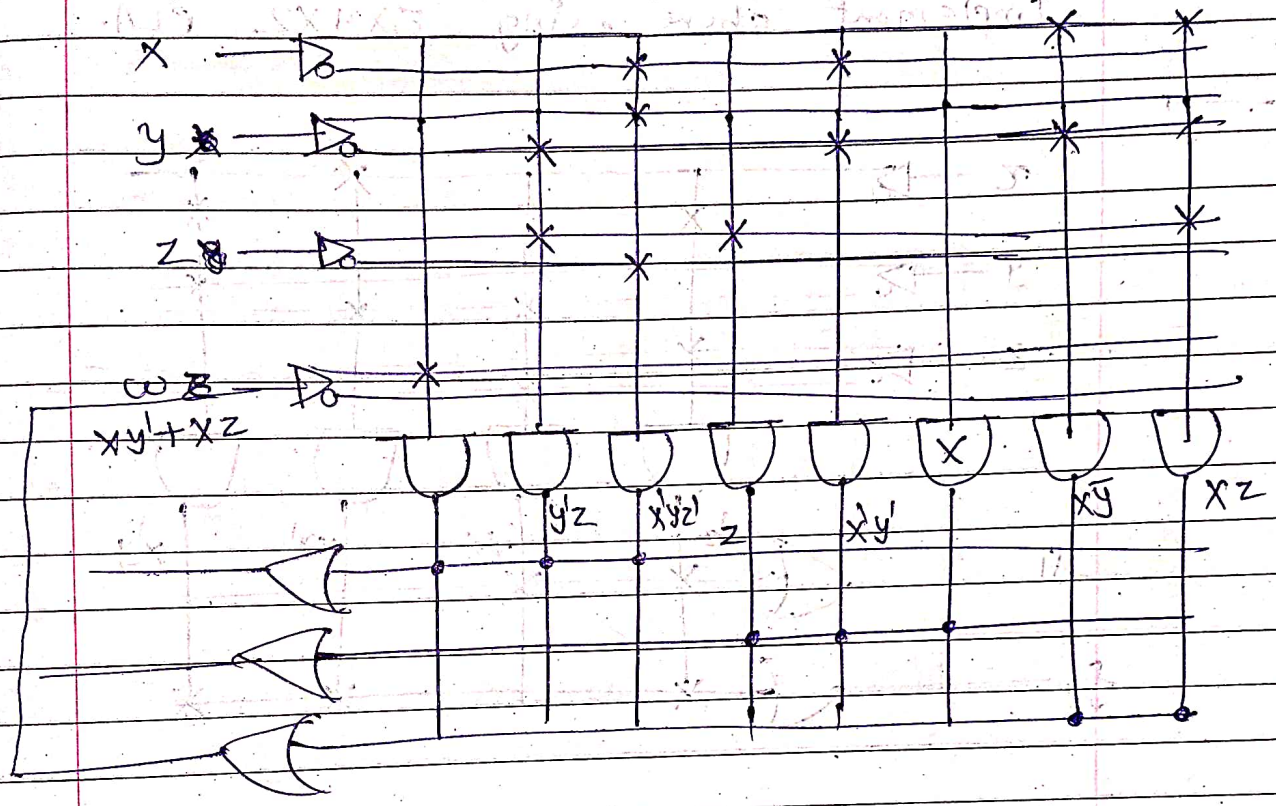
(3) PAL (Programmable Array Logic)

— mxpxm (OR is fixed)

DATE:
PAGE:

que $f_1(x,y,z) = \Sigma(1,2,4,5,7)$ 4X8X3

$f_2(x,y,z) = \Sigma(0,1,3,5,7)$



f_1 :

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

$f_1 = y'z + xz + xy' + x'yz'$

f_2 :

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

$f_2 = y'z + xz$

$f_2 = z + x'y'$

DATE:
PAGE:

que Implement the f^n using PLA

$f_1 = \Sigma m(0,1,3,5)$

$f_2 = \Sigma m(3,5,7)$

3X4X2.

f_1 :

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

$f_1 = x'y' + x'z + xy'z$

f_2 :

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

$f_2 = xy'z + yz.$

que $f_1 = \Sigma m(3,5,6,7)$

$f_2 = \Sigma m(0,2,4,7)$

8X4X2

f_1 :

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

$f_1 = xz + x'yz + xyz'$

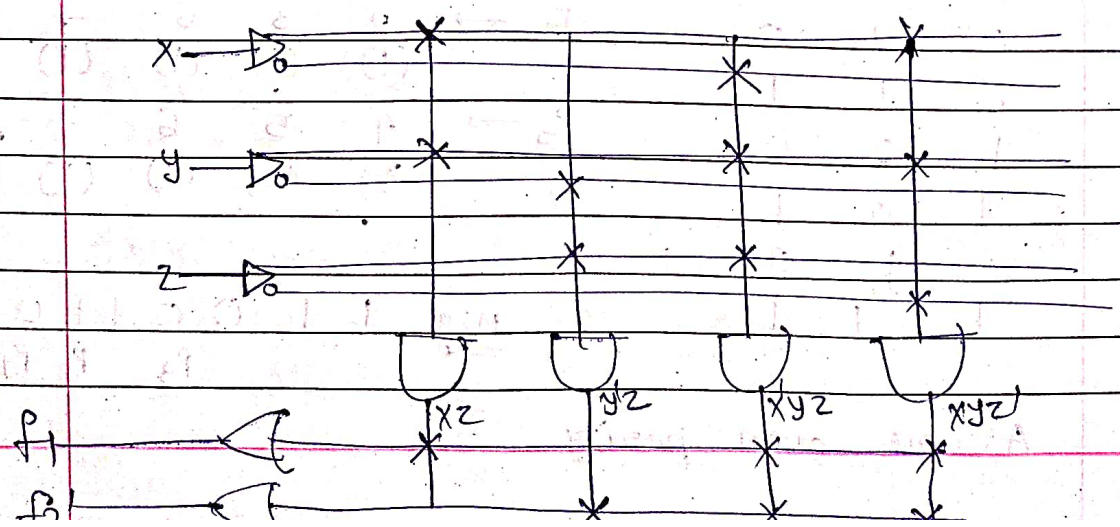
f_2 :

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

f_2' :

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

$f_2' = y'z + x'yz + xyz'$



DATE:
PAGE:

* Hamming Code :

→ two variations : (i) single error correction (SEC)
(ii) Single error correction double error detection (DED)

Tx

1. Calculation of Redundant data

$2^r \geq m+r+1$

where m = data bits
r = redundant bits

eg. $m=4$, $2^r \geq 5+r$
$\Rightarrow \boxed{r=3}$ parity
$m+r = 4+3$
$= 7$ total no. of bits
(7,3) Hamming code

2. Bit position table

				2^2	2^1	2^0
7	6	5	4	3	2	1
D_4	D_3	D_2	P_3	D_1	P_2	P_1
1	1	0		1		

(Redundant)
- parity bits are always placed at position of 2^n.

P_3	P_2	P_1
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

$P_1 \rightarrow$ 1 (0), 3 (1), 5 (0), 7 (1) $\boxed{P_1=0}$

$P_2 \rightarrow$ 2 (1), 3 (1), 6 (1), 7 (1) $\boxed{P_2=1}$

$P_3 \rightarrow$ 4 (0), 5 (0), 6 (1), 7 (1) $\boxed{P_3=0}$

Msg 1 1 0 0 1 1 0
P_3 P_2 P_1

Assume even parity

DATE:
PAGE:

RX

1. Recalculate Redundant bits

$2^r \geq m+r+1$

but $m+r=7 \Rightarrow \boxed{r=3}$

2. Bit position table

7	6	5	4	3	2	1
D_4	D_3	D_2	P_3	D_1	P_2	P_1
1	1	1	0	1	1	0

$P_1'' \rightarrow$ 1, 3, 5, 7 → 0, 1, 1, 1 $\quad \boxed{P_1''=1}$

$P_2'' \rightarrow$ 2, 3, 6, 7 → 1, 1, 1, 1 $\quad \boxed{P_2''=0}$

$P_3'' \rightarrow$ 4, 5, 6, 7 → 0, 1, 1, 1 $\quad \boxed{P_3''=1}$

$P_3'' P_2'' P_1'' \neq 0$

1 0 1 → gives position of error bit which was going to flip.

Ex. $r=8 \Rightarrow 2^8 \geq m+9$

$m \leq 247$

total bits $\Rightarrow 2^8-1 = \boxed{255}$ $\quad [(m+r)]$

no. of databits $\Rightarrow \boxed{m \leq 247}$

DATE:
PAGE:

EX. msg = 101101
encode using hamming code

(I) $2^r \geq m+r+1$

here, $m = 6$

$2^r \geq 6+r+1$

$\boxed{r=4}$ $\quad m+r = 10$

(II)

		2^3				2^2		2^1	2^0
10	9	8	7	6	5	4	3	2	1
1	0		1	1	0		1		
D_6	D_5	P_4	D_4	D_3	D_2	P_3	D_1	P_2	P_1

(III)

P_1 — 1 (0), 3 (1), 5 (0), 7 (1), 9 (0) $\quad \boxed{P_1 = 0}$

P_2 — 2 (0), 3 (1), 6 (1), 7 (1), 10 (1) $\quad \boxed{P_2 = 0}$

P_3 — 4 (0), 5 (0), 6 (1), 7 (1) $\quad \boxed{P_3 = 0}$

P_4 — 8 (1), 9 (0), 10 (1) $\quad \boxed{P_4 = 1}$

transmitted msg: 10111 $\overline{\text{X}}$ 00100

↑ flipping bit

Received msg: 10110 00100
(RX)

DATE:
PAGE:

(I) $2^r \geq m+r+1$

but $m+r = 10$

$\therefore 2^r \geq 11 \implies \boxed{r=4}$

(II)

10	9	8	7	6	5	4	3	2	1
1	0	1	1	0	0	0	1	0	0
D_6	D_5	P_4	D_4	D_3	D_2	P_3	D_1	P_2	P_1

P_1'' — 1 3 5 7 9 → (0) (1) (0) (1) (0) → $\boxed{P_1''=0}$

P_2'' — 2 3 6 7 10 → (0) (1) (0) (1) (1) → $\boxed{P_2''=1}$

P_3'' — 4 5 6 7 → (0) (0) (0) (1) → $\boxed{P_3''=1}$

P_4'' — 8 9 10 → (1) (0) (1) → $\boxed{P_4''=0}$

$P_4'' P_3'' P_2'' P_1'' \neq 0$

0 1 1 0

→ 6th bit is flipped.

Ex. msg = 11011100111100101

(I) $2^r \geq m+r+1$

(14)

$2^r \geq 18 \implies \boxed{r=5}$

(II)

17	16	15	14	13	12	11	10	9	8	7
D_{12}	P_5	D_{11}	D_{10}	D_9	P_8	D_7	D_6	D_5	P_4	P_4
(1)	(1)	(0)	(1)	(1)	(1)	(0)	(0)	(1)	(1)	(1)

6	5	4	3	2	1
D_3	P_2	P_3	D_1	P_2	P_1
(1)	(0)	(0)	(1)	(0)	(1)

P_1'' →	1	3	5	7	9	10	13	15	17
	(1)	(1)	(0)	(1)	(1)	(0)	(1)	(0)	(1)
P_2'' →	2	3	6	7	10	11	14	15	
	(0)	(1)	(1)	(1)	(0)	(0)	(1)	(0)	
P_3'' →	4	5	6	7	12	13	14	15	
	(0)	(0)	(1)	(1)	(1)	(1)	(1)	(0)	
P_4'' →	8	9	10	11	12	13	14	15	
	(1)	(1)	(0)	(0)	(1)	(1)	(1)	(0)	
P_5'' →	16	17							
	(1)	(1)							

$P_5'' P_4'' P_3'' P_2'' P_1'' \neq 0$

0 1 1 0 0

↳ 12th bit

DATE:
PAGE:

Ex. Encode the msg: 1010 1001 1101

(I) $2^r \geq m + r + 1$

$2^r \geq 13 \Rightarrow \boxed{r = 4}$

(II)

12	11	10	9	8	7	6	5	4	3	2	1
D_8	D_7	D_6	D_5	P_4	D_4	D_3	D_2	P_3	D_1	P_2	P_1
1	0	1	0	1	0	0	1	1	1	0	1

P_1'' — 1 3 5 7 9 11 (1) (1) (1) (0) (0) (0) $\boxed{P_1'' = 1}$

P_2'' — 2 3 6 7 10 11 (0) (1) (0) (0) (1) (0) $\boxed{P_2'' = 0}$

P_3'' — 4 5 6 7 12 (1) (1) (0) (0) (1) $\boxed{P_3'' = 1}$

P_4'' — 8 9 10 11 12 (1) (0) (1) (0) (1) $\boxed{P_4'' = 1}$

$P_4'' P_3'' P_2'' P_1'' \neq 0$

1 1 0 1

(I) $2^r \geq m + r + 1$

$2^r \geq 12 + r + 1 \Rightarrow r = 5$ $\quad m + r = 17$

17	16	15	14	13	12	11	10	9	8	7	6	5	4
D_{12}	P_5	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	P_4	D_4	D_3	P_2	P_1
(1)	(0)	(0)	(1)	(0)	(1)	(0)	(0)	(1)		(1)	(1)	(0)	

8	2	1
D_1	P_2	P_1
(1)		

DATE:
PAGE:

P_1 = 1 3 5 7 9 11 13 15 17 → $P_1 = 0$
(0) (1) (0) (1) (1) (0) (0) (0) (1)

P_2 = 2 3 6 7 10 11 14 15 → $P_2 = 0$
(0) (1) (1) (1) (0) (0) (1) (0)

P_3 = 4 5 6 7 12 13 14 15 → $P_3 = 0$
(0) (0) (1) (1) (1) (0) (1) (0)

P_4 = 8 9 10 11 12 13 14 15 → $P_4 = 1$
(1) (1) (0) (0) (1) (0) (1) (0)

P_5 = 16 17 → $P_5 = 1$
(1) (1)

transmitted msg: 11-0101001111 00100.

- **Extended Hamming Code :**

msg : 11011101

(I) $r = 4$ (12, 4) — m+r, redundant bits

(II)

12	11	10	9	8	7	6	5	4	3	2	1
D_8	D_7	D_6	D_5	P_4	D_4	D_3	D_2	P_3	D_1	P_2	P_1
1	1	0	1		1	1	0		1		

P_1 = 1 3 5 7 9 11 → $P_1 = 0$
(0) (1) (0) (1) (1) (1)

P_2 = 2 3 6 7 10 11 → $P_2 = 0$
(0) (1) (1) (1) (0) (1)

P_3 = 4 5 6 7 12 → $P_3 = 1$
(1) (0) (1) (1) (1)

P_4 = 8 9 10 11 12 → $P_4 = 1$
(1) (1) (0) (1) (1)

DATE:
PAGE:

Encoded msg : 1 1 0 1 1 1 1 0 1 1 0 0

Received msg : 1 1 0 1 1 1 1 1 1 0 0

(I) P_1'' = 1 3 5 7 9 11 → $P_1''=1$ (same as transmitter side method)
(1) (1) (1) (1) (1) (1)

P_2'' = 2 3 6 7 10 11 → $P_2''=0$
(0) (1) (1) (1) (0) (1)

P_3'' = 4 5 6 7 12 → $P_3''=0$
(0) (0) (1) (1) (1)

P_4'' = 8 9 10 11 12 → $P_4''=1$
(1) (1) (0) (1) (1)

$P_4'' P_3'' P_2'' P_1''$ ⊕ (XOR) $P_4 P_3 P_2 P_1$

1 0 0 1 ⊕ 1 1 0 0 = 0 1 0 1 (5th bit)

Syndrome calculator

SECDED

1 1 0 1 1 1 1 0 1 1 0 0 0 ← GP ← even no. of 1's ⇐ Encoded msg

case-I : $P_4'' P_3'' P_2'' P_1''$
code / Syndrome = 0 & GP = 0
∴ No error

case-II : code / syndrome = 0 & GP = 1 → only flipped GP

Case-III syn. ≠0 CRP = 1 Single error detection

Case-IV syn ≠0, CRP = 0 Double error detection

	P_4			P_3	P_2	P_1	CRP
1 1 0 1	1	1	1	0	1	0	0
1 1 0 1	1	0	1	1	1	0	0

P_1'' ← 1 3 5 7 9 11

0 1 1 0 1 1 $P_1'' = 0$

P_2'' ← 2 3 6 7 10 11 $P_2'' = 1$

1 1 1 0 0 1

P_3'' ← 4 5 6 7 12 $P_3'' = 1$

1 1 1 0 1

P_4'' ← 8 9 10 11 12 $P_4'' = 1$

1 1 0 1 1

$P_4'' P_3'' P_2'' P_1''$

1 1 1 0

⊕ 1 1 0 0

0 0 1 0

DATE:
PAGE:

12	11	10	9	8	7	6	5	4	3	2	1	
D_8	D_7	D_6	D_5	P_4	D_4	D_3	D_2	P_3	D_1	P_2	P_1	
1	1	0	1	1	1	1	0	1	1	0	0	GCP = ?
			0			0						GCP = ?

$P_1'' = 1\ 3\ 5\ 7\ 9\ 11$
$(0)\ (1)\ (0)\ (1)\ (0)\ (1)$
$P_1'' = 0$

$P_2'' = 2\ 3\ 6\ 7\ 10\ 11$
$(0)\ (1)\ (1)\ (1)\ (0)\ (1)$
$P_2'' = 1$

$P_3'' = 4\ 5\ 6\ 7\ 12$
$(0)\ (0)\ (0)\ (1)\ (1)$
$P_3'' = 0$

$P_4'' = 8\ 9\ 10\ 11\ 12$
$(0)\ (0)\ (0)\ (1)\ (1)$
$P_4'' = 0$

P_4''	P_3''	P_2''	P_1''	GP	
0	0	1	1	0	Received
⊕ 1	1	0	0	0	Transmitted
1	1	1	1	0	

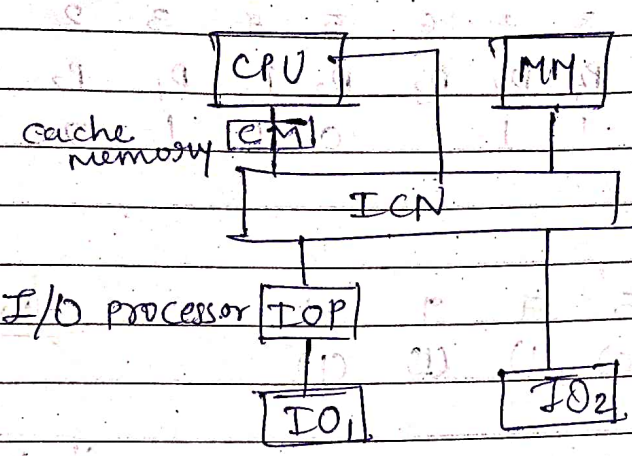
Syn ≠ 0, GP = 0 (Double error detection)

- **Processor Level :**

1. CPU
2. Memories
3. I/O
4. Interconnection network.

DATE:
PAGE:

Single processor System



Block Diagram of Computer System :

also called switching network / communication network.

why

→ **Synchronization Problem:**

- CPU task and IOP task is different
- CPU is faster > Main memory > I/O device. therefore there is no synchronization bet^n them.

Multi processor System