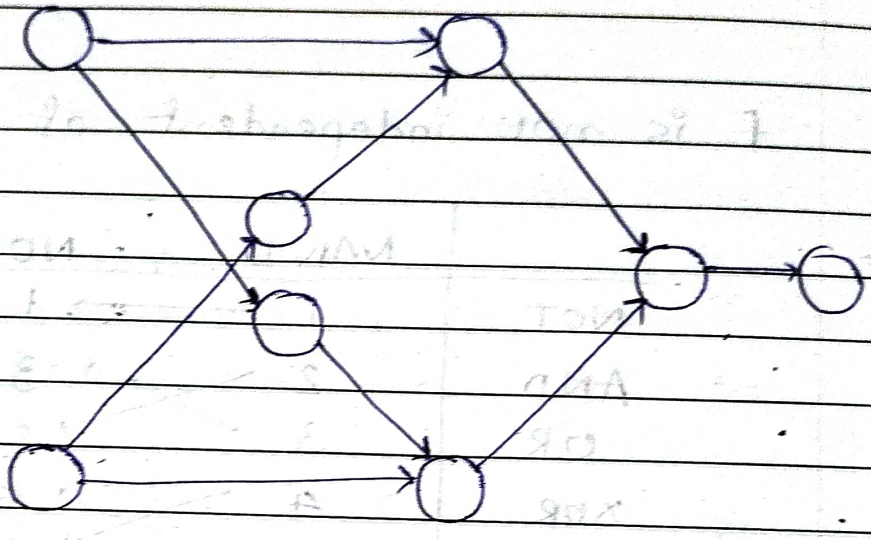
Design Methodology

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

→ Sytem Representation:

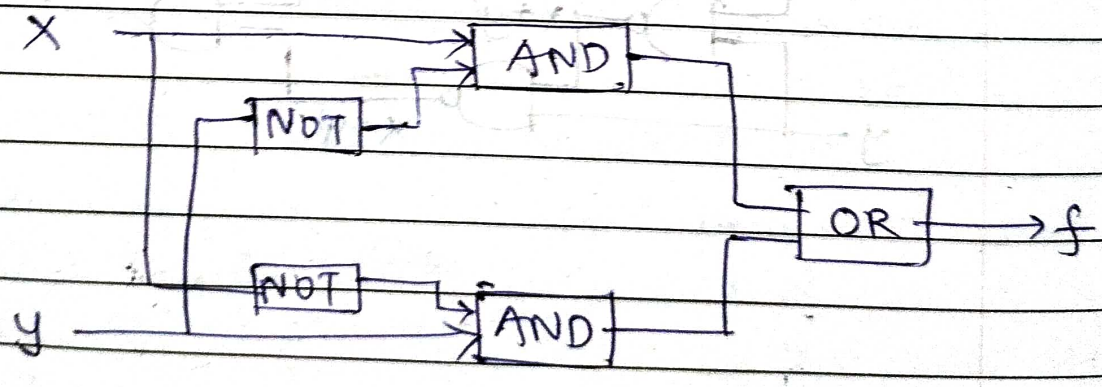


$V = \{v_1, v_2, \ldots\}$

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

$e_1 = \{v_1, v_2\}$

→ Schemetic diagram:



Structure vs Behaviour

↙ mathematical eqn ↘ truth table

two ways of

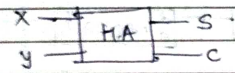
→ Hardware Description language (HDL)
VHDL, VERILOG

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 :

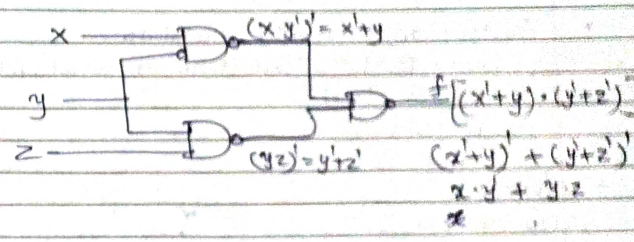
Entity is half-adder
port (x, y : in bit ; sum, carry : out bit) ;
end half adder

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)

→

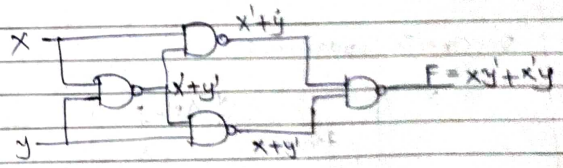


f is not independent of any variable.

→

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

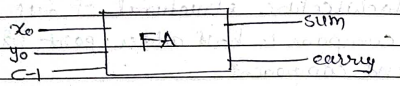
→



DATE:
PAGE:

→ VHDL Description of Full adder :

1. Behavioural VHDL Desc. :



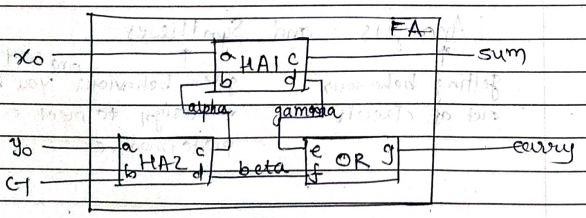
$sum = x_0 \oplus y_0 \oplus C_{-1}$

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

entity is full-adder
port (x_0, y_0, C_{-1} : in bit ; sum, carry : out bit)
end full adder

Architecture behaviour of full adder is
begin
sum ⇐ x_0 XOR y_0 XOR C_{-1} ;
carry ⇐ (x_0 AND y_0) OR (x_0 AND C_{-1}) OR (y_0 AND C_{-1})
end behaviour

2. Structural VHDL Desc :

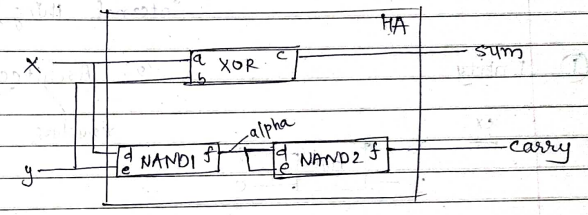


DATE:
PAGE:

→ Structural VHDL description :

⇐ : signal assignment

In half adder we can write.
half adder is
if 'co 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
end component

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

Signal alpha : bit
end half adder

Network list (Netlist)

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

Goals: Design should be correct
Performance maximize
cost minimize
power consumption none
reliability
compatibility

→ CAD ⟹ syntheses a design
prototype of the system editor
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} ns
(2)	Register (Reg. transfer RTL)	Reg, counter comb. circuit seq. circuit	MSI	word	10^{-9} to 10^{-6} s
(3)	Processors (architecture behaviour system)	CPU, Memories, I/O devices	VLSI	block of words	10^{-3} to 10^{3} s

entity is full adder
port (x0, y0, 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

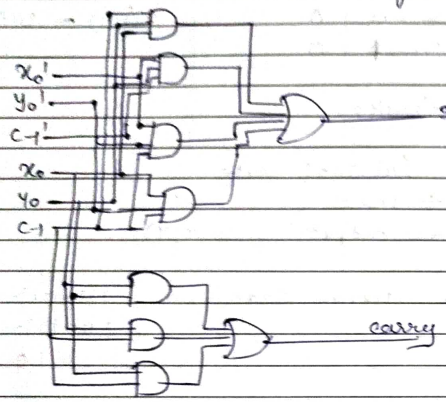
signal α, β, γ : bit)
end full adder
begin
HA1 : half adder portmap (a ⇒ x0, b ⇒ alpha,
c ⇒ sum, d ⇒ gamma);
HA2 : half adder portmap (a ⇒ y0, b ⇒ c-1,
c ⇒ alpha, d ⇒ beta);
OR : OR gate portmap (e ⇒ gamma, f ⇒ beta
g ⇒ carry)
end
end structure

→ Design Process

Analysis and Syntheses
↑ ↑
getting behaviour out of circuit — given behaviour and component you have to design to meet cost & performance

Using AND-OR

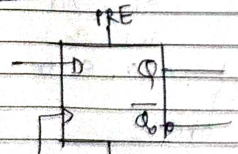
gate level



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 signals

① crate level / hardware design

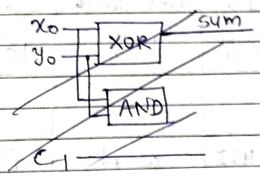
Full adder

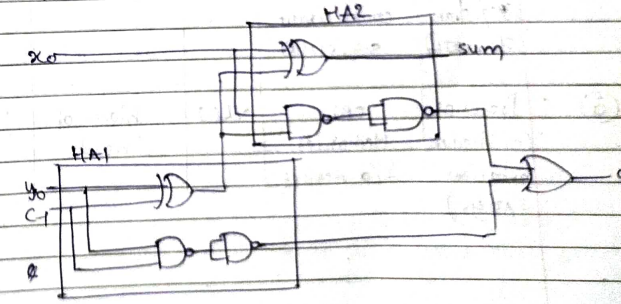
$$S_0 = x_0y_0C_{-1} + x_0'y_0C_{-1}' + x_0'y_0'C_{-1} + x_0y_0'C_{-1}'$$

$$C_0 = x_0y_0 + x_0C_{-1} + y_0C_{-1}$$
$$= (x_0+y_0)\cdot(x_0+C_{-1})\cdot(y_0+C_{-1})$$

Half adder

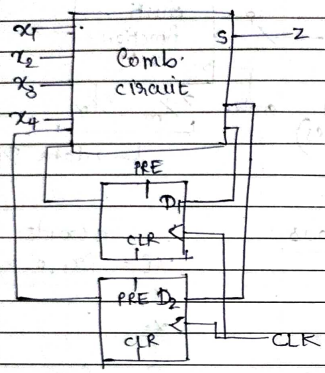
$$S_0 = x_0y_0' + x_0'y_0$$
$$C_0 = x_0y_0$$

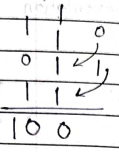




4 gate level

(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 / encoder
adder
ALU, PLD
shift reg, counter

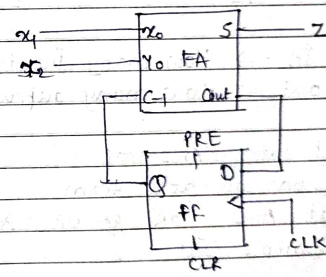
		Di	
		0	1 ← input
S(i)	$S_0=0$	0	1 S(i+1) = D(i)
PS	$S_1=\phi$	0	1

⟶ Seq. circuits.

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

(1) 2 bit stream adder

11
1101011 → x_1
1001111 → x_2
10

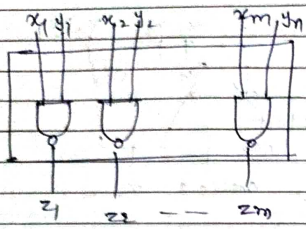


truth table :

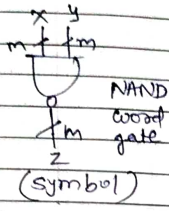
		x_1x_2			
		00	01	10	11
Present State	$S_0=0$	$S_0,0$	$S_0,1$	$S_0,1$	$S_1,0$
	$S_1=1$ (carry)	$S_0,1$	$S_1,0$	$S_1,0$	$S_1,1$

↑ 0/p (sum)

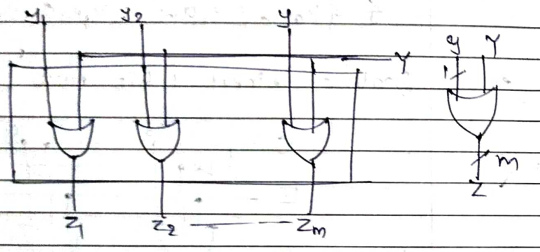
Logic diagram



Block diagram, (symbol)

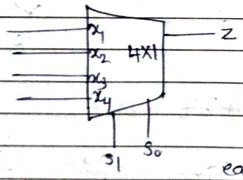


$f = y + Y$ (coord) (1 bit)



Multiplexes

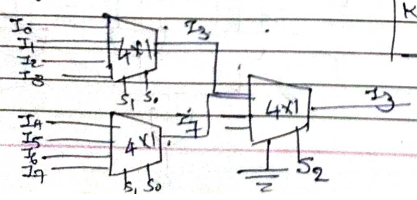
In OS
- data selecting
- routing



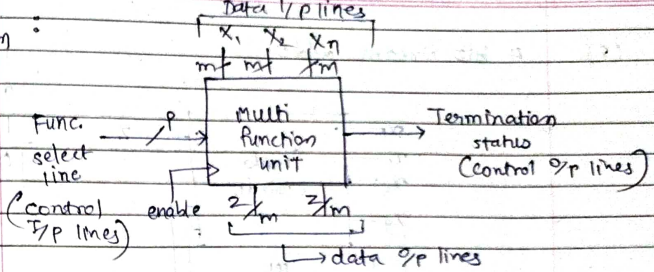
max. no. of data source = k and each IO data line carries m bits

K Input (K way)
m bit MUX

8X1 using only 4X1 MUX



Block representation:



→ Operations

n coords m bits

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

x_1

x_2

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

~~$z = ($~~

$z(x_1, x_2, \ldots, x_n) =$

$[z(x_{1,0}, x_{2,0}, \ldots x_{n,0}),$ } 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 ~~f~~ bit variables and m bit $= 2^{2^{nm}}$

coord gate $X = x_1 x_2 \ldots x_m$

$Y = y_1, y_2 \ldots y_m$

$g = \overline{xy}$

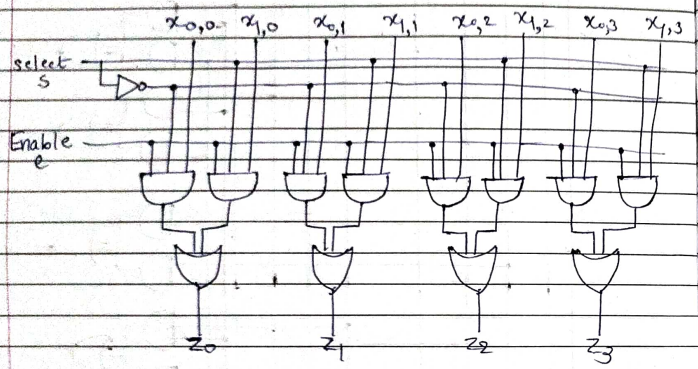
$z(\overline{xy}) = z_1(\overline{x_1 y_1}), z_2(\overline{x_2 y_2}), z_3(\overline{x_3 y_3}) \ldots z_m$

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

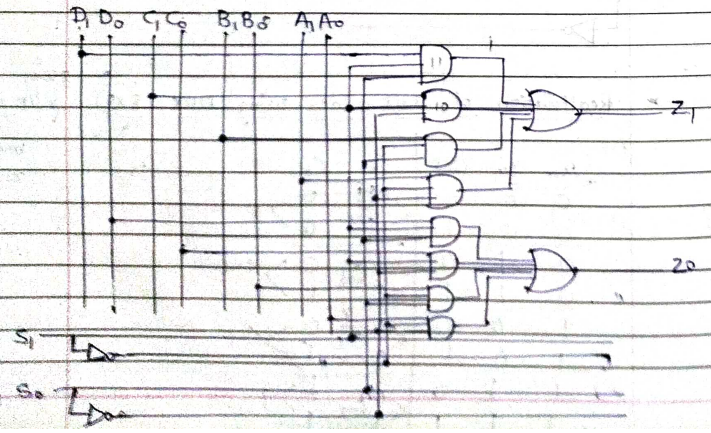
DATE:
PAGE:

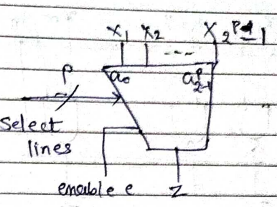
~~4 bit 2 bit MUX~~

* 2 Input 4 bit MUX.



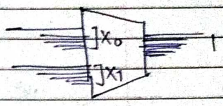
* 4 Input 2 bit MUX





k copy m bit registers

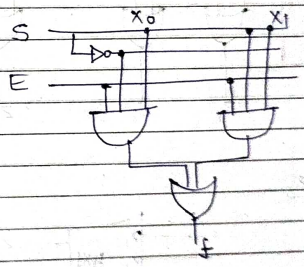
2 i/p 4 bit MUX



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

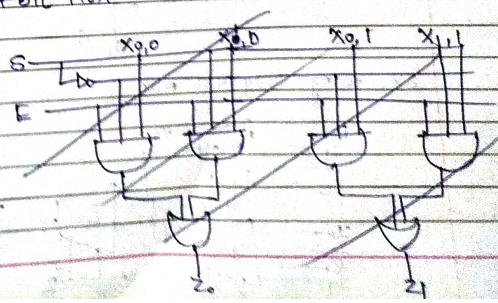
$f = (x_0a_0 + x_1a_1 + \dots x_{2^p-1}a_{2^p-1})E$

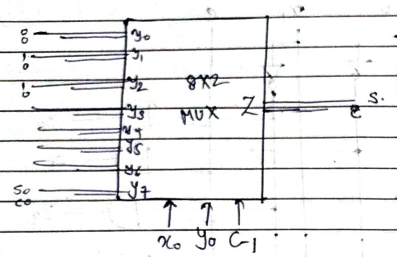
Ex Realize 2 input 1 bit MUX



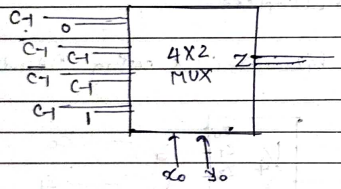
$f = (\bar{S}x_0 + Sx_1)E$

~~2 i/p 4 bit MUX~~

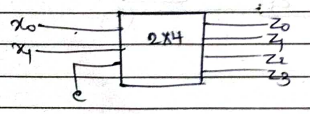




4/p 2 bit MUX

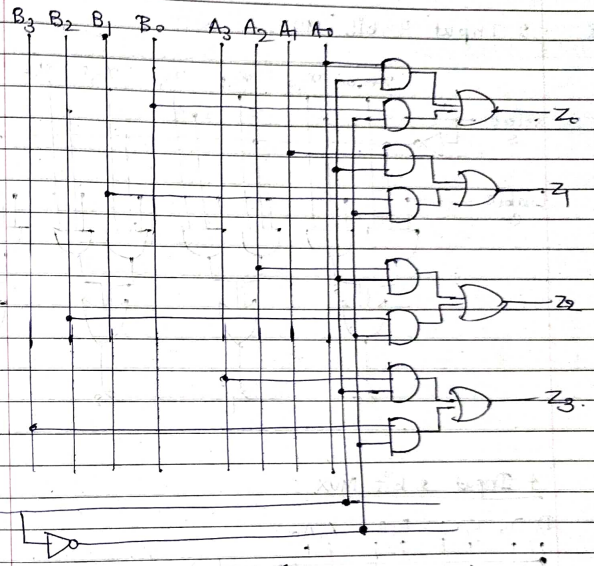


→ Decoder / Encoder :



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

Another Representation of 2 Input 4 bit MUX



→ Realization of full adder using MUX (8X1)

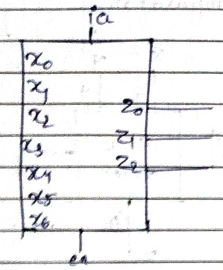
8 way
8 i/p 2 bit
must b
2 bit

x_0	y_0	C_{-1}	S_0	C_0
0	0	0	0	0
0	0	1	1	0
0	1	0	1	C_{-1}
0	1	1	0	1
1	0	0	1	C_{-1}
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Priority Encoder

ex. 8x3 Priority Encoder

Block diagram



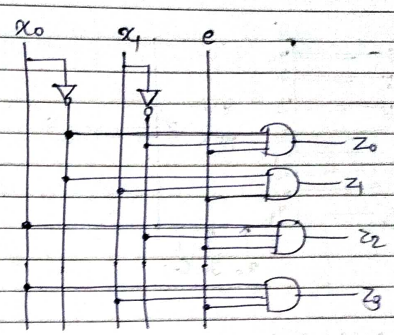
e	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	z_2	z_1	z_0	Ia (active)
0	x	x	x	x	x	x	x	x	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	0	0	1	x	0	0	1	1
1	0	0	0	0	0	1	x	x	0	1	0	1
1	0	0	0	0	1	x	x	x	0	1	1	1
1	0	0	0	1	x	x	x	x	1	0	0	1
1	0	0	1	x	x	x	x	x	1	0	1	1
1	0	1	x	x	x	x	x	x	1	1	0	1
1	1	x	x	x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

$z_2 = \bar{x_7}\bar{x_6}\bar{x_5}x_4 + \bar{x_7}\bar{x_6}x_5 + \bar{x_7}x_6 + x_7$

$= x_7 + x_6 + x_5 + x_4$

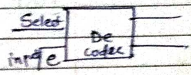
$z_1 = x_7 + x_6 + x_3 + x_2$

$z_0 = x_7 + x_5 + x_3 + x_1$

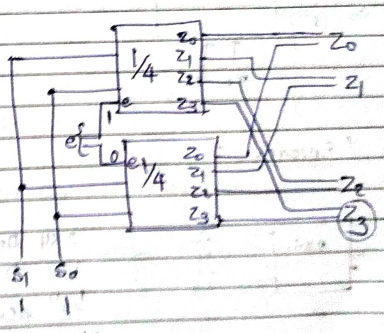


Use : p-ROM (select the address)

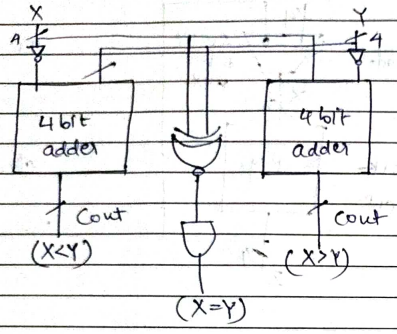
In Demultiplexer



ex. 4 o/p 2 bit DeMUX using Decoder



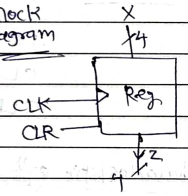
ex. 8 o/p 3 bit DeMUX.



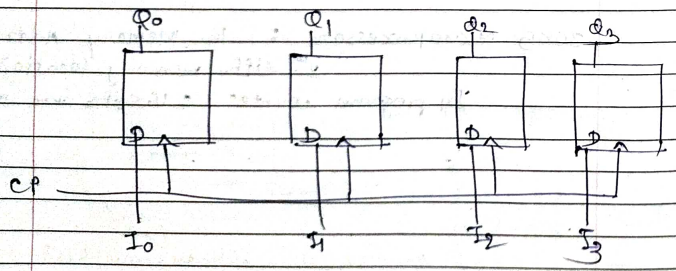
→ Sequential Elements

Registers
→ parallel
→ shift → left
→ Right

Block diagram



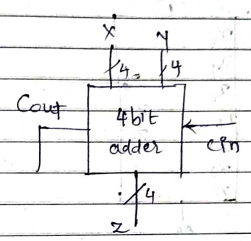
Ex. 4-bit parallel register



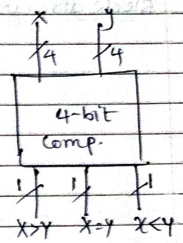
left = x, Z_{m-1}, Z_{m-2}, ... Z_0
right = Z_{m-1}, Z_{m-2}, ..., Z_0, x

→ Arithmetic elements :

1. Adder
2. Comparator



(1)



(2)

case 1 $X > Y$ $\quad X - Y > 0$ ——(1)

$Y = (2^n - 1) - \bar{Y}$ ——(2)

From (1) & (2),

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

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

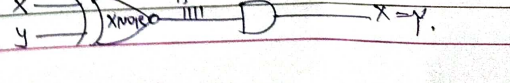
↳ if cout is 1, then X > Y

case 3 $X < Y \Rightarrow Y > X$

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

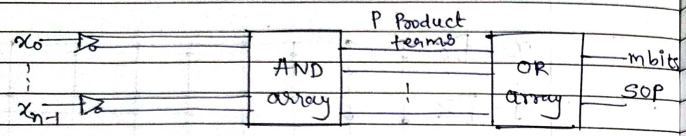
↳ if carry generated,

case 2



DATE:
PAGE:

PLD



n inputs X p product terms X m outputs.

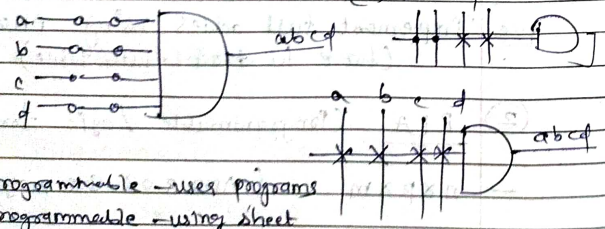
- divided into three types

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

Inflexable

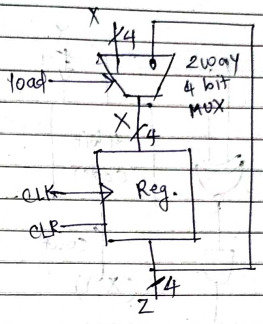
① PROM (Programmable Read only memory)
- n to 2^n decoder.
- n I/p max. product terms 2^n AND gates (fixed)

8X2 → bit



field programmable - users programs
mask programmable - using sheet

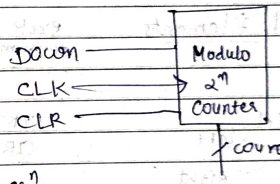
PLA
(Reprogrammable)



IF I/P is 0 ⇒ no change
I/P is 1 ⇒ I/P (new)

characteristic eqn. = I/P

Counter



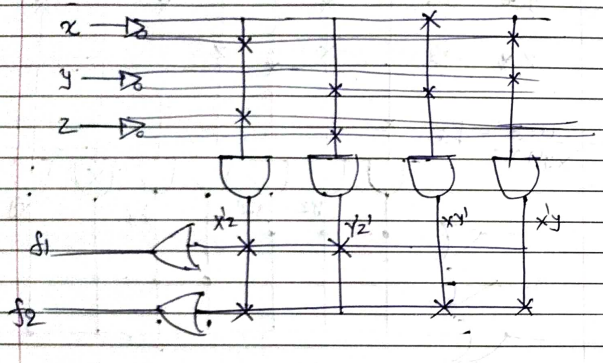
characteristic eqn

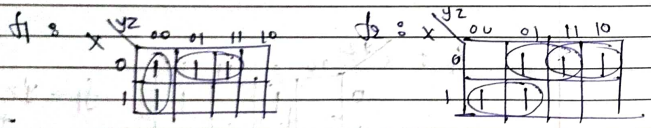
Up counter $S_{i+1} = S_i$ plus 1
down counter $S_{i+1} = S_i$ minus 1

8085 Microprocessor : 16 Memory Address
2^{16} diff. memory locations
In program counter : 16 ffs are requi

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)

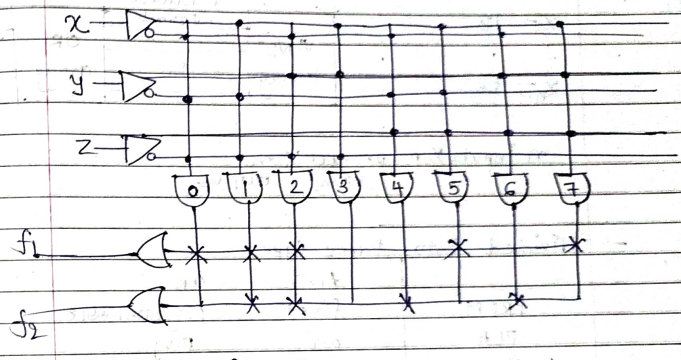
$f_2 = xy' + \underline{x'z} + x'y$ (common)

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)

ex 8X2 (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 addes using PROM.
(have to draco truth table)

(2) PLA (Programmable Logic Array)

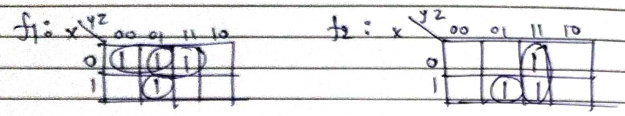
— mxpxm (AND & OR are prog.)

Que Implement the f^n using PLA

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

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

3X4X2



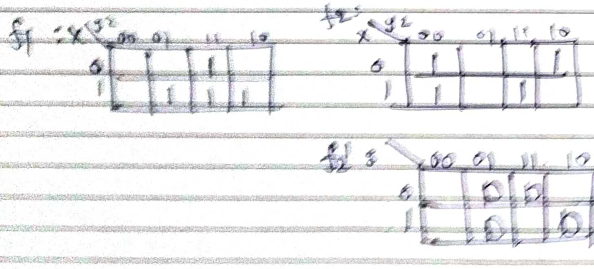
$f_1 = x'y' + x'z + xy'z$

$f_2 = xy'z + yz$

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

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

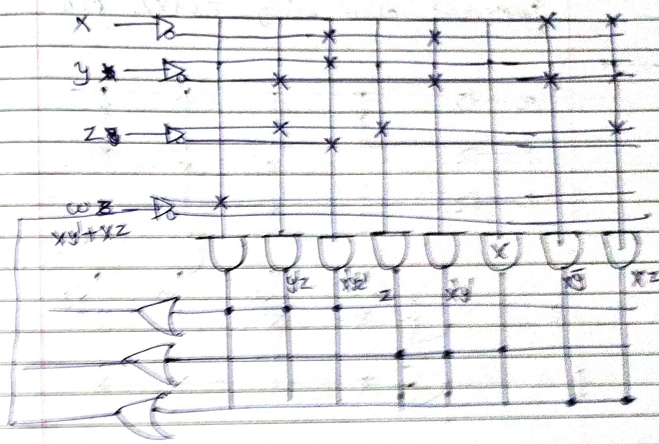
3X4X2

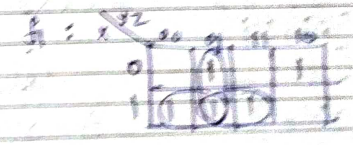


Que $f_1(x,y,z) = \Sigma(1,2,4,5,7)$

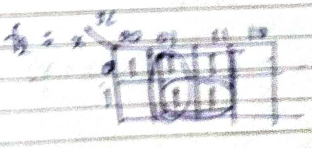
$f_2(x,y,z) = \Sigma(0,1,3,5,7)$

4X8X3





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



$f_2 = z + x'y$