Switching circuits and logic Design

* References

→ Zvi Kohavi & N.K. Jha (Switching & finite Automata Theory).
— Cambridge 3rd edition

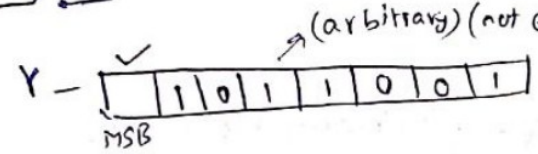
→ Fundamentals of logic Design
— Roth, Thompson.

→ Digital Design by Morris Mano, Pearson

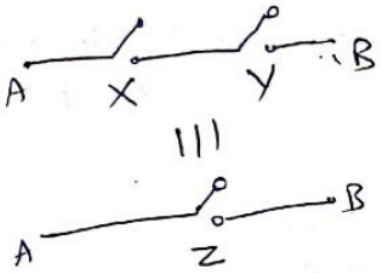
* Digital Design — any book.

*

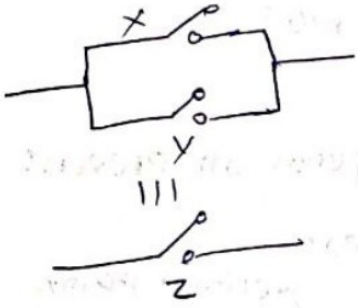
*



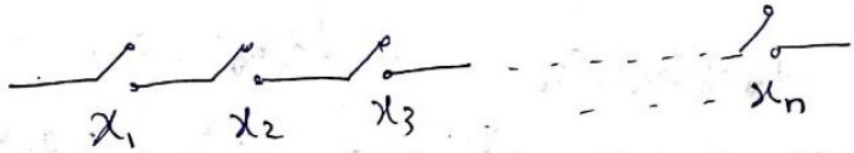
*



*



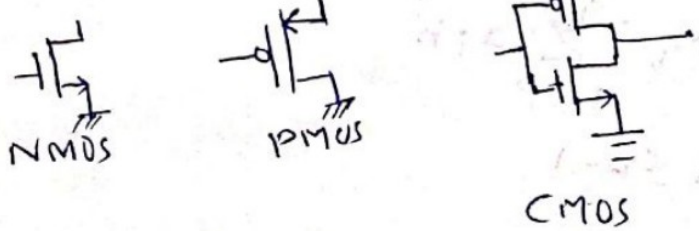
*



x_i → T(1) ⇒ x_i is switching variable
→ F(0)

$x_i = 0$, iff $x_i \neq 1$
$x_i = 1$, iff $x_i \neq 0$

*

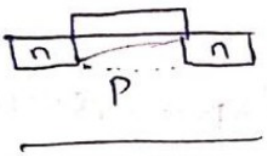


Scanned by CamScanner

* Power Consumption in CMOS is lesser than NMOS & pMOS because at any time only one of the transistors is on in CMOS. So there's no path from V_{DD} to ground

* SSI – Small scale Integration
  MSI – Medium "
  LSI – Large "
  VLSI – Very Large "
  VVLSI – Very Very Large "
  ULSI – Ultra Large "

*



NMOS

$\lambda = \frac{1}{2}$ (channel length)

λ values available are 1 μ, 0.25μ, 0.18μ, 0.13μ

* Switching variable X → 0, 1

using switches, we got AND, OR, NOT

* Combination Ckt. Design

→ present output depends on present input

* <u>Number Systems and Codes:-</u>

→ If $N_b = (a_{q-1} a_{q-2} \cdots a_0 \cdot a_{-1} a_{-2} \cdots a_{-p})$ (decimal point)

Then $N_b = a_{q-1} b^{q-1} + a_{q-2} b^{q-2} + \cdots + a_0 b^0 + a_{-1} b^{-1} + a_{-2} b^{-2} + \cdots + a_{-p} b^{-p}$

$$N_b = \sum_{i=-p}^{q-1} a_i b^i$$

each $0 \le a_i \le b-1$

Scanned by CamScanner

Hexadecimal - 16 → 0, 1, 2, ... 9, A, B, C, D, E, F

* $b_1 > b_2$

$$N_{b_1} = a_{q-1} b_2^{q-1} + a_{q-2} b_2^{q-2} + \cdots + a_0 b_2^0$$

↓ Representation of N in base b_1.

$$\frac{N_{b_1}}{b_2} = \left(a_{q-1} b_2^{q-2} + a_{q-2} b_2^{q-3} + \cdots + a_1 \right) + \frac{a_0}{b_2}$$

$\downarrow Q_0$

$$N_{b_1} = b_2 Q_0 + a_0$$

$$\therefore N_{b_1} = (a_{q-1} \cdots a_2 a_1 a_0)_{b_2}$$

Ex: $\frac{34_{10}}{2} = 17 + 0$

$\frac{17_{10}}{2} = 8 + 1$

$\frac{8_{10}}{2} = 4 + 0 \equiv 100010$

$\frac{4_{10}}{2} = 2 + 0$

$\frac{2_{10}}{2} = 1 + 0$

$\frac{1_{10}}{2} = 0 + 1$

For the RHS of decimal point,

$$N_{b_1} \times b_2 = a_{-1} + a_{-2} b_2^{-1} + a_{-3} b_2^{-2} + \cdots + a_{-p} b_2^{-(p-1)}$$

$34.12_{10} \equiv 100010.00011\ldots\ldots$

$\Rightarrow 0.12 \times 2 = 0.24$

$0.24 \times 2 = 0.48$

$0.48 \times 2 = 0.96$

$0.96 \times 2 = 1.92$

$0.92 \times 2 = 1.84$

Scanned by CamScanner

* Decimal — Binary

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

if $N = x_4 x_3 x_2 x_1$

$N = w_4 x_4 + w_3 x_3 + w_2 x_2 + w_1 x_1$

$w_1, w_2, w_3, w_4 \rightarrow$ weights associated (place value)

$N = (x_4 x_3 x_2 x_1)_8 = x_4 8^3 + x_3 8^2 + x_2 8 + x_1 8^0$

$8^3\ 8^2\ 8^1\ 8^0$

$x_i = 0, 1, 2, \ldots, 7$

* Binary to Octal

$(0\ 0\ 1\ :\ 0\ 1\ 1\ :\ 1\ 0\ 1)_2 = (135)_8$

1 3 5

$(1\ 0\ 1\ 1\ :\ 1\ 0\ 1)_2 = (5D)_{16}$

5 D

* Weighted code

8421-Code ← Binary system is called

BCD code → Binary Coded Decimal Code.

Decimal	2421 code (weights / place values)	642-3 code
0	0000	0000
1	0001	0101
2	0010/1000	0010
3	0011/1001	1001/0111
4	0100/1010	
5	0101/1011	
6	0110/1100	
7	0111/1101	
8	1110	
9	1111	

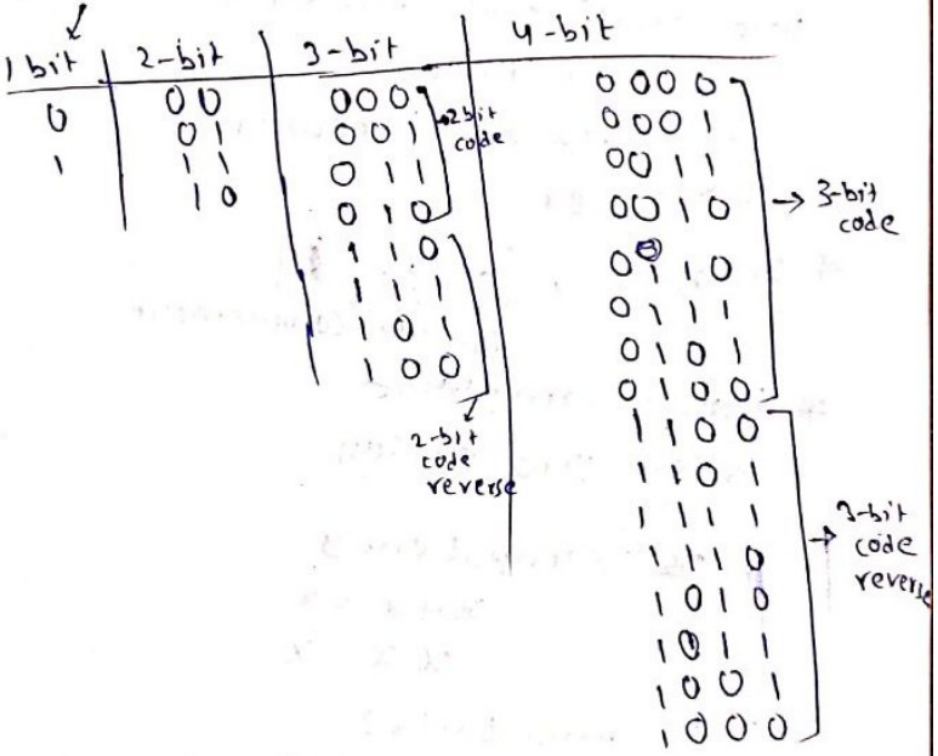
Scanned by CamScanner

* Non-weighted codes :

Decimal	Excess-3	Cyclic code
0	0011	
1	0100	
2	0101	
3	0110	
4	0111	
5	1000	
6	1001	
7	1010	
8	1011	
9	1100	

Cyclic code → Two consecutive codes differ in only one digit.

Gray code → popular cyclic code.



Decimals	Gray code	Binary
0	000	000
1	001	001
2	011	010
3	010	011
4	110	100
5	111	101
6	101	110

Scanned by CamScanner

n-bit Binary to n-bit Gray code (G) Conversion

(B)

$B_n \rightarrow G_n$

$g_i = b_i \oplus b_{i+1}$

digits in G_n — digits in B_n

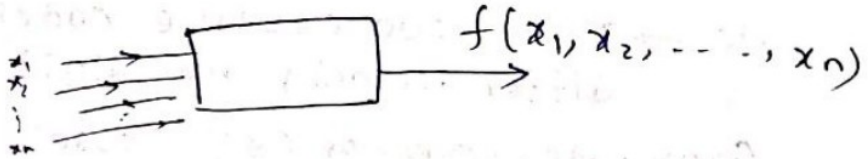
$b_5\ b_4\ b_3\ b_2\ b_1\ b_0$

$\oplus$ 0 1 0 1 1 0 — B

1 1 1 0 1

$g_4\ g_3\ g_2\ g_1\ g_0$

* Switching Algebra



Parameters to be considered

→ Time taken

→ Area (or) Hardware

→ Power.

* B $\{0, 1\}$, $\{+, \cdot, \bar{\ }\}$

OR AND Complement

~~* Basic Compleme~~

* Basic Properties

→ Idempotency

$x + x = x$

$x \cdot x = x$

→ Commutative

$x + y = y + x$

$x \cdot y = y \cdot x$

→ Associative

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

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

Scanned by CamScanner

→Distributive

Product (·) is distributive over Sum (+).

& Sum (+) is distributive over product (·)

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

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

x	y	z	$x+yz$	$(x+y)\cdot(x+z)$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

* Simplification Rules :

$x + 1 = 1 = 1 + x$

$x + 0 = x = 0 + x$

$x \cdot 1 = x = 1 \cdot x$

$x \cdot 0 = 0 = 0 \cdot x$

Absorption

$x + xy = x$

$x \cdot (x+y) = x$

$x + x' = 1$

$x \cdot x' = 0$

Consensus

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

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

Proof:- $xy + x'z + yz$

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

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

$= xy + x'z.$

Scanned by CamScanner

3. $(x+y)\cdot(x'+z)\cdot(y+z)$

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

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

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

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

* If f_1, f_2 are two switching expressions.

$f_2, f_1 = \{x_1, \cdots x_n, +, \cdot, '\}$

meaning is f_1, f_2 are func^ns of $x_1, \cdots x_n$ with operators $+, \cdot, '$

$f_1 < 0, 1$ $\quad f_2 < 0, 1$

then $f_1 \cdot f_2$, $f_1 + f_2$, f_1', f_2' are all switching expressions

*

x y z	f
0 0 0	1 → $x'y'z'$
0 0 1	0
0 1 0	0
0 1 1	0
1 0 0	1 → $xy'z'$
1 0 1	0
1 1 0	0
1 1 1	1 → xyz

$\Rightarrow f = x'y'z' + xy'z' + xyz$

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

$= y'z' + xyz$

Q. If $f_1 = x'y'z' + xy'z' + xyz$

$f_2 = xyz + y'z'$ → Sum of products expression.

$\Rightarrow f_1$ is equivalent to f_2

Check it.

&

* Canonical form of Switching function:

All the terms of the expression have all the literals in their normal form or complemented form.

Scanned by CamScanner

For any expression, there will be only one canonical form.

→ If a product term contains all the variables in normal or complemented form, it's called Minterm.

→ " Sum " " , "

" " " " "

Maxterm.

* Shanon's Expansion Theorem/Decomposition Theorem :-

$$f(x_1, x_2, \cdots x_n) = x_1 \cdot f(1, x_2, x_3 \cdots \cdots, x_n) + x_1' \cdot f(0, x_2, x_3 \cdots, x_n)$$

$$= (x_1 + f(0, x_2, x_3 \cdots, x_n)) \cdot (x_1' + f(1, x_2, x_3 \cdots x_n))$$

$$= x_1 x_2 \cdot f(1,1,x_3, x_4, \cdots x_n) + x_1 x_2' \cdot f(1,0,x_3 \cdots x_n) + x_1' x_2 \cdot f(0,1,x_3, \cdots, x_n) + x_1' x_2' \cdot f(0,0,x_3, \cdots x_n)$$

* De Morgan's Theorem for Complementation :-

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

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

* Dual :-

$$f(x_1, x_2, \cdots, x_n, +, \cdot, 0, 1)$$

$$= f_D(x_1, x_2, \cdots, x_n, \cdot, +, 1, 0)$$

↙ Dual of f

Scanned by CamScanner

* All switching expressions of a two variable function :-

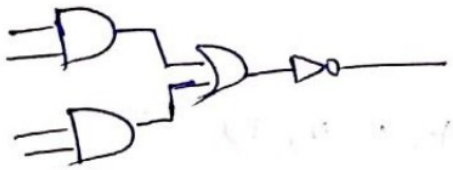
$$f(x,y) = a_0 x'y' + a_1 x'y + a_2 xy' + a_3 xy$$

a_i - 0/1

a_0	a_1	a_2	a_3	$f(x,y)$	Name
0	0	0	0	0	Inconsistency
0	0	0	1	xy	AND
0	0	1	0	xy'	—
0	0	1	1	$xy' + xy = x$	—
0	1	0	0	$x'y$	—
0	1	0	1	y	—
0	1	1	0	$x'y + xy'$	XOR
0	1	1	1	$x + y$	OR
1	0	0	0	$x'y'$	NOR
1	0	0	1	$x'y' + xy$	(XNOR) equivalence
1	0	1	0	y'	—
1	0	1	1	$x + y'$	$y \rightarrow x$ (Implication)
1	1	0	0	x'	NOT
1	1	0	1	$x' + y$	$x \rightarrow y$ (Implication)
1	1	1	0	$x' + y'$	NAND
1	1	1	1	1	Tautology.

* AND OR → Inputs gates 2

AOI2



Scanned by CamScanner

* Functionally Complete set of operations :-

Canonical form of Switching expressions / SOP / POS

$f(x,y,z) = x + y'z = x \cdot 1 \cdot 1 + y'z \cdot 1$

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

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

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

↓ canonical SOP

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

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

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

↓ canonical POS.

$SOP = \Sigma(0,5,6,7) = x'y'z' + xy'z + xyz' + xyz$

↙ Minterms

↙

using AND(·), OR(+), NOT(') , we can represent all the switching expressions

=> Functionally complete set of operations

↙

Def:- A set of operations is functionally complete if any switching expression (SOP/POS) can be represented & realized by ~~use~~ these operations.

$\{\cdot, +, '\}$ (or) $\{\cdot, '\}$ (or) $\{+, '\}$

since $(A+B)' = A' \cdot B'$

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

* NAND is a functionally complete

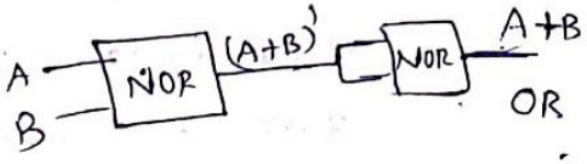
NOR " " "

Scanned by CamScanner

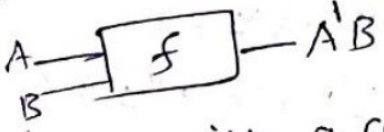
Q. Show that NOR is functionally complete

& $(A+B)'$

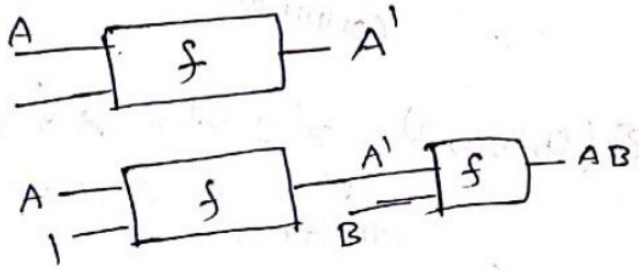
$(A+A)' = A'$



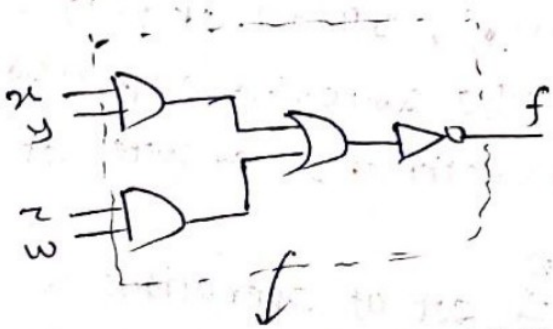
Check ~~Show~~ $f(A'B)$ is functionally complete.



$f(A'B)$ together with a constant state '1' is functionally comple.



A OII →



Q. Check whether the ~~above~~ given switching expression is functionally complete or not.

$f(x,y,z) = xyz + x'yz + x'yz' + x'y'z + x'y'z$

$f = \Sigma(0,1,2,3,7)$

$f' = \Sigma(4,5,6) = xy'z' + xy'z + xyz'$

Scanned by CamScanner

* Analogy b/w Switching Algebra & Boolean algebra:-

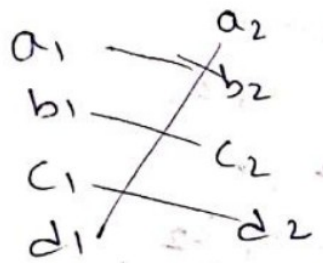
Set of objects/variables = $\{a_1, b_1, c_1, d_1\}$ ---(1)

Set of operations = $\{f_1, f_2, f_3\}$

Set of variables = $\{a_2, b_2, c_2, d_2\}$ ---(2)

Set of operations = $\{f_1', f_2', f_3'\}$

→ Isomorphic Algebraic systems



→ Switching Algebra is 2-valued Boolean Algebra.

* Boolean Algebra:-

Boolean algebra is a distributive, complemented lattice.

distributive → distributive law

complemented → complementation.

lattice → commutative, associative, idempotent.

→ A Boolean Algebra 'B' is a set of variables $\{a, b, c \cdots\}$ and a set of operations $\{+, \cdot\}$ that satisfy the properties of idempotent, commutative, absorption, and associative and mutually distributive.

→ There are two bounds 0 (least) and 1 (greatest).

Scanned by CamScanner

→ It has a unary operation of complementation

* Complementation is Unique

$a + a' = 1$
$a \cdot a' = 0$ — a' is the complement of 'a'.

Proof:- Let there are two complements of a and they are b_1 & b_2.

$\Rightarrow a + b_1 = 1$ $\quad a + b_2 = 1$

$a \cdot b_1 = 0$ $\quad a \cdot b_2 = 0$

$b_1 = b_1 \cdot 1$

$b_1 = b_1 \cdot (a + b_2)$

$b_1 = b_1 \cdot a + b_1 \cdot b_2$

$b_1 = b_1 \cdot b_2$

$b_2 = b_2 \cdot 1$

$= b_2 \cdot (a + b_1)$

$= b_1 \cdot b_2$

$\Rightarrow b_1 = b_2 = b_1 \cdot b_2$

$\Rightarrow$ Complement is Unique.

* Show that B·A satisfies DeMorgan's rule.

Proof: $(a+b)(a' \cdot b') = 0$

~~$a \cdot a' + a \cdot b' + b \cdot a' + b b' = 0$~~

$a \cdot a' \cdot b' + b \cdot a' \cdot b' = 0$

$0 + 0 = 0$

$0 = 0$ ✓

$a + b + a' b' = 1$

$(a+b) + a' \cdot b' = 1$

$(a+b+a') \cdot (a+b+b') = 1$

$1 \cdot 1 = 1$

$1 = 1$ ✓

$(a \cdot b) \cdot (a' + b') = 0$

$a b a' + a b b' = 0$

$0 = 0$ ✓

$(a \cdot b) + (a' + b') = 1$

$(a' + b + a) \cdot (a' + b + b) = 1$

$1 \cdot 1 = 1$

$1 = 1$ ✓

Scanned by CamScanner

+	0	1	a	a' = b
0	0	1	a	b
1	1	1	1	1
a	a	1	a	1
a' = b	b	1	1	b

•	0	1	a	a' = b
0	0	0	0	0
1	0	1	a	b
a	0	a	a	0
a' = b	0	b	0	b

*Switching Algebra a is 2-valued Boolean Algebra.

* XOR = EXOR = $x \oplus y = xy' + x'y$.

* $\oplus, \{A, B, C\}$

If $A \oplus B = C$, then $A \oplus C = B$

Proof: $A \oplus C \Rightarrow$

$= AC' + A'C'$

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

$= A \cdot (A' + B) \cdot (A + B') + A'B$

$= A \cdot (A'B' + AB) + A'B$

$= AB + A'B$

$= B$

Commutative
$A \oplus B = B \oplus A$

Associative
$A \oplus (B \oplus C) = (A \oplus B) \oplus C$.

Distributive $\{\cdot, \oplus\}$
$A \cdot (B \oplus C) = A \cdot B \oplus A \cdot C$

check for $\{+, \oplus\}$.

Kohavi

3.3(d) Simplify

$(a + a'b) + a'b'c + a'b'c'd + \cdots$

$= (a + a') \cdot (a + b) + a'b'c + \cdots$

$= (a + b) + (a + b)'c + a'b'c'd + \cdots$

$= a + b + c + a'b'c'd + \cdots$

$= a + b + c + d + \cdots$

Scanned by CamScanner

3.4(b) Find by inspection the complement of the following and simplify.

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

& $[(x+y'z').(y+x'z').(z+x'y')]'$

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

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

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

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

$= x \oplus y + x \oplus z + y \oplus z$

3.5(b) Validate

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

& LHS

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

~~$= xy.(z+z') + x'y'.(z+z') + xyz$~~

~~$= xyz + xyz' + x'y'z + x'y'z' + xyz$~~

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

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

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

RHS

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

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

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

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

$= yz + xy + x'y'$

Scanned by CamScanner

~~3.6 If $AB' + A'C$~~.

3.7 Find A, B, C, D by ~~sorting~~ solving the following equations.

$$A' + AB = 0$$
$$AB = AC$$
$$AB + AC' + CD = C'D.$$

$A' + AB = 0$

$\Rightarrow A' + B = 0$

$A' = 0, \quad B = 0 \Rightarrow \boxed{A = 1}, \quad \boxed{B = 0}$

$AB = AC \Rightarrow 0 = C \Rightarrow \boxed{C = 0}$

$0 + 1 + 0 = D$

$\boxed{D = 1}$

3.20 A safe has five locks v, w, x, y, z all of which must be unlocked for the safe to open. The keys to the locks are distributed among five executives in the following way

A has key for locks v & x
B " " v & y
C " " w & y
D " " x & z
E " " v & z

(a) Determine number of executives required to open the safe.

(b) Who is "essential executive" without whom the safe cannot be opened?

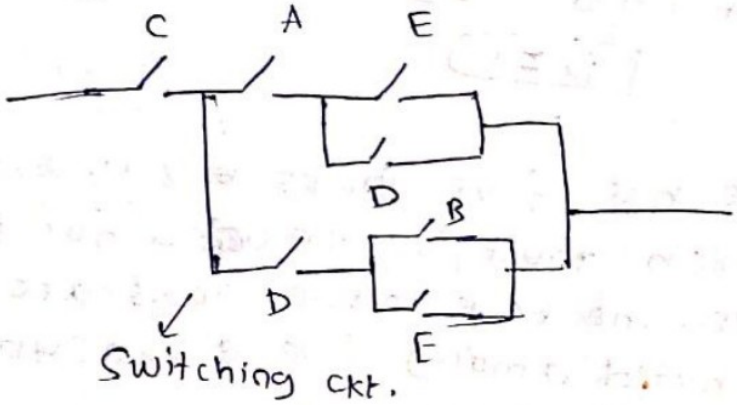
(c) Find all the combinations of executives that can open the safe.

Scanned by CamScanner

Executives \ Keys	V	W	X	Y	Z
A	✓		✓		
B	✓			✓	
C (Essential)		✓		✓	
D			✓		✓
E	✓				✓

(C) CAD + CAE + BCD + CDE

OR



Switching ckt.

* Algebraic properties used to Minimize the s/w expression

→ Minimal Expression
→ Irredundant expression

1. Min. no. of literals in the expression.
2. Min. no. of terms

* $f = \Sigma(2,0,4,3,7,5)$

Truth table

Input x	y	z	output
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

$f(x,y,z) = x'yz' + x'y'z' + xy'z' + x'yz + xyz + x'y'z$

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

$= x'z' + xy' + yz$ – Irredundant & Minimal

Scanned by CamScanner

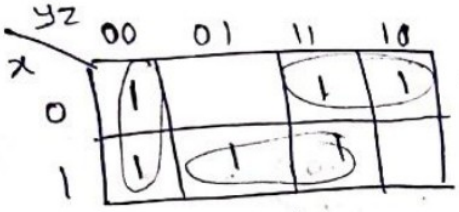
(01) $x'yz' + x'y'z' + xy'z' + x'yz + xyz + xy'z$

$= x'y + y'z' + xz$ → Irredundant & Minimal also

* Karnaugh Map method of minimization

$Aa + Aa' = A$

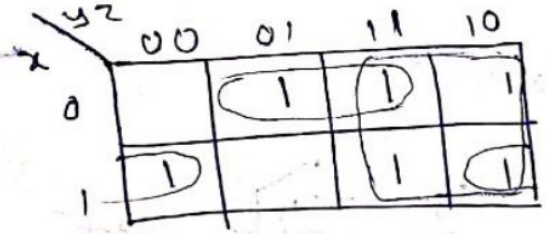
$f = \Sigma(2, 0, 4, 3, 7, 5)$



→ $y'z' + xy + xz$

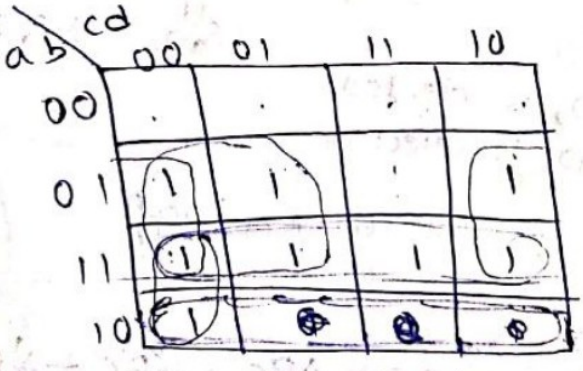
Cube 2^m adjacent cells block is called cube. $m = 1, 2, 3, \ldots$

If f is an n-variable func^n and we can identify a 2^m cube then the minimized term has $(n-m)$ literals.



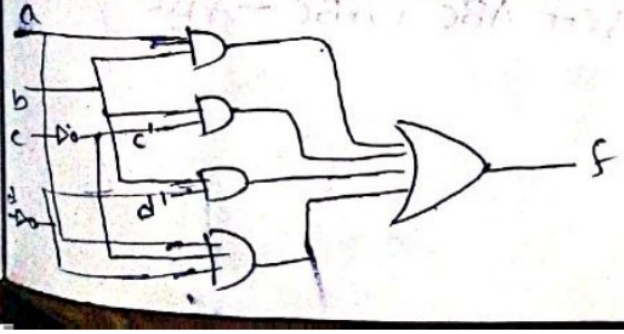
→ $y + x'z + xz'$

$f = \Sigma(4, 5, 6, 8, 12, 13, 14, 15)$



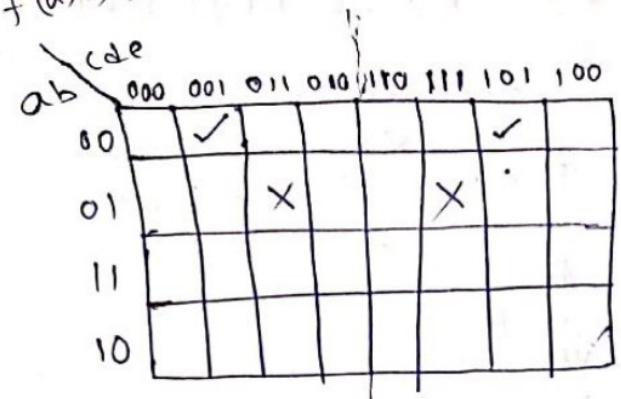
→ $ab + bc' + bd' + ac'd'$

AND-OR expression



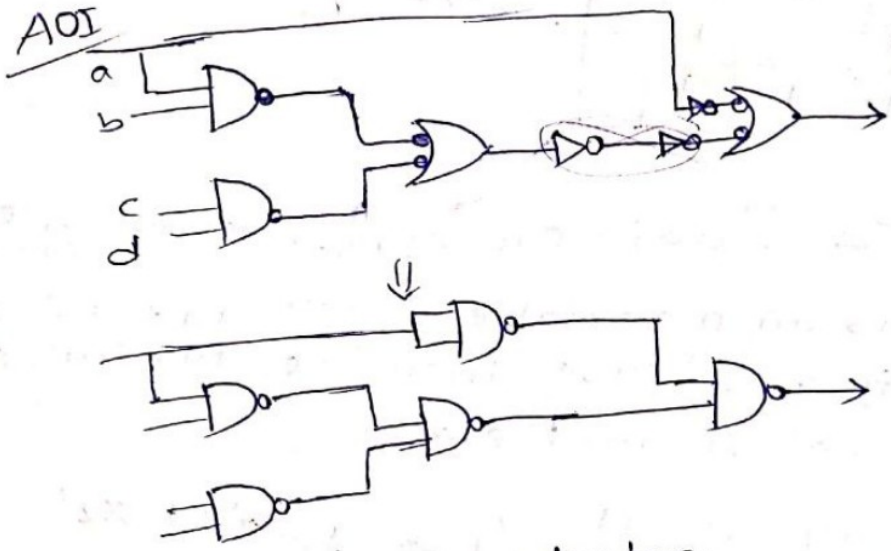
Scanned by CamScanner

* $f(a,b,c,d,e)$

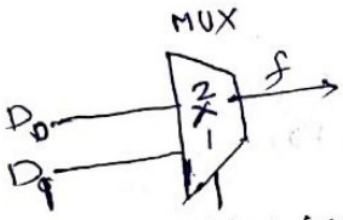


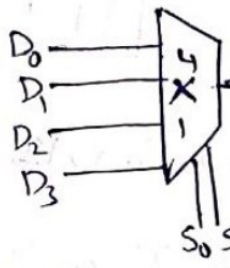
also
✓ - adjacent
× - adjacent

* AOI



* Multiplexer – Data selector.





$\Rightarrow S_0'S_1'D_0 + S_0'S_1D_1 + S_0S_1'D_2 + S_0S_1D_3$

$S'D_0 + SD_1$ ⟸ $S=0$, D_0 is selected
$S=1$, D_1 is selected

n – Inputs => Select line → $\log_2^n$

* Adder

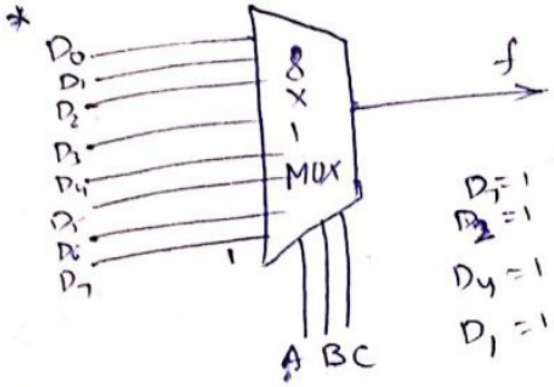
$S = A \oplus B \oplus C = ABC + AB'C' + A'B'C + A'BC'$

Scanned by CamScanner

* 4×1 MUX

$= S_0' S_1' D_0 + S_0' S_1 D_1 + S_0 S_1' D_2 + S_0 S_1 D_3$.

*



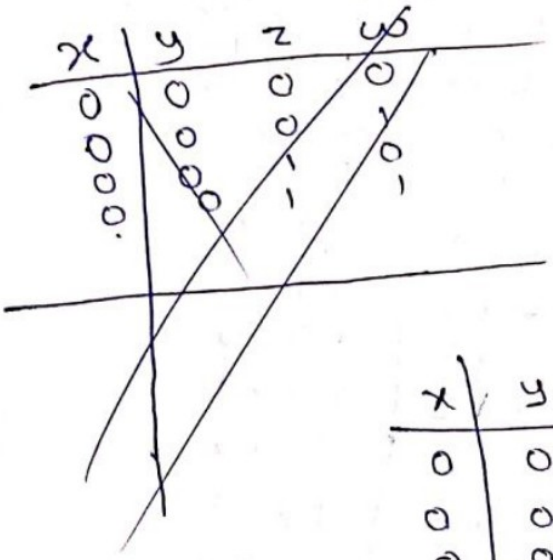
$D_7 = 1$
$D_2 = 1$
$D_4 = 1$
$D_1 = 1$

$f = ABC + A'BC' + AB'C' + AB'C$

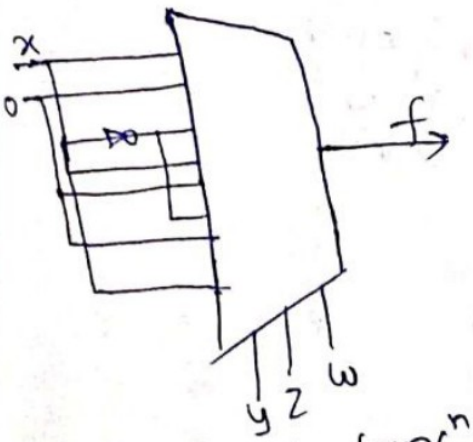
→ Here A, B, C are inputs and selection inputs

* For n-variable func^n to be realized we need 2^n × 1 MUX since the n-variables will be in n-select lines.

→ $f(x,y,z,w) = x'y'zw' + x'yz'w' + xy'z'w' + xy'zw + xyzw$



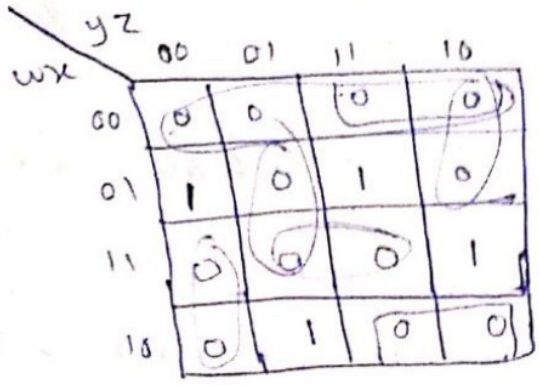
x	y	z	w	f
0	0	0	0	0 = x
0	0	0	1	0 = 0
0	0	1	0	1 = x'
0	0	1	1	0 = x
0	1	0	0	0 = 0
0	1	0	1	1 = x'
0	1	1	0	0 = 0
0	1	1	1	0 = x
1	0	0	0	1 = x
1	0	0	1	0 = 0
1	0	1	0	0 = x'
1	0	1	1	1 = x
1	1	0	0	0 = 0
1	1	0	1	0 = x'
1	1	1	0	0 = x
1	1	1	1	1 = x



n-Variable func^n with n-1 select lines

Scanned by CamScanner

* Minimization with product of sum Expression.



$f = w'xy'z + w'xyz + wxyz' + wx'y'z$

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

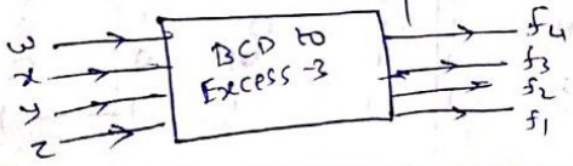
* Ex:

BCD to excess-3 converter:-

Decimal	BCD $w\ x\ y\ z$	Excess-3 $f_4\ f_3\ f_2\ f_1$
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
10	1 0 1 0	
11	1 0 1 1	
12	1 1 0 0	
13	1 1 0 1	
14	1 1 1 0	
15	1 1 1 1	

Input Not possible

UNDEFINED

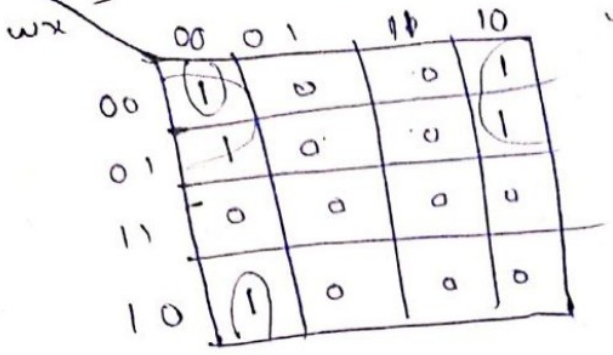


Scanned by CamScanner

K-Maps for f_1

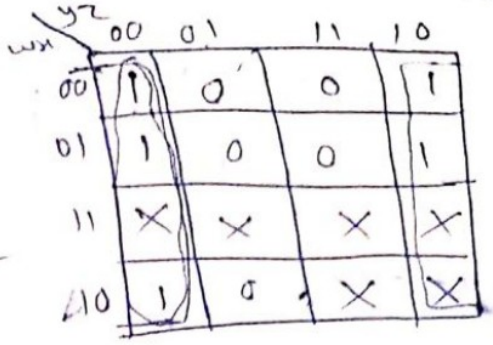
10, 11, 12, 13, 14, 15 → Don't care

without Don't care



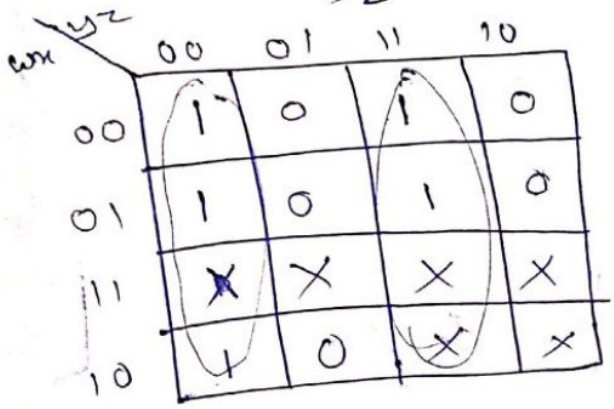
$f_1 = w'z' + x'y'z'$

with Don't care



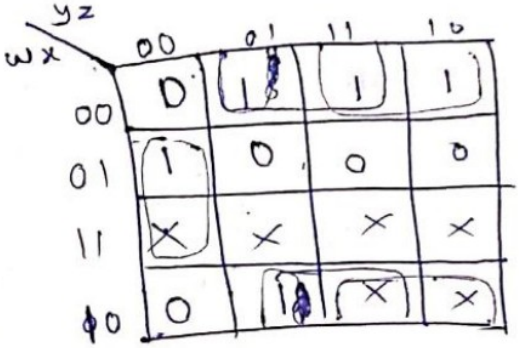
$f_1 = z'$

f_2



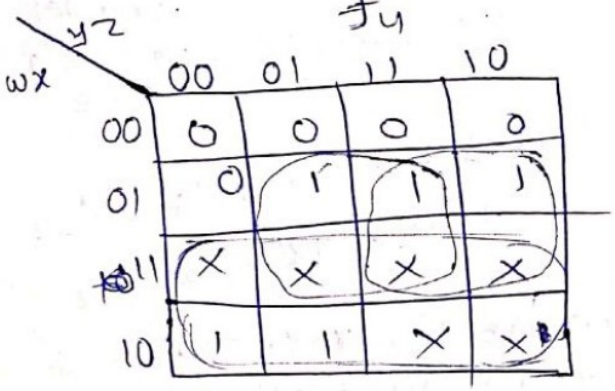
$f_2 = y'z' + yz$

f_3



$f_3 = x'z + x'y + xy'z'$

f_4



$f_4 = w + xz + xy$

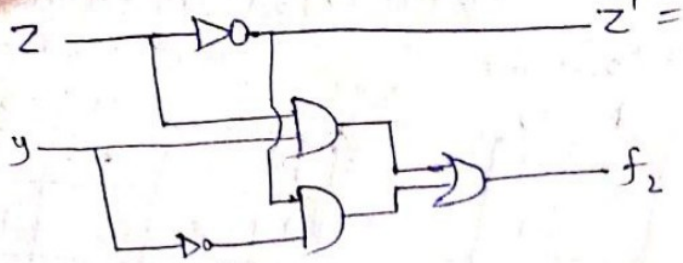
$f_1 = z'$

$f_2 = y'z' + yz$

$f_3 = x'z + x'y + xy'z'$

$f_4 = w + xy + xz$

Scanned by CamScanner



~~* Map-entered variable~~

* K-Map with Map-entered variable:-

wx \ yz	00	01	11	10
00	1	A	A	0
01	0	1	C	0
11	A'	1	1	0
10	B	1	B'	1

f(w, x, y, z, A, B, C)

↓ to solve

First put all the variables '0'

consider A & A' to be different variables.

i.e Both can be put '0' at the same time.

$$\Rightarrow f = wy'z + w'x'y'z' + w'x'yz + wx'yz' + wxyz$$

Now, change the terms formed by wx, y, z i.e previous 1's to don't care.

& put A as 1 & 0 ther variables (A', B, B', C) = 0

wx \ yz	00	01	11	10
00	X	1	1	0
01	0	X	0	0
11	0	X	X	0
10	0	X	0	X

$$f = A \cdot w'x'z$$

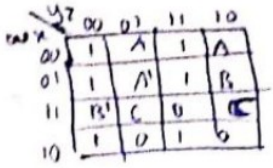
Scanned by CamScanner

Now put B as 1 & other 0 → $f = B$.

" B' as 1 " 0

" C as 1 & " 0

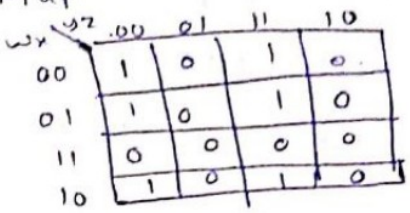
" A' as 1 " 0



K-Map Map entered variables:

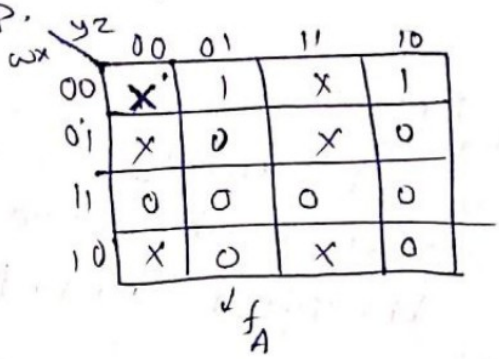
step 1: Put '0' in all the cells containing map-entered variables. Find the minimal expression using K-Map

~~Find~~ f'_{wxyz}



step 2: (How f depends on A)
Make all other map entered variables '0'.
Make all 1's as Don't cares. (don't touch the 0's)

Then get a minimal expression as a product of that particular map-entered variable (Here 'A') and expression we get from K-Map.



f_A

$A \cdot f_A$

step 3: Repeat step 2 for A', B, B', C, - - -

$$\therefore f = f_{wxyz} + A \cdot f_A + A' \cdot f_{A'} + B \cdot f_B + B' \cdot f_{B'} + C \cdot f_C$$

Scanned by CamScanner

* Quine McClansky Method :

Switching or Boolean expression

Minimal expression

Irredundant expression
↳ can't be reduced further

Prime Implicants :

Let $f(x_1, x_2, \ldots x_n)$ and $g(x_1, x_2, \ldots x_n)$ are two switching func^n^s

Now, f covers g if f contains a '1' everywhere where g does (in K-map or minterms); $f \geq g$

If f covers g & g covers f, then f & g are equivalent.

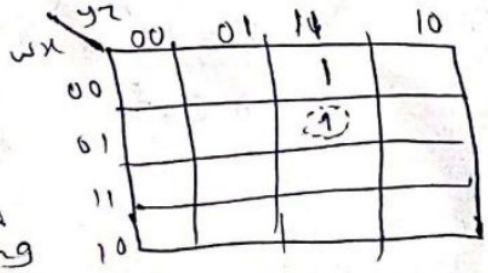
Let $h(x_1, x_2, \ldots x_n)$ be a single product term

If f covers h, then it is said that $h \rightarrow f$,
h is one implicant of 'f'.

Prime Implicant

A prime implicant 'P' is a product term that implies the function f and dropping any literal from P produces another product term that doesn't imply f.

Ex) let's say $P = w'x'yz$

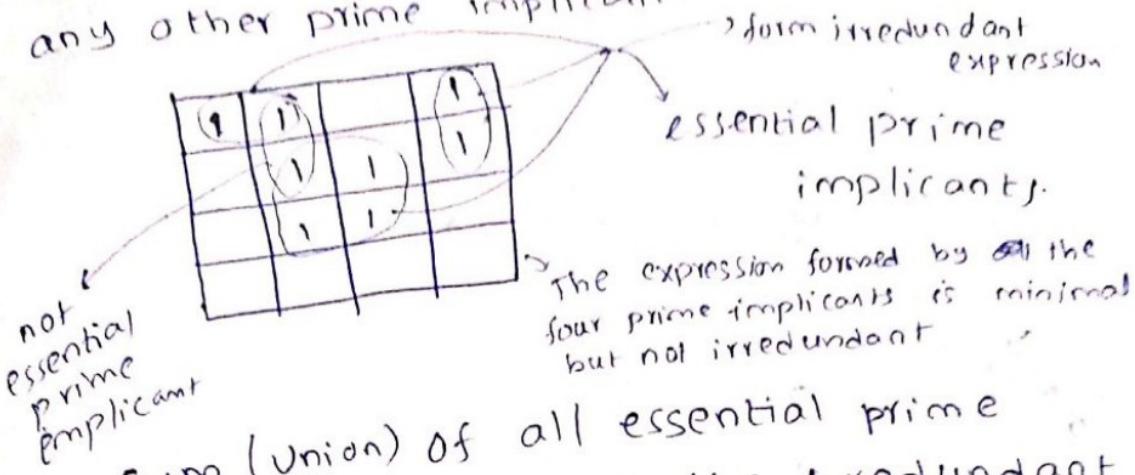


By making
$P_1 = w'yz$
we are adding $w'xyz$ to P. But that '1' may not be there in f to cover P_1.

Scanned by CamScanner

Essential Prime Implicants:

A prime implicant is essential if it covers a cell(min-term) which is not covered by any other prime implicant



Sum (Union) of all essential prime implicants is one of the irredundant expression.

Theorem :-

An irredundant Boolean expression is the sum (union) of all essential prime implicants.

Proof :- Let f_1 be an irredundant expression which has an essential prime implicant P.

~~Assume If I drop one literal from P, then~~

Assume the expression is not irredundat i.e it can be reduced

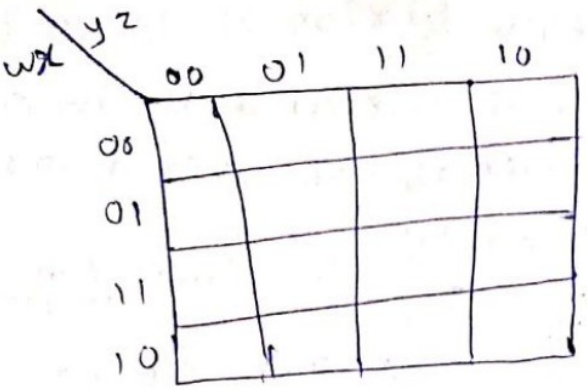
⇒ It leads to dropping a variable from P. & still covers f_1

But it is a contradiction. since P is prime implicant

Quine McClansky Method:

$$f = \sum m(0, 1, 2, 5, 7, 8, 9, 10, 13, 15)$$

Scanned by CamScanner



Step1 : $f = \sum m(0, 1, 2, 5, 7, 8, 9, 10, 13, 15)$

No. of 1's in minterm	Group No.	minterm no. (m)	~~x~~ ~~y~~ w x y z
0	1	0	0 0 0 0
1	2	1	0 0 0 1
		2	0 0 1 0
		8	1 0 0 0
2	3	5	0 1 0 1
		9	~~0 0 0 0~~ 1 0 0 1
		10	1 0 1 0
3	4	7	0 1 1 1
		13	1 1 0 1
4	5	15	~~1 1 1 1~~ 1 1 1 1

(4 is not there in f)

Step2 :-

Adjacent two groups can be combined

m	step2 (combination)
0, 1	0 0 0 – ✓
0, 2	0 0 – 0 ⊗
0, 8	– 0 0 0 *
1, 5	0 – 0 1 ×
1, 9	– 0 0 1 *
2, 10	– 0 1 0 *
8, 9	1 0 0 – ✓
8, 10	1 0 – 0 (×)

Scanned by CamScanner

5, 7	01_1 α
5, 13	-101
9, 13	1_01 x
7, 15	-111
13, 15	11-1

Step 3:-

m	wxyz	
0, 1, 8, 9	-00-	A = x'y'
0, 2, 8, 10	-0-0	B = x'z'
1, 5, 9, 13	--01	C = y'z
5, 7, 13, 15	-1-1	D = xz

⇒ No more minimization is possible.

$$f = A + B + C + D = x'y' + x'z' + y'z + xz$$

Method 2 (Minimization by Decimal Representation) :-

1. Difference of two minterms with different indices is power of 2.
2. If the minterm with smaller index (index = no. of 1's) is larger than the minterm with larger index, then it cannot be combined even if the difference b/w minterms is a power of 2.
3. Minterms with same indices are not considered.

Scanned by CamScanner

$f(v,w,x,y,z) = \sum(13,15,17,18,19,20,21,23,25,27,29,31) + \sum_d(1,2,12,24)$

↓ Don't care conditions

Index	Minterms	variables v w x y z	Groups of difference (2^m)
1	1	0 0 0 0 1	1,17 (16) – A
	2		2,18 (16) – B
2	12		12,13 (1) – C
	17		17,19 (2) ✓
	18		17,21 (4) ✓
	20		17,25 (8) ✓
	24		18,19 (1) – D
		24,25 (1) – F	20,21 (1) – E
3	13		13,15 (2) ✓
	19		13,29 (16) ✓
	21		19,23 (4) ✓
	25		19,27 (8) ✓
			21,23 (2) ✓
			21,29 (8) ✓
			25,27 (2) ✓
			25,29 (4) ✓
4	15		15,31 (16) ✓
	23		23,31 (8)
	27		27,31 (4) ✓
	29		29,31 (2) ✓
5	31		

→ In 1st & 2nd groups no 16 is there i.e no matching difference

So A, B are prime implicants

17, 19, 13, 15 are not taken because low index but large number

17,19,21,23 (2,4) ✓

17,19,25,27 (2,8) ✓

17,21,25,29 (4,8) ✓

13,15,29,31 (2,16) — G

19,23,27,31 (4,8) ✓

21,23,29,31 (2,8) ✓

25,27,29,31 (2,4) ✓

Scanned by CamScanner

17, 19, 21, 23, 25, 27, 29, 31 (2, 4, 8) → H

~~17, 19, 25, 27~~

→ 1, 17 (16) – A
2, 18 (16) – B
12, 13 (1) – C
18, 19 (1) – D
20, 21 (1) – E
24, 25 (1) – F
13, 15, 29, 31 (2, 16) – G
17, 19, 21, 23, 25, 27, 29, 31 (2, 4, 8) – H

* To find "essential" prime implicants. Don't care are not included

	13 ✓	15 ✓	17 ✓	18	19 ✓	20 ✓	21 ✓	23 ✓	25 ✓	27 ✓	29 ✓	31 ✓
vwxyz (16 8 4 2 1) A			X									
w'x'y'z = A				X								
wx'yz' = B												
v'wxy' = C	X											
vw'x'y' = D				X	X							✓
vw'xy' = E						X	X					
vwx'y' = F									X			
wx z = G	X	X								X	X	✓
v z = H			X		X		X	X	X	X	X	✓

G, H, E, B (or) G, H, E, D

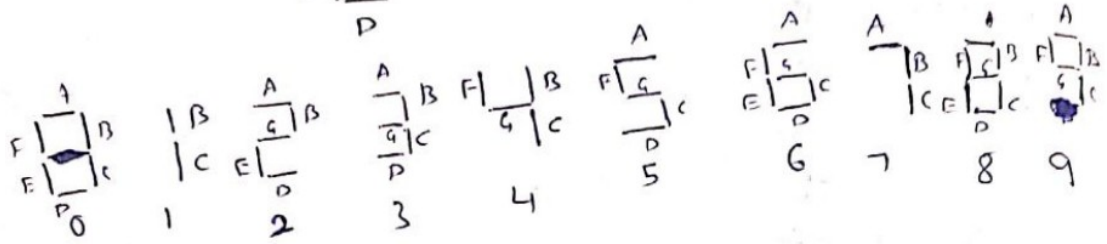
∴ $f = vz + wxz + vw'xy' + w'x'yz'$

(or)

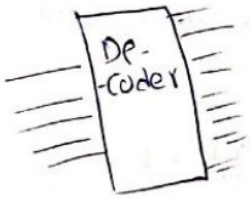
$= vz + wxz + vw'xy' + vw'x'y$

Scanned by CamScanner

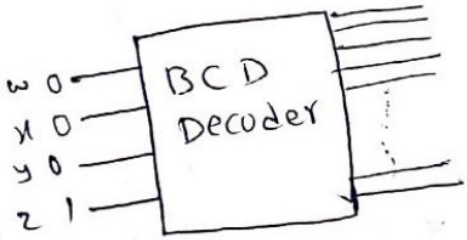
* 7-segement Display:

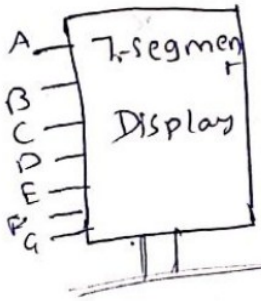


BCD to 7-segment Display



→ n to 2^n





Decimal	BCD w x y z	f_1 A	f_2 B	f_3 C	f_4 D	f_5 E	f_6 F	f_7 G
0	0000	1	1	1	1	1	1	0
1	0001	0	1	1	0	0	0	0
2	0010	1	1	0	1	1	0	1
3	0011	1	1	1	1	0	0	1
4	0100	0	1	1	0	0	1	1
5	0101	1	0	1	1	0	1	1
6	0110	1	0	1	1	1	1	1
7	0111	1	1	1	0	0	0	0
8	1000	1	1	1	1	1	1	1
9	1001	1	1	1	0	0	1	1

10, 11, 12, 13, 14, 15 → Don't care

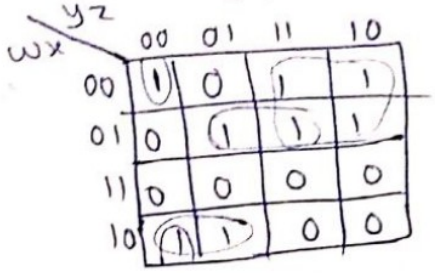
Scanned by CamScanner

f_1

$f_1 = y + w + x'z' + xz$

don't cares not taken since if inputs 29 are given, it may behave erratically

f_1



$f_1 = x'y'z' + w'y + w'xz + wx'y'$

* Method of minimization :

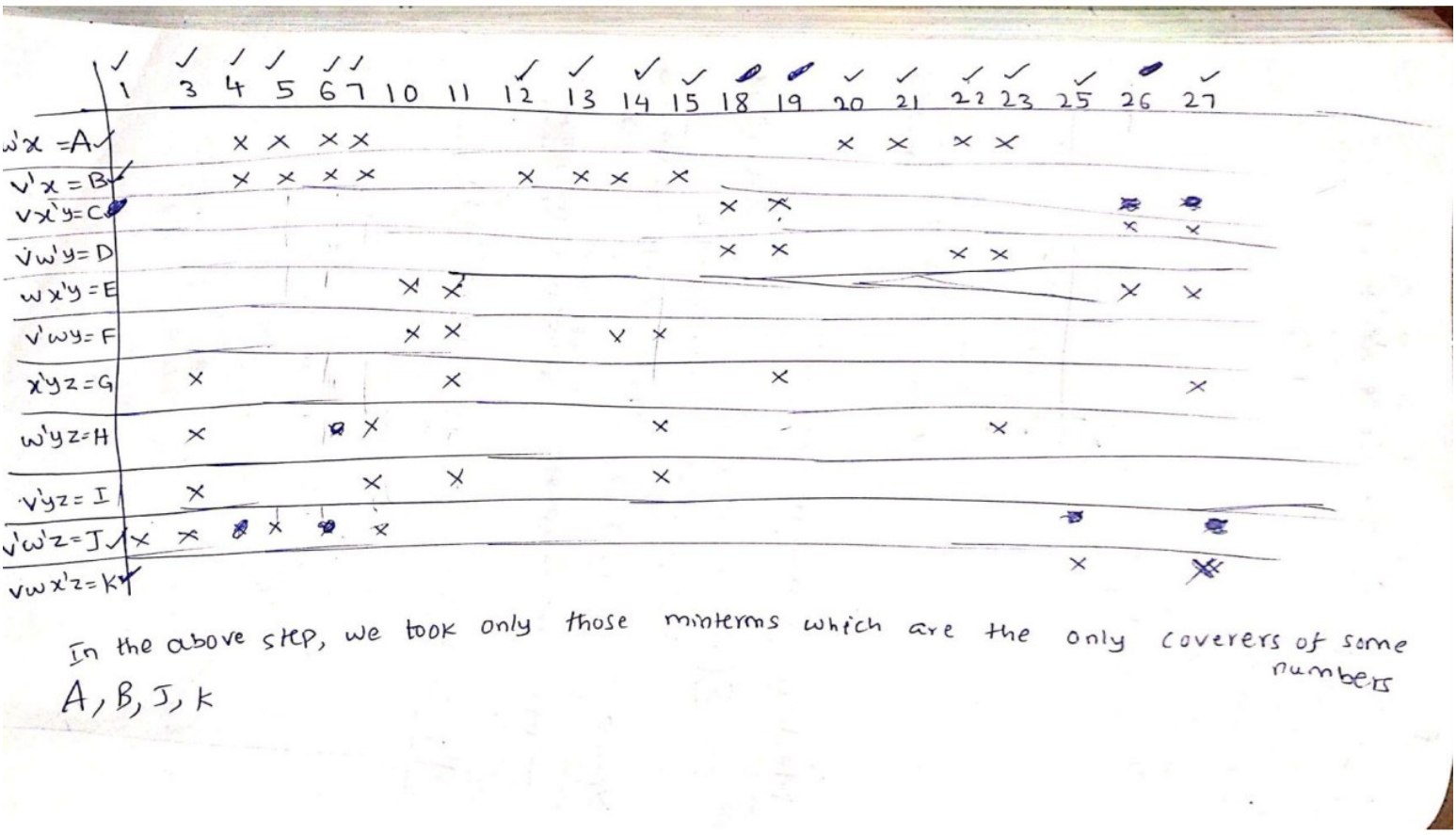
$f = \sum(1, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 18, 19, 20, 21, 22, 23, 25, 26, 27)$.

Minimal expression — First, simply & find out minterms using method 2.

After that, the following table

	1	3	4	5	6	7	10	11	12	13	14	15	18	19

Scanned by CamScanner



In the above step, we took only those minterms which are the only coverers of some numbers

A, B, J, K

Scanned by CamScanner

step 2:

	10	11	18 ✓	19 ✓	26 ✓
✓C			×	×	×
			×	×	
× → D					×
✓E	×	×			
× → F	×	×			
✓G		×		×	
× → H		×			
× → I					

D, F, H, I are dominated prime implicants i.e all their minterms are covered by some other prime implicants.

= ~~(They are subsets of other prime implicants)~~

	10 ✓	11 ✓	18 ✓	19 ✓	26 ✓
✓C			×	×	×
✓E	×	×			×
G		×		×	

→ A, B, ~~E~~, J, K, C, E

we get irredundant form but we lose some minimal terms due to this step

$f = w'x + v'x + v'w'z + vwxz + vx'y + wx'y$

*Ex $f(v,w,x,y,z) = \sum(0,1,3,4,7,13,15,19,20,22,23,29,31)$.

£ To not lose minterms, follow ↓

Scanned by CamScanner

	0	1	3 ✓	4	7 ✓	13 ✓	15 ✓	19 ✓	20	22	23 ✓	29 ✓	31 ✓
A ✓						×	×					×	×
dominated × → B					×		×				×		×
C ✓			×		×			×			×		
D										×	×		
E									×	×			
F				×					×				
G		×	×										
H	×			×									
I	×	×											

	0	1	4	20	22
D					×
E				×	×
F			×	×	
G		×			
H	×		×		
I	×	×			

two-valued variable which tells which of the above terms are included

$$P = (H+I)\cdot(G+I)\cdot(F+H)\cdot(E+F)\cdot(D+E)$$

(0, 1, 4, 20, 22)

~~$= (GH+I)$~~

$= (GH+I)\cdot(F+H)\cdot(DF+E)$

$= (FI+HI+IFGH+GH)\cdot(E+DF)$

$= (FI+HI+GH)\cdot(E+DF)$

$= EFI + EHI + EGH + DFI + DFHI + DFGH$

Scanned by CamScanner

∴ function

$f = A + C + (E+F+I)$

(or)

$= A + C + (E+H+I)$

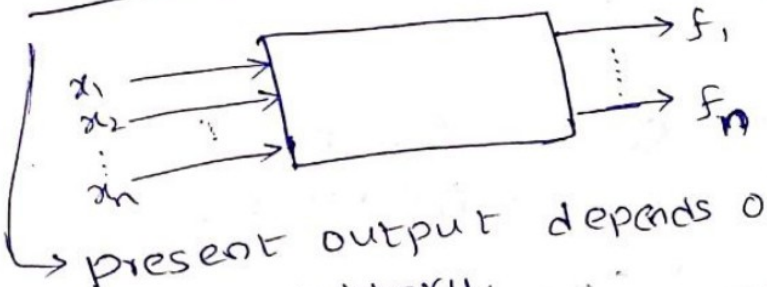
$= A + C + (E+G+H)$ (or)

$= A + C + (D+F+I)$ (or)

$= A + C + (D+F+G+H)$ ✗

(since it has four terms)

* Sequential circuit Design :-



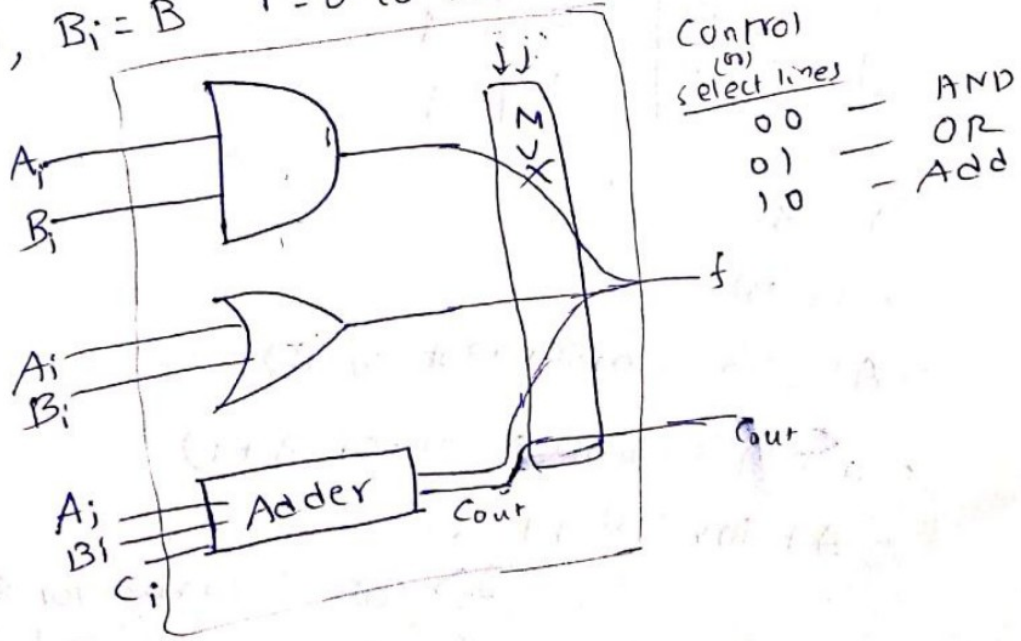
→ present output depends on present input & past history.

* Arithmetic Logic Unit (ALU) :-

+, −, *, /, <, >, AND, OR, ...

Design of 1-bit ALU :-

$A_i = A$, $B_i = B$ $i = 0$ to 15 [∵ 16 Bit]

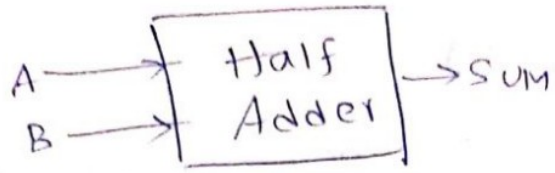


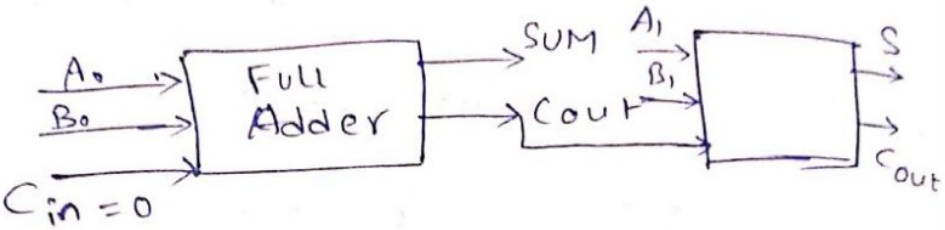
Scanned by CamScanner

A	B	SUM
0	0	0
0	1	1
1	0	1
1	1	10

A	B	carry out	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$= A \oplus B$





For Full adder, $Sum = A \oplus B \oplus C$

$Cout = AB + BC + CA$

A	B	C	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$A - B$

$= A + (-B)$

$= A +$ (2's complement of B)

$= A +$ (1's complement of $B + 1$)

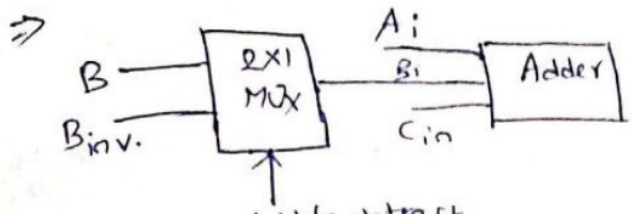
$= A +$ inv.$(B) + 1$

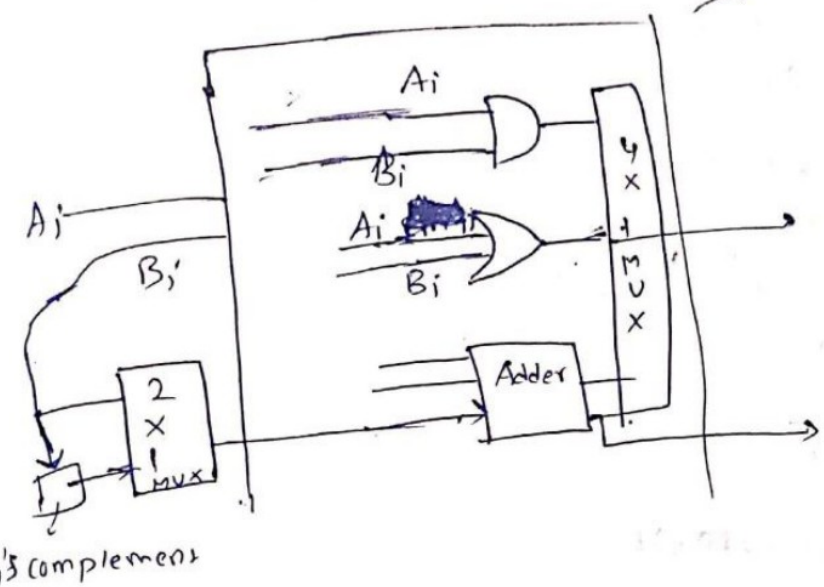
$\Rightarrow$ initial carry for subtraction $= 1$

Scanned by CamScanner

Scanned by CamScanner

⇒

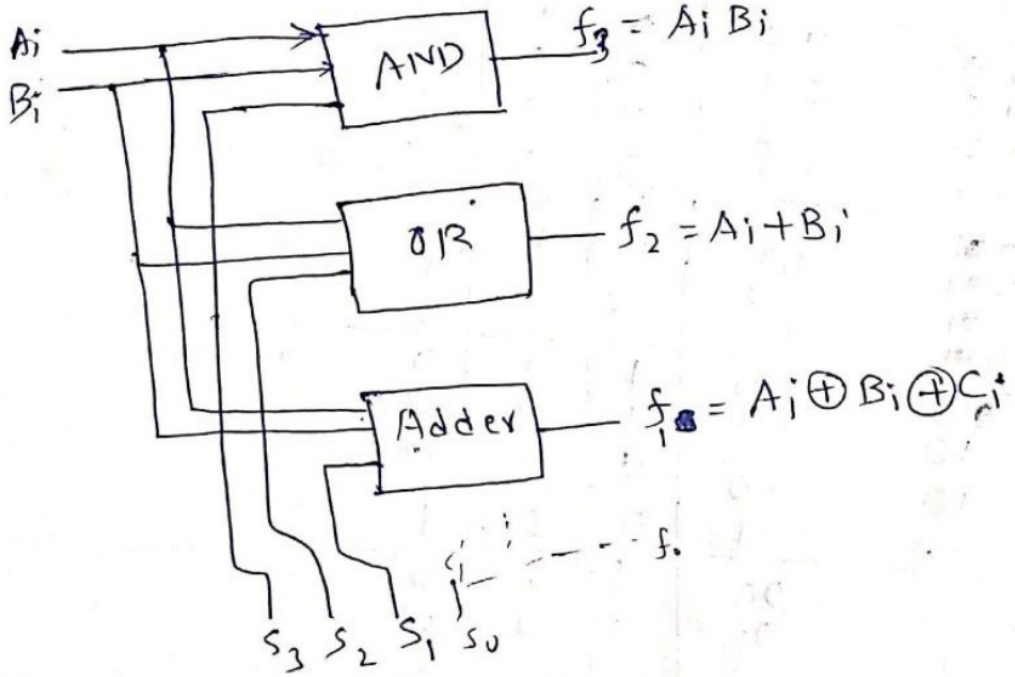




→ Disadvantage: All the AND, OR, Adder are computed. We select only one out of them.

→ Combinational ckt. → very fast

→ Memory access → always time consuming



$f = S_3 \cdot f_3 + S_2 f_2 + S_1 f_1 + S_0 f_0$

↓ (or)

~~$S_0' S_0 f_0 + S_0' S_1' f_1 + S_0 S_1' f_0$~~

$S_0' S_1' f_3 + S$

* Comparator

<, =, >

SSI → AND, OR, NOT, NAND, NOR, ... chip < 10 (less than 10 gates in a chip)

MSI → <100

LSI → few 1000

VLSI → > 10000

→ i^{th} stage 1-bit ALU

x_i, y_i

$f_2 = x_i y_i + x_i' y_i'$ (To say if $x_i == y_i$)

$(x_i > y_i)$ $f_1 = x_i y_i'$

$(x_i < y_i)$ $f_3 = x_i' y_i$

→ 2-bit comparator

x: $x_1 x_2$ y: $y_1 y_2$

$x_1 x_2$	$y_1 y_2$	f_1 (>)	f_2 (=)	f_3 (<)
00	00	0	1	0
00	01	0	0	1
00	10	0	0	1
00	11	0	0	1
01	00	1	0	0
01	01	0	1	0
01	10	0	0	1
01	11	0	0	1
10	00	1	0	0
10	01	1	0	0
10	10	0	1	0
10	11	0	0	1
11	00	1	0	0
11	01	1	0	0
11	10	1	0	0
11	11	0	1	0

Scanned by CamScanner

x_1x_2 \ y_1y_2	00	01	11	10
00	2	3	3	2
01	1	2	3	3
11	1	1	2	1
10	1	1	3	2

for $f_2 = 1$

$$f_1 = x_1 y_1' + x_1 x_2 y_2' + x_2 y_1' y_2' = x_1 y_1' + x_2 y_2' (x_1 + y_1')$$

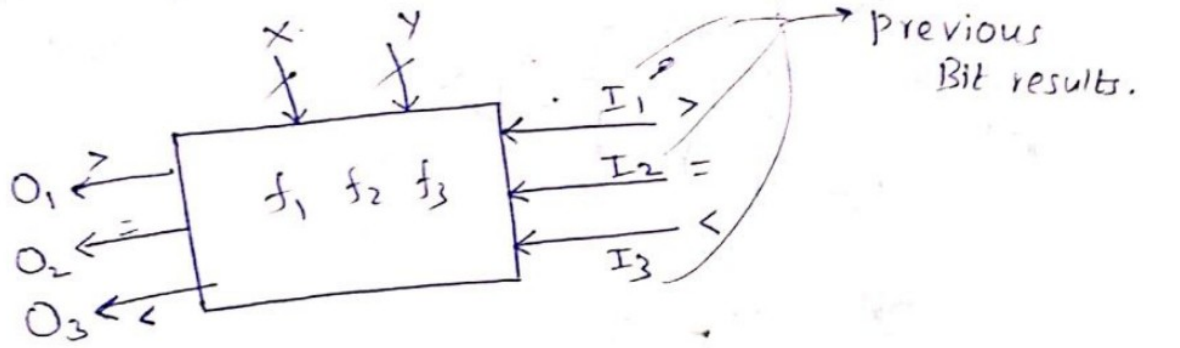
$$f_2 = x_1' x_2' y_1' y_2' + x_1' x_2 y_1' y_2 + x_1 x_2' y_1 y_2' + x_1 x_2 y_1 y_2$$

$$= x_1' y_1' (x_2' y_2' + x_2 y_2) + x_1 y_1 (x_2' y_2' + x_2 y_2)$$

$$= (x_2' y_2' + x_2 y_2)(x_1' y_1' + x_1 y_1)$$

$$= (\overline{x_1 \oplus y_1}) \cdot (\overline{x_2 \oplus y_2})$$

$$f_3 = x_1' y_1 + x_1' x_2' y_2 + x_2' y_1 y_2 = x_1' y_1 + x_2' y_2 (x_1' + y_1)$$



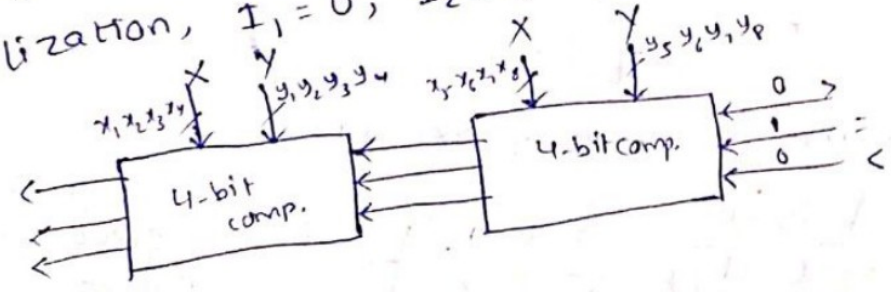
*

$$O_1 = f_1 + f_2 I_1$$

$$O_2 = f_2 I_2$$

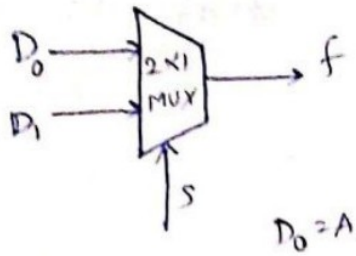
$$O_3 = f_3 + f_2 I_3$$

Initialization, $I_1 = 0$, $I_2 = 1$, $I_3 = 0$



Scanned by CamScanner

*

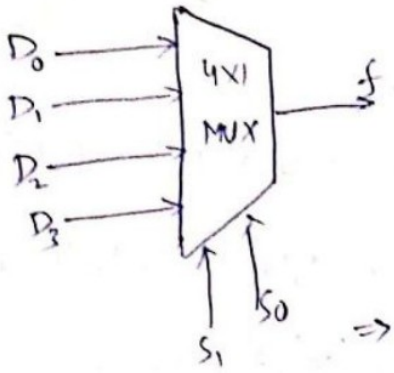


$S = 0, \; f = D_0$

$S = 1, \; f = D_1$

$\rightarrow f = D_0 S' + D_1 S$

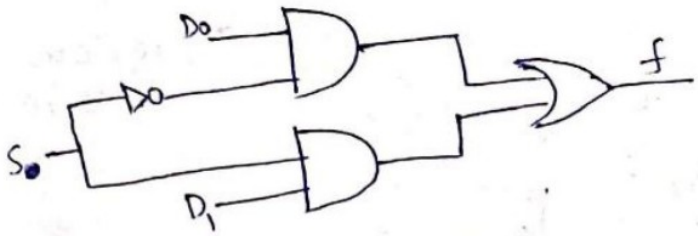
$D_0 = A, D_1 = A' \Rightarrow f = AS' + A'S = A \oplus S$



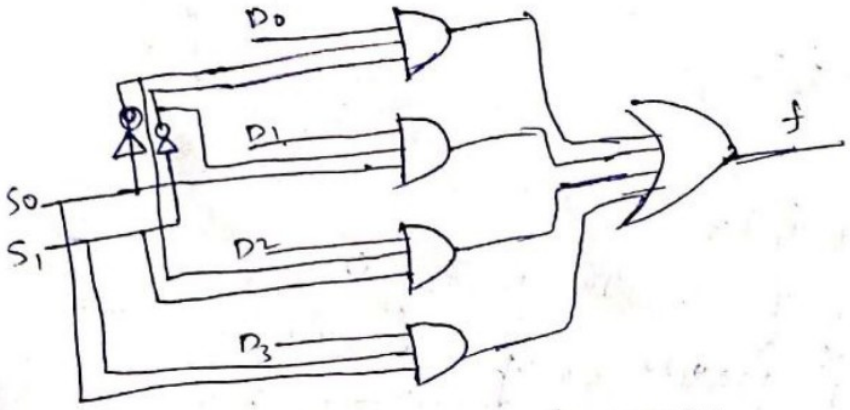
S_1	S_0	f
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

$\Rightarrow f = S_1' S_0' D_0 + S_1' S_0 D_1 + S_1 S_0' D_2 + S_1 S_0 D_3$

8x1 MUX $\Rightarrow f = S_2' S_1' S_0' D_0 + S_2' S_1' S_0 D_1 + S_2' S_1 S_0' D_2$
$+ S_2' S_1 S_0 D_3 + S_2 S_1' S_0' D_4 + S_2 S_1' S_0 D_5$
$+ S_2 S_1 S_0' D_6 + S_2 S_1 S_0 D_7$



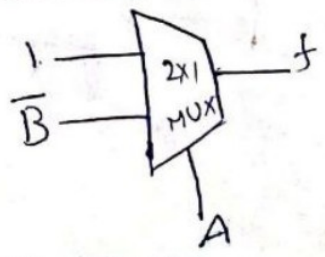
2x1 MUX



4x1 MUX

* 2-input NAND using 2x1 MUX.

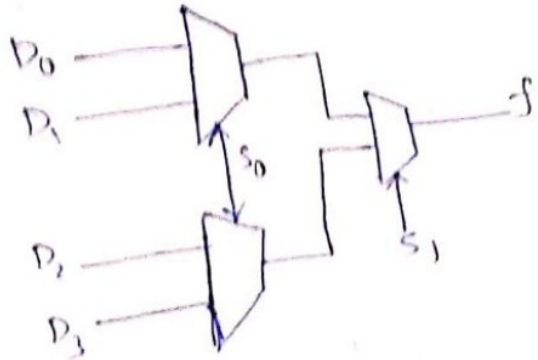
A	B	f
0	0	1
0	1	1 = 1
1	0	1
1	1	0 = $\bar{B}$



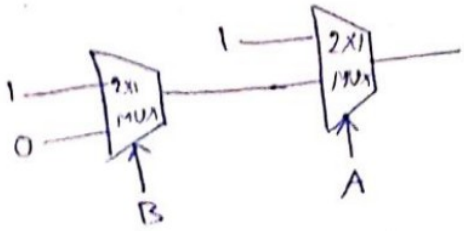
Input lines & select lines should not have same variable [Ex:- A, $\bar{A}$, A, A ...]

Scanned by CamScanner

* 4×1 MUX using 2×1 MUX:

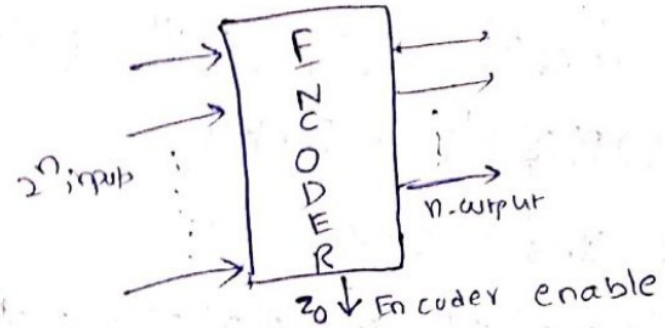


*



* Encoder:

2^n input – n output ckt.



Input lines – P_i

If $i > j$, then input P_i has higher priority than P_j.

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	f_0	f_1	f_2

Scanned by CamScanner

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	f_0	f_1	f_2	Z_0
1	0	0	0	0	0	0	0	0	0	0	.
∅	1	0	0	0	0	0	0	0	0	1	OR ($P_0 P_1 \ldots P_7$)
∅	∅	1	0	0	0	0	0	0	1	0	
∅	∅	∅	1	0	0	0	0	0	1	1	
∅	∅	∅	∅	1	0	0	0	1	0	0	$= P_0 + P_1 + P_2 + P_3 + P_4 + \ldots + P_7$
∅	∅	∅	∅	∅	1	0	0	1	0	1	
∅	∅	∅	∅	∅	∅	1	0	1	1	0	
∅	∅	∅	∅	∅	∅	∅	1	1	1	1	

Don't care since if $i > j$ P_i has higher priority

$$f_0 = P_4 P_5' P_6' P_7' + P_5 P_6' P_7' + P_6 P_7' + P_7$$

$$= P_4 P_5' P_6' P_7' + P_5 P_6' P_7' + (P_6 + P_7)$$

$$f_0 = P_4 + P_5 + P_6 + P_7$$

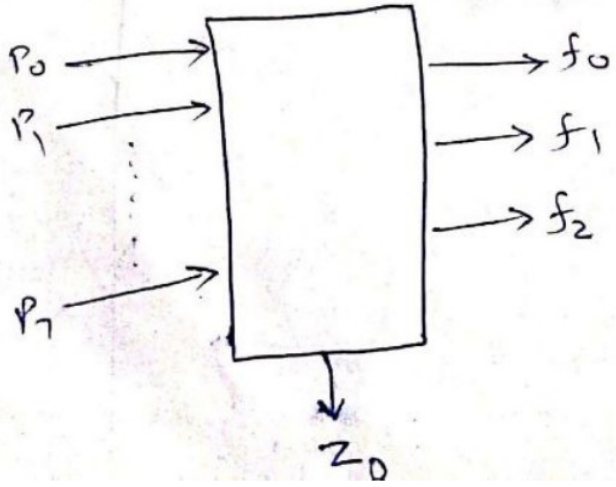
$$f_1 = P_2' P_3' P_4' P_5' P_6' P_7' + P_3 P_4' P_5' P_6' P_7' + P_6 P_7' + P_7$$

$$= (P_2 + P_3) P_4' P_5' P_6' P_7' + (P_6 + P_7)$$

$$f_1 = (P_2 + P_3) P_4' P_5' + P_6 + P_7$$

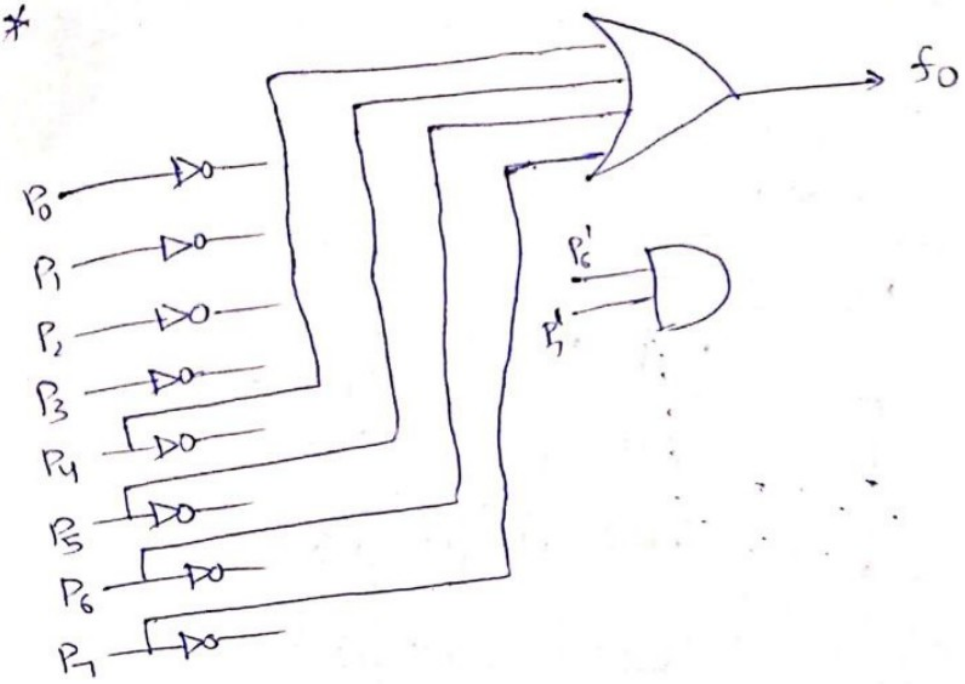
$$f_2 = P_1 P_2' P_3' P_4' P_5' P_6' P_7' + P_3 P_4' P_5' P_6' P_7' + P_5 P_6' P_7' + P_7$$

$$= P_4' P_5' P_6' P_7' (P_3 + P_1 P_2') + P_7 + P_5 P_6'$$

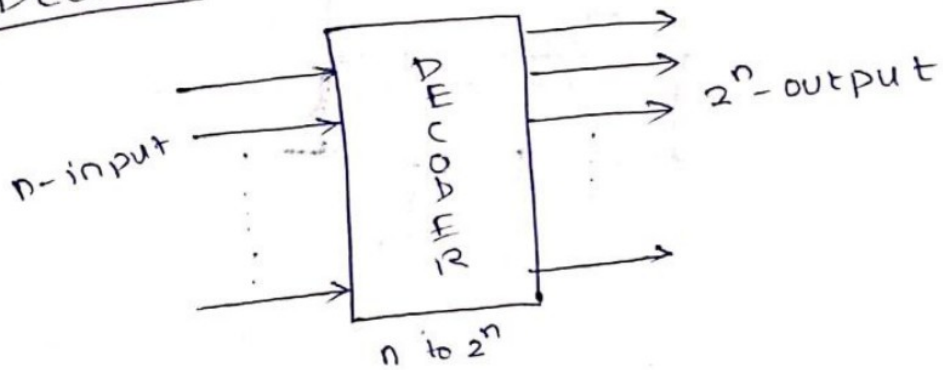


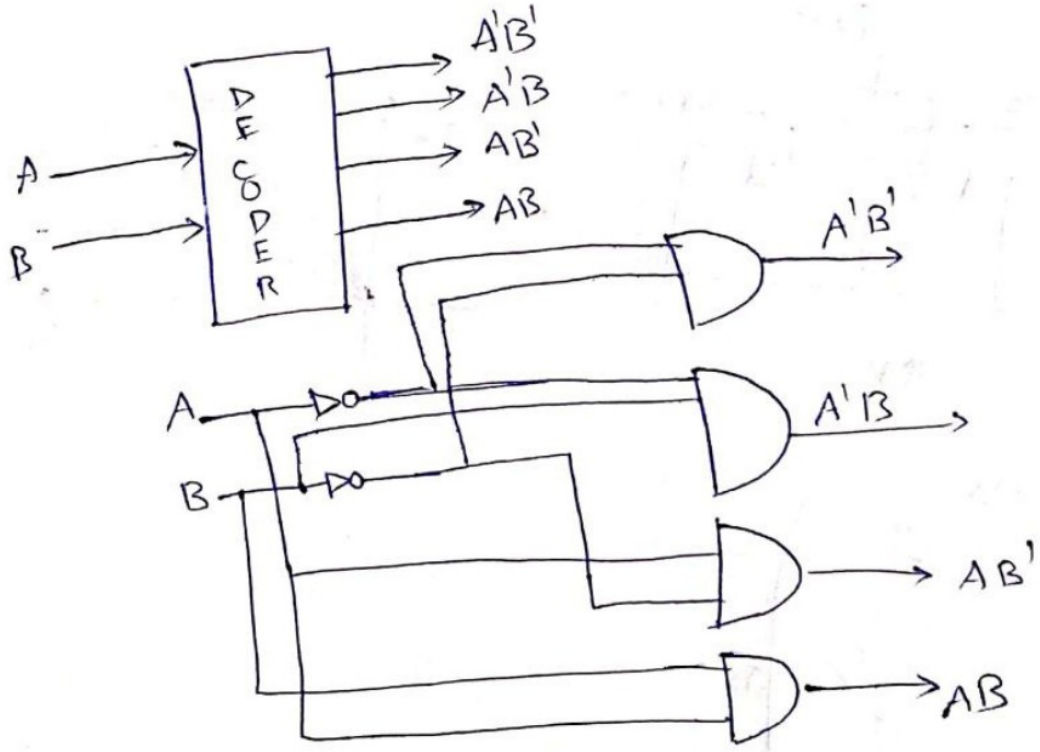
Scanned by CamScanner

*



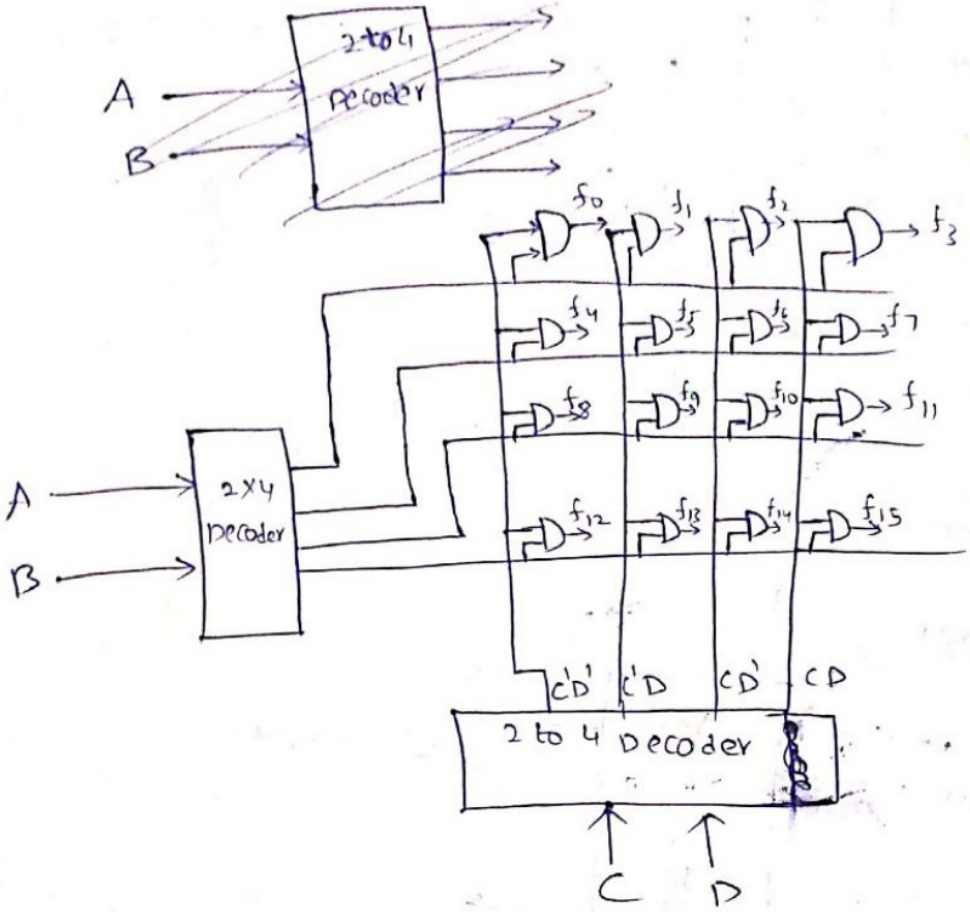
* Decoder



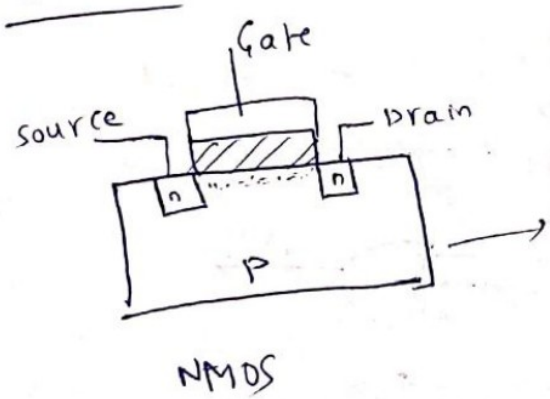


Scanned by CamScanner

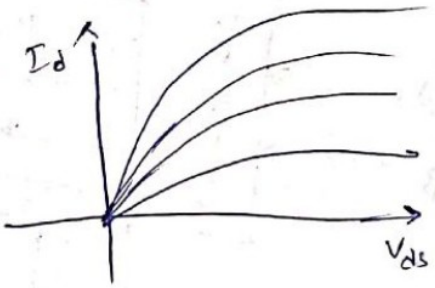
* 4 to 16 decoder using 2 to 4 Decoders.

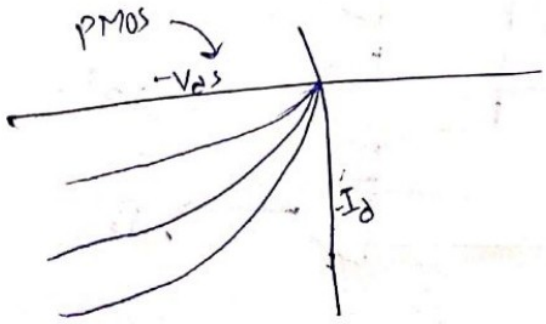


* CMOS

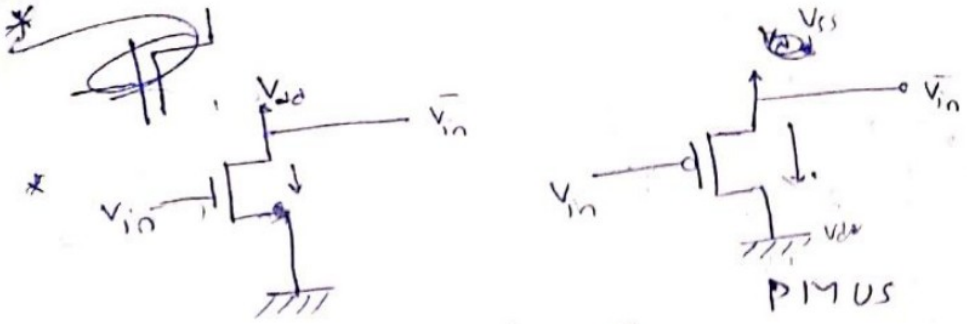


NMOS

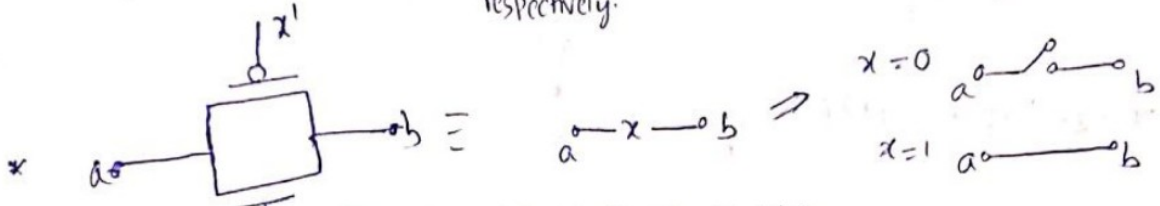




Scanned by CamScanner

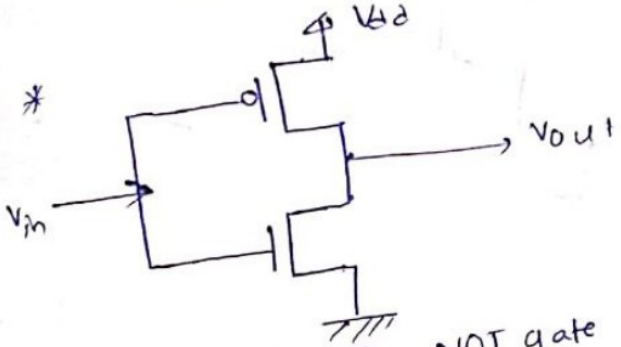


When V_{in} = 1, 0, current flows ⇒ power consumption respectively.

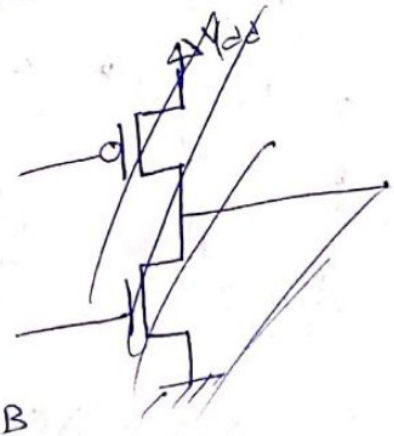


x → Complementary switch

Transmission gate.

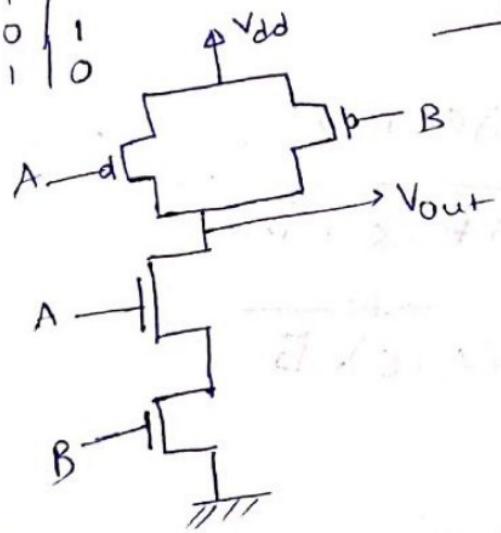


CMOS NOT gate

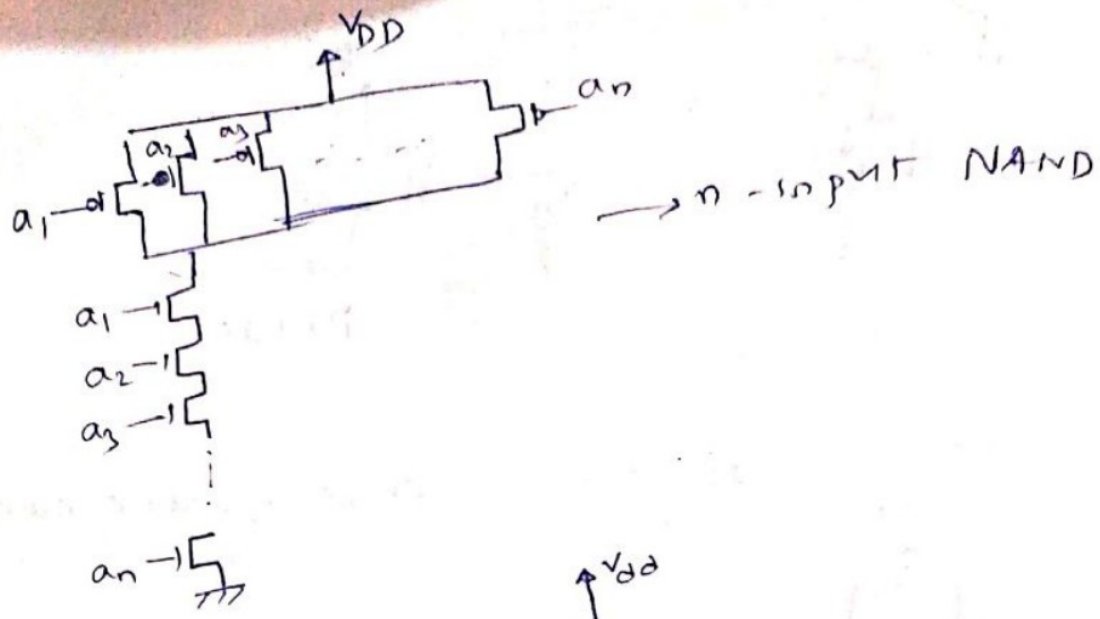


* 2 input NAND

A	B	NAND
0	0	1
0	1	1
1	0	1
1	1	0

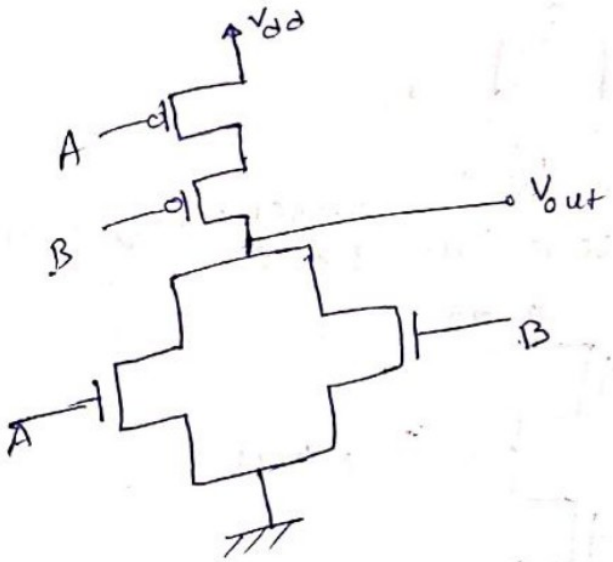


Scanned by CamScanner

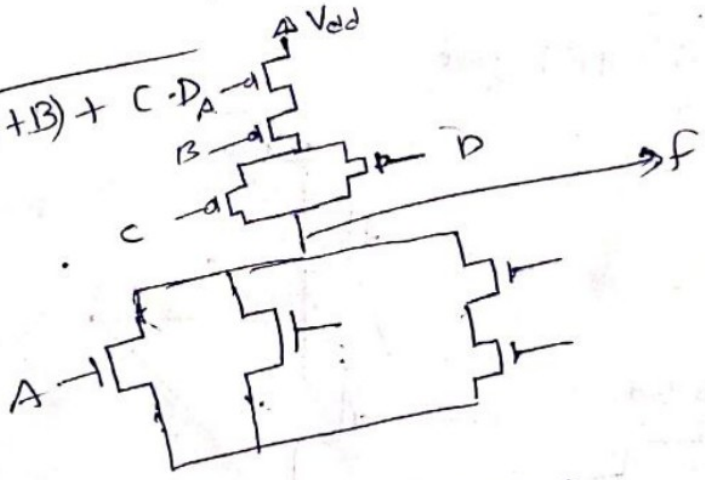


→ n - input NAND

* 2- input NOR



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



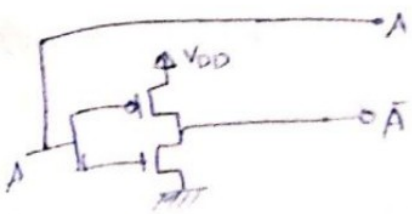
* $f = A\bar{B} + \bar{A}C + D$

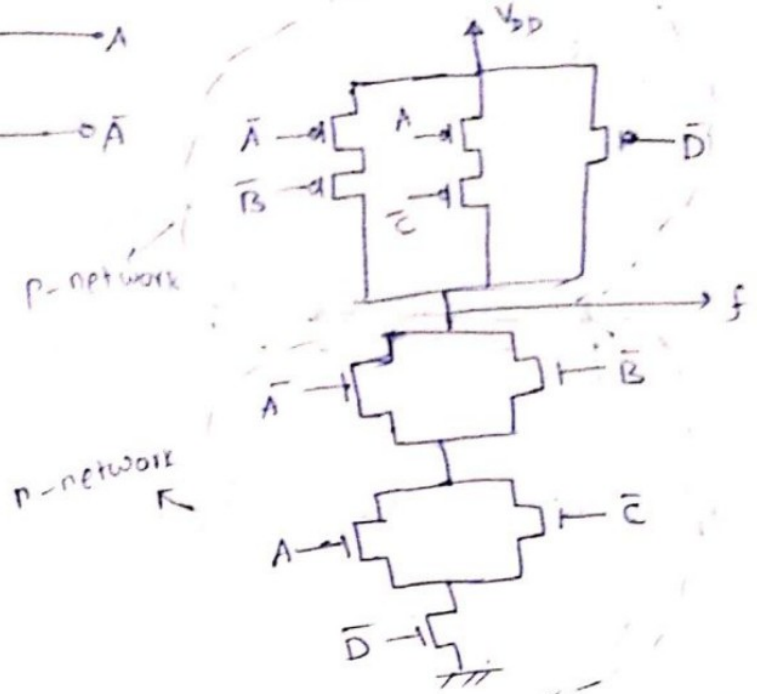
$\bar{f} = \overline{(\bar{A}\bar{B} + \bar{A}C + D)}$

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

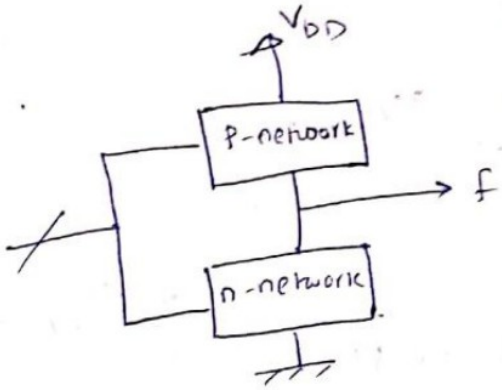
$= \overline{(\bar{A}+\bar{B}) \cdot (A+\bar{C}) \cdot \bar{D}}$

Scanned by CamScanner

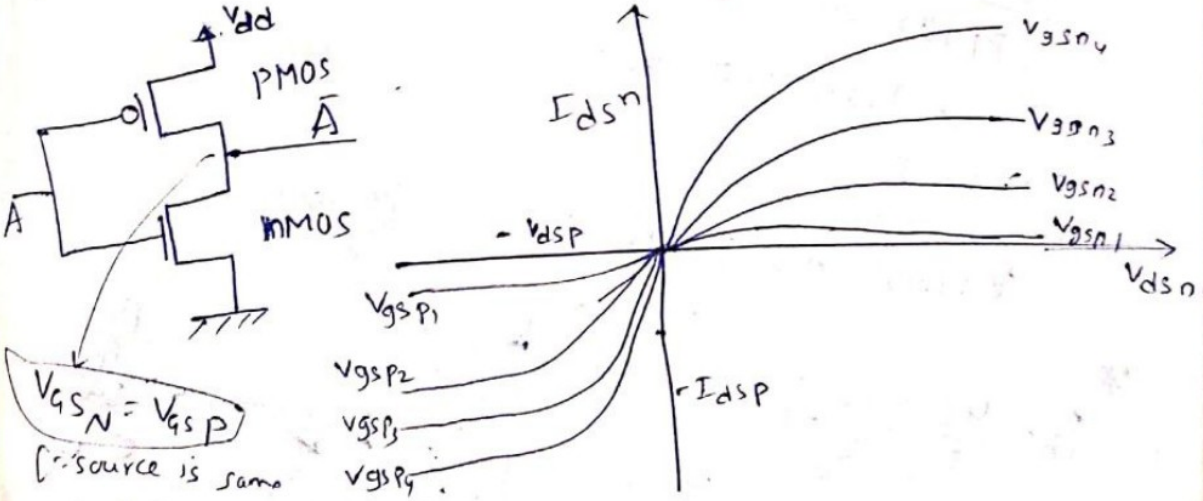




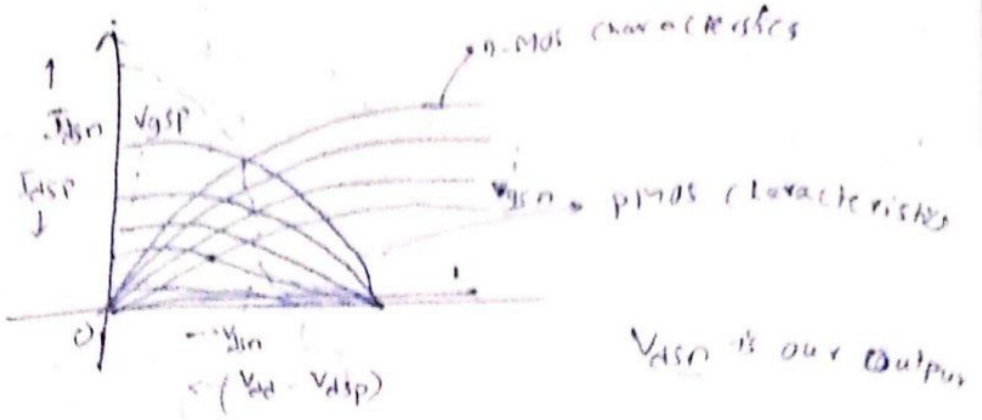
*



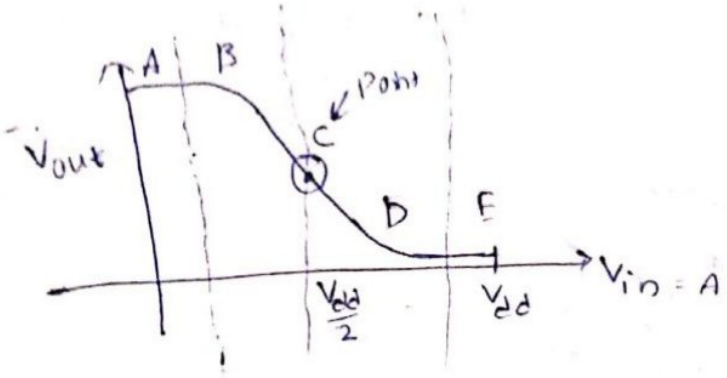
* DC characteristics of NOT (CMOS Inverter):-



Scanned by CamScanner



$V_{gsn} = V_{gsp}$, $I_{dsn} = I_{dsp}$



Region A

PMOS is saturated (on), NMOS is ~~off~~ cutoff (off).

Region B, C, D

PMOS on, NMOS on

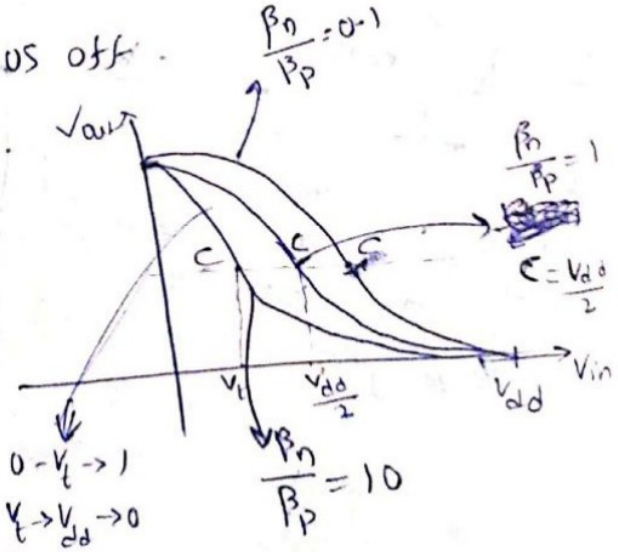
C ⟶ transition point.

Region E

NMOS on, PMOS off.

* $\beta_n = \frac{\mu \epsilon}{t}\left(\frac{W}{L}\right)$

μ → Mobility of e^-



Scanned by CamScanner

* Noise Margin(NM)

$$NM_L = |V_{IL\,Max} - V_{OL\,Min}|$$

$$NM_H = |V_{IH\,Max} - V_{OH\,Max}|$$

* Adder:

$$A = 0110$$
$$B = 1011$$
$$10001$$
C S

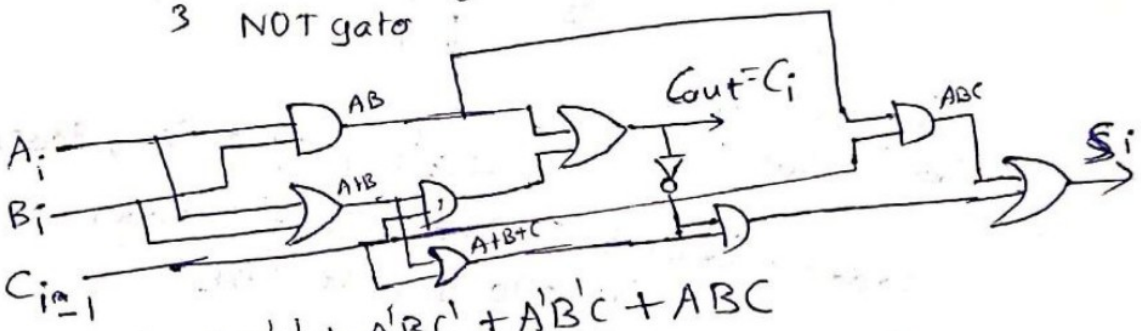
$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = AB + BC_{in} + C_{in}A = AB + C(A+B)$$

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

$$C_{out} = A'BC + AB'C + ABC' + ABC$$

4 3-input AND gates ⇒ 8 2-input AND
1 4-input OR gate
3 NOT gate



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

$$\Rightarrow S = (A+B+C)(AB + C(A+B))' + ABC$$

computation: $S = (A+B+C)(AB + C(A+B))' + ABC$

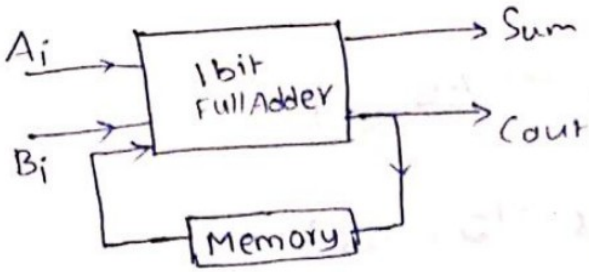
$$S = (A+B+C)\cdot(A'+B')\cdot(A'+C')\cdot(B'+C') + ABC$$

$$S = (A'B + A'C + AB' + B'C)\cdot(A'B' + A'C' + B'C' + C') + ABC$$

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

4 2-input AND gates
4 2-input OR gates
1 NOT gate

Scanned by CamScanner



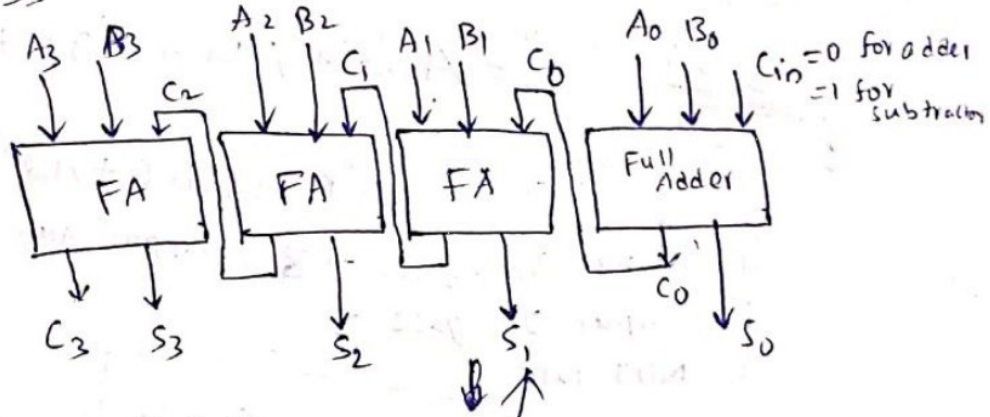
*

$C_2\ C_1\ C_0$

$A_3\ A_2\ A_1\ A_0$

$B_3\ B_2\ B_1\ B_0$

$C_3\ S_3\ S_2\ S_1\ S_0$



C_3 — Cout

$S_3 S_2 S_1 S_0$ — Sum

4-bit Ripple Carry Adder

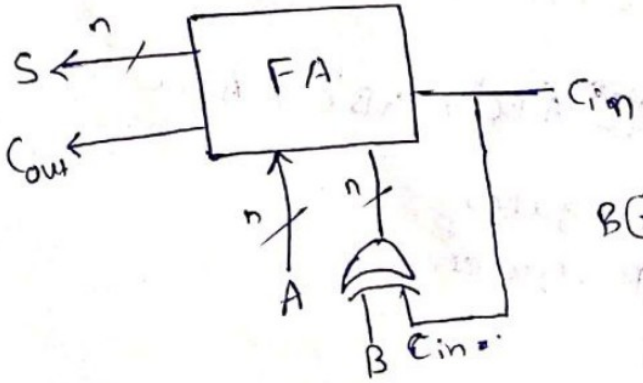
t_{add} → addition time of a full adder block. (time to generate carry).

t_{prog} → carry propagation time.

n-bit Ripple carry Adder.

time required = $n\,t_{add} + (n-1)\,t_{prog}$

*



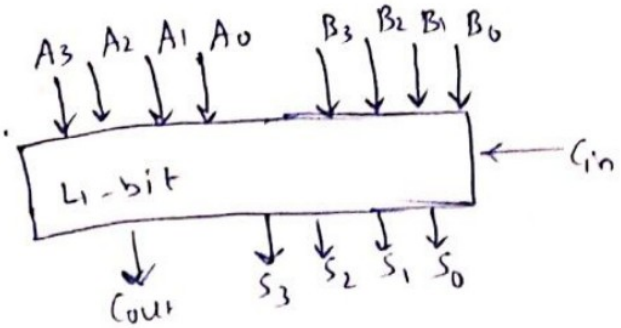
$B \oplus C_{in} = BC_{in}' + B'C_{in}$

$C_{in}=0 \Rightarrow B$

$C_{in}=1 \Rightarrow B'$

Scanned by CamScanner

*



* Carry look Ahead Adder:

$$C_i = AB + BC_{i-1} + AC_{i-1}$$

Carry generate $g_i = A.B$

Carry propagate. $P_i = A + B$

$$C_i = A_i B_i + C_{i-1}(A_i + B_i)$$

$$c_i = g_i + C_{i-1} P_i$$

$$C_0 = g_0 + C_{in} P_0$$

$$C_1 = g_1 + P_1 C_0$$

$$c_1 = g_1 + P_1 (g_0 + P_0 C_{in})$$

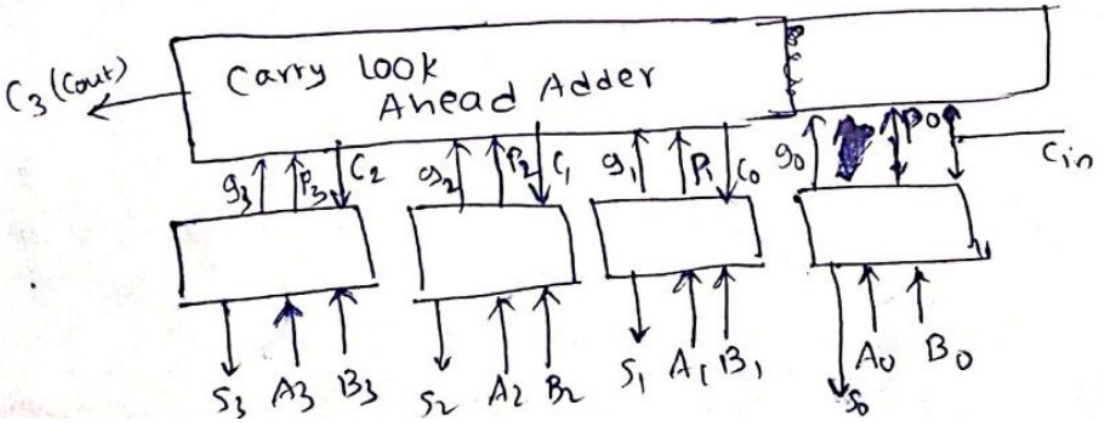
$$C_1 = g_1 + P_1 g_0 + P_1 P_0 C_{in}$$

$$C_2 = g_2 + P_2 C_1$$

$$C_2 = g_2 + P_2 (g_1 + P_1 g_0 + P_1 P_0 C_{in})$$

$$C_2 = g_2 + P_2 g_1 + P_2 P_1 g_0 + P_2 P_1 P_0 C_{in}$$

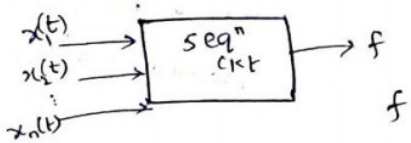
$$C_3 = g_3 + P_3 g_2 + P_3 P_2 g_1 + P_3 P_2 P_1 P_0 C_{in}$$



Scanned by CamScanner

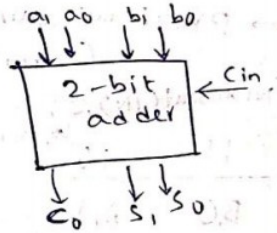
* Sequential Circuit Design :-

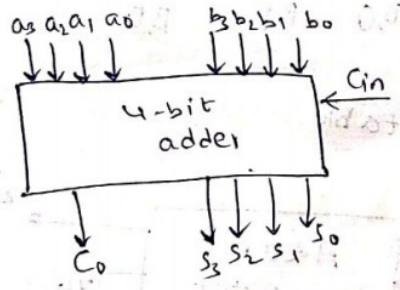
→ The output depends on the current input and some stored information (can be states of ckt or any other information)

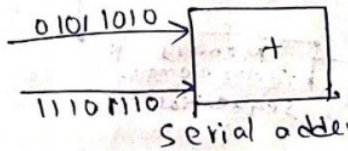


$f(x_i(t), y_j(t-1))$

Adder





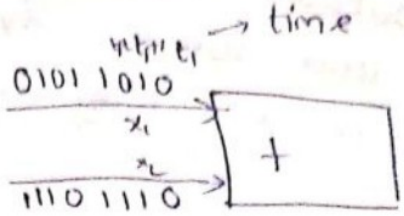


→ data is added bit by bit

Serial adder

Scanned by CamScanner

```
  1 1
  0 1 1 0 0 1
+ 0 1 1 1 0 0
-------------
  1 1 0 1 0 1
```



Addition = $x_1^{t,t+1} \rightarrow x_2^{t,t+1} + c^{t-1}$ (carry)

* Present input, Stored info
  x_1^t, x_2^t c^{t-1}

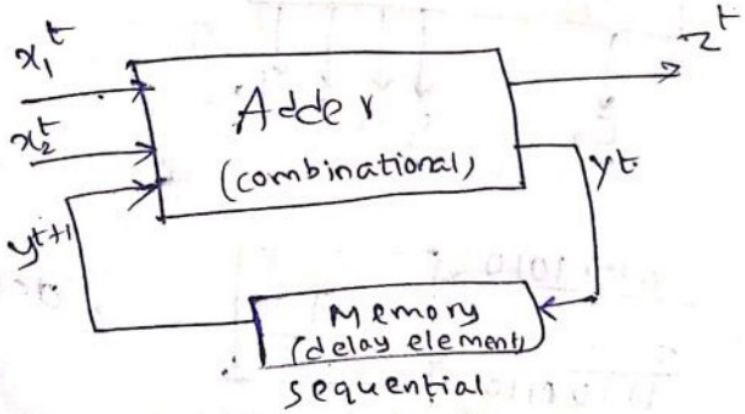
State A → previous carry is 0 (carry absent).

state B → previous carry is 1 (carry present)

present State	prev. carry	x_1x_2 = 00	01	11	10
		(Next state (NS), Z)			x_1+x_2+ Carry
A	(C=0)	A,0	A,1	B,0	A,1
B	(C=1)	A,1	B,0	B,1	B,0

State Assignment

State table.



Scanned by CamScanner

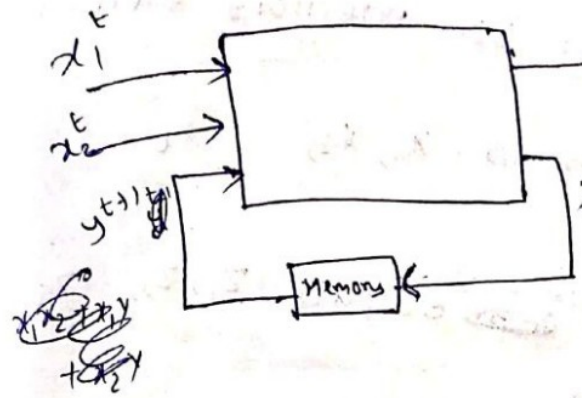
$y(t+1)$

y \ x_1x_2	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$y = y^{(t)} = x_1x_2 + x_1y + x_2y$

y \ x_1x_2	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$Z = x_1 \oplus x_2 \oplus y$

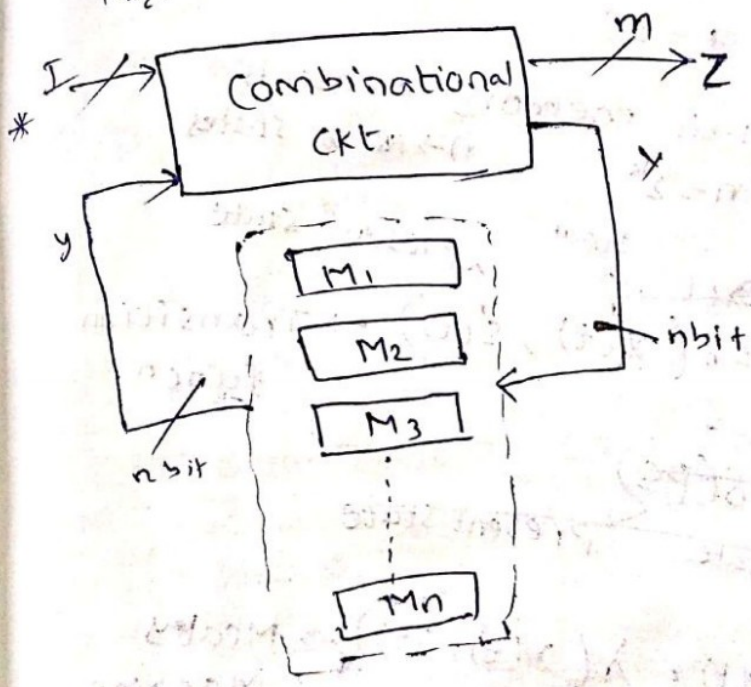


$Z = x_1^t \oplus x_2^t \oplus y^t$

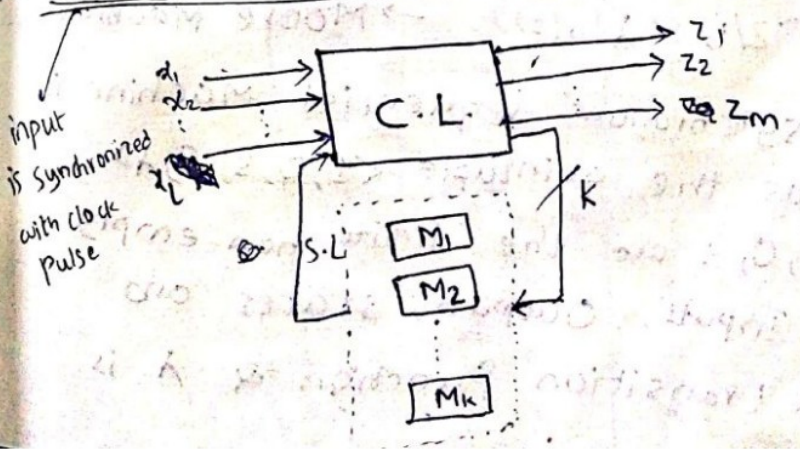
y^t

$x_1x_2 + x_1y^{t-1} + x_2y^{t-1}$

$(y^{t+1} = y^t)$



* Synchronous Sequential machines :-



C.L → Combinational Logic

S.L → Sequential Logic

Scanned by CamScanner

$P = 2^L$

O → q no.of ~~outp~~ m-bit outputs $z_1, z_2, \ldots z_q$

$q = 2^m$

S = k no.of n-bit memory elements

* I – L- 1-bit inputs $x_1, x_2, \ldots x_L$

input Alphabet (set of all possible values $x_1, x_2, \ldots x_L$ can take)

$P = 2^L$

O → m – 1-bit ~~outputs~~ outputs $z_1, z_2, \ldots z_q$

Output Alphabet

$q = 2^m$

S → K – 1-bit memory elements

Set of states

$n = 2^k$ n → No. of states

~~$S(t+1) = \delta(x(t))$~~

* $S(t+1) = \delta(x(t), S(t))$ → Transition funcn

Input, present state

Next state

~~$NS = \delta(PS)$~~

~~Next state~~ ~~present state~~

$Z(t) = \lambda(x(t), S(t))$ → Meaty Machine

(or)

output

$Z(t) = \lambda(S(t))$ → Moore Machine

* A ~~sequ~~ synchronous sequential Machine is defined as the quintuple $\langle I, O, S, \delta, \lambda \rangle$ where I, O, S are the finite non-empty sets of Inputs, Outputs, states and δ is the transition function & λ is

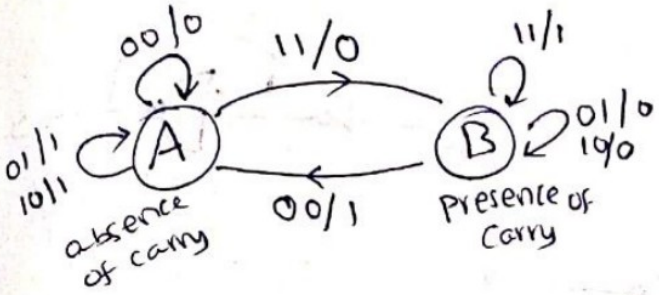
Scanned by CamScanner

the output function & they are defined as

$\delta: I \times S \rightarrow S$

$\lambda: I \times S \rightarrow O$ (for mealy machine)

$\lambda: S \rightarrow O$ (for Moore machine)

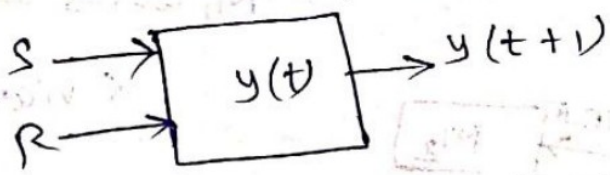


*Memory elements

1. SR latch
2. D flip-flop
3. T flip-flop
4. J-K flipflop

* Set-Reset Latch :-

Excitation Table of R-S latch



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

Scanned by CamScanner

Scanned by CamScanner

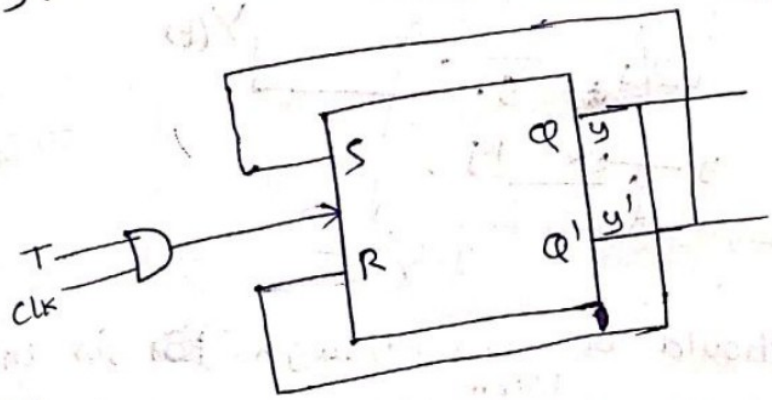
* Trigger or T-latch :

Input is 0, output is 1 } when excited
($y(t)$ $y(t+1)$)

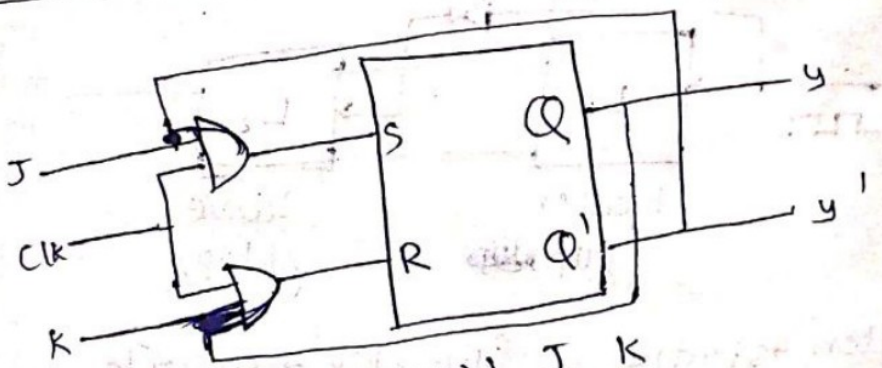
Input is 1, output is 0

$y(t)$	$y(t+1)$	T
0	0	0
0	1	1
1	0	1
1	1	0

$y(t+1) = T\,y'(t) + y(t)T' = T \oplus y(t)$.



* J-K Latch → SR + T merge :



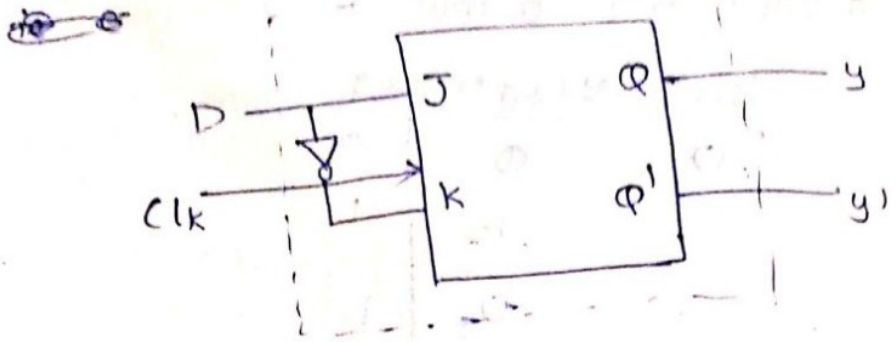
$y(t)$	$y(t+1)$	J	K
0	0	0	—
0	1	1	—
1	0	—	1
1	1	—	0

Scanned by CamScanner

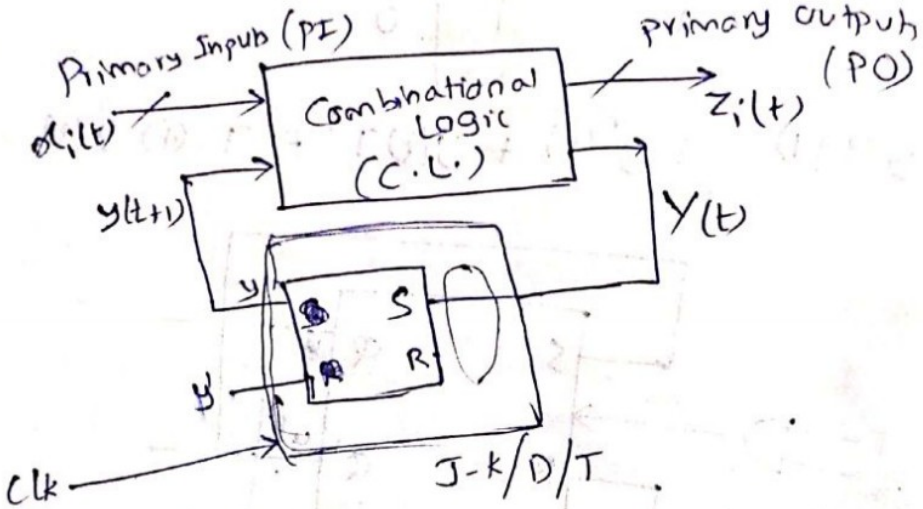
* D - latch :-

~~$y(t+1) = D(t)$~~

$y(t+1) = D(t)$



*

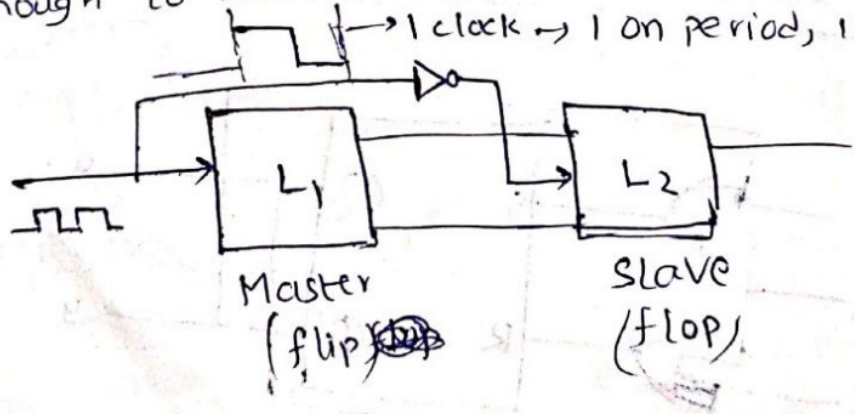


↓

period should be long enough ~~for~~ for the state of the ~~machine~~ latch to change & short enough to not change ~~twice~~.

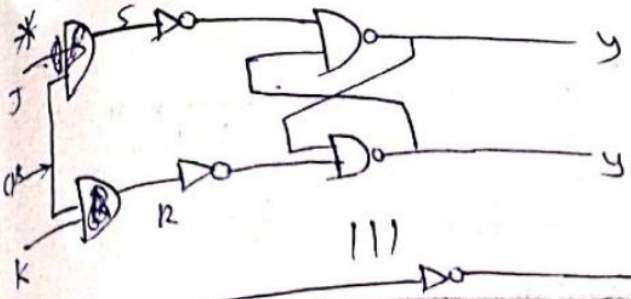
→ 1 clock → 1 on period, 1 off period

*

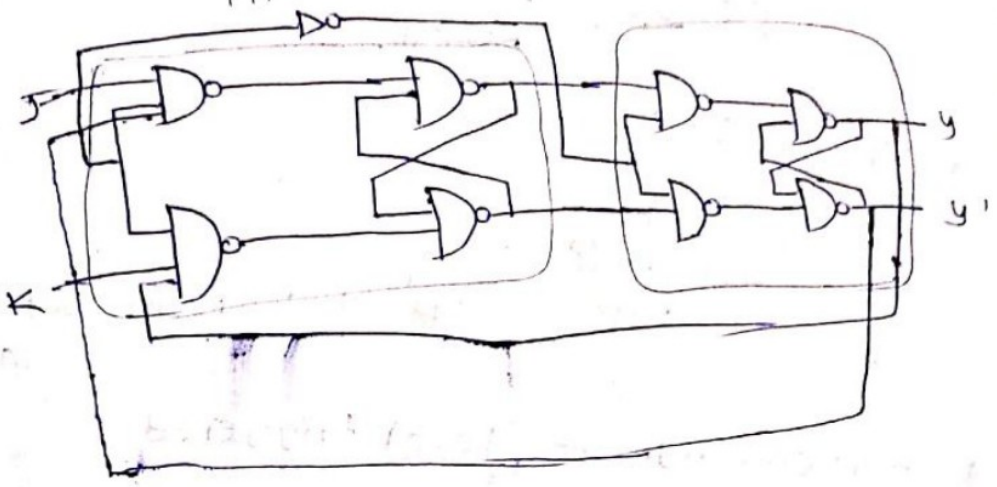


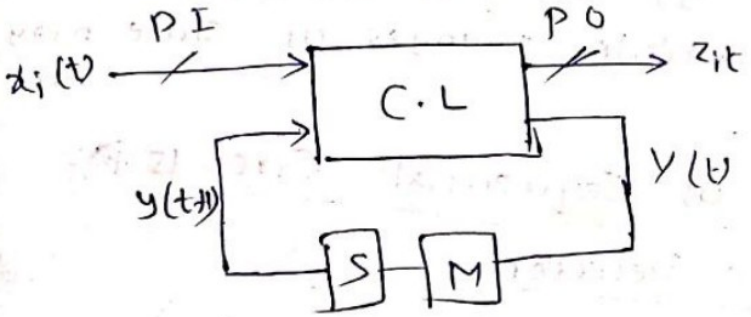
→ latch → single / flip flop → double.

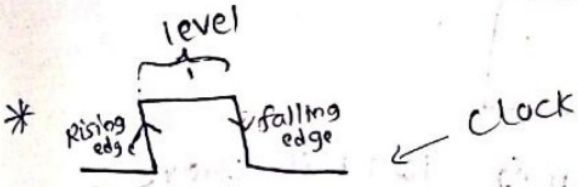
Scanned by CamScanner



III

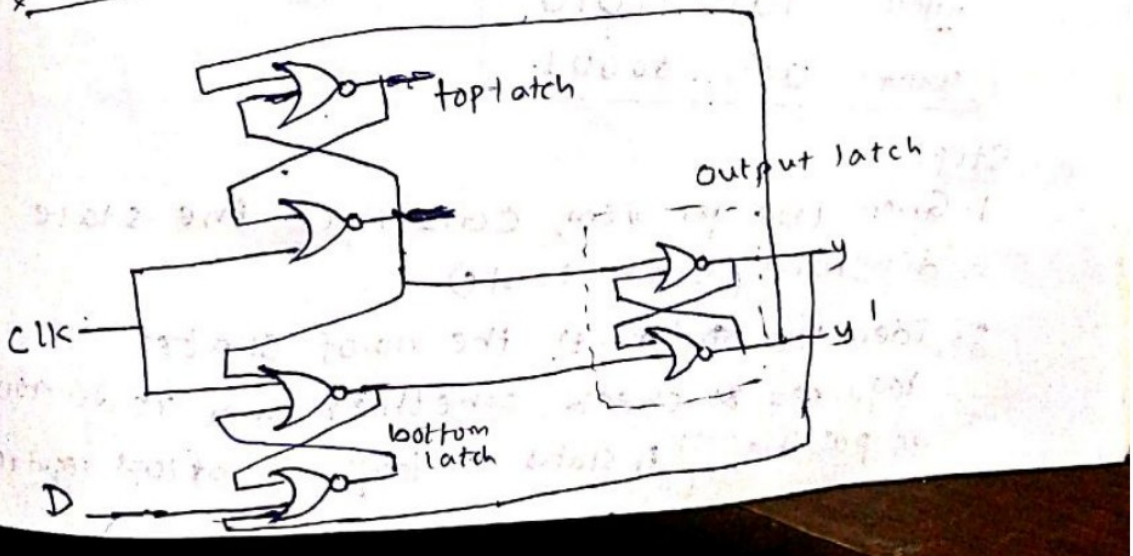






Negative

* Edge triggered flip-flop :- (triggered by transition from 0 to 1 or 1 to 0)



Scanned by CamScanner

→ Clk high

D input is applied

⇒ bottom $y' = D'$, inputs to output latch are

top $y = D$

00

RS - 00

⇒ latched

→ Clk low (high to low)

~~RS = 00~~

bottom → D

top → D' ⇒ S = 1 for output latch

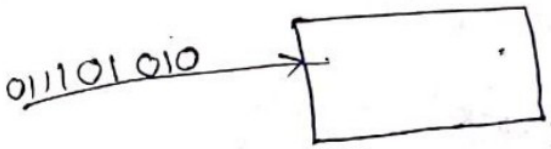
* short comings of level triggered

↓

If it is at high(1) for a long time, multiple changes of state may occur.

* Synthesis of Sequential Circuits:-

* Sequence Detector:-



To find if substring 1010 is there

input: 1010 11010

output: 0001 00001.

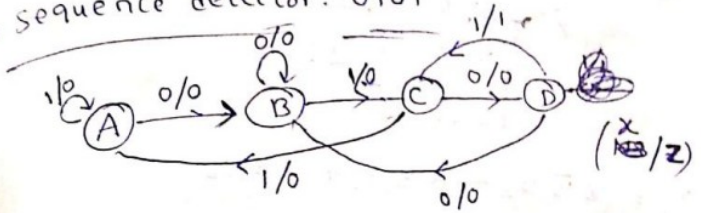
Steps:-

1. Given the problem, construct the state diagram (state table).
2. Identify (count) the no. of states required & check whether any reduction is possible. (n states ⇒ $\log_2^n$ flipflops required)

Scanned by CamScanner

5. Identify the type of flipflops suited to your application
4. State assignment
5. Form the excitation table
6. Minimize the NS (next states) & Outputs.
7. Draw the circuit

* Sequence detector: 0101



PS	x=0	x=1
00 A	B,0	A,0
01 B	B,0	C,0
11 C	D,0	A,0
10 D	B,0	C,1

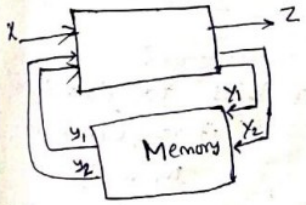
→ (NS, Z)

Y_1Y_2 · Y_1Y_2 Y_1Y_2

A → 00, B → 01, C → 11

Y_1Y_2

D → 10



PS y_1y_2	x=0 Y_1Y_2, z	x=1 Y_1Y_2, z
00	01, 0	00, 0
01	01, 0	11, 0
11	10, 0	00, 0
10	01, 0	11, 1

Scanned by CamScanner

Y_1

x \ y_1y_2	00	01	11	10
0	0	0	1	0
1	0	1	0	1

Y_2

x \ y_1y_2	00	01	11	10
0	1	1	0	1
1	0	1	0	1

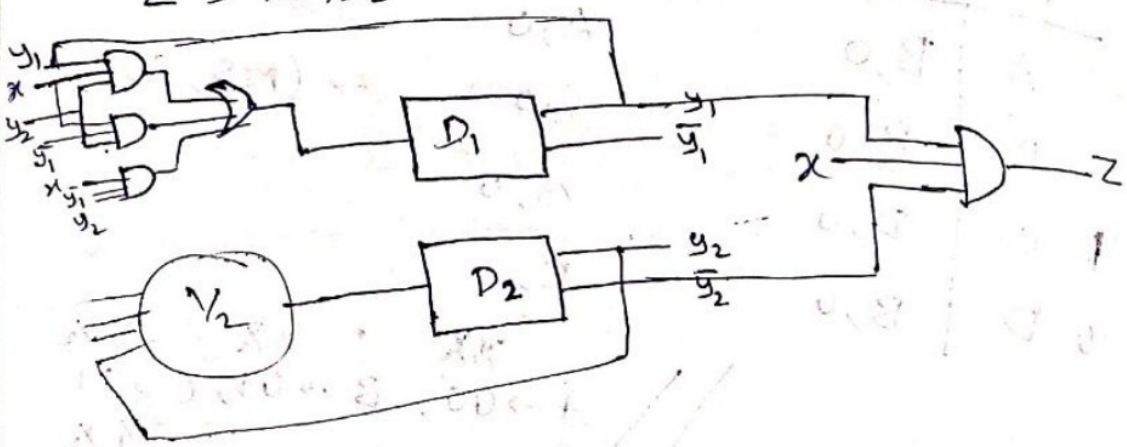
Z

x \ y_1y_2	00	01	11	10
0	0	0	0	0
1	0	0	0	1

$$Y_1 = x\bar{y}_1y_2 + x y_1\bar{y}_2 + x y_1y_2$$

$$Y_2 = \bar{x}\,\bar{y}_1 + \bar{y}_1 y_2 + y_1\bar{y}_2$$

$$Z = x y_1\bar{y}_2$$



* Parity-bit Generator

m-bits → to be transferred

we transfer m+c = n bits to identify if any error in m bits

c-bits → redundant

← m bits → ← c →

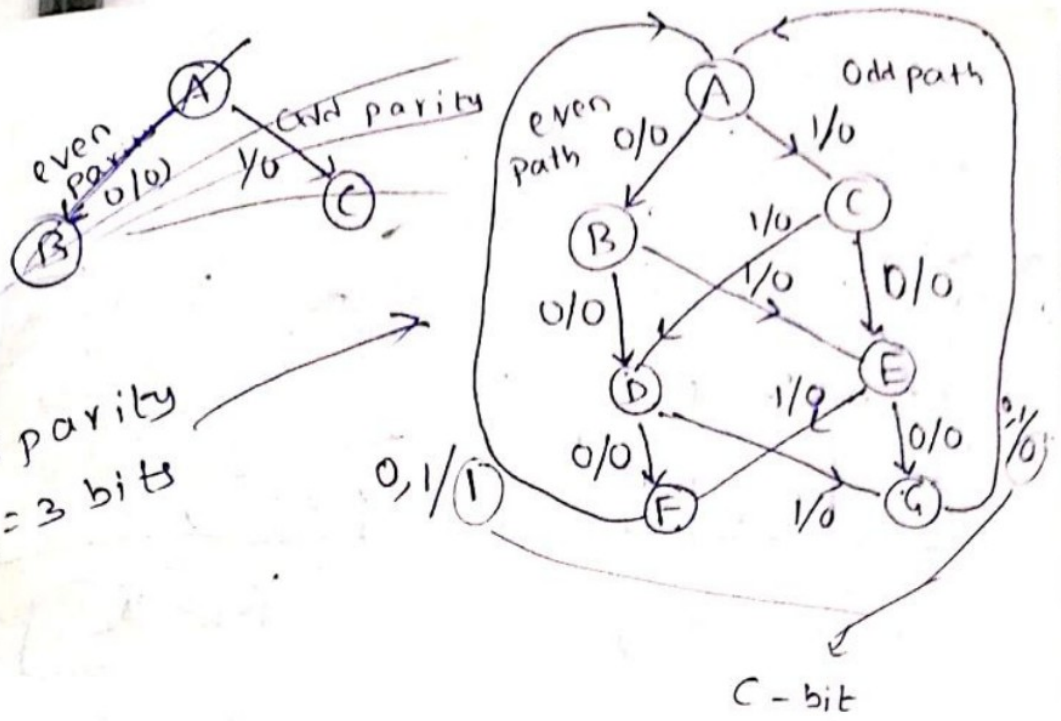
0	1	0	1	0
0	1	1	1	1

} even parity

⇒ Total no. of 1's in m+c bits should be even

Scanned by CamScanner

1/0.



state assignment $y_1 y_2 y_3$	NS $(Y_1 Y_2 Y_3)$ $x=0$	NS $(Y_1 Y_2 Y_3)$ $x=1$	output (z) $x=0$	output (z) $x=1$
000 A	B 010	C 011	0	0
010 B	D 110	E 111	0	0
011 C	E 111	D 110	0	0
110 D	F 100	G 101	0	0
111 E	G 101	F 100	0	0
100 F	A 000	A 000	1	1
101 G	A 000	A 000	0	0

J-K flipflop

	$x=0$			$x=1$		
	$J_1 K_1$	$J_2 K_2$	$J_3 K_3$	$J_1 K_1$	$J_2 K_2$	$J_3 K_3$
000	0 –	1 –	0 –	0 –	1 –	1 –
010	1 –	– 0	0 –	1 –	– 0	– 1
011	1 –	– 0	– 0	1 –	– 0	– 1
110	– 0	– 1	0 –	– 0	– 1	1 –
111	– 0	– 1	– 0	– 0	– 1	– 1
100	– 1	0 –	0 –	– 1	0 –	0 –
101	– 1	0 –	– 1	– 1	0 –	– 1

Scanned by CamScanner

$Z = y_1 \bar{y}_2 \bar{y}_3$

For J_1

xy_1 \ y_2y_3	00	01	11	10
00	0	X	1	1
01	X	X	X	X
11	X	X	X	X
10	0	X	1	1

$J_1 = y_2$

For K_1

xy_1 \ y_2y_3	00	01	11	10
00	X	X	X	X
01	1	1	0	0
11	1	1	0	0
10	X	X	X	X

$K_1 = \bar{y}_2$

For J_2

xy_1 \ y_2y_3	00	01	11	10
00	1	X	X	X
01	0	0	X	X
11	0	0	X	X
10	1	X	X	X

$x\bar{y}_1 + x\bar{y}_1$

$J_2 = \bar{y}_1$

For K_2

xy_1 \ y_2y_3	00	01	11	10
00	X	X	0	0
01	X	X	1	1
11	X	X	1	1
10	X	X	0	0

$K_2 = y_1$

For J_3

xy_1 \ y_2y_3	00	01	11	10
00	0	X	X	0
01	0	X	X	0
11	0	X	X	1
10	1	X	X	1

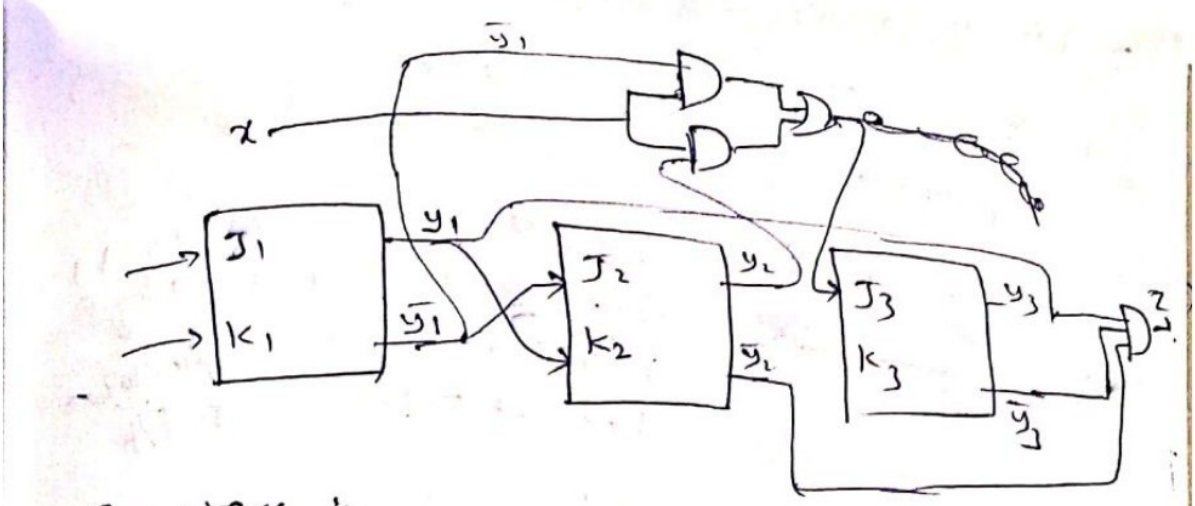
$J_3 = x\bar{y}_1 + xy_2$

For K_3

xy_1 \ y_2y_3	00	01	11	10
00	X	X	0	X
01	X	1	0	X
11	X	1	1	X
10	X	X	1	X

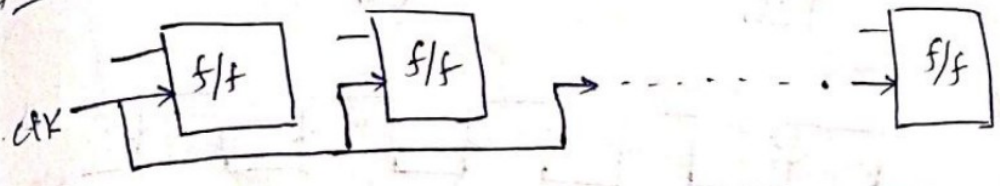
$K_3 = \bar{y}_2 + x$

Scanned by CamScanner



* Counters :

synchronous :



→ Lock out

→ mod 6 counter

	3-bit
0	000
1	001
2	010
3	011
4	100
5	101
→ 6	110 → 000
→ 7	111 → 001

these 2 are invalid states ⇒ these must be locked out.

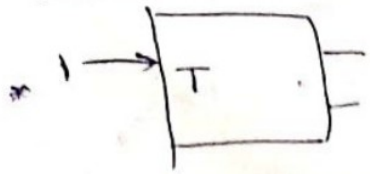
→ Asynchronous Counter :

→ upcounter → 0, 1, 2, 3, 4, 5, 0, 1, 2, 3, ...

→ down counter → 5, 4, 3, 2, 1, 0, 5, 4, 3, ...

Scanned by CamScanner

2-bit up counter (-ve edge triggered)

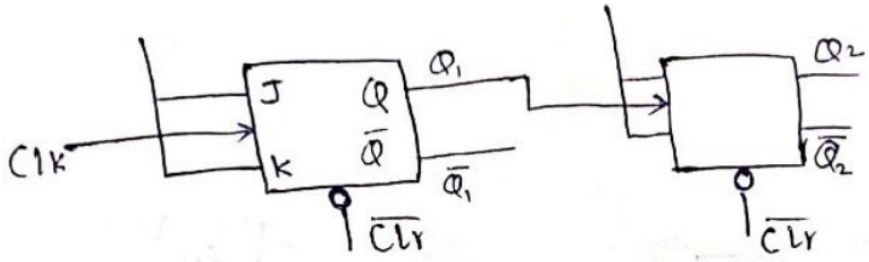


i/p T = 1 ⇒ 0 → 1
1 → 0

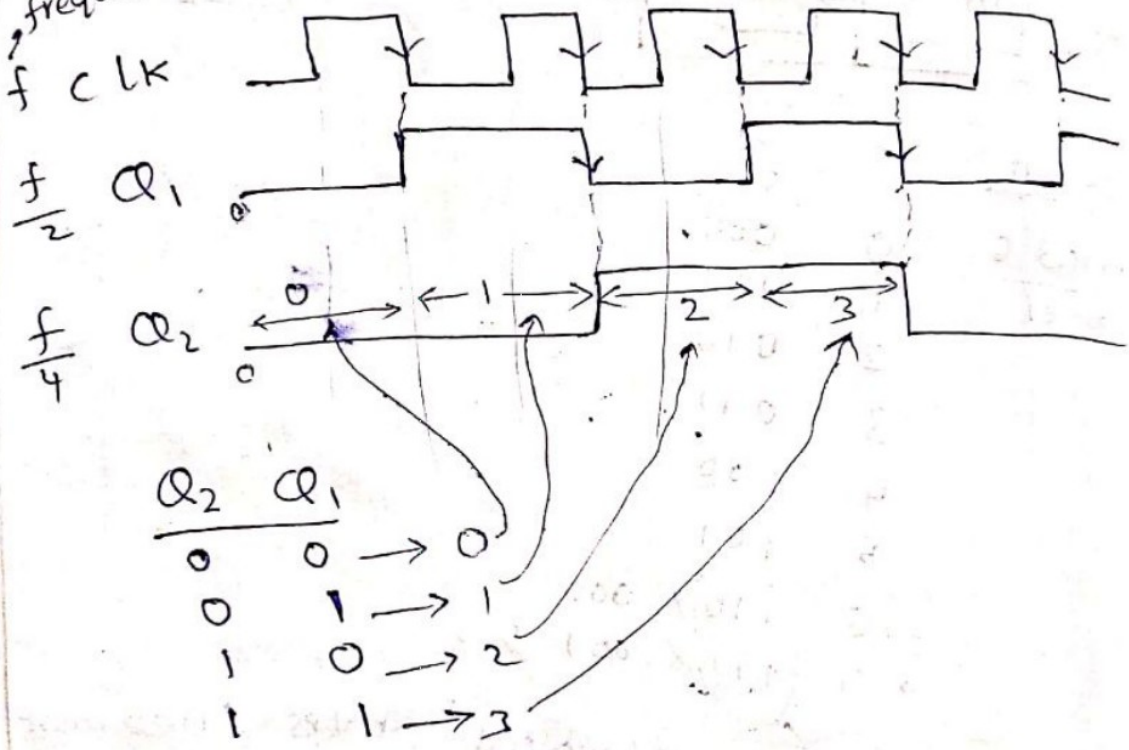
0 → 0 ⇒ 0 —
0 → 1 ⇒ 1 —
1 → 0 ⇒ — 1
1 → 1 ⇒ — 0

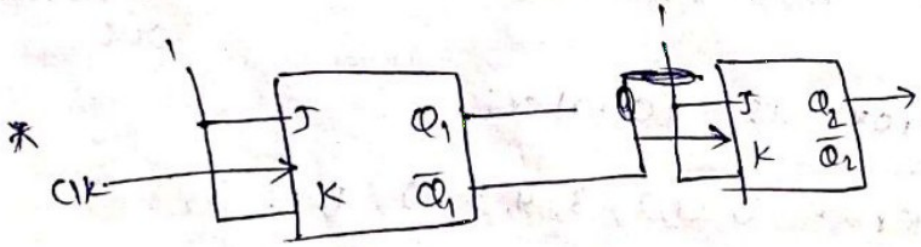
⇒ 0 → 1, 1 → 0 } 11

Similar to T-f/f

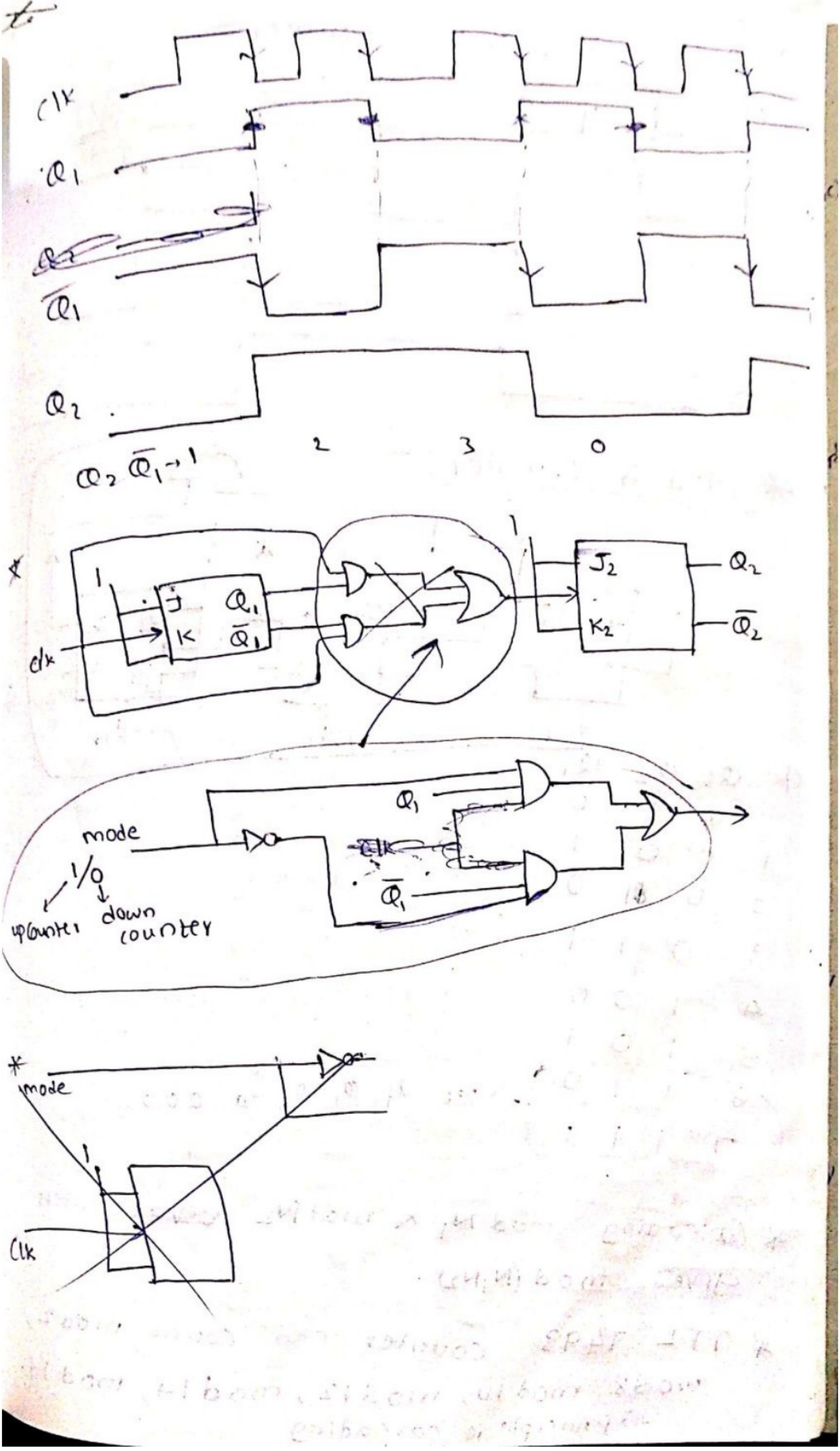


frequency



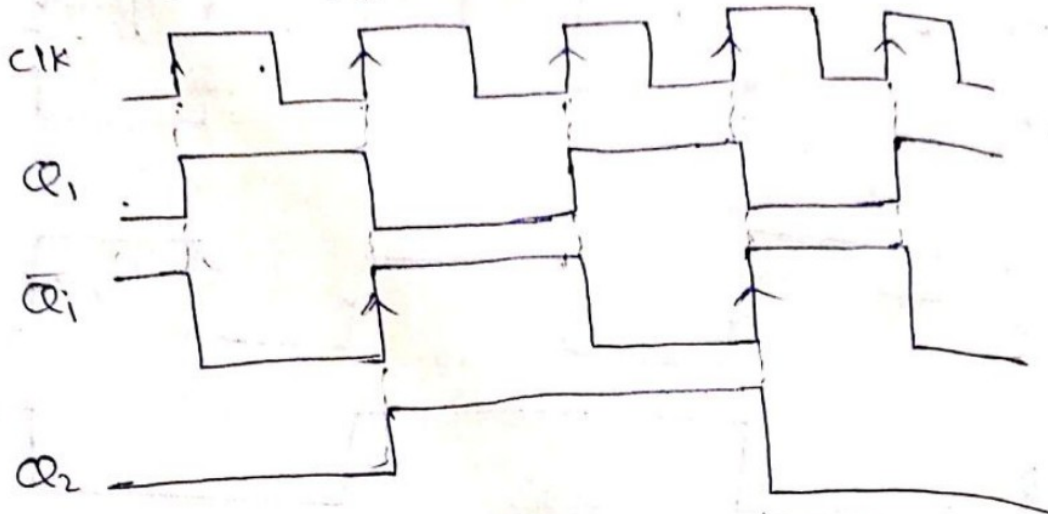


Scanned by CamScanner

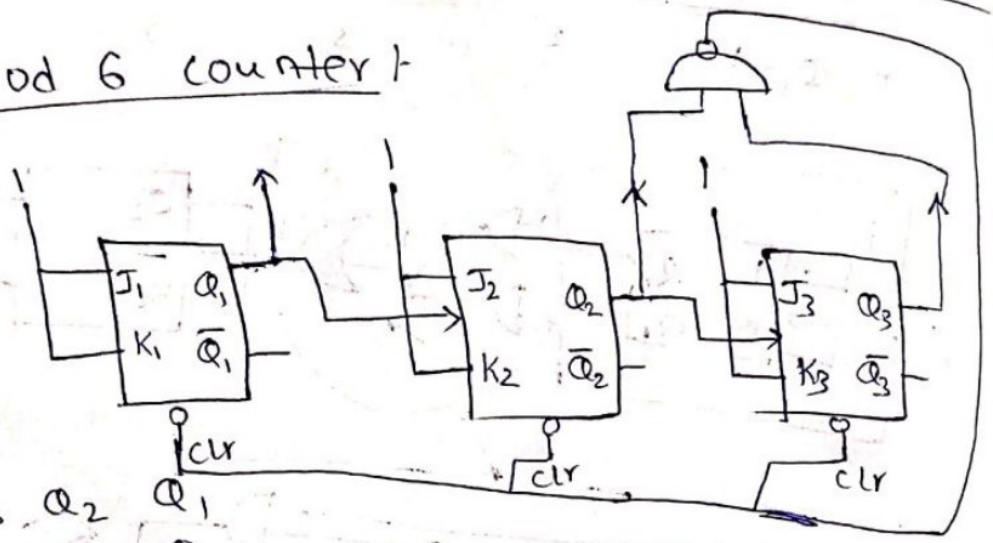


Scanned by CamScanner

→ +ve edge triggered upcounter



* Mod 6 counter :-



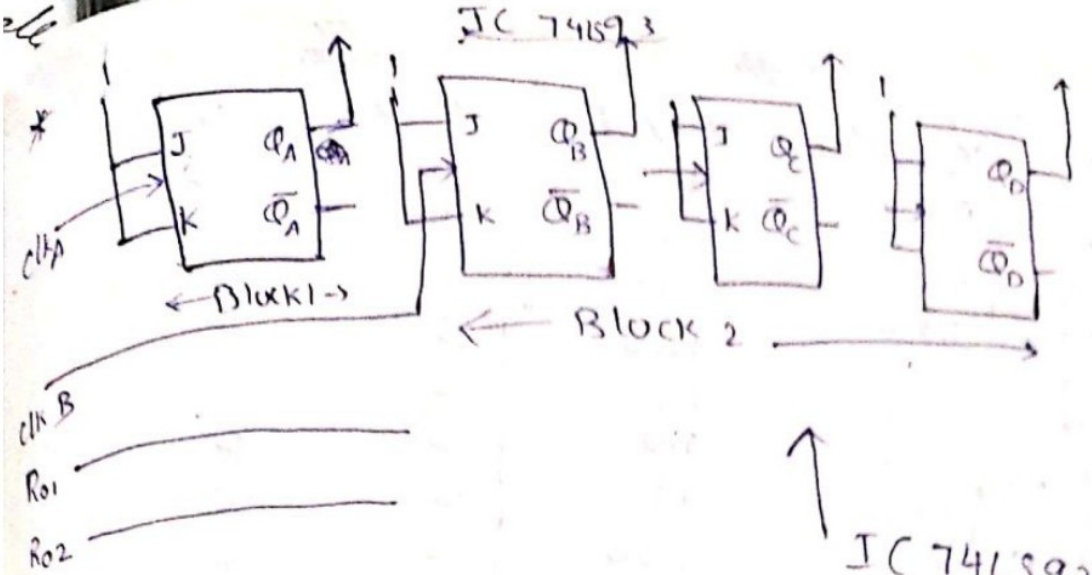
clk	Q_3	Q_2	Q_1
	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

→ set Q_1, Q_2, Q_3 to 000

* Cascading mod N_1 & mod N_2 ~~give~~ counters gives mod $(N_1 N_2)$.

* TTL 7493 counter can count mod 2, mod 8, mod 10, mod 12, mod 14, mod 16.
→ principle is cascading

Scanned by CamScanner



IC 74LS93

mod 2 → use
clk A, R_{01}, R_{02} - grounded

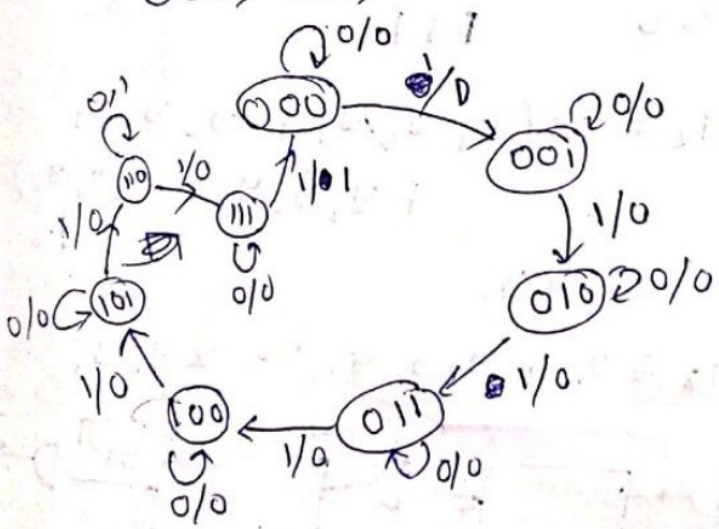
mod 8
clk B, R_{01}, R_{02} - grounded
↓use

*Synchronous Counters:

Binary

Mod 8 counter

000, 001, ... , 111.



'1' is output after every 8 clock pulses.

Scanned by CamScanner

Count Comerion ... Z

PS $y_3 y_2 y_1$	NS $x=0$ $Y_3 Y_2 Y_1$	NS $x=1$ $Y_3 Y_2 Y_1$	$x=0$	$x=1$
000	000	001	0	0
001	001	010	0	0
010	010	011	0	0
011	011	100	0	0
100	100	101	0	0
101	101	110	0	0
110	110	111	0	0
111	111	000	0	1

$$Z = x y_3 y_2 y_1$$

* Excitation

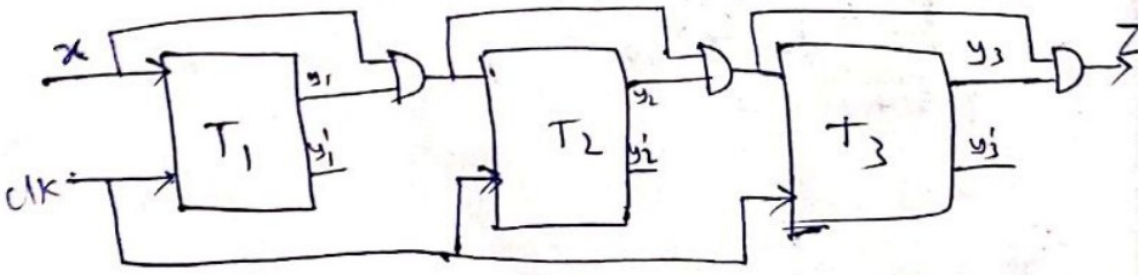
T - flip flop

PS $y_3 y_2 y_1$	$T_3 T_2 T_1$ $x=0$	$T_3 T_2 T_1$ $x=1$
000	000	001
001	000	011
010	000	001
011	000	111
100	000	001
101	000	011
110	000	001
111	000	011

$$\therefore T_1 = x \quad , \quad T_2 = x(\bar{y}_3 \bar{y}_2 y_1 + \bar{y}_3 y_2 y_1 + y_3 \bar{y}_2 y_1 + y_3 y_2 y_1)$$

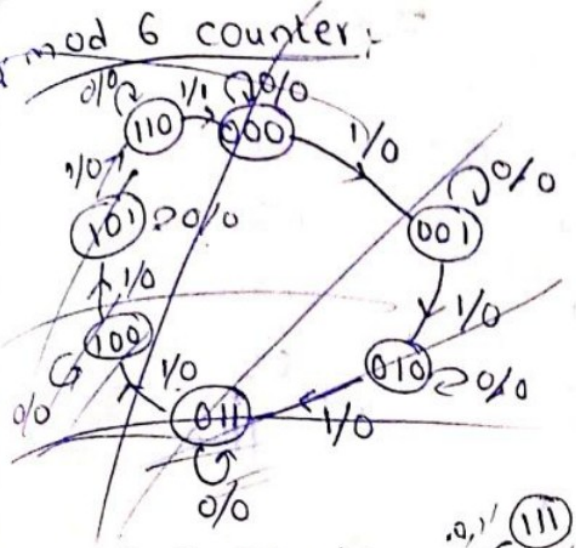
$$T_2 = x y_1$$

$$T_3 = x(\bar{y}_2 y_2 y_1 + y_3 y_2 y_1) = x y_2 y_1$$

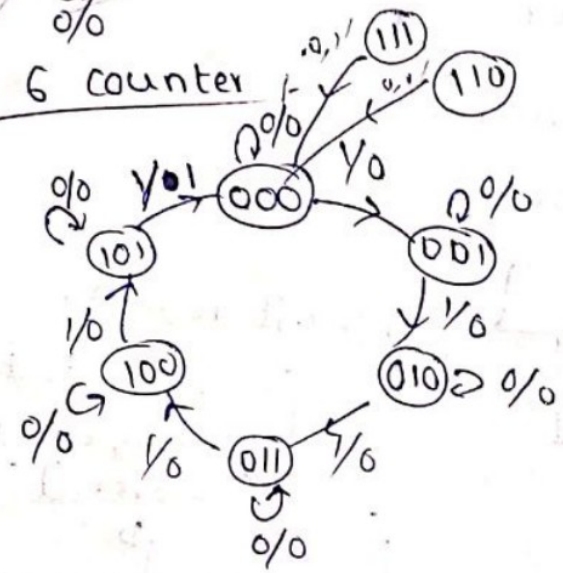


Scanned by CamScanner

* mod 6 counter :-

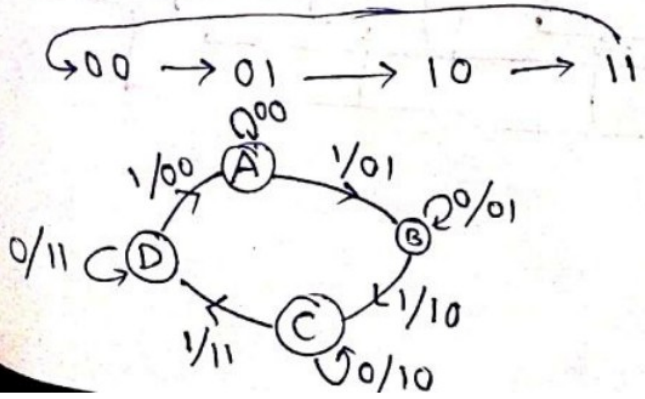


* mod 6 counter



PS	NS x=0	NS x=1
000		
001		
010		
011		
100		
101		
110	000	000
111	000	000

* 2-bit up counter :- (1-input → 2-output)



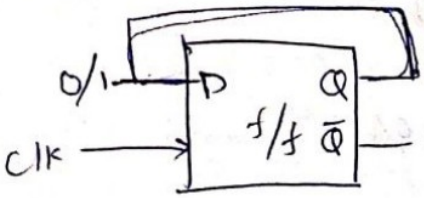
Scanned by CamScanner

PS	NS $x=0$	NS $x=1$	$Z_1 Z_2$ $x=0$	$Z_1 Z_2$ $x=1$
00 A	A 00	B 01	00	01
01 B	B 01	C 10	01	10
10 C	C 10	D 11	10	11
11 D	D 11	A 00	11	00

* shift register :-

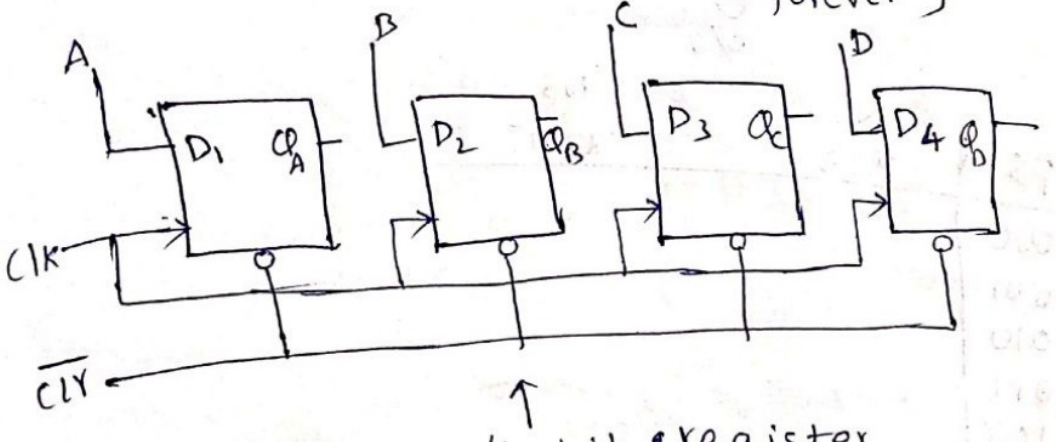
→ storage (Memory)

→ Data Transfers.

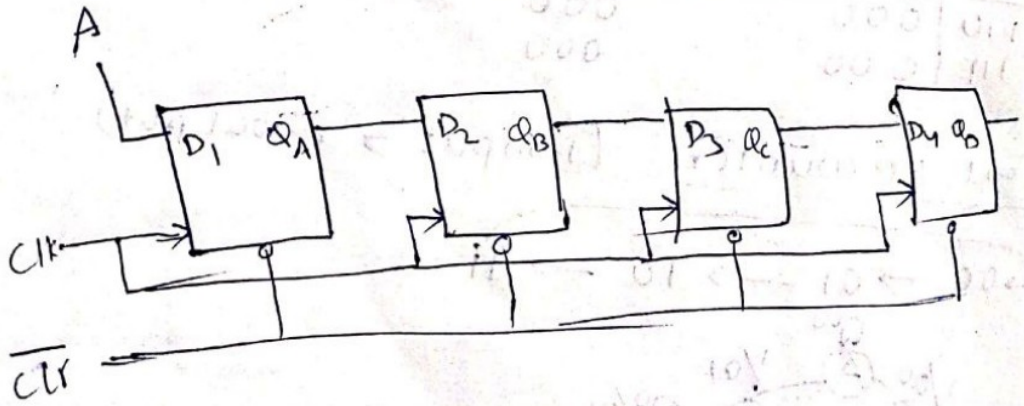


→ 1-bit register

[∵ Q = D is transferred along the line forever]



↑ 4-bit register



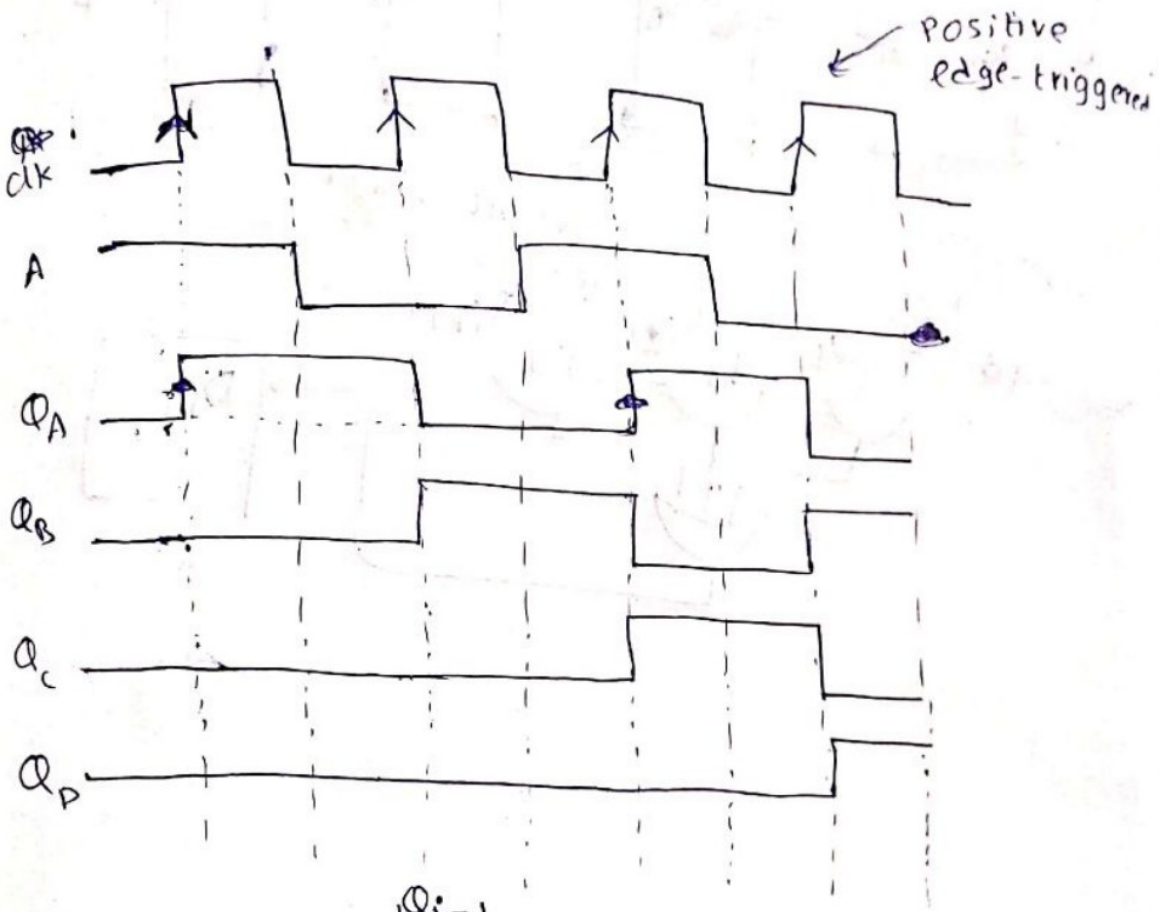
Scanned by CamScanner

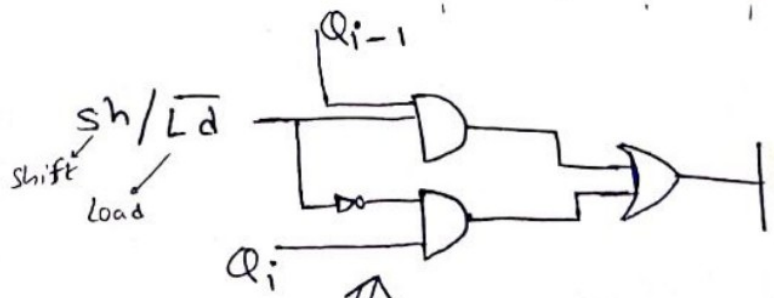
A. 1010

Clk	A	Q_A	Q_B	Q_C	Q_D
0	1	0	0	0	0
1	0	1	0	0	0
2	1	0	1	0	0
3	0	1	0	1	0
4		0	1	0	1

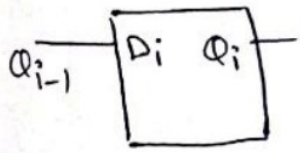
clock pulse number

Right shift

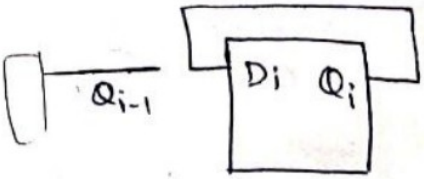




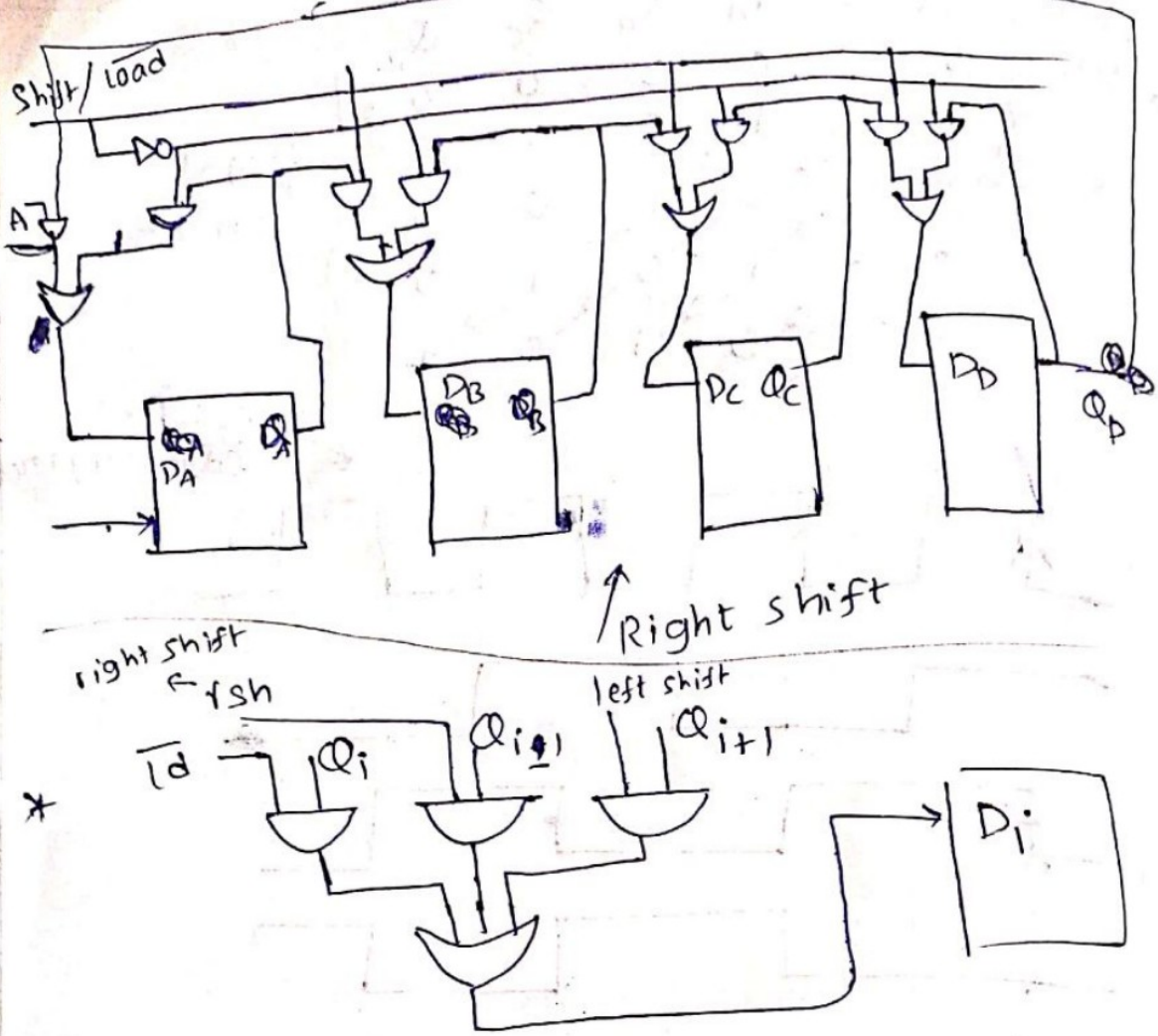
Shift (sh)



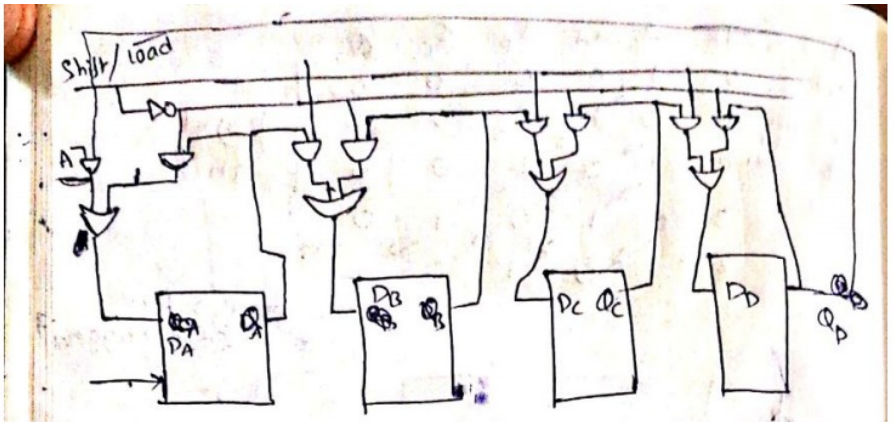
Load (ld)

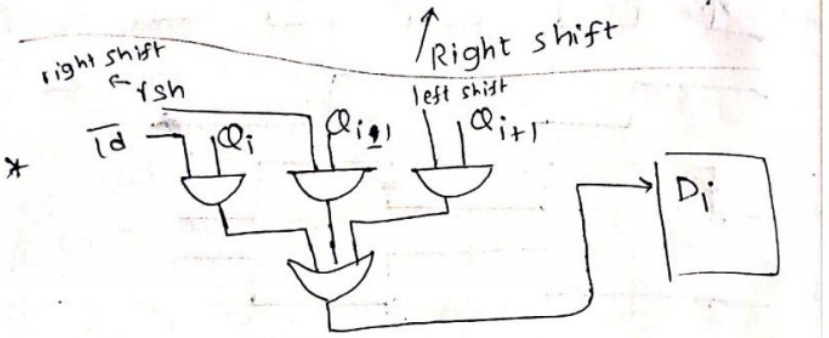


Scanned by CamScanner

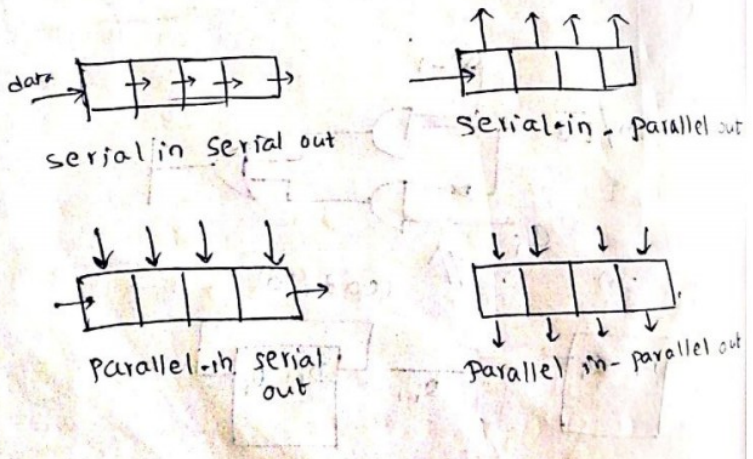


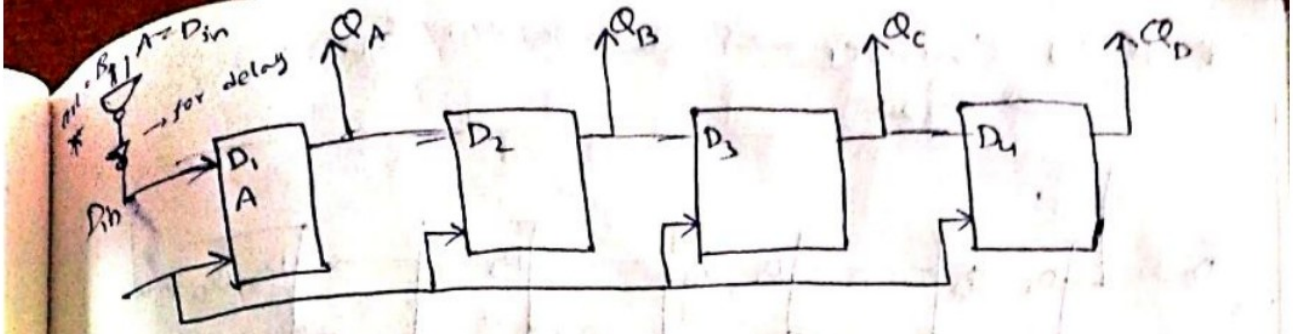
Scanned by CamScanner





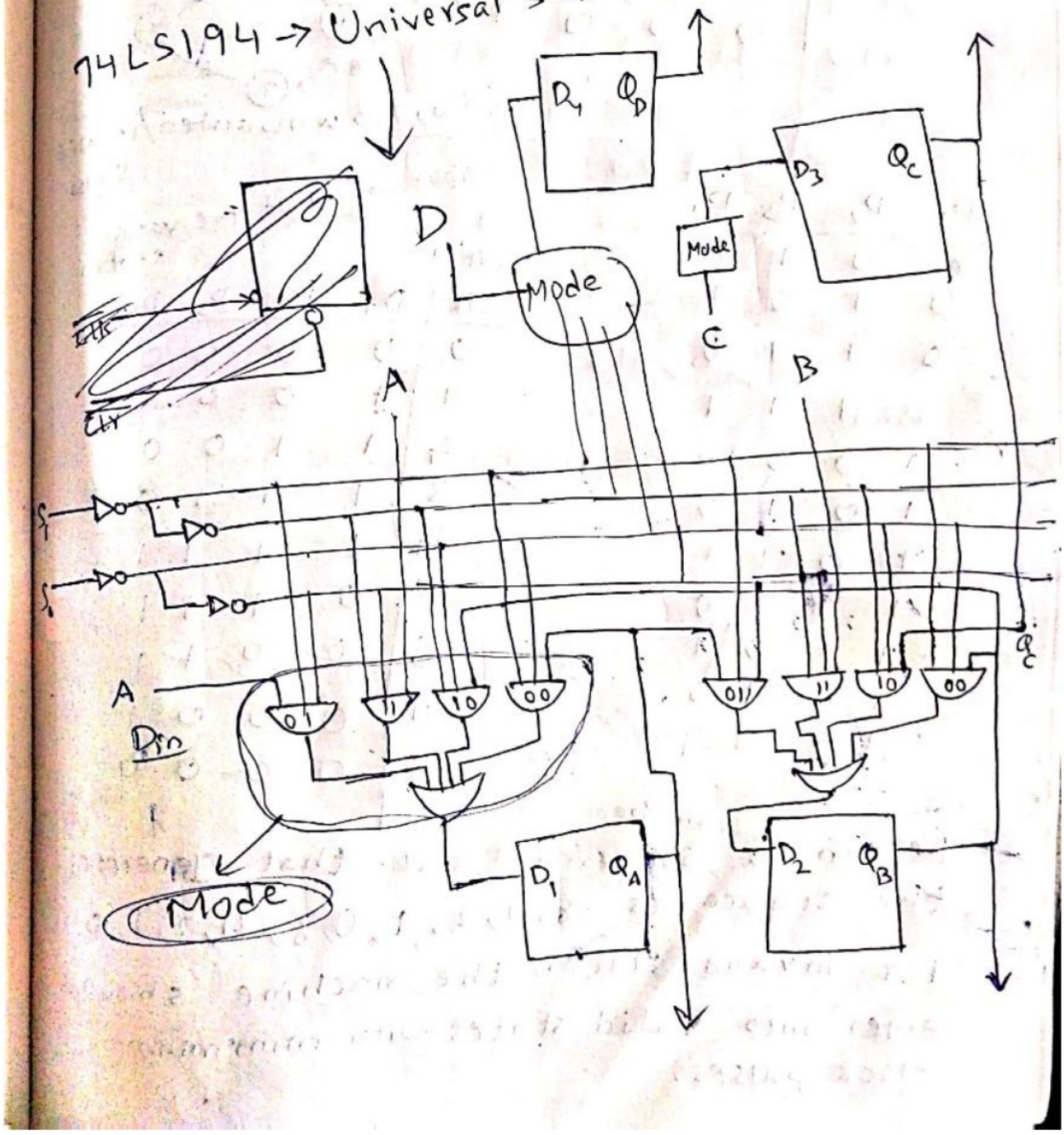
* Uses of shift Register in Data Transmission





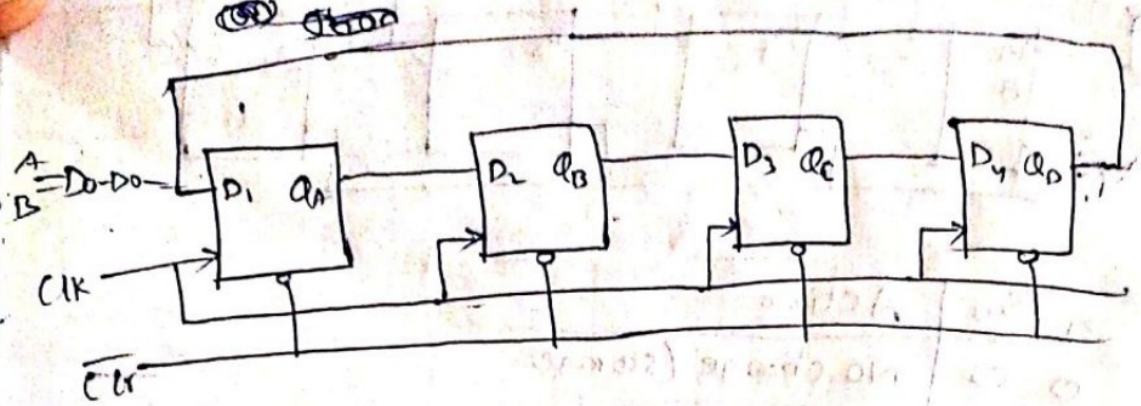
S_1	S_0	Action
0	0	No change (storage)
0	1	right shift
1	0	left shif
1	1	parallel data-in (paralled load-in)

74LS194 → Universal shift Register



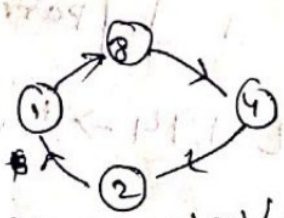
Scanned by CamScanner

* Ring Counter → feed back for Q_D



D_1	D_2	D_3	D_4
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

$f = \Sigma(1, 2, 4, 8)$



3, 5, 6, 7 → unwanted/invalid states

reset to any of the valid states

D_1	D_2	D_3	D_4
0	0	1	1
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

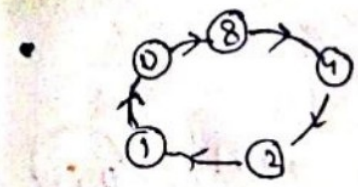
Input = 1

CLK	D_1	D_2	D_3	D_4
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
8	0	0	0	0

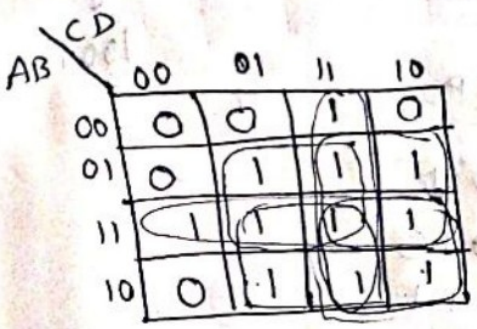
Q. Design a synchronous sequential ckt. that generates the sequences 8, 4, 2, 1, 0, 8, 4, 2, 1, 0

& For invalid states, the machine should enter into valid states with minimum clock pulses.

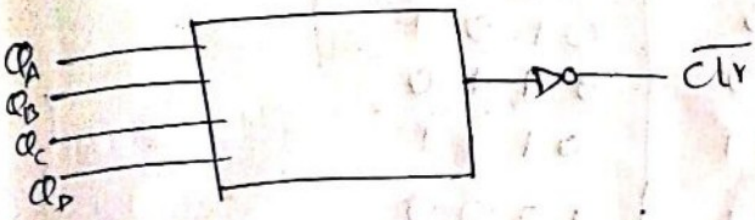
Scanned by CamScanner



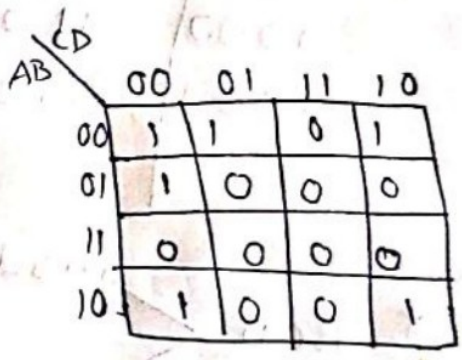
3, 5, 6, 7, 9 – 15



$f = AB + CD + BD + BC + AC + AD$ ....



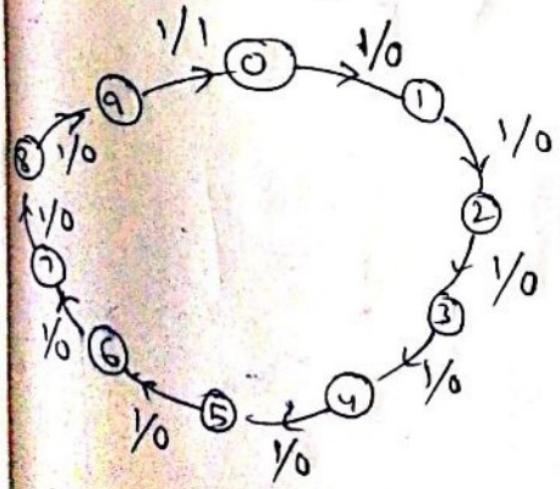
PS	NS	D f/f
0000	1000	1000
1000	0100	0100
0100	0010	0010
0010	0001	0001
0001	0000	0000



BCD counter

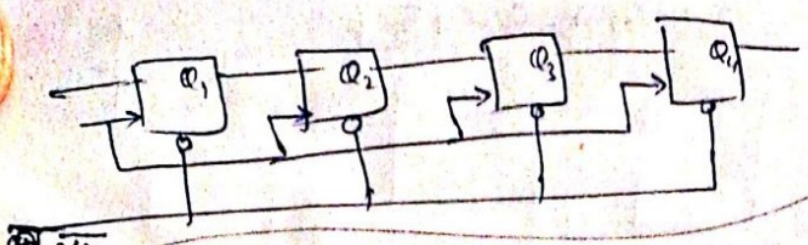
0 – 9 valid states

10 – 15 < Assume that these inputs never appear
it ~~state~~ should enter into valid states



PS	NS	NS (BCD)
0000	0001	1000
1001	0010	10001
:	:	:
1000	1001	
1001	1010	

Scanned by CamScanner



0010 → 0011 → 1001

PS	NS Johnson	NS BCD
0000	1000	0001
0001	0000	0010
0010	1001	0011
0011	0001	0100
0100	1010	0101
0101	0010	0110
0110	1011	0111
0111	0011	1000
1000	1100	1001
1001	0100	0000

* BCD
  - modulo counter
  - shift register (right shift)
    - ↓ HW

Direct Logic (Detecting 1001)

→ Identify the repetitive sequence

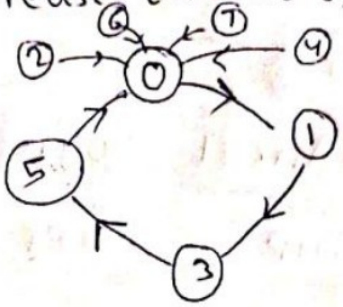
→ Identify the minimum(m) no. of flipflops.
(eg: for 0111, we need 2 f/fs)

→ Do the unique state assignment with min no. of f/fs

→ If it is not possible, increase the no. of f/fs.

0 0
0 1
1 1
1 0

0 0 0
0 0 1
0 1 1
1 0 1



PS Q_3 Q_2 Q_1	NS	J_3 K_3	J_2 K_2	J_1 K_1
0 0 0	0 0 1	0 X	0 X	1 X
0 0 1	0 1 1	0 X	1 X	X 0
0 1 1	1 0 1	1 X	X 1	X 0
1 0 1	0 0 0	X 1	0 X	X 1
0 1 0	0 0 0	X 1	X 1	0 X
0 1 0	0 0 0	X 1	X 1	0 X
1 1 1	0 0 0	X 1	X 1	X 1
1 0 0	0 0 0	X 1	0 X	0 X

$\overline{CLR}$

J_3

Q_3 \ Q_2Q_1	00	01	11	10
0	0	0	1	0
1	X	X	X	X

$K_3 = 1$

$K_2 = 1$

$K_1 = Q_3$

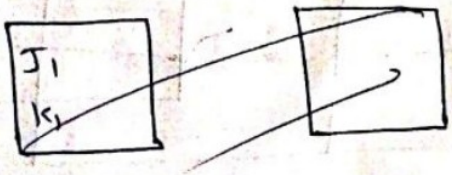
Only from the first half of table

$J_3 = \overline{Q}_3 Q_2 Q_1$

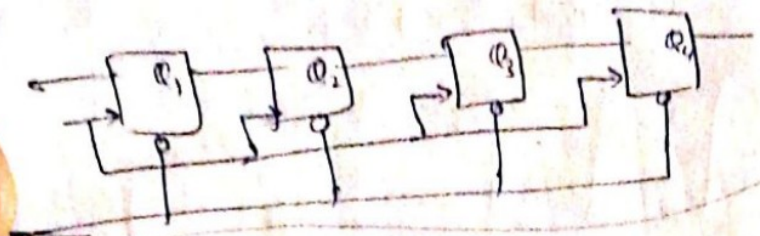
$J_2 = \overline{Q}_3 \overline{Q}_2 Q_1 \,/\, \overline{Q}_3 Q_1$

$J_1 = \overline{Q}_3 \overline{Q}_2 \overline{Q}_1 \,/\, \overline{Q}_3$

By considering Bottom half also



Scanned by CamScanner

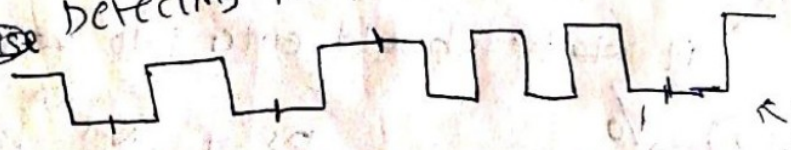


0010 → 0011 → 1001

PS	NS Johnson	NS BCD
0000	1000	0001
0001	0000	0010
0010	1001	0011
0011	0001	0100
0100	1010	0101
0101	0010	0110
0110	1011	0111
0111	0011	1000
1000	1100	1001
1001	0100	0000

* BCD
  - mod10 counter
  - shift register (right shift)
    - ↳ HW

* Detecting the pattern 1001 (may be overlapping).



by pulse train Generator.

* Pulse Train Generator
  1. Direct Logic
  2. Indirect Logic.

Scanned by CamScanner

Direct Logic (Detecting 1001)

→ Identify the repetitive sequence

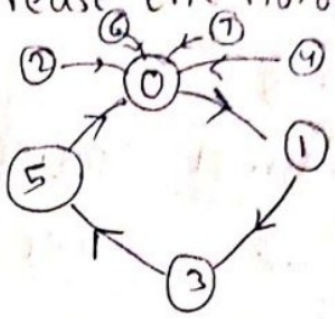
→ Identify the minimum(m) no. of flipflops.
(eg: for 0111, we need 2 f/fs)

→ Do the unique state assignment with min no. of f/fs

→ If it is not possible, increase the no. of f/fs

0 0
0 1
1 1
1 0 1

0 0 0
0 0 1
0 1 1
1 0 1



PS $Q_3 Q_2 Q_1$	NS	$J_3 K_3$	$J_2 K_2$	$J_1 K_1$
0 0 0	0 0 1	0 X	0 X	1 X
0 0 1	0 1 1	0 X	1 X	X 0
0 1 1	1 0 1	1 X	X 1	X 0
1 0 1	0 0 0	X 1	0 X	X 1
0 1 0	0 0 0	X 1	X 1	0 X
0 1 1 0	0 0 0	X 1	X 1	0 X
1 1 1	0 0 0	X 1	X 1	X 1
1 0 0	0 0 0	X 1	0 X	0 X

CLr

J_3

Q_3 \ Q_2Q_1	00	01	11	10
0	0	0	1	0
1	X	X	X	X

$K_3 = 1$

$K_2 = 1$

$K_1 = Q_3$

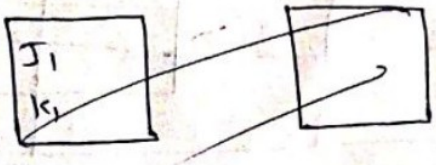
Only from the first half of table

$J_3 = \bar{Q}_3 Q_2 Q_1$

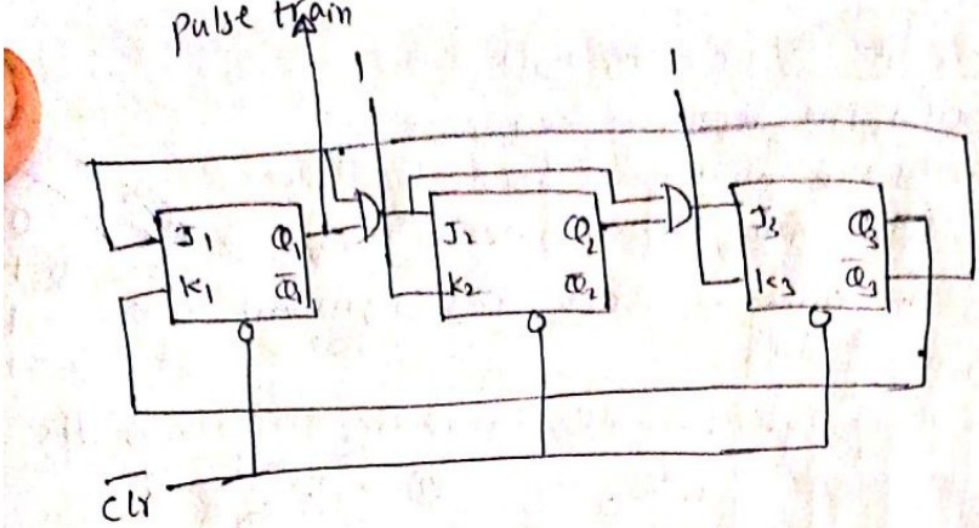
$J_2 = \bar{Q}_3 \bar{Q}_2 Q_1 \,/\, \bar{Q}_3 Q_1$

$J_1 = \bar{Q}_3 \bar{Q}_2 \bar{Q}_1 \,/\, \bar{Q}_3$

By considering Bottom half also



Scanned by CamScanner



Indirect Logic :-

→ chip - counter (Using counter) :-

10110 10110 10110 . . . . .

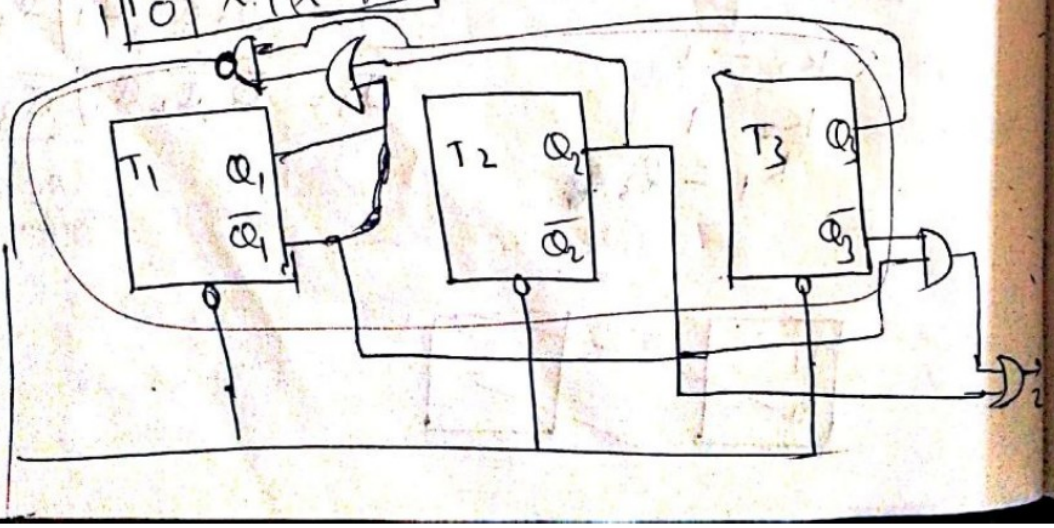
5-bit => atleast 3-bit counter is required (mod-5)

Q_3 Q_2 Q_1	Z
0 0 0	1
0 0 1	0
0 1 0	1
0 1 1	1
1 0 0	0
1 0 1	X
1 1 0	X
1 1 1	X

(invalid states)

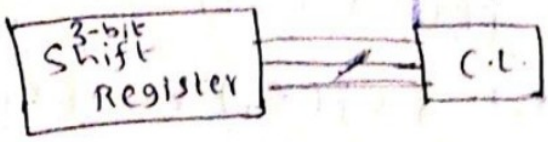
Q_3 \ Q_2Q_1	00	01	11	10
0	1	0	1	1
1	0	X	X	X

$$Z = Q_2 + \bar{Q}_3 \bar{Q}_1$$

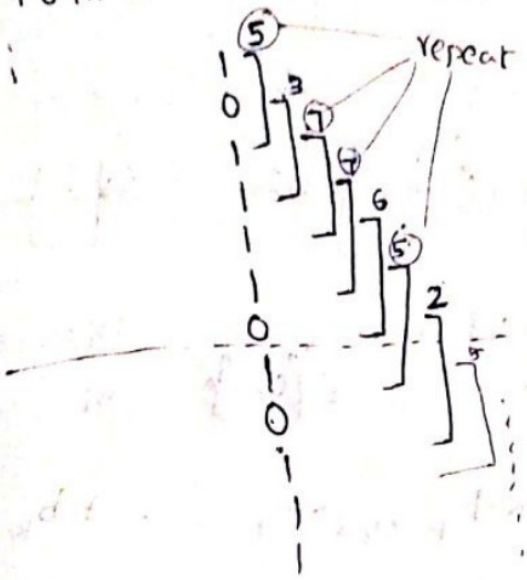


Scanned by CamScanner

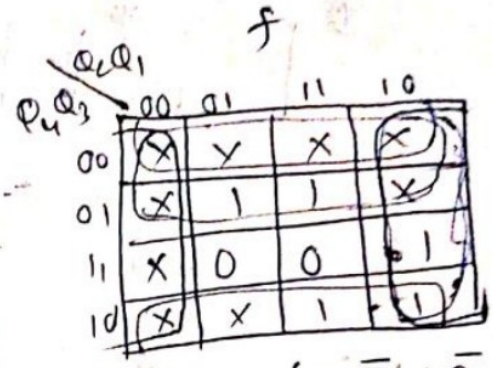
→ Using Shift Register :-



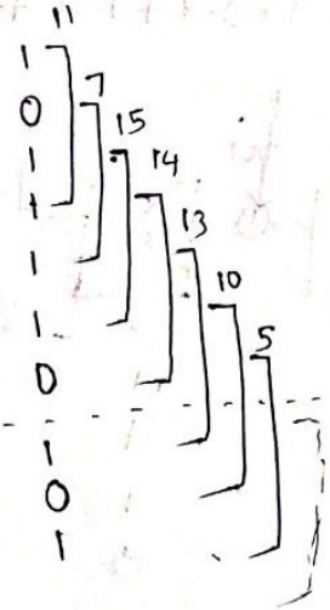
1011110 1011110 1011110 .....



repeat ⇒ we need 4 bit shift register



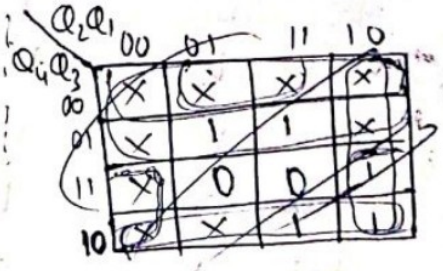
$f = \bar{Q}_1 + \bar{Q}_4 + \bar{Q}_3$

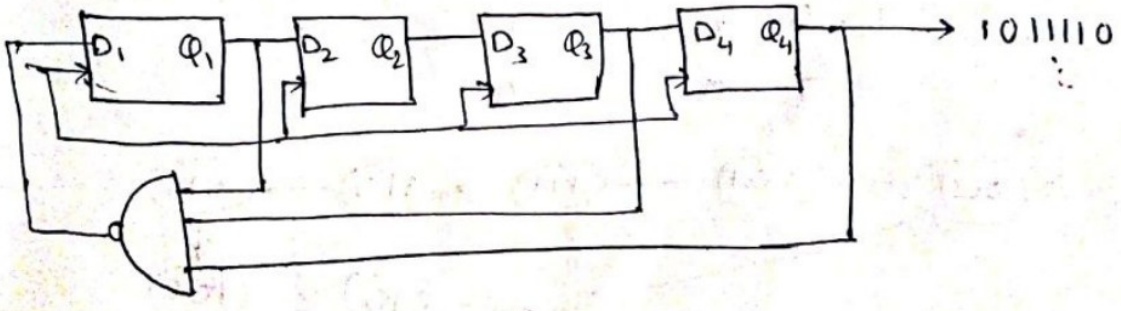


Q_4	Q_3	Q_2	Q_1	f
1	0	1	1	1
0	1	1	1	1
1	1	1	1	0
1	1	1	0	1
1	1	0	1	0
1	0	1	0	1
0	1	0	1	1

1011

$f = \overline{Q_1 Q_3 Q_4}$





Scanned by CamScanner

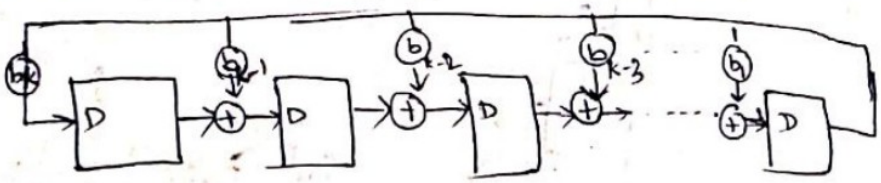
Linear Feedback Shift Register (LFSR)

$f(x_1 + x_2) = f(x_1) + f(x_2)$ → linear function

Ex: XOR

$f(ax_1 + bx_2) = af(x_1) + bf(x_2)$

⊕ - XOR



k-stage LFSR.

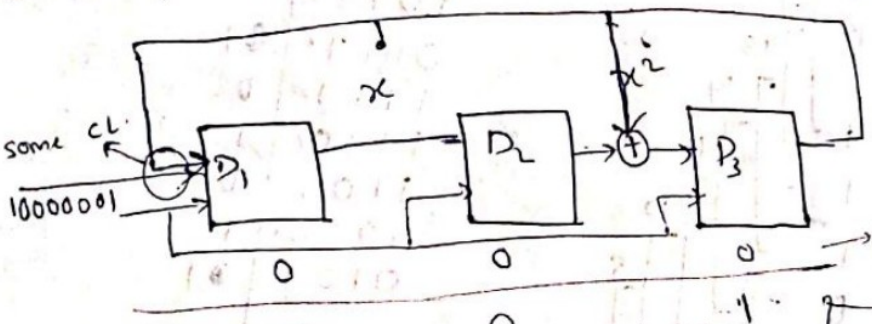
$b_1, b_2, \ldots \ldots b_k \in \{0, 1\}$

$b_i = 0 \Rightarrow$ open switch

$b_i = 1 \Rightarrow$ closed switch

LFSR $p(x) = x^k + b_1 x^{k-1} + b_2 x^{k-2} + \ldots\ldots + b_k$.

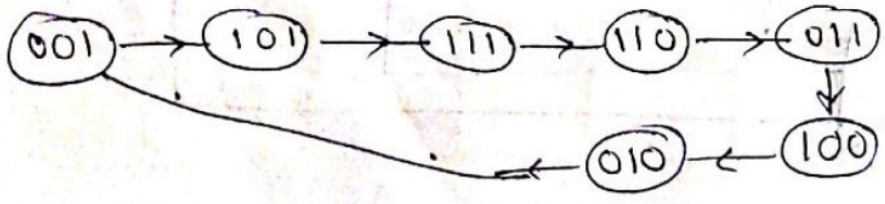
Ex. $p(x) = 1 + x^2 + x^3 = 1 + 0 \cdot x + 1 \cdot x^2 + 1 \cdot x^3$



→ start with 000 ⇒ remain there only

→ start with 001

Clk=0	0	0	1
Clk=1	1	0	1
Clk=2	1	1	1
Clk=3	1	1	0
clk=4	0	1	1
Clk=5	1	0	0
Clk=6	0	1	0
Clk=7	0	0	1



Scanned by CamScanner

→ 7 states (2^3-1) are obtained but for $1+x^3$, these many states are not obtained.

→ max. states for primitive polynomial.

→ A k-degree irreducible polynomial is primitive if it divides x^m+1 when $m = 2^k-1$

(irreducible: which has no factors other than 1 & itself)

$P(x) = 1+x^2+x^3$

$m = 2^3-1 = 7.$

$$1+x^2+x^3 \;\big)\; x^7+1 \;\big(\; x^4-x^3+x^2-2x+1$$

$$x^7+x^6+x^4$$

$$-x^6-x^4$$

$$-x^6-x^5-x^3$$

$$x^5-x^4+x^3$$

$$x^5+x^4+x^2$$

$$-2x^4+x^3-x^2$$

$$-2x^4-2x^3-2x$$

$$x^3-x^2+2x+1$$

$$x^3+x^2+1$$

$$-2x^2+2x$$

$1+x^2+x^3 = 1+0\cdot x+1\cdot x^2+1\cdot x^3 = 1101$

$1+x^7 = x^7+1 = 10000001$

Scanned by CamScanner

→ 7 states (2^3-1) are obtained but for $1+x^3$, these many states are not obtained.

- max: States for primitive polynomial.

→ A k-degree irreducible polynomial is primitive if it divides x^m+1 when $m = 2^k-1$ (which has no factors other than 1 & itself)

$P(x) = 1+x^2+x^3$

$m = 2^3-1 = 7$

$$\begin{array}{r|l} 1+x^2+x^3 \;)\; x^7+1 & x^4-x^3+x^2-2x+1 \\ \underline{-x^7+x^6+x^4} & \\ -x^6-x^4 & \\ \underline{+x^6+x^5+x^3} & \\ x^5-x^4+x^3 & \\ \underline{+x^5+x^4+x^2} & \\ -2x^4+x^3-x^2 & \\ \underline{+2x^4+2x^3+2x} & \\ x^3-x^2+2x+1 & \\ \underline{+x^3+x^2+x} & \\ -2x^2+2x & \end{array}$$

Not normal division. It's Binary division

$x^7+x^7 = (1+1)x^7 = 0x^7 = 0$

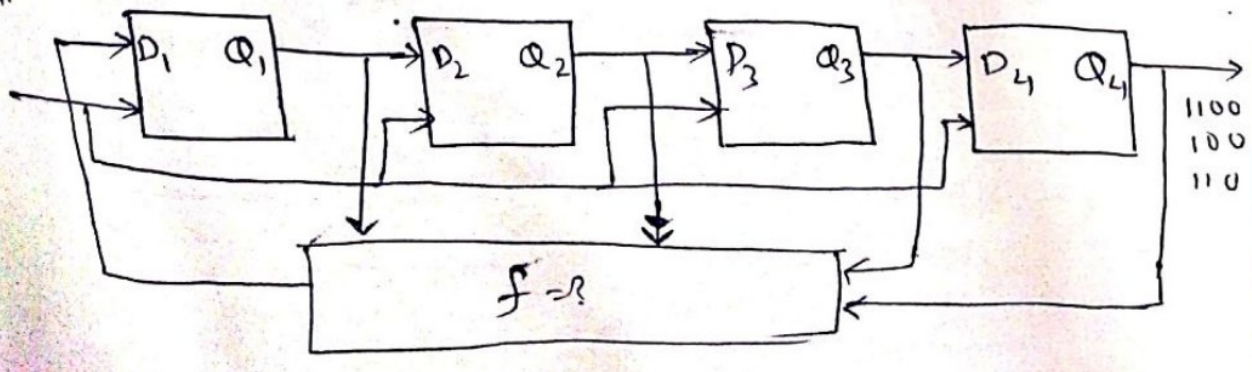
$(1+0)x^7 = 1\cdot x^7$

$(0+0)x^7 = 0x^7$

$1+x^2+x^3 = 1+0\cdot x+1\cdot x^2+1\cdot x^3 = 1101$

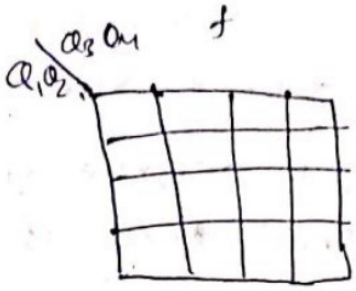
$1+x^7 = x^7+1 = 10000001$

*



Scanned by CamScanner

Q_1	Q_2	Q_3	Q_4	f
0	0	1	1	1
1	0	0	1	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	1
1	0	0	0	1
0	1	0	0	1
1	1	1	0	1
1	0	1	1	0
1	1	0	1	0
0	1	1	1	0
0	0	1	1	1



1100 | 00111

* To get 000 in 3-bit LFSR along with other 7-patterns.

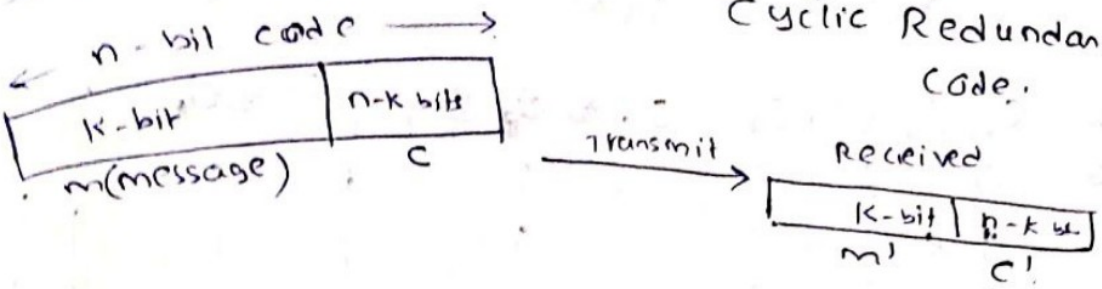
1 1 0
0 0 1
1 0 1
1 1 0 1
0 1 0
0 1 1
1 0 0

*HW Insert 0000 to LFSR (4-bit)

Scanned by CamScanner

Uses of LFSR to generate CRC code

CRC → Cyclic Redundancy Code.

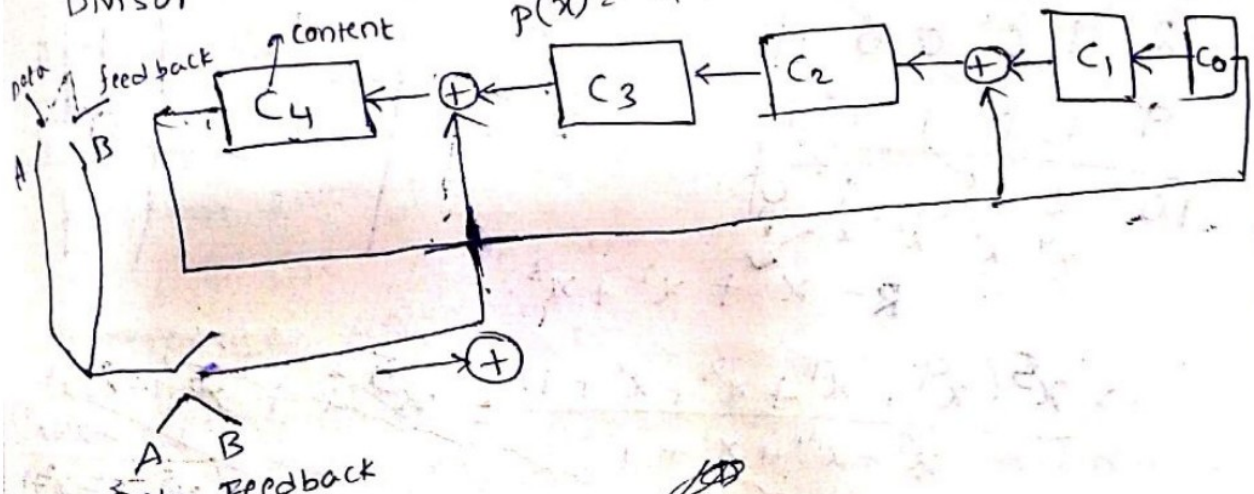


Ex: $D(x) = x^9 + x^7 + x^3 + x^2 + 1 \Rightarrow 1010001101$

predetermined $P(x) = x^5 + x^4 + x^2 + 1 \Rightarrow 110101$

$$\frac{x^{n-k} \cdot D(x)}{P(x)} = Q(x) + \frac{R(x)}{P(x)}$$

Dividend ← $x^{n-k} \cdot D(x)$; Remainder ← $R(x)$; Divisor ← $P(x)$; Quotient ← $Q(x)$



$$x^5 + x^4 + x^2 + 1 \;\big)\; x^9 + x^7 + x^3 + x^2 + 1 \;\big(\; x^4 + x^3 + x$$

$x^9 + x^8 + x^6 + x^4$

$1 + 1 = 0$

$x^8 + x^7 + x^6 + x^4 + x^3 + x^2 + 1$

$x^8 + x^7 + x^5 + x^3$

$x^6 + x^5 + x^4 + x^2 + 1$

$x^6 + x^5 + x^3 + x$

$x^4 + x^3 + x^2 + x + 1$

Scanned by CamScanner

for degree polynomial

10-bits	5-bits

$\Rightarrow n = 15, k = 10$

Remainder of degree 4

$$\frac{x^{n-k} D(x)}{P(x)} = \frac{x^5 \cdot D(x)}{P(x)}$$

next clk feedback.

clk	C_4	C_3	C_2	C_1	C_0	$C_4 \oplus d \oplus C_3$	$C_4 \oplus d \oplus C_1$	$C_4 \oplus d$	d
0	0	0	0	0	0	1	1	1	1
1	1	0	1	0	1	1	1	1	0
2	1	1	1	1	1	1	1	0	1
3	1	1	1	1	0	0	0	1	0
4	0	1	0	0	1	1	0	0	0
5	1	0	0	1	0	1	0	1	0
6	1	0	0	0	1	0	0	0	1
7	0	0	0	1	0	1	0	1	1
8	1	0	0	0	1	1	1	1	0
9	1	0	1	1	1	0	1	0	1
10	0	1	1	1	0				

$x^4\ x^3\ x^2\ x^1\ x^0$

$R = x + x^2 + x^3.$

$$\Rightarrow \frac{x^5(x^9 + x^7 + x^3 + x^2 + 1)}{x^5 + x^4 + x^2 + 1} = x^5 Q + \frac{x^5(1 + x + x^2 + x^3 + x^4)}{x^5 + x^4 + x^2 + 1}$$

$$= \frac{x^{14} + x^{12} + x^8 + x^7 + x^5}{x^5 + x^4 + x^2 + 1}$$

$$= x^5 \left(1 + \cdots\right)$$

$x^5 + x^4 + x^2 + 1 \;\big)\; x^{14} + x^{12} + x^8 + x^7 + x^5 \;\big(\; x^9 + x^8 + x^6 + x^4 + x^2 + x$

$x^{14} + x^{13} + x^{11} + x^9$

$x^{13} + x^{12} + x^{11} + x^9 + x^8$

$x^{13} + x^{12} + x^{10} + x^8$

$x^{11} + x^{10} + x^9 + x^7$

$x^{11} + x^{10} + x^8 + x^6$

$x^9 + x^8 + x^7 + x^6 + x^5$

$x^9 + x^8 + x^6 + x^4$

$x^7 + x^5 + x^4$

$x^7 + x^6 + x^4 + x^2$

$x^6 + x^5 + x^2$

$x^6 + x^5 + x^3 + x$

Scanned by CamScanner

for degree k polynomial

10-bits	5-bits

Remainder of degree 4.

$\Rightarrow n = 15, k = 10$

$$\frac{x^{n-k} D(x)}{P(x)} = \frac{x^5 \cdot D(x)}{P(x)}$$

next clk feedback.

clk	$C_4 C_3 C_2 C_1 C_0$	$C_4 \oplus d \oplus C_3$	$C_4 \oplus d \oplus C_1$	$C_4 \oplus d$	d
0	0 0 0 0 0	1	1	1	1
1	1 0 1 0 1	1	1	1	0
2	1 1 1 1 1	1	1	0	1
3	1 1 1 1 0	0	0	1	0
4	0 1 0 0 1	1	0	0	0
5	1 0 0 1 0	1	0	1	0
6	1 0 0 0 1	0	0	0	1
7	0 0 0 1 0	1	0	1	1
8	1 0 0 0 1	1	1	1	0
9	1 0 1 1 1	0	1	0	1
10	0 1 1 1 0				

$x^4\ x^3\ x^2\ x^1\ x^0$

$R = x + x^2 + x^3.$

$$\Rightarrow \frac{x^5(x^9 + x^7 + x^3 + x^2 + 1)}{x^5 + x^4 + x^2 + 1} = x^5 Q + \frac{x^5(1 + x + x^2 + x^3 + x^4)}{x^5 + x^4 + x^2 + 1}$$

$$= \frac{x^{14} + x^{12} + x^8 + x^7 + x^5}{x^5 + x^4 + x^2 + 1} = x^5 \left(1 + \cdots\right)$$

$$x^5 + x^4 + x^2 + 1 \;\Big)\; x^{14} + x^{12} + x^8 + x^7 + x^5 \;\Big(\; x^9 + x^8 + x^6 + x^4 + x^2 + x$$

$$x^{14} + x^{13} + x^{11} + x^9$$

$$x^{13} + x^{12} + x^{11} + x^9 + x^8$$

$$x^{13} + x^{12} + x^{10} + x^8$$

$$x^{11} + x^{10} + x^9 + x^7$$

$$x^{11} + x^{10} + x^8 + x^6$$

$$x^9 + x^8 + x^7 + x^6 + x^5$$

$$x^9 + x^8 + x^6 + x^4$$

$$x^7 + x^5 + x^4$$

$$x^7 + x^6 + x^4 + x^2$$

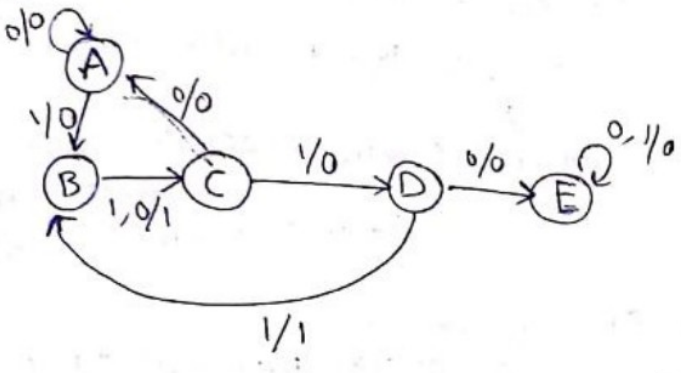
$$x^6 + x^5 + x^2$$

$$x^6 + x^5 + x^3 + x$$

$$x^3 + x^2 + x$$

Scanned by CamScanner

* Capabilities & limitations of Synchronous Sequential machine :-



no other states are accessible from the sink ← Sink : E → Terminal states

Source

~~No other~~ state which is not accessible from any other state.

S_i & S_j are two states of 'M'.
if applying input-sequence 'X', S_j is reached from S_i, then S_j is called X-successor of S_i.

* Strongly connected Machines :-

If for each pair of (S_i, S_j) of M/c 'M', if S_j is the X-successor of S_i, then M is strongly connected.

* Limitations

→ Apply P no. of 1's, 1111.....1, P > n (n: no. of states)

Limitation ⇒ By pigeonhole principle, At least one state repeats.
finite memory, we can design the m/c.

Ex:- Design a m/c that generates a '1', when the no. of 1's applies is $\frac{K(K+1)}{2}$, K = 1, 2, 3, ...

{ Input : 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
output : 1 0 1 0 0 1 0 0 0 1 0 0 0 1

output is not repetitive ⇒ not possible to create the m/c.
(i.e no repetitive state).

Scanned by CamScanner

*Multiplication → N_1, N_2 given, $N_1 \times N_2$ is to be found.
If N_1, N_2 are finite numbers,
I can get $N_1 \times N_2$ with finite memory
with ripple carry adder we can find product.
(because input is finite)

*Serial Multiplier (synchronous)
Assume that a sequential multiplier can be designed for a serial multiplier.
Let $N_1 = N_2 = 2^p$ → Binary → 10000.... (p no.of zeroes) → p+1 bits.
⇒ product = 2^{2p} → 1000..... (2p no.of zeroes) → 2p+1 bits

Input :

Clk Pulse → (2p+1) ..... (p+2)	(p+1) p 4 3 2 1
	1 0 ..... 0 0 0 0 → N_1
×	1 0 ..... 0 0 0 0 → N_2
(1) 0 0 0 0	0 0 0 0 ..... 0 0 0 0 → $N_1 \times N_2$

1 is not possible ← should repeat ← zerostate

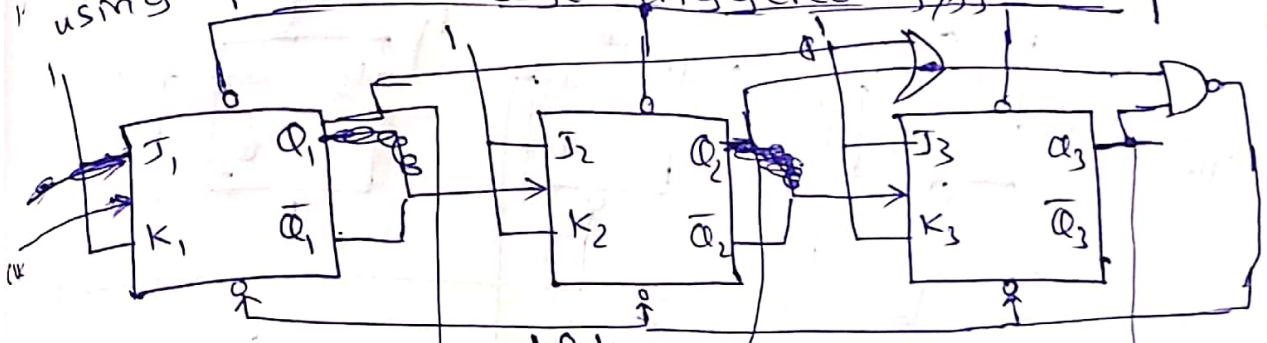
Characteristics of synchronous sequential M/c
zero state repeats ⇒ It can never generate '1'.

⇒ Serial Multipler is not possible to design.

* Multiplication of two finite no-s with finite memory elements is possible (i.e inputs are given at first, not serially)
But multiplication of serial inputs is not possible

Scanned by CamScanner

1. Design an asynchronous mod 5 upcounter by using positive edge triggered f/f's



- invalid states = 101, 110, 111

$= Q_3 Q_2 Q_1' + Q_3 Q_2 Q_1 + Q_3 Q_2' Q_1$

$= Q_3 Q_2 + Q_3 Q_2' Q_1$

$= Q_3 (Q_2 + Q_1)$

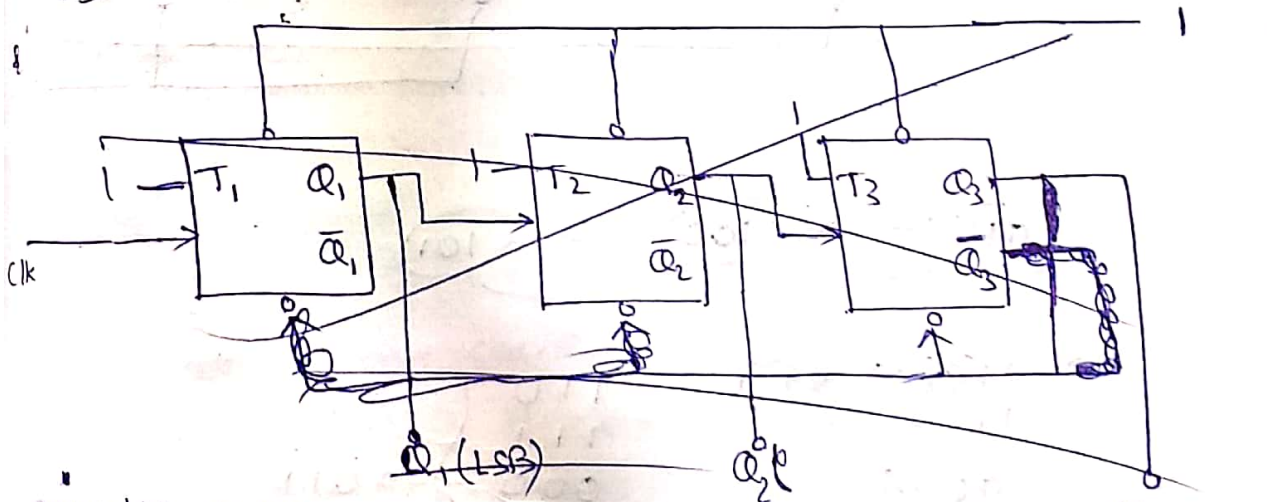
Number → $Q_3 Q_2 Q_1$

Q_3 (MSB)

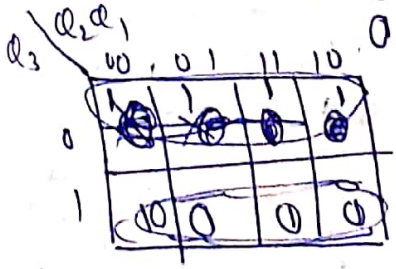
Q_2

(LSB) Q_1

2. Design a 3-bit asynchronous down mod4 counter. by using +ve. edge triggered T f/f's.

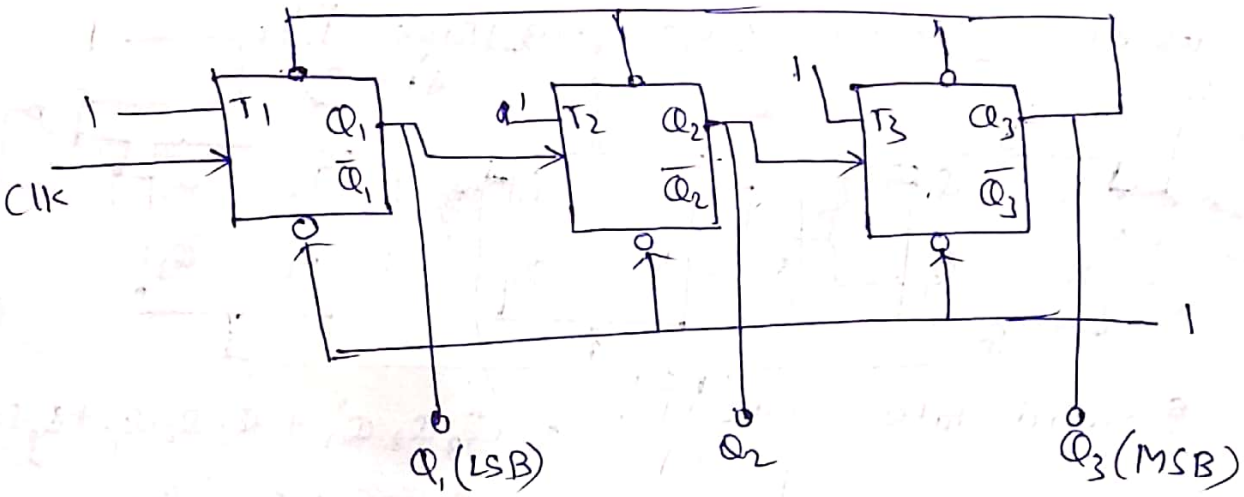


Invalid states = 000, 001, 010, 011

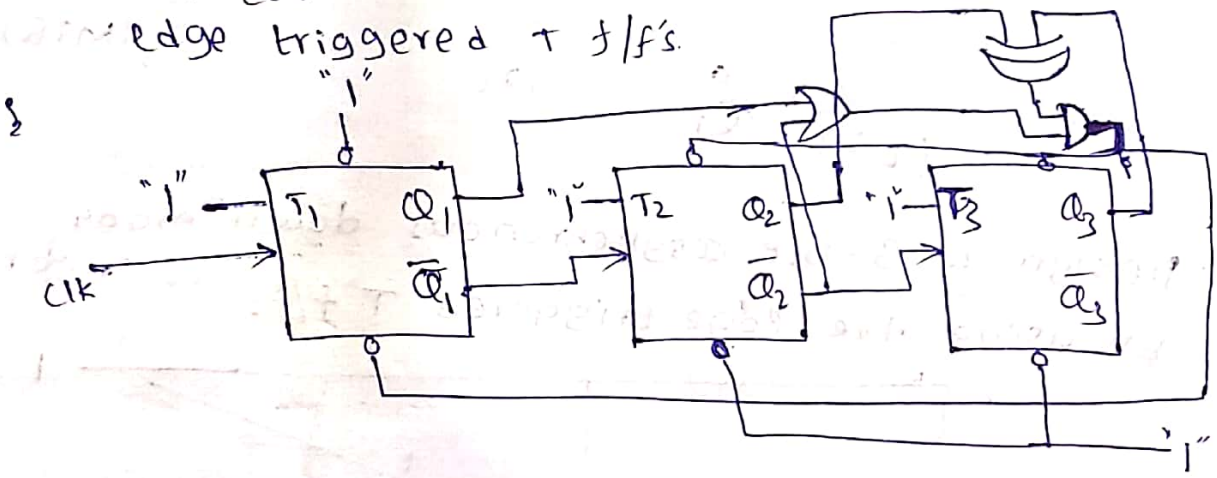


$\Rightarrow \bar{Q}_3 \Rightarrow$ preset = Q_3

[∵ it should be set to 111 & not 000]



Q3. Design an asynchronous up counter which will count 011 to 101 by using +ve edge triggered T f/f's.



⇒

110
111
000
001
010
→ 011

Q_3 \ Q_2Q_1	00	01	11	10
0	1	1	D	1
1	0	0	1	1

$CLr_1' = Pr_2' = Pr_3' = Q_3'Q_2'$
$+ Q_3Q_2$
$+ Q_2'Q_1'$
$= (Q_3 \oplus Q_2)' + Q_2Q_1'$
$= ((Q_3 \oplus Q_2) \cdot (Q_2' + Q_1))'$

(6) 4. Design a sequential ckt with 4 f/f s defined as follows

(a) MSB = $A(t+1) = (C(t) \oplus D(t)) \cdot x + (C(t) \odot D(t))\bar{x}$

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

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

LSB : $D(t+1) = C(t)$

(b) Obtain the ~~sequ~~ state sequence starting from $x = 1$ & ABCD = 0001. Write down the decimal equivalent for all the state.

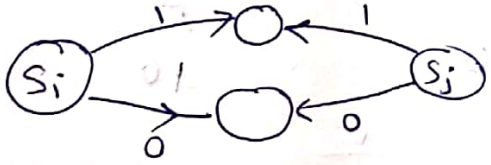
* State equivalence & minimization:-

n-State machine

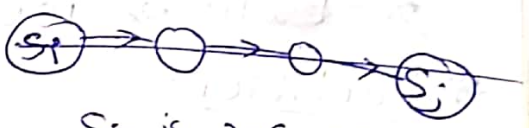
⇒ the no. of memory elements required,

$$k = \lceil \log_2 n \rceil$$

In machine M, S_i & S_j are two states



* ~~*~~K-equivalence
* K- distinguishable



~~S_j is 3-successor of S_i~~

→ If S_i & S_j, the two states of m/c M produce the same output sequences on application of same K-no. of inputs (k-sequence) then S_i & S_j are K-equivalent.

i.e input seq. of length = k

~~→ If S_i & S_j produce the same o~~

→ If S_i & S_j are k-equivalent and the produce diff. output for $(k+1)^{th}$ input, they are (k+1) - distinguishable.

PS	NS, z	
	$x=0$	$x=1$
A	E, 0	D, 1
B	F, 0	D, 0
C	E, 0	B, 1
D	F, 0	B, 0
E	C, 0	F, 1
F	B, 0	C, 0

⇒ A, E are 1-equivalent

Input sequence: $\frac{x}{00}$ $\frac{x}{01}$

Output: $\frac{A}{00}$ $\frac{E}{00}$ $\frac{A}{01}$ $\frac{E}{01}$

⇒ A, E are 2-equivalent

$\frac{x}{10}$ $\frac{x}{11}$

$\frac{A}{10}$ $\frac{E}{10}$ $\frac{A}{10}$ $\frac{E}{10}$

$x = 111 \Rightarrow \frac{A}{100}$ $\frac{E}{101}$ ⇒ A, E are 3-distinguishable

* Partition:

One partition is ~~same~~ some block of states which are k-equivalent.

$P_k = \{A, E\}$, k=2 in above example

Partition	Block of states
P_0	(A B C D E F)
P_1	(A C E) (B D F)
P_2	(A C E) (B D) (F)
P_3	(A, C) (E) (BD) (F)
P_4	(A C) (E) (B D) (F)

To check if two states are k-equivalent, go to their output states for each input 0,1 and check if they're in same block in P_{k-1}.

If $P_k = P_{k+1}$ ⇒ The same partition will continue for all k. ↓ called equivalent partition.

Theorem 1

The equivalent partition is unique.

Proof:

Let there be two equivalent partitions P_a & P_b.

$P_a \neq P_b$.

⇒ We have atleast two states S_i & S_j where S_i & S_j belong to same ~~partition~~ block in P_a and S_i, S_j are in different ~~partitions~~ blocks in P_b

P_a	P_b	
$\{S_i, S_j\}$	$\{S_i\}, \{S_j\}$	let $a-b=k$

⇒ Atleast some I_j successor exists (k-sequence) for which S_i & S_j are k-distinguishable.

S_i & S_j cannot belong to same ~~group~~ block in P_a. ⇒ contradiction

Theorem:

If S_i & S_j are two distinguishable states of a machine 'M', then they are distinguishable by (n-1) or less where n is the no. of states of 'M'.

* We consider M_1 & M_2 to be two machines and S_i of M_1 and S_j of M_2 are k-equivalent.

If for all states of M_1, there are some k-equivalent states of M_2, then M_1 & M_2 are k-equivalent.