Page No.:

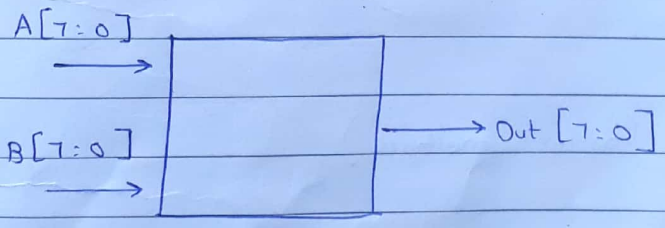
* Boolean formulae

1. $0 + a = a$, $1 \cdot a = a$ Axiom
   $1 + a = 1$, $0 \cdot a = 0$
2. $a + \bar{a} = 1$, $a \cdot \bar{a} = 0$ Axiom
3. $a(b+c) = ab + ac$ Axiom
   $a + bc = (a+b)(a+c)$ Axiom
4. $a \cdot a = a + a = a$
5. $a + \bar{a}b = a + b$
6. Consensus
7. De Morgan.

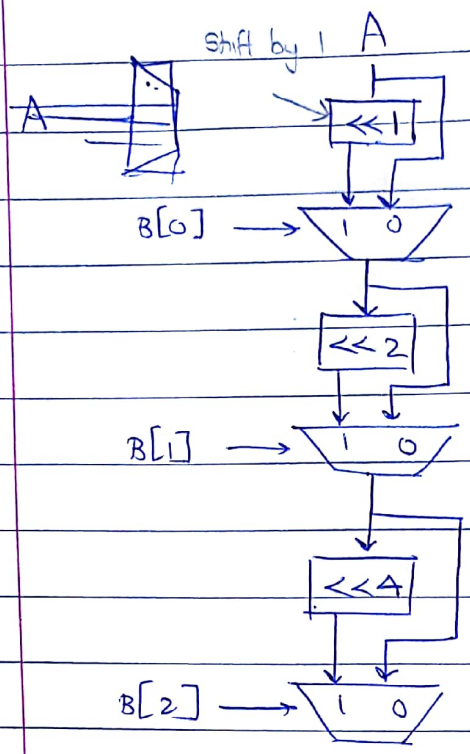
5/12

→ Shift - to - left
(1011 → 0110)
= Multiplication by 2 (without MSB)

* Shift - to - right :- Division by 2 (without remainder)



- Number at B indicates number of places to shift
- If B ≥ 8, Out = 00000000
  - Only B[2:0] are important.



- We could have another I/p 'Right', which decides whether shift left (when 0) and right (when 1)

Meth1 Build left shifter and right-shifter, use MUX to choose.

Meth2 Build only left shifter.
If you want to shift right, first reverse all digits

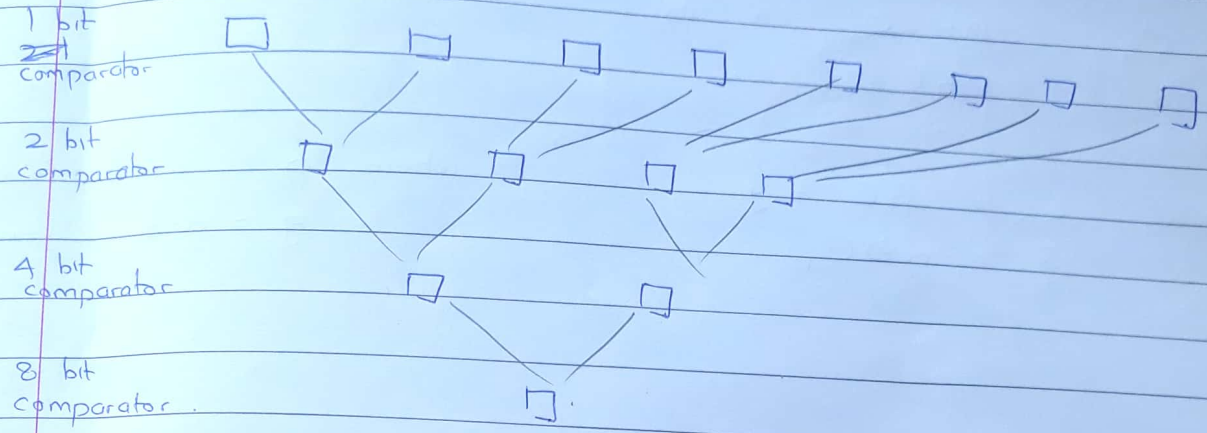
→ Comparator (better method)

Compare highest 4 bits (A_H & B_H) and lowest 4 bits (A_L & B_L) to obtain G_H, E_H, G_L, E_L
Then compare A_{total} & B_{total}

$$G_{total} = G_H + E_H \cdot G_L$$

- ~~Cos~~

Cascade these comparators in binary tree



∴ • In this method, more computation is required but delay is lower

Delay ~ O(log n)

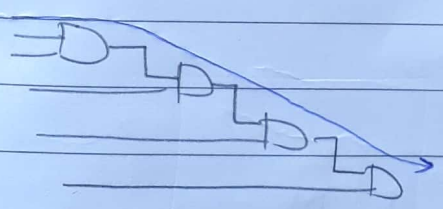
Meth ('' ~ O(n)

• No. of gates required for both methods ~ O(n)

∴ • It is difficult to position this tree on a circuit board.

* Critical Path :-

The path which takes longest time (determines delay)



- In adder, the 'carry' path is critical ($C_0 \rightarrow C_1 \cdots\cdots \rightarrow C_7$)

Page No.:

FASTER ADDER

* For adder,

$C_0 = (a \oplus b) C_1 + ab$

$S = (a \oplus b) \oplus C_1$

Write $C_0 = pC_1 + g$ where $p = a \oplus b$ = 'propag

$S = p \oplus C_1$ $g = a \cdot b$ = 'generat

6/2 • $p = a \oplus b$ can be calculated in parallel for all 8-bits.

• To make critical path (carry overs) faster,

$C_1 = p_0 C_0 + g_0$

$C_2 = p_1 p_0 C_0 + p_1 g_0 + g_1$

$C_3 = p_2 p_1 p_0 C_0 + p_2 p_1 g_0 + p_2 g_1 + g_2$

C_0 was propagated if $p_2 = p_1 = p_0 = 1$

Carry was 'generated'

$C_n = (p_{n-1} p_{n-2} \cdots p_0) C_0 + [g_{n-1} + p_{n-1} g_{n-2} + p_{n-1} p_{n-2} g_{n-3} \cdots]$

$O(\log n)$ product of n literals

sum of n terms

• 'Divide and conquer' gives ↓ delay, requires ↑ no. of gates.

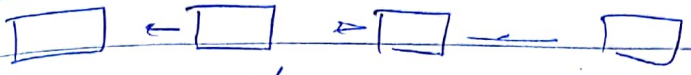
Delay $\sim O(\log n)$

* No. of gates $\sim O(n^3)$

" " " for one $C \sim O(n^2)$

Page No.:

*



n/k in series

Delay $\sim O\left(\frac{n}{k} \log k\right)$

No. of gates $\sim O\left(\frac{n}{k} k^3\right)$

- To reduce no. of gates, ~~carry~~ calculate only C_n carry fast using log method. Calculate remaining at leisure.

"Delay V/s Cost"

MULTIPLIER.

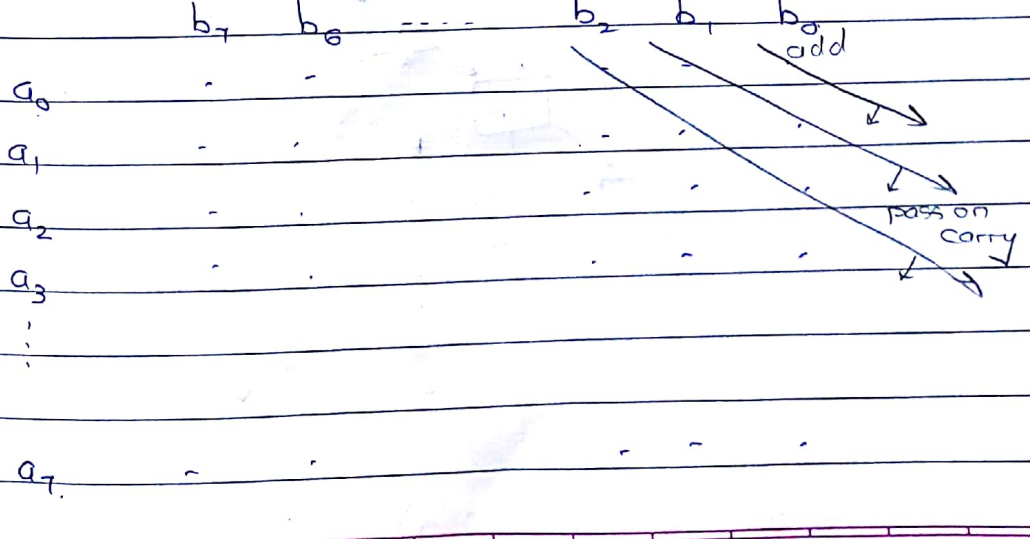
$a_7 \ldots a_0$

$\times \; b_7 \ldots b_0$

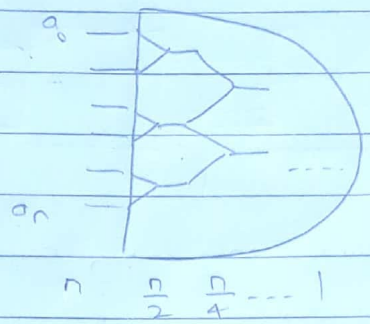
$= a_0b_0 + 2(a_1b_0 + a_0b_1) + 2^2(a_2b_0 + a_1b_1 + a_0b_2) \ldots$

* No of gates $\sim O(n^2)$

A] Array Multiplier



- The fastest possible time is $O(\log n)$

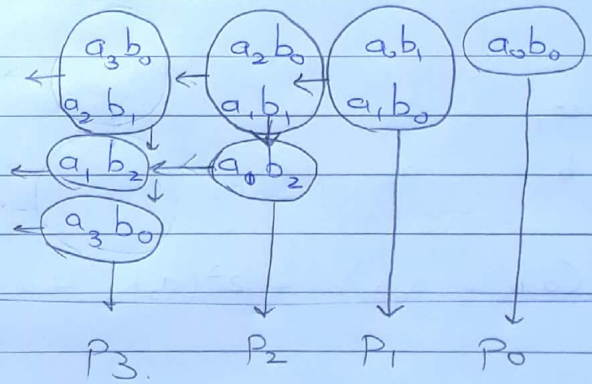


8/2

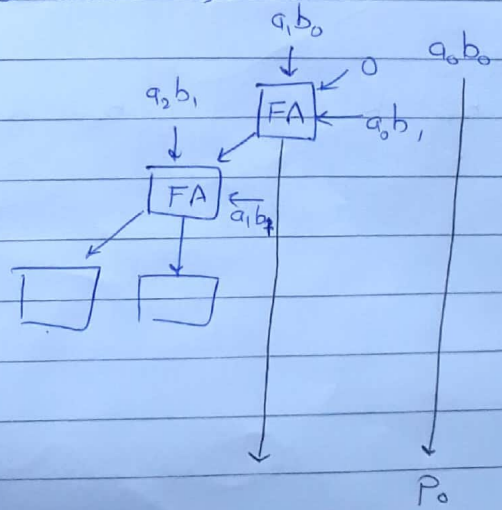
$a_7\ a_6\ a_5\ a_4\ \ldots\ a_0$

$b_7\ b_6\ b_5\ b_4\ \ldots\ b_0$

$P_{15}\ \ldots\ P_0$

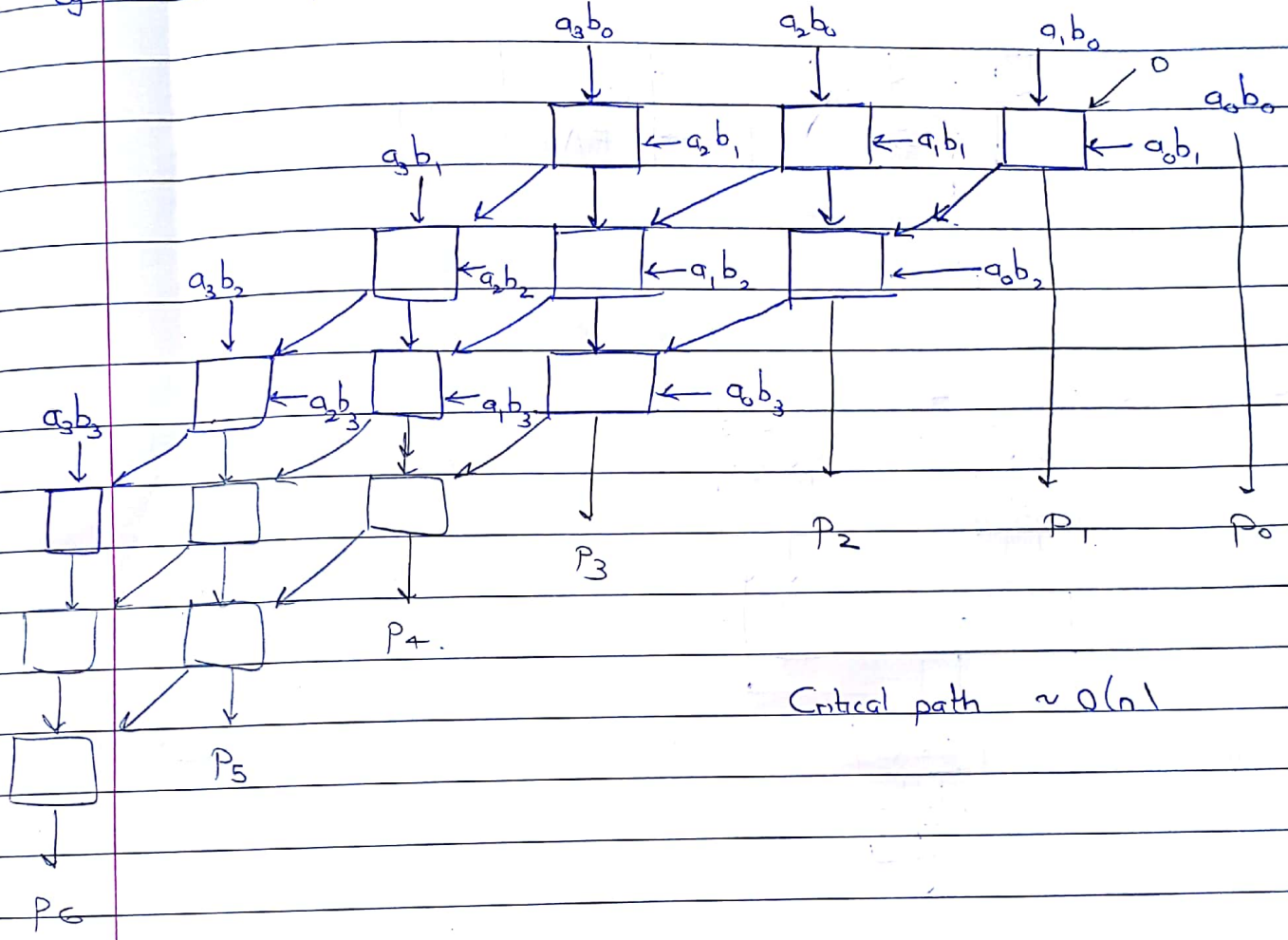


Use Full adders to achieve this:-



- Critical path of carry-over $\sim O(n)$

eg 3 × 3 multiplication



Critical path $\sim O(n)$

→ To improve the critical path length, better way to add :-

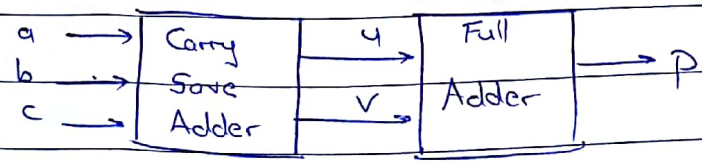
'Carry Save Adder'

$$
\begin{array}{cccccc}
 & c_4 & a_3 & a_2 & a_1 & a_0 \\
+ & b_4 & b_3 & b_2 & b_1 & b_0 \\
+ & w_4 & w_3 & w_2 & w_1 & w_0 \\
\hline
 & u_4 & u_3 & u_2 & u_1 & u_0 \quad \text{--- Sum} \\
+ & v_5 & v_4 & v_3 & v_2 & v_1 \quad \text{--- Carry} \\
\hline
 & p_5 & p_4 & p_3 & p_2 & p_1 \; p_0
\end{array}
$$

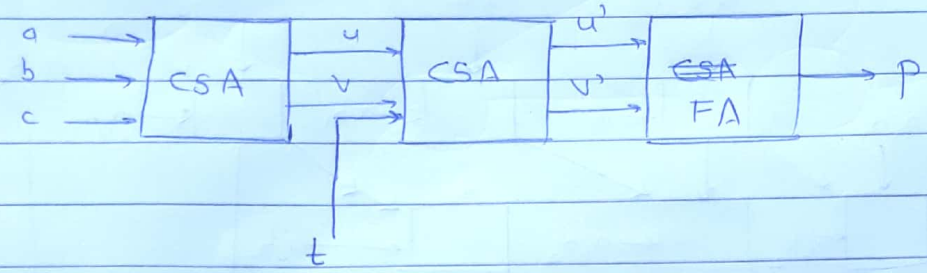
Sum of 3

↓

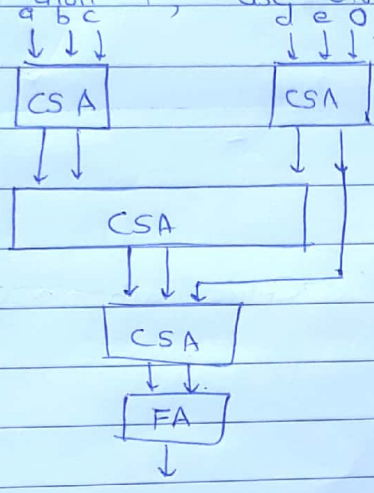
Sum of 2



For more than 3 numbers,



- For more than 4, use divide and conquer.

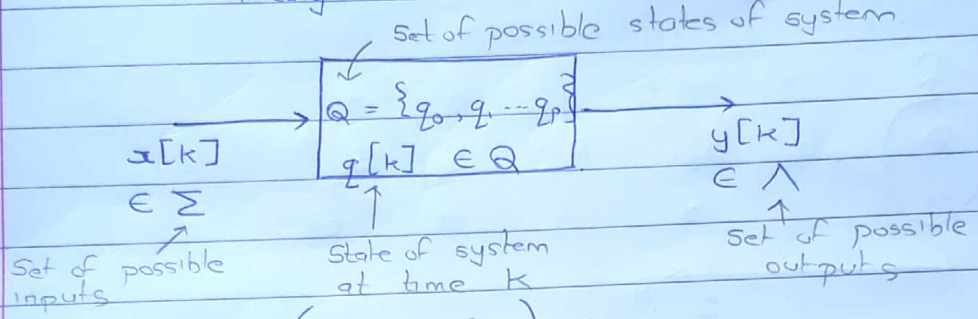


SEQUENTIAL CIRCUITS

- Have memory

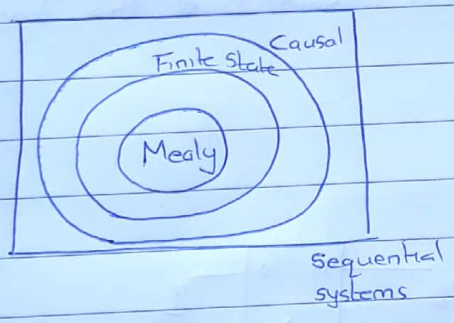
I FINITE STATE MACHINE = Finite sets of I/P, O/P, state.

'Mealy Machine'



1. $y[k] = \lambda(x[k], q[k])$
2. $q[k+1] = \delta(x[k], q[k])$
   = Define as 'nq[k]' (next q)

3.



eg Design a finite state machine.

$x[k] \in \Sigma \equiv \{a, b\}$

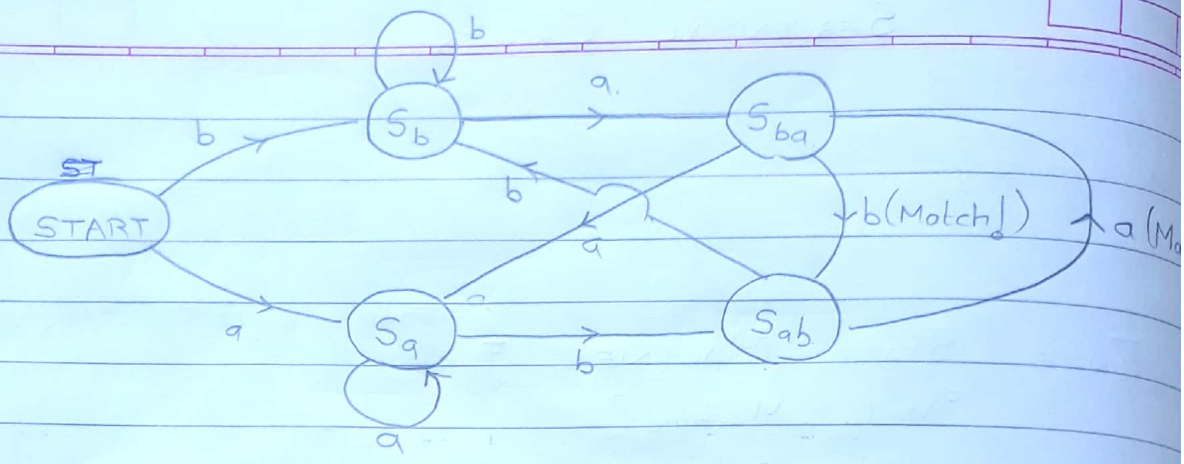
$y[k] \in \Lambda \equiv \{y, n\}$

such that $y[k] = y$ if last three inputs ~~are~~ are 'aba' or 'bab'

$= n$ otherwise

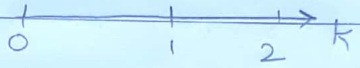
eg-

	→ k								
x[k]	a	b	a	a	b	a	b	a	a
y[k]	n	n	y	n	n	y	y	y	n



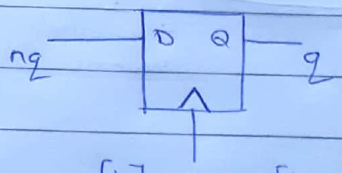
eg

$y[k]$		n	n	y	n	n
$q[k]$	$\emptyset$	S_a	S_a	S_{ab}	S_{ba}	S_a
$x[k]$	a	a	b	a	a	b



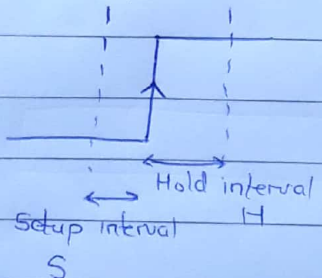
- The discrete time 'k' can be provided using edges of clock cy
  - The signal must have settled (be constant) at the times of sampling.

II D Flip Flops



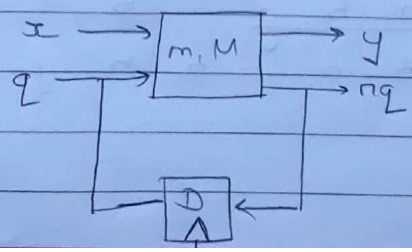
$q[k]$ $\quad q[k+1] = nq[k]$

- Clock :



- Changes in signal should occur before S interval
  The input value expected to be read must be held till after Hinte
  $\Rightarrow$ Input must be stable in [S, H]

eg Above example

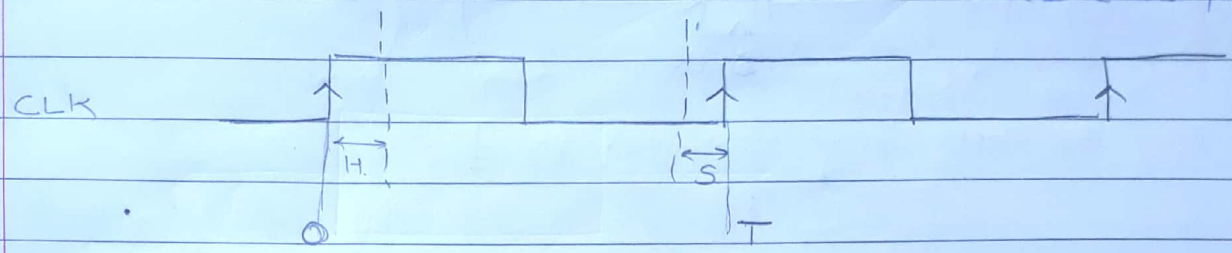


- Say $q[k+1] = nq[k]$

Take D = Delay of flip-flop

m = Minimum delay in finite state machine (shortest path)

M = Maximum " " " (~~too~~ critical path)



q value is updated at $t = D$.

- Then nq value is updated soonest at $D+m$
  latest at $D+M$

For $t \in (D+m, D+M)$, nq may hold (possibly wrong) updated values

- Hence, for correct functioning of circuitry, ① $D + m \geq H$ ② $D + M \leq T - S$

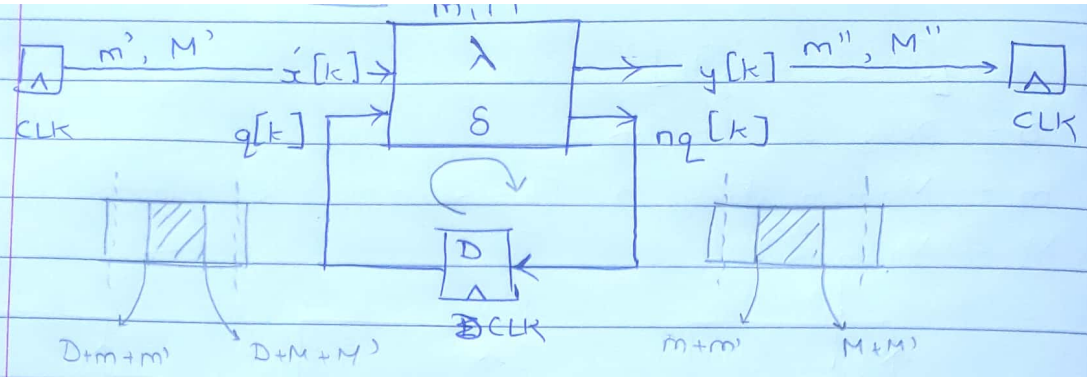
- It is easy to rectify, ~~and~~ if ② is not satisfied
  → Increase T.

19/2

Write $m + D \sim m$ and $M + D \sim M$ $\Rightarrow$ $m \geq H$ and $M \leq T - S$

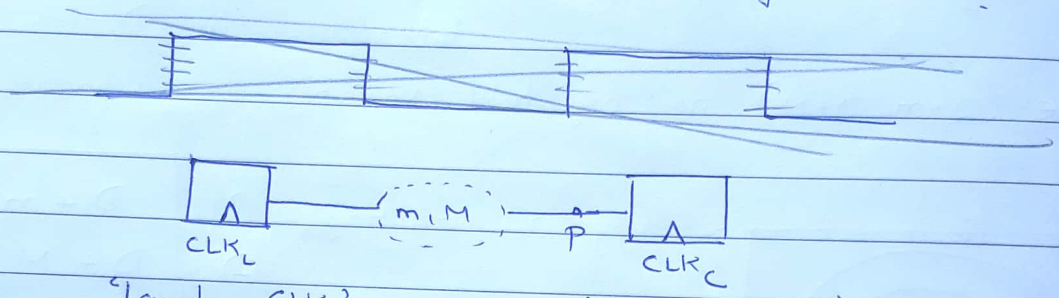
- We cannot let $x[k]$ change any ~~time~~ where in the time period, as the consequential update of $nq[k]$ might fall in $[S, H]$ of flip flop

Similarly, we cannot let $y[k]$ change at any arbit time.

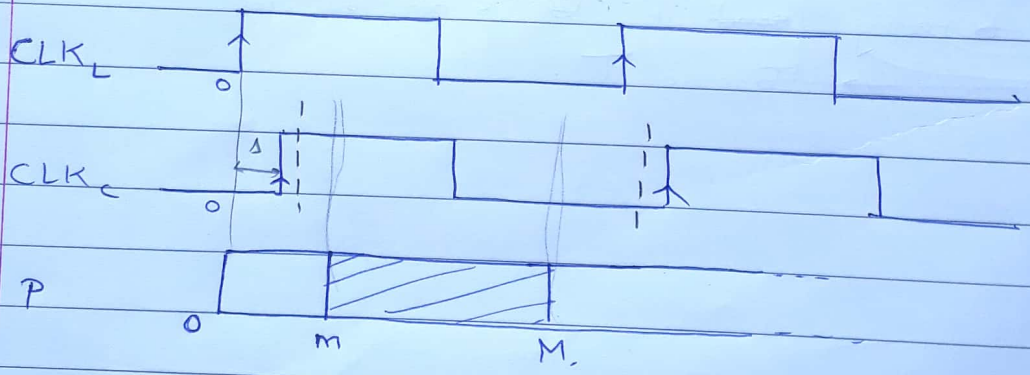


Ensure these intervals do not intersect with [S, H]

→ What if clocks themselves are not synchronous?



'Launching CLK' 'Capturing CLK'



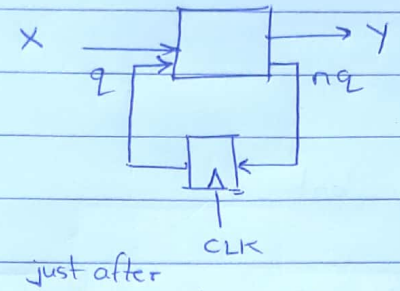
For correct capture of P, we worry about [S, H] of CLK_C

1. $m \geq H + \Delta$ --- More difficult
2. $M \leq (T - S) + \Delta$ --- Easier

- If Δ is positive, and if we can ensure that $m \geq H + \Delta$, then we can choose a lower T for '2' ☺

We can en

II TRACEFILES FOR FINITE STATE MACHINES

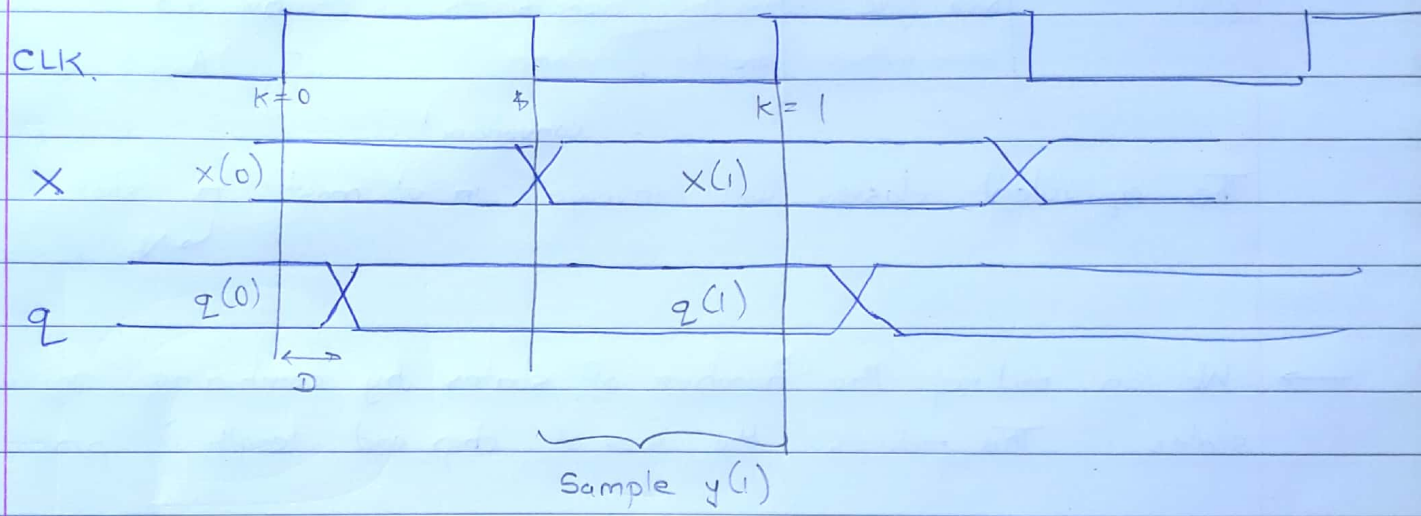


just after

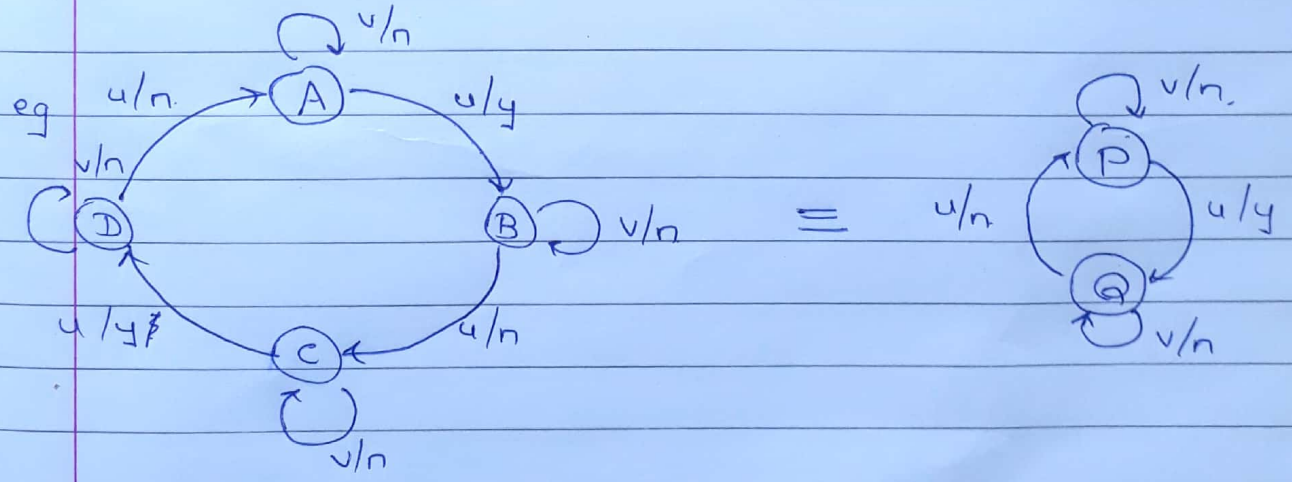
~~and~~ q is changing ~~at~~ positive edge of CLK.

Make X change at negative edge of CLK.

Sample Y after a predecided delay after negative edge of CLK



III EQUIVALENCE OF FSMs



In LHS figure, A and C are not distinct states

B and D

- Two states A and B are distinct if the same input produces different output.

0-equivalent	1-equivalent	2-equivalent
{A, B, C, D}	{A, C} {B, D}	{A, C} {B, D}
All are equivalent before input is given	After giving 1 input x_0, if both give same output	After giving 1 new input to 1-equivalents, their next states are 1-equivalent

If next I/P is 4
A → B
and C → D
But {B, D} are 1-
∴ {A, C} are 2-e

---converged

The equivalent classes will 'converge' in at most 'n' steps
↳ No. of states

→ We can reduce the number of states by combining equivalent states. This reduces the size of chip and length of program.