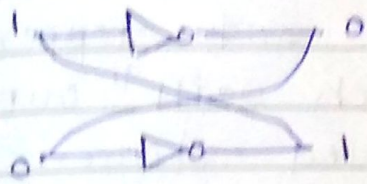
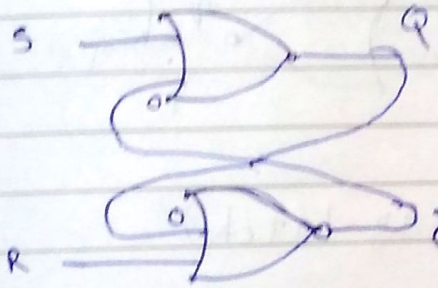
sites.google.com/site/ec2sunit.

⇒ Memory cell -



→ 0 will remain 0 and 1 will remain 1 only.

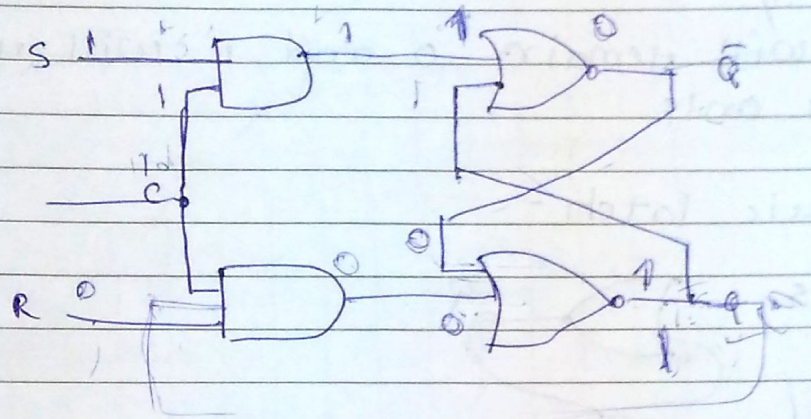
⇒ Basic latch -



R	S	Q(t)	Q(t+1)	$\overline{Q(t)}$
0	0	0	0	1
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	0	0
1	1	1	0	0

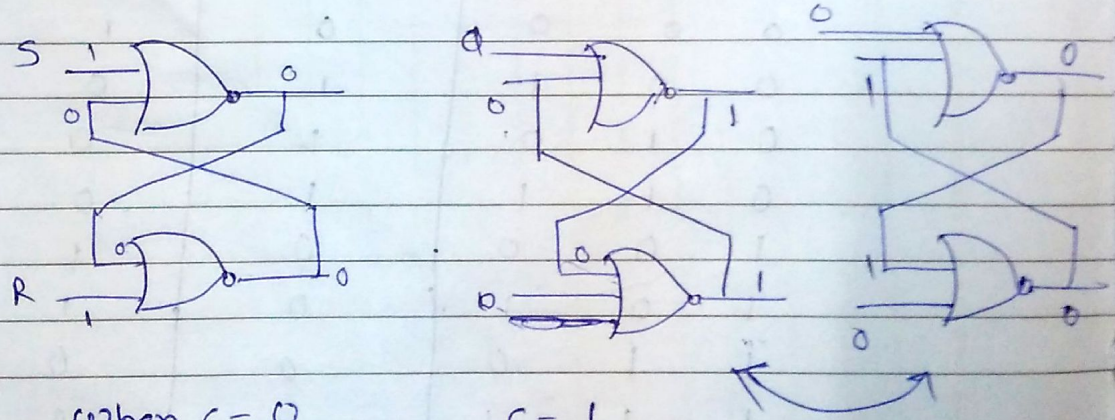
(1) Clocked SR latch -

→ If C=0 then latch is not responding to inputs. So, Latch will give output of previously stored data.



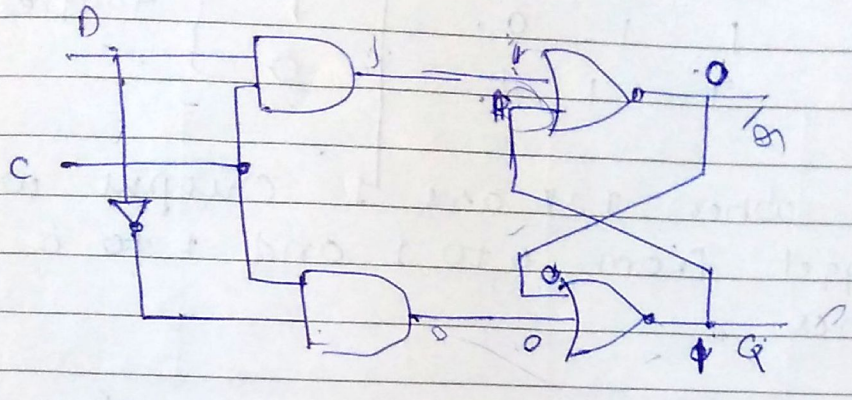
⇒ Oscillations :- (In SR latch)

When S = R = 1.



When C=0 C=1.

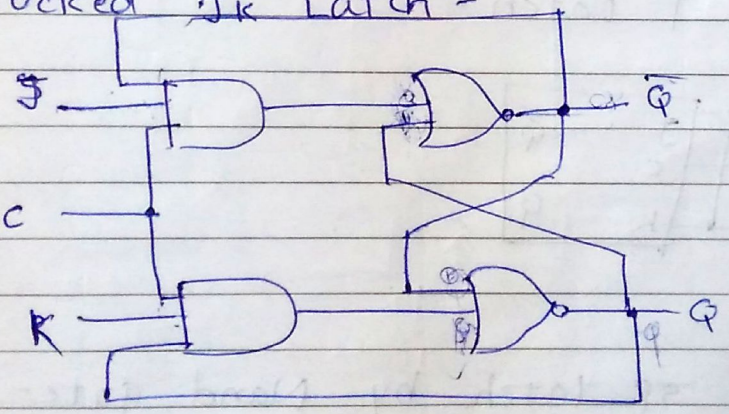
(2) Clocked D latch –



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

$Q(t+1) = D$.

(3) clocked JK Latch –



C=1.

J	K	Q(t)	Q(t+1)	
0	0	0	0	Memory
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set

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

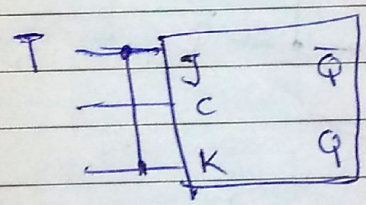
toggle

→ Here, when J, K are 1 Output will toggel from 0 to 1 and 1 to 0 multiple times.

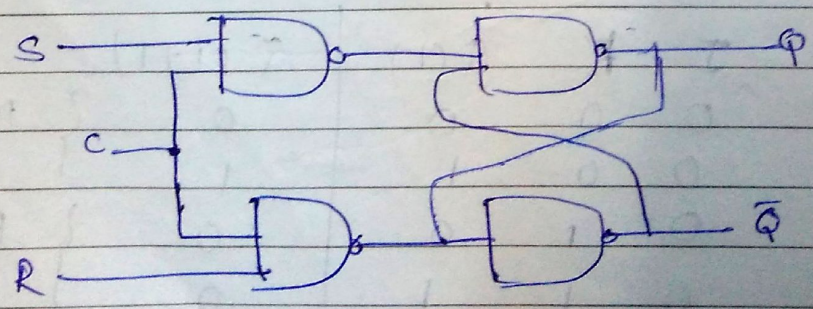
→ Solution 1: We have to keep clock cycle width very narrow. so, J, K value will remain 1 for only lesser time and Output will toggle very less times.

→ Solution 2: Use flip-flop rather than lo

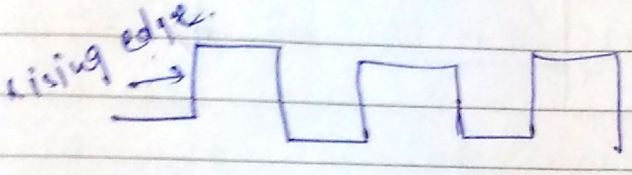
(4) Clocked T Latch -



⇒ Clocked SR latch by Nand gate-



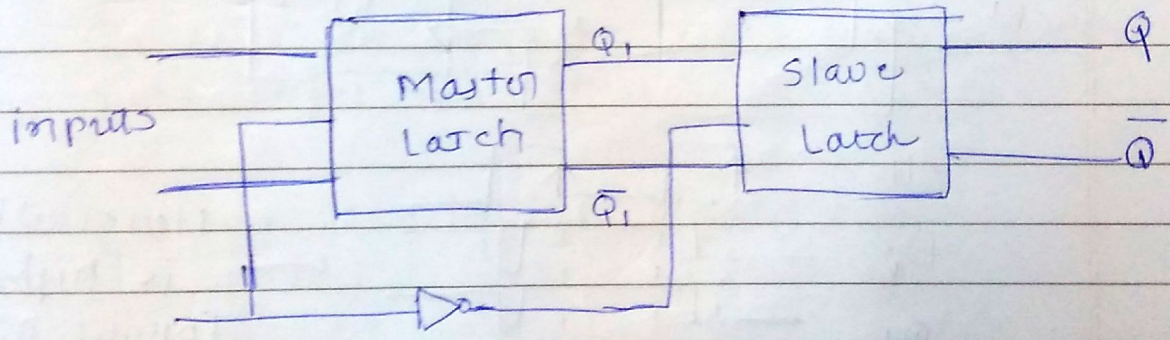
⇒ Difference between flip-flop and latches is Latches respond inputs when clock is high means it is level sensitive while flip-flop respond inputs when rising edge occurs.



(i) rising edge flip-flop.
(ii) falling edge flip-flop.

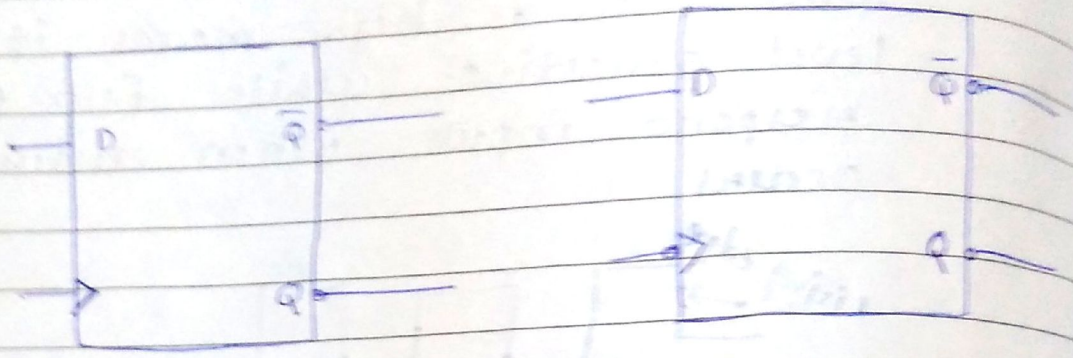
→ So, Latch can't be used when we want to transfer only one bit because it is level dependent. So, for long level it will toggle more than once.

⇒ Flip-flop Master slave (edge sensitive)



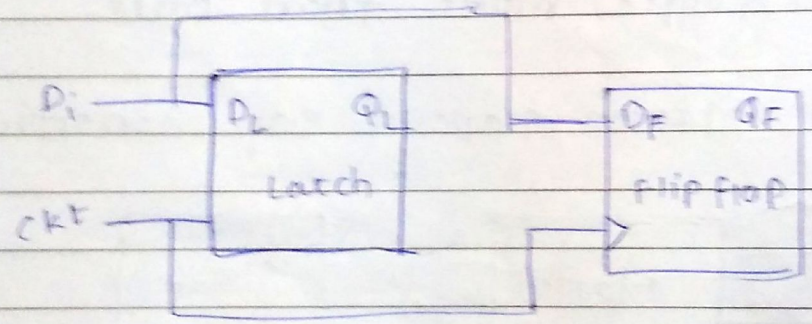
→ When slave latch is enabled, Master latch will be disabled.

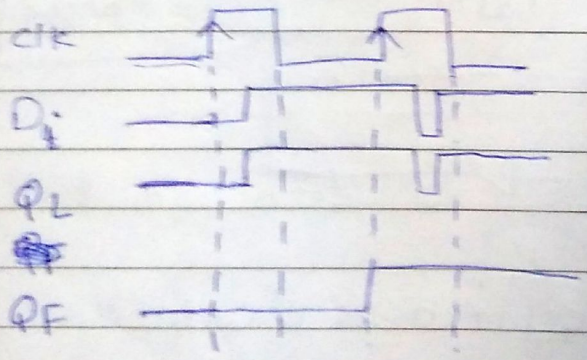
⇒ Symbol of flip-flop –



Rising edge flip-flop. falling edge flip flop

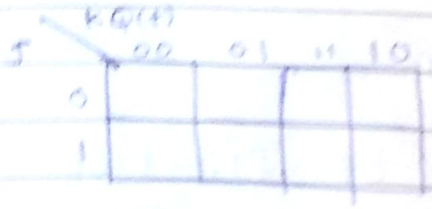
e.g. Draw the waveform of output of latch Q_L and flip-flop Q_F. If both $Q_L = Q_F = 0$ initially





Here when clock is high, Q_L will follow Di and when clock is low, Q_L will show previous output

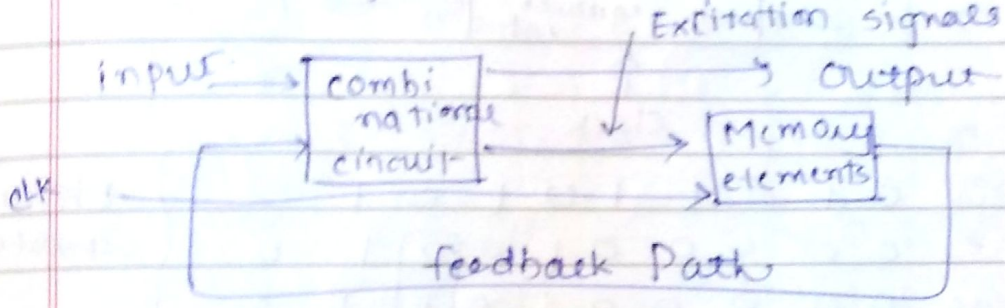
⇒ k-map for JK latch –



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

Sequencial Circuit

→ Block-diagram of Sequencial circuit.



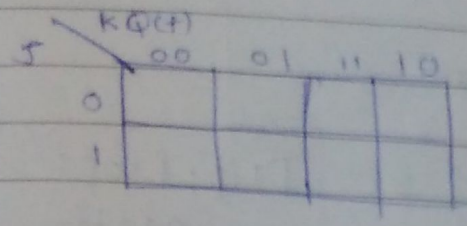
→ Reverse of characteristic table is "Excitation Table"
Input side will be Q(t) and output side will be J, K (OR S, R).

⇒ Flip-flop excitation tables :–

(a) RS

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

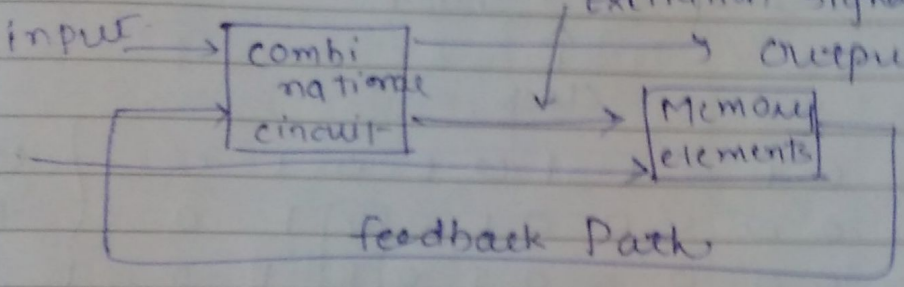
⇒ k-map for JK latch –



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

Sequencial Circuit

→ Block-diagram of Sequencial circuit.



→ Reverse of characteristic table is "Excitation Table"
Input side will be Q(t) and output side will be J, K (OR S, R).

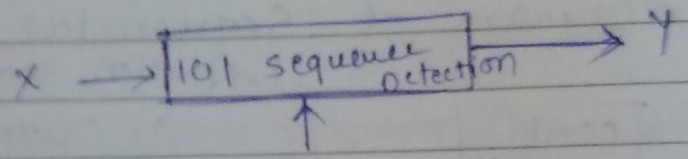
⇒ Flip-flop excitation tables :–

(a) RS

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

⇒ Design logic circuit to -

(1) state diagram
(2) state table
(3) Type & no of flip-flops
(4) excitation values & o/p values. → (input ff) assign symbols
(5) Obtain boolean exp. signals & o/ps

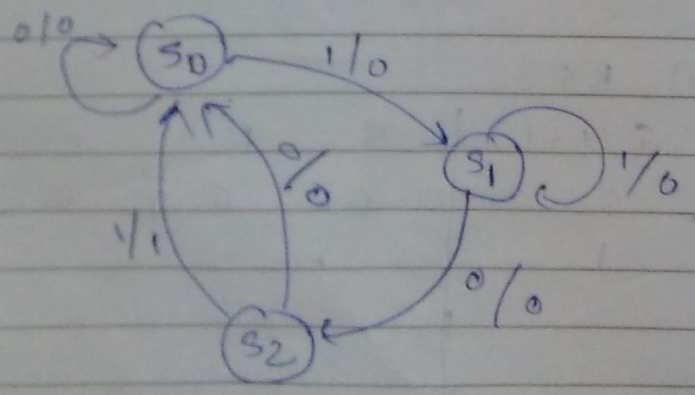


(without overlap)

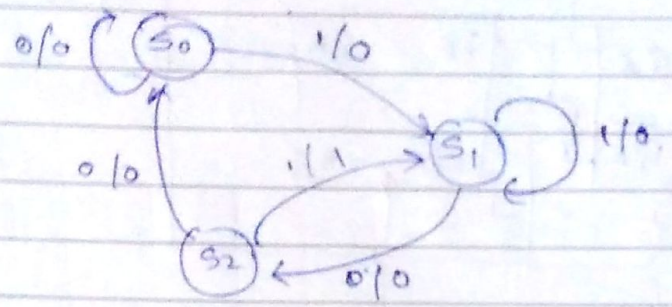
X -	0	1	0	1	1	0	1	0	1	0	
z_1 -	0	0	0	0	0	1	0	0	1	0	
z_2 -	0	0	0	0	0	1	0	1	1	0	

~~state table~~

→ First start with s_0 and if we detect 1 s_1 move to $s_0 \rightarrow s_1$., if we detect 0 then $s_1 \rightarrow s_2$ and then if 1 then $s_2 \rightarrow s_0$.



State diagram without overlap



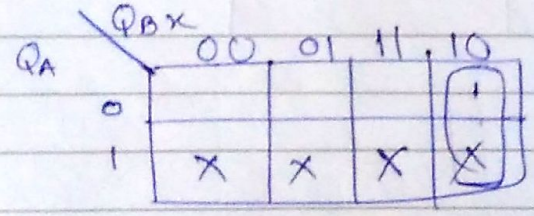
state diagram with overlap.

we have to check for every input 0 & 1.

present		i/p	Next state		o/p	excitation			
Q_A	Q_B	x	$Q_A(t+1)$	$Q_B(t+1)$	z	J_A	K_A	J_B	K_B
0	0	0	0	0 (S_0)	0	0	0	0	0
0	0	1	0	1 (S_1)	0	0	0	1	0
0	1 (S_1)	0	1	0 (S_2)	0	1	0	0	1
0	1 (S_1)	1	0	1 (S_1)	0	0	0	0	0
1	0	0	0	0 (S_0)	0	0	1	0	1
1	0	1	0	0 (S_0)	1	0	1	0	1

→ If m = states and n = i/p (x) then there are ~~six~~ $m \times 2^n$ cases in state table.

→ Draw k-map for J_A, K_A, J_B, K_B, z



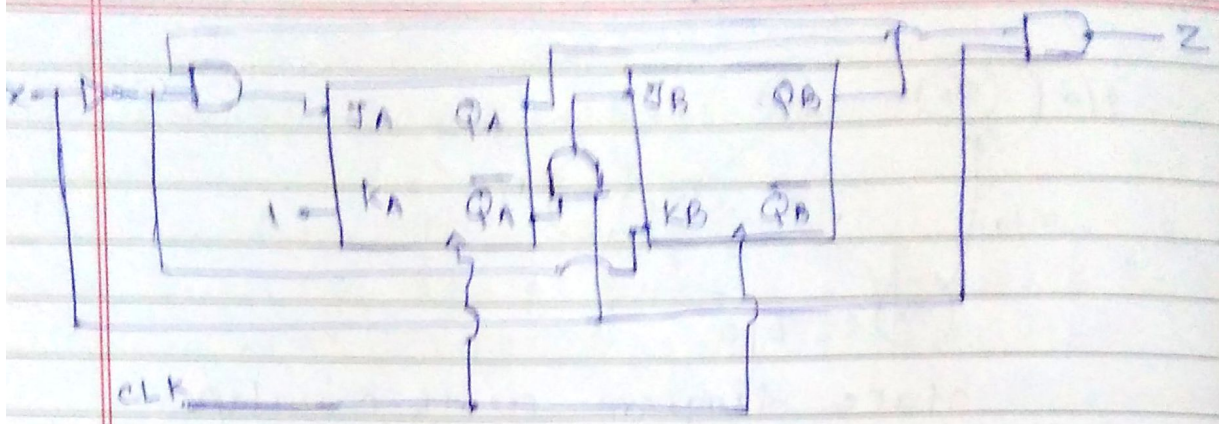
$J_A = Q_B \overline{x}$

if Q_A & Q_B = 1 then that box should n't consider.

(i) Without Overlap :-

$J_A = x\bar{Q}_B$, $K_A = 1$

$J_B = x\bar{Q}_A$, $K_B = \bar{x}$, $z = xQ_A$



⇒ Analysis problem - Sequencial circuit -

(1) first find out the values of S_s and R_s

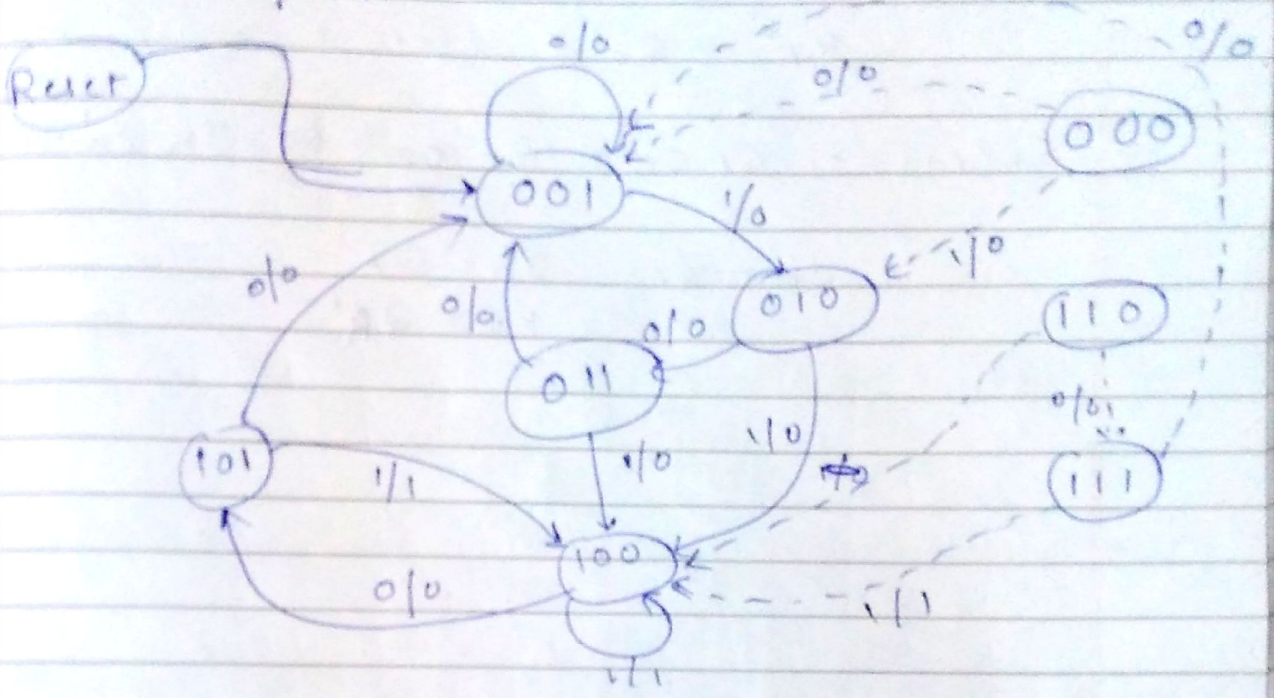
$S_A = x.B$ $\quad S_B = x\bar{A}\bar{B}$ $\quad S_C = \bar{x}$

$R_A = \bar{x}C$ $\quad R_B = xB + BC$ $\quad R_C = x$

(2) Then make state-Table.

Present			x	Next			y = x. A(t).
A(t)	B(t)	C(t)		A(t+1)	B(t+1)	C(t+1)	
0	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0
0	1	0	0	0	1	1	
0	1	0	1	1	0	0	
⋮			⋮				

⇒ State diagram —



Example $Q_A(t+1) = Q_A' Q_B + Q_C$. (State equation)

$Q_B(t+1) = Q_A Q_B + Q_A' Q_B'$

$Q_C(t+1) = Q_B Q_C'$

$Z = (Q_B + Q_C) Q_A'$

→ first of all, we have to convert $Q_A(t+1)$ in terms of Q_A and Q_A'

so, $Q_A(t+1) = Q_A' Q_B + Q_C (Q_A + \overline{Q_A})$

$= Q_A' Q_B + Q_A Q_C + \overline{Q_A} Q_C$

$= \underline{Q_A'} (Q_B + Q_C) + \underline{Q_A} (Q_C)$ — (1)

→ Match eq. (1) with general equation,

$Q_A(t+1) = Q_A' J_A + Q_A K_A'$ [characteristic equation]

SO, $J_A = Q_B + Q_C$

$K_A = Q_C'$

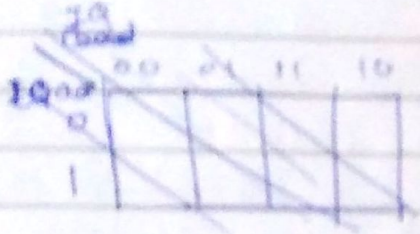
$Q_{B(t+1)} = Q_A Q_B + Q_A' Q_B' \leftrightarrow J_B Q_B' + K_B' Q_B$

~~$J_B = Q_A$~~. $J_B = Q_A'$

$K_B = Q_A'$

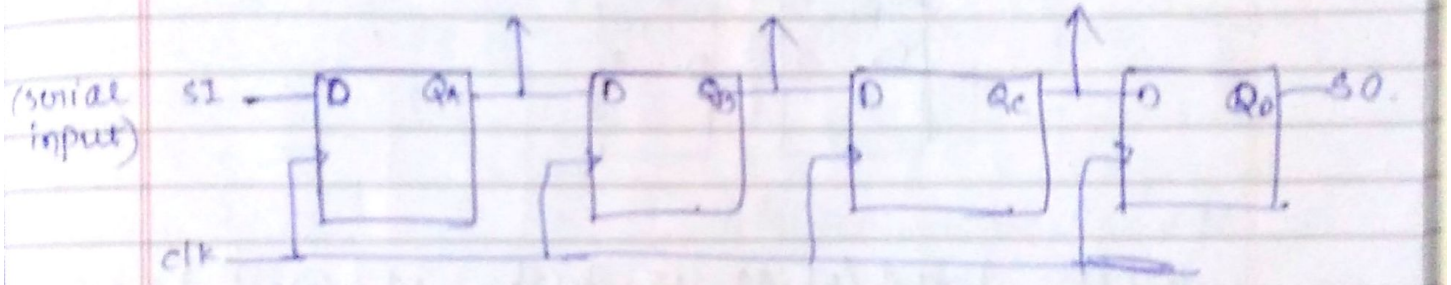
Register with Parallel load (PIPO - reg)

I_i	Q_i	Load	D_i
0	X	1	0
1	X	1	1
X	0	0	0
X	1	0	1

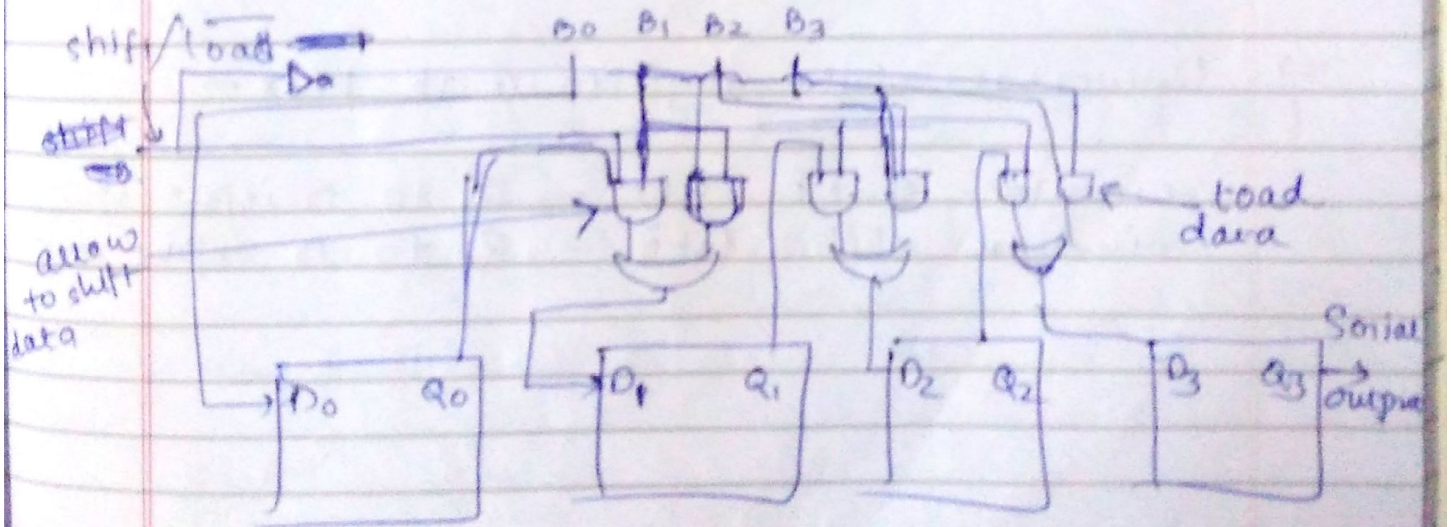


$D_i = I_i \cdot load + Q_i \cdot load$

⇒ Shift - registers — (Serial In, Serial out)



⇒ Parallel in, Serial out —



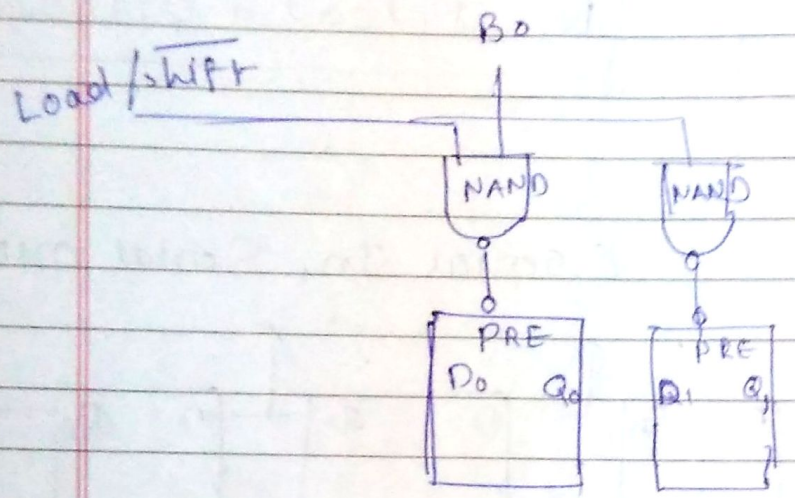
shift/load = 0 = Load data in flip-flops

load = 1 = 1 shift data to right (in diff flip-flop)

→ Preset (PRE) and clear are active Low.

→ We can make a circuit with PRE and clear

for shifting



when Load/shift is high it will load data.

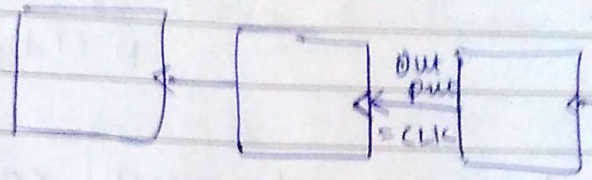
⇒ Universal shift Register with MUX –

circular shift Right – Q_3 to D input of Q_0
circular shift left – Q_0 to D input of Q_3

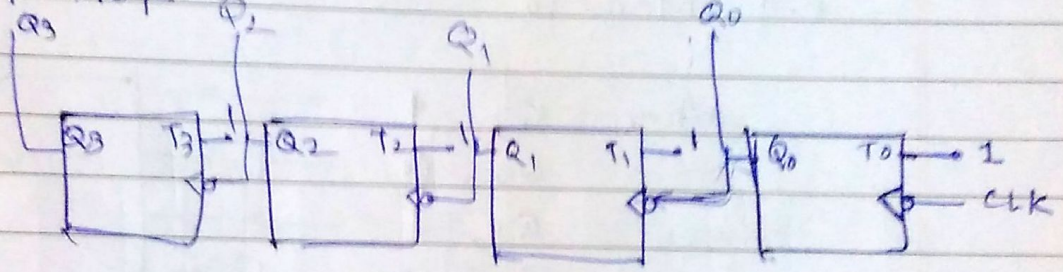
⇒ Mod-N Asynchronous Counter —

→ When clock is given to only one bit is called as Ripple counter. (Binary Counter) Mod 2^N counter)

e.g.

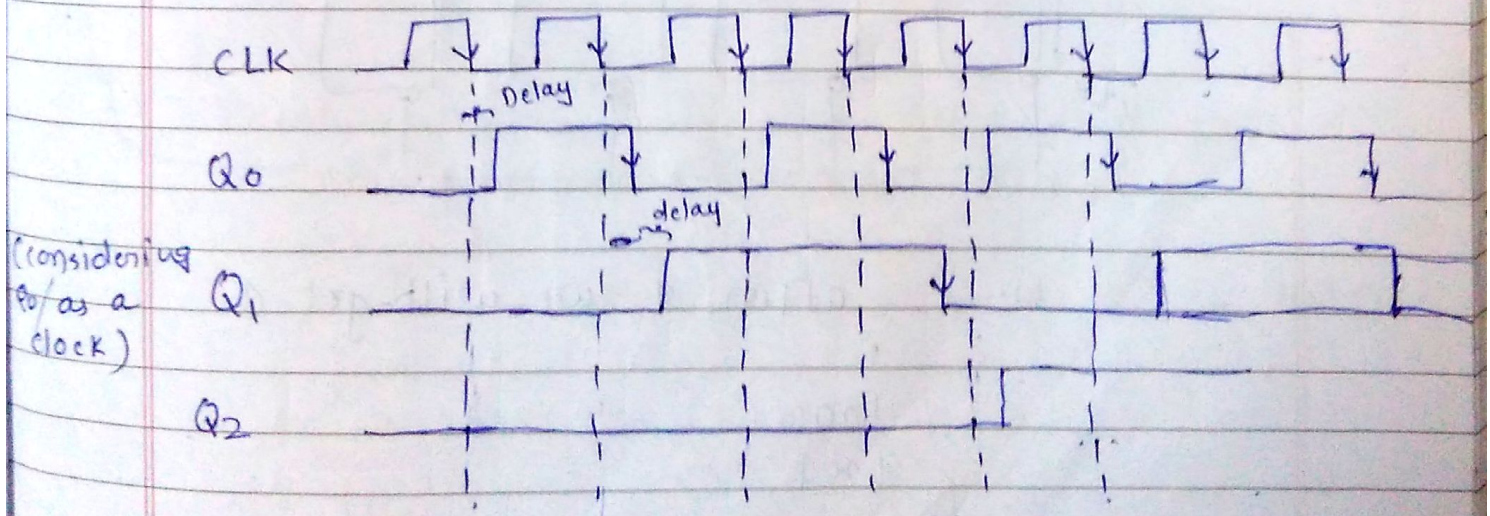


→ In counters we have to use only J-K or T flip-flop



Asynchronous Counter

Q_3	Q_2	Q_1	Q_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1



→ Here frequency depends on the Time delay.

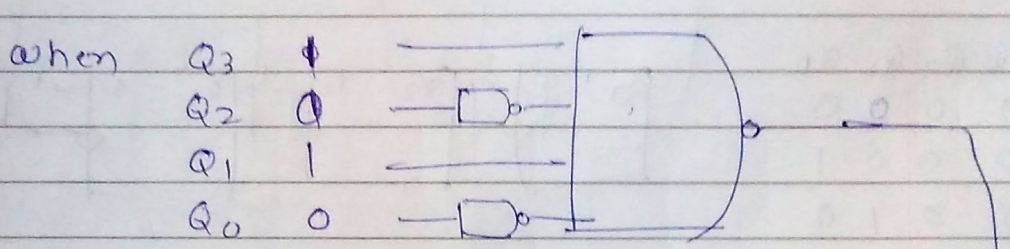
$T_{min} = N \cdot (t_d)_{ff}$ ($\because (t_d)_{ff}$ = time delay of 1 flipflop)

and $f_{max} = \frac{1}{T_{min}}$

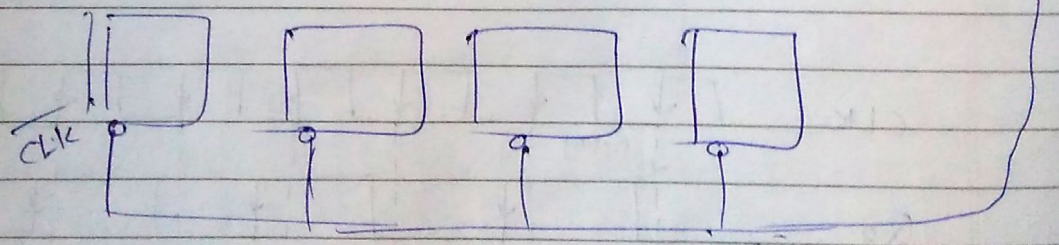
$= \frac{1}{N \cdot (t_d)_{ff}}$

⇒ Mod-N Asynchronous counter –

e.g. Mod-10



Then all bits will be 1. This is Unique condition.

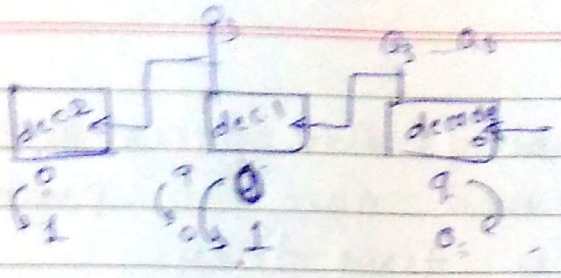


Here, after 9 we will get 0.

1000

1001

0000



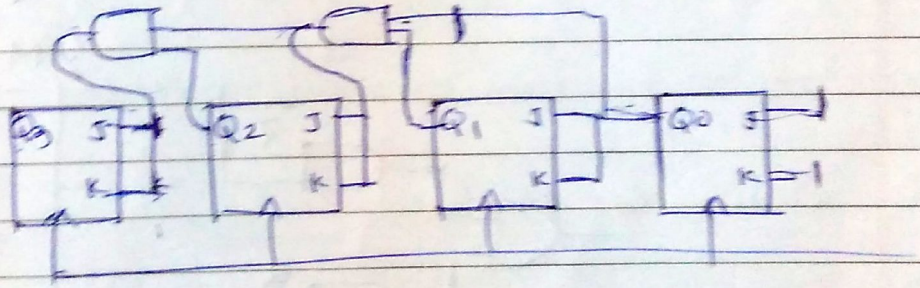
decade 0 goes from 9 to 0 at that time decade 1 goes from 0 to 1.

⇒ Synchronous Counter → Serial Carry

→ Here, we are applying clk simultaneously to all the flip-flops.

Q_3	Q_2	Q_1	Q_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0

when $Q_0 = 1 \Rightarrow Q_1$ will toggle.
$Q_0, Q_1 = 1 \Rightarrow Q_2$ will toggle
$Q_0, Q_1, Q_2 = 1 \Rightarrow Q_3$ will toggle.

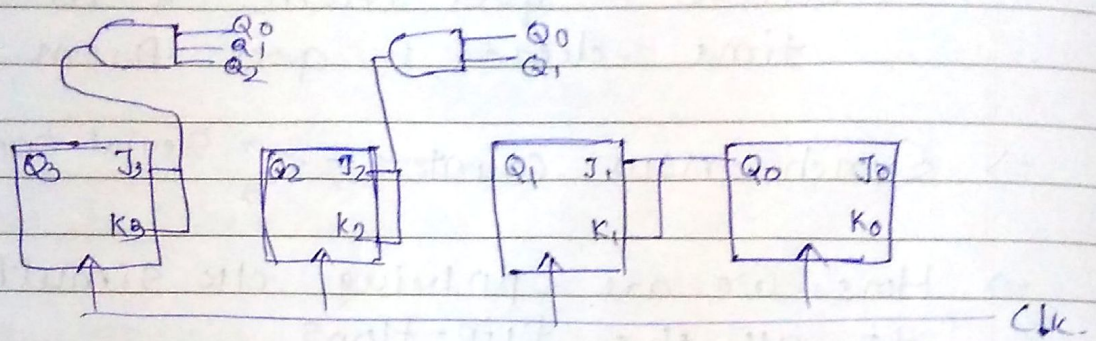


→ There will be 1 ff and two And gates delay in worst case.

$T_{min} = (t_d)_{ff} + (N-2)\, t_d$ AND

$\therefore f_{max} = \frac{1}{T_{min}}$.

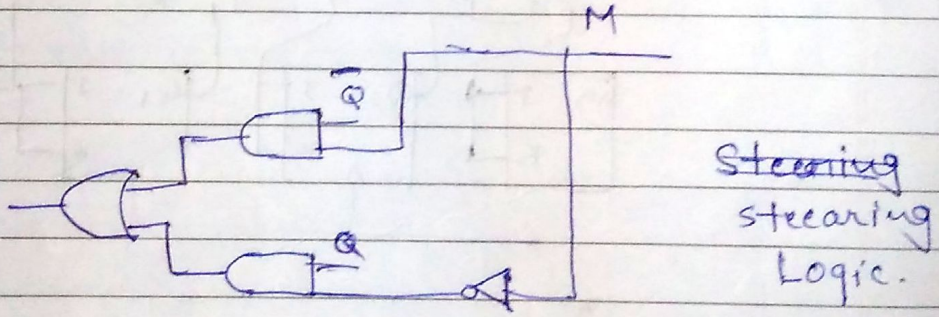
→ Instead of using And gates, we can use only one And gate seperately to produce products without any time-delay.



$T_{min} = (t_{d_1})_{ff} + t_{d_1}$ AND

$\therefore f_{max} = \frac{1}{T_{min}}$

⇒ Up-down counters –

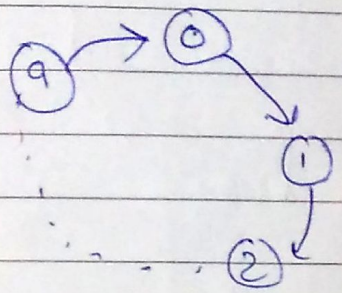


→ when M=0, Q will propogate
That is called as Up-counter.

→ when M=1, $\bar{Q}$ will propogate
That is called as down-counter.

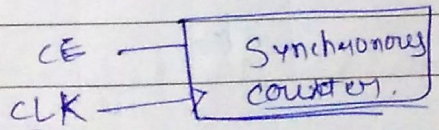
→ For N flip-flops, we require N-1 "steering logic" circuits.

⇒ Mod-N Synchronous Counter –

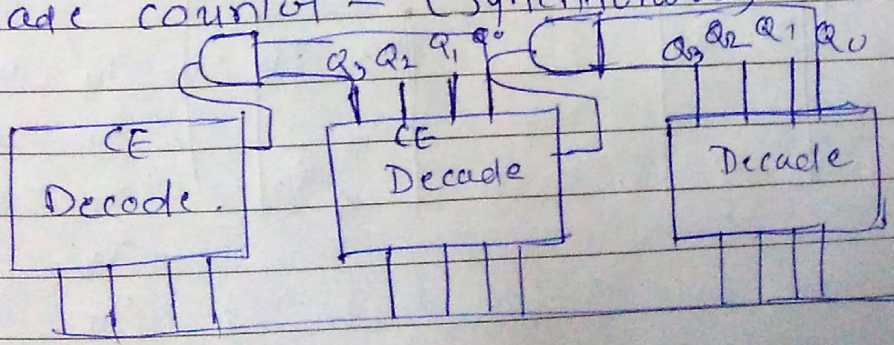


→ Random Sequence can be generated very easily ~~by~~ using Synchronous counter but it is very difficult to generate using Ripple counter.

⇒ Counter Enable :- Used to suspend or to continue counting.

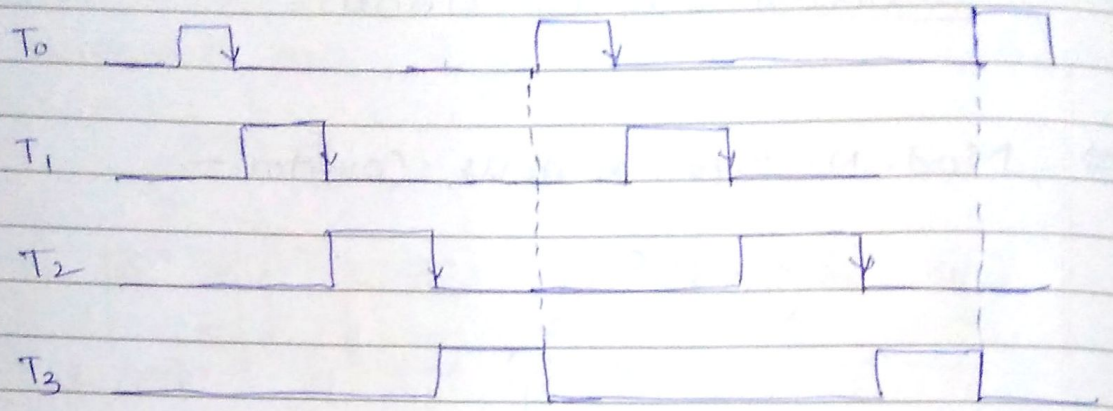


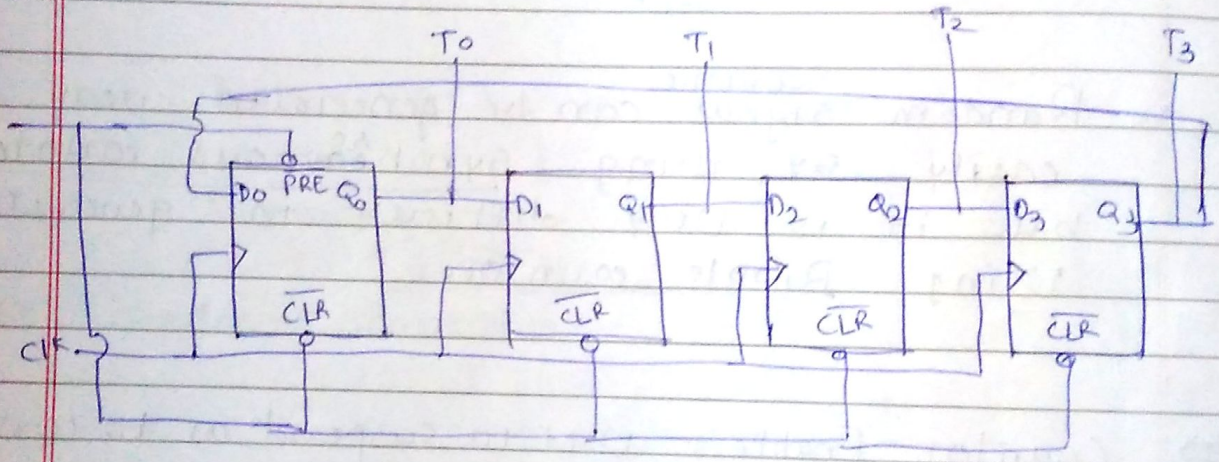
⇒ Decade counter – (synchronous)



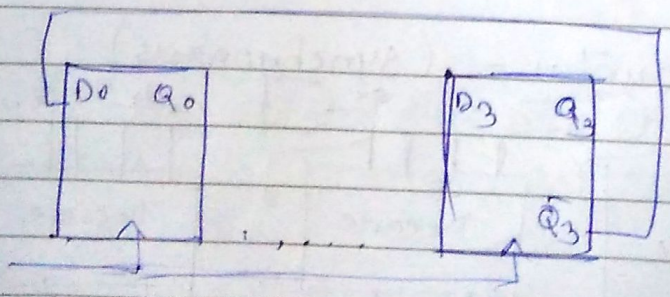
when 9 (1001) is detected, CE will be high and counter will be Activated after than for 0 (0000) counter will be Suspended.

⇒ Ring Counter – (Non-overlapped timing signals)

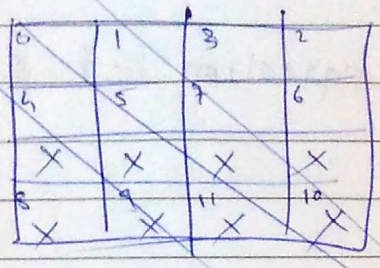




⇒ Johnson counter –



	Q_3	Q_2	Q_1	Q_0
T_0	0	0	0	0
T_1	0	0	0	1
T_2	0	0	1	1
T_3	0	1	1	1
T_4	1	1	1	1
T_5	1	1	1	0
T_6	1	1	0	0
T_7	1	0	0	0
	0	0	0	0



Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	T_0	T_1	T_2	X
01	Y	X	T_3	X
11	T_c	X	T_4	T_5
10	T_7	X	X	X

$T_0 = \overline{Q_3}\,\overline{Q_0}$

$T_1 = \overline{Q_1}\,Q_0$

$T_2 = Q_1\,\overline{Q_2}$

$T_3 = Q_2\,\overline{Q_3}$

⇒ Sequence Generator — $s \le 2^n - 1$, n = No. of ffs.

1010100101

(whatever is at D_1 is going to be set in next clock cycle)

Q_1	Q_2	Q_3	Q_4	D_1
1	1	0	1	0
0	1	1	0	1
1	0	1	1	0
0	1	0	1	1
1	0	1	0	0
0	1	0	1	0
0	0	1	0	1
1	0	0	1	0
0	1	0	0	1
1	0	1	0	1

Here for same input we have different output. So, can't use sequence generator using 4 flip-flops.

→ find out the equation of D_1.

Register Transfer Logic

→ Higher order notations used to describe digital operations are called the "Register Transfer ~~Logic~~ statement" Language (RTL)

→ RT statement consist of (i) control function (ii) micro-operations.

control function :- List of micro-operations seperated by commas.

⇒ Register Notation :-

Registers: A(8), B(8), PC(16)
↓ 8 bits ↓ 16 bits

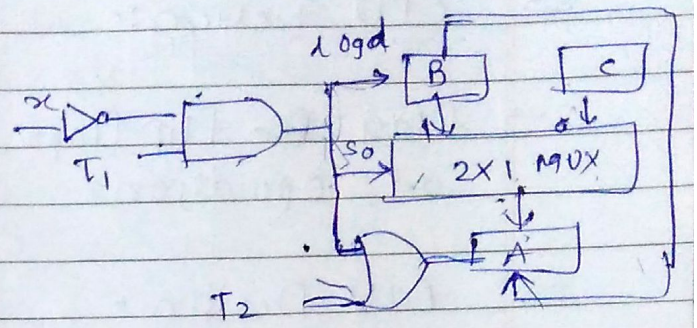
A(5-8) OR A5-8 ⇒ Upper 4 bits of reg A

A_3 ⇒ 3rd bit of reg A.

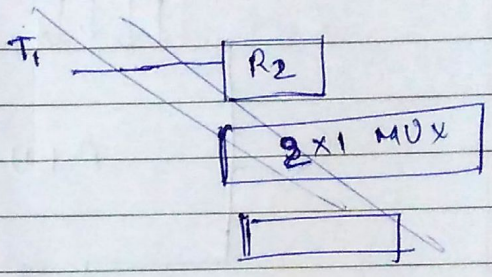
PC(H) ⇒ Higher byte of PC.

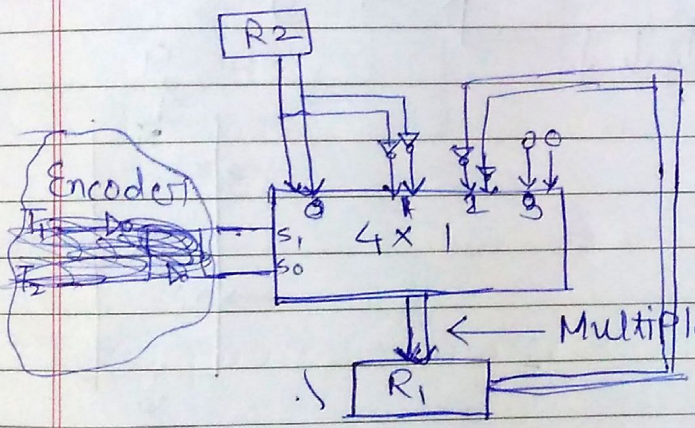
⇒ RTL Description :-

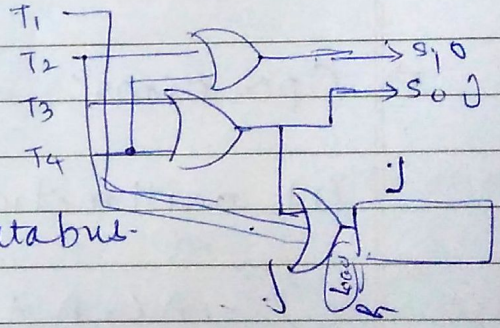
(1) control condition
$x'T_1$: $A \leftarrow B$, $B \leftarrow A$.
T_2 : $A \leftarrow C$.



(2) T_1 : $R_1 \leftarrow R_2$
T_2 : $R_1 \leftarrow \overline{R_2}$
T_3 : $R_1 \leftarrow \overline{R_1}$
T_4 : $R_1 \leftarrow 0$







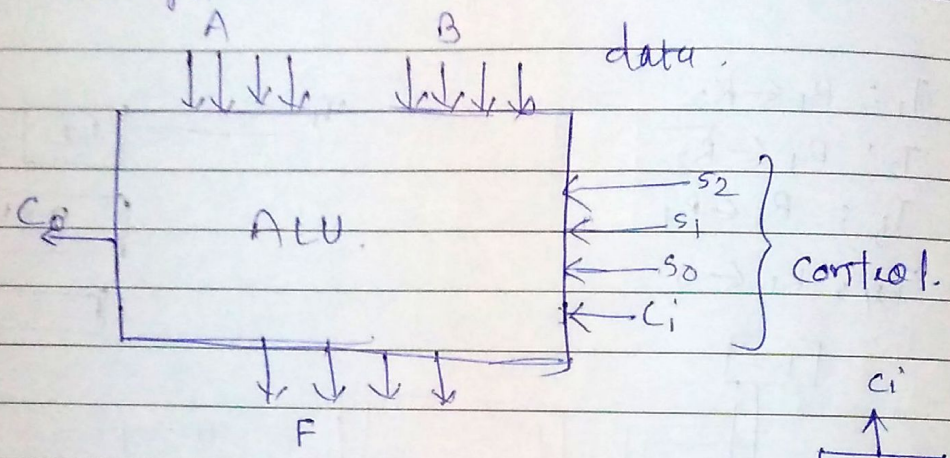
T_1	T_2	T_3	T_4	S_0	S_1
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

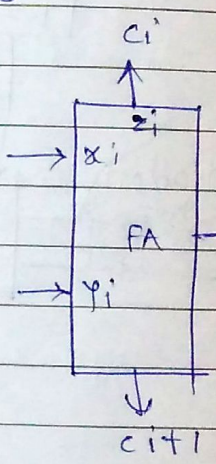
$S_0 = T_3 + T_4$.
$S_1 = T_2 + T_4$.

⇒ CPU Design :-

→ Flag flip-flops — Indicates the answer of operation.

⇒ ALU Design :-





⇒ Operations :

(1) arithmetic.

$A \leftarrow A + B$.
$A \leftarrow A + B + c_i$
$A \leftarrow A + \overline{B} + 1$.

$x_i = A_i$
$y_i = S_0 B_i + S_1 B_i'$
$z_i = c_i$.

(2) Logic

$A \leftarrow$ Not A
$A \leftarrow A \vee B$.
$A \leftarrow A \wedge B$.

for logic operation, carry-in = 0.

so, $z_i = S_2' c_i$.

(3) Shift & Rotate

$A \leftarrow shr\ A$, $A_n (MSB) \leftarrow 0$.
$A \leftarrow shl\ A$, $A_1 ('LSB) \leftarrow 0$.

1111
1111
1110

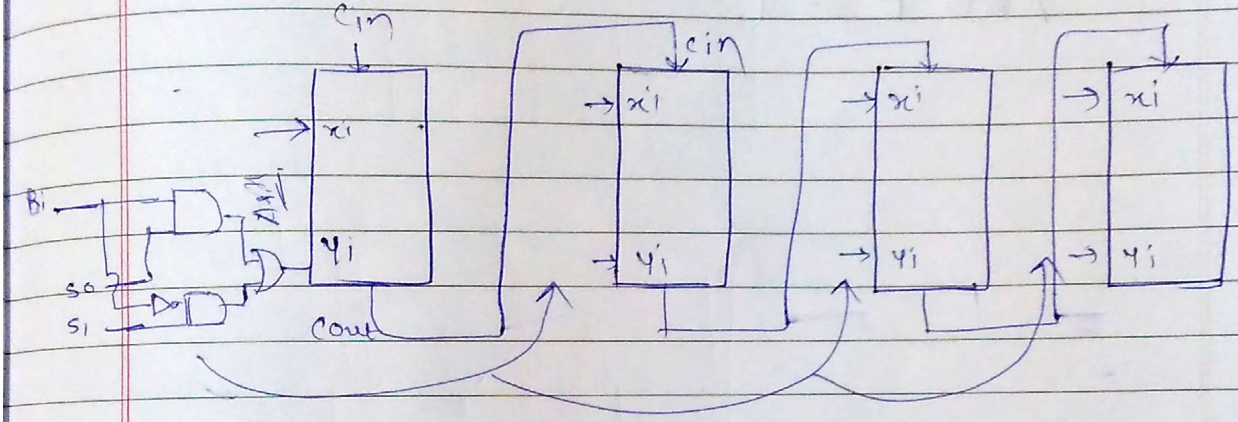
1011
1111
1010

A-1

(4) Rotate Right : A ← shr A, An ← A
Rotate Left : A ← shl A, A1 ← An

→ $S_2 = 0$ for Arithmetic operation
$S_2 = 1$ for Logical operation.

⇒ 4-bit ALU :-



S_2	S_1	S_0	Xi	Yi	C_i	$F_i = X_i \oplus Y_i$	Operation
1	0	0	Ai	0	0	Ai	need to modify
1	0	1	Ai	Bi	0	$A_i \oplus B_i$	XOR
1	1	0	Ai	$\bar{B_i}$	0	$A_i \oplus B_i'$	need to modify
1	1	1	Ai	1	0	A_i'	NOT

→ we need to modify ; with conditions :

S_2	S_1	S_0
1	0	0
1	1	0

To get AND Operation, $X_i = A_i + B_i'$. (when 100)
To get OR operation, $X_i = A_i + B_i$ (" 110)

In General, $X_i = A_i + S_2 S_1' S_0' B_i + S_2 S_1 S_0' B_i'$
$Y_i = S_0 B_i + S_1 B_i'$.

(zero)

Z = 0-flag.

C = carry-flag

A > B	c = 1 and z = 0.
A < B	c = 0 and z = 0
A = B	z = 1.