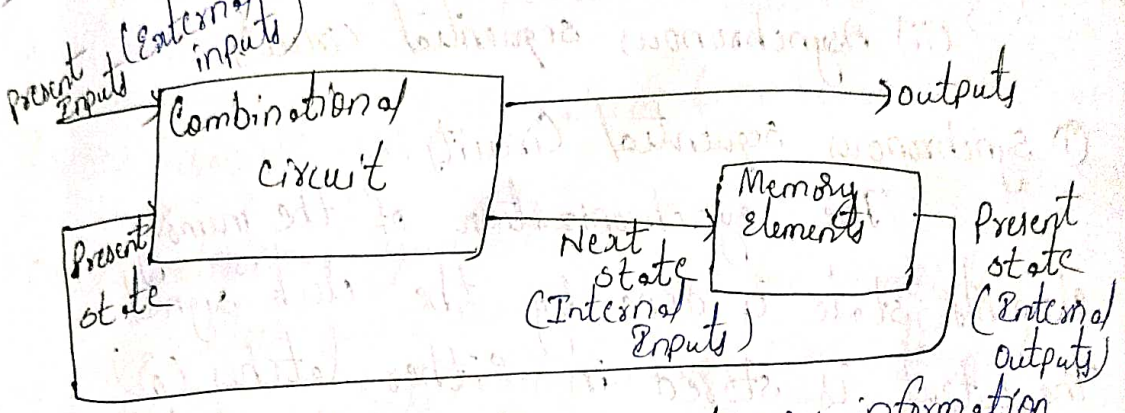
Sequential Circuits:

⇒ A circuit in which output depends on present input is called sequential circuits

⇒ Here output depends on present input and present states.

⇒ Block diagram of a sequential circuit:-



⇒ The sequential circuit receives binary information from external inputs that, together with the present state of the storage elements determine the binary value of the outputs.

⇒ A sequential circuit (S.C) doesn't need to always contain a combinational circuit (C.C). So SC can contain only memory elements also (storage elements)

Differences between Combinational circuit and Sequential circuits.

Combinational circuit	Sequential circuit
① Output depends on only present input.	① Output depends on both present input and present state (previous state output)
② The feedback path is not present	② Feedback path is present
③ Memory elements not required	③ Memory elements (storage elements) required.

Combinational Circuit	Sequential Circuit
④ Clock signal is not required	④ Clock signal is required
⑤ Easy/simple to design	⑤ Difficult to design

*** Types of sequential circuits:

(i) Synchronous sequential circuits

(ii) Asynchronous sequential circuits

① Synchronous Sequential Circuits:-

The synchronization of the memory elements state is done by the clock signal. The output is stored in either latches (or) Flipflops (Memory devices). The synchronization is done by using either positive edges (or) negative edges of the clock signal.

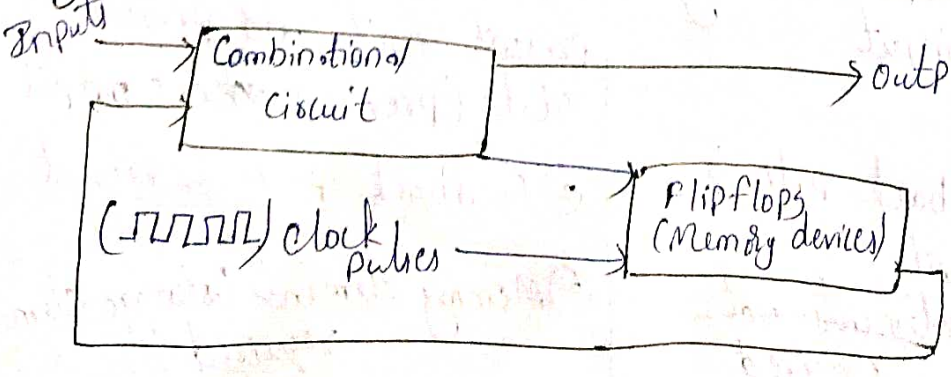
{ Synchronization: Output will change at instance of time }.

⇒ Latches and flipflops are memory devices. They store 1 bit information.

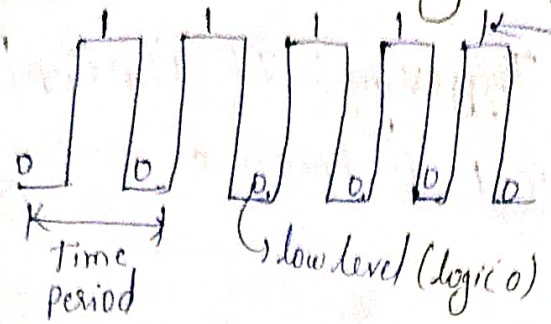
(or)

Latches and flipflops are basic memory storage elements.

Synchronous sequential circuit:-



* Pulse wave form (or) Timing diagram of clock pulses



⇒ The sum of logic 1 & logic '0' is called one complete cycle

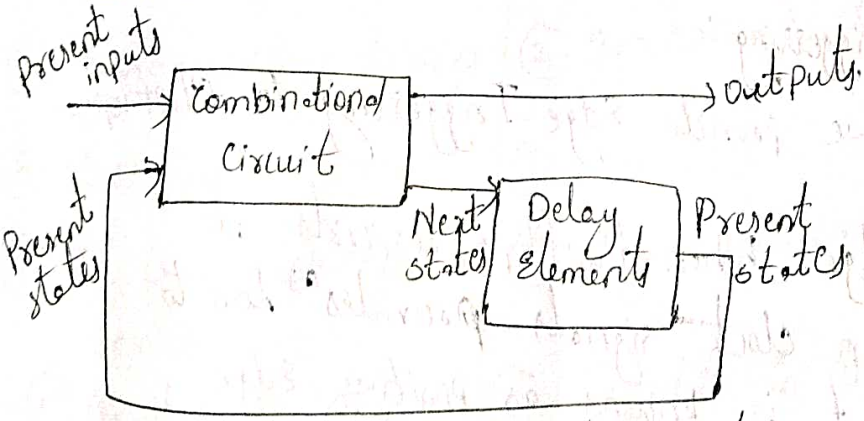
(a) ⇒ Time period is called one complete cycle

⇒ Inverse of time period is called frequency

⇒ One complete cycle is called time period

(2) Asynchronous Sequential circuit:-

Clock pulses are not used for memory devices. The "unclocked flipflops" (or) time delayed are the memory elements used. The Asynchronous sequential circuits is a combinational circuit with feedback



(1) Synchronous sequential circuits:-

* Types of Triggering:-

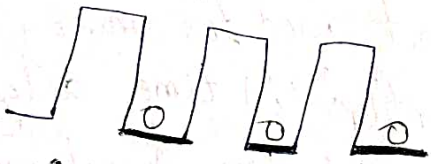
(i) Level-Triggering
(ii) Edge-triggering

> given to memory elements

(1) Level-Triggering :- (Mostly for latches)

→ Positive level-Triggering :- If flipflop / latches responds at logic 1 is known as positive level triggering

→ Negative level triggering :- If flipflop responds at logic 0 is known as negative level triggering.



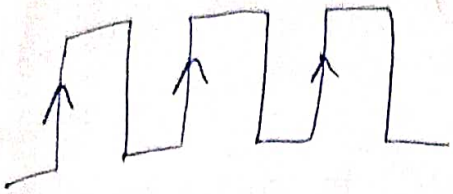
⇒ If we provide negative level triggering then flipflop / latch responds at logic 0 or low level or at negative level.

(2) Edge-Triggering :-

⇒ Mostly we provide Edge-Triggering to flipflops.

→ Positive Edge :- When flipflops responds at when clock signals provides as low to high that is knowns as positive edge triggering. (or)

When flipflop responds at low to high clock pulse signals that is known as Positive edge triggering.



→ Negative Edge:

when flipflop responds at high to low clock pulse signals is known as Negative Edge triggering.

* Latches & flipflops:

→ Latches & flipflops are the storage elements. They store 1 bit of information (either 0 or 1).

→ Differences between latches & flipflops:

Latches	Flipflops
① Do not require clock signal	① Requires clock signal
② Asynchronous devices	② Synchronous devices
③ Uses Enable signal	③ Uses clock signal
④ It is a level sensitive device	④ Edge sensitive device
⑤ Simpler to design	⑤ Difficult to design
⑥ Operation is faster	⑥ Operation is slower
⑦ Power requirement is less	⑦ Power requirement is more.

* S-R Latch (Using NOR gates)

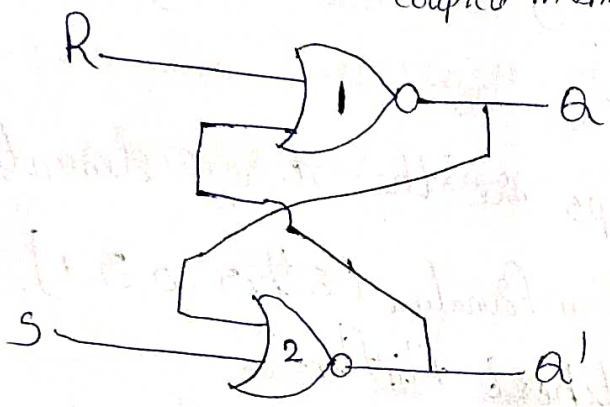
⇒ S - SET(1) → output

⇒ R - RESET(0) → output

⇒ We can design S-R Latch using NAND gates also. There that Latch is called S'R' Latch.

Diagram:- (Using NOR gates)

⇒ Here NOR gates connected in cross coupled manner.

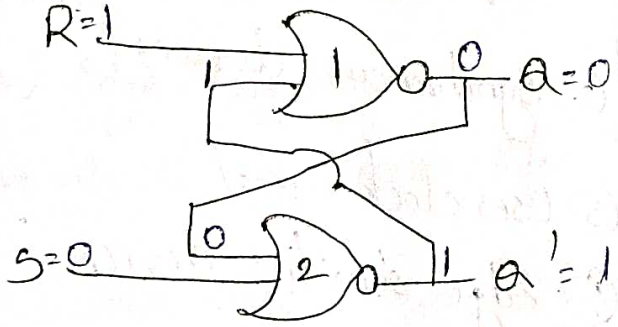


Truth Table for NOR gate

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

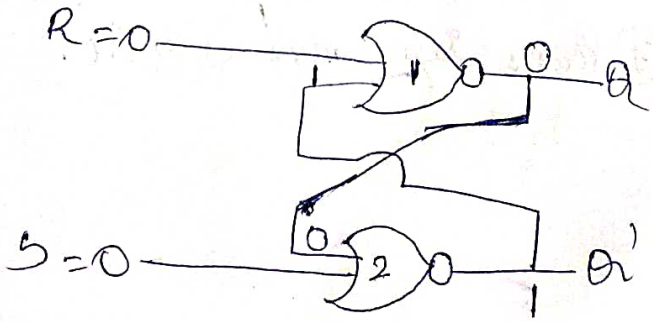
Case (i):-

S=0, R=1 ⇒ Inputs



∴ Q=0, Q'=1 → outputs

Case (ii):- To check whether it is storing or not, take S=0, R=0 (here we are removing Elements).



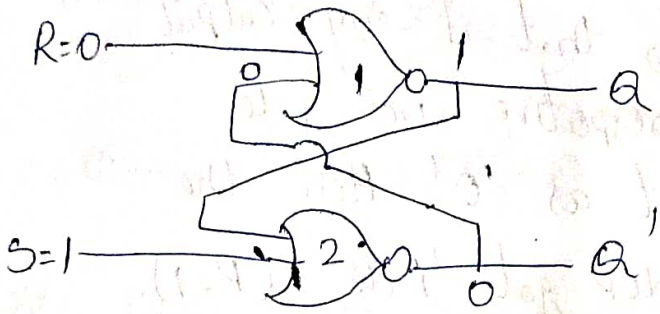
∴ Q=0 and Q'=1

⇒ Here after removing elements also it is storing previous values.

⇒ The combination of S=0 and R=0 is memory state because it is storing previous values.

Case (ii) :-

S=1, R=0



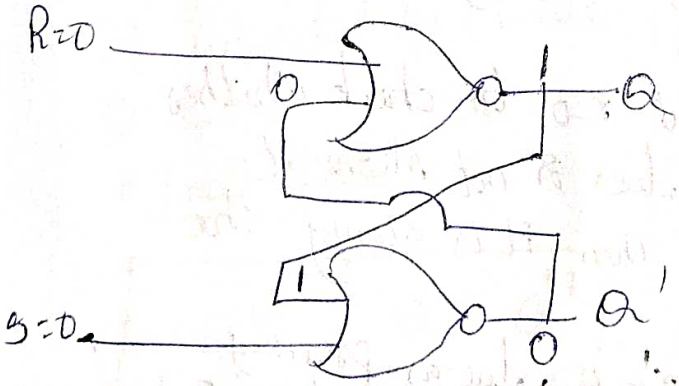
here, Q=1

Q'=0

Note: If any one input it 1 then its respective NOR gate output is 0. Here S=1, so its respective 2nd NOR gate output is 0. (Q'=0)

→ Removing elements and place

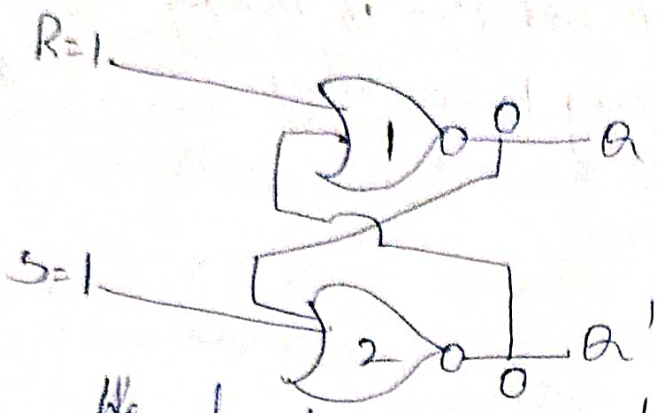
S=0, R=0



Here we take the output of Q that we got for S=1 and R=0. There the output of Q=1. By taking that we check whether we are getting same value for Q' or not. But here we get Q'=0 as previous itself. So here it is storing previous values and that why it is memory state.

Case (iii):

S=1, R=1



We already know that if any output is 1 then its respective NOR gate will give the output as '0'. Here the first input of NOR gate '1' is 1 (R=1). That's why its output is '0'.

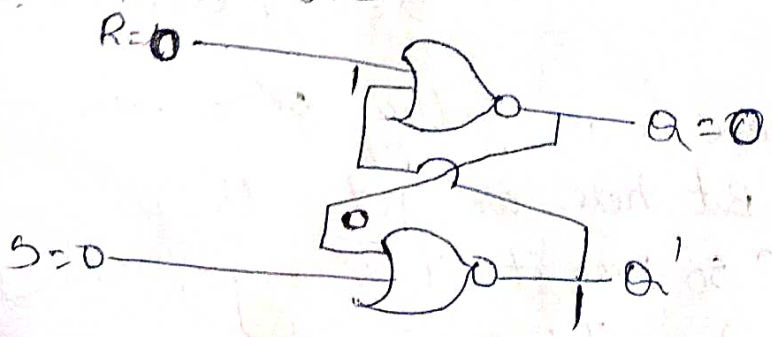
The first input of 2nd NOR gate is S=1. So its output is also '0'.

$\therefore Q = 0$ and $Q' = 0$

Here Q and Q' outputs are trying to get same value.

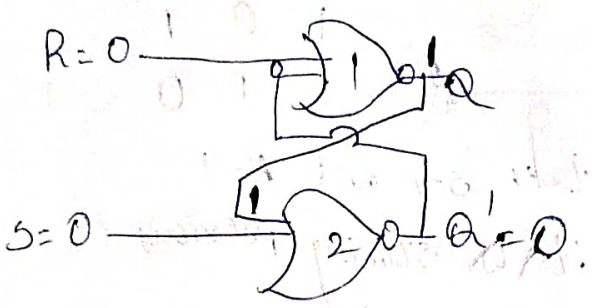
Now take S=0 and R=0 to check whether it is storing the values or not. Means it gives same outputs then it is storing the values.

∴ S=0 and R=0. It take its Q value as previous output of Q. i.e. Q=0



Here $Q=0$ and $Q'=1$. So, here it is not storing previous values. Here we get unpredictable results.

Now take Q' values as its previous output i.e $Q'=0$ when $R=1$ and $S=1$.



Here $Q=1, Q'=0$

Here also we are getting unpredictable values.

⇒ In Digital circuits, we should not get unpredictable values. So the input combination of $R=1$ and $S=1$ should be avoided. we can avoid this $R=1$ and $S=1$ combination using SR latch with Enable input. (In this way, we can avoid that result in small manner)

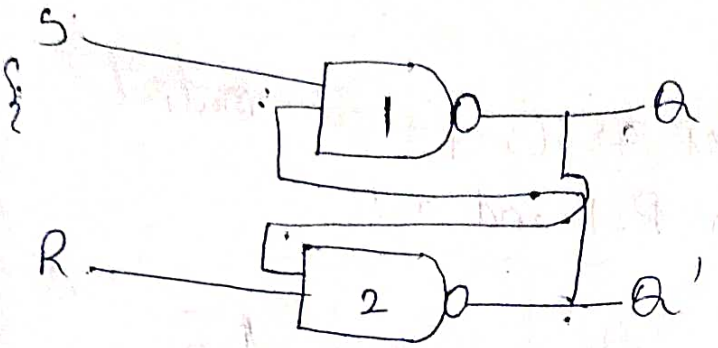
⇒ To avoid $R=1$ and $S=1$ combination's outputs we can use completely as unpredictable value

D latch.

* Truth Table:

S	R	Q	Q'	
0	0	Memory state (values as before)		
0	1	0	1	→ Reset state
1	0	1	0	→ set state
1	1	Not used (or) invalid state		

SR Latch Using NAND gates:

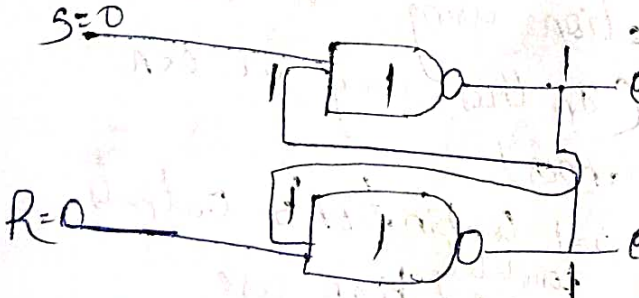


Truth Table for NAND gates

x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

⇒ Here we have to take S=1 and R=1 to check whether it is storing previous values or not in Every case.

Case(i): S=0 and R=0



Here Q=1 and Q'=1.

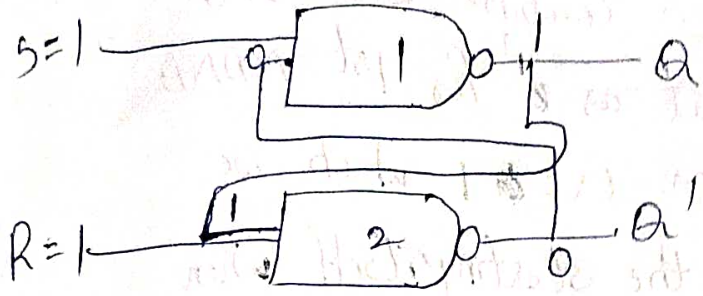
⇒ Here in NAND gates if any one output is 0 then its output is 1. Here for 1st NAND gates one of the input is S=0, so its output is 1. i.e Q=1 and for 2nd NAND gates one of the input is R=0. So its output is 1. i.e Q'=1

⇒ Now check for S=1 and R=1 to check whether it is storing previous values or not.

⇒ S=1 and R=1.

(i) Take Q value as its previous output.

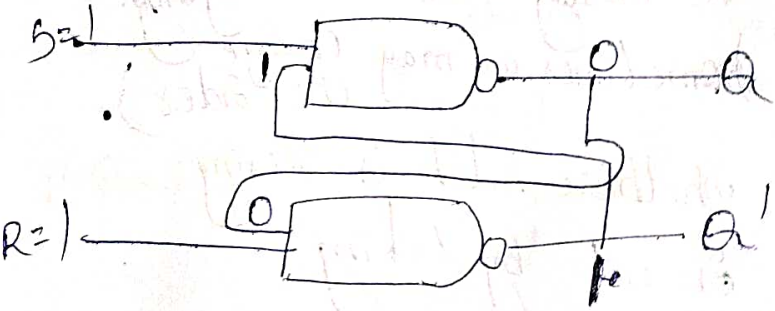
∴ Q=1



Here Q=1 and Q'=0. It is unpredictable value (Invalid)

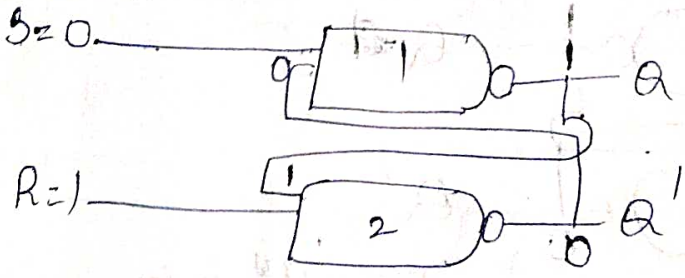
(ii) Take Q' value as its previous output.

Q'=1.



Here Q=0 and Q'=1. It is not storing the previous values what we get for S=0 and R=0 so it is unpredictable value (Invalid).

Case-(ii):- R=1 and S=0



outputs:

- Q=1
- Q'=0

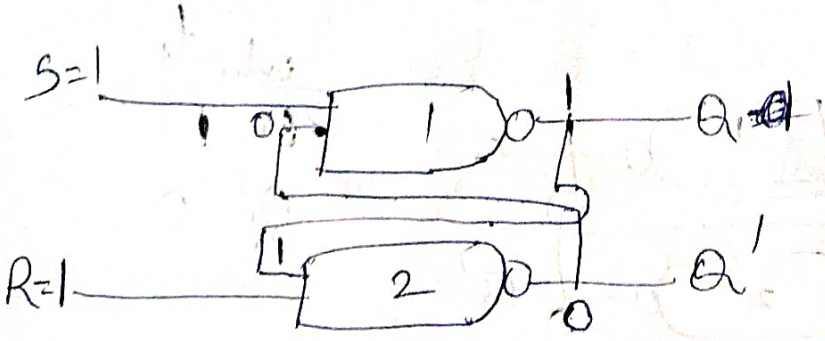
We know that if any one input is 0 then its output is '1'. Here one of the input of 1st NAND gate is '0'. so it's output

will be 1, automatically. That means Q=1.
Combing that Q=1 with second NAND gates first output R=1 we get output as 0 for second NAND gate. That means Q'=0. That, Q'=0 combine with S=0 and give output as 1 for 1st NAND gate. That means Q=1, which we get already in the starting itself when we directly found the output considering its first input itself. (This is the procedure that we mostly used for Every case here. Sometimes we may change the sides).

⇒ Now check whether latch is storing previous values or not by taking S=1 and R=1.

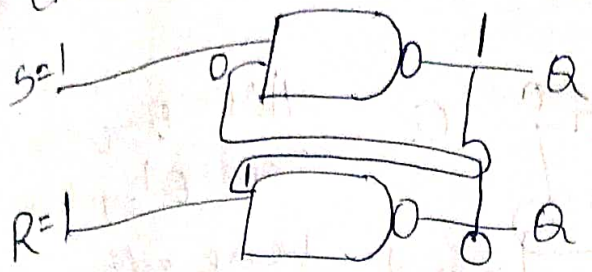
(i) Take Q=1 and check.

{ we get previously Q output value as 1. that's why we take Q=1 }



Here Q=1 and Q'=0. Here it is storing previous values as it is. So it is set state.

(ii) Take $Q' = 0$ and check.

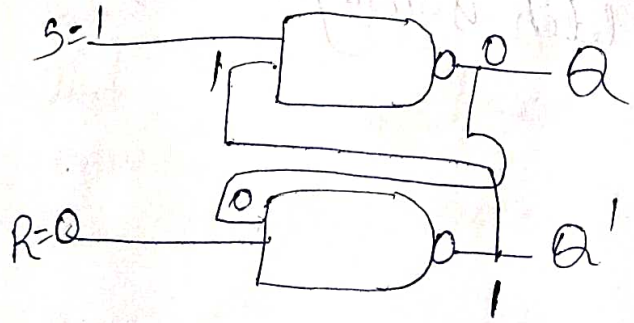


Here $Q=1$ and $Q'=0$. Here also it is storing same ~~values~~ ~~previous~~ as it is previous values.

$\therefore$ $S=0$ and $R=1$ is a set state.

Case (iii):-

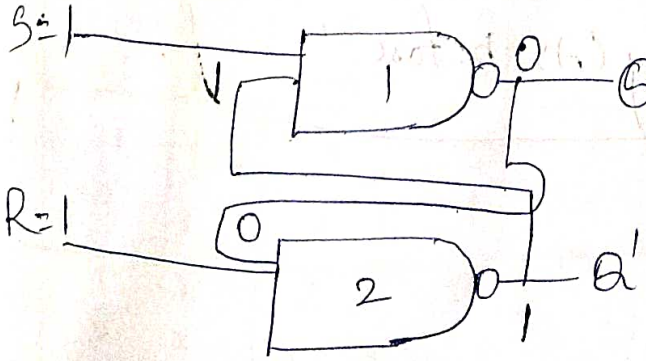
$S=1$ and $R=0$



Here $R=0$. So its respective NAND gate value is '1'. Using that output we found Q value by combing $Q'=1$ with $S=1$ $\therefore$ $Q=0$ and $Q'=1$ here.

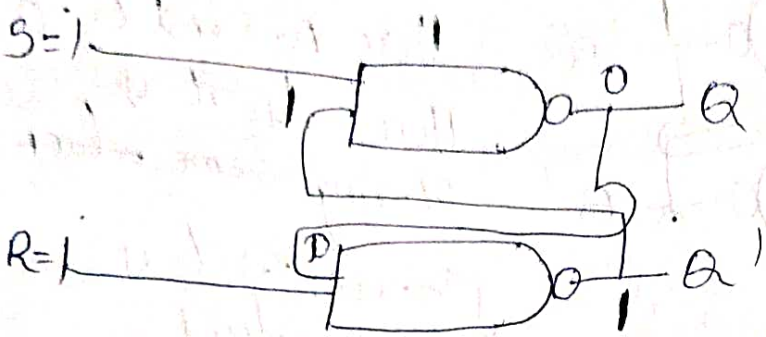
⇒ Now check using $S=1$ and $R=1$.

(i) Take $Q=0$



Here $Q=0$ and $Q'=1$.

(ii) Take $Q' = 1$.



Here $Q = 0$ and $Q' = 1$. We are getting same values.

So, it is storing previous values.

And it is reset state. (i.e. if $Q = 0$ when it is storing same values as before then it is Reset state. If $Q = 1$ then it is SET state if and only if the latch is storing previous values).

Truth Table:-

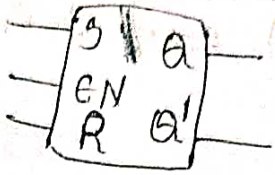
S	R	Q	Q'
0	0	Not used (Invalid)	
0	1	1	0 (set state)
1	0	0	1 (Reset state)
1	1	Memory (As: before)	

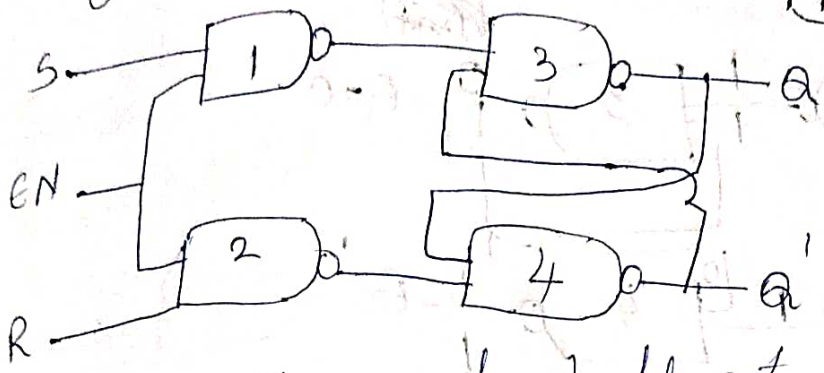
* Gated SR Latch (SR latch with Enable)

* Input:

* Logic diagram:

Logic symbol:





⇒ In this the inputs should not reach the output. There should be some control. That control can be done by the Enable (EN) input. So, Enable input works as controlling power. (Gated means controlling power).

⇒ Truth table:

EN	S	R	Qn	Qn+1	state
1	0	0	0	0	No change (NC)
1	0	0	1	1	
1	0	1	0	0	Reset
1	0	1	1	0	
1	1	0	0	1	set
1	1	0	1	1	
1	1	1	0	X	Indeterminate
1	1	1	1	X	
0	X	X	0	0	No change (NC)
0	X	X	1	1	

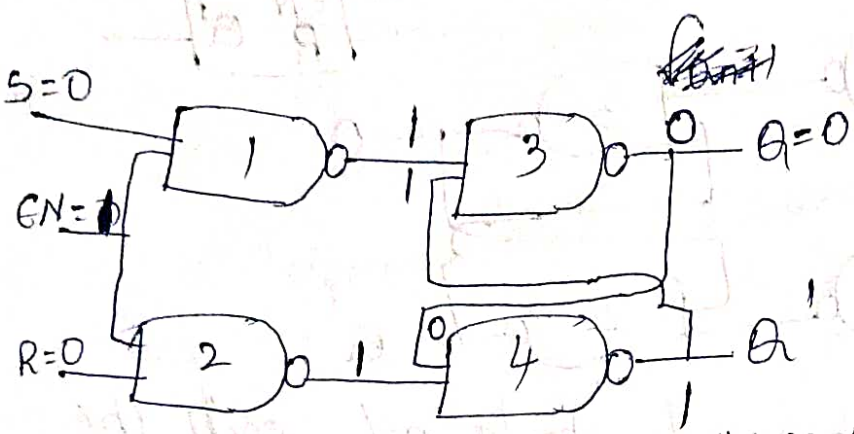
Here, Qn = Present state

Qn+1 = Next state

Verification:

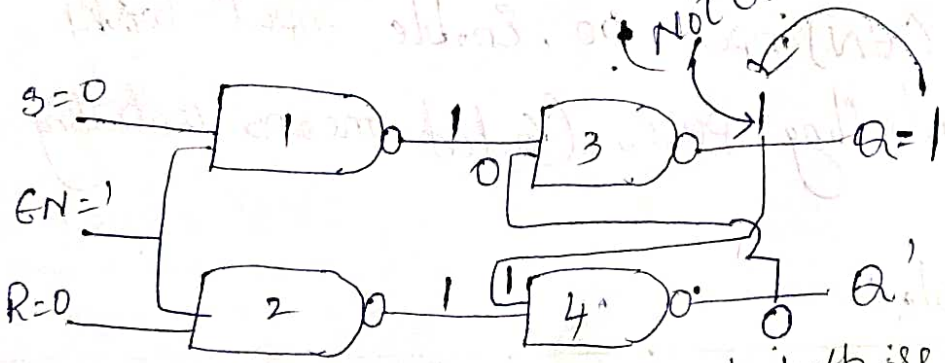
Case (i):

(i) EN=0, S=0, R=0, Qn=0 (Qn value will assigned to Q)



Qn=0 and Qn+1=0. Both are same so No change

(ii) EN=1, S=0, R=0, Qn=1



Qn=1, and Qn+1=1 and both are same so No change.

∴ It is No change state.

1st NAND gate output is the combination of S and EN.

2nd NAND gates output is the combination of EN and R.

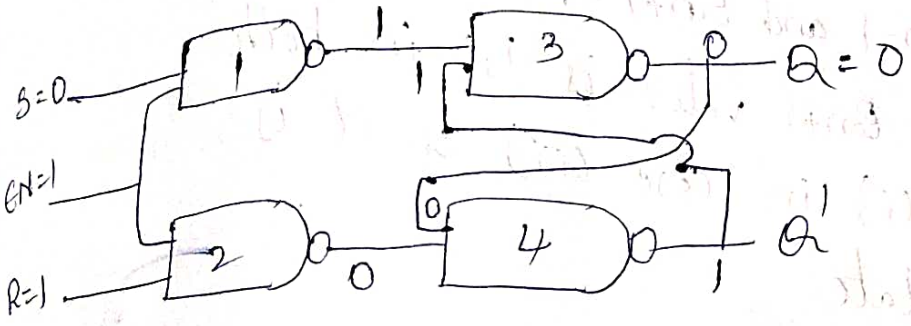
4th NAND gate output is the combination of Q value and second gate output.

3rd NAND gates output is the combination of 4th NAND gate output and 1st NAND gate output and that output

is considered as Q_{n+1}. We don't mention it in the diagram. We directly take that value. This is the procedure and order we mostly followed here.

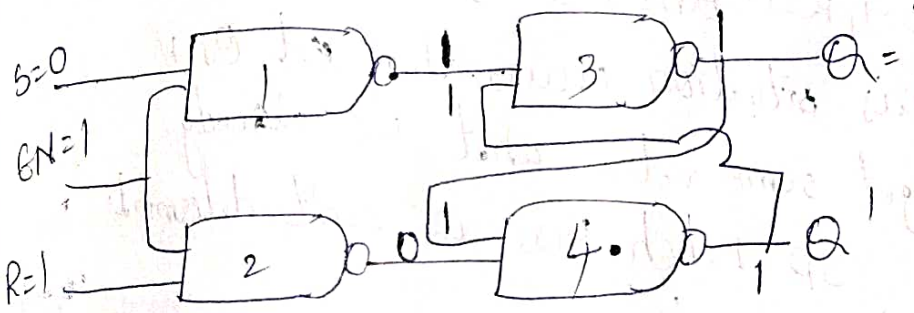
Case (ii):-

(i) $EN=1, S=0, R=1, Q_n=0$



$Q_n=0$ and $Q_{n+1}=0$.

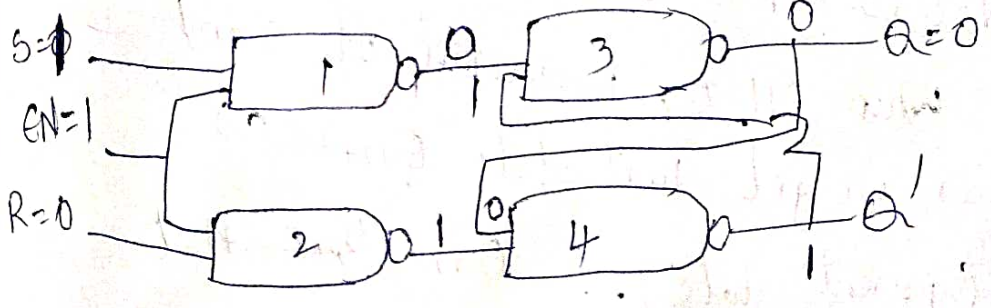
(ii) $EN=1, S=0, R=1, Q_n=1$



$Q_n=1$ and $Q_{n+1}=0$

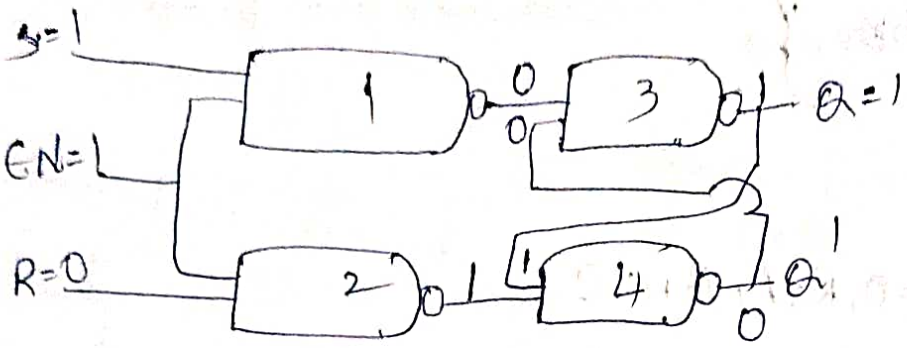
∴ Case (ii) is Reset state because Q_{n+1} ($Q_{n+1}=0$) values are same though we change Q_n value.

Case (iii):- (i) $EN=1, S=1, R=0, Q_n=0$



$Q_n = 0$ and $Q_{n+1} = 1$

(ii) $EN = 1, S = 1, R = 0, Q_n = 1$



$Q_n = 1$ and $Q_{n+1} = 1$.

Here Q_{n+1} value is 1, 1 in both (i) and (ii) in case (iii). So it is SET state.

Case (iv):

i) $EN = 1, S = 1, R = 1, Q_n = 0$

ii) $EN = 1, S = 1, R = 1, Q_n = 1$

} Q_n this both cases $S = 1$ and $R = 1$.

When this situation occurs then Q_{n+1} tries to get same value what we already saw in SR Latch. Here we can't determine Q_{n+1} values exactly.

So, It is intermediate state.

[ENABLE used to avoid combination of $S = 1$ and $R = 1$ but not to eliminate]

Case (v):

(i) $EN = 0, S = X, R = X, Q_n = 0$

(ii) $EN = 0, S = X, R = X, Q_n = 1$

When $EN = 0$, irrespective of S and R values we get next state Q_{n+1} as Q_n (present state). $\therefore$ $Q_n = 0$ then $Q_{n+1} = 0$

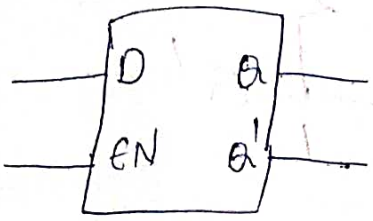
$Q_n = 1$ then $Q_{n+1} = 1$

we verified

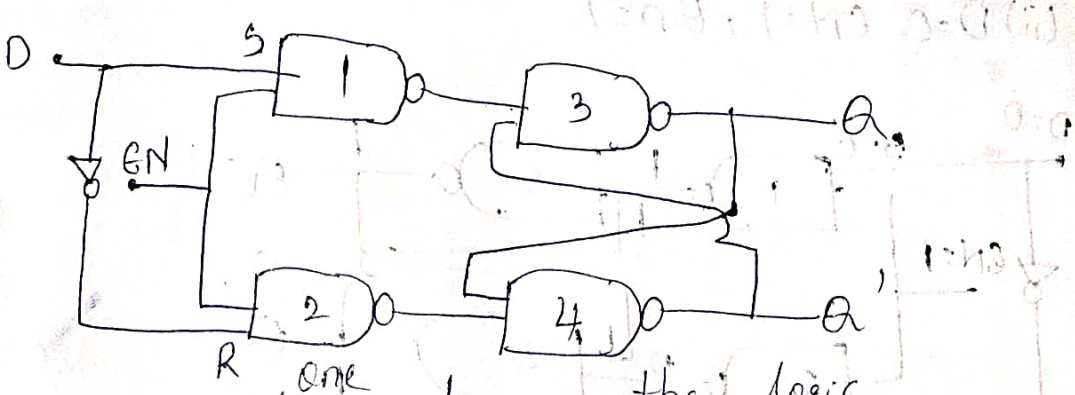
⇒ Now we are going to discuss about D latch which is used to eliminate that "indeterminate state.

* Gated D-Latch (D means Data) (Data Latch)

⇒ Logic symbol:-



⇒ Logic diagram:-



⇒ Here we add inverter in the logic diagram when compared with SR Enable Latch.

⇒ We add inverter in order to Eliminate S=1 and R=1 indeterminate state. And for that reason only we take D values.

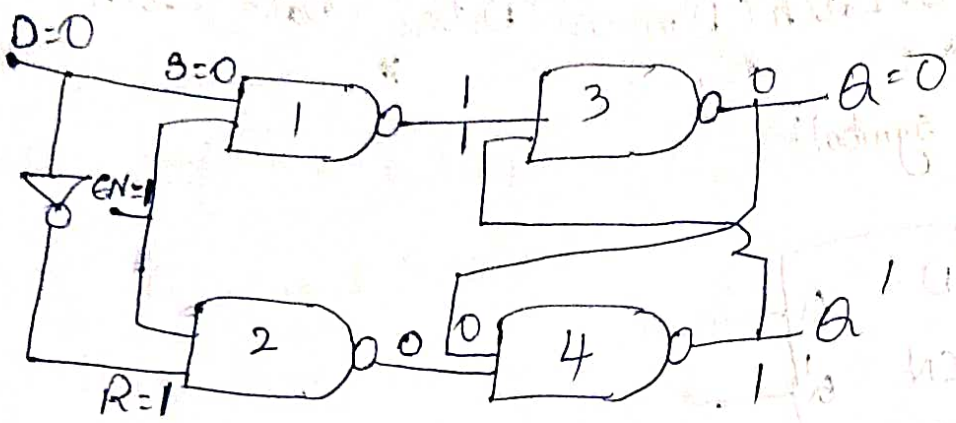
⇒ In D latch or D flipflop, the output always follows input.

Qn+1 is D is input and outputs.

Case (i) :-

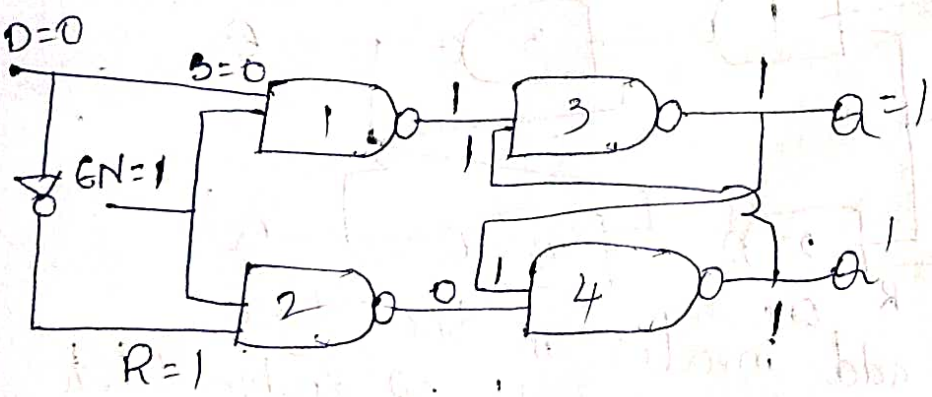
D = 0, EN = 1 . Q_n = X (Either 0 or 1)

(i) D = 0, EN = 1, Q_n = 0 (Q_n value will assigned to Q here)



here Q_n = 0 and Q_{n+1} = 0

(ii) D = 0, EN = 1, Q_n = 1



Q_n = 1 and Q_{n+1} = 0

Here for Q_n = 0 and Q_n = 1 the Q_{n+1} values is '0' only.

∴ Q_n = X and Q_{n+1} = 0 : In both cases $Q_{n+1} = D$ → Re st

⇒ S value is equal to D value.

⇒ R value is equal to compliment of D.

i.e, D = 0 then R = 1

D = 1 then R = 0 because inverter is placed from D to R.

⇒ 1st NAND gate output is the combination of S and EN values.

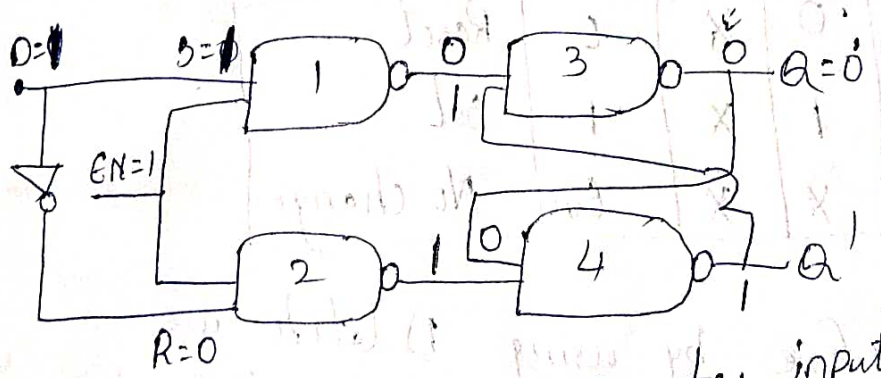
⇒ 2nd NAND gate output is the combination of EN and R values.

⇒ 4th NAND gate output is the combination of Q and 2nd NAND gate output values.

⇒ 3rd NAND gate output is the combination of 4th NAND gate output and 1st NAND gate output. And that 3rd NAND gate output will consider as Q_{n+1}.

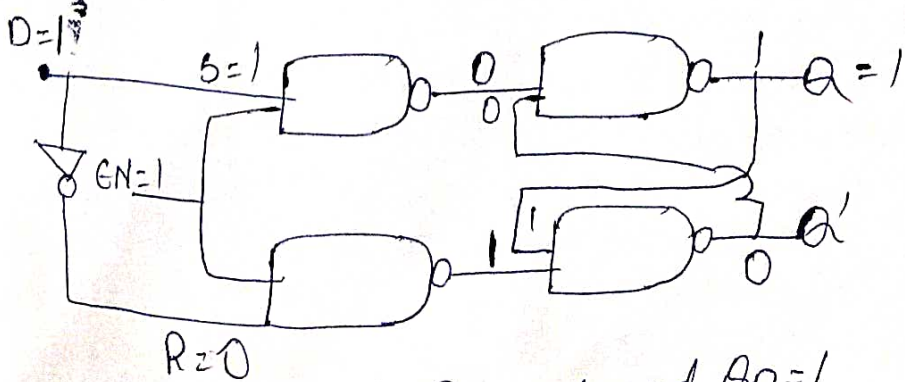
Case(ii):- EN=1, D=1, Q_n=X (Either 0 or 1)

(i) EN=1, D=1, Q_n=0



Here for 3rd NAND gates inputs are 0, 1. So output Q_{n+1} = 1.

(ii) EN=1, D=1, Q_n=1



∴ Q_{n+1}=1 and Q_n=1

in (i) & (ii) in case (iii),

$Q_{n+1} = D = 1$

So it is set state

Case (iii)

EN=0, D=X (0 or 1); ~~Qn (0 or 1)~~ Qn=X (0 or 1),

~~Qn+1~~

(i) EN=0, D=0, Qn=0 then $Q_{n+1}=0$

(ii) EN=0, D=0, Qn=1 then $Q_{n+1}=1$

(iii) EN=0, D=1, Qn=0 then $Q_{n+1}=0$

(iv) EN=0, D=1, Qn=1 then ~~Qn+1=1~~.

Truth Table:-

EN	D	Qn	Qn+1	State
1	0	X	0	Reset
1	1	X	1	set
0	X	X	Qn	No change

∴ Therefore, by using D latch we can totally avoid indeterminate state (S=1 and R=1).

Flip flops:

⇒ Flip flops are constructed from "latches".

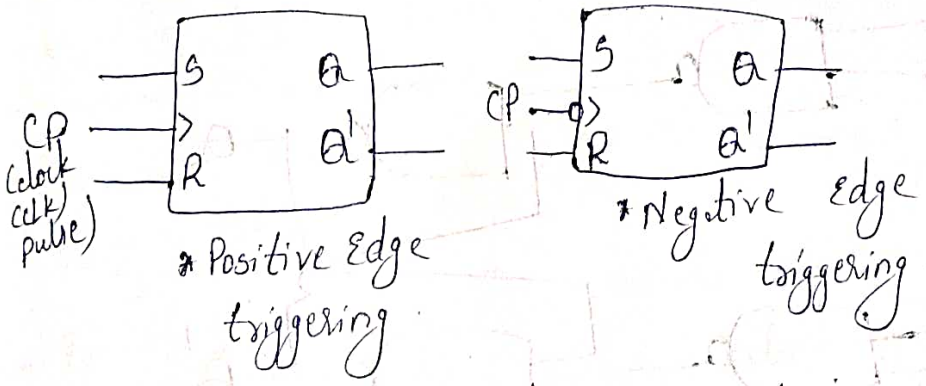
⇒ We apply clock pulse for the flip flops.

⇒ Flip flops contains edges. i.e positive edge triggering and Negative edge triggering.

⇒ We can construct SR Flipflops, D flip flops, J k flip flops and T Flipflops.

* SR flip flops :-

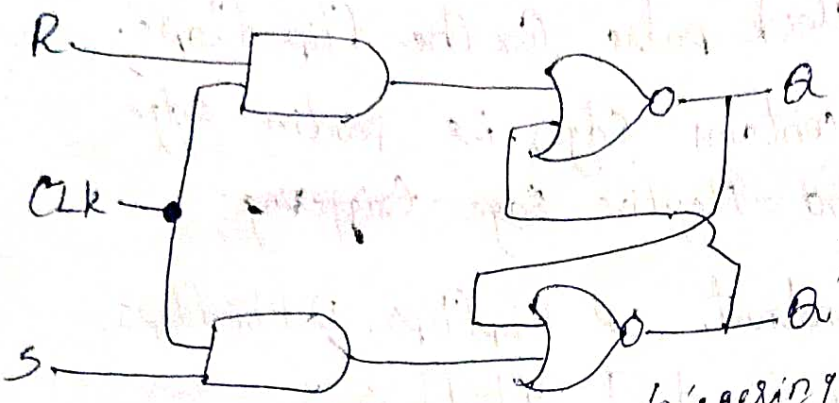
⇒ Logic symbol :-



⇒ If we place CP as '→' then it is positive edge triggering.

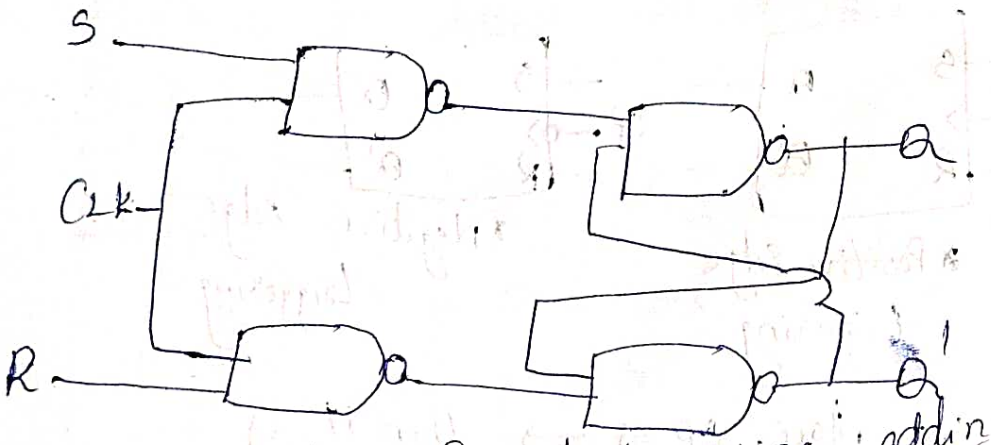
⇒ If we place CP as '—○' then it is Negative edge triggering. Here bubble is connected.

* Logic diagram using NOR gates:



⇒ It is Negative Edge triggering.

* Logic diagram using NAND gates:-



⇒ This circuit is formed by ~~using~~ adding two NAND gates to NAND based SR flip-flop. The inputs are active high as the Extra NAND gate inverts the inputs. A clock pulse is given as input to both the Extra NAND gates.

⇒ Assuming it is a positive Edge triggering device, the truth table for this flip-flop is shown as follows:-

* Truth table:

CLK	S	R	Qn	Qn+1	State
↑	0	0	0	0	No change (NC)
	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	X	Indeterminate
	1	1	1	X	
↓	X	X	0	0	No change (NC)
	X	X	1	1	

⇒ The same truth table can be achieved by using NOR gates also.

→ In truth table, ↑ represents positive edge triggering. i.e, CLK (clock pulse) = 1.

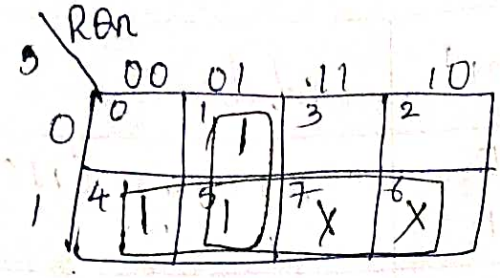
↓ represents Negative edge triggering. i.e, CLK (clockpulse) = 0

* Simplification using k-map:

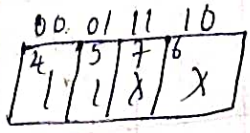
Take S, R, Qn inputs and Qn+1 output.

S.NO	S	R	Qn	Qn+1
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	X
7	1	1	1	X

$Q_{n+1} = \Sigma(1,4,5) + d(6,7)$



Group-1



$\Rightarrow S$

Group-2

$\Rightarrow R'Q'$

$$Q_{n+1} = S + R'Q_n$$

↳ This is the characteristic Equation (or) Output Equation of both SR latch and SR flipflop.

D Flip flop:

* Logic symbol:

clock may be → (or) —○>

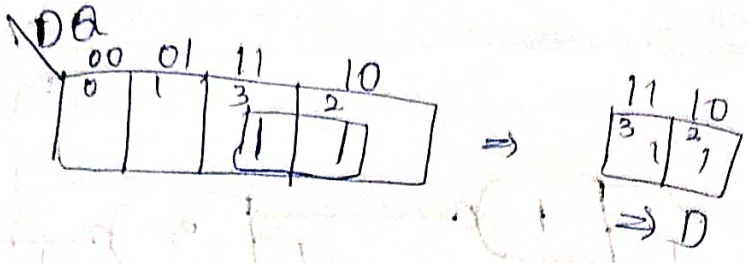
⇒ Let us consider clock is '→' means positive edge triggering then,

Truth table:

CLK	D	Q	Q_{n+1}
↑	0	0	0
↑	1	0	1
↑	0	1	0
↑	1	1	1

K-map simplification:-

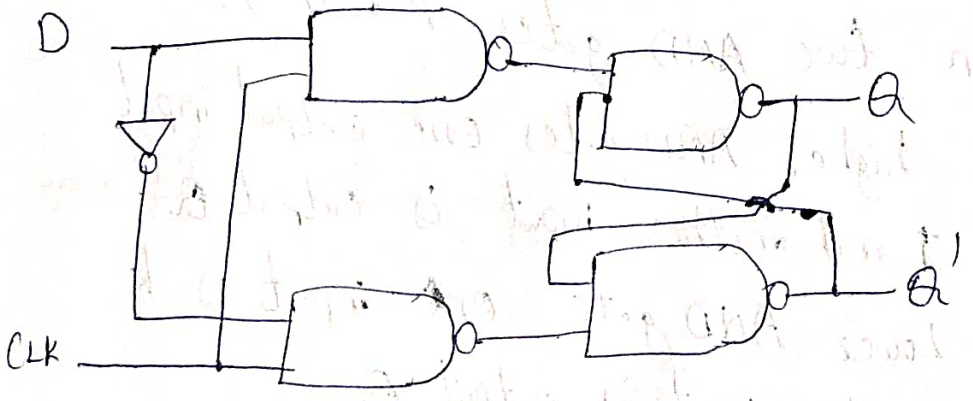
$Q_{n+1} = \Sigma(2,3)$



$\therefore$ **$Q_{n+1} = D$**

⇒ We can predict this directly from the Truth table itself. Q_{n+1} is equal to 'D' in truth table.

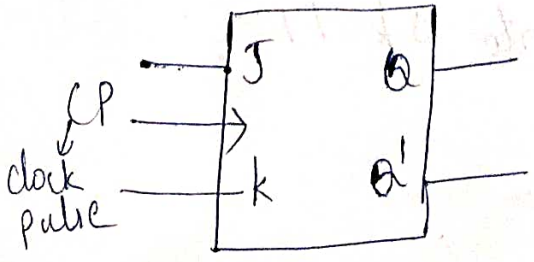
Logic diagram:- (Using NAND gates)



JK flip flop:-

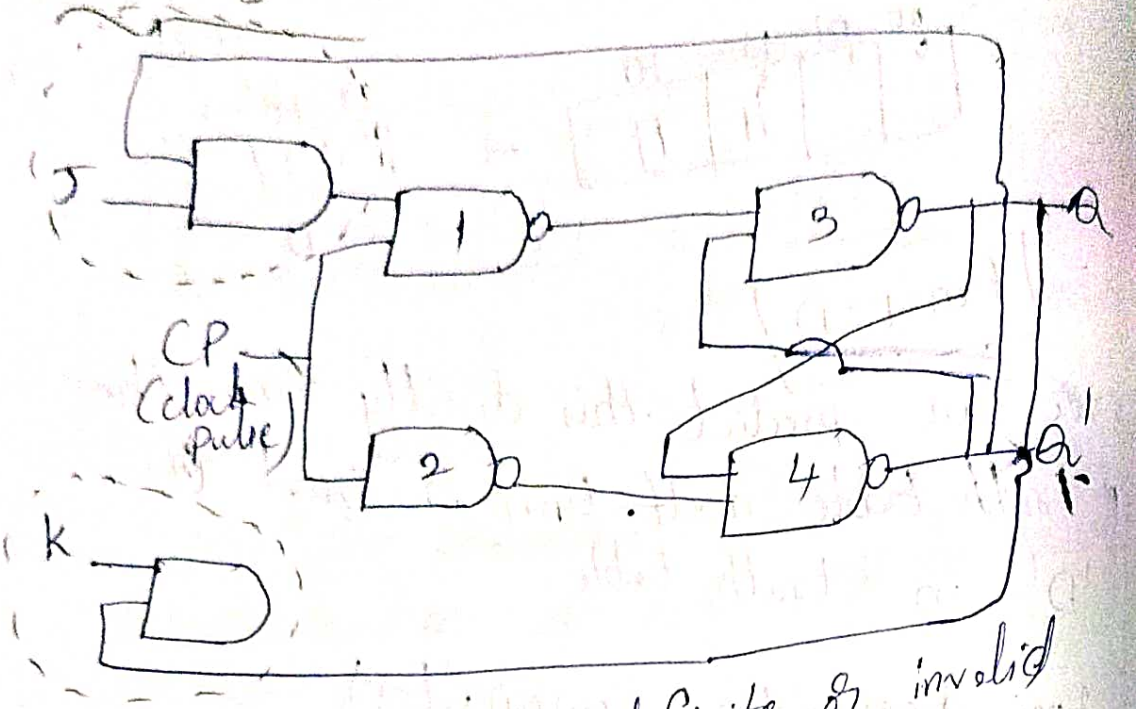
⇒ Jk flip flop is constructed from SR slip flop & from D flipflip.

* Logic Symbol:-



Here J, k are inputs.
Q, Q' are outputs.

Logic diagram:



⇒ In order to avoid indefinite or invalid input values (R=1 and S=1) we have taken two AND gates k and J.

⇒ For higher AND gates one output input is J and another input is output Q'.

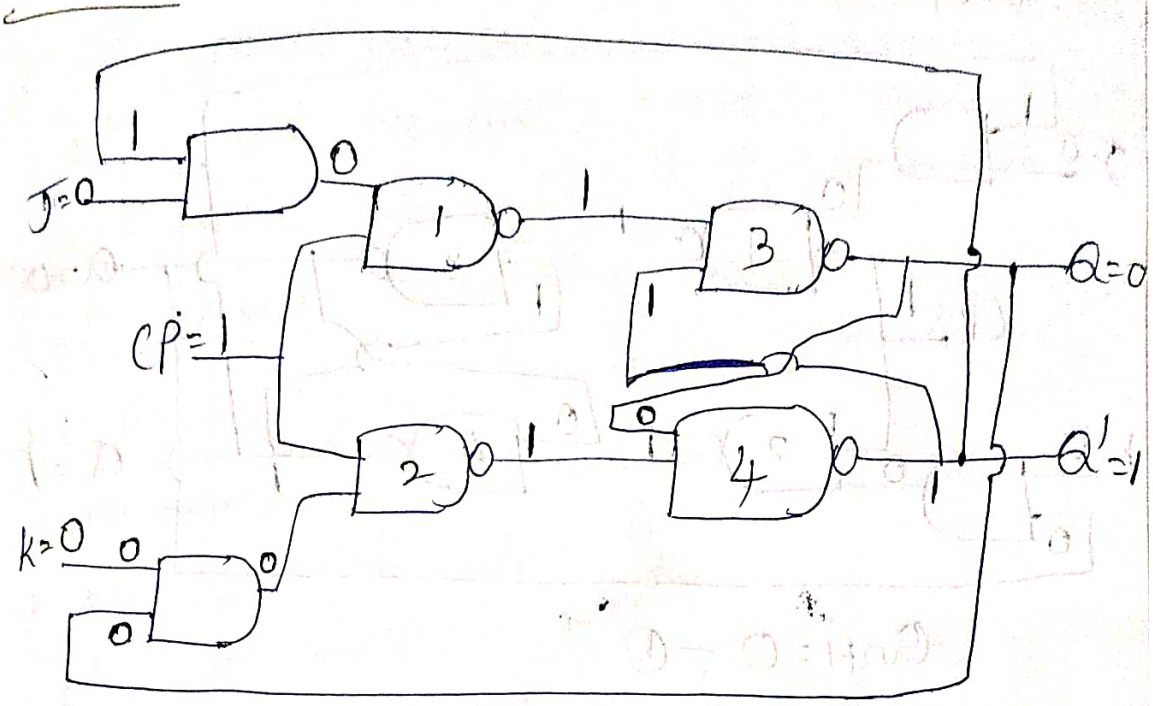
⇒ For Lower AND gate one input is k and another input is output Q.

⇒ Here we are applying Positive edge triggering. So we get output at positive edge only. So, CP= here.

⇒ * The uncertainity in the state of SR FF when S=R=1 can be eliminated by Converting SR FF into JK FF.

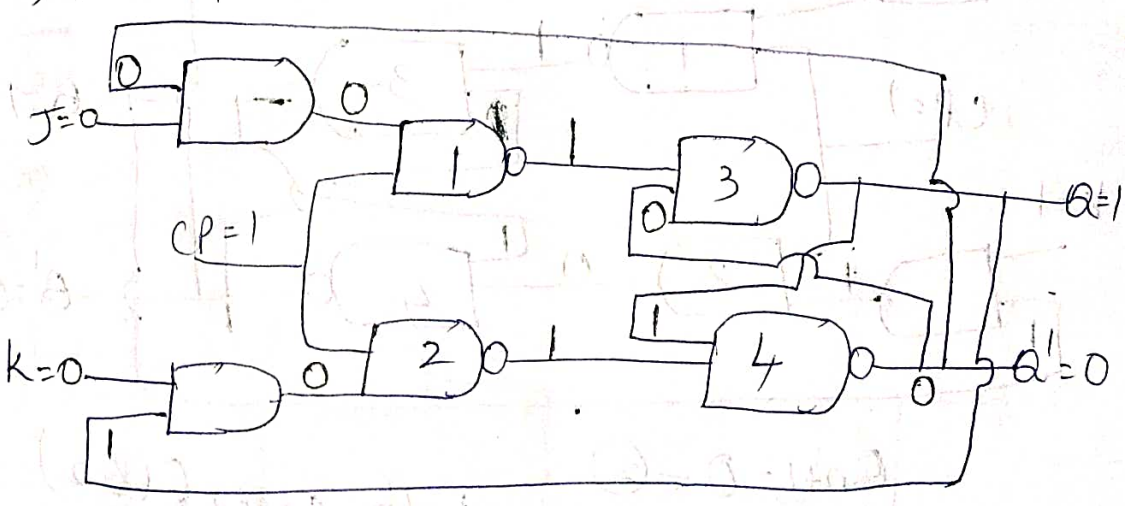
Operations:

Case (i):- 1) $J=k=0$, $Q_n=0$ (Present state)



$Q_{n+1}=0$ (next state).

2) $J=k=0$, $Q_n=1$.

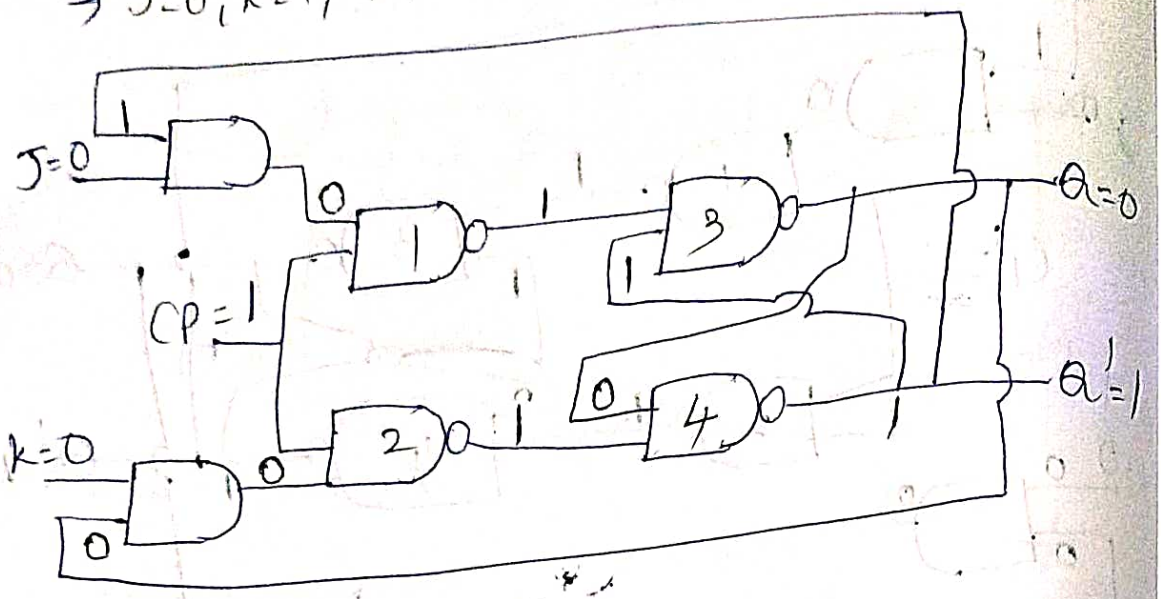


$Q_{n+1}=1$

$\therefore$ $J=0, k=0$, $Q_n=0, Q_{n+1}=0$ } No change.
$Q_n=1, Q_{n+1}=1$ $Q_n=Q_{n+1}$

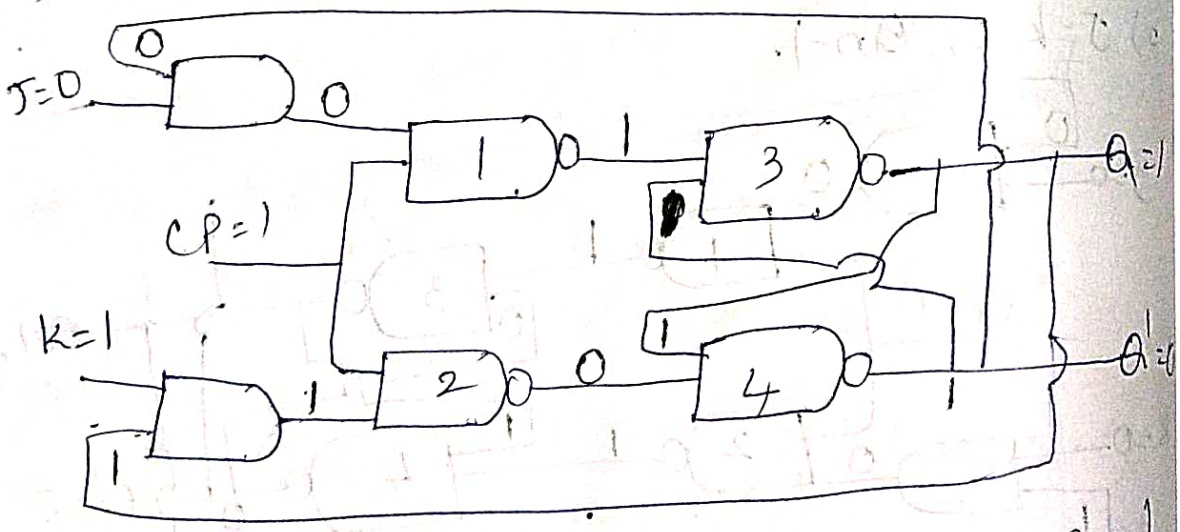
Case (ii):

⇒ J=0, k=1, Qn=0



Qn+1 = 0 — ①

⇒ J=0, k=1, Qn=1



Qn+1 = 0 — ②

∴ Qn+1 is zero in both cases (1 & 2)

So, it is Reset state

⇒ here Q value is Equal to Qn·Q' is complement of Q'. i.e. Q=0 then Q'=1
Q=1 then Q'=0

⇒ k combines with Q and gives its respective AND gate output.

⇒ J combines with Q' and gives its respective AND gates output.

⇒ 1st NAND gate output is the input combination of CP and J's AND gate output.

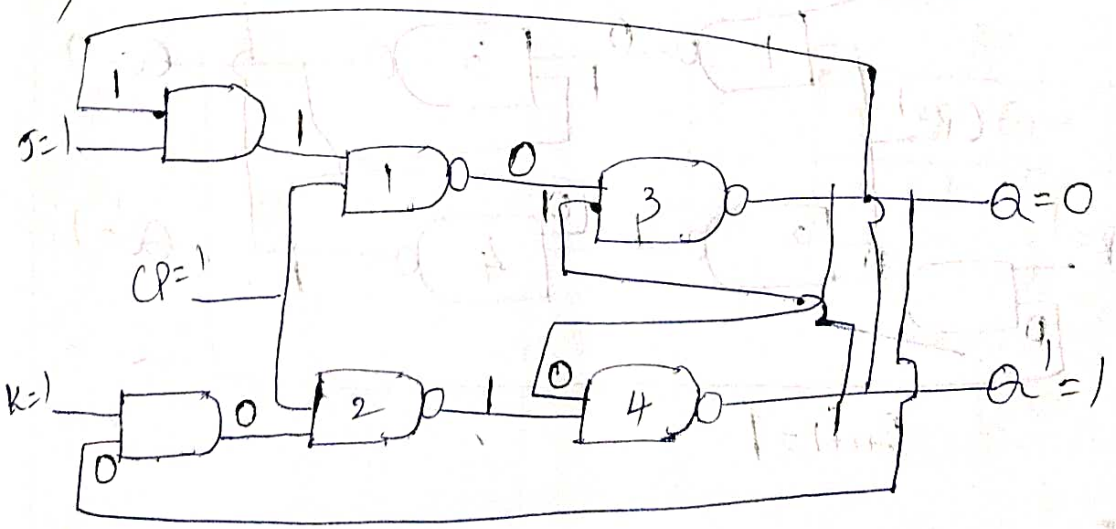
⇒ 2nd NAND gate output is the combination of CP and k's AND gate output.

⇒ 4th NAND gate output is the combination of 2nd NAND gate output and Q.

⇒ 3rd NAND gate output is the combination of 4th NAND gate output and 1st NAND gate output. And that 3rd NAND gate output is Q_{n+1} (Next state). }

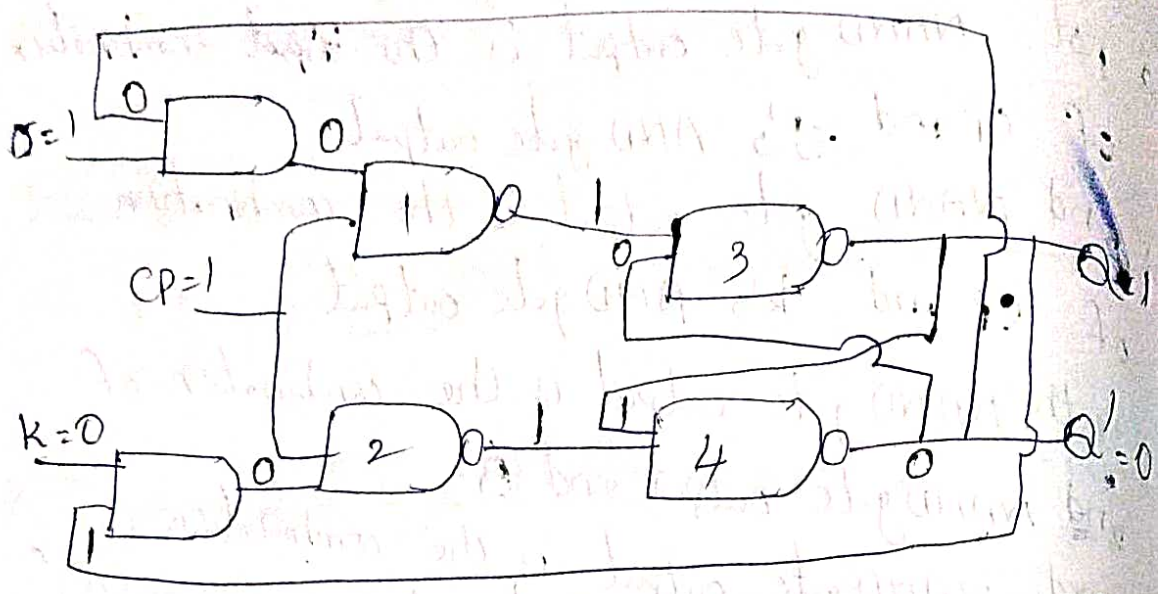
Case (iii):-

⇒ J=1, k=1, Q_n=0



$Q_{n+1} = 1$ —①

⇒ J=1, k=0, Qn=1

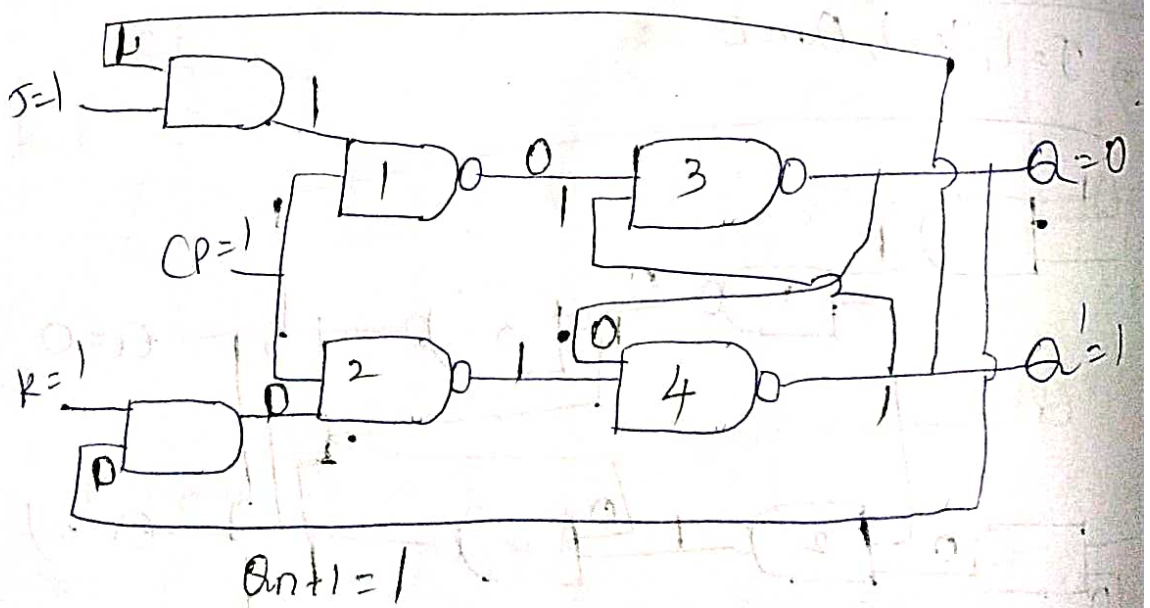


Qn+1 = 1 — ②

In both ① & ② Qn+1 = 1. So it is SET state.

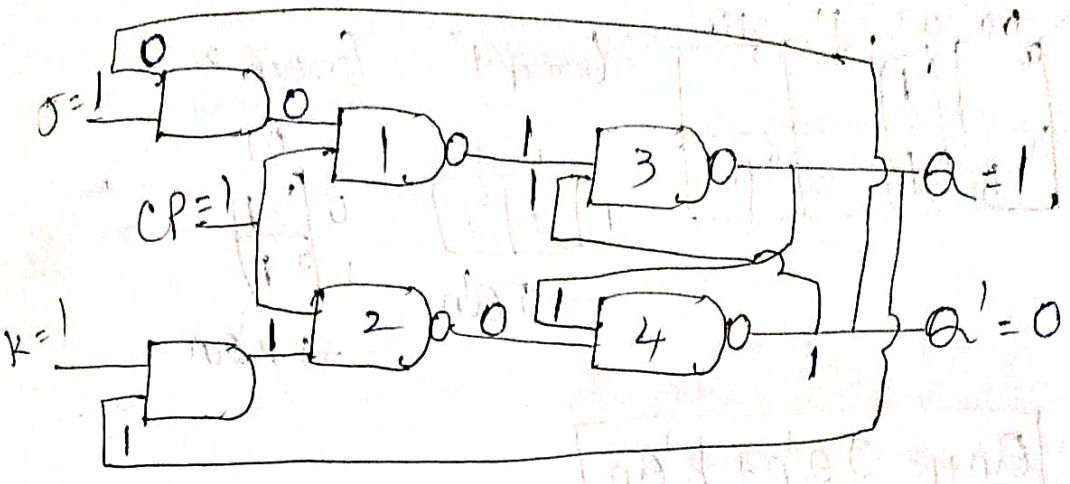
Case (iv):

⇒ J=1, k=1, Qn=0



Qn+1 = 1

⇒ ~~Qn=1~~ J=1, k=1, Qn=1.



Qn+1 = 0.

∴ Qn = 0 → Qn+1 = 1 } Toggling (Toggle state)
Qn = 1 → Qn+1 = 0 } (Inverting)

⇒ Here uncertainity of J=1 and k=1 is eliminated. We didn't get any indeterminate state.

Truth Table:

CP	J	k	Qn	Qn+1
↑	0	0	0	0
↑	0	0	1	1
↑	0	1	0	0
↑	0	1	1	0
↑	1	0	0	1
↑	1	0	1	1
↑	1	1	0	1
↑	1	1	1	0

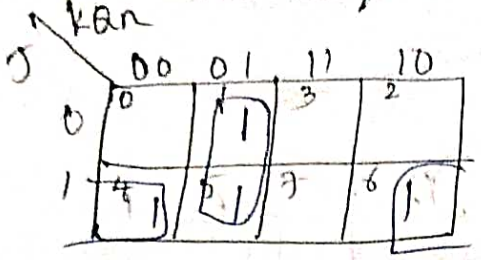
⇒

* Simplified Truth Table.

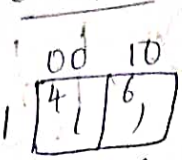
J	k	Qn+1
0	1	Qn
0	1	0
1	0	1
1	1	Qn'

Qn+1 = Σ(1, 4, 5, 6)

* Characteristic Equation using k-map:

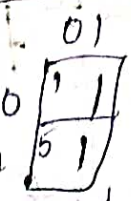


Group-1:



$\Rightarrow J\bar{Q}_n$

Group-2:



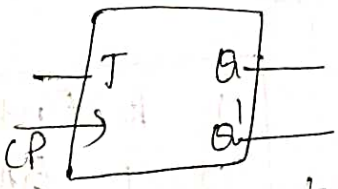
$\Rightarrow K'Q_n$

$\therefore \boxed{Q_{n+1} = J\bar{Q}_n + K'Q_n}$

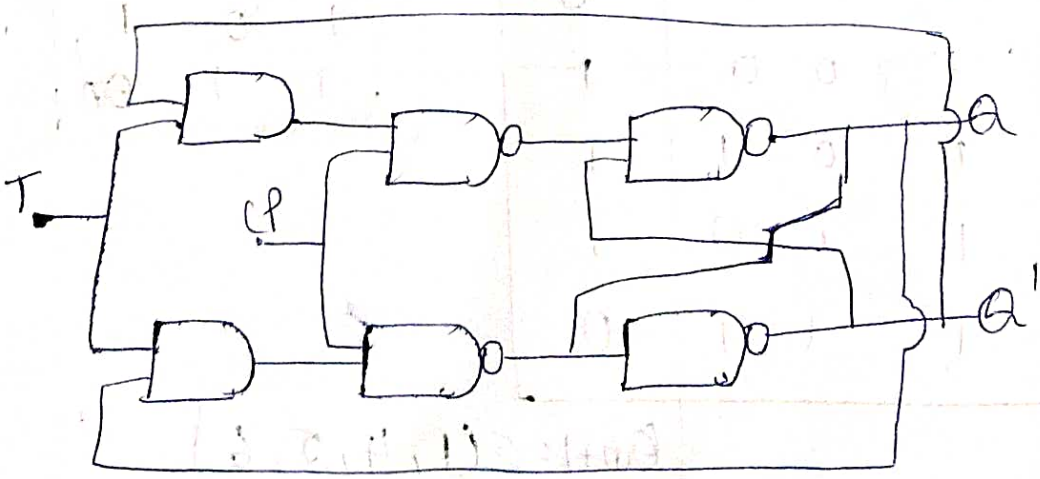
* T-flip flop (Toggle FF):

⇒ By combining JK flip flop we are going to give common input T and we are constructing T-flip flop with that.

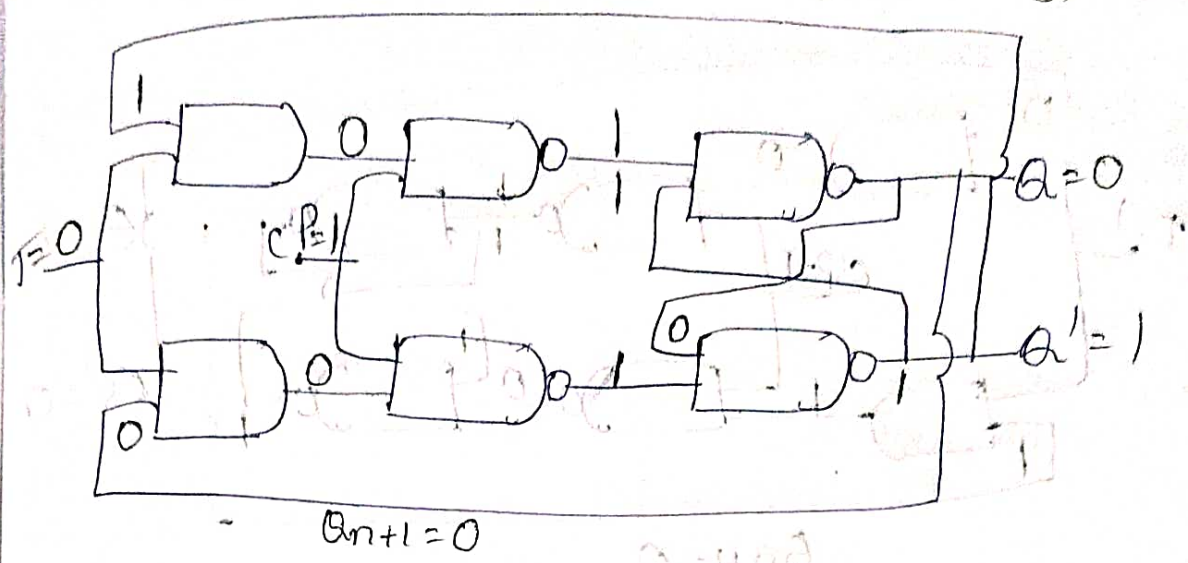
* Logic symbol:



* Logic diagram:

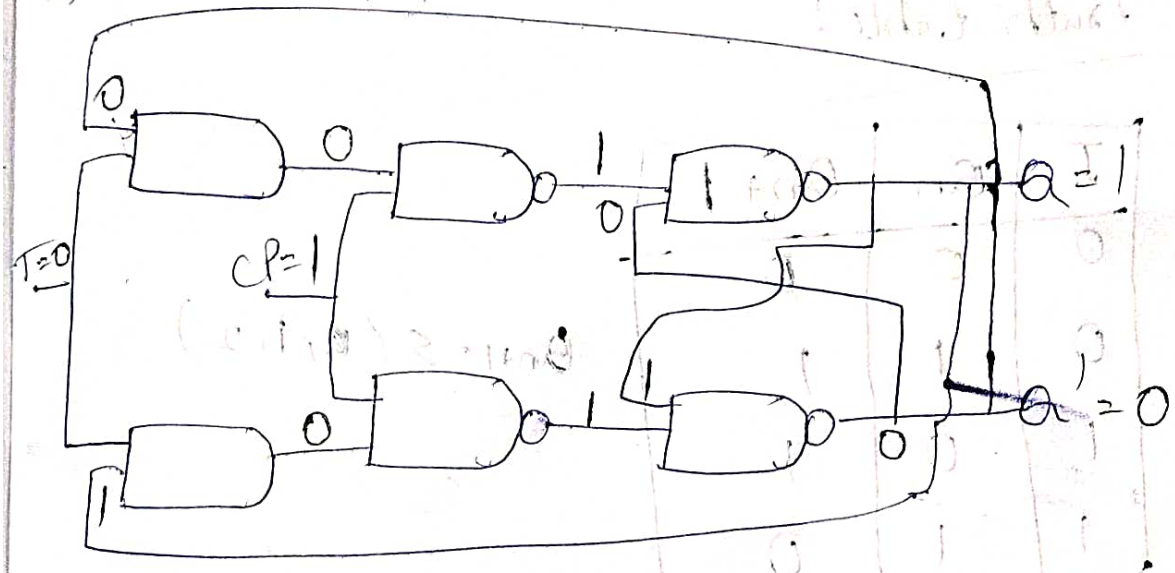


Case (i): 1) $T=0, Q_n=0, CP=1$ (Because it is positive edge triggering)



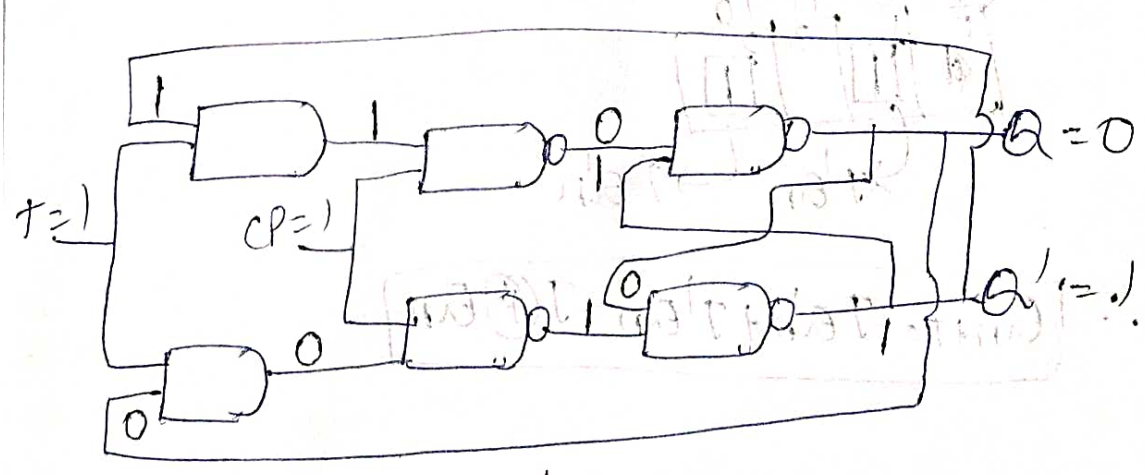
$Q_{n+1}=0$

2) $T=0, Q_n=1, CP=1$



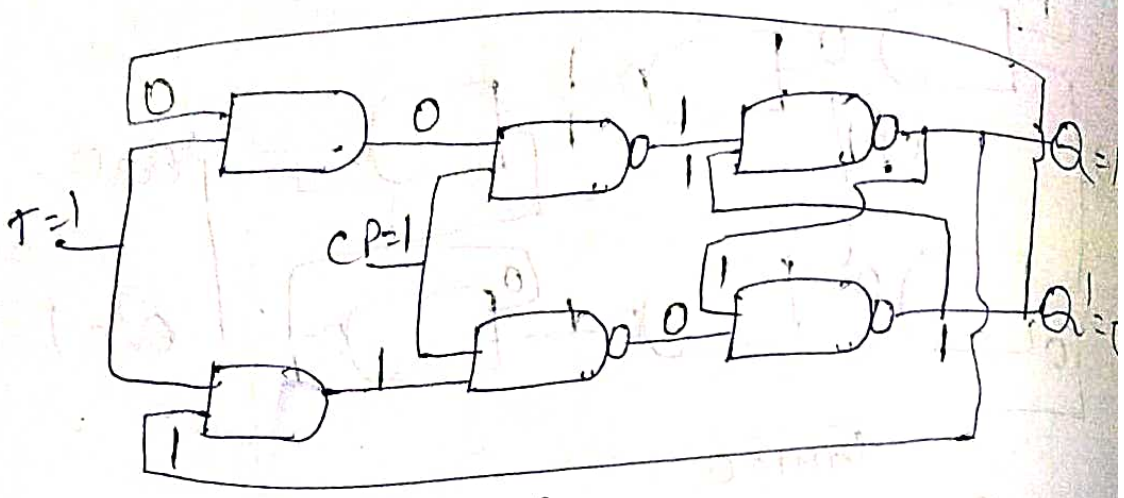
$Q_{n+1}=1$

Case (ii): 1) $T=1, Q_n=0, CP=1$



$Q_{n+1}=1$

(2) $T=1, Q_n=1, CP=1$

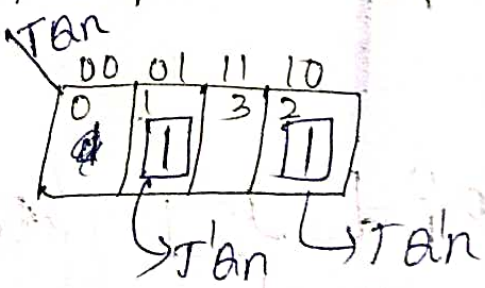


Truth table:

T	Qn	Qn+1
0	0	0
0	1	1
1	0	1
1	1	0

$Q_{n+1} = \Sigma(0, 1, 2)$

K-map simplification:



$\therefore \boxed{Q_{n+1} = TQ'_n + T'Q_n = T \oplus Q_n}$

* Asynchronous inputs:-

⇒ These are direct inputs/overiding inputs

⇒ They won't use any clock.

⇒ Here, storage elements responds without clock pulse.

⇒ In SR flipflop, the synchronous inputs are SR. In Jk flipflop, synchronous inputs are Jk.

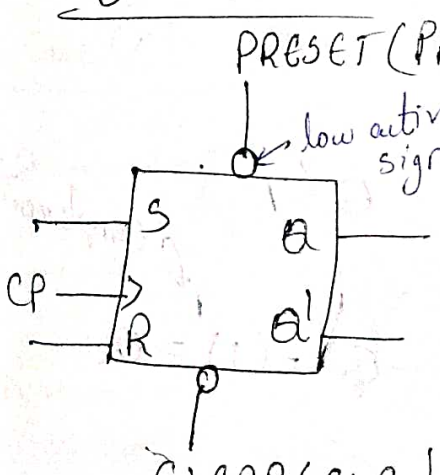
⇒ We can provide Asynchronous inputs to flip flops.

Ex:- SR flip flop:-

⇒ Now we will provid asynchronous inputs to the SR flipflop. The asynchronous inputs here are "PRESET & CLEAR".

⇒ Using this PRESET & CLEAR, the flipflop can respond without having impact of synchronous inputs.

Logic symbol:

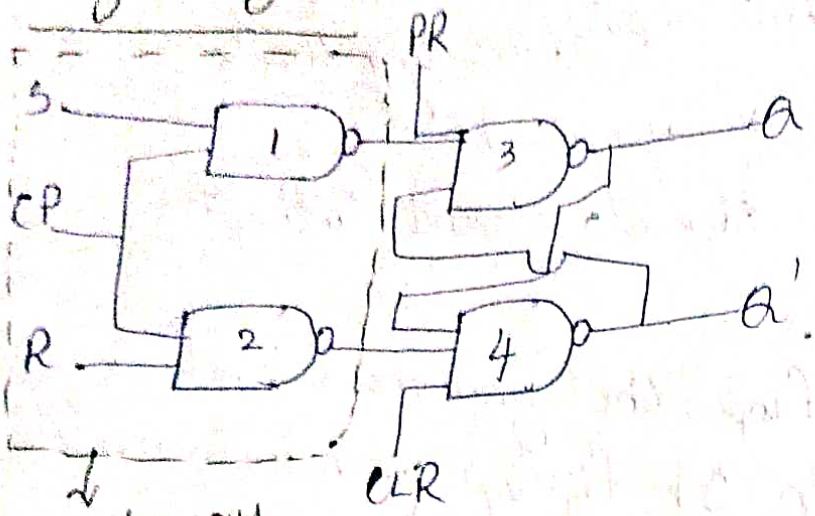


⇒ PRESET & CLEAR are Active low signals.

⇒ If we provide their input as '0' then only the flipflop will active and provide output

⇒ If we provide PRESET & CLEAR inputs as '1' then flipflop will be inactive and don't give any Output

Logic diagram:



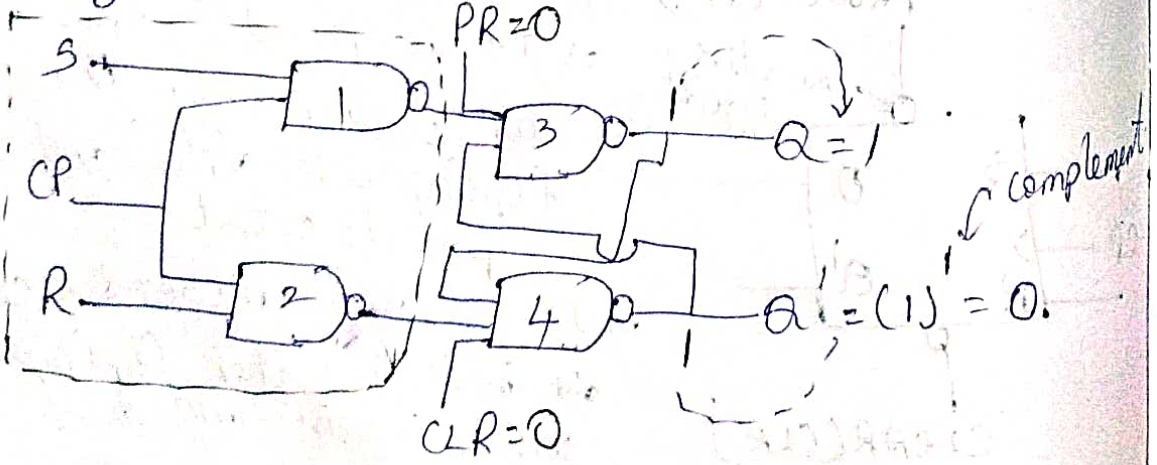
Truth table:

PRESET	CLEAR	Qn+1
0	0	Not used
0	1	1
1	0	0
1	1	Flipflop operates Normally

Check:

Case (i):

PRESET = 0 and CLEAR = 0. Both are '0's. SO they are activate.



⇒ Here both PRESET and CLEAR are active. So they don't consider synchronous block.

⇒ And at third nand gates one of the input, PR=0. We know that if one input is zero then the only will be one for NAND gates. So third NAND gate output is 1.

Q = 1.

⇒ And at third gate also, one input, CLR=0. So its Output is 1. But it is not 'Q', it is Q's complement means output complement will be its final output

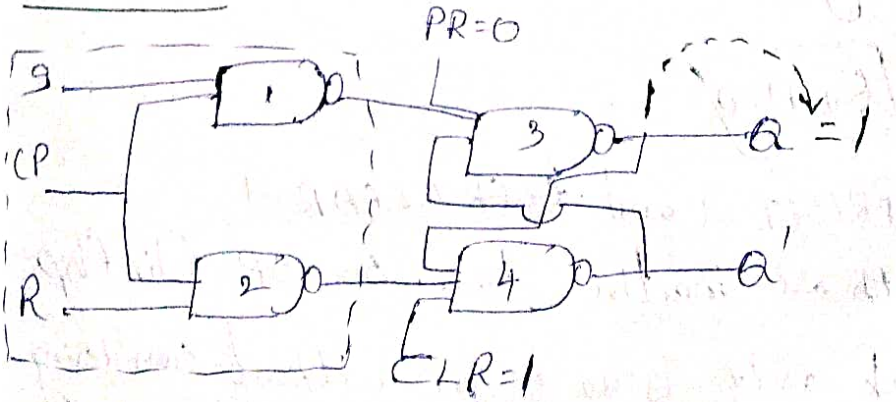
so Q' = complement of 4th gate output

⇒ Q' = complement of 1 = 0.

∴ Q=1, Q'=0. Here we have 2 final states. So we are not using them.

(Note: Here at 4th NAND gate, it is connected to Q'. So we have to complement the output that we get for 4th NAND gate and consider that value as Q' value).

Case (ii): PRESET=0 and CLEAR=1

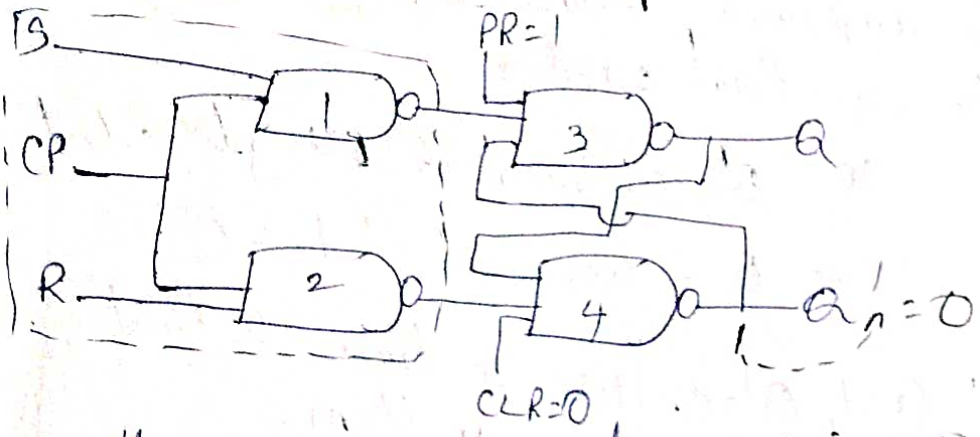


⇒ Here CLR=1. So it is inactive. And PR=0. So it is active. And at 3rd NAND gate one of the input is '0'. So its output is automatically '1'.

∴ Q=1. Here we are getting only one final next state. So it will be our next state.

$Q_{n+1}=1$

Case (iii): PRESET=1 and CLEAR=0



Here PR=1. So inactive, CLR is 0. So that 4th NAND gate is active. One of its complement input is '0'. So its output is '1' (complement of 1 is 0.

∴ Q'=0. Here only only next state we are getting. So it will be our final next state.

∴ $Q_{n+1}=0$

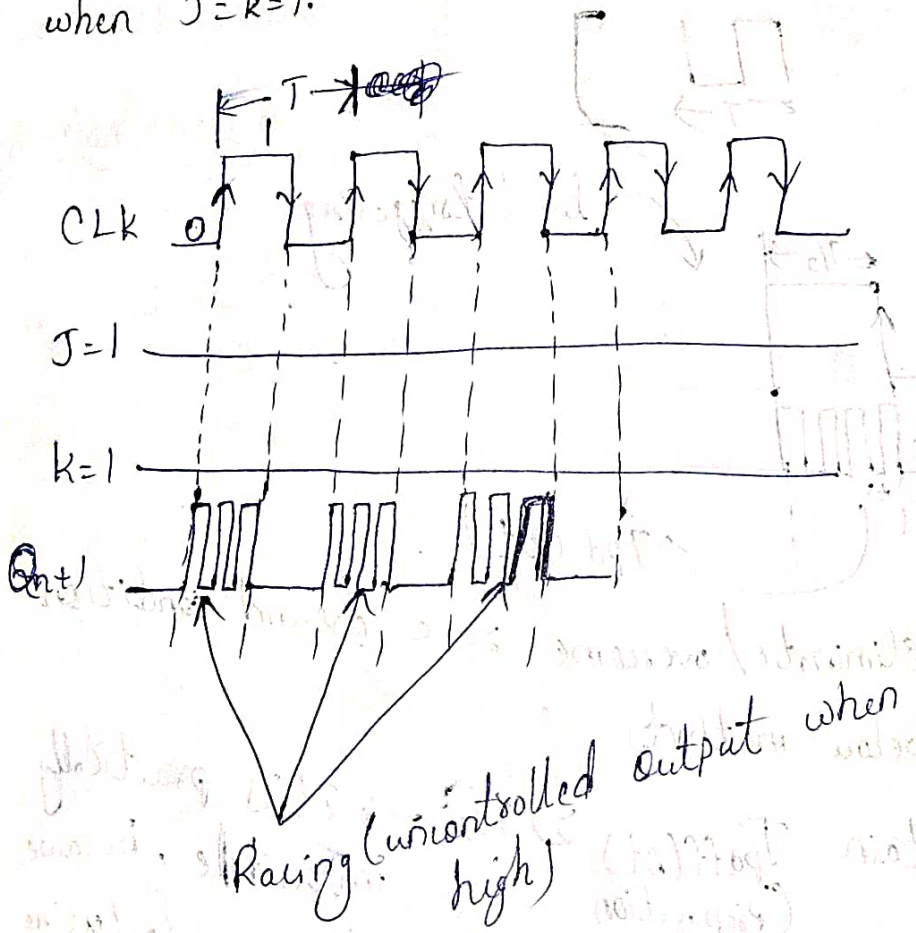
Case (iv): PRESET=1 and ~~CLR~~ CLEAR=1.

⇒ Both are inactive here. So SR flipflop will work as it is as before without considering PR and CLR(CLEAR). ∴

* Race Around Condition:-

In JK Flipflop, if J=k=1 & if clock is too long then state of the Flipflop is keep toggles which leads to uncertainity in determining the output state of flip flop. This problem is called "Race-Around condition".

Timing diagram of JK flip flop owing racing when J=k=1:-



* Race around condition occurs when:-

① Half of the time period of the clock pulse (CLK) is much more greater than propagation delay flip flop.

$T/2$ of CLK >> Tpdff

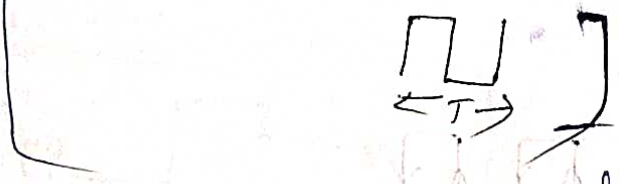
(T_{pdff} – Propagation delay of flipflop)

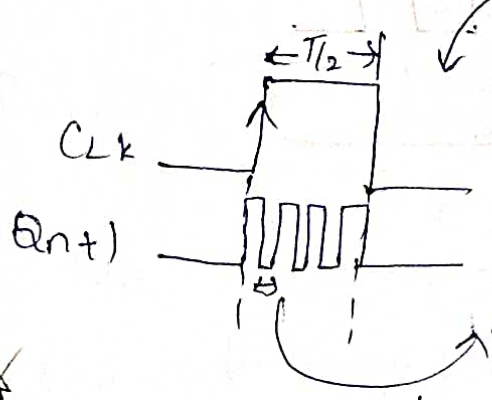
T_{pdff} :- Time taken by the inputs to be propagated to the output

(2) Use of level triggering.

→ When we use level triggering (positive & Negative), then race around condition occurs.

Time period :- One ← Time taken for complete cycle.





* How to Eliminate/overcome race around condition

→ Using below methods

(i) Maintain $T_{pdff}(\Delta t) >> T/2$ (Propagation delay) → It is practically impossible, because while manufacturing the chips they will set some propagation delay. We can't able to change it.

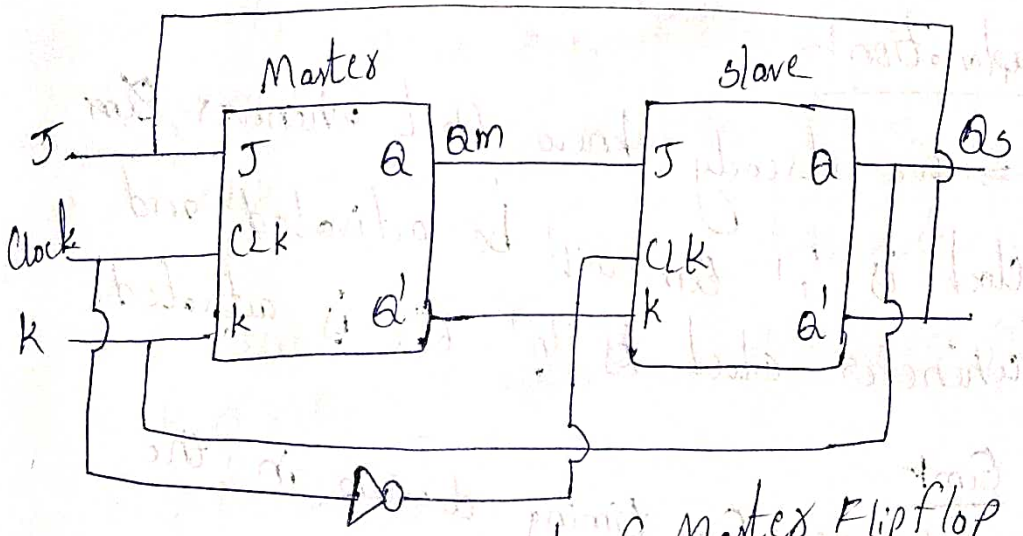
But critically we can eliminate race around condition.

(ii) Use of Edge Triggering :- Here the half of the time period is very less C in both positive and negative Triggering.

(iii) Use of Master-slave configuration (-ve edge triggering (it may be in SR FF, D FF & T FF also)).

Now, by using Master-slave ~~method~~ JK flipflop we will eliminate race around condition.

* Master-slave JK flip flop



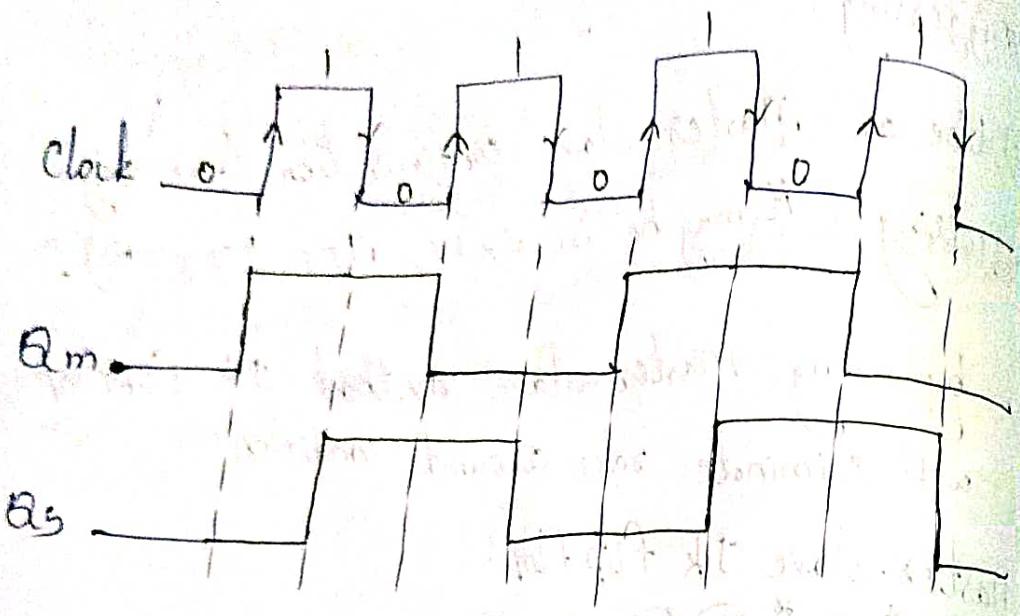
Here, Qm is the output of Master Flipflop

Qs is the output of slave flip flop

→ When clock = 1, Master FF will be activated & slave FF is disabled because clock is directly connected to Master FF and it's inverter is connected to slave FF.

→ When clock = 0, Master FF will be disabled & slave FF is activated.

* Timing diagrams of master-slave JK flip flop (when J=K=1):



{ Explanation:-

⇒ We already knew that whenever ~~Qm~~ clock is '1' Qm will be activated and whenever clock is '0' Qs is activated.

⇒ Qm:-

⇒ Consider Qm timing diagram in the above figure.

⇒ When CLK = 1, it activates and get output 1.

⇒ ~~When~~ ~~CLK~~ In the next, CLK = 0 then Qm is disabled. That's why it doesn't change it output and carries the previous memory (previous output 1).

⇒ Again value of CLK changes further and

CLK=1. Now, Qm will activated and changes its output. That means the previous memory '1' changes to output '0'.

⇒ Next CLK changes to '0' again and Qm is disabled and carries its previous memory (previous output '0').

⇒ Like this process will takes place.

Qs:

⇒ Here Qs will activate when Clk = 0.

⇒ Initially at CLK=1, Qs doesn't change its output because it doesn't activate at '0' that's why it is zero.

Later,

⇒ At CLK=0, Qs will activate and gives output '1'. Later at CLK=1, it is disabled and carries previous memory '1'. Later at CLK=0, it will activate and changes it output to '0'.

⇒ Like this the proces will continues.

* Toggling will occurs when clock changes from 0 to 1 or from 1 to '0'.

⇒ In Master flip flop, toggling occurs when clock changes from 0 to 1.

⇒ In Slave Flip flop, toggling occurs when clock changes from 1 to 0

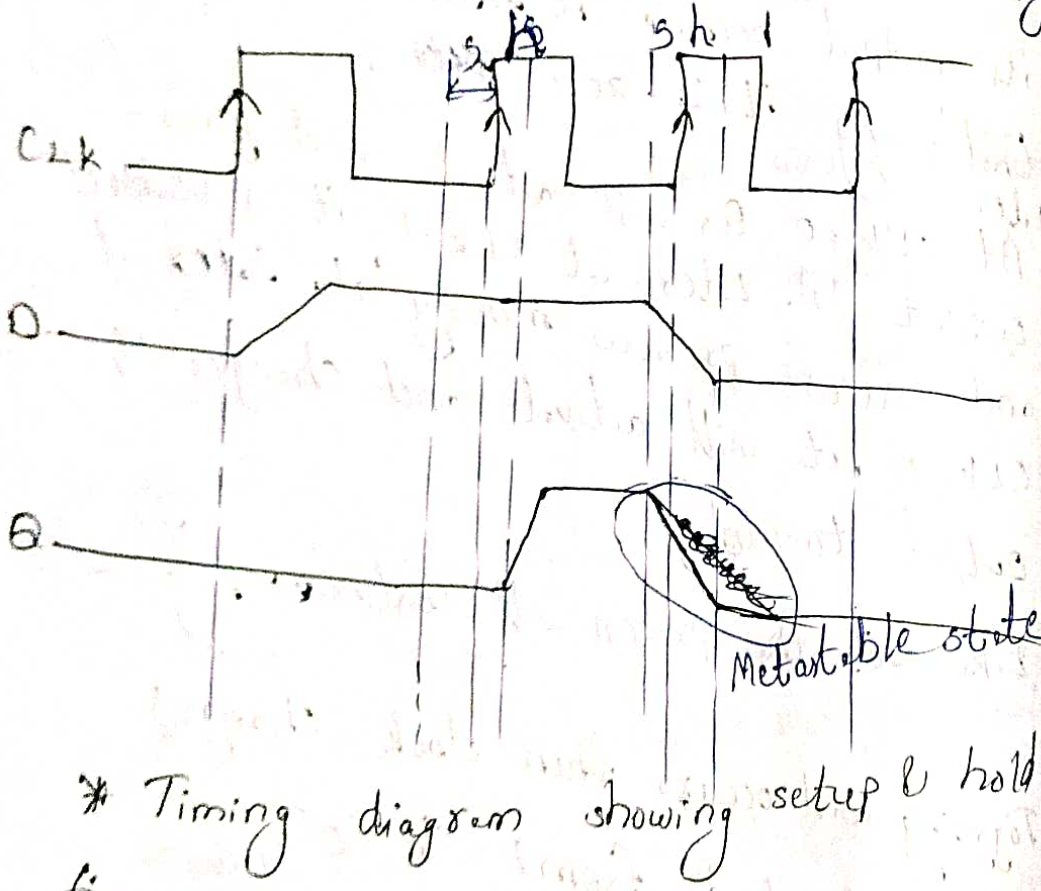
⇒ In Master Flip flop, the previous memory is carried when CLK=0

⇒ In Slave Flipflop, the output is previous memory when CLK=1

Toggling :- Controlled output when clock changes.

* Setup time (s) :- Minimum time required input to be stable prior to clock edge.

* Holde time (h) :- Minimum time required input to be stable after clock edge



* Timing diagram showing setup & hold time

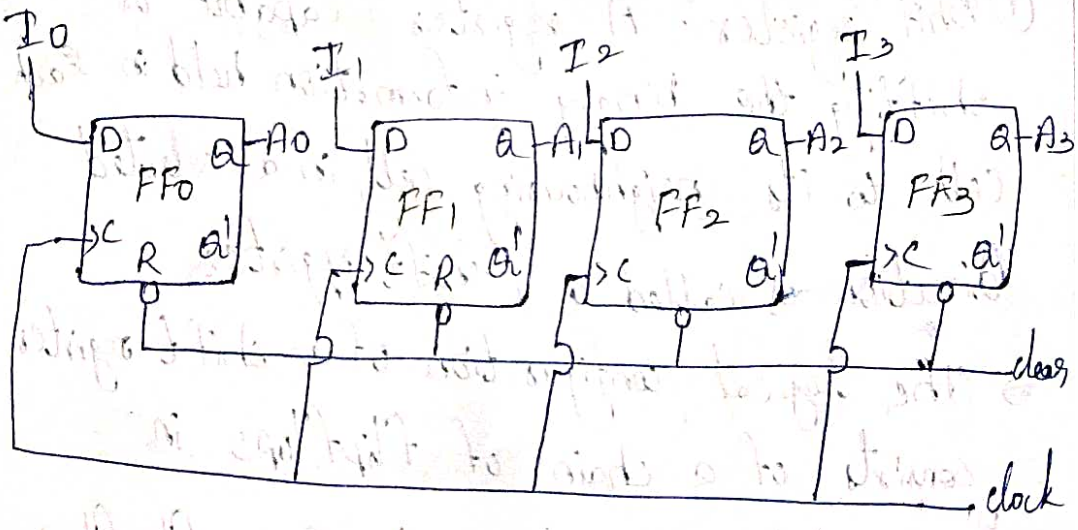
→ To calculate setup time and hold time for any flip flop, the input should be stable. But in the above diagram when input changes from 1 to 0 it is not stable. It is in metastable state. so, at that situation we can't able to find setup time and hold time, we can find setup time and hold time, we can define particular state output.

*** Registers & counters:-

→ Register:- A register is a group of flipflops. Each one of shares a common clock and capable of storing one-bit of information.

* An n-bit register consists of a group of 'n' flipflops capable of storing 'n' bits of binary information.

Ex: 4-bit register - it requires 4 flip flops.



⇒ Here "clear" is a input and it is active low signal.

⇒ By providing clear=0, we can RESET flip flops.

⇒ Based on triggering (positive & negative) values (input data) stores in respective flipflops.

For FF_0 → input is I_0, output A_0

For FF_1 → input = I_1, output = A_1

For FF_2 → input = I_2, output = A_2

For FF_3 → input = I_3, output = A_3

Applications:

We can use register in two categories:-

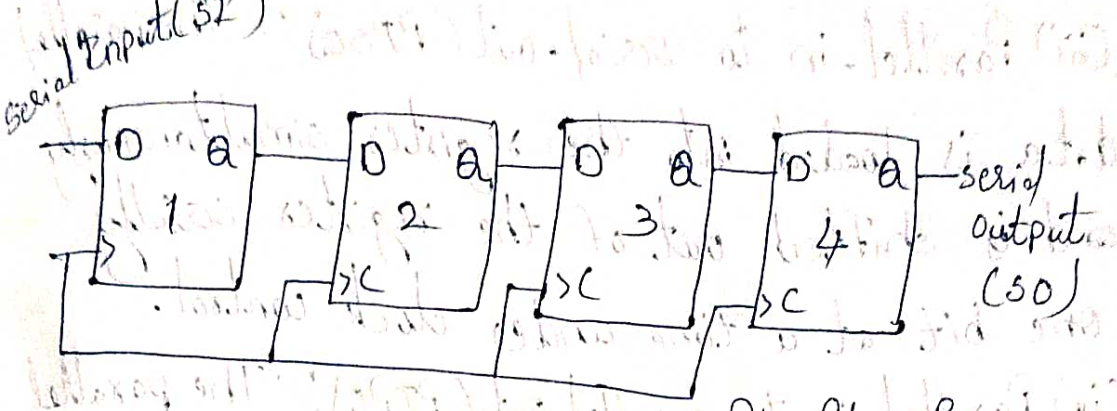
1. As a shift register
2. As a storage register.

① Shift register:- A register capable of shifting the binary information held in each cell to its neighbouring cell, in a selected direction is called a shift register.

⇒ The logical configuration of a shift register consists of a chain of flipflops in cascade, with the output of one flip flop connected to the input of next flipflop.

⇒ All flipflops receives a common clock pulses, which activate the shift of data from one state to the next.

Ex: 4-bit shift register



⇒ Here, we prefer only D flipflop. Because in a 'D' Flipflop, the output is same as input irrespective of present state.

* ~~Shift~~ register may operate in any one of following modes: (Based upon input and output configurations)

(i) Serial-in to parallel-out (SIPO):-

The register is loaded with serial data, one bit at a time, with stored data being available at the output in parallel form.

{ Ex:- Data:-

1 0 1 0 → we transfer this data bit by bit (serial), and it will be collected in ~~bit~~ by all at a time (parallel) & by using clock pulse)

(ii) Serial in to serial-out (SISO) :- The data is shifted serially "IN" & "OUT" of the register one bit at a time in either left (or) right direction under clock control.

(iii) Parallel-in to serial-out (PISO) :- The parallel data is loaded into the register simultaneously and is shifted out of the register serially one bit at a time under clock control.

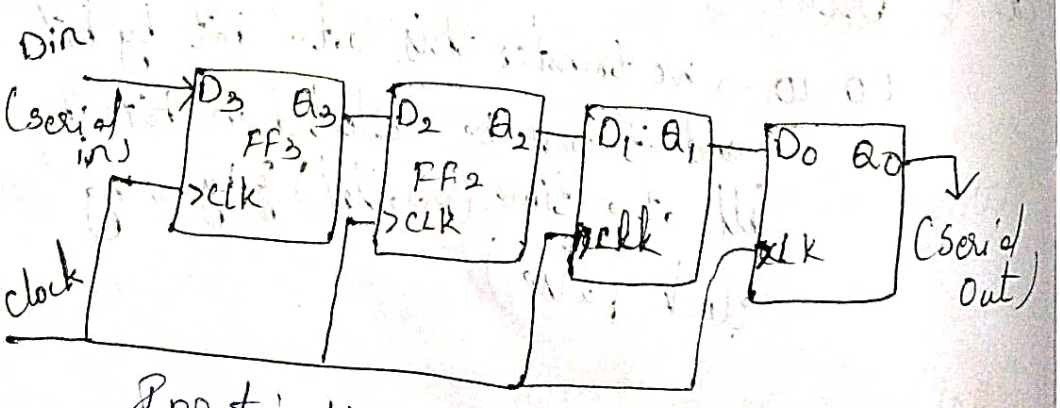
(iv) Parallel-in to parallel out (PIPO) :- The parallel data is loaded simultaneously into the register and transferred together to their respective outputs by the same clock pulse.

Note :- SISO can be used as a shift register.

* Serial In Serial Out (SISO) :

We can shift data from left to right (or) right to left.

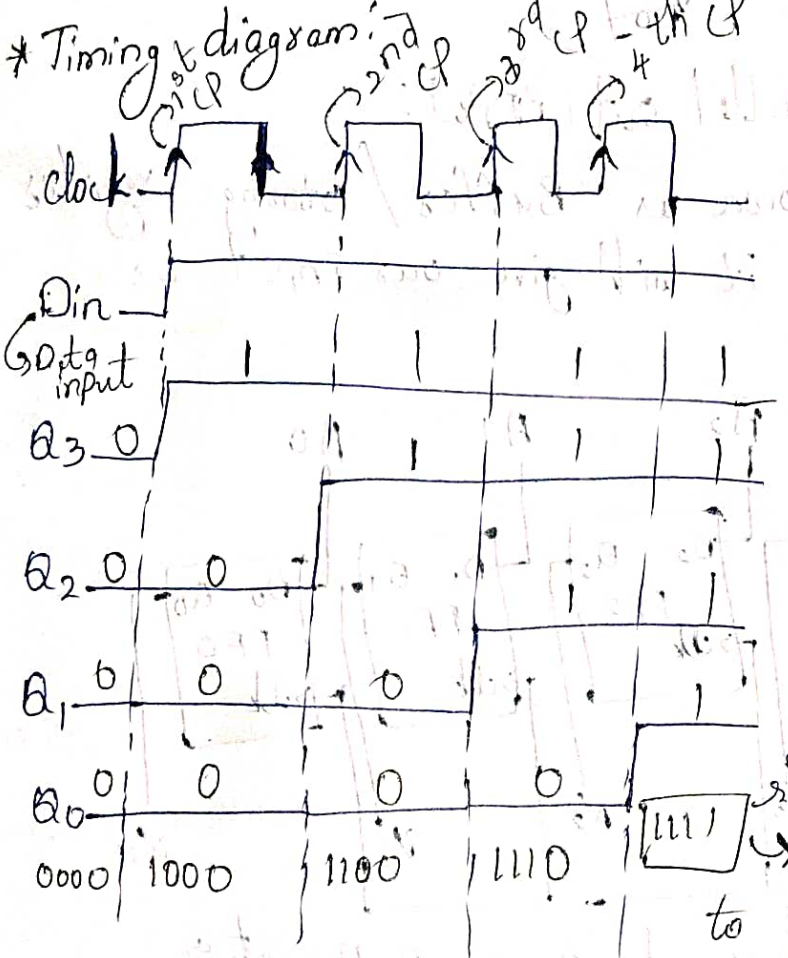
Here we are going to shift from left to right (right shift)



Input : 1111

clock	Q_3	Q_2	Q_1	Q_0
Initially:	0	0	0	0
1st CP ↑	1	0	0	0
2nd CP ↑	1	1	0	0
3rd CP ↑	1	1	1	0
4th clock pulse (CP) ↑	1	1	1	1

* Timing diagram:

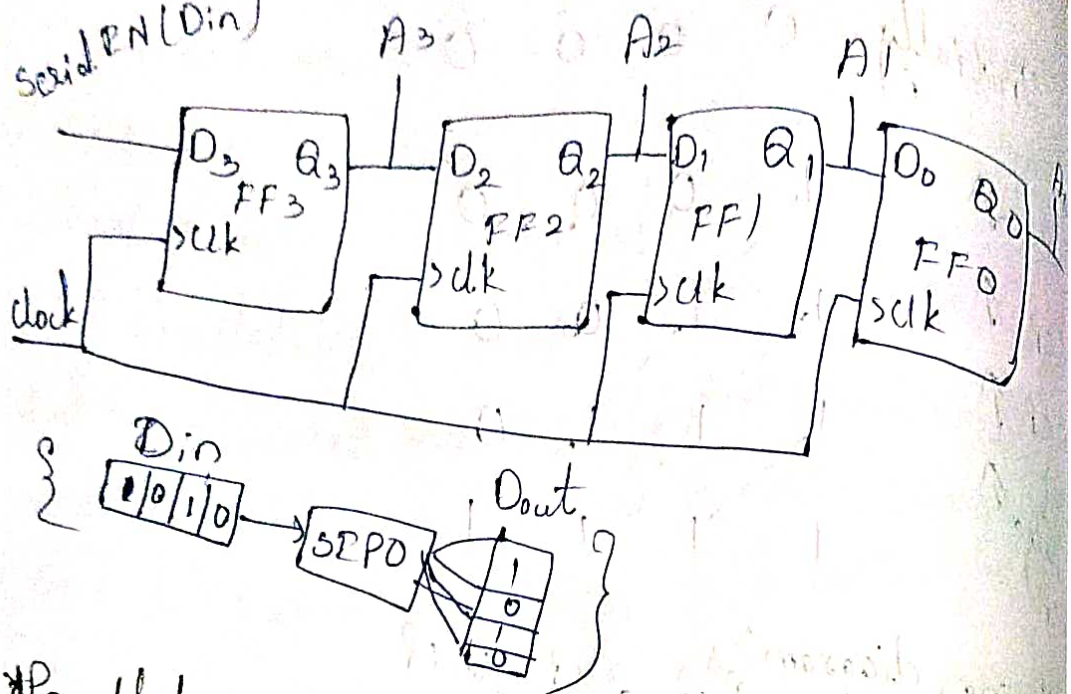


* Data is from shifted left to right and we get the final output data 1111.

* Here for 4-bit data we required 4 flipflops.
* Here the data stored at the End of the 4th clock pulse.

(HINT: Refer NESO ACADEMY youtube channel for more clarity)

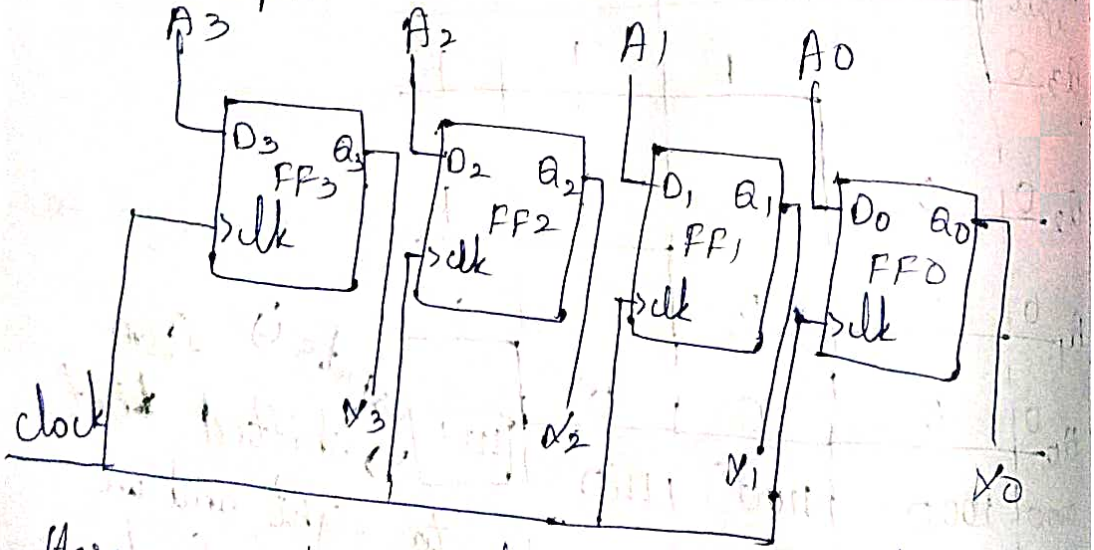
* Serial in parallel out (SIPO):



*Parallel-in-Parallel out (PIPO):

⇒ It is also known as Buffer / Storage register

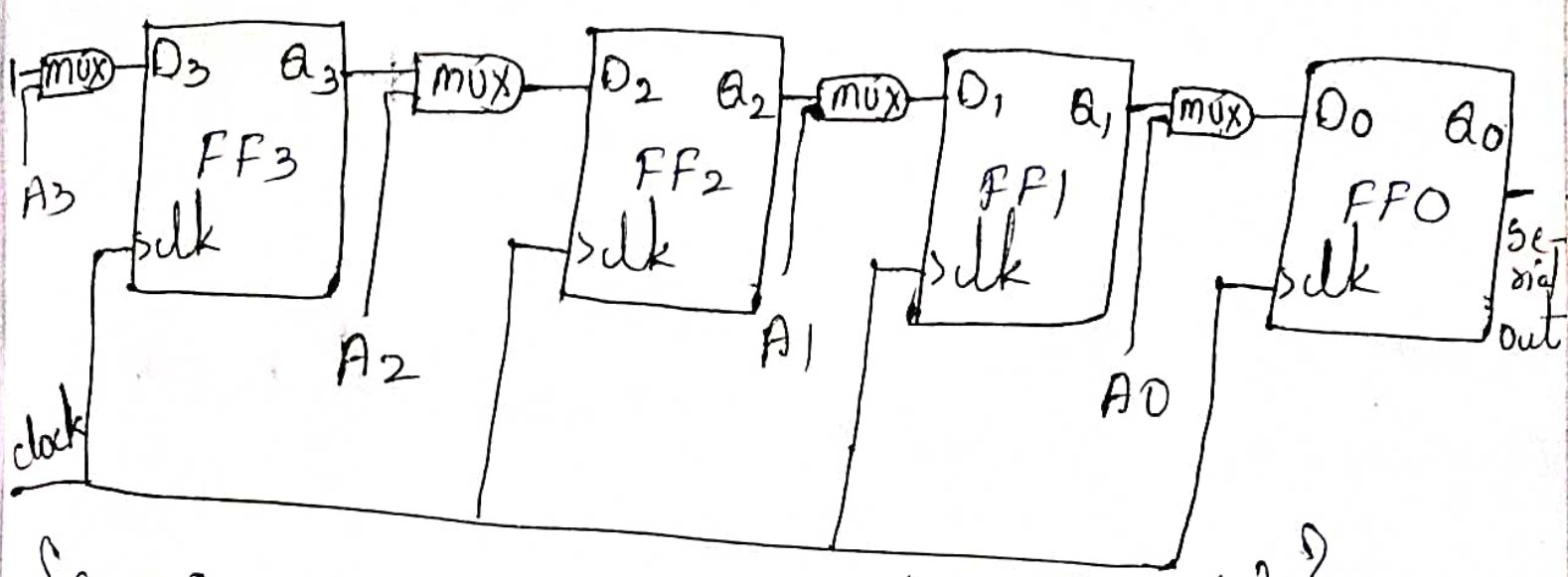
⇒ Buffer means it will give our Input as the output



Here we give Inputs A_3, A_2, A_1, A_0 at a time and we will get output y_3, y_2, y_1, y_0 at a time

Input ——▷ output

* Parallel-in Serial Out (PISO):-



{(Refer NESO ACADEMY youtube channel) }

⇒ For far distances better to prefer ~~dat~~ serial data transmissions.

Ex:- if the distance 10 cm. Then for ~~parallel~~ serial transmision it requires 1 conducting wires for 4 bits. But for parallel transmission it requires 4x10 = 40 conducting wires.

* State diagram:- It is a visual representation (Graphical representation) of how sequential circuits behave. It clearly says that the transition of states from one state to the next, as well as the output for a given input.

In this visual representation we need to remember 3 points:

(i) Each current (present) state is represented by circle.

(ii) The transition from present state to the next state is represented by a directed line connecting the circles.

(iii) If the directed line connects itself, which indicates there is no change in the state (the next state is same as the present state).

* State diagram is very helpful in analysing and designing the given sequential circuit.

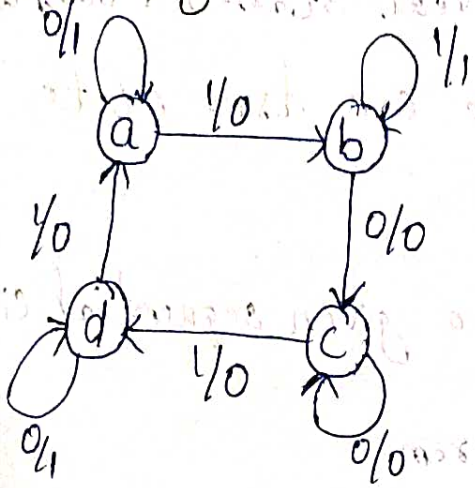
* State diagram can be draw in 2 models:-

① Mealy Model

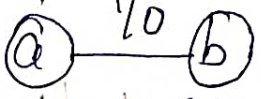
② Moore Model.

That two models supports Finite state Machines.

① State diagram for Mealy Model



Here, states are a, b, c, d.
Inputs are 1, 0
Outputs are 1, 0.

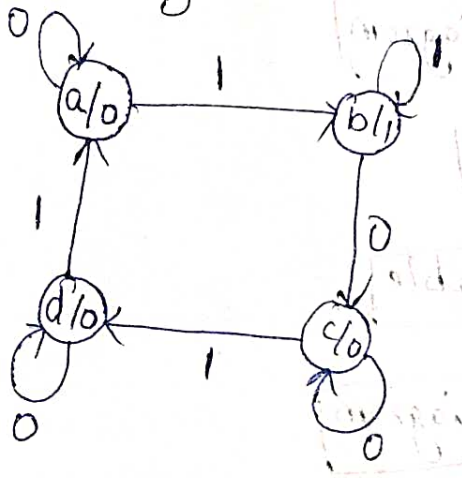


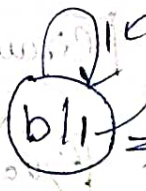
⇒ This means transition from a to b on input '1' and output is 0.

Here, $\frac{1}{0}$ → 1 → Numerator - Input
0 - denominator - output

⇒ * Here "output is a function of both present state & input".

② State diagram for Moore Model





← input
→ output
⇒ Transition from b on the input '1' gives b and with output is 1.

⇒ * Here "Output is a function of only ~~present~~ present states".

{ states can act as Present state or next state }

* State table: It consists of Present state, inputs, Next states & outputs.

* Using state table we can draw state diagram & vice versa.

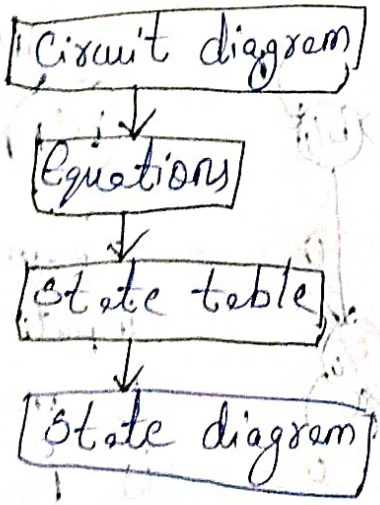
⇒ Steps for analysis of a given sequential circuit

Step-1: Draw circuit diagram.

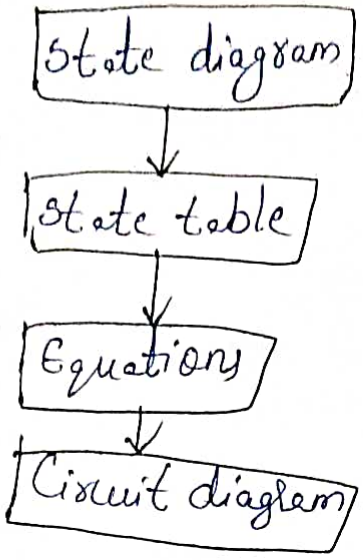
Step-2: Using circuit diagram of a given sequential circuit, draw next state equations.

Step-3: We can draw state table using equations.

Step-4: Using state table, draw state diagram.



* steps to design a sequential circuit.



*** Excitation Table:- (sequential circuits design for counters)

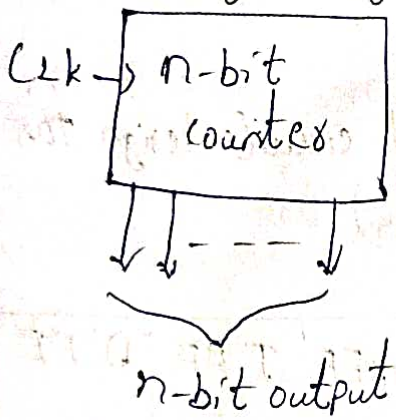
Present state (Qn)	Next state (Qn+1)	SRFF S R	JKFF J K	T-FF T	D-FF D
0	0	0 X	0 X	0	0
0	1	1 0	1 X	1	1
1	0	0 1	X 1	1	0
1	1	X 0	X 0	0	1

* Counters :- A counter is a register capable of counting the number of clock pulses arriving at its clock input.

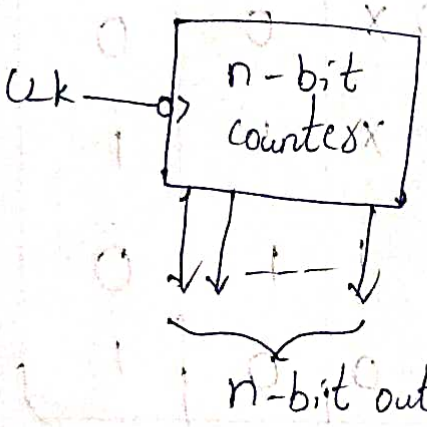
→ On arrival of each clock pulse, the counter is incremented by 1 (up counter).

→ In down counter, value is incremented by 1.

* Positive edge triggered n-bit counter



* Negative edge triggered n-bit counter



→ n-bit counter has 'n' flipflops & it has 2^n distinct states of outputs.

→ for Ex :-

2-bit binary counter has 2 Flip flops and distinct states of outputs $= 2^n = 2^2 = 4$ states.

They are:- 00, 01, 10, 11

→ The Maximum count that a binary counter counts = $2^n - 1$.

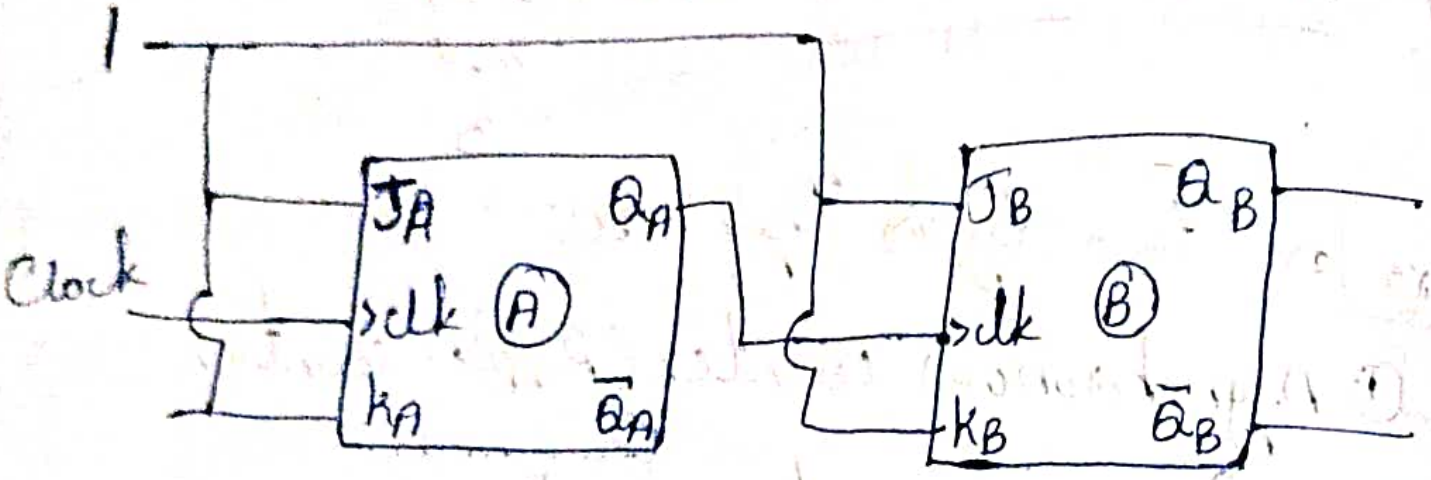
Ex: n = 2, then Max count = $2^2 - 1$
= 4 - 1
= 3

*Counters are two types They are:
① Asynchronous counter (Ripple counter)
② Synchronous counter

Asynchronous counter	synchronous counter
① External clock is applied to 1st FF & output of 1st FF acts as clock to Next FF	① External clock is applied to all FF's simultaneously.
② FF's are not clocked simultaneously	② FF's are clocked simultaneously
③ Circuit is simple for more number of states	③ Circuit is complex as number of states increases
④ Speed is slow (Operates slowly)	④ speed is high

* Asynchronous counter

⇒ We are taking JK FlipFlop here, because toggling is possible in JK FlipFlop.



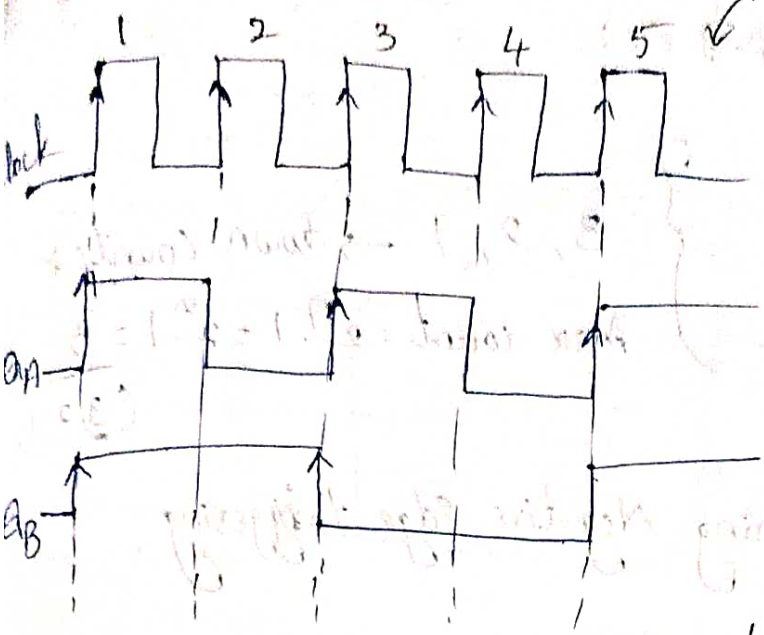
⇒ For FF 'B' clock is Q_A can be acts as clock. For FF 'A' clock we use clock to apply.

⇒ In counters there are 3 types of countings:-

① up counter :- 1, 2, 3, ——

② Down counter :- 3, 2, 1, ——

③ UP/Down counter.



↙ Timing diagram

⇒ Here we apply positive edge triggering for clock. so whenever clock = 1, then toggle will occurs for Q_A and Q_A will changes it value. i.e, if it is in 0(↓) then after toggle it value is 1 (↑) and wise versa:

If the clock value is 0 (↓) then Q_A will carries previous memory (previous output)

⇒ Q_B will follows Q_A. whenever Q_A is 1 (↑) then Q_B toggles and changes its states from 0 to 1 & 1 to 0.

⇒ And when Q_A is 0(↓), it carries previous memory (output).

↙ Truth table.

Clock pulse	Q_B	Q_A
0	0	0
1	1	1
2		
3		0
4	0	1
5	0	0
	1	1

} some outputs will come as above

Here the outputs are:

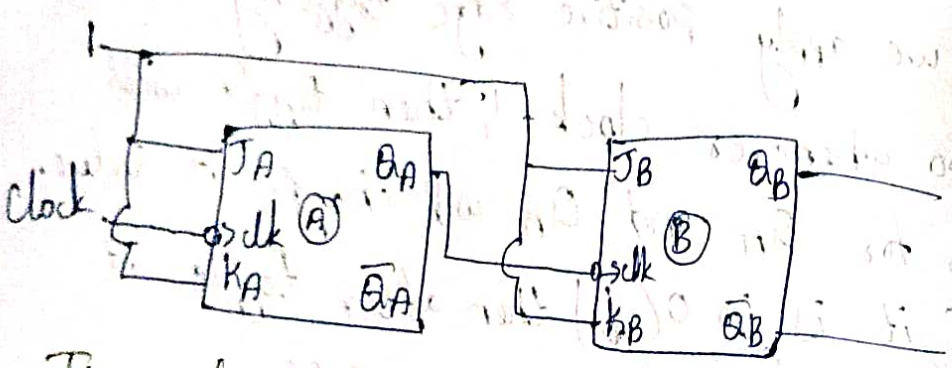
0 0 = 0
1 1 = 3
1 0 = 2
0 1 = 1
0 0 = 0

} 3, 2, 1 → down counter

Max count = $2^n - 1 = 2^2 - 1 = 3$

(3, 2, 1)

* 2 bit ripple using Negative Edge triggering.



Timing diagram

⇒ Here Q_A toggles when clock = 0 (↓) because we used Negative Edge triggering.

⇒ Q_B toggles and changes its value from 0 to 1 & 1 to 0 when $Q_A = 0$ (↓) because Q_A acts as clock for Q_B.

Clock pulse	Q_B Q_A	
Initially	0 0	= 0
1	0 1	= 1
2	1 0	= 2
3	1 1	= 3

← Truth table

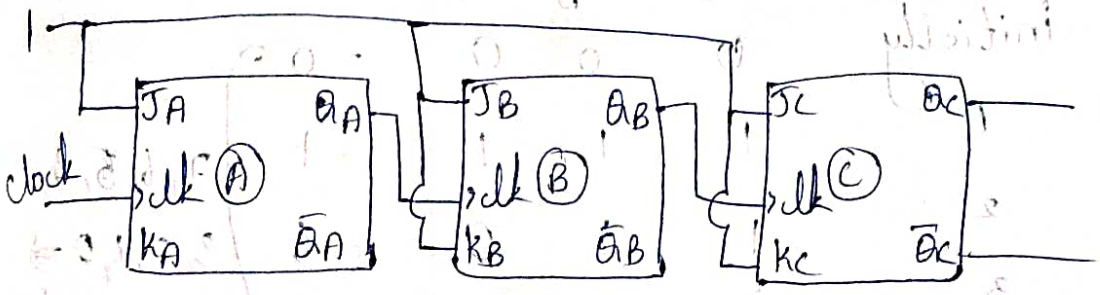
1, 2, 3 ⇒ up counter

Max count = 3 $(2^2 - 1 = 3)$.

* 3 bit ripple counter:

① Using positive Edge triggering.

⇒ Take 3 Flip flops.

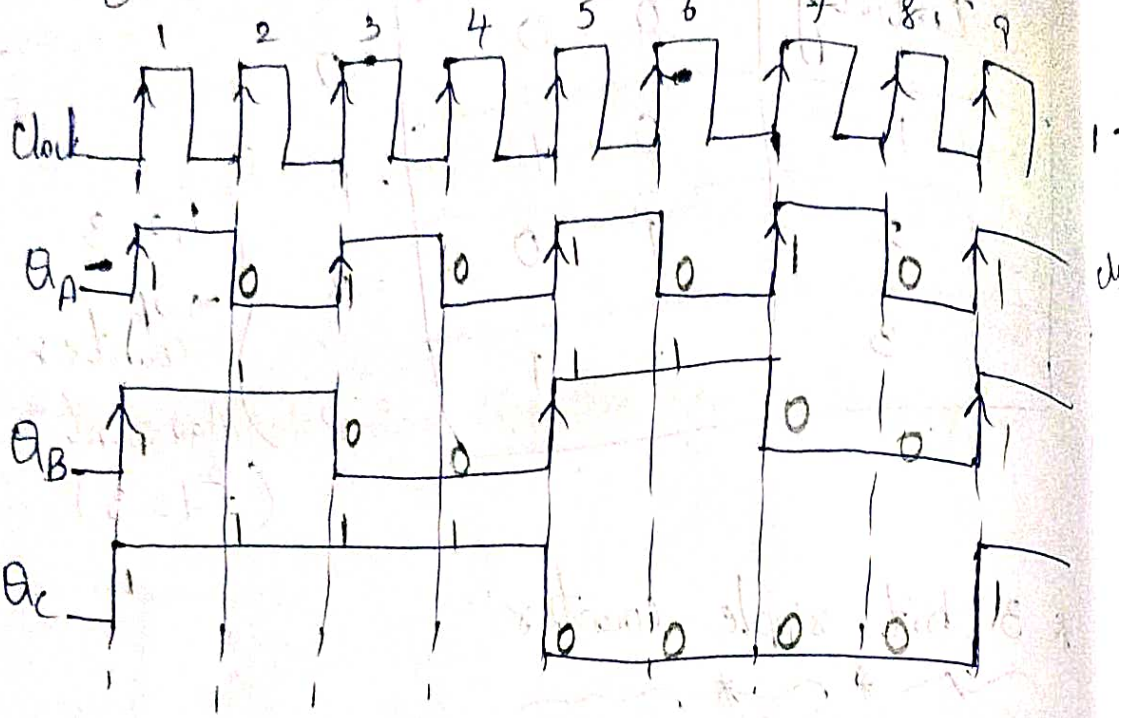


⇒ Here Q_A toggles when clock = 1 (↑)

⇒ Q_A acts as clock to Q_B. So Q_B toggles when $Q_A = 1$ (↑)

⇒ Q_B acts as clock to Q_C. So Q_C toggles and changes its value from 1 to 0 or 0 to 1 when $Q_B = 1$ (because we used positive edge triggering here).

Timing diagram:-



Truth Table:-

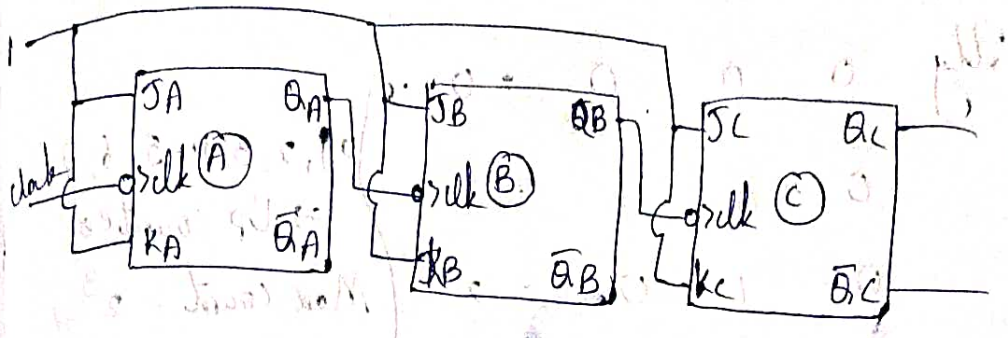
Clock pulse	Q_C	Q_B	Q_A	
initially	0	0	0	= 0
1	1	1	1	= 7
2	1	1	0	= 6
3	1	0	1	= 5
4	1	0	0	= 4
5	0	1	1	= 3
6	0	1	0	= 2
7	0	0	1	= 1
8	0	0	0	= 0
9	1	1	1	= 7

7, 6, 5, 4, 3, 2, 1, 0 ⇒ down counter.

$(2^3 - 1 = 7$ ↓ Max. Count)

Some will repeat like above format

② Using Negative Edge triggering :-

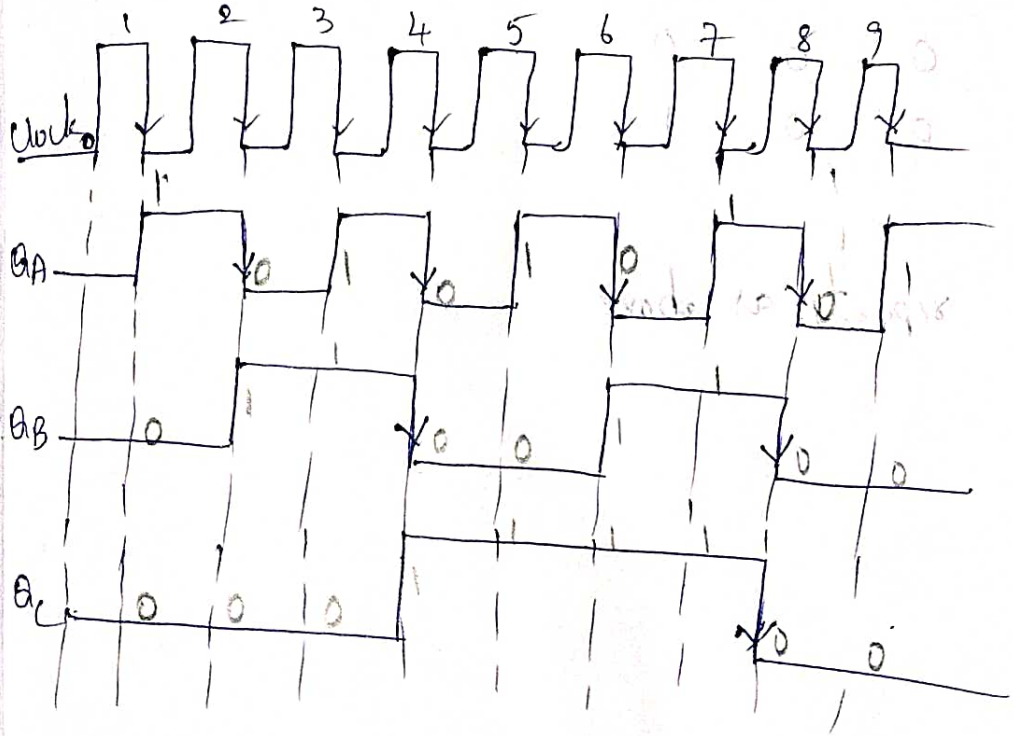


⇒ Here Q_A toggles when clock = 0 (↓)

⇒ Q_B toggles when $Q_A = 0$ (↓)

⇒ Q_C toggles when $Q_B = 0$ (↓)

Timing diagram :-



Truth Table :-

Clock pulse	Q_C	Q_B	Q_A	
initially	0	0	0	= 0
1	0	0	1	= 1
2	0	1	0	= 2
3	0	1	1	= 3
4	1	0	0	= 4
5	1	0	1	= 5
6	1	1	0	= 6
7	1	1	1	= 7
8	0	0	0	
9	0	0	1	

0, 1, 2, 3, 4, 5, 6
⇒ Up counter
Max count = 2

repeat as above

* Design a 3-bit ripple counter ~~up/down~~ up down counter:-

⇒ To design this we required a control input say 'M'

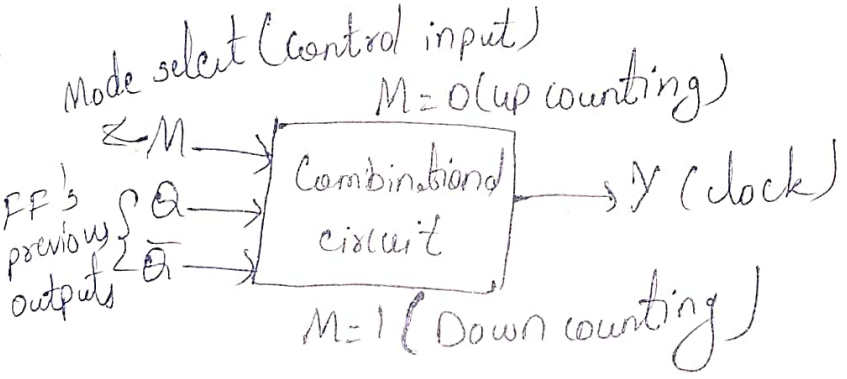
⇒ Let us assume a logic,

whenever $M=0$, then it is up counting and Q connected to the clock.

(i: when $M=0$, we consider Q values)

whenever $M=1$, then it is down counting and $\bar{Q}$ connected to the clock.

(i: when $M=1$, we consider $\bar{Q}$ values)

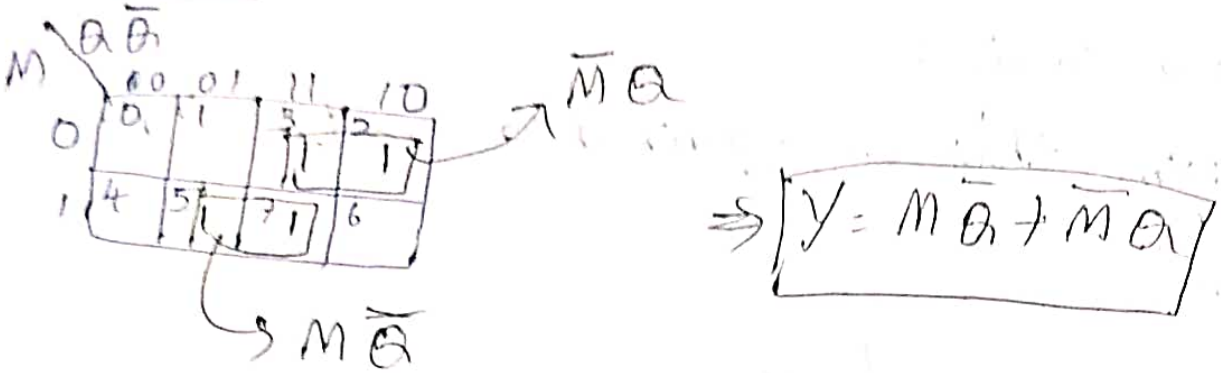


Truth Table:-

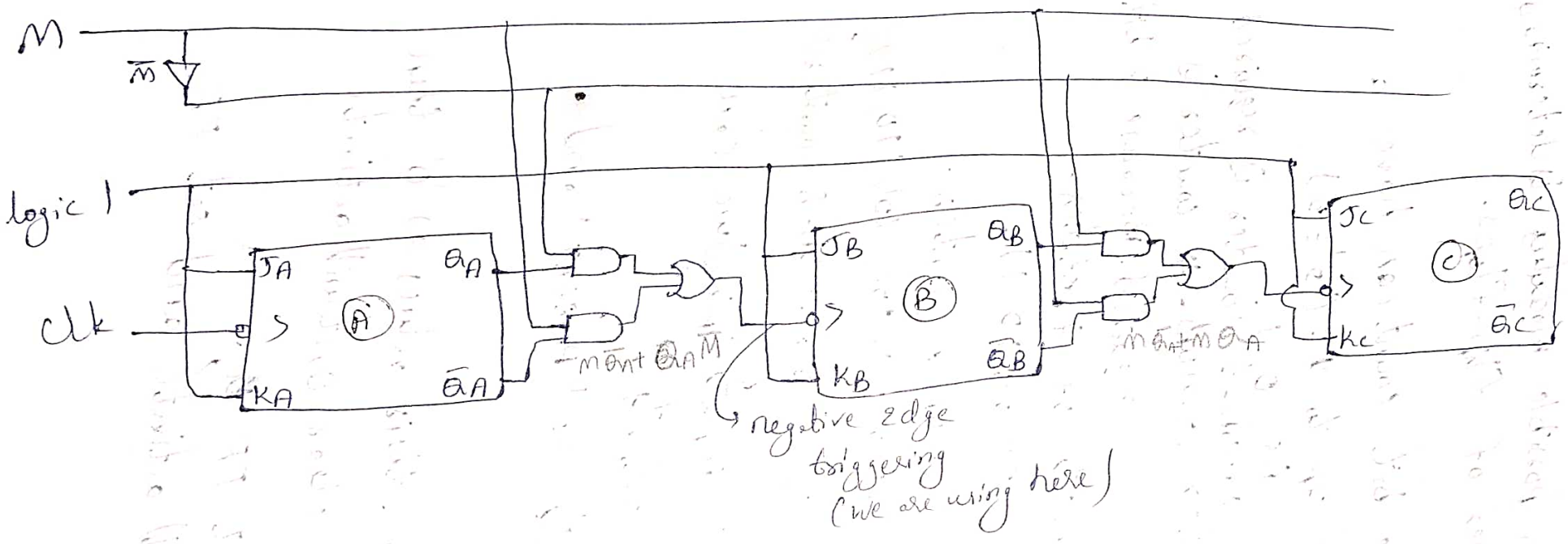
Input			Output	
M	Q	$\bar{Q}$	Y	
0	0	0	0	Q
0	0	1	0	
0	1	0	1	
0	1	1	1	
1	0	0	0	$\bar{Q}$
1	0	1	1	
1	1	0	0	
1	1	1	1	

$Y = \Sigma(2, 3, 5, 7)$

K-Map :-



We need to implement this Binomial Expression $Y = M\bar{Q} + \bar{M}Q$ to logical diagram (3 bit ripple counter).



* Circuit diagram

(Note :- For more clarity refer "Ekeeda" you tube channel - 3 bit Asynchronous Up down counter)

* Modulus of a Counter (MOD):

⇒ The modulus of a counter is the Number of states in its count sequence.

⇒ The maximum possible modulus is determined by the number of flip-flops.

Ex ⇒ A one bit count can have a modulus of up to 2^1. ($2^1 = 2$).

i.e, 0, 1.

So, 1 bit counter → MOD-2 counter is required.

⇒ 2 bit counter → MOD-4 counter is required. Because 2 bit consists of 4 states.

00, 01, 10, 11.

⇒ 3 bit counter → MOD-8 counter.

⇒ 8 states are:- 000, 001, 010, 011, 100, 101, 110, 111.

* Design MOD-6 counter using MOD-8 counter:-

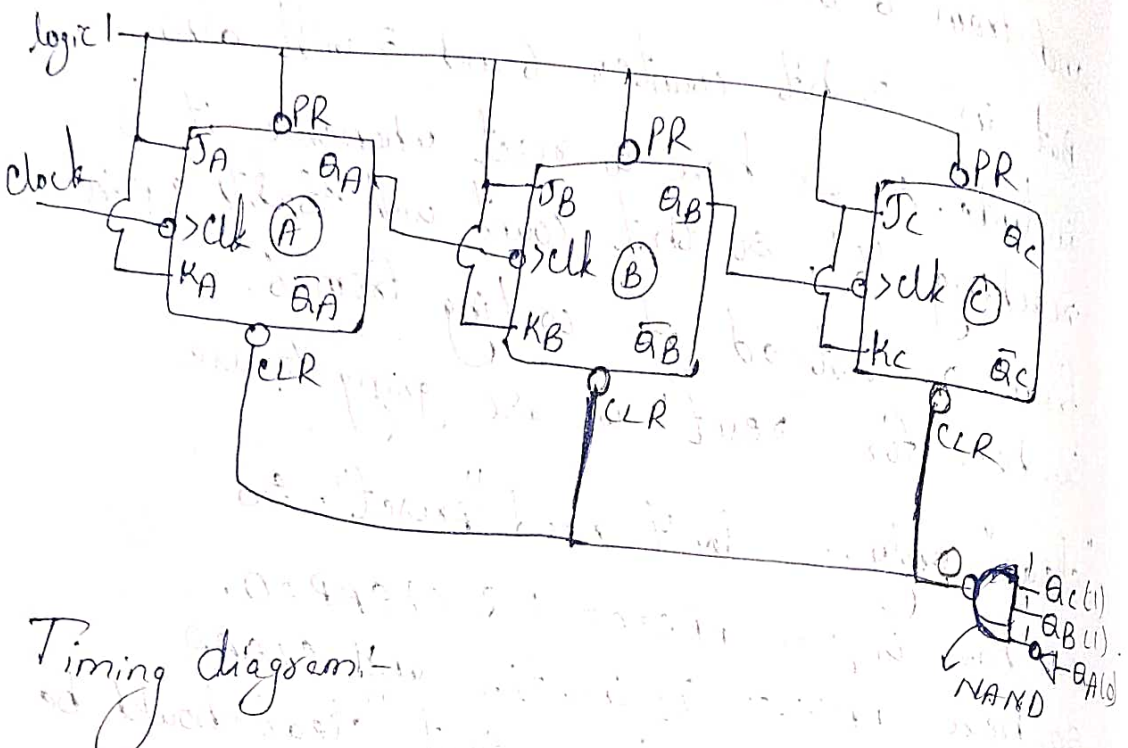
⇒ Using Asynchronous inputs (pins) we can "reset" the FF outputs.

⇒ Asynchronous inputs are "PRESET" and "CLEAR".

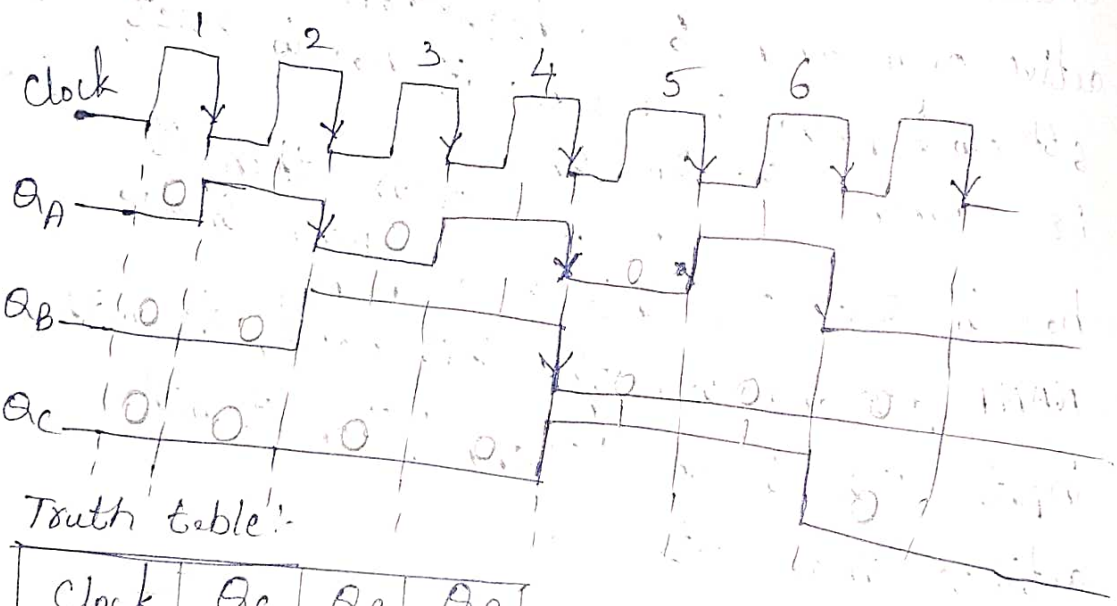
⇒ Whenever we assign their values as '0' then they will active. i.e clear=0, it will activate and we can reset FF & preset=0, it will activate and set FF.

Here we need count upto 6 counts, i.e 0, 1, 2, 3, 4, 5
and from 6 onwards we don't require them.
But in 3-bit counter 6 and 7 will also
include. So, we have to reset whenever it is
reaching '6'. so that count will start again
from 0 instead of conting from 6.
So, here for 'reset' we are going to use
"clear" only, we don't need "preset". So
we have to give PRESET=1 & CLEAR=0.
so, here PRESET is inactive and CLEAR
is active. And we know that "clear" should be
active only when 6 reaches. Means whenever
6th clock pulse is reaches it has to reset.
i.e when Q_C Q_B Q_A occurs then it
has to reset. For this we are going to use
1 1 0
NAND gate and connecting that nand gate with
input 110 to the "CLEAR". Then it will
activate and reset.

*Circuit diagram:-



Timing diagram:-



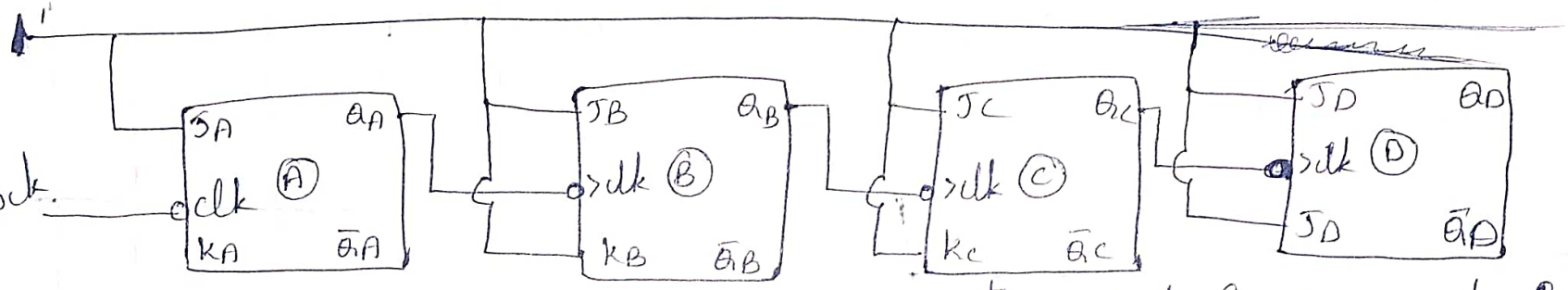
Truth table:-

Clock	Q_C	Q_B	Q_A
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1

RESET and
count from
0 again.

* 4 bit ripple counter :-

⇒ Using Negative edge triggering.



⇒ Here Q_A acts as clock for Q_B and Q_B acts as clock for Q_C and Q_C acts as clock for Q_D.

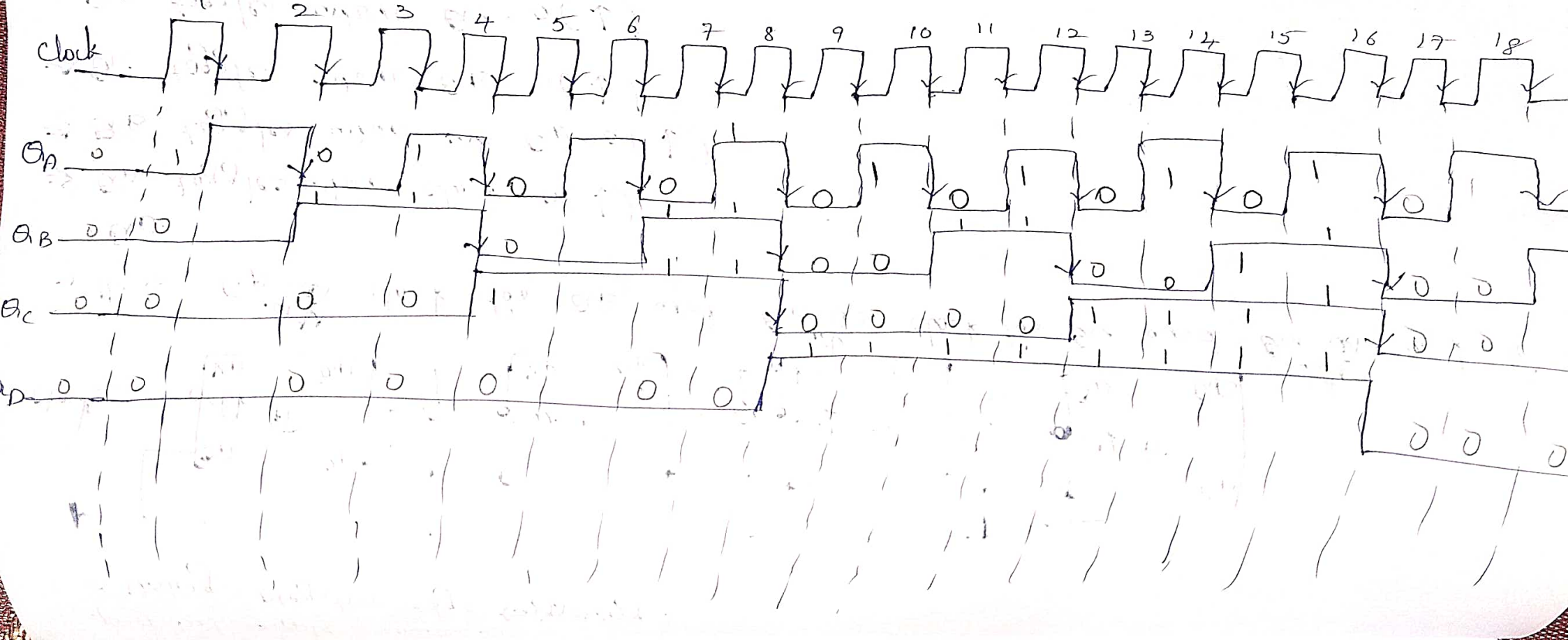
⇒ Q_A toggles when clock = 0 (↓)

⇒ Q_B toggles when Q_A = 0 (↓)

⇒ Q_C toggles when Q_B = 0 (↓)

⇒ Q_D toggles when Q_C = 0 (↓)

Timing Diagram:-



Truth Table:-

Clock pulse	Q_D	Q_C	Q_B	Q_A	
initially	0	0	0	0 = 0	
1	0	0	0	1 = 1	
2	0	0	1	0 = 2	
3	0	0	1	1 = 3	
4	0	1	0	0 = 4	
5	0	1	0	1 = 5	
6	0	1	1	0 = 6	
7	0	1	1	1 = 7	
8	1	0	0	0 = 8	
9	1	0	0	1 = 9	
10	1	0	1	0 = 10	
11	1	0	1	1 = 11	
12	1	1	0	0 = 12	
13	1	1	0	1 = 13	
14	1	1	1	0 = 14	
15	1	1	1	1 = 15	
16	0	0	0	0 = 0	
17	0	0	0	1 = 1	

UP counting
Max count
$= 2^4 - 1$
$= 15$
(0-15)

repeats as above

* Design of Synchronous Counters:

Steps:

1) Decide the number of FF's
2) Excitation table of FF
3) State diagram & circuit Excitation table
4) Obtain simplified Equations using k-map
5) Draw the logic diagram.

Ex:- 2-bit synchronous upcounter:

Step-1:- Let us consider 2 JK FF's

Step-2:- Excitation table of JK FF

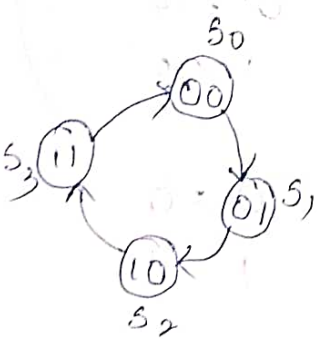
Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

(Taken from Excitation table we had studied already)

Step-3:- State diagram:-

For 2 bits, number of states $= 2^2 = 4$ states

i.e, 00, 01, 10, 11.



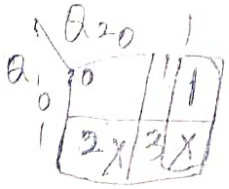
Circuit Excitation table:

Q_1	Q_2	Q_1^*	Q_2^*	J_1	K_1	J_2	K_2
0	0	0	1	0	X	1	X
0	1	1	0	1	X	X	1
1	0	1	1	X	0	1	X
1	1	0	0	X	1	X	1

(In the above table Q_1 Q_2 are present states & Q_1^* Q_2^* are the next state. For Ex, if present state is 01 (1), then next state will be 10 (2) like that. It counts upto 3 and starts counting again from zero. And J_1K_1 is the combination of Q_1 Q_1^* values. i.e., $Q_1=0$ & $Q_1^*=0$ then $J_1K_1=00$ and its value has to be taken from the Excitation table. In Excitation table for 00 the value is 0X like that we will perform. And for J_2K_2, the combination values are Q_2 Q_2^*)

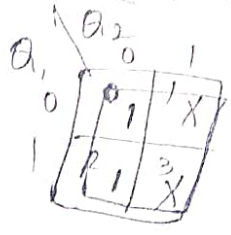
Step-4: Simplified Equations using k-map

For $J_1 = \Sigma(1) + d(2,3)$



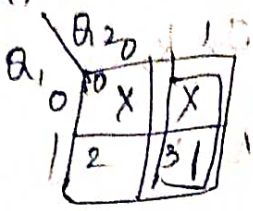
$J_1 = Q_2$

For $J_2 = \Sigma(0,2) + d(1,3)$



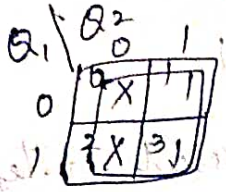
$J_2 = 1$

For k_1 : $\Sigma(3) + d(0,1)$



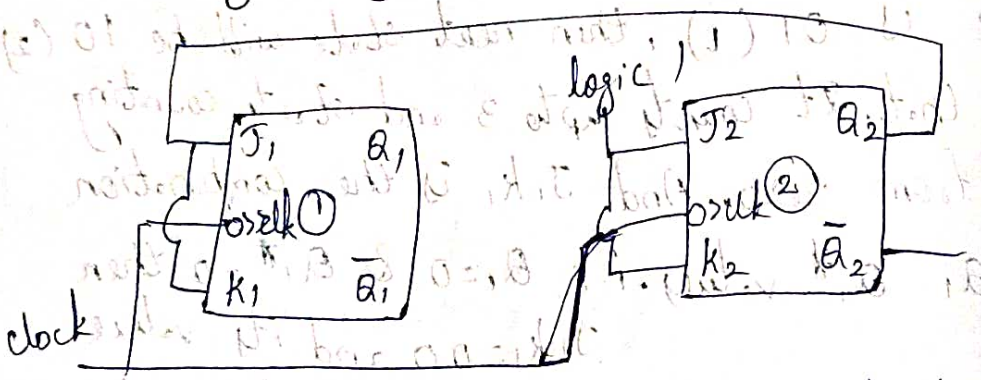
$\Rightarrow k_1 = Q_2$

For k_2 : $\Sigma(1,3) + d(0,2)$



$\Rightarrow k_2 = 1$

Step-5 : logic diagram for 2 bit synchronous up counter



* Synchronous BCD Counter (Decade counter)

⇒ This consists numbers from 0-15 in which "0 to 9" are minterms and 10-15 are don't care terms.

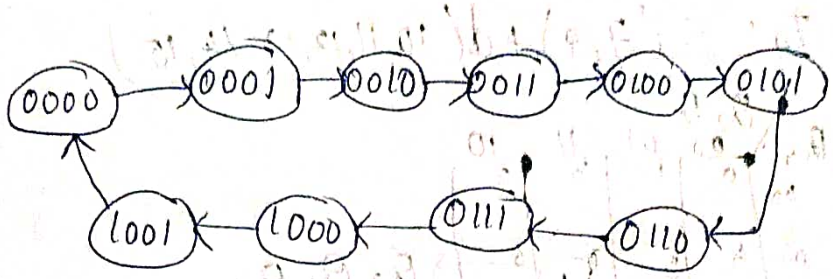
⇒ 4 inputs, $2^4 = 16$ combinations

Step-1 : Take 4 TFFs.

Step-2 : Excitation table for TFF

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Step-3: State diagram:-



Circuit Excitation table:-

S.No	Q_4	Q_3	Q_2	Q_1	Q_4^*	Q_3^*	Q_2^*	Q_1^*	T_4	T_3	T_2	T_1
0	0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	1	0	0	1	0	0	0	1	1
2	0	0	1	0	0	0	1	1	0	0	0	1
3	0	0	1	1	0	1	0	0	0	1	1	1
4	0	1	0	0	0	1	0	1	0	0	0	1
5	0	1	0	1	0	1	1	0	0	0	1	1
6	0	1	1	0	0	1	1	1	0	0	0	1
7	0	1	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	1	0	0	1	0	0	0	1
9	1	0	0	1	0	0	0	0	1	0	0	1

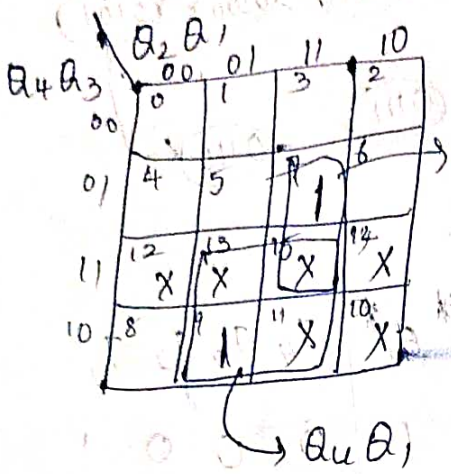
(In the above, T_4 is the combination of $Q_4 Q_4^*$, T_3 is the combination of $Q_3 Q_3^*$, For T_2 the combination is $Q_2 Q_2^*$, For T_1 the combination is $Q_1 Q_1^*$. If the combination values are same then resultant is '0', if different then '1'.

Ex:- $T_4 \Rightarrow$

Q_4	Q_4^*	T_4
0	0	0
0	1	1

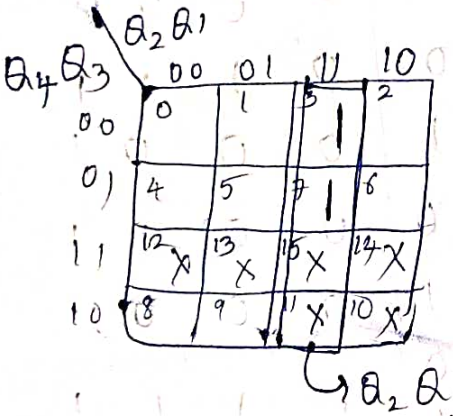
Step-4 :- Simplified Equations using k-map :-

For $T_4 :- \Sigma(7,9) + d(10,11,12,13,14,15)$



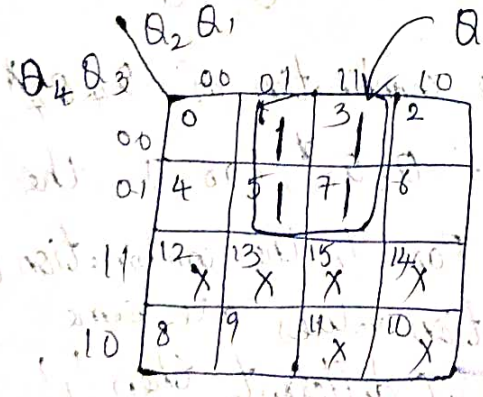
$\Rightarrow T_4 = Q_4 Q_1 + Q_3 Q_2 Q_1$

For $T_3 :- \Sigma(3,7) + d(10,11,12,13,14,15)$



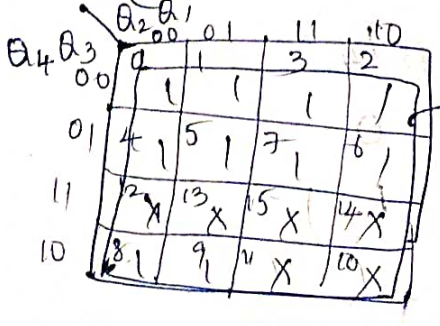
$\Rightarrow T_3 = Q_2 Q_1$

For $T_2 :- \Sigma(1,3,5,7) + d(10,11,12,13,14,15)$

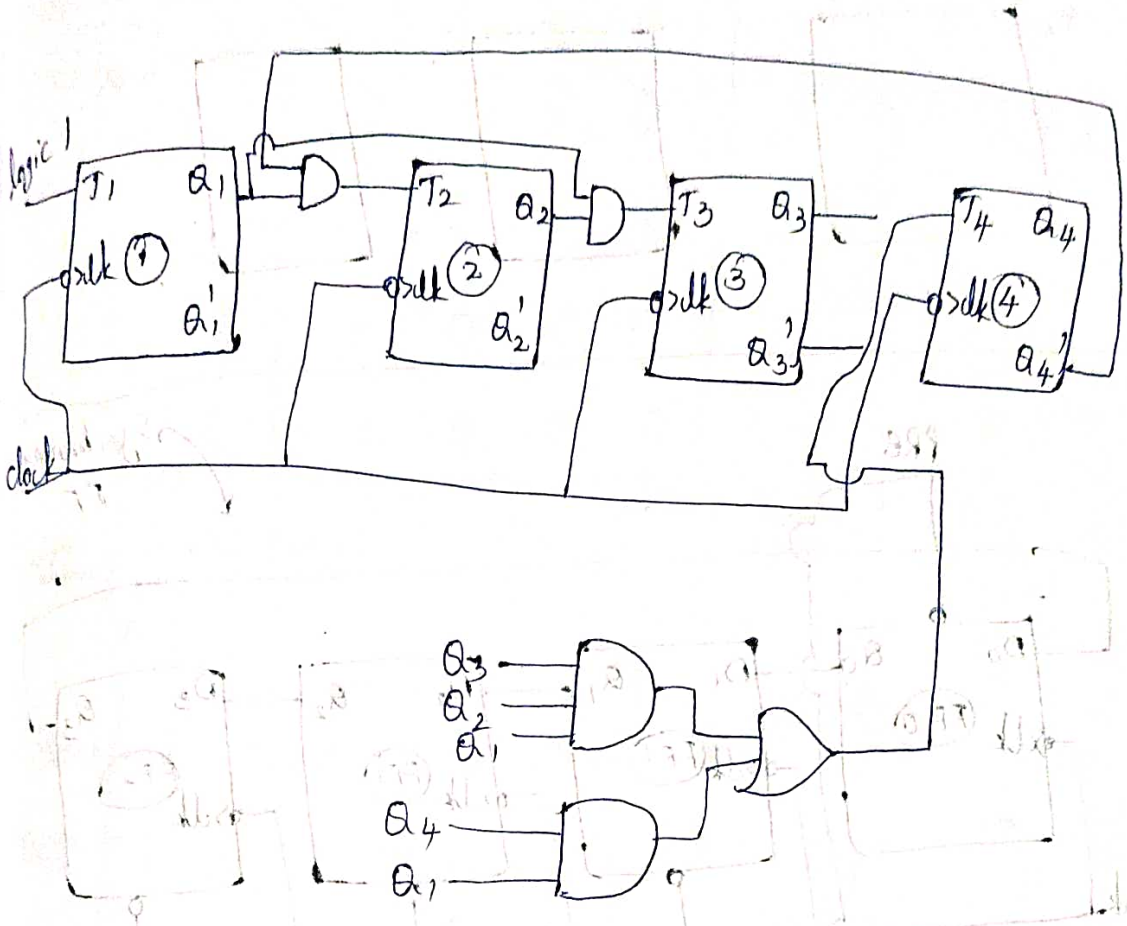


$\Rightarrow T_2 = Q_4' Q_1$

For $T_1 :- \Sigma(0,1,2,3,4,5,6,7,8,9) + d(10,11,12,13,14,15)$



step-5:- logic diagram based on required Expressions.



* Other Counter (special counters):-

There are 2 special counters:-

① Ring counter

② Johnson Counter (Twisted ring / switch tail counter)

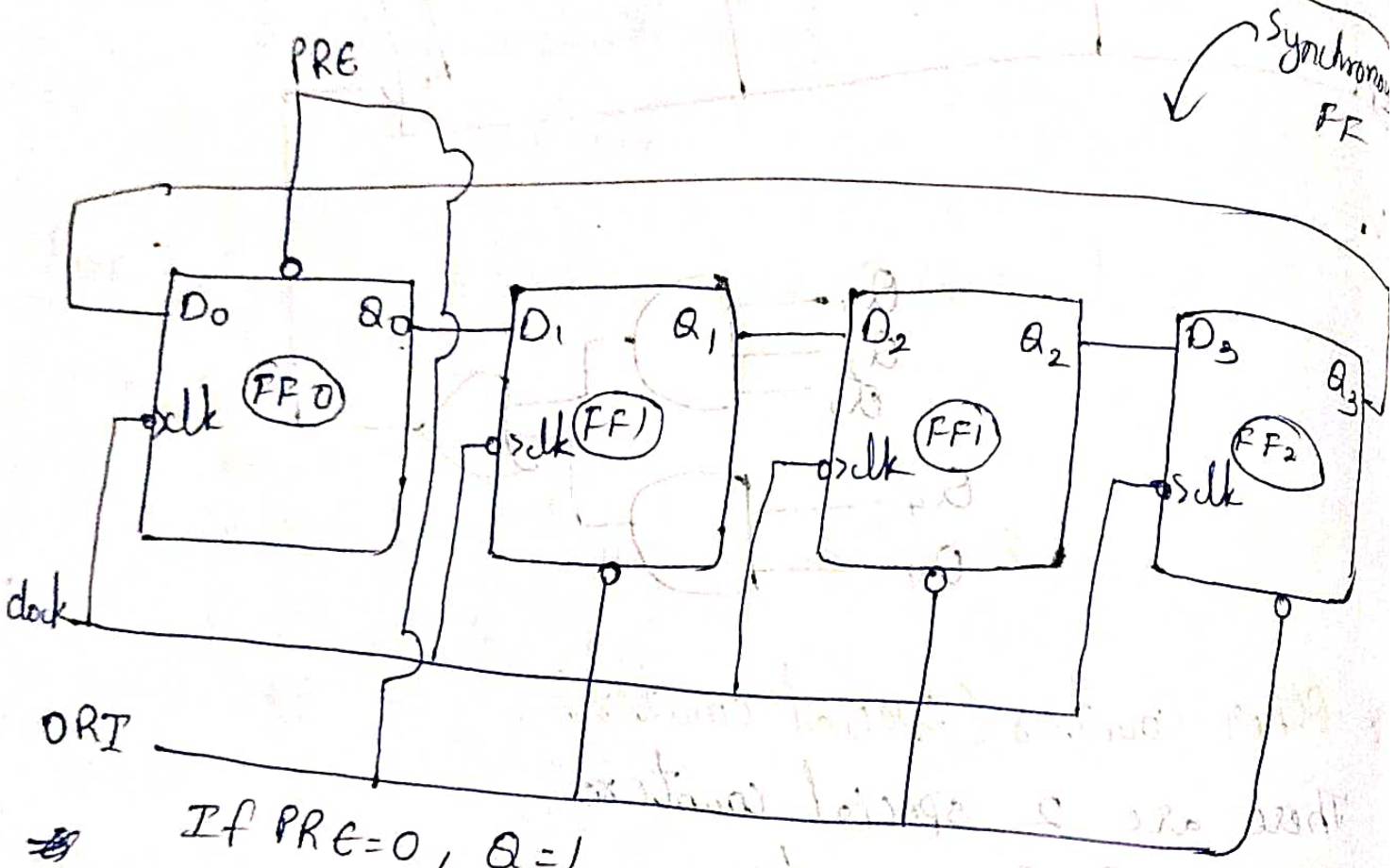
⇒ Counters can be designed to generate any desired sequence of states.

* Ring Counter:- It is one of the shift register counter.

⇒ * The output of last FF is connected to the input of first FF.

⇒ Here number of states = Number of FF's.

⇒ Here we are taking Over Riding Input (ORI) which is Asynchronous pin.



If PRE=0, Q=1

CLR=0; Q=0

→ Here we considered D FF ~~which~~. For DFF Output is same as Input means, Input will be the Output.

⇒ Here, the initial value is 1000, $Q_0\ Q_1\ Q_2\ Q_3$

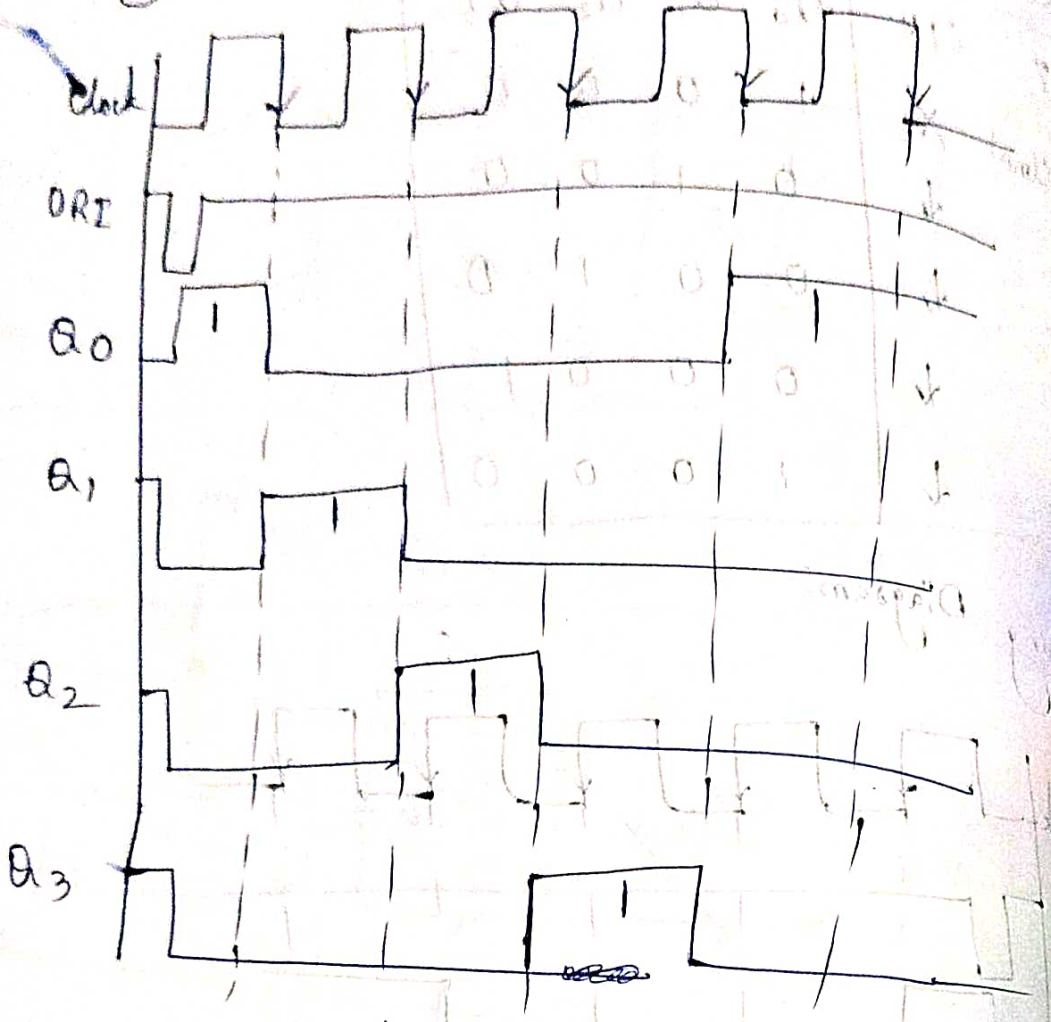
So we connect PRE to Q_0 and CLR to Q_1, Q_2, Q_3. PRE(PRESET) & CLR(clear) are active low signals.

→

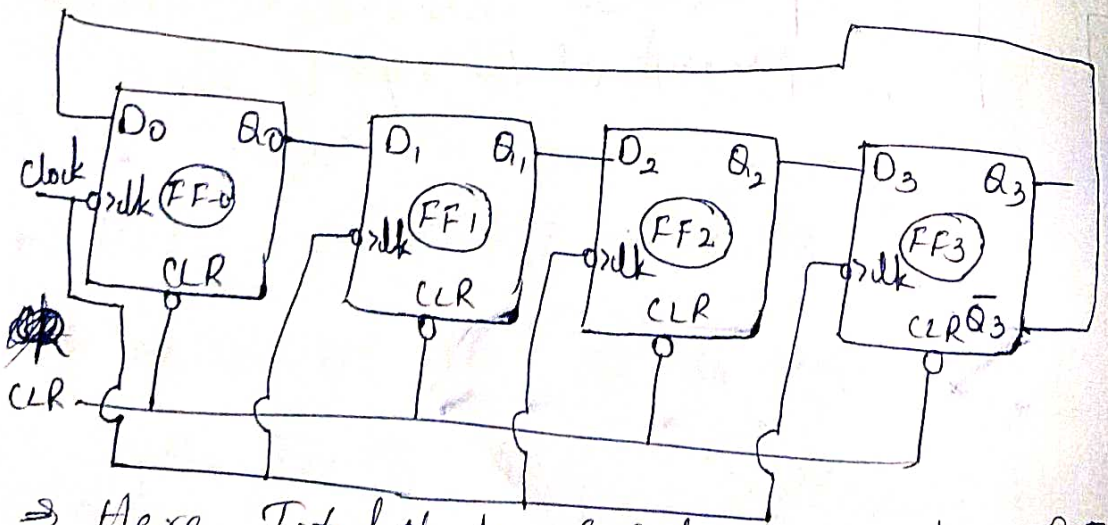
Truth Table:-

ORI	CLK	Q_0	Q_1	Q_2	Q_3
0 (low)	X	1	0	0	0
1	↓	0	1	0	0
1	↓	0	0	1	0
1	↓	0	0	0	1
1	↓	1	0	0	0

Timing Diagram:



* Johnson Counter:- Complemented output of last FF is connected as input to the 1st FF.



⇒ Here, Total Number of states = 2 × Number of FF's

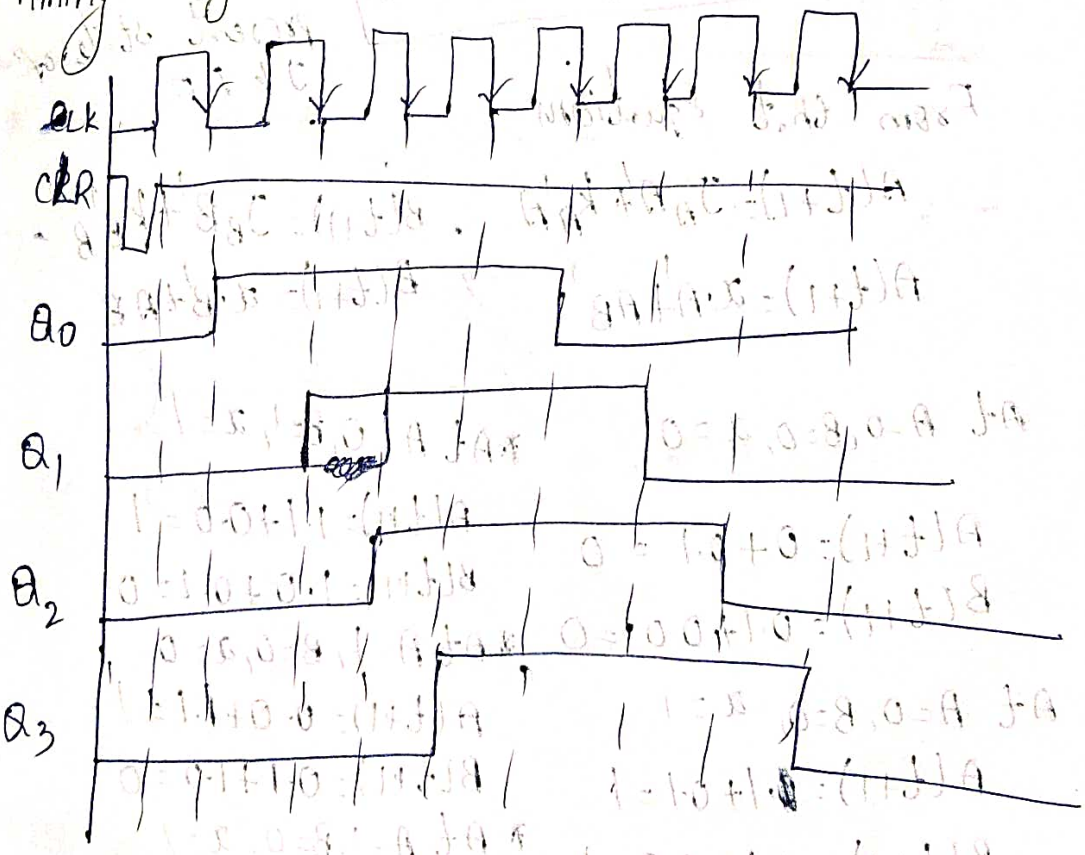
here FF's = 4

∴ Total Number of states = 2×4
= 8 states

Truth Table:

CLR	CLK	Q_0	Q_1	Q_2	Q_3
⊓	X	0	0	0	0
1	↓	1	0	0	0
1	↓	1	1	0	0
1	↓	1	1	1	0
1	↓	1	1	1	1
1	↓	0	1	1	1
1	↓	0	0	1	1
1	↓	0	0	0	1
1	↓	0	0	0	0

Timing Diagram:



Problems:

(1) A sequential circuit has two JK FF's A & B & one input x. The circuit is described by the following FF input Equations

$J_A = x,\ K_A = B;\ J_B = x$ & $K_B = A'$

(a) Derive the state equations A(t+1) & B(t+1) by substituting the input equations for the J & K variables.

(b) Draw the state diagram of the circuit

(A) Input Equations,

$J_A = x,\ K_A = B$ & $J_B = x,\ K_B = A'$

→ Charateristic Equation for JK FF is

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

← Q is the present state of JK FF

From that Equations,

$A(t+1) = J_A A' + K_A' A$. $B(t+1) = J_B B' + K_B' B$

$A(t+1) = x \cdot A' + AB'$ / $B(t+1) = x B' + AB$

* At A=0, B=0, x=0

$A(t+1) = 0 + 0 \cdot 1 = 0$

$B(t+1) = 0 \cdot 1 + 0 \cdot 0 = 0$

* At A=0, B=0, x=1

$A(t+1) = 0 \cdot 1 + 0 \cdot 1 = 1$

$B(t+1) = 1 \cdot 1 + 0 \cdot 0 = 1$

* At A=0, B=1, x=0

$A(t+1) = 0 \cdot 1 + 0 \cdot 0 = 0$

$B(t+1) = 0 \cdot 0 + 0 \cdot 0 = 0$

* At A=0, B=1, x=1

$A(t+1) = 1 \cdot 1 + 0 \cdot 0 = 1$

$B(t+1) = 1 \cdot 0 + 0 \cdot 1 = 0$

* At A=1, B=0, x=0

$A(t+1) = 0 \cdot 0 + 1 \cdot 1 = 1$

$B(t+1) = 0 \cdot 1 + 1 \cdot 0 = 0$

* At A=1, B=0, x=1

$A(t+1) = 1 \cdot 0 + 1 \cdot 1 = 1$

$B(t+1) = 1 \cdot 1 + 1 \cdot 0 = 1$

* At A=1, B=1, x=0

$A(t+1) = 0 \cdot 0 + 1 \cdot 0 = 0$

$B(t+1) = 0 \cdot 0 + 1 \cdot 1 = 1$

At A=1, B=1, x=1

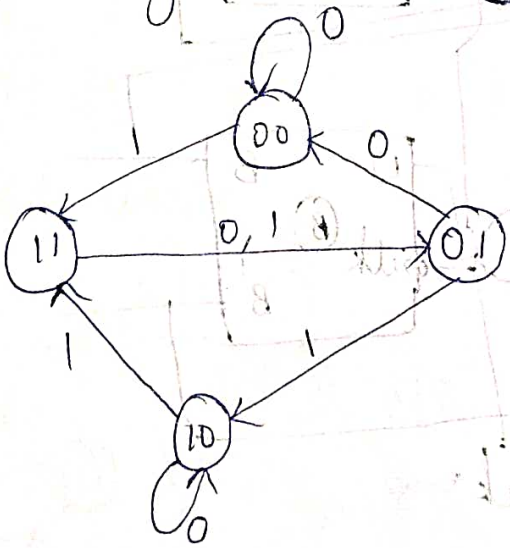
A(t+1) = 1.0+1.0 = 0

B(t+1) = 1.0+1.1 = 1

* state table:

Present state		Input	Next state	
A	B	x	A(t+1)	B(t+1)
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	0	1
1	1	1	0	0

* state diagram:



* A sequential circuit with two DFF's A & B, two inputs x & y & one input z is specified by the following next state & output equations.

$A(t+1) = xy' + xB$

$B(t+1) = xA + xB'$

$z = A$

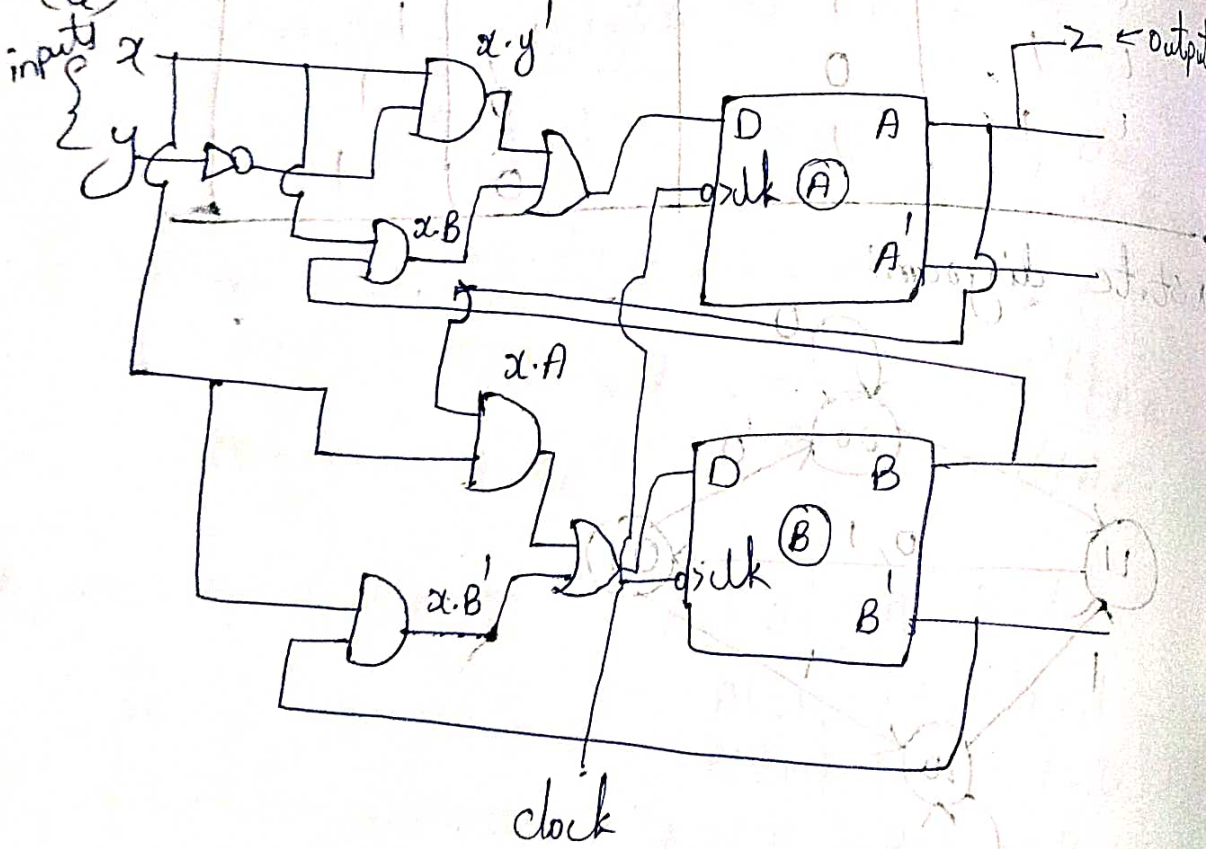
(a) Draw the logic diagram of the circuit

(b) List the state table for the sequential circuit

(c) Draw the corresponding state diagram

(A) $Q(t+1) = D$ for DFF's.

(a)



b) state table:

Present state A	Present state B	Inputs x	Inputs y	Next state A(t+1)	Next state B(t+1)	Output z
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	1	1	0
0	1	0	0	0	1	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	1	0	0
1	0	0	0	1	0	1
1	0	0	1	0	0	1
1	0	0	0	0	0	1
1	0	1	1	1	1	1
1	1	0	0	0	1	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	1	1	1

(c) state diagram:-