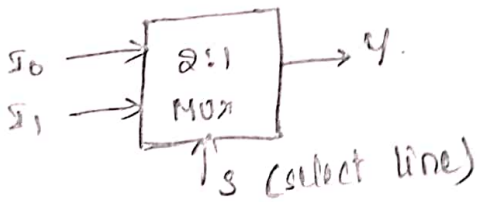
29/1/21

Multiplexers:

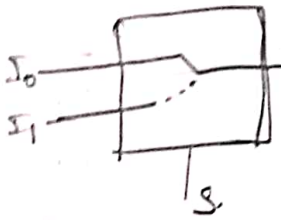
Selects binary input from many input lines.

(input : output ⇒ 2:1, 4:1, 8:1, 16:1, ...).

2:1 Mux:



(ii)

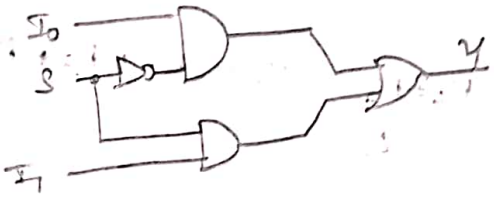


Output 'y' can be connected to only one input (I_0/I_1) at a time (to depends on select line).

(iii) eg: if $S=0$, $y=I_0$.

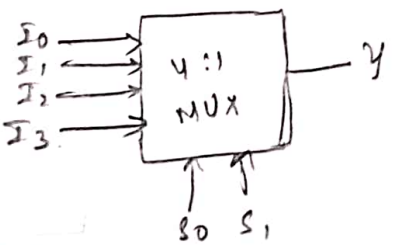
if $S=1$, $y=I_1$

Here, $y = I_0\bar{S} + I_1 S$



⇒ 2:1 multiplexer.

4:1 MUX:

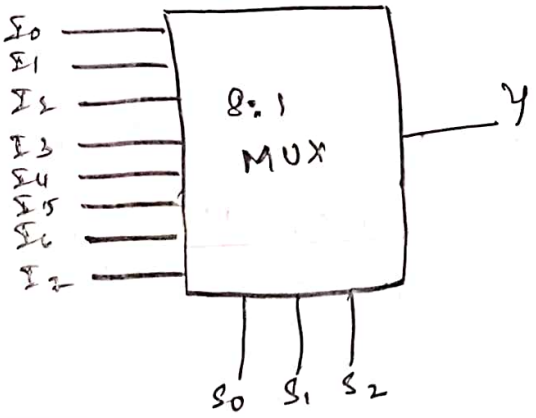


If no. of select lines = n, then, input lines = 2^n.

S_0	S_1	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

$Y = I_0 \bar{S}_0 \bar{S}_1 + I_1 \bar{S}_0 S_1 + I_2 S_0 \bar{S}_1 + I_3 S_0 S_1$.

8:1 MUX :

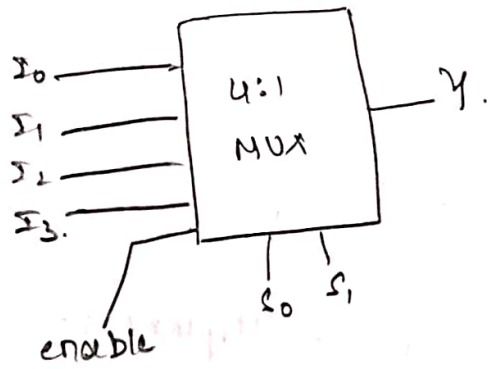


S_0	S_1	S_2	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

$Y = I_0 \bar{S}_0 \bar{S}_1 \bar{S}_2 + I_1 \bar{S}_0 \bar{S}_1 S_2 + I_2 \bar{S}_0 S_1 \bar{S}_2 + I_3 \bar{S}_0 S_1 S_2 + I_4 S_0 \bar{S}_1 \bar{S}_2 + I_5 S_0 \bar{S}_1 S_2 + I_6 S_0 S_1 \bar{S}_2 + I_7 S_0 S_1 S_2$.

Enable : (eg : for 4:1 MUX).

E	S_0	S_1	Y.
0	x	x	0
1	0	0	I_0
1	0	1	I_1
1	1	0	I_2
1	1	1	I_3

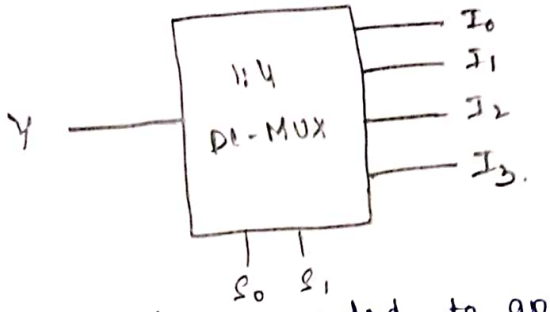


We can have enables for 8:1 & 16:1 --- multiplexers.

De-Multiplexers:

(input : output ⇒ 1:2, 1:4, 1:8, 1:16 ---)

eg: for 1:4 De-MUX:



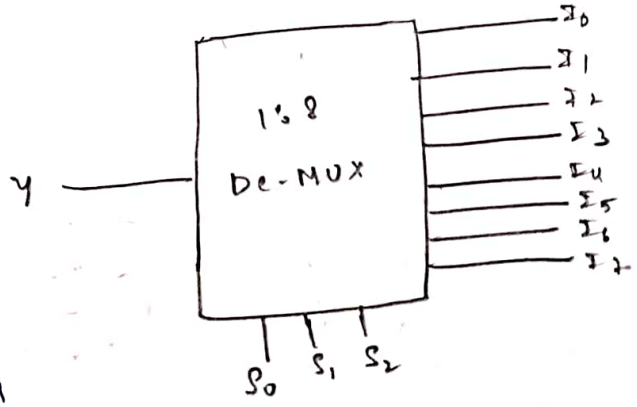
y	S_0	S_1	I_0	I_1	I_2	I_3
1(0)	0	0	1(0)	x	x	x
1(0)	0	1	x	1(0)	x	x
1(0)	1	0	x	x	1(0)	x
1(0)	1	1	x	x	x	1(0)

input gets connected to any one of the outputs.

Outputs : $I_0 = y\bar{I}_0\bar{S}_1$; $I_1 = y\bar{I}_0 S_1$; $I_2 = y S_0\bar{S}_1$; $I_3 = y S_0 S_1$

eg: for 1:8 De-MUX:

y	S_0	S_1	S_2	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
1	0	0	0	1	x	x	x	x	x	x	x
1	0	0	1	x	1	x	x	x	x	x	x
1	0	1	0	x	x	1	x	x	x	x	x
1	0	1	1	x	x	x	1	x	x	x	x
1	1	0	0	x	x	x	x	1	x	x	x
1	1	0	1	x	x	x	x	x	1	x	x
1	1	1	0	x	x	x	x	x	x	1	x
1	1	1	1	x	x	x	x	x	x	x	1



$I_0 = y\bar{S}_0\bar{S}_1\bar{S}_2$

$I_1 = y\bar{S}_0\bar{S}_1 S_2$

$I_2 = y\bar{S}_0 S_1\bar{S}_2$

$I_3 = y\bar{S}_0 S_1 S_2$

$I_4 = y S_0\bar{S}_1\bar{S}_2$

$I_5 = y S_0\bar{S}_1 S_2$

$I_6 = y S_0 S_1\bar{S}_2$

$I_7 = y S_0 S_1 S_2$

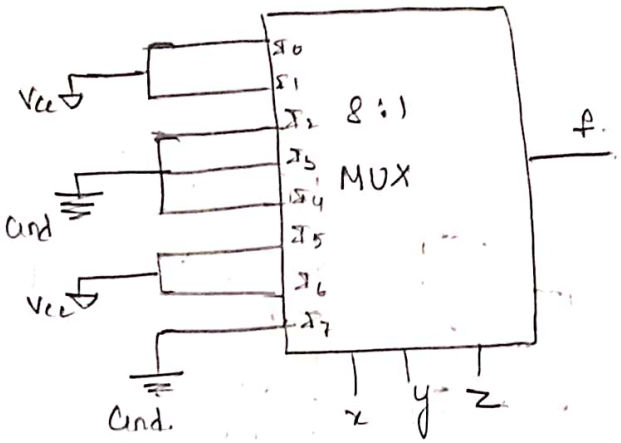
1/2/21

Example problems on multiplexers:

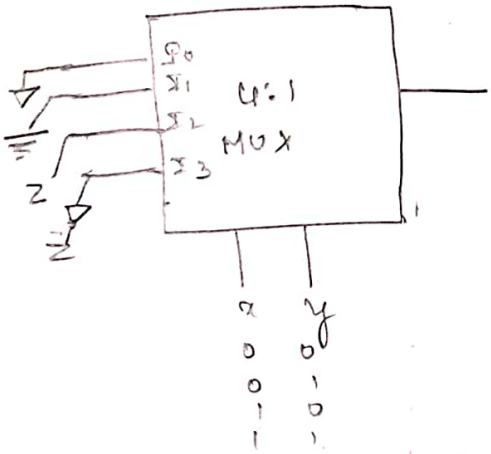
(1) Implement $f(x, y, z) = \pi(2, 3, 4, 7)$ using 8:1 MUX, 4:1 MUX, 8:1 MUX.

x	y	z	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Using 8:1 mux:

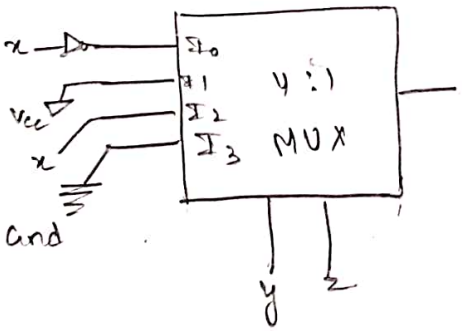


Using 4:1 MUX:



x	y	z	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Using 4:1 MUX:

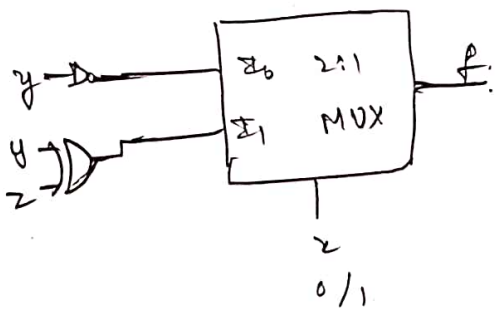


x	y	z	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Why we can give z & x to selection pins.

Using 2:1 MUX:

x	y	z	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



y	z	f
0	0	1
0	1	1
1	0	0
1	1	0

$f = \bar{y}\bar{z} + \bar{y}z$

$= \bar{y}(z + \bar{z})$

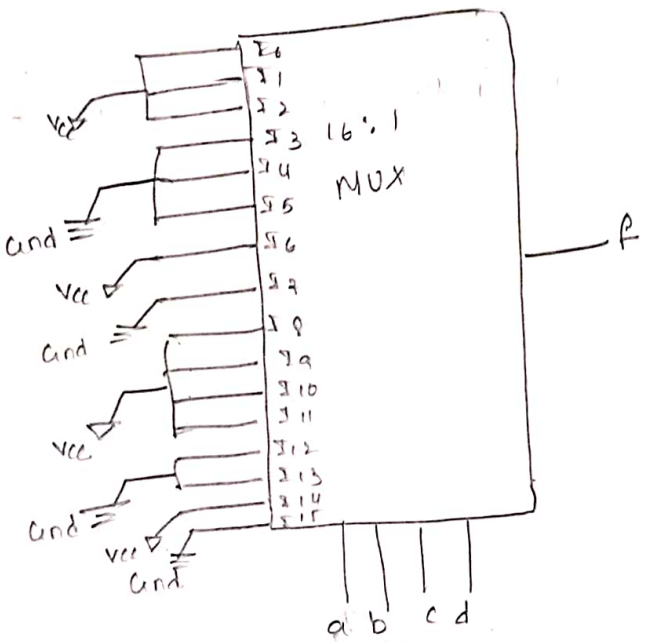
$= \bar{y}$

(for I_0)

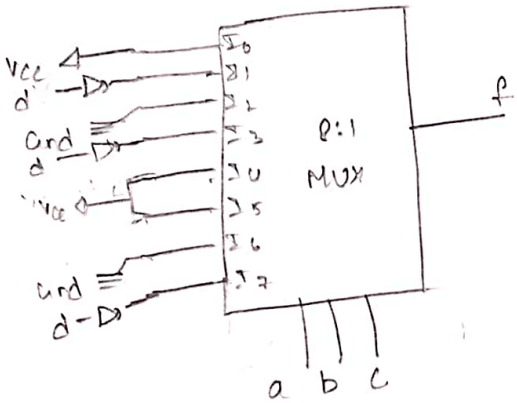
(2) Implement $f(a,b,c,d) = a\bar{b}c + \bar{b}\bar{c} + c\bar{d}$ using 16:1 MUX.

a	b	c	d	f
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

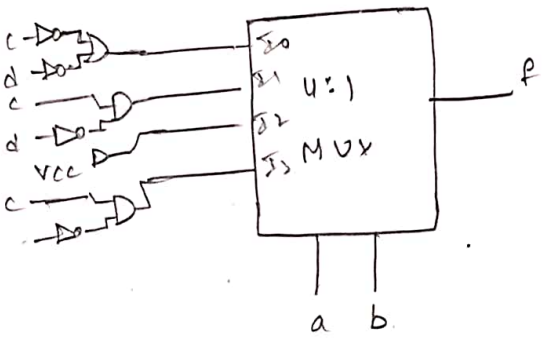
Using 16:1 MUX:



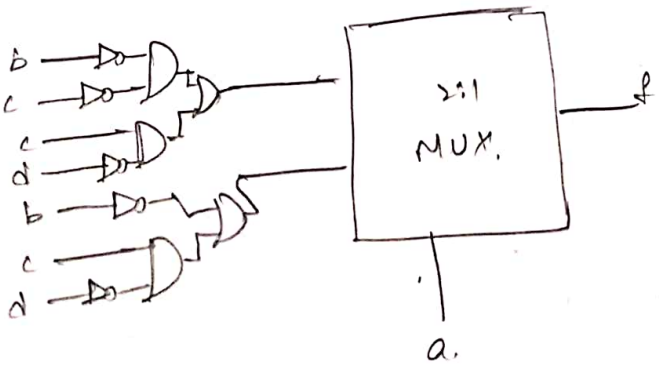
Using 8:1 MUX:



Using 4:1 MUX.

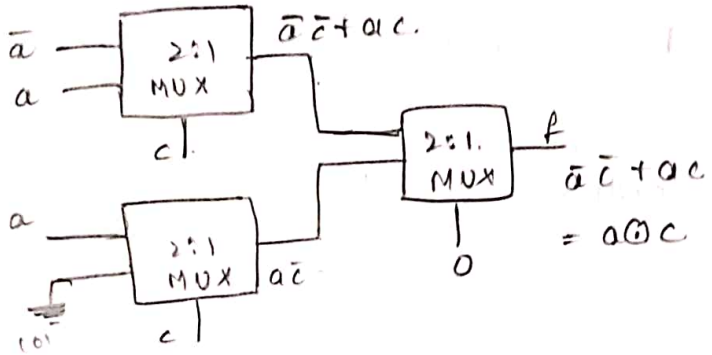


Using 2:1 MUX:

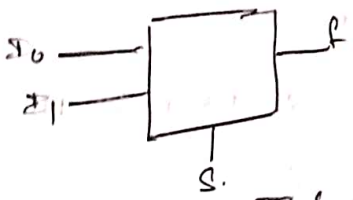


(3)

(3) find f.



since

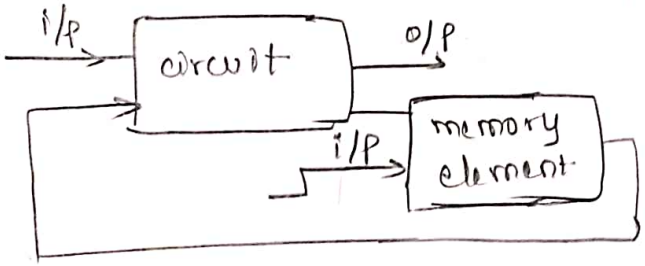


$f = I_0 \bar{s} + I_1 s$

2/2/21

Sequencial circuits

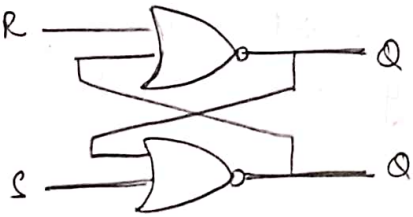
A combinational circuit along with a memory element is called sequencial circuits.



SR latch:

There are two types of SR latches: NOR based, NAND based.

NOR based; (cross coupled circuit)



NOR Gate:

input		output
0	0	1
0	1	0
1	0	0
1	1	0

if any one of the inputs is high the output is 0.

S	R	Q	Q'
0	0	x	x
1	0	1	0
0	0	1	0
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1

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

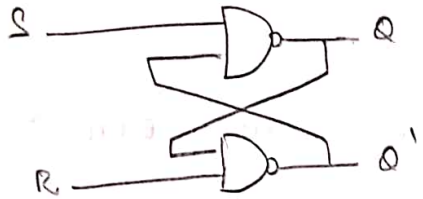
(not valid)

(∵ Q = Q' which is not possible)

Being in 0 0 state is same as being in memory state for the SR latch. The output donot change when S & R are in 0 0 state.

Here 1 bit is stored in the output 'Q' (or Q').

SR latch (NAND):



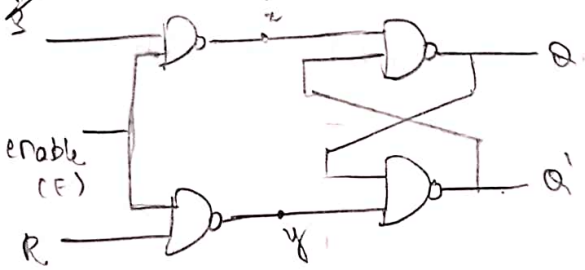
Input		output
0	0	1
0	1	1
1	0	1
1	1	0

S	R	Q	Q'
1	1	x	x. (don't know)
0	1	1	0
1	1	1	0
1	0	0	1
1	1	0	1
1	1	0	1
0	0	1	1 (invalid)

For NAND based SR latch 1 1 is the memory state.

S	R	Q	Q'
1	1	memory	
0	1	1	0
1	0	0	1
0	0	invalid	

5/2/21



$x = \overline{S \cdot E} = \bar{S} + \bar{E}$

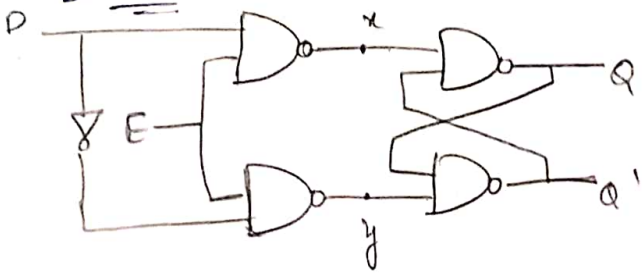
$y = \overline{R \cdot E} = \bar{R} + \bar{E}$

when enable = 0, $x = y = 1$ (in memory state)

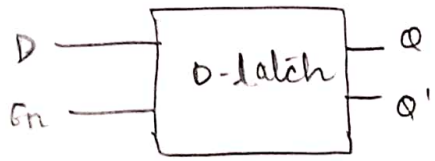
when enable = 1, $x = \bar{S}$, $y = \bar{R}$

E	S	R	x	y	Q	Q'
1	1	1	0	0	invalid	
1	0	0	1	1	memory	
1	1	0	0	1	1	0
1	0	0	1	1	1	0
1	0	1	1	0	0	1
1	0	0	1	1	0	1

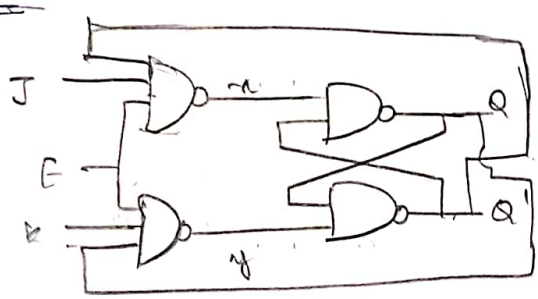
D-latch:



E	D	Q	Q'
0	x	memory state	
1	0	0	1
0	x	0	1
1	1	1	0
0	x	1	0



J-K latch:



$x = \overline{J \cdot E \bar{Q}} = \bar{J} + \bar{E} + Q$

$y = \overline{K E Q} = \bar{K} + \bar{E} + \bar{Q}$

If enable = 0; $x = y = 1$ (in memory state)

If enable = 1; $x = \bar{J} + Q$, $y = \bar{K} + \bar{Q}$

inputs | intermediate inputs | outputs

Q	J	K	x	y	Q	Q'
1	1	0	1	1	1	0
1	0	1	1	0	0	1
0	0	0	1	1	0	1
0	1	1	0	1	1	0
1	0	0	1	1	1	0
1	1	1	1	0	0	1
0	1	1	0	1	1	0
1	1	1	1	0	0	1
1	1	1	1	0	0	1
0	1	1	0	1	1	0
0	1	1	0	1	1	0
1	1	1	1	0	0	1
1	1	1	1	0	0	1

$J = k = 0 \Rightarrow$ memory state

$J = 1, k = 0 \Rightarrow Q = 1 \quad Q' = 0$

$J = 0, k = 1 \Rightarrow Q = 0, Q' = 1$

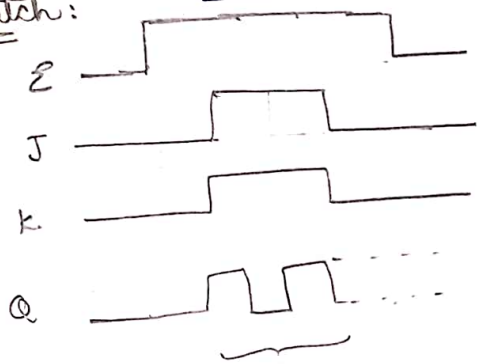
$J = k = 1 \Rightarrow Q$ is complement of previous output Q.

↳ racing condition

Q	J	k	Q	Q'
0	0	0	0	1
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
0	0	0	0	1
0	0	1	0	1
1	1	0	1	0
0	1	1	0	1

Timing diagrams:

8/2/21 JK latch:

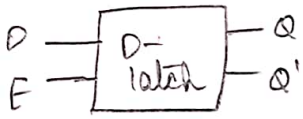


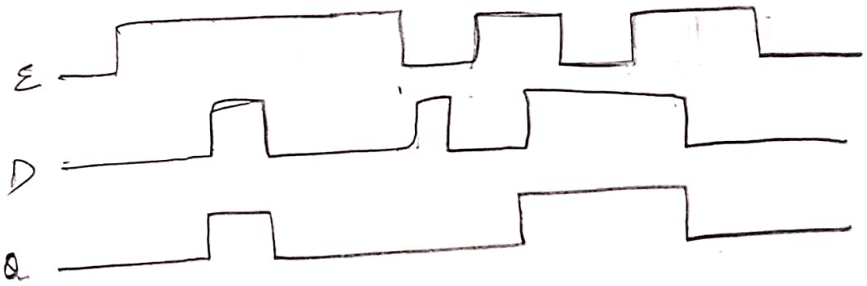
E	J	k	Q	Q'
0	0	0	0	1
1	0	0	0	1
1	1	1	1	0
1	1	1	0	1
1	0	0	1	0
0	0	0	1	0

y axis - amplitude
x axis - time

↳ racing behaviour.

For D-latch:





Enable: It is a random signal which can be changed by the user.

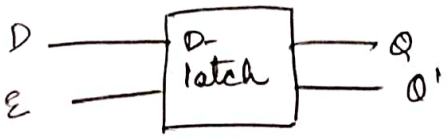
random enable signal :

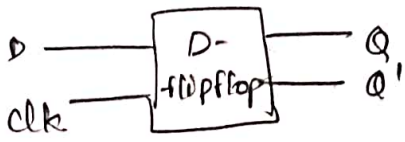
periodic enable signal :

↳ fixed time period $\left(T = \frac{1}{f}\right)$

↳ clock input

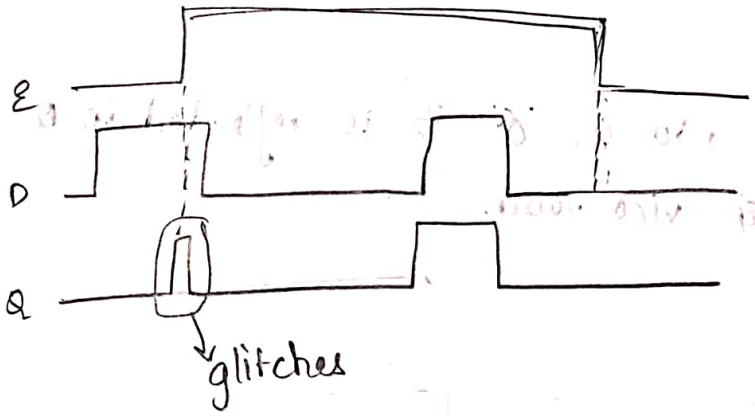
If the enable ~~is~~ turns to be the clock input, the D-latch becomes D-flipflop.





for D-latch: (disadvantage)

latches are level sensitive circuits (depends on i/p (or) enable signals being high or low). → two types — high (en)
— low (en)

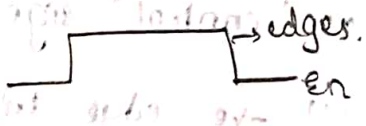


⊗ when 'Q' becomes high for a very small period of time, the jerks produced in 'Q' are called glitches.

So, to avoid these unwanted disturbances and glitches, we use edge-sensitive circuits instead of level-sensitive circuits.

9/2/21

edges can be, high to low or low to high

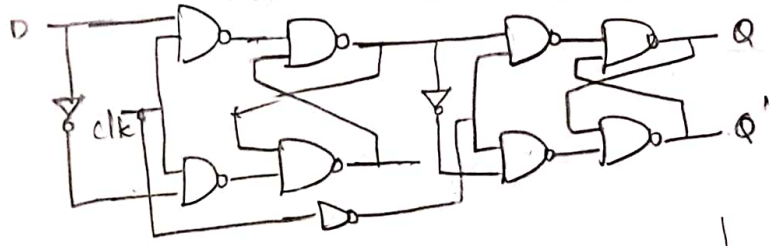


→ there are two types of edge sensitive circuits:

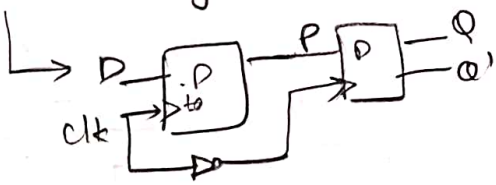
(1) positive (0 to 1) } → enable
(2) negative (1 to 0) }

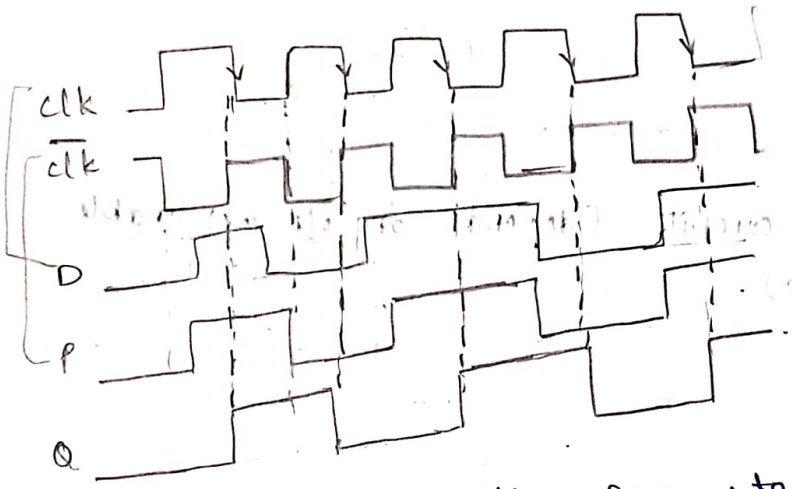
→ Converting level-sensitive to edge-sensitive

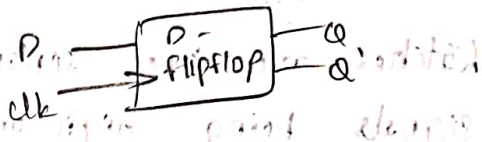
-ve edge sensitive D-flipflop:



} combining two d-flipflops make it edge-sensitive



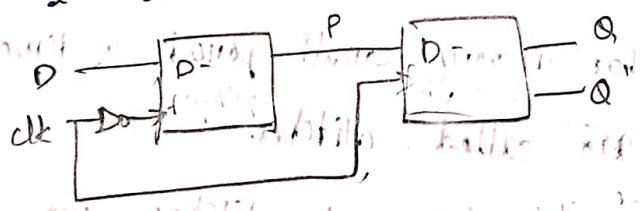


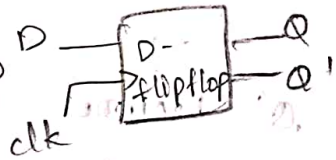


(edge sensitive) (-ve)
↳ wrt clk

whenever, clk is falling from 1 to 0, if it is reflected in Q if Q is low it becomes high & vice versa

+ve edge sensitive:



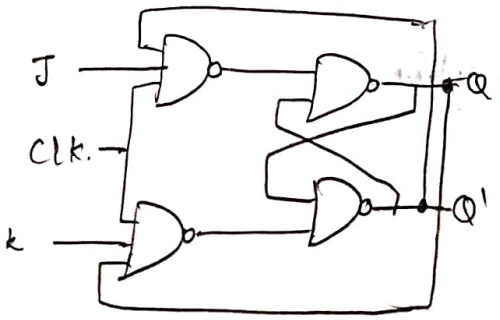


If clk is involved, that signal is called 'triggering' signal or 'control' signal. Two types can be called:

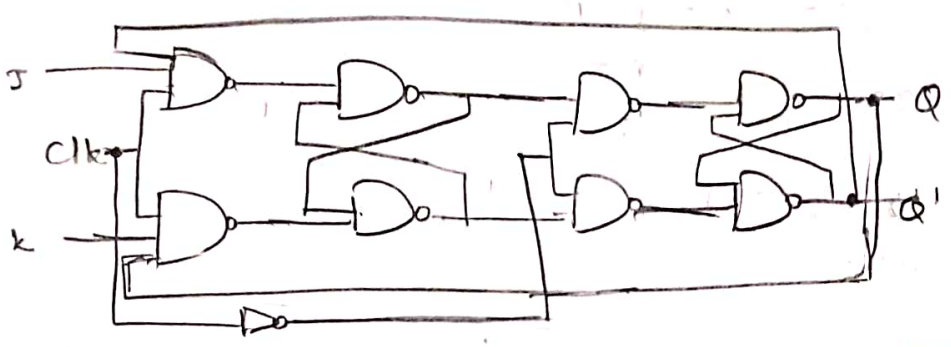
a) -ve edge triggering (or) falling edge triggering
b) +ve edge triggering (or) raising edge triggering

11/2/21

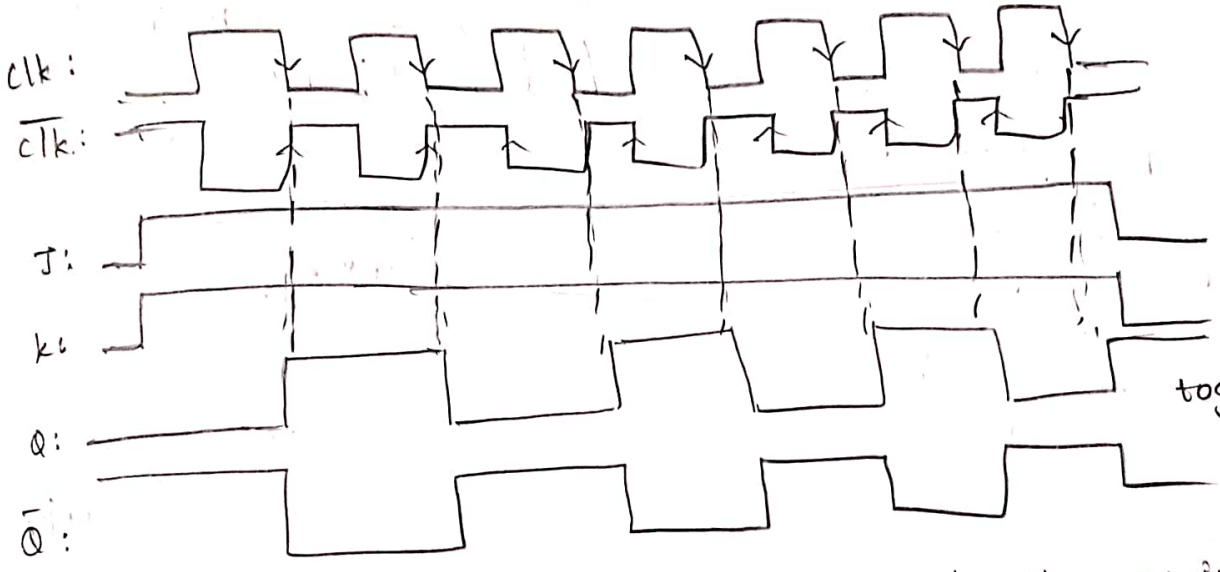
Difference b/w JK latch & JK flipflop:



clk = J = k = 1
→ racing condition



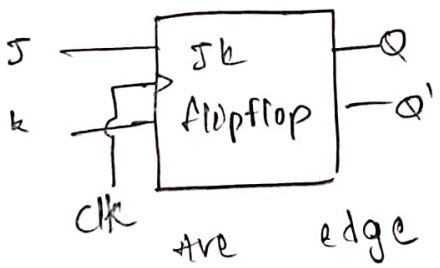
Negtive edge
Jk flipflop



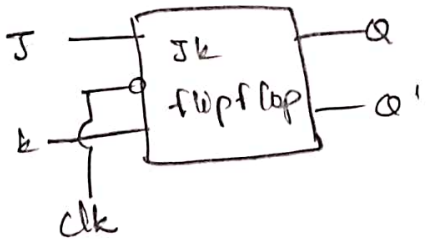
toggling behaviour

when clk goes to 0, J = k = 1, Q complements its previous value.

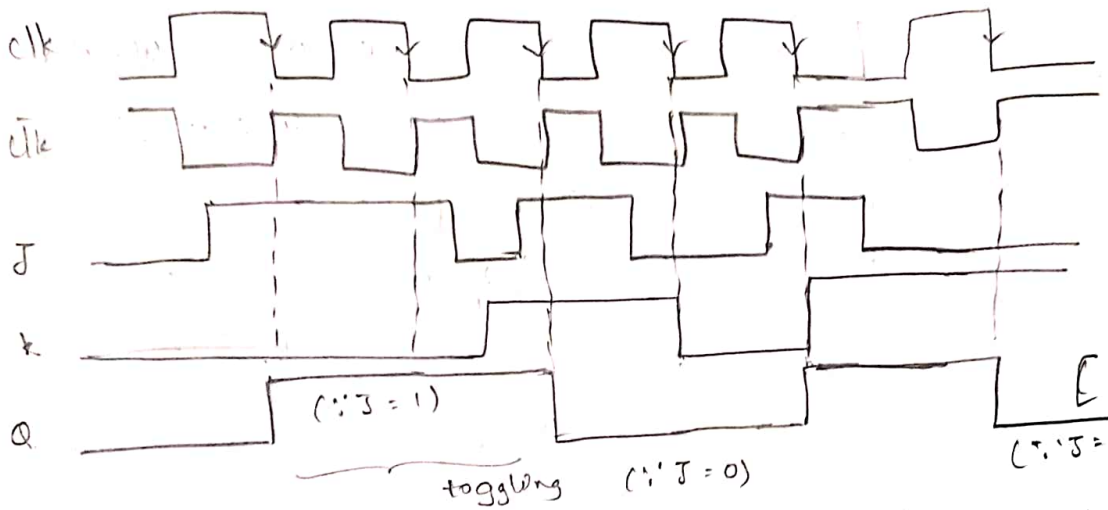
To get a positive edge Jk flipflop, just interchange the clock signal & complement of clock signal inputs (i) ~~to~~ $\overline{clk}$ - given to 1st Jk flipflop & clk - given to 2nd Jk flipflop.



-ve edge



latch	flipflop
(1) Enable signal	(1) clk signal
(2) level sensitive	(2) edge sensitive / level sensitive



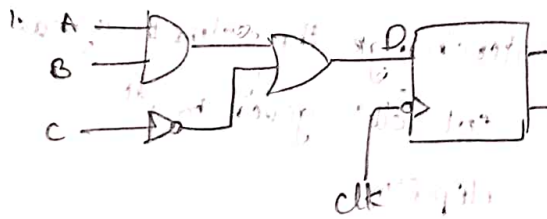
J	k	Q	Q'
1	0	1	0
0	1	0	1
0	0	memory state	
1	1	complement	

(ii) if clk = 2 Hz (eg)
Q = 1 Hz
if Q = clk
output = 0.5 Hz and so on } in case of toggling

15/2/21

Applications:

1.



A	B	C	D	after one clock pulse Q(t+1) Q	Q'
0	0	0	1	1	0
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	0	0	1

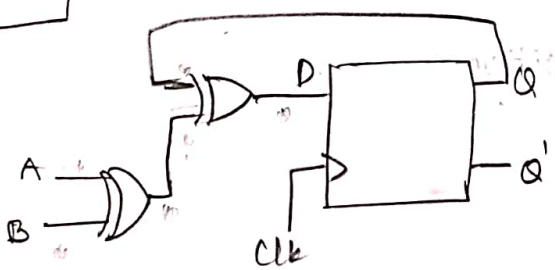
(a) A
(b) B
(c) C
clk
d
Q

$D(t) = AB + \bar{C}$ (state eqn).

The truth table here is called state table.

Q(t+1) = next state

2.



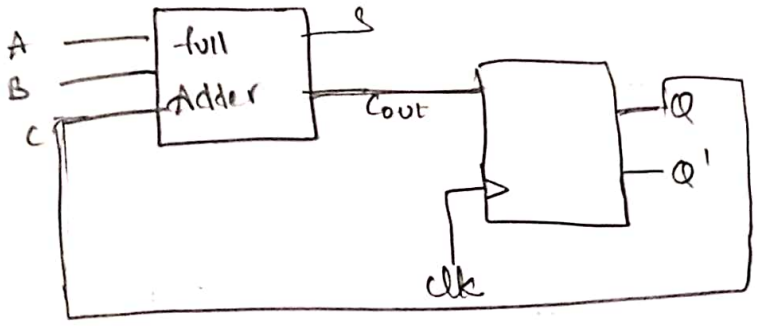
A	B	Q	(P.S) D(t)	(NS) Q'(t+1)
0	0	0	0	0
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
0	0	1	1	1
1	0	1	0	0

and so on

$D(t) = A \oplus B \oplus Q$

Q(t) → present state (∵ o/p going to i/p)

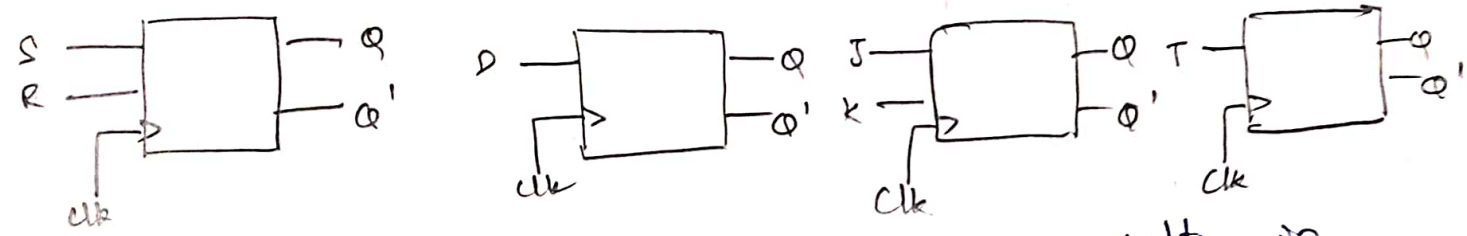
3



A	B	C	S (PS)	Cout (PS)	Q(t+1) (NS)
0	0	0	0	0	0
0	1	0	1	0	0
1	0	0	1	0	0
1	1	0	0	1	1
1	1	1	1	1	1

and so on.

16/2/21



When j,k combine to be a single input, it results in T-flipflop.

T=0 (J=k=0) → circuit is in memory state.

T=1 (J=k=1) → output toggles

Counters:

In BCD to 7-segment decoder, the output changes according to the input

Counters can count the numbers itself (0 1 2 3 4 5 ...).

without changing inputs.

Applications: digital clock (0→60) (0→60 min) (0→12 hrs)

sec, min, hrs all have different frequencies.

Designing of a counter:

→ We use T-flipflops.

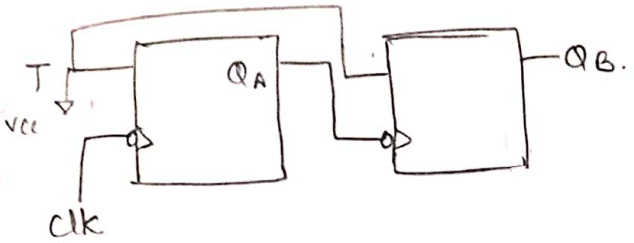
There are two types of counters:

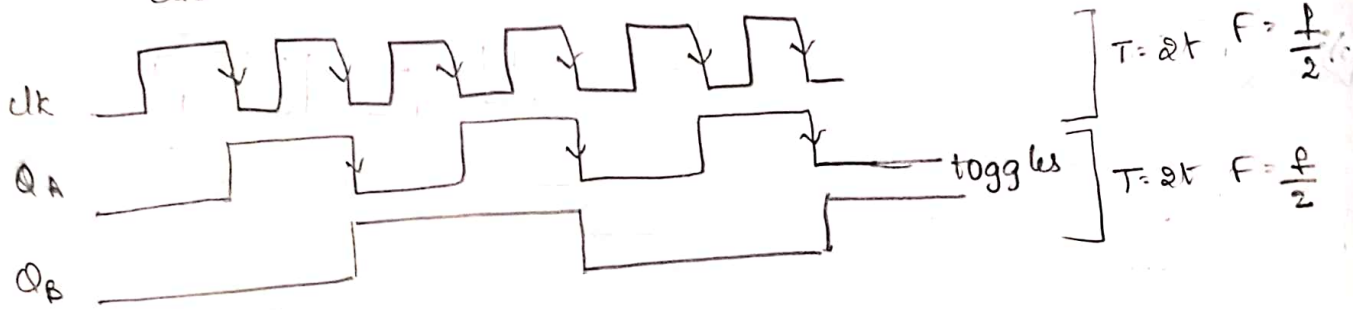
(1) asynchronous counter (or) ripple counter

(2) synchronous counter.

Asynchronous counters:

→ 2 flip-flops are used



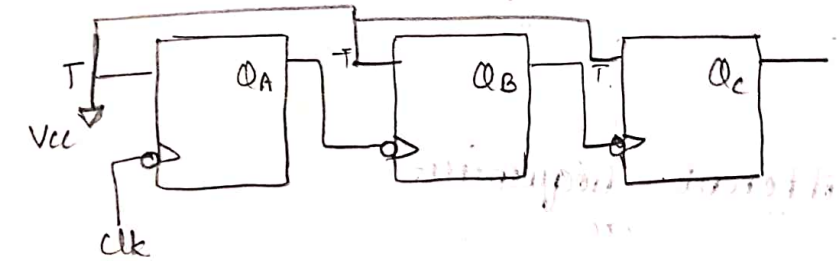


Q_B	Q_A	
0	0	] after 1st clk pulse
0	1	] after 2nd clk pulse
1	0	] after 3rd clk pulse
1	1	
0	0	
0	1	
1	0	
1	1	

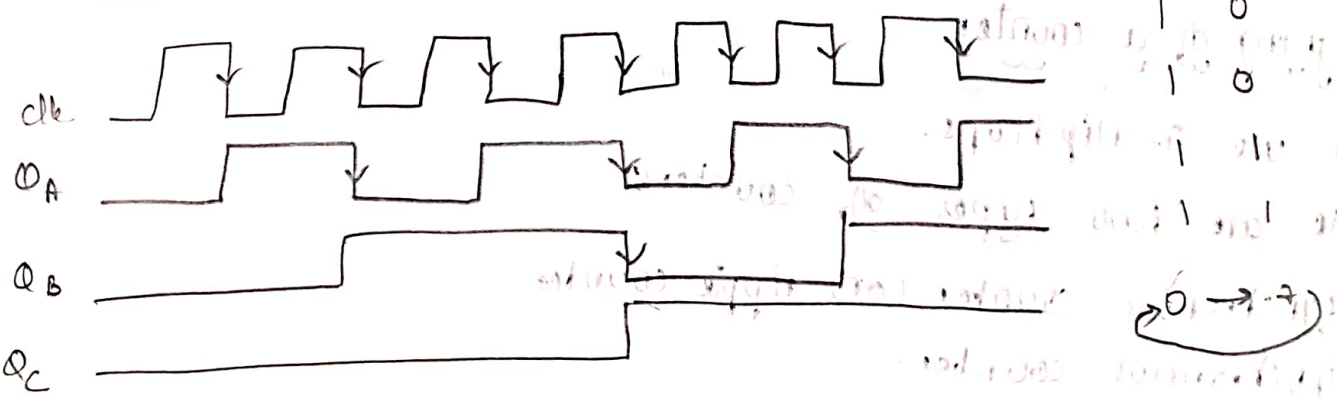
0
1
2
3
0
1
2
3

→ 0 → 1 → 2 → 3

If we give the 1st two of 4 inputs in a 7-segment decoder as Q_B, Q_A. without changing inputs manually output changes from 1 to 3.

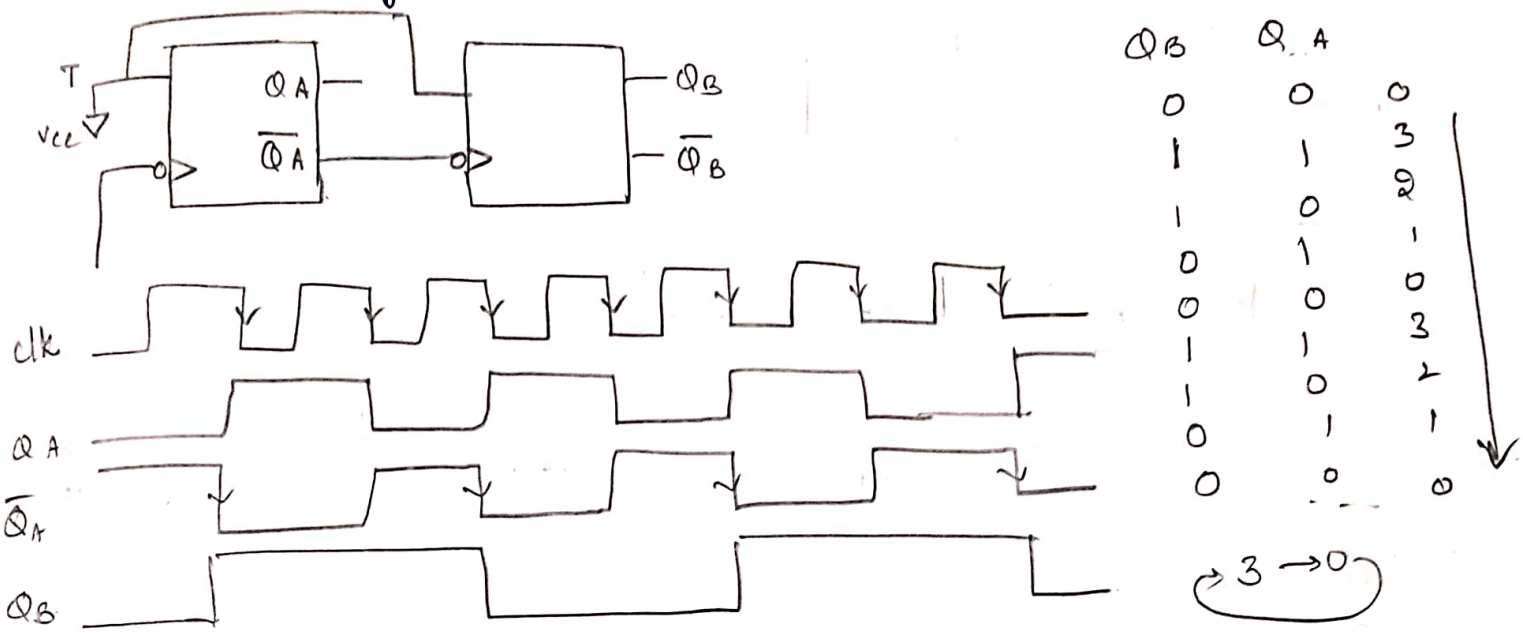


Q_C	Q_B	Q_A
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1



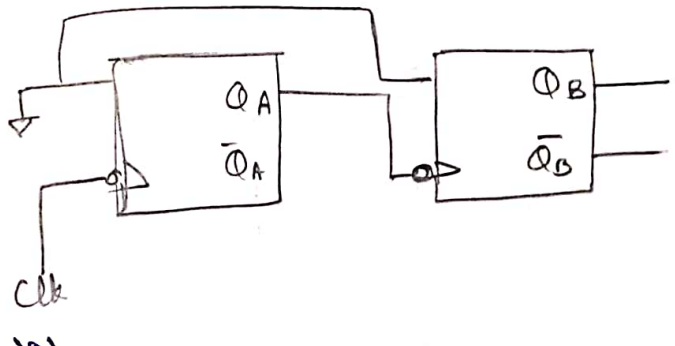
If no. of flipflops = n, count range = 0 to $2^n - 1$.

For down counting:



Q_B	Q_A	
0	0	0
1	1	3
1	0	2
0	1	1
0	0	0
1	1	3
1	0	2
0	1	1
0	0	0

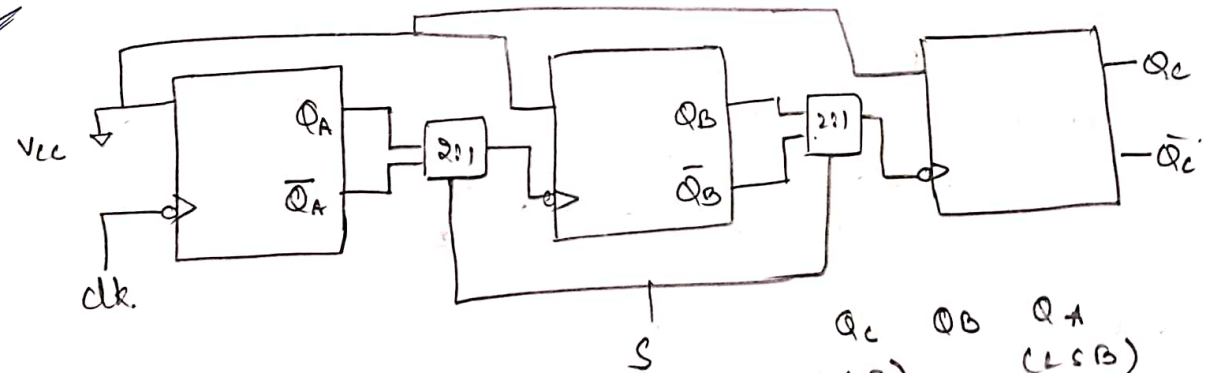
3 → 0



$\bar{Q}_B$	$\bar{Q}_A$	
1	1	3
1	0	2
0	1	1
0	0	0
1	1	3
1	0	2

(down down counting)

19/2/21



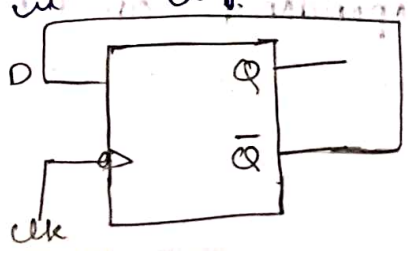
S = 0; upcounting upto 7. (Q_C Q_B Q_A) (MSB) (Q_C Q_B Q_A) (LSB)

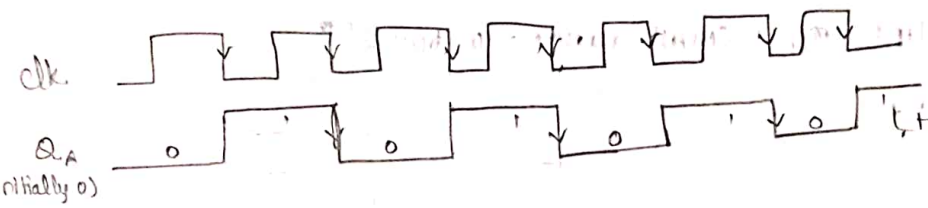
S = 1; downcounting upto 7. (Q_C Q_B Q_A)

S = 0, down → ($\bar{Q}_C$ $\bar{Q}_B$ $\bar{Q}_A$)

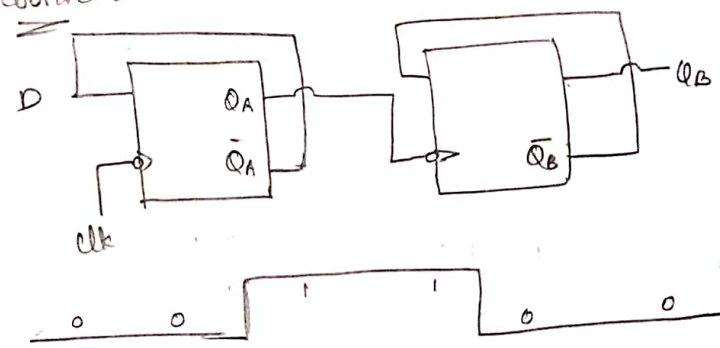
S = 1, up → ($\bar{Q}_C$ Q_B Q_A)

Counter using D-flip flop:





Upcounter:

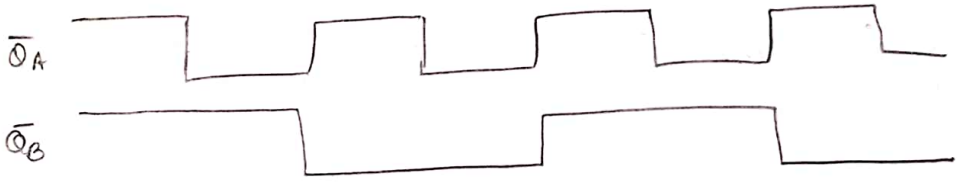


Q_B	Q_A
0	0
0	1
1	0
1	1
0	0
0	1

upcounting
0 → 3

Down counter:

Considering $\bar{Q}_A$ $\bar{Q}_B$ instead of Q_A Q_B.



$\bar{Q}_B$	$\bar{Q}_A$
1	1
1	0
0	1
0	0
1	1

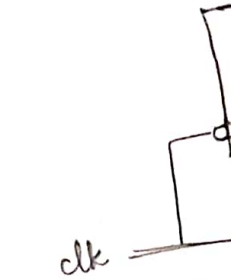
down counting
3 → 0

Negative edge

	Up	down	LSB/NSB
(T=1) T	Q → clk of nxt flipflop	$\bar{Q}$ → clk of nxt flipflop Q → clk of nxt flipflop	outputs from Q_A & Q_B Outputs from $\bar{Q}_A$ & $\bar{Q}_B$
(D=$\bar{Q}$) D	Q → clk of nxt flipflop ↓ outputs from Q_A & Q_B	Q → clk of nxt flipflop ↓ Outputs from $\bar{Q}_A$ & $\bar{Q}_B$	

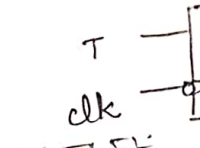
In asynchronous counters, each flipflop is dependent on the other flip flop. (∵ clk of 1 flipflop depends on output of previous flipflop)

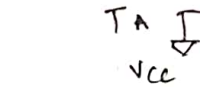
Synchronous



Complexity

→ Using T





T

clk

Both

both

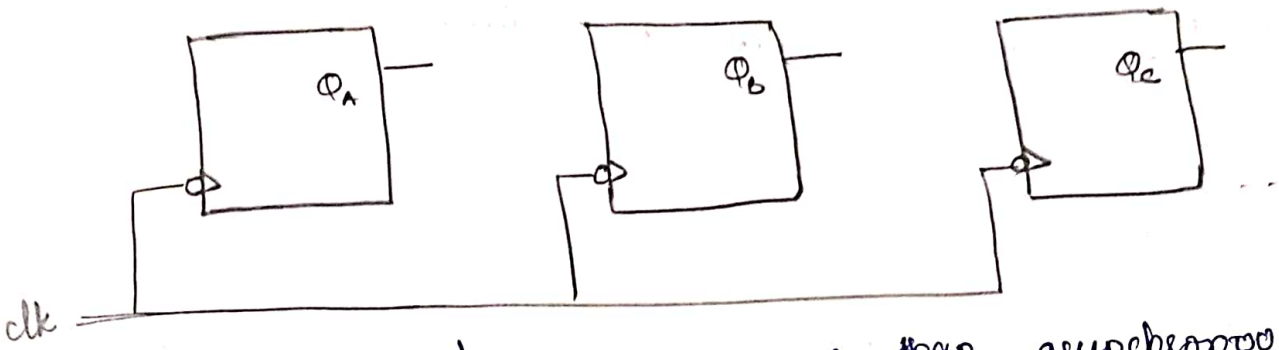
be

T.

V_{cc}

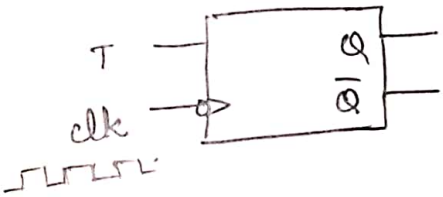
clk

Synchronous counters (faster than asynchronous)
(∵ clk of ff donot depend on previous output)

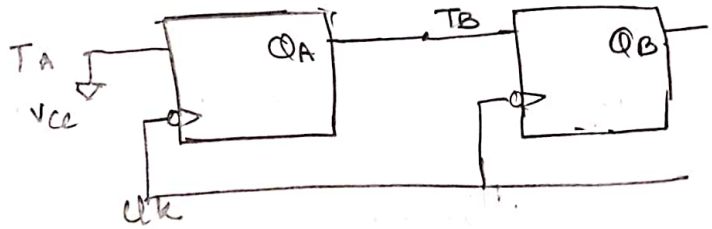


Complexity of synchronous is more than asynchronous.

→ Using T-flipflop.

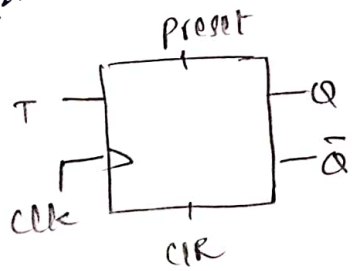


T	(PS) present state	(NS) next state
0	0	0
0	1	1
1	0	1
1	1	0



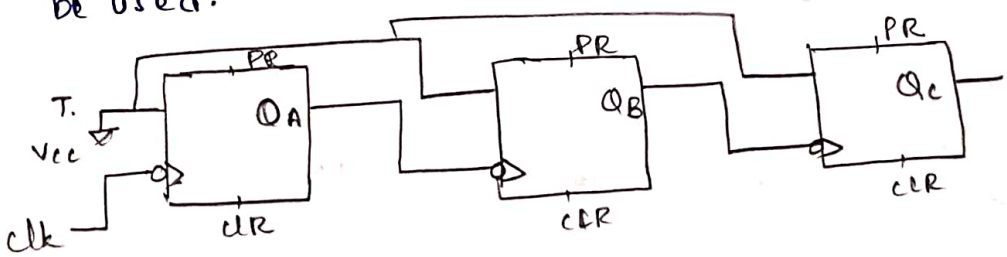
(PS) Q_B	(PS) Q_A	(NS) Q_B	(NS) Q_A	T_B	T_A
0	0 (0)	0	1 (1)	0	1
0	1 (1)	1	0 (2)	1	1
1	0 (2)	1	1 (3)	0	1
1	1 (3)	0	0 (0)	1	1

22/12/21



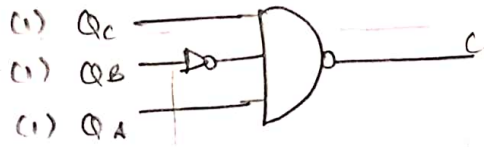
Preset = 0, Q = 1 [regardless of input values]
Preset = 1, Q depends on input
clear = 0, Q = 0 (regardless of inputs)
clear = 1, Q depends on input

Both preset & clear are not given simultaneously (ie) both cannot be used simultaneously, any one of them can be used.

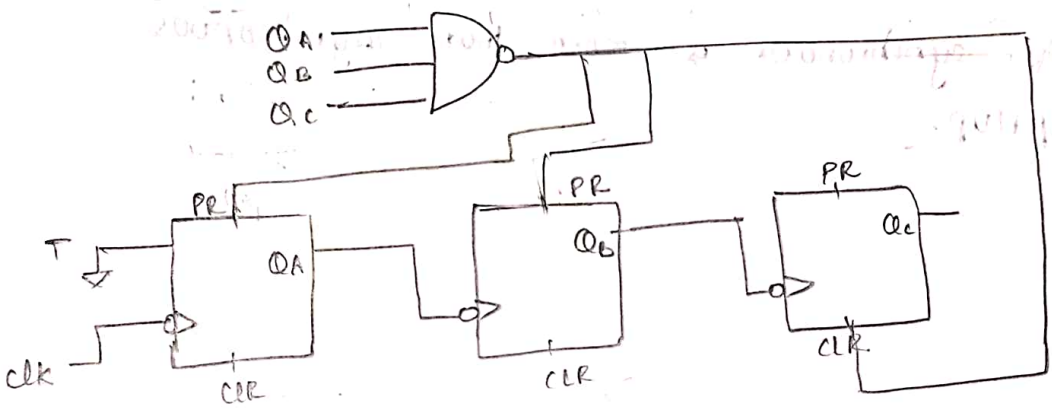


Q_C	Q_B	Q_A	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

→ To count from 0 to 4, [5 = 101 → all to be 1's]

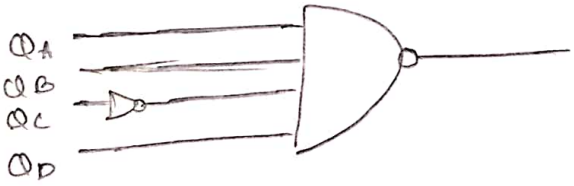


→ To count from 3 to 6 [7 = 111 → all to be is for input of nand]

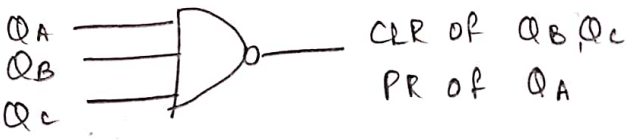


$Q_C Q_B Q_A$
3 = 0 1 1
↓
forcing $Q_C = 0$
↓
output of nand is connected to CLR of Q_C

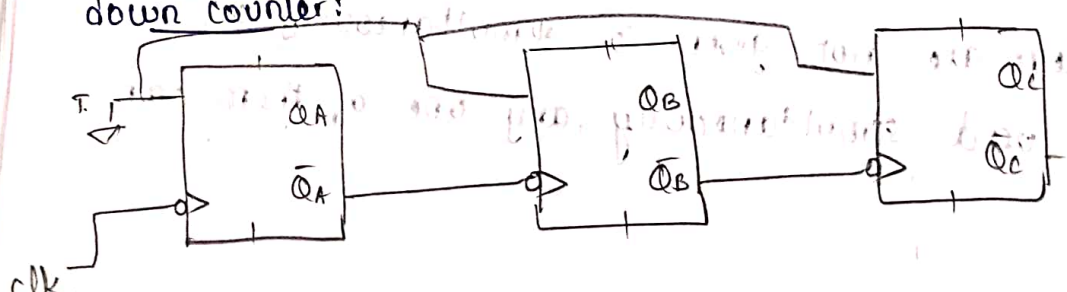
→ To count from 6 to 12 (13 = 1101) (6 = 0110) $Q_D Q_C Q_B Q_A$

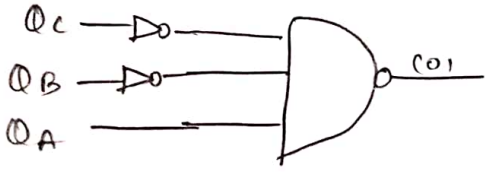


→ To count from 1 to 6 [7 = 111] [1 = 001]



down counter:





Q_C	Q_B	Q_A	
1	1	1	
1	1	0	2
1	0	1	
1	0	0	
0	1	1	
0	1	0	6
0	0	1	
0	0	0	

23/2/21

Synchronous Counter:

PS [Q(t)]			NS. [Q(t+1)]					
Q_C	Q_B	Q_A	Q_C	Q_B	Q_A	T_C	T_B	T_A
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

$T_A = 1$

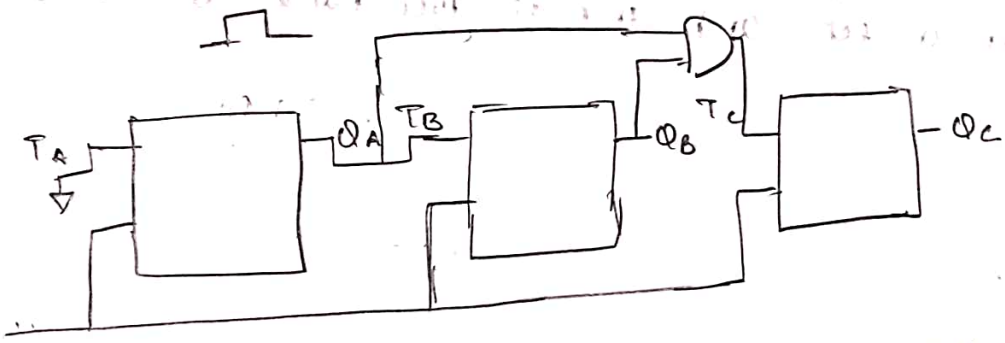
$T_B = Q_A$

$T_C = Q_A \cdot Q_B$

$T_C = \bar{Q}_C Q_B Q_A + Q_C Q_B Q_A$

$T_C = Q_A Q_B (\bar{Q}_C + Q_C)$

$T_C = Q_A Q_B$



Upcounter from 0 to 7

→ The circuit remains same for -ve or +ve edge flipflop.

Down counter:

(PS)			(NS)					
Q_C	Q_B	Q_A	Q_C	Q_B	Q_A	T_C	T_B	T_A
1	1	1	1	1	0	0	0	1
1	1	0	1	0	1	0	1	1
1	0	1	1	0	0	0	0	1
1	0	0	0	1	1	1	1	1
0	1	1	0	1	0	0	0	1
0	1	0	0	0	1	0	1	1
0	0	1	0	0	0	0	0	1
0	0	0	1	1	1	1	1	1

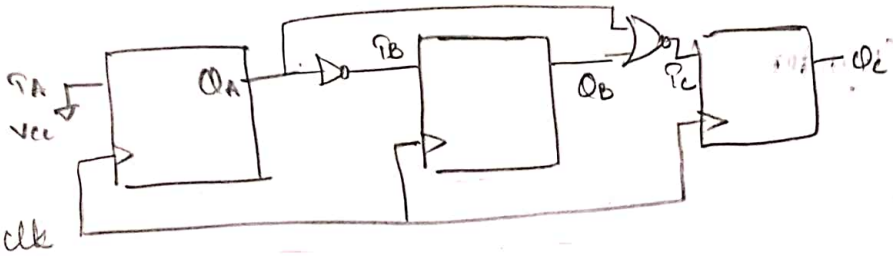
$T_A = 1$

$T_B = \bar{Q}_A$

$T_C = Q_C \bar{Q}_A \bar{Q}_B + \bar{Q}_C Q_B \bar{Q}_A$

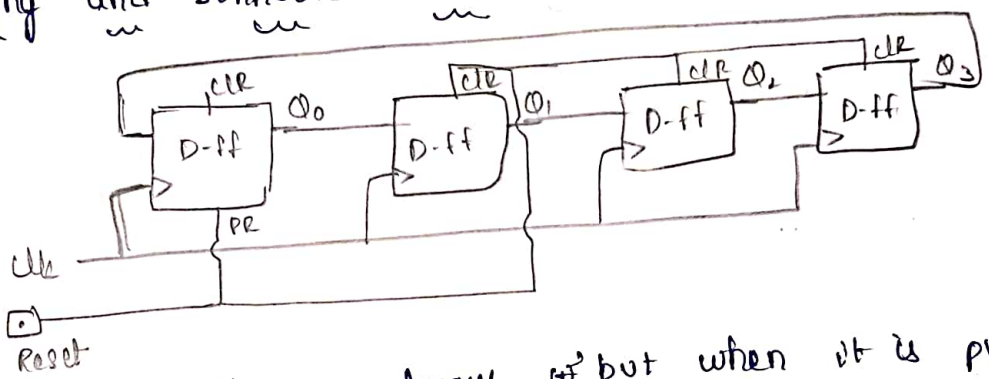
$= \bar{Q}_A \bar{Q}_B$

$= \overline{(Q_A + Q_B)}$

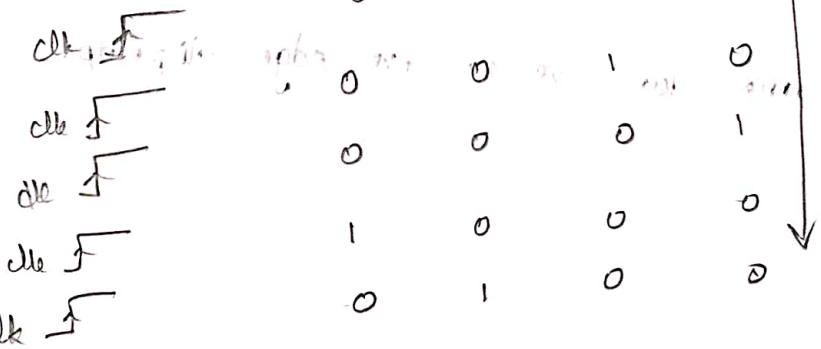


Ring and Johnson Counter:

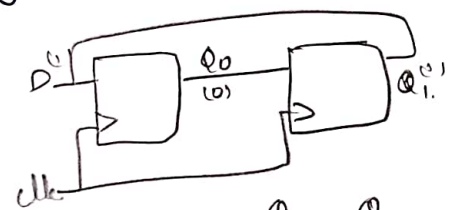
ring counter.



Reset: It will be 1 always but when it is pushed once it becomes '0' for a sec and then it goes back to normal



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



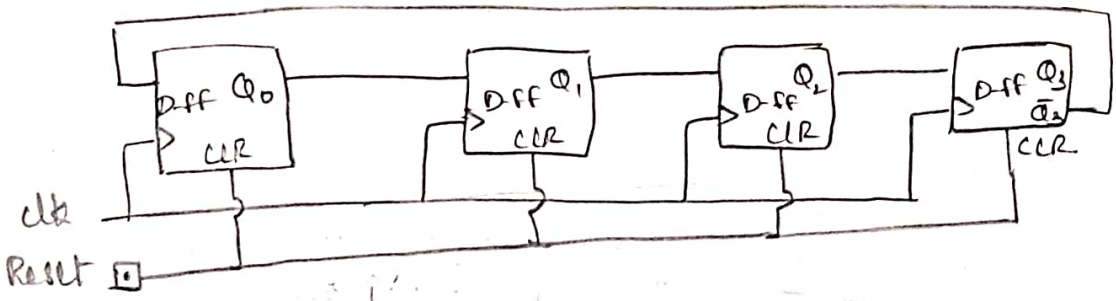
Q_0	Q_1
0	1
1	0
0	1
1	0

→ for 5 flip flops:

Q_0	Q_1	Q_2	Q_3	Q_4
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1
1	0	0	0	0
0	1	0	0	0

26/2/21

Johnson counter:



Reset	clk	Q_0	Q_1	Q_2	Q_3
	x	0	0	0	0
		1	0	0	0
		1	1	0	0
		1	1	1	0
		1	1	1	1
		0	1	1	1
		0	0	1	1
		0	0	0	1
		0	0	0	0

Synchronous Counters:

→ Divided into two types:

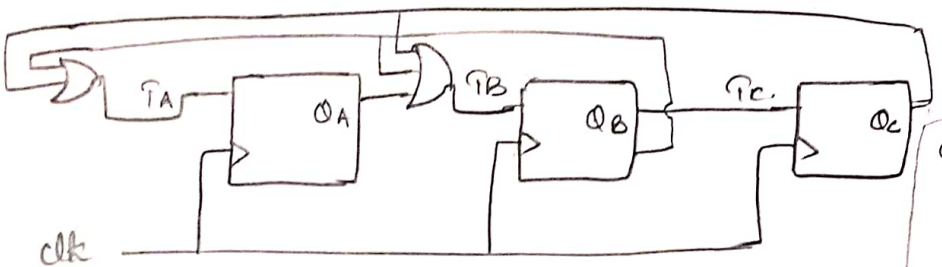
(1) Sequential (0-3, 0-7, ...)

(2) Non-sequential (skipping some no's in b/w).

eg: 0-3-4-5-7

0-1-2-4-7

(PS)			(NS)					
Q_C	Q_B	Q_A	Q_C	Q_B	Q_A	T_C	T_B	T_A
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	1	0	0	1	1	0
0	1	1	x	x	x	x	x	x
1	0	0	1	1	1	0	1	1
1	0	1	x	x	x	x	x	x
1	1	0	x	x	x	x	x	x
1	1	1	0	0	0	1	1	1



T_B:

Q_C \ $Q_B Q_A$	00	01	11	10
0	0	1	X	1
1	1	X	1	X

$T_B = Q_B + Q_C + Q_A$

T_C:

Q_C \ $Q_B Q_A$	00	01	11	10
0	0	0	X	1
1	0	X	1	X

$T_C = Q_B$

T_A:

Q_C \ $Q_B Q_A$	00	01	11	10
0	1	1	X	0
1	1	X	1	X

$T_A = \bar{Q}_B + Q_C$