16/10/20

Eg: Find no of 3×8 decoders req to design 10×1024 decoder.

Sol:

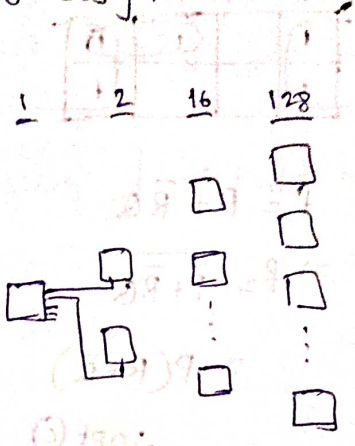
$$\frac{1024}{8} = 128$$

$$\frac{128}{8} = 16$$

$$\frac{16}{8} = 2$$

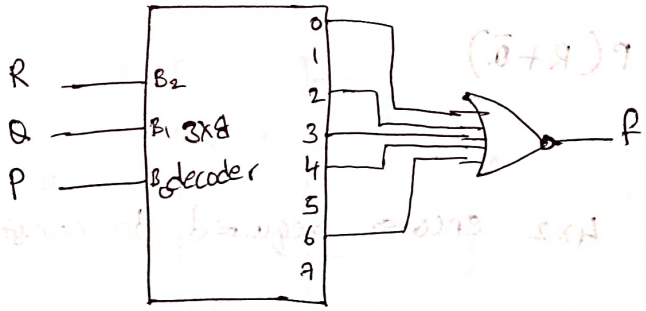
$$\frac{2}{8} = \lceil 0.25 \rceil = 1$$

$$\therefore 128 + 16 + 2 + 1 = 147$$



→ No of smaller decoders req to build a larger decoder is always odd.

(828) The minimal POS form of the o/p function

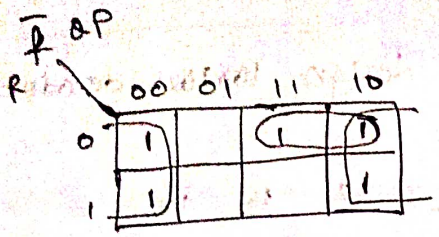


Designing logical function using decoder

a) $P(Q+\bar{R})$ b) $R(\bar{Q}+P)$ c) $P(\bar{Q}+R)$ d) $R(Q+\bar{P})$

R	Q	P	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	1	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	0	0	1	0

$$f = \overline{Y_0 + Y_2 + Y_3 + Y_4 + Y_6}$$



$$\bar{f} = \bar{P} + \bar{R}Q$$

$$\Rightarrow f = \overline{\bar{P} + \bar{R}Q}$$

$$= P(R + \bar{Q})$$

$\therefore$ opt ©

Method 2:

$$f = \overline{Y_0 + Y_2 + Y_3 + Y_4 + Y_6}$$

$$= \overline{\Sigma m(0,2,3,4,6)}$$

$$= \Pi M(0,2,3,4,6)$$

$$f = P(R + \bar{Q})$$

** *Q) Find min no of 4x2 encoder required to construct 64x6 encoders. **

Sol:

$$\frac{64}{4} = 16$$

$$\left(\frac{\text{req i/p}}{\text{given i/p}}\right)$$

$$\frac{16}{4} = 4$$

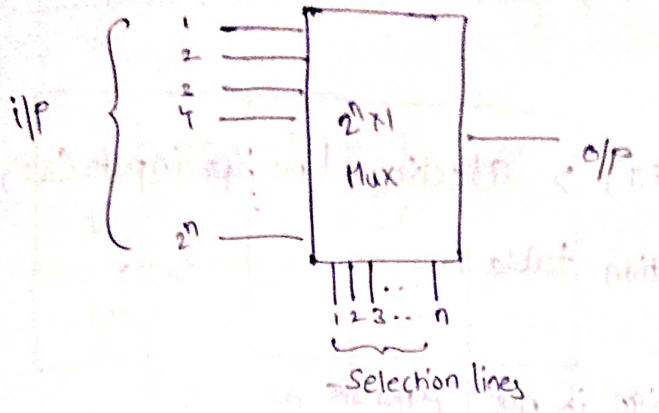
$$\frac{4}{4} = 1$$

$\therefore 16 + 4 + 1 = 21$

Multiplexer & Demultiplexer:

Multiplexer (Mux):

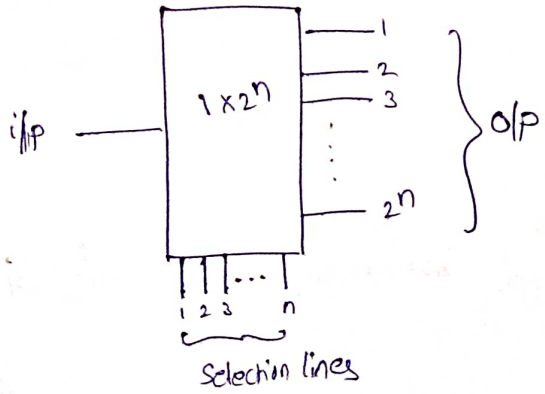
Multiplexer is a CLC which consists of 2^n i/p and single o/p and 'n' selection line i/p



Multiplexing: sending many signals through single channel

Demultiplexer:

It is a CLC which consists of single i/p & 2^n o/p with 'n' selection line inputs.



Mux: Parallel i/p to serial o/p ; P

DeMux : serial i/p to parallel o/p

Mux:

→ used for converting parallel data to serial data.

→ Data selector.

Demux:

→ Serial to Parallel data convertor.

→ Data distributor.

17/10/20

Multiplexer Design:

Model 1 :

If functional inputs (FI) = selection line inputs (SI) then design directly.

Model 2 :

If functional inputs (FI) > selection line inputs (SI) then design using implementation table.

→ If FI < SI then design is not possible

Q) Design $f(A,B,C) = \Sigma m[2,4,6]$ using

(i) 8×1 mux

(ii) 4×1 mux

Sol:

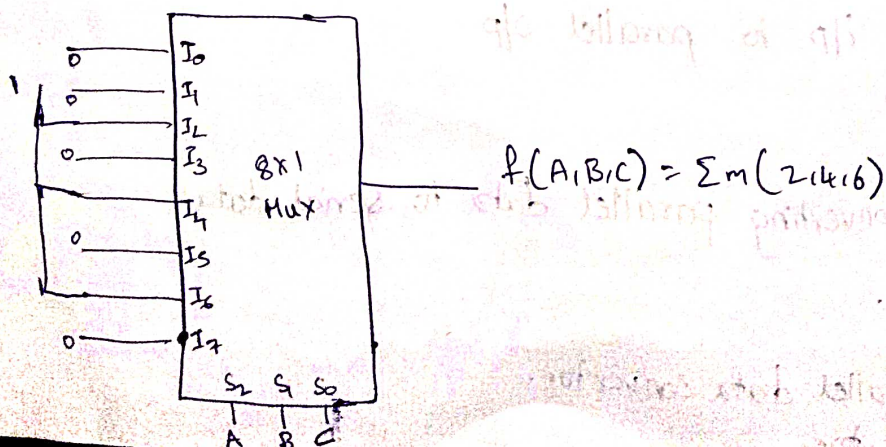
(i) Using 8×1 mux :

no of FI = 3

no of SI = 3

Here FI = SI.

So we design directly



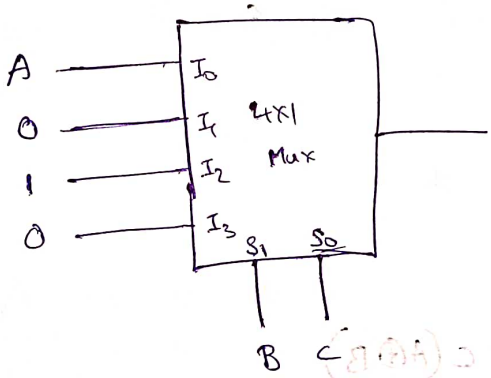
4x1 Mux :

Here FI = 3
SI = 2
FI > SI

So we design this using implementation table.

Implementation table (by removing 'A')

	I_0 $B\bar{C}$	I_1 $\bar{B}C$	I_2 $B\bar{C}$	I_3 BC
$\bar{A}$ (0)	0	1	2	3
A (1)	4	5	6	7
	A	0	1	0



Verification

$f = \bar{B}\bar{C}I_0 + \bar{B}CI_1 + B\bar{C}I_2 + BCI_3$

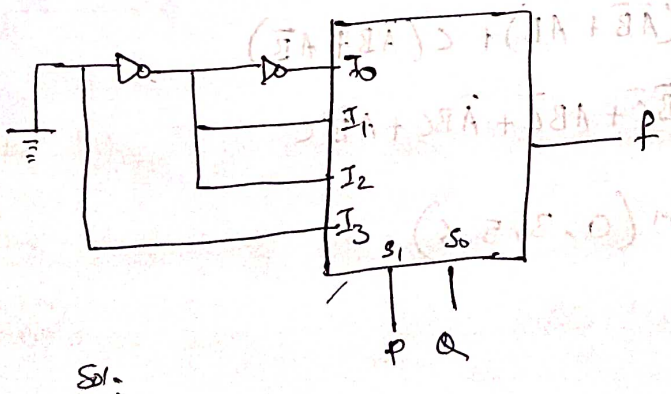
$= A\bar{B}\bar{C} + 0 + B\bar{C}$

$= A\bar{B}\bar{C} + AB\bar{C} + \bar{A}B\bar{C}$

$= \Sigma m(2,4,6)$

* *

(Q39) The logic function implemented by the circuit shown below is

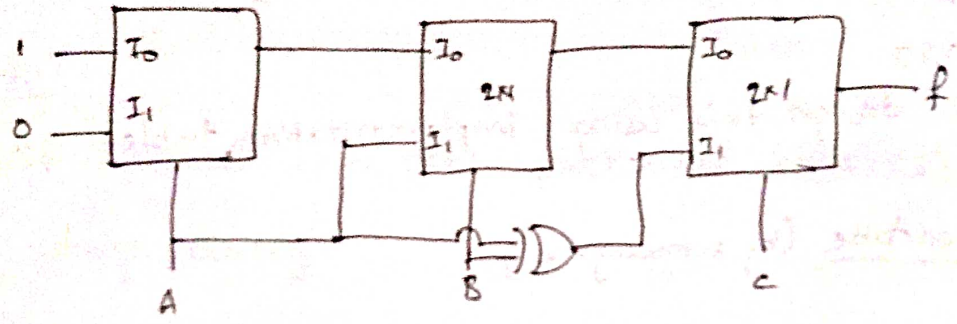


Sol:

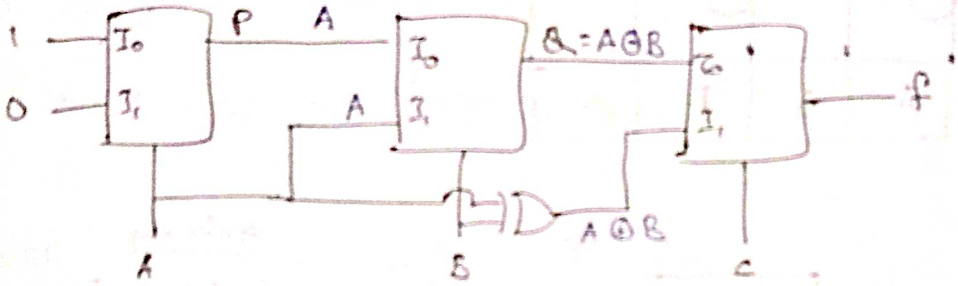
(A) $I_0 = 0$ $I_1 = 1$ $I_2 = 1$ $I_3 = 0$

$\Rightarrow f = \Sigma m(1,2) = P \oplus Q$

(Q40) In the following digital circuit, find f.



Sol:

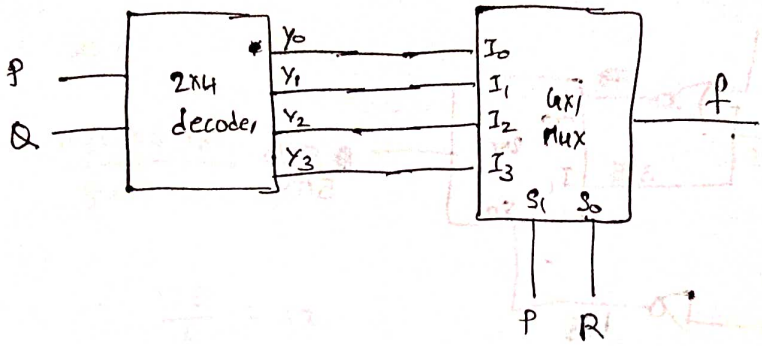


$$B = \bar{B}\bar{A} + BA$$
$$= AB + \bar{A}\bar{B}$$
$$= A \odot B$$

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

$$= \bar{C}(\bar{A}\bar{B} + AB) + C(\bar{A}B + A\bar{B})$$
$$= \bar{A}\bar{B}\bar{C} + AB\bar{C} + \bar{A}BC + A\bar{B}C$$
$$= \Sigma m(0, 3, 5, 6)$$

(Q41) The minterm $f(P, Q, R)$ are



Sol:

$Y_0 = \bar{P}\bar{Q}$; $Y_1 = \bar{P}Q$; $Y_2 = P\bar{Q}$; $Y_3 = PQ$

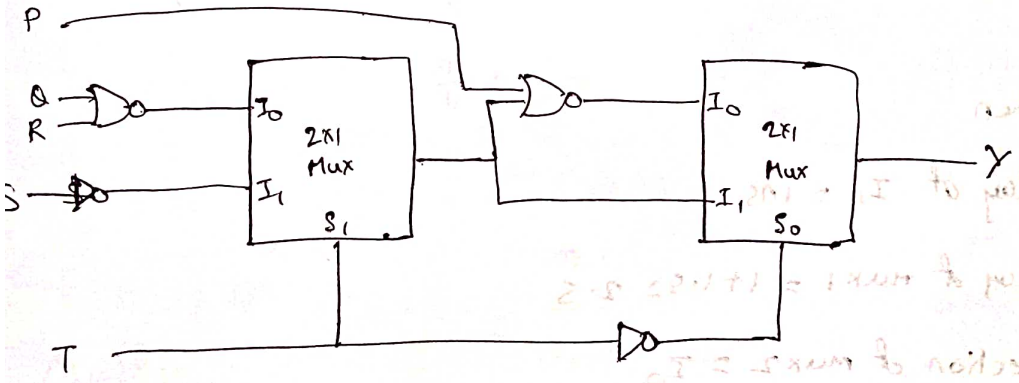
$\Rightarrow f = \bar{P}\bar{R}I_0 + \bar{P}RI_1 + P\bar{R}I_2 + PRI_3$

$= \bar{P}\bar{R}\bar{P}\bar{Q} + \bar{P}R\bar{P}Q + P\bar{R}P\bar{Q} + PRPQ$

$= \bar{P}\bar{Q}\bar{R} + \bar{P}QR + P\bar{Q}\bar{R} + PQR$

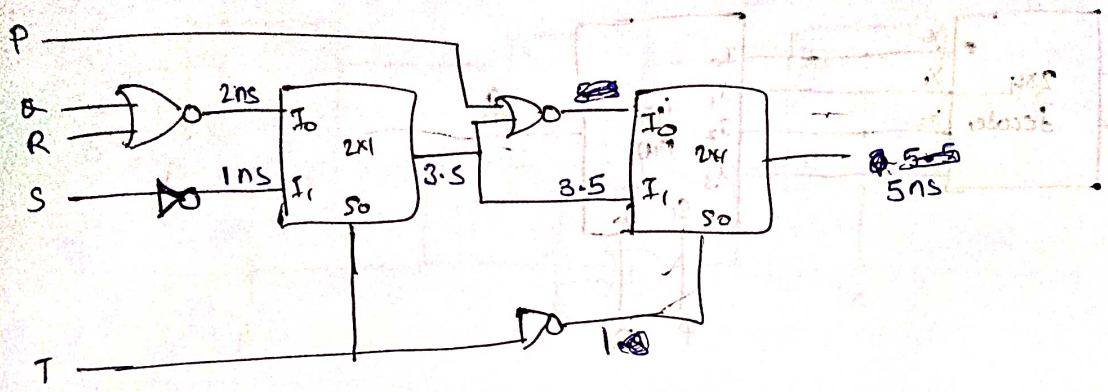
$= \sum m(0, 3, 4, 7)$

(Q42) The circuit shown below



The delays of NOR gate, Mux & inverter are 2ns, 1.5ns & 1ns respectively. If all the inputs are applied, then the maximum ~~prog~~ propagation delay (in ns) is _____

Sol:



~~The fig shows max possible delay at each step when T=1.~~
~~i.e. 5 ns~~

IF T=0 then,

delay for I_0 = 2ns

delay for o/p of 1st mux = 2 + 1.5 = 3.5

Selection of Mux 2 will be 0

~~∴ delay of Nor = 2ns + 2.5 = 4.5~~

~~∴ delay of 2nd mux =~~

delay of mux 2 will be 3.5 + 1.5 = 5.

(∵ Selection of mux 2 is 1)

If T=1 then

delay of I_1 = 1ns

delay of mux 1 = 1 + 1.5 = 2.5

selection of mux 2 = I_0

∴ delay of nor = 2.5 + 2

mux 1 ↓ NOR ↓

= 4.5

delay of mux 2 = 4.5 + 1.5 = 6

∴ Max delay = 6

18/10/20

(843) To design 128×1 mux min no of 4×1 mux req is?

Sol:

$$\frac{\text{required i/p}}{\text{given i/p}}$$

$$\therefore \frac{128}{4} = 32$$

$$\frac{32}{4} = 8$$

$$\frac{8}{4} = 2$$

$$\frac{2}{4} = 0.5$$

$$\therefore 1+2+8+32 = 43$$

(844) To design 1×64 Demux, min no of 1×4 demux req is

Sol:

$$\frac{\text{req o/p}}{\text{given o/p}}$$

$$= \frac{64}{4} = 16$$

$$\frac{16}{4} = 4$$

$$\frac{4}{4} = 1$$

$$\therefore 21$$

Overflow:

In the signed 2's complement arithematic, if the result exceeds the range

$-(2^{n-1})$ to $(2^{n-1}-1)$ then overflow occurs

18/10/20

Q43 To design 128×1 mux min no of 4×1 mux req is?

Sol:

$$\frac{\text{required i/p}}{\text{given i/p}}$$

$\therefore \frac{128}{4} = 32$

$\frac{32}{4} = 8$

$\frac{8}{4} = 2$

$\frac{2}{4} = 0.5$

$\therefore 1+2+8+32 = 43$

Q44 To design 1×64 Demux, min no of 1×4 demux req is

Sol:

$$\frac{\text{req o/p}}{\text{given o/p}}$$

$= \frac{64}{4} = 16$

$\frac{16}{4} = 4$

$\frac{4}{4} = 1$

$\therefore 21$

Overflow:

In the signed 2's complement arithematic, if the result exceeds the range

$-(2^{n-1})$ to $(2^{n-1}-1)$ then overflow occurs

Case (i):

Addition of a +ve and a -ve number:

Eg: Add +6 & -2

$+6 \rightarrow 0110$

$-2 \rightarrow 1110$

$10100 = (+4)$

∴ no overflow

→ Thus addition of a positive and a negative never leads to overflow.

Case II: Addition of two +ve numbers:

Add +4, +5

$+4 \rightarrow 0100$

$+5 \rightarrow 0101$

$+9 \quad 1001$

Here we have 4 bits

∴ range -2^{n-1} to $2^{n-1}-1$

-8 to 7

∴ overflow occured

Eg: Add +4, +2

$+4 \rightarrow 0100$

$+2 \rightarrow 0010$

$+6 \quad 0110$

+6 is within the range

∴ no overflow.

∴ Addition of two positive numbers may cause overflow.

Case III: Addition of both -ve numbers.

Eg: add -7 & -5

$-7 \rightarrow$ 1001

$-5 \rightarrow$ 1011

-12 10100

-12 doesn't lies in the range

∴ overflow occurs.

Eg: add -5 & -2

$-5 \rightarrow$ 1011

$-2 \rightarrow$ 1110

-7 11001

∴ -7 lies within the range

∴ no overflow.

Thus addition of two -ve numbers may lead to overflow.

Observation:

From the above examples, we had overflow whenever C_3 & C_4 are ~~equal~~ different.

∴ Condition to detect overflow is $C_3 \oplus C_4 = 1$

i.e.

∴ for n-bit numbers, condition for overflow is

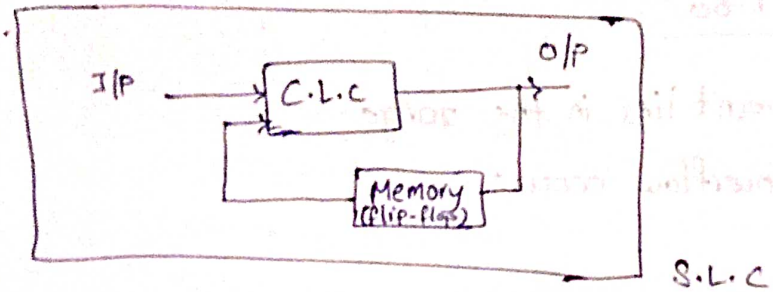
$$C_{in} \oplus C_{out} \text{ (or) } C_{n-1} \oplus C_n$$

→ one more method to detect overflow is by observing sign bits

i.e., $\boxed{A_{n-1} B_{n-1} \overline{S_{n-1}} + \overline{A_{n-1}}\,\overline{B_{n-1}} S_{n-1} = 1}$

Sequential Logic Circuits:

Defn: The circuit in which present o/p depends on both present i/p as well as previous data.



S.L.C = C.L.C + Memory

Flip-Flop:

It is a memory element

Types of flip-flops:

(i) SR FF
(ii) JK FF
} Two i/p FF

(iii) T FF
(iv) D FF
} one i/p FF

} Two o/ps $[Q_{n+1}, \overline{Q}_{n+1}]$

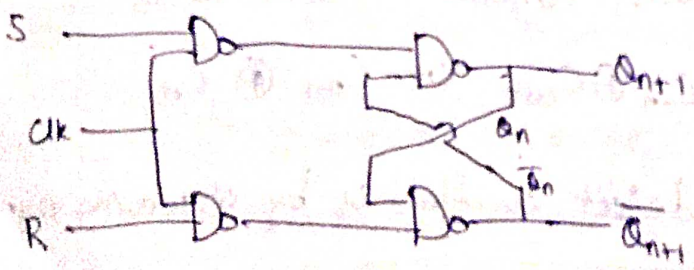
∴ FF is also called as bi-stable device.

Condition in FF:

always value of Q_{n+1}, $\overline{Q}_{n+1}$ are complement to each other.

If ~~value~~ $Q_{n+1} == \overline{Q}_{n+1}$ then it is invalid o/p.

SR FlipFlop:



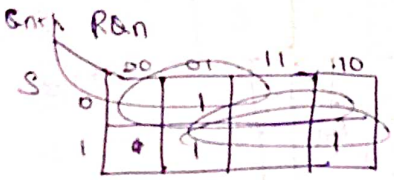
S - Set

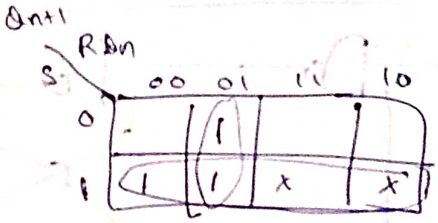
R - Reset

Characteristic table (Truth table)

Decimal	S	R	(R.S)' Q_n	(N.S) Q_{n+1}	State name
0	0	0	0	0	No change
1	0	0	1	1	
2	0	1	0	0	Reset
3	0	1	1	0	
4	1	0	0	1	Set
5	1	0	1	1	
6	1	1	0	x	Invalid
7	1	1	1	x	

Expression for Q_{n+1}





$\therefore Q_{n+1} = S + \bar{R} Q_n$

Excitation table (or) Transition table:

Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Note:

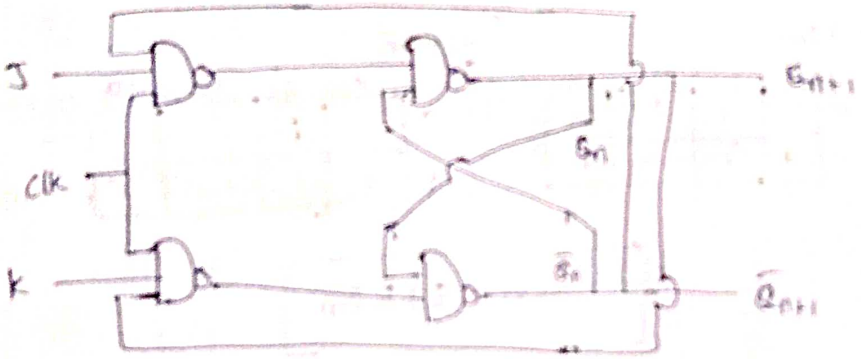
→ Disadvantage of SRFF is when S=1 & R=1 the O/P is invalid. We can overcome this problem using JKFF

Simplified truth table:

S	R	Qn+1
0	0	Qn
0	1	0
1	0	1
1	1	x

20/10/20

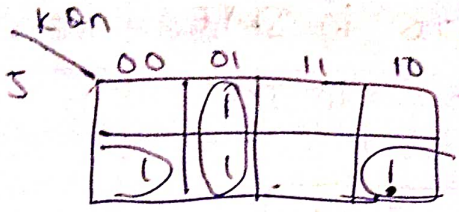
Jk FlipFlop:



Characteristic Table:

J	k	Qn	Qn+1	State Name
0	0	0	0	No change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	set
1	0	1	1	
1	1	0	1	Toggle
1	1	1	0	

$Q_{n+1}[J,k,Q_n] = \sum m(1,4,5,6)$



$$Q_{n+1} = J\bar{Q}_n + \bar{k}Q_n$$

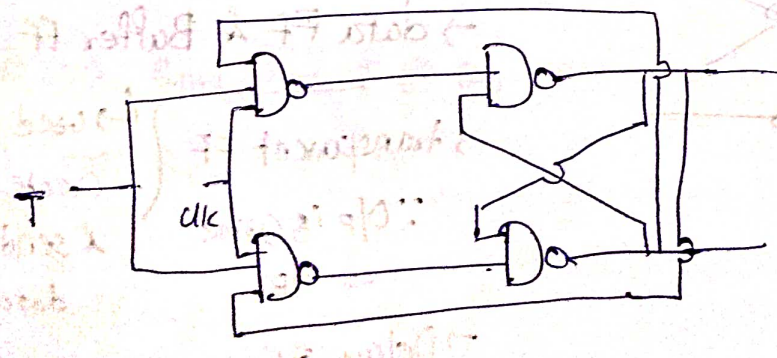
Excitation table:

Q_n	Q_{n+1}	J	k
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Simplified Truth table

S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\bar{Q}_n$

T Flip flop:



T → Toggle

Characteristic table:

$T=0 \Leftrightarrow J=0$ & $K=0$ in JK-FF

$T=1 \Leftrightarrow J=1$ & $K=1$ in JKFF

Characteristic table:

T	Q_n	Q_{n+1}	State
0	0	0	No change
0	1	1	
1	0	1	Toggle
1	1	0	

$$Q_{n+1} = T \oplus Q_n$$

Excitation table:

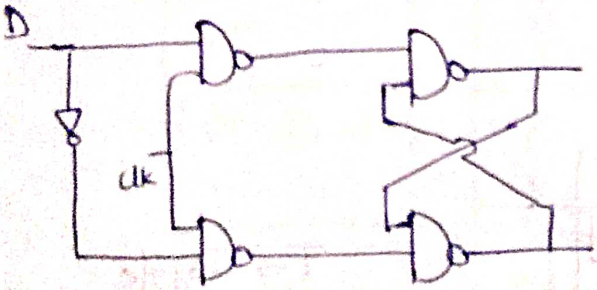
Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Simplified Truth table:

T	Q_{n+1}
0	Q_n
1	$\bar{Q}_n$

→ data/ delay

D FlipFlop:



DFF is called

→ data FF & Buffer FF

→ transparent FF (↳ used to create delay & send the data.)

∵ O/p is same as i/p.

→ Delay FF

Characteristic Table :

D	Q_n	Q_{n+1}	State
0	0	0	reset
0	1	0	
1	0	1	set
1	1	1	

$D=0 \cong (S=0 \& R=1)$
↳ reset

$D=1 \cong (S=1 \& R=0)$
↳ set

$$Q_{n+1} = D\bar{Q}_n + DQ_n = D$$

~~$\therefore Q_n = D$~~ $\quad \therefore Q_{n+1} = D$

Excitation table

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

Simplified Truth Table :

D	Q_{n+1}
0	0
1	1

Note :

SR $\Rightarrow$ $Q_{n+1} = S + \bar{R}Q_n$

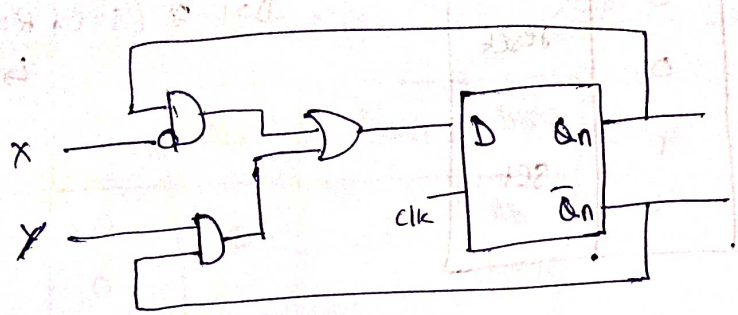
JK $\Rightarrow$ $Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$

T $\Rightarrow$ $Q_{n+1} = T \oplus Q_n$

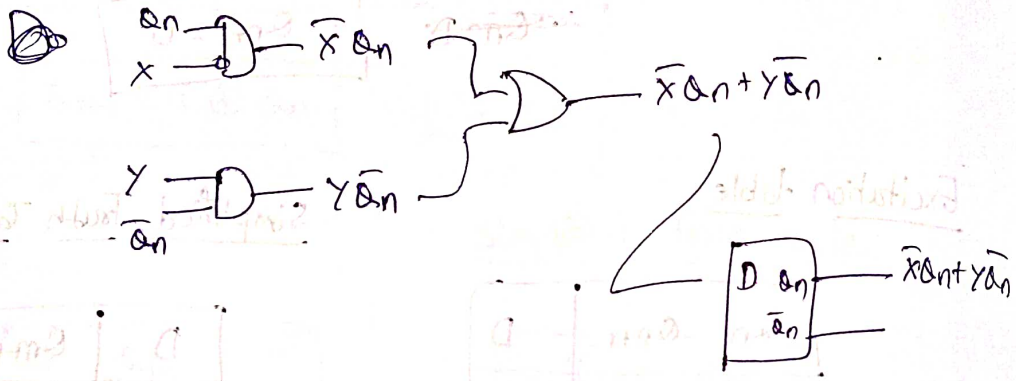
D $\Rightarrow$ $Q_{n+1} = D$

Flipflop Conversion Related Problems

Eg: Find how the below circuit acts



Sol:



i.e., $Q_{n+1} = \bar{X}Q_n + Y\bar{Q}_n$

i.e.,

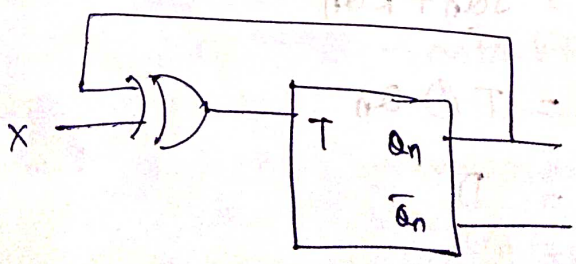
for JKFF

$$Q_{n+1} = J\bar{Q}_n + \bar{R}Q_n$$

i.e., The given ckt is JKFF

where X is K & Y is J

Eg: Find how the below ckt works



Sol:

From the figure

$$Q_{n+1} = T \oplus Q_n$$

we have $T = X \oplus Q_n$

$$\therefore Q_{n+1} = X \oplus Q_n \oplus Q_n$$

$$= X \oplus 0$$

$$= X$$

i.e., $Q_{n+1} = X$

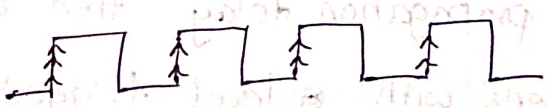
∴ The given fig corresponds to D flipflop where $X = D$.

Clock:

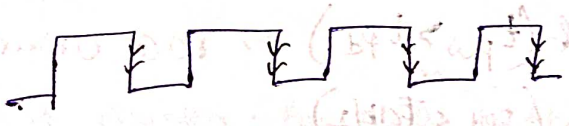
Clock is used to change the o/p of any sequential circuit at a specified point on a triggering.

Types of clocks:

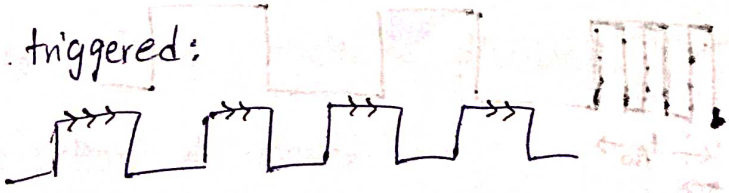
(i) Positive Edge triggered:



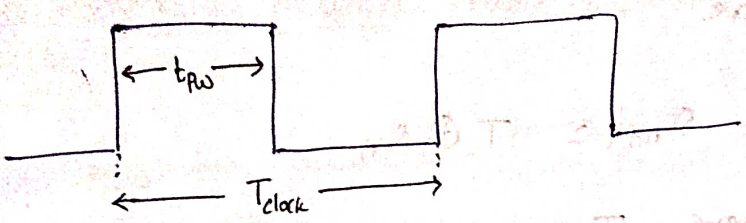
(ii) Negative Edge triggered:



(iii) Positive level triggered:



(iv) Negative level triggered:



t_{pw} → clock pulse width

T_{clock} → Clock Time period

$T_{clock} = 2t_{pw}$

Propagation delay (t_{pd}):

Time required for an input to reach (~~prog~~ propagate) the o/p is called propagation delay.

Note:

During one clock time period, it required that o/p changes only once.

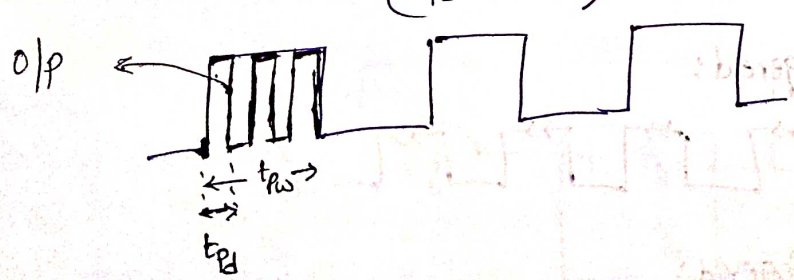
Race Around Problem:

For ~~JK~~ JKFF, if J=1, k=1 and if clock pulse width (t_{pw}) is more than propagation delay then o/p keeps on toggling. This happens with ~~@~~ level triggering. This problem is known as race around problem.

J=1, k=1 & ($t_{pw} > t_{pd}$) ⇒ race around problem.

or

($t_{pd} < T_{clk}/2$)



→ This problem occurs only in JKFF & TFF.

→ SRFF & DFF are free from this problem.

condition to avoid race around problem.

Method 1: $t_{pw} < t_{pd} < T_{clock}$

(or)

$\frac{T_{clock}}{2} < t_{pd} < T_{clock}$

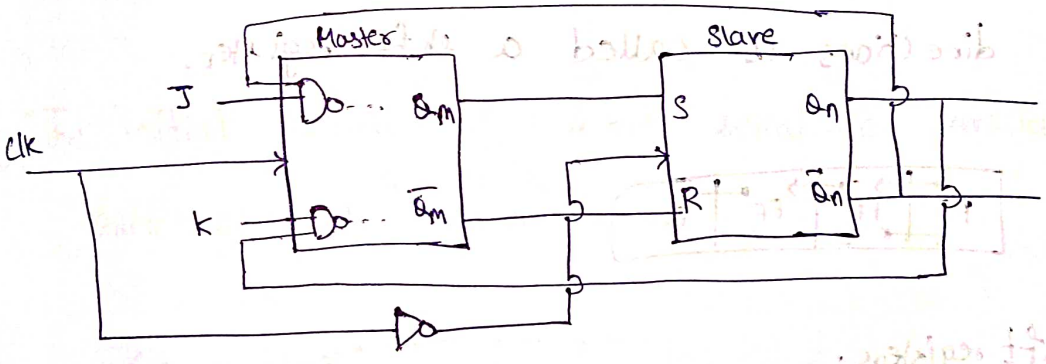
} This is not possible practically.

Method 2: Using Master-slave Flip Flop

↳ Complex Design

Method 3: Using Edge triggered clock.

Master Slave FF:



→ Master slave FF is constructed using 2 FFs.

→ Master ⟹ JKFF

Slave ⟹ JK/SR

→ when master is enabled (functional) the slave is disabled.

when slave is enabled, master is disabled.

→ Slave copies action of master.

→ Master & slave O/ps never change at same time (∵ at given time only one is functional)

→ Though we have two flipflops, master slave flipflop stores only one bit.

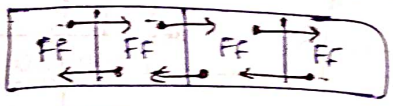
Register:

→ group of flipflips is called a register.

→ An n-bit register contains 'n' FFs and is used to store n-bit data.

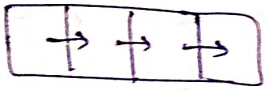
Shift register:

A register in which shifting of data is possible in one or both the directions is called a shift register.

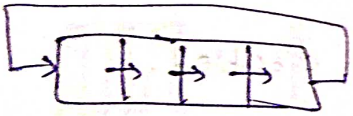


Types of shift registers:

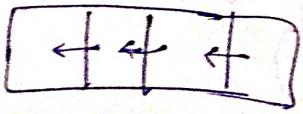
(i) Serial Right shift:



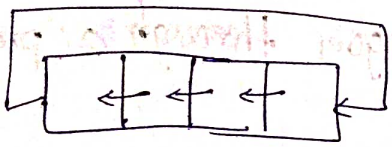
(ii) Rotate right shift:



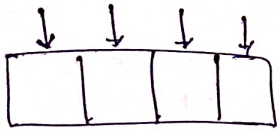
(iii) Serial left shift:



(iv) Rotate left shift.

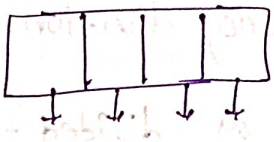


(v) Parallel shift in :



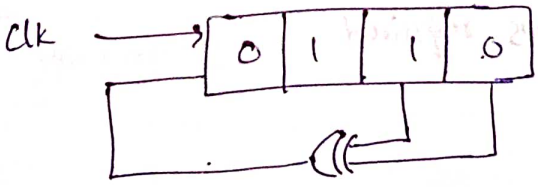
Data is entered into all FFs at the same time.

(vi) Parallel shift out :



Data is sent out from all FFs at the same time.

Eg: The initial content of 4 bit serial-in parallel out right shift register is shown below



Find contents of shift register after 3 clock pulses.

Sol :

After 1st clock pulse : 1 0 1 1

After 2nd clock pulse : 0 1 0 1

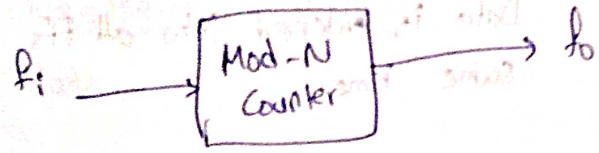
After 3rd clock pulse : 1 0 1 0

∴ 1010

Counters:

Counter is a register that goes through a predetermined sequence of states.

→ A mod-n counter has 'n' different states



$$f_o = \frac{f_i}{N}$$

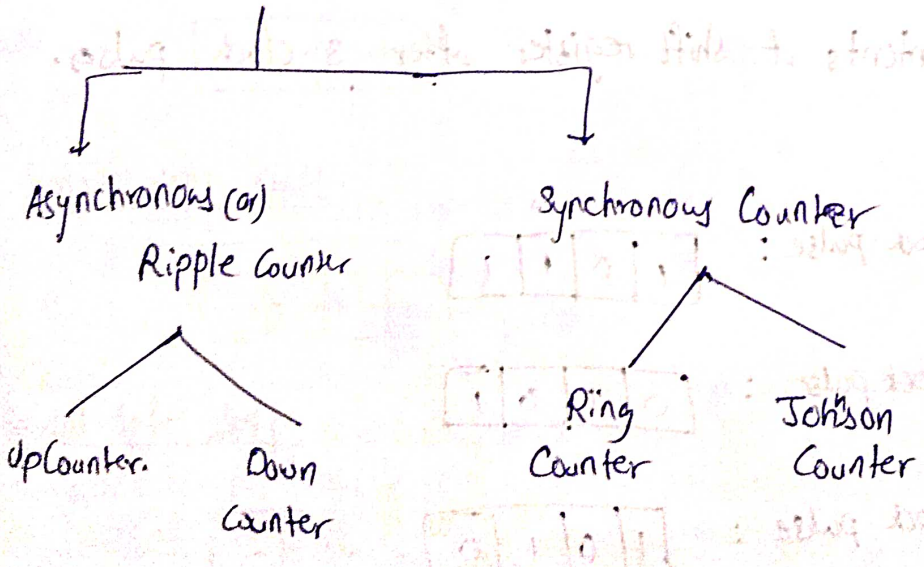
i.e., Counter is used in frequency division.

Thus Mod-N counter is also known as divided-by-N counter.

→ To design a mod-N counter we need 'n' bits register such that $N \leq 2^n$

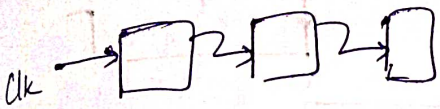
N → no of states
n → no of flip-flops required

Types of Counters



Asynchronous Counter

→ O/p of one FF is given as clock i/p for other FF

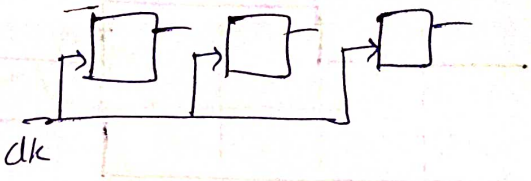


→ Here O/p of all the FFs change at different times.

→ If propagation delay of each FF is t_{pd} then propagation delay of an n-bit asynchronous counter is $n * t_{pd}$

Synchronous Counter

→ Same clock i/p is given for all FFs



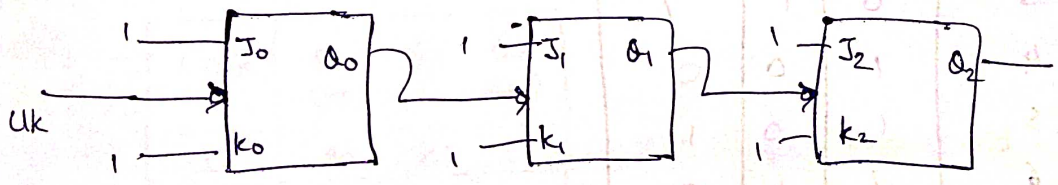
→ Here output of all the FFs changes at the same time.

→ If propagation delay of each FF is t_{pd} then for n-bit synchronous counter's time delay $\Phi = t_{pd}$

Asynchronous (or) Ripple Counter:

3-bit Up Counter:

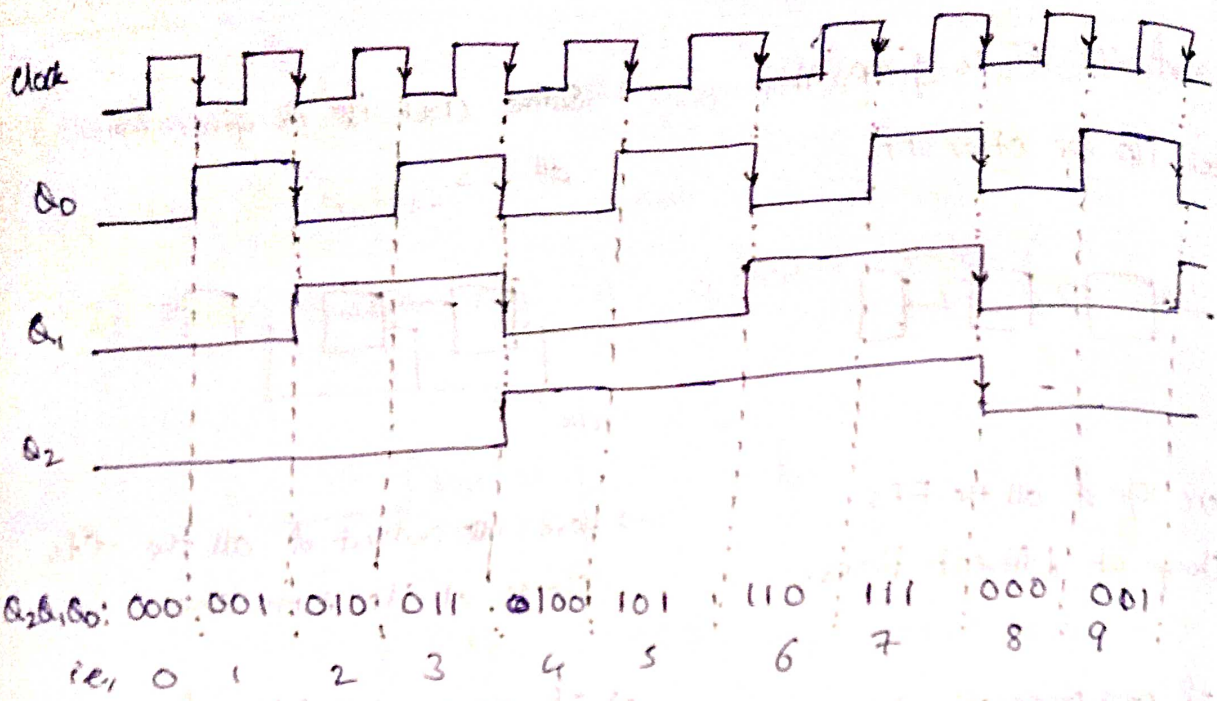
→ : Negative edge trigger.



Assume the initial configuration is

$Q_0 = Q_1 = Q_2 = 0$

Clock wave forms:



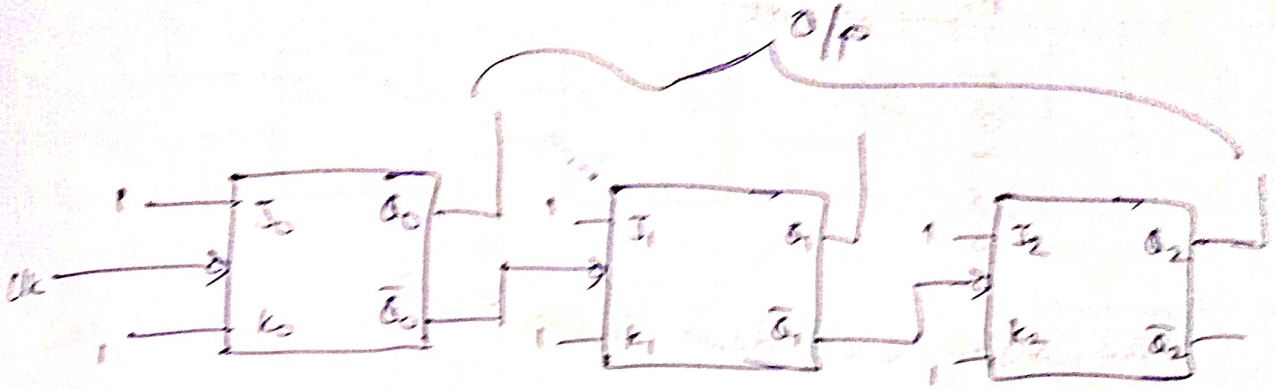
→ 3 bits asynchronous counter repeats its states after every $8 = 2^3$ clock pulses.

→ Thus an n-bit asynchronous counter repeats its state after every 2^n clock pulses.

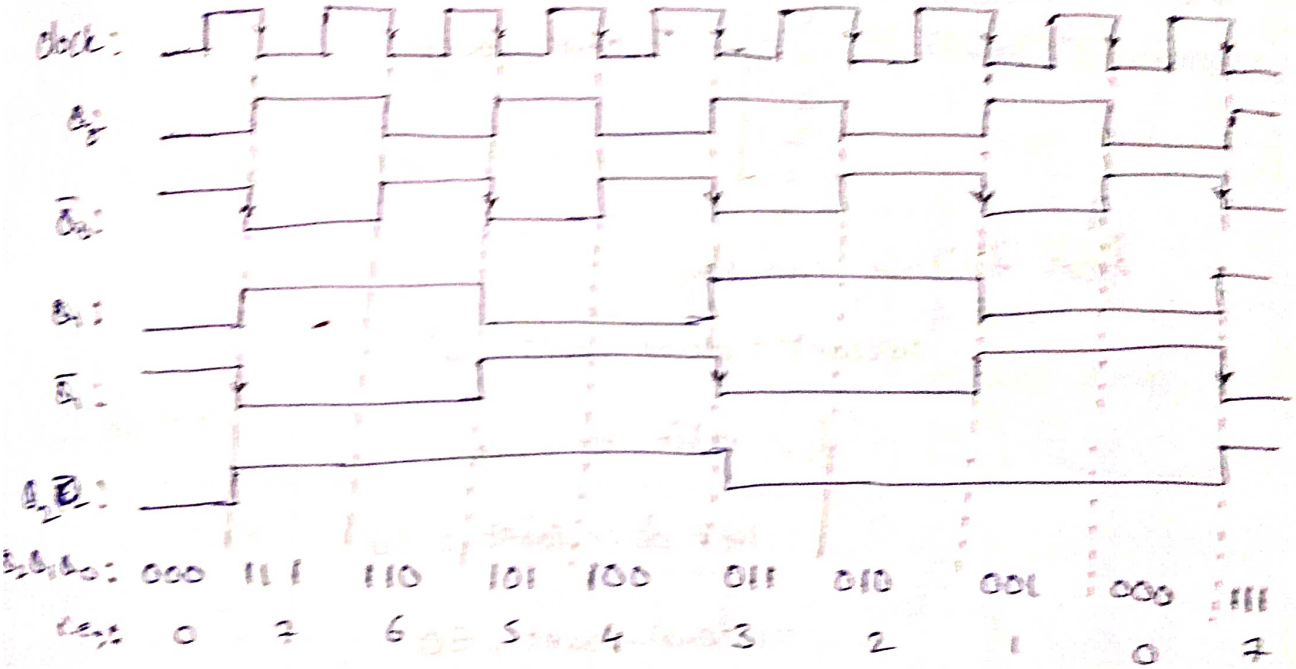
Clock	Q_2 (MSB)	Q_1	Q_0 (LSB)
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0
9	0	0	1

∴ This is called mod 8 counter.

Down Counter:



Assume initially $Q_0 = Q_1 = Q_2 = 0$



Clock	Q_2 (MSB)	Q_1	Q_0 (LSB)
0	0	0	0
1	1	1	1
2	1	1	0
3	1	0	1
4	1	0	0
5	0	1	1
6	0	1	0
7	0	0	1
8	0	0	0
9	1	1	1

∴ This is known as Mod-8 down counter.

(Q45) In a 6 bit ripple counter, the value of upcounting is 14. Then the value of down counting is ____

Sol:

Consider a 3 bit upcounter with upcounting:

0
1 } 1 upcount
2
3
4
5
6
7

Now consider 3 bit down counter

0
7
6
5
4 } 7 down count
3
2
1
0

from this we can say

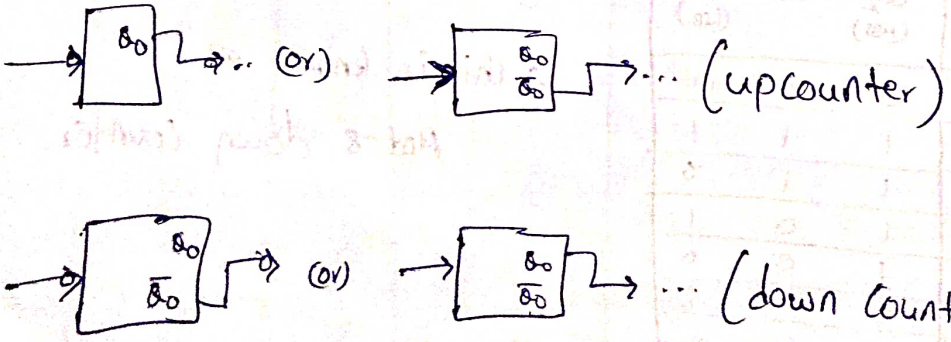
upcount + down count = 2^n

for n bit ⊗ Counter

$\therefore 14 + \text{down\_count} = 64$

$\Rightarrow \text{down\_count} = 50$

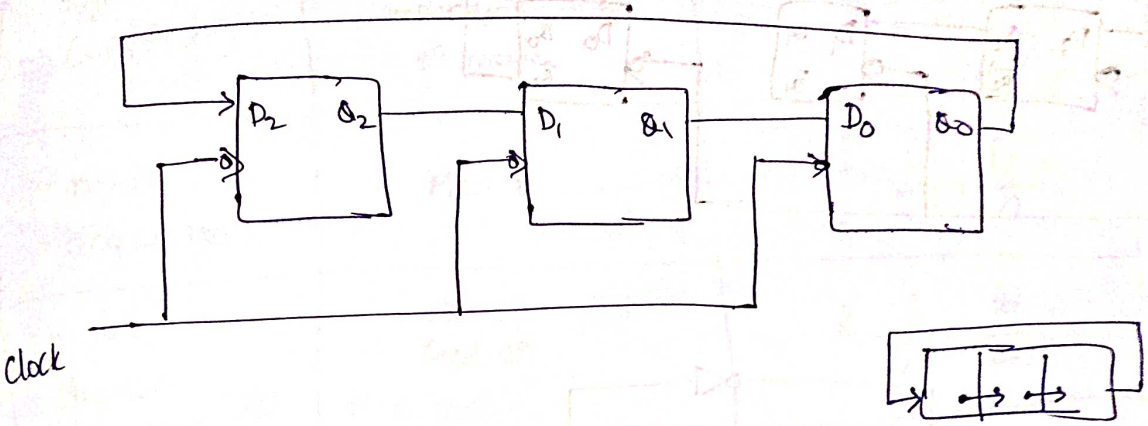
Note:



Synchronous Counters:

Ring Counter:

Consider the 3-bit ring counter as shown below.



let, initially $Q_2Q_1Q_0 = 100$ ~~(we should not take $Q_2Q_1Q_0 = 000$ as initial condition (think why))~~

from figure $D_2 = Q_0$

$D_1 = Q_2$

$D_0 = Q_1$

A mod-n ring counter requires 'n' FFs

Truth table

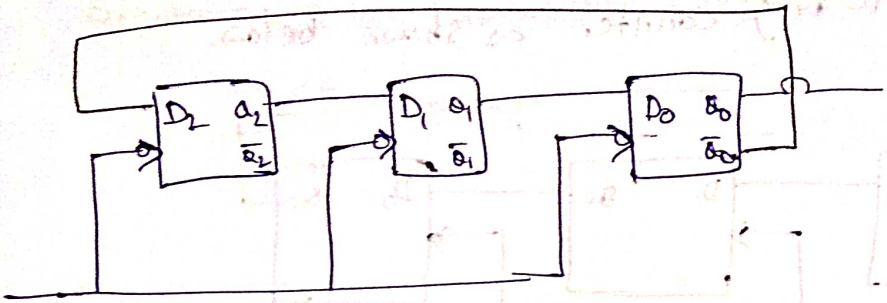
clock	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	-	-	-	1	0	0
1	0	1	0	0	1	0
2	0	0	1	0	0	1
3	1	0	0	1	0	0

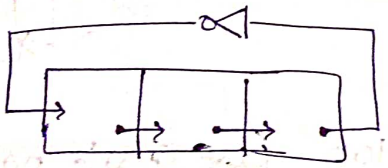
→ Thus in 3-bit ring counter, the value repeats after 3 clock cycles.

→ Thus in n-bit ring counter the value repeats after n clock pulses

Jhonson Counter / Twisted Ring Counter / Moebius Counter:

Consider below 3-bit Jhonson counter





Assume $Q_2Q_1Q_0 = 000$

Clock	Q_2	Q_1	Q_0
0	0	0	0
1	1	0	0
2	1	1	0
3	1	1	1
4	0	1	1
5	0	0	1
6	0	0	0

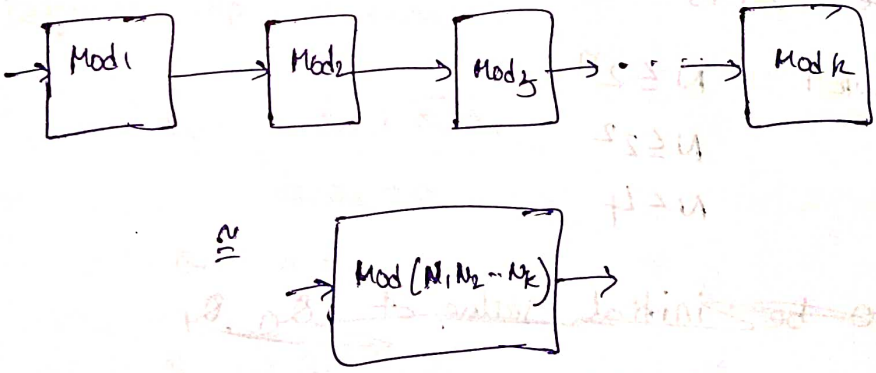
∴ 3-bit Jhonson Counter repeats same value after 6 clock pulses

⇒ Thus, an n bit Jhonson Counter repeats value after 2n clock pulses.

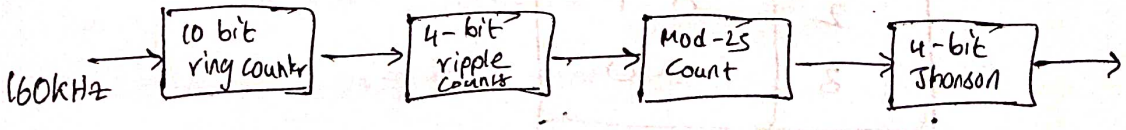
Summary :

Counter Name	Mod	o/p frequency $f_0 = f_i/N$
n-bit asynchronous (up/down)	Mod 2^n i.e., repeats after 2^n values	$f_0 = \frac{f_i}{N} = \frac{f_i}{2^n}$
n-bit ring counter	Mod n	$f_0 = \frac{f_i}{N} = \frac{f_i}{n}$
n-bit Johnson Counter	Mod 2n	$f_0 = \frac{f_i}{N} = \frac{f_i}{2n}$

→ If k counters Mode N_1, Mod N_2, ... Mod N_k are cascaded, then the resultant counter is Mod $(N_1 N_2 \cdots N_k)$



Q46 The frequency of the pulse at z shown in the fig is



Sol :

resultant counter is

$$\text{Mod}\left((10)(2^4)(25)(8)\right) = \text{Mod}\left(32000\right)$$

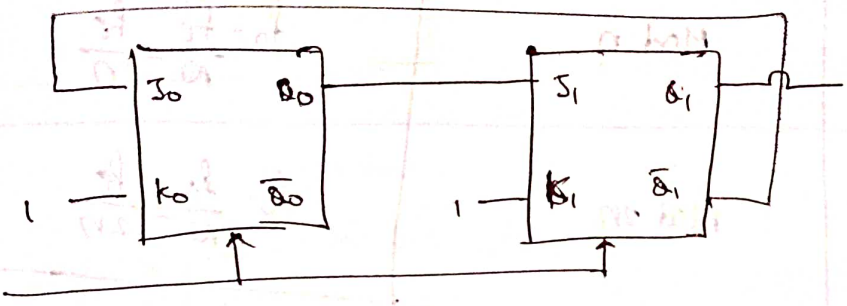
Counter.

$$\therefore f_o = \frac{f_i}{N} = \frac{160\,kHz}{\cancel{16000}\ 32000} = \frac{160\times10^3}{\cancel{16000}\ 32000}\ Hz = \cancel{160\ Hz}$$

$$= \frac{160}{32} = 5\ Hz$$

**
(Q47)
***

Fig shows Mod k counter



the value of 'k' is equal to ——

(Assume initially $Q_0 = Q_1 = 0$)

a) 2 b) 3 c) 4 d) 5

Sol:

we have 2 FFs

WKT $N \le 2^n$

$N \le 2^2$

$N \le 4$

~~Let~~ ~~00 be~~ initial value of $Q_0\ Q_1$

Clock	Q_0	Q_1
0	0	0
1	1	0
2	0	1
3	0	0

$\therefore$ Mod 3 Counter.

* Method II : Standard method to solve this type of Questions

Solution

i) Form relations b/w variable

$J_0 = \bar{Q}_1$ $\quad J_1 = Q_0$

$K_0 = 1$ $\quad K_1 = 1$

Clock	J_0	K_0	J_1	K_1	Q_0	Q_1
0	-	-	-	-	0	0
1	1	1	0	1	1	0
2	1	1	1	1	0	1
3	0	1	0	1	0	0

∴ Mod 3 Counter

Method III :

Determine O/p expressions

$$Q_0 = J_0\bar{Q}_0 + \bar{K}_0 Q_0$$

$$= \bar{Q}_1\bar{Q}_0 + 0$$

$$Q_0 = \bar{Q}_1\bar{Q}_0$$

$$Q_1 = J_1\bar{Q}_1 + \bar{K}_1 Q_1$$

$$= Q_0\bar{Q}_1 + 0$$

$$= Q_0\bar{Q}_1$$

∴ $Q_0 = \bar{Q}_1\bar{Q}_0$ $\quad Q_1 = Q_0\bar{Q}_1$

using these formulas

Q_0	Q_1
0	0
1	0
0	1
0	0

∴ Mod 3 Counter.

* *

→ Counting sequence of the counter is: $Q_1 Q_0$

Note that Q_0 is LSB and Q_1 is MSB

Q_1	Q_0
0	0
0	1
1	0
0	0

∴ Counting sequence : 0, 1, 2, 0.

→ For the above counter find the o/p after 334th clock pulse.

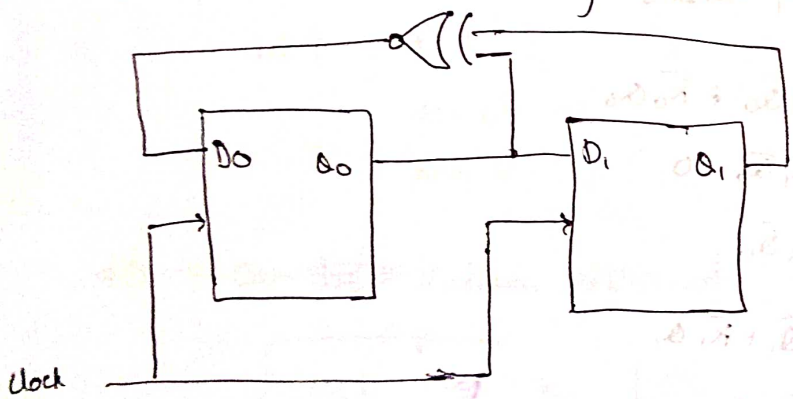
Since it is mod 3 counter,

∴ o/p at 334th clock pulse = o/p at 334 mod 3 clock pulse

= o/p at 1st clock pulse

i.e., $Q_0 = 1$; $Q_1 = 0$;

(Qus) The modulus of the following counter is



Assume initial values are $Q_0 = Q_1 = 0$

Sol:

$Q_0 = D_0 = Q_0 \odot Q_1$

$Q_1 = D_1 = Q_0$

Q_0	Q_1
0	0
1	0
0	1
0	0

∴ Mod 3 counter.

(Q49) The minimum no of FFs req to ~~~~ construct counter with count sequence (0,0,1,1,2,2,3,3,00) is ____

Sol:

Let Q_0 be a dummy bit

we require this additional FF to differentiate

Q_2	Q_1	Q_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1
0	0	0
0	0	1
⋮		

Here, however we take o/p only from Q_2 & Q_1

∴ Min no of FFs req = 3

(Q50) Find min no of FFs req to construct a counter with count sequence (0,1,0,2,0,3,...) is ____

Sol:

Here '0' is repeated thrice.

So we need to differentiate 3 0's.

So we need 2 dummy FFs.

and we have 4 uniques ~~state~~ state (i.e., 0,1,2,&3)

∴ we need 2 FFs.

∴ 4 FFs

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

Dummy FFs