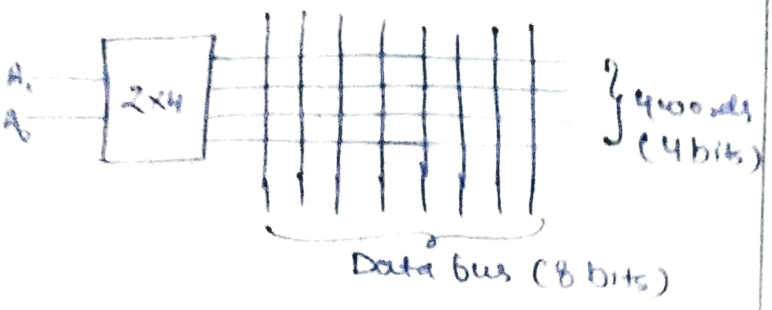
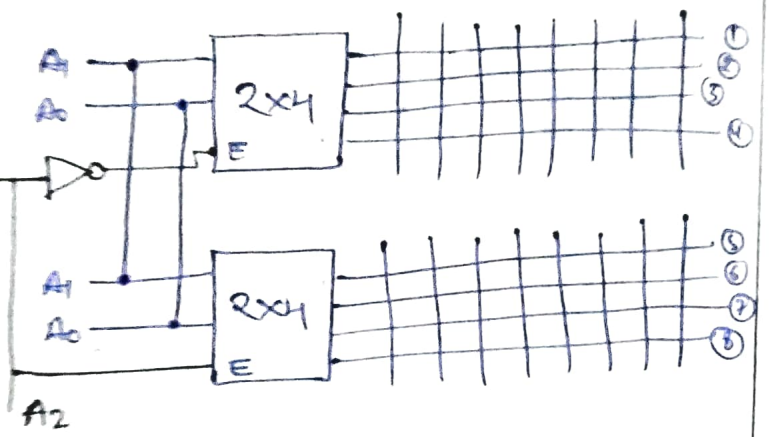
Address Expansion of ROM



M/m size = 8×4 = 25 bits

Use above ROM to construct a ROM of size (8×8) = 2⁶ bits

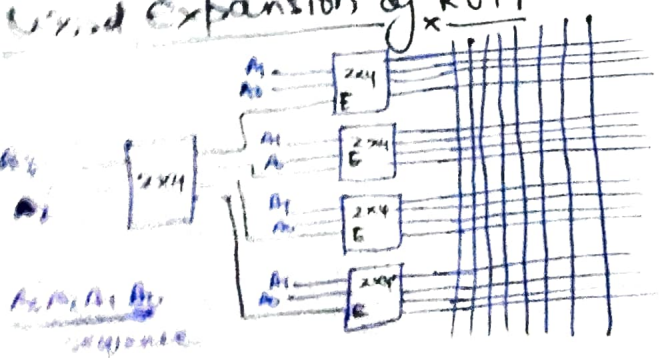


A_2	A_1	A_0	o/p line
0	0	0	①
⋮	⋮	⋮	⋮
1	1	1	⑧

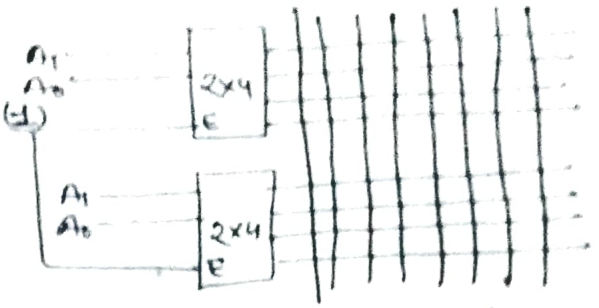
→ We are increasing no. of words (same as no. of address) without increasing size of word.

→ So, we can say that 3×8 decoder works as 8×8 ROM.

Word Expansion of ROM



→ It is a 16×8 ROM.



→ If you set the enable as 1 then only four combinations are possible.

A_1	A_0
0	0
0	1
1	0
1	1

And at each combination 8 buses from each decoder get selected and total 16 buses get selected.

→ So, ROM side becomes (4 words × 16 bits)

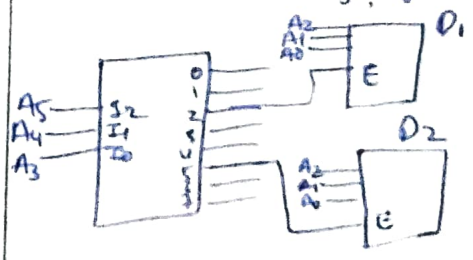
This way we can expand the word size too.

→ This method can apply to RAM also.

→ We can extend this concept upto larger memory.

Finding address range of devices

If $A_5 A_4 A_3 A_2 A_1 A_0$ are addresses, then ranges does D_1 and D_2 get?



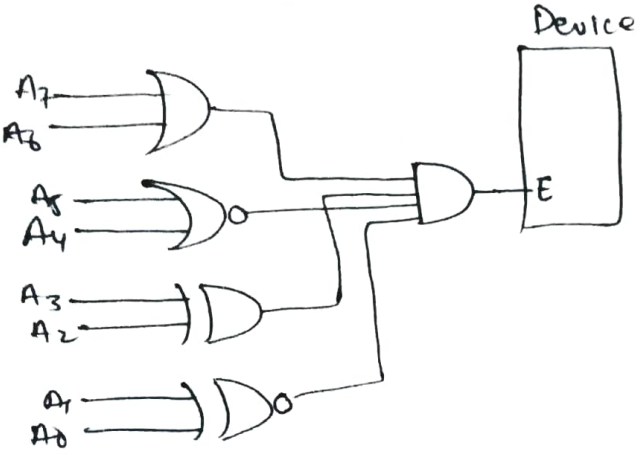
	A_5	A_4	A_3	A_2	A_1	A_0	
D_1 →	0	1	0	0	0	0	start
D_2 →	1	0	1	0	0	0	start

$D_1 \rightarrow (16 \text{ to } 23)$

$D_2 \rightarrow (40\text{-}47)$

Example on enabling a device

Consider a device that is enabled using 8 address bits as shown below. For how many address the device is enabled?



a) 1 b) 8 c) 12 d) None

A_6	A_7
0	1
1	0
1	1

③

A_5	A_4
0	0

①

A_3	A_2
0	1
1	0

②

A_0	A_1
0	0
1	1

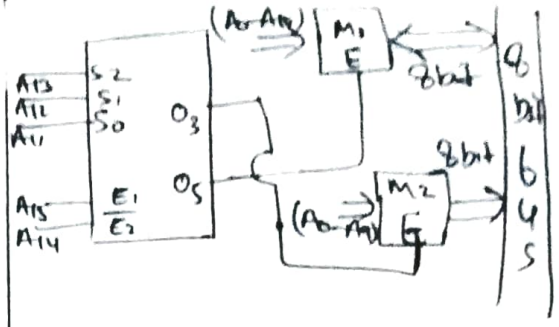
②

$3 \times 2 \times 2 \times 1$

Total 12 address make device enabled.

Finding the address range of memory devices

Consider the following interface of two m/m devices with 3×8 decoder. Compute the nature and address range of each m/m device.



Soln M_1 connects with a dual direction bus means we can perform both read and write on it.

M_2 Connects with one directional bus so we can only read from it

$M_2 \rightarrow$ ROM $M_1 \rightarrow$ RAM.

$M_1 \rightarrow (A_0 - A_{10})$ 11 address bit means 2^{11} words can store

$M_2 \rightarrow (A_0 - A_9)$ 10 bit address means 2^{10} words can store

Size of RAM $\rightarrow 2^{11} \times 8 = 2^{14}$ bits

Size of ROM $\rightarrow 2^{10} \times 8 = 2^{13}$ bits

Address range for M_1

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_0
1	0	1	0	1	0	0
1	0	1	0	1	1	1

$E_1 = 1, E_2 = 0$ O_5

Address range for M_2

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_0
1	0	0	1	1	ϕ_0	0	0
1	0	0	1	1	ϕ_1	1	1

[9800H – 9FFFH]

Here we choose ϕ that comes first as '0' bcoz we are seeking towards getting starting add. and ϕ that comes at last equals to 1 to get final address.

→ But we can give any value to dont care so each m/m location can get two address.

→ In ROM $2^{11} \times 8$ → m/m locations were there. and each m/m location will get 2 address.

Total no. of address in ROM = 2 as $2 \times 2^{10} \times 8$

Introduction to Encoders

→ Encoders are of two types-
1. Non-priority
2. Priority

Ex - 4x2 Encoder (Non-Priority)

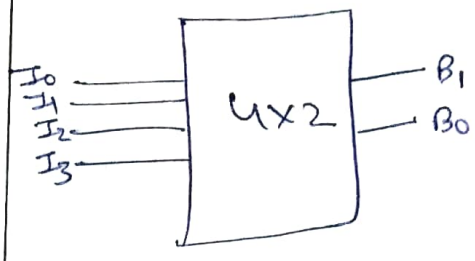
I_0	I_1	I_2	I_3	B_1	B_0
0	0	0	1	1	1
0	0	1	0	1	0
0	1	0	0	0	1
1	0	0	0	0	0

$B_1(I_0, I_1, I_2, I_3) = I_0' I_1' I_2' I_3 + I_0' I_1' I_2 I_3'$

$= I_0' I_1' (I_2 \oplus I_3)$

$B_2(I_1, I_0, I_2, I_3) = I_0' I_1' I_2' I_3 + I_0' I_1 I_2' I_3'$

$\Rightarrow I_0' I_2' (I_1 \oplus I_3)$



Encoders

→ They convert one code to another

→ They perform lossless compression.

→ They are of two types

a) Non-priority (Doesn't support simultaneous i/p activation).

b) Priority (Support simultaneous i/p activation and used for interrupt servicing).

→ Due to static priorities, the lower priorities i/p is exposed to starvation.

Priority Encoders:

Say, priority is $I_3 > I_2 > I_1 > I_0$

Highest suffix input gives the highest priority.

I_0	I_1	I_2	I_3	B_1	B_0
ϕ	ϕ	ϕ	1	1	1
ϕ	ϕ	1	0	1	0
ϕ	1	0	0	0	1
1	0	0	0	0	0

$B_1 = I_3 + I_2 I_3' = I_2 + I_3$

$B_0 = I_3 + I_1 I_2' I_3'$

$= (I_3 + I_1) \cdot (I_3 + I_2')$

→ We can extend it to any no. of inputs, logic will be same.

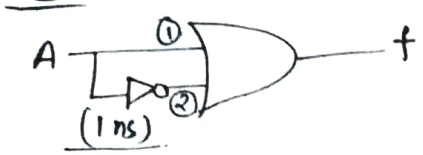
Introduction to Hazards

Hazards are of two type

① Temporary
② Permanent

① Temporary Hazard

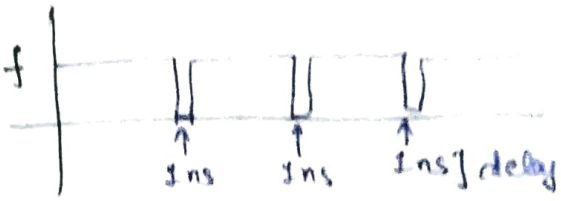
Ex-



Say we have a 'not gate' that takes 1ns to pass a signal then,

assume initially A=1 then value at ① = 1 and value at ② = 0. It might take 1ns to provide value at ② as 0. After setting values we get out put at f=1.

And then suddenly we switch the value at A=0 then value at ① becomes '0' at the same instant but value at ② take 1ns to become '1'. for that one second both location ① and ② will show '0' and f becomes 0 for 1ns, called as hazards.



It is a temporary hazard, and it is going to correct itself.

Permanent Hazard

(Short ckt (always 1) or open ckt (always 0))

Say link get broken at B, so it will always show '0' whatever input you gives, also called struck at '0'.

These errors needs to remove by self.

Hazards in digital circuit

→ The anal function of a digital circuit is called hazard. They can be permanent or temporary. The temporary hazards is due to uneven delays of i/ps.

→ The permanent hazards are resulted due to open circuit or short circuits of the connecting leads (terminals).

→ The hazards can be struck at 0 (s-a-0) or struck at 1 (s-a-1)

→ The single fault analysis is made using "path sensitization technique". This techniques provides test vector.

Procedure to find Test Vector

Cont—

① Inactivate all the other paths expect the tested one. For 'AND' or 'NAND' apply the logic 1 for inactivation.

For 'OR' or 'NOR' apply the logic 0 from inactivation.

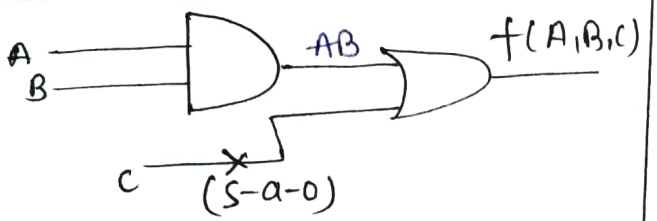
② Apply the opposite logic value at the tested place.

③ The input combinations satisfying the above requirements from the test vector.

9mp

Examples on test vector

①

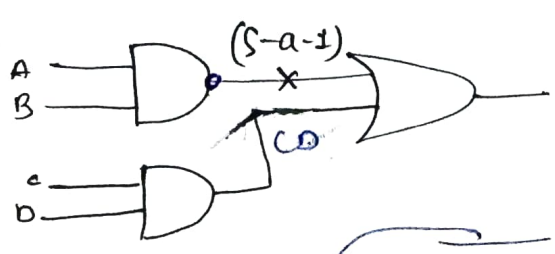


Make AB = 0 so that o/p become depend completely on C value.

A	B	C
0	0	1
0	1	1
1	0	1

Choose value of C as opposite to other values

②



Put CD = 0

C	D	(S-a-1)
0	0	0
0	1	0
1	0	0

A	B
1	0
0	1
0	0

So, for fig ②

A	B	C	D	
0	0	0	0	9 Test Vectors
0	1	0	0	
1	0	0	0	
0	0	0	1	
0	1	0	1	
1	0	0	1	
0	0	1	0	
0	1	1	0	
1	0	1	0	

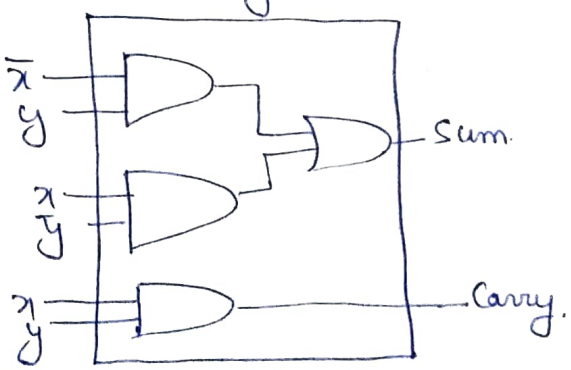
Half Adder

It can do sum of 2 bits only.

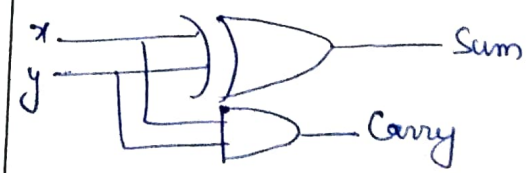
Inputs x	Inputs y	Outputs sum	Outputs carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$S = \bar{x}y + x\bar{y} = x \oplus y$

$Carry = xy$



or



Full Adder

It can perform the sum of 3 bits.

→ It can be constructed by using two half adders

→ It is 3 i/p two o/p device

x	y	z	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

K-map for S

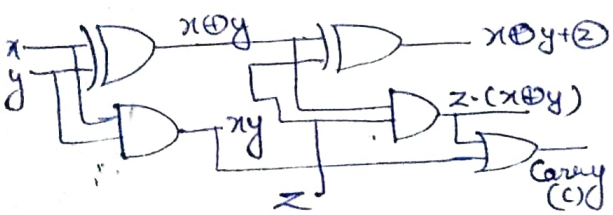
z \ xy	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$S = \bar{x}y\bar{z} + \bar{x}\bar{y}z + xyz + x\bar{y}\bar{z}$$

$$S = x \oplus y \oplus z$$

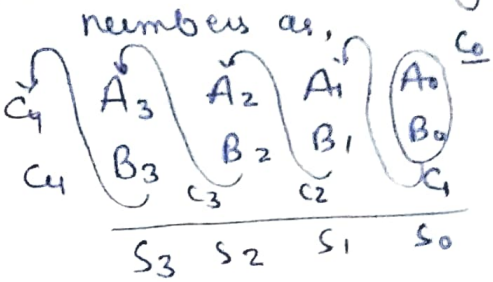
and, $C = xy + yz + zx$

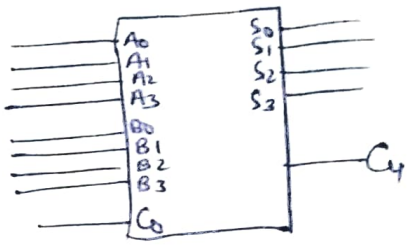
$$= z(x \oplus y) + xy$$



Ripple Carry Adder

Conventions: We are taking 2, 4 bit numbers as,

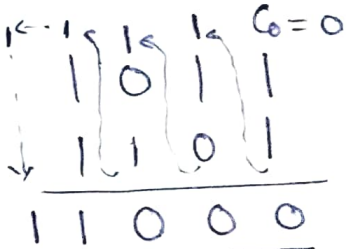


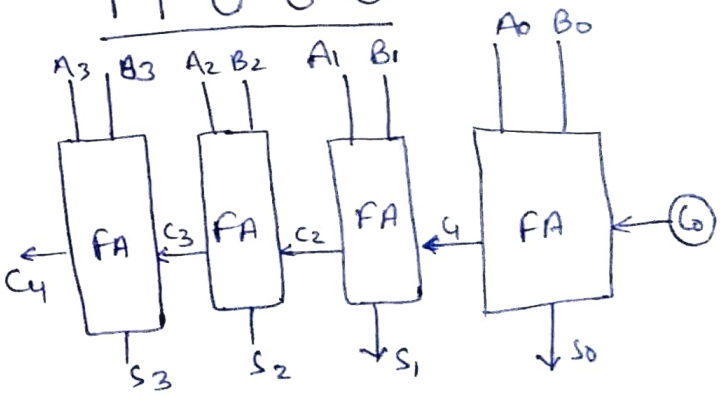


If you above machine it will take 9 inputs means 29 combination So, truth table become very large. So, we will use concept of full adder bcoz it performs add of 3 bits.

Time taken in shifting of carry is latency time

Ex

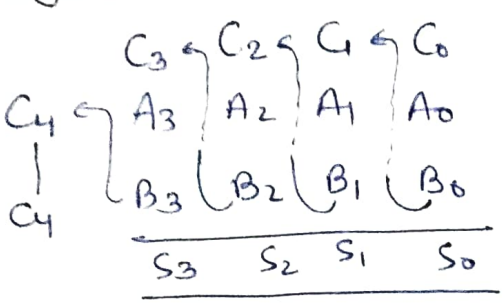




→ Time taken by complete system to add the numbers is high because all the FA works sequentially.

→ To solve this problem of time we have carry look ahead adders.

Carry Look Ahead Adder



$$G_1 = C_0 (A_0 \oplus B_0) + A_0 B_0$$

$$C_2 = G_1 (A_1 \oplus B_1) + A_1 B_1$$

$$C_3 = C_2 (A_2 \oplus B_2) + A_2 B_2$$

$$C_4 = C_3 (A_3 \oplus B_3) + A_3 B_3$$

say, $G_i = A_i B_i$ (↳ Generate fn), $P_i = A_i \oplus B_i$ (↳ Propagation fn)

$$C_1 = C_0 P_0 + G_0$$

$$C_2 = C_1 P_1 + G_1$$

$$C_3 = C_2 P_2 + G_2 \qquad C_4 = C_3 P_3 + G_3$$

In terms of C_0,

$$C_1 = C_0 P_0 + G_0$$

$$C_2 = (C_0 P_0 + G_0) P_1 + G_1$$

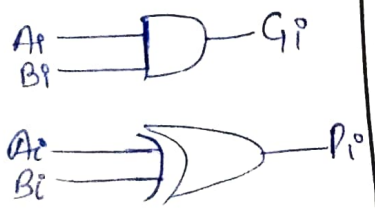
$$= C_0 P_0 P_1 + G_0 P_1 + G_1$$

$$C_3 = (C_0 P_0 P_1 + G_0 P_1 + G_1) P_2 + G_2$$

$$= C_0 P_0 P_1 P_2 + G_0 P_1 P_2 + G_1 P_2 + G_2$$

$$C_4 = C_0 P_0 P_1 P_2 P_3 + G_0 P_1 P_2 P_3 + G_1 P_2 P_3 + G_2 P_3 + G_3$$

Implementation

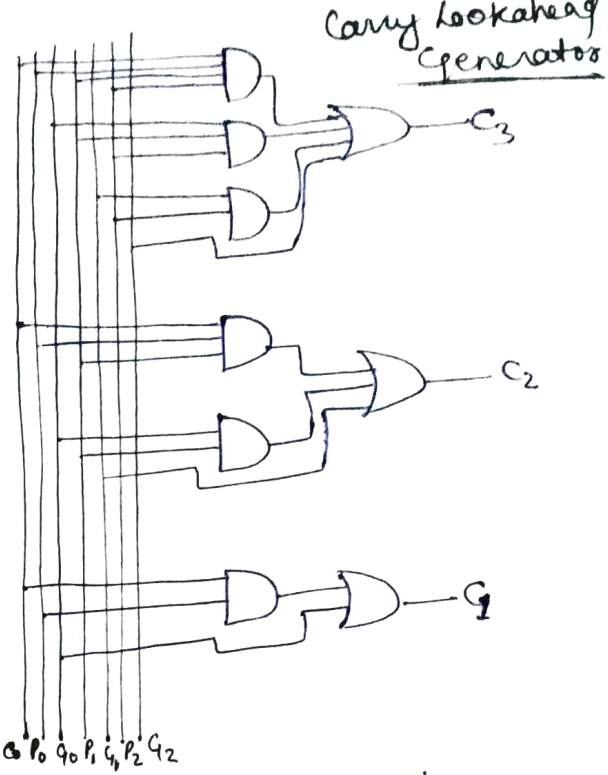


for n-bit no's adder

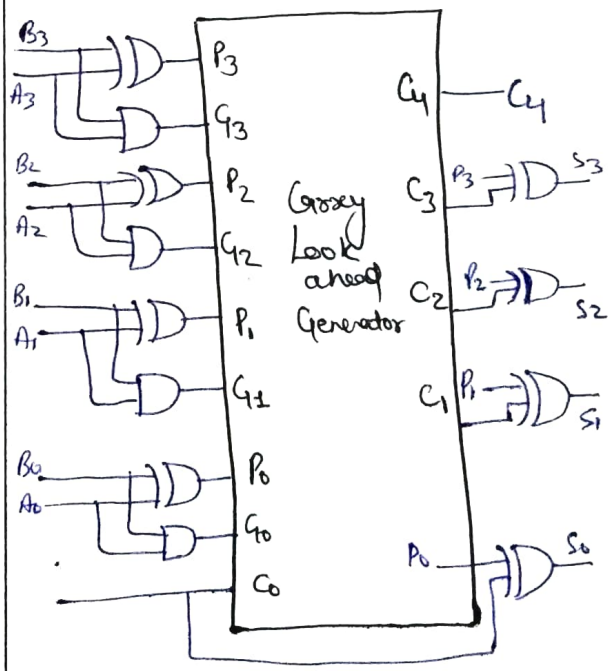
AND Gate $= \frac{n(n+1)}{2}$

OR Gate $= f(n)$

Carry Lookahead Generator



It is better than ripple carry adder because it doesn't work in a sequential manner. It depends only on starting input parameters.



Advantages:- ① There is no rippling bcoz no one is waiting for the response from previous stages.

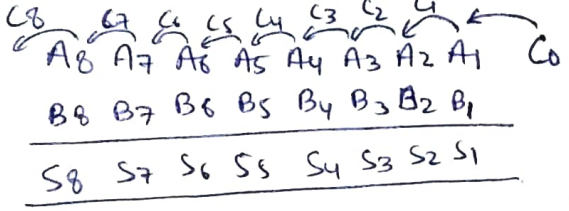
Hybrid Adder

→ Ripple carry adder is simple and cheap but slower.

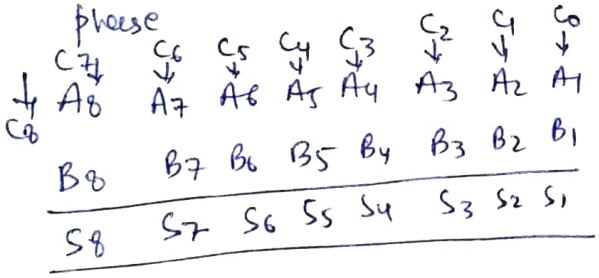
→ Carry look ahead is complex and costly but fast.

→ So, both have its own pros and cons.

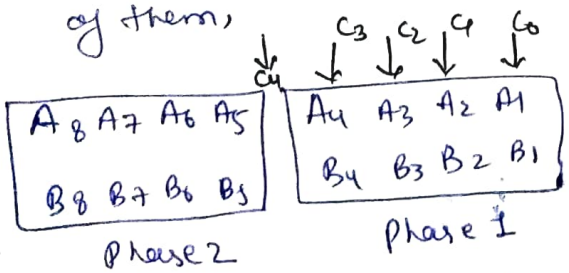
Suppose, we are adding two 8 bit numbers, in ripple adder.



In carry look ahead generator all the carries generated directly with no dependency on previous phase



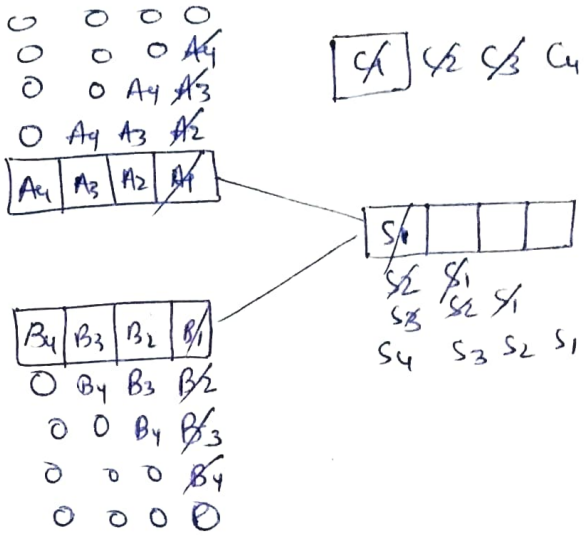
In hybrid we combine both of them,



So here we compute the phase 1 first and then whatever the final carry we get (C_4) we provide that value as C_0 to another phase.

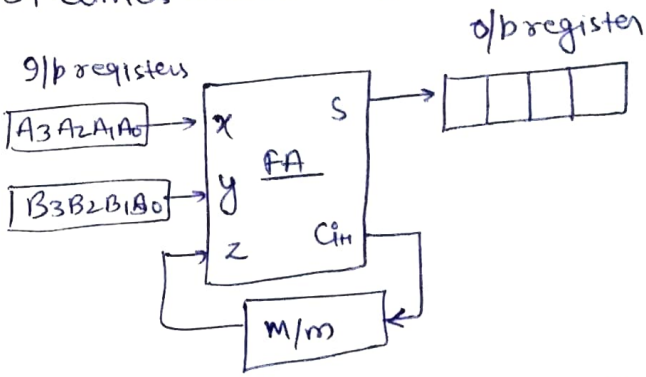
Since, 8 bit adder must have complex ckt but 4 bit adder have less complex ckts.

Serial Adder



Everything will perform in shift registers.

It comes under sequential ckts



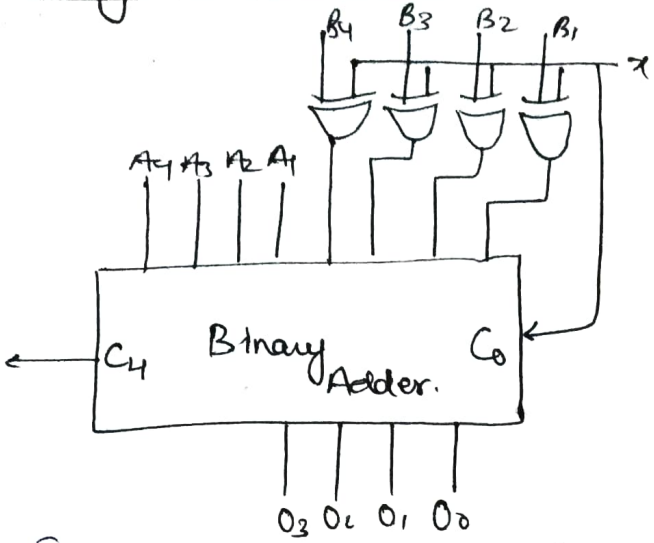
Output is depends on inputs and carry of previous inputs.

→ It is a serial adder and very slow as compare to all the others.

→ It is cheapest one bcoz we need only one FA.

→ We will read about in sequential ckts.

Binary Adder / Subtractor



So, this ckt can work as both adder and subtractor as, we know -

$B \oplus 0 = B\bar{0} + \bar{B}0 = B$

$B \oplus 1 = B\bar{1} + \bar{B}1 = \bar{B}$

If $x = 0$ we receive all the bits as it is, and adder will perform $(A + B)$.

If $x = 1$ then it will perform $(A + \bar{B} + 1)$

$\uparrow$ C_0

$\Rightarrow$ A + 2's complement of B

$= \underline{A - B}$.

→ So, we can implement various operation using binary adder.

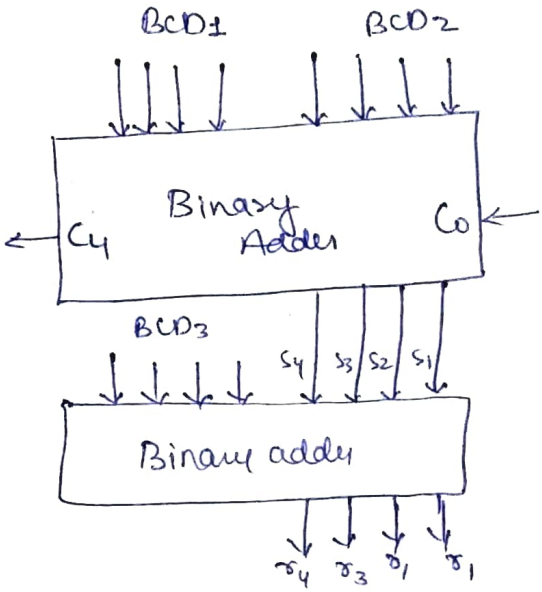
Ex

	A	B	x	function
a)	BCD	0011	0	→ (BCD to XS-3 Converter)
b)	XS-3	0011	1	→ (XS-3 to BCD Converter)
c)	1001	BCD	1	→ (9's complement of BCD)
d)	1010	BCD	1	→ (10's complement of BCD)

BCD Adder

If output is greater than 9 or we are generating a carry then output is incorrect or not in BCD.

→ for correction we add 6 in output.



If there is no need of correction then give BCD_3 as '0' (zero).

Correction is required if

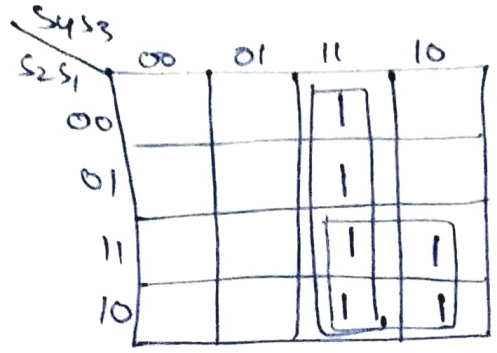
$S_4S_3S_2S_1 > 9$ or

$C_4 = 1$.

To detect $S_4S_3S_2S_1 > 9$,

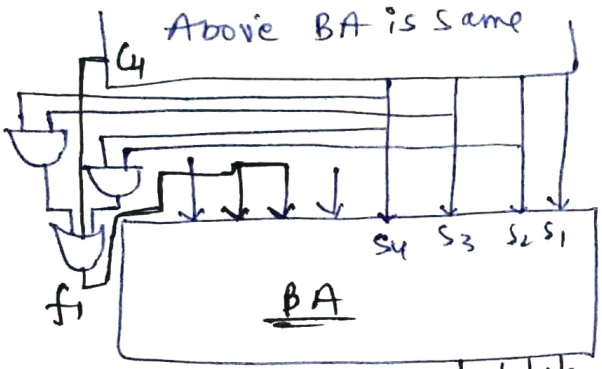
S_4	S_3	S_2	S_1	f
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

f is a function that gives '1' for $S_4S_3S_2S_1 > 9$ otherwise gives '0'



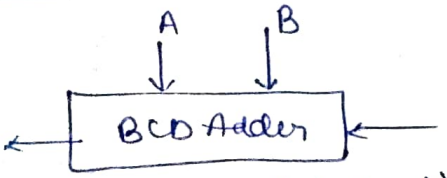
$f = S_4S_2 + S_4S_3$ will acts as alarm to output >9.

→ Solve do following changes in previous ckt o



$f_1 = f + C_4$ → for generated Carry correction

Invalid Combinations for BCD Adder



How many invalid combinations at A, B can occur to give incorrect output?

Invalid comb^n = Total Combination − Valid comb^n

Ex: 4 bits numbers A and B

Total Comb^n = 16 × 16 = 256

Valid Comb^n = 10 × 10 (0–9)
= 100

Invalid comb^n = 256 − 100
= 156

(2) Approach

(A) 4 bit = 2^4 = 16 combinat
(0–9) valid (10–15) invalid for BCD

A → (10–15)

B → Any 16 comb^n

(16 × 6) + (6 × 16)

↓ by B belong to (10–15).

Total invalid =
(16 × 6) + (16 × 6) − (6 × 6)
= 156 [for common in both]

2-Bit Comparator

→ for unsigned number

A_2A_1	B_2B_1	A>B	A=B	A<B
0	0			
0	1			
1	0			
1	1			

→ by we go like this it will take a lot of time to analyse if we have input size is more than 2 bit. so we go for another approach

X-NOR can be used as two bit Comparators as.

a	b	a ⊙ b	
0	0	1	→ same bits
0	1	0	→ Differed bits
1	0	0	
1	1	1	→ " → same bits

$\frac{A}{A_2\ A_1}$ $\frac{B}{B_2\ B_1}$] 2 bits numbers.

Case(i) $A = B$ if $A_2 = B_2$ and $A_1 = B_1$

So, $x_1 = x_2 = 1$ or $x_1 \cdot x_2 = 1$

Case(ii) $(A > B)$ if $(A_2 > B_2)$ or
(1) (0)

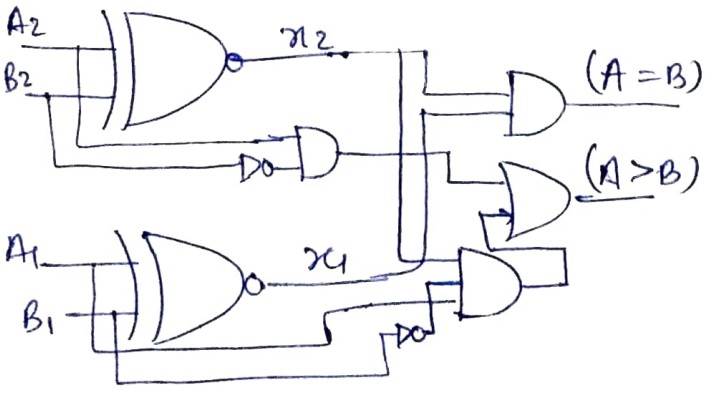
$(A_2 = B_2$ and $A_1 > B_2)$
(1) 0

$\Rightarrow (A_2 \overline{B_2} + x_2 \cdot A_1 \overline{B_1}) = 1$

Case3 $(A < B)$ if $(A_2 < B_2)$ or
0 1

$(A_2 = B_2$ and $A_1 < B_1)$
0 1

$\Rightarrow \overline{A_2} B_1 + (x_2 \cdot \overline{A_1} B_1) = 1$



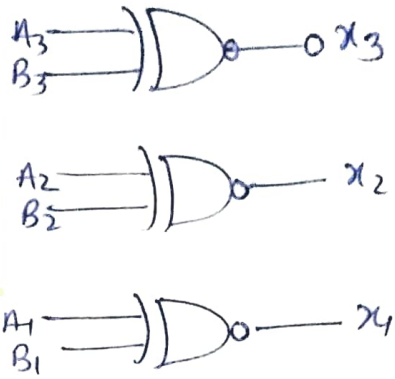
Similarly we can design for $(A < B)$.

3,4 bit Comparator

$A = A_3 A_2 A_1$

$B = B_3 B_2 B_1$

We assume that we already have 1 bit comparator as



Case 1: $A = B$. if

$x_1 \cdot x_2 \cdot x_3 = 1$

Case 2: $(A > B)$ if

$A_3 > B_3 = A_3 \overline{B_3}$
(1) (0)

(or)

$A_3 = B_3$ and $A_2 > B_2$

$= x_3 \cdot A_2 \overline{B_2}$

(or)

$A_3 = B_3$ and $A_2 = B_2$ and

$A_2 > B_1$

$= x_3 \cdot x_2 \cdot \overline{B_1} A_1$

Case3 $(A < B)$ iff

$\overline{A_3} B_3 + x_2 \overline{A_2} B_2 + x_2 x_3 B_1 \overline{A_1}$

$= 1$

Similarly we can designed for 4 bit comparators

Analyzing all the cases of Comparator

2-bit comparator

A_2	A_1	B_2	B_1	
A		B		
0	0	0	0	(0)
0	0	0	1	(1)
0	0	1	0	(2)
0	0	1	1	(3)
0	1	0	0	(4)
0	1	0	1	(5)
0	1	1	0	(6)
0	1	1	1	(7)
1	0	0	0	(8)
1	0	0	1	(9)
1	0	1	0	(10)
1	0	1	1	(11)
1	1	0	0	(12)
1	1	0	1	(13)
1	1	1	0	(14)
1	1	1	1	(15)

A=B have 4 combinations.

$16 - 4 = 12$

$\frac{12}{2}$ = (A>B and A<B)

bcoz both have symmetric combinations.

for n bit comparion

A = B for 2^n combination.

Remaining combinations $2^{2n} - 2^n$

$A > B = A < B = \frac{2^{2n} - 2^n}{2} = 2^{2n-1} - 2^{n-1}$

Time complexity of ripple carry Adder = $O(n)$

(n → no. of Full Adders)

Time complexity of look ahead adder = $O(\log_k n)$

k = fan-in

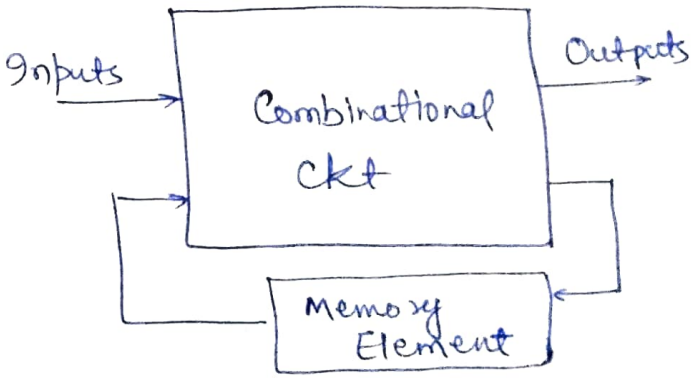
n → no. of inputs

no. of levels $l = \log_k (n+1)$

Sequential Circuits

→ The external o/p of a sequential circuit depends on external i/ps and on present content of "m/m elements".

→ The present contents of the m/m elements are called as present state and new content of m/m elements are obtained by taking external i/p's and present state is called as next state.



Latch and flip flop

1 bit memory cell: A m/m element is some medium in which one bit of information

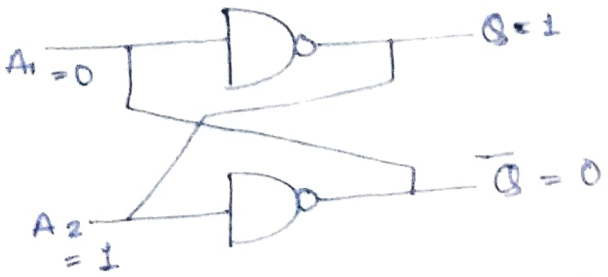
Can be stored or retained until necessary, and thereafter its content can be replaced by another value. It means flip flop can store a bit for long time and also modify it if required.

→ The binary or digital memory circuit is known as flipflop.

→ It has 2 stable states which are known as '1' or '0'. So it is also called bistable multivibrator.

→ flip-flop is basic block of RAM.

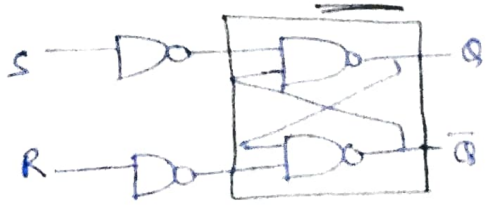
→ We can use any functionally complete gate to develop a flipflop, but generally we use NAND gate.



In flipflop we can change the i/p's but in latches we can't change the inputs so each value get locked after setting an inputs.

SR- flipflop

Latch



Q_n = present state

Q_{n+1} = next state

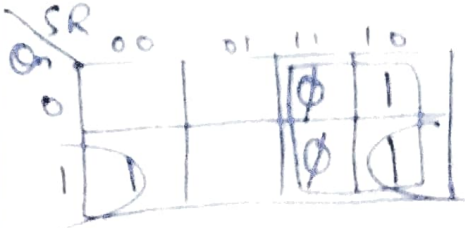
$Q_{n+1} = f(S, R, Q_n)$

Characteristic table

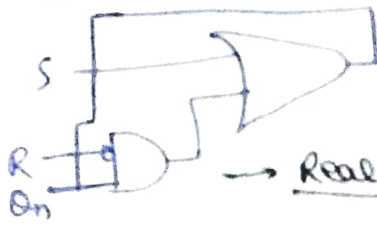
S	R	Q_n	Q_{n+1}	
0	0	0	0	Latch
0	0	1	1	Latch
0	1	0	0	Reset
0	1	1	0	Reset
1	0	0	1	set
1	0	1	1	set
1	1	0	∅	Invalid
1	1	1	∅	Invalid

Set { $Q_n \to Q_{n+1}$: 0 → 1, 1 → 1 } Reset { 0 → 0, 1 → 0 }

∅ → S = R = 1 set the value of Q, Q̄ both as 1 and that is invalid so we don't care.



$Q_{n+1} = S + \bar{R} Q_n$ → Characteristic eqn



→ Realisation

Q_n	Q_{n+1}	S	R	
0	0	0	∅	Excitation table
0	1	1	0	
1	0	0	1	
1	1	∅	0	

function table

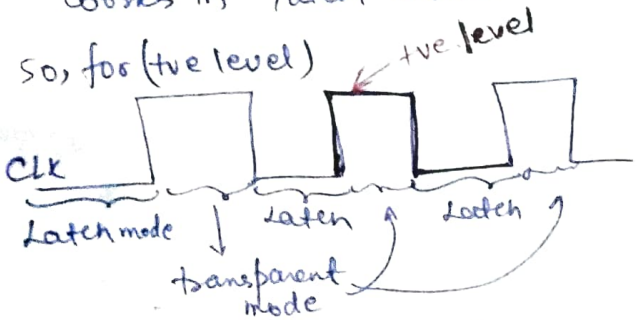
S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\emptyset$

Clocked flip flops

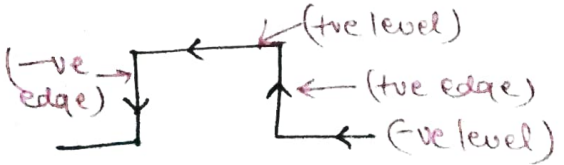
Sometimes it might be possible that S and R both have need of a constant value for long so both sets its value as required but due to noise value get changed and output get fluctuated.

To resolve above issue we use clock signal as CLK. In the initial CLK = 0, so SR flipflop is in latch mode similarly at any time if CLK = 0 irrespective of values of S & R, flipflop works in latch mode.

So, for (tve level)

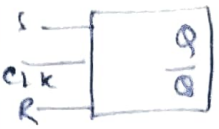


In transparent mode value of Q_{n+1} and o/p depends on S/R.



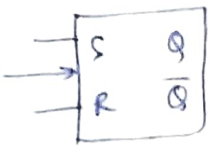
tve level triggered

tve level

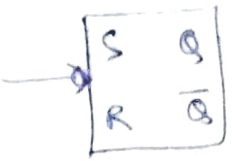


-ve Level

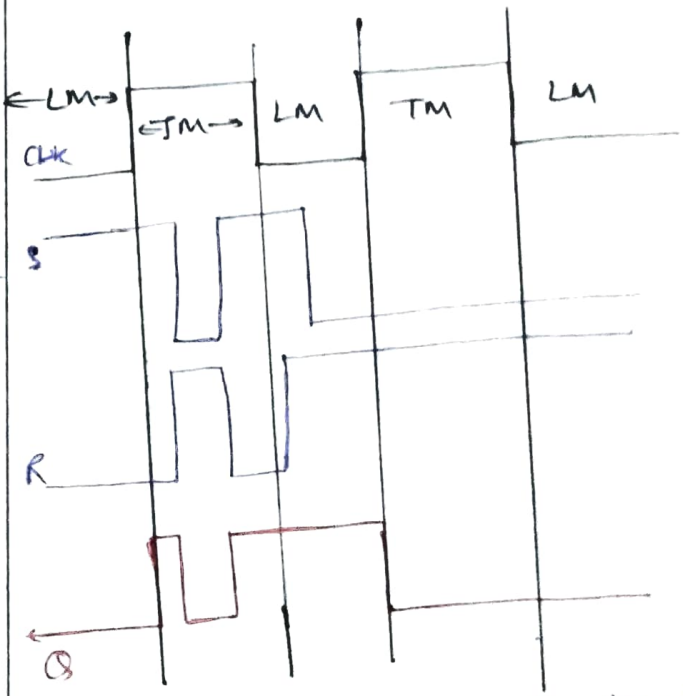
tve edge



-ve edge



Positive Level Triggered



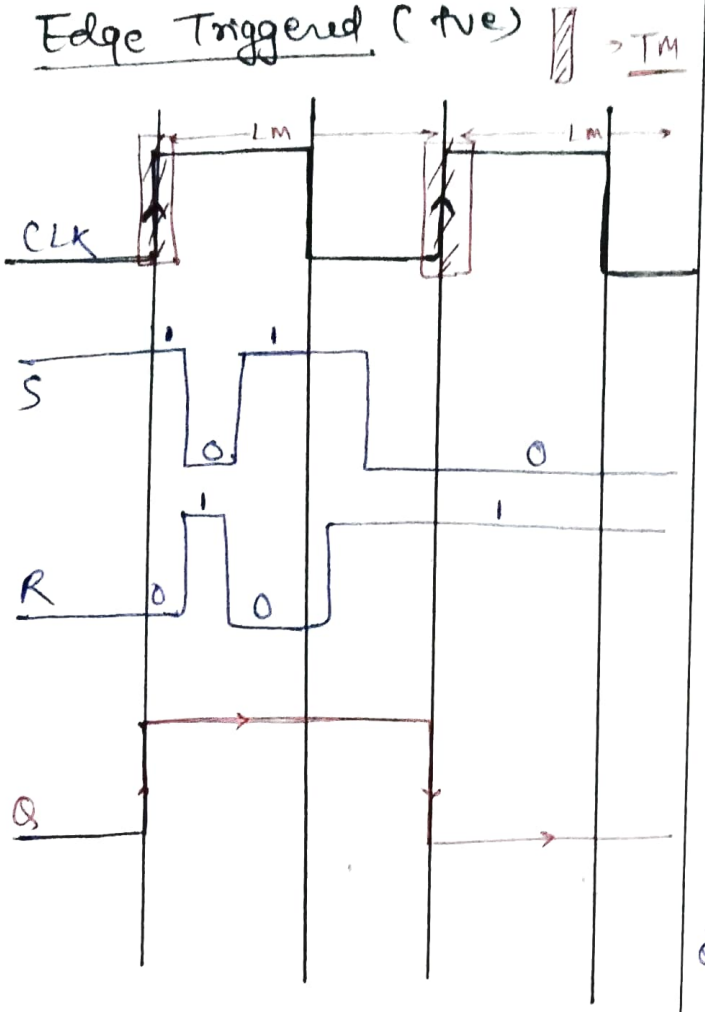
LM → Latch mode TM → Transparent mode

CLK, S, R are input signals and Q follows the all 3 to decide o/p.

→ Q can change its value in TM only and LM it will continue the present state independent of inputs.

→ Initial value Q assumes as '0'.

Edge Triggered (+ve)



So we can say that Latch and transparent mode changes accoseding to mode of flipflop

JK Flip Flop

JK is same as SR flipflop ($J=S, K=R$). It is also defined $J=K=1$ and the flipflop perform complementation.

J	K	Q_{n+1}
0	0	Q_n
1	0	1
0	1	0
1	1	$\overline{Q_n}$

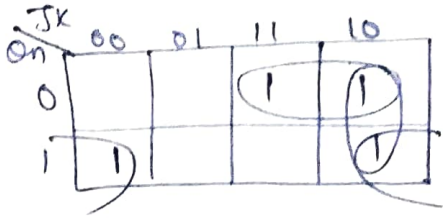
function table of JK

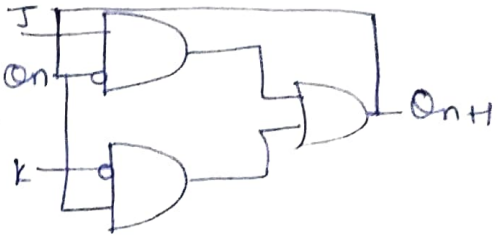
Characteristic table

J	K	Q_n	Q_{n+1}	
0	0	0	0	Q_n
0	0	1	1	
0	1	0	0	0
0	1	1	0	
1	0	0	1	1
1	0	1	1	
1	1	0	1	$\overline{Q_n}$
1	1	1	0	

Characteristic eqn

$Q_{n+1} = J\overline{Q_n} + \overline{K}Q_n$] essential prime implicant

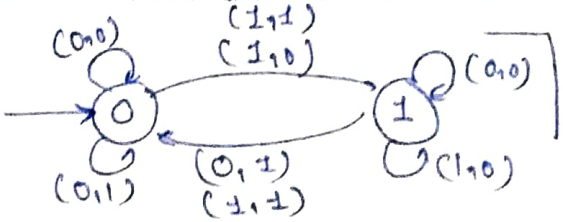




Excitation Table

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

On the basis of Characteristic table

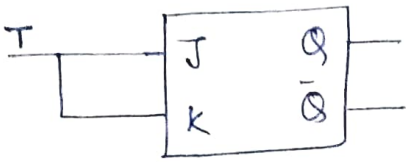


State Diagram

(0) → denotes reset state

(1) → denotes set state

T- flipflop (Toggle FF)



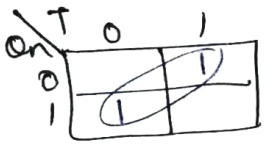
Combine both input

fanction Table

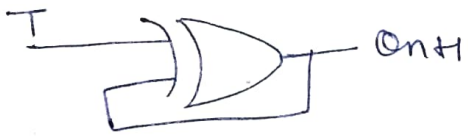
T	Q_{n+1}	
0	Q_n	] Latching
1	$\bar{Q}_n$	] Complementing

Chasacteristic table

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



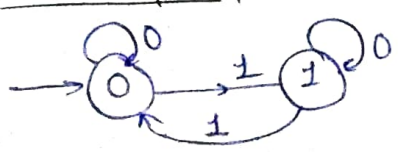
$Q_{n+1} = T \oplus Q_n$



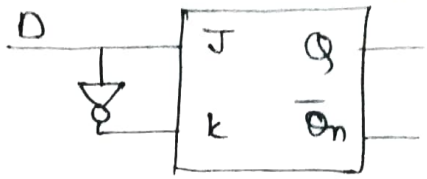
Excitation Table

Q	Q_{n+1}	T
0 →	0	0
0 →	1	1
1 →	0	1
1 →	1	0

State diagram



D flipflop (Delay F F)



D	Q_{n+1}	
0	0	(J=0, K=1)
1	1	(J=1, K=0)

Characteristic Table

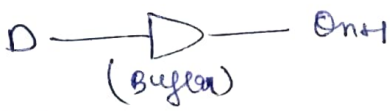
D	Q_n	Q_{n+1}	
0	0	0	} D=0
0	1	0	
1	0	1	} D=1
1	1	1	

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

$= D$

→ Output is directly depends on input. so no need of m/m element or feedback.

→ It use to provide delay only.

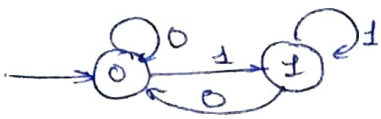


→ It is use to remove temporary hazards due to delays.

Excitation table

Q_n	Q_{n+1}	D	
0 →	0	0	} Depends on Q_{n+1}
0 →	1	1	
1 →	0	0	
1 →	1	1	

State diagram



Example on flip flop. (1)

Characteristic eqn of a flipflop is $Q_n = \bar{x}_1 \bar{Q} + \bar{x}_2 Q$. Define the behaviour of this.

function table

x_1	x_2	Q_n	Behaviour
0	0	1	→ Set
0	1	$\bar{Q}$	→ toggle
1	0	Q	→ Latch
1	1	0	→ Reset

Characteristic table

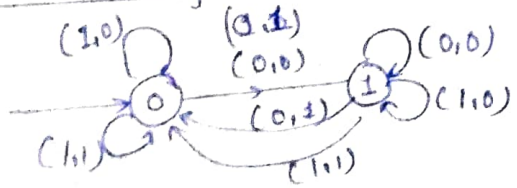
x_1	x_2	Q	Q_n	
0	0	0	1	} set
0	0	1	1	
0	1	0	1	} toggle
0	1	1	0	
1	0	0	0	} Latch
1	0	1	1	
1	1	0	0	} Reset
1	1	1	0	

$Q_n = \bar{Q}\bar{x}_1 + \bar{x}_2 Q$ → satisfying the Characteristic table.

Excitation table

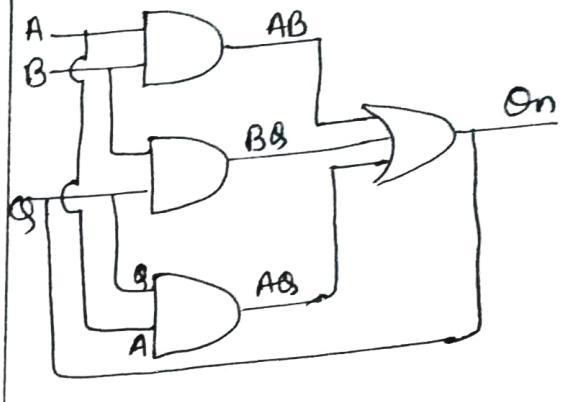
Q	Q_n	x_1	x_2
0 → 0		1	ϕ
0 → 1		0	ϕ
1 → 0		ϕ	1
1 → 1		ϕ	0

State diagram



Example (2)

Consider the following realisation. construct the excitation tables



Soln

Characteristic eqn

$Q_n = AB + BQ + AQ$

function table

A	B	Q_n	
0	0	0	→ Reset
0	1	Q	} Latch
1	0	Q	
1	1	1	→ Set

Characteristic table

A	B	Q	Q_n
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Excitation table

Q	Q_n	AB
0 → 0		ϕ ϕ
0 → 1		1 1
1 → 0		0 0
1 → 1		ϕ ϕ

→ Wrong table

We can write ØØ bcoz it also includes (1,1) but have only 3 combination as (0,0), (0,1), (1,0). So we write it separately.

Q	Qn	A	B	
0	0	0	Ø	≠ (ØØ)
0	0	1	0	
1	0	0	0	
0	1	1	1	
1	1	1	Ø	≠ [Ø,Ø]
1	1	0	1	

Introduction to flipflop Inter Conversion

Procedure

1. Get the characteristic of target Flipflop
2. Replace the next state using excitation of given FF.
3. Obtain the expression for i/p of given FF and realise them.

Ex→ JK to T FF

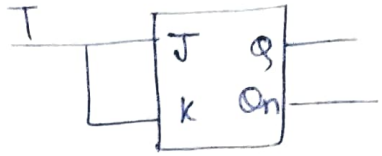
T	Qn	Qn+1	J	K
0	0	0	0	Ø
0	1	1	Ø	0
1	0	1	1	Ø
1	1	0	Ø	1

J = T

Qn \ T	0	1
0	Ø	1
1	Ø	Ø

K = T

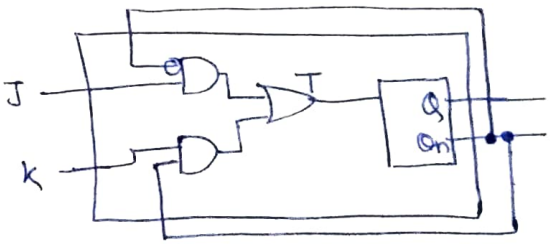
So, J = K = T



Convert T to JK

J	K	Qn	Qn+1	T
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	1

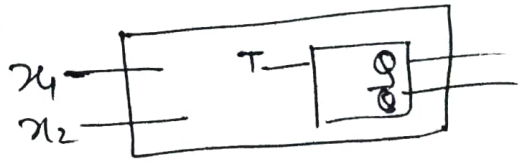
$T = Qn\,K + J\,\overline{Qn}$



Example on conversion

A new flipflop x_1x_2 has Characteristic expression $Qn = \overline{x_1}\,Q + \overline{x_2}\,Q$. Realise it using T flipflop.

Soln

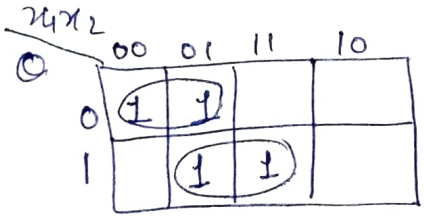


function Table

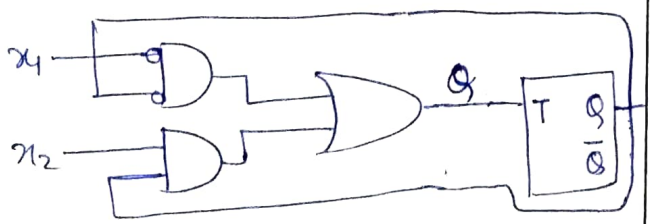
x_1	x_2	Q_n
0	0	1
0	1	$\bar{Q}$
1	0	Q
1	1	0

Characteristic table

x_1	x_2	Q_n	Q_{n+1}	T
0	0	0	1	1
0	0	1	1	0
0	1	0	1	1
0	1	1	0	1
1	0	0	0	0
1	0	1	1	0
1	1	0	0	0
1	1	1	0	1

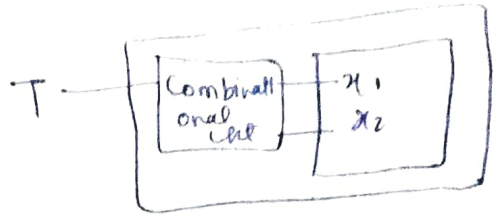


$T = \bar{x}_1\bar{Q} + x_2Q$



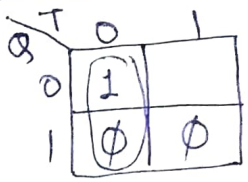
→ follow the convention given in question for Q_n and Q_{n+1}

Ex-2 Convert $x_1x_2 \longrightarrow T$
given that $Q_n = \bar{x}_1\bar{Q} + \bar{x}_2Q$



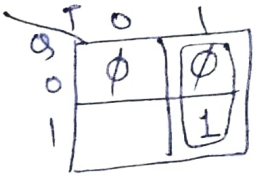
T	Q	Q_n	x_1	x_2
0	0	0	1	ϕ
0	1	1	ϕ	0
1	0	1	0	ϕ
1	1	0	ϕ	1

for x_1

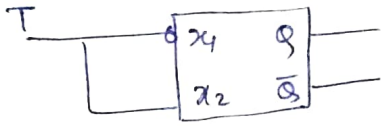


$x_1 = T'$

for x_2

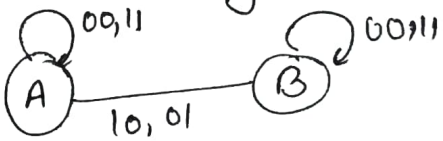


$x_2 = T$



gmb

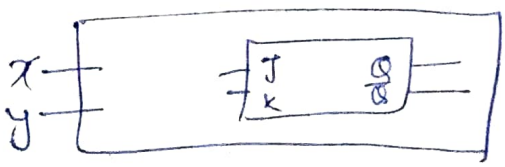
An x,y flipflop is represented by the following state diagram



To realise it using JK ff.

Find J & K.

→ Either A, B = {0, 1} or there is no direction on edge joining (A) and (B) and it's not a typing mistake.



x	y	Q_n	Q_{n+1}	J	K
0	0	0	→ 0	0	∅
0	0	1	→ 1	∅	0
0	1	0	→ 1	1	∅
0	1	1	→ 0	∅	1
1	0	0	→ 1	1	∅
1	0	1	→ 0	∅	1
1	1	0	→ 0	0	∅
1	1	1	→ 1	∅	0

functional table

x	y	Q_{n+1}
0	0	Q_n
0	1	$\overline{Q_n}$
1	0	$\overline{Q_n}$
1	1	Q_n

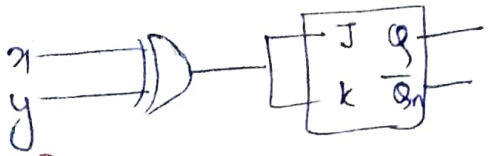
] from given state diagram

Using minimisation from k-map we get

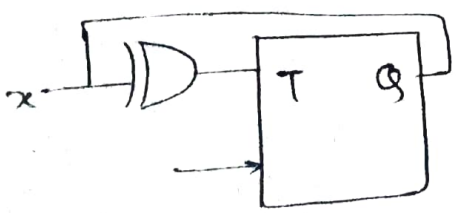
$J = x'y + xy' = x \oplus y$

$K = x'y + xy' = x \oplus y$

So, $J = K = x \oplus y$



Q What is the behaviour of following one-input flipflop ?



a) D-FF b) T-FF

c) Inverted D-FF d) Inverted T-FF

Soln

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

$Q_{n+1} = x$

Delay provided only so one in D-FF.

Counters

The counters are used to provide accurate timing and control signals.

→ These are of two types :-
a) Synchronous b) Asynchronous

→ In synchronous counters all the flipflops respond to the same clock instances.

→ In asynchronous counters, the output of one flipflop drives the clock of another flipflop.

→ Synchronous counters are faster than asynchronous counters.

→ Due to simplicity of design, asynchronous counters are used in IC fabrication.

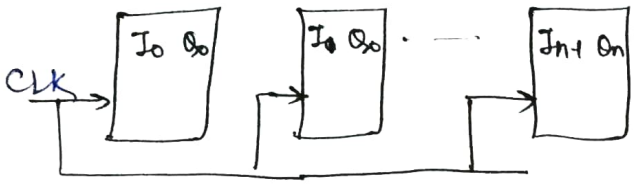
→ The simplified version of Synchronous counter is called shift counters.

→ D-FF is basic element of shift counters.

The ring and Johnson counters are further simplified version of shift counters

Asynchronous and Synchronous Counters

① Synchronous



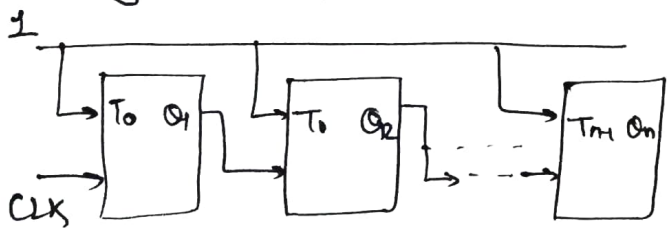
Input can be function of o/p but we are just explaining the concept of counter here.

Propagation delay in synchronous counter is,

$$T_{bdsyn} = T_{FF} + T_{Combinational}$$

$$T_{CLK} \geq T_{bdsyn}$$

② Asynchronous



Propagation delay in asynchronous counter (rippling effect) is,

$$T_{bdasyn} = N \times T_{FF} + T_{Combinational\ Ckt}$$

$$T_{CLK} \geq T_{bdasyn}$$

T_{CLK} → After how much time you will give IInd input.

Shift Counters

In synchronous counters,

$$Input\ (I_i) = f(Q_0, Q_1 \cdots Q_{N-1})$$

$$0 \leq i \leq N-1$$

→ It's design is more complex So, simplification are done as follows.

1) $I_i = Q_{i-1}$

$\therefore I_1 = Q_0$

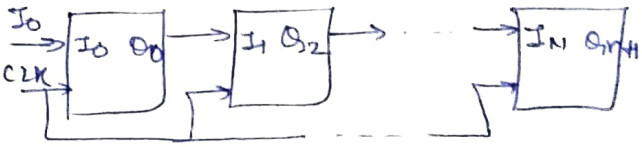
$I_2 = Q_1$

$\vdots$

$I_N = Q_{N-1}$

Here data is shifted from one counter to another next FF. So it is called as shift counter.

→ All are D-FF in shift register



In shift counter

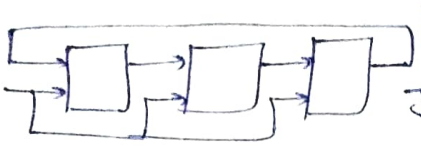
$$I_0 = f(Q_{N-1})$$

$I_0 = Q_{N-1}$ (Ring Counter)

$I_0 = \overline{Q}_{N-1}$ (Johnson Counter)

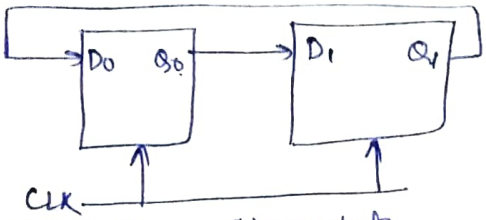
→ In we have 'n' flipflops then 2^n states are possible then counter size should be mod (2^n).

Ring counter

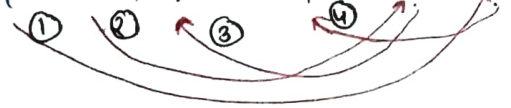


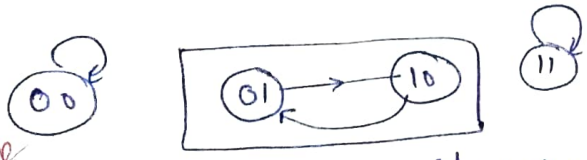
Johnson's Counter.

Mod 2 ring counters



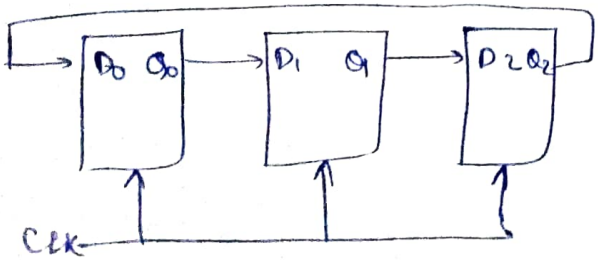
Present State		Next State			
Q_1	Q_0	Q_{1N}	Q_{0N}	D_1	D_0
0	0	0	0	0	0
0	1	1	0	1	0
1	0	0	1	0	1
1	1	1	1	1	1





So, even we have 2^N states for (N-ff) but we get only N useful state so, it is (mod N) counter. and here N is 2. (Ring Counter)

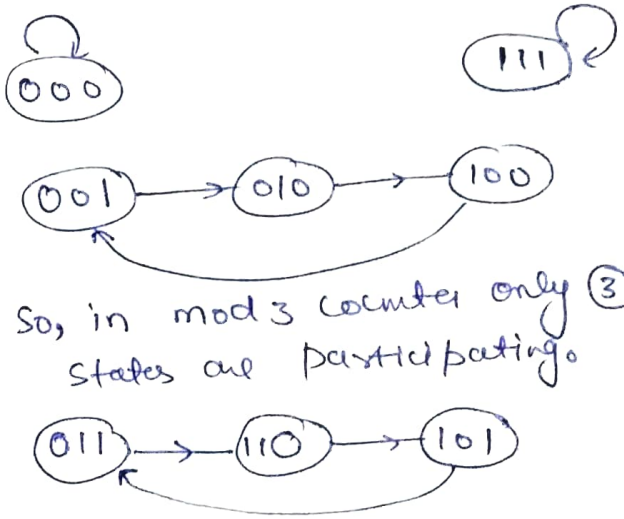
Mod 3 Counter



Q_2	Q_1	Q_0	Q_{2N}	Q_{1N}	Q_{0N}	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	1	0
0	1	0	1	0	0	1	0	0
0	1	1	1	1	0	1	1	0
1	0	0	0	0	1	0	0	1
1	0	1	0	1	1	0	1	1
1	1	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	1

Present | Next

$D_0 = Q_2$, $D_1 = Q_0$, $D_2 = Q_1$



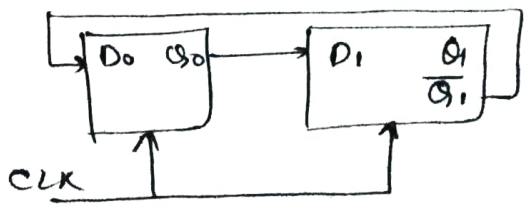
So, in mod 3 counter only ③ states are participating.

→ Transitions are from present state to next state.

→ At each clock bits get shift to next flip flop and we reach the next state of DFF.

→ we can use this counter in Round-Robin Scheduling bcoz each time only one device get active.

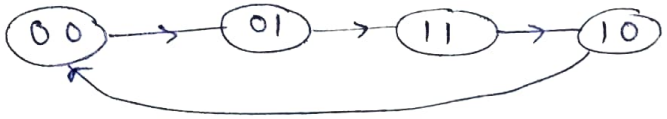
Mod 4 Johnson Counter



→ It is also called twisted ring counter or Johnson's Counter.

Q_1	Q_0	Q_{1N}	Q_{0N}	D_0	D_1
0	0	0	1	1	0
0	1	1	1	1	1
1	0	0	0	0	0
1	1	1	0	0	1

$D_0 = \overline{Q_1}$ $D_1 = Q_0$

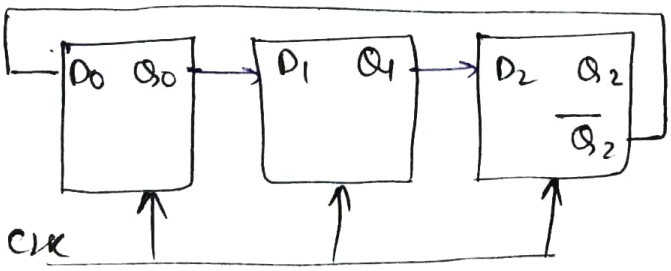


→ 4 states are involved here so it is mod 4 counter.

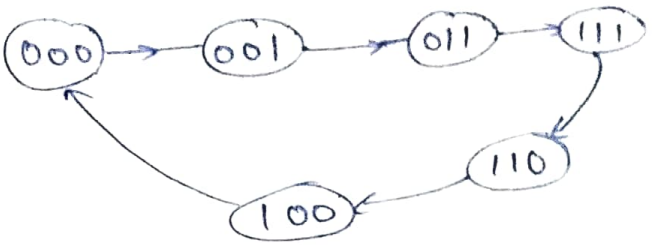
It means after each trip of 4 shift we reach to initial state.

Prob → In Johnson counter if N flipflop are present then it is (mod 2N) counter.

Mod 6 Johnson Counter



Q_2	Q_1	Q_0	Q_{2N}	Q_{1N}	Q_{0N}	D_2	D_1	D_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	1	0	1	1
0	1	0	1	0	1	1	0	1
0	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	1	0
1	1	0	1	0	0	1	0	0
1	1	1	1	1	0	1	1	0



→ So after 6 shifts we get the first input again.

→ So it is mod 6 counter.

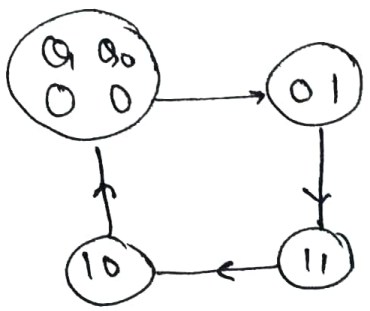
→ We will get reverse o/p on reversing LSB & MSB

Mod 4 gray counter using T-FF

Designing a counter from functions

1. Get the state table from state diagram.
2. Identify flipflops to be used and replace the concerned next state using excitation table of associate flip flop.
3. Get the expression for ips and realize them.

Q Design a synchronous counter for the following using T-FFs.



Soln Cont—

State table

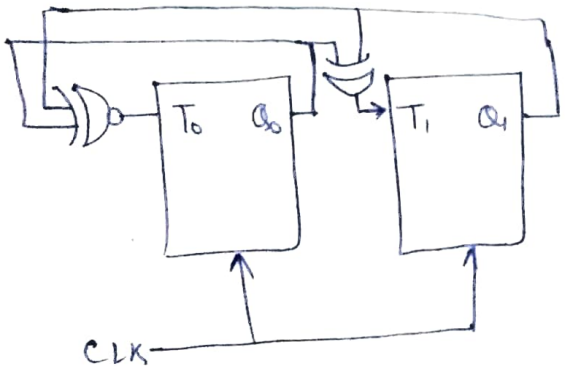
Q_1	Q_0	Q_{1N}	Q_{0N}	T_1	T_0
0	0	0	1	0	1
0	1	1	1	1	0
1	1	1	0	0	1
1	0	0	0	1	0

for T_1 entries Consider $Q_1 \rightarrow Q_{1N}$

for T_0 entries —— $Q_0 \rightarrow Q_{0N}$

$T_1 = \bar{Q}_1 Q_0 + Q_1 \bar{Q}_0 = Q_1 \oplus Q_0$

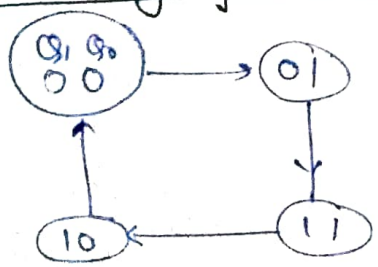
$T_0 = \bar{Q}_1 \bar{Q}_0 + Q_1 Q_0 = Q_1 \odot Q_0$



→ Since in each transition in DFA only 1 bit is changing and it is called gray code. [00 → 01 → 11 → 10 → (00)]

→ So it is mod 4 gray Counter.

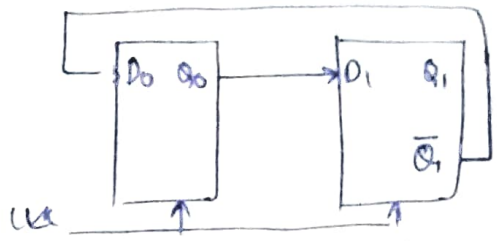
Mod 4 gray Counter using D-FF



Q_1	Q_0	Q_{1N}	Q_{0N}	D_1	D_0
0	0	0	1	0	1
0	1	1	1	1	1
1	1	1	0	1	0
1	0	0	0	0	0

$D_1 = Q_0 \bar{Q}_1 + Q_1 Q_0 = Q_0$

$D_0 = \bar{Q}_1 \bar{Q}_0 + \bar{Q}_1 Q_0 = \bar{Q}_1$



→ Johnson's counter for mod 4 gray code.

→ So, DFF is better than T-FF.

Mod 4 gray counter using 1 T and 1 D Flip Flop

Q_1	Q_0	Q_{1N}	Q_{0N}	D_1	T_0
0	0	0	1	0	1
0	1	1	1	1	0
1	0	0	0	0	0
1	1	1	0	1	1

$D_1 = Q_0$ $\quad T_0 = \bar{Q}_1 \bar{Q}_0 + Q_1 Q_0 = Q_1 \odot Q_0$