Digital Logic & Design (GATE)

1. Boolean Algebra

→ Distributive law : $x+yz = (x+y)(x+z)$

→ Demorgan's law : $\overline{A+B} = \bar{A}\cdot\bar{B}$, $\overline{A+B+C} = \bar{A}\cdot\bar{B}\cdot\bar{C}$

$\overline{AB} = \bar{A}+\bar{B}$, $\overline{ABC} = \bar{A}+\bar{B}+\bar{C}$

Logic Gates

Basic	Universal	Derived
NOT	NAND	XOR
AND	NOR	XNOR
OR		

→ AND, OR, NAND, NOR – single i/p signal controlled gates

Thus not suitable for staircase operati

XNOR → Suitable for staircase operation

→ propagation delay : time taken by a gate to produce o/p from i/p.

→ Bubbled NAND = OR, bubbled NOR = AND

→ Bubbled AND = NOR, bubbled OR = NAND

→ Single bubbled XOR = X-NOR, Single bubbled X-NOR = X-OR

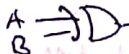
→ Bubbled XOR = XOR, Bubbled X-NOR = X-NOR

Min no of 2ip NAND to get n ip NAND = 2n-3
Silly for NOR !

→ X-OR gate : o/p is high for odd no of 1's (non coincidence gate)

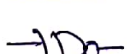
→ X-NOR gate : o/p is high for even no of 1's (coincidence gate)

XOR



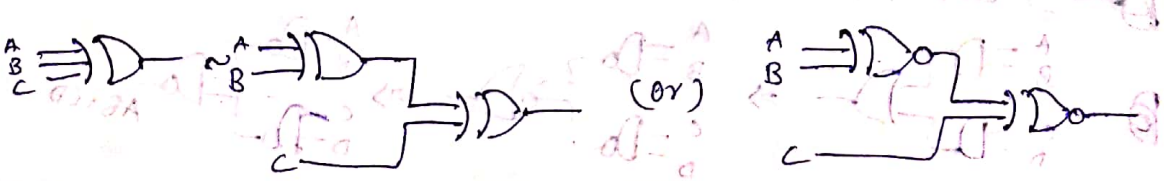
$A \oplus B = \bar{A}B + A\bar{B}$

X-NOR

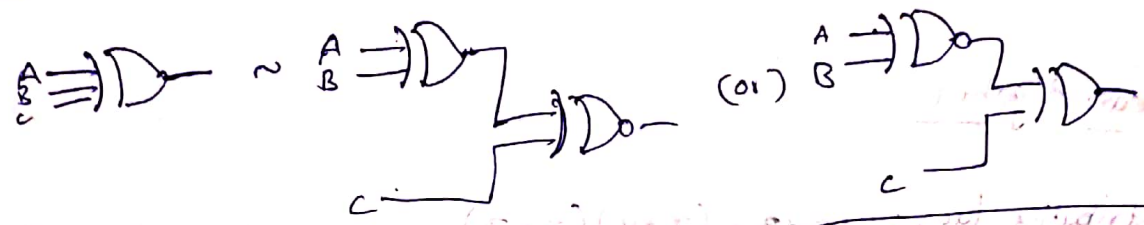


$A \odot B = AB + \bar{A}\bar{B}$

3-i/p Ex-OR



3-i/p X-NOR



fan-in: no of i/p to a gate

fan-out: no of o/p connected to a gate.

operating speed: The max frequency at which digital data can be applied to a gate.

→ Venn diagrams for logic gates

Obtaining expression from given truth table

Four methods:

(i) S.O.P (minterm i.e., 1)
(ii) P.O.S (maxterm i.e., 0)
(iii) K-Map
(iv) Tabulation method

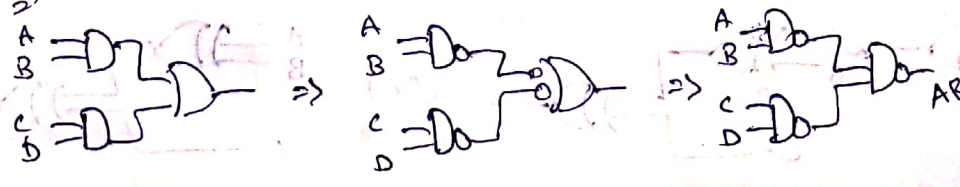
	I/P		Term Designation			
			Σ S.O.P '1'		Π P.O.S. '0'	
0	0	0	$\bar{x}\bar{y}$	m_0	$x+y$	M_0
1	0	1	$\bar{x}y$	m_1	$x+\bar{y}$	M_1
2	1	0	$x\bar{y}$	m_2	$\bar{x}+y$	M_2
3	1	1	xy	m_3	$\bar{x}+\bar{y}$	M_3

→ Sum of all minterms is '1'.
→ Product of all maxterms is '0'.

→ Redundant term: It is a repeated term whose removal does not disturb orignal canonical form.

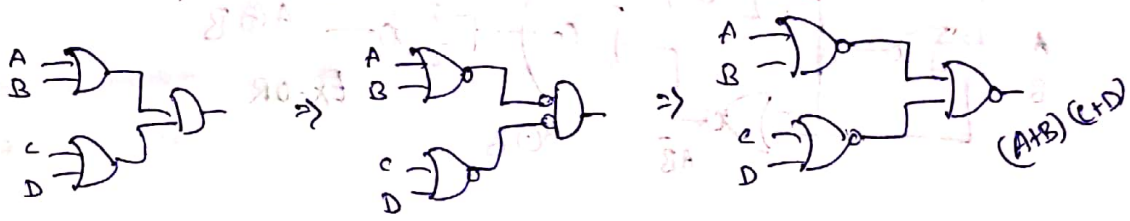
→ Two level SOP expression can be implemented 2-level NAND gates

Eg: Consider AB+CD



→ Two level P.O.S expression can be ~~represented~~ implemented with 2 level NOR gates

Eg: consider (A+B)(C+D)



Dual Function (F^D)

→ Dual function for a given function is generated by replacing '+' with '.' and '.' with '+'. and '0' with '1' and '1' with '0'

→ If all minterms / maxterms produced by the i/p func and dual function are same then the i/p func is called a self Dual Function (SDF)

→ Max no of SDFs that can be implemented with n binary variables = $2^{2^{n-1}}$

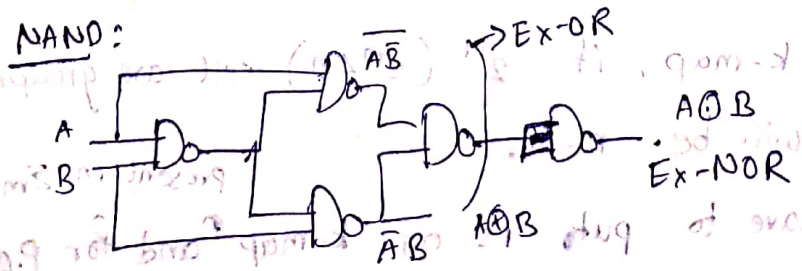
→ Max no of normal boolean func than can be implemented with n binary variables: 2^{2^n}.

Universal Gates

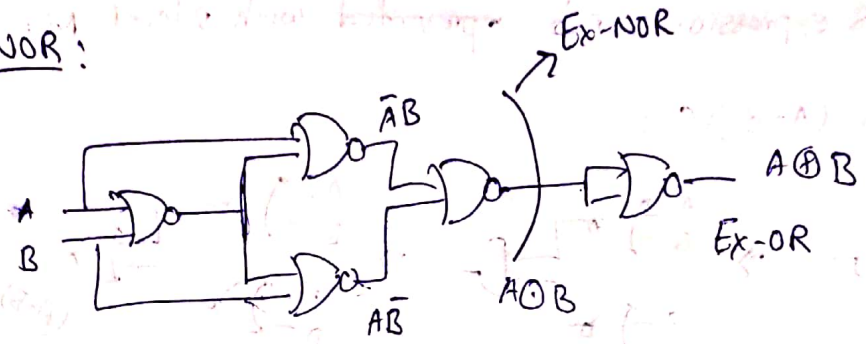
→ Check implementation of NOT, OR, AND, NAND, NOR with NAND & NOR

→ Exclusive gates with NAND/NOR gates:

NAND:



NOR:



→ NAND gates for Ex-OR: 4

NAND gates for Ex-NOR: 5

NOR gates for Ex-OR: 5

NOR gates for Ex-NOR: 4

2. K-Maps:

→ In n-variable k-map, each cell has n chances for grouping with other cells.

→ Std method:

(i) get all PIs (Prime implicants: All possible groupings)

(ii) get EPIs (Essential PI: PIs which are grouped only once)

(iii) if EPIs are not sufficient get selective PIs.

→ k-Maps give minimized expression in most of the cases but not in all the cases.

→ Remember to take 0 as A and 1 as $\bar{A}$ while determing P.O.S expression.

→ In a n-variable k-map, if 2^m ($2^m \leq n$) cell are grouped then term size will be $n-m$.

→ For S.O.P we have to put 1's on k-map present in Σm and for P.O.S we need to put 0's on k-map present in π_M.

i.e, $F(ABC) = \Sigma_m (0,1,5,7)$ (S.O.P) $\Rightarrow F(ABC) = \pi_M (2,3,4,6)$ (P.O.S)

3. Number System:

→ Conversion blw base r and r^n system:

* Direct conversion is possible by replacing each symbol in r^n system with n bits.

Eg: $(24)_8 = (010\ 100)_2$
(12)

$(01101101)_2 = (001101101)_2 = (155)_8$

↳ $(6B)_{16}$

* While converting from r to r^n if bits can be partitioned into n groups, then add zeroes to start of integer part, but for decimal part add at the end.

Eg: $(11010.1101)_2 = (011010.110100)_2 = (32.64)_8$

Fractional data Conversions:

$(26.8125)_{10} = (?)_2$

$(26)_{10} = (11010)_2$

for decimal part

$.8125 \times 2$

1	1.6250×2
1	1.25×2
~~0~~	~~0.5×2~~
0	0.5×2
1	1.0

1101

$\Rightarrow (26.8125)_{10} = (11010.1101)_2$

$(11010.1101)_2 = (?)_{10}$

$\cdots 2^4\ 2^3\ 2^2\ 2^1\ 2^0\ 2^{-1}\ 2^{-2}\ 2^{-3}\ 2^{-4}$
$1\ 1\ 0\ 1\ 0 .\ 1\ 1\ 0\ 1$

* In decimal part power starts from '-1'.

Signed Data Representation:

→ In any representation the data is represented normally.

→ Result for given data is same if 1's or 2's complement is performed twice.

(i) Signed Magnitude Notation:

0000 } to
⋮ } to
0111 } +7

Range: $(-2^{n-1}-1)$ to $(2^{n-1}-1)$

1000 } -0
⋮ } to
1111 } -7

→ It contain an invalid notation i.e., -0

(ii) Signed 1's Complement Notation:

0000 } to
⋮ } to
0111 } +7

Range: $(-2^{n-1}-1)$ to $(2^{n-1}-1)$

1000 } -7
⋮ } to
1111 } -0

→ It also contains -0.

(iii) Signed 2's Complement Notation:

0000 } to
⋮ } +7
0111 }

Range: (-2^{n-1}) to $(2^{n-1}-1)$

1000 } -8
⋮
1111 } -1

→ It permits 8421 weighted code considering M.S.B (sign bit) as -ve weight.

i.e., 1011 → $\overset{-8\ 4\ 2\ 1}{1\ 0\ 1\ 1}$ ⇒ -8+2+1 = -5

→ permits sign bit extension for converting smaller size signed data to larger size

i.e., 1011 = 11011 = 111011 = 1111011 = ...

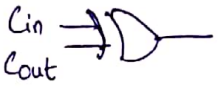
These padded bits are known as dummy bits / Gaurd bits.

Overflow (OVFL) in signed 2's complement Notation:

→ OVFL does not occur on adding one +ve & one -ve data.

→ OVFL inverts the target sign bit. i.e., if target is '1', result sign bit is 0.

Overflow detection:



Cin: Obtained by adding data.

Cout: obtained by adding sign bits.

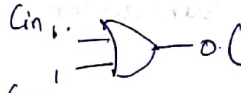
Eg: Cin 1 1 1 1 1 (-1)

1 1 1 1 (-1)

Cout 1 1 1 1 0

↓

Answer



if res is -ve we'll obtain it in 2's comp form.

If it is +ve we'll get in normal form

→ Check examples in notes for better understanding

4. Combinational Circuits (Better to refer notes for perfection)

→ faster, no memory, no clock

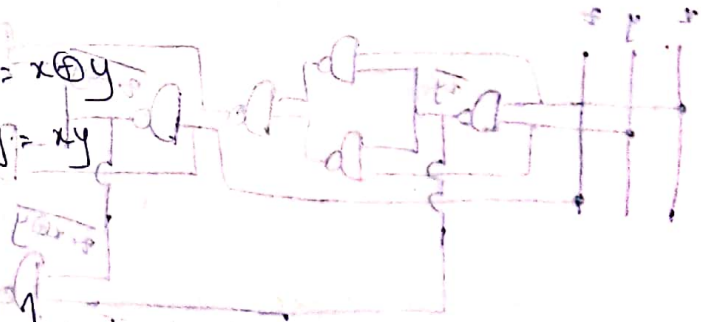
→ Eg: Adders, subtractors, Mux, Demux, Encoder, Decoder etc. magnitude comparator

→ Mux is universal combinational circuit.

Half Adder:

Sum = x ⊕ y

Carry = xy



implementation:

→ 5 NAND gates
→ 5 NOR gates

} try to implement

↳ difficult

Half Subtractor:

Diff = $x \oplus y$

Borrow = $\bar{x}y$

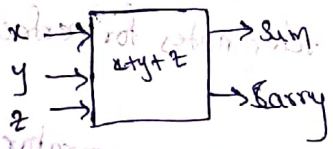
Implementation:

→ 5 NAND gates
→ 5 NOR gates

Note:

☆ For half adder / half subtractor min no of NAND/NOR gates required is 5.

Full Adder:



Sum = $x \oplus y \oplus z$

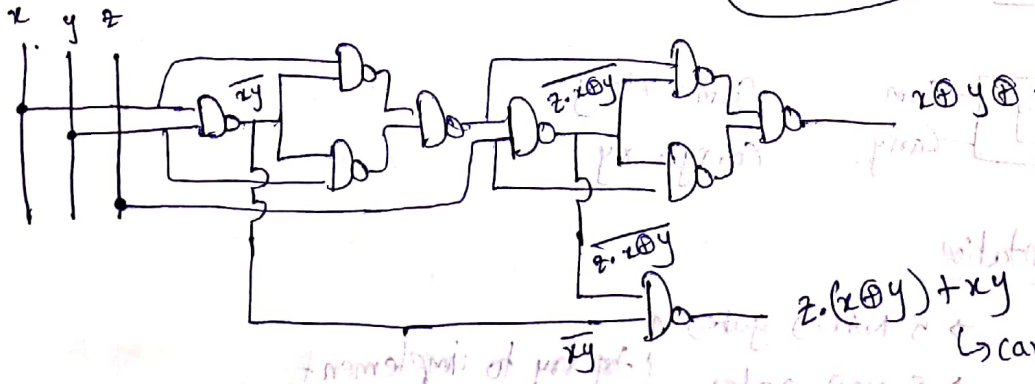
Carry = $z \cdot (x \oplus y) + xy$ [By writing min terms]

$= yz + xz + xy$ [from k-map]

This is used in implementation

implementation with NAND & NOR:

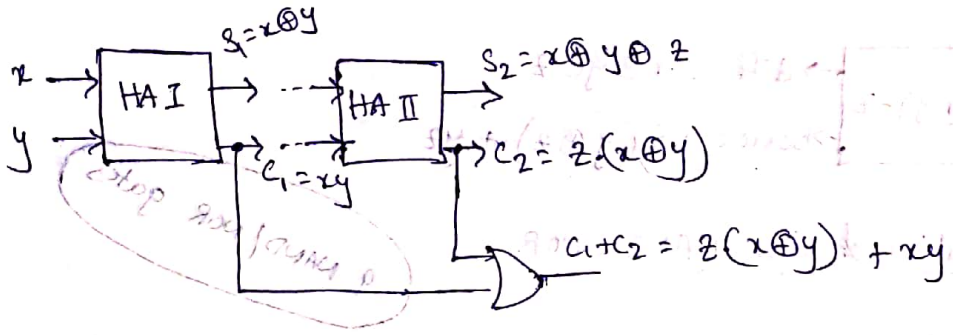
9 NAND/NOR gates



Note:

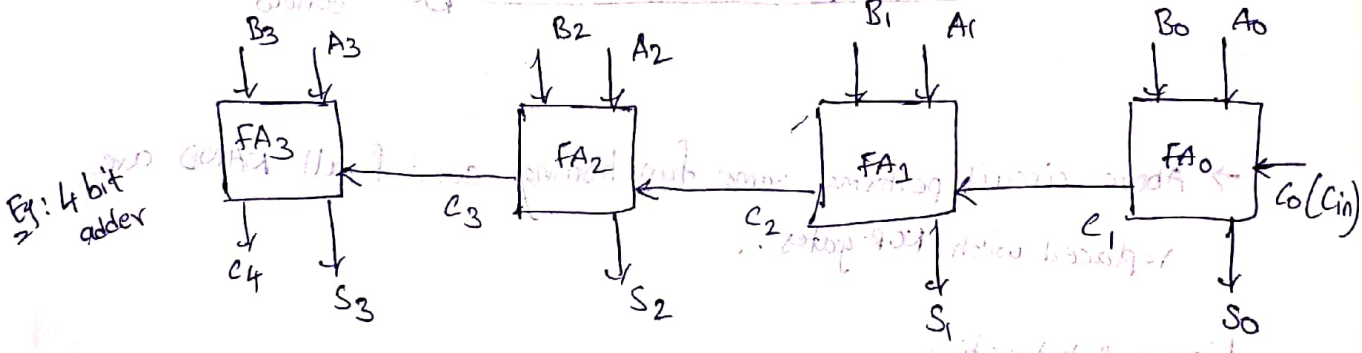
Above circuit Produces same o/p even if we replace all NAND gates with NOR gates.

FA with HA:



* FA is known as one full bit adder
* HA is known as one half bit adder

n-bit adder:



result: $C_4 S_3 S_2 S_1 S_0$

C_{in}: initial carray

$C_1, C_2 \ldots C_{n-1}$: Ripple carries

→ n-1 ripple carries are req to start operation in MSB adder.

→ ~~ripple~~ Total ripple carry time: $(n-1) T_{cy}$

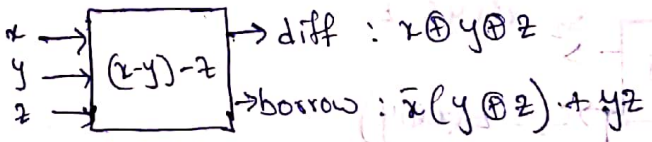
T_{cy}: time taken by FA to produce carry.

→ woost time to stabilize result

$$T_{worst} = [(n-1) T_{cy}] + \max(T_{sum}, T_{cy})$$

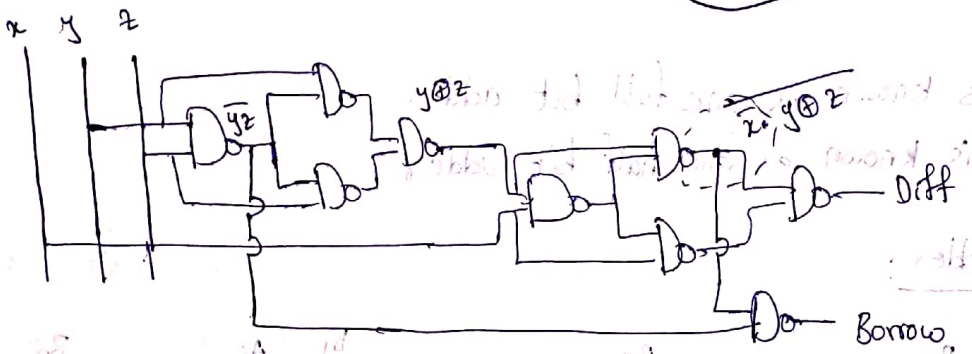
T_{sum}: time taken by FA to produce sum

Full Subtractor:



implementation of with NAND & NOR:

9 NAND/NOR gates



→ Above circuit performs same functioning even if all NAND are replaced with NOR gates.

binary substraction:

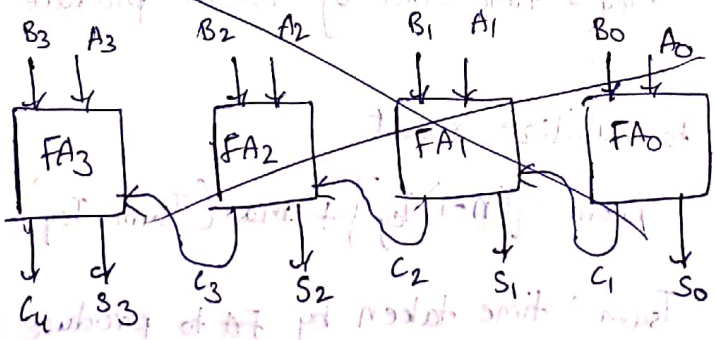
→ A − B = A + B̄ + 1 (i.e., adding 2's comp of B to A)

→ If carray is generated result is +ve & available in normal form.

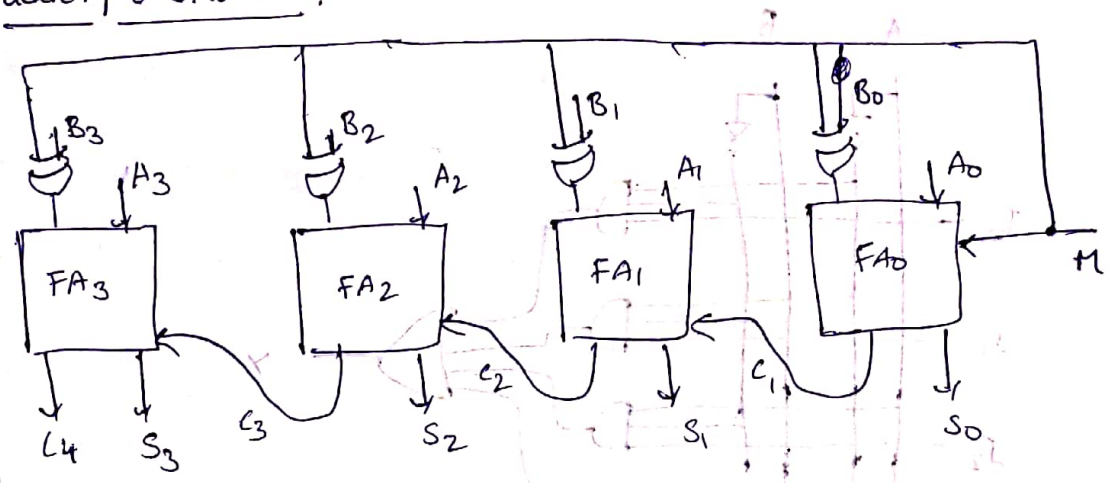
→ If carry is not generated result is -ve & available in 2's comp.

Check Eg if needed

4 bit adder/sub:



4 bit adder / subtractor:



Analyze

M: Mode bit if M=0 → adder
if M=1 → subtractor

$x \oplus 0 = x$
$x \oplus 1 = \bar{x}$

for M=0

$A_3\ A_2\ A_1\ A_0$
$B_3\ B_2\ B_1\ B_0$
$0 (M)$
} adder

for M=1

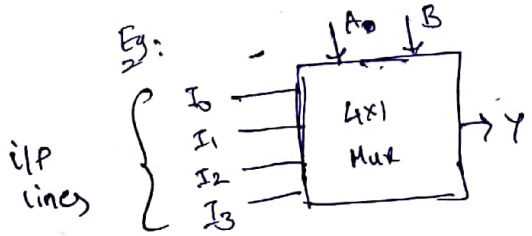
$A_3\ A_2\ A_1\ A_0$
$\bar{B}_3\ \bar{B}_2\ \bar{B}_1\ \bar{B}_0\ (\bar{B})$
} 2's comp } Sub

Multiplexer:

*also known as universal combinational circuit, data selector, many to one selector, parallel to serial convertor

→ Its size is of form $2^n \times 1$

Eg:

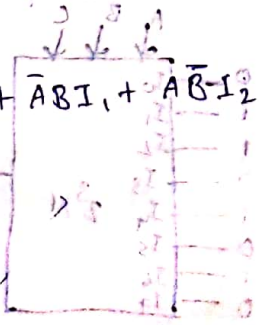


A,B → selection lines

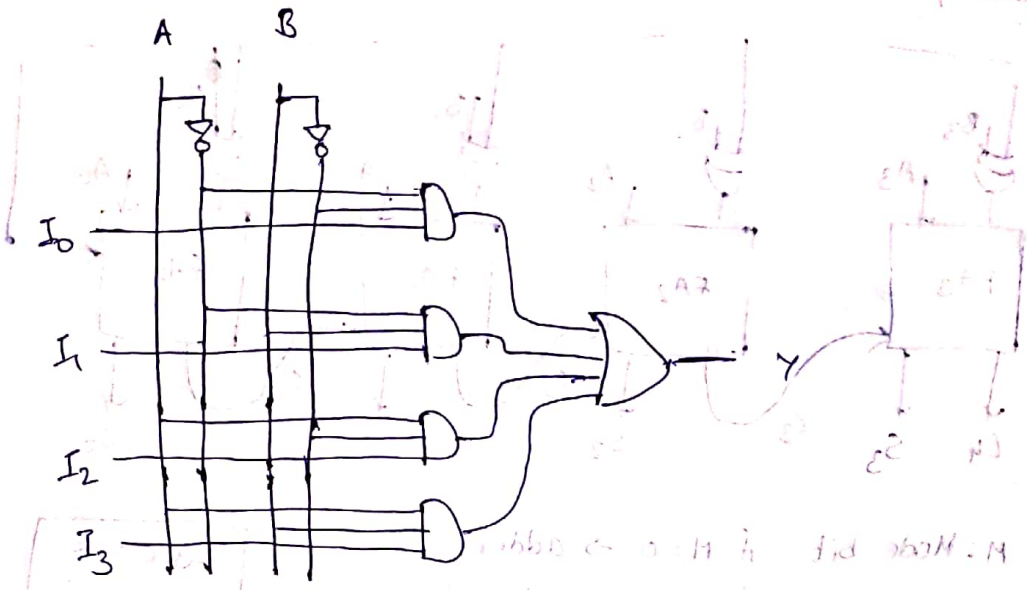
A	B	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

~~$Y = \bar{A}\bar{B}I_0$~~

$Y = \bar{A}\bar{B}I_0 + \bar{A}BI_1 + A\bar{B}I_2 + ABI_3$



implementation:



for $2^n \times 1$ Mux find

No of AND gates req = 2^n

Size of each and gate = n+1

no of NOT gates req = n

no of OR gates req = 1

Size of the OR gate = 2^n

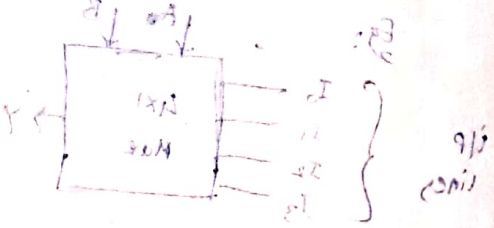
Examples

Mux as Function Selector:

To implement n variable function

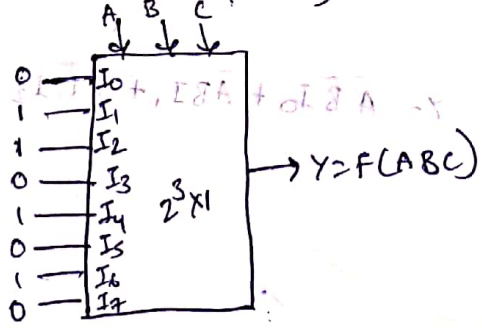
(i) $2^n \times 1$ Mux

(ii) $2^{n-1} \times 1$ Mux



$2^n \times 1$:

Eg: $F(ABC) = \Sigma m(1,2,4,6)$



$2^{n-1} \times 1$ (Req less gates compared to $2^n \times 1$)

* data i/p may be 0's, 1's, free i/p, complement of free i/p

Eg: $F(ABC) = \Sigma m(1,2,4,6)$

if A = free i/p

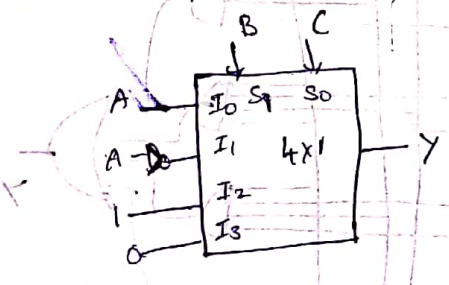
$\bar{A}$	0	(1)	(2)	3
A	(4)	5	(6)	7

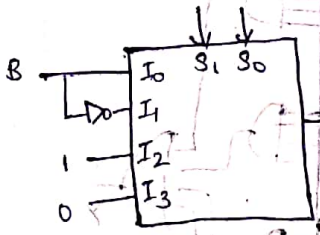
A $\bar{A}$ 1 0

if B = free i/p

$\bar{B}$	0	(1)	(4)	5
B	(2)	3	(6)	7

B $\bar{B}$ 1 0

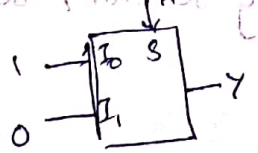




* ~~For~~ clear check notes if you don't understand

Logic Gates with Muxes ($2^{n-1} \times 1$):

NOT gate:

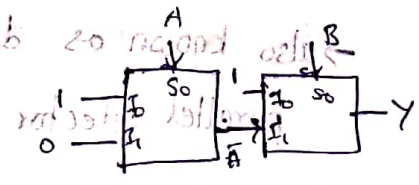


NAND gate

$\bar{A}$	(0)	(1)	
A	(2)	3	

1 $\bar{A}$

↳ Not gate req

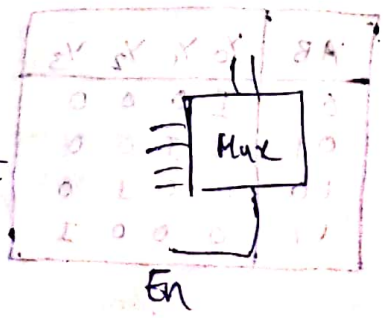


* All basic gates req only 1 / 2x1 Mux
* All universal & derived gates req 2 Muxes.

Enable i/p in Muxes:

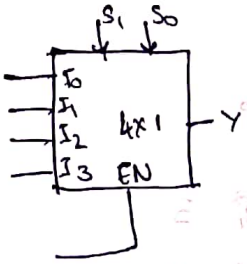
→ Special i/p called enable i/p is used.

→ If mux is disabled, no matter what i/p & selection lines you give, o/p is forced to zero.

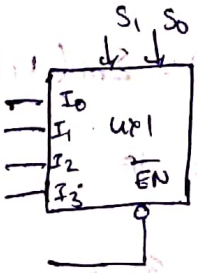


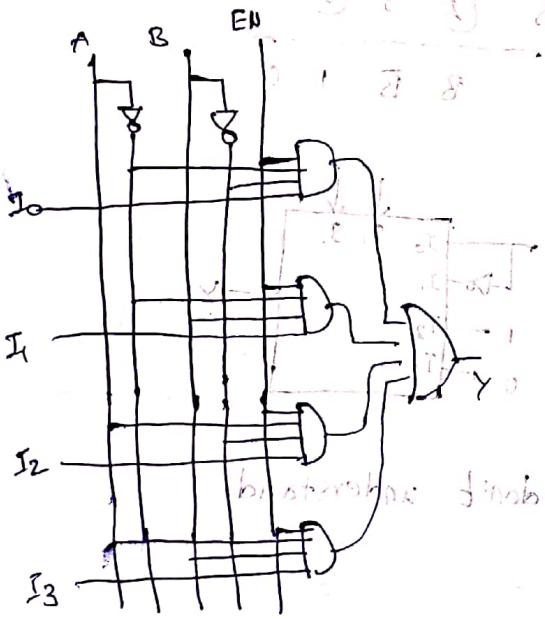
two types of enable i/p:

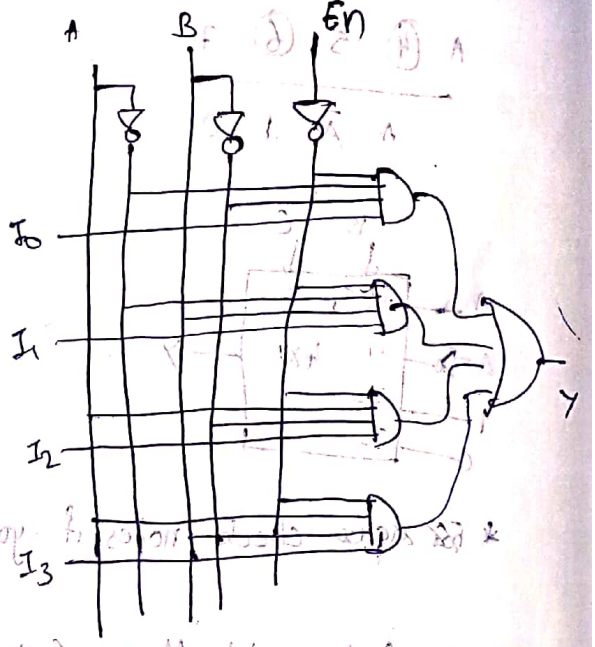
(i) Active high enable



(ii) Active low Enable





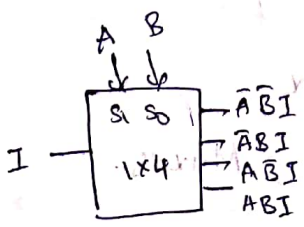


Demux:

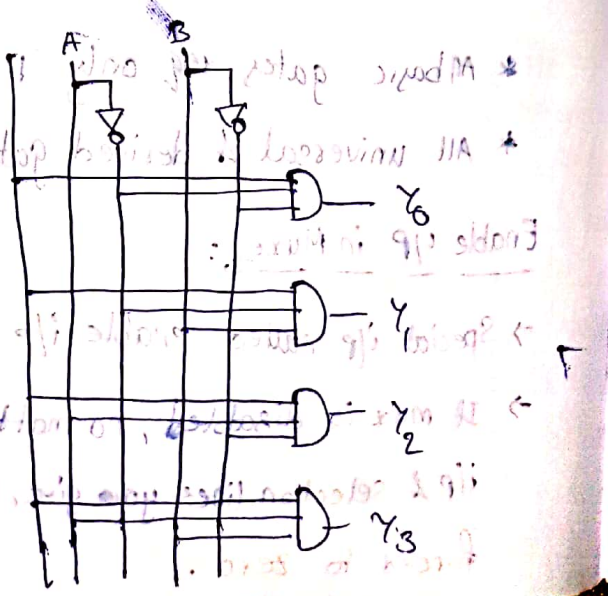
→ also known as data distributor, one to many selector, serial to parallel selector

→ size: 1×2^n

Eg:



AB	Y_0	Y_1	Y_2	Y_3
00	I	0	0	0
01	0	I	0	0
10	0	0	I	0
11	0	0	0	I



Decoder:

→ used for memory addressing & chip selection.

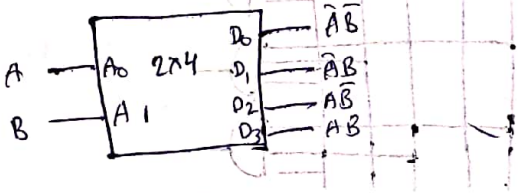
→ No. selection lines, size: $n \times 2^n$

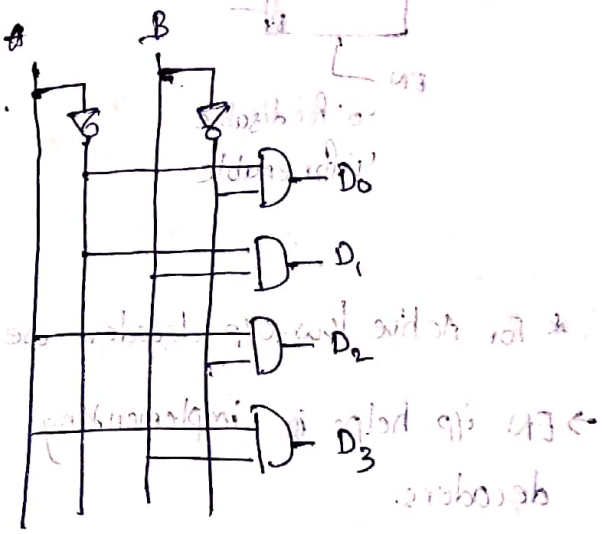
→ two types:

i) Active high o/p decoder → only 1 o/p bit is high

ii) Active low o/p decoder → only 1 o/p bit is low.

Active high o/p decoder





A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

→ Try to do the same above implementation for Active low o/p decoder (use NAND gates)

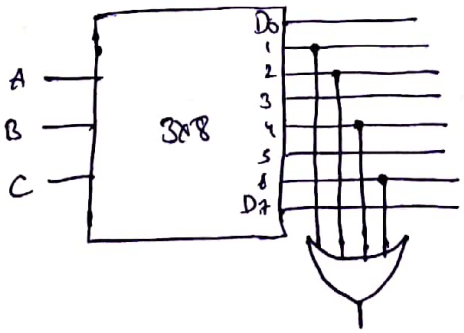
Decoder as function selector:

→ $n \times 2^n$ is used for n variable function

Eg: $F(ABC) = \Sigma m(1,2,4,6)$

ABC	F
000	0
001	1
010	1
011	0
100	1
101	0
110	1
111	0

Active high o/p decoder

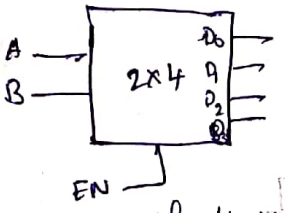


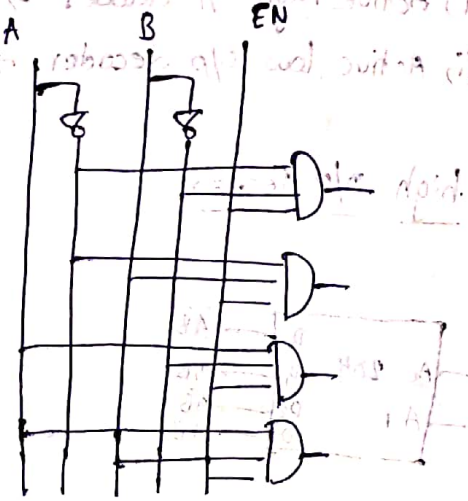
→ For Active low o/p decoder we use NAND gate (think why)

Enable i/p in decoder:

→ If a decoder is disabled
an active high decoder produces '0' o/p in all lines
an active low o/p decoder produces '1' o/p in all lines

Active high o/p decoder

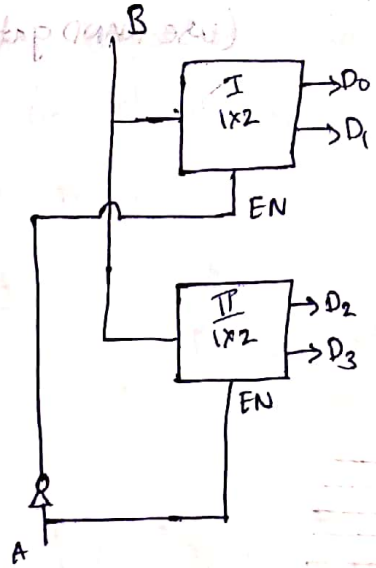




* For Active low o/p decoder we negate EN cuz we use NAND gates.

→ EN i/p helps in implementing larger size decoders with smaller size decoders.

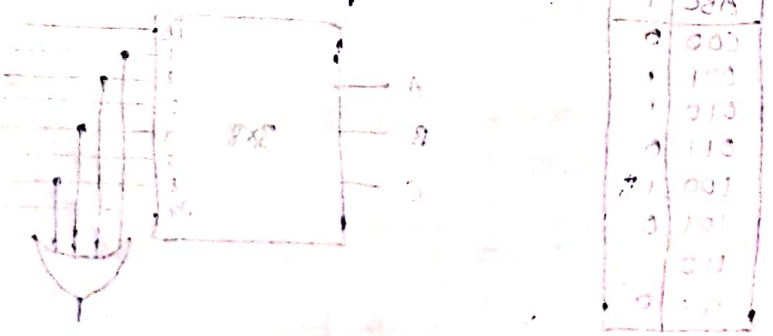
2×4 with 1×2:



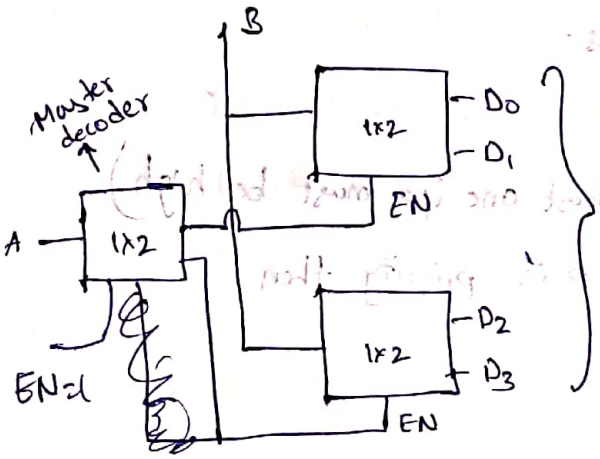
Disabled when A=1

Disabled when A=0

A B	D_0	D_1	D_2	D_3
0 0	1	0	0	0
0 1	0	1	0	0
1 0	0	0	1	0
1 1	0	0	0	1



without using NOT gate :



Eg :

Req : 4x16
avail : 1x2

$\frac{4\times16}{1\times2} = 4 + 8$

No of 1x2 req = 8+4+2+1 = 15

Req : 3x8
avail : 1x2

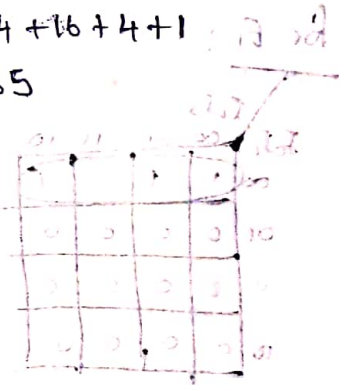
$\frac{8}{2} = 4$

No of 1x2 req = 4+2+1 = 7

Req : 16x256
avail : 2x4

$\frac{256}{4} = 64$ (From this divide it by 4 until you get 1)

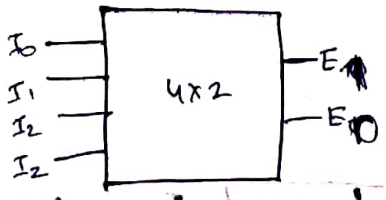
No of 2x4 req = 64+16+4+1 = 85



Encoder :

→ Reverse of decoder , size : $2^n \times n$

→ If two or more i/p line have high signal then it is invalid i/p.

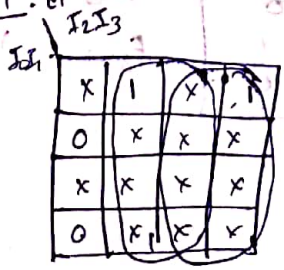


I_0	I_1	I_2	I_3	E_1	E_0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

$E_1 = I_2 + I_3$

$E_0 = I_1 + I_3$

for E_1 :



Priority Encoder

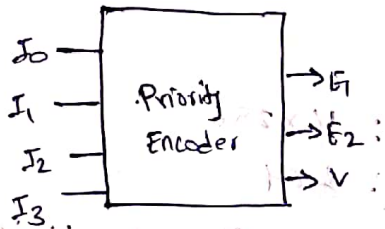
→ Permits one or more high i/p signals.

→ Consists of special o/p bit (V).

→ O/p is valid only if V=1 (i.e., atleast one i/p must be high)

→ If i/p lines are I_0, I_1, I_2, I_3 and P is priority then

$$P_{I_0} > P_{I_1} > P_{I_2} > P_{I_3}$$



	I_0	I_1	I_2	I_3	E_1	E_0	V
	0	0	0	0	x	x	0 (invalid)
(8 to 15)	1	x	x	x	0	0	1
(4 to 7)	0	1	x	x	0	1	1
(2 to 3)	0	0	1	x	1	0	1
(1)	0	0	0	1	1	1	1

for E_1 :

I_0I_1 \ I_2I_3	00	01	11	10
00	x	1	1	1
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

$E_1 = \bar{I}_0 \bar{I}_1$

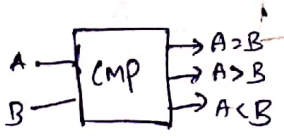
Sily find E_2

$V = I_0 + I_1 + I_2 + I_3$

Magnitute Comparator:

→ Use to compare two magnitudes

one bit emp :



A	B	A=B	A<B	A>B
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1
1	1	1	0	0

$F_{A=B} = A \odot B$

$F_{A<B} = \bar{A}B$

$F_{A>B} = A\bar{B}$

silly find out how design a 2-bit comparator.

For a n-bit comparator

Total no of combinations $= 2^{2n}$

No of A=B combinations $= 2^n$

No of A>B combinations $= \frac{2^{2n} - 2^n}{2}$

No of A<B Combination $= \frac{2^{2n} - 2^n}{2}$ } same

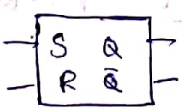
5. Sequential Circuits

→ Seq circuit is a clocked ckt and o/p arrival time depends on frequency of the clock pulse.

→ If clock i/p is '0', no changes are present at the o/p.

→ For seq ckt reliable operation, max clock frequency must be less than $1/t_{pd}$. t_{pd} → ckt propagation delay.

SR latch :

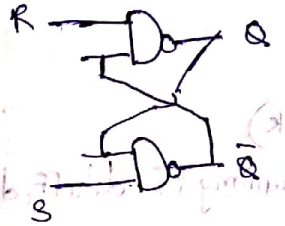


'S' sets Q

'R' reset Q

(o/p is invalid (x) if Q = Q̄)

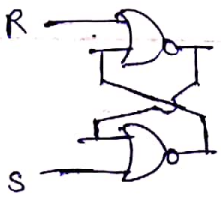
SR NAND latch



S	R	Q
0	0	X → invalid
0	1	0
1	0	1
1	1	N.C

N.C → No change

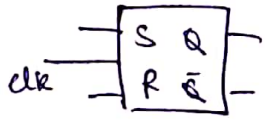
SR NOR latch



S	R	Q
0	0	N.C
0	1	0
1	0	1
1	1	X

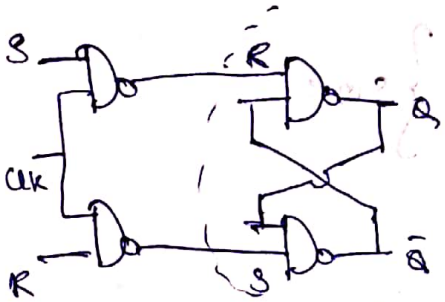
SR Flipflop

→ Obtained by connecting a clock to SR latch



S	R	Q
0	0	N.C
0	1	0
1	0	1
1	1	x

S	R	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	x



Q_{t+1} → Next state

Q_t → Present state

→ To get all the above o/p in table the clk must be high otherwise o/p will be unchanged irrespective of i/p

Jk Flipflop

J	k	Q_{t+1}	
0	0	Q_t	Same as SRFF
0	1	0	
1	0	1	
1	1	$\bar{Q}_t$	→ Toggle

Single i/p Flipflops

JKFF

TFF (J=k)

DFF (J≠k)

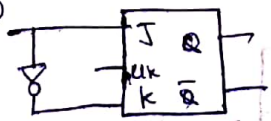
(direct FF, dummy FF, data FF, dumbFF)

T

T	J	k	Q_{t+1}
0	0	0	Q_t
1	1	1	$\bar{Q}_t$

T	Q_{t+1}
0	Q_t
1	$\bar{Q}_t$

D



D	J	k	Q_{t+1}
0	0	1	0
1	1	0	1

D	Q_{t+1}
0	0
1	1

FlipFlop Tables

(i) General tables: Tables that were written so far.

(ii) Characteristic tables: Tells relation blw Q_t & Q_{t+1}

(iii) Excitation tables: Tells what i/p to be given to get desired Q_{t+1} from given Q_t

Characteristic Tables:

S	R	Q_t	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	x
1	1	1	x

$Q_{t+1} = S + \bar{R}Q_t$

J	k	Q_t	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

$Q_{t+1} = J\bar{Q}_t + \bar{k}Q_t$

Characteristic Equations

D	Q_t	Q_{t+1}
0	0	0
0	1	0
1	0	1
1	1	1

$Q_{t+1} = D$

T	Q_t	Q_{t+1}
0	0	0
0	1	1
1	0	1
1	1	0

$Q_{t+1} = T \oplus Q_t$

Excitation Tables:

Q_t	Q_{t+1}	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

Q_t	Q_{t+1}	J	k
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Q_t	Q_{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Counter

two types:
- Synchronous Counter
- Asynchronous Counter (Ripple Counter)

→ used to count states and each state is available for one clock cycle time.

→ If n bit counter is used to count all 2^n states, it is called binary counter, else it is mod counter.

→ Mod k counter is used to count k no of states.

→ Design of Mod k counter is expensive, cuz it req more gates.

Design of Synchronous Counter:

3 bit upcounter (binary counter)

Q_2	Q_1	Q_0	T_2	T_1	T_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	1	1	1
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	0	0	1
1	1	1	1	1	1

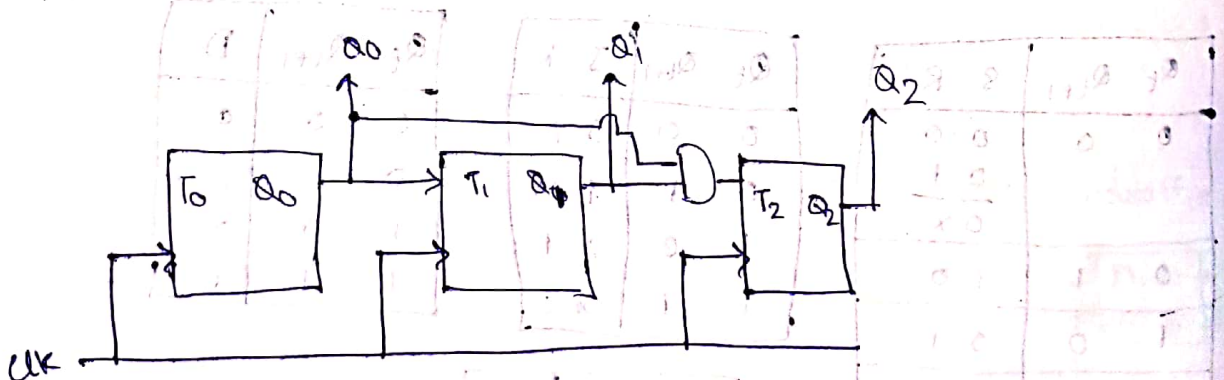
$T_0 = 1$

$T_1 = Q_0$

$T_2 = Q_1 Q_0$

if you are designing n bit binary upcounter then we can use general formula for $T_i = Q_{i-1} Q_{i-2} \cdots Q_1 Q_0$

Eg: $T_5 = Q_4 Q_3 Q_2 Q_1 Q_0$



Mod 6 upcounter

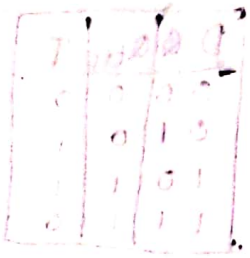
Mod 6 — 0,1,2,3,4,5 (∴ we req 3 bits)

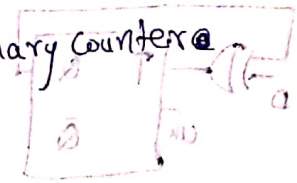
Q_2	Q_1	Q_0	T_2	T_1	T_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	1	1	1
1	0	0	0	0	1
1	0	1	1	1	1
1	1	0	x	x	x
1	1	1	x	x	x

$T_2 = Q_0(Q_1+Q_2)$

Slly T_1 & T_2

Now design a seq ckt as it was designed for 3-bit binary counter.





→ When it is asked to design a counter for given states check if any state is repeated. If no state is repeated then design process is as you have seen above. But if any state is repeated then follow below procedure.

Eg: Design counter for 0,1,2,1,3

To distinguish two 1's we used extra flipflop (dummy ff)

Q_D	Q_1	Q_0	T_D	T_1	T_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	1	1	1
1	0	1	1	1	0
0	1	1	0	1	1

$T_D = Q_0 + Q_1\overline{Q_0}$

$T_1 = Q_0 + Q_1$

$T_0 = \overline{Q_D}$

Flipflop Conversions:

→ Any FF can be implemented with a given FF.

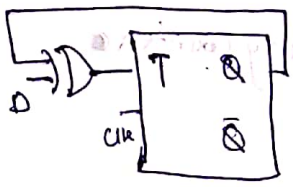
→ For this, first write characteristic table for target FF.

→ Then write excitation table for available FF.

DFF using TFF

D	Q	Q_{t+1}	T
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

$T = D \oplus Q$

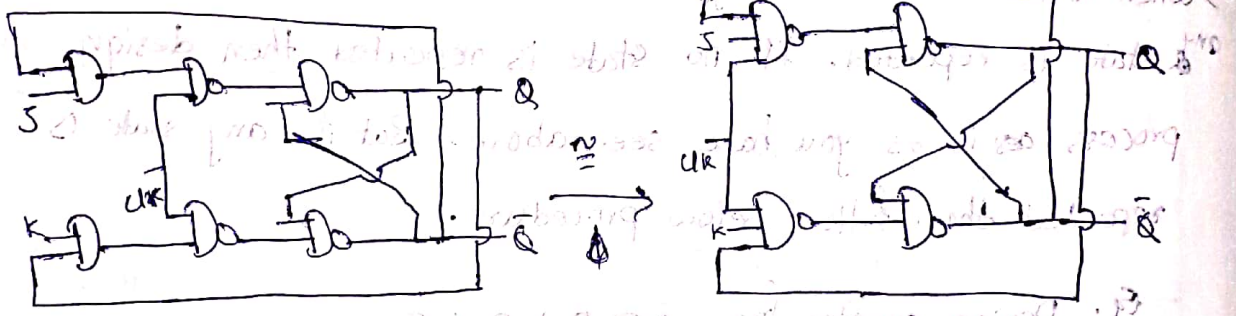


JKFF using SRFF

J	k	Q_t	Q_{t+1}	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

$S = J\bar{Q}$

$R = kQ$



→ Observe above fig and understand.

→ In 2nd fig which represents JKFF, if we remove 3i/p NAND gates and place 2ip NAND gate then the figure represents SRFF.

→ JKFF : two 3 i/p NAND, two 2 i/p NAND

SRFF : four 2 i/p NAND.

Flipflop Triggering:

→ It process of applying high clock signal at desired time.

→ 4 techniques

(i) +ve level trigger
(ii) -ve level trigger
(iii) +ve edge trigger
(iv) -ve edge trigger

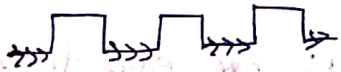
→ RAC occurs in level triggered FF, hence edge triggered are used most.

→ To use edge triggered FF, we ~~need~~ have to use pulse trainer.

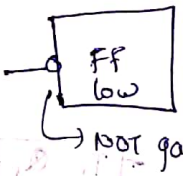
+ve level trigger:

→ Enabled during high signal.

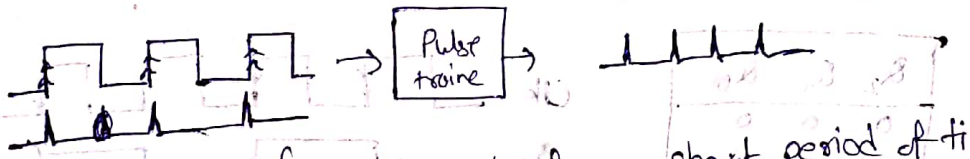
-ve level trigger



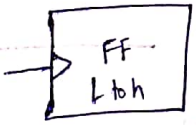
→ Enabled during low signal.



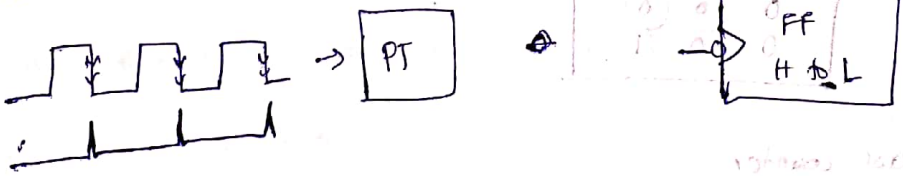
+ve edge trigger



→ Enabled once for each cycle for a short period of time

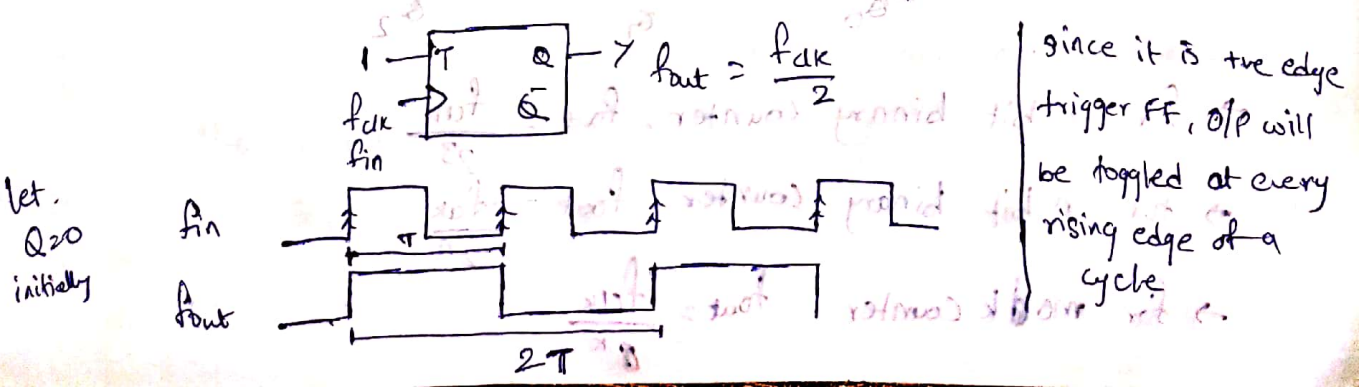


-ve edge trigger



TFF:

→ It known as frequency divider or mod 2 counter.



since it is +ve edge trigger FF, O/P will be toggled at every rising edge of a cycle.

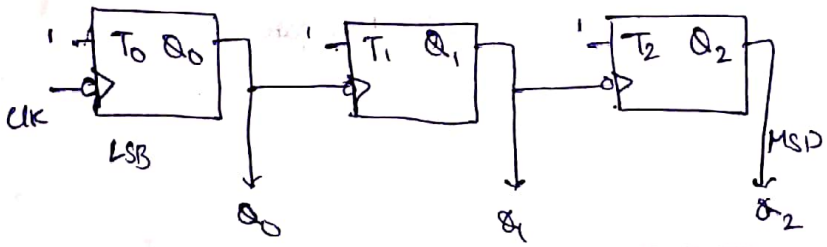
Design of Asynchronous Counter (Ripple Counter):

→ Main clock generator is connected to LSB FF only.

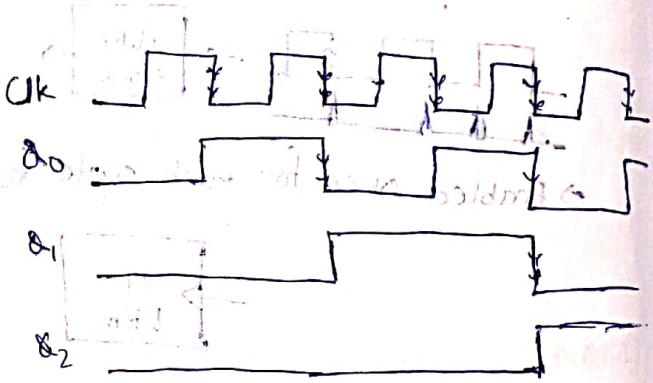
→ Design cost is cheaper and is used for freq division application.

→ up counter: -ve edge trigger
down counter: +ve edge trigger

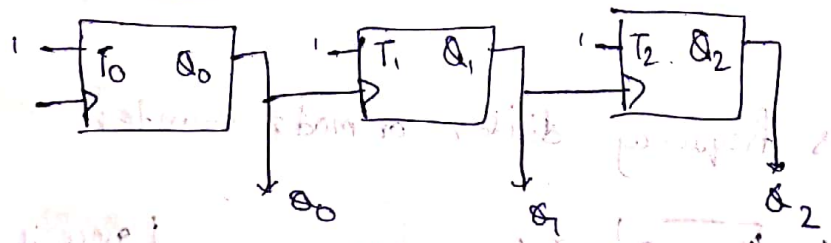
3bit up counter



Q2	Q1	Q0
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



3 bit down counter



→ for 3 bit binary counter, $f_{out} = \frac{f_{clk}}{2^3}$

→ for n bit binary counter $f_{out} = \frac{f_{clk}}{2^n}$

→ for mod k counter $f_{out} = \frac{f_{clk}}{2^k}$

Design of asynchronous mod counters:

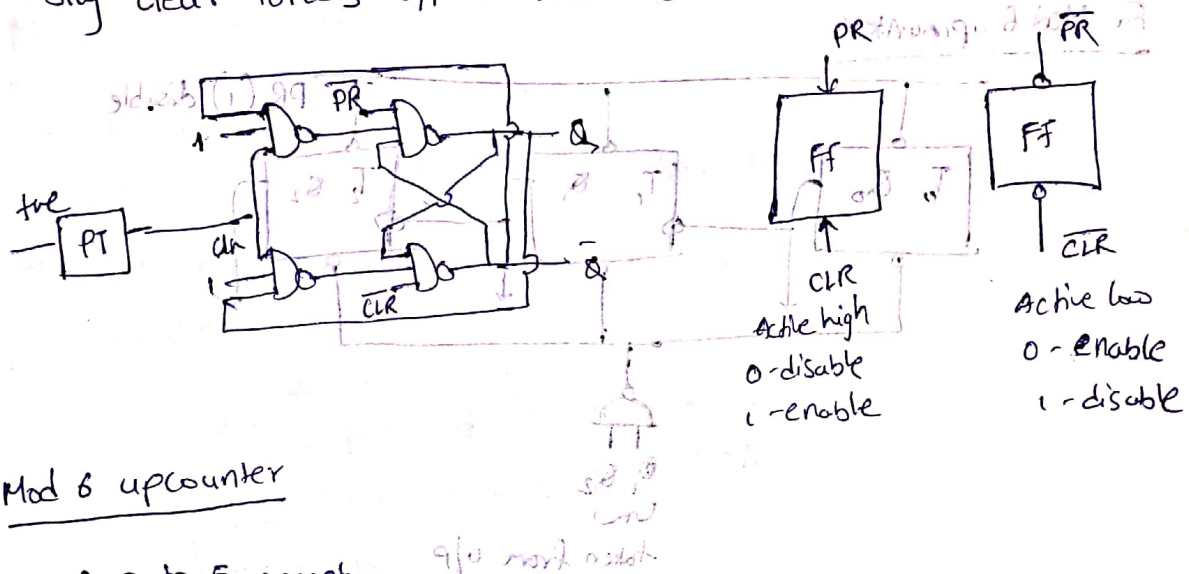
→ we use two special i/p

(i) preset (pr) (ii) clear (clr)

→ ~~preset~~ Both are generally disabled.

→ when preset is enabled it forces o/p to high regardless of i/p and clock and hence used in down counter.

→ Slly clear forces o/p to low and hence is used in upcounters.



Mod 6 upcounter

→ 0 to 5 accept

$6 = (110)_2$
$Q_2 Q_1 Q_0$

to we have to stop counting exactly at 6.

Binary value of 6 has 2 1's at Q_2 & Q_1.

For active low enable $\overline{CLR}$ should be given i/p 0 when 6 occurs.

Hence we connect $Q_2 Q_1$ to NAND gate and connect it to $\overline{CLR}$.

for 6 $\overline{Q_2 Q_1} = \overline{11} = 0 = \overline{CLR}$

$\overline{CLR} = 0$ Hence enable and o/p set to 000.

Also we can use OR gate in place of NAND gate and give

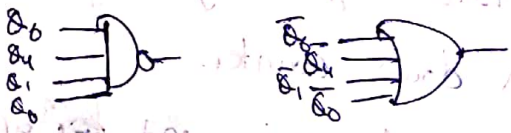
i/p $\overline{Q_2}$ $\overline{Q_1}$

Mod 83 upcounter :

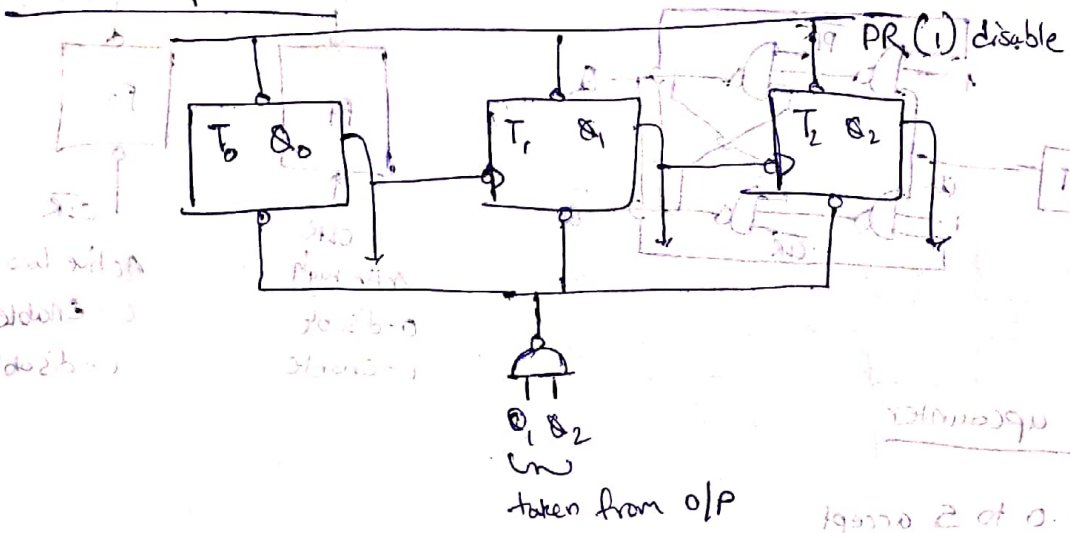
→ 0 to 82 accept

$83 = (1010011)_2$
$Q_6\ Q_5\ Q_4\ Q_3\ Q_2\ Q_1\ Q_0$

4 1's. Hence 4 ip NAND or OR is req



For Mod 6 upcounter



Race Around Condition (RAC):

→ when FF o/p is toggled more than once in one clk cycle then we say RAC has occured.

→ RAC is diff from toggling. RAC is uncontrolled toggling.

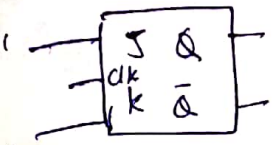
→ SRFF & DFF are free from RAC.

→ RAC never occurs in edge trigged FFs.

→ So RACO occurs only in level triggered JKFF & level triggered TFF.

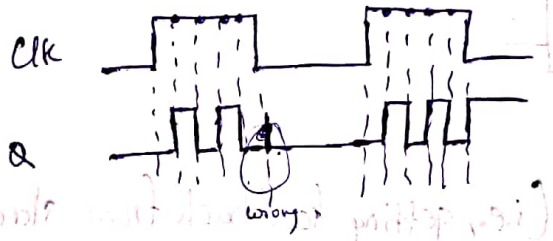
→ Master slave FF is used to prevent RAC.

Eg for RAC:



Let $t_{pd} = 1ns \Rightarrow f_{pd} = 1GHz$

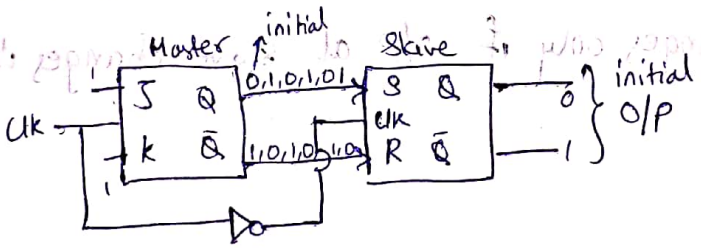
$t_{clk} = 10ns \Rightarrow f_{clk} = 0.1GHz$ or $100MHz$

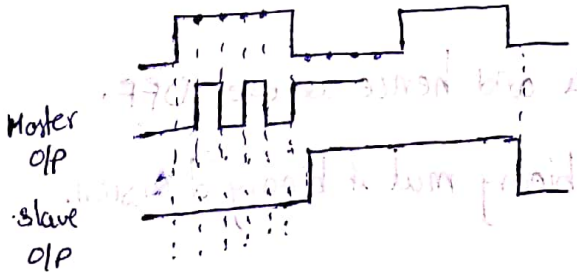


Master-Slave JkFF:

→ Master FF: JkFF
Slave FF: JkFF or SRFF (gets i/p from Master FF & gives final o/p)

→ when clock is high master is enabled & Slave is disabled and vice versa



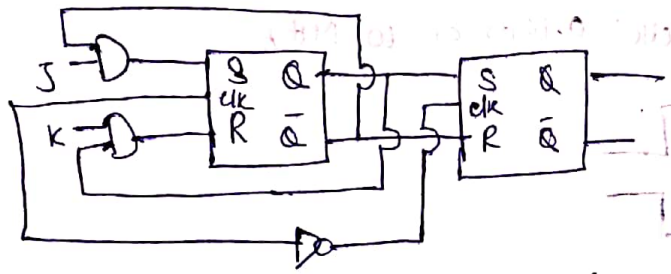


→ But the above circuit works good only if Master toggles o/p for odd no of times. If it toggles even no of times i/p for Slave remains same all time it is enabled and hence o/p is same i.e., straight line.

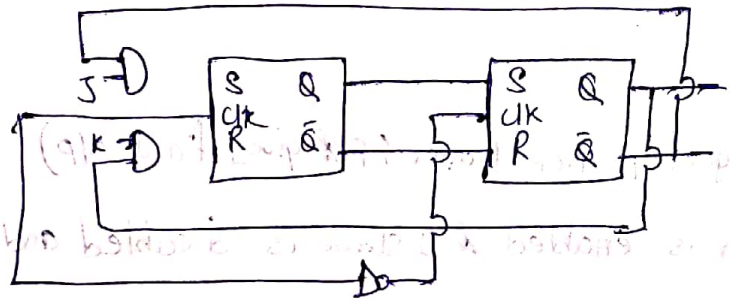
→ So we modify it as shown below so that o/p at Master toggles odd no of times (i.e., one time)

Modification:

The previous diagram can be seen as



↓ Modification (i.e., getting feedback from slave instead of getting it from itself)



→ Now i/p at JK changes only if o/p at slave changes i.e., only once for clock cycle.

Registers:

→ Used to save applied data and hence we use DFF.

→ Data shifting can perform binary mul & binary division.

→ types of shifting:

(i) Serial in Serial out.

(ii) Serial in Parallel out (most used in real time)

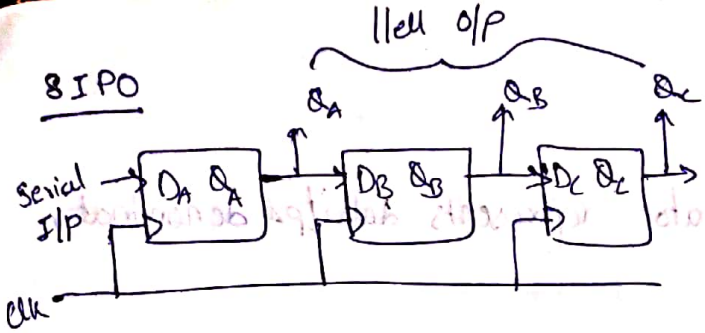
(iii) Parallel in Serial out

(iv) Parallel in Parallel out.

→ 010 (2) → left shift → 100 (4)

→ 100 (4) → right shift → 010 (2)

SIPO



D_A = Serial I/P
$D_B = D_A\ (Q_A)$
$D_C = D_B\ (Q_B)$

(observe diff in the tables for SIPO, RC, JC)

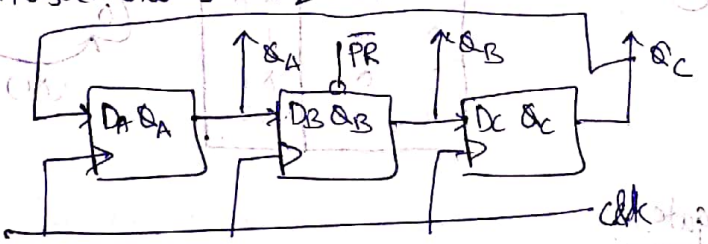
Ring Counter (RC):

→ Same as SIPO except Q_C is connected to serial i/p and one of the flip flops is connected with preset.

→ only one o/p bit is high and hence it counts n states

→ Useful states → n

→ unused states → $2^n - n$



$D_A = Q_C$
$D_B = Q_A (Q_A)$
$D_C = Q_B (Q_B)$

D_A D_B D_C	clk	Q_A Q_B Q_C
initial		0 1 0
0 0 0	↑	0 0 1
1 0 0	↑	1 0 0
0 1 0	↑	0 1 0

→ $\overline{PR}$ is used to make Q_B '1' initially.

Johnson Counter (JC):

→ Also known as Twisted RC, Ringtail Counter, Moebius counter.

→ No preset is used.

→ Useful states : 2n, Unused states : 2^n

→ Same as RC except Q_C is toggled and connected to D_A and PR is not used

D_A D_B D_C	Clk	Q_A Q_B Q_C
initial		0 0 0
1 0 0	↑	1 0 0
1 1 0	↑	1 1 0
1 1 1	↑	1 1 1
0 1 1	↑	0 1 1
0 0 1	↑	0 0 1
0 0 0	↑	0 0 0

$D_A = \overline{Q_C}$
$D_B = D_A$ or Q_A
$D_C = D_B$ or Q_B

State table & State diagrams:

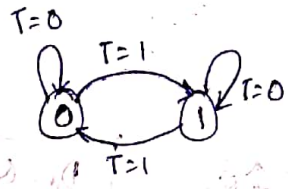
→ One transition arrow numerator represents data i/p & denominator represents function o/p.

→ To draw state diagram of n bit binary counter we req. 2^n states. (try to draw).

State diagrams for FFs

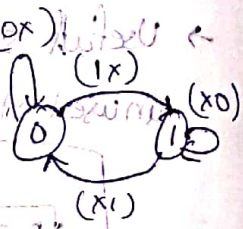
TFF

Q	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0



JKFF

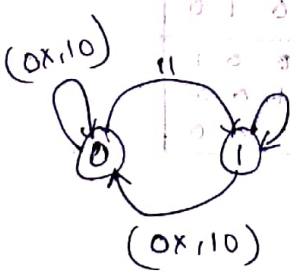
Q_t	Q_{t+1}	Jk
0	0	0X
0	1	1X
1	0	X1
1	1	X0



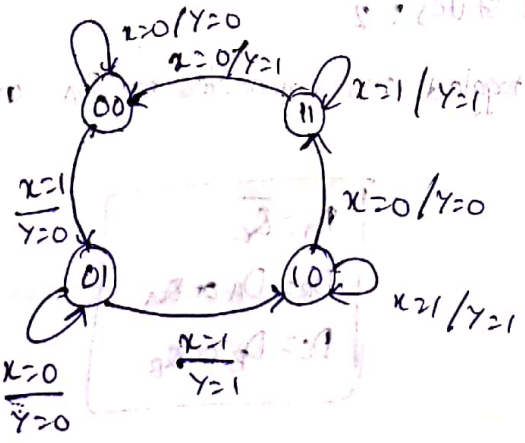
State diagrams for logic gates:

AND gate

Q_t	Q_{t+1}	AB
0	0	(0X,10)
0	1	11
1	0	(0X,10)
1	1	11



Eg: Design logic diagram for below state diagram with TFF

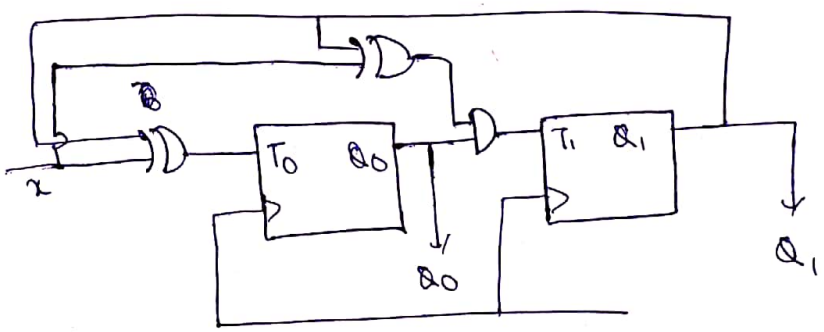


P.S Q_t			N.S Q_{t+1}				
Q_1	Q_0	x	Q_1	Q_0	T_1	T_0	Y
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	0	1	0	0	0
0	1	1	1	0	1	1	1
1	0	0	1	1	0	1	0
1	0	1	1	0	0	0	1
1	1	0	0	0	1	1	1
1	1	1	1	1	0	0	1

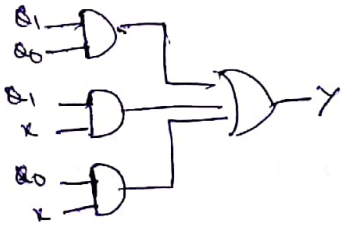
from fig — excitation table — from fig

$T_1 = Q_0(Q_1 \oplus x)$

$T_0 = Q_1 \oplus x$



for O/P Y connect below ckt to the above ckt



→ The sequential synchronous circuit is also known as Finite State Machine (FSM).