Number System

There are 4 number systems:

i.) **Decimal Number System:**
* Includes 10 digits: (0, 1, 2, ..., 9)
* Base = 10

ii) **Binary Number System:**
* Includes 2 digits: (0 and 1)
* Base = 2

iii) **Octal Number System:**
* Includes 8 digits: (0, 1, 2, ..., 7)
* Base = 8

iv.) **Hexadecimal Number System:**
* Also called as alphanumeric number system
* Includes 16 characters: (0, 1, ..., 9, A, B, ..., F); where A ~ 10 and B ~ 11 and so on.
* Base = 16

nb: **Base**: It is the number of digits present in a particular number system.
: It is also called as radix

Conversion from one system to other:

Decimal to Binary:

Divide the number by 2 continously and write the remainders until you get 1. Then write the remainders from bottom to top.

eg: $(96)_{10} = (?)_2$

$\therefore (96)_{10} = (1100000)_2$

2 | 96
2 | 48 → 0
2 | 24 → 0
2 | 12 → 0
2 | 6 → 0
2 | 3 → 0
1 → 1

Eg: $(123)_{10} = (?)_2$

$\therefore = (1111011)_2$

2|123
2|61 → 1
2|30 → 1
2|15 → 0
2|7 → 1
2|3 → 1
1 → 1

Binary to Decimal:

From left to right and give index no. starting from zero. Then the formula for decimal equivalent is

$$\sum \text{bitvalue} \times 2^n \quad ; n = \text{bit position of the corresponding bitvalue}$$

Eg:

$2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$ ← bit weightage
$6\ \ 5\ \ 4\ \ 3\ \ 2\ \ 1\ \ 0$ ← bit position

$(1\ 1\ 1\ 0\ 1\ 1\ 1)_2$

$= (1\times 2^6 + 1\times 2^5 + 1\times 2^4 + 0\times 2^3 + 1\times 2^2 + 1\times 2^1 + 1\times 2^0)_{10}$

$= (64 + 32 + 16 + 0 + 4 + 2 + 1)_{10} = (119)_{10}$

Fractional Decimal to Binary:

First evaluate left part of decimal as done before. For right part multiply 2 and continously write the part before decimal point. Follow this until a continum is reached.

Eg: $(10.7)_{10} = (?)_2$

2|10
2|5 → 0
2|2 → 1
1 → 0

$0.7 \times 2 = 1.4 \rightarrow 1$
$0.4 \times 2 = 0.8 \rightarrow 0$
$0.8 \times 2 = 1.6 \rightarrow 1$
$0.6 \times 2 = 1.2 \rightarrow 1$
$0.2 \times 2 = 0.4 \rightarrow 0$

So Ans = $(1010.10110)_2$

Eg: $(14.3)_{10} = (?)_2$

$= (1110.01001)_2$

2|14
2|7 → 0
2|3 → 1
1 → 1

$0.3 \times 2 = 0.6 \rightarrow 0$
$0.6 \times 2 = 1.2 \rightarrow 1$
$0.2 \times 2 = 0.4 \rightarrow 0$
$0.4 \times 2 = 0.8 \rightarrow 0$
$0.8 \times 2 = 1.6 \rightarrow 1$
0.6

Fractional binary to decimal :

Here left past of decimal point is analysed the same, but for right past apply the same with index starting from 1 and using formula $\sum$ bit value $\times 2^{-n}$, n = bit position

$2^3\ 2^2\ 2^1\ 2^0\ 2^{-1}\ 2^{-2}\ 2^{-3}$
$3\ 2\ 1\ 0\ 1\ 2\ 3$ ← bit position

Eg: $(1101.101)_2$

$= (1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3})_{10}$

$= 8 + 4 + 0 + 1 + 0.5 + 0 + 0.125$

$= (13.625)_{10}$

Eg:

Decimal to Octal :

Devide the decimal number by 8 until we get a number less than 8 in quotient past and write the remainders.

Eg: $(319)_{10} = (?)_8$

8 | 319
8 | 39 → 7
4 → 7

$\therefore$ Ans $= (477)_8$

Eg: $(471)_{10} = (?)_8$

8 | 471
8 | 58 → 7
7 → 2

$= (727)_8$

Eg: $(512)_{10} = (?)_8$

8 | 512
8 | 64 → 0
8 | 8 → 0
1 → 0

$= (1000)_8$

Eg: $(946)_{10} = (?)_8$

8 | 946
8 | 118 → 2
8 | 14 → 6
1 → 6

$= (1662)_8$

Eg: $(1049)_{10} = (2031)_8$

8 | 1049
8 | 131 → 1
8 | 16 → 3
2 → 0

Octal to Decimal:

form right to left give index numbers stasting from 0 and then use the formula $\sum (\text{bit value}) \times 8^n$, n = index value's position

Eg: $(727)_8 = 7\times8^2 + 2\times8^1 + 7\times8^0$

$= 448 + 16 + 7 = (471)_{10}$

Eg: $(477)_8 = 4\times8^2 + 7\times8^1 + 7\times8^0 = 256 + 56 + 7 = (319)_{10}$

Eg: $(1000)_8 = 1\times8^3 + 0\times8^2 + 0\times8^1 + 0\times8^0 = (512)_{10}$

Eg: $(1662)_8 = 1\times8^3 + 6\times8^2 + 6\times8^1 + 2\times8^0 = 512 + 384 + 48 + 2 = (946)_{10}$

Eg: $(2031)_8 = 2\times8^3 + 0\times8^2 + 3\times8^1 + 1\times8^0 = 1024 + 24 + 1 = (1049)_{10}$

Decimal to Hexadecimal

Deride by 16 to the decimal number and write the remainder until a quotient less than 16 is obtained.

Use 10 → A, 11 → B, 12 → C, 13 → D, 14 → E, 15 → F in remainders

Eg: $(319)_{10} = (?)_{16}$

16 | 319
16 | 19 → 15 → F
1 → 3

Ans = $(13F)_{16}$

Eg: $(471)_{10} = (?)_{16}$

16 | 471
16 | 29 → 7
1 → 13 → D

$= (1D7)_{16}$

Eg: $(512)_{10} = (?)_{16}$

16 | 512
16 | 32 → 0
2 → 0

$= (200)_{16}$

Eg: $(946)_{10} = (?)_{16}$

$= (3B2)_{16}$

16 | 946
16 | 59 → 2
3 → 11 → B

Eg: $(1049)_{10} = (?)_{16}$

$= (419)_{16}$

16 | 1049
16 | 65 → 9
4 → 1

Hexadecimal to Decimal

Give index values from right to left and then use the formula. $\sum$ (bit value) $\times 16^n$, n = bit value's position. but use converse of A, B, C, D, E & F as 10, 11, 12, 13, 14 & 15 respectively

Eg: $(13F)_{16} = 1\times16^2 + 3\times16^1 + 15\times16^0 = 256 + 48 + 15 = (819)_{10}$

Eg: $(1D7)_{16} = 1\times16^2 + 13\times16^1 + 7\times16^0 = 256 + 208 + 7 = (471)_{10}$

Eg: $(200)_{16} = 2\times16^2 + 0\times16^1 + 0\times16^0 = (512)_{10}$

Eg: $(3B2)_{16} = 3\times16^2 + 11\times16^1 + 2\times16^0 = 768 + 176 + 2 = (946)_{10}$

Eg: $(419)_{16} = 4\times16^2 + 1\times16^1 + 9\times16^0 = 1024 + 16 + 9 = (1049)_{10}$

Binary Addition:

rules: $0+0=0$, $0+1=1$, $1+0=1$, $1+1=10$ i.e 0 with 1 as carry

Eg:
$$\begin{array}{r} (1010)_2 \\ +(0111)_2 \\ \hline (10001)_2 \end{array}$$

Eg:
$$\begin{array}{r} (1110)_2 \\ +(1100)_2 \\ \hline (11010)_2 \end{array}$$

Eg:
$$\begin{array}{r} (10110)_2 \\ +(10110)_2 \\ \hline (1000010)_2 \end{array}$$

Eg:
$$\begin{array}{r} (10011)_2 \\ +(1001)_2 \\ \hline (11100)_2 \end{array}$$

Eg:
$$\begin{array}{r} (1001)_2 \\ +(1101)_2 \\ \hline (10110)_2 \end{array}$$

Eg:
$$\begin{array}{r} (11110)_2 \\ +(11100)_2 \\ \hline (111010)_2 \end{array}$$

Binary Subtraction:

Rules: $0-0=0$, $1-0=1$, $1-1=0$, $0-1=1$ with borrow 1 and add 2

Eg:
$$\begin{array}{r} 10011 \\ -\ 1001 \\ \hline (1010)_2 \end{array}$$

Eg:
$$\begin{array}{r} (1100)_2 \\ -(1101)_2 \\ \hline (1100)_2 \end{array}$$

Eg:
$$\begin{array}{r} (11110)_2 \\ -(11100)_2 \\ \hline (10)_2 \end{array}$$

Eg:
$$\begin{array}{r} (10110)_2 \\ -(10110)_2 \\ \hline (10100)_2 \end{array}$$

Eg
$$\begin{array}{r} (1110)_2 \\ -(1100)_2 \\ \hline (10)_2 \end{array}$$

Binary Multiplication

Rules: $0\cdot0=0$, $0\cdot1=0$, $1\cdot0=0$, $1\cdot1=1$

Eg:
$$\begin{array}{r} (1001)_2 \\ \times (100)_2 \\ \hline 0000 \\ 0000 \\ 1001 \\ \hline (100100)_2 \end{array}$$

Eg:
$$\begin{array}{r} (1101)_2 \\ \times (1001)_2 \\ \hline 1101 \\ 0000 \\ 0000 \\ 1101 \\ \hline (1110101)_2 \end{array}$$

Eg:
$$\begin{array}{r} (11011)_2 \\ \times (100)_2 \\ \hline 00000 \\ 00000 \\ 11011 \\ \hline (1101100)_2 \end{array}$$

Eg:
$$\begin{array}{r} (10110)_2 \\ \times (1111)_2 \\ \hline 10110 \\ 10110 \\ 10110 \\ 10110 \\ \hline (10100101010)_2 \end{array}$$

[illegible] 100 0 with co

Eg:
$$\begin{array}{r} (1110)_2 \\ (110)_2 \\ \hline 0000 \\ 1110 \\ 1110 \\ \hline (1010100)_2 \end{array}$$

Binary Division:

Rule : $0 \div 0 = 0$, $0 \div 1 = 0$, $1 \div 1 = 1$

Eg: $(1100)_2 \div (11)_2$

$$11 \overline{)1100} \quad (100)$$
$$-11$$
$$0000$$

$\therefore$ Ans = $(100)_2$

Eg: $(1010)_2 \div (11)_2$

$$11 \overline{)1010} \quad (11)$$
$$-11$$
$$100$$
$$-11$$
$$1$$

Ans: $(11)_2$ with remainder $(1)_2$

Eg: $(11010)_2 \div (111)_2$

$$111 \overline{)11010} \quad (11)$$
$$-111$$
$$1100$$
$$-111$$
$$101$$

= $(11)_2$ with remainder $(101)_2$

LOGIC GATES:

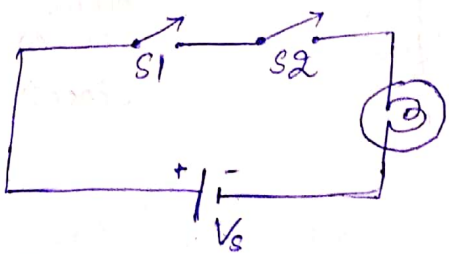
*These are the basic building blocks of a digital system.

- Fundamental Gates (AND, OR, NOT, XOR)
- Universal Gates. (NAND, NOR)

Fundamental Gates:

(a) AND Operation:

* It represents two switches in series combination.



* Truth Table :

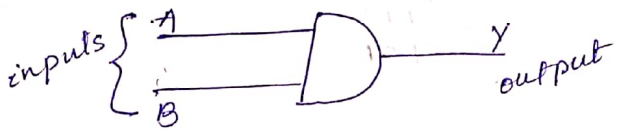
S1	S2	Bulb
open (0)	open (0)	not glow (0)
open (0)	close (1)	not glow (0)
close (1)	open (0)	not glow (0)
close (1)	close (1)	glow (1)

0 ≈ low
1 ≈ high

* In case of AND operation when both the switches are closed, bulb will glow and in any other cases bulb will not glow.

AND Gate :

* It has two or more inputs and one output.
* It is same as the two switches in series combination
* Output is 1 only when all the inputs are 1 (or high) otherwise output is 0 for all other combination of inputs.
* Symbol :



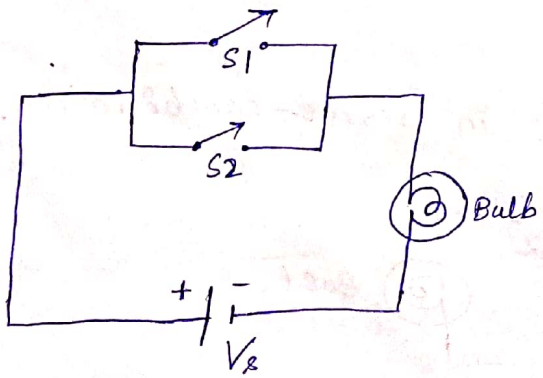
$Y = A \cdot B$

Truth Table :

Input		output
A	B	$Y = A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

(b) OR Operation :

* Represents two switches in parallel combination



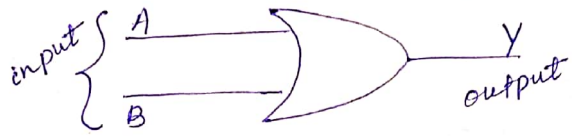
Truth Table :

S1	S2	Bulb
open (0)	open (0)	not glow (0)
open (0)	close (1)	glow (1)
close (1)	open (0)	glow (1)
close (1)	close (1)	glow (1)

* In this operation bulb will glow when both the switches are closed or any one of the switches is closed
* Bulb will not glow when both switches are open.

OR Gate:

* It has two or more inputs and one output
* Represents two switches in ||el combination.
* It is also called as inclusive OR-gate.
* Its output is high when all the inputs are high or any one of the inputs is high; otherwise output is low.
* Symbol:



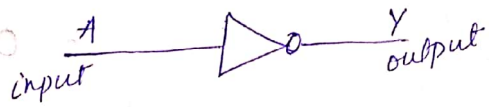
$Y = A + B$

Truth Table

input		output
A	B	Y = A+B
0	0	0
0	1	1
1	0	1
1	1	1

(c) NOT Gate:

* It has one input and one output.
* Output represents the compliment of the input, i.e, if input is 0 then output is 1 and vice-versa.
* Symbol:

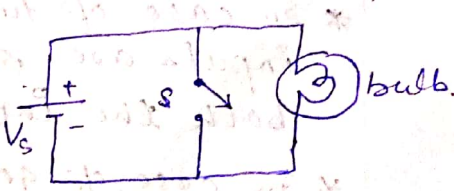


$Y = \bar{A}$ or $Y = A'$

Truth Table

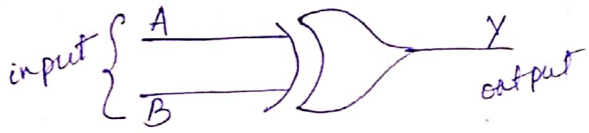
input	output
A	$Y = \bar{A}$
0	1
1	0

* In an electronic circuit it represents a switch connected parallel to a bulb or a shorted switch



(d) XOR Gate / EXOR Gate:

* It has two or more inputs and one output.
* It performs binary operations (addition).
* Its output is high when it contains odd number of 1's in the input.
* Or in other ways, output will be high when both the inputs are different and low when they are same; in case of 2 inputs.
* Output will be low when there exists even number of 1's in input
* Symbol:



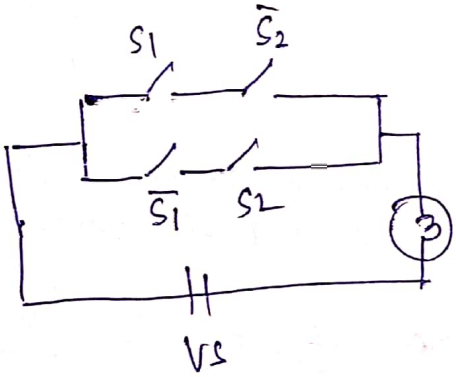
$$Y = A \oplus B$$
$$= \bar{A}B + A\bar{B}$$
$$= (A+B)\cdot(\bar{A}+\bar{B})$$

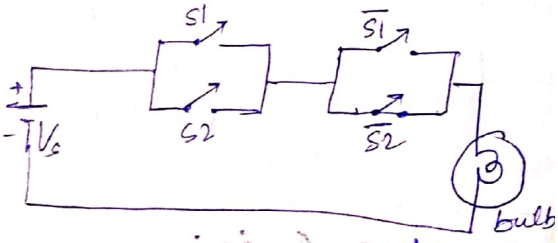
(algebraic equation)

Truth Table.

input		output
A	B	$Y = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

* Its equivalent circuit representation will be like a hybrid network:



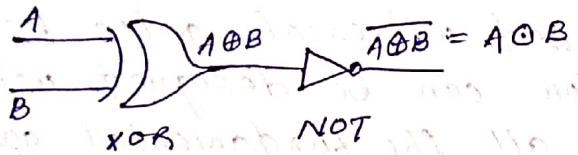


$$(A+B)\cdot(\bar{A}+\bar{B})$$
$$= A\cdot\bar{A} + A\bar{B} + B\bar{A} + B\cdot\bar{B}$$
$$= 0 + A\bar{B} + \bar{A}B + 0$$
$$= A\bar{B} + \bar{A}B$$

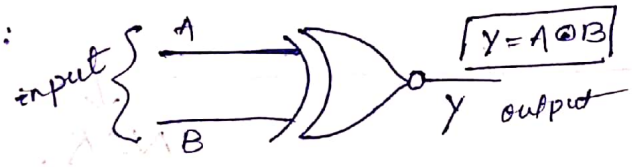
(e) X-NOR Gate: (Extra)

* It has got two or more inputs and one output
* In case of two inputs, output is high when both the inputs are same and output is low (or 0) when both the inputs are different.
* In case of generalised X-NOR gate, output is 1 (or high) if it contains even number of 1s or 0s, and output is 0 when it contains odd number of 1s or 0s.

* It is equivalent to X-OR + NOT, i.e.



* Symbol:



now, $Y = A \odot B$

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

$= (A + \bar{B}) \cdot (\bar{A} + B)$

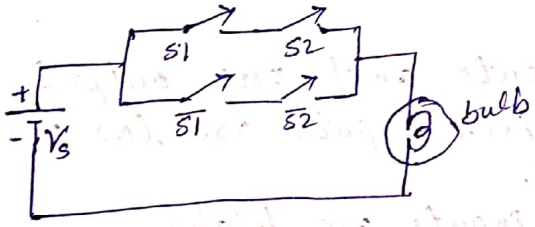
(algebraic equation)

Truth Table

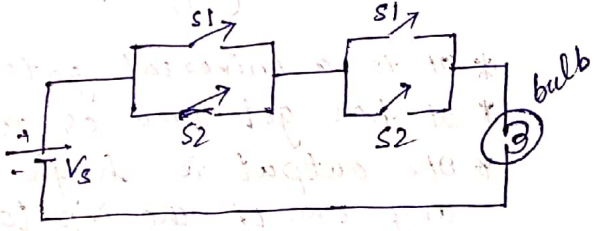
input		output
A	B	$Y = A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

* There can be two equivalent circuit representation

$A \cdot B + \bar{A} \cdot \bar{B}$



$(A + \bar{B}) \cdot (\bar{A} + B)$



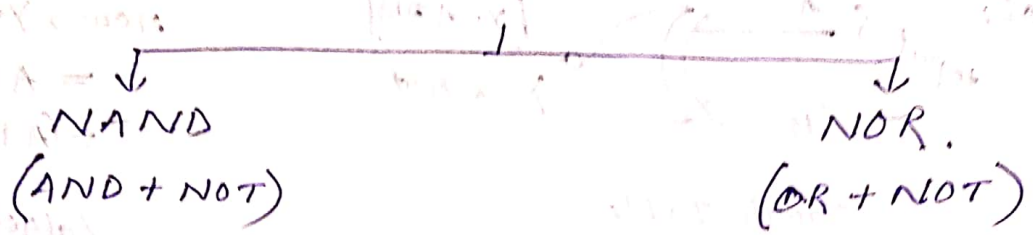
nb: How to find algebraic equations from given truth table.

1. We have to see where '1' is there in the output and ignore whenever output is '0'.
2. To find equation of the operation, add all the individual equations obtained whenever output is 1.
3. Remember while finding individual equations, consider '1' as direct variable and '0' as complemented variable.

Eg: If 'x' is the variable then take $x = 1$ and $\bar{x} = 0$

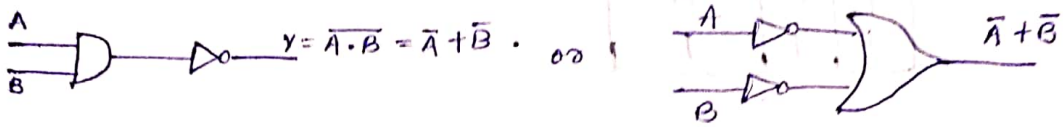
Universal Gate:

* These gates are called universal gates because any boolean expression can be designed using universal gates as well as all the fundamental gates can be designed too.

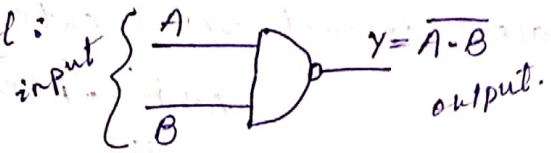


(a) NAND Gate:

= AND + NOT = 2 NOT + OR



* Symbol:

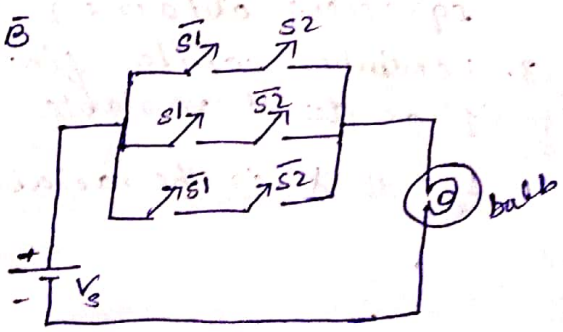


* It is a universal gate
* It has got two or more inputs and one output.
* Its output is high when both inputs are low or when any one of the inputs is low.
* Output is low when all the inputs are high.

Truth Table

Input		Output
A	B	$Y = \overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

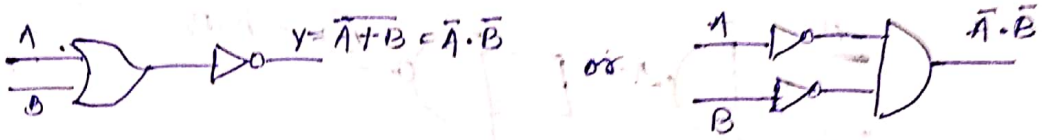
* Algebraic expression: $A\bar{B} + \bar{A}B + \bar{A}\bar{B}$
* The corresponding electronic circuit representation is:



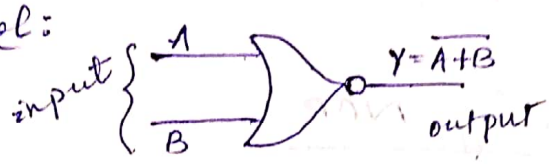
(b) NOR Gate:

* It is a universal gate.
* It has two or more inputs and one output.
* Output is high when all inputs are low and output is low in all other cases.

NOR = OR + NOT = 2NOT + AND



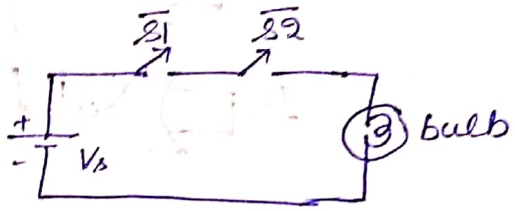
* Symbol:



Truth Table

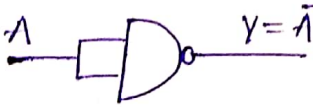
Input		Output
A	B	Y = $\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

* Algebraic expression: $\bar{A}\cdot\bar{B}$
* Electric circuit representation:



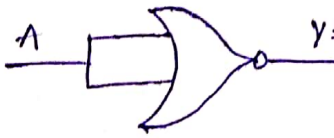
---

i) Design of NOT gate using NAND:



((∵ $X\cdot X = X$

ii) Design of NOT gate using NOR:



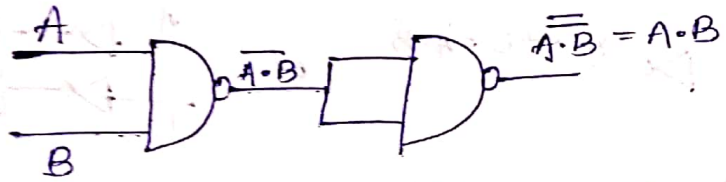
((∵ $X+X = X$

iii) Design of AND gate using NAND:

now we know $NAND(A,B) = \overline{A \cdot B}$

Again, $AND(A,B) = A \cdot B = \overline{\overline{A \cdot B}}$ (∵ $\overline{\overline{X}} = X$)

$= NOT(NAND(A,B))$



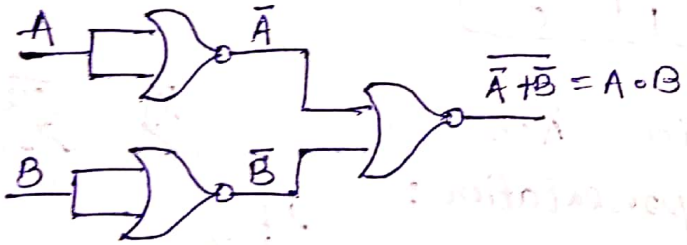
iv) Design of AND gate using NOR:

we know, $NOR(A,B) = \overline{A+B}$

so $AND(A,B) = A \cdot B = \overline{\overline{A} \cdot \overline{B}}$ (∵ $\overline{\overline{X}} = X$)

$= \overline{\overline{A} + \overline{B}}$ (∵ De Morgan's laws)

$= NOR(\overline{A}, \overline{B})$



24.9.19

1's Complement

* complement refers to negative of a number, i.e, for example if 12 is the number then its complement refers to -12
* for calculating 1's complement follow the rule : convert 1→0 and 0→1.

Eg: 1's complement of $(12)_{10}$

$(12)_{10} = (1100)_2$

1's complement = $(0011)_2 = -(12)_{10}$

Addition using 1's complement:

* when addend number is greater than the number to be subtracted then only result is positive and in other cases result will be negative.
* case : (a-b)
  - → If there generates an End Around carry (EAC) then that has to be added to LSB and after addition whatever result obtained is the final result.
  - → If no EAC is generated then answer will be a negative value so to check the result just take the 1's complement of the result which will be the +ve of the result.

Eg: $(7)_{10} - (3)_{10}$

$(7)_{10} = (0111)_2$, $(3)_{10} = (0011)_2 \xrightarrow{1's} (1100)_2 = -(3)_{10}$

so, $(7)_{10} = 0111$

$-(3)_{10} = 1100$ (+)

$(1)\ 0011$ (case 1)

EAC → $+\ 1$

$(0100)_2 = (4)_{10}$ (Ans)

Eg: $(3)_{10} - (7)_{10}$

$(3)_{10} = (0011)_2$, $(7)_{10} = (0111)_2 \xrightarrow{1's} (1000)_2 = -(7)_{10}$

so, $(3)_{10} = 0011$

$-(7)_{10} = 1000$ (+)

$(1011)_2$ (Ans) (case 2)

now $(1011)_2 \xrightarrow{1's} (0100)_2 = (4)_{10}$ so $(1011)_2 = -(4)_{10}$ is right answ

* Case (-a-b)

→ here follow the same rules for EAC addition, but remember here the obtained result will always be negative. Hence to check the result always take the 1's complement of the result which will be a +ve value.

→ Be careful about no. of bit representation to be used

Eg: $-(7)_{10} - (3)_{10}$

$(7)_{10} = (00000111)_2 \xrightarrow{1's} (11111000)_2 = -(7)_{10}$

[here 8 bit is followed as result is -20 where 20>1

$(13)_{10} = (00001101)_2 \xrightarrow{1's} (11110010)_2 = -(13)_{10}$

$-(7)_{10} = 11111000$

$-(13)_{10} = 11110010$ (+)

11101010

EAC → 1

$+ 1$

$(11101011)_2$ (Ans)

now $(11101011)_2 \xrightarrow{1's} (00010100)_2 = (20)_{10}$ = +ve of value obtain

2's Complement:

* Here also complement means the same, i.e, negative of the number taken.
* Rules: everything for logic part is same as that in 1's complement about the cases, except the fact that when an EAC is encountered it is not added rather ignored.
* Logic: 2's complement = 1's complement + 1., i.e, find 1's compl first and then add 1 to it.
  : OR other method is scan or copy all bits from LSB till '1' appears and then make complements of the rest. bits.

Eg: $(7)_{10} - (3)_{10}$:

$(7)_{10} = (0111)_2$, $(3)_{10} = (0011)_2 \xrightarrow{2's} (1101)_2 = -(3)_{10}$

$(7)_{10} = 0111$

$-(3)_{10} = 1101$ (+)

10100

EAC is ignored.

$(0100)_2 = (4)_{10}$ (Ans)

Eg: $(3)_{10} - (7)_{10}$

$(3)_{10} = (0011)_2$, $(7)_{10} = (0111)_2 \xrightarrow{2's} (1001)_2 = -(7)_{10}$

$(3)_{10} = 0011$

$-(7)_{10} = 1001$

$(1100)_2$ (Ans)

now $(1100)_2 \xrightarrow{2's} (0100)_2 = (4)_{10}$ = +ve value of answer.

so Ans is correct.

Eg: $-(7)_{10} - (3)_{10}$

$(7)_{10} = (0111)_2 \xrightarrow{2's} (1001)_2 = -(7)_{10}$, $(3)_{10} = (0011)_2 \xrightarrow{2's} (1101)_2 = -(3)_{10}$

$-(7)_{10} = 1001$

$-(3)_{10} = 1101$ (+)

0110

(1) EAC is ignored

$(0110)_2$ (Ans)

now $(0110)_2 \xrightarrow{2's} (1010)_2 = (10)_{10}$ = +ve of value obtained

Hence ans is correct

Binary Codes:

* Binary codes are simply the codes of numeric or alpha-numeric numbers in form of '0' and '1'.
* It refers to forms of representing binary codes.

- Weighted Code
  - In such codes every bit-position has some weight or value
  - Eg: 8421-code
    : 5421-code
    : BCD-code
- Non-weighted code
  - In such codes every bit position doesn't possess any value or weight
  - Eg: Grey code
    : Excess-3 code.

Weighted Code :

8421 - code

* It means 4-bit representation of binary codes, i.e, from 0 to 15.
* Leftmost side is MSB (value = 8) and rightmost side is LSB (value = 1)

Decimal No.	Binary Equivalent $B_3(8)$	$B_2(4)$	$B_1(2)$	$B_0(1)$
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
⋮	⋮	⋮	⋮	⋮
15	1	1	1	1

5421 - code

* It means 4-bit representation of binary codes from 0 to 12
* Leftmost value = 5 = MSB , rightmost value = 1 = LSB

Decimal No.	$B_3(5)$	$B_2(4)$	$B_1(2)$	$B_0(1)$
0	0	0	0	0
1	0	0	0	1
⋮	⋮	⋮	⋮	⋮
12	1	1	1	1

BCD - Code (Binary Coded Decimal)

* Here each digit of the decimal number is converted to its corresponding 4-bit representation binary code.
* So BCD means representation upto 9 from 0.

Eg : $(73)_{10} = (0111 \quad 0011)_2$

BCD addition :

→ add each digit in 4-bit code.
→ If the number obtained in form of binary 4-bits, exceeds 9 then to that particular digit add 6 (i.e., 0110)
→ If CAC is generated in any digit then to that particular digit also add 6 (i.e, 0110).

Eg:

$(76)_{10} = 0111 \quad 0110$
$+(47)_{10} = 0100 \quad 0111$

$1100 \quad 1101$ (as >9 so add 6)
$+0110 \quad 0110$
$0010 \quad 0011$

as we obtain a carry so add 6 more but we have already added 6

so ans = $(0001 \quad 0010 \quad 0011)_2$

Q. Perform following BCD addition.

(a) $(36)_{10} + (93)_{10}$

$(36)_{10}$ = 0011 0110

$(93)_{10}$ = 1001 0011

1100 1001

+0110

1 0010 1001

$\Rightarrow$ ans: $(0001\ 0010\ 1001)_2$

(b) $(175)_{10} + (437)_{10}$

$(145)_{10}$ = 0001 0111 0101

$(437)_{10}$ = 0100 0011 0111

0110 1011 1100

+1 0110 0110

0110 0001 0010

$\Rightarrow$ ans: $(0110\ 0001\ 0010)$

NB: decimal no. with factions are also done in same way.

Non-weighted Code

Gray Code:

* It is not suitable for arithmatic operations like BCD, rather it is a cyclic code and each successive code differs in only one bit position only.
* These are used in I/O devices, shaft encoders etc.

Binary to Gray

Binary: B_3 (MSB) B_2 B_1 B_0 (LSB)

Gray: G_3 G_2 G_1 G_0

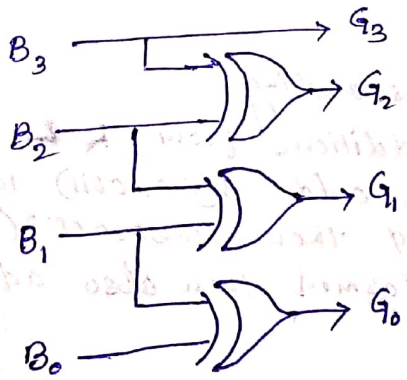
so $G_3 = B_3$

$G_2 = B_3 \oplus B_2$

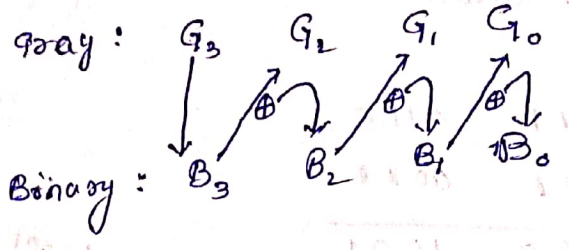
$G_1 = B_2 \oplus B_1$

$G_0 = B_1 \oplus B_0$

In form of logic gates:

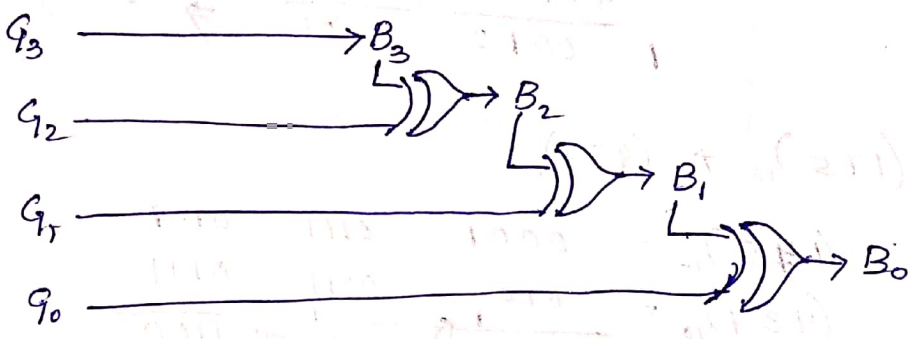


Gray to Binary



$B_3 = G_3$
$B_2 = B_3 \oplus G_2$
$B_1 = B_2 \oplus G_1$
$B_0 = B_1 \oplus G_0$

In boom of gate:



Eg:
Binary = $(1\ 0\ 0\ 1\ 1\ 0\ 1)_2$
Gray = $(1\ 1\ 0\ 1\ 0\ 1\ 1)_2$

EXCESS THREE (XS-3) code :

* It is a non-weighted BCD code, obtained by adding each decimal bit with '3' and then writing its corresponding BCD
* The vice-versa is done by subtracting 3(0011) from the XS-3 then converting the BCD to decimal.

Eg: $(1\ 7)_{10}$
+3 +3
$(4\ 10) \longrightarrow (0100\ \ 1010)_2$ (Ans).

addition :

→ convert each into corresponding
→ take individual 4-bit additions from R→L
→ If EAC is generated then add 3(i.e, 0011) to it
→ If no EAC is generated then subtract 3(i.e, 0011) from it.
→ When a new 4-bit is formed then also add 3 (i.e, 0011) to it.

Eg: $(17)_{10} + (45)_{10}$

$(17)_{10} \xrightarrow{XS-3} (4\ 10) \longrightarrow 0100\quad 1010$

$(45)_{10} \xrightarrow{XS-3} (7\ 8) \longrightarrow 0111\quad 1000$

$1100\quad 0010$

$-0011\quad +0011$

$(1001\quad 0101)$ (AW)

here in rightpart an EAC was generated, so 3 was added to it whereas in left part 3 was subtracted because no EAC was generated.

Eg: $(39)_{10} + (78)_{10}$

$(39)_{10} \xrightarrow{XS-3} (6\ 12) \longrightarrow 0110\quad 1100$

$(78)_{10} \xrightarrow{XS-3} (10\ 11) \longrightarrow 1010\quad 1011$

$0001\quad 0001\quad 0111$

$0011\quad 0011\quad +0011$

$(0100\quad 0100\quad 1010)$ (EAW)

here as in leftmost part a new 4-bit occured so 3 was again added to it.

BOOLEAN ALGEBRA

* Boolean algebra is a system of mathematical logic, i.e, used to represent any complex logic statement by a simpler boolean function.
* Logic operations used are: AND (·), OR (+), NOT (¯), XOR (⊕).
* Rules: $1 \cdot 0 = 0 \cdot 1 = 0$; $1 + 0 = 0 + 1 = 1$

$\bar{0} = 1$; $\bar{1} = 0$; $\bar{\bar{1}} = 1$; $\bar{\bar{0}} = 0$.

* $\bar{\bar{X}} = X$ (complementation law)

Pf:

LHS		RHS
X	$\bar{X}$	$\bar{\bar{X}}$
1	0	1
0	1	0

LHS = RHS

* $1 + X = 1$

Pf:

LHS		RHS
1	X	X+1
1	1	1
1	0	1

LHS = RHS

* $X \cdot \bar{X} = 0$ (Negation or complement law)

LHS		
X	$\bar{X}$	$X \cdot \bar{X}$
1	0	0
0	1	0

LHS = 0 = RHS.

* $X + X = X$

RHS		LHS
X	X	X+X
1	1	1
0	0	0

LHS = RHS

* $X \cdot X = X$

RHS		LHS
X	X	X·X
1	1	1
0	0	0

LHS = RHS

AND laws:

4 AND laws are: $A \cdot 0 = 0$, $A \cdot 1 = A$, $A \cdot A = A$ and $A \cdot \bar{A} = 0$

(proofs can be found easily by T-T)

OR laws:

4 OR laws are: $A + 0 = A$, $A + 1 = A$, $A + A = A$, $A + \bar{A} = 1$

(proofs are found by T-T)

Commutative laws:

2 commutative laws are: $A + B = B + A$ and $A \cdot B = B \cdot A$.

Associative laws

The two laws are: $A + (B + C) = (A + B) + C$ and $A \cdot (B \cdot C) = (A \cdot B) \cdot C$.

Idempotent law

2 laws are: $A \cdot A = A$ and $A + A = A$.

Distributive laws:

The two distributive laws: $A+(B\cdot C)=(A+B)\cdot(A+C)$

$A\cdot(B+C)=(A\cdot B)+(A\cdot C)$

(above can also be proved simply by drawing our T.T)

Absorption law:

The two absorption laws are:

(i) $X+X\cdot Y=X$

Pf: $LHS = X+X\cdot Y = X\cdot(1+Y) = X\cdot 1 = X = RHS$ ∵ $1+A=A$ and $A\cdot 1=A$.

(ii) $X\cdot(X+Y)=X$

Pf: $LHS = X\cdot(X+Y) = X\cdot X + X\cdot Y = X + X\cdot Y$ ∵ $AA=A$

$= X(1+Y) = X\cdot 1 = X = RHS$ ∵ $1+A=1$ and $1\cdot A=A$.

Demorgan's laws

i) $\overline{A+B}=\bar{A}\cdot\bar{B}$

Pf

					LHS	RHS
A	B	$\bar{A}$	$\bar{B}$	A+B	$\overline{A+B}$	$\bar{A}\cdot\bar{B}$
0	0	1	1	0	1	1
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	1	0	0

LHS = RHS (Proved)

ii) $\overline{A\cdot B}=\bar{A}+\bar{B}$

Pf

					LHS	RHS
A	B	$\bar{A}$	$\bar{B}$	A·B	$\overline{A\cdot B}$	$\bar{A}+\bar{B}$
0	0	1	1	0	1	1
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	0	0

LHS = RHS (Proved)

Q Solve: $\overline{\overline{x\bar{y} + xyz} + x(y + x\cdot\bar{y})}$

Given: $\overline{\overline{x\bar{y} + xyz} + x(y + x\cdot\bar{y})}$

$= \overline{\overline{x\bar{y}}\cdot\overline{xyz} + (xy + x\cdot x\cdot\bar{y})}$ ∵ Demorgan's laws $\overline{A+B} = \bar{A}\cdot\bar{B}$

$= \overline{(\bar{x}+\bar{y})\cdot(\bar{x}+\bar{y}+\bar{z}) + (xy + x\bar{y})}$ ∵ Demorgan's laws $\overline{A\cdot B} = \bar{A}+\bar{B}$ & $A\cdot A = A$

$= \overline{(\bar{x}+y)\cdot(\bar{x}+\bar{y}+\bar{z}) + x(y+\bar{y})}$ ∵ $\bar{\bar{A}} = A$

$= \overline{\bar{x}\cdot\bar{x} + \bar{x}\cdot\bar{y} + \bar{x}\cdot\bar{z} + y\bar{x} + y\bar{y} + yz + x\cdot 1}$ ∵ $A + \bar{A} = 1$

$= \overline{\bar{x} + \bar{x}\cdot\bar{y} + \bar{x}\bar{z} + y\bar{x} + 0 + yz + x}$ ∵ $A\cdot A = A$ & $A\cdot 1 = A$

$= \overline{(x+\bar{x}) + x\bar{y} + \bar{x}\bar{z} + \bar{x}y + yz}$

$= \overline{1 + (x\bar{y} + \bar{x}\bar{z} + \bar{x}y + yz)}$ ∵ $A + \bar{A} = 1$

$= \bar{1} = 0$ ∵ $1 + A = 1$

(Ans)

Any boolean expression can be expressed in a standard or cannonical or expanded sum (OR) of products (AND) : SOP form, or a standard or canonical or expanded product (AND) of sums (OR) : POS form.

Sum Of Products (SOP)

* In this form there are a number of product terms, each one of which contains all the variables of function either in complemented or non-complemented form, are summed Together.
* Each product term is called as a minterm, usually represented by 0s and 1s (1 for non complemented & 0 for complemented ones)
* Their Decimal equivalent is expressed as a subscript of m, i.e, $m_0, m_1, \ldots$ etc.

Eg: $f(A,B) = \bar{A} + AB$

$= \bar{A}(B+\bar{B}) + AB$

$= \bar{A}B + \bar{A}\bar{B} + AB$ ((is the SOP form in variable state

$= 01 + 00 + 11$ ((SOP in binary state

$= m_1 + m_0 + m_3$ ((SOP in decimal state

$= \sum m(0,1,3)$ ((SOP in actual representation

Q Convert to SOP form:

(i) $f(A,B,C) = \bar{A}C + A\bar{B} + B\bar{C}$

$= \bar{A}(B+\bar{B})C + A\bar{B}(C+\bar{C}) + (A+\bar{A})B\bar{C}$

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

$= 011 + 001 + 101 + 100 + 110 + 010$

$= m_3 + m_1 + m_5 + m_4 + m_6 + m_2$

$\Rightarrow \boxed{f(A,B,C) = \sum m(1,2,3,4,5,6)}$ (Ans)

(ii) $f(A,B,C,D) = \bar{A}CD + \bar{B}\bar{D} + A\bar{D}$

$= \bar{A}(B+\bar{B})CD + (A+\bar{A})B(C+\bar{C})\bar{D} + A(B+\bar{B})(C+\bar{C})\bar{D}$

$= (\bar{A}B+\bar{A}\bar{B})CD + (AB+\bar{A}B)(C\bar{D}+\bar{C}\bar{D}) + (AB+A\bar{B})(C\bar{D}+\bar{C}\bar{D})$

$= \bar{A}BCD + \bar{A}\bar{B}CD + AB\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D} + \bar{A}BC\bar{D} + \bar{A}B\bar{C}\bar{D} + ABCD + AB\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D}$

$= 0111 + 0011 + 1110 + 1100 + 0110 + 0100 + 1000 + 1010$

$= m_7 + m_3 + m_{14} + m_{12} + m_6 + m_4 + m_8 + m_{10}$

$\Rightarrow f(A,B,C,D) = \sum m(0,2,3,7,8,10,12,14)$

(iii) $f(A,B,C,D) = B\bar{C}\bar{D} + A\bar{D}\bar{C} + \bar{C}\bar{B} + \bar{D}$

$= (A+\bar{A})B\bar{C}\bar{D} + A(B+\bar{B})\bar{C}\bar{D} + (A+\bar{A})B\bar{C}(C+\bar{D}) + (A+\bar{A})(B+\bar{B})(C+\bar{C})\bar{D}$

$= AB\bar{C}\bar{D} + \bar{A}B\bar{C}\bar{D} + AB\bar{C}\bar{D} + AB\bar{C}\bar{D} + (A\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C})(D+\bar{D}) + (AB+A\bar{B}+\bar{A}B+\bar{A}\bar{B})(C\bar{D}+\bar{C}\bar{D})$

$= AB\bar{C}\bar{D} + \bar{A}B\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D} + AB\bar{C}D + A\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D} + ABC\bar{D} + A\bar{B}C\bar{D} + \bar{A}BC\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}BC\bar{D} + \bar{A}\bar{B}C\bar{D} + AB\bar{C}\bar{D}$

$= 1100 + 0100 + 1000 + 1001 + 0001 + 0000 + 1110 + 0010 + 0110 + 1010$

$= m_{12} + m_4 + m_8 + m_9 + m_1 + m_0 + m_{14} + m_2 + m_6 + m_{10}$

$\Rightarrow f(A,B,C,D) = \sum m(0,1,2,4,6,8,9,10,12,14)$

Product Of Sum (POS):

* It is the standard form in which a number of sum terms, each one of which contains all variables of the function either in complemented or non-complemented form, are multiplyed togat
* Each sum term is called as maxterm, usually represented in 0s and 1s (non complemented is 0 and complimented is 1).
* The decimal equivalent is expressed as a subscript of M, i.e, $M_0, M_1, M_2, \ldots$ etc.
* First step : convert expression to SOP form
  Second step : replace '.' with '+' & vice-versa in obtained SOP.

Eg: $f(A,B) = \bar{A} + AB$

$= \bar{A}(B+\bar{B}) + AB$

$= \bar{A}B + \bar{A}\bar{B} + AB$ // SOP form

$= (\bar{A}+B)\cdot(\bar{A}+\bar{B})\cdot(A+B)$ // POS form algebraically

$= 10 \cdot 01 \cdot 00$

$= M_2 \cdot M_3 \cdot M_0$

$\Rightarrow \boxed{f(A,B) = \prod M(0,2,3)}$ (Ans).

Karnaugh Map

* It is a systematic method of simplifying Boolean expression
* It is a chart or a graph, composed of an arrangement of adjacent cells, each representing a particular combination of variables in sum or product form.
* If there are 'n' variables then there are 2^n squares.
* The binary numbers given on the squares are in gray codes.

n = 2

Total no. of squares = $2^2 = 4$

A \ B	$(\bar{B})$ 0	(B) 1
$(\bar{A})$ 0	00 $(\bar{A}\bar{B})$ 0	01 $(\bar{A}B)$ 1
(A) 1	10 $(A\bar{B})$ 2	11 (AB) 3

↝ Gray codes.

↓ Gray codes.

n = 3

Total no. of squares = $2^3 = 8$

A \ BC	$(\bar{B}\bar{C})$ 00	$(\bar{B}C)$ 01	(BC) 11	$(B\bar{C})$ 10
$(\bar{A})$ 0	000 $(\bar{A}\bar{B}\bar{C})$ 0	001 $(\bar{A}\bar{B}C)$ 1	011 $(\bar{A}BC)$ 3	010 $(\bar{A}B\bar{C})$ 2
(A) 1	100 $(A\bar{B}\bar{C})$ 4	101 $(A\bar{B}C)$ 5	111 (ABC) 7	110 $(AB\bar{C})$ 6

AB \ C	$(\bar{C})$ 0	(C) 1
$(\bar{A}\bar{B})$ 00	000 0	001 1
$(\bar{A}B)$ 01	010 2	011 3
(AB) 11	110 6	111 7
$(A\bar{B})$ 10	100 4	101 5

n = 4

Total no. of squares = $2^4 = 16$

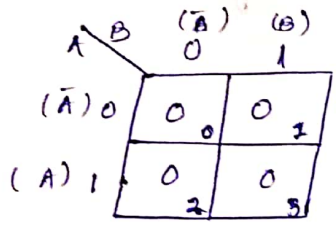
AB \ CD	$(\bar{C}\bar{D})$ 00	$(\bar{C}D)$ 01	(CD) 11	$(C\bar{D})$ 10
$(\bar{A}\bar{B})$ 00	0000 $(\bar{A}\bar{B}\bar{C}\bar{D})$ 0	0001 $(\bar{A}\bar{B}\bar{C}D)$ 1	0011 $(\bar{A}\bar{B}CD)$ 3	0010 $(\bar{A}\bar{B}C\bar{D})$ 2
$(\bar{A}B)$ 01	0100 $(\bar{A}B\bar{C}\bar{D})$ 4	0101 $(\bar{A}B\bar{C}D)$ 5	0111 $(\bar{A}BCD)$ 7	0110 $(\bar{A}BC\bar{D})$ 6
(AB) 11	1100 $(AB\bar{C}\bar{D})$ 12	1101 $(AB\bar{C}D)$ 13	1111 $(ABCD)$ 15	1110 $(ABC\bar{D})$ 14
$(A\bar{B})$ 10	1000 $(A\bar{B}\bar{C}\bar{D})$ 8	1001 $(A\bar{B}\bar{C}D)$ 9	1011 $(A\bar{B}CD)$ 11	1010 $(A\bar{B}C\bar{D})$ 10

representing an expression in a K-map and loops of K-map:

* convert into SOP form first.
* in the K-map put '1' only at those places where the expression of SOP obtained matches with the squares of K-map.
* the find loops either in vertical or horizontal manner with number of squares of form 2, 4, 8, 16, 32, etc..
* for every loop obtained, find the common in the squares of that loop and then assign that value to the loop.
* finally add all the loop expressions which represents the actual function taken.

Eg: $f(A,B) = 0 = \sum m()$

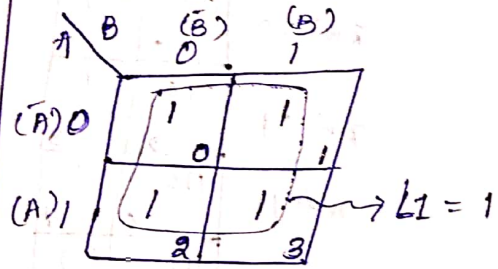
⇒ 2 variable K-map ⇒ 4 squares



no loops ⇒ $f(A,B) = 0$

① Solve using K-map:

i) $f(A,B) = 1$, $= \sum m(0,1,2,3)$

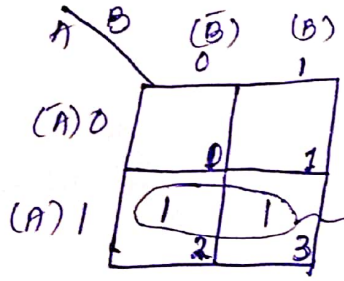


here all square boxes are given 1.

$\therefore f(A,B) = L_1 = 1$

ii) $f(A,B) = A$

$= A(B+\bar{B}) = AB + A\bar{B} \quad = \sum m(2,3)$

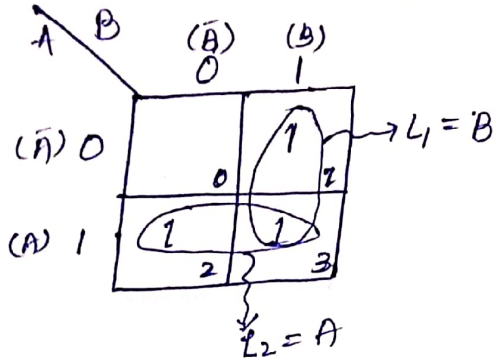


as A is common to both the squares.

$\therefore f(A,B) = L_1 = A$

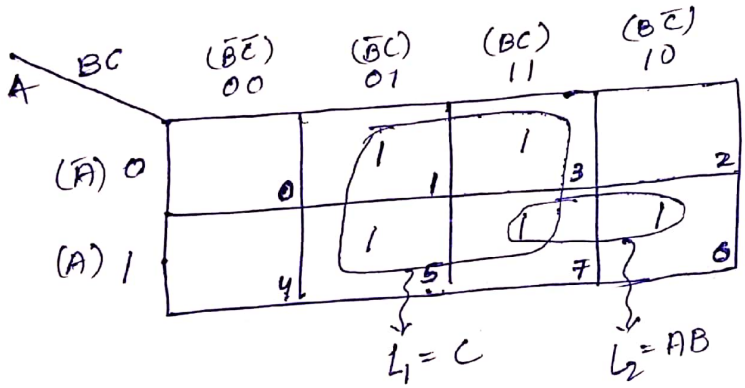
iii) $f(A,B) = A+B = A(B+\bar{B}) + B(A+\bar{A})$

$= AB + A\bar{B} + \bar{A}B = \sum m(1,2,3)$



$\therefore f(A,B) = L_1 + L_2 = A+B$

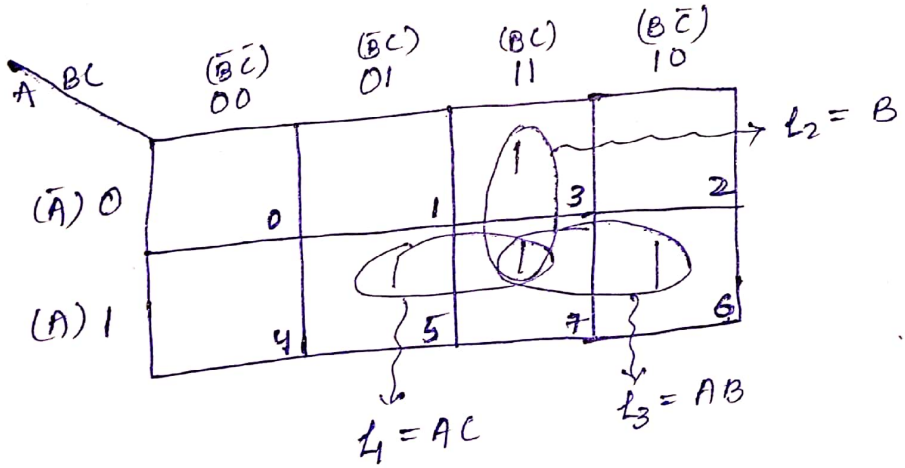
iv) $f(A,B,C) = ABC + A\bar{B}C + A\bar{B}\bar{C} = \sum m(4,5,7)$



[here 1st go for higher number of loops. then lower ones, i.e, 16 > 8 > 4 > 2.

v) $f(A,B,C) = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

$= \sum m(3,5,6,7)$



$\therefore f(A,B,C) = L_1 + L_2 + L_3$

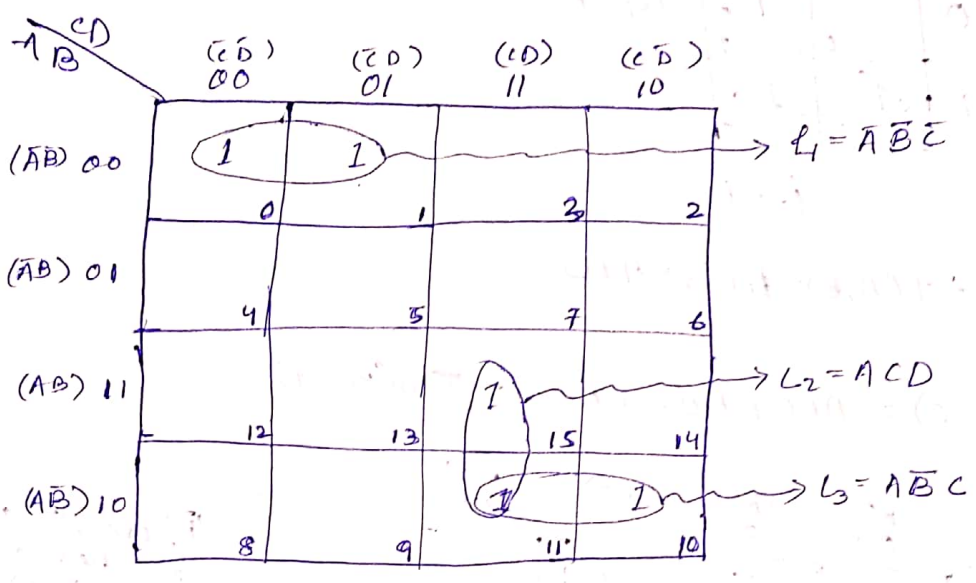
$\Rightarrow \boxed{f(A,B,C) = B + AC + AB}$

(Ans)

1) Solve by using K-map and then draw the expression in terms of basic logic gates.

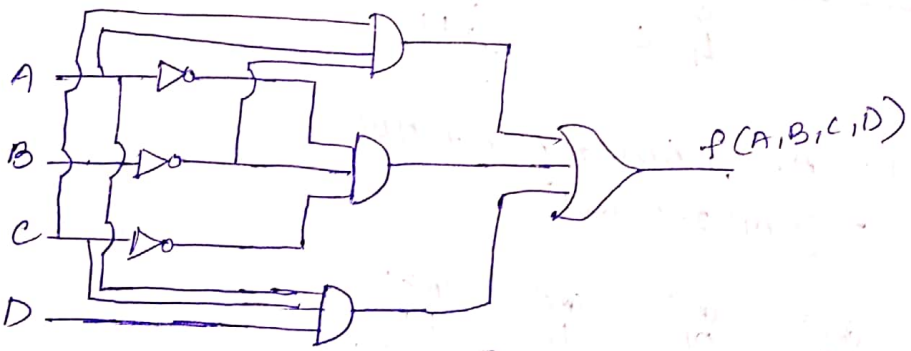
$f(A,B,C,D) = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + A\bar{B}\bar{C}\bar{D} + A\bar{B}CD + ABCD$

$= \sum m(0, 1, 10, 11, 15)$

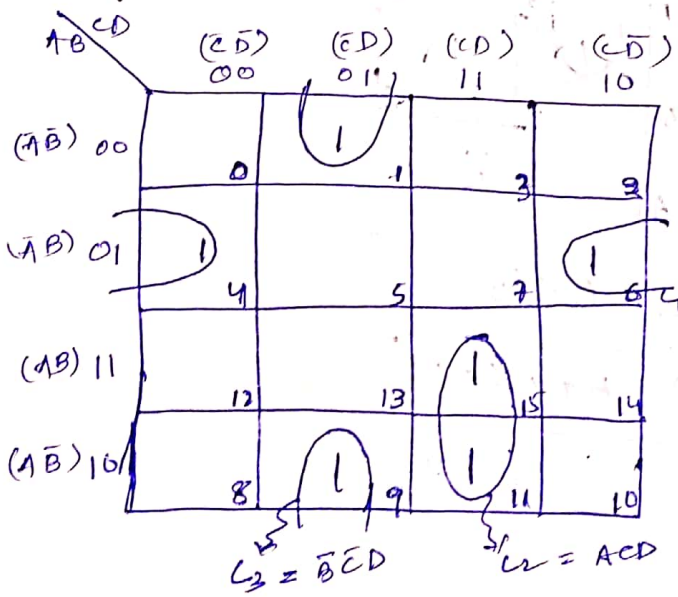


$\therefore f(A,B,C) = L_1 + L_2 + L_3$

$\Rightarrow \boxed{f(A,B,C) = \bar{A}\bar{B}\bar{C} + ACD + A\bar{B}C}$

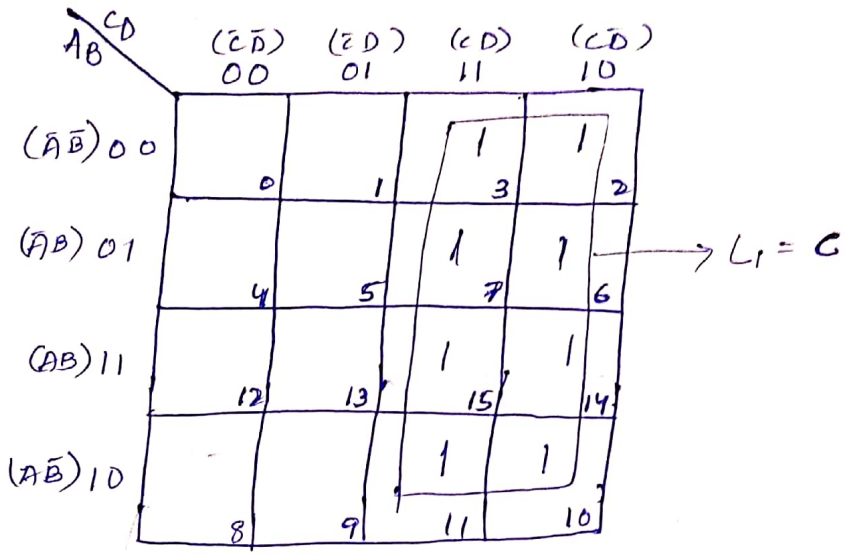


2) Solve using K-map: $f(A,B,C,D) = \sum m(1,4,6,11,9,15)$



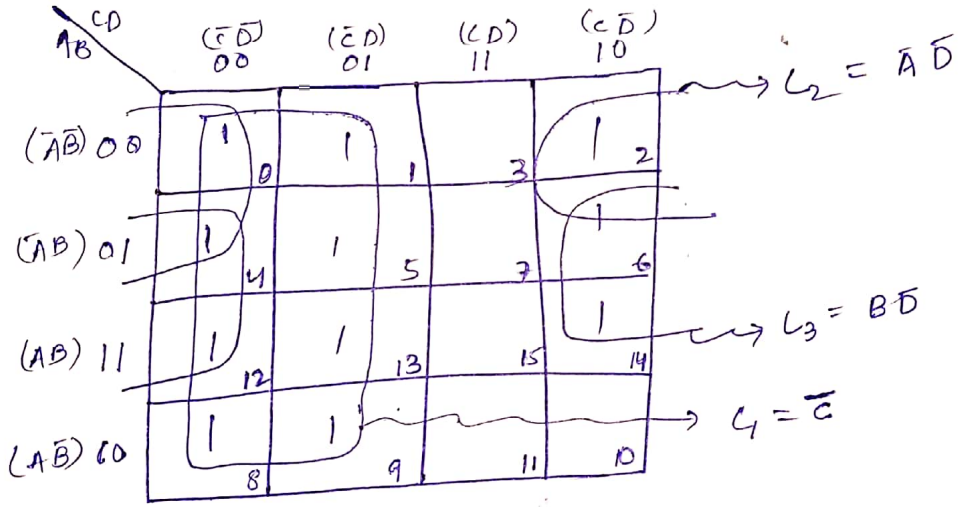
$\boxed{f(A,B,C,D) = \bar{A}B\bar{D} + \bar{B}\bar{C}D + ACD}$

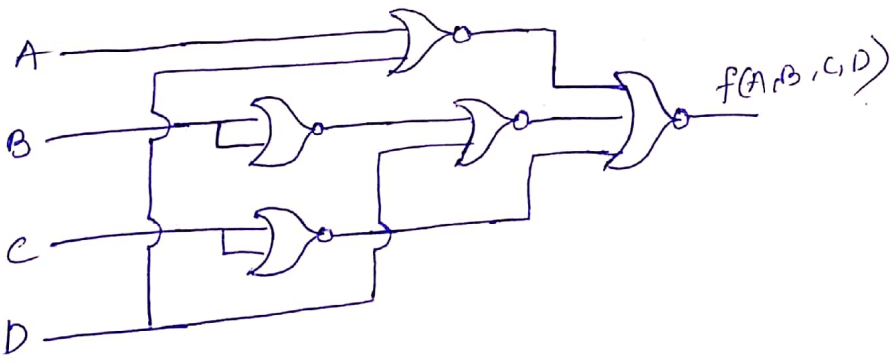
Q Solve using K-Map method $f(A,B,C,D) = \sum m(2,3,6,7,10,11,14,15)$



$\therefore f(A,B,C,D) = C$

Q Solve $f(A,B,C,D) = \sum m(0,1,4,5,12,13,8,9,2,6,14)$ and then design the minimised function using only NOR gate.





conversions are

$L_3 = B\bar{D} = \overline{\bar{B} + D}$

$L_2 = \bar{A}\bar{D} = \overline{A + D}$

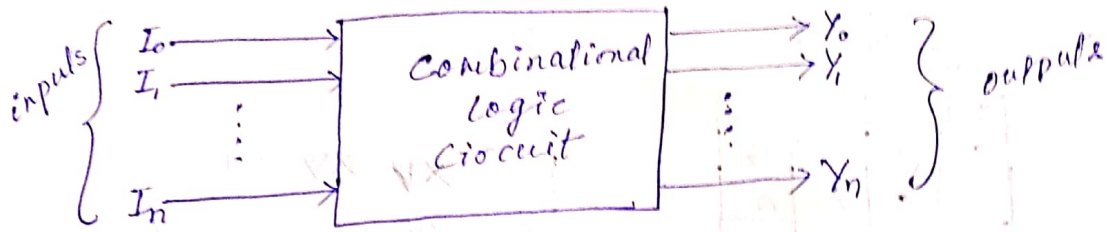
$L_1 = \bar{C} = \overline{C + C}$

Comination Logic Circuits

Combinational logic ckt.

* It is a logic circuit in which the output at any instant depends upon the input present at that instant only.
* It doesn't depend upon the previous input or past output.
* So, it doesn't require any memory element to store the previous input or past output.

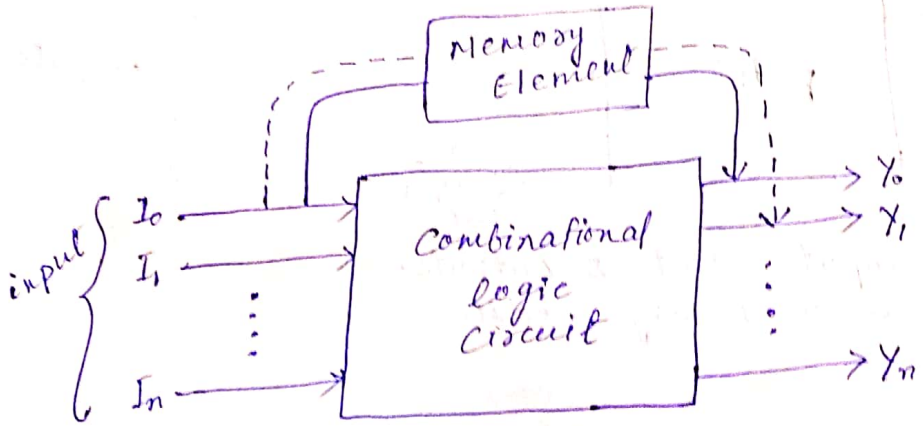
Eg: Half-Adder, Full-Adder, Half-Subtractor, Full-Subtractor, Binary parallel Adder circuit, Multiplexer, Demultiplexer.



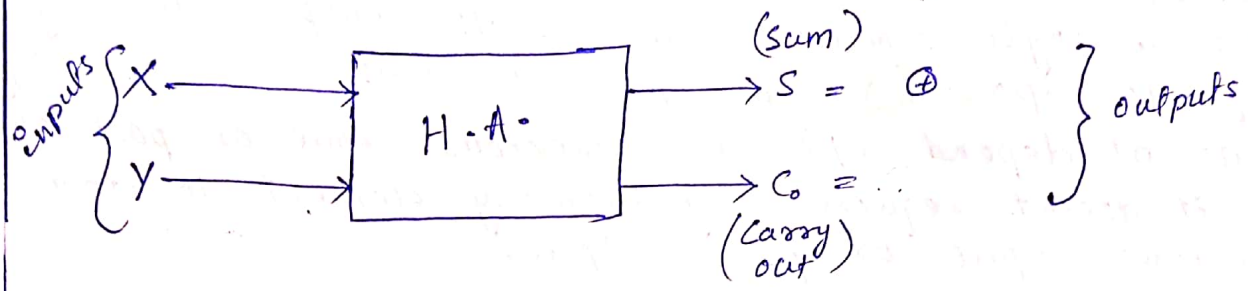
Sequential logic Circuit

* It is a logic circuit in which output at any instant not only depends upon the present input at that instant, but also depends on previous input or past output.
* It requires a memory element to store the previous input as well as past output.
* So, it is a combination of combinational logic circuit and a memory element.

Eg: Flip-flop, Shift Registers, Counters



Half Adder (H.A)



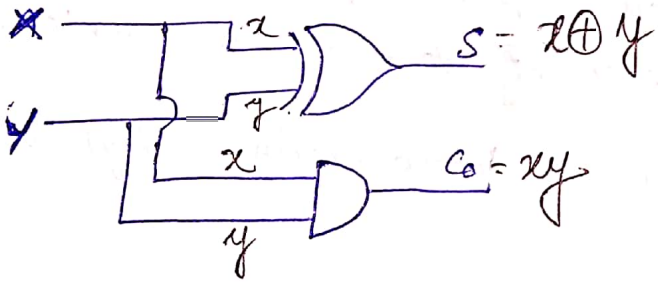
★ It has two inputs and two outputs.
★ It performs binary addition

Truth Table

Inputs		Outputs	
X	Y	S	C_o
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

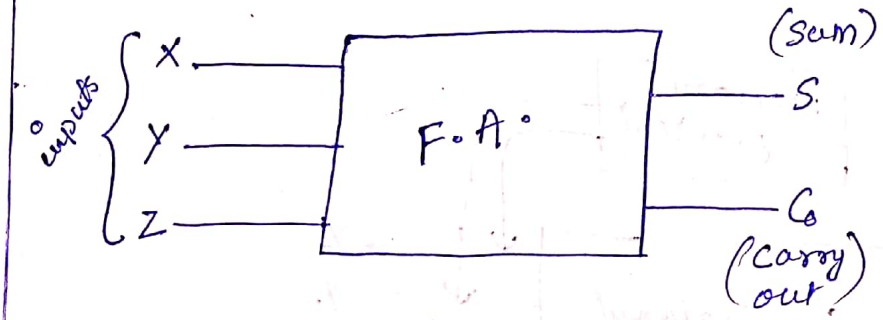
$S = \bar{X}Y + X\bar{Y} = X \oplus Y$
$C_o = XY$

★ logic gate representation:



(which can also be represented with use of universal gates)

Full Adder



★ There are 3 inputs and 2 outputs.
★ It also performs binary addition

Truth Table

X	Y	Z	S	C_o
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

$\therefore S = \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ$

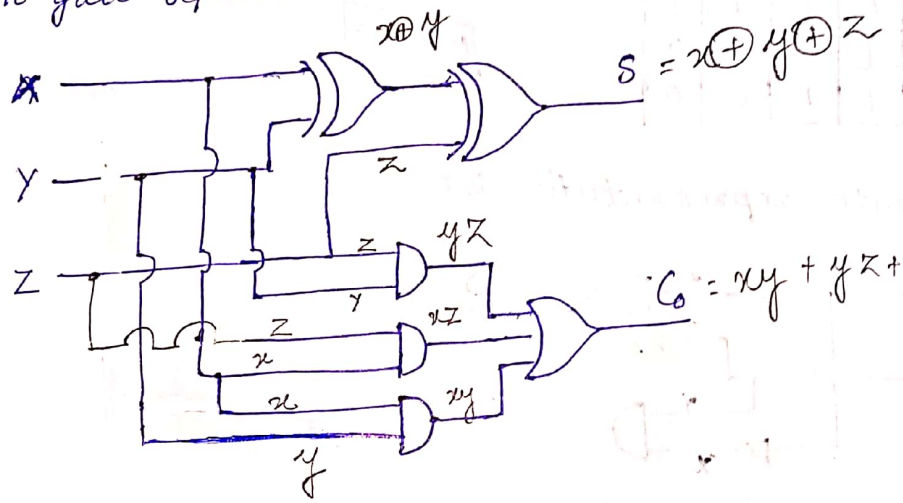
$\Rightarrow \boxed{S = A \oplus B \oplus C}$

$C_o = \bar{X}YZ + X\bar{Y}Z + XY\bar{Z} + XYZ$

$= (\bar{X}YZ + XYZ) + (X\bar{Y}Z + XYZ) + (XY\bar{Z} + XYZ)$

$\Rightarrow \boxed{C_o = XY + YZ + XZ}$

★ logic gate representation is:

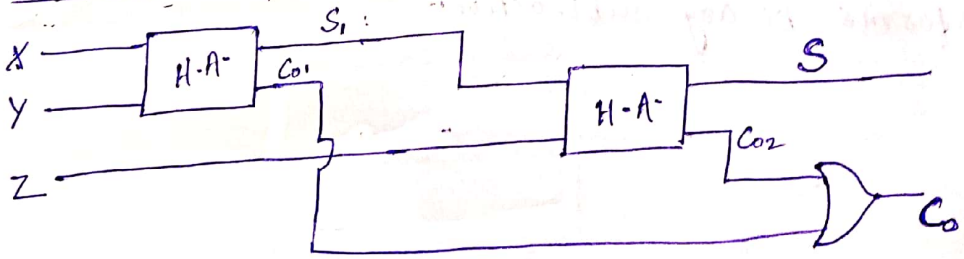


nb $A \oplus B \oplus C = (\bar{A}B + \bar{B}A) \oplus C = (\bar{A}B + A\bar{B}) \cdot \bar{C} + \overline{(\bar{A}B + A\bar{B})} \cdot C$

$= \bar{A}B\bar{C} + A\bar{B}\bar{C} + \overline{\bar{A}B} \cdot \overline{A\bar{B}} \cdot C = \bar{A}B\bar{C} + A\bar{B}\bar{C} + (A + \bar{B})(\bar{A} + B) \cdot C$

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

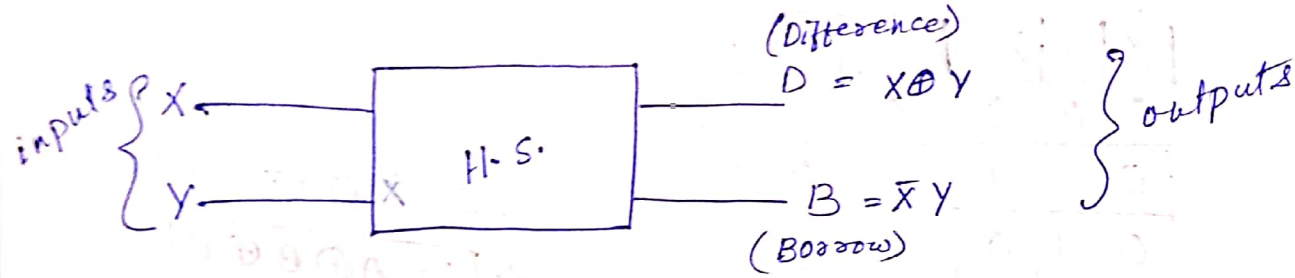
★ Full Adder using Half Adder



$S_1 = X \oplus Y$, $S = X \oplus Y \oplus Z$, $C_{o1} = XY$, $C_{o2} = (X \oplus Y) \cdot Z = XZ + YZ$, $C = C_{o1} + C_{o2}$

$\Rightarrow C = XY + YZ + ZX$

Half Subtractor:



* It takes 2 inputs and gives 2 outputs.
* It performs binary subtraction.

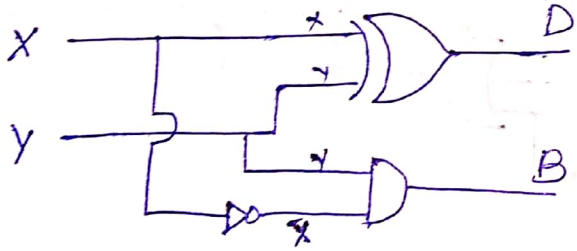
Truth Table

Input		Output	
X	Y	B	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

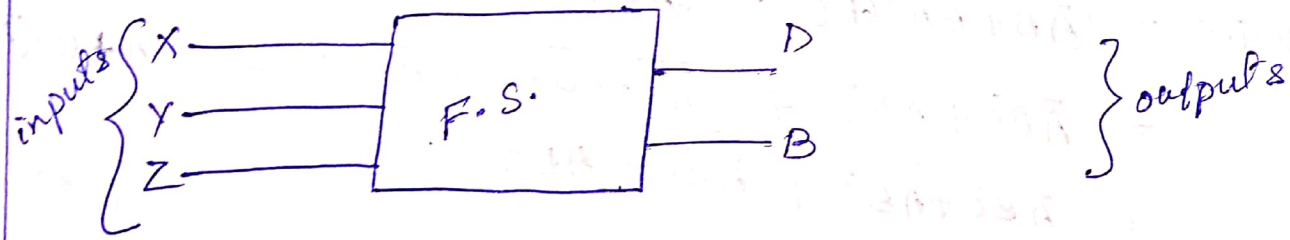
$\therefore D = \bar{X}Y + X\bar{Y} = X \oplus Y$

$B = \bar{X}Y$

* Logic gate representation is:



Full Subtractor



* There are 3 inputs and 2 outputs.
* It performs binary subtraction.

Truth Table

X	Y	Z	D	B
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$\therefore D = \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ$

$\Rightarrow \boxed{D = X \oplus Y \oplus Z}$

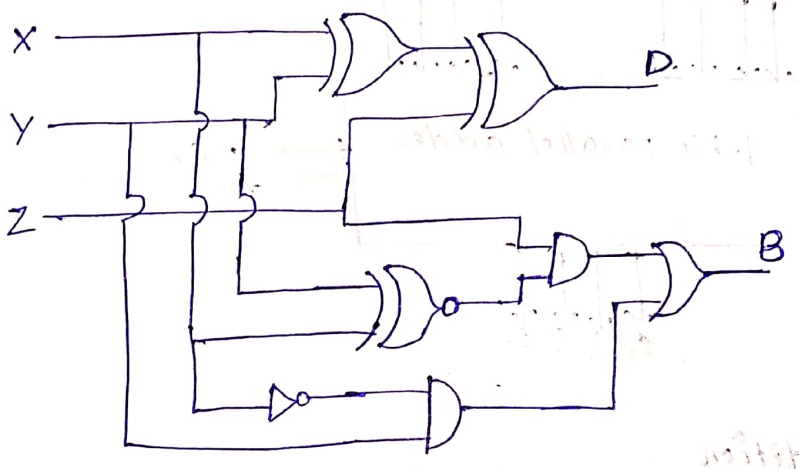
$B = \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + \bar{X}YZ + XYZ$

$= (\bar{X}\bar{Y}Z + XYZ) + (\bar{X}Y\bar{Z} + \bar{X}YZ)$

$= (\bar{X}\bar{Y} + XY)Z + \bar{X}Y$

$\Rightarrow \boxed{B = \overline{X \oplus Y} \cdot Z + \bar{X}Y}$

* Logic gate representation.



Binary Parallel Adder Ckt.

*A binary full adder can add two one-bit binary numbers and a carry in. So extending this concept we can add two n-bit binary numbers with 'n' full adders and in such the 1st carry in, i.e., at the least significant stage, is grounded.

Eg: 4 bit binary parallel Adder ckt.

$C_2 \quad C_1 \quad C_0 \quad (C_{in} = 0)$

$A_3 \quad A_2 \quad A_1 \quad A_0$

$B_3 \quad B_2 \quad B_1 \quad B_0$

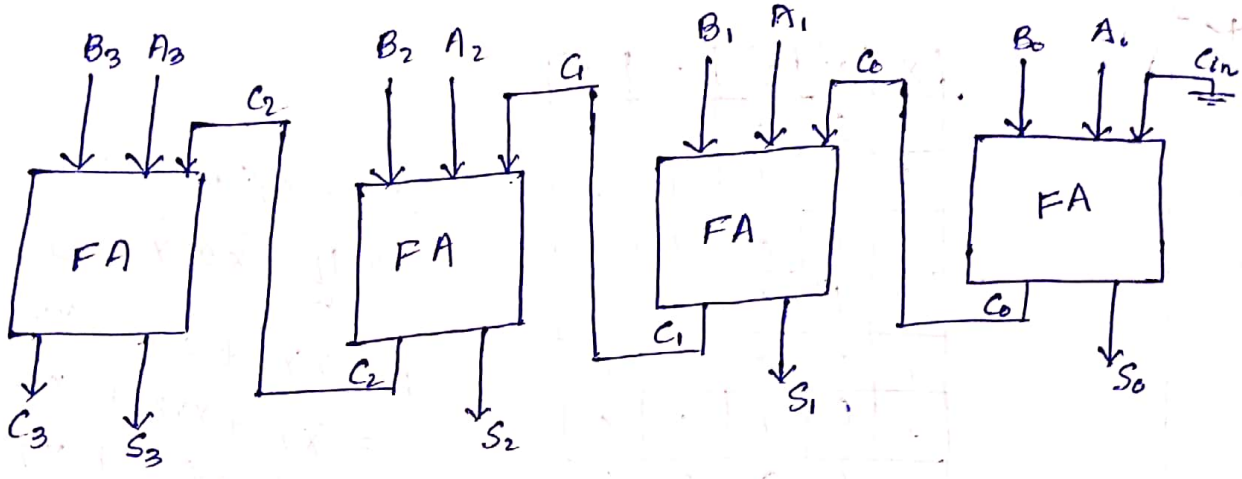
$C_3 \quad S_3 \quad S_2 \quad S_1 \quad S_0$

$S_3 \rightarrow A_3 + B_3 + C_2$

$S_2 \rightarrow A_2 + B_2 + C_1$

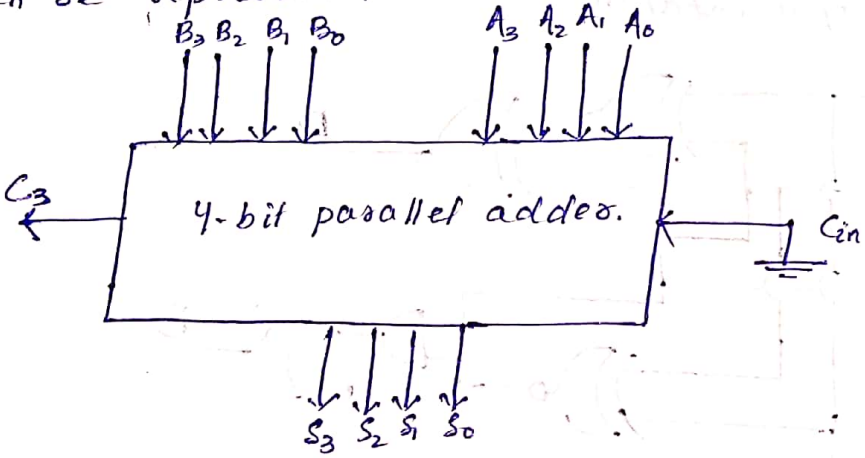
$S_1 \rightarrow A_1 + B_1 + C_0$

$S_0 \rightarrow A_0 + B_0 + C_{in}$



(4-bit Binary Ilel adder ckt.)

It can be represented as



Excess 3 addition

			C_2	C_1	C_0	($C_{in}=0$)
Let	excess 3 A =		A_3	A_2	A_1	A_0
	excess 3 B =		B_3	B_2	B_1	B_0
	(+)					
	C_3	S_3	S_2	S_1	S_0	

case1 if $C_3 = 1$ then to $(S_3 S_2 S_1 S_0)_2$, $(0011)_2$ will be added.

case2 if $C_3 = 0$ then to $(S_3 S_2 S_1 S_0)_2$, $(0011)_2$ will be subtracted, i.e, it's 2's form = $(1101)_2$ will be added.

so analysing $(0011)_2$ and $(1101)_2$ in respective cases in any of cases we need to add. $\bar{C}_3\ \bar{C}_3\ C_3\ \bar{C}_{in}$

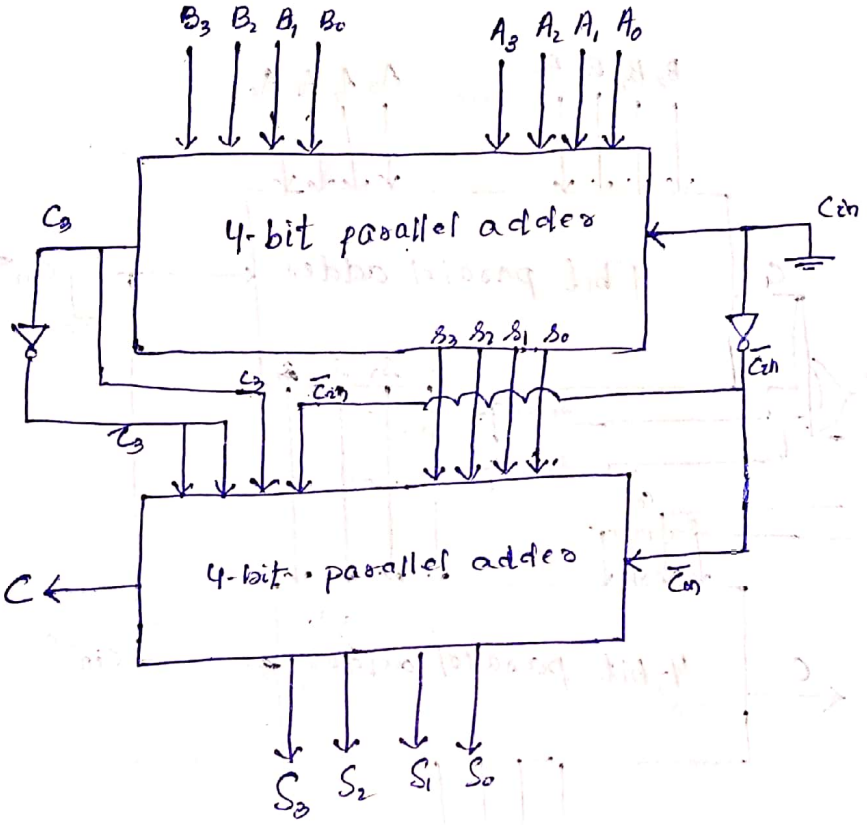
∴

$A_3\ A_2\ A_1\ A_0$ $(C_{in}=0)$

$B_3\ B_2\ B_1\ B_0$

$+\ \bar{C}_3\ \bar{C}_3\ C_3\ \bar{C}_{in}$

$C\ S_3\ S_2\ S_1\ S_0$



BCD Addition

$(C_{in}=0)$

Let BCD of A = $A_3\ A_2\ A_1\ A_0$

BCD of B = $B_3\ B_2\ B_1\ B_0$

$C_4\ S_3\ S_2\ S_1\ S_0$

The range of $(C_4 S_3 S_2 S_1 S_0)_2$ for which addition of $(0110)_2$ is required to $(S_3 S_2 S_1 S_0)$ is from $(10)_{10}$ to $(18)_{10}$.

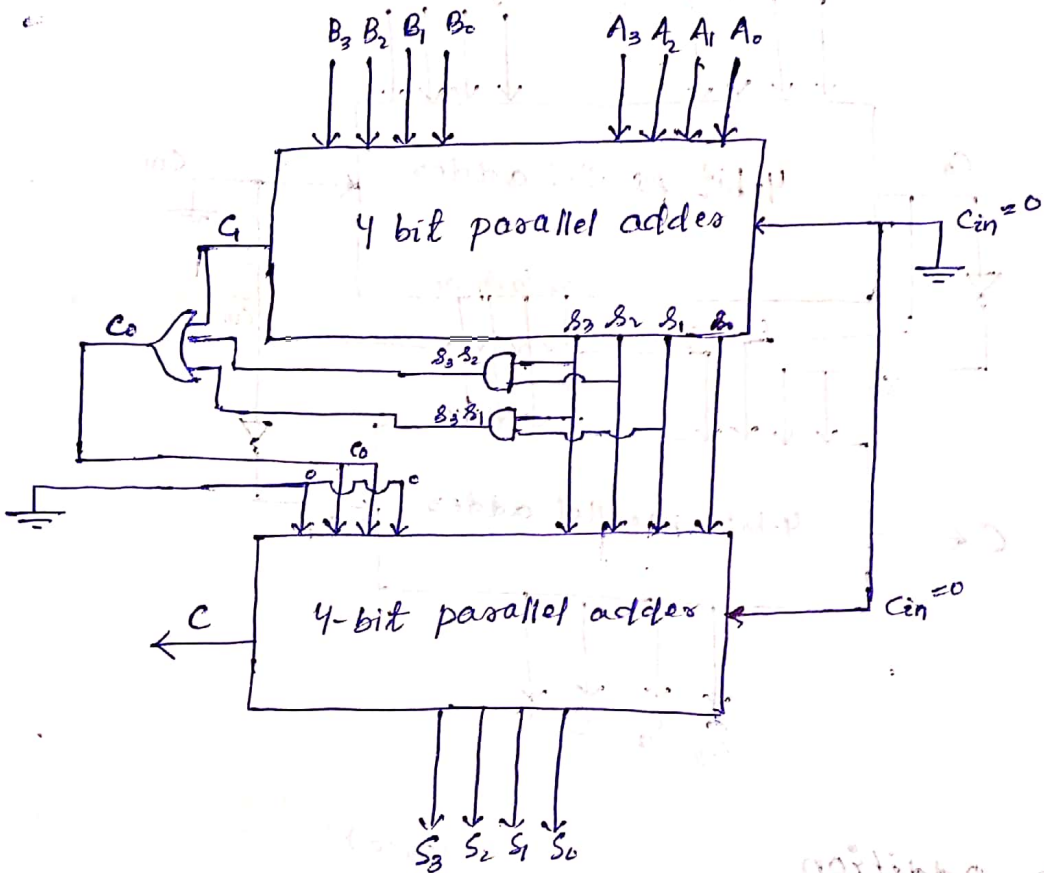
C_4	S_3	S_2	S_1	S_0	decimal
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
0	1	0	0	0	16
1	0	0	0	1	17
1	0	0	1	0	18

As we require to add 0 1 1 0 = 0 C_0 C_0 0 (say)

analysing the table $C_0 = C_1 + S_3 \cdot (S_2 + S_1)$

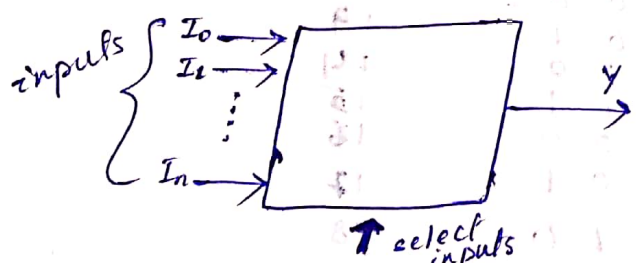
∴ next addition is

	S_3	S_2	S_1	S_0
	0	C_0	C_0	0
C	S_3	S_2	S_1	S_0



Multiplexers: (data selectors)

* Multiplexing means sharing,
* Multiplexers (MUX) or data selectors is a logic circuit that accepts several data inputs and allows only one of them at a time to get through the output.
* the process from input to output is controlled by SELECT inputs or the SELECT lines (or called as ADDRESS inputs)



*A multiplexer acts like a digitally controlled multi-position switch, i.e, it selects one of the N-inputs and transmits to the output (called as multiplexing).

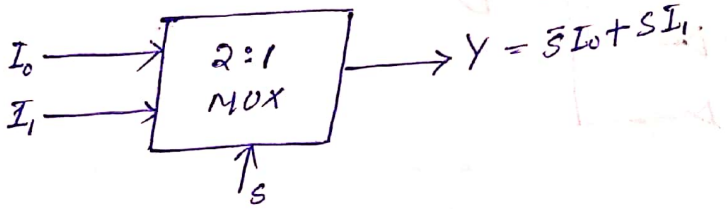
nb If 'N' is no. of inputs then 'n' is no. of select lines such that $N = 2^n$.

2:1 MUX

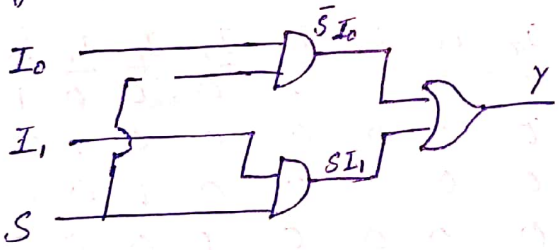
no. of inputs = 2 = 2^1 ⇒ no. of select lines = 1

S	I_1	I_0	Y
0	0	1	$I_0 = \bar{S} I_0$
1	1	0	$I_1 = S I_1$

$$\therefore \boxed{Y = \bar{S} I_0 + S I_1}$$



In logic gates, we can represent it as

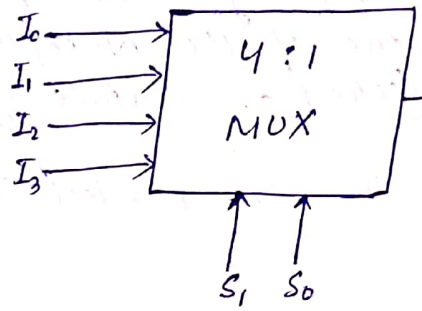


4:1 MUX

∴ N = 4 ⇒ n = 2

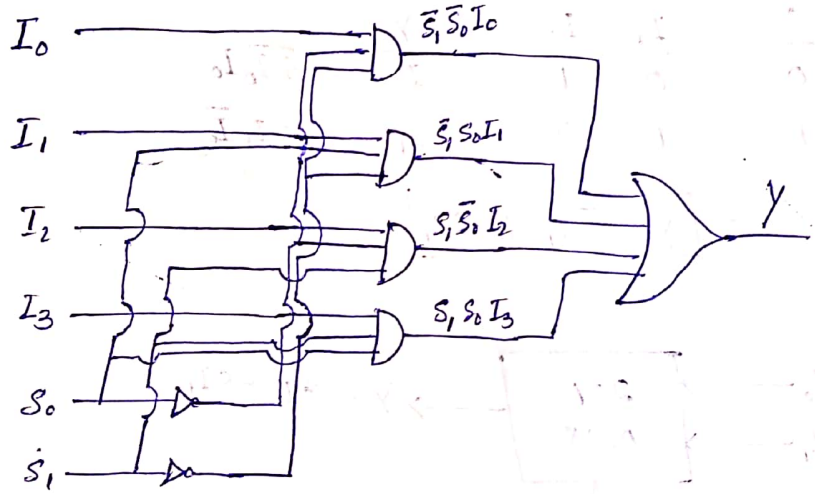
S_1	S_0	I_3	I_2	I_1	I_0	Y
0	0	0	0	0	1	$I_0 = \bar{S}_1 \bar{S}_0 I_0$
0	1	0	0	1	0	$I_1 = \bar{S}_1 S_0 I_1$
1	0	0	1	0	0	$I_2 = S_1 \bar{S}_0 I_2$
1	1	1	0	0	0	$I_3 = S_1 S_2 I_3$

$$\therefore \boxed{Y = \bar{S}_1 \bar{S}_0 I_0 + \bar{S}_1 S_0 I_1 + S_1 \bar{S}_0 I_2 + S_1 S_2 I_3}$$



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

logic gate representation:

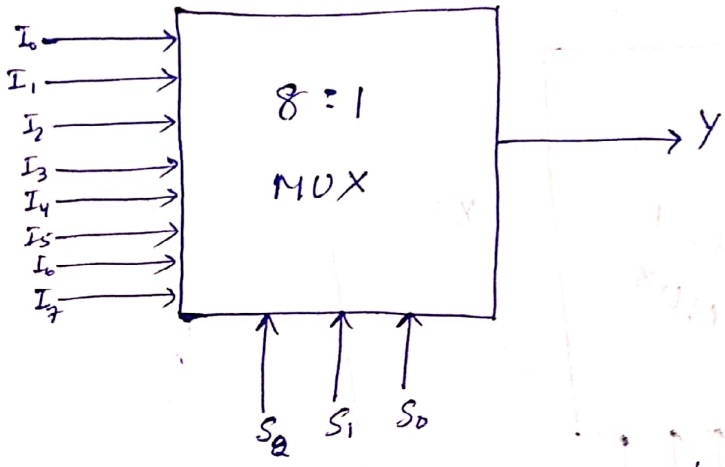


8 : 1 MUX

$N = 8 = 2^3 \Rightarrow n = 3$

S_2	S_1	S_0	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	Y
0	0	0	0	0	0	0	0	0	0	1	$I_0 = \bar{S}_2 \bar{S}_1 \bar{S}_0 I_0$
0	0	1	0	0	0	0	0	0	1	0	$I_1 = \bar{S}_2 \bar{S}_1 S_0 I_1$
0	1	0	0	0	0	0	0	1	0	0	$I_2 = \bar{S}_2 S_1 \bar{S}_0 I_2$
0	1	1	0	0	0	0	1	0	0	0	$I_3 = \bar{S}_2 S_1 S_0 I_3$
1	0	0	0	0	0	1	0	0	0	0	$I_4 = S_2 \bar{S}_1 \bar{S}_0 I_4$
1	0	1	0	0	1	0	0	0	0	0	$I_5 = S_2 \bar{S}_1 S_0 I_5$
1	1	0	0	1	0	0	0	0	0	0	$I_6 = S_2 S_1 \bar{S}_0 I_6$
1	1	1	1	0	0	0	0	0	0	0	$I_7 = S_2 S_1 S_0 I_7$

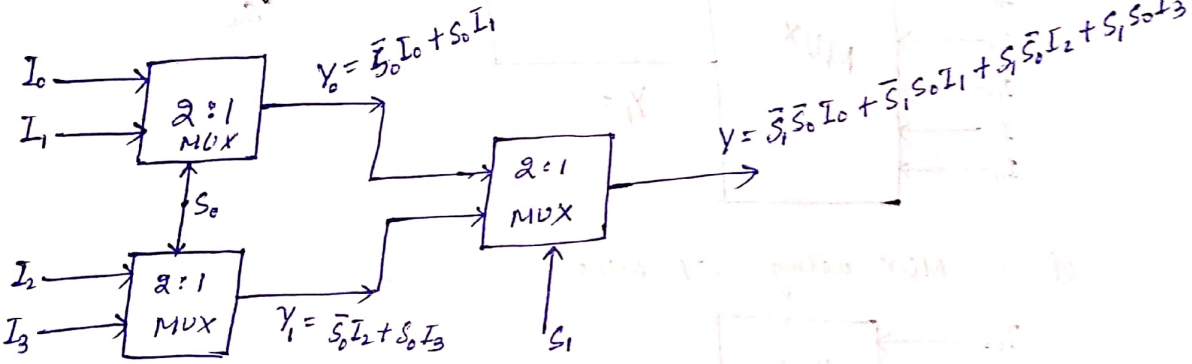
$$\therefore Y = \bar{S}_2 \bar{S}_1 \bar{S}_0 I_0 + \bar{S}_2 \bar{S}_1 S_0 I_1 + \bar{S}_2 S_1 \bar{S}_0 I_2 + \bar{S}_2 S_1 S_0 I_3 + S_2 \bar{S}_1 \bar{S}_0 I_4 + S_2 \bar{S}_1 S_0 I_5 + S_2 S_1 \bar{S}_0 I_6 + S_2 S_1 S_0 I_7$$



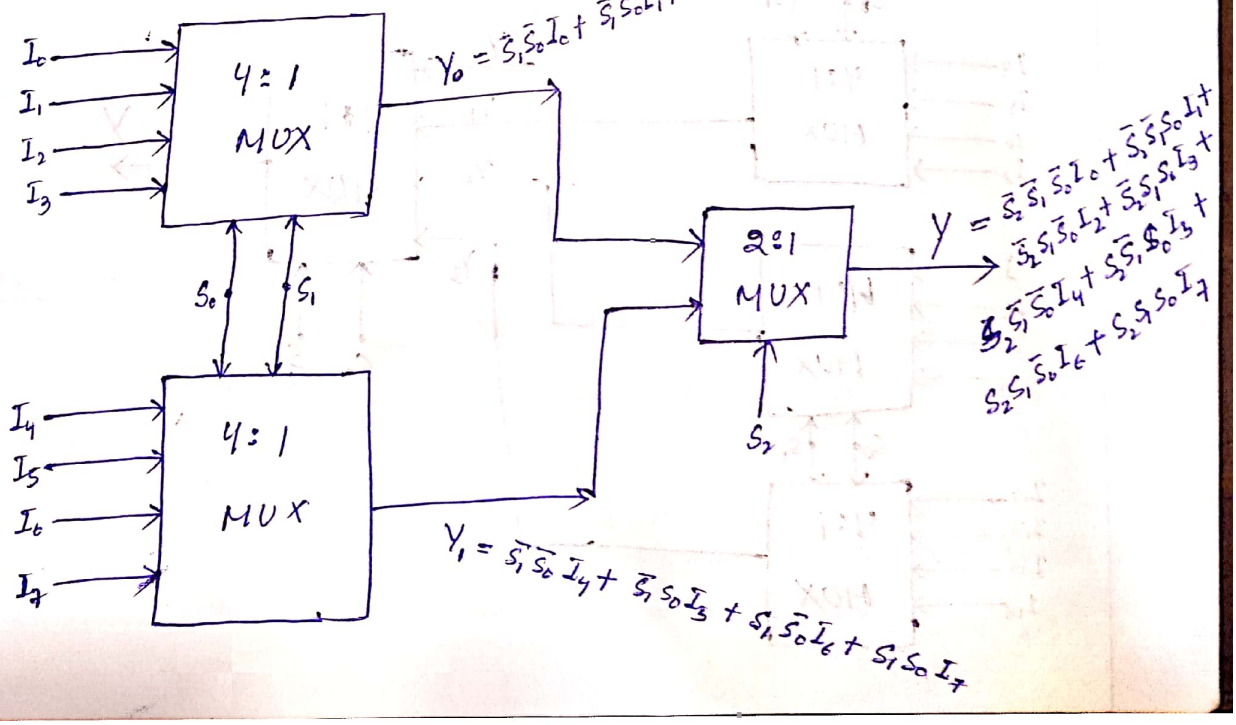
similar fashion, the logic diagram can be drawn.

Q: Design the following:

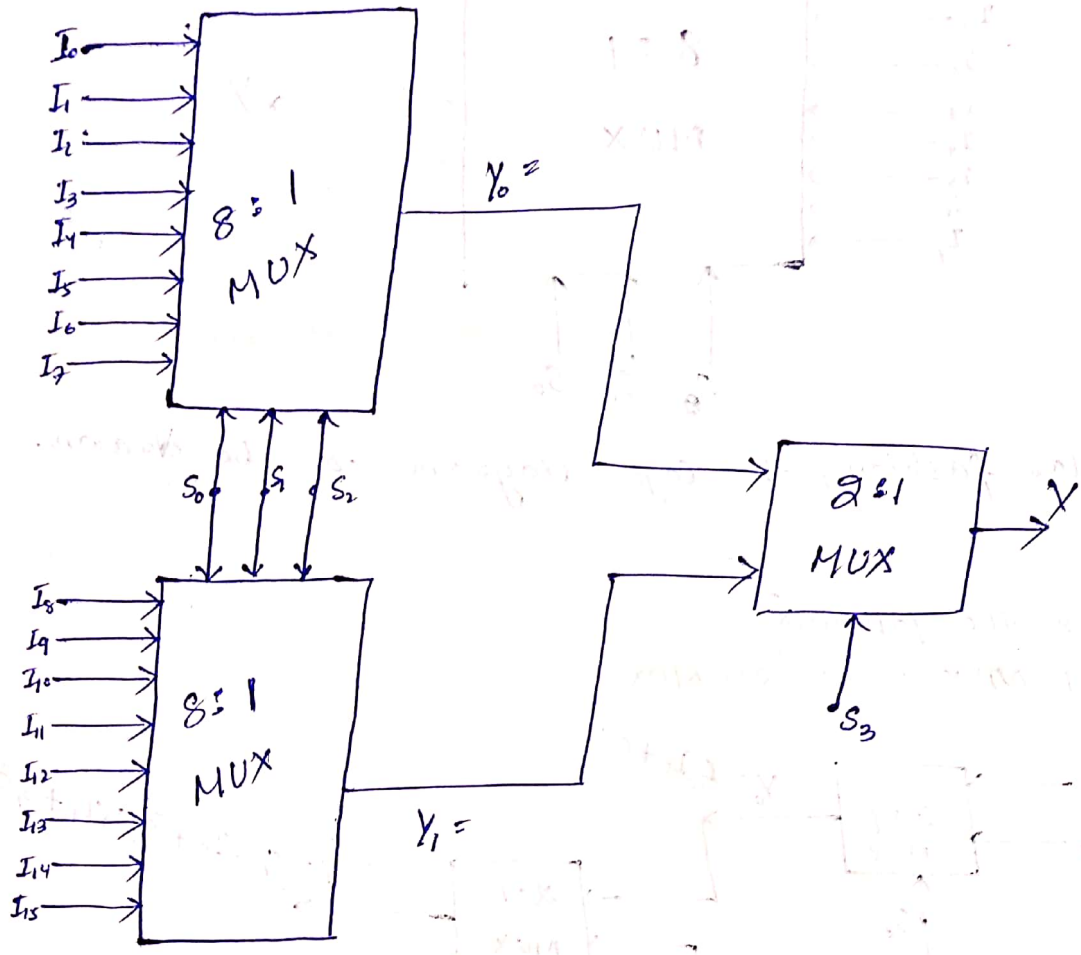
(a) 4:1 MUX using 2:1 MUX



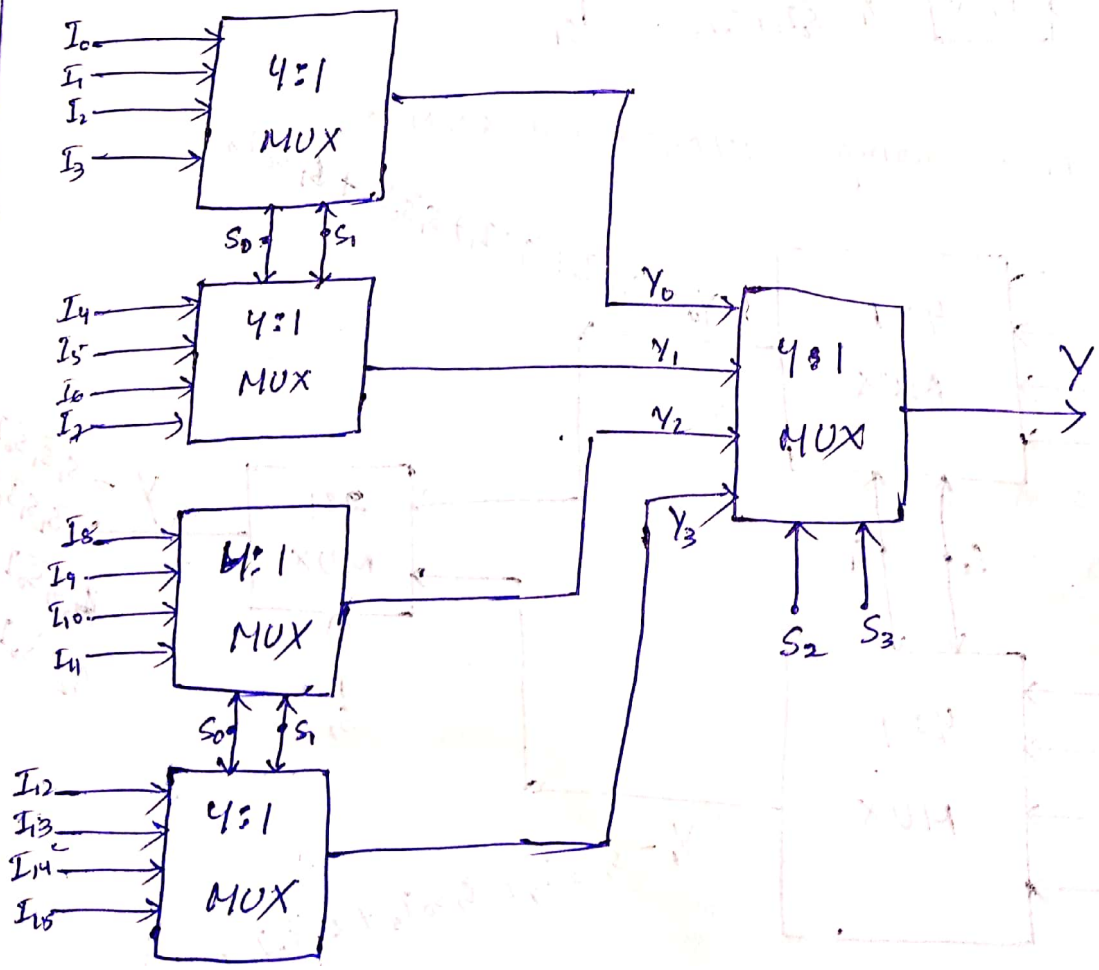
(b) 8:1 MUX using 4:1 MUX and 2:1 MUX.



© 16:1 MUX using 8:1 MUX

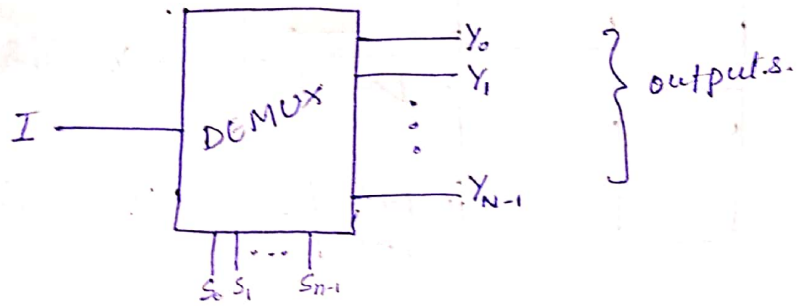


(d) 16:1 MUX using 4:1 MUX

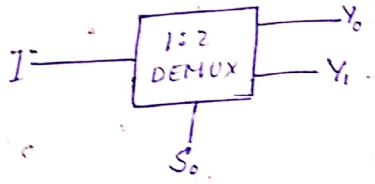


Demultiplexers:

* These are just reverse of multiplexers, i.e., data distributes
* There is only one input and there are several numbers of outputs. And select lines control the output flow.
* Here if no. of outputs = 'N' = 2^n then 'n' is no. of select lines.



1:2 DEMUX

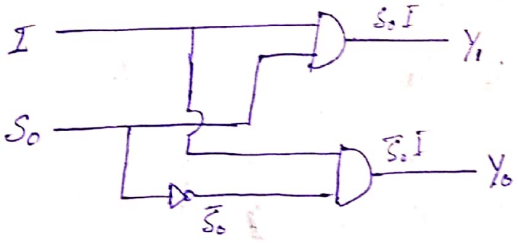


S_0	I	Y_0	Y_1
0	I	I	0
1	I	0	I

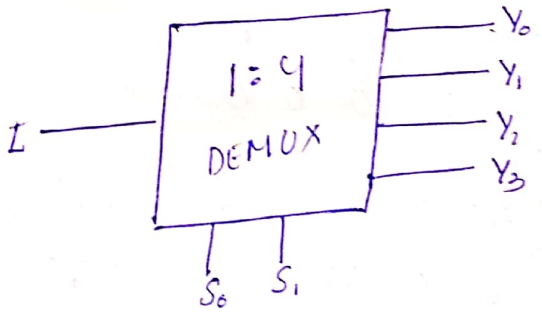
so $Y_0 = \bar{S}_0 I$

$Y_1 = S_0 I$

∴ In logic gates:



1:4 DEMUX

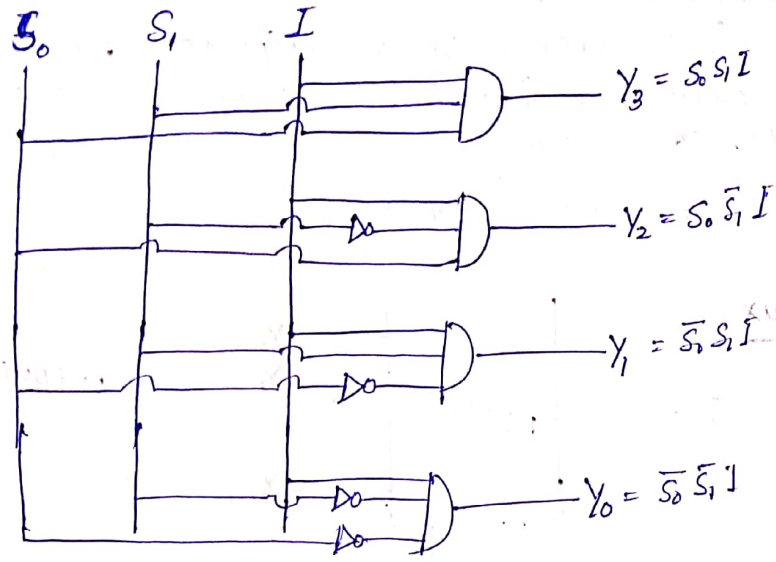


S_0	S_1	I	Y_3	Y_2	Y_1	Y_0
0	0	I	0	0	0	I
0	1	I	0	0	I	0
1	0	I	0	I	0	0
1	1	I	I	0	0	0

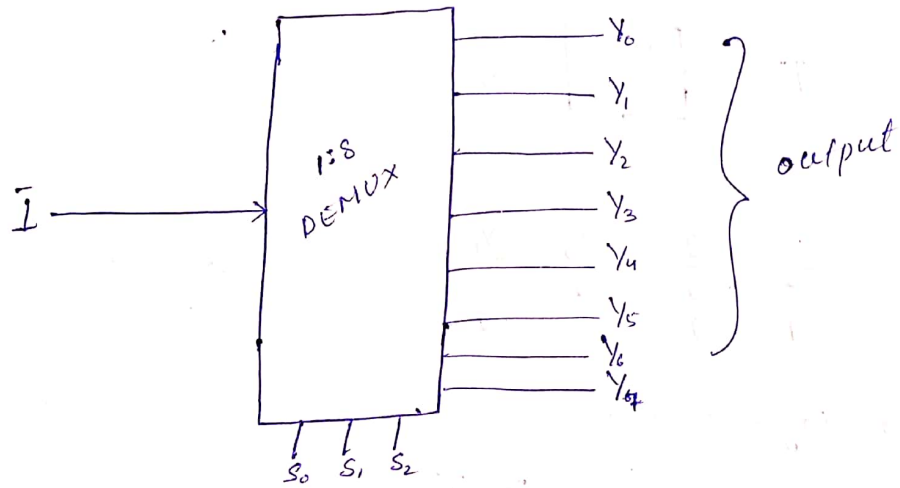
So, $Y_0 = \bar{S}_0 \bar{S}_1 I$, $Y_1 = \bar{S}_0 S_1 I$

$Y_2 = S_0 \bar{S}_1 I$, $Y_3 = S_0 S_1 I$

In form of logic gates:



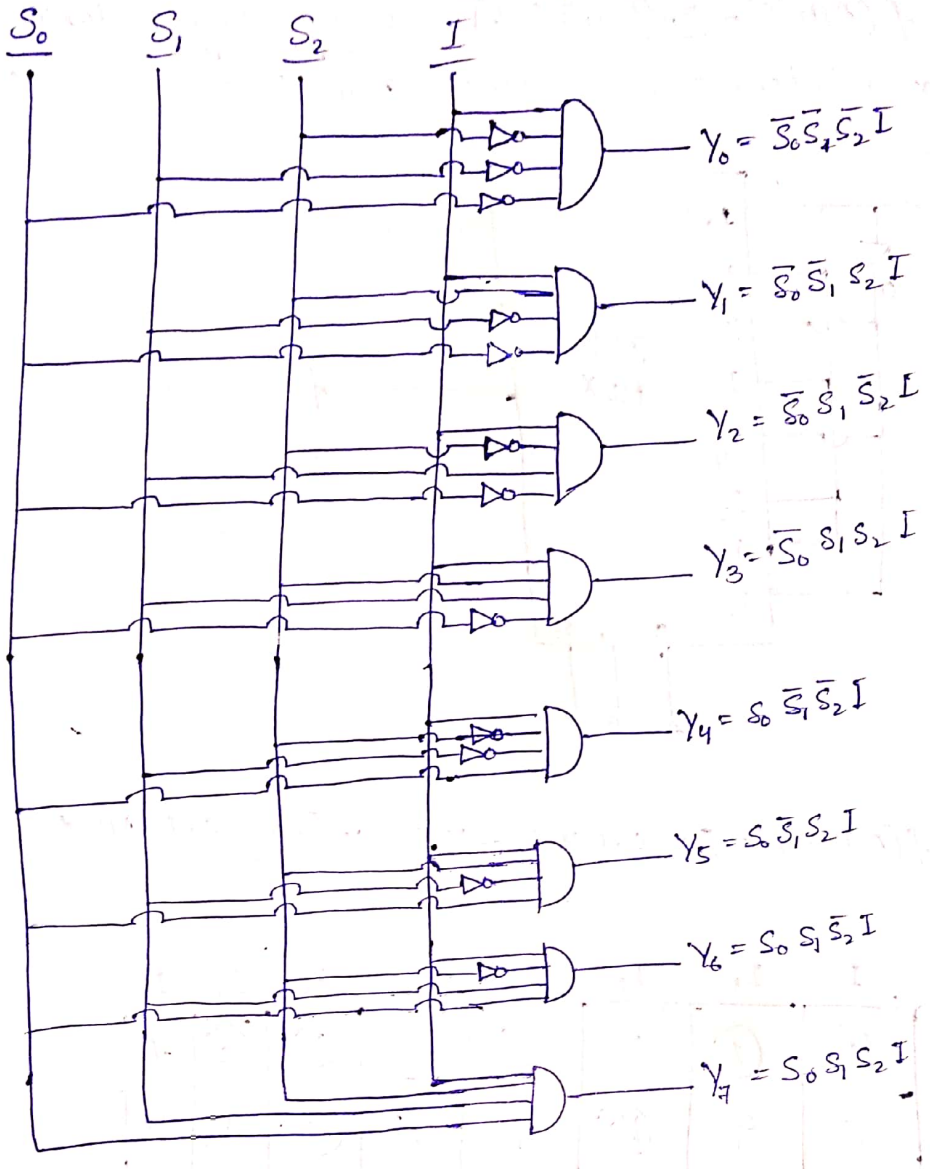
1 : 8 DEMUX



S_0	S_1	S_2	$\bar{I}$	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	I	0	0	0	0	0	0	0	I
0	0	1	I	0	0	0	0	0	0	I	0
0	1	0	I	0	0	0	0	0	I	0	0
0	1	1	I	0	0	0	0	I	0	0	0
1	0	0	I	0	0	0	I	0	0	0	0
1	0	1	I	0	0	I	0	0	0	0	0
1	1	0	I	0	I	0	0	0	0	0	0
1	1	1	I	I	0	0	0	0	0	0	0

$Y_0 = \bar{S_0} \bar{S_1} \bar{S_2} I$, $Y_1 = \bar{S_0} \bar{S_1} S_2 I$, $Y_2 = \bar{S_0} S_1 \bar{S_2} I$, $Y_3 = \bar{S_0} S_1 S_2 I$

$Y_4 = S_0 \bar{S_1} \bar{S_2} I$, $Y_5 = S_0 \bar{S_1} S_2 I$, $Y_6 = S_0 S_1 \bar{S_2} I$, $Y_7 = S_0 S_1 S_2 I$



Expressing Boolean Functions using MUX

→ First find the minterms of the expression given.

→ Then if any MUX can be used then directly analyse the no. of minterms given, respectively choose the MUX type. Then directly put 1 to the minterm inputs and then 0 to the remaining. The select lines are the variables.

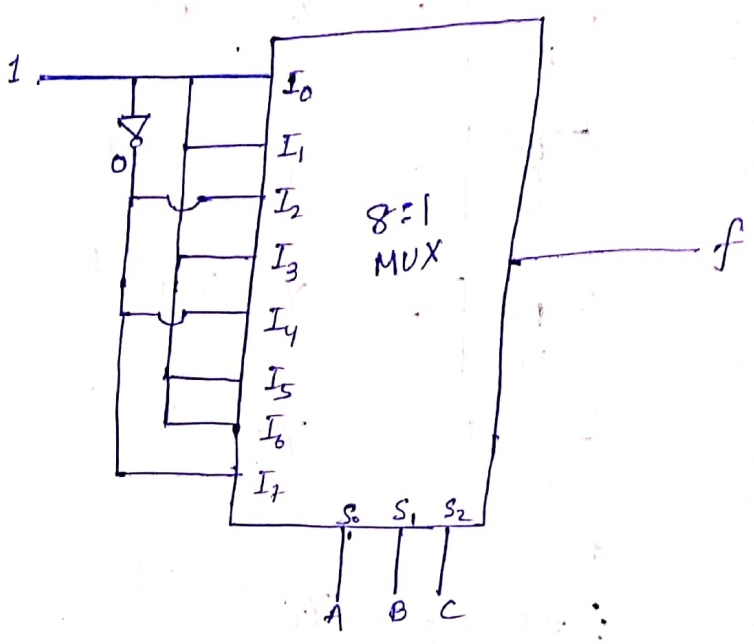
→ If any particular MUX is given then accordingly (N:1) then take N columns and two rows. Columns ($I_0, I_1, \cdots, I_{N-1}$) and rows ($A$ and $\bar{A}$). Follow the following rules:

ⓐ if both minterms in a column are circled ⇒ take it 1.
ⓑ if both minterms in a column are not circled ⇒ take it 0.
ⓒ if only upper row's minterm is circled ⇒ take it as $\bar{A}$
ⓓ if only lower row's minterm is circled ⇒ take it as A

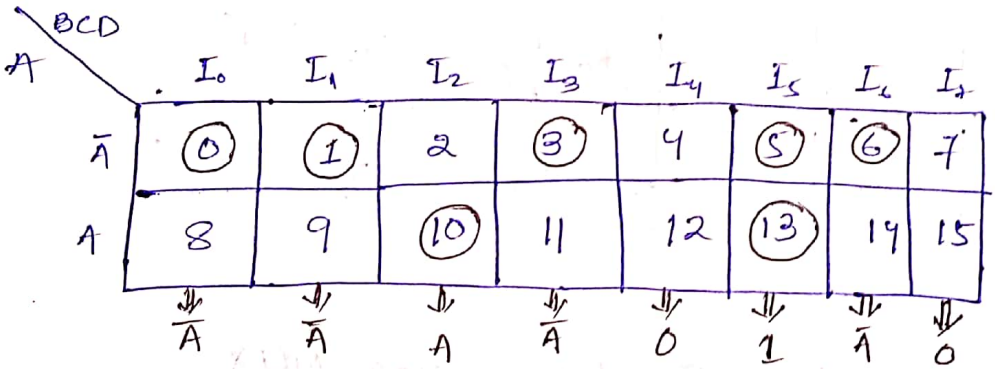
finally take the rest variables as the select lines.

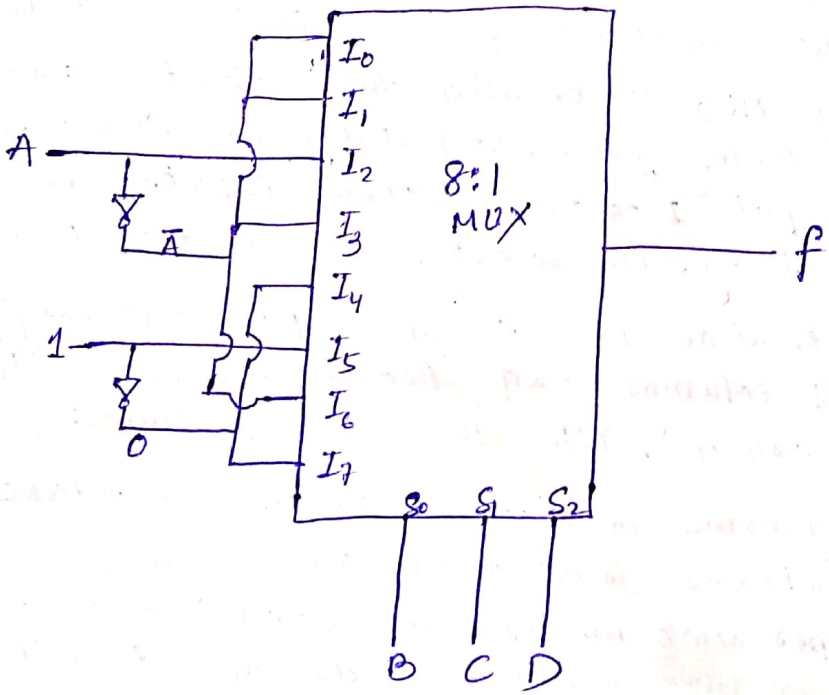
Eg: Design $f(A,B,C) = \sum m(0,1,3,5,6)$ using any suitable MUX.

as max. mintcom = 6 so total 8 inputs to be considered.

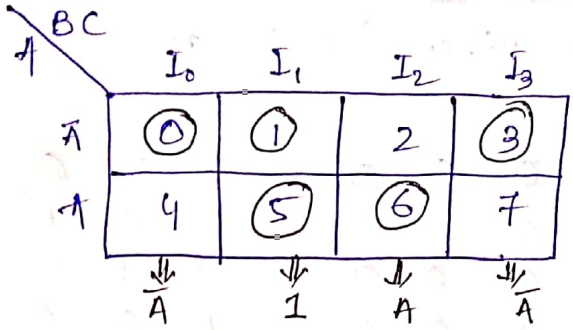


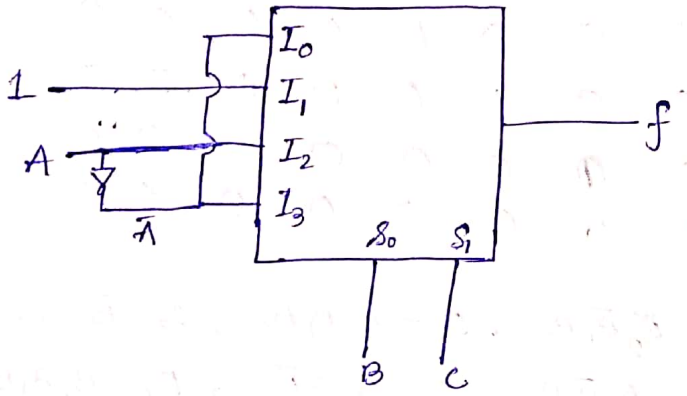
Eg: Design $f(A,B,C,D) = \sum m(0,1,3,5,6,10,13)$ using 8:1 MUX.





2. Design using 4:1 MUX for $f(A,B,C) = \sum m(0,1,3,5,6)$



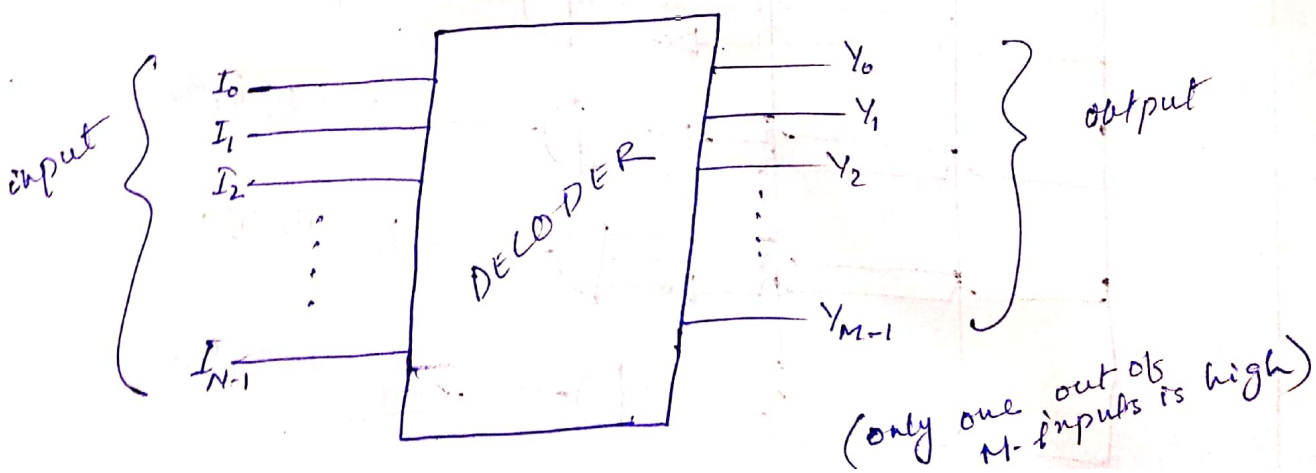


Applications of MUX

(i) Data routing
(ii) Logic function generator
(iii) Parallel to Series converter
(iv) Control sequencer.

Decoders:

* A decoder is a logic circuit that converts an N-bit binary input code M output lines such that only one output line is activated for each one of the possible combinations of inputs.



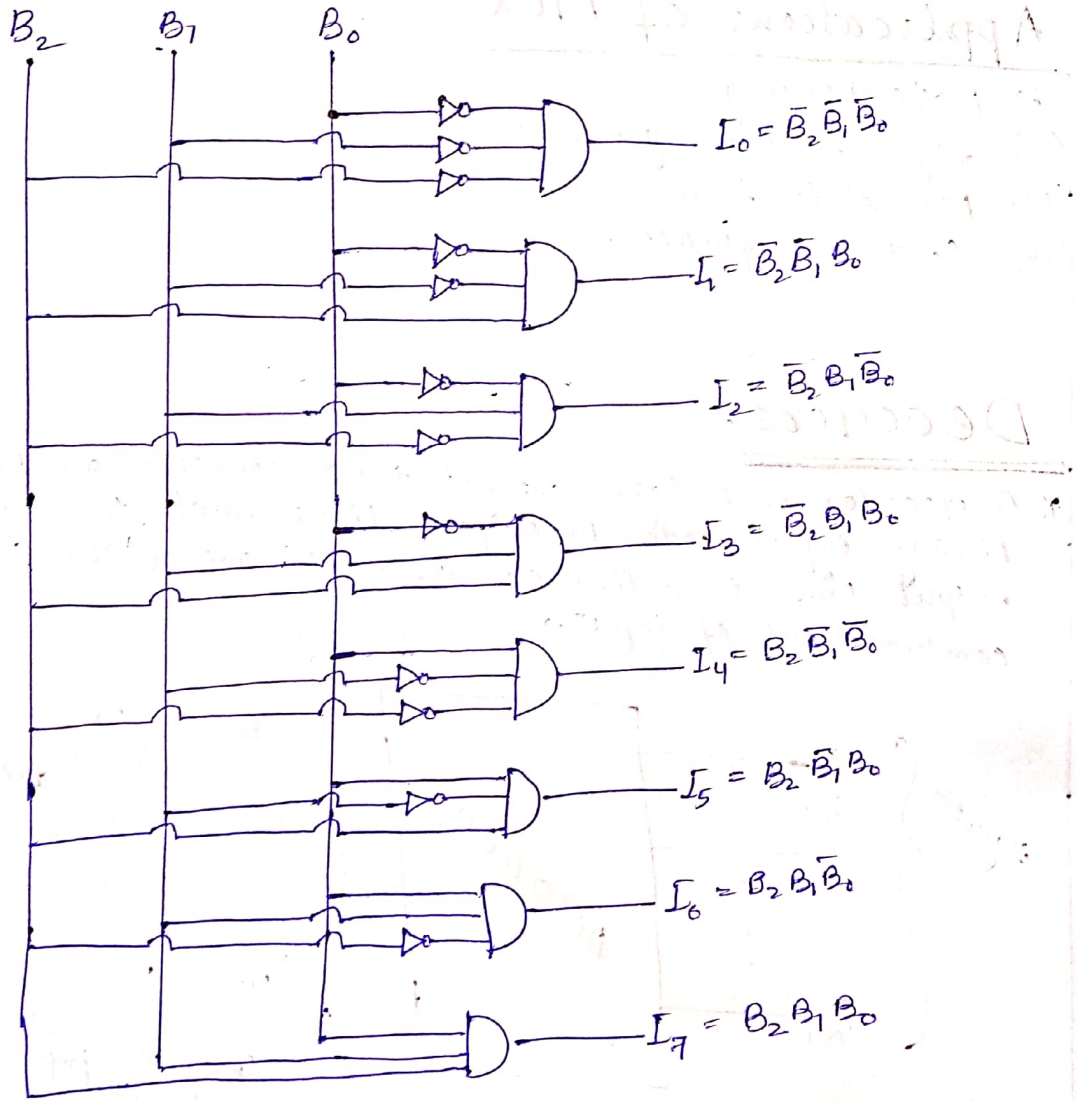
3 Line to 8 line Decoder

* It is a decoder with three inputs and eight outputs.

Inputs			Outputs							
B_2	B_1	B_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

$\therefore D_0 = \bar{B}_2 \bar{B}_1 \bar{B}_0$, $D_1 = \bar{B}_2 \bar{B}_1 B_0$, $O_2 = \bar{B}_2 B_1 \bar{B}_0$, $O_3 = \bar{B}_2 B_1 B_0$

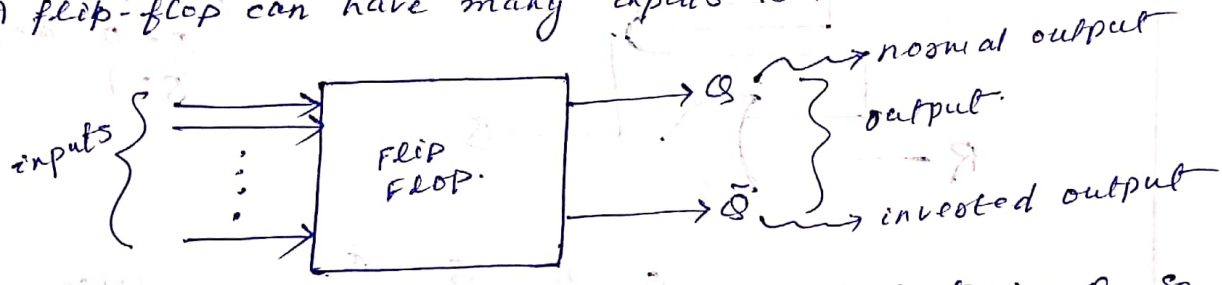
$O_4 = B_2 \bar{B}_1 \bar{B}_0$, $O_5 = B_2 \bar{B}_1 B_0$, $O_6 = B_2 B_1 \bar{B}_0$, $O_7 = B_2 B_1 B_0$



* This is nothing but the binary to octal converter.

FLIP-FLOPS

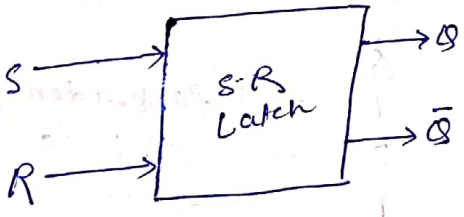
* It is a type of sequential logic circuit, which acts basically as a memory element, made up of assembly of logic gates.
* It stores one bit of data at a time (either 0 or 1)
* It acts as a bi-stable multivibrator, i.e., it can remain in either of states.
* A flip-flop can have many inputs but two outputs.

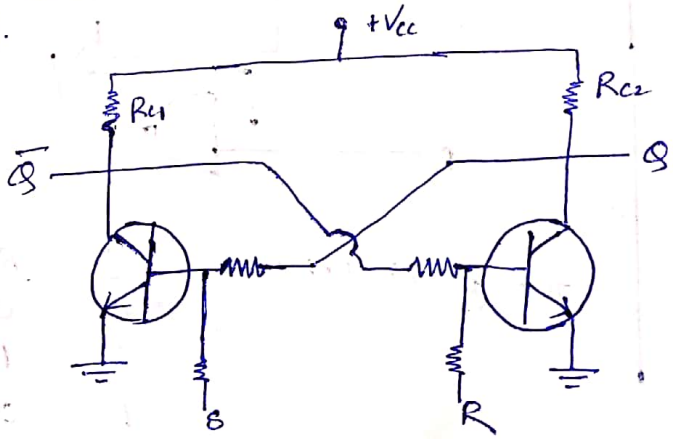


* Flip flop always refers to state of normal output, i.e., Q. So when Q=1 ⇒ flip flop is said to be HIGH or SET state and when Q=0 ⇒ flip flop is said to be LOW or RESET state.
* Flip flop is the name as it switches between possible output states.
* **Synchronous:** if a clock pulse is applied to a flip flop then it is recognized as synchronous or clocked flip flop
* **Asynchronous:** if a clock pulse is applied to a flip-flop then it is called asynchronous or unclocked flip-flop.
* **Latch:** it is used for certain flip flops which are non-clocked. as they latch on to a 1 or 0 immediately upon receiving the input pulse SET or RESET.

S-R LATCH

* It is the simplest flip flop called as SET-RESET latch, which has two inputs S and R.





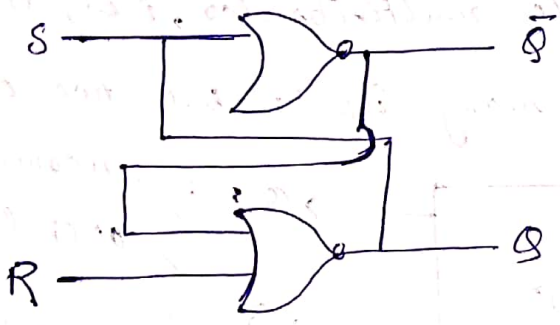
Truth table

S	R	Q	Comment
0	0	Q_0	— No Change
0	1	0	— RESET
1	0	1	— SET
1	1	-	— Forbidden undesired state

Q_0 = state of flip flop output before applying inputs.

NOR Gate SR Latch :

Two cross-coupled NOR gates.



Assume initialy
$Q = 0$ and
$\bar{Q} = 1$

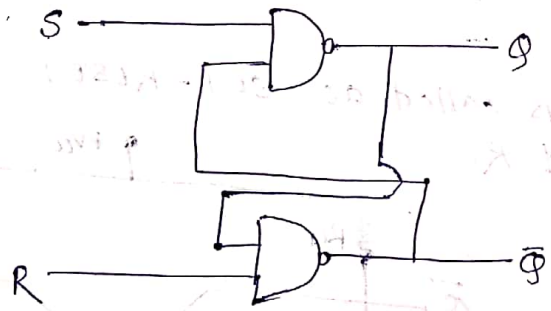
The truth table :

S	R	Q	$\bar{Q}$	
0	0	0	1	→ No change
0	1	0	1	→ RESET (as Q=0)
1	0	1	0	→ SET (as Q=1)
1	1	0	0	→ Forbidden

* It is called as active HIGH SR latch, because no chang state is when S & R are at 0 and either of the inputs are made high to transfer the state.

NAND GATE SR LATCH

* Two cross-coupled NAND gates can be used to obtain an active LOW SR Latch.



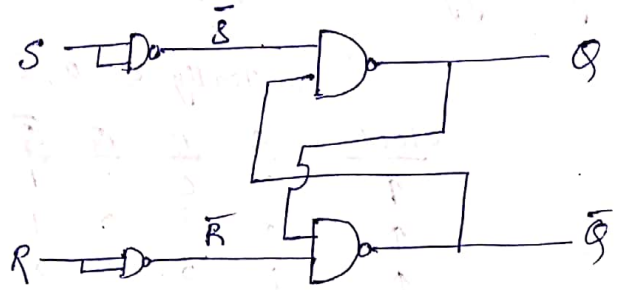
Assume initially
$Q = 0$ and $\bar{Q} = 1$

The truth table:

S	R	Q	$\bar{Q}$	
0	0	1	1	→ forbidden
0	1	1	0	→ SET (as Q=1)
1	0	0	1	→ RESET (as Q=0)
1	1	0	1	→ No change

* It is called active LOW because no change cond^n comes when S=R=1 and to transist to SET or RESET either of the inputs is made low. (0).

* An active HIGH SR latch is also possible with NAND gate but only with inverted inputs.



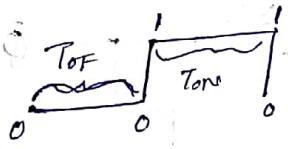
initially assume $Q = 0$, $\bar{Q} = 1$

The truth table:

S	R	Q	$\bar{Q}$	
0	0	0	1	→ No change
0	1	0	1	→ RESET
1	0	1	0	→ SET
1	1	ϕ	1	→ forbidden.

Clock Pulse:

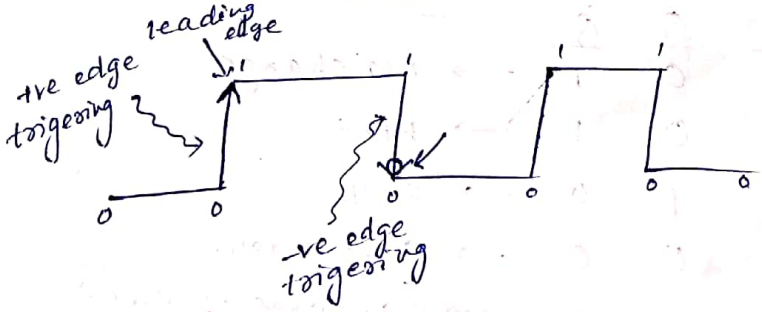
* Transition between two states '0' and '1' where each state has a certain time limit, i.e., T_{ON} while at '1' and T_{OFF} while at '0'; is called a clock pulse.



* Total time period, $T = T_{ON} + T_{OFF}$ [[0 → 0 → 1 → 1 → 0

The duty cycle, $$D = \frac{T_{ON}}{T_{ON} + T_{OFF}}$$

*



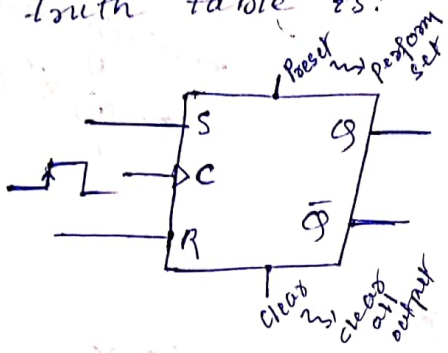
sequence of clock pulses called as 'clock trains'.

* —— or ——▷ represents +ve edge triggering and

——○ represents -ve edge triggering.

S-R Flip-Flop (Edge Triggered)

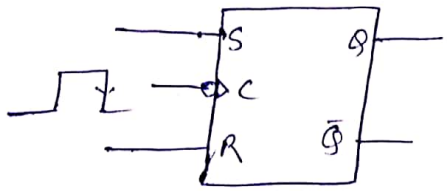
* Logic symbol for -ve edge triggered flip-flop and corresponding truth table is:



initially $Q = 0$, $\bar{Q} = 1$

clock	S	R	Q	$\bar{Q}$	
↑	0	0	0	1	→ No change
↑	0	1	0	1	→ RESET
↑	1	0	1	0	→ SET
↑	1	1	0	0	→ forbidden

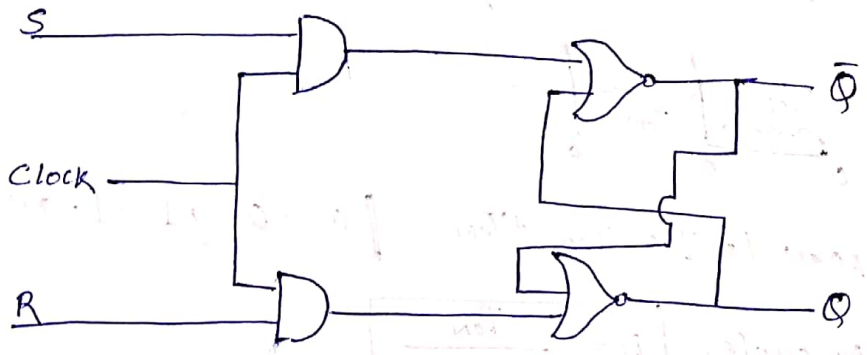
* The S & R inputs are called as synchronous control inputs because they affect the output only upon triggering and without clock, S & R are useless.
* The -ve edge triggered flipflop (SR) is as shown



initially $Q = 0$, $\bar{Q} = 1$

clock	S	R	Q	$\bar{Q}$	
↓	0	0	0	1	→ No change
↓	0	1	0	1	→ RESET
↓	1	0	1	0	→ SET
↓	1	1	0	0	→ forbidden

Using NOR:

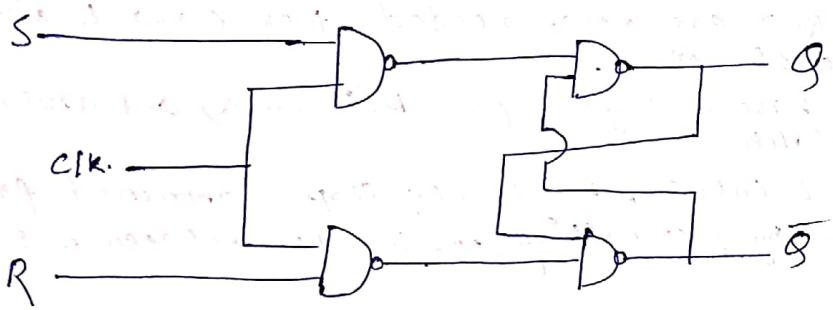


Initially $Q = 0$, $\bar{Q} = 1$

clk	S	R	Q	$\bar{Q}$	
↑	0	0	0	1	→ No change
↑	0	1	0	1	→ RESET
↑	1	0	1	0	→ SET
↑	1	1	0	0	→ forbidden.
↓	–	–	–	–	→ (no change of state)

whenever clock is triggered -ve, no change occurs to state.

Using NAND:



initially $Q=0, \bar{Q}=1$

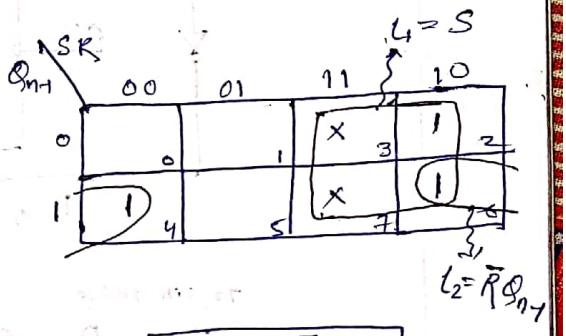
Clock	S	R	Q	$\bar{Q}$	
↑	0	0	0	1	→ No change
↑	0	1	0	1	→ RESET
↑	1	0	1	0	→ SET.
↑	1	1	✱	✱	→ forbidden
↓	–	–	–	–	→ (no change in state)

* So the characteristic/state table of SR flipflop can be found with inputs Q_{n-1} (previous state), S and R, S output Q_n (present state)

characteristic table

Q_{n-1}	S	R	Q_n
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	x
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	x

x ⇒ don't care or undefined



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

* from the characteristic table, the excitation table can be drawn with inputs (Q_{n-1} and Q_n), and outputs S and R

excitation table

Q_{n-1}	Q_n	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

x ⇒ don't care either 0 or 1

similarly drawing separate K-maps we obtain

$S = Q_n$

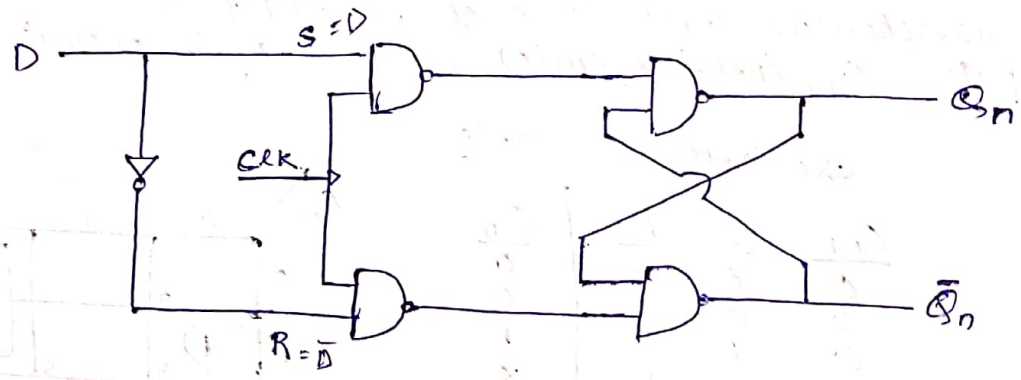
$R = \bar{Q}_n$

Gated D-Latch and D-Flip Flop (Edge triggered)

* In many applications of SR Latch, the input combinations S=R=1 and S=R=0 are never needed, then S and R always complement each other.
* Hence we can have a single input D (for Data) and device is called as D-Latch.
* Just like the D-latch, the D-Flip flop is obtained from SR flip-flop by just putting one inverter between S & R terminals

* Symbol

* the circuit

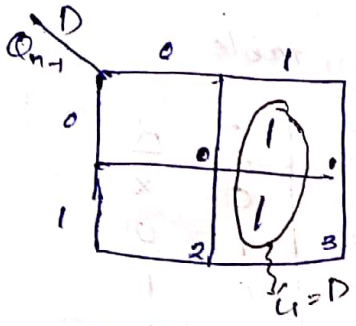


Truth table

Clk	D	Q_n	
↓	X	Q_{n-1}	(no change)
↑	0	0	(RESET)
↑	1	1	(SET)

* Characteristic/State Table

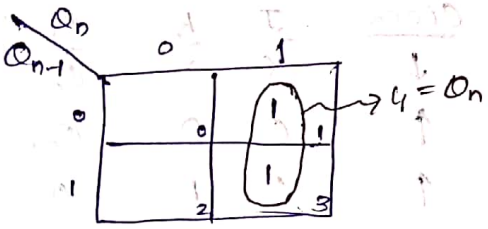
Q_{n-1}	D	Q_n
0	0	0
0	1	1
1	0	0
1	1	1



$\therefore$ $Q_n = D$ (characteristic equation)

Excitation table

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

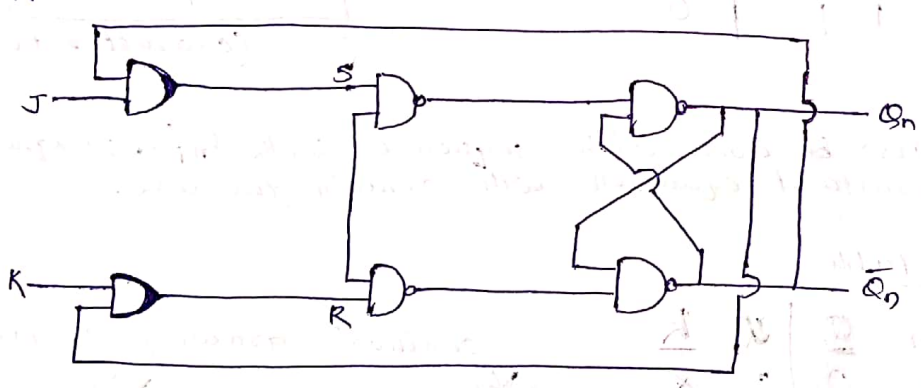


So $\boxed{D = Q_n}$ (excitation equation)

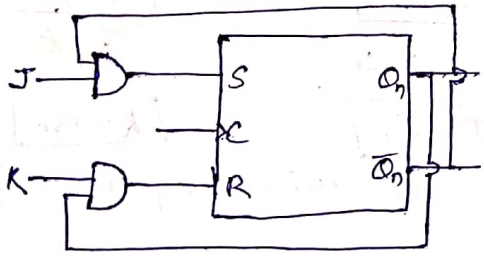
* Negative edge triggered flip flop also operates in same way as positive edge triggered, except the change of state takes place at negative going edge of clock pulse.

Edge Triggered J-K Flip-Flop :

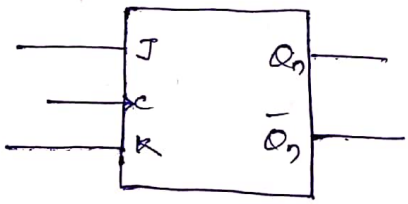
* JK is very versatile and also most widely used.
* JK's functioning is same as SR FlipFlop except the fact that here we don't get any invalid state.
* The requirements are $S = J\bar{Q}_n$ and $R = KQ_n$, so the circuit is



* symbol



or

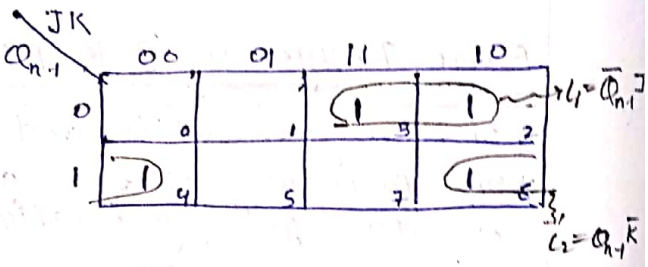


* Truth table

Clock	J	K	Q_n	$\bar{Q}_n$	
↓	X	X	Q_{n-1}	$\bar{Q}_{n-1}$	(No change)
↑	0	0	Q_{n-1}	$\bar{Q}_{n-1}$	(No change)
↑	0	1	0	1	(SET)
↑	1	0	1	0	(RESET)
↑	1	1	$\bar{Q}_{n-1}$	Q_{n-1}	(Toggle : complement of previous state)

* Characteristic table / State table

Q_{n-1}	J	K	Q_n
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



$$\therefore Q_n = \bar{Q}_{n-1} J + Q_{n-1} \bar{K}$$

(characteristic equation)

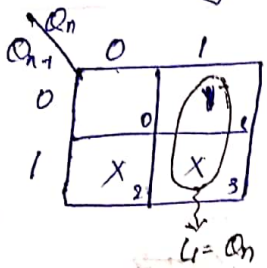
nb : It can also be done with sequence (J, K, Q_{n-1}) as inputs, however resultant equation will remain the same.

* excitation table

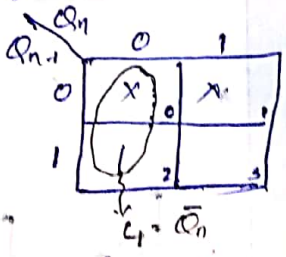
x ⇒ don't care

Q_{n-1}	Q_n	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

similarly drawing K-maps.



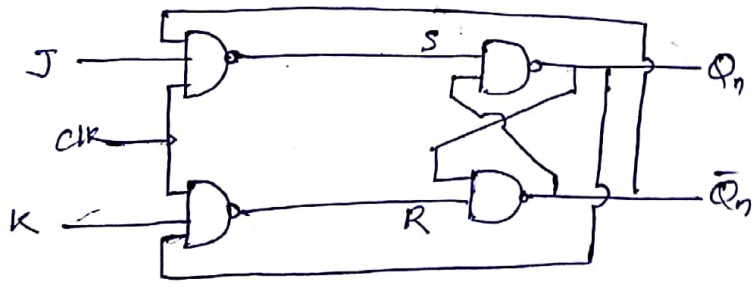
$$\therefore J = Q_n$$



$$K = \bar{Q}_n$$

(excitation equations)

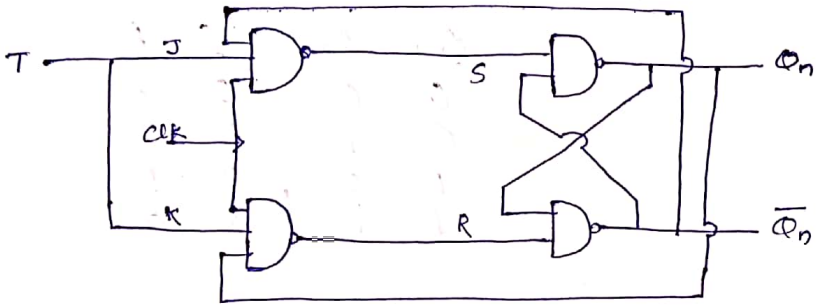
* The reduced circuitory can be :



Edge Triggered T-Flip Flop

* It is a single control input, labelled T for toggle.
* It is obtained by simply short circuiting the J and K inputs of a JK flip flop.
* In this case whenever T is low(0) then there is no change but when T is high(1), the output toggles, i.e, we get the complement of previous state.

* The circuit



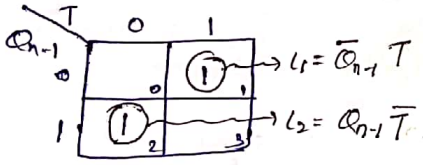
* Symbol:

* Truth table

Clock	T	Q_n	$\bar{Q}_n$	
↓	x	Q_{n-1}	$\bar{Q}_{n-1}$	(No change)
↑	0	Q_{n-1}	$\bar{Q}_{n-1}$	(No change)
↑	1	$\bar{Q}_{n-1}$	Q_{n-1}	(Toggle)

* Characteristic Table

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



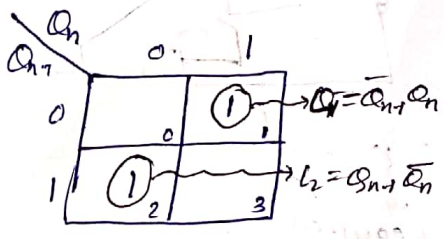
$\therefore Q_n = \bar{Q}_{n-1} T + Q_{n-1} \bar{T}$

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

(characteristic eqn)

* Excitation Table

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



$\therefore T = \bar{Q}_{n-1} Q_n + Q_{n-1} \bar{Q}_n$

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

(excitation equation)

Conversion Of Flip-Flops :

* To convert one type flip-flop into another type, a combinational circuit is designed such that if inputs of required FF are fed as input to combinational ckt, and resulting output is fed as input to actual FF, then output finally obtained is similar to output of required flip flop.

SR to D

Truth table of D-flipflop and excitation of SR flip flop

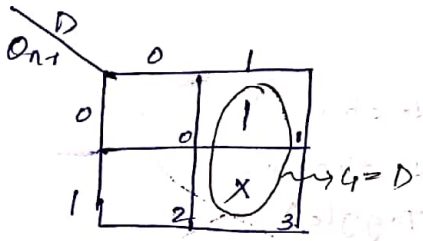
Q_{n-1}	D	Q_n
0	0	0
0	1	1
1	0	0
1	1	1

Q_{n-1}	Q_n	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

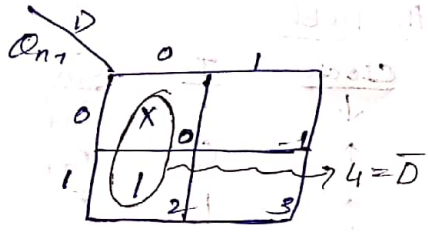
for the conversion we'll consider D as input & SR as output

	Inputs			Outputs	
from truth table	Q_{n-1}	D	Q_n	S	R
	0	0	0	0	x
	0	1	1	1	0
	1	0	0	0	1
	1	1	1	x	0

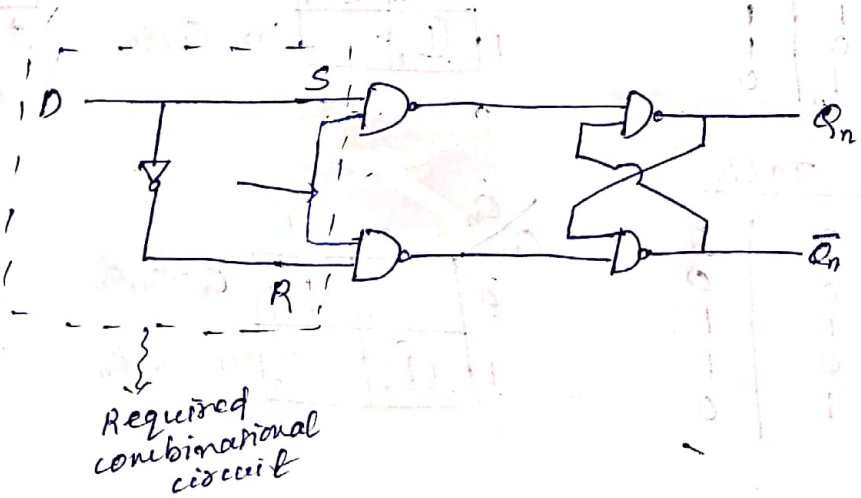
from excitation table



$S = D$



$R = \bar{D}$



SR to JK

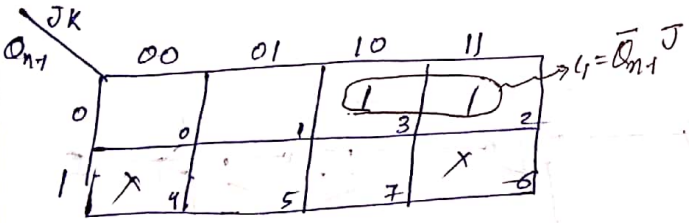
Truth table of JK flip flop and excitation table of SR

Q_{n-1}	J	K	Q_n
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

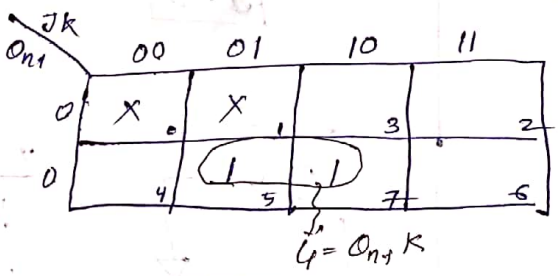
Q_{n-1}	Q_n	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

We'll consider JK as input and SR as output

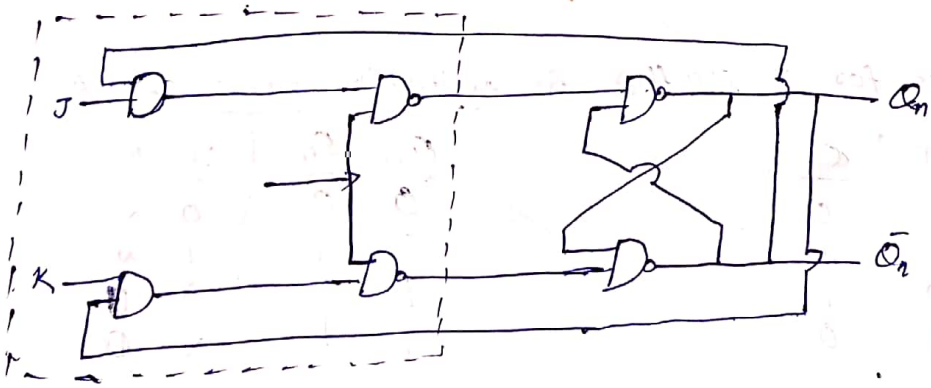
Inputs				Outputs	
Q_{n-1}	J	K	Q_n	S	R
0	0	0	0	0	X
0	0	1	0	0	X
0	1	0	1	1	0
0	1	1	1	1	0
1	0	0	1	X	0
1	0	1	0	0	1
1	1	0	1	X	0
1	1	1	0	0	1



$S = \bar{Q}_{n-1} J$



$R = Q_{n-1} K$



JK to D

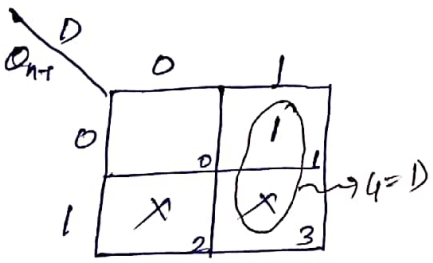
Truth table of D flip-flop and excitation table of JK

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

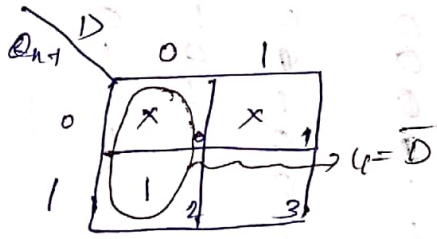
Q_{n-1}	Q_n	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

input as D and output as JK.

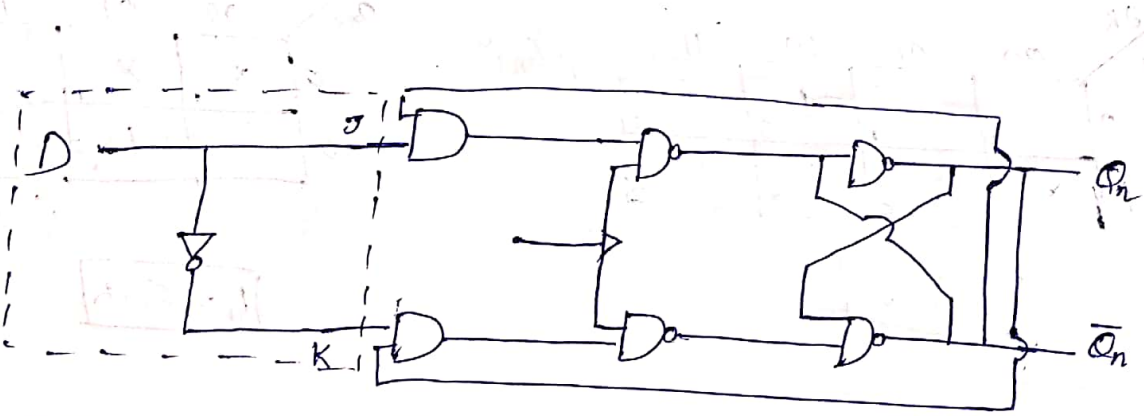
Q_{n+1}	D	Q_n	J	K
0	0	0	0	x
0	1	1	1	x
1	0	0	x	1
1	1	1	x	0



$J = D$



$K = \bar{D}$



JK to T

Truth table for T flipflop & excitation table of JK

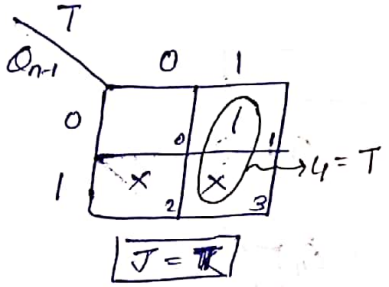
Q_{n+1}	T	Q_n
0	0	0
0	1	1
1	0	1
1	1	0

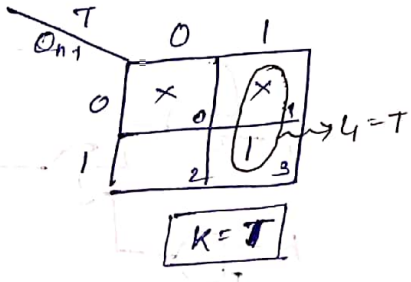
Q_{n-1}	Q_n	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

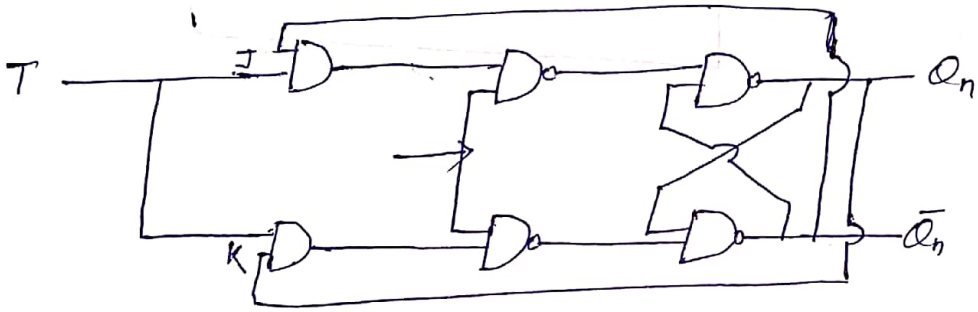
we take T side as input and JK as output

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

J	K
0	X
1	X
X	0
X	1







T to JK

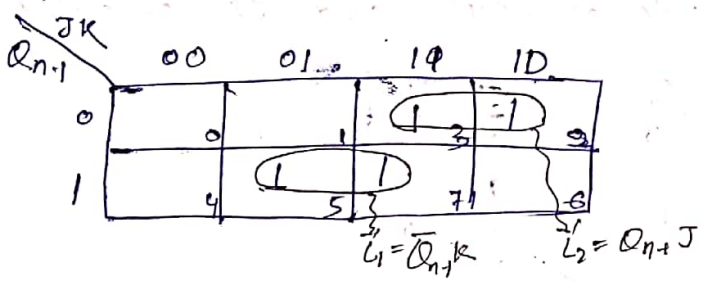
Truth table of JK and excitation table of T

Q_{n-1}	J	K	Q_n
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

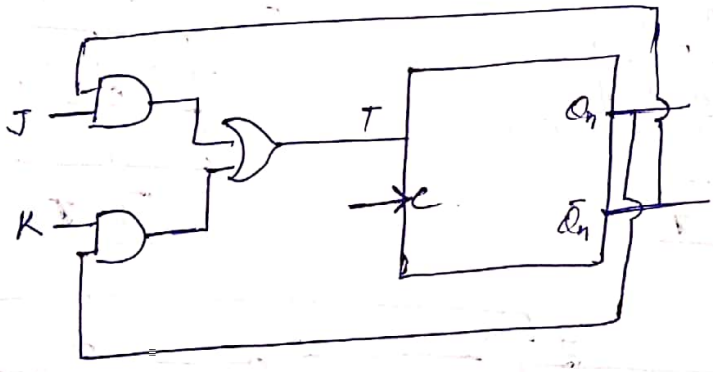
Q_{n-1}	Q_n	T
0	0	0
0	1	1
1	0	0
1	1	1

taking JK as inputs at T as outputs

Q_{n-1}	J	K	Q_n	T
0	0	0	0	0
0	0	1	0	0
0	1	0	1	1
0	1	1	1	1
1	0	0	1	0
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1



$$T = \bar{Q}_{n+1} K + Q_{n+1} J$$



Counters

* A digital counter is a set of flip-flops (FFs) whose state change in response to pulses applied at the input to the counter.
* A counter is used to count pulses, because the flip flops are interconnected such that their combined state at any time is the total number of pulses that have occured upto that time.
* It can also be used as a frequency divider to obtain waveforms with frequencies that are specific fractions of clock frequency.
* other uses:
  - → timing functions as in digital watches
  - → to create time displays.
  - → to produce non-sequential binary counts
  - → to generate pulse trains
  - → to act as frequency counters.

counter types

- Asynchronous Counters (Ripple counters)
- Synchronous Counters

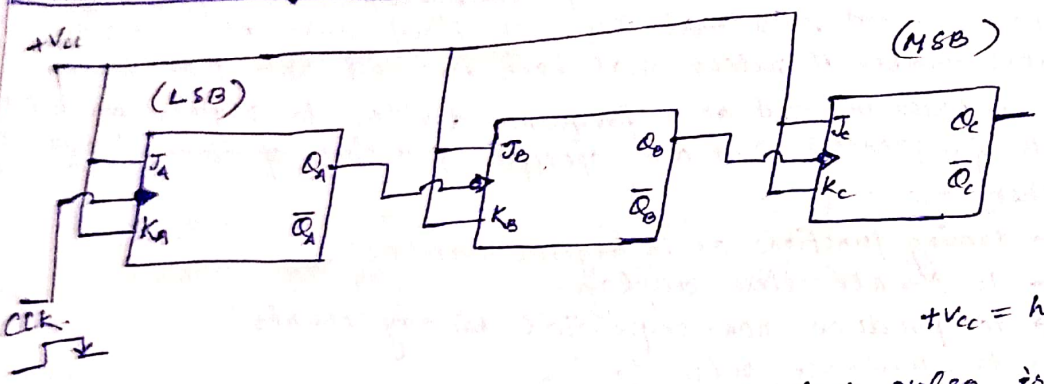
* A counter may be an up-counter or a down-counter. The up-counter counts in upward direction, i.e, 0, 1, 2, .., N. But the down-counter counts in downward direction, ie, N, N-1, N-2, ..., 1, 0.
* Each count is called as 'state' of counter. The no. of states through which the counter passes before returning to the starting state is called the '**modulus**' of the counter.

modulus = no. of distinct states that the counter counts. including zero. that counter can store.

* The LSB of any counter is that bit which changes most often. In ripple counters, the LSB is the Q output of the flip flop to which external clock is applied.

Asynchronous Counters:

4-bit binary counters: (up)



+Vcc = high input (=1

* As edge triggering is -ve so whenver clock pulse is given, the 1st output Q_A changes from prev. state and accordingly its consequtive outputs also change, i.e., An will change only if its pre-flipflop's output toggles from 1 to 0.
* Eg: Q_B will change if Q_A changes from 1 to 0.
* This change happens so because given trigerring is -ve.
* In this way the series of flipflops counts the number of counts of clockpulse in up direction from 0 to 15.

Q_D	Q_C	Q_B	Q_A	CLK	
0	0	0	0	0th	initial state
0	0	0	1	1st	
0	0	1	0	2nd	
0	0	1	1	3rd	
0	1	0	0	4th	
0	1	0	1	5th	
0	1	1	0	6th	
0	1	1	1	7th	
1	0	0	0	8th	
1	0	0	1	9th	
1	0	1	0	10th	
1	0	1	1	11th	
1	1	0	0	12th	
1	1	0	1	13th	
1	1	1	0	14th	
1	1	1	1	15th	

* After 15th clock when we apply 16th clock pulse, we see all bits change and the system moves to initial state again, i.e, to (0 0 0).
* It is called as MOD16 asynchronous counter. or MOD 16 ripple counter

MOD-3 ripple counter (up)

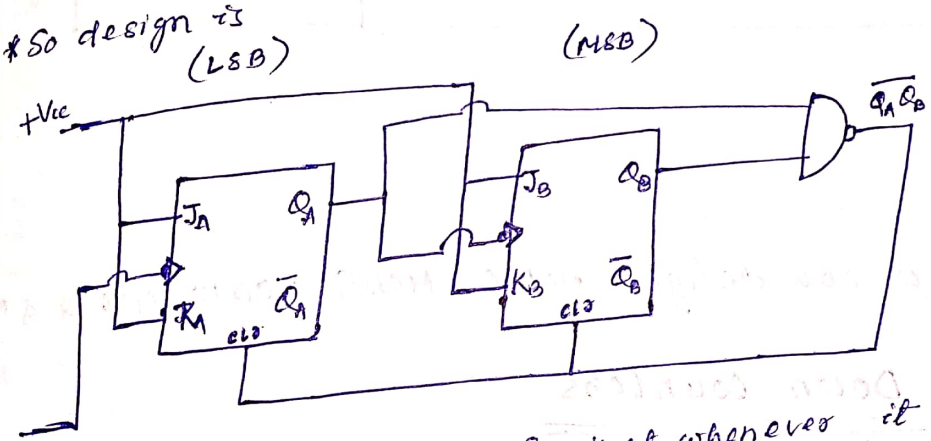
* Here mod 3 speaks that counter will count only 3 clock pulses, i.e, 0th, 1st and 2nd; so the design needs a bit of modified
* $3 < 4 = 2^2$ so 2 flip flops are required.

Q_B	Q_A	clk
0	0	0th
0	1	1st
1	0	2nd
1	1	3rd

→ here we need (0 0) not (1 1) so needs to be eliminated.

→ To eliminate, we see both Q_A and Q_B are high in 3rd state which needs to be changed, so both are passed to a NAND gate which goes to CLEAR terminal of both the flipflops.

* So design is

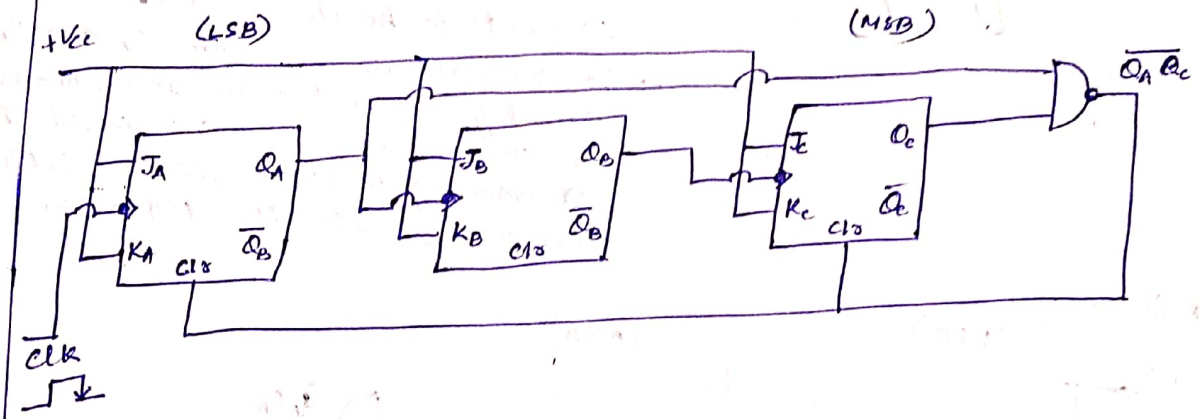


* Work of clear-terminal is that whenever it recieves '0' as input then it resets the flipflop to '0' from present state.
* Rest logic is same as 4-bit ripple counter or MOD16 ripple counter
* When the 3rd clock applied here, then it counts (0 0 0) and hence, the counting goes to 0. then 1 and 2 and then again 0 and 1 and so on..

MOD 5 ripple counter (UP)

* So a counter needs to be designed which counts upto 4th clock pulse, i.e, 5 clock pulses.
* As $5 < 2^3$ so 3 flipflops to be used.

Q_C	Q_B	Q_A	Clk
0	0	0	0th
0	0	1	1st
0	1	0	2nd
0	1	1	3rd
1	0	0	4th
1	0	1	5th → this need to be (0 0 0), so as we see Q_A and Q_C are high here so needs to given through NAND to clear terminals of flipflop A & C.



nb Similarly we can design MOD 6, MOD 9, MOD 10, MOD 12 & MOD 14

Design of Down Counters

* It is counters where counting starts from the topmost value that can be counted. Eg: MOD 3 down counter will 1st count 2 then 1 then 0 then 2 then 1 and so on.
* Design of down counter involves a slight change than that of its up counter version.
* In down counter, clock pulse applied is +ve edge triggered and the 2nd flipflop onwards use the clockpulse coming from complemented output of its previous flipflop.

Eg: flipflop B will have a clock pulse from $\overline{Q_A}$ and flipflop C from $\overline{Q_B}$ and so on.

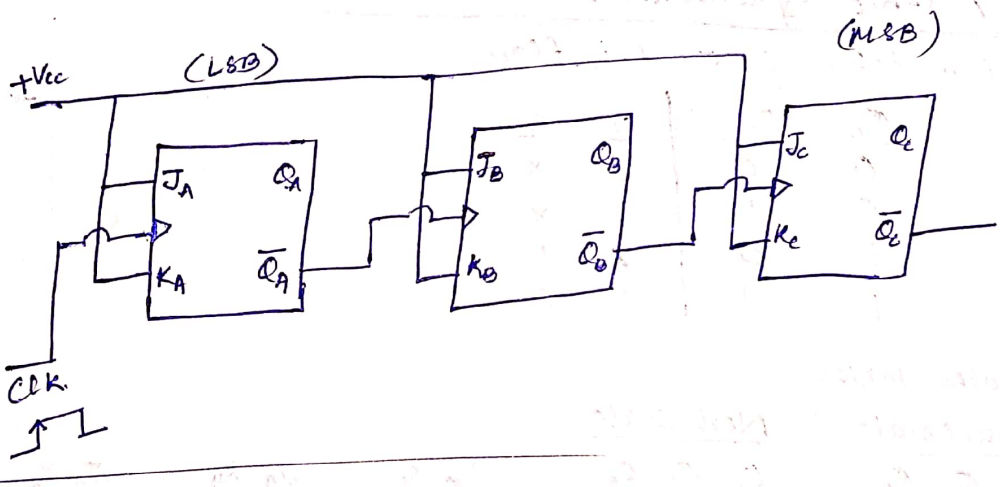
3 bit down counter :

* 3 bits means mod 8, i.e, 3 flip flops.

$\bar{Q}_C$	$\bar{Q}_B$	$\bar{Q}_A$	Clk
1	1	1	0th
1	1	0 (-ve)	1st
1	0 (-ve)	1 (+ve)	2nd
1	0	0	3rd
0	1	1	4th
0	1	0	5th
0	0	1	6th
0	0	0	7th

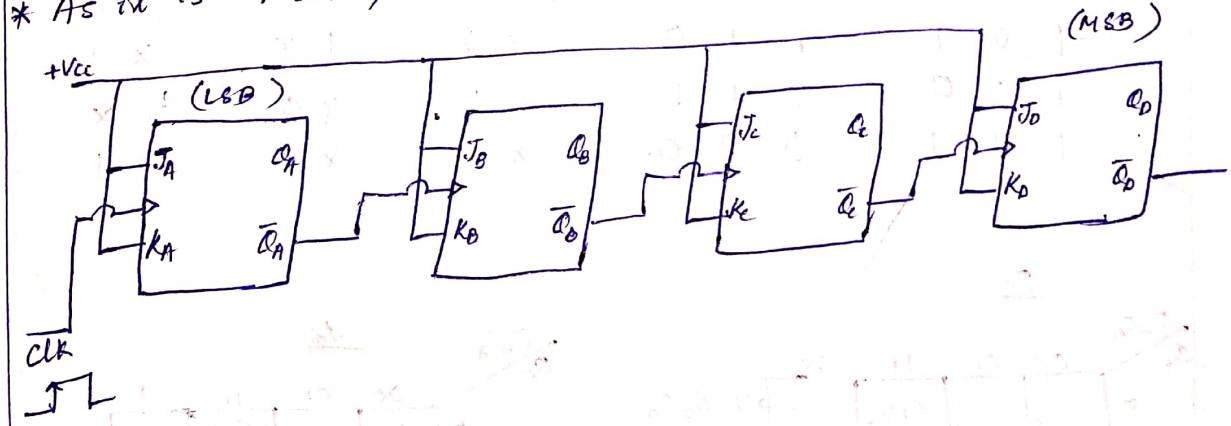
Here toggle of states involve just the opposite as that in case of the down counters, because here a +ve edge trigger is given

* from 8th clockpulse we will be again getting (111) which is the initial state.



4 bit down counter :

* Very similar to 3 bit down counter, it also counts in downward direction, i.e, here it counts from 15 to 0.
* Logic is same as 3 bit down ripple counter.
* As it is 4 bit, so no. of flipflops = 4.



Synchronous Counters

* In synchronous counters each flip flop is supplied with individual clock pulses and the same clock pulse that is given to the 1st flipflop
* The problem with asynchronous flip flops is that it cannot operate properly at high clock frequencies and they are also very slow because all FFs are interdependant.
* This is overcome in synchronous or parallel counters.
* **Steps to design:**
  - → 1st draw excitation table of chosen flip flop type to be used.
  - → Draw a table with Present state, Next state for flip flop then for each FF-parameter write down values using excitation table
  - → Draw k-maps for FF-parameters in the table considering inputs to be the Present state and then find equations for each.
  - → from that equations design the flip flop.
* In general we use JK-FF.

MOD 8 / 3-bit synchronous counter (UP)

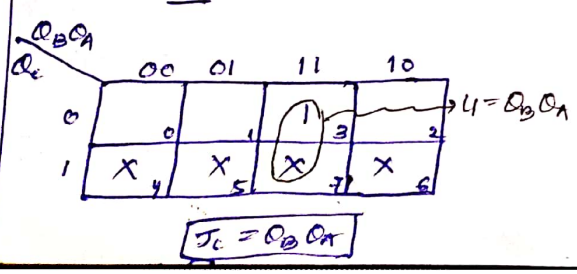
* excitation table of JK flip flop.

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

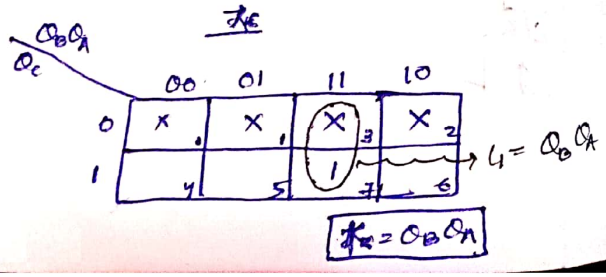
* The states tables:

Present state			Next state			J_C K_C	J_B K_B	J_A K_A
Q_C	Q_B	Q_A	Q_C	Q_B	Q_A			
0	0	0	0	0	1	0 X	0 X	1 X
0	0	1	0	1	0	0 X	1 X	X 1
0	1	0	0	1	1	0 X	X 0	1 X
0	1	1	1	0	0	1 X	X 1	X 1
1	0	0	1	0	1	X 0	0 X	1 X
1	0	1	1	1	0	X 0	1 X	X 1
1	1	0	1	1	1	X 0	X 0	1 X
1	1	1	0	0	0	X 1	X 1	X 1

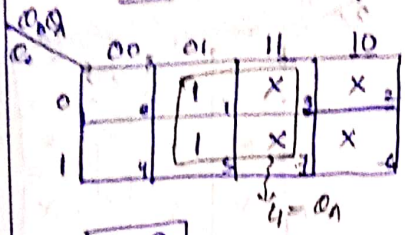
J_C



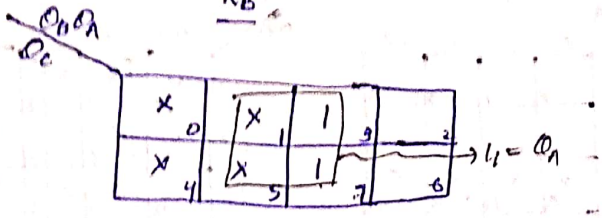
J_B



J_A

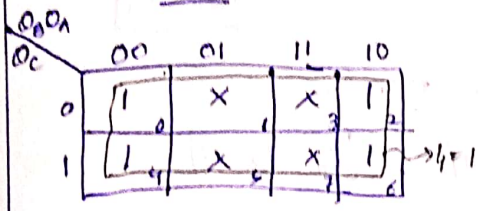


K_B



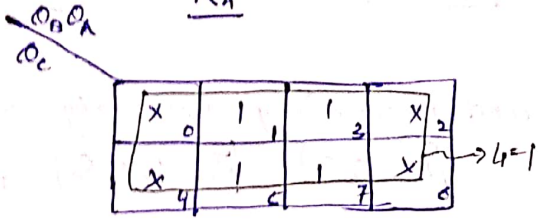
$K_B = Q_A'$

J_A



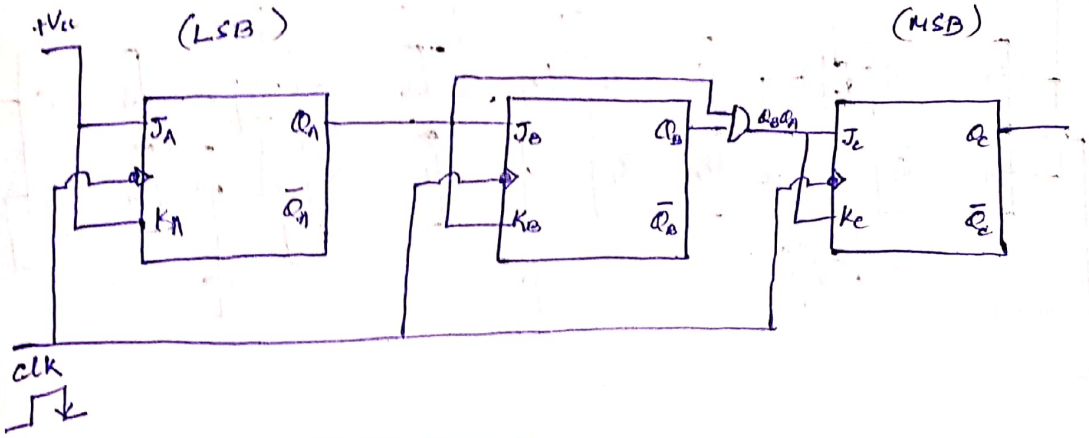
$J_A = 1$

K_A



$K_A = 1$

* The design is

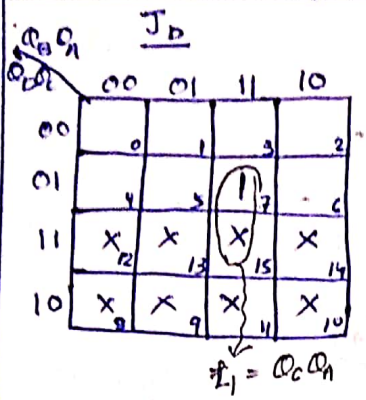


DECADE / MOD 10 Synchronous Counters

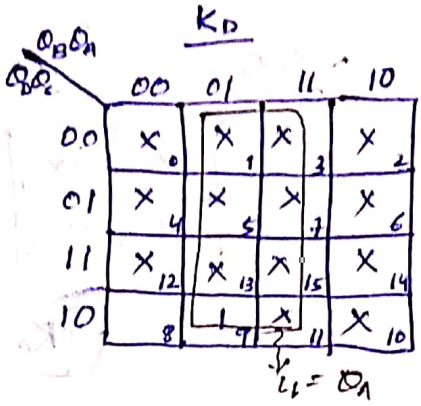
* same JK to be used. and as $10 < 2^4 \Rightarrow$ 4 FF to be used.
* states tables:

Present state				Next state				J_D	K_D	J_C	K_C	J_B	K_B	J_A	K_A
Q_D	Q_C	Q_B	Q_A	Q_D	Q_C	Q_B	Q_A								
0	0	0	0	0	0	0	1	0	x	0	x	0	x	1	x
0	0	0	1	0	0	1	0	0	x	0	x	1	x	x	1
0	0	1	0	0	0	1	1	0	x	0	x	x	0	1	x
0	0	1	1	0	1	0	0	0	x	1	x	x	1	x	1
0	1	0	0	0	1	0	1	0	x	x	0	0	x	1	x
0	1	0	1	0	1	1	0	0	x	x	0	0	x	x	1
0	1	1	0	0	1	1	1	0	x	x	0	x	0	1	x
0	1	1	1	1	0	0	0	1	x	x	1	x	1	x	1
1	0	0	0	1	0	0	1	x	0	0	x	0	x	1	x
1	0	0	1	0	0	0	0	x	1	0	x	0	x	x	1

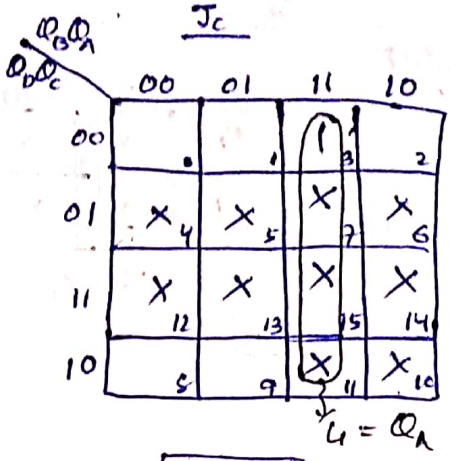
* for rest of the states, i.e. from (1010) to (1111), put all inputs of FFs to be x (don't care)



$J_D = Q_C Q_A$



$K_D = Q_A$

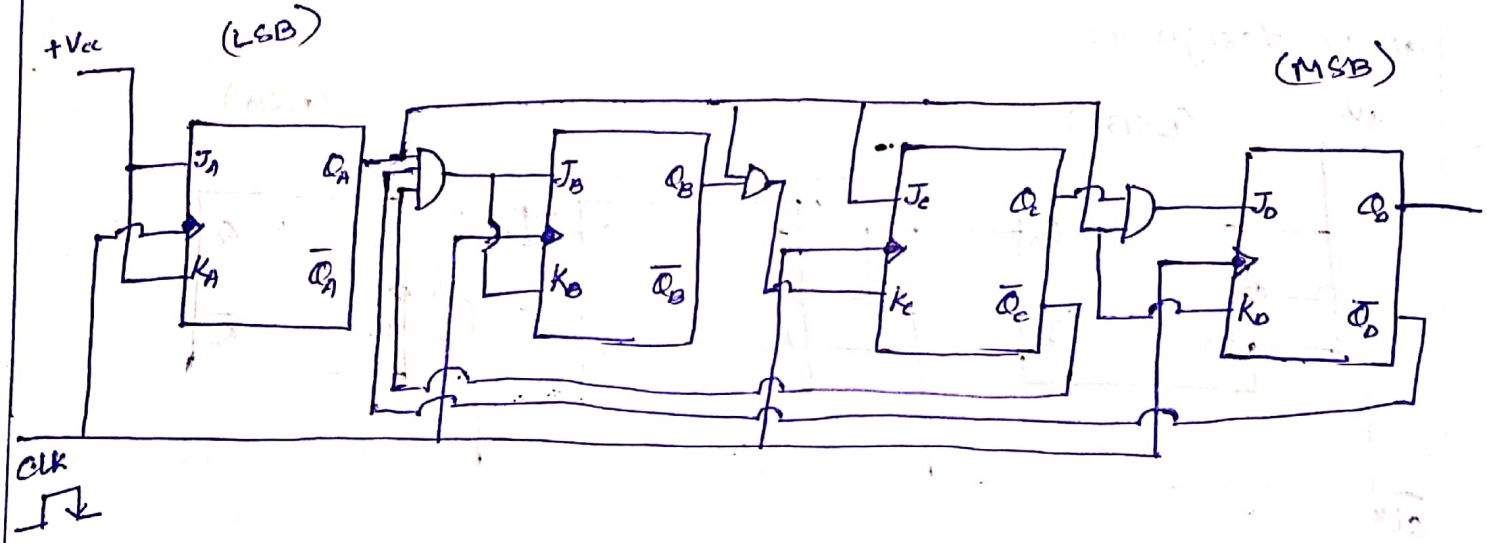


$J_C = Q_A$

and proceeding similiarly we will obtain

$K_C = Q_B Q_A$, $J_B = \bar{Q}_D \bar{Q}_C Q_A$, $J_D = Q_A$, $J_A = K_A = 1$

* So the design is:

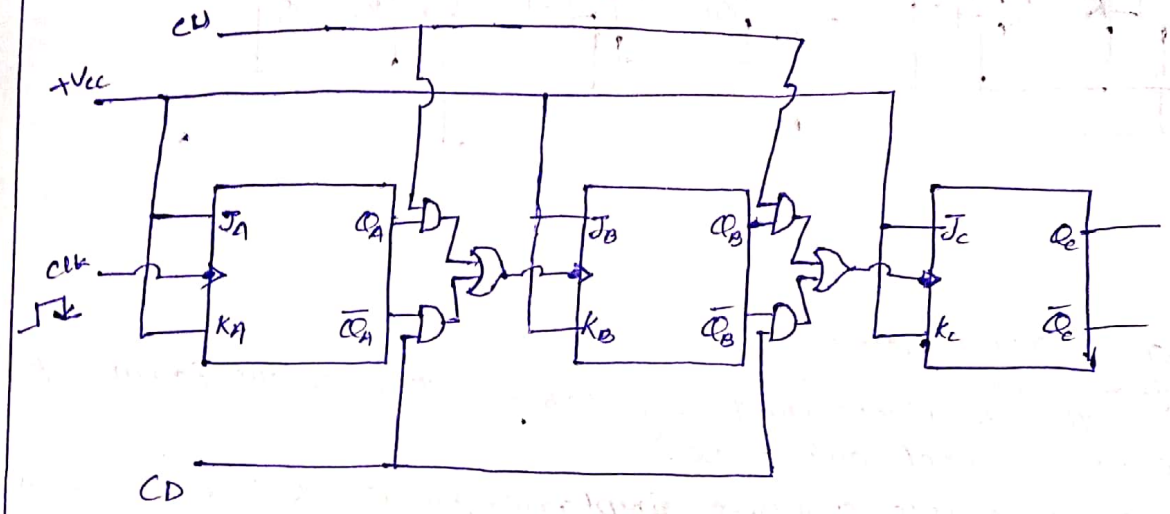


N.B similiarly MOD 6, MOD 7, MOD 9, MOD 12 and MOD 14 can be designed.

Asynchronous up/down counter:

* An asynchronous counter that can count in both upward and downward direction is called up/down asynchronous counter.
* There are two extra inputs called CU (Control Up) and CD (Control Down) with the relation $CU = \overline{CD}$.
* When $CU = 1$, the counter counts upward direction and when $CU = 0 \Rightarrow CD = 1$, the counter counts downward direction.

Eg 3-bit up-down counter (asynchronous & -ve edge triggered)



nb: for +ve edge triggering just put them (CU & CD) in opposite places.

Synchronous up/down counter

* In this case, let $M = CU$ so, $\overline{M} = CD$. When $M = 1$ it will count upward and when $M = 0$, it will count downward.
* 1st table needs to be drawn then, K-maps, then the design.

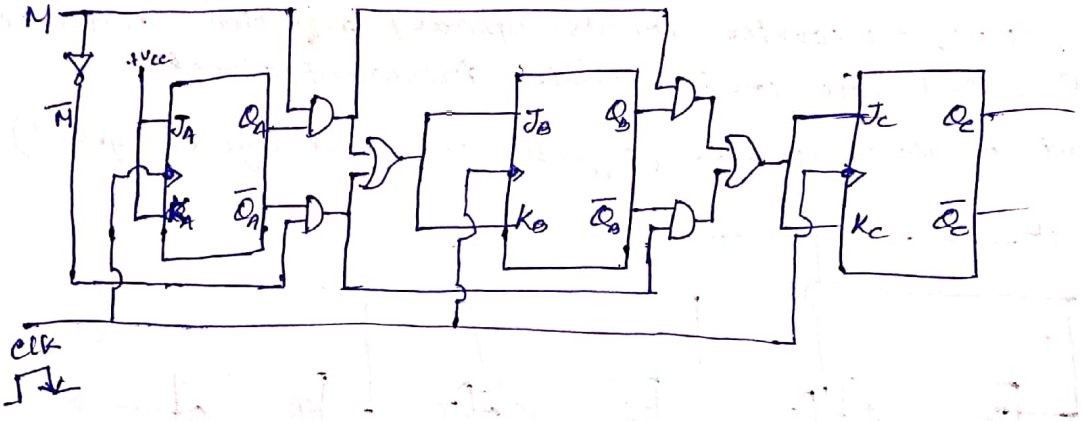
Eg: 3 bit up/down synchronous counter (-ve edge triggered)

Present State			M	Next State			J_C	K_C	J_B	K_B	J_A	K_A
Q_C	Q_B	Q_A		Q_C	Q_B	Q_A						
0	0	0	0	1	1	1	1	X	1	X	1	X
0	0	0	1	0	0	1	0	X	0	X	1	X
0	0	1	0	0	0	0	0	X	0	X	X	1
0	0	1	1	0	1	0	0	X	1	X	X	1
0	1	0	0	0	0	1	0	X	X	1	1	X
0	1	0	1	0	1	1	0	X	X	0	1	X
0	1	1	0	0	1	0	0	X	X	0	X	1
0	1	1	1	1	0	0	1	X	X	1	X	1
1	0	0	0	0	1	1	X	1	1	X	1	X
1	0	0	1	1	0	1	X	0	0	X	1	X
1	0	1	0	1	0	0	X	0	0	X	X	1
1	0	1	1	1	1	0	X	0	1	X	X	1
1	1	0	0	1	0	1	X	0	X	1	1	X
1	1	0	1	1	1	1	X	0	X	0	1	X
1	1	1	0	1	1	0	X	0	X	0	X	1
1	1	1	1	0	0	0	X	1	X	1	X	1

from K-maps we obtain : $J_C = \bar{Q}_B \bar{Q}_A \bar{M} + Q_B Q_A M = K_C$;

$J_B = K_B = \bar{Q}_A \bar{M} + Q_A M$; $J_A = K_A = 1$.

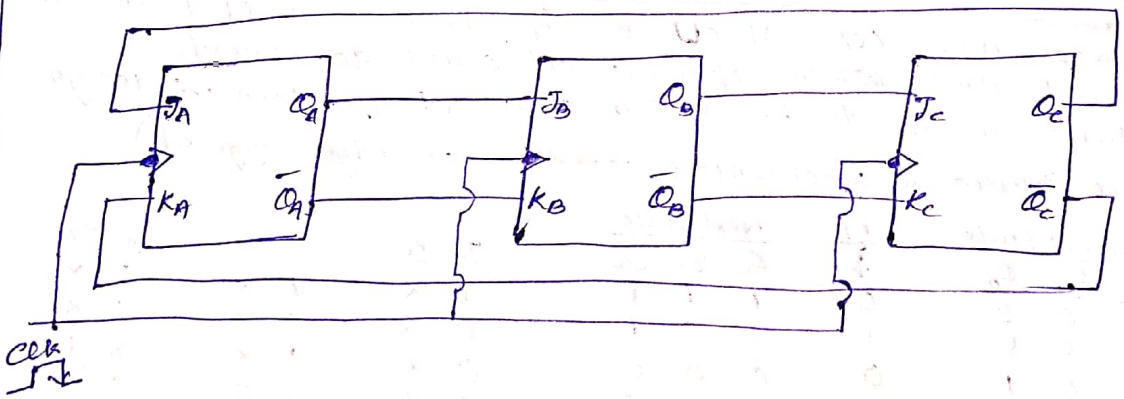
so design is



Ring Counters

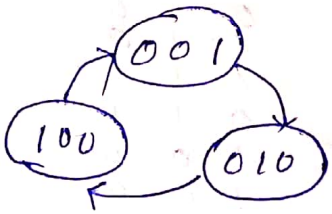
* This is the simplest shift register counter of the form of serial in - serial out type.
* There is a simple structure arrangement of FFs as in shift registers, but just that the last output is connected to the 1st FF's input.

Eg : 3-bit ring counter



state tables

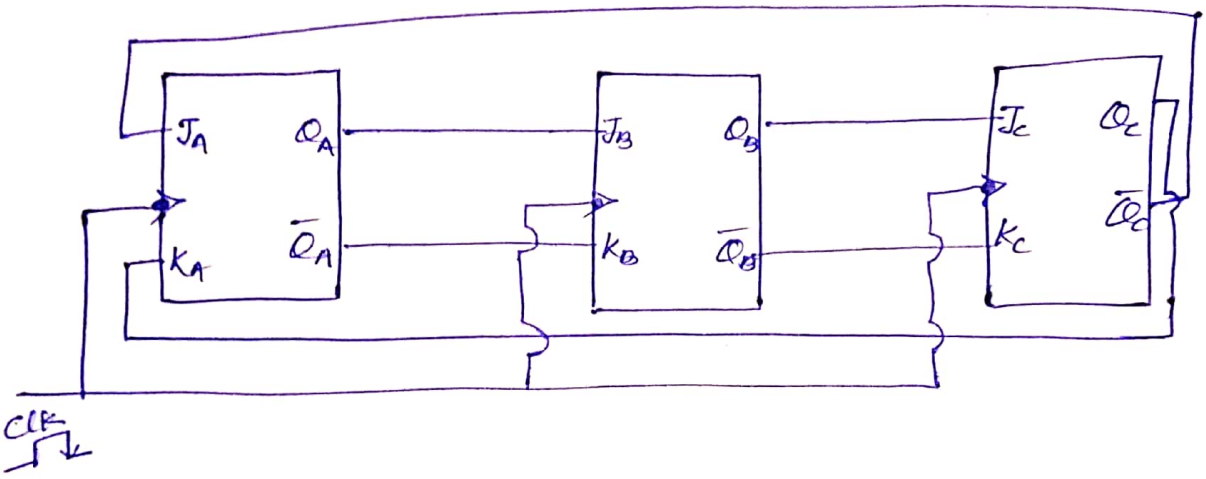
Q_C	Q_B	Q_A	clk
0	0	1	0th
0	1	0	1st
1	0	0	2nd
0	0	1	3rd



* An n-bit ring counter has only n-states to count

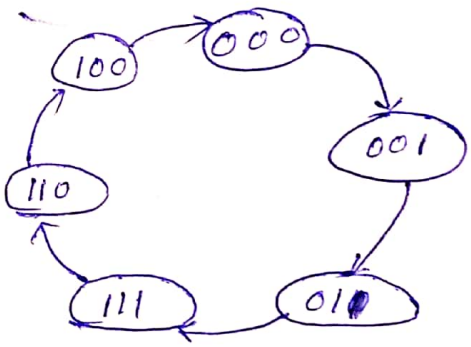
TWISTED RING COUNTERS:

* It is otherwise called as Johnson counter. or switch tail ring counter
* Its feedback is exactly same as that of the ring counters, except that the last ouput's invession goes to the input of 1st FF.
* Eg: 3bit twisted ring counter.



State table

Q_C	Q_B	Q_A	CLK
0	0	0	0th
0	0	1	1st
0	1	1	2nd
1	1	1	3rd
1	1	0	4th
1	0	0	5th
0	0	0	6th



* An n-bit twisted ring counter can count 2n distinct states.