Number System

Systematic way to represent and manipulate numbers

Examples of Number Systems

→ Decimal
→ Binary
→ Roman
→ Sexagesimal (base 60)

Broad Classification of number system

→ Weighted - decimal, binary, etc
→ Non-weighted - Roman, Gray code, etc

Eg 1) $235 = 2\times10^2 + 3\times10^1 + 5\times10^0$ → weighted.
(2, 1, 0)

2) I → 1, V → 5, X → 10 → XIV = 14 → not weighted.

Base or Radix

Decimal → 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 → 10 base

Binary → 0, 1 → 2 base.

~~General~~ → In general, a number system of base (radix) r uses r distinct digits (0, 1, 2, ... (r-1)) and each digit has some weight r^k, ~~k → 0~~

→ $k \geq 0$ for integer part
→ $k < 0$ for fractions

$$D = d_{n-1} d_{n-2} \ldots d_1 d_2 \cdot d_{-1} d_{-2} \ldots d_{-m}$$

(2) $\sqrt{22} = 6$ given, what is base of the number system?

(Ans. 7)

$2 \times 2 = (6)$
$2 + 2$
$2x = 30$
$x = 15$

Signed and Unsigned Binary Number Representation

→ Binary Number System is important for designing digital circuits.

→ Why are binary number system important?

- At the low level, the circuit is implemented using Transistors.

→ A transistor has two states, either 'ON' or 'OFF'

- As the transistor has two states and can be denoted by two digits (0 or 1) in binary system

→ There are some convention.

- Open Switch = 0
- Closed Switch = 1
- Low Voltage = 0
  High Voltage = 1
- Absence of current = 0
  Presence of current = 1

Bit → Single binary digit (0 or 1)
Nibble → 4 bits
Byte → 8 bits.

Unsigned Binary Number

→ An n-bit binary number system can have 2^n distinct numbers.

Minimum = 0 Maximum = $2^n - 1$

Eg n=3 → 0, 1, 2 ---- (8-1) 7

or

000, 001, ----- 111

→ An n bit binary integer can be denoted as:

$b_{n-1}\ b_{n-2} \cdots b_2\ b_1\ b_0$

$D = b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + \cdots + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 1$

→ Each digit position has a weight that is same power of 2

Signed Integer Representation.

There are three possible approaches to represent signed integers.

→ Sign Magnitude representation
→ 1's complement representation
→ 2's complement representation.

Sign Magnitude Representation

→ For an n-bit number representation, the most significant bit (MSB) represent sign (0 ⇒ +ve, 1 ⇒ -ve)

→ The rest (n-1) bits denote the magnitude of the number

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

+7	0111
+6	0110
+5	0101
	0110
+4	0101
+3	0100
+2	0011
+1	0010
	0001
+0	0000
-0	1000
-1	1001
-2	1010
-3	1011
-4	1100
-5	1101
-6	1110
-7	1111

→ A problem with 'sign-magnitude' representation: There are two possible representation of zero

+0 = -0 → 0000, 1000

1's complement Representation

→ Positive numbers are represented exactly as in sign-magnitude representative form

→ Negative numbers are represented in 1's complement form

→ 1's complement of a number is obtained by complementing every bit of a number (1 → 0 and 0 → 1)

→ MSB will indicate the sign of the number

→ In 1's complement representation, the number ranges from

$-(2^{n-1}-1)$ to $+(2^{n-1}-1)$ n-bits representation

+7 → 0111	−7 → 1000
+6 → 0110	−6 → 1001
+5 → 0101	−5 → 1010
+4 → 0100	−4 → 1011
+3 → 0011	−3 → 1100
+2 → 0010	−2 → 1101
+1 → 0001	−1 → 1110

0 → 0000 −0 → 1111

→ Similar to Sign-magnitude representation, 1's complement representation has two zeros

→ Advantage of 1's complement representation is that subtraction can be done using addition which leads to saving in circuitry

~~5 − 3 = 0101 + 1100 = 0001~~

2's complement Representation

→ Positive Numbers are represented same as before

→ Negative numbers are represented in 2's complement

→ To get Number's magnitude → 1's complement → add 0001 → 2's comple.

→ MSB will indicate the sign of the number (0 → +ve, 1 → −ve)

7 → 0111	−7 → 1001
6 → 0110	−6 → 1010
5 → 0101	−5 → 1011
4 → 0100	−4 → 1100
3 → 0011	−3 → 1101
2 → 0010	−2 → 1110
1 → 0001	−1 → 1111

0 → 0000

→ Range : $(2^{n-1}-1)$ to -2^{n-1}

→ Advantages of 2's complement

1) Unique representation of zero
2) Subtraction can be done using addition, which saves circuitry

Eg 5 − 3 = 0101 − 00101 = 0101 + 1101 → 2's complement of 3

= 0010

→ Additional features of 2's complement representation:

- Weighted number representation with MSB having weight $= -2^{n-1}$

$$D = b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + \cdots + b_1 \times 2^1 + b_0 \times 2^0$$

Eg 0101 = 4 + 1 = 5

1101 = −8 + 4 + 0 + 1 = −3

- Shift left by k positions with zero padding multiplies the number by 2^k

00001001 = 9 —shift 2 pos left→ 00100100 = 36

11110111 = −9 —shift 2 pos left→ 11011100 = −36

- Shift right by k position with sign bit padding divides the number by 2^k

00100100 = 36 —shift 2 pos right→ 00001001 = 9

- The sign bit can be copied as many times as required at the beginning to extend the size of the number (called sign extension)

$X = 0111 = 0000\ 0111$

Binary Arithmetic

- The binary Number System is widely used in computer system.

Input		X + Y		X − Y		X . Y
X	Y	sum	carry	difference	borrow	Product
0	0	0	0	0	0	0
0	1	1	0	1	1	0
1	0	1	0	1	0	0
1	1	0	1	0	0	1

Binary Addition.

Eg:

```
  0 1 0 1
+ 0 1 1 1
---------
1 0 0 1 0
```

```
  1 0 1 1 0 1
+ 1 0 0 1 1 1
-------------
1 0 1 0 1 0 0
```

Binary Subtraction

Eg:

```
  0 1 0 1
- 0 1 0 1
---------
0 0 0 0
```

```
  1 0 1 1 0
- 0 1 1 0 0
-----------
0 0 1 1 0
```

Substraction using 1's complement

Suppose we want to perform $A - B$ using 1's complement. We have to follow the steps given below

- Compute 1's complement of $B = \bar{B}$ (say)
- Perform $R = A + \bar{B}$
  - If a carry is obtained after addition. Add carry with R (i.e $R = R + 1$). The result is positive number
  - Else The result is a negative and is in 1's complement form of R.

Eg: $6 - 2$; $6_{10} \rightarrow (0110)_2$, $+2 = (0010)_2$, $-2 \xrightarrow{(1's)} (1101)_2$

$\therefore 6 - 2 \rightarrow$

$$\begin{array}{r} 0110 \\ +\ 1101 \\ \hline 10011 \end{array}$$

→ There is a carry and R is +ve.

$R = 0011 + 0001 = 0100 = 4_{10}$

Eg: $3 - 5$; $(3)_{10} = (0011)_2$, $(5)_{10} = (0101)_2$

$(-5)_{1's} = (1010)_2$

$\therefore 3 - 5 \rightarrow$

$$\begin{array}{r} 0011 \\ +\ 1010 \\ \hline 1101 \end{array}$$

→ no carry, R is in 1's complement and is -ve

$\therefore R = -(0010)_2 = -(2)_{10} = -2$

Substraction using 2's complement

Suppose we want to perform A - B using 2's complement

Steps

- Compute 2's complent of B (let it be $\bar{B}$)
- Compute $R = A + \bar{B}$
  - If carry is generated, ignore it, R is +ve
  - else R is -ve and is in 2's complement form

Eg 6 - 2 ; $(6)_{10} = (0110)_2$, $(2)_{10} = (0010)_2$

$(-2)_{10_{2's}} = (1101)_2 + (0001)$

$= (1110)_2$

$\therefore 6 - 2 \rightarrow$

$$\begin{array}{r} 0\,1\,1\,0 \\ +\ 1\,1\,1\,0 \\ \hline 1\ 0\,1\,0\,0 \end{array}$$

$\rightarrow$ Carry is present

$\therefore R$ is +ve, $R = (0100)_2 = (4)_{10}$

Eg 3 - 5 ; $(3)_{10} = (0011)_2$, $(5)_{10} = (0101)_2$

$(-5)_{10_{2's}} = (1010)_2 + (0001)_2$

$= (1011)_2$

$\therefore 3 - 5 \rightarrow$

$$\begin{array}{r} 0\,0\,1\,1 \\ +\ 1\,0\,1\,1 \\ \hline 1\,1\,1\,0 \end{array}$$

$\rightarrow$ no carry

$\therefore R$ is -ve, $R = (-1)_{10} \times (1000 + 0001)_2$

$= (-1)_{10} \times (0010)_2$

$= (-1)_{10} * (2)_{10} = (-2)_{10}$

Gray Code

0	0000	1 bit change
1	0001	2 bit change
2	0010	1 bit change
3	0011	3 bit change
4	0100	

→ Change of bit can cause problem

→ That's why Gray Code is invented, where only 1 bit change occur.

→ This makes Gray Code a non-weighted representation

Application

i) Gray Code reduces error during Analog to Digital

ii) Easy to convert Gray to Binary

→ Gray is called self-reflecting code

→ suppose you have gray code of 2 bits.

00
01
10

to make gray codes of 3 bit we do the following.

$$\left.\begin{matrix}000\\001\\011\\010\end{matrix}\right\} gp1$$

$$\left.\begin{matrix}110\\111\\101\\100\end{matrix}\right\} gp2 \rightarrow \text{mirror of } gp1$$

→ i) mirror gp1 to form gp2.
ii) add 0 to gp1 and 1 to gp 2
iii) 3 bit gray code is formed.

Generalising, we can do same for m bit group to form a (m+1) bit gray code

Conversion of Binary to Gray ~~and vice versa~~

let gray code be $g_{n-1}\ g_{n-2}\ g_{n-3} \cdots g_0$

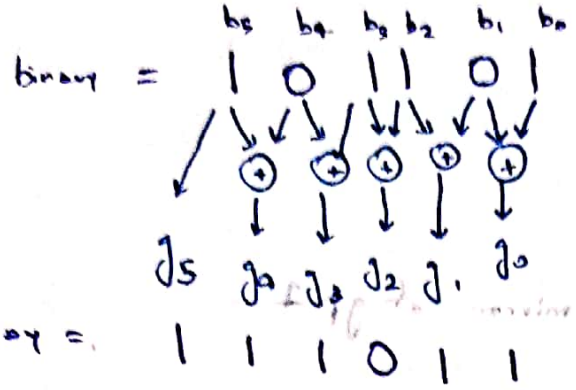
and it's corresponding binary $b_{n-1}\ b_{n-2}\ b_{n-3} \cdots b_0$

$$g_i = b_i \oplus b_{i+1} \quad 0 \le i \le n-2$$ ← XOR gate

$$g_{n-1} = b_{n-1}$$

m	Y	m⊕Y
0	0	0
0	1	1
1	0	1
1	1	0

Eg ~~b5~~



Gray to Binary Conversion

Eg gray = 0 1 0 1 1 0 0 1

binary = 0 1 1 0 1 1 1 0

i) start from MSB and go to LSB

$$b_i = \begin{cases} g_i & \text{(if no. of 1's preceding } g_i \text{ is even)} \\ \overline{g_i} & \text{(if no. of 1's preceding } g_i \text{ is odd)} \end{cases}$$

Error Detection Code

→ binary code can change due to noise when travelling through media

→ we can check it by giving extra bits. (Parity Bit)

even parity tries to make no. of 1's even

odd parity tries to make no. of 1's odd

Eg: 0 0 0 → 1 (0) (0) ← even parity

0 0 1 → 0 (1) (1) ← odd parity

if no. of 1's even → 0

if no. of 1's odd → 1

Eg: 0 1 0 → 0

→ this can check for odd no. of changes in binary code.

without checking

Eg :- signal : 0 0 0 1 —travelling through media→ 0 1 0 1

with checking signal → 0 0 0 1 1 —travelling→ 0 0 1 1 (1)

even parity ↑ ↑ parity bit

Message

Even Parity	Even Parity	Odd Parity
Eg 0 0 0	0	1
0 0 1	1	0
0 1 0	1	1
0 1 1	0	0
1 0 0	1	1
1 0 1	0	0

→ checked by counting no. of 1's in message.

Boolean Algebra

Q) Prove $x + \bar{x}y = x + y$

LHS $= x.(y+\bar{y}) + \bar{x}y$ $[1 = y + \bar{y}]$

$= xy + x\bar{y} + \bar{x}y$ $[x + x = x]$

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

$= x + y$ = RHS

Q) Prove $x.(\bar{x}+y) = xy$

LHS $= x.\bar{x} + xy = xy$ = RHS

minimisation

Eg $A.B + ABC + \bar{B}C$

$\Rightarrow AB(1+C) + \bar{B}C = AB + \bar{B}C$

Principal of Duality

Eg: $E_1 \rightarrow X + \bar{X}Y = X + Y$

$E_{1\,dual} \rightarrow$ dual $\rightarrow X.(\bar{X}+Y) = XY$

if E_1 is true, then $E_{1\,dual}$ is also true.

Eg $X + 1 = 1$

dual $X.0 = 0$

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

Eg minimise $F(A,B,C) = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

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

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

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

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

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

$= BC + AC + AB$

minimize

Eg $F(A,B,C) = \bar{A}B + \bar{B}\bar{C} + AB + B\bar{C}$

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

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

$= B + \bar{B}$

$= 1$

Demorgan's Theorem

→ $\overline{x+y} = \bar{x} \cdot \bar{y}$

→ $\overline{x \cdot y} = \bar{x} + \bar{y}$

x	y	$\overline{x+y}$	$x' \cdot y'$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

→ $\overline{x+y+z} = \bar{x} \cdot \bar{y} \cdot \bar{z}$

→ $(x \cdot y \cdot z)' = x' + y' + z'$

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

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

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

Eg $F = \overline{(A\bar{B} + \bar{A}B)}$

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

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

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

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

Q) $F(A,B,C) = (ABC)'(A+C)(A+C')$

simplify F

$$F = (A'+B'+C')(A+AC'+AC)$$

$$= AB' + AB'C' + ABC + AC'$$

$$Ans = AB' + AC'$$

Minterm and Maxterm

$f(x, y, z)$

→ 8 minterms

($\bar{x}=0$, $x=1$)

$\bar{x}\bar{y}\bar{z}$	m_0	0	0
$\bar{x}\bar{y}z$	m_1	0	1
$\bar{x}y\bar{z}$	m_2	1	0
$\bar{x}yz$	m_3	1	1
$x\bar{y}\bar{z}$	m_4		
$x\bar{y}z$	m_5		
$xy\bar{z}$	m_6		
xyz	m_7		

→ 8 maxterm

($x=0$, $\bar{x}=1$)

$x+y+z$	M_0
$x+y+\bar{z}$	M_1
$x+\bar{y}+z$	M_2
$x+\bar{y}+\bar{z}$	M_3
$\bar{x}+y+z$	M_4
$\bar{x}+y+\bar{z}$	M_5
$\bar{x}+\bar{y}+z$	M_6
$\bar{x}+\bar{y}+\bar{z}$	M_7

Eg $f = ab'c + abc'$

$ab'c = 1$, only if $a=1, b=0, c=1$

Eg

	x	y	z	s = f(x,y,z)	
m_0	0	0	0	0	←
m_1	0	0	1	1	← $\bar{x}\bar{y}z$
m_2	0	1	0	1	← $\bar{x}y\bar{z}$
m_3	0	1	1	0	←
m_4	1	0	0	1	← $x\bar{y}\bar{z}$
m_5	1	0	1	0	←
m_6	1	1	0	0	←
m_7	1	1	1	1	← xyz

→ here m_1, m_2, m_4, m_7 are true minterms
and m_0, m_3, m_5, m_6 are false minterm.

$\therefore f(x,y,z) = \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$

$f(x,y,z) = \Sigma(1,2,4,7)$

$f(x,y,z) = (x+y+z)(x+\bar{y}+\bar{z})(\bar{x}+y+\bar{z})(\bar{x}+\bar{y}+z)$

Book: Digital Logic and Computer Design, Mavrice Mno.

BCD ~~and Excess Code~~ (Binary Coded Decimal)

→ It is sometime desirable to manipulate numbers in decimal instead of converting them to binary.

→ Decimal to binary and binary to decimal conversion is complex.

→ One popular code to represent decimal is BCD. Each decimal digit is represented by 4-bit binary equivalent. conversion is much easier.

Eg 391 = 0011 ~~010~~ 1001 0001

13.34 = 0001 0011 . 0011 0100

→ There are six unused combinations.

1010 1011 1100 1101 1110 1111

(10) (11) (12) (13) (14) (15)

Addition of BCD Numbers

→ When two BCD numbers are added, there may be need for correction step where 6 (0110) will be added to one of the nibble

→ This correction is required when a nibble is one of the six unused combination or there is carry in from the previous nibble.

Eg 1) 23 + 46 = 69

0010 0011 + 0100 0110 = 0110 1001

(6) (9)

2) 23 + 48 = 71

0010 0011 + 0100 1000 = 0110 1011 → This is unused so we add 6 (0110)

(6) (11)

0000 + 0110 = 0111 0001

(7) (1)

3) 28 + 39 = 67

0010	1000	
+ 0011	1001	
0110	0001	→ here carry flowed from one nibble to other
+ 0000	0110	→ add 6
0110	0111	
(6)	(7)	

Gray Code

→ There are some applications, where if multiple bits are changing between two consecutive number, that may create problem. To elevate the problem, a new code was introduced, which is called Gray Code.

→ Gray Code is a type of non-weighted binary code where successive code words differ in only one bit. Any code with this property is called cyclic code.

→ Gray Code is used in many practical applications that require analog to digital conversion.

→ To reduce error in conversion

→ also, Binary ↔ Gray Conversions are easy.

Eg: to measure the angle of rotation of a wheel.

→ Gray Code also called self-reflecting code. Suppose we have Gray Code representation for m-bits.

→ To obtain the Gray Code representation for (m+1) bits, we write down two m-bit representation one below the other with second one being the mirror image of the first one.

→ we then add 0 at the beginning of every code in first group and 1 at the beginning of the second group.

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

Binary to Gray Code Conversion

→ Let $g_{n-1} g_{n-2} \cdots g_2 g_1 g_0$ and $b_{n-1} b_{n-2} \cdots b_2 b_1 b_0$ denote n-bit Gray Code and it's equivalent binary representation respectively.

$$g_i = b_i \oplus b_{i+1} \quad \text{for } 0 \le i \le n-2$$

and $g_{n-1} = b_{n-1}$

x	y	x ⊕ y
0	0	0
0	1	1
1	0	1
1	1	0

Decimal	binary Representation	Gray Code
0	0000	000
1	0001	001
2	0010	011
3	0011	010
4	0100	110
5	0101	111
6	0110	101
7	0111	100

Gray Code to Binary Code conversion.

→ Start with MSB and proceed to the LSB and set

→ $b_i = g_i$, if number of 1's preceeding g_i is even

$b_i = \overline{g_i}$, if number of 1's preceeding g_i is odd

$g_2 g_1 g_0$	$b_2 b_1 b_0$
000	000
001	001
011	010
010	011
110	100
111	101
101	110
100	111

Boolean Algebra

→ An algebraic system defined on the set $\{0,1\}$ with two binary operators (AND or OR) and one unary operator NOT. AND operator is called Logical Product. OR operator and NOT operation are called Logical sum and complement respectively.

→ Boolean (switching) variables are two valued variables that can take two distinct values 0 and 1.

→ Boolean expression is an expression consisting of Boolean variables, constants and operators.

→ Given a Boolean expression, how to prove it?

Eg: Prove $x + xy = x$

i) By verifying the expression for all possible values of the variable. Called truth table verification or perfect induction.

ii) By using algebraic manipulation using some rules

Basic Laws of Boolean Algebra

Basic Identity

$x + 1 = 1$

$x \cdot 0 = x$

$x \cdot 1 = x$

$x \cdot 0 = 0$

Idempotent Law

$x + x = x$

$x \cdot x = x$

Commutative Law (Order of variable doesn't matter)

$(x+y) = y + x$

$x \cdot y = y \cdot x$

Complementation Law

$x + \bar{x} = 1$

$x \bar{x} = 0$

Associative Law

$(x+y)+z = x+(y+z)$

$(x \cdot y) \cdot z = x \cdot (y \cdot z)$

Distributive Law

$x \cdot (y+z) = x \cdot y + x \cdot z$

$x + (y \cdot z) = (x+y) \cdot (x+z)$

Absorption Law

1) $xy + x\bar{y} = x$

Proof LHS $= x(y+\bar{y}) + xy = xy + x\bar{y} = x(y+\bar{y}) = x = $ RHS

2) $x \cdot (x+y) = x$

Proof LHS $= x + xy = x$

Useful Laws

1) $x + \bar{x}y = x+y$

Proof LHS $= x(y+\bar{y}) + \bar{x}y$

$= xy + x\bar{y} + \bar{x}y$

$= xy + x\bar{y} + xy + \bar{x}y$

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

$= x + y =$ RHS

2) $x \cdot (\bar{x}+y) = xy$

Proof: LHS $= x\bar{x} + xy = xy =$ RHS

Consensus Theorem

i) $xy + \bar{x}z + yz = xy + \bar{x}z$

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

Involution

$\overline{(\bar{x})} = x$

Boolean Function Minimisation

Given a boolean expression, we can simplify it by using basic laws of boolean algebra. We can reduce the number of terms in the expression. We can also reduce the number of literals (variables)

Eg: $xy + xyz + \bar{y}z = xy(1+z) + \bar{y}z = xy + \bar{y}z$

Principle of Duality

→ Principle of Duality states that a Boolean expression E_2 can be obtained from a given Boolean expression E_1 by interchanging the operation AND and OR and constants 0 and 1. E_1 and E_2 are said to be dual of each other

Eg: i) given : $x + \bar{x}y = x + y$
dual : $x(\bar{x} + y) = xy$

ii) given : $x + 1 = 1$
dual : $x \cdot 0 = 0$

Simplification Examples

1) $F = x\bar{y} + xy + yz$
$= x(\bar{y} + y) + yz$
$= x + yz$

2) $F = \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz$
$= \bar{x}yz + x\bar{y}z + xyz + xy\bar{z} + xyz$
$= yz + xz + xy$

(0 0 0)	$\bar{x}\bar{y}\bar{z}$
(1 0 0)	$x\bar{y}\bar{z}$
(0 1 0)	$\bar{x}y\bar{z}$
(1 1 0)	$xy\bar{z}$
(0 0 1)	$\bar{x}\bar{y}z$
(1 0 1)	$x\bar{y}z$
(0 1 1)	$\bar{x}yz$
(1 1 1)	xyz

De-Morgan's Theorem

For two variables x and y :

i) $\overline{x+y} = \bar{x} \cdot \bar{y}$

ii) $\overline{x \cdot y} = \bar{x} + \bar{y}$

Eg: 1) $F = \overline{x+y} \cdot (\bar{x} + \bar{y})$
$= \bar{x}\bar{y} \cdot (\bar{x} + \bar{y})$
$= \bar{x}\bar{y} + \bar{x}\bar{y} = \bar{x}\bar{y}$

2) $F = \overline{(x\bar{y} + \bar{x}y)}$
$= \overline{(x\bar{y})}\ \overline{(\bar{x}y)}$
$= (\bar{x} + \bar{y})(x + \bar{y})$
$= \bar{x}x + \bar{x}\bar{y} + xy + y\bar{y}$
$= \bar{x}\bar{y} + xy$

Minterm and Maxterm

→ In a boolean function, a set literal is defined as variables in uncomplemented or complemented form.

Eg: $x, \bar{x}, y, \bar{y}$, etc.

→ Consider an n-variable boolean function $f(x_1, x_2, \ldots x_n)$

Minterm: A product term (AND operation) of all the n literals is called minterm

Maxterm: A sum term (OR operation) of all the n literals is called maxterms

→ consider a function $f(x,y,z)$

minterms:		maxterm:	
$\bar{x}\bar{y}\bar{z}$	(000)	$x+y+z$	(000)
$\bar{x}\bar{y}z$	(001)	$x+y+\bar{z}$	(001)
$\bar{x}y\bar{z}$	(010)	$x+\bar{y}+z$	(010)
$\bar{x}yz$	(011)	$x+\bar{y}+\bar{z}$	(011)
$x\bar{y}\bar{z}$	(100)	$\bar{x}+y+z$	(100)
$x\bar{y}z$	(101)	$\bar{x}+y+\bar{z}$	(101)
$xy\bar{z}$	(110)	$\bar{x}+\bar{y}+z$	(110)
xyz	(111)	$\bar{x}+\bar{y}+\bar{z}$	(111)

Properties of minterm and maxterm

→ A particular minterm assumes value 1 for exactly one combination of variables

Eg: $f(x,y,z) = x\bar{y}z + xy\bar{z} + xyz$

$x\bar{y}z$	$xy\bar{z}$	xyz
x=1	x=1	x=1
y=0	y=1	y=1
z=1	z=0	z=1

→ A particular maxterm assumes 0 for exactly one combination of variables

Eg: $f(x,y,z) = (x+\bar{y}+z)(\bar{x}+\bar{y}+\bar{z})(x+y+z)$

$(x+\bar{y}+z)$	$(\bar{x}+\bar{y}+\bar{z})$	$(x+y+z)$
x=0	x=1	x=0
y=1	y=1	y=0
z=0	z=1	z=0

→ All the minterms that have value 1 are called the true minterms

→ All the minterms that have value 0 are called the false minterms

Eg: $f(x,y,z) = \bar{x}y + xyz$

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

$= \bar{x}yz + \bar{x}y\bar{z} + xyz.$

True minterms → $\bar{x}yz$, $\bar{x}y\bar{z}$, xyz

→ All the maxterms that have value 1 are called the true maxterm

→ All the maxterm that have value 0 are called the false maxterm

Canonical Form of Representing Function

→ A canonical form is unique representation of a function

→ We can obtain two canonical representation directly from the truth table.

i) Canonical Sum of product (disjunction normal form)

ii) Canonical product of sum (conjunctive normal form)

Canonical Sum-of-Product

→ From the truth table, identify all the true minterms - corresponding to rows for which the output of the function is 1.

→ Take the sum of all true minterms.

Eg:

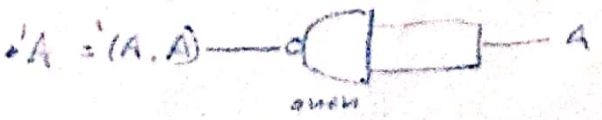
x	y	z	s	
0	0	0	0	← $x+y+z$
0	0	1	1	← $\bar{x}\bar{y}z$
0	1	0	1	← $\bar{x}y\bar{z}$
0	1	1	0	← $x+\bar{y}+\bar{z}$
1	0	0	1	← $x\bar{y}\bar{z}$
1	0	1	0	← $\bar{x}+y+\bar{z}$
1	1	0	0	← $\bar{x}+\bar{y}+z$
1	1	1	1	← xyz

$s = \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$
(001) (010) (100) (111)

$= \Sigma(1,2,4,7)$

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

$= \Pi(0,3,5,6)$



Canonical Product of Sum

→ From the truth table identify the false minterm - corresponding to the rows for which the output of the function is 0

→ For each false minterm, form a sum term where a variable will appear in uncomplemented form if it has value 0

SOP → 1 dominant → 1 = non complement
0 = complement
↓
minterm

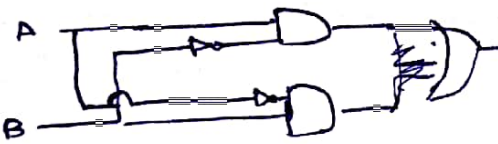
POS → 0 dominant → 0 = no complement
1 = complement
↓
max term

Function Completeness

→ Three basic operations: {AND, OR, NOT}

→ hence the set {AND, OR, NOT} is complete. As any boolean expression can be made using the operations in the set

Eg: $A\bar{B} + \bar{A}B$



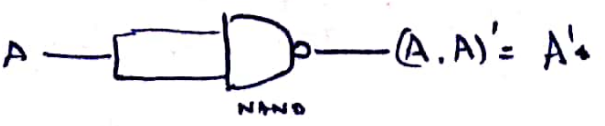
→ Now we can observe that {NAND}, and {NOR} gate are also functionally complete.

A[A (NAND) B = $\overline{A \cdot B}$ = $\bar{A} + \bar{B}$]

[A (NOR) B = $\overline{A + B}$ = $\bar{A} \cdot \bar{B}$]

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

→ Eg: A NOT(A) = A'

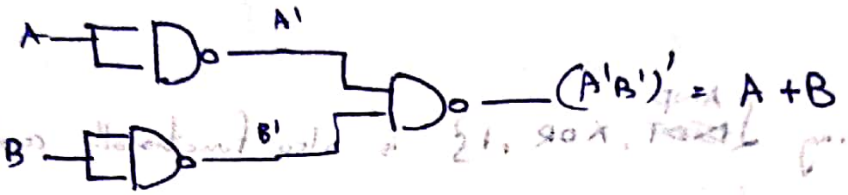


→ AND (A,B) = $[(A.B)']'$

$= [(AB)'(AB)']'$

So

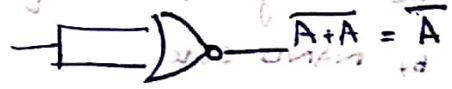
→ OR (A,B) ⇒



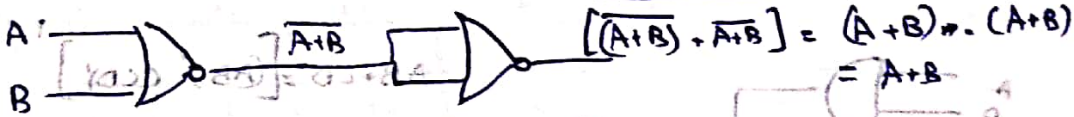
→ So using ~~NAND~~ NAND Gate, we can make {OR, NOT, AND}, and hence {NAND} is also functionally complete. And It is called a Universal Gate.

NOR

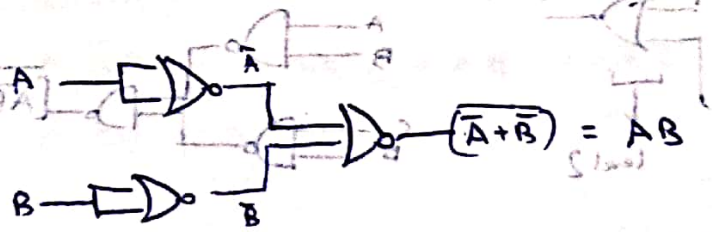
NOT → A



OR →



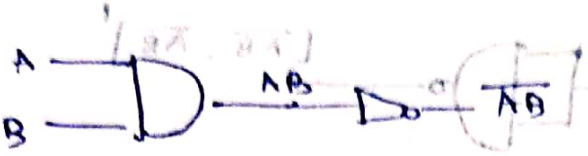
AND →



→ hence {NOR} is also functionally complete and NOR is also a universal gate

now knowing {NAND} and {NOR} as functionally complete.

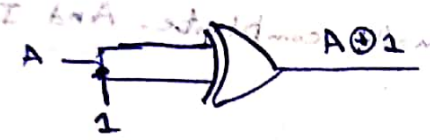
2) {NOT, AND} is also functionally complete.



b) {NOT, OR} is also functionally complete.

~~Sim~~ → Now proving {~~NOT~~ AND, XOR, 1} is also functionally complete

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

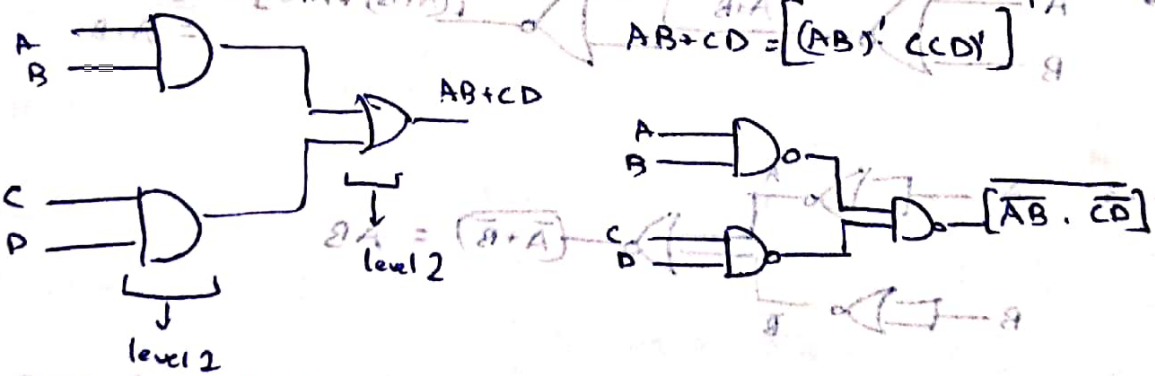


$A \oplus 1 = A \cdot 0 + \bar{A} \cdot 1 = \bar{A}$ → This along with AND can make NAND

2 level

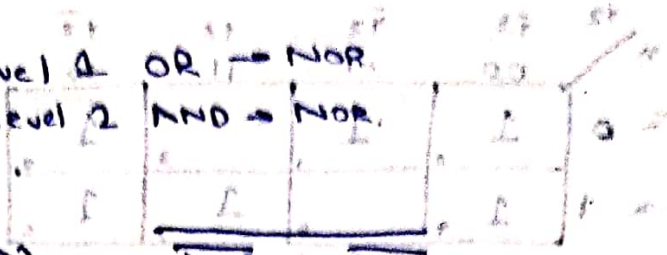
→ Any {AND, OR} circuit can be changed to it's equivalent circuit by changing AND and OR Gates by NAND Gate.

level 1 level 2

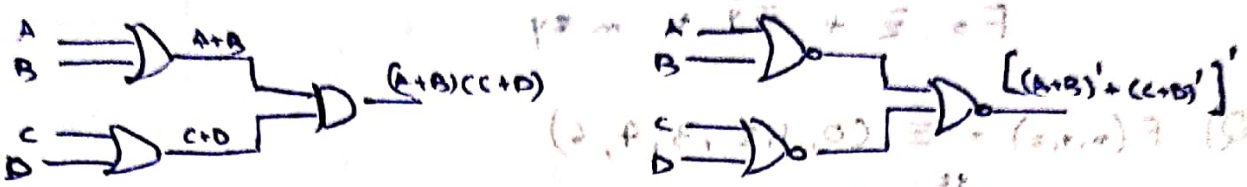


→ Any {OR, AND} is equivalent to {NOR, NAND}
↓
two level

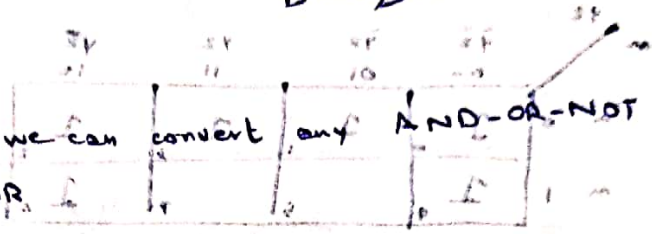
→ This means level 1 OR → NOR
level 2 AND → NOR.



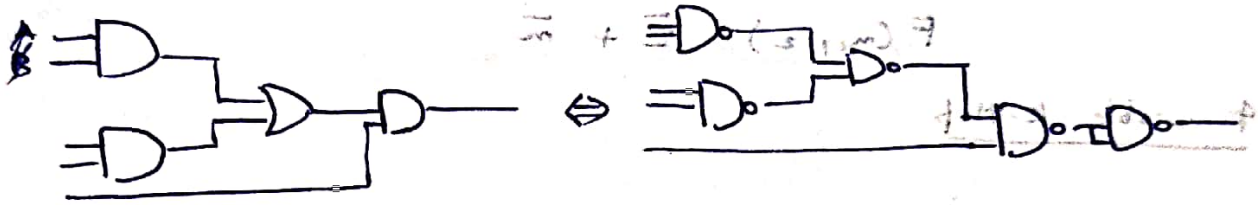
Eg: $(A+B)(C+D) = \overline{\overline{A+B} + \overline{C+D}}$



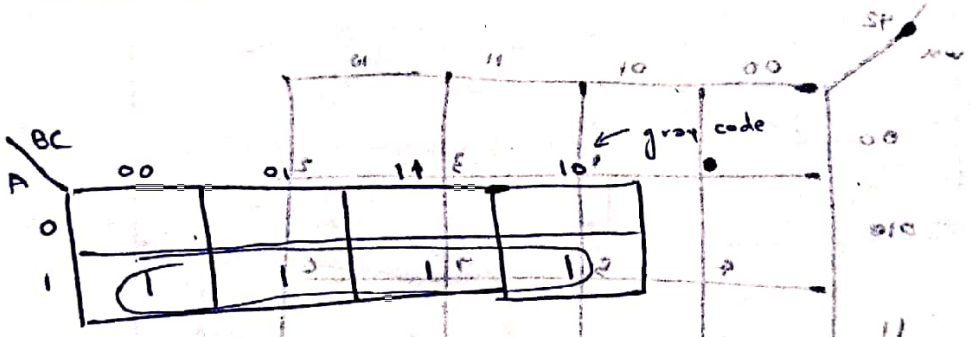
→ Using Boolean Algebra, we can convert any AND-OR-NOT to only NAND or only-NOR.



Eg:



K Map

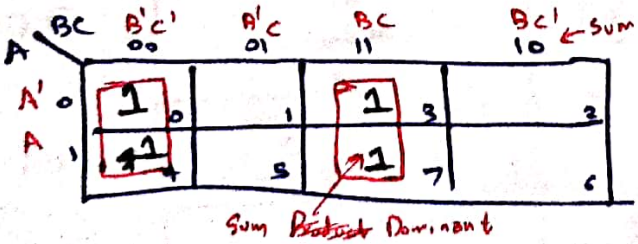


→ $AB'C + AB'C + ABC + AB'C'$

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

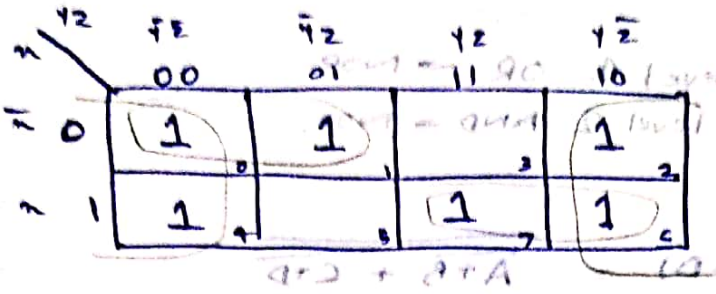
Eg:

$f(A,B,C) = \Sigma\ (0,3,4,7)$
↓
SOP



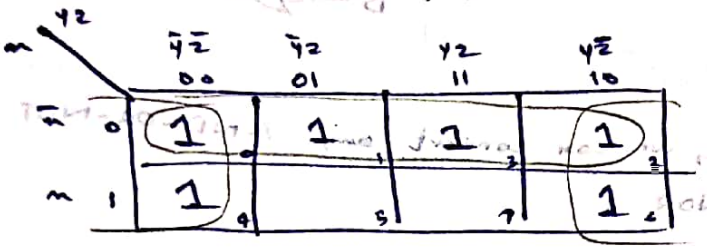
$f(A,B,C) = \bar{B}\bar{C} + BC$

Q) $F(x,y,z) = \Sigma(0,1,2,4,6,7)$



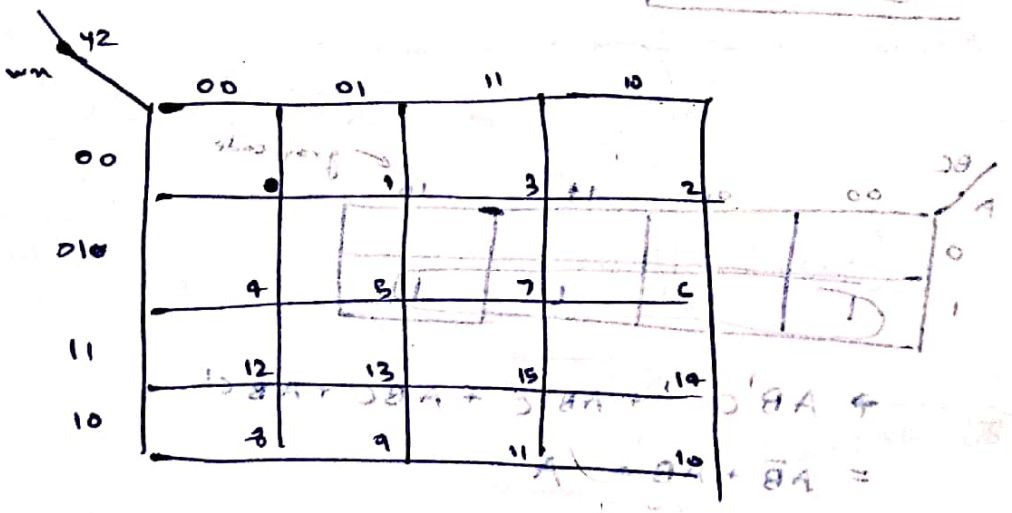
$F = \bar{z} + \bar{x}\bar{y} + x\bar{y}$

Q) $F(x,y,z) = \Sigma(0,1,2,3,4,6)$

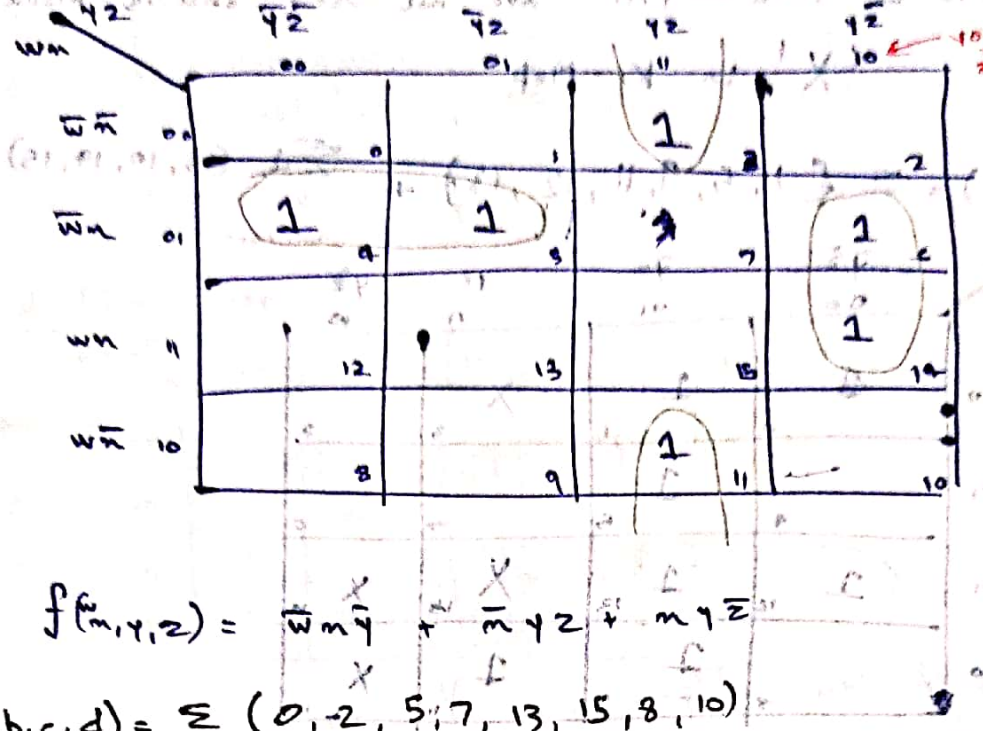


$F(x,y,z) = \bar{z} + \bar{x}$

4 variable K-Map

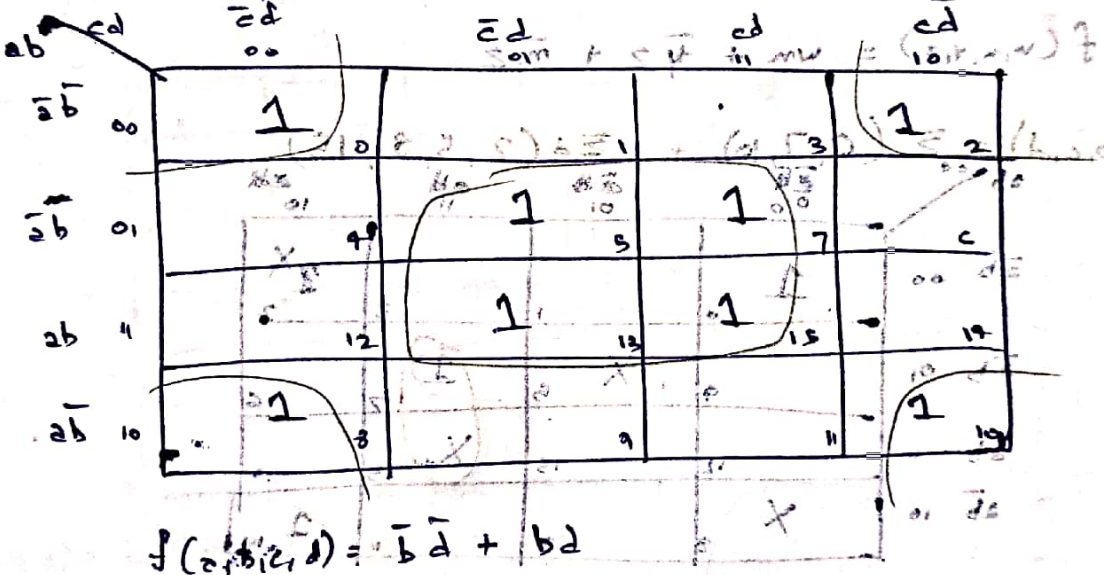


Eg $f(w,x,y,z) = \Sigma(8, 9, 5, 6, 11, 14)$



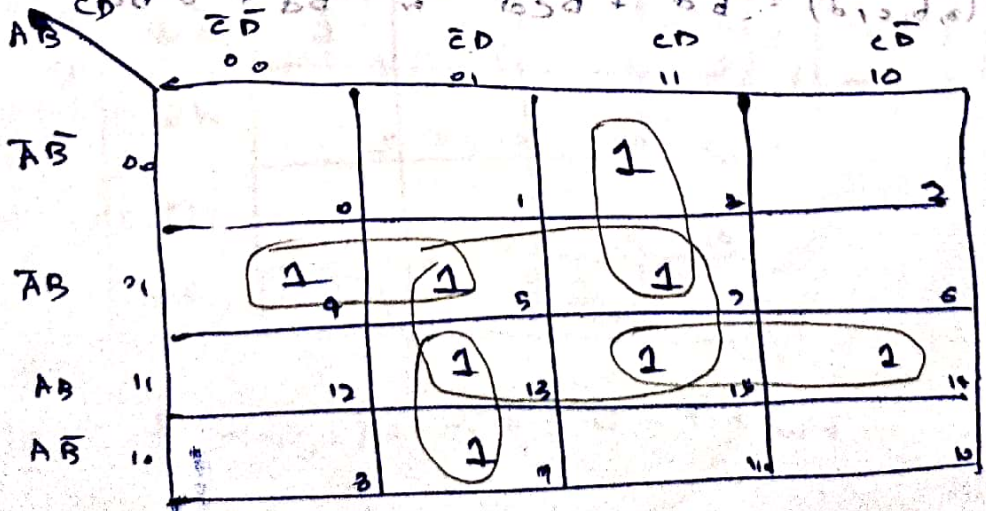
$f(w,x,y,z) = \bar{w}x\bar{y} + \bar{x}yz + xy\bar{z}$

Q) $f(a,b,c,d) = \Sigma(0, 2, 5, 7, 13, 15, 8, 10)$



$f(a,b,c,d) = \bar{b}\bar{d} + bd$

Q) $f(A,B,C,D) = \Sigma(3, 4, 5, 7, 9, 13, 14, 15)$



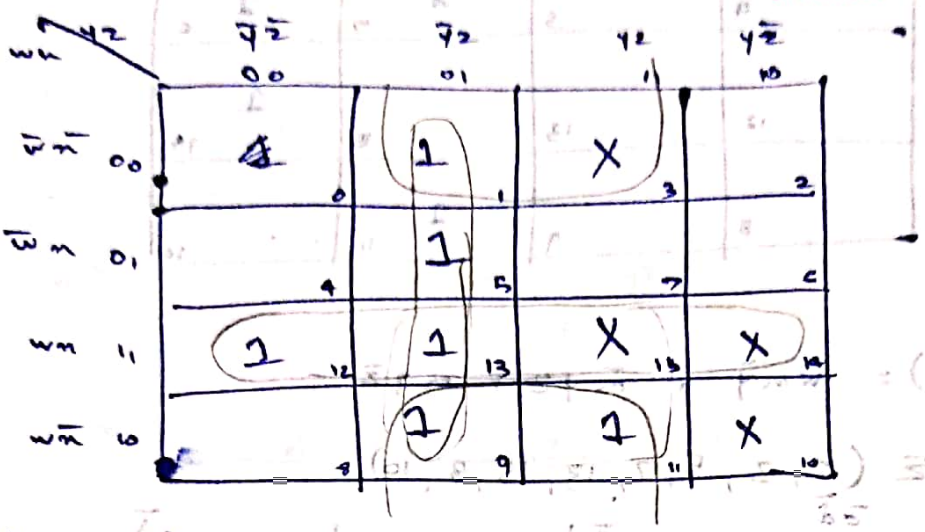
$f(A,B,C,D) = \bar{A}B\bar{C} + \bar{A}CD + ABC + A\bar{C}\bar{D} + BD$ → is extra

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

Handling Don't care Input

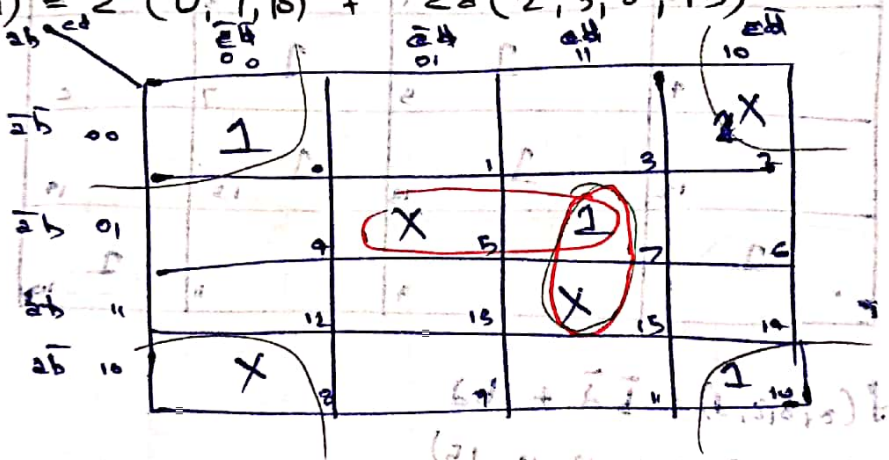
In BCD → 1010 — 1111 are not used and is marked as 'X' in K-Map

Eg $f(w,x,y,z) = \Sigma\{1,5,9,11,12,13\} + \Sigma d(3,10,14,15)$ ← don't care

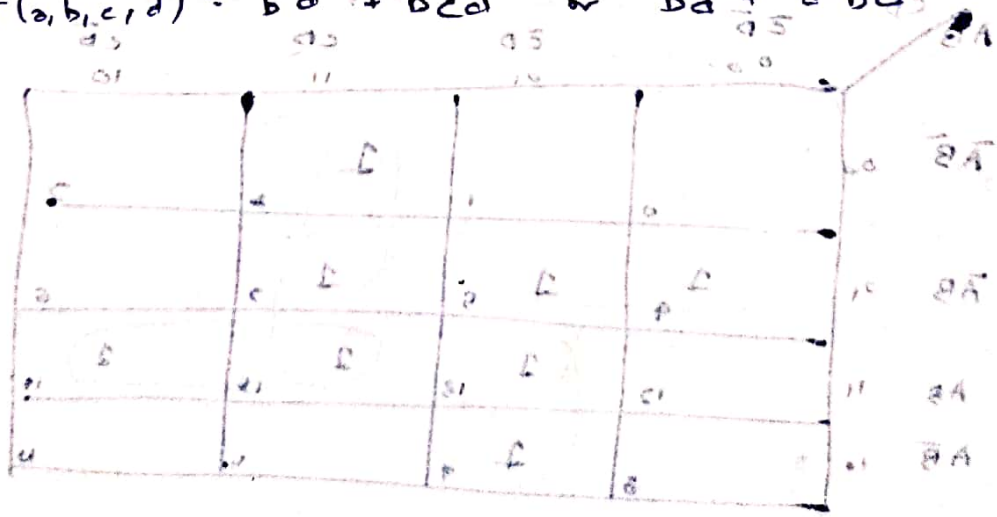


$f(w,x,y,z) = wx + \bar{y}z + \bar{x}z$

Q) $f(a,b,c,d) = \Sigma(0,7,10) + \Sigma d(2,5,8,15)$

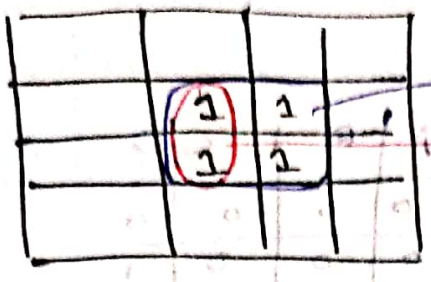


$f(a,b,c,d) = \bar{b}\bar{d} + bcd$ or $\bar{b}\bar{d} + a'bd$



Prime Implicant

Eg



→ This is a prime implicant

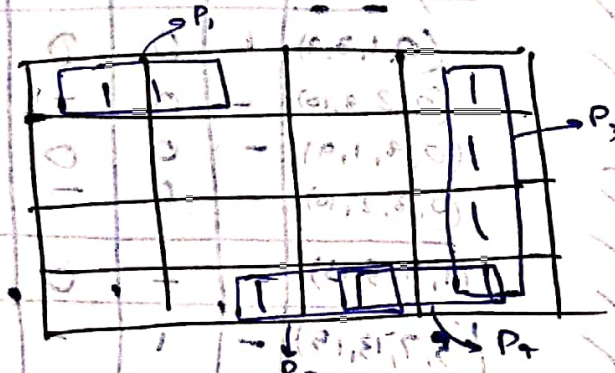
→ This is not a prime implicant

Essential Prime Implicant

Eg:

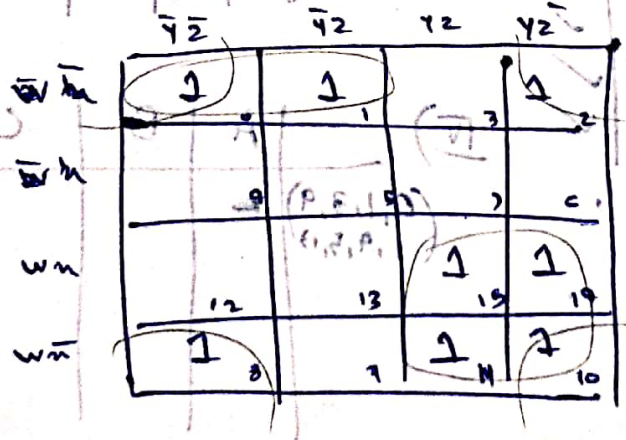
→ P_1, P_2, P_3 are essential prime implicants.

Eg



→ here P_4 is non essential

a) $F(w,x,y,z) = \Sigma\,(0,1,2,3,10,11,14,15)$



$$F(w,x,y,z) = x̄z̄ + w̄x̄ȳ + wy$$

Q) $F(A,B,C,D) = \Sigma(0,1,2,5,7,8,9,10,13,15)$

solve using not K-Map. find prime implicants.

	A	B	C	D
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
5	0	1	0	1
7	0	1	1	1
8				

I)

	A	B	C	D
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
8	1	0	0	0
5	0	1	0	1
9	1	0	0	1
10	1	0	1	0
7	0	1	1	1
13	1	1	0	1
15	1	1	1	1

II)

	A	B	C	D	
(0,1)	0	0	0	-	✓
(0,2)	0	0	-	0	✓
(0,8)	-	0	0	0	✓
(1,5)	0	-	0	1	✓
(1,9)	-	0	0	1	✓
(2,10)	-	0	1	0	✓
(8,9)	1	0	0	-	✓
(8,10)	1	0	-	0	✓
(5,7)	0	1	-	1	✓
(5,13)	-	1	0	1	✓
(7,13)	1	-	0	1	✓
(7,15)	-	1	1	1	✓
(13,15)	1	1	-	1	✓

III)

	A	B	C	D	
(0,1,8,9)	-	0	0	-	
(0,2,8,10)	-	0	-	0	
(0,8,1,9)	-	0	0	-	
(0,8,2,10)	-	0	-	0	
(1,9,5,13)	-	-	0	1	→ (15,9,13) = -01
(5,7,13,15)	-	1	-	1	
(5,13,7,15)	-	1	-	1	

IV)

	A	B	C	D
(0,1,8,9) (1,9,5,13)	-			

I)

	A	B	C	D
0	0	0	0	0 ✓
1	0	0	0	1 ✓
2	0	0	1	0 ✓
8	1	0	0	0 ✓
5	0	1	0	1 ✓
9	1	0	0	1 ✓
10	1	0	1	0 ✓
7	0	1	1	1 ✓
13	1	1	0	1 ✓
15	1	1	1	1

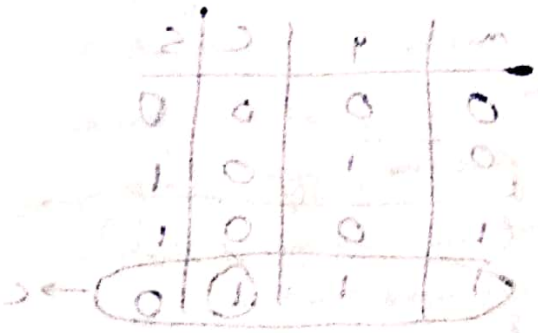
II)

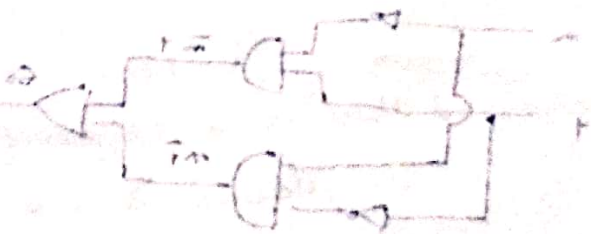
	A	B	C	D
(0,1)	0	0	0	-
(7,10)	-	0	1	0
(0,8)	-	0	0	0
(5,7)	0	1	-	1
(1,13)	1	-	0	1
(5,13)	1	1	-	1

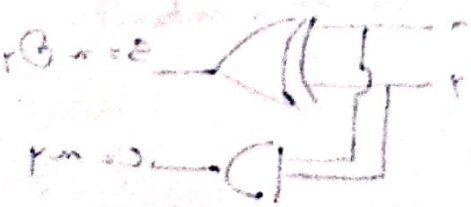
III)

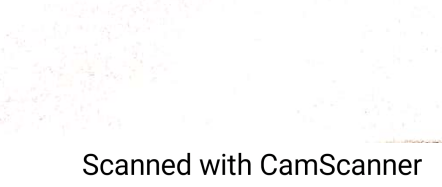
A	B	C	D

(wrong)

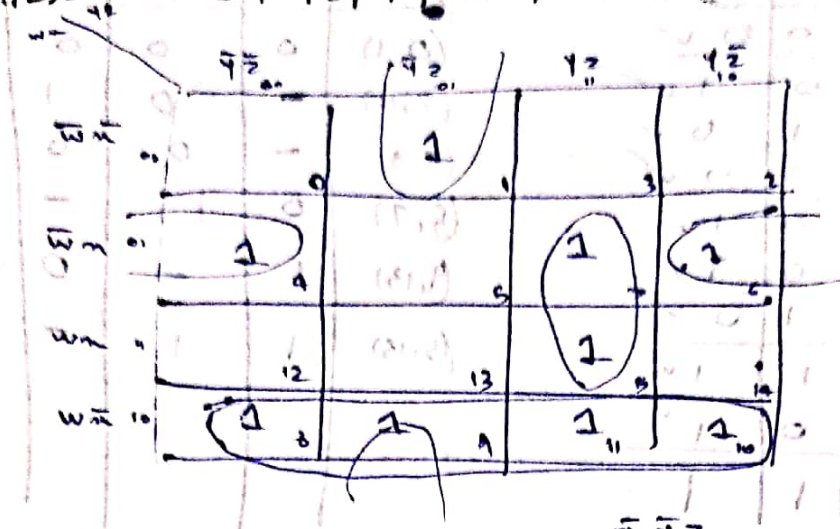






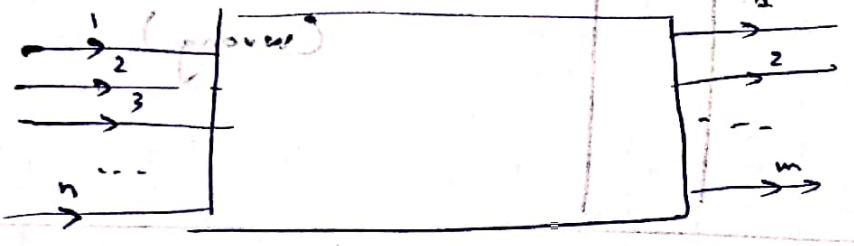


Q) $f(w,x,y,z) = \Sigma(1,4,6,7,8,9,10,11,15)$



$f(w,x,y,z) = w\bar{x} + xyz + \bar{w}x\bar{z} + \bar{x}\bar{y}z$

Combinational Circuit.



Half Adder

two inputs → x & y

0+0 = 0 0
0+1 = 0 1
1+0 = 0 1
1+1 = 1, 0
(1 → Carry, 0 → Sum)

two output → Sum and Carry

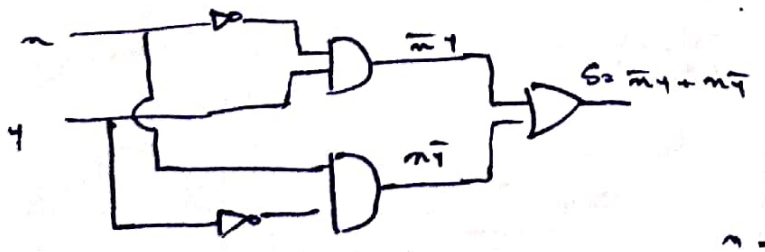
x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

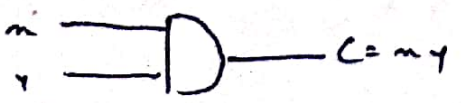
S → 0 1 1 0

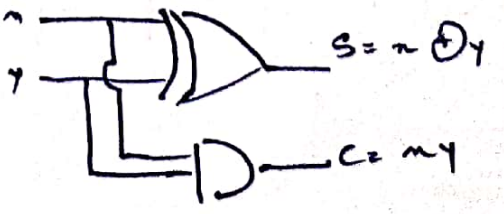
C → 0 0 0 1

$S = x\bar{y} + \bar{x}y = XOR(x,y) = x \oplus y$

$C = xy$







Full Adder

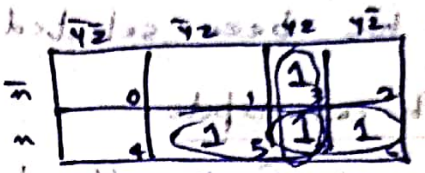
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1 ←
0	1	0	0	1 ←
0	1	1	1 ←	0
1	0	0	0	1 ←
1	0	1	1 ←	0
1	1	0	1 ←	0
1	1	1	1 ←	1 ←

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

$= \Sigma(1,2,4,7)$

$C = \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz$

$= \Sigma(3,5,6,7)$



Karnaugh Map

→ A Karnaugh Map may be considered as a pictorial representation of a truth table

→ It provides a straightforward procedure for minimizing boolean Function.

→ For an 'n'-variable function there are 2^n cells in the map (one for each minterm)

→ 2 adjacent 2 cell differ in only one variable

→ Adjacent $2^2 = 4$ cell differ in 2 variables

→ Adjacent 2^m cells differ in m variables.

→ We try to group 2^m cells corresponding true minterms. Try to make groups bigger ensuring all minterms are covered.

How to make sure this property is ensured?

→ two adjacent differ in the value of one variable.

→ helps in combining cells to group

Eg: $AB\bar{C} + AB\bar{C} = AC$

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

Drawback: Since it is pictorial approach, it is difficult to visualise function with more than 5 or 6 variables

Basic Approach

→ Fill up the cells of K-Map with true minterms of the function

→ Group the true minterms of the function into cubes such that

→ The size of cubes are maximised

→ Every true minterms is covered by at least one cube

→ Write down the minimized sum-of-product expression for the function by creating one product term out of every cube that has been selected.

Handling don't care inputs

→ There exists functions for which some of the inputs are treated as don't cares, and the corresponding output values does not matter.

→ Such input never appear

Eg: BCD values 1010 (10) to 1111 (15) never comes as input.

→ The don't care minterms are labelled as 'X' in the K-Map.

→ When creating the cubes, we can include cells marked 'X' along with those marked as '1' to make a larger cube.

→ But it is not necessary to cover all the cells marked by 'X'.

Implicant

→ Given a function f of n variables, a minterm p is an implicant of f iff, for all combination of n variables for which $p = 1$, $f = 1$

Eg: $f = A\bar{B}C + A\bar{C} + \bar{B}\bar{C}A$

→ all minterm here are implicants

Prime Implicant

→ An implicant is said to be prime implicant if after removing any literal from it, the resulting product is no longer an implicant.

→ With respect to K-Map, it is a cube that is not completly covered by another implicant representing a larger cube.

Eg: $f = \bar{A}B + AC + \bar{B}\bar{C}$

→ $\bar{A}B$ is an prime implicant

→ for $(A, B, C) = (0,1,0)$ and $(0,1,1)$, $\bar{A}B = 1$

→ if we remove $\bar{B}$ from $\bar{A}B$, resulting min term $\bar{A}$ is not an implicant, as $(0,0,0)$ and $(0,0,1)$ as input doesn't give 1 as output for $\bar{A}$.

Essential Prime Implicant

→ A prime implicant of a function is said to be essential prime implicant if it covers at least one minterm of the function that is not covered by any other prime implicant.

Some Results

→ Every irredundant SoP expression equivalent to a function F is a union of prime implicants of F.

→ The set of all essential prime implicants must be present in any irredundant SoP expression

→ Any prime implicant covered by the sum of essential prime implicants must not be present in an irredundant SoP expression

Simplification of Boolean Functions Using Tabulation method.

Motivation

- The k-Map method of minimization is convinient when the number of variables is less than six.
- The map method is a trial-and-error procedure that depends on human ability to recognize certain pattern.
- It is also difficult to automate the method.
- For large functions, a more systematic procedure is required and for that Quine - McCluskey method can be used.
- The Quine - McCluskey method is easy to automate.

Basic Concept

- Repeated application of $\bar{A}\bar{X} + AX = A$ to all adjacent pair terms. This will produce the set of all prime implicants at the end.
- This is a two step process. First generate all the prime implicants and then, select the prime implicant that will cover the all minterms of the function.
- We can go combing any pair of product terms that differ in the value of single literal.

1) Two k-variable can be combined into (k-1) variable term, iff they only differ in one variable.
2) We use binary representation of minterms for convinience
3) Two minterms can be combined if the binary representation differ in only one position.
4) we use symbol '-' to indicate absence of literal.

Identification of all Prime Implicants (Step-1)

→ Arrange all the minterms in groups based on the number of 1's. The number of 1's in a term is called it's index.

→ Compare every term of a group (index i), with each term of another group having index (i+1)

⇒ Merge the two term using rule, $AX + A\bar{X} = A$.

→ Place a check mark to each term that has been used in combining.

→ Now compare the terms that are generated as the output of the previous iteration in same fashion, generate a new term by combining two terms that differ by only one position.

→ The process continues until no further combinations are possible.

→ The remaining unchecked terms are the prime implicants of the function.

Eg: $F(w,x,y,z) = \Sigma(0,1,2,8,10,11,14,15)$

	w	x	y	z
0	0	0	0	0 ✓
1	0	0	0	1 ✓
2	0	0	1	0 ✓
8	1	0	0	0 ✓
10	1	0	1	0 ✓
11	1	0	1	1 ✓
14	1	1	1	0 ✓
15	1	1	1	1

	w	x	y	z
0,1	0	0	0	- ✓
0,2	0	0	-	0 ✓
0,8	-	0	0	0 ✓
2,10	-	0	1	0 ✓
8,10	1	0	-	0 ✓
10,11	1	0	1	- ✓
10,14	1	-	1	0 ✓
11,15	1	-	1	1 ✓
14,15	1	1	1	- ✓

	w	x	y	z
0,2,8,10	-	0	-	0
0,8,2,10	-	0	-	0
10,11,14,15	1	-	1	-
10,14,11,15	1	-	1	-

∴ The set of prime implicants are $\{\bar{w}\bar{x}\bar{z} + \bar{x}\bar{y}, wy\}$

Prime Implicit Chart (Step-2)

→ It is a tabular data structure, which pictorially depicts the covering relationship between prime implicants and minterms.

→ Useful to select the minimum set of prime implicants.

→ Minterms are listed along columns, while prime implicants listed along rows.

→ A 'X' is entered in the table if corresponding prime implicant covers corresponding minterm.

→ If a column has a single 'X', the prime implicant corresponding to the row in which the 'X' appears is an essential prime implicant.

Selection of essential Prime Implicants.

→ A check mark is placed in the chart to the essential prime implicants to indicate that they have been selected.

→ Next check each column whose minterm is covered by the essential prime implicants.

→ If the essential prime implicants do not cover all the minterms of the function, select the prime implicant that will cover the uncovered minterms.

Eg $F(w,x,y,z) = \Sigma(1,4,6,7,8,9,10,11,15)$

Prime implicant $\bar{x}\bar{y}z$, $\bar{y}x\bar{z}$, $\bar{w}xy$, xyz, wyz, $w\bar{x}$

		1	4	6	7	8	9	10	11	15
✓	$\bar{x}\bar{y}z$	1,9	X				X			
✓	$\bar{w}x\bar{z}$	4,6		X	X					
	$\bar{w}xy$	6,7			X					
✓	xyz	7,15			X					X
	wyz	11,15							X	X
✓	$w\bar{x}$	8,9,10,11				X	X	X	X	
			✓	✓	✓	✓	✓	✓	✓	✓

∴ essential pr

∴ $F(w,x,y,z) = \bar{x}\bar{y}z + \bar{w}x\bar{z} + xyz + w\bar{x}$

Q) Express the boolean function $F = xy + \bar{x}z$ as product of maxterm

$$F = xy + \bar{x}z$$
$$= (xy + \bar{x})(xy + z)$$
$$= (x + \bar{x})(y + \bar{x})(x + z)(y + z)$$
$$= (y + \bar{x})(x + z)(y + z)$$
$$= (\bar{x} + y + z\bar{z})(x + y\bar{y} + z)(x\bar{x} + y + z)$$
$$= (\bar{x} + y + z)(\bar{x} + y + \bar{z})(x + y + z)(x + \bar{y} + z)(x + y + z)(\bar{x} + y + z)$$
$$= (\bar{x} + y + z)(\bar{x} + y + \bar{z})(x + y + z)(x + \bar{y} + z)$$

Q) Find 10's complement of 132900.

→ 1's complement = ~~no.~~ 999999

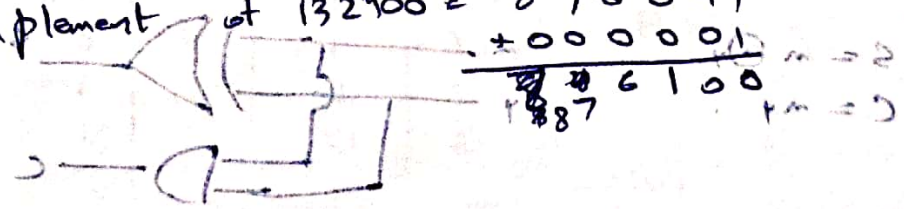
→ 2's complement = 1's complement + 1

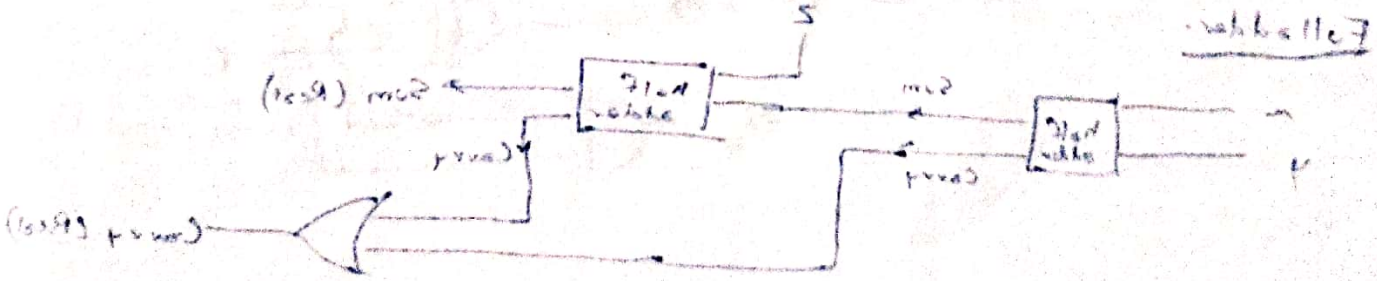
Similarly, 9's complement = 9999.... - no.

10's complement = 9's complement + 1

9's complement of 132900 = 999999
- 132900
867099

10's complement of 132900 = 867099
+ 000001
867100





Full adder Circuit

$S = x'y'z + x'yz' + xy'z' + xyz$

$= x \oplus y \oplus z$

$x \rightarrow (x \oplus y) \oplus z$

$= (xy' + yx') \oplus z$

$= z\overline{(xy' + yx')} + z'(xy' + yx')$

$= z(xy' \cdot \overline{yx'}) + xy'z' + x'yz'$

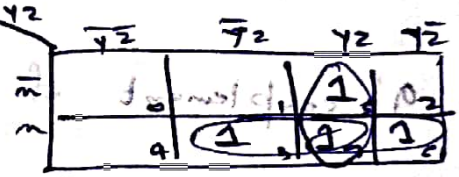
$= z(x' + y)(y' + x) + xy'z' + x'yz'$

$= x'y'z + x'yz' + xy'z' + xyz$

→ $S = \Sigma(1, 2, 4, 7)$

$C = \Sigma(3, 5, 6, 7) = x'yz + xy'z + xyz' + xyz$

$= xz + yz + xz$

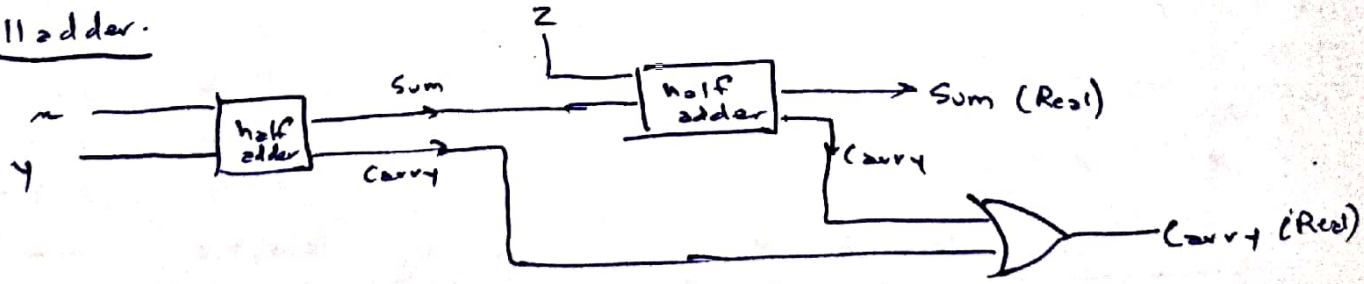


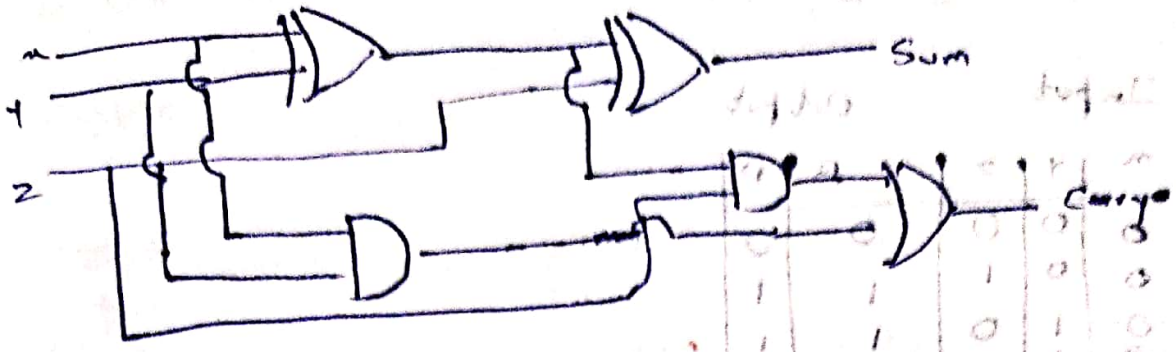
Half adder

$S = x \oplus y$

$C = xy$

Full adder.





x	y	z	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$C = z(x \oplus y) + xy$$
$$= z(xy' + yx') + xy(z+z')$$
$$= zxy'z + x'yz + xyz + xyz'$$

Half Subtractor

→ find x − y to give difference and borrow

Two cases:

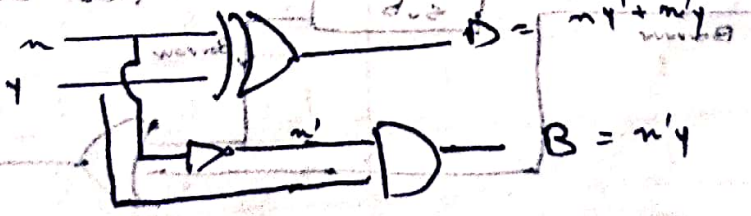
- $x \geq y$
  - $0 - 0 = 0$
  - $1 - 0 = 1$
  - $1 - 1 = 0$
- $x < y$
  - $0 - 1 =$ borrow 1 → $2 - 1 = 1$

x	y	B	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

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

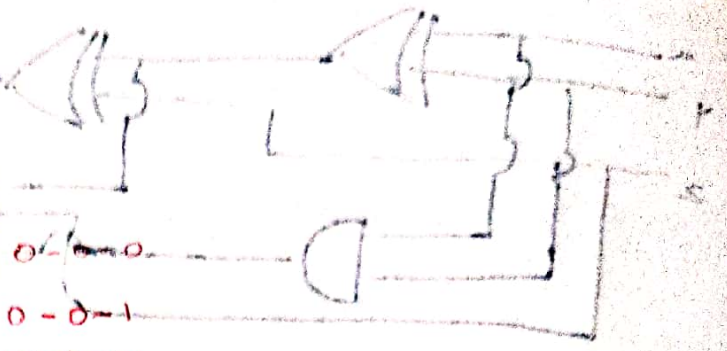
Same as S for half adder

$$B = x'y$$



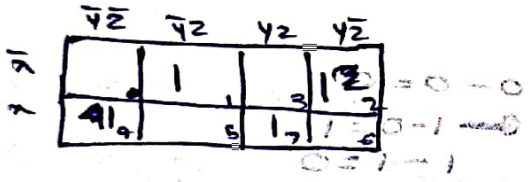
Full Subtractor (x-y-z)

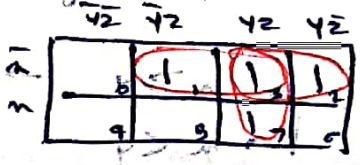
Input			Output	
x	y	z	B	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



$D = \Sigma(1,2,4,7)$

$B = \Sigma(1,2,3,7)$



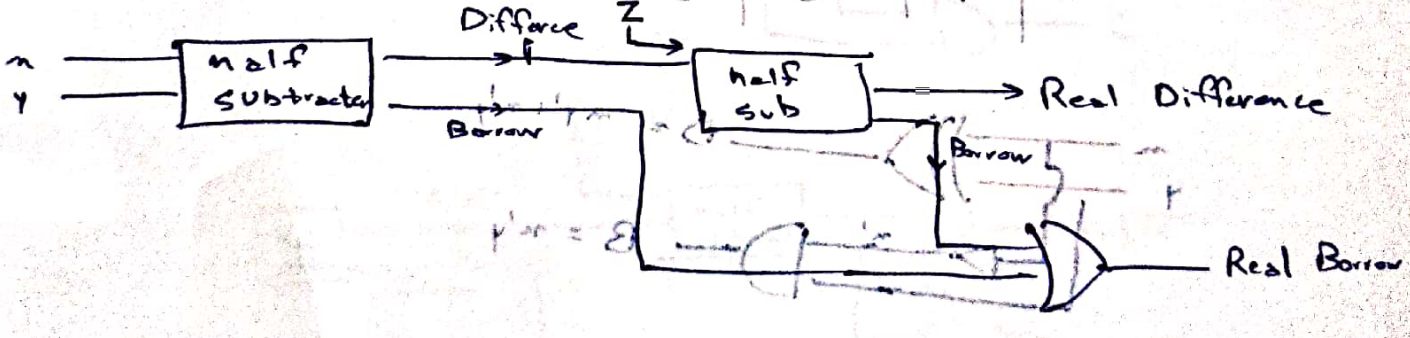


$D = x'y'z' + x'yz' + xy'z + xyz$

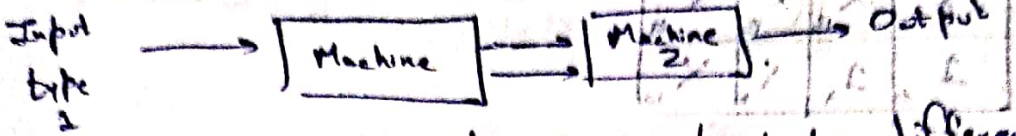
$B = x'z + yz + x'y$

Comparing Full Adder's Carry with Borrow

$C = xy + yz + xz$		0	0
$B = x'z + yz + x'y$	, i.e ,	only invert x in	Carry
1		0	
0		1	1



Code Conversion



→ some times 'Machine 2' take different type of input

→ we need 'code conversion'

excess 3 (add 3)

0 → 0000 + $(3)_{10}$ = 0011

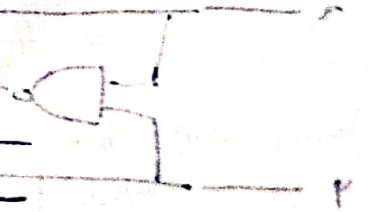
1 ⟶ 0100

2 ⟶ 0101

BCD to excess 3 BCD

number	A	B	C	D	excess 3 w	x	y	z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

(6 combination [10, 11, 12, 13, 14, 15] are don't care)



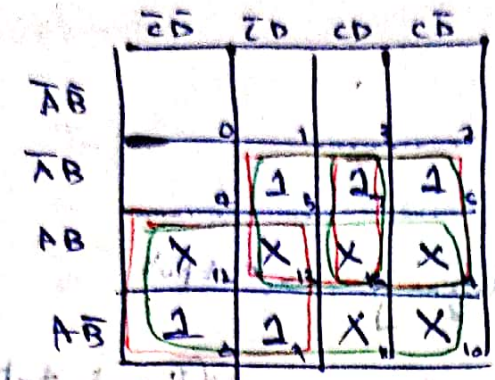
now $w = \Sigma(5,6,7,8,9) + \Sigma d(10,11,12,13,14,15)$

$x = \Sigma(1,2,3,4,9) + \Sigma d(\ldots)$

$y = \Sigma(0,3,4,7,8) + \Sigma d(\ldots)$

$z = \Sigma(0,2,4,6,8) + \Sigma d(\ldots)$

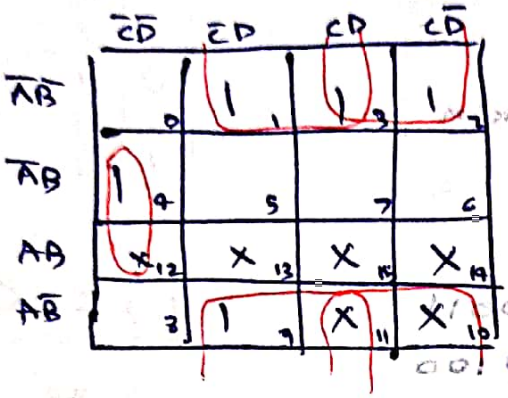
1/



$w = AC' + BD + BC$

shorter → $w = A + BD + BC$

2/

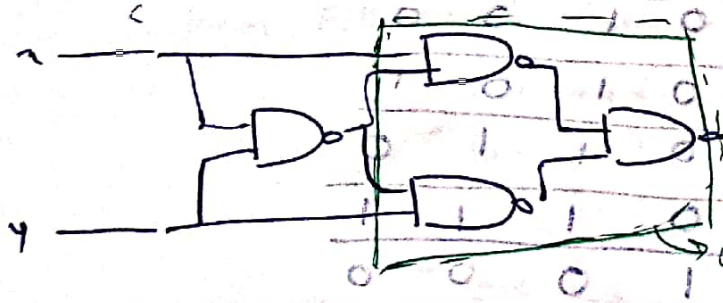


$x = B'D + B'C + BC'D'$

similarly $y = CD + CD'$

$z = D'$

XOR



0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	1	2
0	0	1	1	0	3
0	1	0	0	1	4
0	1	0	1	0	5
0	1	1	0	0	6
0	1	1	1	1	7
1	0	0	0	1	8
1	0	0	1	0	9
1	0	1	0	0	10
1	0	1	1	1	11
1	1	0	0	0	12
1	1	0	1	1	13
1	1	1	0	1	14
1	1	1	1	0	15

∴ simplification → $\overline{xy} = \bar{x} + \bar{y}$

$x(\bar{x} + \bar{y}) + y(\bar{x} + \bar{y})$

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

~~$(A \oplus B)'$~~

$A \oplus B \oplus C \oplus D = (A \oplus B) \oplus (C \oplus D)$

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

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

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

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

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

$= A'B'CD' + A'B'C'D + ABCD' + ABC'D$

$+ AB'C'D' + A'BC'D' + AB'CD + A'BCD$

$= \Sigma(1, 2, 4, 7, 8, 11, 13, 14)$

comparing with $A \oplus B \oplus C = \Sigma(1, 2, 4, 7)$

$\therefore$ no. of terms in $A_1 \oplus A_2 \oplus \dots A_n = \frac{2^n}{2}$

also $\Sigma(1, 2, 4, 7, 8, 11, 13, 14)$ =

0	0	0	1	
0	0	1	0	
0	1	0	0	
0	1	1	1	→ all have odd no. of 1
1	0	0	0	
1	0	1	1	
1	1	0	1	
1	1	1	0	

$\therefore$ all XOR output will have all odd number of one.

$\therefore$ also XNOR output will have even number of one.

and also has $\frac{2^n}{2}$ terms.

Eg: using generate even parity generator →

x	y	z	p
0	0	0	0
0	0	1	1 ←
0	1	0	1 ←
0	1	1	0
1	0	0	1 ←
1	0	1	0
1	1	0	0
1	1	1	1 ←

all have odd 1 in (x,y,z)

∴ $p = x \oplus y \oplus z$

Eg: Even parity checker → no error → 0
↳ error → 1

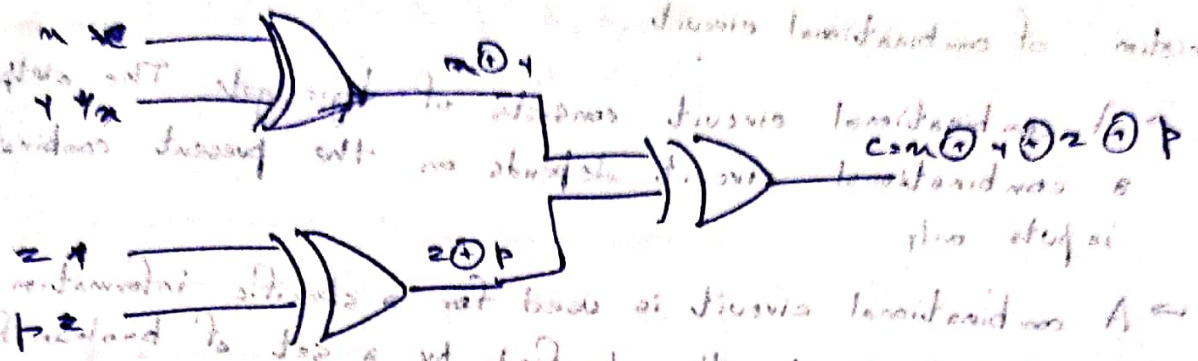
Input				Output
x	y	z	p	c
0	0	0	0	0
0	0	0	1	1 ←
0	0	1	0	1 ←
0	0	1	1	0
0	1	0	0	1 ←
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1 ←
1	0	0	0	1 ←
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1 ←
1	1	0	0	0
1	1	0	1	1 ←
1	1	1	0	1 ←
1	1	1	1	0

all have odd no. of 1

∴ $c = x \oplus y \oplus z \oplus p$

i.e same circuit will act as both parity checker and generator

→ Parity Checker circuit



→ to make it parity generator, put $p = 0$, which makes

$$z \oplus p = z$$

Odd parity generator → make no. of 1 → odd

x	y	z	p
0	0	0	1 ←
0	0	1	0
0	1	0	0
0	1	1	1 ←
1	0	0	0
1	0	1	1 ←
1	1	0	1 ←
1	1	1	0

all will have input (x,y,z) containing even. no. of 1

→ similarly we can make odd parity checker with $c = x \text{ XNOR } y \text{ XNOR } z \text{ XNOR } p$

COMBINATIONAL LOGIC

Definition of combinational circuit

→ A combinational circuit consists of logic gate. The outputs of a combinational circuit depends on the present combination of inputs only.

→ A combinational circuit is used for a specific information processing task which is logically specified by a set of boolean function

Eg i) Adder / Subtractor
ii) Multiplexer
iii) De-multiplexed / Decoder.

Design Procedure

→ The design of a combinational circuit starts from the verbal (written) specification of the problem and ends in a logic diagram or a set of Boolean Functions from which the logic diagram can be easily obtained. We have to follow the steps given below for designing a combinational circuit.

→ Identify the numbers of input and output variables.
→ denote each of the variable with a letter symbol
→ Obtain the truth table which defines the relationship between input and output variables.
→ Simplify the boolean expression for each of the output variable.
→ Draw the logic diagram.

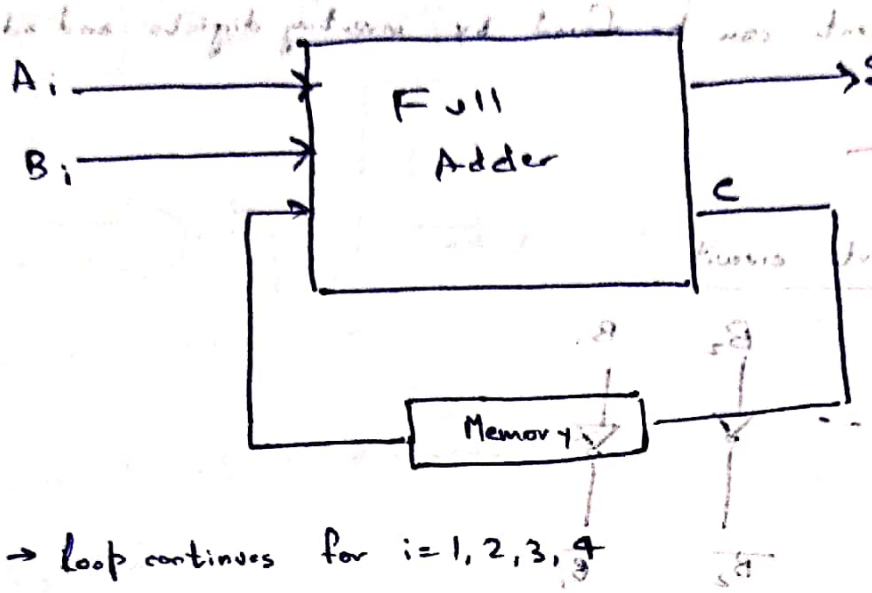
Exclusive OR Function

→ $x \oplus 0 = x$
→ $x \oplus x = 0$
→ $x \oplus \bar{y} = \overline{x \oplus y}$
→ $x \oplus 1 = \bar{x}$
→ $x \oplus \bar{x} = 1$
→ $\bar{x} \oplus y = \overline{x \oplus y}$

Ripple Carry Adder

let we need to add $A = A_1, A_2, A_3, A_4$

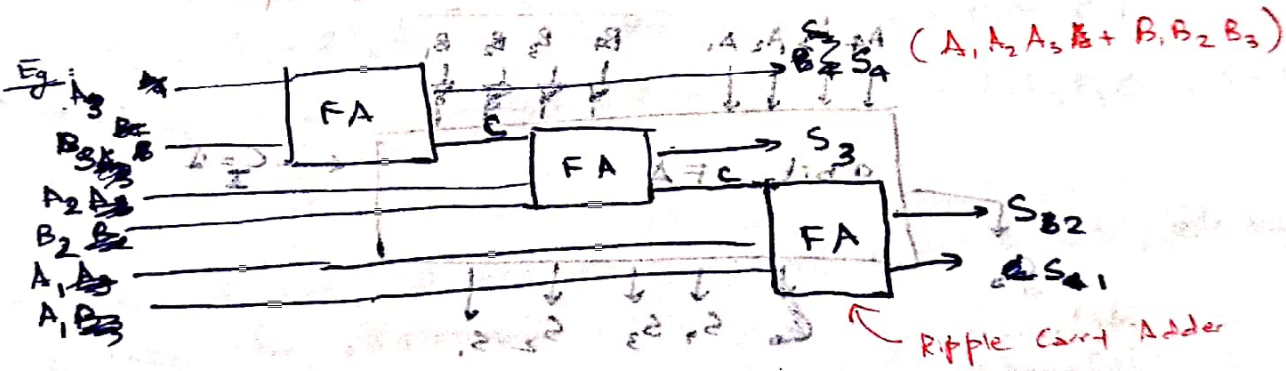
$B = B_1, B_2, B_3, B_4$



→ Series Adder

→ loop continues for $i = 1, 2, 3, 4$

→ Parallel adder also exists. Eg: Ripple Carry Adder.



Time Consumption

→ Let one FA take t time for it's work

$\therefore C_2 = t$

$C_3 = 2t$

$C_4 = 3t$

$\vdots$

$C_n = (n-1)t$

$C_{n+1} = nt$

→ C_{n+1} exist as 4 bit adding can make 5 bit ans.

4 bit Subtractor

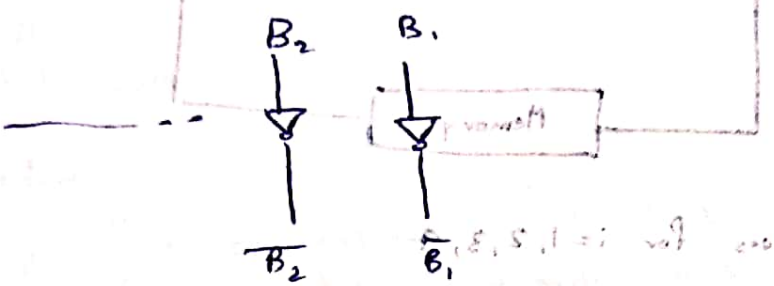
$A_1A_2A_3A_4 - B_1B_2B_3B_4 \Leftrightarrow A - B$

→ can be done by adding A and 2's complement of B

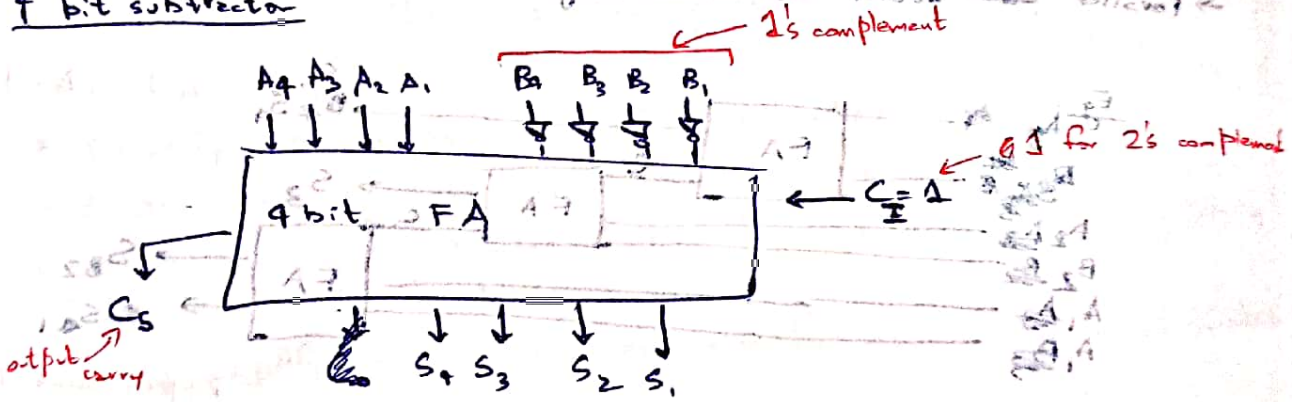
→ 2's complement can be found by inverting digits and adding 1 by

1.

→ one complement circuit



→ 4 bit subtractor



→ i.e same circuit can act as a adder and subtracter.

→ we no know that $B \oplus 0 = B$

and $B \oplus 1 = \overline{B}$

→ so we can use XOR gate to make the inversion and carry pass

Drawback of Ripple carry Adder

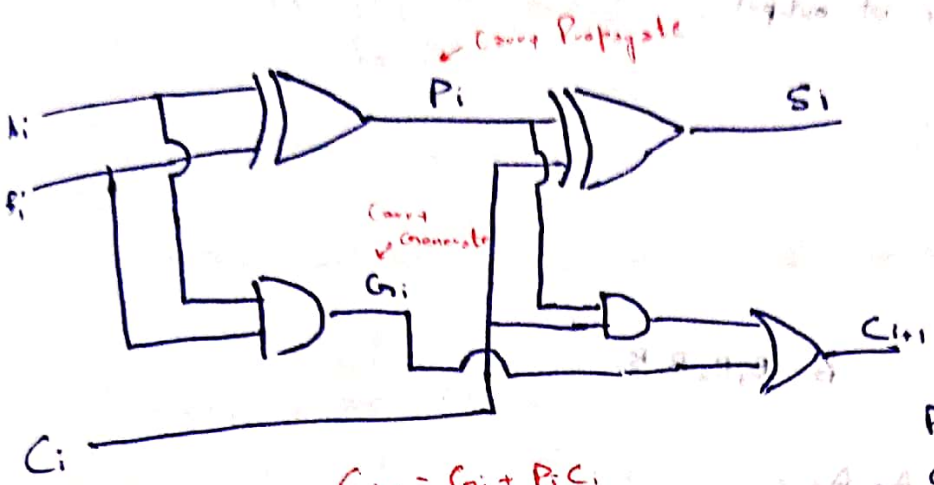
→ Total delay ∝ no. of bits (n)

→ Performance bad for high n

→ Carry is bottleneck, which propagates from one stage to another

Look- Ahead Carry Adder.

⇒ normal Full Adder



$C_{i+1} = G_i + P_i C_i$

$P_i = A_i \oplus B_i$

$G_i = A_i \cdot B_i$

$C_{i+1} = G_i + P_i C_i$

→ Consider 4 bit parallel adder.

~~$C_2 = G_1 +$~~

$C_2 = G_1 + P_1 C_1$

$C_3 = G_2 + P_2 C_2 = G_2 + P_2 (G_1 + P_1 C_1) = G_2 + G_1 P_2 + P_1 P_2 C_1$

$C_4 = G_3 + B_3 P_3 C_3 = G_3 + B_3 P_3 (G_2 + P_2 G_1 + P_1 P_2 C_1)$

$= G_3 + B P_3 G_2 + P_3 P_2 C_1 + P_1 P_2 P_3 C_1$ (SOP)

→ This way we can calculate all carry before the start, ~~etr~~ decreasing time taken for addition

→ After initial delay, all additin can be done in parallel

→ Time Complexity = $O(1)$, i.e delay $\propto$ initial step

→ Hardware complexity high for high n

Magnitude Comparator

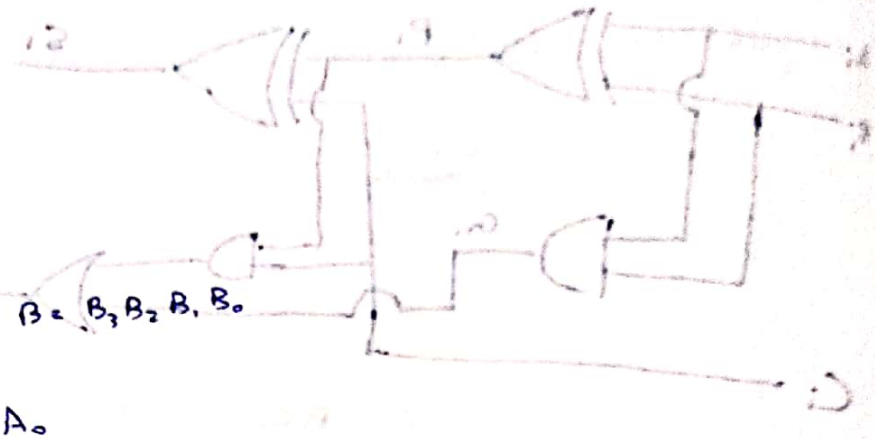
→ gives magnitude difference between two numbers (let these be A and B)

→ Three possibilities of output

i) $A = B$

ii) $A > B$

iii) $A < B$



Eg: $A = A_3 A_2 A_1 A_0$ $\quad B = B_3 B_2 B_1 B_0$

$\therefore$ $A_3\ A_2\ A_1\ A_0$

$B_3\ B_2\ B_1\ B_0$

I) equality

→ check B_3 and A_3

↳ if $B_3 = A_3$, check $A_2 B_2$

↳ if $A_2 = B_2$, check $A_1 B_1$

↳ if $A_1 = B_1$, check A_0, B_0

↳ if $A_0 = B_0$ ⟶ $A = B$

⇒ To check equality, we use XNOR

$X_i = 1$ only if $A_i = B_i$

$\therefore A = B$ only if $A_3 \odot B_3$, $A_2 \odot B_2$, $A_1 \odot B_1$, $A_0 \odot B_0$ is 1

→ all for XNOR gate is fed to four input AND Gate, this will give 1 only if All Digits are same.

II) $A > B$ ~~check~~ or $A < B$

→ check A_3, B_3

↳ if $A_3 = B_3$, check A_2, B_2

↳ if

→ There will be a case when $A_i \neq B_i$, then $A_i > B_i$ or $A_i < B_i$.

∴ we can do the following

$$(A>B) = \underbrace{A_3 \cdot B_3'}_{\text{first term}} + \underbrace{m_3 A_2 B_2'}_{\text{second term}} + \underbrace{m_3 m_2 A_1 B_1'}_{\text{third time}} + \underbrace{m_3 m_2 m_1 A_0 B_0'}_{\text{fourth term}}$$

will be 1 if any $A_i > B_j$

where $m_i = A_i B_i + A_i' B_i'$ → XNOR gate → gives 1 if $A_i = B_i$

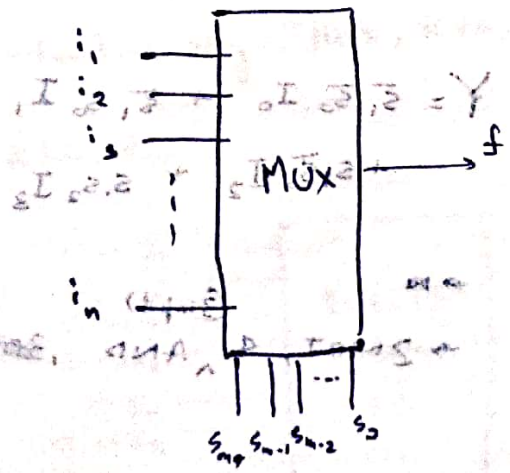
Similarly,

$$(A<B) = A_3' B_3 + m_3 A_2' B_2 + m_3 m_2 A_1' B_1 + m_3 m_2 m_1 A_0' B_0$$

↑ will be 1 if any $A_i < B_j$

~~→ 4 XNOR, 4 AND, 1 OR gate required.~~

Multiplexer



0	0	x	x	x	0	0
1	1	x	x	x	0	0
0	1	0	x	x	1	0
1	x	1	x	x	1	0
0	x	x	0	x	0	1
1	x	x	1	x	0	1
generally i_1	x	x	x	0	1	1
1	x	x	x	1	1	1

$n = 2^m$

2-to-1 MUX

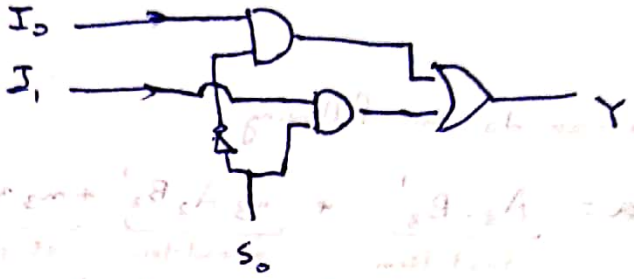
2 input, 1 select line, 1 otput line.

S_0 \ $I_1 I_0$	$\bar{I}_1\bar{I}_0$	$\bar{I}_1 I_0$	$I_1 I_0$	$I_1\bar{I}_0$
$\bar{S}_0$	0	1	1	0
S_0	0	0	1	1

$$Y = \bar{S}_0 I_0 + S_0 I_1$$

→ 1 NOT, 2 AND, 1 OR

S_0	I_1	I_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



4-to-1 MUX

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

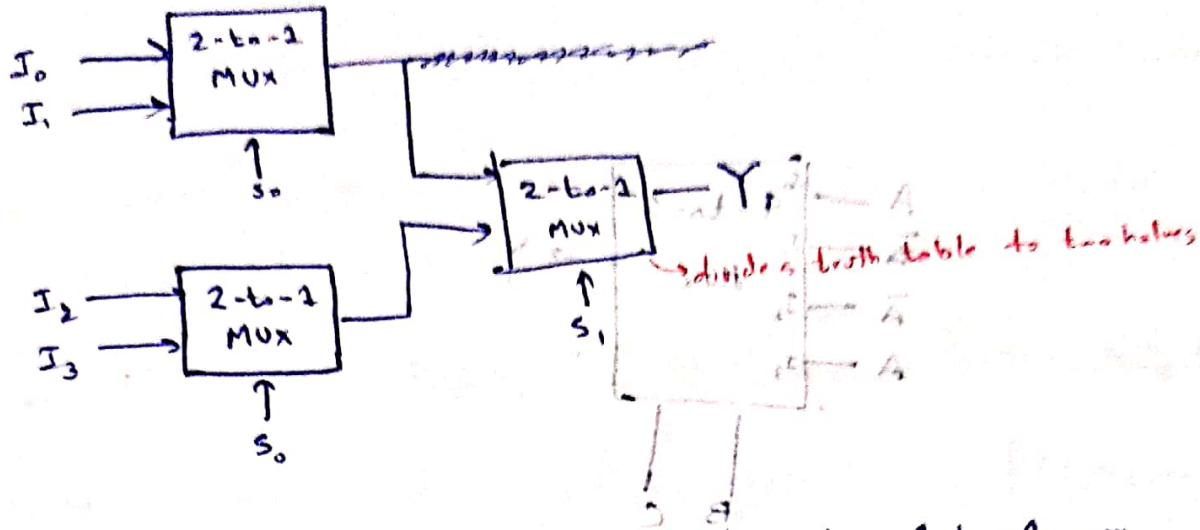
x → any value

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

(3 input) 1x 4 input OR

→ 2 NOT, 4 AND, ~~3 OR~~ 4 m

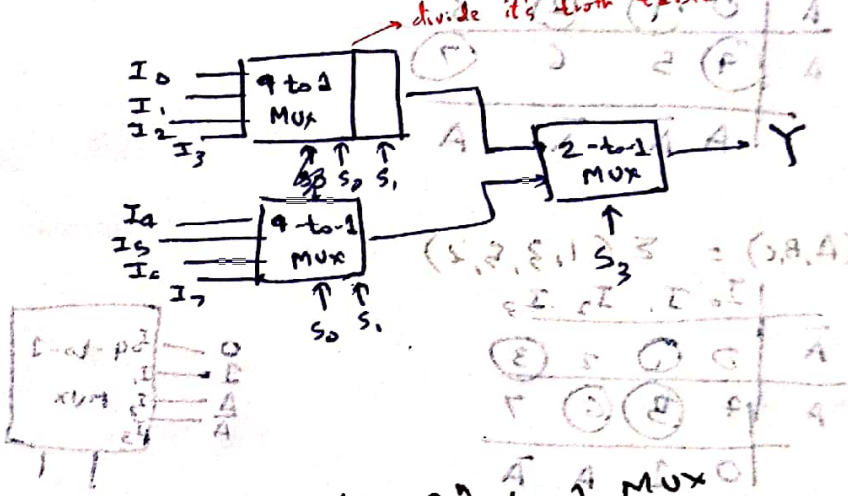
we can use 2-to-1 MUX's to make 4-to-1 MUX



similarly we can make 8-to-1 MUX to make 4-to-1 mux

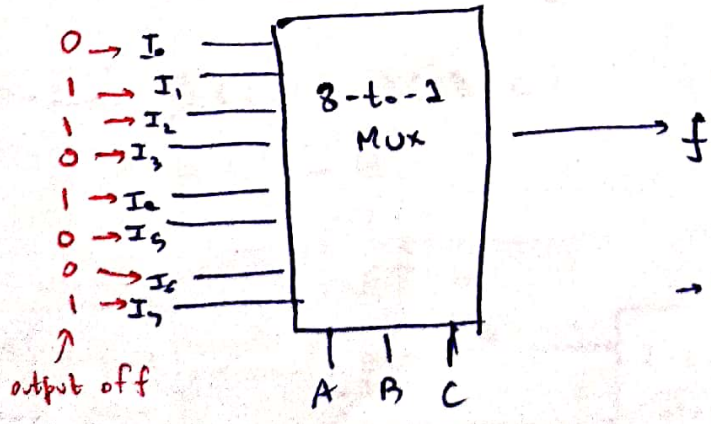
8-to-1

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



Forming logic function using MUX, n variable 2^n-to-1 MUX

Eg: Full Adder Sum

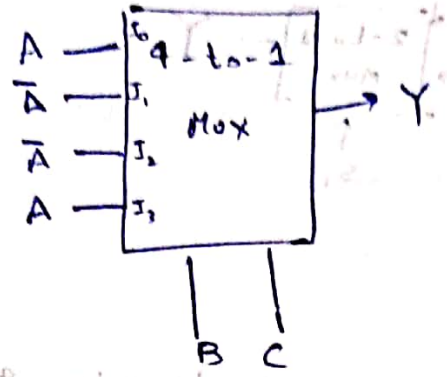


→ if A, B, C = 0, 0, 0, I_0 will be selected

hence I_0 should be output of f corresponding to (0,0,0)

n-variable, 2^{n-1}-to-1 MUX

Eg: $f(A,B,C) = \Sigma(1,2,4,7)$ (full adder)



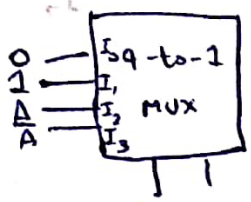
	I_0	I_1	I_2	I_3
$\bar{A}$	0	(1)	(2)	3
A	(4)	5	6	(7)
	A	$\bar{A}$	$\bar{A}$	A

→ implementation table

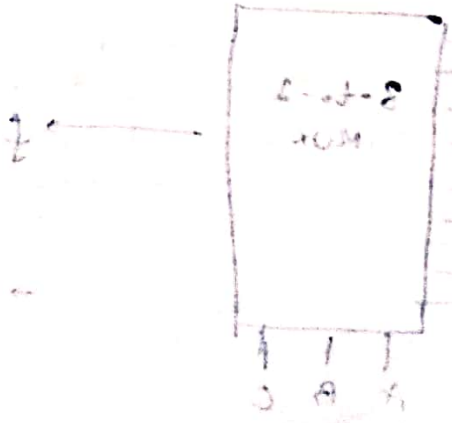
Eg: $f(A,B,C) = \Sigma(1,3,5,7)$

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

$A \oplus B$



can be interchanged as there is significance



* 42 is not 101010, but 0101010, we have to show magnitude form for positive number.

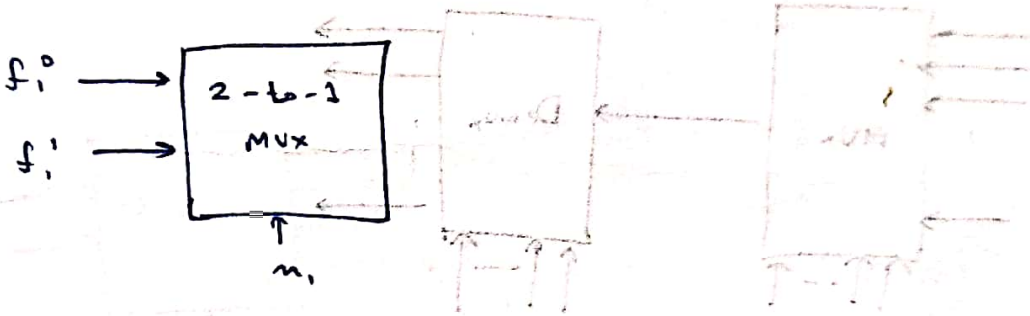
* ~~SOP~~ POS for is not $\Pi(0,1)$, but is $(\_\_).(\_\_).(\_\_)$

* only write gray code numbers in binary in gray, no letters or decimal numbers, sir might cut marks ☹

Shannon's Decomposition Theorem, and expression of any function using MUX

$$f(x_1, x_2, \ldots x_n) = \bar{x}_1 f(0, x_2, x_3, \ldots x_n) + x_1 f(1, x_2, \ldots x_n)$$

$$= x_1' f_1^0 + x_1 f_1'$$



Eg: $f = A'B + BC + AC'$

$f(A,B,C) = A'B + BC + AC'$

$= A' f(0,B,C) + A f(1,B,C)$

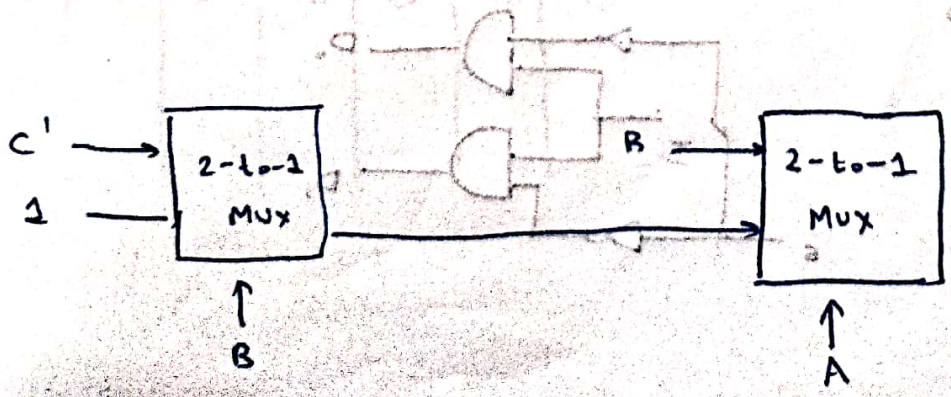
$= A'(B + BC) + A(BC + C')$

$= A'(B) + A(BC + C')$

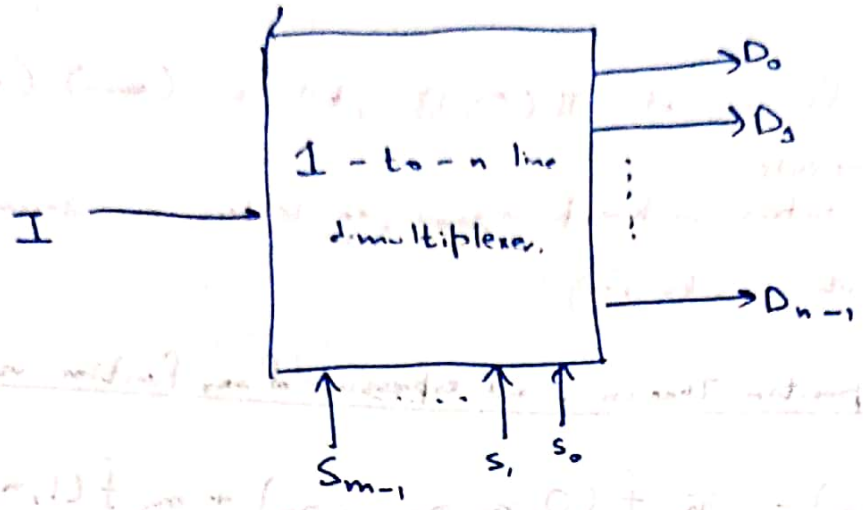
$\because xx'y = xy$

now, $B + C' = B'(C') + B(1 + C')$ (by Shannon's Theorem)

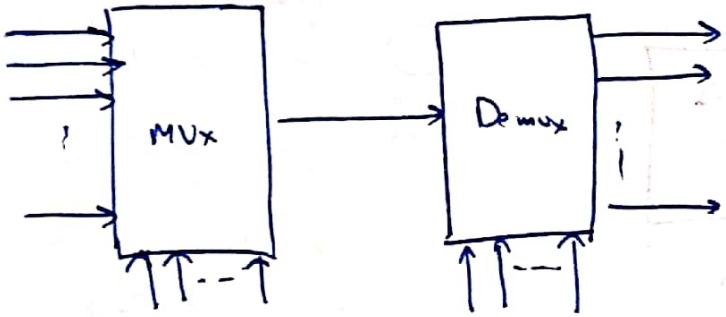
$= B'C' + B$



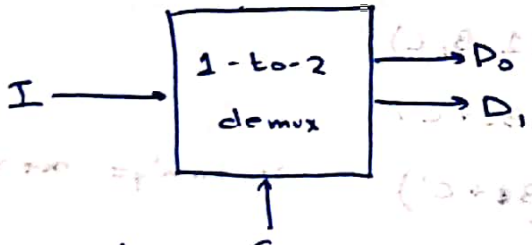
Demultiplexer



general use case



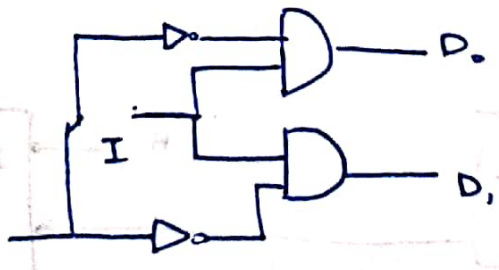
1 - to - 2 line demux



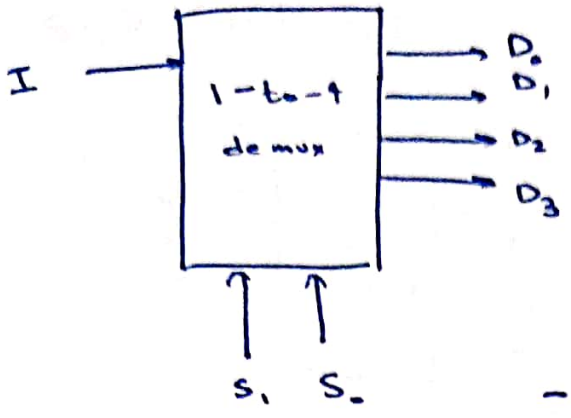
$D_0 = \overline{S_0} I$

$D_1 = S_0 I$

Input		output	
I	S_0	D_1	D_0
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	0



1-to-4 line demuxer



$D_0 = \bar{S}_1 \bar{S}_0 I$

$D_1 = \bar{S}_1 S_0 I$

$D_2 = S_1 \bar{S}_0 I$

$D_3 = S_1 S_0 I$

- 2 not, 4 three input and

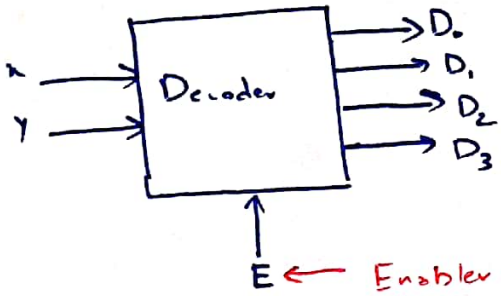
$S_1S_0 = 00 \rightarrow D_0$

$S_1S_0 = 01 \rightarrow D_1$

$S_1S_0 = 10 \rightarrow D_2$

$S_1S_0 = 11 \rightarrow D_3$

Decoder



E	x	y	D_0	D_1	D_2	D_3
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1
1	X	x	0	0	0	0

$D_0 = x'y'$

$D_1 = x'y$

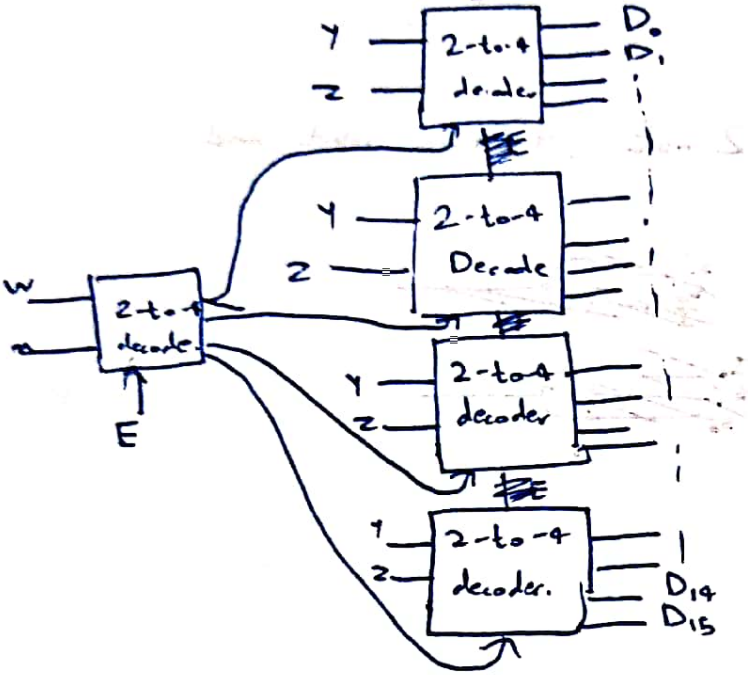
$D_2 = xy'$

$D_3 = xy$

4-to-16 line decoder using 2-to-4 line decoder

Input: w x y z

Output: $D_0, D_1, \ldots D_{15}$



→ we observe that

$D_0 = x'y'$

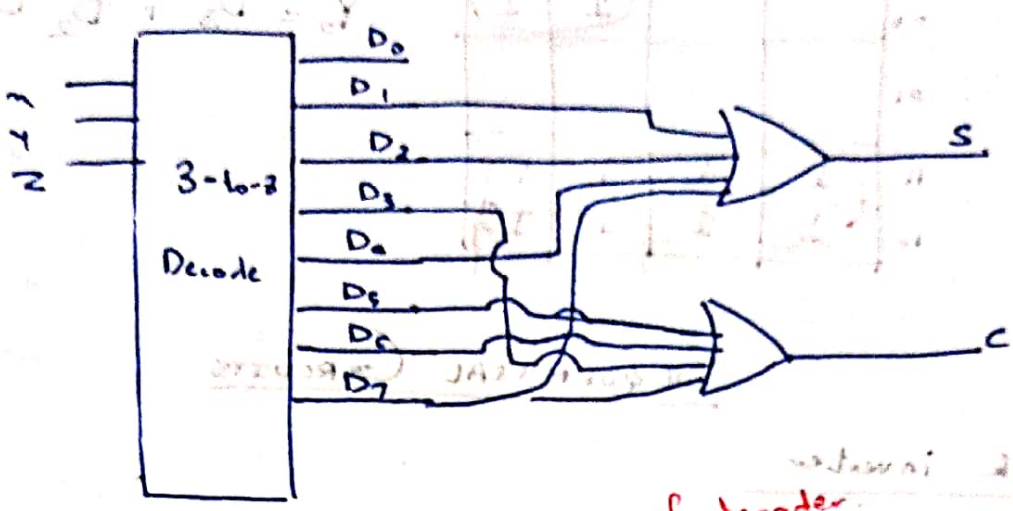
$D_1 = x'y$

$D_2 = xy'$

$D_3 = xy$ → minterms

Full Adder using Decoder.

$S = \Sigma(1,2,4,7)$, $C = \Sigma(3,5,6,7)$



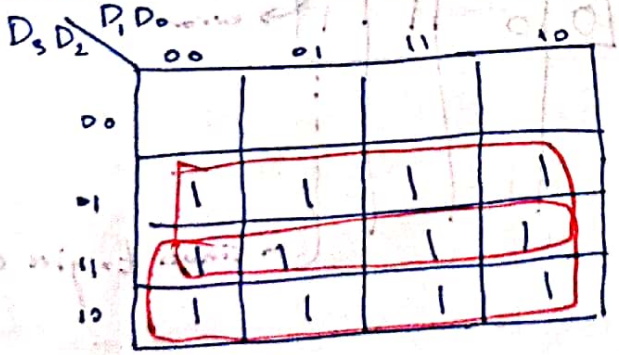
4 to 2 line Priority Encoder → inverse of decoder

D_3	D_2	D_1	D_0	Y_1	Y_0
0	0	0	0	X	X
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1

X → any value (either 0 or 1)

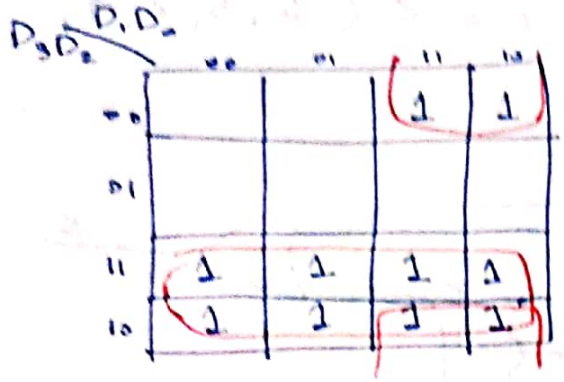
priority : $D_i > D_j$ $(i > j)$

Solving Y_1



$Y_1 = D_2 + D_3$

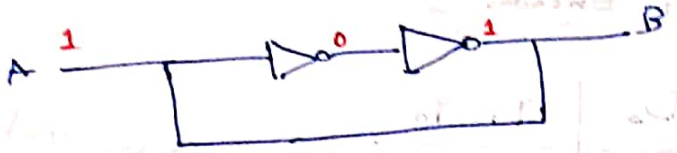
Solving Y_0



$$Y_0 = D_3 + \overline{D_2} D_1$$

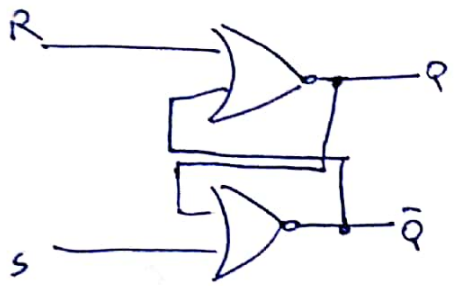
SEQUENTIAL CIRCUITS

Feedback inverter



→ This will give and store 1, as long as we are giving this power

S-R-Latch



→ NOR: any input 1, output 0

Consider initial output 00

S	R	Q	$\bar{Q}$	
1	0	1	0	then every output cascade to next one
0	0	1	0	
0	1	1	0	
0	0	0	1	
1	1	0	0	
0	0	1	1	→ error

(Continues to give error)

∴ if Q and $\bar{Q}$ comes to be same, output will continue to oscillate between 00 and 11, giving error

→ To avoid this, we make speed of gate different, both will not have same output, this is called race condition.

→ also, we avoid putting 11 in S-R Latch

~~S-R Latch~~

→ also 00 just copies previous out-put

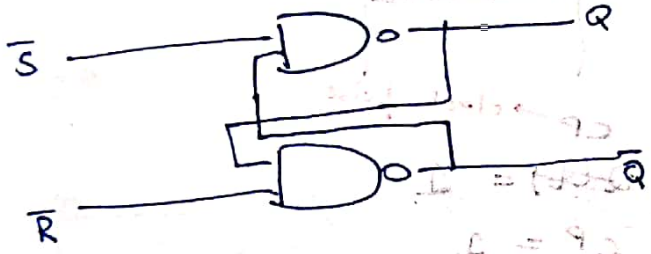
Set condition : S=1, R=0 ; Q=1 $\bar{Q}$=0

Rear condition : S=0, R=1 ; Q=0 $\bar{Q}$=1

Copy : S=0, R=0

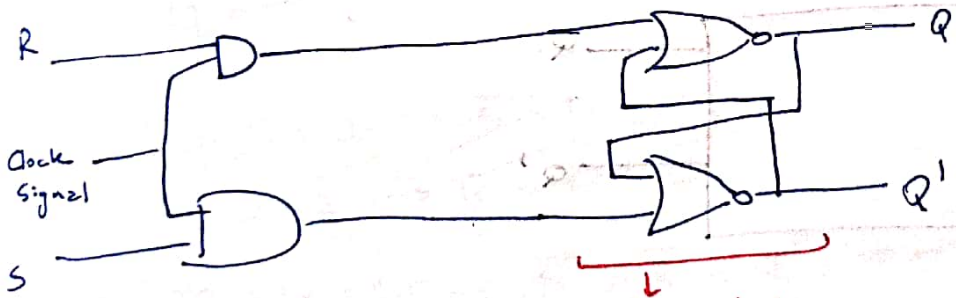
Invalid : S=1, R=1

S-R Latch NAND

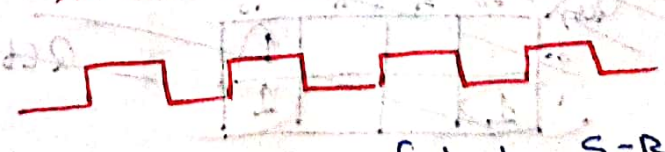


$\bar{S}$	$\bar{R}$	Q	$\bar{Q}$	
1	0	1	0	→ Set
0	1	0	1	→ Rear.
0	0			→ copy
1	1	x	x	→ invalid

S-R-Flip Flop



§ Clock Signal:



→ clock = 0 → no S and R will be fed to S-R Latch

→ clock = 1 → S and R fed to Latch

Present Q(t)	S	R	output Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	indeterminate
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	indeterminate

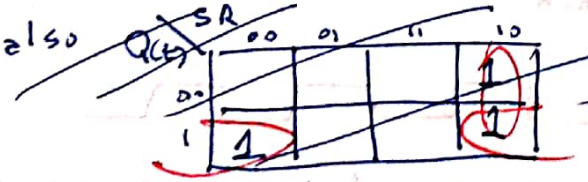
CP → clock pulse

here Set: $S=1, R=0,$ ~~$Q(t)$~~ $= 1$

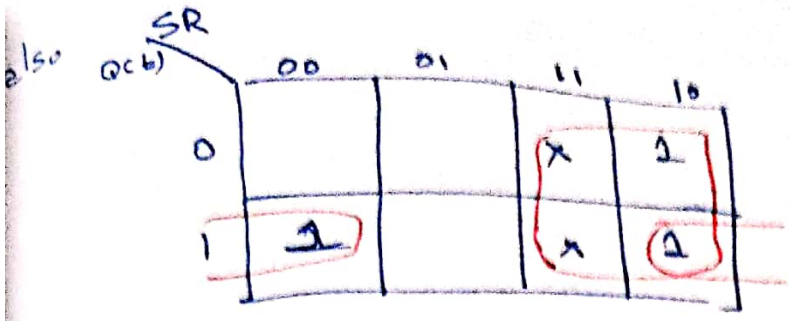
clear ~~Rear~~: $S=0, R=1,$ $CP = 1$

Symbolic Diagram

also

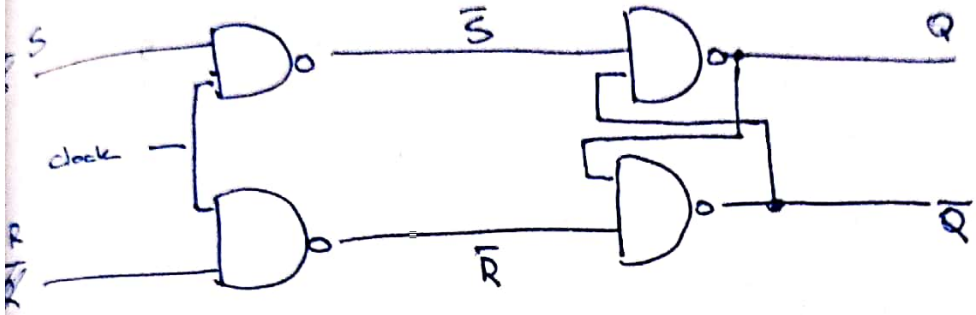


$Q(t+1) = Q(t)R' + SR'$

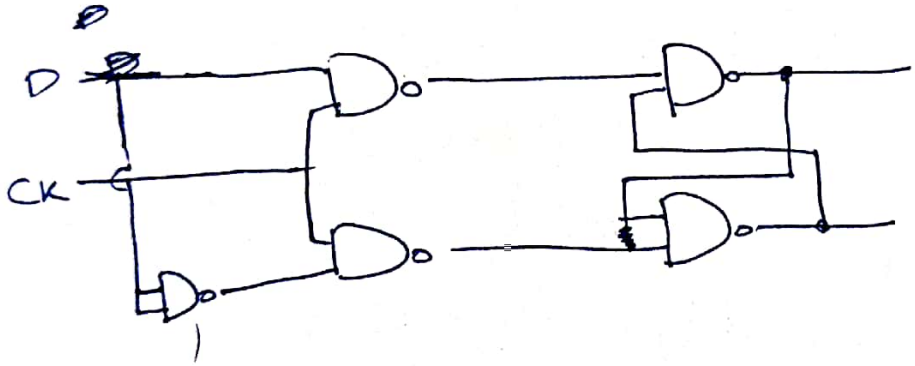


$Q(t) = S + Q(t)\,R'$

S-R- Flip Flop [NAND]



D flip flop



if D = 0 → $\bar{S}$ = 1, $\bar{D}$ = 1

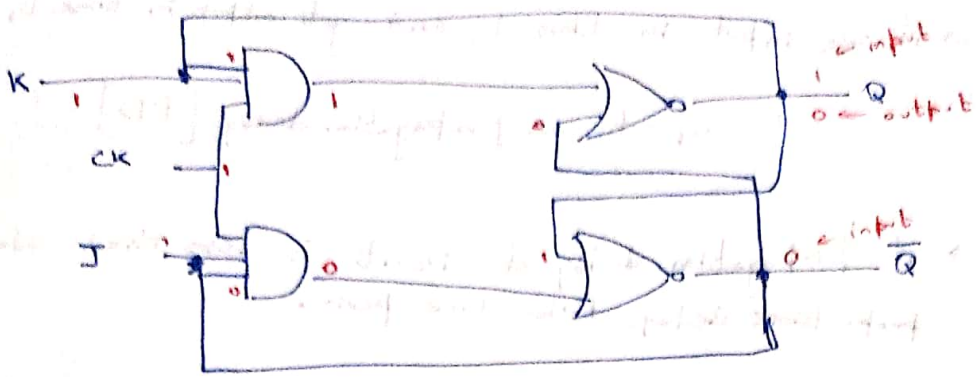
D = 1 → $\bar{R}$ ← D = 0

→ CK = 0 → $\bar{S} = \bar{R} = 1^{0}$ → no change

→ CK = 1 → D = 1 → $\bar{S} = 0$ → set
$\bar{R} = 1$

D = 0 → $\bar{S} = 1$ → clear state
$\bar{R} = 0$

J-K Flip Flop



→ This avoids the non-stable output when J = K = 1 (just flips the output)

Eg: Q = 1, $\bar{Q}$ = 0, J = 1, K = 1, CK = 1

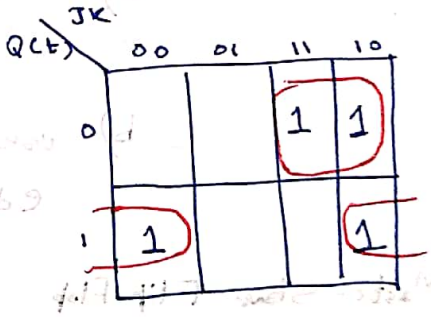
output Q = 0, $\bar{Q}$ = 1

Eg: Q = 0, $\bar{Q}$ = 1, J = K = CK = 1

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

Characteristic Table

Q(t)	J	K	Q(t+1)
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



→ when S = 1, R = 0 → Set condition.

∴ J = 1, K = 0, Q = 1

→ when S = 0, R = 0, Q = 0 → Reset Condition.

∴ J = 0, K = 1, Q = 0

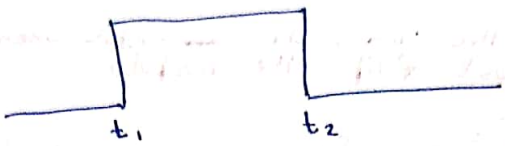
→ From K-Map: $Q(t+1) = Q(t)'J + Q(t)K'$

→ suppose J=K=1

→ if we input in time t and get output in time t_1

$\therefore t_1 - t$ = propagation delay [PD]

→ if propagation delay of circuit is less than clock ~~prop. time delay.~~ Pulse time period



clock pulse time period ← $(t_2 - t_1) > PD$

→ we may get multiple change within a single clock pulse. This is called ~~"racing condition"~~ "race-in condition"

→ This can be avoided by using a clock with

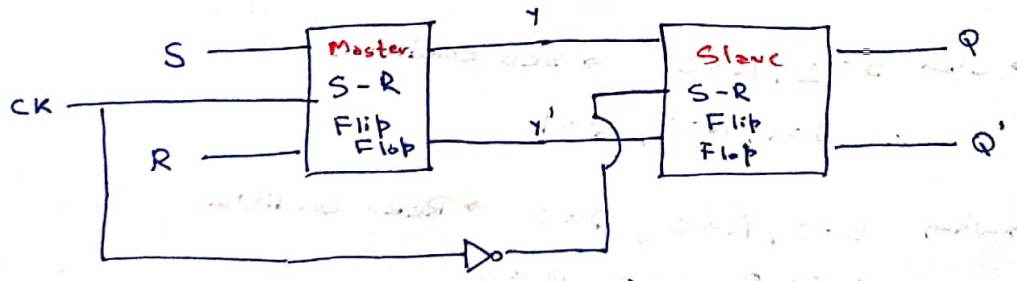
a) clock Time period ≤ Propagation delay

(Not a wise way)

b) use master-slave flip-flop or edge-triggered flip-flop

SR Master-Slave Flip flop

→ uses two flip flops, one is master and other is slave

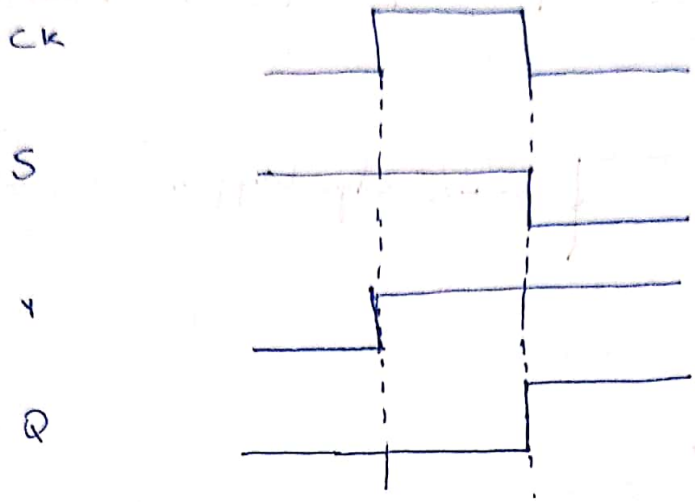


→ This solves the ~~rcc~~ "race-in condition"

end result

→ ~~only~~ output chage only if ck=0

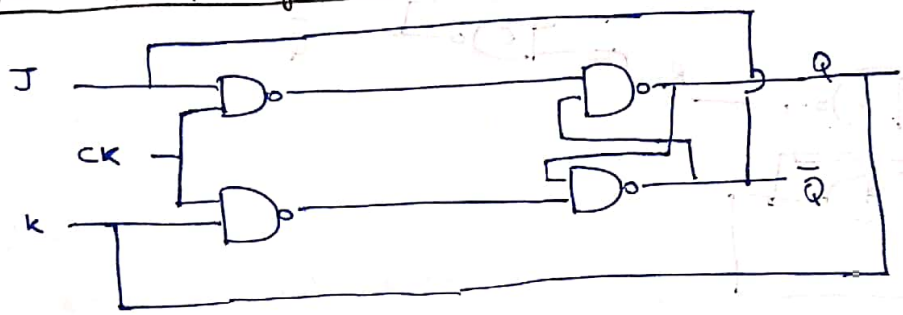
Timing diagram



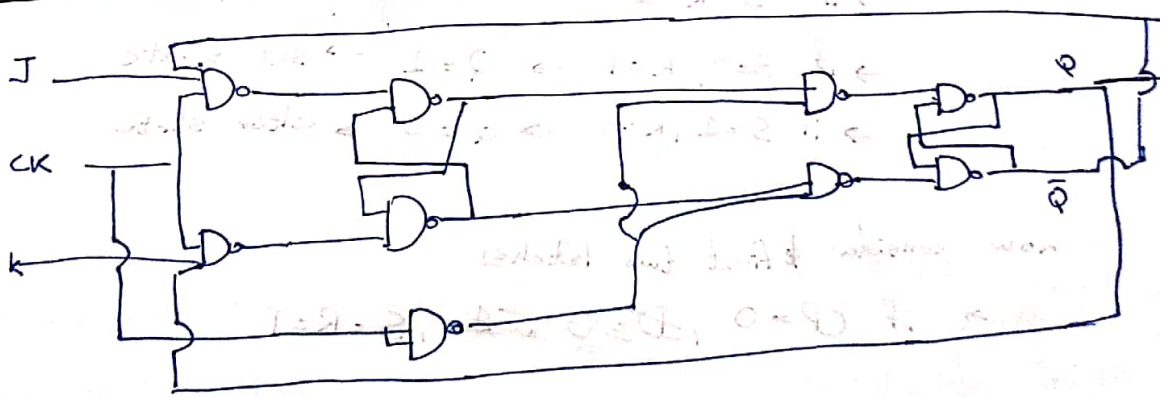
initial : y = Q = 0

→ This is a negative edge-triggered flip flop

J-K Flip Flop using only NAND

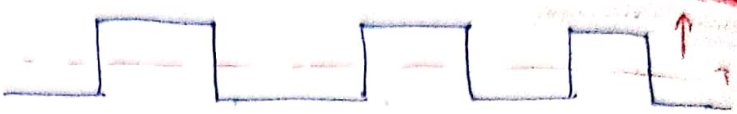


J-K Master Slave using NAND

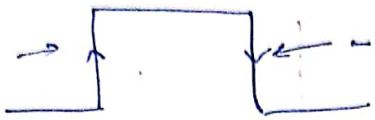


Edge - Triggered D Flip Flop

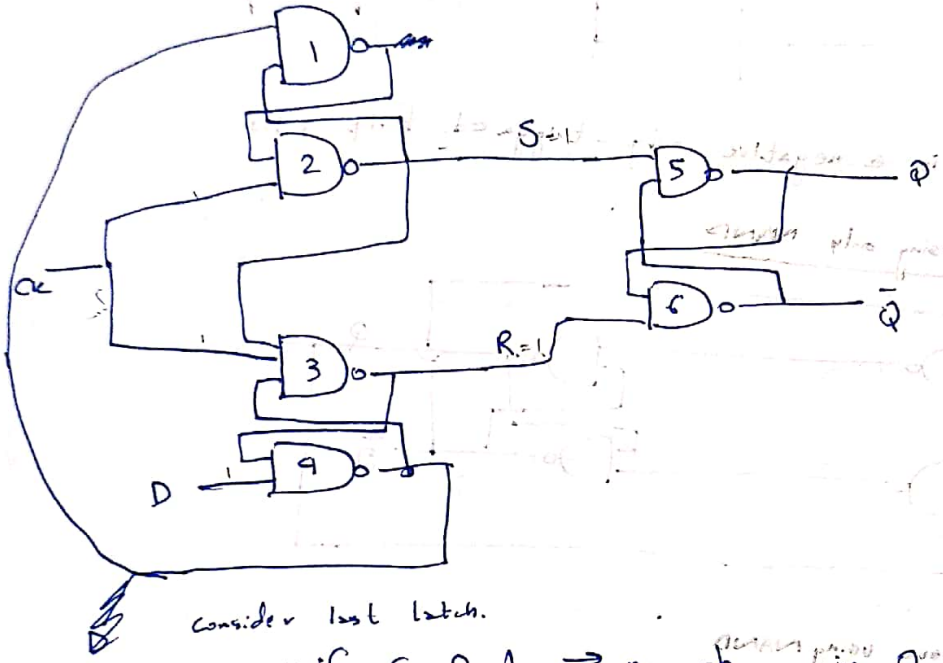
CP :



+ve edge trigger →



← -ve edge trigger.



consider last latch.

→ if S = R = 1 → no change in Q

→ if S=0, R=1 → Q = 1 → set state

→ if S=1, R=0 → Q = 0 → clear state

now consider first two latches

→ if CP = 0, ~~D = 0 or 1~~, S = R = 1

Setup time: initial time, where D needs to be constant (before application of clock)

setup time = propagation delay of signal from gate 4 to gate 1

→ assume D doesn't change during setup time, ~~D=D, CP=1~~

→ D=0, CP=1, → S=1, R=0 → Q=0 → set

Hold time = time where D should be held after application of clock pulse.

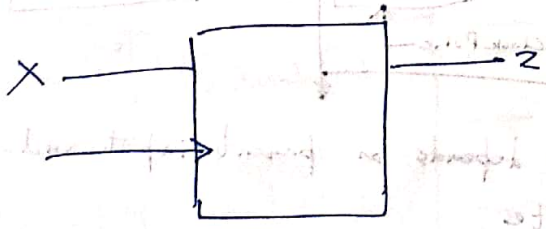
= Propagation delay of Gate 3 as it ensures R change after change of S.

→ CP=1, D=1 → S=0, R=1 → Q=1 → clear.

→ assume initial S=R=1 as it makes no change, and CP=0 between two CP=1

Design of Sequential Circuit

Q) Desing a circuit to detect 3 or more consecutive 1's in a serial bit stream.



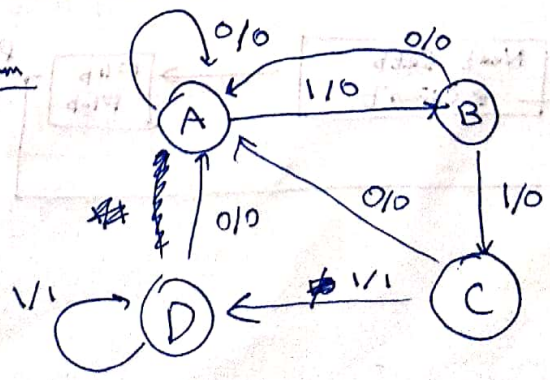
X = 0 1 1 0 1 1 1 1 0 1 0 1 1 1

Z = 0 0 0 0 0 0 1 1 0 0 0 0 1

State Transition Diagram

or

State ~~Transition~~ Diagram



State Table

present state	next state		output	
	X=0	1	X=0	1
A	A	B	0	0
B	A	C	0	0
C	A	D	0	1
D	A	D	0	1

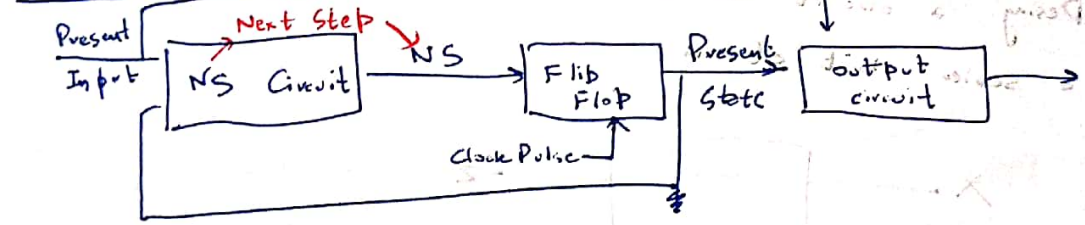
Mealy and Moore FSM

$$\delta : \Sigma \times S$$

Mealy $\Rightarrow$ $\omega : S \times \Sigma \rightarrow \Gamma$

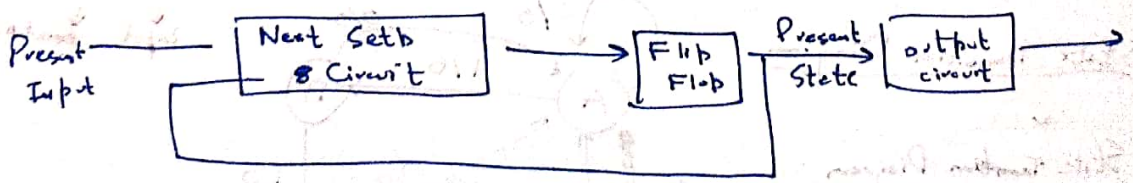
Moore $\Rightarrow$ $\omega : S \rightarrow \Gamma$

Mealy



→ output depends on present input and present state

Moore



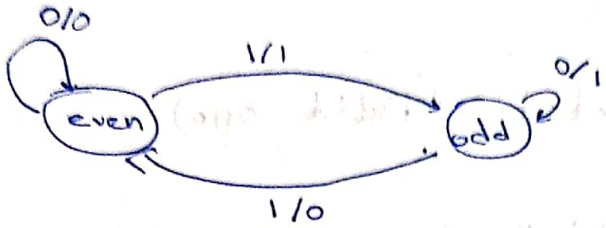
Eg 1 Design a ~~be~~ serial parity generator.

Input: 0 0 1 1 1 0 1 0 0 1

output: 0 0 1 0 1 1 0 0 0 1

even
~~odd (or) 1~~
(1 → odd
0 → even)

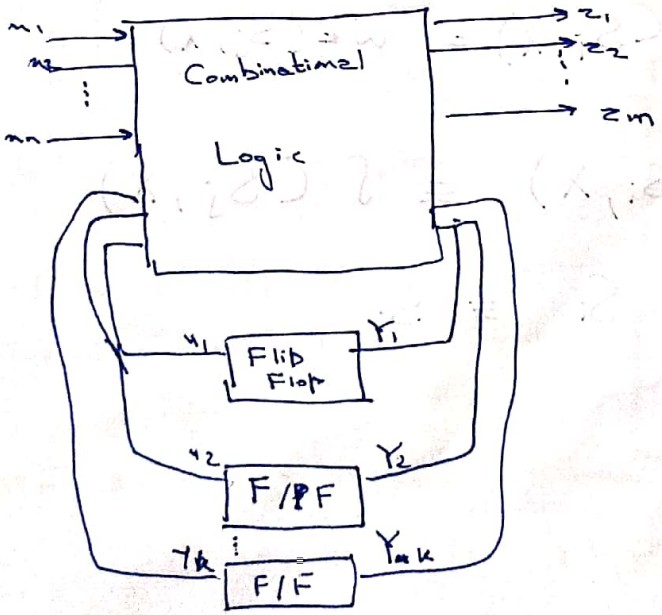
state diagram



state table

present state	next step		output	
	X=0	X=1	X=0	X=1
Even	Even	Odd	0	1
odd	odd	Even	1	0

Representation of a Sequential logic



$\Sigma = 2^n$, $\Gamma = 2^m$

$PS = y_1, y_2, \ldots y_k$

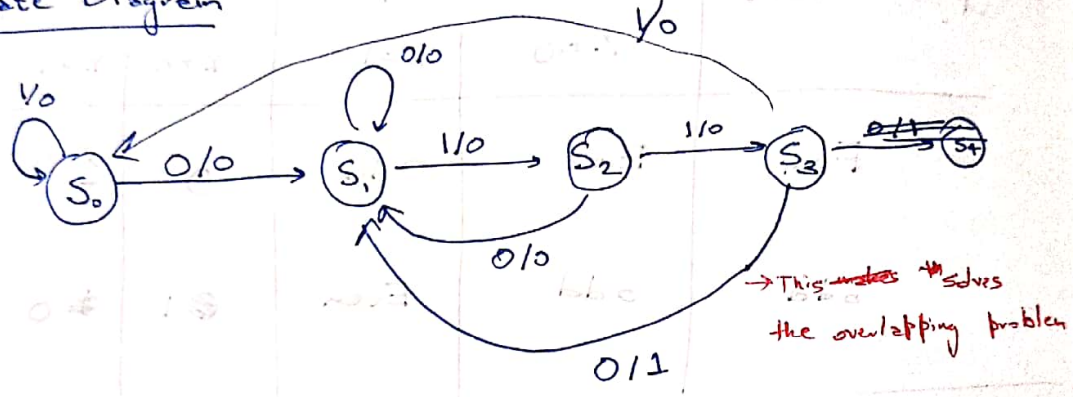
$NS = Y_1, Y_2, \ldots Y_k$

Eg 2 Sequence Detector (Detect 0110)

→ X = 0 0 1 1 0 0 1 0 0 1 1 0 1 1 0 0

Z = 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0

State Diagram



→ This ~~makes~~ solves the overlapping problem

State Table Minimisation

$$w(S_i, X) = w(S_j, X)$$

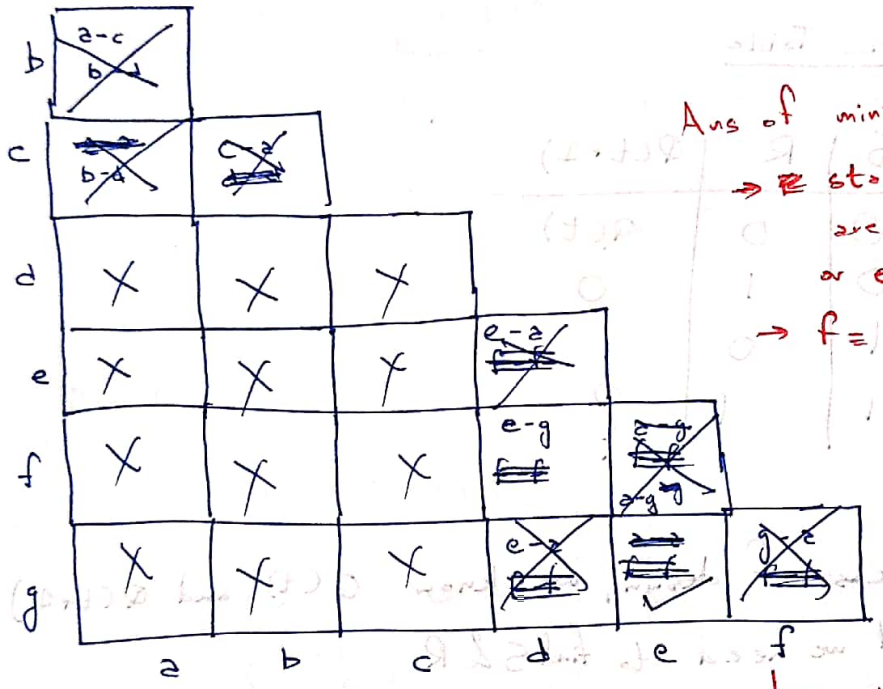
$$\delta(S_i, X) \equiv \delta(S_j, X)$$

$$S_i \equiv S_j$$

Implication Table

Eg 1

PS	I/P = 0 NS	O/P	I/P = 1 NS	O/P
a	a	0	b	0
b	c	0	d	0
c	a	0	d	0
d	e	0	f	1
e	a	0	f	1
f	d	0	f	1
g	a	0	f	1



Ans of minimization.

→ state e and g are same or e ≡ g

→ f ≡ d.

final Implicat Table.

PS	IP=0 NS	OP	IP=1 NS	OP
a	a	0	b	0
b	e	0	d	0
c	a	0	d	0
d	e	0	f	1
e	a	0	f	1
~~f~~	~~e~~	~~0~~	~~f~~	~~1~~

I) (making table)

→ don't write same implication

Eg a-a, b-b, etc.

→ If any o/p output diffvent, put cross

→ Implications (a-c) have no order

Eg a-c = c-a

→ If all implication on square is same.

Eg a-a & e-e, put ✓, they are equivalent

II) (checking Table)

→ check every not X box with it's implied box, if implied box is X, the box is also X

→ there are two square
  → d-f
  → e-g
which are not ~~related~~ (i.e they have no difference)

∴ $d \equiv f$ and $e \equiv g$

and hence state diagram can be adjusted to minimize states.

Excitation Table of Flip-Flop

Characteristic Table

S	R	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	?

→ in case of design, we know Q(t) and Q(t+1) and we need to find S & R

SR excitation →

Q(t)	Q(t+1)	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

JK excitation

Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

D flip flop

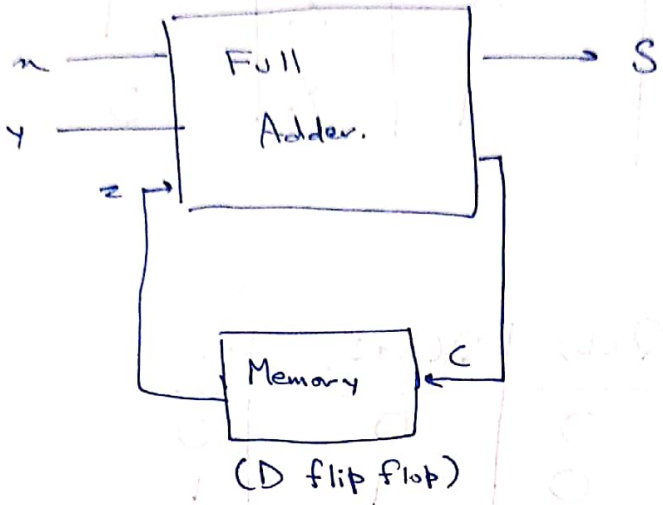
Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

T flip flop

Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

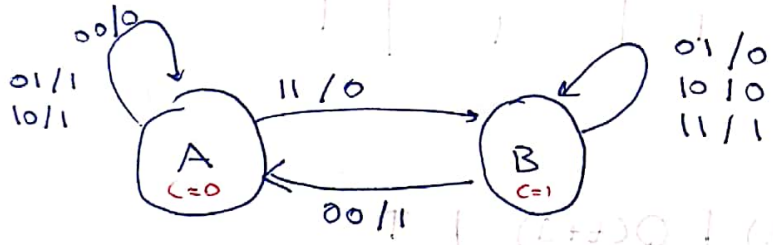
Eg: Serial Adder

Eg A = 1010
B = 0101 → 1111

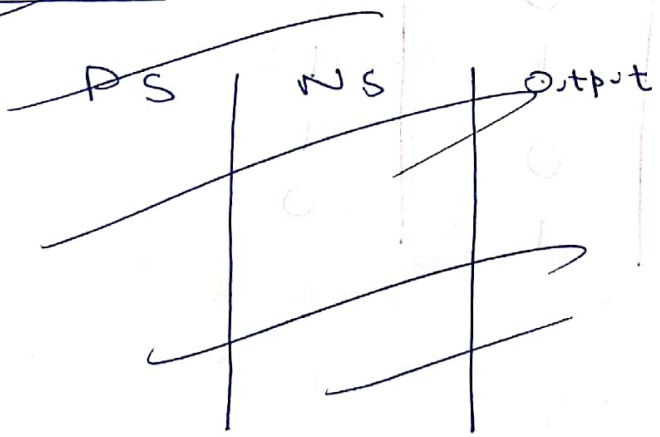


(D flip flop)

State diagram



State Table



PS	NS, Output			
	00	01	10	11
A	A,0	B,1	A,1	B,0
B	A,1	B,0	B,0	B,1

State Assignment Table.

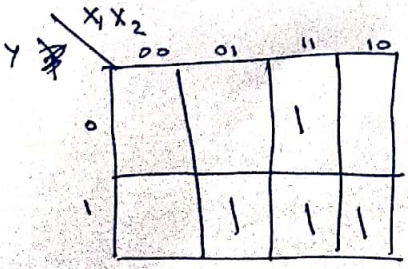
A = 0 B = 1

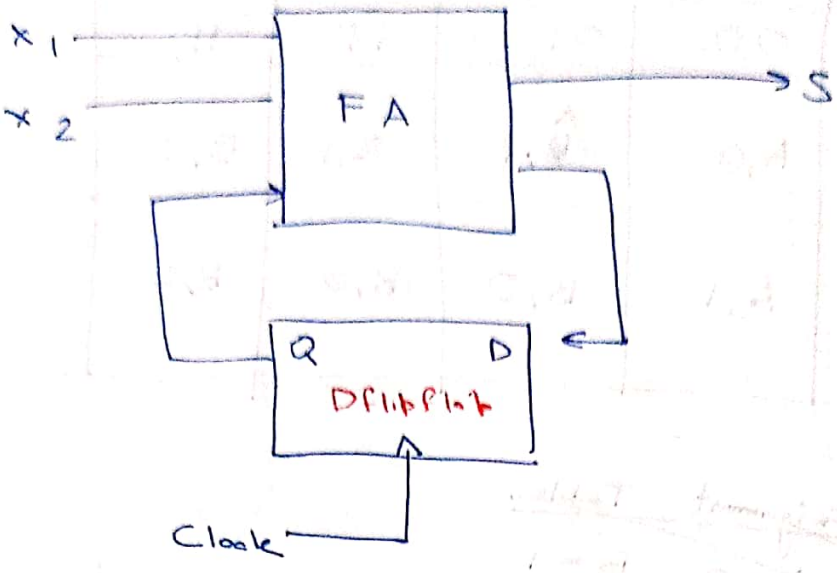
PS NS ~~Output~~

$Y_{(PS)}$	X_1	X_2	Y(NS)	D	Output (S)
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	1	1

$S = \Sigma(1,2,4,7) = Y \oplus X_1 \oplus X_2$

$D = \Sigma(3,5,6,7) = X_1X_2 + YX_2 + YX_1$



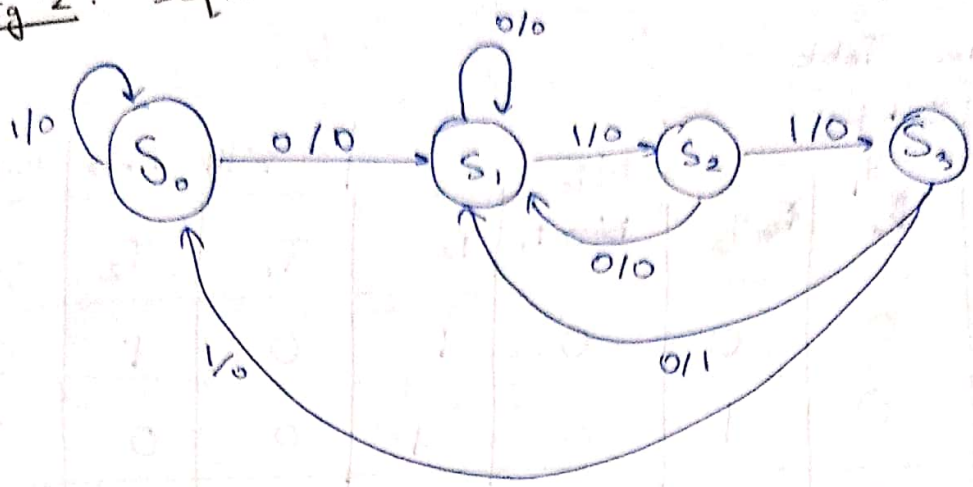


If we used JK Flip Flop instead of D, the excitation table would have been this:

y	X_1	X_2	Y(NS)	J	K	S
0	0	0	0	0	X	0
0	0	1	0	0	X	1
0	1	0	0	0	X	1
0	1	1	1	1	X	0
1	0	0	0	X	1	1
1	0	1	1	X	0	0
1	1	0	1	X	0	0
1	1	1	1	X	0	1

X → anything

Eg 2: Sequence Detector (0110)



State Table

PS	NS, Z (output)	
	n=0	n=1
S_0	$S_1, 0$	$S_0, 0$
S_1	$S_1, 0$	$S_2, 0$
S_2	$S_1, 0$	$S_3, 0$
S_3	$S_1, 1$	$S_0, 0$

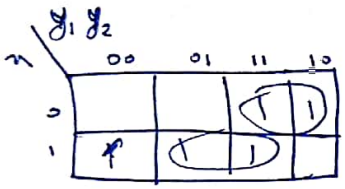
State Assignment: $S_0 = 00$; $S_1 = 01$; $S_2 = 10$; $S_3 = 11$

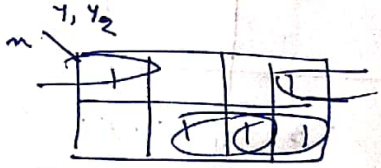
using D Flip Flop

~~Transition Tab~~

Excitation Table

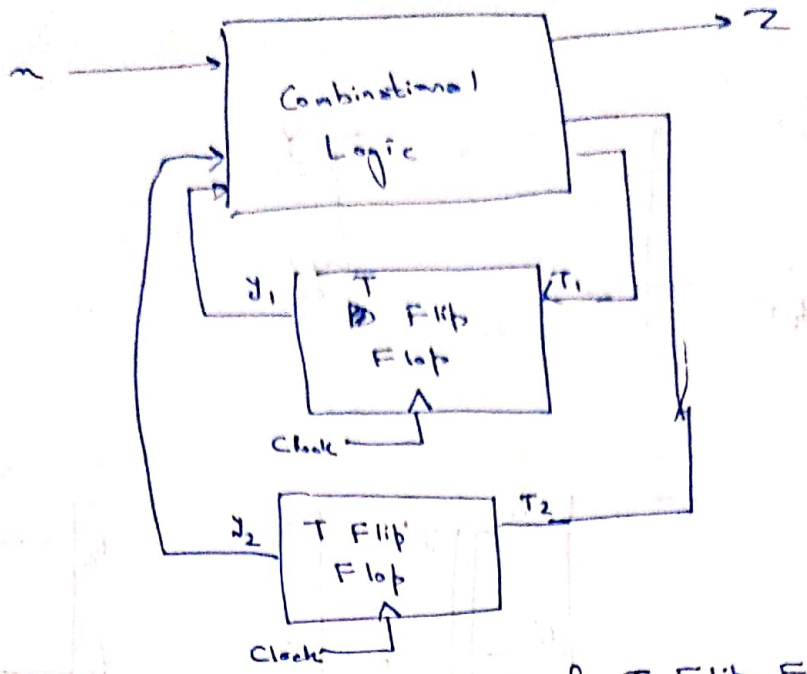
	PS		NS		Flip Flops		output
x	y_1	y_2	Y_1	Y_2	T_1	T_2	Z
0	0	0	0	1	0	1	0
0	0	1	0	1	0	0	0
0	1	0	0	1	1	1	0
0	1	1	0	1	1	0	1
1	0	0	0	0	0	0	0
1	0	1	1	0	1	1	0
1	1	0	1	1	0	1	0
1	1	1	0	0	1	1	0





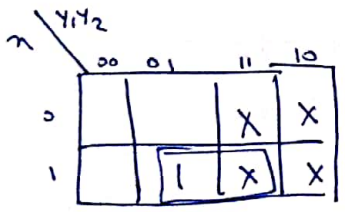
$T_1 = x'y_1 + xy_2$

$T_2 = x'y_2' + xy_2 + xy_1$

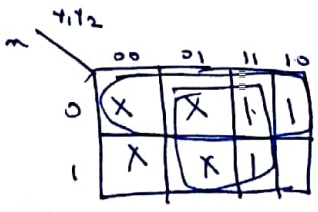


if we use JK Flip Flop instead of T Flip Flop

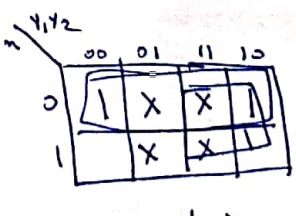
x	Y_1	Y_2	Y_1	Y_2	J_1	K_1	J_2	K_2	z
0	0	0	0	1	0	X	1	X	0
0	0	1	0	1	0	X	X	0	0
0	1	0	0	1	X	1	1	X	0
0	1	1	0	1	X	1	X	0	1
1	0	0	0	0	0	X	0	X	0
1	0	1	1	0	1	X	X	1	0
1	1	0	1	1	X	0	1	X	0
1	1	1	0	0	X	1	X	1	0



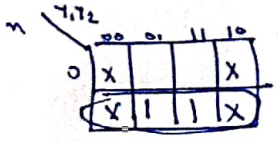
$J_1 = x Y_2$



$K_2 = x' + Y_2$

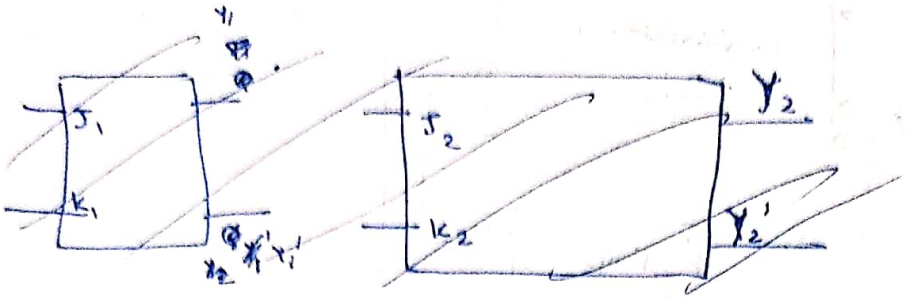


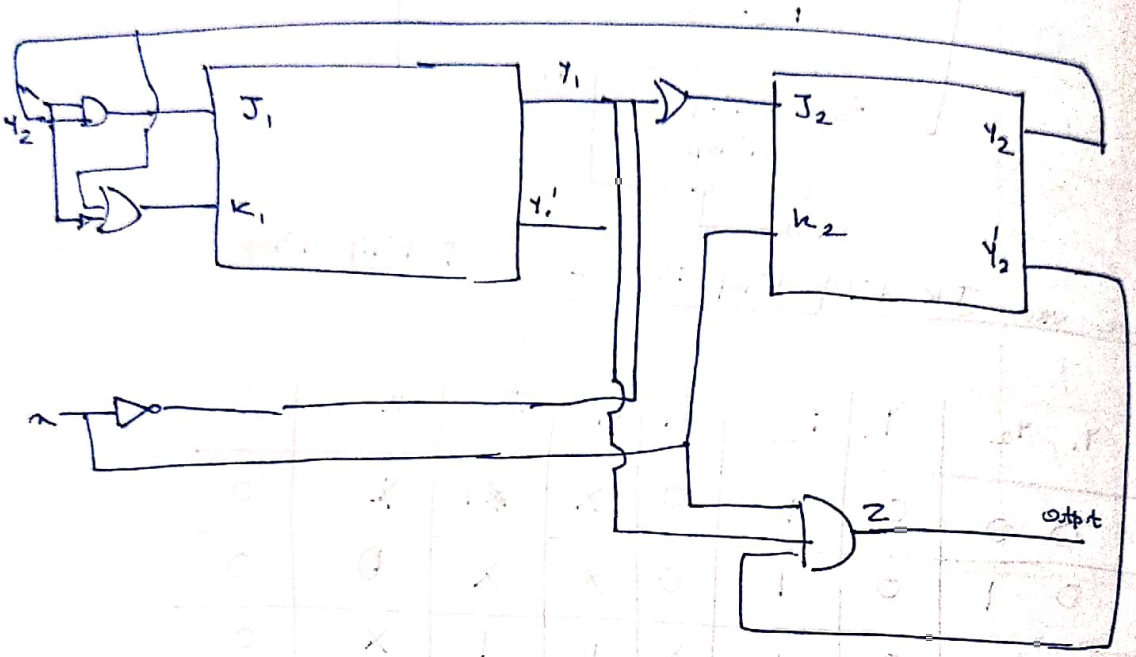
$J_2 = x' + Y_1$



$k_2 = X$

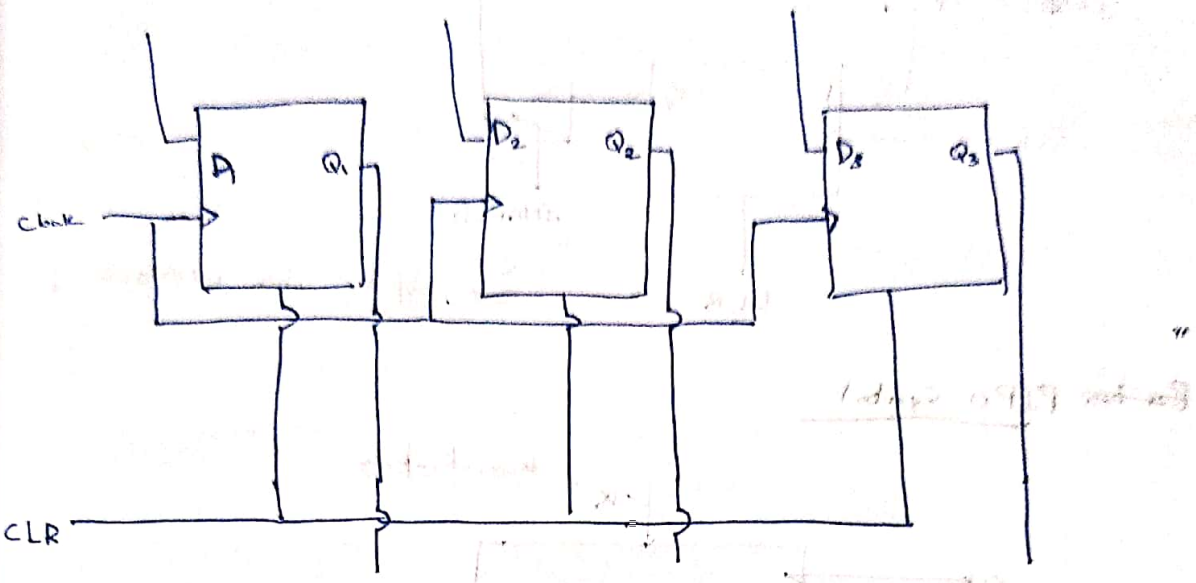
Circuit





Register Design

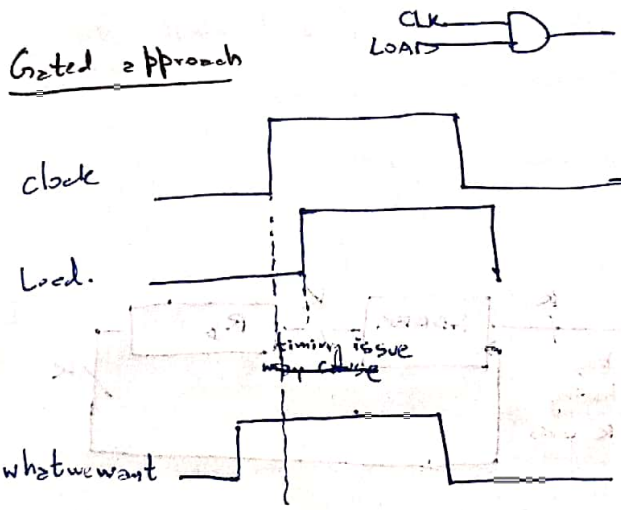
3 bit PIPO Register (using D flip flop)



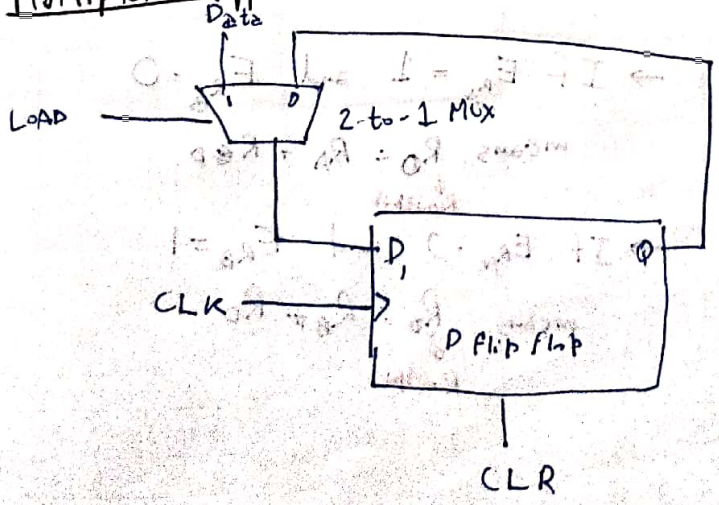
→ when CLR = 1, flip flops resets.

Addition of Load in PIPO

Gated approach

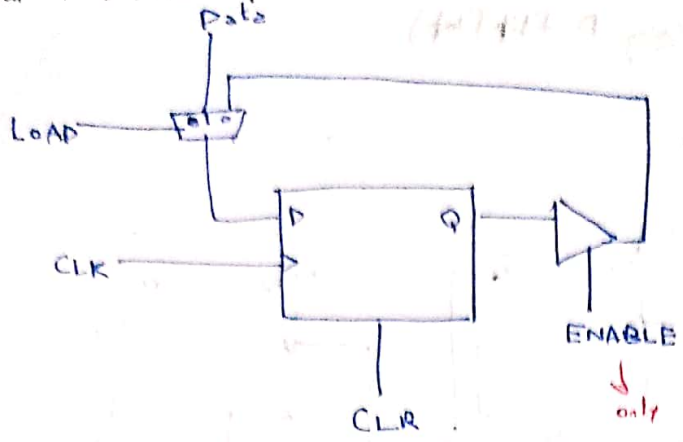


Multiplexer approach

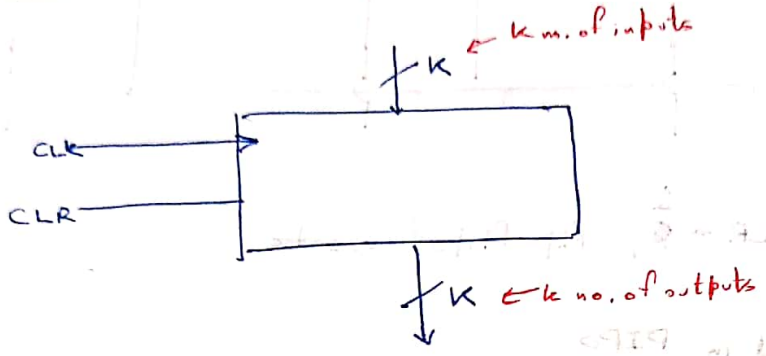


Load=0 → no change
Load=1 → Data get fed.

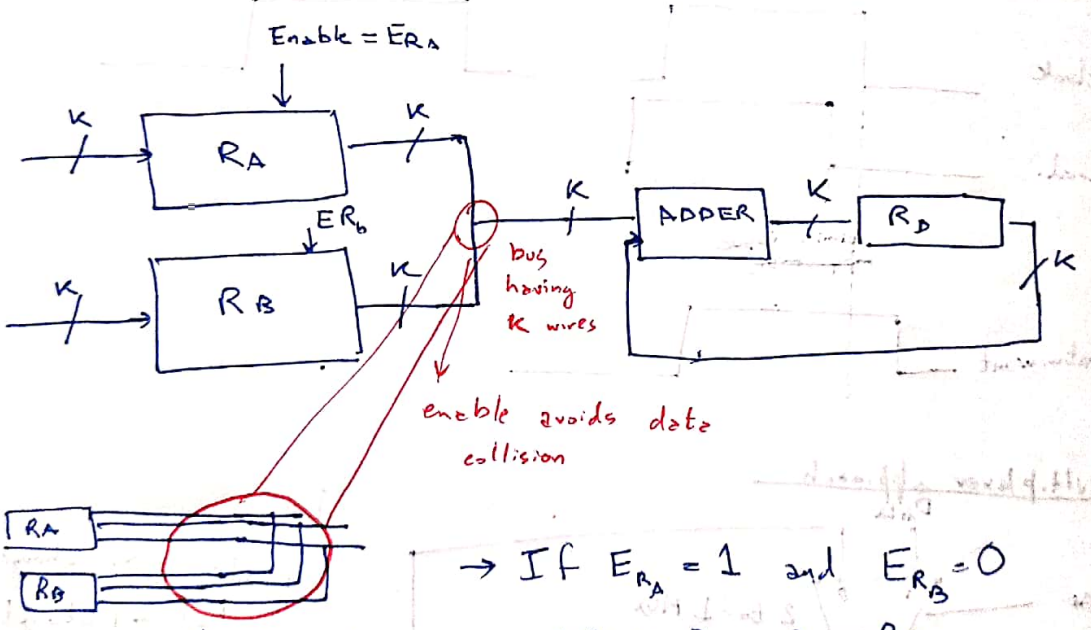
Addition of enable input



~~Bus base~~ PIPO Symbol



Bus based Architecture



→ If $E_{R_A} = 1$ and $E_{R_B} = 0$

means $R_D = R_A + R_D$

(R_D ↓ Routput)

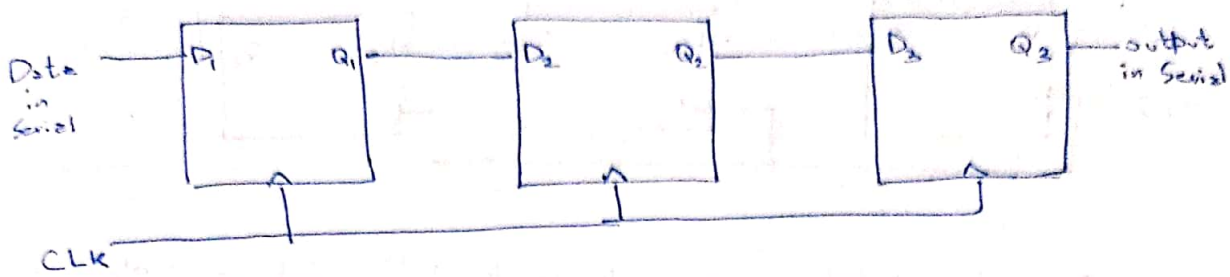
→ If $E_{R_A} = 0$ and $E_{R_B} = 1$

means $R_D = R_B + R_D$

(R_D ↓ Routput)

→ Enable acts as switch.

Shift ~~Desire~~ Register (SISO) (Right shift Register)



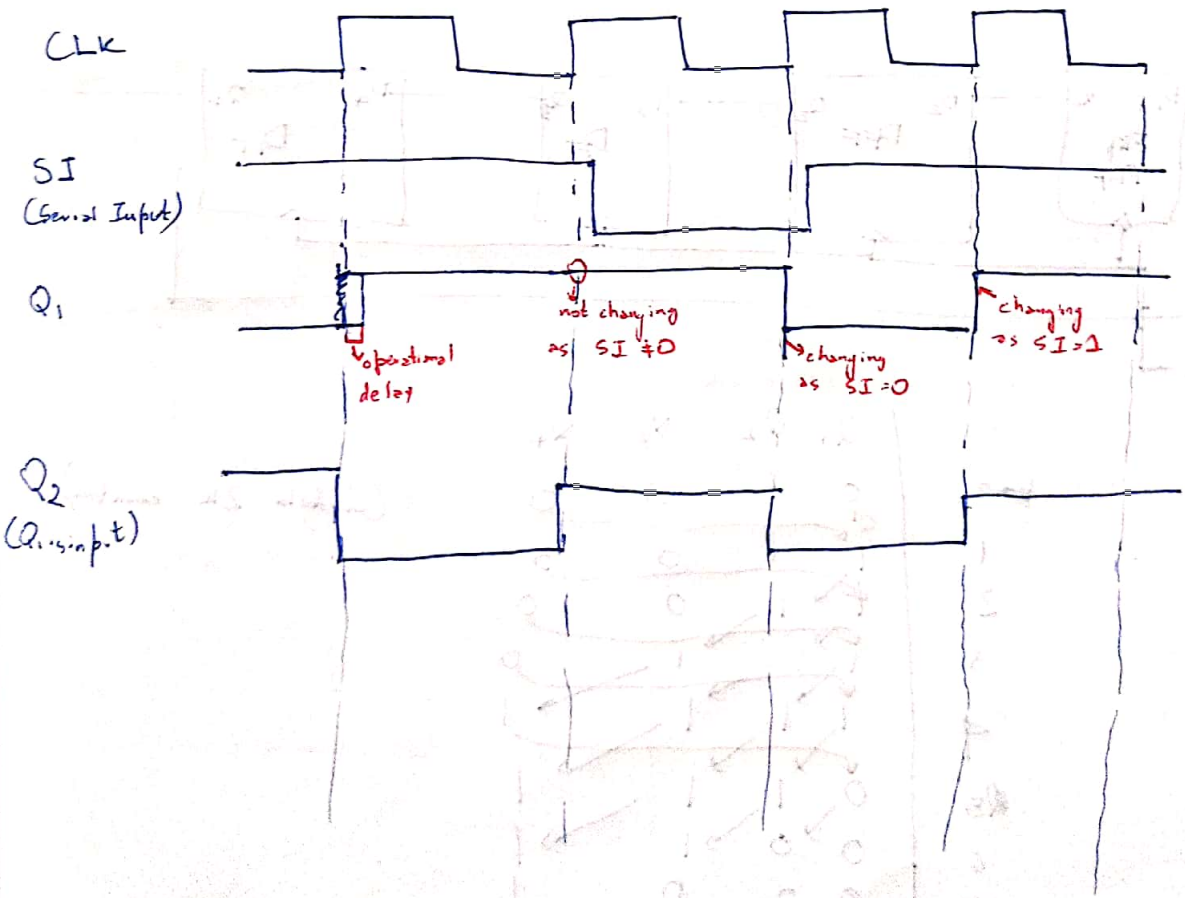
Eg initially

Data = 01011

	Q_1	Q_2	Q_3
first clk	0	0	0
second clk	1	0	0
third clk	1	1	0
4th clk	0	1	1

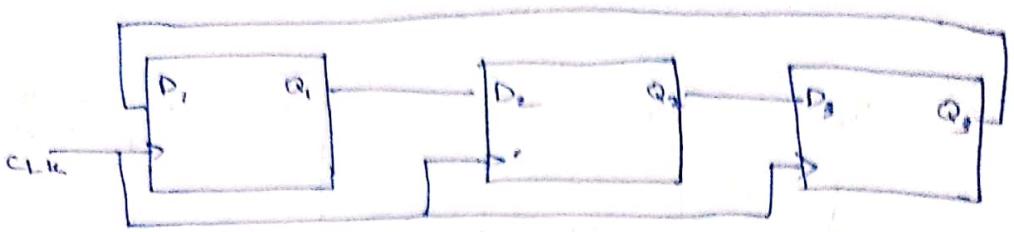
→ adds delay by 1

Sample Timing Diagram (initial $Q_1Q_2Q_3Q_4 = 0101$)
(+ve edge trigger Flip Flop)



Ring Counter

→ SISO shift



→ generally initialized with single 1 and all FF's 0

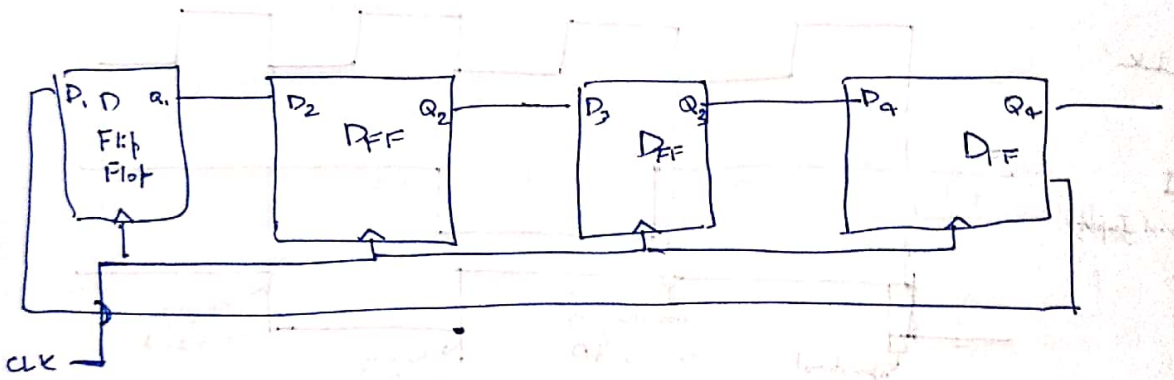
Eg:

	Q_1	Q_2	Q_3
initial	1	0	0
1st CLK	0	1	0
2nd CLK	0	0	1
3rd CLK	1	0	0

and repeat

Jhonson Counter

Initial condition : $0000 = Q_1 Q_2 Q_3 Q_4$



	Q_1	Q_2	Q_3	Q_4
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
8	0	0	0	0

Q'

(modulo 2k counter)

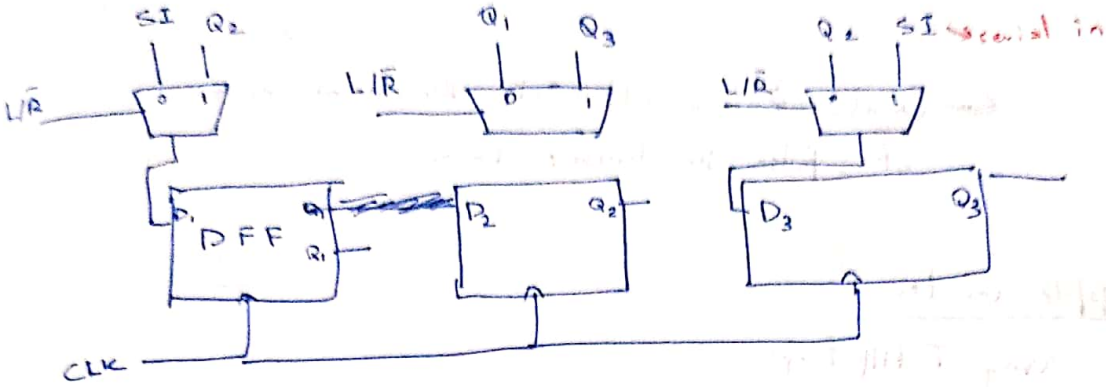
Bidirectional Shift Register

Control Signal $L/\bar{R}$

left / right

$L/\bar{R} = 1 \rightarrow$ Left shift register

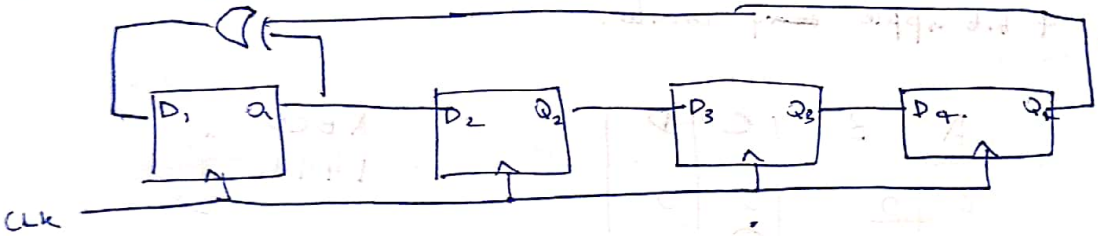
$L/\bar{R} = 0 \rightarrow$ right shift register



Universal Shift Register

S_1	S_0	output
0	0	no remains same.
0	1	right sh. f
1	0	left shift
1	1	parallel loading.

~~Timer~~ & LFSR



Initial condition $1000 = Q_4\,Q_3\,Q_2\,Q_1$

produces $2^n - 1$ combinations.

Counter Design

Refill ~~Counter~~ ~~Ripple Counter~~

Synchronous : CP applied to All FF simulta

Async : ~~same~~ CP applied to only one FF

← Counter ⊕ gives output of the no. of occurrence of a pulse in binary form.

Ripple Counter

using T filp flops

Eg: it will go from

1111 → 0111 → 0011 → 0001 → 0000

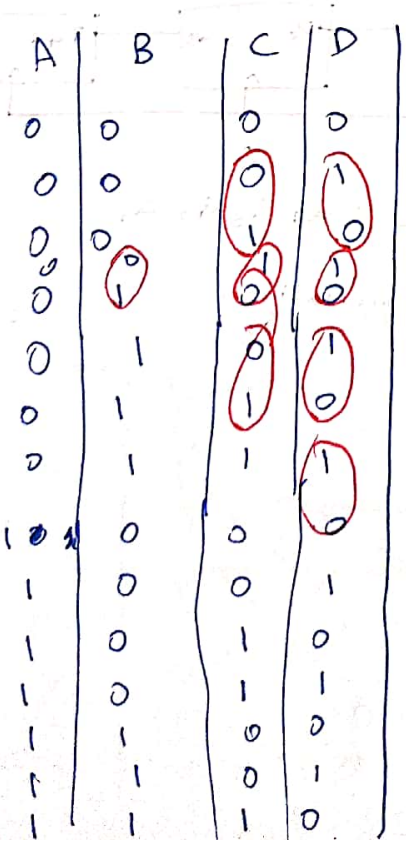
↓

intermidiate steps.

→ similar ~~for~~ propagation method like Ripple Carry Adder.

→ Asynchrous; not all fip flop has same clock.

Eg: 4 bit ripple ~~carry~~ counter.



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

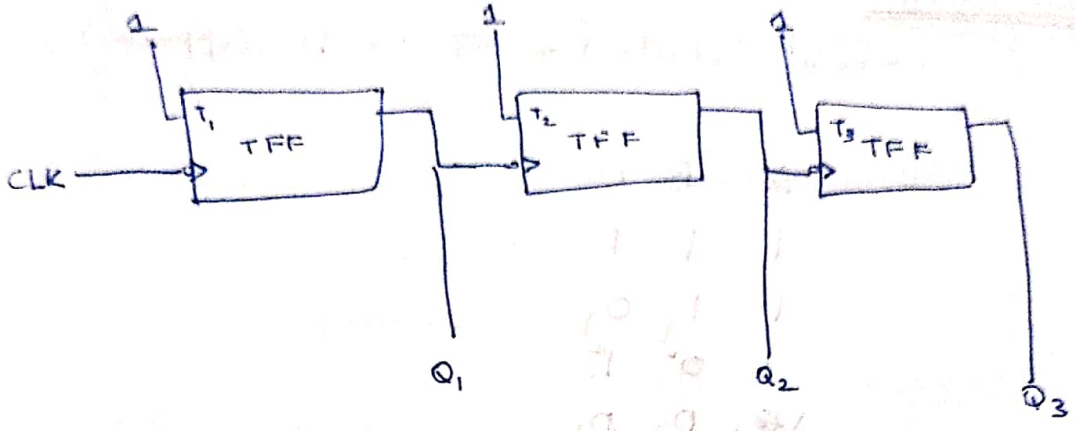
A B C D
1 1 1 1

Obs

→ if there is change of $1 \xrightarrow{\text{to}} 0$ in lower bit, there will be change in higher bits.

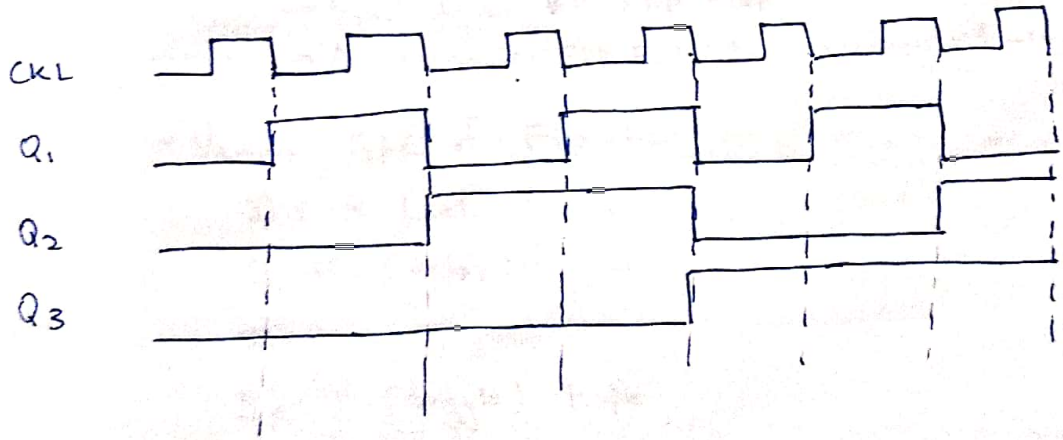
Using this observation: (we use -ve edge trigger Flip Flop)

Eg: 3 bit



initial	Q_1	Q_2	Q_3
	0	0	0
	0	0	1
	0	1	0
	0	1	1
	1	0	0
	1	0	1
	1	1	0
	1	1	1

Time diagram



Same thing by using positive edge trigger flipflop:

→ we invert clock pulse

→ by connecting ~~outputs~~ Q_1', Q_2', Q_3' to next Flip Flop instead of Q_1, Q_2, Q_3.

down Counter

→ count starts from 7 to 0 (opposite)

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

Observation

→ change from 0 to 1 in LSB → change in it's higher order

→ can be done by +ve edge trigger T Flip Flop

Latch And Flip Flop

Definition of Sequential Circuits

→ The output of a sequential circuit depends not on the present input, but also the past history of the circuit. The circuit memorizes it's present state.

→ Latches and Flip Flops are the basic building block of storage device.

→ There are two types of sequential circuits:

i) Synchronous

ii) Asynchronous

→ Basic idea behind storing bit

Design of Latches

→ A Latch is a temporary storage device that has two stable states, 0 and 1

→ A Flip-Flop is a special kind of latch where a clock signal triggers the change in stored value.

→ Synchronous: All Flip Flop controlled by same clock

→ Asynchronous: Flip Flops have different timings.

→ Various Type of Flip-Flop

I) S-R (Set - Reset)

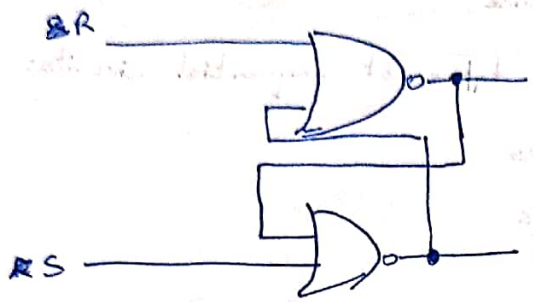
II) D (delay)

III) J-K

IV) T (toggle) type.

S-R Latch

- It has a pair of cross-coupled NOR or NAND gates.
- → has two inputs (S and R) and two outputs (Q and $\bar{Q}$)
- → The output can be set to 0 or 1 by applying suitable ~~va~~ S and R values.



(initial Q=0, $\bar{Q}$=0)
(cause no power)

S	R	Q	$\bar{Q}$
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	0	0 ← ☹

→ output of NOR is 0 if any input 1, and is 1 if all input 0

(Truth Table is sequential, value of previous Q, affects present Q)

→ Latch has two useful states.

i) Q=1, $\bar{Q}$=0 ; set state (S=1, R=0)

ii) Q=0, $\bar{Q}$=1 ; ~~reset state~~ clear state (S=0, R=1)

→ When S=1, R=1, we get Q=1, $\bar{Q}$=1, which violates the definition of latch and is an invalid state

→ When S=0, R=0, output is just copied.

→ consdier, first we apply S=1, R=1 to get Q=$\bar{Q}$=0, then we apply S=0, R=0, we get Q=$\bar{Q}$=1. If the speed of two gates be same, output will oscillate between Q=$\bar{Q}$=0 and Q=$\bar{Q}$=1.

→ But in reality, two gates can never have same speed. In that case, we cannot predict the output. This phenomena is known as race condition.

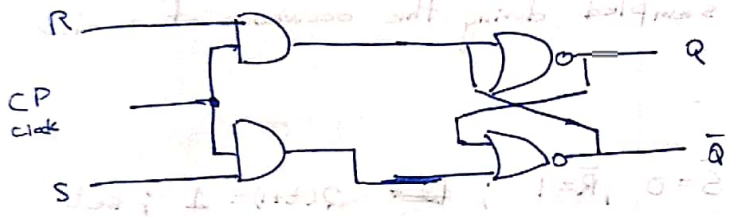
Race condition

→ A scenario where the final output depends on the relative speed of the gate.

→ If we apply S=1, R=1, ~~we get~~ then we apply S=0, R=0 output will either be $Q=0, \bar{Q}=1$ or $Q=1, \bar{Q}=1$, depending about relative speed of gate.

S-R FlipFlop

The S-R latch can be modified by providing an additional control input that determines when the state of the circuit is to be changed.



→ when clock pulse goes 1, info. from S and R input are allowed to reach to the latch.

→ Characteristic Table of S-R Flip Flop

Q(t)	S	R	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	undefined.
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	undefined.

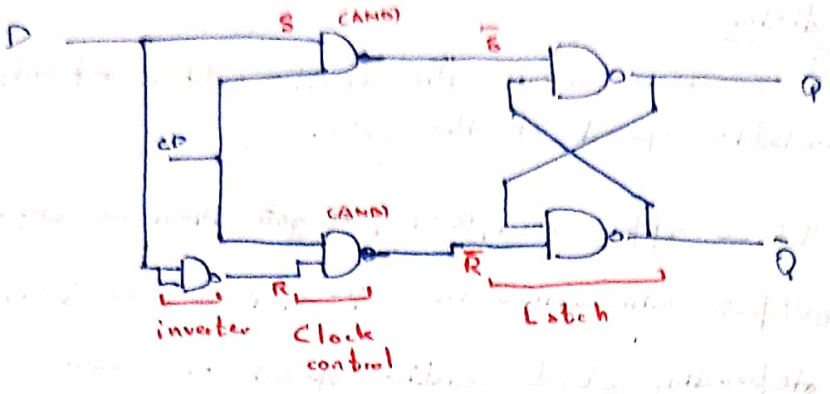
(only happens when CP=1)

from truth table: $Q(t+1) = S + \bar{R}\,Q(t)$

→ Set : S = 1, R = 0 , CP = 1 , Q(t+1) = 1

→ clear : S = 0, R = 1, CP = 1 ; Q(t+1) = 0

D Flip Flop



(will only work when CP = 1)

→ As long as clock pulse input is 0, $\bar{S} = \bar{R} = 0$. This makes sure two input of SR Latch remains ~~same~~ ~~intr~~ 0 initially

→ The D input is sampled during the occurence of a clock pulse.

→ D = 1, CP = 1 ; $\bar{S} = 0$, $\bar{R} = 1$; ~~D=1~~ Q(t+1) = 1 ; set

→ D = 0, CP = 1 ; Q(t+1) = 0 ; clear state.

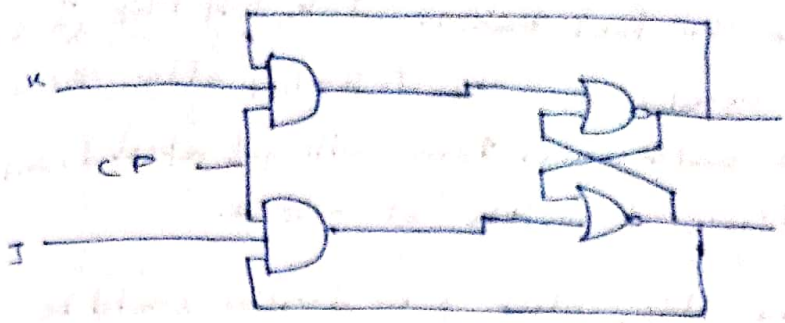
→ characteristic Table:

Q(t)	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

Q(t+1) = D

(multiple D Flip Flops can be used to delay the signal)((resistor)

J-K Flip Flop

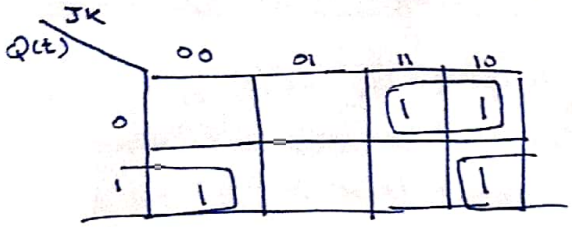


→ This Flip Flop solves S=1, R=1 problem, in case of S-R Flip Flop

→ when J=1, K=1, Q complements. ; all others same as S,R ⁂

→ Characteristic Table (initial state, everything 0)

Q(t)	J	K	Q(t+1)	
0	0	0	0	(copies Q(t))
0	0	1	0	
0	1	0	1	
0	1	1	1	(inverts Q(t))
1	0	0	1	(copy Q(t))
1	0	1	0	
1	1	0	1	
1	1	1	0	(invert Q(t))



$Q(t+1) = \bar{Q}(t)J + Q\bar{K}$

Race in Condition

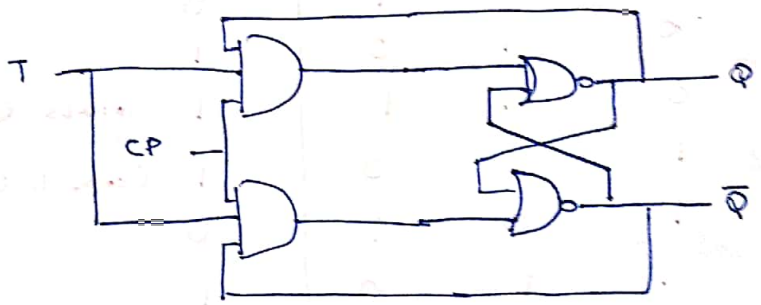
→ Due to the feed back in J-K Flip Flop if the Clock Signal remains 1 (while J=K=1), after the output will be complemented & once will get repeated. and a continous transition of outputs.

→ To avoid this, clock pulse duration should be less than propagation delay of J-K Flip Flop

→ another solution is to use master-slave or edge-triggered flip flop.

T Flip Flop

→ The T flip-flop is a single-input version of J-K Flip Flop. The T input is obtained from a JK type, if both inputs are tied together.



T = 1 , invert

T = 0 , copy

Triggering of Flip-Flop

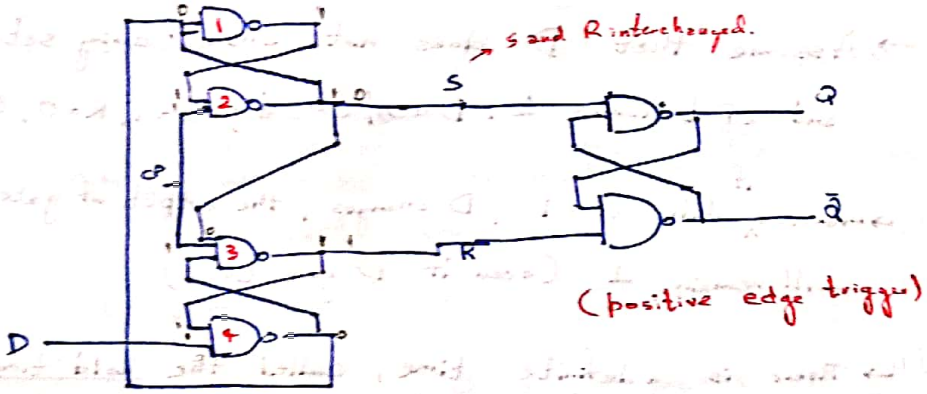
→ The state of a Flip Flop is changed by a momentary change in the input signal. This momentary change is called a trigger.

→ Clocked Flip Flops are triggered by pulses. A pulse stats from an initial value 0, goes to 1, and after a short time, returns to 0

→ If a Flip Flop changes it's state at rising edge of the clock pulse, it is called a positive edge triggered flip flop.

→ If Flip Flop changes it's state at falling edge of clock pulse, it is negative edge trigger Flip Flop.

Edge Trigger D Flip Flop



→ In this type of Flip Flop, the output transition occur at a specific level of the clock pulse. When the pulse input level exceeds this threshold level, the inputs are locked and Flip Flop is unresponsive to further changes in input, until clock returns to 0 and another pulse occurs.

→ when

i) S=0, R=0 → copy
ii) S=1, R=0 → Q=1; set
iii) S=0, R=1 → Q=0; clear

(S and R gets determined by D)

CP	D	S	R	
0	0	1	1	
0	1	1	1	
1	0	1	0	(initial SR = 11)
1	1	0	1	(initial SR = 11)

(change only when CP = 1, otherwise output SR = 11)

(when CP = 1, $S = \bar{D}$, $R = D$)

→ There is a definite time, called setup time, in which D must be kept constant, prior to the application of pulse

setup time = propagation delay for signal from gate 4 to gate 1

→ Assume that D does not change during setup time and CP becomes 1. D = 0, CP = 1; S = 1, R = 0, Q = 0

→ ~~no~~ now, if while CP = 1, D changes, the output of gate 4 ~~changes~~ will remain 1 (even if D goes to 1)

→ There is a definite time, called the <u>hold time</u>, that the D should not change after application of pulse.

Hold time = propagation delay of Gate 3 output to gate 4.

→ it must ensure that R becomes 0 in order to maintain output of gate 4 at 1, regardless of D.

Design Of Synchronous Sequential Circuits

→ In case of sequential circuits, the outputs depends not only on the present inputs but also on the internal states of the circuit.

→ The internal states also changes with time.

→ The number of states of a circuit is finite, and hence a sequential circuit is also called a Finite State Machine (FSM)

→ FSM can be represented in either state table or state diagram.

Mealy and Moore FSM

→ Formal Definition of FSM. A deterministic FSM can be mathematically defined as a 6 Tuples $(\Sigma, \Gamma, S, s_0, \delta, \omega)$

where → Σ ⇒ set of input combination

→ Γ ⇒ set of output combination

→ S ⇒ finite set of states.

→ $s_0 \subset S$ ⇒ initial state

→ δ ⇒ state transition function (it gives the next state)

→ ω ⇒ output function.

→ $\delta: S \times \Sigma$ → present state and present inputs determine the next state (NS)

Mealy Machine

$\omega: S \times \Sigma \rightarrow \Gamma$

→ ~~present state~~

→ output depends on Present state and inputs.

Moore Machine

$\omega: S \rightarrow \Gamma$

→ output depends on present state.

Design of Synchronous Sequential Circuit

i) From a description of the problem, obtain state table and state diagram.

ii) Check whether the table contains any redundant states, if so, remove them

iii) Select a state assignment and determine Type of Flip Flop.

iv) Derive transition and output tables

v) Derive the excitation table and obtain excitation and output function.

vi) Minimize the function and the circuit diagram.

Construction of State diagram / Tables from specification

→ A state diagram specifies the different states of a FSM, the condition under which state change occur, and the corresponding output.

→ States are denoted as circles, labelled with unique symbols

→ Transitions are represented as directed arrows between pairs of states.

→ Each transition is labelled by α/β, where α denotes input combination and β denotes output combination.

→ A state table is an alternative way of representating state diagram

→ For each value of present state, it specifies next state and output for every input combination.

State Table minimization

→ **Basic Concepts:**

→ Identify equivalent states and merge them into single state.

→ Can lead to more compact representation.

→ **State Equivalence:**

→ Two states S_i and S_j are said to be equivalent iff for every single input sequence X, the outputs are same and next states are equivalent

$$w(S_i, X) = w(S_j, X)$$
$$\delta(S_i, X) = \delta(S_j, X)$$

Steps for minimization using Implication Table

1) Construct an implication table that contains a square for every pair of states

2) → Compare each pair of rows in the state table, if the output associated with state S_i and S_j are different (for every ~~input~~ any input), place 'X' in square S_i, S_j to ~~Sin~~ indicate S_i and S_j are not equivalent.

→ If outputs are same, place the implied pair in the square S_i, S_j (if the next state of S_i and S_j are ~~n~~ m and n respectively for some input x, then m-n is an implied pair for $S_i - S_j$)

3) Go through table square-by square, if square $S_i - S_j$ contains implied pair m-n and ~~so~~ and square m-n contains 'X', then $S_i - S_j$ are ~~are~~ not equivalent, place X in that square

4) repeat step 3 until ~~pass~~ no more 'x' can be added.

5) For each square $S_i - S_j$ that doesn't contains 'X', the $S_i - S_j$ states are equivalent

Mod-M counter

Mode: 000, 001, 010, 011, 100, 101

→ if we use 4 Flip flop, we get mode 16 counter
→ so for mod-8, we take 3 Flip Flops

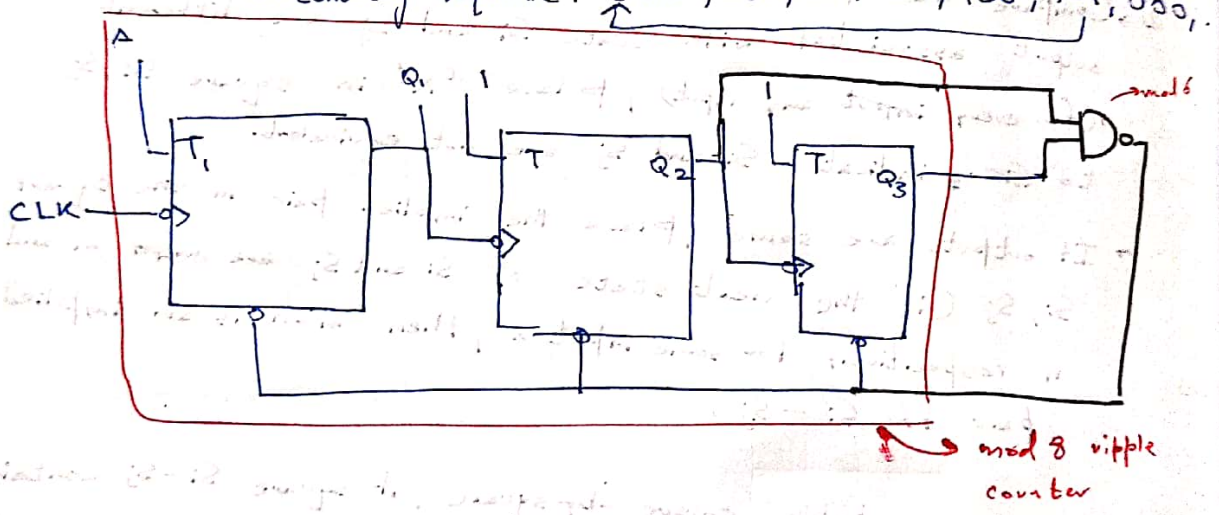
→ first we make a ripple counter of $\lceil \log_2 M \rceil$ stages

→ we go from 000, 001, ----- M-1, then give signal to CLR of all Flip Flop to return to 000

Eg Mod 6

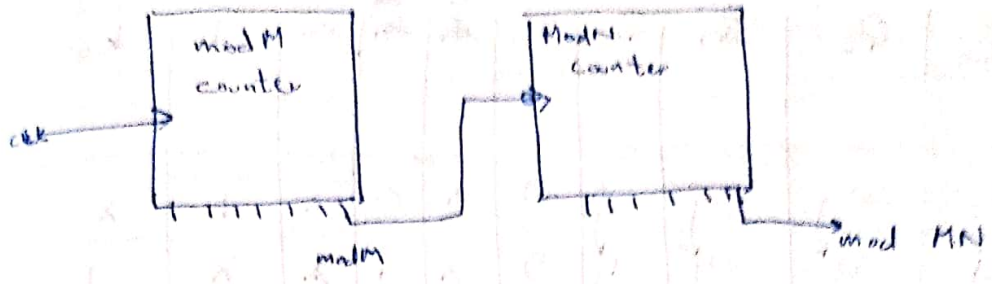
→ $\lceil \log_2 6 \rceil = 3$

→ counting sequence: 000, 001, 010, 011, 100, 101, 000, ..

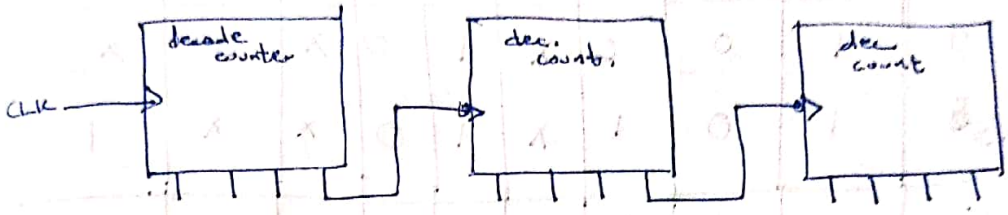


Eg: 0 → 999

and we have decade counter (0→9)



So for 0-999 → (10×10×10)



→ Ripple counter is asynchronous as only one FF connected to clk

Lockout

Eg: Mod 5 : 000 001 010 011 100

→ and 101, 110, 111 are unused.

→ if counter goes to any unused state (due to noise) counter will not know the next step, so the counter might cycle between unused states, This is called lockout.

Synchronous Counter will never reset, ← goes to 000, if counter goes to unused state

Eg: Mod 5 (using JK)

Counter Sequence S	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	X	0	X	1	X
1	0	0	1	0	X	1	X	X	1
2	0	1	0	0	X	X	0	1	X
3	0	1	1	1	X	X	1	X	1
4	1	0	0	X	1	0	X	0	X
US	1	0	1	X	1	0	X	X	1
US	1	0	1	X	1	X	1	0	X
US	1	1	0	X	1	X	1	X	1
~~US~~									

J_2 :

Q_2 \ Q_1Q_0	00	01	11	10
0			1	
1	X	X	X	X

$J_2 = Q_1 Q_0$

K_2 : Q_2 \ Q_1Q_0

Q_2 \ Q_1Q_0	00	01	11	10
0	X	X	X	X
1	1	1	1	1

$K_2 = 1$

J_1 :

Q_2 \ Q_1Q_0	00	01	11	10
0		1	X	X
1			X	X

$J_1 = Q_2' Q_0$

K_1 : Q_2 \ Q_1Q_0

Q_2 \ Q_1Q_0	00	01	11	10
0	X	X	1	
1	X	X	1	

$K_1 = Q_0 + Q_2 = \overline{Q_0' Q_2'}$

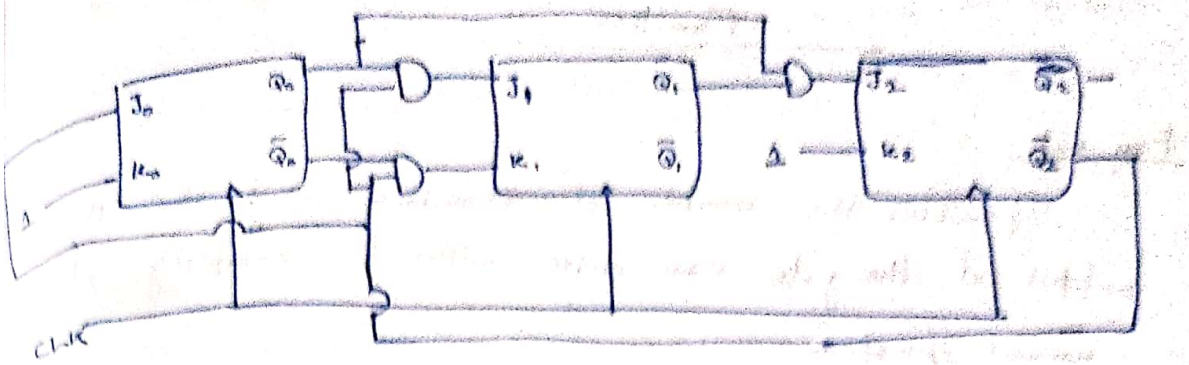
J_0 :

Q_2 \ Q_1Q_0	00	01	11	10
0	1	X	X	1
1		X	X	

$J_0 = \bar{Q}_2$

Q_2 \ Q_1Q_0	00	01	11	10
0	X	1	1	X
1	X	1	1	X

$K_0 = 1$

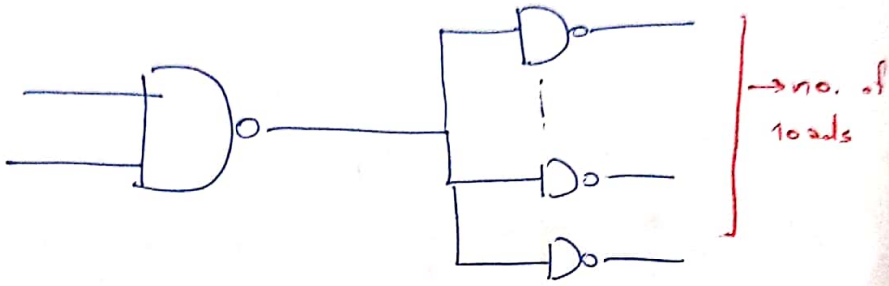

J_0
Q_0
K_0
$\bar{Q}_0$
J_1
Q_1
K_1
$\bar{Q}_1$
J_2
Q_2
K_2
$\bar{Q}_2$
A
CLK

IC Digital Logic Families

Fan out

Specifies the number of standard load that the output of the gate can drive without hampering it's normal operation.

Eg



→ as signal gets divided, after a threshold no., the gates will recive very less power to actually work. That threshold is fan out.

Power Dissapation

The power consumed by a gate

Propagation Delay

The average transition period for the signal to propagate from input to output, when the signal changes it's value.

	A	B	Y
t_1	1	1	1
$t_1+\delta$	0	1	1
t_2	0	1	0

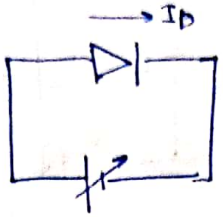
Eg

propagation delay = $t_2 - t_1$

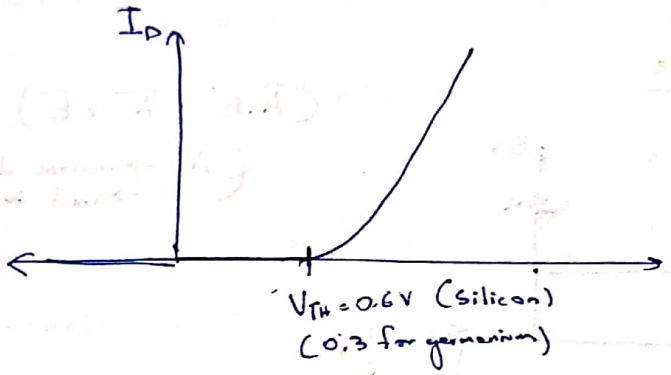
Noise Margin

The limit of a noise voltage which may be present without hampering the proper operation of the circuit.

Diode

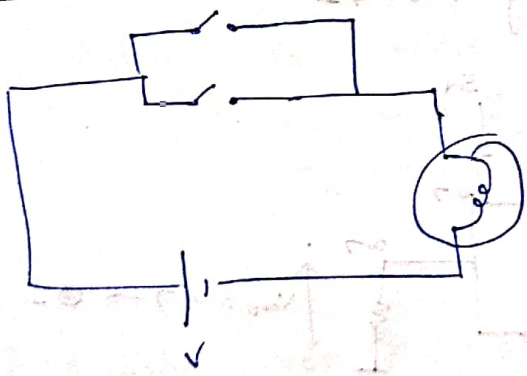


(assume silicon semiconductor)
(threshold V = 0.6 V)
(for germanium, V_{th} = 0.3 V)



∴ it can be used as swithch.

OR gate ~~diode~~



if any switch closes bulb will glow.

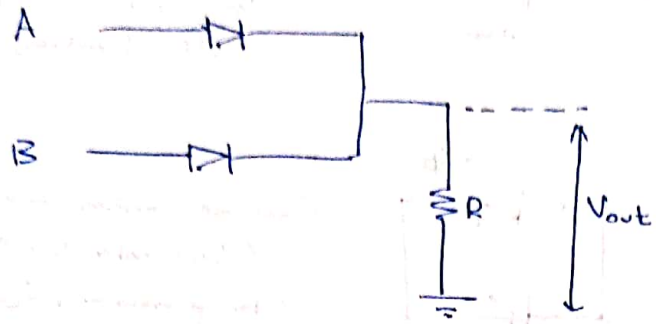
glow → 1
off → 0

OR gate Diode

0 V → off → 0 (0 – 0.6 V) (assuming Si)

~~more than 1~~

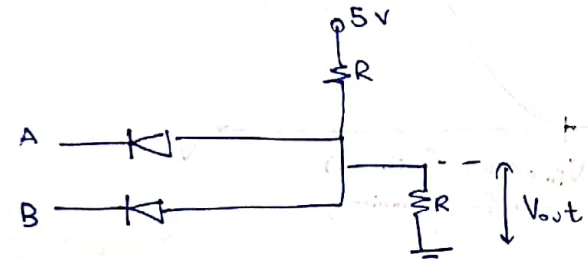
5 V → logic 1. (more than 0.6V)



NAND gate Diode

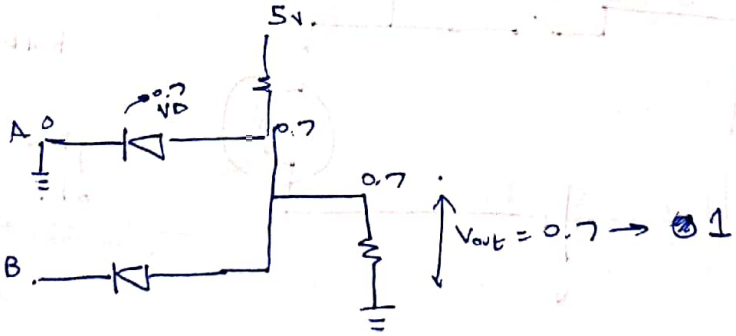
$(\overline{A.B} = \overline{A} + \overline{B})$

($\overline{A}$ → reverse diode → add +5V in end)



if A becomes ground (A=0), current flow through A (almost 0 res), making Vout = 1

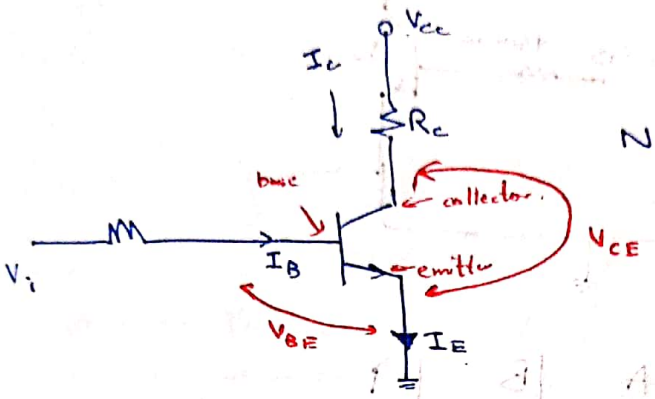
Eg



→ if A, B = 5 V, no current flow through diode,
making $V_{out} = \frac{5 \times 0}{2R} = 2.5\ V \rightarrow 0$

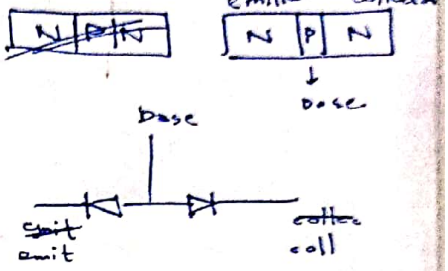
Transistor

→ have 3 phase.
- → cutoff (logic 0)
- → active → (not used, used as amp)
- → saturation (logic 1)



(remember Tap Analogy)

NPN Transistor.



$I_E = I_C + I_B$

→ depending on I_B, transistor goes to cutoff or saturation

cut off: $V_{BE} < 0.6$

$V_{CE} \Rightarrow$ open. (∞ V)

$I_B = I_C = 0$

Active: $V_{BE} \rightarrow 0.6 - 0.7$ V

$V_{CE} > 0.8$ V

$I_C = h_{FE} I_B$

h_{FE} → DC current gain. (Transistor parameter)

Saturation: V_{BE} (0.7 − 0.8) V

$V_{CE} = 0.2$ V

$I_B \neq \frac{I_C}{h_{FE}}$

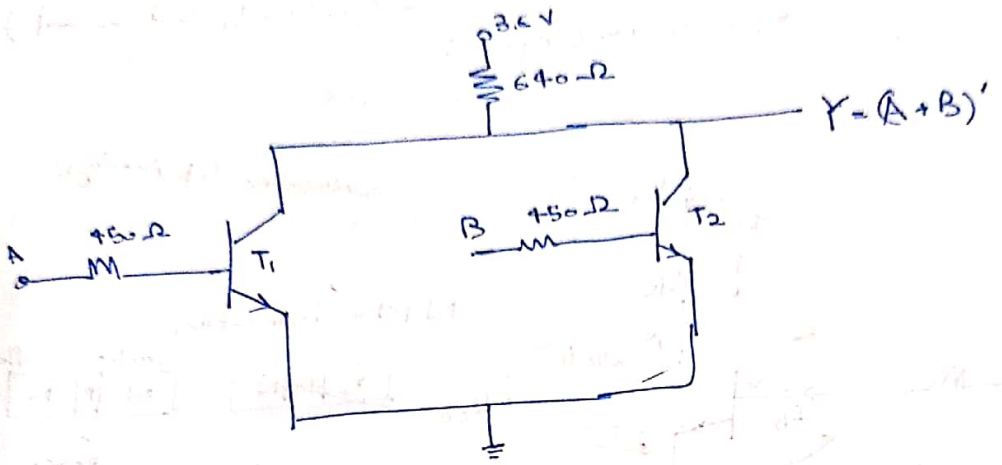
→ Transistor is called a current controlled device.
(we control it by I_B)

Resistor Transistor Logic (RTL)

→ basic gate : NOR gate

→ Logic 0 : 0.2 V

→ Logic 1 : ~~1.2 3.6 V~~ 1 - ~~2.8~~ 3.6 V



A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

→ if A = 3.6 , B = 0 ; T_1 → saturation (Tap open) → Y = 0

A = 0 , B = 3.6 , T_2 → saturation → Y = 0

A = 3.6 , B = 3.6 , T_1, T_2 → saturation → Y = 0

→ A = 0 , B = 0 → T_1, T_2 → cutoff (Tap off) → Y = 1

→ if any of A , B = 3.6 , V_{CE} = 0.2 , making

Y = ~~3.6 - 0.2~~ 0.2 → 0

~~640~~

→ if both A , B = 0 , cutoff $V_{CE} = \infty$

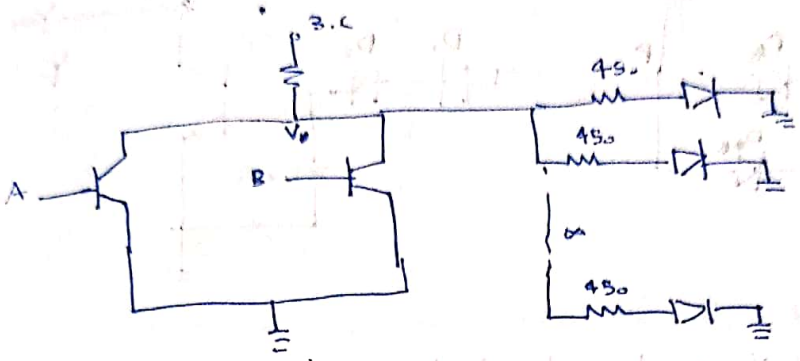
Y = 3.6 V → 1

Fan out = 5

Power Dissipation : 12 mW

Propagation Delay : 25 ns

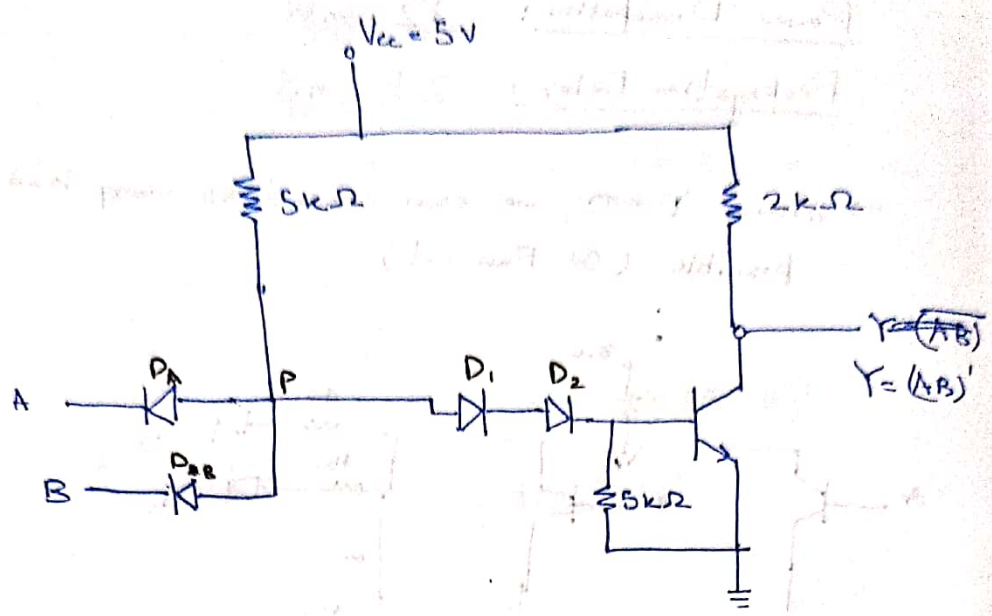
→ when Y = 0, we can connect as many load as possible (∞ Fan out)



→ when Y=1, as we keep increasing load, Voltage for load decreases, when it become less than 0.2, it hampers the output., hence Y=1 is limiting for fan out.

(more load, more current, V decrese, after limit Y changes)

Diode Transistor Logic (DTL)



0 - 0.2 V : Low level Logic

4 - 5 V : High Level Logic

Basic Gate: NAND Gate.

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

→ if A = 0.2 V, B = 5 V

V_{PA} — V_P = 0.5 V V_A = 0.2 V

$\therefore V_{PA}$ =

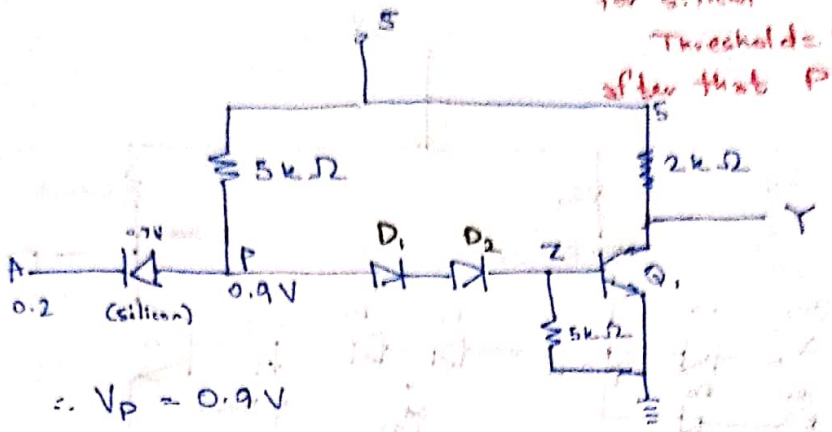
→ D_A will conduct

hence

I) if A = 0.2 V , B = 5 V

→ D_A will conduct

for silicon
Threshold = 0.6 V
after that PD = 0.7 V



∴ $V_P = 0.9$ V

~~also $V_Z = 0.9$~~

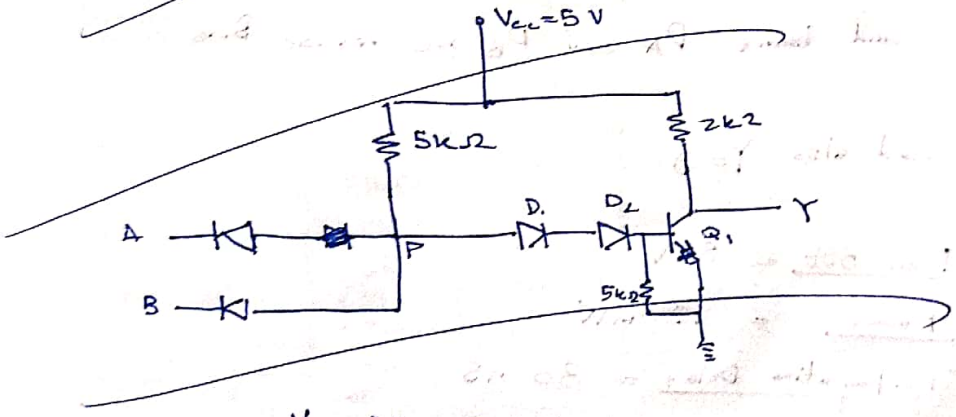
→ To run Q_1, we need min. $V_P = 1.8$ V

as we need 0.6 × 2 V for D_1 and D_2 and more 0.6 V for Q_1 to run.

→ but $V_P = 0.9$ V, so Y will remain at logic 1

II) if A = 5 V , B = 5 V

→ D_A and D_B will not conduct

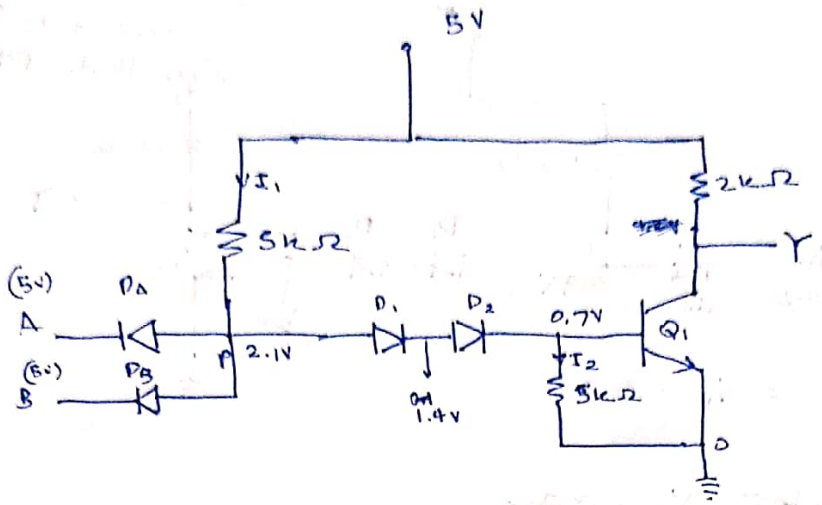


∴ $V_P = 5$ V > 1.8 V

∴ Q_1 will be active ; Y = 0

II) if A = B = 5V

∴ D_A and D_B will not flow



∴ $I_B = I_1 - I_2$ (Transistor saturation)

∴ $V_{CE} = 0.2$ V

∴ ~~$V_C = 5$ ∴ $V_E =$~~

as both diode D_1 and D_2 and Q_1 are flowing,

Voltage Drop = $0.7 \times 3 = 2.1$V

∴ ~~$V_P = 5 + 2.1 V_C = +2.9$ V~~

$V_{DA} = 2.1 - 5 = -2.9 = V_{DB}$

and hence D_A and D_B are reverse Bias

and also $Y = 0$

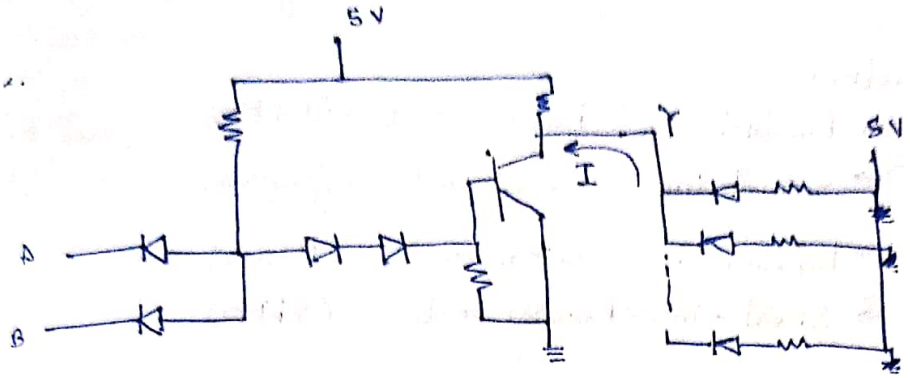
Fan out = 8

Power = 12 mW

Propagation Delay = 30 ns

Limiting Fan out

Let Load be



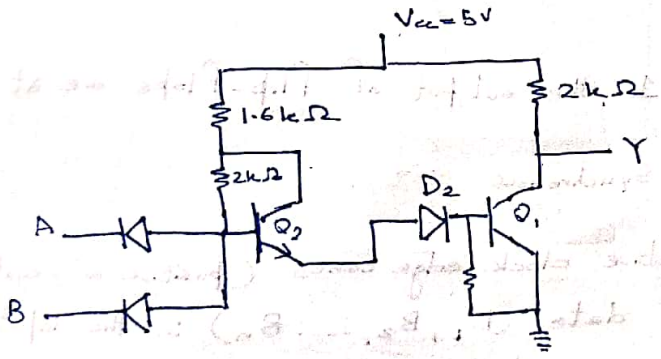
when $Y = 1$, PD between load diode is ~~flte~~ not flowing

when $Y = 0$, current flowing to DTL, hence I_C will increase

as $I_B h_{fe} \geq I_C$ for saturation

⇒ as $I_C \uparrow$, after some load, transistor come out of saturation, hampering DTL

Alternative circuit to increase fan out



→ when Q_1 is saturated, Q_2 is active

Q_2 active, $I_E = I_B + I_C$

$I_E = h_{FE_2} I_B$; I_B can be increase

→ hence I_E can be increased., which is I_B for Q_1

∴ for Q_1 : $I_B h_{fe_1} \geq I_C$

↓ can be increased using Q_2 ; ↓ will increase due to load

→ and Q_2 saturation can be maintained.

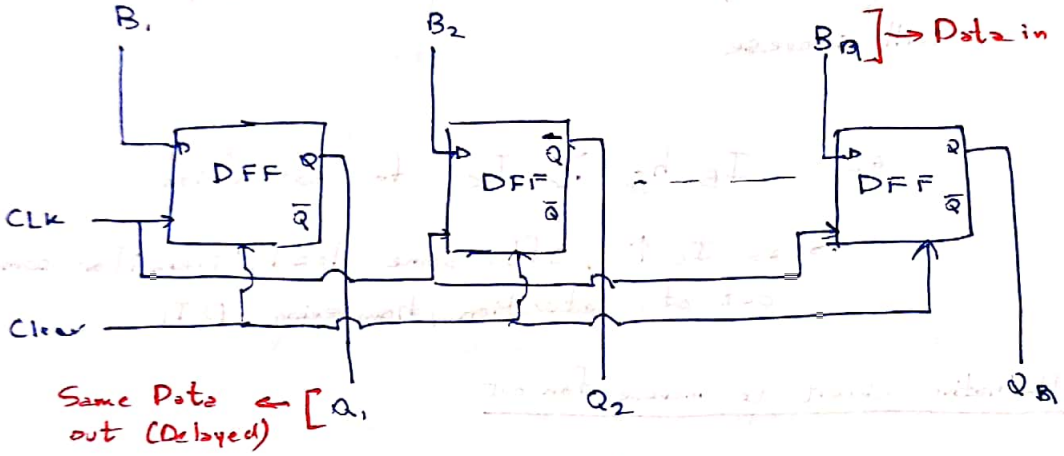
Design Of Registors

→ A resister consists of a group of flip-flops with a common clock input, used for storing binary information (data)

→ Variation:
- → Parallel-in-Parallel-out (PIPO)
- → Seriel-in-Serial-out (SISO)
- → Parallel-in-Serial-out (PISO)
- → Sireal-in-Parallel-out (SIPO)

→ If a register support either serial-in or serial-out, or both, it is called a shift register.

PIPO



→ when Clear = 1, the output of flip-flops are at 0

→ Same clock: Synchronous

→ When the active clock edge comes (positive or negative) the available data ($B_1, B_2, \ldots B_n$) in the inputs is stored in the register and also available at it's output

Addition of Load Signal with Basic PIPO

→ In practice, the clock is coming continuously and there is a seperate signal LOAD that specifies when register is to be loaded with new data.

→ Two possible ways to introduce load.

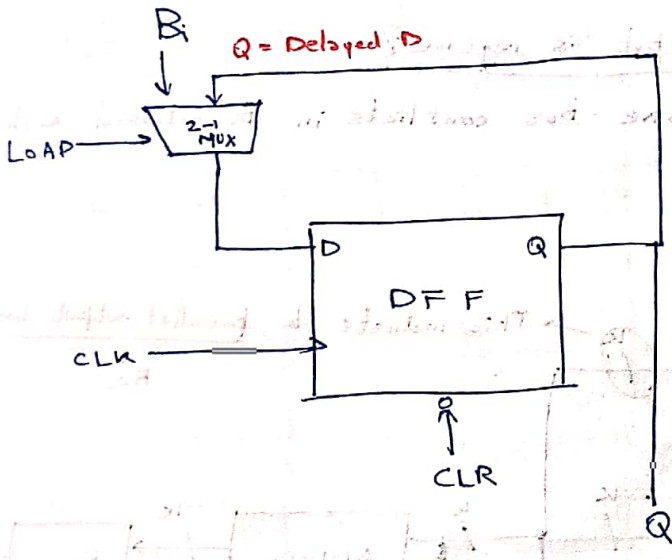
i) Use a gated clock (like LOAD && CLK)

→ not a good sol^{n}, as gating the clock with another signal can cause timing problem. LOAD has to come before clock arrives

ii) Seperate out clock and LOAD using MUX's

→ Better and recommended solution

→ Each Flip-Flop in the registers gets replaced by the following.



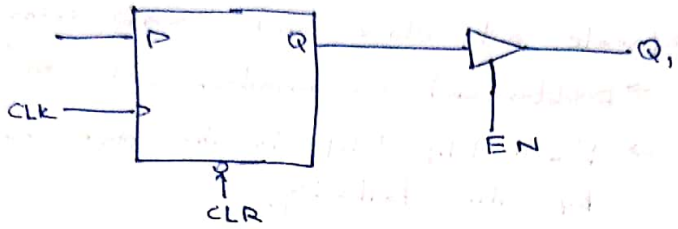
Addition of Enable with the PIPO resistor

→ Many registers have an enable input that can be used to force the output lines into high impedance state, if required.

↓
(A very high impedance means we make it so that output is neglected)

→ We need tri state buffer with every flip-flop output.

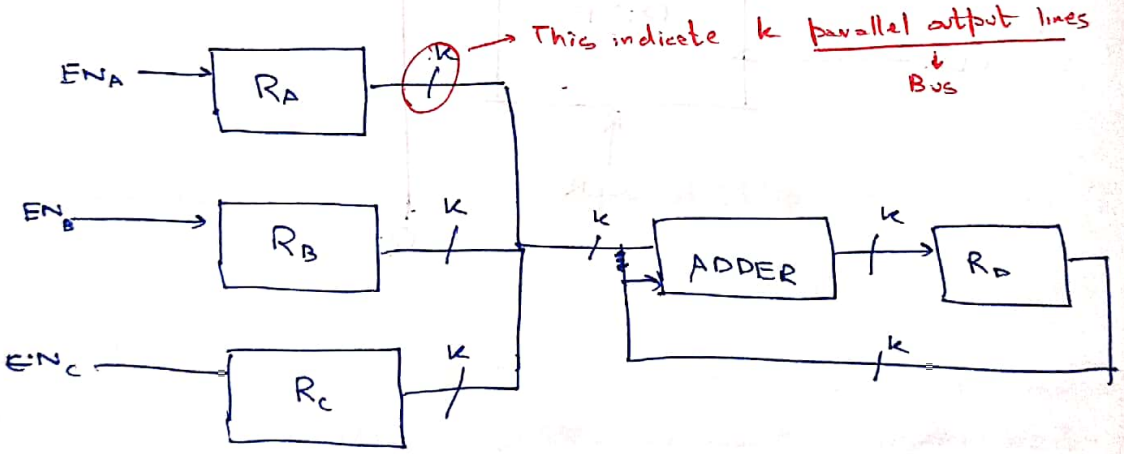
When ENABLE = 1 , $Q_1 = Q$
ENABLE = 0 , Q_1 = High Impedance (neglect)



→ Why Enable input is require?

→ To resolve bus conflicts in bus-based architecture.

Example



If $EN_A = 1$, then $R_D = R_D + R_A$
$EN_B = 1$, then $R_D = R_D + R_B$
$EN_C = 1$, then $R_D = R_D + R_C$

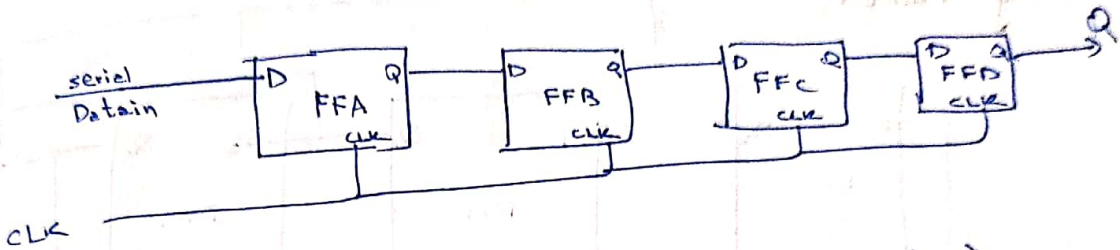
→ at most one of EN_A, EN_B, $EN_C = 1$, so as to avoid data collision.

Shift Register

→ A Shift Register is a register in which the binary data can be stored and the data can be shifted to the left (or right) when a shift signal is applied.

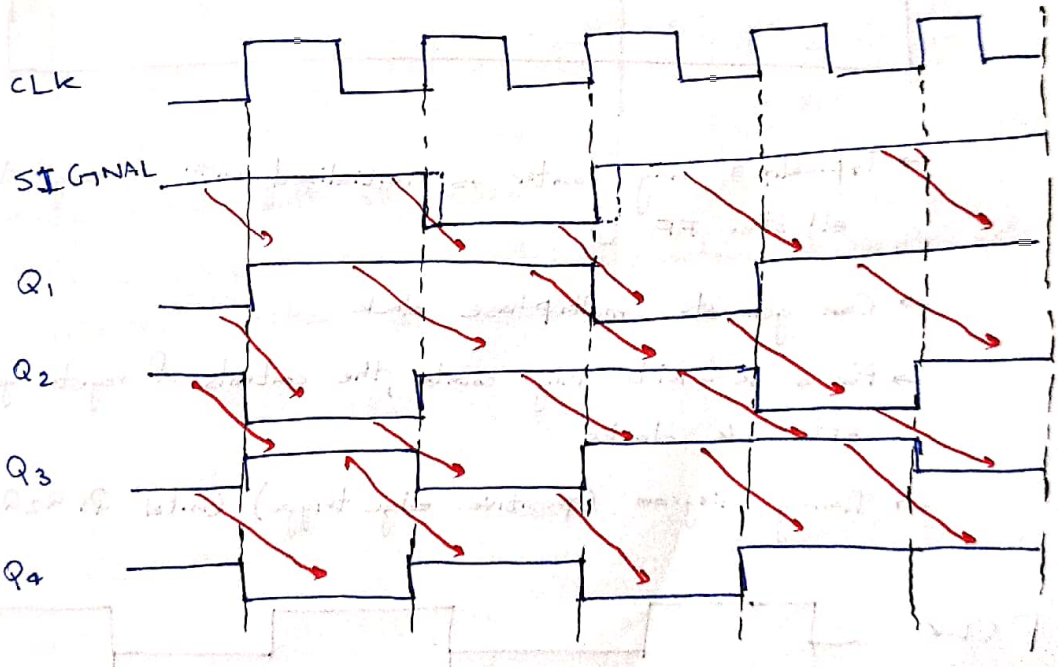
→ It can be constructed by connecting D, SR, or JK FF in cascade.

⇒ Eg: 4 bit shift register.



⇒ Eg: sample Timing diagram (for positive edge)

→ Initail $Q_1Q_2Q_3Q_4 = 0101$

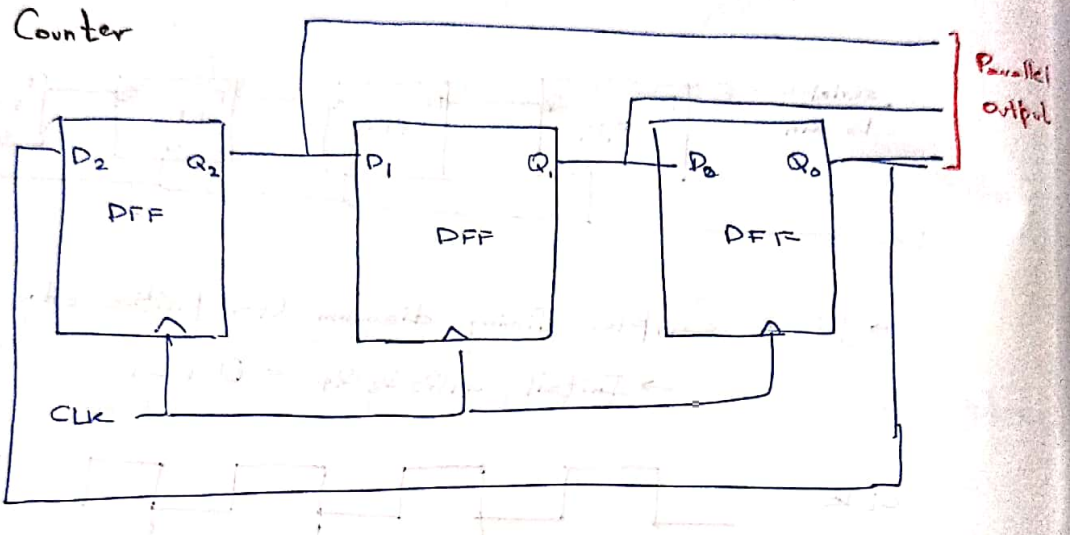


Different Types of Flip-Flops

Depending upon the way various storage devices are connected, there can be different types of shift-registers:

i) Ring Counter
ii) Twisted Ring ~ Jhanson Counter
iii) Bidirectional Shift Register
iv) Universal Shift Register
v) Linear Feed back Shift Register.

Ring Counter

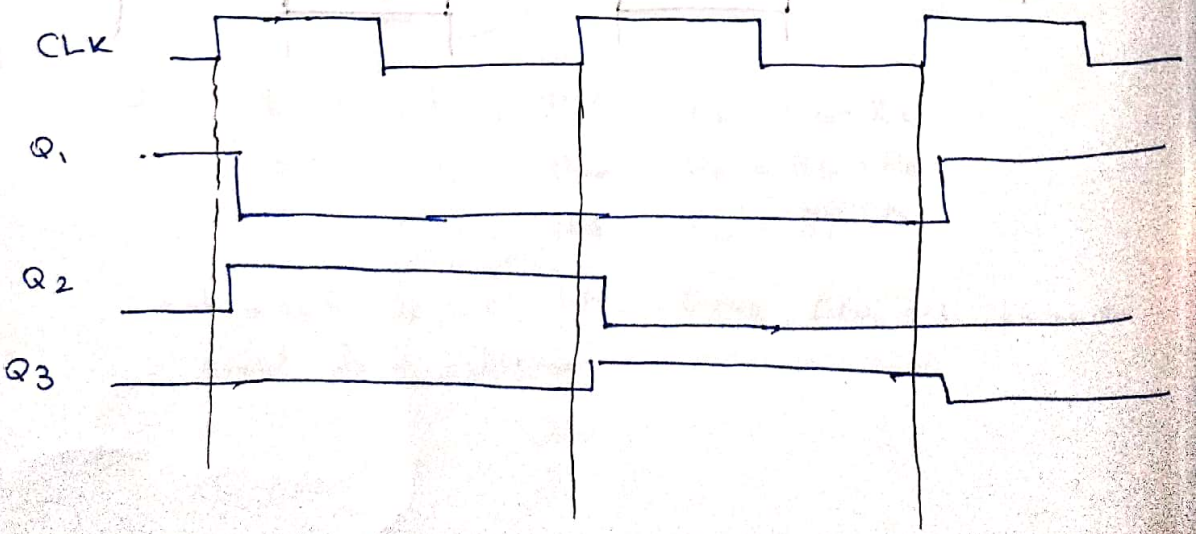


→ Typically a ring counter is initialised with a single 1 and all other FF 0

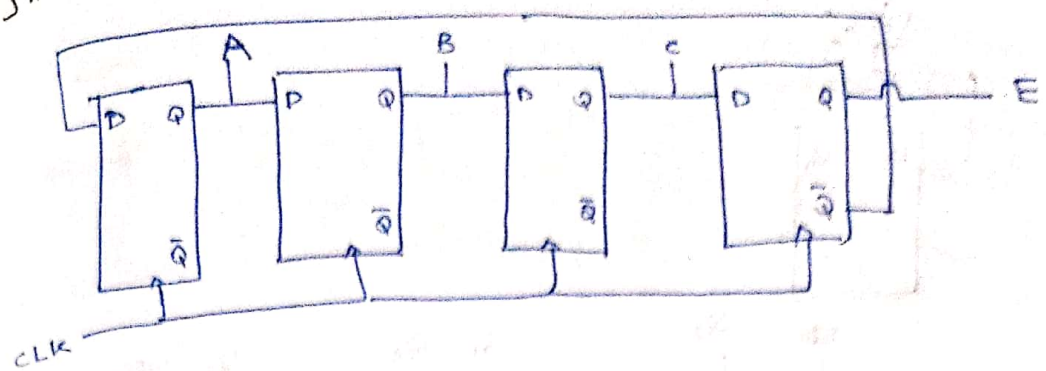
→ Can generate multiphase clock

→ For a k-bit ring counter, the contents of register get repeated after k clocks.

→ Timing diagram (positive edge trigger) (initial $Q_1 Q_2 Q_3 = 100$)



Jhonson Counter



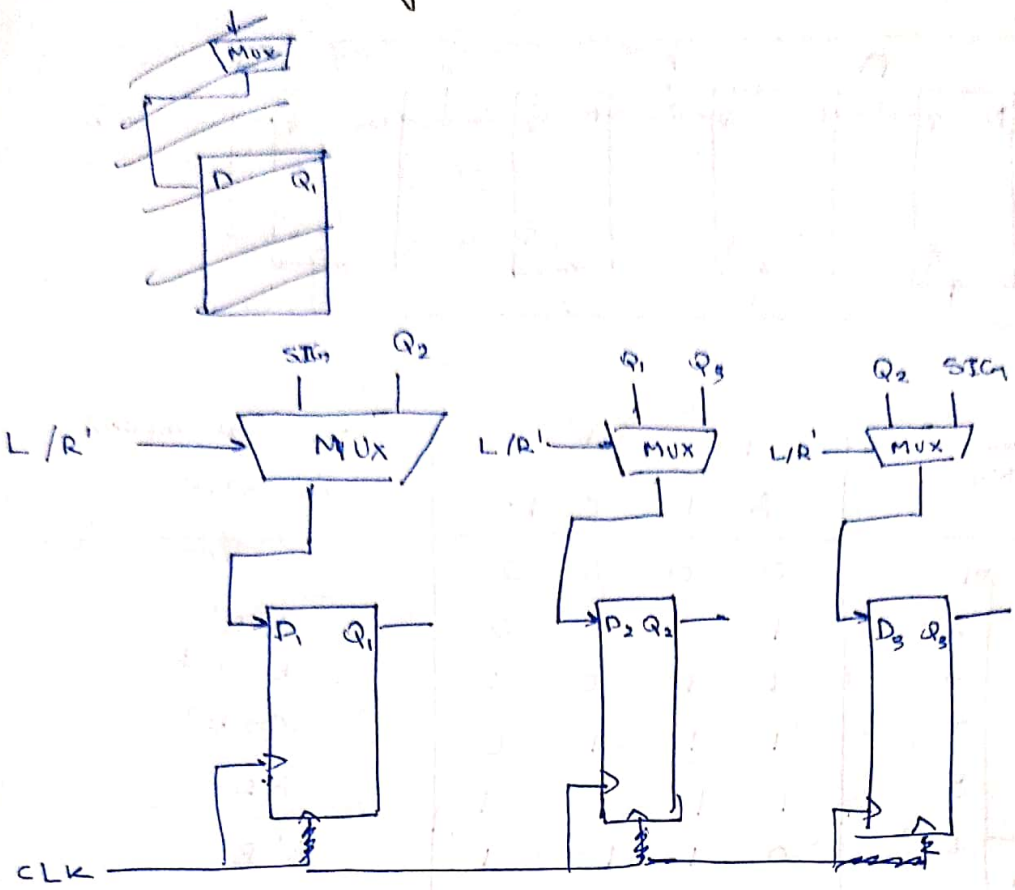
Sequence No	FF output A	B	C	E	AND gate required for output
1	0	0	0	0	$A'E'$
2	1	0	0	0	AB'
3	1	1	0	0	BC'
4	1	1	1	0	CE'
5	1	1	1	1	AE
6	0	1	1	1	$A'B$
7	0	0	1	1	$B'C$
8	0	0	0	1	$C'E$
9	0	0	0	0	$A'E'$

1 to 9 same
diff = 8 = 2×4

→ Initialised with all zero

→ For a K-bit Jhonson Counter, contents of registers gets repeated after 2k Clock sequence.

Bidirectional Shift Register

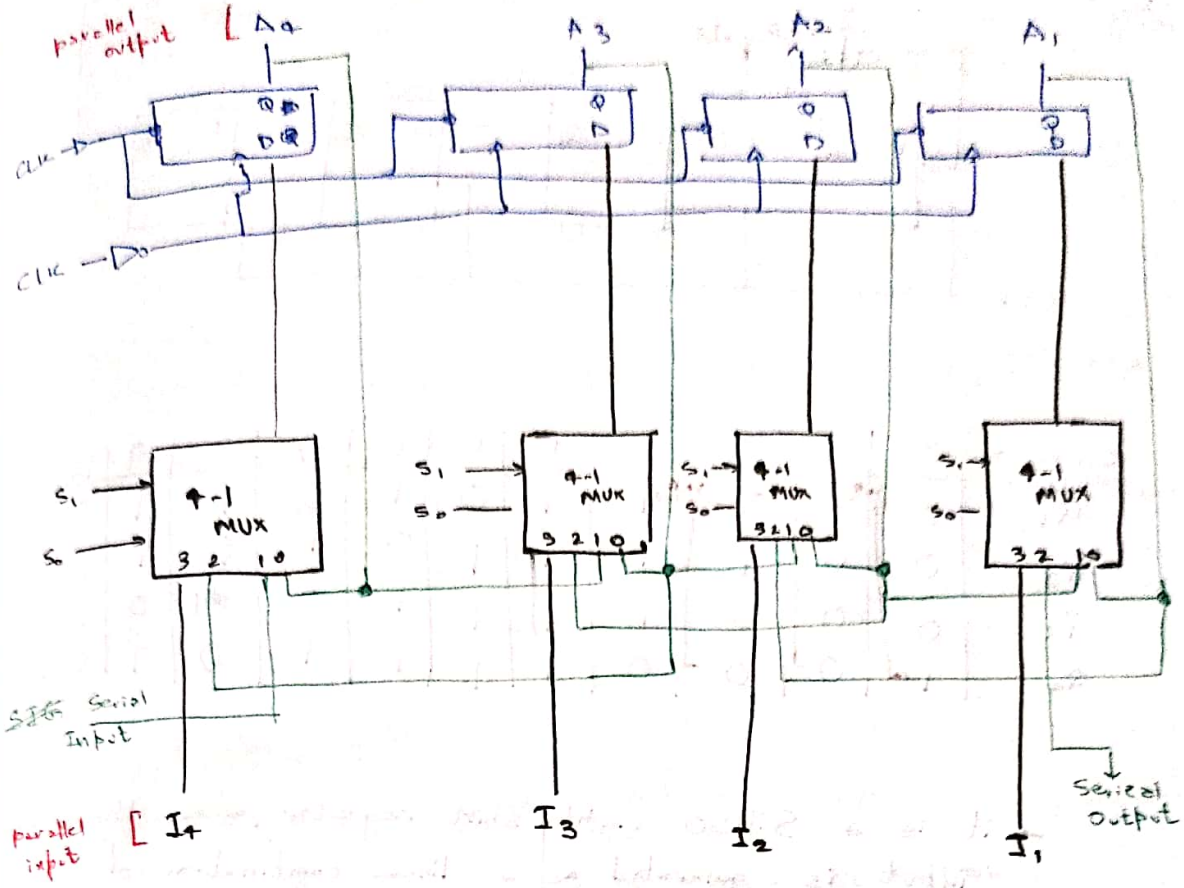


→ Control Signal $L/\bar{R}$ is used as select for MUX

$L/\bar{R} = 0$ ⇒ left shift

$L/\bar{R} = 1$ ⇒ right shift

Universal Shift Register

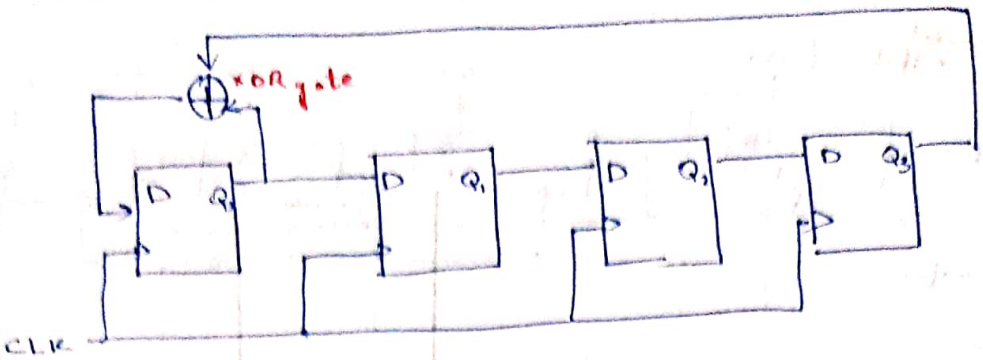


→ A universal shfit register is a ~~digi~~ bidirectional shift register whose input can be either in serial or in parallel and whose output can also be either in serial or in parallel.

Mode Control		Operation
S_0	S_1	
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel Mode.

Linear Feedback Shift Register

Eg:



CLK →	0	1	2	3	4	5	6	7	8	9	10	11
Q_1	0	1	1	1	1	0	1	0	1	1	0	0
Q_2	0	0	1	1	1	1	0	1	0	1	1	0
Q_3	0	0	0	1	1	1	1	0	1	0	1	1
Q_4	1	0	0	0	1	1	1	1	0	1	0	1

→ it is a SISO right shift register, where the serial input is generated as a linear combination of some of the Flip Flop output.

→ This is used to create random patterns

→ generally initialized to $Q_4 Q_3 Q_2 Q_1 = 1000$

Design Of Counters

Design of Binary Counter of any arbitrary Modulus

→ We want to design a modulo M-counter for any M

→ counter will count from 0 to M-1, then back to 0

→ Assume FF's have CLR inputs.

Basic idea

→ We first design a ripple counter with $\lceil \log_2 M \rceil$ stages.

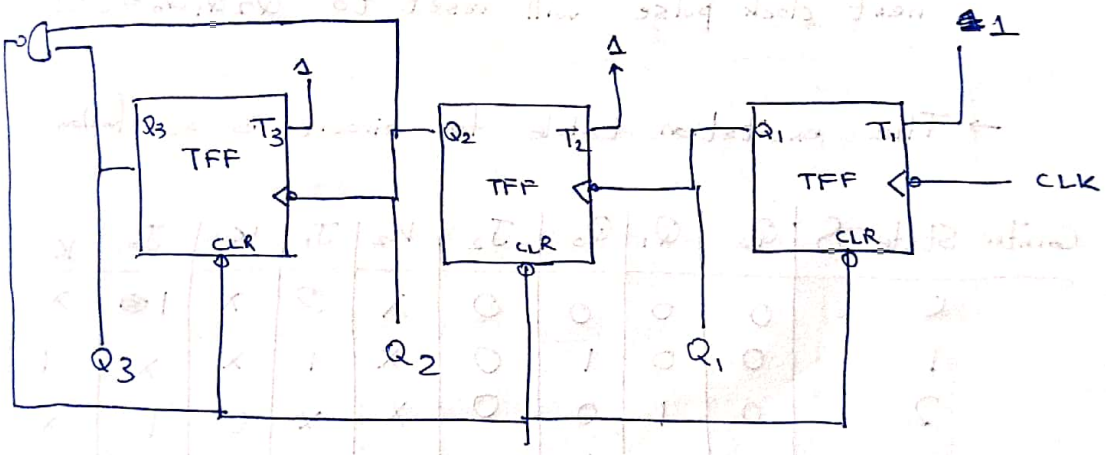
→ Then, we will design a gating circuit which takes input from the output and generates a signal, whenever the value reaches M.

→ connect output of gating circuit to CLR of FF's.

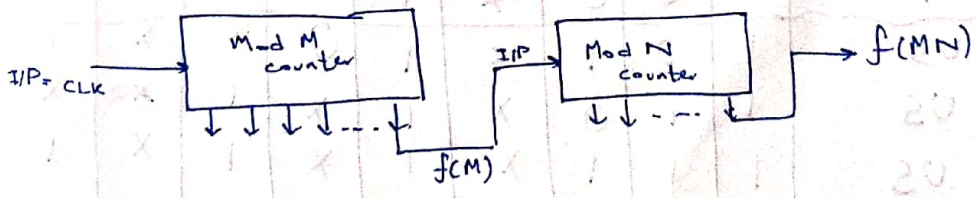
Mod-6 Ripple counter

→ No. of stages: $\lceil \log_2 6 \rceil = 3$

→ 000 → 001 → 010 → 011 → 100 → 101



Cascading of Ripple counter.



→ If a mod M counter and a mod-N counter are connected in cascade, combination will count mod-MN

Lockout

→ For a Mod-5 counter, $Q_2Q_1Q_0$ = 101, 110, 111 are not used. Suppose that due to external noises, the counter happend to find itself in invalid state. we do not know the next state for it.

→ ~~It might happen that~~

→ It might happen that output for unused state also gives unused state, and continue to cycle through unused states, never arriving at used state.

→ A counter whose unused states have this property, is said to suffer from lockout.

Example of Synchronous Counter

→ Design a Mod-5 synchronous counter such that if the unused states $Q_2Q_1Q_0$ = 101, 110, 111 occur, next clock pulse will reset to $Q_2Q_1Q_0$ = 000.

→ The excitation table for circuit is as follows

Counter State S	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	X	0	X	1	X
1	0	0	1	0	X	1	X	X	1
2	0	1	0	0	X	X	0	1	X
3	0	1	1	1	X	X	1	X	1
4	1	0	0	X	1	0	X	0	X
US	1	0	1	X	1	0	X	X	1
US	1	1	0	X	1	X	1	X	X
US	1	1	1	X	1	X	1	X	1

from this table and using K-Map, we get

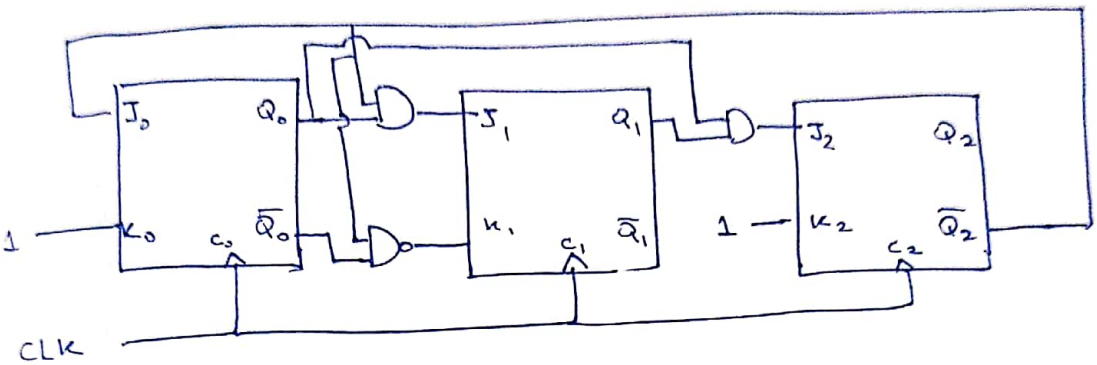
$J_2 = Q_1 Q_0$ $\quad J_1 = Q_2' Q_0$

$K_2 = 1$ $\quad K_1 = Q_2 + Q_0 = \overline{\overline{Q_2}\,\overline{Q_0}}$

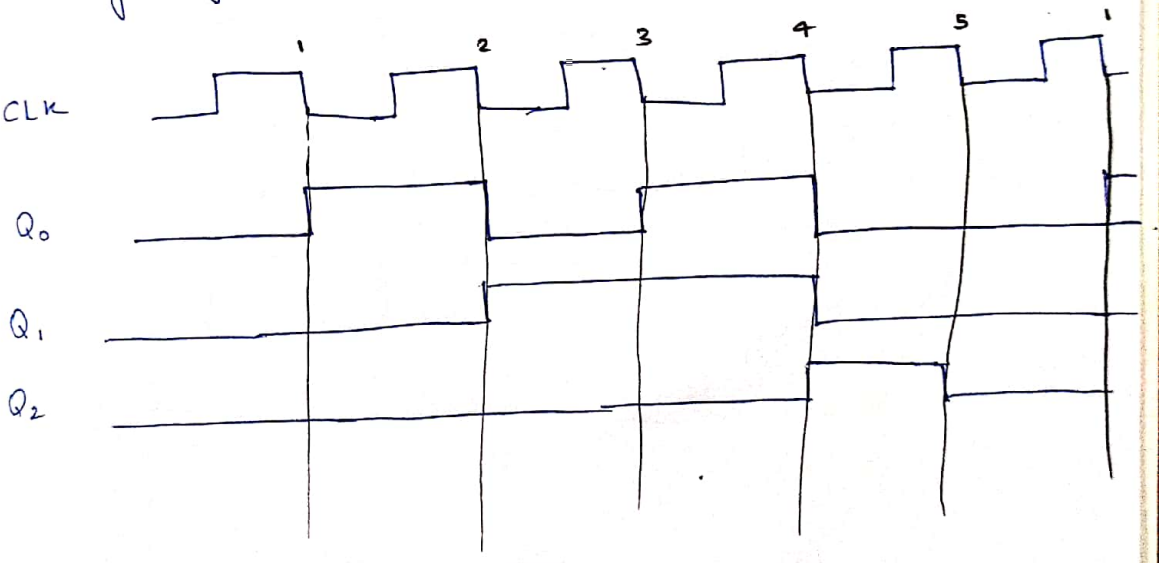
$J_0 = \overline{Q_2}$

$K_0 = 1$

so, the circuit diagram



→ Timing Diagram (-ve edge trigger)



TTL

→ Transistor Transistor Logic

→ Types
  → Standard TTL

Standard TTL

Propagation Delay: 10 nS
Power Dissapation: 10 mW

Low Power TTL

Propagation Delay: 33nS
Power Dissapation: 1 mW

High-Speed TTL

Propagation Delay: 6 nS
Power Dissapation: 22mW

Schottky TTL

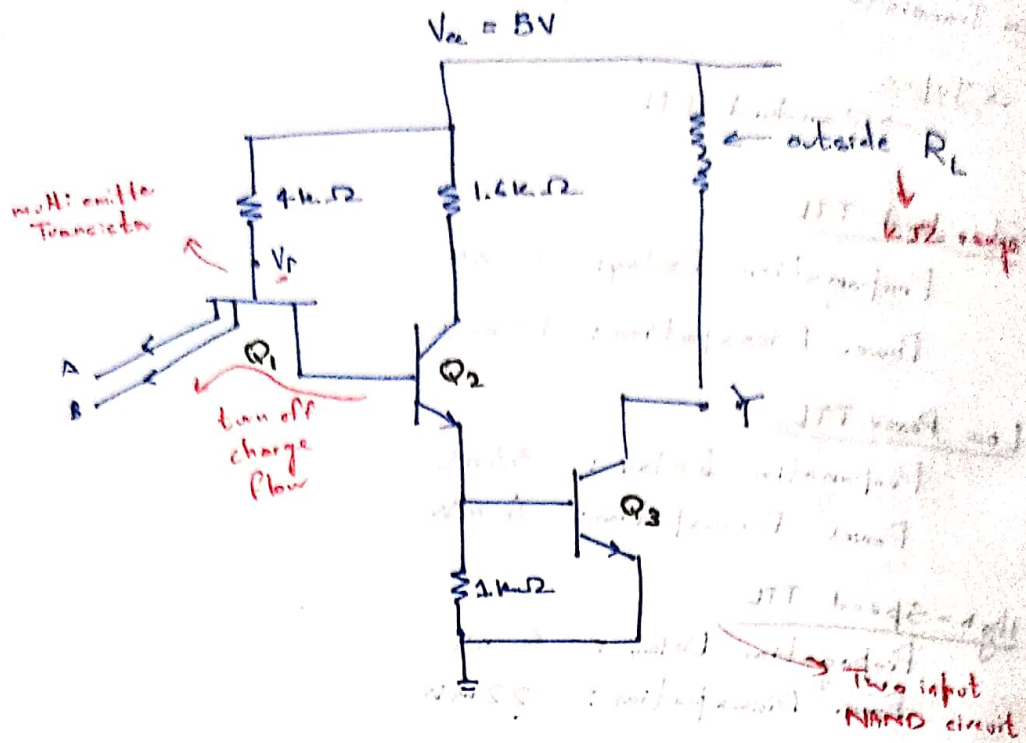
Propagation Delay: 3nS
Power Dissapation: 19 mW

Low Power Schottky TTL

Propagation Delay: 9.5nS
Power Dissapation: 2mW

→ all TTL family has three types of output configs:
  → Open-Collector TTL
  → Totem-Pole TTL
  → Tri-State TTL

Open Collector TTL



→ To get output you have to externally connect resistor from V_{cc} to Y.

→ Low Level = 0.2 V

→ High Level = 2.4 - 5 V

→ Basic Circuit : NAND gate

→ if A = 0.2 V, B = 5 V (A = 0, B = 1, Y should by 1)

V_P = 0.9 V (0.2 + 0.7) flowing V

→ To get Y = 0, we need to cross the transistor

∴ min V_P = 3 × 0.6 = 1.8 V

threshold V

∴ V_P = 0.9 V

∴ Y = 1

→ if both A = 0.2 V, B = 0.2 V

V_P = 0.9 V

∴ Y = 1

→ if A=5V, B=5V

→ $V_p = 5V$

∴ Y=0 [Q_1, Q_2, Q_3 saturation]

($V_Y = 0.2V$ if Q_3 saturate)

→ This is similar to DTL but propagation Delay in DTL is 30 ns, but here it 10ns, why?

→ consider initially A=5, B=5, Q_1, Q_2, Q_3 saturate

→ why any of A or B (say A) is become 0.2V

Q_1 goes to active, base of Q_2 gives charge to Q_1, making turnoff faster.

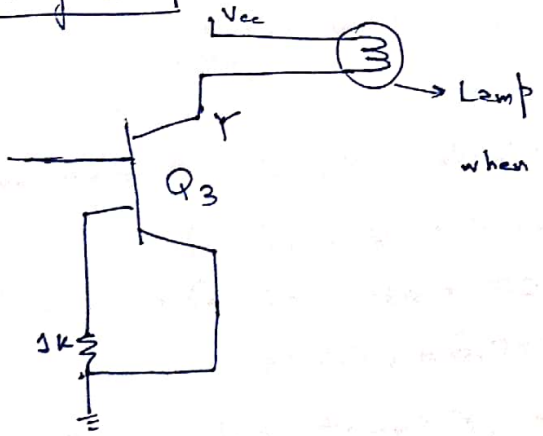
→ Application of Open Collector.

→ Driving Lamp or Relay

→ Wired Logic.

→ Construction of Common Bus System.

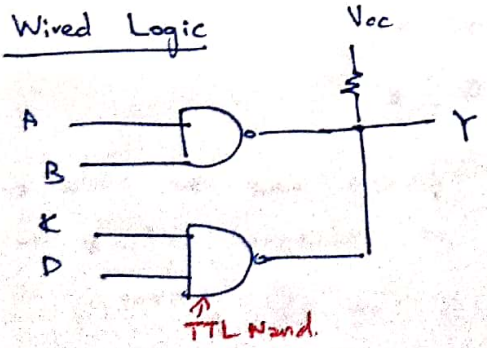
Driving Lamp



when Y=0, Lamp glow

Y=1, Lamp off

Wired Logic

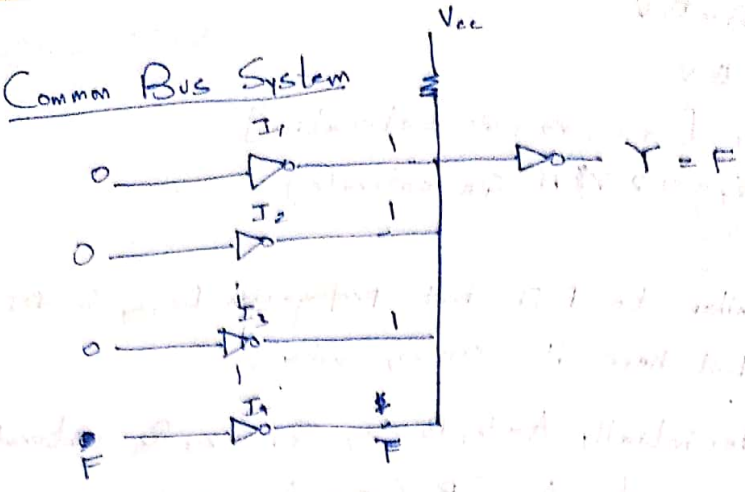


→ output of TTL's can be connected

$Y = \overline{(AB)}' \cdot \overline{(CD)}$

$= (AB + CD)'$

Common Bus System



→ keep input for all I_{n-1} to 0 and use I_n as signal

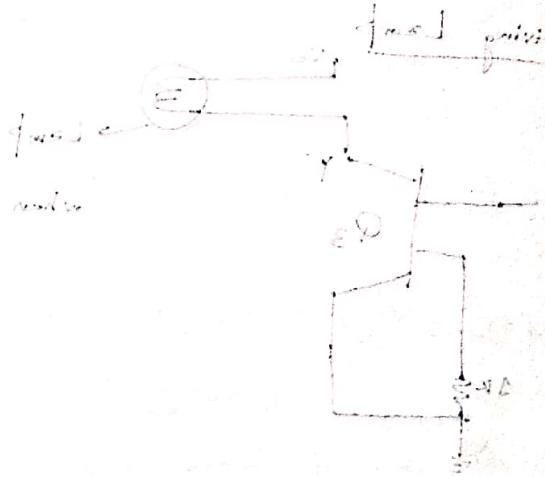
→ $Y = I_n$ input

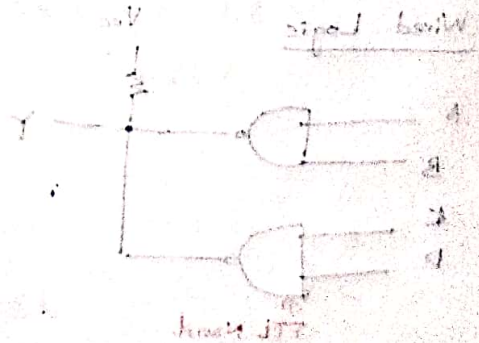
→ hence we can connect outputs of several system to one output

here system is inverter

~~→ Phase not gate can be made by TTL~~

→ Systems can be made by TTL NAND GATE





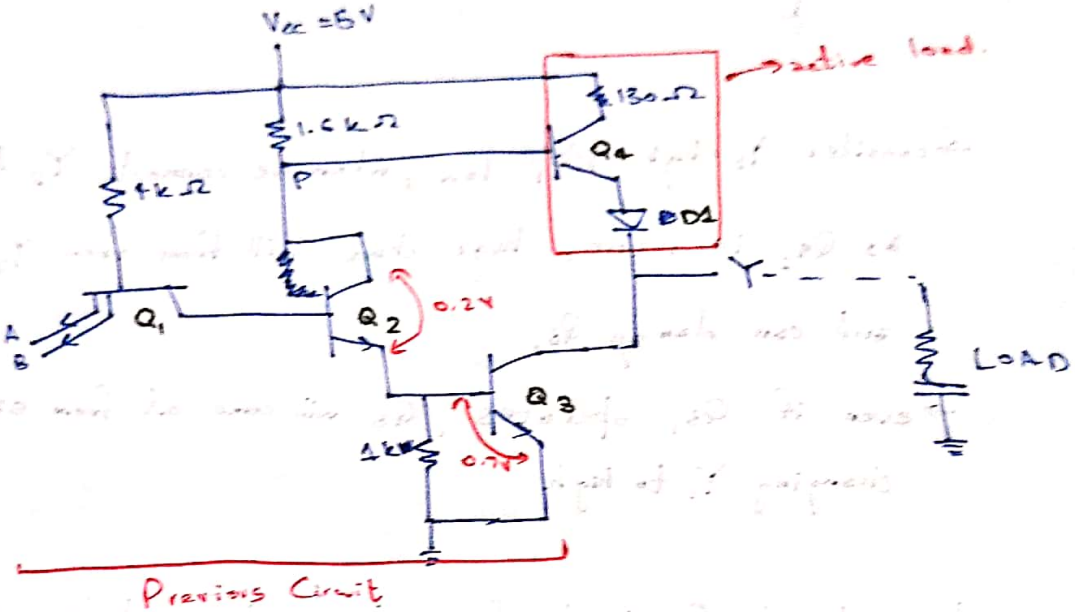
Tofem - Pole TTL

→ Why Tofem - Pole.

→ In open circuit, propagation delay depends on output R_L - Load Resistance and Capacitance. (RC)

→ ~~R_L~~ ; if we could reduce value of R, delay could decrease.

→ Tofem - Pole achives this by changing passive R_L to an active component.



→ active load.

→ when A=5=B , Q_2, Q_3 saturation (Y=0)

→ for Q_3 , $V_{BE} = 0.7V$

→ for Q_2 , $V_{CE} = 0.2V$

∴ $V_p = 0.9V = (0.7+0.2)$

→ To conduct through Y, we pass ~~I_{base}~~ Q_4 & D_1

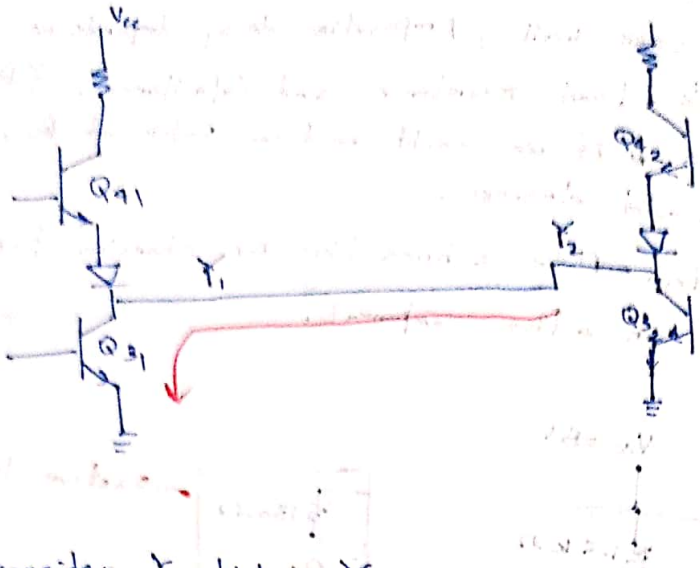
∴ min $V_p = 0.6 \times 2 = 1.2$

* ∴ Resitance of active load high.

→ ~~when A=B=5 any go off (Y=1)~~

→ when connected to load, Y=0, capacitor will discharge, making Q_4 go to active.

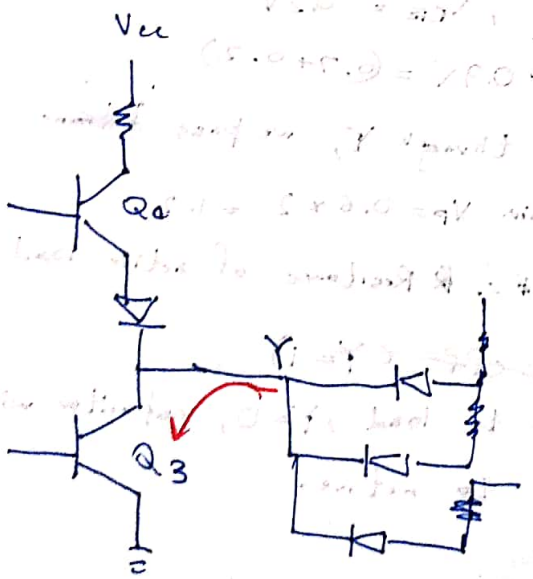
→ we can't do wired Logic with Totem-pole



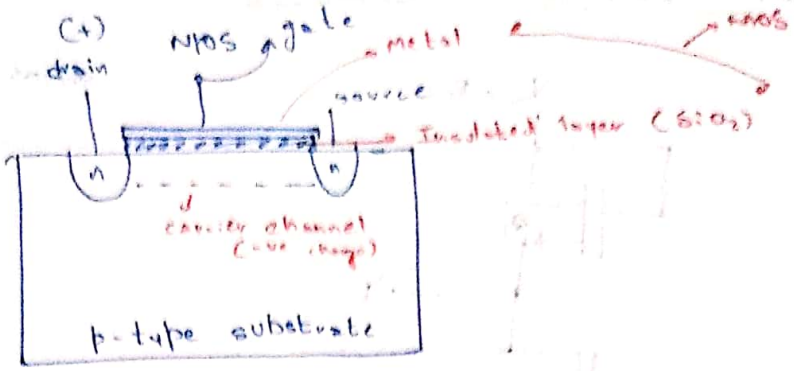
→ consider Y_2 high, Y_1 low, when we connect Y_2 & Y_1 as Q_{A_2} is active, huge charge will from from Y_2 to Y_1 and can damage Q_{B_1}

→ even if Q_{B_1} operates, Q_{B_2} will come out from saturation, changing Y_1 to high.

→ Logic Level 0 limit fan out for totem-pole, as $Y=0$, more load will inject current to Q_3, after a limit, Q_3 goes out of saturation, changing output



n-type channel

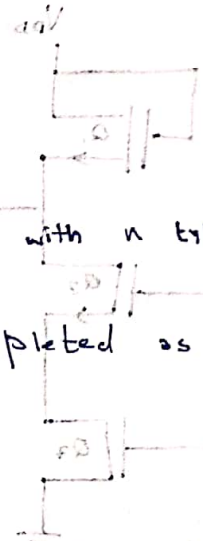


→ when +ve voltage applied on drain, p type substrate and Metal layer becomes like capacitor, forming a channel of carriers.

→ we can also make a p-type channel where all
i) n type becomes p
ii) p become n
iii) source is +ve
iv) channel is of +ve charge.

Two types Mos can operate

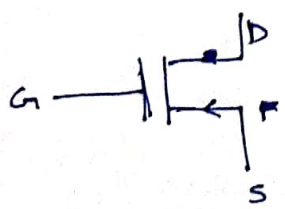
i) Depletion mode
ii) Enhancement mode



Depletion mode → Channel slightly doped with n type, it decreases threshold
→ The channel gets depleted as @Mos becomes -ve volt.

Enhancement mode
→ No channel initially
→ as we increase voltage, Channel gets made, enhancing channel region.

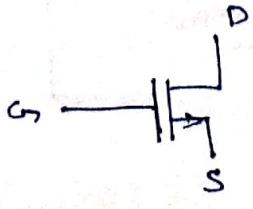
symbol of p-channel Mos



V_{GS} = -ve $\overset{>}{\not<}$ V_T

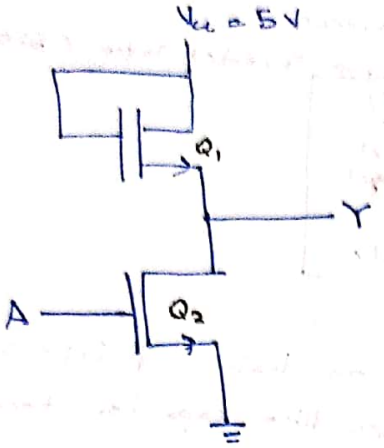
Eg conduct when V_T = -1 & V_{GS} = -1.5

n-channel



conduct when
V_{GS} = +ve > V_T
↓
Threshold

→ MOS transistor as ~~Register~~ Inverter



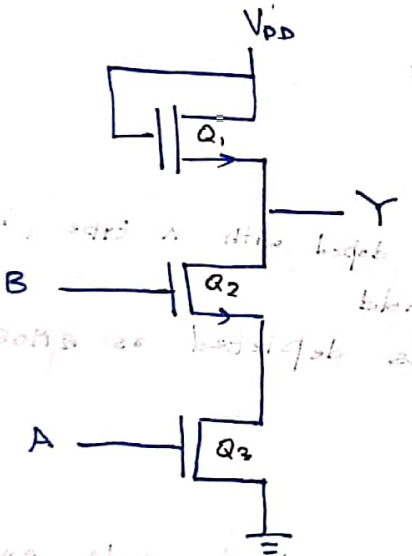
assume $V_T = 1V$

→ Q_1 is always on as Gate connected to V_{cc}

→ when A = 0, Q_2 will no conduct, Y = 1

→ when A = 1 (5V), Q_2 will conduct, Y = 0

→ NAND



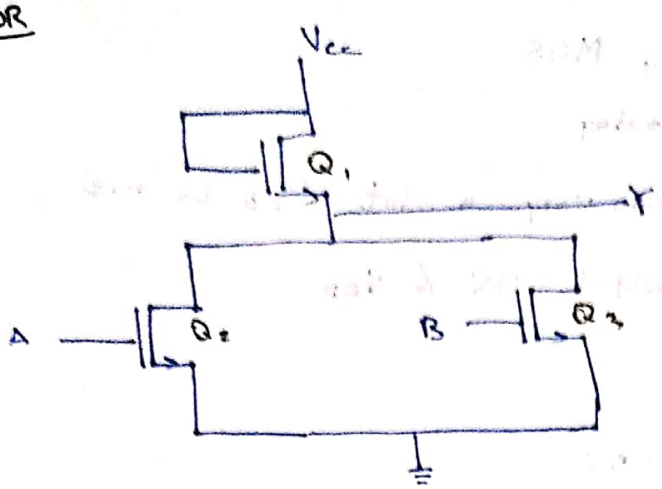
→ Q_1 is always ON

→ when B = 0, A = 5V
→ Y = high
→ Q_2 will not conduct

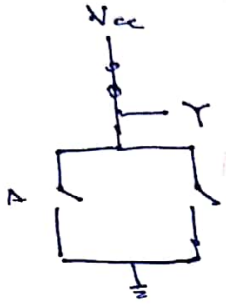
→ when B = 5V, A = 0V
→ Q_3 will not conduct
→ Y = high

→ when B = A = 5V
→ both Q_2 & Q_3 conduct
→ Y = low

NOR



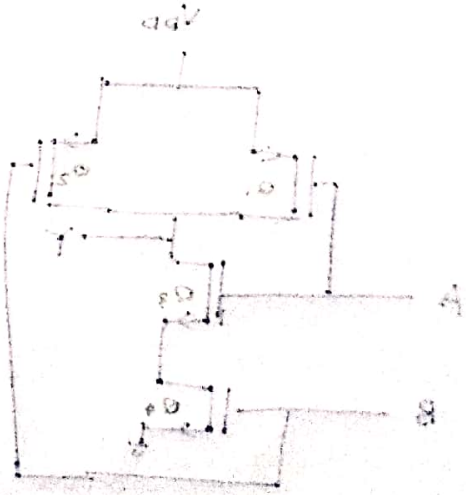
simple diagram



→ Q_1 always conduct

→ when any A or B, 0 → Y = high

→ when none A and B 0 → Y = low.



CMOS

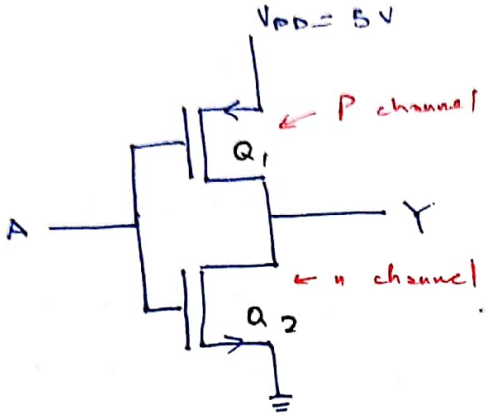
→ complementary MOS

→ widely used today

→ V_{cc} / V_{DD} can vary a lot (+3 to +18)

→ two output 0V & V_{DD}

Inverter



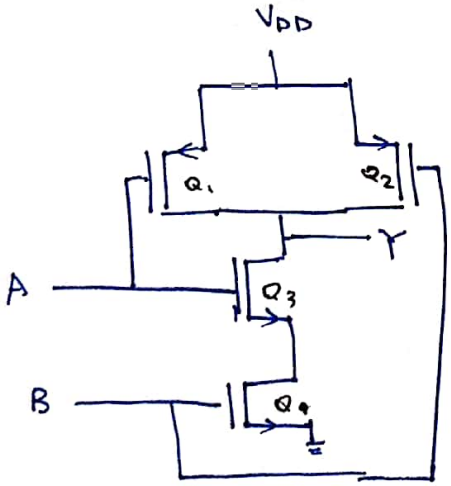
→ when A = 0

for Q_1 → $V_{GS} = -5$ → conduct

for Q_2 → $V_{GS} = 0$ V → not conduct

∴ Y = 1

NAND



when A = B = 0, Q_3, Q_4 conduct

Y = 1

When A = 5, B = 0

Q_1 ~~conduct~~ not

Q_2 ~~not~~ conduct

Q_3 conduct

Q_4 not

∴ Y = high

→ when A = B = 5

Q_1, Q_2 not Q_3 Q_4 conduct

∴ Y = 0

Tri-State TTL

→ As Totem-Pole can't handle wired input, we use Tri-State.

→ There are three possible output.

Y = 0, 1 or high impedance.

→ here Q_6, Q_7, Q_8 are open collector TTL and Q_1, Q_2, Q_3, Q_4, Q_5 are totem-pole TTL

→ When C=0 (0 V or 0.2 V)

→ Q_7 and Q_8 not flow, Q_6 flow

→ base ~~voltage~~ Q_6 → V_{BE} = conducting

~~V_{CC}~~ = voltage at the base of Q_6 = 0.9V (0.2+0.7)

(0.7 = conducting Threshold)

→ but for Q_8 to flow, we need at least 0.6×3 = 1.8 V, so Q_7 ∧ Q_8 not flow (similar to open collector TTL)

→ hence for D1 ~~5V~~ (something) ~~0V~~ high (can be ignored)

→ now when A = high

Q_2, Q_3 → on

Q_4 → off Y = low

→ now when A = low (0.2 V)

Y = high.

C=0, NOT gate

→ when C=1

→ Q_6 not flow, Q_7, Q_8 flow (saturation)

∴ D1 conducts

→ Q_1 conducts → Q_3 off

→ Q_5 base voltage = 0.9 V, but we need 0.6+2 V

C=1, high impedance ∴ Q_4 not conduct.

∴ Y is basically open (high impedance)

Circuit:

→ This circuit can be used for wire logic

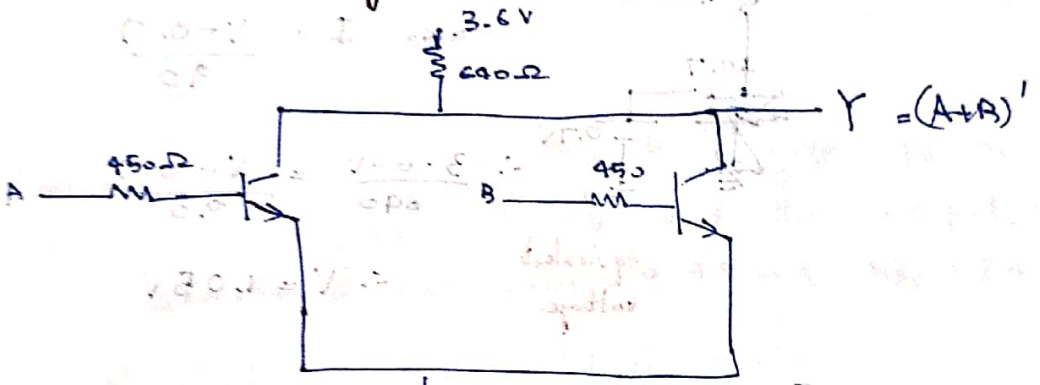
→ for Bus, all but one should be in high impedance and one should give data.

DTL question

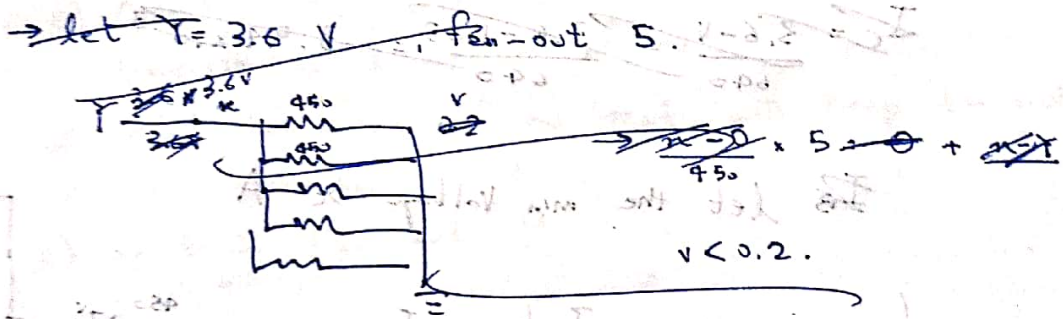
Q) a) Determine high-level output voltage of RTL for a fan-out of 5

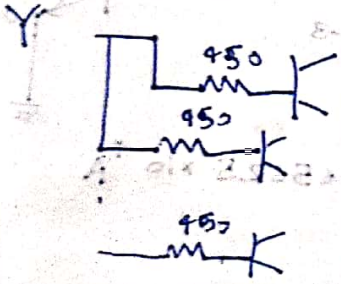
b) Determine the min. input voltage required to driven an RTL Transistor to saturation when $h_{FE} = 20$.

c) find noise margin, when input high and Fan out 5



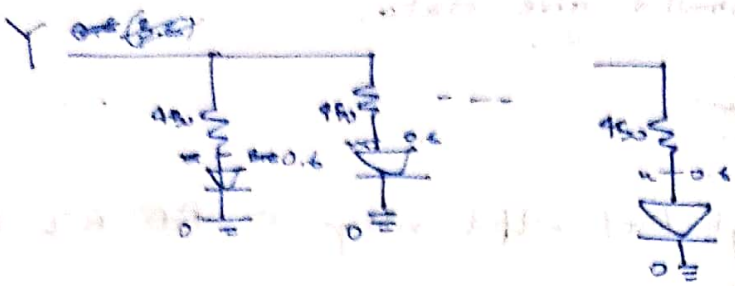
→ LOAD → similar inputs will be connected.



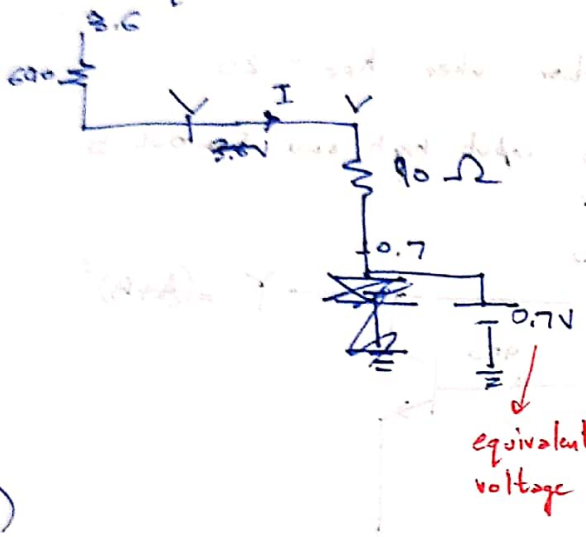


(5)

simplifying.



→ equivalent circuit



$I = \frac{3.6 - V}{640}$

also $I = \frac{V - 0.7}{90}$

$\therefore \frac{3.6 - V}{640} = \frac{V - 0.7}{90}$

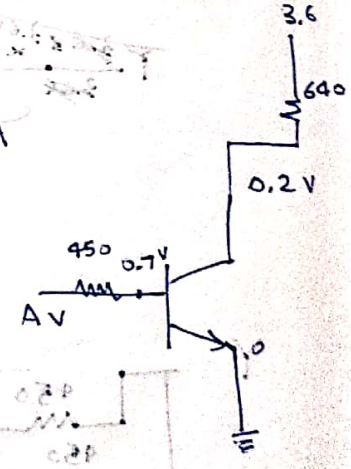
$\therefore V = 1.05\,V$

b)

Saturation: $I_B \geq \frac{I_C}{h_{FE}}$, $h_{FE} = 20$, $V_{CE} = 0.2V$

~~$I_C = \frac{3.6 - V}{640} = \frac{3.6 - 1.05}{640} = 3.98 \times 10^{-3}$~~

~~I_B~~ let the min voltage be A



for min. case $I_B h_{FE} = I_C$

$\therefore I_C = \frac{3.6 - 0.2}{640} = 5.31 \times 10^{-3}$

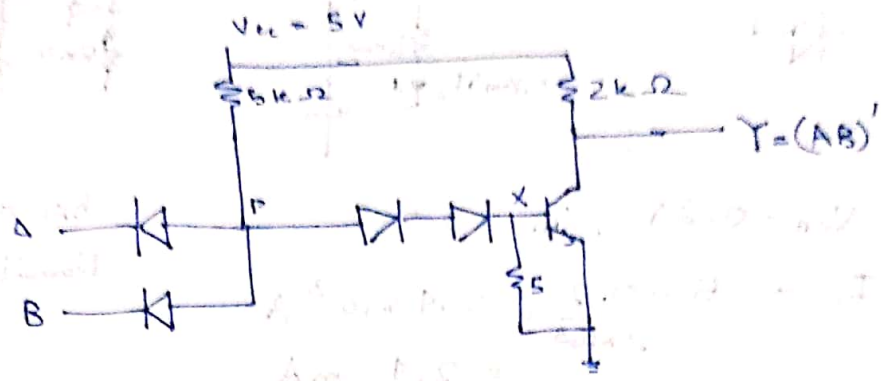
$\therefore I_B = \frac{I_C}{h_{FE}} = \frac{3.4}{640 \times 20} = 2.65625 \times 10^{-4}\,A$

now $I_B = \frac{A - 0.7}{450}$ $= \frac{3.4}{640 \times 20}$

$A = 0.7 + \frac{3.4 \times 450}{640 \times 20} = 0.819\,V$

c) Noise Margin = 1.05 V − 0.819 V
= 0.23 V.

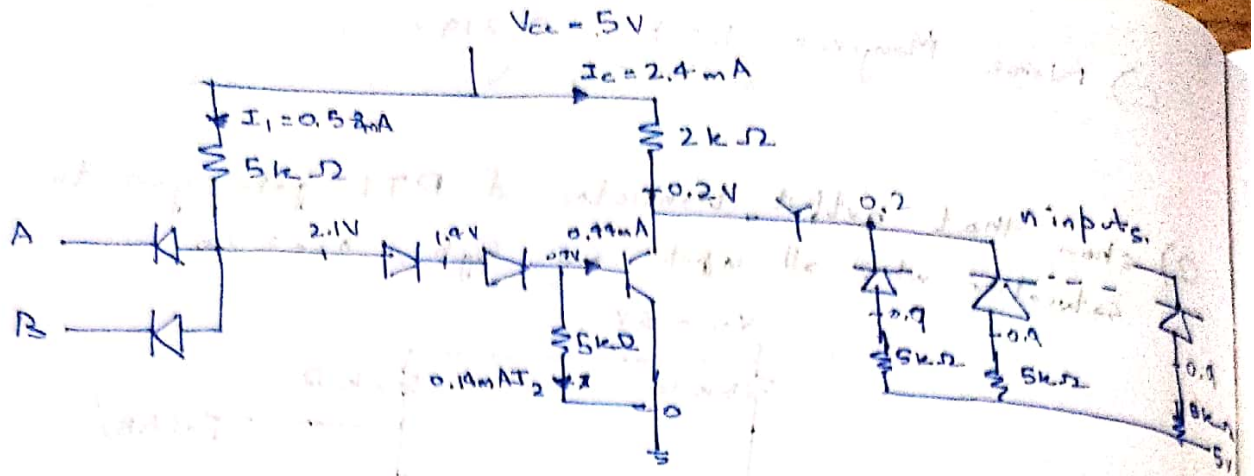
Q) show that output transistor of DTL gate goes to saturation, when all inputs are high, $h_{FE} = 20$.



when $A = B = 5V$, $V_P = 2.1V$, $V_x = 5 - 1.4V = 3.6V$
(done in Theory)

Q) Connect the output Y of the DTL gate to N inputs of other similar gate. Assume that the output transistor is saturated & base current = 0.44 mA. $h_{FE} = 20$

a) calculate current in 2kΩ resistor.
b) Calculate the current coming from each input connected to the gate
c) Calculate the total collector current in the output transistor as a function of N.
d) Find the value of N that will keep transistor in saturation
e) what is Fan-out ?



$V_{CE} = 0.2V$, $V_{BE} = 0.7V$; $h_{FE} = 20$

Transistor saturated

$$I_c = \frac{5 - 0.2}{2 \times 10^3} = 2.4 \times 10^{-3} A$$

$$= 2.4 \text{ mA}$$

$$I_1 = \frac{5 - 2.1}{5 \times 10^3} = 0.58 mA$$

$$I_2 = \frac{0.7}{5 \times 10^3} = 0.14 mA$$

$\rightarrow I_B = 0.58 - 0.14 = 0.44 mA$

b) when Y = 0, current coming from each gate

$$= \frac{5 - 0.9}{5k} = 0.82 mA$$

c) for n gates, $I_{tot} = 0.82 \times n$ mA

d) To keep in saturation $I_B h_{FE} \geq I_c$

$$I_c = 2.4mA + n \times 0.82 mA$$

$I_B = 0.44mA$, $h_{FE} = 20$

limit $\rightarrow$ $I_B h_{FE} \geq I_c$

$\therefore$ $0.44 \times 20 \geq 2.4 + n \times 0.82$

$n \leq 7.80$

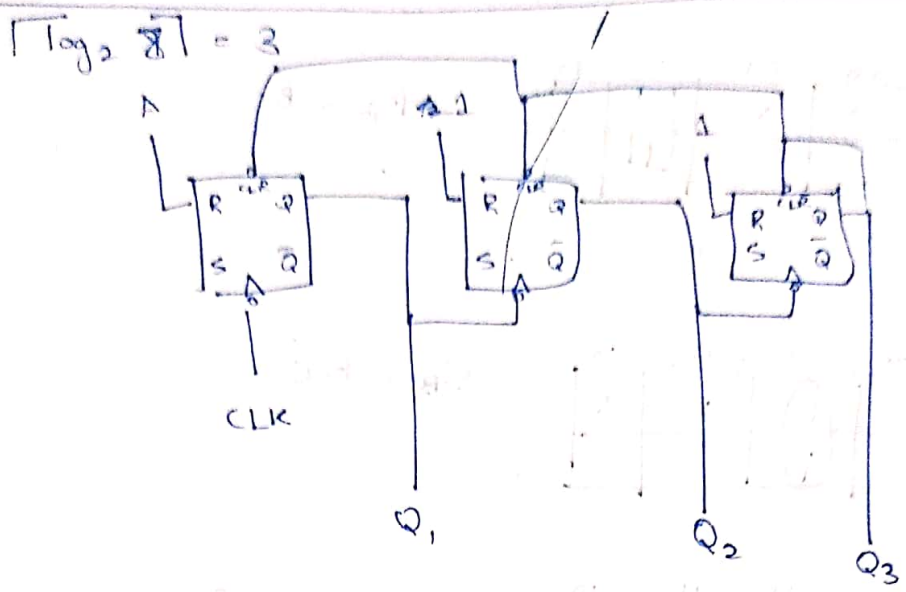
$\therefore$ n should be kept 7

e) Fan out = 7

Q6) b) Design a counter with following binary sequence:

0, 1, 3, 2, 6, 9, 5, 7 and repeat.

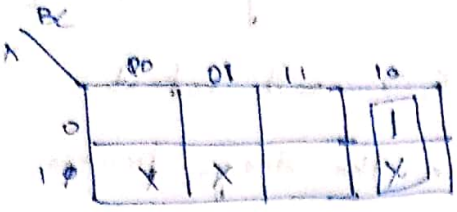
Use RS Flip Flop in the design process



(Gray code → binary could also be done)

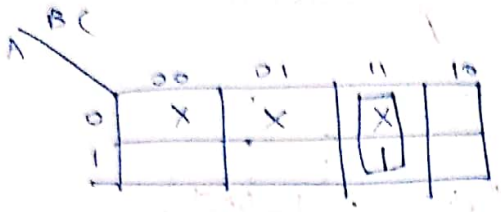
A	B	C	S_A	R_A	S_B	R_B	S_C	R_C
0	0	0	0	X	0	X	1	0
0	0	1	0	X	1	0	X	0
0	1	1	0	X	X	0	0	1
0	1	0	1	0	X	0	0	X
1	1	0	X	0	0	1	0	X
1	0	0	X	0	0	X	1	0
1	0	1	X	0	1	0	X	0
1	1	1	0	1	0	1	0	1

S_A



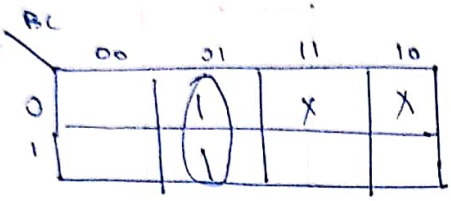
$S_A = B\bar{C}$

R_A



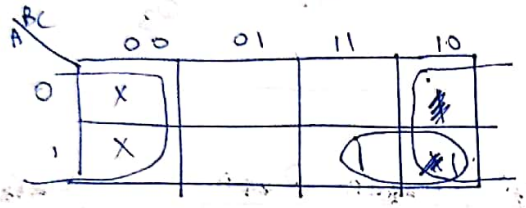
$R_A = \bar{B}C$

S_B



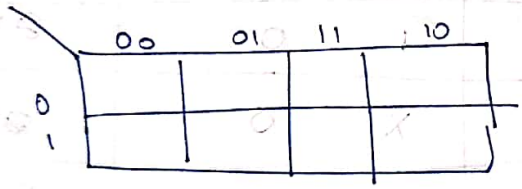
$S_B = \bar{B}C$

R_B



$R_B = A\bar{B}$

S_C



(complete)