1. Binary Number Systems

Camlin | Page
Date / /

⇒ Analog system is called as continuous systems
⇒ Digital system is called as discrete systems.

Decimal System :→ 0 → 9 (highest digit = base - 1) = 10 - 1

$(518.425)_{10} = 5\times10^2 + 1\times10^1 + 8\times10^0 + 4\times10^{-1} + 2\times10^{-2} + 5\times10^{-3}$

Octal system :→ (highest num = 8 - 1 = 7) 0 → 7

$(467)_8 = 4\times8^2 + 6\times8 + 7\times8^0$
$= 256 + 48 + 7$
$= (311)_{10}$

$(345)_8 = 3\times8^2 + 4\times8 + 5\times8^0$
$= 192 + 32 + 5$
$= (229)_{10}$

Hexadecimal system :→ (highest num = 16 - 1 = 15) 0 → 15

$(A)_{16} = (10)_{10}$ $(B)_{11} = (11)_{10}$ $(C)_{16} = (12)_{10}$ $(D)_{16} = (13)_{10}$ $(E)_{16} = (14)_{10}$ $(F)_{16} = (15)_{10}$

$(4AC2)_{16} = 4\times16^3 + A\times16^2 + C\times16^1 + 2\times16^0$
$= 16384 + 2560 + 192 + 2$
$= (19138)_{10}$

$(5C.D4)_{16} = 5\times16^1 + C\times16^0 + D\times16^{-1} + 4\times16^{-2}$
$= .$

Camlin Page
Date / /

Binary systems :→ (highest num = 2 - 1) 0, 1

→ 0, 1 are called bits

→ 8 bits are one byte

$$(11011)_2 = 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$$
$$= 16 + 8 + 2 + 1$$
$$= (27)_{10}$$

Generalized eqn :→

The General eqn for converting any number which is written in other systems into decimal

$$(a_5a_4a_3a_2a_1a_0)_r = a_5r^5 + a_4r^4 + a_3r^3 + a_1r^1 + a_0r^0 + a_1r^{-1} + a_2r^{-2} \ldots$$

8

Decimal to Binary :→

Find the LCM to the given number by 2.
Take remainders & write in reverse order.

Ex:

2 | 49	1
2 | 24	0
2 | 12	0
2 | 6	0
2 | 3	1
	1

$(49)_{10} = (110001)_2$

2 | 97	1
2 | 48	0
2 | 24	0
2 | 12	0
2 | 6	0
2 | 3	0
	1

$(97)_{10} = (1100001)_2$

$(84.63)_{10}$ =

Integer

2 | 84	0
2 | 42	0
2 | 21	0
2 | 10	1
2 | 5	1
2 | 2	0
	1

1011000.

$0.63 \times 2 = 1.26$

fraction part

1	0.26
0	0.52
1	0.04
0	0.08
0	0.16
0	0.32

$(84.63)_{10} = (1011000.101000)_2$

$(75.54)_{10}$ =

Integer

2 | 75	1
2 | 37	1
2 | 18	0
2 | 9	1
2 | 4	0
2 | 2	0
	1

1001011.

$0.54 \times 2 = 1.08$

fraction part

1	0.08
0	0.16
0	0.32
0	0.64
1	0.28

10001

$(75.54)_{10} = (1001011.10001)_2$

Converting from diff. bases to diff bases. using decimal

$(\quad)_n = (\quad)_m$

↓ convert to base 10 using eqn. → using LCM. convert from base 10 to m base.

$(8A3)_{16} = (\ ?\ )_8$

$8A3 = 8 \times 16^2 + A \times 16^1 + 3 \times 16^0$

$= 2048 + 160 + 3$

$(8A3)_{16} = (2211)_{10}$

$(2211)_{10} = (4243)_8$

8 | 2211 · 3
8 | 276 4
8 | 34 4
4

Camlin Page
Date / /

Interconversions without using decimal:

Hexadecimal to binary → 4 bits of all numbers

$(8\ A\ 3)_{16} = (1000\ 1010\ 0011)_2$
1000 1010 0011

$= (4\ 2\ 4\ 3)_8$

Binary to Octal :→ 3 bit combination

1) $(748)_8 = (111\ 100\ 011)$
111 100 011
$= (1E3)_{16}$

2) $(8A3.B2)_{16} = (1000\ 1010\ 0011 . 1011\ 0010)$ Logging zero
1000. 1010 0011 . 1011 0010
$= (4243.544)_8$

3) 1001 0011 1001 . 0001 1100 1000
$(4471.071)_8$
$(939.1C8)_{16}$

Camlin Page
Date / /

Bases ⇒	Decimal (10)	Binary (2)	Octal (8)	Hexadecimal (16)
	0	0000	0	0
	1	0001	1	1
	2	0010	2	2
	3	0011	3	3
	4	0100	4	4
	5	0101	5	5
	6	0110	6	6
	A	0111	7	7
	8	1000	10	8
	9	1001	11	9
	10	1010	12	A
	11	1011	13	B
	12	1100	14	C
	13	1101	15	D
	14	1110	16	E
	15	1111	17	F

Binary Arthematic :→

```
  1 1
  1010
+ 1011
------
 10101
```

2 → 10
3 → 11
4 → 100

```
  1 1 1
  10110
+ 01111
-------
 100101
```

```
 1010
 0111
 0111
```

```
  0 10
  0110
- 0101
------
  0001
```

```
  0 10
  0110
  0111
```

```
  0 101
  0100
  0111
------
  0101
```

```
   1010
 x  101
------
   1010
  0000+
 1010++
-------
 110010
```

→ Generally computers follow addition to subtract numbers i.e adding -ve numbers.
these are compliments.

16/18

Compliments :-

(i) Diminished Radix compliment :-> ($r-1$)'s compliment
The diminished Radix compliment of the num N with base r having n digits in it is expressed as
$(r^n - 1) - N$.

Decimal

$(10-1)$ 9's compliment
$r-1$

$(492)_{10}$ =

9's compliment $= (10^3 - 1) - 492$
$= 1000 - 493$
$= 507$

$(2845)_{10}$
$= 10^4 - 1 - 2845$
$= 10000 - 2848$
$= 7154$

Binary

$(2-1)$ 1st compliment

$(10101)_2$
$= (2^5 - 1) - 10101$
$= 31 - 10101$
$= 11111 - 10101$
$= 01010$

$(011011)_2$
$= 2^6 - 1 - 011011$
$= 63 - 011011$
$= 100100$
$= 111111 - 011011$
$= 100100$

Camlin Page
Date / /

Octal

8's complement

$(456)_8$

$= 321$

$= (8^3 - 1) - 456$

$= (511) - (456)_8$

$= (777)_8 - (456)_8$

$= 321$

Hexadecimal

15's complement.

$(5AB)_{16}$

$= 16^3 - 1 - 5AB$

$4096 - 1 - 5AB$

$= (1000 - 1) - 5AB$

$= FFF - 5AB$

$= A54$

4096 - 0
16 | 256 - 0
16 | 16 - 0
1

~~(ii) Diminished R.~~

(ii) Radix complement :→ (r's complement)

The Radix complement of the num N with base r having n digits in it is expressed as

$(r^n) - N$

Decimal

10's complement

$(243)_{10}$

10's compl. $= 10^3 - 243$

$= 757$

B.

Binary

2's complimat

$(1.00110)_2$

$= 2^6 - 100110$

$= 64 - 100110$

$= 1000000 - 100110$

$= 011010$

0111111_0
1000000
100110
011010

2|64 - 0
2|32 - 0
2|16 - 0
2|8 - 0
2|4 - 0
2|2 - 0
1

Octal

$(672)_8$

$= 8^3 - 6A2$

$= (512) - 6A2$

$= 1000 - 6A2$

$= 106$

Hexadecimal

$(4AD)_{16}$

$= 16^3 - 4AD$

$= 4096 - 4AD$

$= 1000 - 4AD$

16 | 4096
16 | 256 - 0
16 | 16 - 0
0 - 1

B53

$= B53$

Subtraction using radix compliment :-

$x - y = x + (-y)$

$= x + (r^n - y)$ carry neglected

$= x + (-y + r^n)$

$= x - y$

when $x > y$

$-y$ → converts to 2^k compliment and gets added to x

r^n → causes as carry so final carry is neglected so we get actual result

Ex: $492 - 342$

$= 492 + (-342)$

$342 = 657 + 1 = 658$

$= 492 + 1000 - 658$

$= 492 + 658$

$= 150$

$+1000 - 342$

492
658
1150

Camlin Page
Date / /

other eg.

342 - 492

342 + (-492)

342 + 508 = 850

= 850 → find each complement

150

342
508
850

849+1 = 150

Binary

Ex: 011100 + (-001010)

= 011100 + 110110

= 1010010

001010 - 011100

= 001010 + (-011100)

= 001010 + 100100

= 101110

= 010010

-001010

2's complement is

= 110101 + 1 = 110110

011100
110110
1010010

111111
011100
100011+1
100100

111111
011100
100100

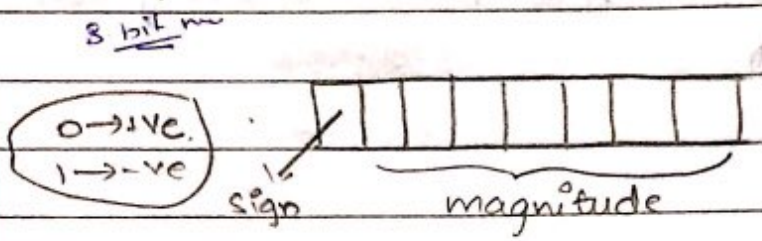
111111
101110
010001+1
010010

2 - 001010

(64)
1000000
001010
110110

Representation of signed binary numbers:-

(i) Signed magnitude method :→



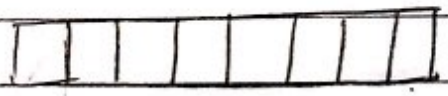
The other 7 bits have 2^7 combinatia i.e 128. for each sign. We can write -1 to -127

+1 to +127

-0/+0.

The maximum number which can be written is ±127

(ii) 1's compliment method :→



leading bits ← +ve ← 0

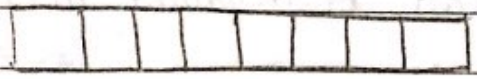
-ve ← 1

Here there is no -ve or +ve num. and bits are used as magnitude.

the -ve numbers are converted to 1's compliment

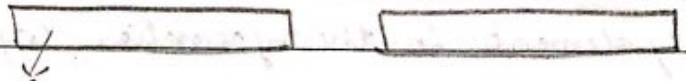
(ii) 2's complement method :→

-128 → 0 → +127



Examples

Sign extension :→



keep sign.

all 8 bits should be written as same for sign.

(most significant method is written)

8-bit number

Binary codes :→

→ Coding is a process of assigning a group of binary bits to represent multivalued items of information.

→ The assaingment of bit patterns to discrete elements of information is generally referred to as binary coding

→ With an n no of bits 2^n elements of information can be created.

→ for x no of elements in the information $\log_2^x$ nog of bits are required to represent.

(i) BCD Code :- It is also called 8421 code (Binary coded decimal)

Decimal digits	(4 bits) BCD bits 8 4 2 1
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Camlin Page
Date / /

25 in BCD is → 00100101

25 → 11001 → binary
→ 00100101 → BCD

12 34
0001 0010 0011 0100

~~Add~~ BCD Arthematic :-)

Addition →

4587	→	0100	0101	1000	0111
3253	→	0011	0010	0101	0011
7840		0111	1000	1110	1010
				0110	0110
				0100	0000

when result is beyond limit of 9. then add 6 to result. & will get actual result

4597
3953

add 6 when carry over addition on it

~~0485~~
~~0245~~

Subtraction :→

It is applied by changing 2nd num to 10's compliment and add it to actual num. (1st num)

0485	→	0485	This addition is done
−0245	→ (10's compli)	+ 9755	by BCD adding.
		(1)0240	

neglected

decimal

Other binary codes :→

flipping we get its 1's complement

self complementing code → weighted code

weighted code

Decimal	8421	2421	84-2-1
0	0000	0000	0000 0000
1	0001	0001	0111 0111
2	0010	0010/1000	0110 0110
3	0011	0011/1001	0100 0101
4	0100	0100/1010	1011 0100
5	0101	0101/1011	1010 1011
6	0110	1100/0110	1010 1010
7	0111	0111/1101	1001 1001
8	1000	1110	1000 1000
9	1001	1111	1111 1111

add 3 to BCD ← Excess 3-code :-

0	0011
1	0100
2	0101
3	0111
4	1000
5	1001
6	1010
7	1011
8	1100
9	

Inverting ← number in half & got 2nd half

so it is self complementing code

Gray code :→

⇒ to avoid switching of switches many times and to avoid confusion gray code is used.

⇒ It is used to minimize switching operations

⇒ It is reflective code

0	0000	0000
1	0001	0001
2	0011	0010
3	0010	0011
4	0110	0100
5	0111	0101
6	0101	0110
7	0100	0111
8	1100	1000
9	1101	1001
10	1111	1010
11	1110	1011
12	1010	1100
13	1011	1101
14	1001	1110
15	1000	1111

27/06/18

Binary → gray code :-

→ The 1st bit of any gray code is same as their binary.

B: 0 1 0 0

G: 0

→ Then compare 2 bits if equal put 0, unequal put 1.

B: 0 1 0 0

G: 0 1

B: 0 1 0 0

G: 0 1 1

B: 0 1 0 0

G: 0 1 1 0

Gray → Binary code :-

→ The 1st bit is same.

G: 0 1 1 0

B: 0

→ Compare 2nd gray bit and 1st binary bit.

if equal = 0, unequal = 1.

G: 0 1 1 0

B: 0 1

→ Repeat the same.

G: 0 1 1 0

B 0 1 0 0

ASCII Code :→

→ American standard code for Informatia Interchage

→ It is 7-bit code

→ $2^7 = 128$ combination can be used.

↓

94 → printable characters

34 → non-printable.

94 → 26+26 (upper & lower case) → 52 english characters

→ 0-9 → 10 digits.

→ 32 → other printable characters.

ASCII Codes

0 → 30H

1 → 31H = 011 0001

9 → 39H = 011 1001

A → Z : 41H ↔ 5A H

a → z : 61H → 7A H

⇒ later this code is extended to 8-bits.

we get 2^8 combination = 256

0 → 30H = 0011 0001

9 → 39H = 0011 1001

⇒ Keeping 1 in the 1st position they are alloted to other signs.

Camlin Page
Date / /

Error detection code :→

→ while sending the code there might be some errors or disturbances in between. then the error detection code is used.

→ All the digits in the code are added. i.e called parity (1st num added)

Ex: 31 H — even parity — if no of one's are odd, one is added to make it even parity

011 0001 — P 1011 0001

odd parity — P 0011 0001

011 0001 — even. P 0011 1001

odd. P 1011 1001

→ If a number is transmitted.

For odd parity

all no of one's are odd. including the parity number.

For even parity

all no of ones are even including the parity number.

→ If there are even parity for odd (or) vice versa error is occured. (It identifies error)

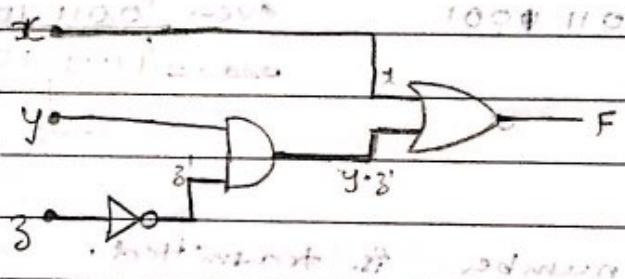
→ But it can identify a single bit error.

(i) $F = x + yz'$

OR AND

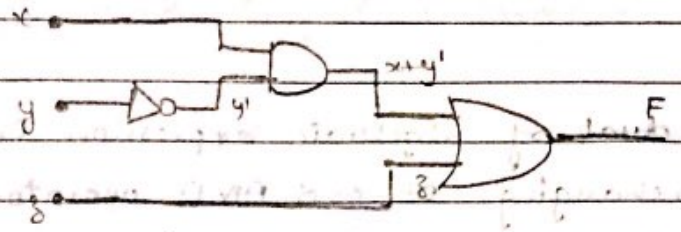
Truth Tables :-

x	y	z	z'	$y \cdot z'$	F
0	0	0	1	0	0
0	1	0	1	1	1
1	0	0	1	0	1
0	0	1	0	0	0
1	1	0	1	1	1
1	0	1	0	0	1
0	1	1	0	0	0
1	1	1	0	0	1



(ii) $F = (x + y') \cdot z$

y'	x	y	z	$x + y'$	F
1	0	0	0	1	0
1	1	0	0	1	0
0	0	1	0	0	0
1	0	0	1	1	1
0	1	1	0	1	0
1	1	0	1	1	1
0	0	1	1	0	0
0	1	1	1	1	1



Canonical and Standard forms :-

Boolean Algebra :-

These are the algebraic structures defined by a set of elements B together with 2 binary operators (+) and (·) provided that the following postulates are satisfied.

(i) The structure is closed w.r.t the operators. These are '+' and (·) [OR and AND]

(ii) The element zero is an identity element w.r.t OR (+) and the element '1' is an identity element w.r.t AND

$$\forall x: \quad x + 0 = x$$
$$x \cdot 1 = x$$

(iii) The structure is commutative w.r.t OR (+) and AND (·)

(iv) The operation OR and AND are distributive one on another.

(v) For every x belongs to B there exist an element x' which belongs to B such that

$$x + x' = 1$$
$$x \cdot x' = 0$$

(vi) There exists 2 elements x, y belonging to B such that $x \neq y$

~~(vii)~~

The dual of algebraic expression is obtained by interchanging OR and AND operators and replacing 1's by 0's and 0's by 1's.

4 Postulates of Boolean Algebra:-

(i) $x + 0 = x$ $\quad x \cdot 1 = x$

(ii) $x + x' = 1$ $\quad x \cdot x' = 0$

(iii) $x + y = y + x$ $\quad x \cdot y = y \cdot x$

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

(v)

Theorems of Boolean Algebra :-

(i) $x + x = x$ $\quad x \cdot x = x$

(ii) $x + 1 = 1$ $\quad x \cdot 0 = 0$

(iii) $(x')' = x$ (Involution).

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

(v) a $(x+y)' = x' \cdot y'$, b $(x \cdot y)' = x' + y'$

(vi) $x + xy = x$, $x(x+y) = x$

$x(1+y) = x \cdot 1$

$1 + 1 = 1$

$0 \cdot 0 = 0$

$x(1+1)$

$x \cdot$

$0 \cdot 0 = 0$

$1 \cdot 1 = 1$

Cannonical and Standard forms:-

Cannonical forms are used to represent a complex function.

These are 2 types (i) min. terms
(ii) max terms.

It consists of literals (variables x, y, z)

Min terms:-

0 → x' 1 → x.

min terms are sum of product of individual variables

Ex: x	y	z	min term	
0	0	0	m_0	$x'y'z'$
0	0	1	m_1	$x'y'z$
0	1	0	m_2	$x'yz'$
0	1	1	m_3	$x'yz$
1	0	0	m_4	$xy'z'$
1	0	1	m_5	$xy'z$
1	1	0	m_6	xyz'
1	1	1	m_7	xyz

$(x + y') \cdot z$. $F = m_1 + m_5 + m_7 = \Sigma(m_1, m_5, m_7)$

→ Add the min terms whose result is 1.

(ii) Max terms :→

$x \rightarrow 0 \qquad x' \rightarrow 1$

⇒ These are addition of individual terms.

Ex:

x	y	z	max terms
0	0	0	$M_0 (x+y+z)$
0	0	1	$M_1 (x+y+z')$
0	1	0	$M_2 (x+y'+z)$
0	1	1	$M_3 (x+y'+z')$
1	0	0	$M_4 (x'+y+z)$
1	0	1	$M_5 (x'+y+z')$
0	1	0	$M_6 (x'+y'+z)$
1	1	1	$M_7 (x'+y'+z')$

$(x+y').z$

$F = M_0 M_1 M_2 = \Pi (M_0, M_1, M_2)$

→ A compliment of min term is its corresponding max terms.

Standard to canonical conversion (without Truth Table)

$F = x + yz'$

$= x \cdot 1 \cdot 1 + 1 \cdot y \cdot z'$

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

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

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

$F = m_7 + m_6 + m_5 + m_4 + m_2$

$F = x + yz'$

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

$= (x+y+0)(x+0+z')$

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

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

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

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

$= M_0 M_1 M_3$

Camlin Page
Date / /

Reducing the function :-

(i) $F = xyz' + xy' + xyz$

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

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

$= x(y + y')$

$F = x$

(ii) $F = \Sigma(0, 2, 4, 5, 6)$

$= \Sigma m_0 m_2 m_4 m_5 m_6$

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

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

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

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

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

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

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

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

$= xy' + xz' + x'z'$

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

$= xy' + z'$

(OR)

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

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

$= z' + xy'z$

$= (z' + z)(z' + xy')$

$= xy' + z'$

$F = x + yz'$

x	y	z	F	min terms	Max terms
0	0	0	0	m_0	M_0
0	0	1	0	m_1	M_1
0	1	0	1	m_2	M_2
0	1	1	0	m_3	M_3
1	0	0	1	m_4	M_4
1	0	1	1	m_5	M_5
1	1	0	1	m_6	M_6
1	1	1	1	m_7	M_7

$F = m_2 + m_4 + m_5 + m_6 + m_7$

$= \Sigma(2, 4, 5, 6, 7)$

$= M_0 M_1 M_3$

$= \Pi(0, 1, 3)$

To get compliment

$F' = m_1 + m_3 + m_0$

$= \Sigma(1, 3, 0)$

$= \Pi(2, 4, 5, 6, 7)$

$F' = M_2 M_4 \cdot M_5 \cdot M_6 \cdot M_7$

By using demorgan's ~~laws~~ theorm.

$F = m_2 + m_4 + m_5 + m_6 + m_7$

$F' = (m_2 + m_4 + m_5 + m_6 + m_7)'$

$= m_2' \cdot m_4' \cdot m_5' \cdot m_6' \cdot m_7'$

$= M_2 M_4 M_5 M_6 M_7$

2. Gate-level Minimization

Camlin Page
Date / /

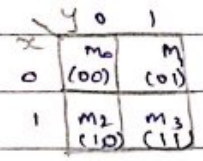
→ The no of gates are reduced in designing circuits. This is gate-level minimization.

→ Using boolean algebra also function can be reduced but it is complicated.

→ Other method to reduce gates and reduce complexicity is Karna map ie K'map.

→ K map

2 Variable K' map :->



normal method

$F = \Sigma(0,2)$

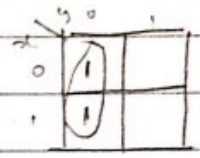
$= m_0 + m_2$

$= x'y' + xy'$

$= y'(x'+x)$

$= y'$

K'map



$F = y'$

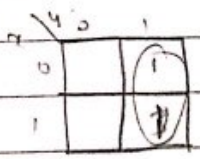
→ K map follows adjacency rules, put 1 at min term places.

→ combine two adj numbers who are same.

→ eliminate the variable who is chaging its value among the combined squares

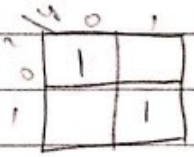
→ then. if the variable num and 1 are same then put the remaing squares have value 0 put x' if 1 put x. /// 'y' to "y" i.e

Eg:. $F = \Sigma(1,3)$



$F = y$

$F = \Sigma(0,3)$



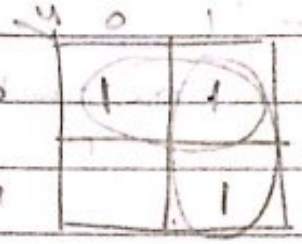
$F = x'y' + xy$

→ XNOR & XOR cannot be reduced.

Camlin Page
Date / /

$F = (0, 1, 3)$

$= m_0 \; m_1 \; m_3$

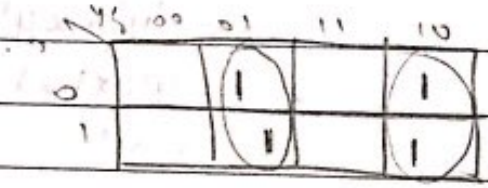


$F = x' + y$

3 variable K'map:- It follows gray code.

yz / x	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

$F = \Sigma(1, 2, 5, 6)$

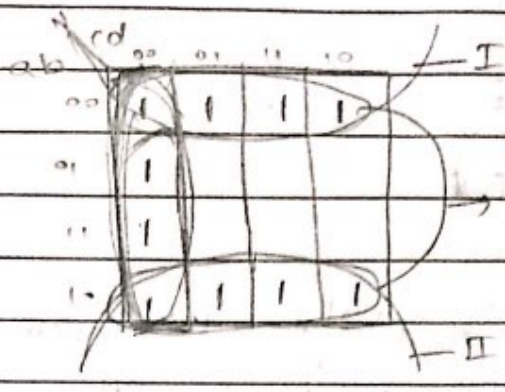


$F = y'z + yz'$

4-variable K'map:→

ab \ cd	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

$F = (0, 1, 2, 4, 8, 9, 10, 11, 12)$

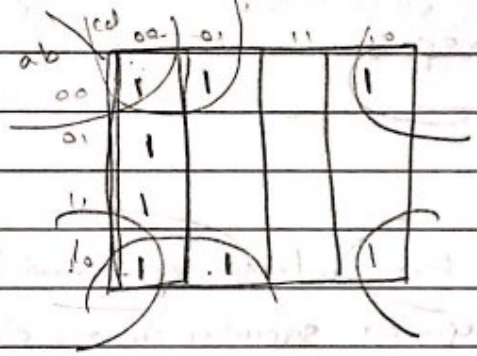


→ extreme left and right and top and bottom squares are adjacent.

$F = c'd' + b'$

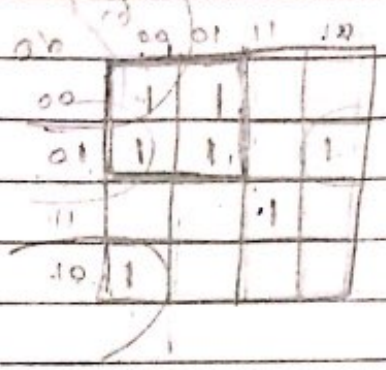
→ If a square having 1 is left out without any connection then take the mid term function exactly:

$F = \Sigma(0, 1, 2, 4, 8, 9, 10, 12)$



$F = c' * b' + c'd' + d' * b'$

$F = b'c' + c'd' + b'd'$



~~$F = c'a' + c'd' + a'b + abcd$~~

$F = a'c' + b'c'd' + a'bd' + abcd$

$= a'c' + ac)(bd' + b'c'd' + a'bd'$

$= (a'c' + ac)(a'c' + bd) + d'(b'c' + a'b)$

$=$

Implicant :->

It is the product term formed by combining adjacent squares in the k' map.

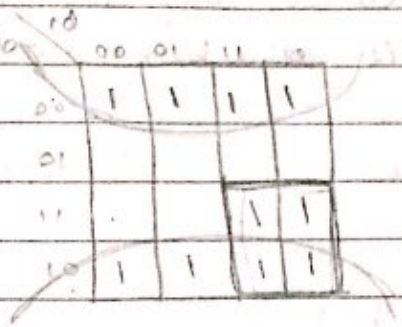
Prime Implicant :->

It is the product term formed by combining the maximum possible no of adjacent squares in the k' map.

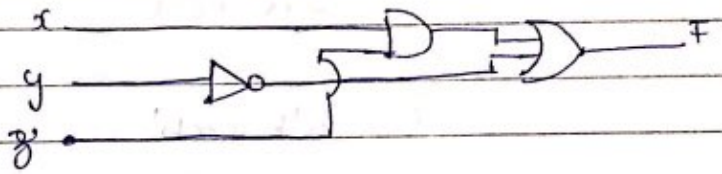
Essential prime Implicant :->

The Essential prime Implicant is the prime Implicant in which ~~atleast there exist~~ there exist one min term which is not covered by other prime Implicants.

$F(x,y,z,w) = \Sigma(0,1,2,3,8,9,10,11,14,15)$



$F = b' + ac = y' + xz$

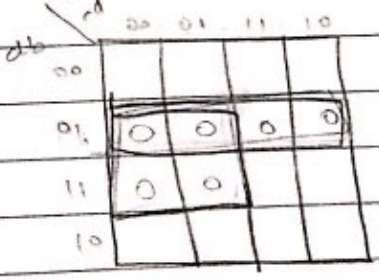


→ In sum of products 1st → AND 2nd → OR

→ In product of sum 1st → OR 2nd → AND

Reduce the function $F(x,y,z,w) = \Sigma(0,1,2,3,8,9,10,11,14,15)$ & to product of sums.

$F = \Sigma(0,1,2,8,9,10,11,14,15)$



$\bar{F} = a'b + c'b$

$\bar{F} = x'y + yz'$

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

$F = (\bar{F})'$

$= [y \cdot (x' + z')]'$

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

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

$F = y' + xz.$

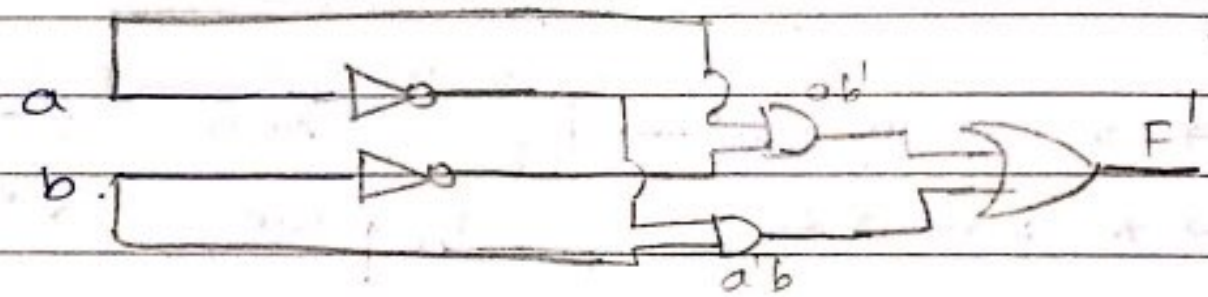
Camlin Page
Date / /

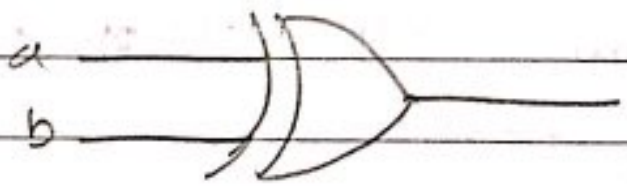
Q. For the function $F(a,b,c,d) = \Sigma(0,1,2,3,12,13,14,15)$ get the reduced exp for $\bar{F}$ using K'map and implimat it using basic gates

F =

ab \ cd	00	01	11	10
00	1			
01	0	0	0	0
11				
10	0	0	0	0

$\bar{F} = a'b + ab'$





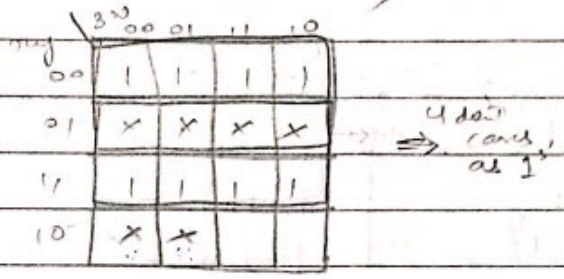
Camlin | Page
Date / /

Don't care functions:→

In some functions some terms will not be cared.

In Binary to BCD conversion numbers after 9 are don't cares.

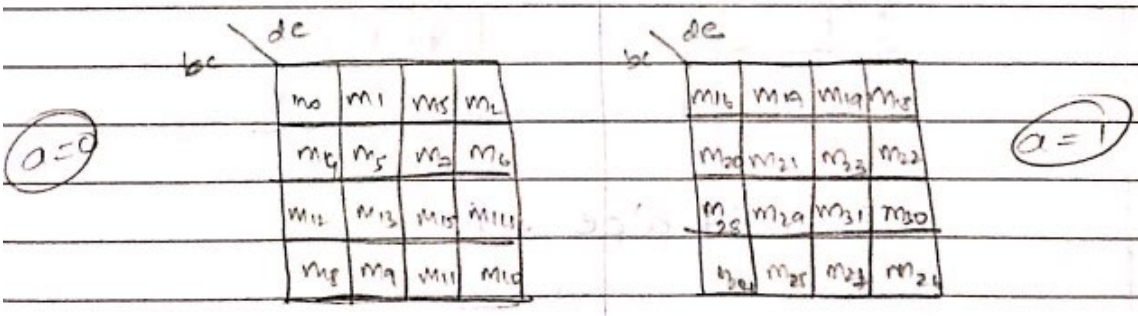
Q. $\phi = \Sigma(4,5,6,7,9)$ $F = \Sigma(0,1,2,3,12,13,14,15)$



$F = x' + y$

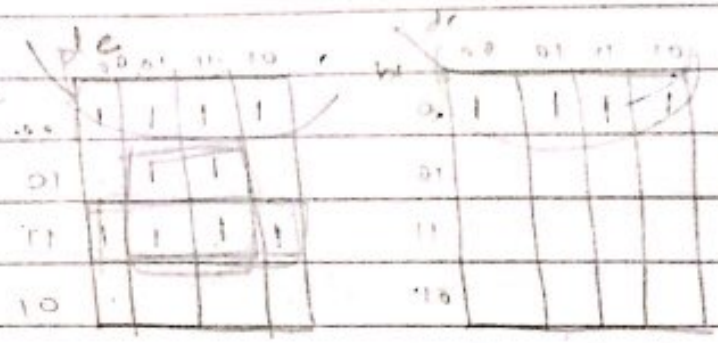
5-Variable k'map:→

A 5-variable k'map consists of 2 → 4 variable k'maps, The MSB is zero for one 4 variable k'map and it is 1 for other 4 variable k'map.



All the adjacency rules that are applicable for 4 variable k'map are also applicable in case of 5 variable k'map along with them. each square in 4-variable k'map is adjacent to the corresponding square in the other 4 variable k'map.

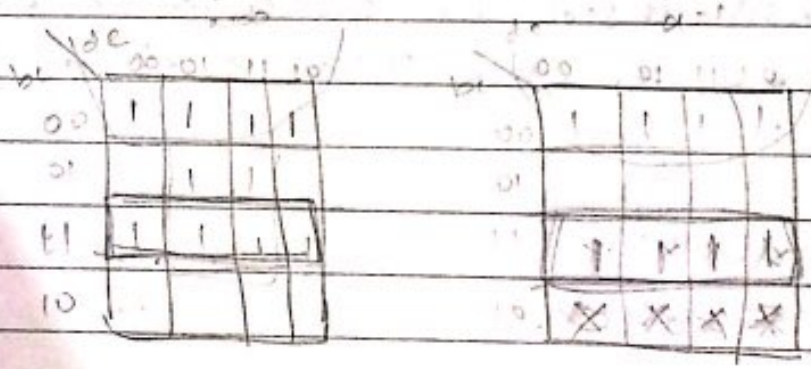
$F(a,b,c,d,e) = \Sigma(0,1,2,5,5,7,12,15,14,15,16,17,18,19)$



$F = b'c' + a'bc + d'ee$

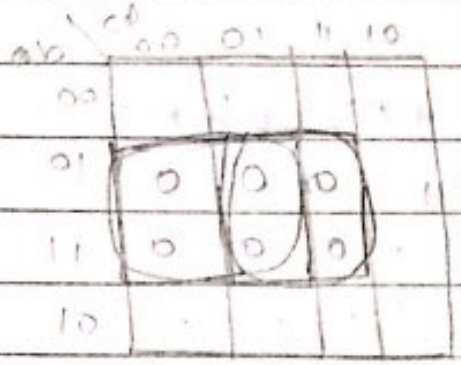
Reduce F if it contains don't care ϕ

$\phi = \Sigma(24, 25, 26, 27, 28, 29, 30, 31)$



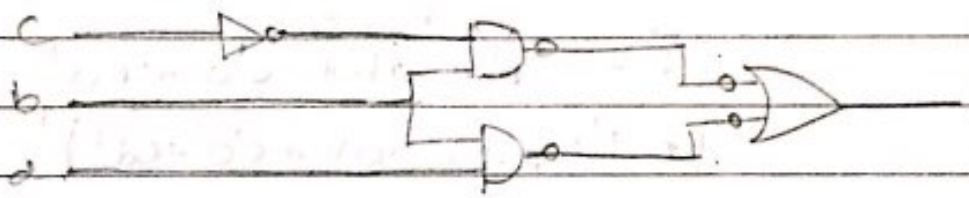
$F = b'c' + a'ce + bc$

Implement the compliment of the func" $F(a,b,c,d) = \Sigma(1,2,3,6,8,9,10,11,14)$ using NAND gates

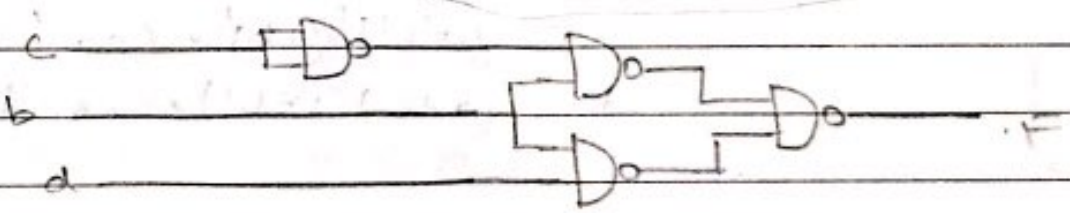


$\bar{F} = c'b + bd$

$\bar{\bar{F}} = bc' + bd$



NOT OR → AND NOT



Don't care functions:→

In some functions some terms will not be cared.

In Binary to BCD conversion numbers after 9 are don't cares.

Q. $\phi = \Sigma(4,5,6,7,9)$ $F = \Sigma(0,1,2,3,12,13,14,15)$

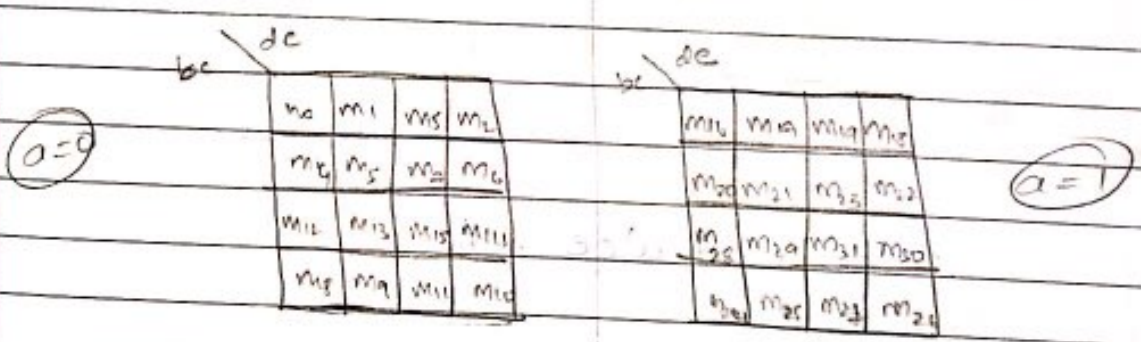
yz \ wx	00	01	11	10
00	1	1	1	1
01	x	x	x	x
11	1	1	1	1
10	x	x		

⇒ 4 don't cares as 1

$F = x' + y$

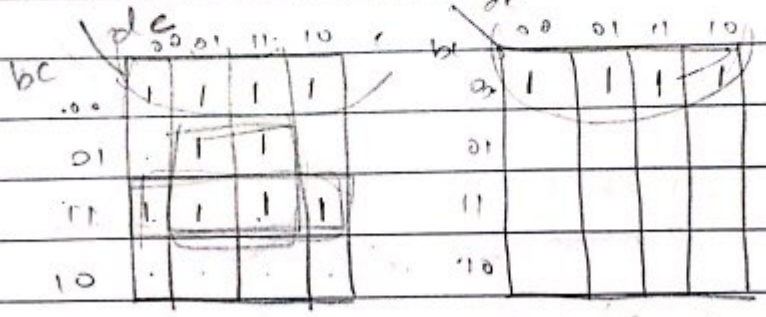
5-Variable k'map:→

Rule 1 A 5-variable k'map consists of 2 → 4 Variable k'maps, The MSB is zero for one 4 variable k'map and it is 1 for other 4 Variable k'map.



Rule 2 All the adjacency rules that are applicable for 4 variable k'map are also applicable in case of 5 variable k'map along with them. each square in 1-4 variable k'map is adjacent to the corresponding square in the other 4 variable k'map.

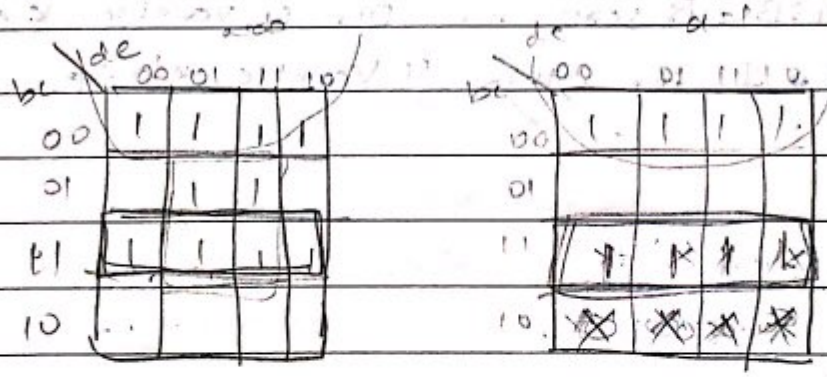
Q. $F(a,b,c,d,e) = \Sigma(0,1,2,5,5,7,12,13,14,15,16,17,18,19)$



$$F = b'c' + a'bc + d'ce$$

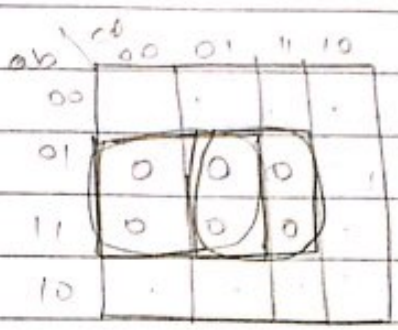
Reduce F if it contains don't care ϕ

$$\phi = \Sigma(24, 25, 26, 27, 28, 29, 30, 31)$$



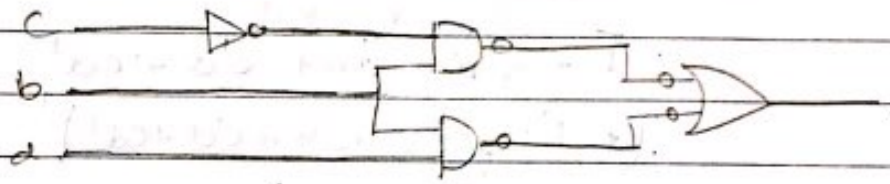
$$F = b'c' + a'ce + bc$$

Implement the compliment of the fun^{n} $F(a,b,c,d) = \Sigma(1,2,3,6,8,9,10,11,14)$ using NAND gates

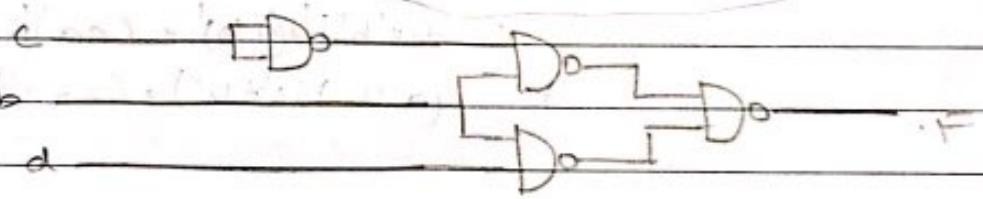


$\bar{F} = c'b + bd$

$\bar{\bar{F}} = bc' \cdot bd$ =



NOT OF → AND NOT



Implement the func^n using NOR gates

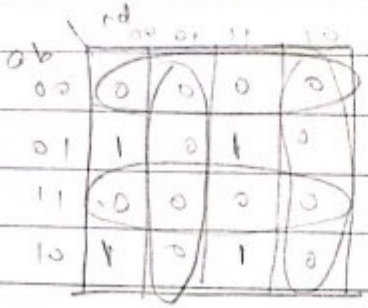
$F = (A \oplus B)(C \oplus D)'$

$= (AB' + A'B)(A + B')(A' + B)$
$(C' + D)(D' + C)$

$= (AB' + A'B)(C'D' + CD)$

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

$F = m_8 + m_4 + m_{11} + m_7$



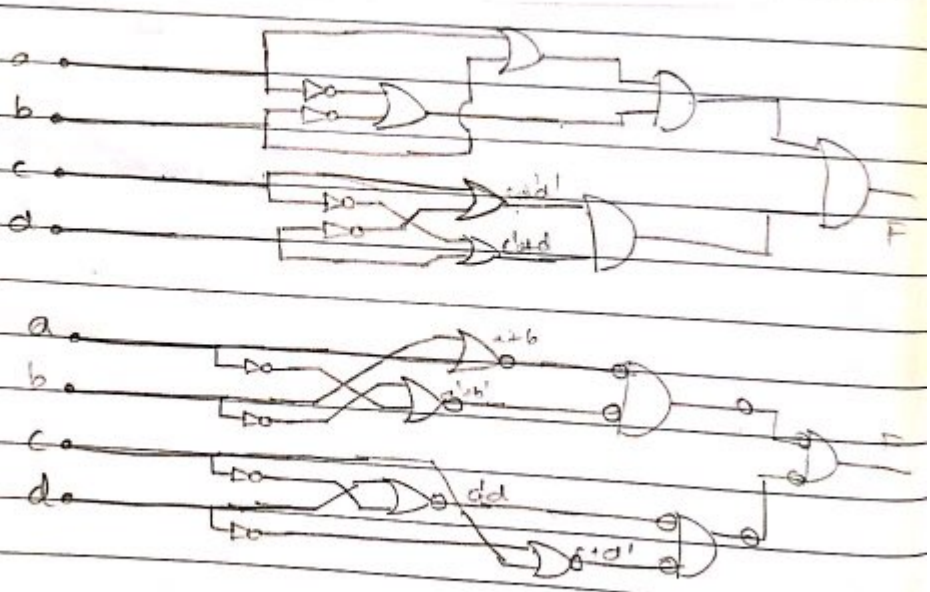
$\overline{F} = a'b' + ab + c'd + cd'$

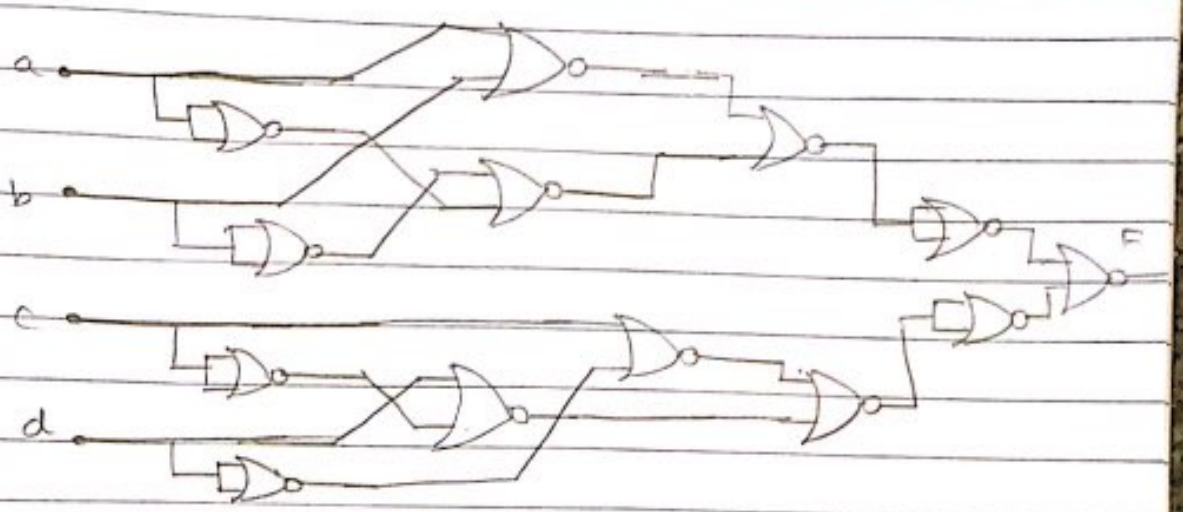
$(F')' = (a'b' + ab + c'd + cd')'$

$= (a'b' + ab)' \cdot (c'd + cd')'$

$= (a'b')' \cdot (ab)' \cdot (c'd)' \cdot (cd')'$

$F = (a+b)(a'+b') \cdot (c+d')(c'+d)$





12/12/18

XOR properties:-

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

(i) $X \oplus 0 = x$

(ii) $x \oplus 1 = x'$

(iii) $x \oplus x = 0$

(iv) $x \oplus x' = 1$

(v) $(x \oplus y') = (x' \oplus y) = (x \oplus y)'$

(vi) $A \oplus B = B \oplus A$

(vii) $A \oplus (B \oplus C) = (A \oplus B) \oplus C$.

→ A multiple variable XOR operation is defined as odd function

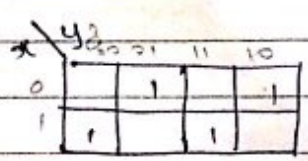
→ An n variable XOR func^n is an odd func^n defined as the logical sum of $\frac{2^n}{2}$ terms whose binary values have an odd no of 1's.

$F(x, y, z)$

$m_0 \rightarrow m_7$ no of 1's

0 → 000 — 0
1 → 001 — 1 → odd
2 → 010 — 1 → odd
3 → 011 — 2
4 → 100 — 1 → odd
5 → 101 — 2
6 → 110 — 2
7 → 111 — 3. → odd.

$F = \Sigma(1, 2, 4, 7)$ odd fun^{n}.

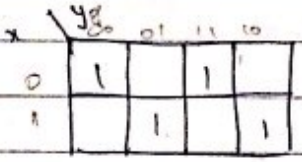


$F \Rightarrow xy'z' + x'y'z + xyz + x'yz'$

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

$x \oplus (y \oplus z)$

$F = \Sigma(0, 3, 5, 6)$ even fun^{n}.



$F = (x \oplus y \oplus z)'$

→ The function which have odd num of 1's in binary representation of their min term is called odd fun^{n}.

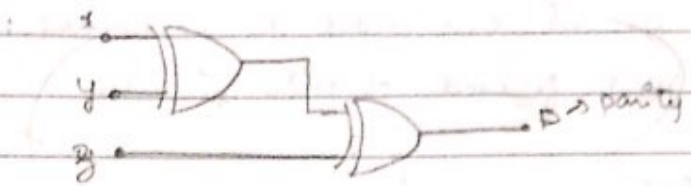
→ Ily for even function

→ Odd and Even functions can be Implemented by XOR and XNOR gates.

Application of XOR:-)

Parity Generator:-) It generates parity at transmitter

Even 3-bit 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	1	1

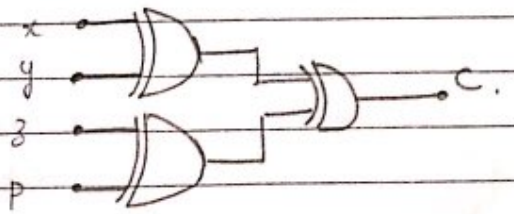
The parity generator gives
output - 1 if (odd num of 1's)
output - 0 if (even num of 1's)

Even parity checker:->

At the recieving end. It checks the parity for error detection.

yz \ P	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

$C = x \oplus y \oplus z \oplus P.$



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

The parity checker can be used for generation and checking of parity

If we ground one input in parity checker its eqn is used as parity generation

$P = 0$.

$x \oplus 0 = x$.

$x \oplus y \oplus z \oplus P = x \oplus y \oplus z \oplus 0$

$F = x \oplus y \oplus z$.

Digital Circuits :→

These are 2 types

→ combinational

→ sequential.

Combinational :→

The o/p depends on the present i/p.

Sequential :→

→ The o/p depends not only depend on present i/p but also on past i/p.

→ There is a storage element which is a memory element which stores past i/p.

(i) Combinational circuits.

Analysis procedure :->

(i) The 1st step is to make sure that the given circuit is combinational or not.

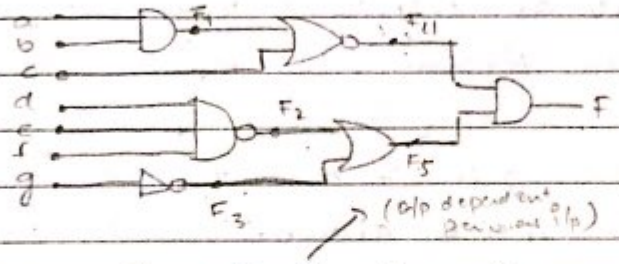
(ii) label all the gate o/p that are function of i/p variables. and determine the boolean function for each gate o/p.

(iii) label the o/p's of the gates that are function of i/p variables and previously labelled gates and determine the boolean function of these gates.

(iv) repeat the above process until the final o/p of the circuit are obtained.

(v) By repeated substitution of previously defined function the final o/p boolean function can be obtained. in terms of i/p variables.

Problem.



→ As there is no feedback or memory element it is a combinational circuit.

→ we name the intermediate outputs.

→ find intermediate functions in terms of i/p

$F_1 = a \cdot b$

$F_2 = (abc)' (d \cdot e \cdot f)'$

$F_3 = g'$

$F_4 = (F_1 + c)'$

$F_5 = F_2 + F_3$

$F = F_4 \cdot F_5$

$F_4 = (a \cdot b + c)'$

$F_5 = (d \cdot e \cdot f)' + g'$

$F = (ab + c)' \left[(def)' + g'\right]$

$= (ab)' \cdot c' \left[(def)' + g'\right]$

$= (a' + b')(c') \left[d' + e' + f' + g'\right]$

$= (ab)' \cdot c' \cdot (defg)'$

$\boxed{F = (ab + c + defg)'}$

Designing Procedure:-

(i) The design starts from converting the design objective into concrete specifications

(ii) From specifications required no of inputs and outputs are to be determined and a symbol is attached to each of them.

(iii) Truth table that defines the relationship between the inputs and outputs is to be determined.

(iv) Simplified boolean function for each output as a function of i/p variables is to be obtained.

(v) appropriate gates are to be chosen and the logic diagram is to be drawn.

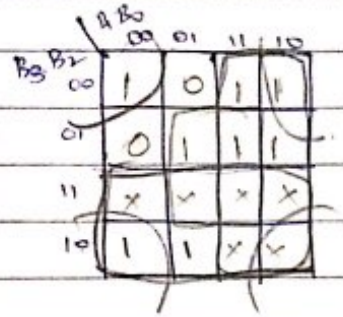
(vi) Finally the correctness of the logic diagram is to be verified.

Code converter:-

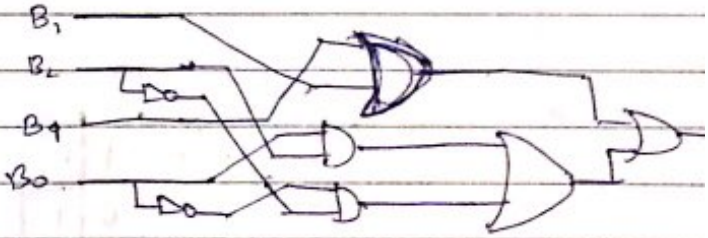
Ex: BCD to 7 segment display decoder.

BCD	7 segment display inputs							
$B_3 B_2 B_1 B_0$	a	b	c	d	e	f	g	
0 0 0 0	1	1	1	1	1	1	0	0
0 0 0 1	0	1	1	0	0	0	0	1
0 0 1 0	1	1	0	1	1	0	1	2
0 0 1 1	1	1	1	1	0	0	1	3
0 1 0 0	0	1	1	0	0	1	1	4
0 1 0 1	1	0	1	1	0	1	1	5
0 1 1 0	1	0	1	1	1	1	1	6
0 1 1 1	1	1	1	0	0	0	0	7
1 0 0 0	1	1	1	1	1	1	1	8
1 0 0 1	1	1	1	1	0	1	1	9

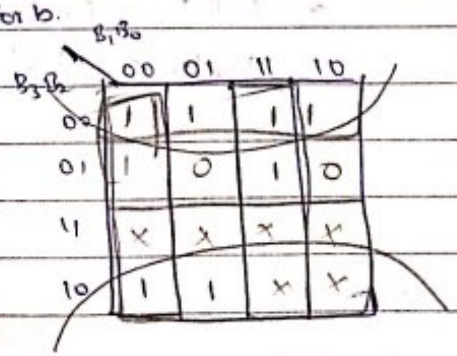
for a.



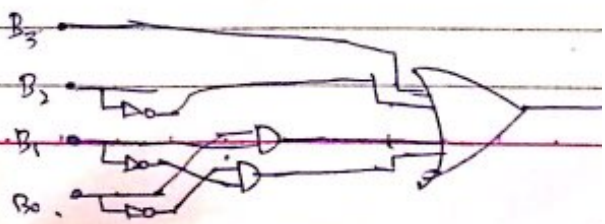
$a = B_0'B_2' + B_1 + B_3 + B_0 B_2$



for b.



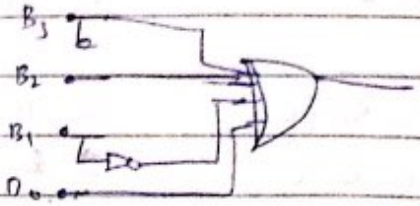
$b = B_1'B_0' + B_1B_0 + B_2' + B_3$



for c

B_3B_2 \ B_1B_0	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	x	x	x	x
10	1	1	x	x

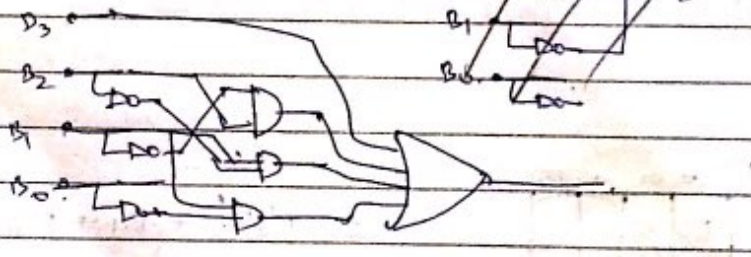
$C = B_1' + B_0 + B_2 + B_3$



for g

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	x	x	x	x
10	1	1	x	x

$g = B_0'B_1 + B_3'B_2'B_1 + B_1'B_2 + B_3B_1'$

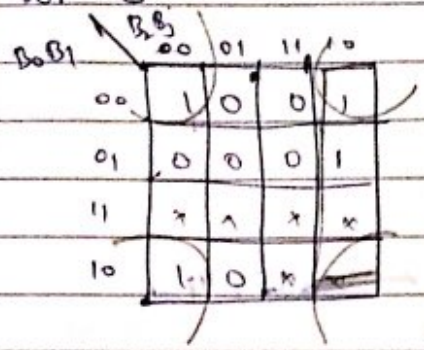


for d.

B_0B_1 \ B_2B_3	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	x	x	x	x
10	1	1	x	x

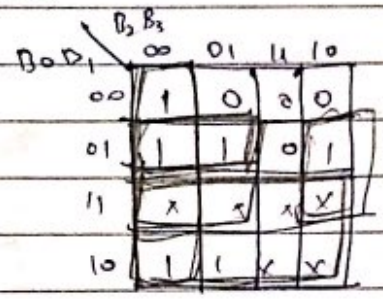
$d = B_0'B_1'B_2 + B_2B_3' + B_0B_1 + B_1B_3' + B_1B_2'B_3$

for e



$e = B_2 B_3' + B_1' B_3'$

for f



$f = B_2' B_3' + B_0 + B_1 B_2' + B_1 B_2 B_3'$

Adder – Subtractor :→

4- bit binary adder :→

A = a_3 a_2 a_1 a_0

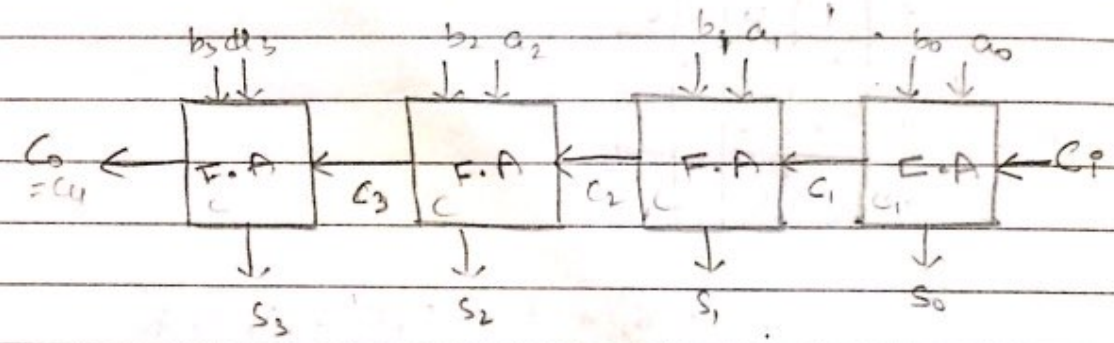
B = b_3 b_2 b_1 b_0 ← Ci : Carry i/p

C_0 S_3 S_2 S_1 S_0

↓

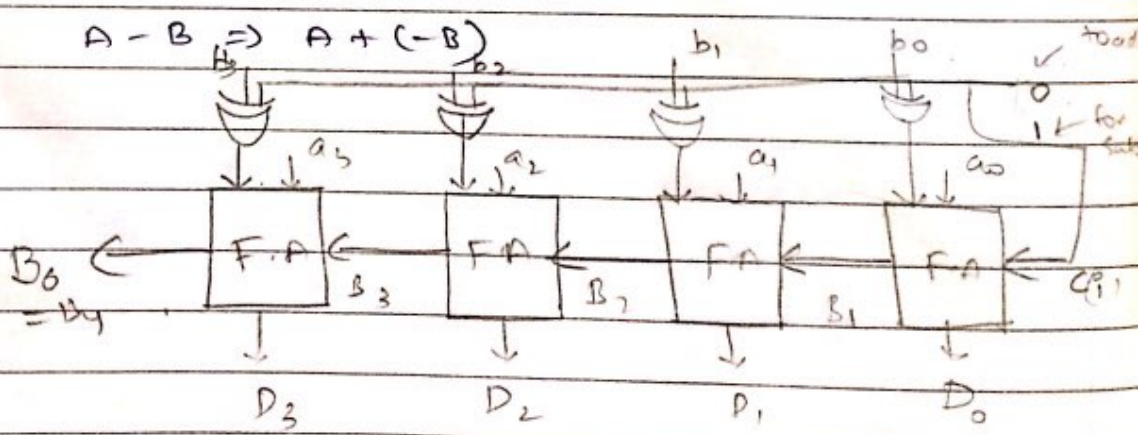
Carry output

9 inputs → 5 outputs



4-bit binay subtractor :→

A – B ⇒ A + (–B)



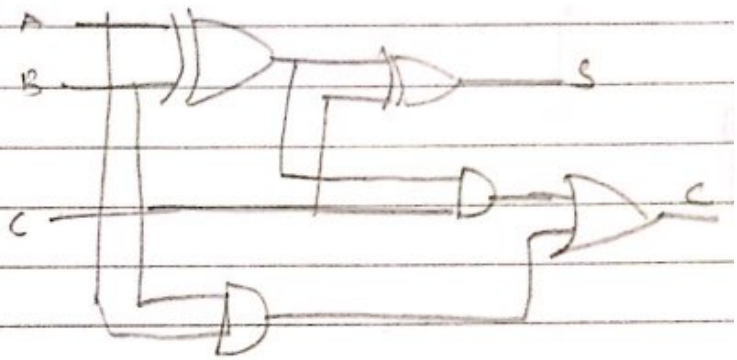
Generally it gets delay to get the o/p carry to avoid this there is another adder

Carry look ahead adder:-

In full adder

$S_i = A_i \oplus B_i \oplus C_i$

$C_{i+1} = (A_i \oplus B_i) C_i + A_i \cdot B_i$



$P_i = A_i \oplus B$

↳ carry propogater

$G_i = A_i \cdot B_i$

↳ carry generate

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

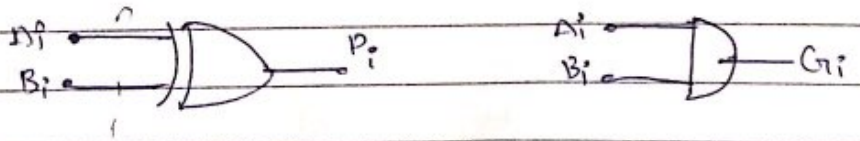
$C_1 = G_0 + P_0 C_0$

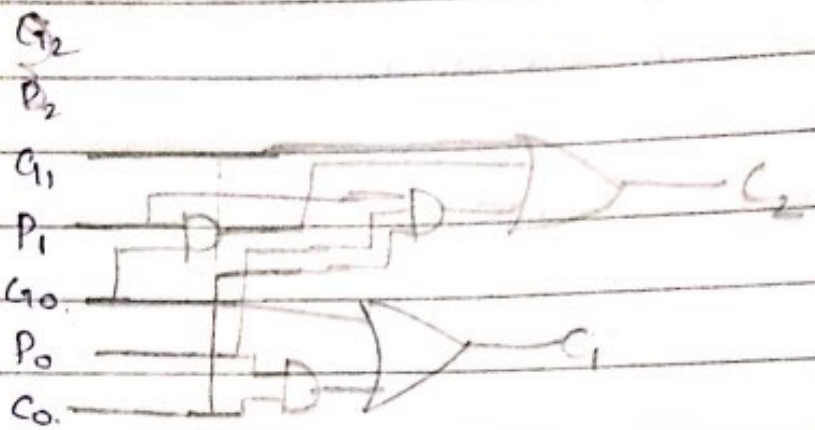
$C_2 = G_1 + P_1 C_1$

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

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

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

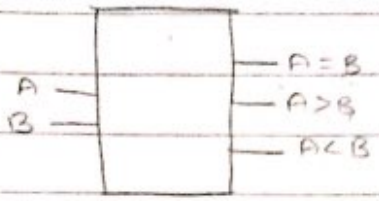




Magnitude comparator :→

4-bit → 8 i/p → 4 bit bits

It checks or compares the i/p values



$x \oplus x = 0$

$(x \oplus x)' = 1$

for checking equality of i/p we use XNOR gate

$x_i = (A_i \oplus B_i)' = A_i B_i + A_i' B_i'$

$A = A_3 \; A_2 \; A_1 \; A_0$

$B = B_3 \; B_2 \; B_1 \; B_0$

$x_3 x_2 x_1 x_0 = 1$ for ~~all bits of~~ $A = B$.

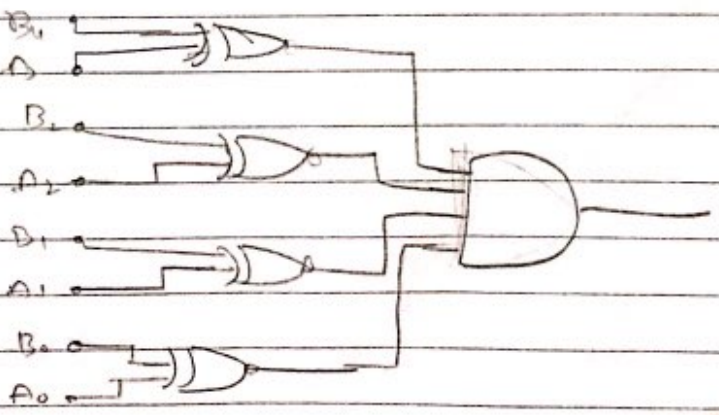
for $(A > B)$

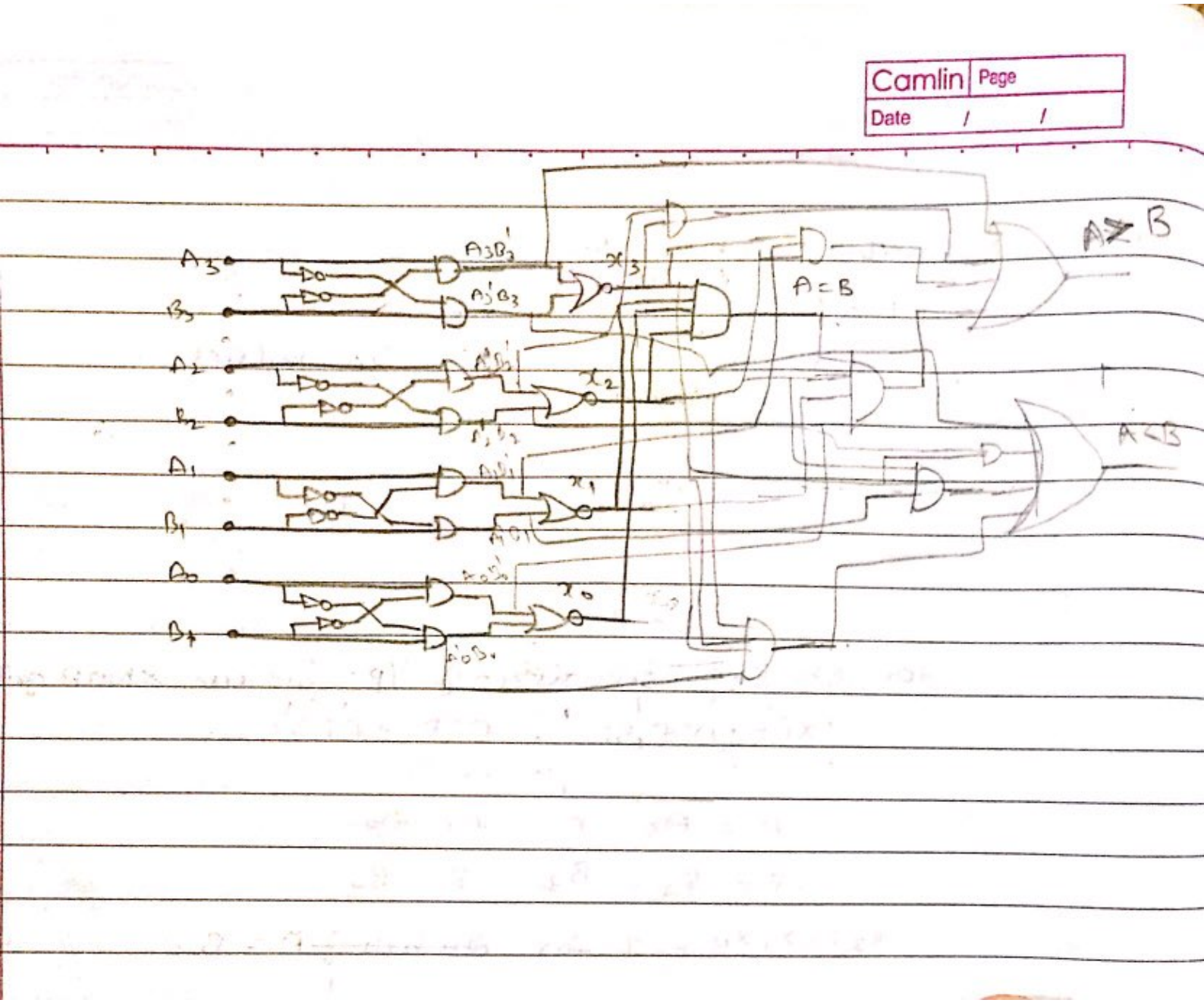
$= A_3 B_3' + x_3 A_2 B_2' + x_2 x_3 A_1 B_1' + x_3 x_2 x_1 A_0 B_0'$

for $(A < B)$

$= A_3' B_3 + x_3 A_2' B_2 + x_3 x_2 A_1' B_1 + x_3 x_2 x_1 A_0' B_0$

$(A = B)$

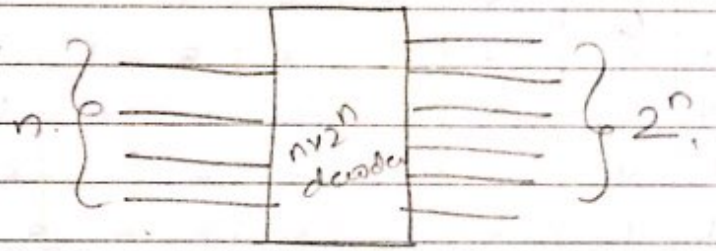




Decoder:→ $(n \times 2^n)$

→ Decoder is a logic circuit that converts binary information from n coded inputs to a maximum of 2^n unique outputs.

→ It simply changes a code into set of signals.



2x4 decoder:→

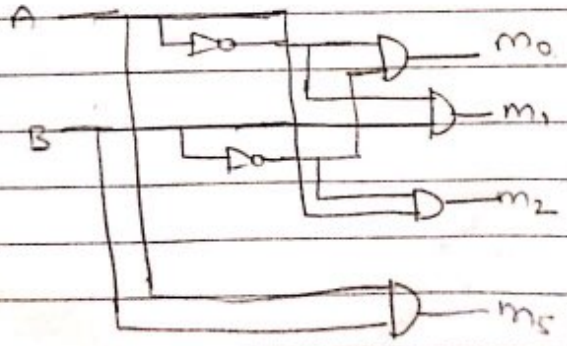
A	B	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

$D_0 = A'B' = m_0$

$D_1 = A'B = m_1$

$D_2 = AB' = m_2$

$D_3 = AB = m_3$



3×8 decoder :-

A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

$D_0 = A'B'C'$

$D_1 = A'B'C$

$D_2 = A'BC'$

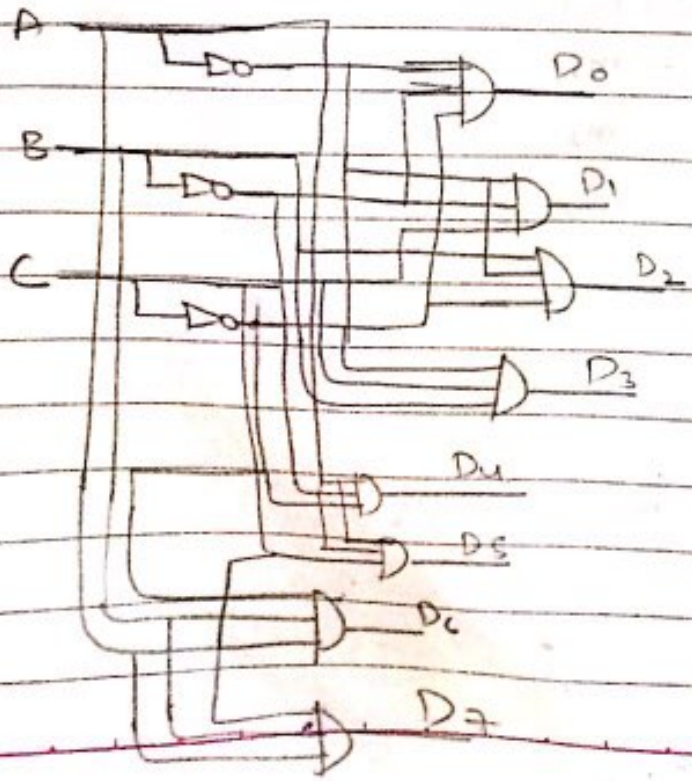
$D_3 = A'BC$

$D_4 = AB'C'$

$D_5 = AB'C$

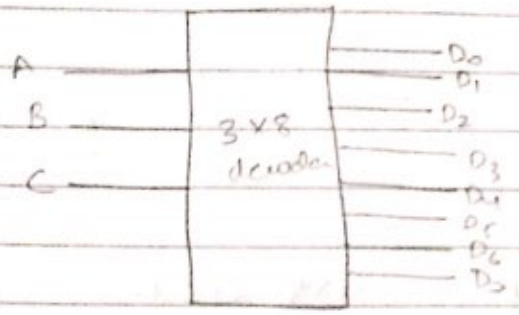
$D_6 = ABC'$

$D_7 = ABC$.

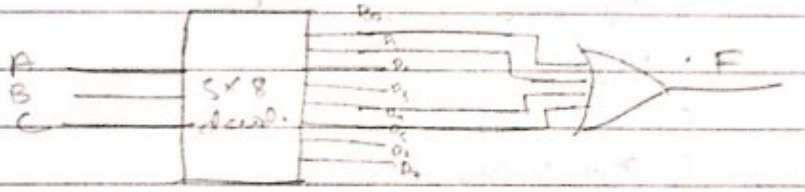


Camlin | Page
Date / /

Implimenting a fun^{n} using decoder:-



Implement the fun^{n} $F = \Sigma(0,1,4,5)$ using 3 to 8 decoder.

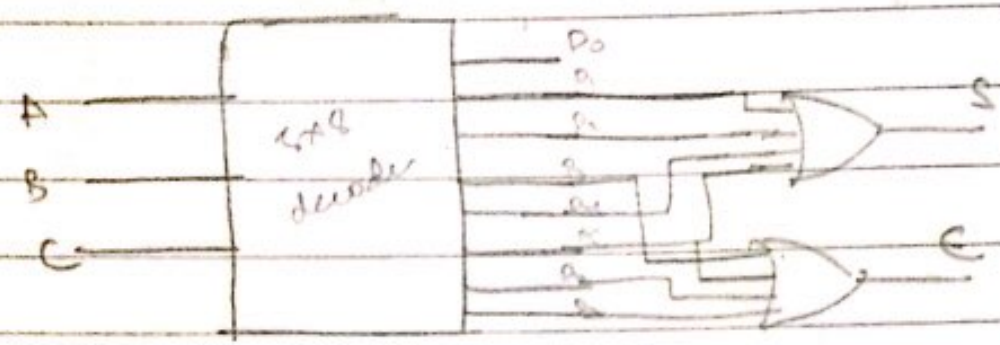


Implement a full adder using 3x8 decoder:-

x	y	z	s	c
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

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

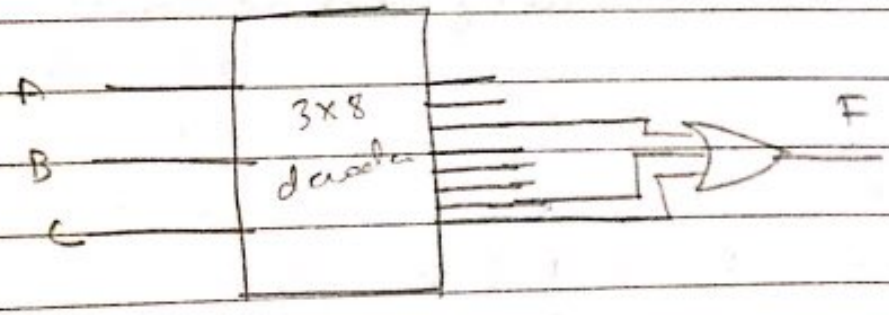
$carry = \Sigma(3,5,6,7)$



→ If 4 digits exist the 4x8 decoder.

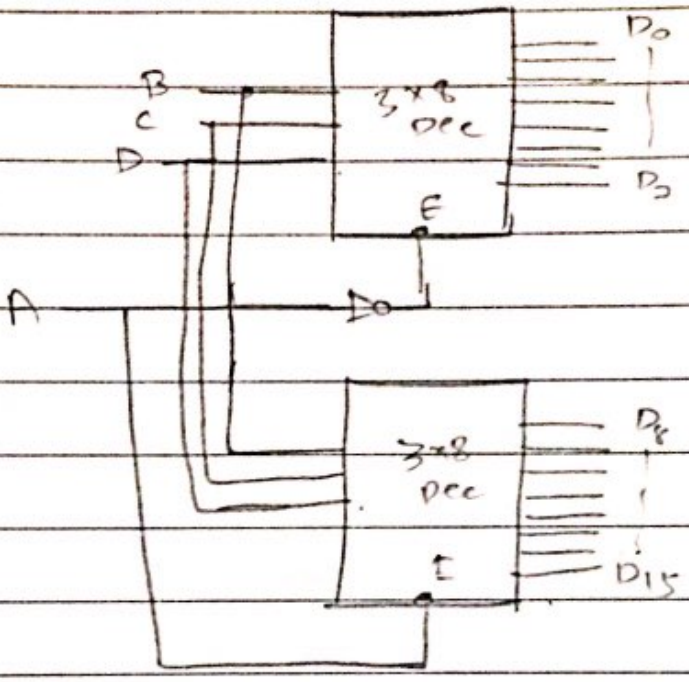
Q10. $F = xy + yz'$ Implimed using 3x8 decode.

$$F = xy(z+z') + yz'(x+x')$$
$$= xyz + xyz' + xyz' + x'yz'$$
$$= xyz + xyz' + x'yz'$$
$$= m_7 + m_6 + m_2$$
$$F = \Sigma(2,6,7)$$



Camlin | Page
Date / /

4x16 decoder using 3x8 decoder:-



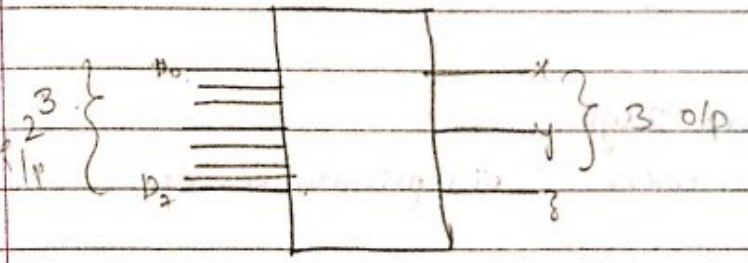
D_0	D_1	D_2	D_3	D_4 D_5 D_6 D_7 D_8 - - - -	D_{15}
1	0	0	0		0
0	1	0	0		0
0	0	1	0		0
0	0	0	1		0
0	0	0	0		1

Encoder :- $(2^n \times n)$

→ Encoder is a combinational circuit that generates a specified code at its o/p such as binary or BCD ..

→ It changes set of signals into a code.

8 × 3 Encoder :-

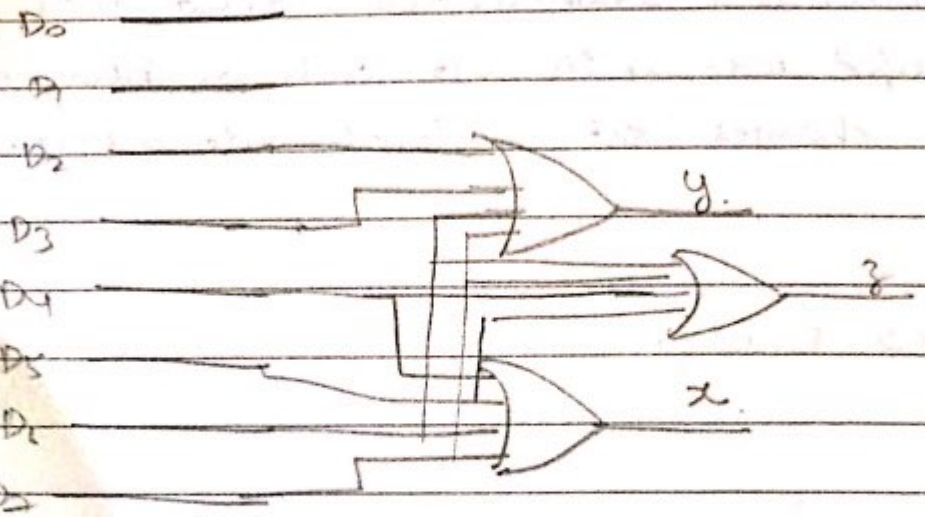


D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$x = \Sigma(4, 5, 6, 7)$

$y = \Sigma(2, 3, 6, 7)$

$z = (1, 3, 5, 7)$



→ Encoders are 2 types

(i) Binary encoder (ii) priority encoder

→ Binary encoder is a multi i/p combinational logic circuit that converts the logic level '1' data at its i/p into an equivalent binary code at its o/p.

Limitations:→

(i) If we give more than one i/p at same time, it confuses to give o/p.

Eg:

For D0 if we give 1 x, y, z are zeroes

0 x, y, z are zeroes

⇒ we get 2 same o/p for 2 diff i/p.

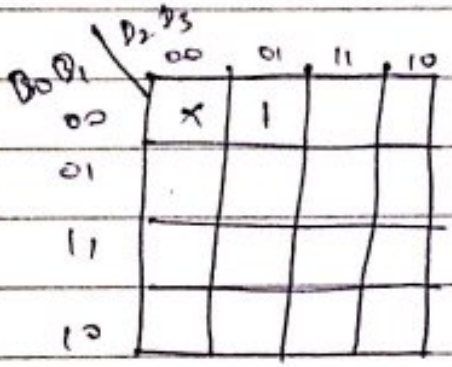
→ The operation of priority coder solves this problem.

Priority encoder :->

This encoder only considers the High Value of the i/p having highest priority. The rest of all i/p with lowest priority are lowered or ignored.

4X2 priority encoder :->

D_0	D_1	D_2	D_3	x	y	V → i/p's which are high
0	0	0	0	x'	x	0
1	0	0	0	0	0	1
x	1	0	0	0	1	1
x	x	1	0	1	0	1
x	x	x	1	1	1	1

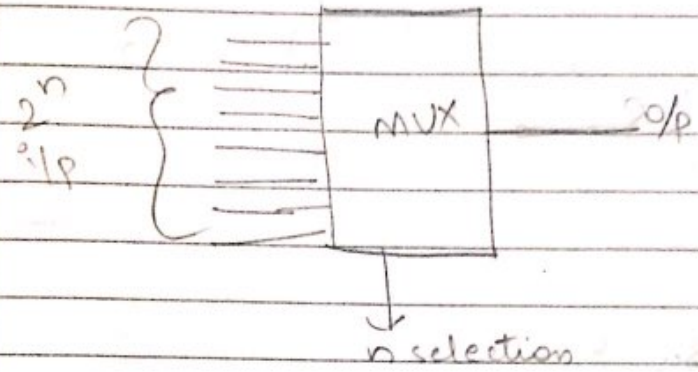


→ If two or more i/p are equal to 1 at the same time, the i/p having highest priority will take precedence.

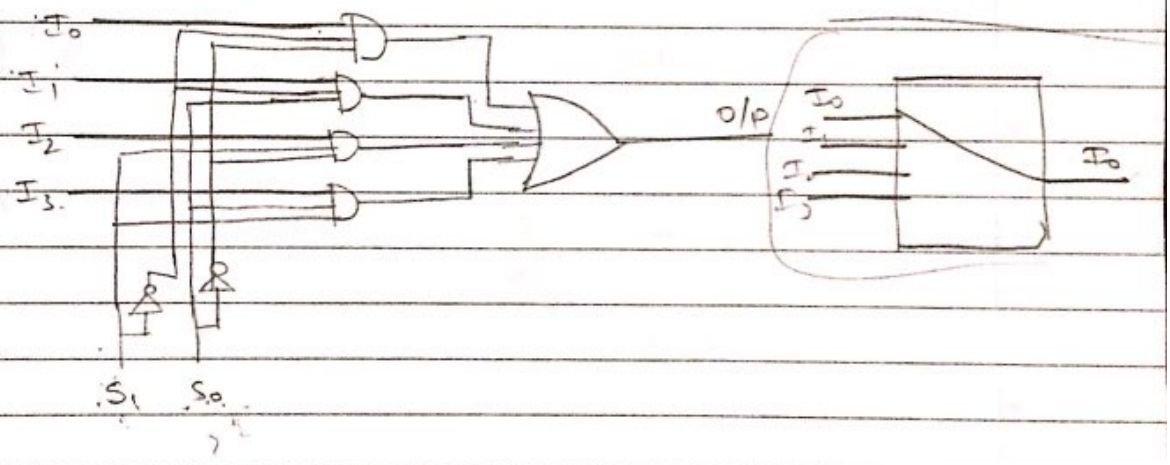
Multiplexers :→ (MUX)

It is a combinational logic circuit designed to switch one of several input lines through a single common output line by application of a control signal.

$2^n \to 1$

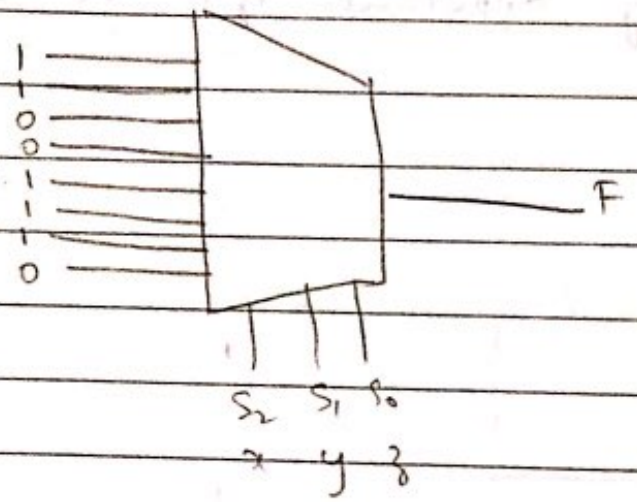


4x1 Mux :→



Camlin Page
Date / /

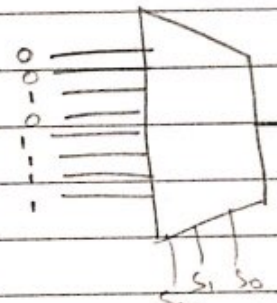
Implement the fun $F = \Sigma(0,1,4,5,6)$ using 8x1 Mux



S_2	S_1	S_0	S
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Implement the fun^n : $x + yz'$ using 8x1 MUX

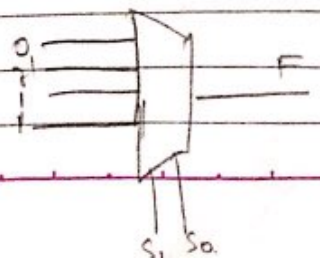
x	y	z	z'	F
0	0	0	1	0
0	0	1	0	0
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	1	1
1	1	1	0	1



$F = \Sigma(1,4,5,6,7)$

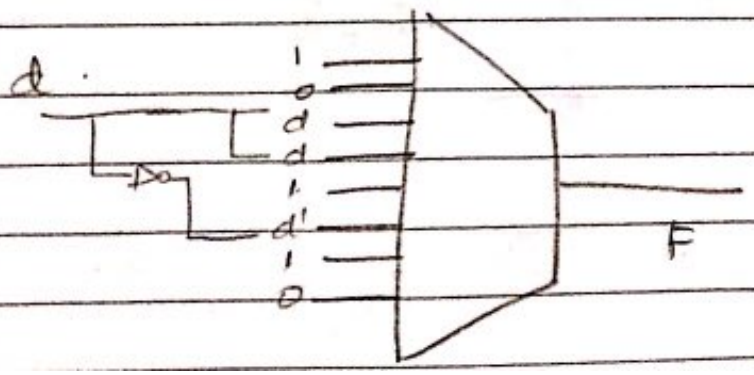
Implement above using 4X1 MUX

x	y	z	F	
0	0	0	0	F = 0
0	0	1	0	
0	1	0	1	F = z'
0	1	1	0	
1	0	0	1	F = 1
1	0	1	1	
1	1	0	1	F = 1
1	1	1	1	



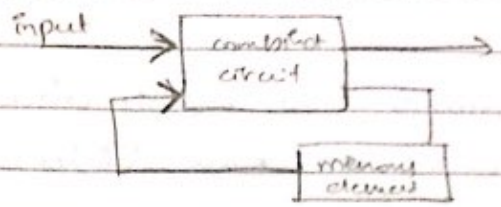
$F = \Sigma(0,1,5,2,8,9,10,12,13)$ 8×1 MUX

a	b	c	d.	F	
0	0	0	0	1	F = 1
0	0	0	1	1	
0	0	1	0	0	F = 0
0	0	1	1	0	
0	1	0	0	0	F = d
0	1	0	1	1	
0	1	1	0	0	F = d
0	1	1	1	1	
1	0	0	0	1	F = 1
1	0	0	1	1	
1	0	1	0	1	F = d'
1	0	1	1	0	
1	1	0	0	1	F = 1
1	1	0	1	1	
1	1	1	0	0	F = 0
1	1	1	1	0	



3. Synchronous Sequential logics

(ii) Sequential circuits :-



⇒ The o/p depends not only on present input but also on past i/p. (stores the element in memory element and feedbacks the o/p)

→ There are 2 types

(i) Synchronous ⟶ clock is present (o/p change when clock comes)

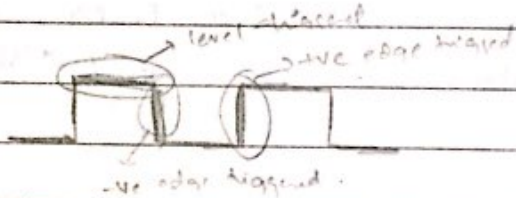
(ii) asynchronous → no clock (diff in i/p, o/p changes)

Synchronous S.C	Asynchronous S.C
(i) The change in i/p signals can effect the memory element upon the activation of clock signal.	(i) change in i/p signals can effect memory element at any instant of time.
(ii) Memory elements are clocked flip flops (FF's).	(ii) Memory elements are either unclocked flip flops or time delay elements.
(iii) The speed of the circuit depends upon the clock speed.	(iii) These circuits doesn't depend upon the clock. (so they are fast w.r.t S.S.C)
(iv) Easier to design.	(iv) they are difficult to design.
(v) circuit operation is stable.	(v) ~~circuit~~ Instability problem are present.
(vi)	

Storage Elements :->

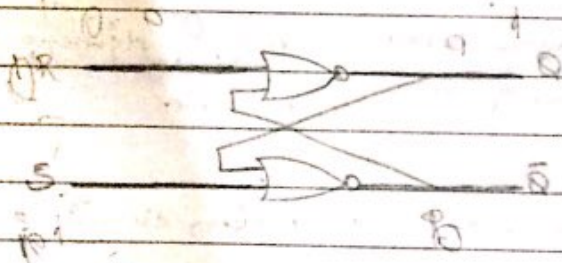
Latch :-> Storage elements that operate with signal levels (level-sensitive) are called latches.
level - triggered

Flip Flop (FF's) :- Storage elements that operate with signal transitions (Edge-sensitive) are called FF's.
Edge - triggered



Logic Circuits :->

(i) **S R latch : with NOR gate :->** (S R latch)



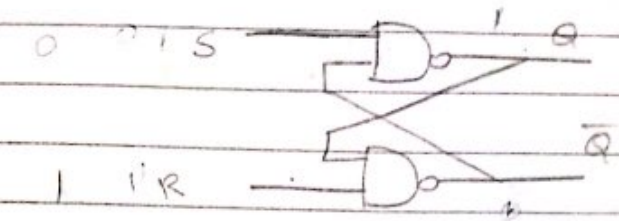
NOR

x	y	o/p
0	0	1
0	1	0
1	0	0
1	1	0

	S	R	Q	Q̄	
storing one ←	1	0	1	0	- set state
don't change the memory ←	0	0	1	0	- memory state
storing zero ←	0	1	0	1	- reset state
	0	0	0	1	- memory state
	1	1	—	—	- not used

because Q, Q̄ should be opposite

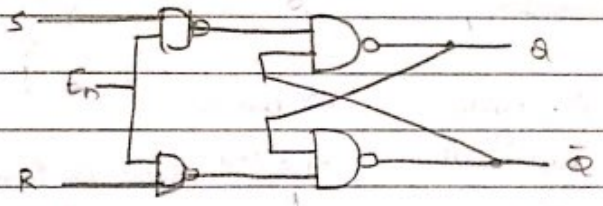
(ii) SR latch with NAND gate :- (S'-R' latch)



x	y	o/p
0	0	1
0	1	1
1	0	1
1	1	0

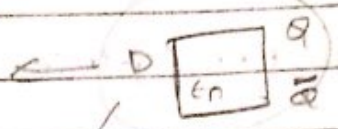
S	R	Q	$\bar{Q}$	
1	0	0	1	- reset state
1	1	0	1	- memory state
0	1	1	0	- set state
1	1	1	0	- memory state
0	0	—	—	not used

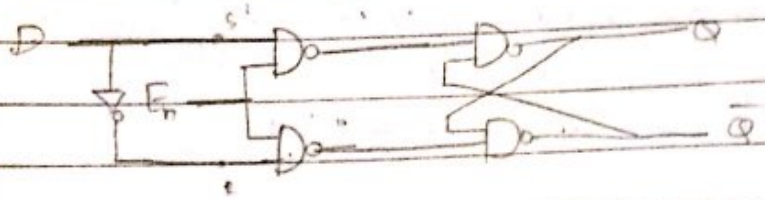
(iii) SR latch with Enable Input :- (popular SR latch)



En	S	R	Q	$\bar{Q}$	
0	——				memory state
1	1	0	1	0	set state
1	0	0	1	0.	memory state
1	0	1	0	1	reset state
1	0	0	0	1	memory state
1	1	1	——		not used

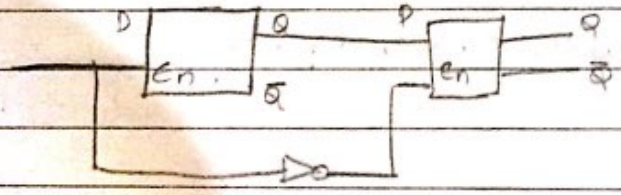
(iv) D latch:-





D	En	Q	$\bar{Q}$	
0	1	0	1	→ reset
1	1	1	0	→ set
-	0	-	-	→ memory state

(v) Negative Edge triggered D-F.F :-

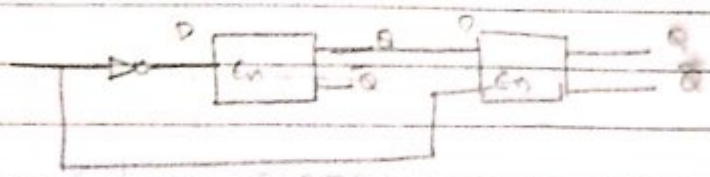


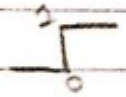
En	1st D latch	2nd D latch
0	not Enable	Enabled. (But no information)
1	Enable	not Enable (no information)

When 1 is give 1st D latch is enabled we get i/p for 2nd D latch

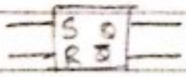
now when 0 is given 2nd D latch activate and takes i/p from 1st D latch

(vii) Positive Edge triggered D-F-F :->

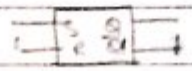




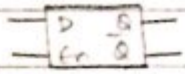
Logic Symbols :->



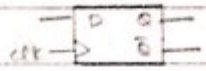
(i) SR latch



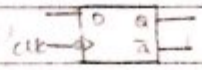
(ii) S'R' latch



(iii) D latch



(iv) +ve triggered D-FF



(v) -ve triggered D-FF

Setup time :->

Minimum time during which the D i/p must be maintained at a const value. prior to the occurance of the clock transition

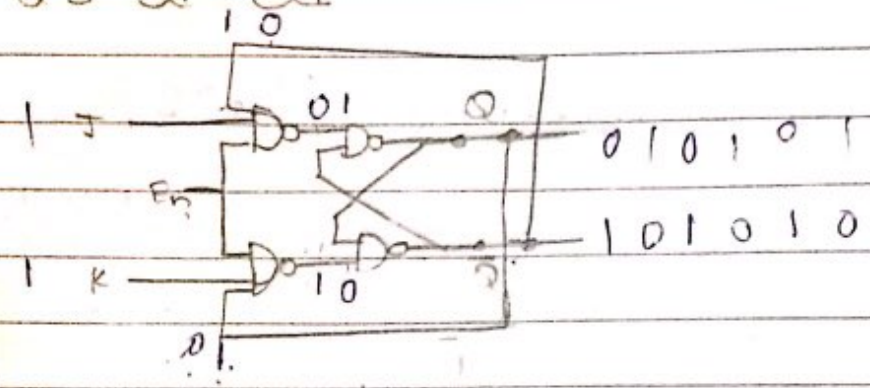
Hold time :->

Minimum time during which D i/p must not change after the application of the clock transition

Propogation delay :->

It is the interval b/w the edge of the trigger and the stabilization of the o/p to a new state.

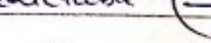
JK Flip Flop



En	J	K	Q	$\bar{Q}$	
0	—	—	—	—	memory state
1	1	0	1	0	→ set state
1	0	0	—	—	memory state
1	0	1	0	1	reset state
1	1	1	1	0	set state

En	J	K	Q	$\bar{Q}$	
0	—	—	—	—	memory state
1	1	0	1	0	set state
1	0	0	—	—	memory state
1	0	1	0	1	reset state
1	1	1	⊕	⊕	~~set state~~

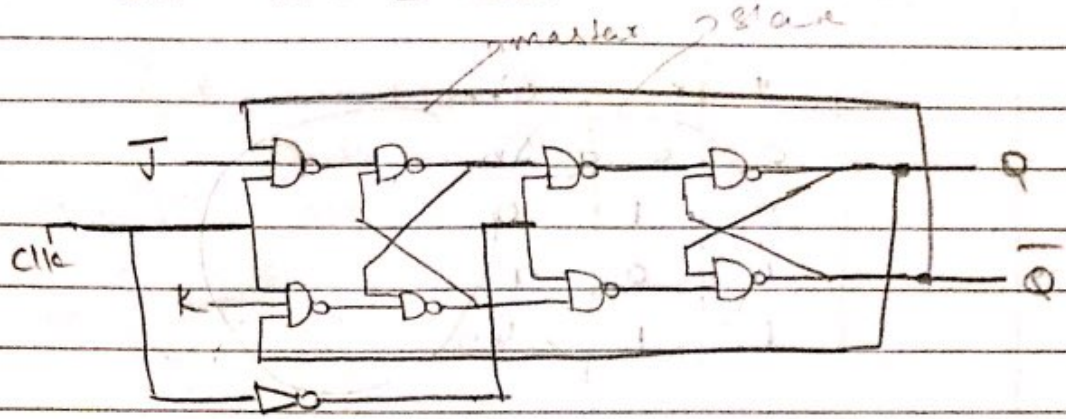
→ Race around condition

→ Race around is not used in digital circuit as the o/p is unpredictable. () (Rarely)

→ This condition is rectified using master slave JK flipflo

Camlin Page
Date / /

Master slave J K Flip Flop :→



These are 2 stage JK FF's.
a clock is used to enable the chip.

J	K	Q	$\bar{Q}$	
0	1	0	1	reset state
1	0	1	0	set state
0	0	——		memory state
1	1			

→ When the the clock goes from high to low the master FF is active and slave is inactive.
i.e When the clock goes high to low at that instant the o/p will flip.
(→ Toggling the o/p)
the o/p is controlled by the clock.

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

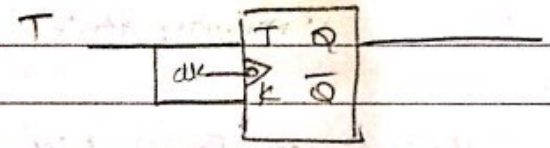
Camlin Page
Date / /

Characteristic Table :-

J	K	Q_{n+1} (next state)
0	0	Q_n → memory
0	1	0 → reset
1	0	1 → set
1	1	$\bar{Q}_n$ → controlled complement to be toggling

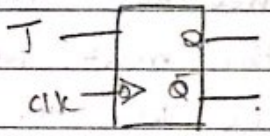
The o/p will be flipped by using clock.
i.e clock controls the o/p..
(toggles)

T Flip Flop :- (Toggle flip flop)



(or)

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



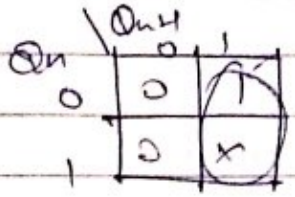
Excitation Table :-

We know the present state & past state variables but should predict the ~~past state value~~ i/p values

Excitation Table for SR latch :-

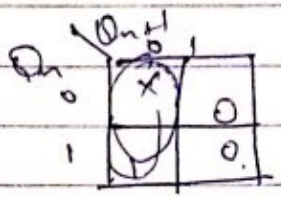
Q_n	Q_{n+1}	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

for S



$S = Q_{n+1}$

for R



$R = \bar{Q}_{n+1}$

Excitation table for D latch :→

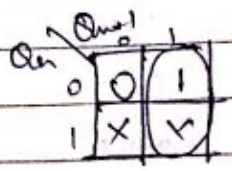
Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

$D = Q_{n+1}$

Excitation table for J K FF :→

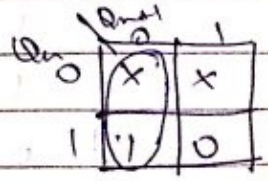
Q_n	Q_{n+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

for J



$J = Q_{n+1}$

for K



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

27/08/19

Camlin Page
Date / /

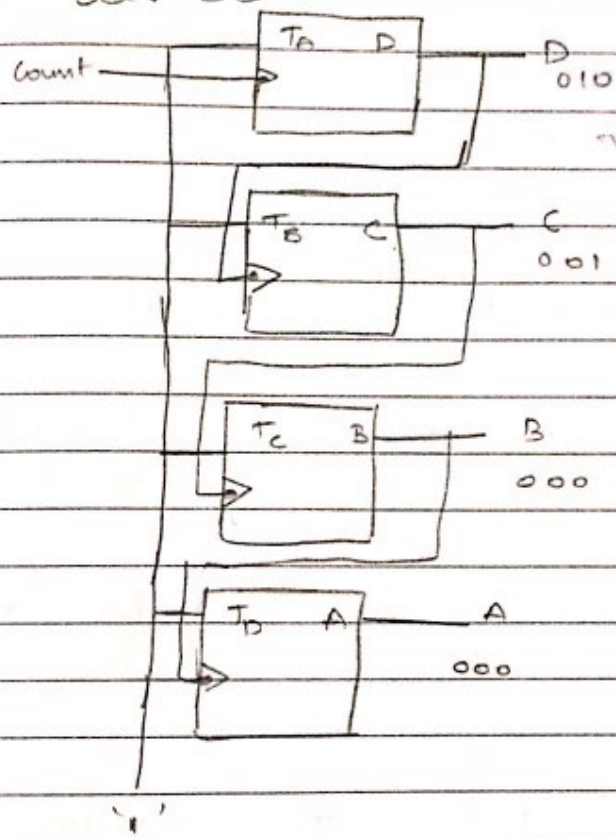
Counters:-

There are 2 types asynchronous and synchronous counter
(Ripple)

Ripple counter:-> (or) Asynchronous counter:-

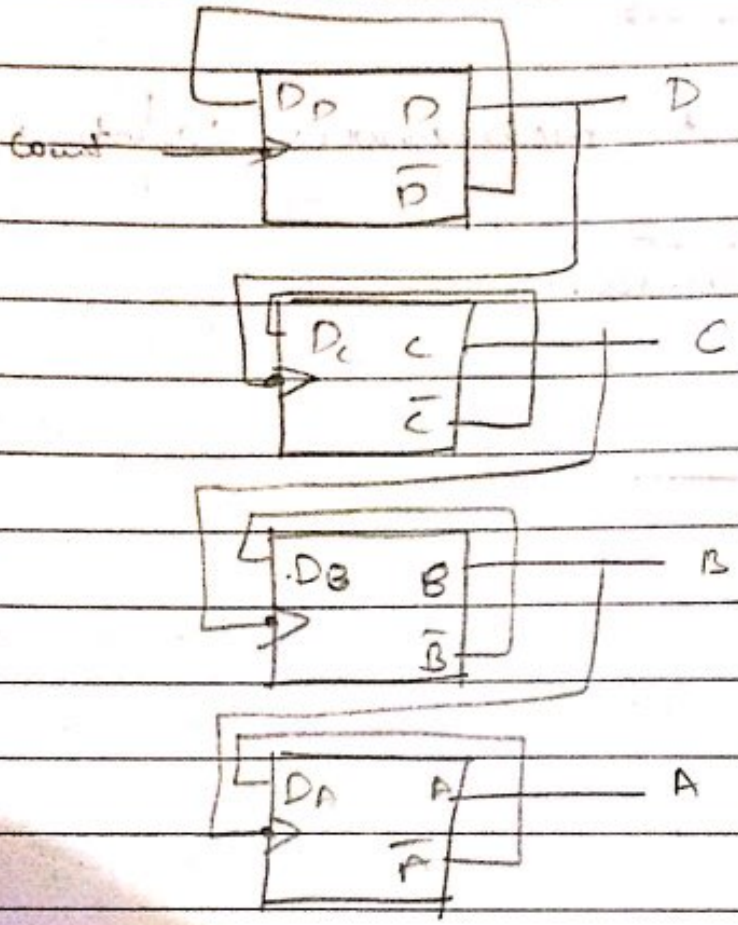
1. four bit Ripple up counter:-
(binary)

using T FF



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

using D FF

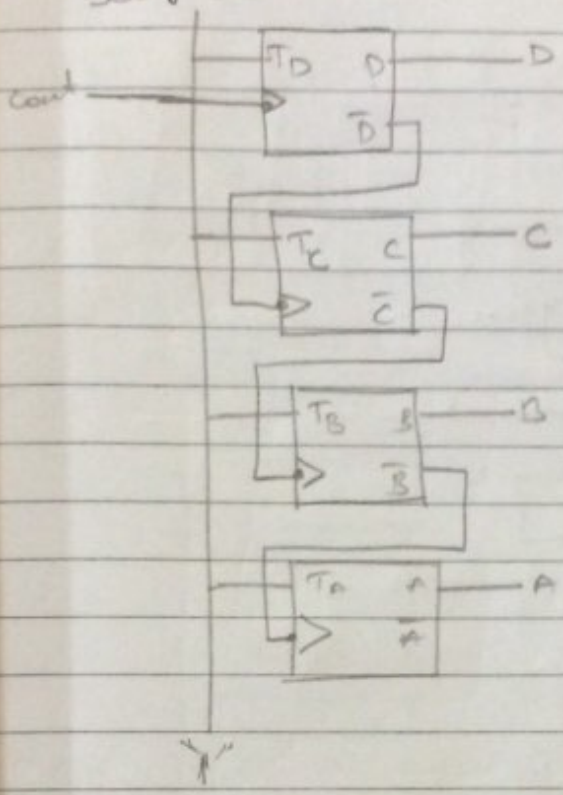


using TK FF

23/08/22

2. 4-bit binary ripple decon counter :->

using T FF



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

As we apply -ve edge triggering

If we take clock as compliment then it is down counter

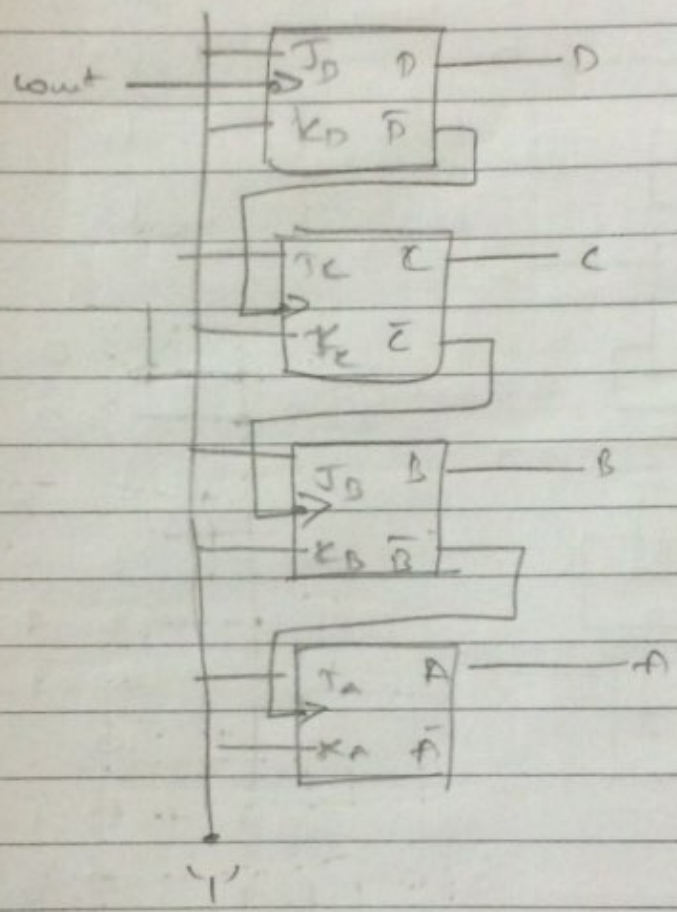
If it is not compliment then it is up counter.

for +ve edge triggering

clock as compliment then it is up counter

not compliment then it is down counter.

using JK FF

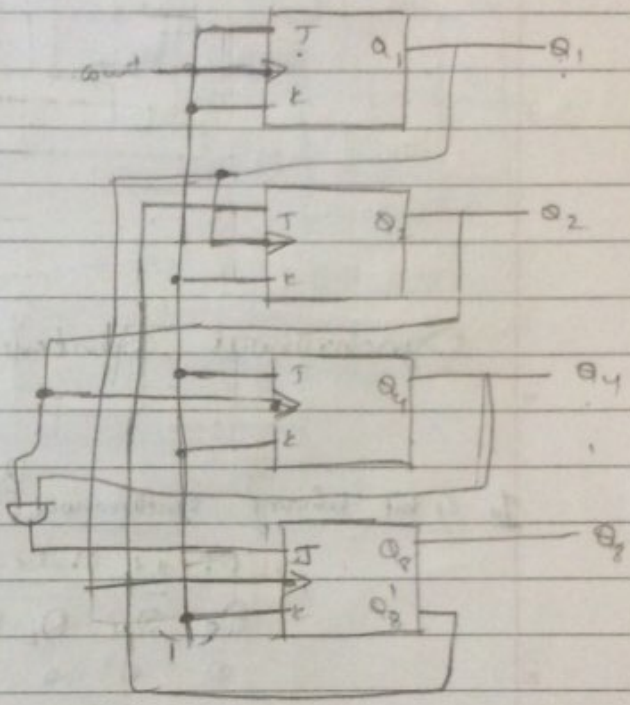


3. BCD Ripple up counter :- also called Modulo 10

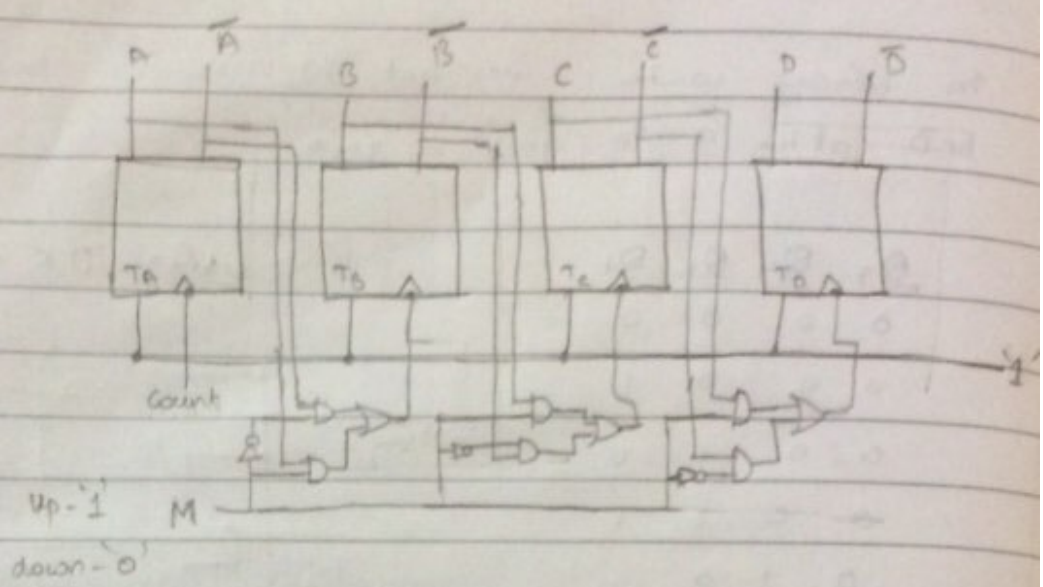
In binary counter we get 10, 11, 12 ..... but for BCD after 9 it goes to zero.

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

using JK FF



4. Binary ripple up-down counter :-



Synchronous counters :-

1. 4 bit binary synchronous Up counter :-

Next state				← Previous state			
Q_8	Q_4	Q_2	Q_1	Q_8	Q_4	Q_2	Q_1
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	1
0	0	1	1	0	0	1	0
0	1	0	0	0	0	1	1
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	1
0	1	1	1	0	1	1	0
1	0	0	0	0	1	1	1
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	1
1	0	1	1	1	0	1	0
1	1	0	0	1	0	1	1
1	1	0	1	1	1	0	0
1	1	1	0	1	1	0	1
1	1	1	1	1	1	1	0
0	0	0	0	1	1	1	1

T_8	T_4	T_2	T_1
0	0	0	1
0	0	1	1
0	0	0	1
0	1	1	1
0	0	0	1
0	0	1	1
0	0	0	1
1	0	0	0
0	0	0	1
0	0	1	0
0	0	0	1
0	1	1	1
0	0	0	1
0	0	1	1
0	0	0	1
1	1	1	1

T_2

Q_8Q_4 \ Q_2Q_1	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

$T_2 = Q_1$

$T_1 = 1$

T_8

Q_8Q_4 \ Q_2Q_1	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	0	0	1	0
10	0	0	0	0

$T_8 = Q_4 Q_2 Q_1$

T_4

Q_8Q_4 \ Q_2Q_1	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	0	0	1	0
10	0	0	1	0

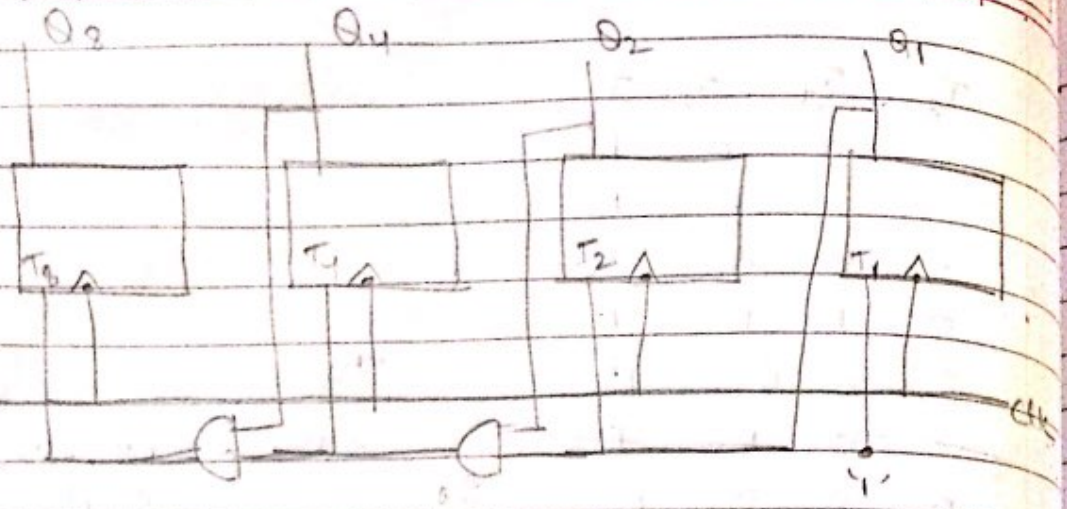
$T_4 = Q_2 Q_1$

$T_8 = Q_4 Q_2 Q_1$

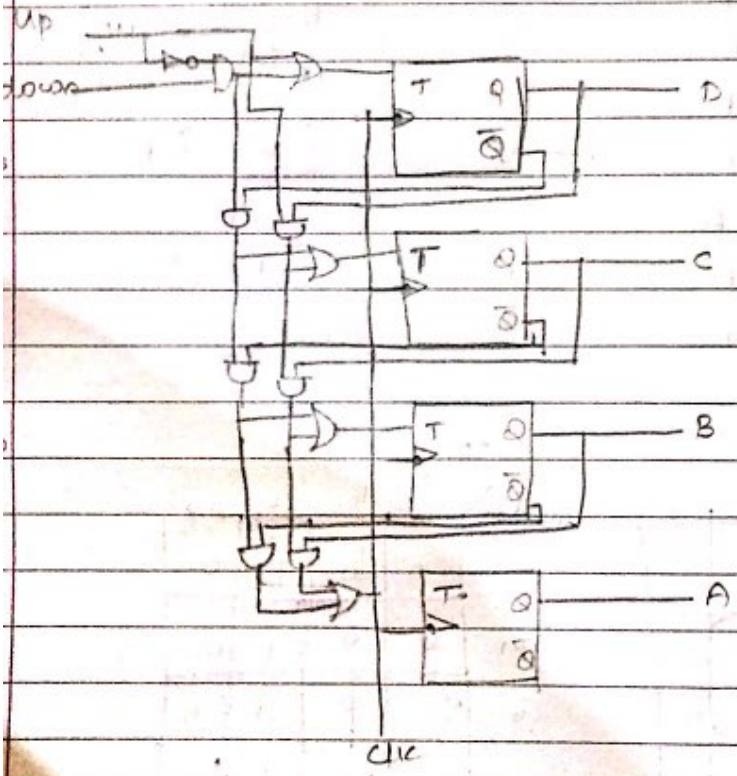
$T_4 = Q_2 Q_1$

$T_2 = Q_1$

$T_1 = 1$



Q) 4 bit synchronous up-down counter.



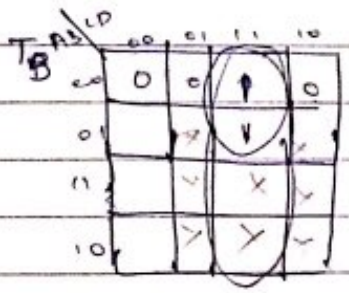
→ For up county 1 is given to up irrespective of down

→ For down county 0 is given to up and 1 to down

3) Synchronous BCD up counter:-

P.S				N.S				FF i/p			
A	B	C	D	A	B	C	D	T_A	T_B	T_C	T_D
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	1
0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	1
1	0	0	1	0	0	0	0	1	0	0	1

$T_D = 1$



$T_B = \cdot CD$

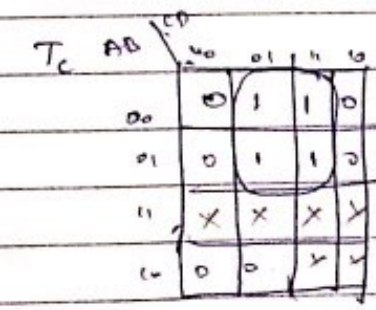
$T_A = A'BCD + AB'C'D$

$= D(A'BC + AB'C')$

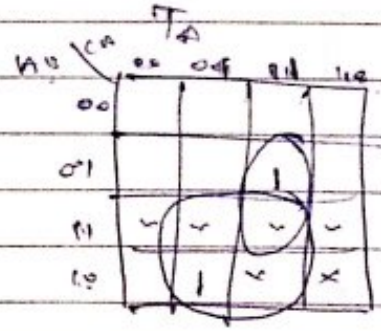
$= D[(A' + AB'C')(BC + B'C')]$

$= D[(A'+A)(A'+B'C')(BC+B'C')]$

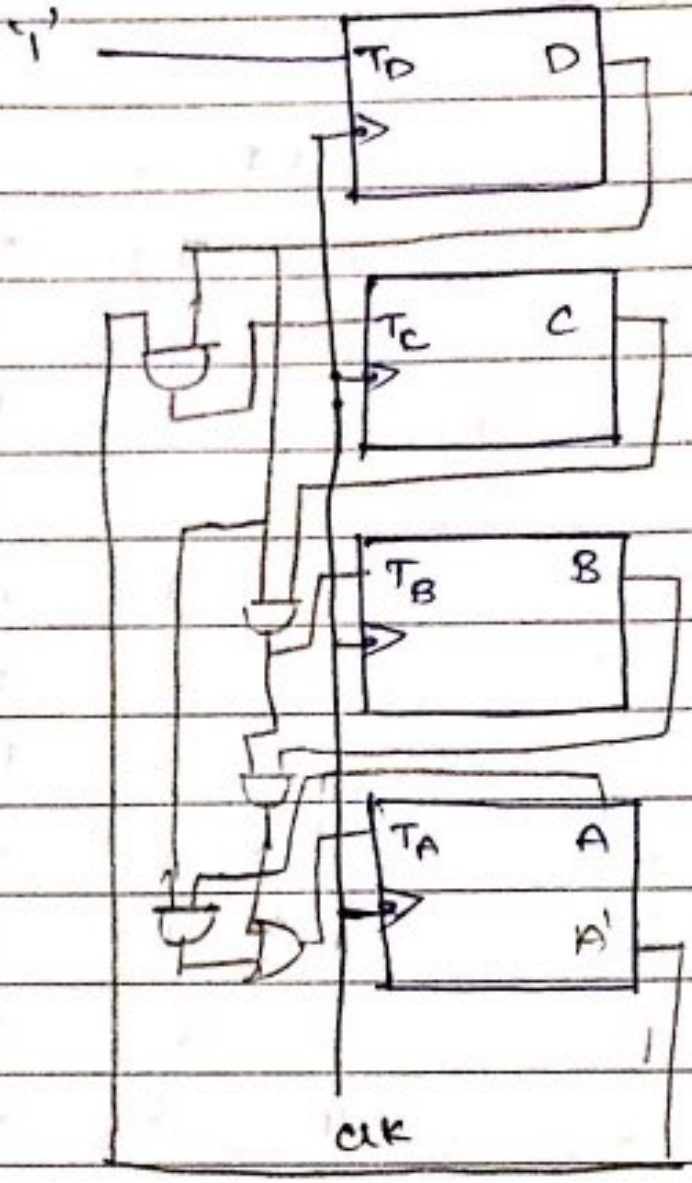
$= D[(A'+B'C')(B+B')(B+C')]$



$T_C = A'D$



$T_A = AD' + BCD$

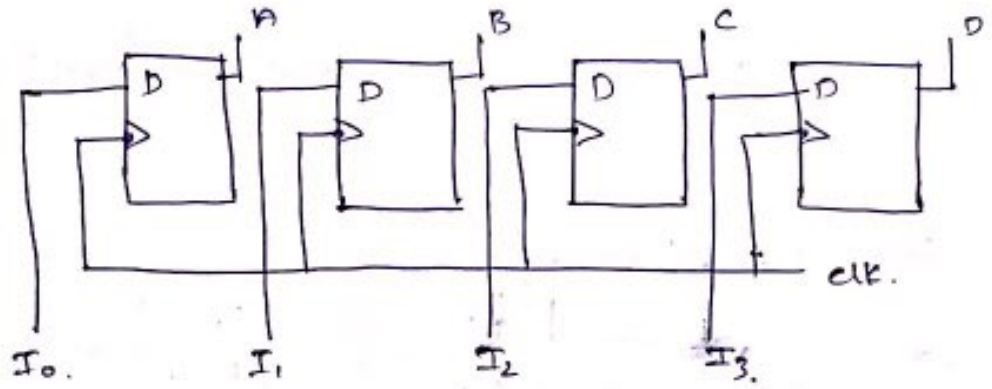


4. Registers and Memories

Registers :→

It is a group of flip flop's each one of which shares a common clock and is capable of storing ~~one bit~~ n-bit information using n-FF's

4-bit register :→



→ As we apply the clock the value might change.

(There are 2 types
synchronous → clk↑, load↑
asynchronous → only load↑)

→ So we use register with parallel load.

taking that if

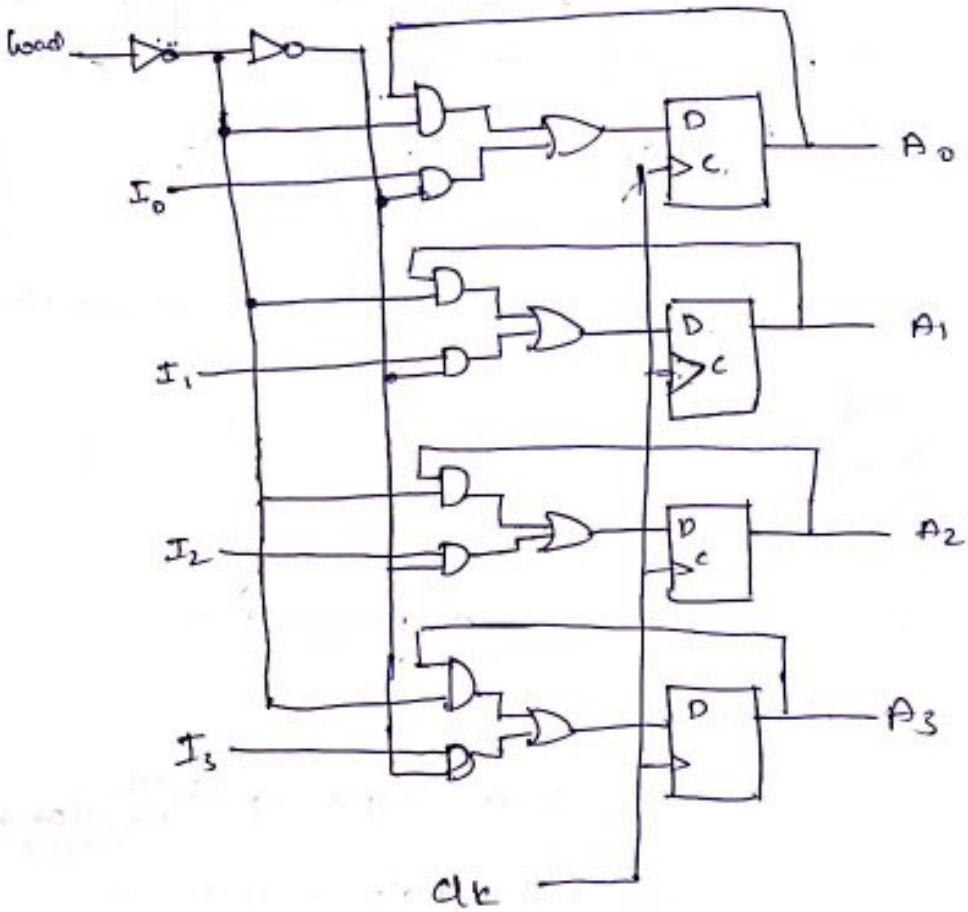
load is 0 o/p is i/p (holds previous value)

load is 1 o/p is next value.

Register with Parallel load :-

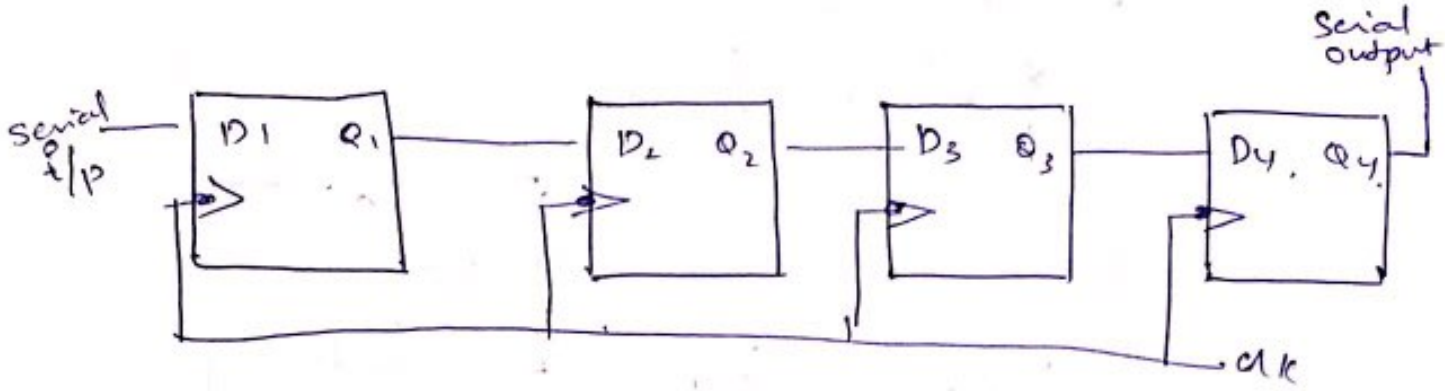
load	I_i	Q_i	D_i	
0	0	0	0	load = 0, $D_i = Q_i$
0	0	1	1	
0	1	0	0	
0	1	1	1	
1	0	0	0	load = 1, $D_i = I_i$
1	0	1	0	
1	1	0	1	
1	1	1	1	

Taking K-map $D_i = load\ I_i + \overline{load}\ Q_i$

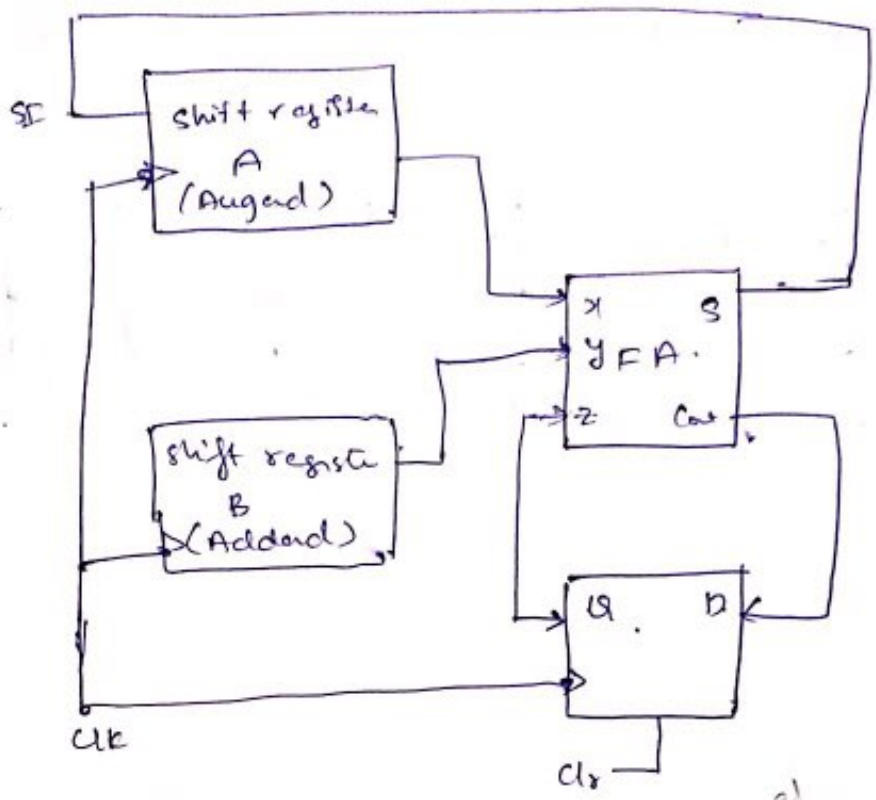


Shift register :→

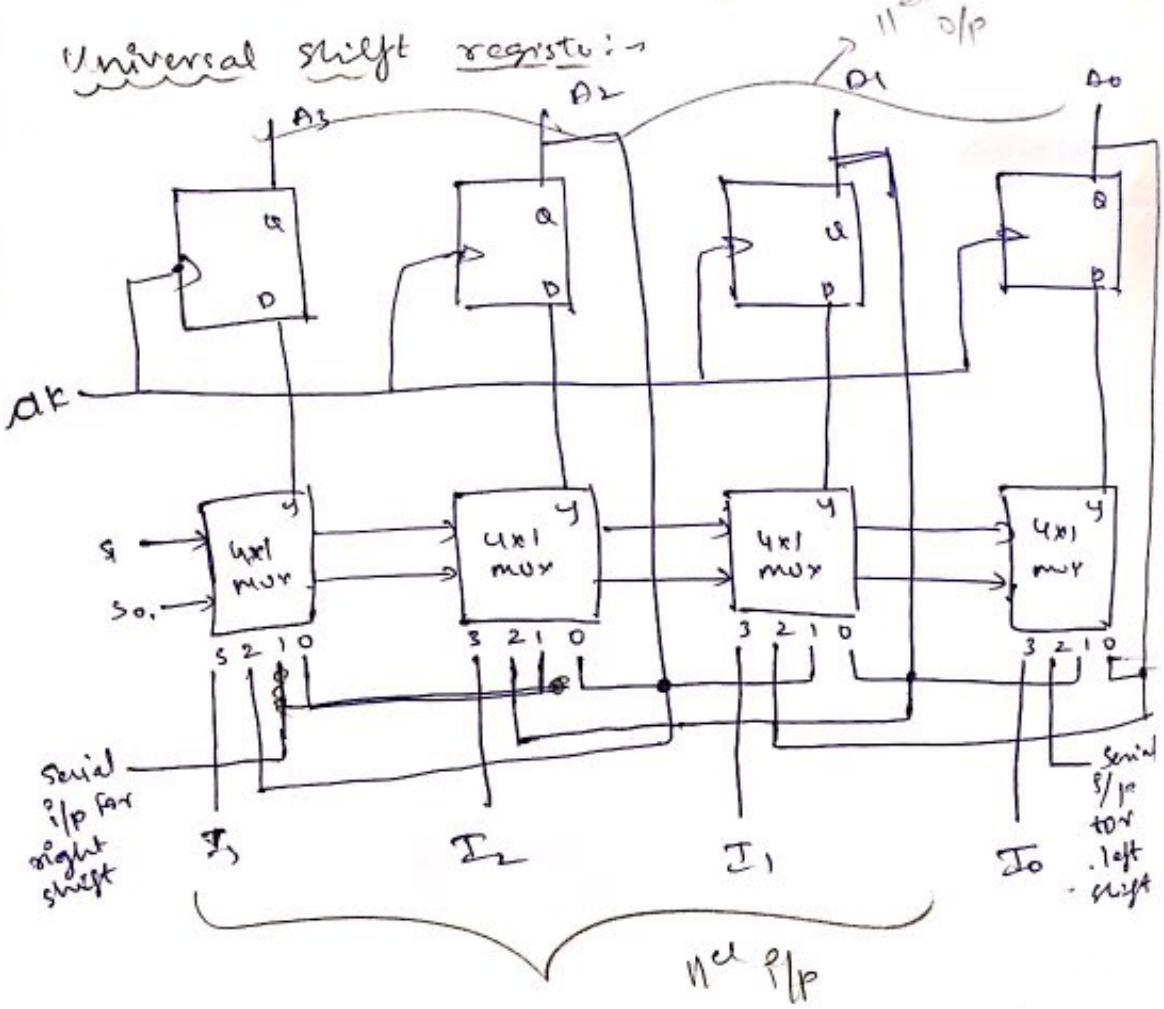
This register is capable of shifting memory to its neighborhood FF in the specified direction.



Serial adder :-



Universal shift register :-



Memories:-

→ A memory unit is a device to which binary information is transferred for storage and from which information is retrieved when needed.

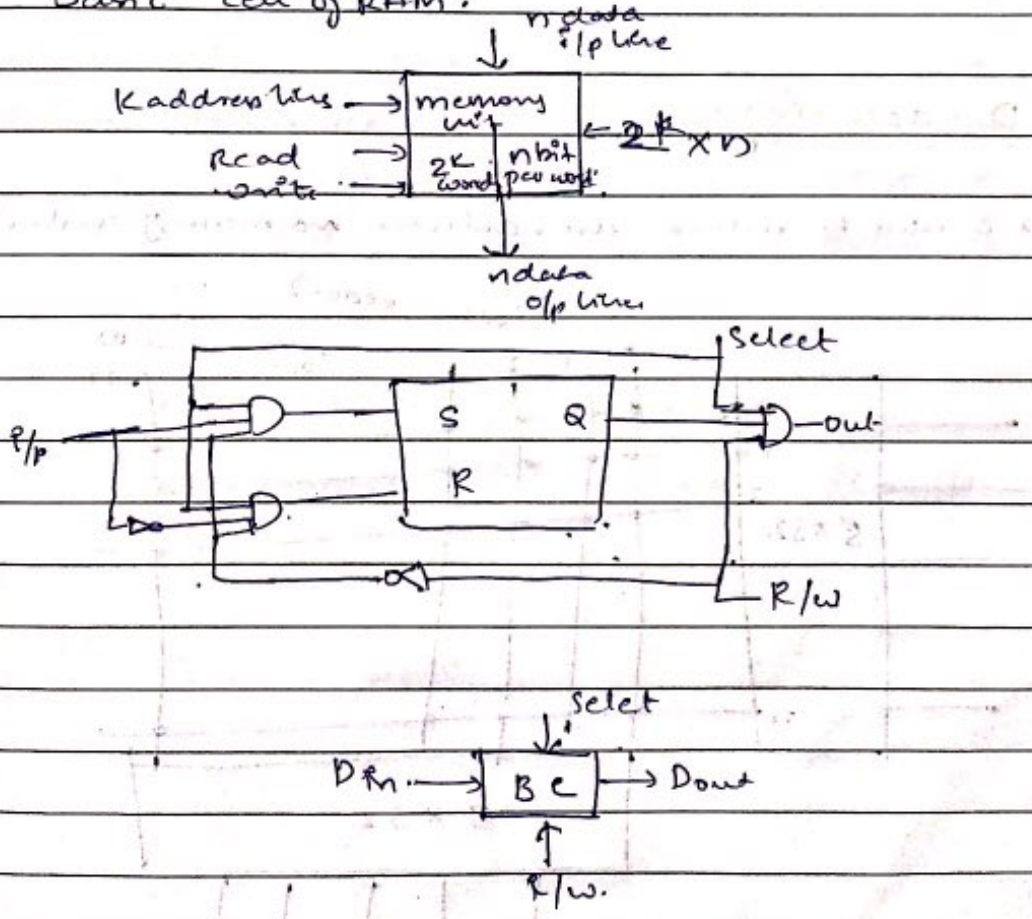
→ It is a collection of cells capable of storing a large quantity of binary information.

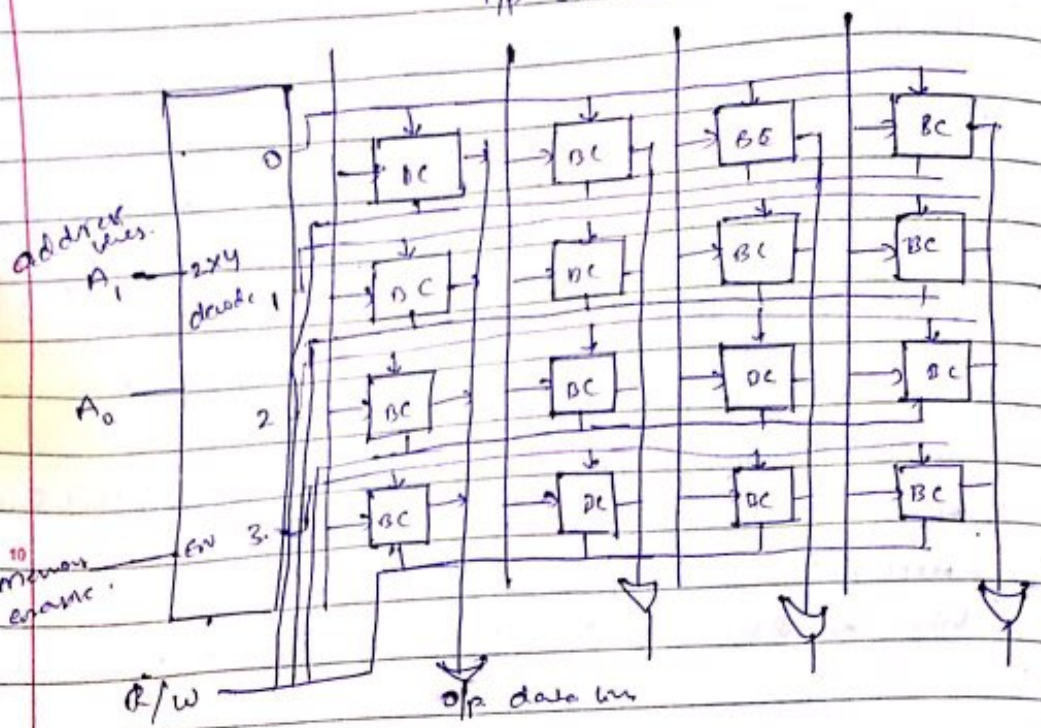
→ We use 2-types of memories mainly in Digital System RAM and ROM.

→ RAM (read and write) stores new information for later use where as ROM is a programable logic device. (only read)

Random Access Memory:- (RAM)

Basic cell of RAM.



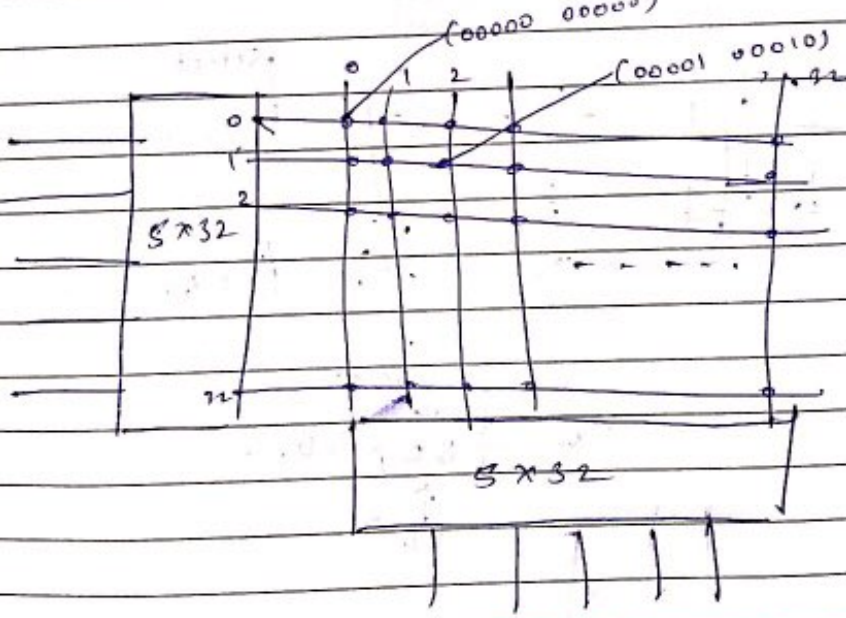


→ 4x4 → 4 memory locations in 4 rows and 4 bits in each row.

12/9/24

2 D - decoding :-

It is used to reduce the hardware and memory construction



we have 32 x 32 intersection

= 1024 locations

AND gates used as 64 → 5 i/p AND gate

instead of 1024 → 10 i/p AND gate

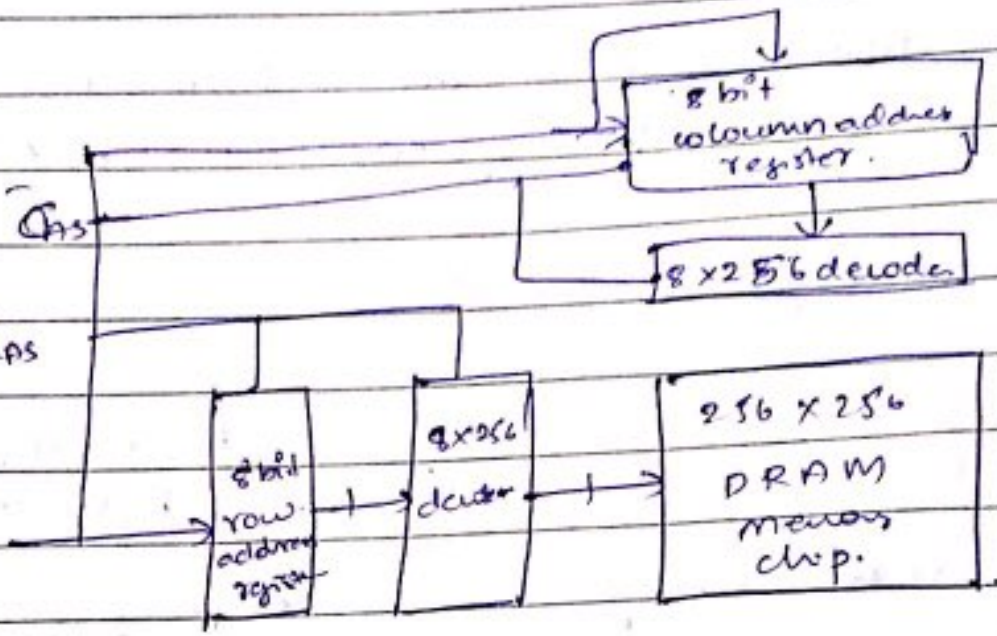
→ There are 2 types of RAM's.

SRAM (Static)	DRAM (Dynamic)
(i) Stored Bits in memory cells. composed of FF	(i) Memory cells composed of capacitor and transistor
(ii) Requires 6 Transistor to store 1 bit	(ii) Requires 1 transistor and 1 capacitor to store 1 bit
(iii) more costly	(iii) less costly
(iv) faster access time	(iv) lower access time
(v) more power consumption	(v) less power consumption.
(vi) low density per chip	(vi) high density per chip.
(vii) does not need to be refreshed periodically	(vii) should be refreshed every few seconds.

→ RAM is always volatile.

→ ROM, memory disk are non volatile.

Refreshing cycle of D-RAM's:-



$\bar{C}_{AS}$ → Colom address i/p selection

$\bar{R}_{AS}$ → Row address i/p selection

Error detection and Correction code:-

→ Detects and corrects code.

→ Hamming code.

K - parity bits

n → data bits

Total (n+k) data bits

Date / /

Hamming Code:→

we have k parity bits and n data bits

Total bits (k+n)

parity bits are placed at 2^x position. $x = 0, 1, 2, \ldots$

and others are data bits.

→ If we consider 8 bit binary numbe. (.0011 1000)

P_1 P_2 D_3 P_4 D_5 D_6 D_7 P_8 D_9 D_{10} D_{11} D_{12}

(we take 8 data bits)

P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D_{10}	D_{11}	D_{12}
		0		0	1	1		1	0	0	0

Calculating parity bits

→ write the position of data bits in binary form.

	P_8 P_4 P_2 P_1
D_3	0 0 1 1
D_5	0 1 0 1
D_6	0 1 1 0
D_7	0 1 1 1
D_9	1 0 0 1
D_{10}	1 0 1 0
D_{11}	1 0 1 1
D_{12}	1 1 0 0

for P_1 EXOR the data bits whose units position is 1

$P_1 = D_3 \oplus D_5 \oplus D_7 \oplus D_9 \oplus D_{11}$

0 0 1 1 0

→ If there are even no of 1's EXOR = 0

odd no of 1's EXOR = 1

$\boxed{P_1 = 0}$

$P_2 = D_3 \oplus D_6 \oplus D_7 \oplus D_{10} \oplus D_{11}$

$= 0 \oplus 1 \oplus 1 \oplus 0 \oplus 0$

$\boxed{P_2 = 0}$

$P_4 = D_5 \oplus D_6 \oplus D_7 \oplus D_{12}$

$= 0 \oplus 1 \oplus 1 \oplus 0$

$\boxed{P_4 = 0}$

$P_8 = D_9 \oplus D_{10} \oplus D_{11} \oplus D_{12}$

$= 1 \oplus 0 \oplus 0 \oplus 0$

$\boxed{P_8 = 1}$

P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D_{10}	D_{11}	D_{12}
0	0	0	0	0	1	1	1	1	0	0	0

for P_8, P_4, P_2, P_1 there are check bits called C_8, C_4, C_2, C_1

$C_1 = D_3 \oplus D_5 \oplus D_7 \oplus D_9 \oplus D_{11} \oplus P_1$

$= 0$

$C_2 = 0$

$C_4 = 0$

$C_8 = 0$

If all the check bits are zero then there is no error

If $D_5 = 1$

$C_1 = 1$
$C_2 = 0$
$C_4 = 1$
$C_8 = 0$

C_8	C_4	C_2	C_1	
0	1	0	1	→ D_5

→ It cannot detect double error.

Advanced Hamming code (PLD's)

PLD → Programmable logic Devices.

→ detects 2 errors and corrects 1.

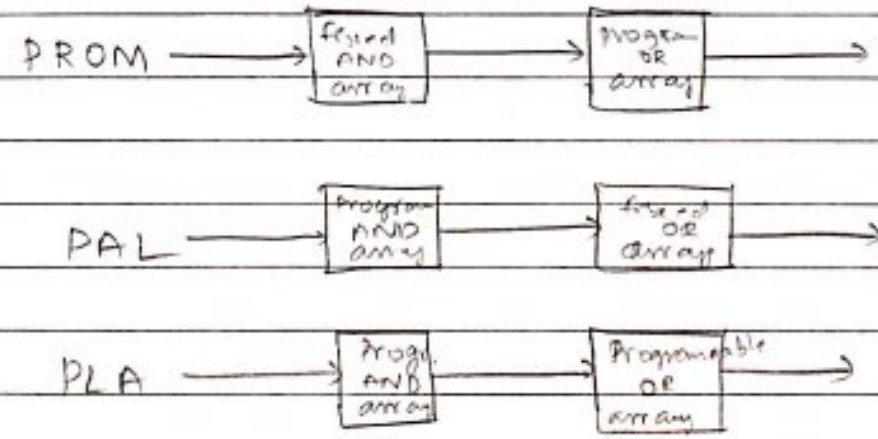
→ there are 3 PLD's

PROM → programmable read only memory

PAL → programmable array logic

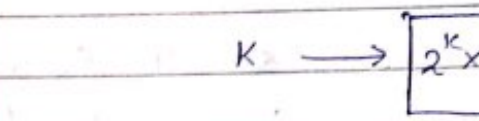
PLA → Programmable logic array.

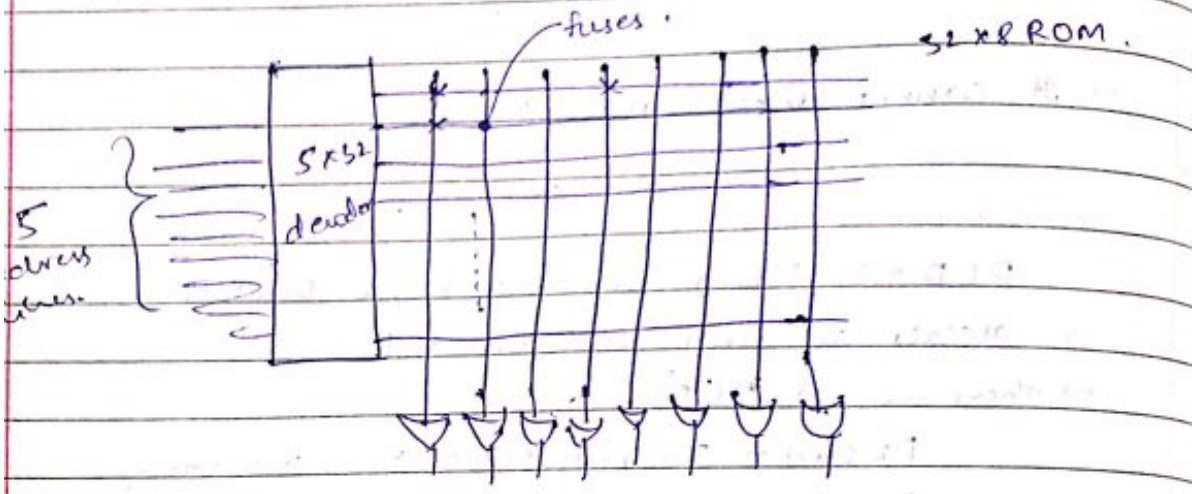
→ There are AND arrays and OR arrays, based on which we decide our PLD.



Camlin Page
Date / /

Read Only Memory :→ ROM





→ when there is fuse present then it is 0.

→ when fuse is blown it is 1.

→ If the power goes, there is no information lost hence it is non volatile.

Programmable array logic (PAL) :-

Find PAL for

$F_1 = \Sigma(1,2,3,4)$

$F_2 = \Sigma(1,2,4,5)$

$F_3 = \Sigma(1,7)$

F_1

AB \ C	0	1
0	0	1
1	1	0

$F_1 = A'B + A'C + AB'C'$

F_2

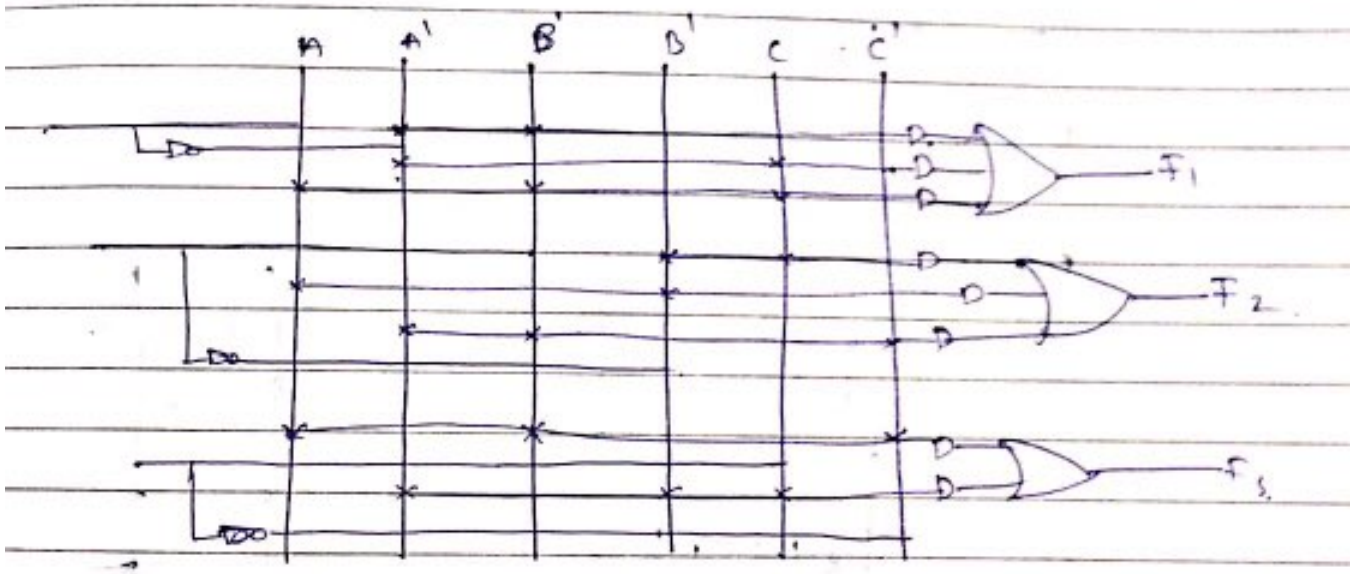
AB \ C	0	1
0	0	1
1	1	0

$F_2 = B'C + AB' + A'BC'$

F_3

AB \ C	0	1
0		1
1		1

$F_3 = A'B'C + ABC'$



Programmable logic array (PLA):-

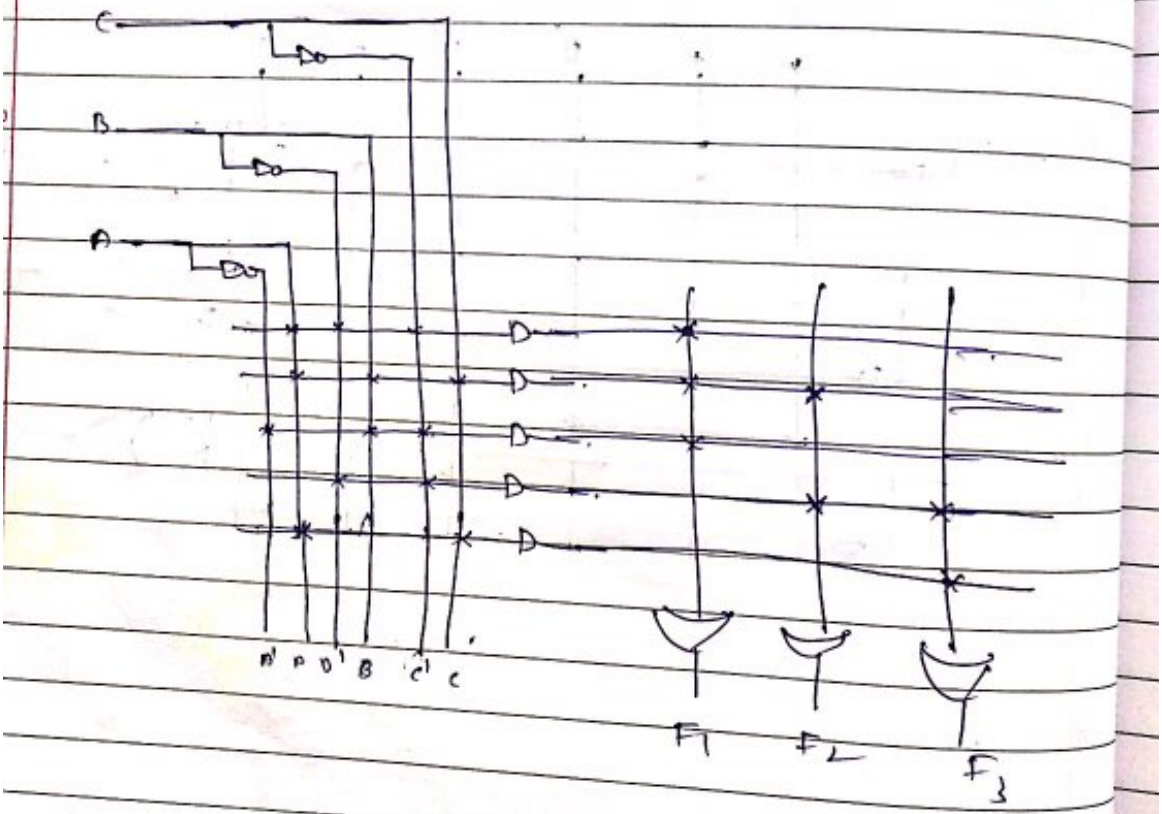
Implement the fuction using PLA

$F_1 = AB'C' + ABC + A'BC'$

$F_2 = ABC + B'C'$

$F_3 = AC + B'C'$

Product	I/P A	B	C	O/P F_1	F_2	F_3
$AB'C'$	1	0	0	1	-	-
ABC	1	1	1	1	1	-
$A'BC'$	0	1	0	1	-	-
$B'C'$	-	0	0	-	1	1
AC	1	-	1	-	-	1



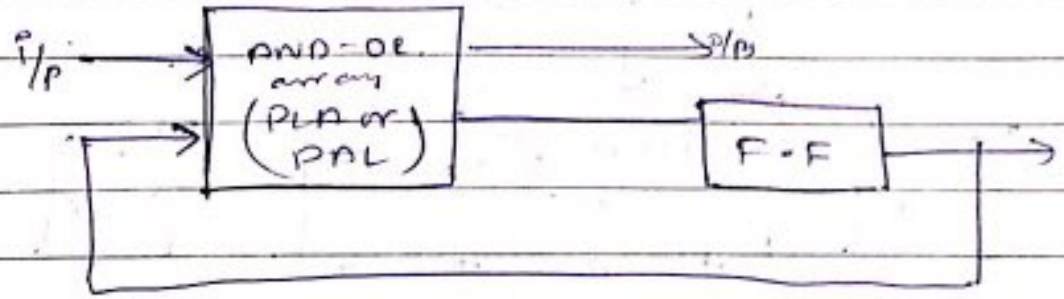
Sequential programmable devices :-

These are designed with flip flops and gate.

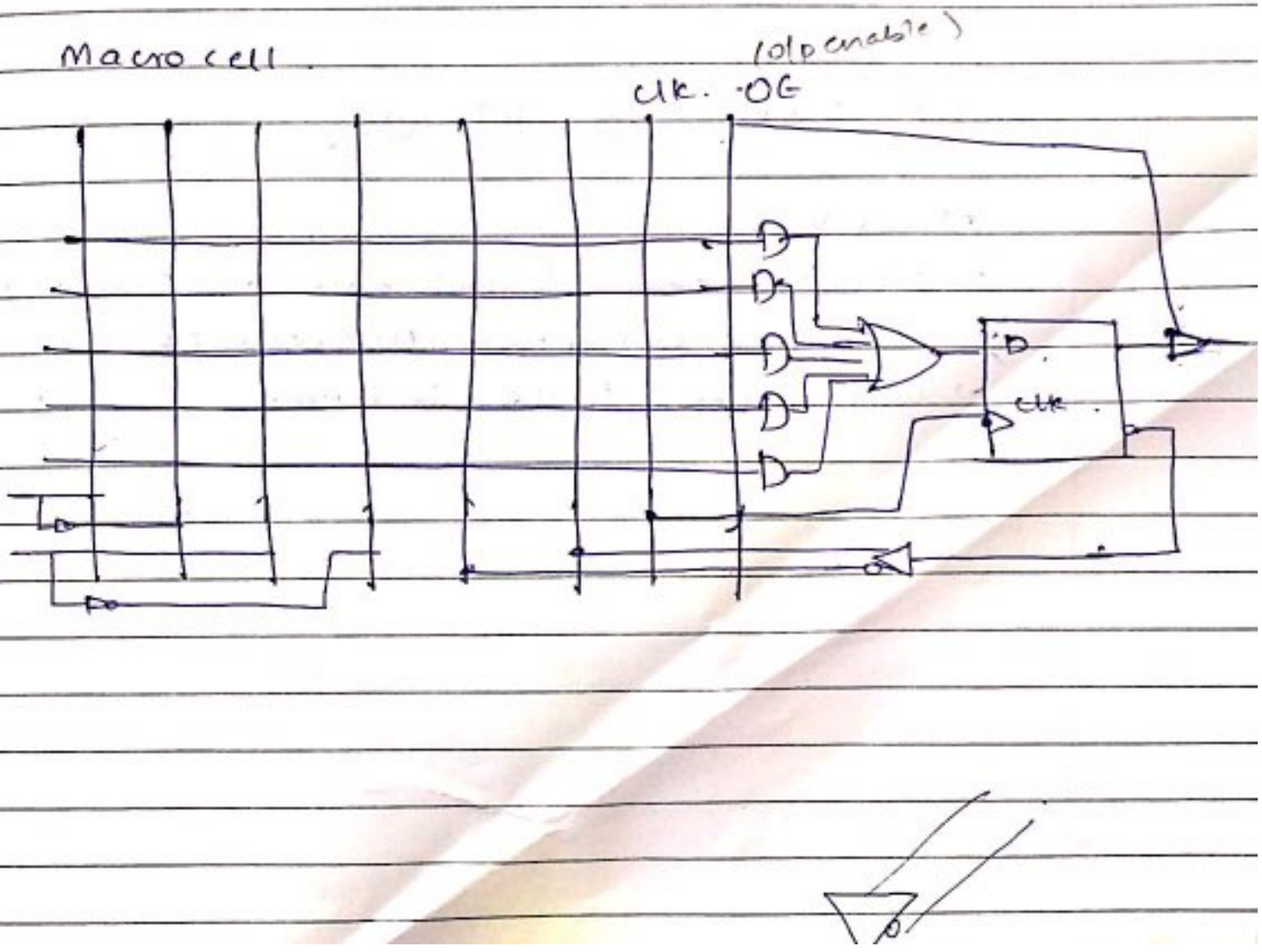
There are 3 types :

. (SPLD, CPLD, FPGA)

(1) Sequential programmable logic device :-



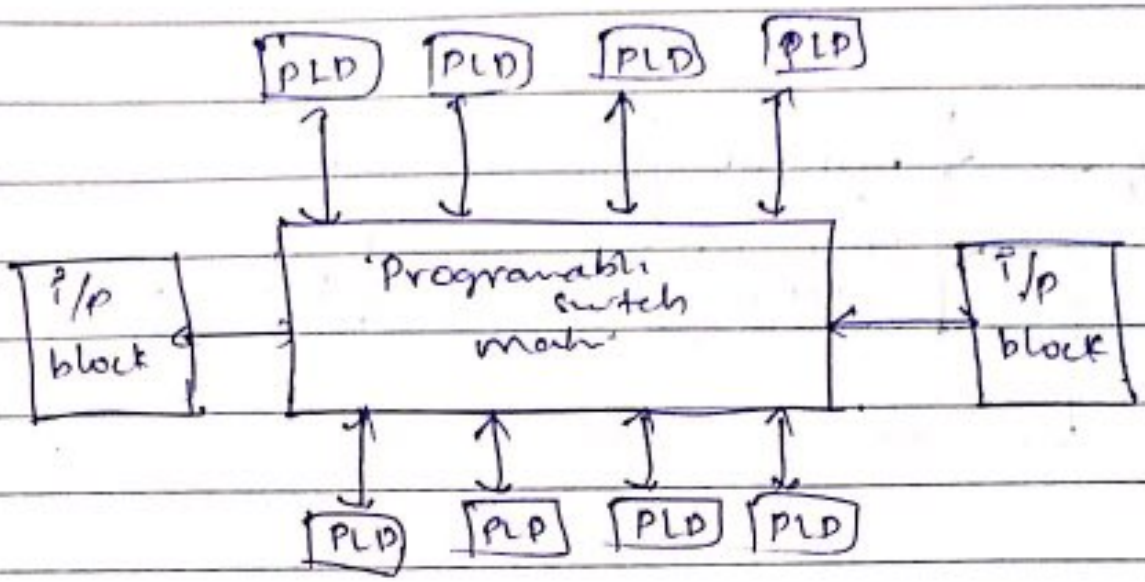
Macro cell



(ii) Complex Programable logic devic (CPLD):-

→ connection of several individual PLD's.
(PAL + PLA)

→ The PLD's are interconected thrash programmable switch matrix.



(iii) Field programable gate array:-

→ uses VLSI
→ Elimentary element is logic block and is in 100's or 1000's
→ It consists of look up table, MUX, gates, FF, etc.
→ look up table is stored in SRAM.

5. Digital Integrated Circuits

Camlin | Page
Date / /

The gates (logic gates / universal gates) are made by the below technologies.

RTL → Resistor Transistor logic
DTL → Diode Transistor logic
TTL → Transistor Transistor logic
MOS → metal oxide semiconductor < nMOS / pMOS
CMOS → complementary metal oxide semiconductor
ECL → Emitter coupled logic.

→ We use BJT's and MOSFET's in these circuits.
→ We commonly use TTL, ECL, CMOS now a days.

Basic characteristics :→

Fanout → allows the gate to connect connection of other gates → As HIGH as possible.
Power dissipation → power consumed → LOW
Propagation delay → the time taken to reflect the change → LOW
Noise Margin → max noise that can be allowed →

fanout :→ It is a no of standard nodes that can be connected to the output of the gate without degrading its normal operation. (ratio of o/p to i/p current)

Power dissipation :→

$$P_{D(avg)} = I_{CC(avg)} \cdot V_{CC}$$

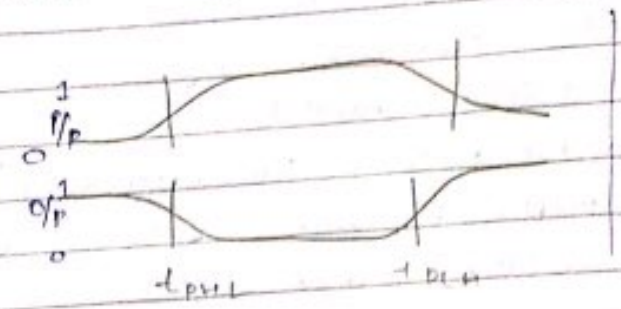
$$I_{CC(avg)} = \frac{I_{CCH} + I_{CCL}}{2}$$

(I_{CCH} → high, I_{CCL} → low)

The current drawn from power supply when output of gate is in high voltage level is I_{CCH}. If it is in low voltage level it is I_{CCL}.

The power dissipation represents the amount of power needed by the gate.
It is in (mW).

Propogation delay :→



Propagation delay $= \frac{t_{PHL} + t_{PLH}}{2}$.

~~t_{PHL} :→~~

The avg transition delay time for the signal to propogate from i/p → o/p when the binary i/p signal changes its value)

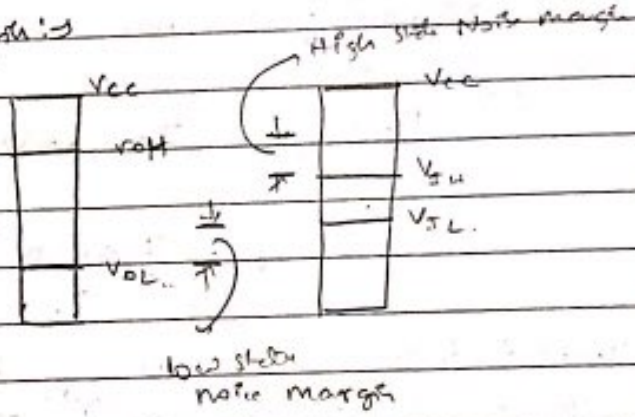
t_{PHL} :→

The propogation delay from i/p → o/p when the o/p chages from high to low

t_{PLH} :→

The propogation delay from i/p → o/p when o/p chages from low to high.

Noise Margin :→



noise margin = $V_{OH} - V_{IH}$

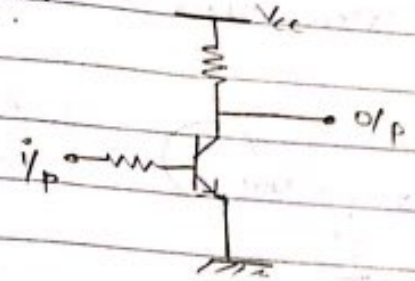
or,

$V_{IL} - V_{OL}$ } whichever is smaller

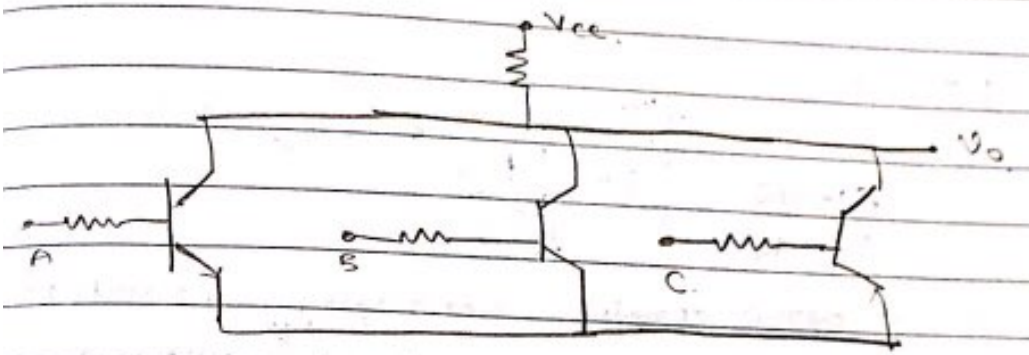
→ It is the maximum noise voltage that can be added into the signal that does not cause any undesirable change in the output.

RTL :→ Resistor Transistor logic :→

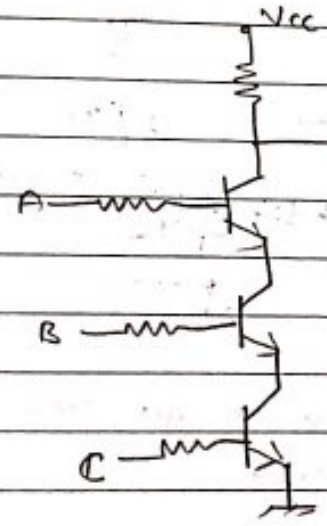
NOT gate :→



NOR gate :→



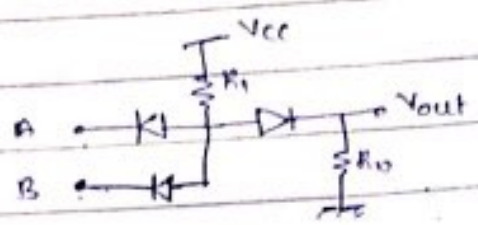
NAND gate :→



DTL:→ Diode transistor logic:-

First started as DL later converted to DTL.

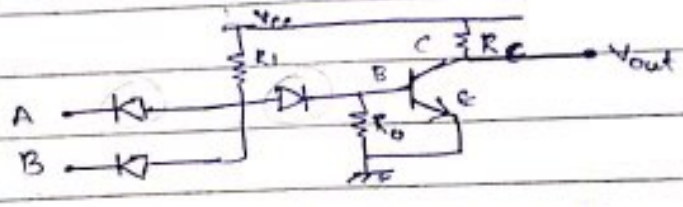
DL:→



A	B	out
0	0	0
0	1	0
1	0	0
1	1	1

→ This operation is AND gate

DTL:→

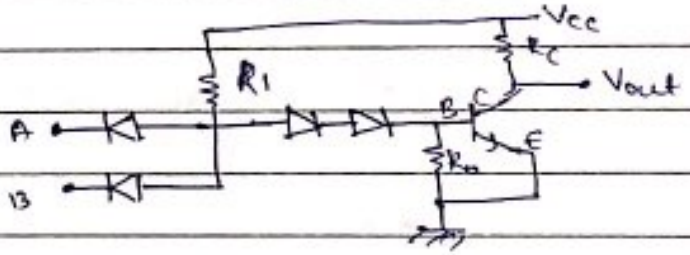


→ This operation is NAND gate (2 diff cut in voltages)

→ Sometimes due to changes in cut-in voltage of diodes which may effect the o/p.

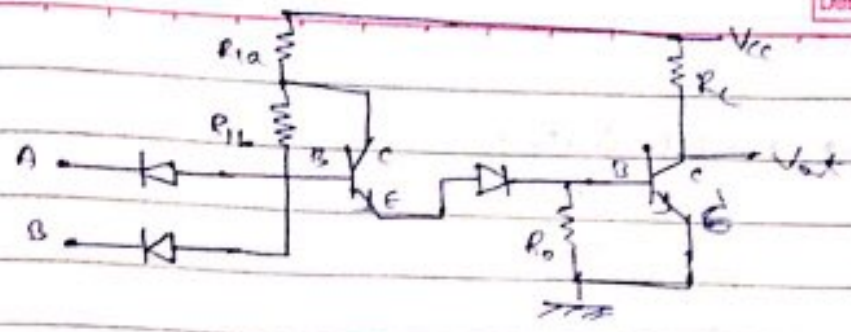
So. to correct this they added another diode

DTL - NAND gate



→ we use transistor as we get more current i.e fanouts are more.

→ Further to increase current drive capability we made some modifications.



TTL :→ Transistor Transistor logic

There are 74 Series

74L → low power TTL.

SSI → Small Scale Integrate

MSI → medium Scale Integrate

→ In BJT's when it acts as switch

Saturation – ON

cutoff – OFF

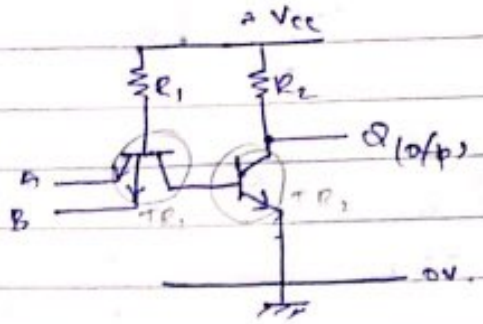
→ But it is hard to bring the transistor from saturation to cut off

So we use active region for ON of switch

(It is easy to by one diode for F.B → R.B)

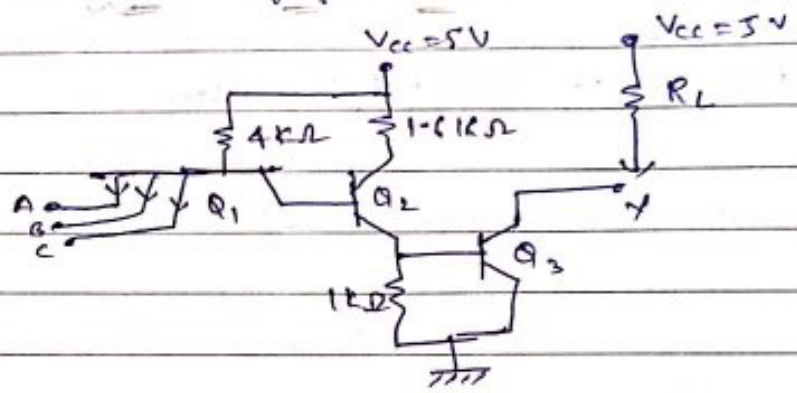
i.e called Schottky Transistor.

2 input NAND Gate :-



when A & B is 0 o/p is high.
as TR1 is R.B
A and B is 1 o/p is low
TR2 is F-B (TR1 drives TR2 Rto saturation)

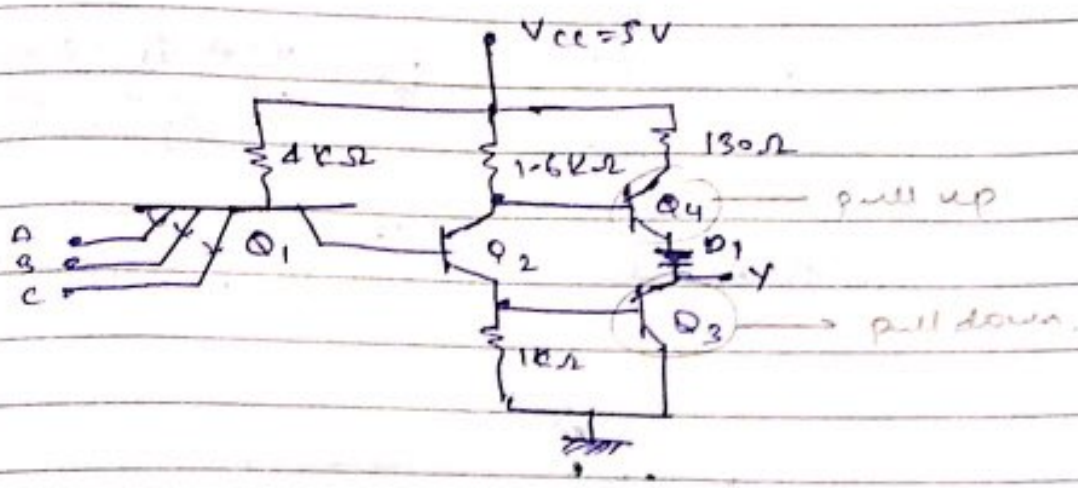
open collector output Gate :-



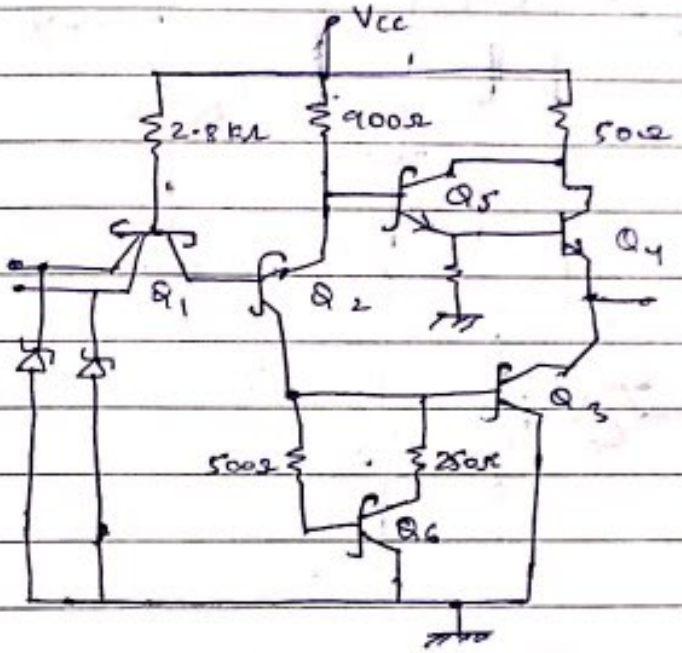
NAND gate

when A & B & C is 0 .o/p is high.
A, B, C is 1 o/p is low. (as Q1 drives Q2, Q3 Rto saturation)

TTL gate with totem-pole output :→



Schottky TTL gate :→

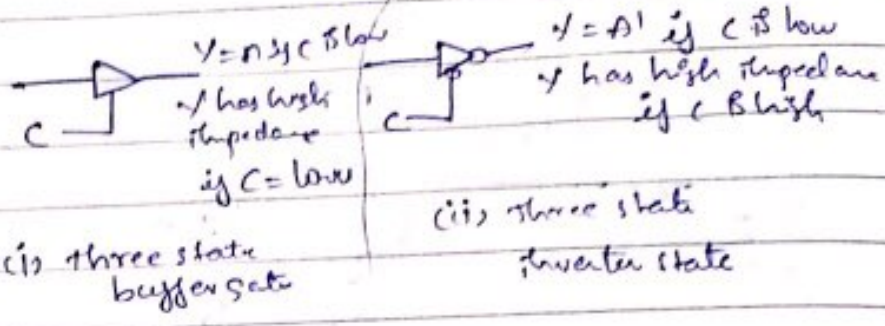


→ has more speed

→ Because schottky Transistor are used
(switches blw active and cut off)

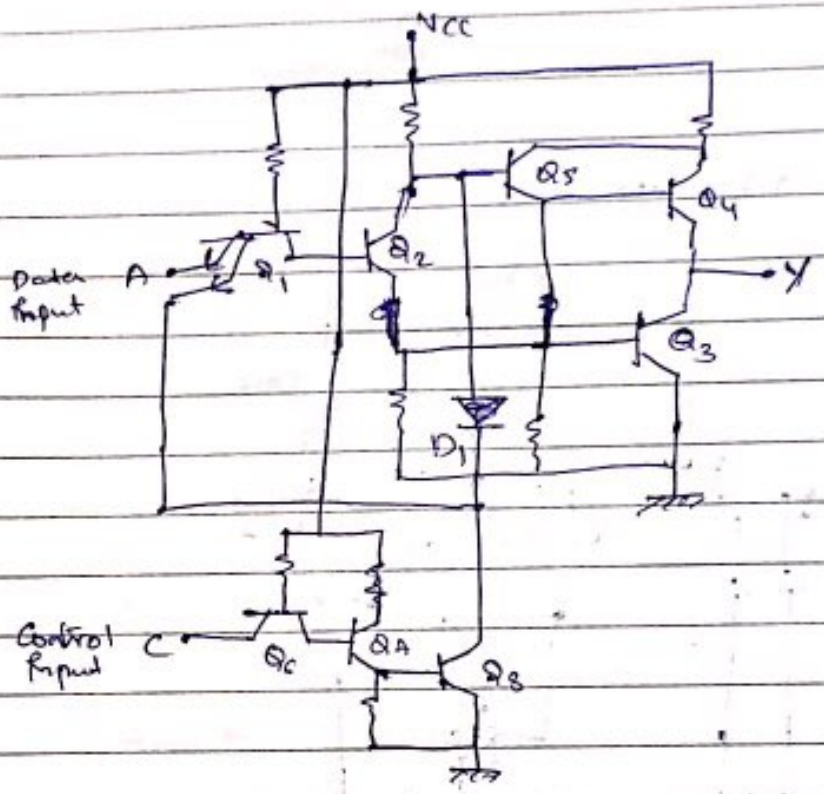
Camlin Page
Date / /

Three state TTL gate :-



(i) three state buffer gate

(ii) three state inverter state



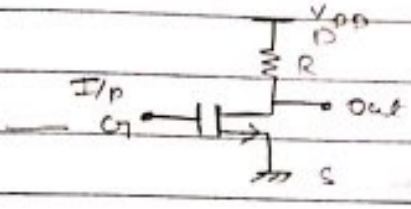
22/09/16

MOS :: Metal Oxide Semiconductor

We prefer NMOS to PMOS because
→ the charge carrier moving are e⁻
and mobility of e⁻ is more than holes
→ easy size reduction
→ less noise compared to pmos.

NMOS logic gates :: → These are positive logic gate devices.

S → low → nMOS
→ high → pMOS.

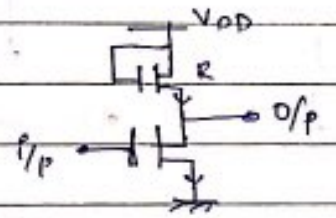


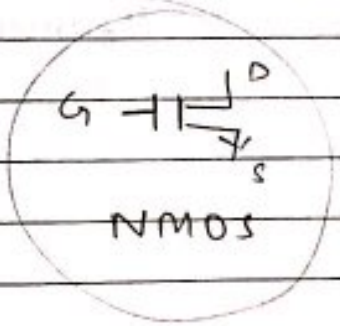
I/P	O/P
0	1
1	0

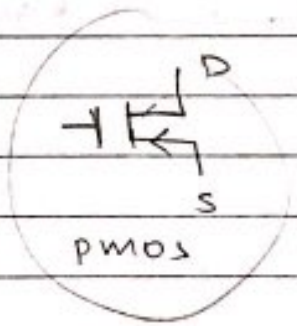
Inverter

PUN → pulls up the O/p to VDD (O/p → supply) (use PMOS)
PDN → pulls down the o/p to ground (O/p → ground)
(use NMOS)

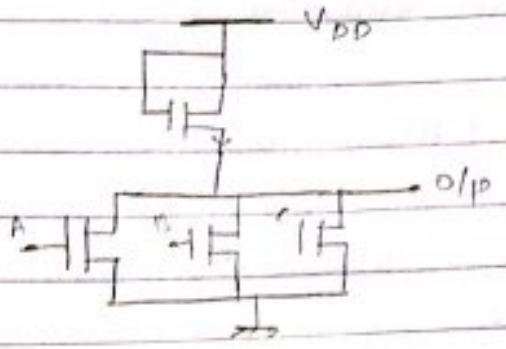
Instead of Resistor R we use nMOS resistor



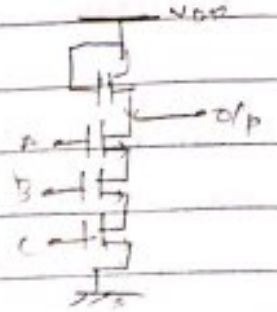




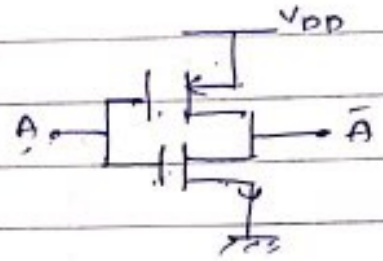
3 i/p NOR gate



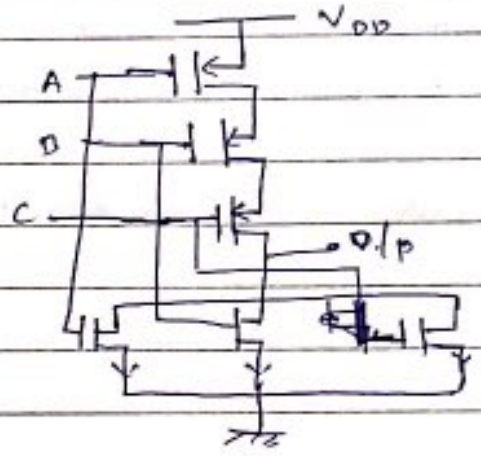
3 i/p NAND gate



CMOS :→ Complementary metal Oxide Semiconductor

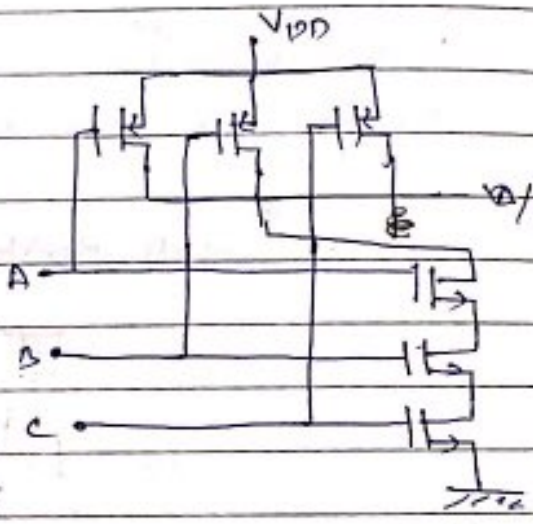


CMOS logic Rules



3 i/p CMOS NOR gate

(pmos → Nel
nmos → Se.)

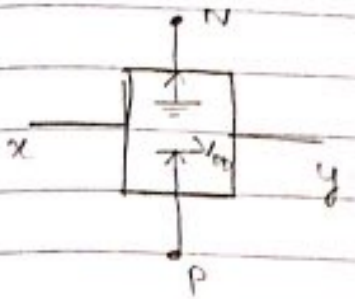


3 i/p CMOS NAND Gate :-

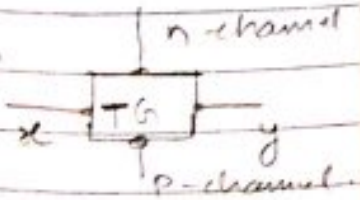
(nmos → 11°1
pmos ↔ (...)

Camlin | Page
Date / /

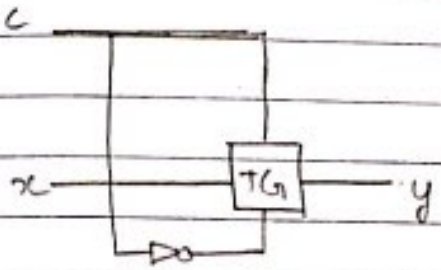
CMOS Transmission gates :-



Logic symbol :-



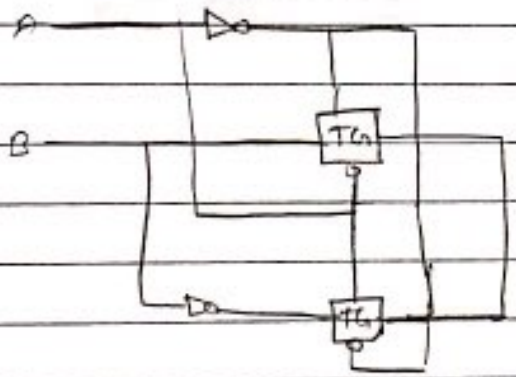
→ It is bilateral switch.
→ It conducts both the sides. (x → y or y → x)



C=0 Switch open (OFF)

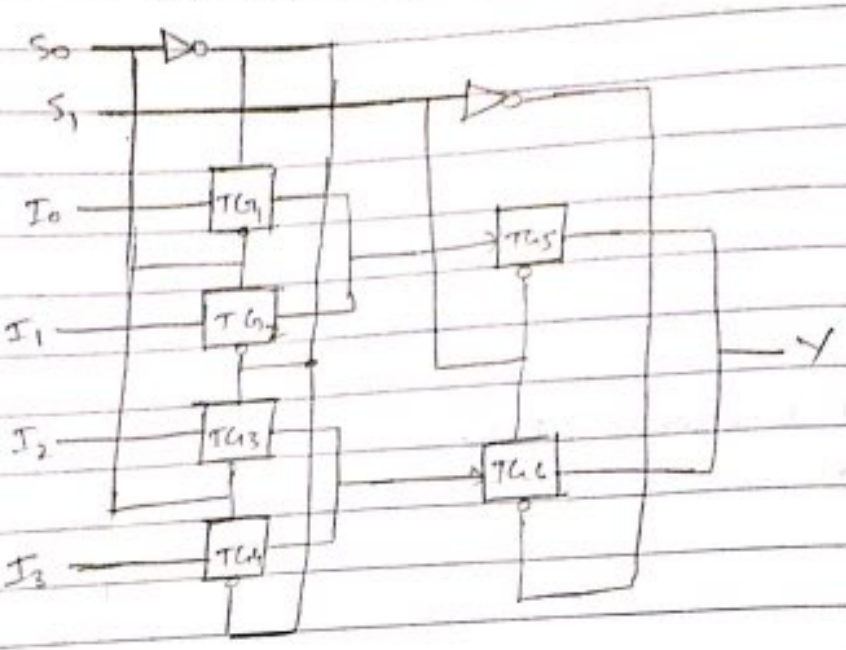
C=1 Switch closed (ON)

EXOR Gate :-

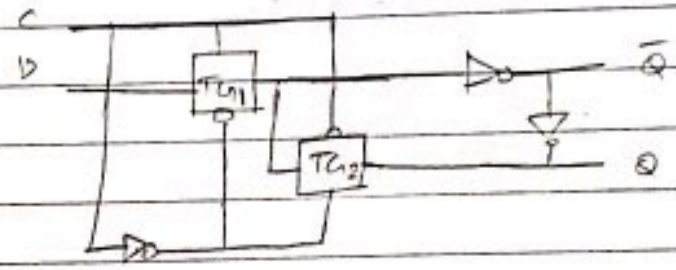


A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

4x1 MUX using T.G:-



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

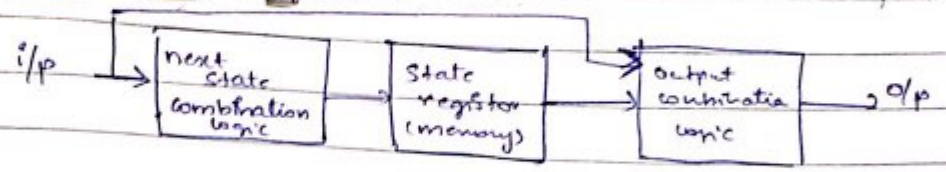


Camlin Page
Date / /

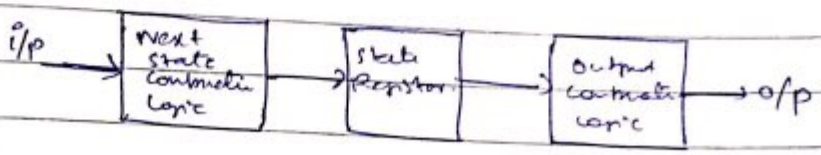
State Machines:-

These are 2 types
- Mealy state machine → o/p depends on present & next
- Moore state machine → o/p depends on present state

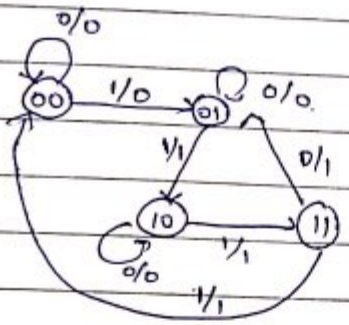
Mealy: block diagram:-



Moore: block diagram

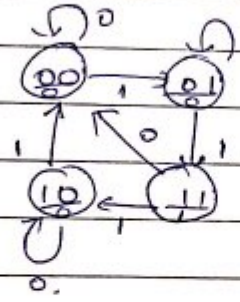


Mealy state diagram:-



→ o/p depends on i/p also

Moore state diagram:-



→ o/p is not influenced by i/p.

→ only influenced by state.

State Equations :→

D, T K and T FF are mostly used.

→ The state equation are on next state, i/p and present state.

For DFF

P.S	i/p	NS
Q_n	D	Q_{n+1}
0	0	0
0	1	1
1	0	0
1	1	1

→ Irrespective of past or present state the o/p is same as i/p i.e D value.

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

for JK FF

Q_n	J	K	Q_{n+1}	
0	0	0	0	memory
0	0	1	0	reset
0	1	0	1	set
0	1	1	1	$\bar{Q}_n$
1	0	0	1	memory
1	1	0	1	set
1	1	1	0	$\bar{Q}_n$
1	0	1	0	reset

Qn \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

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

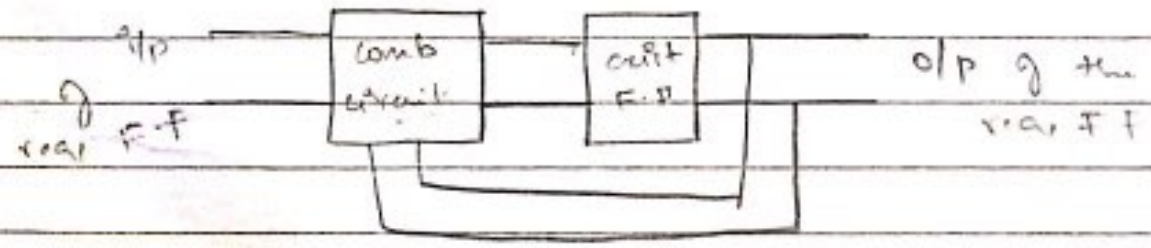
$$Q_{n+1} = Q_n' J + Q_n K'$$

for T FF

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

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

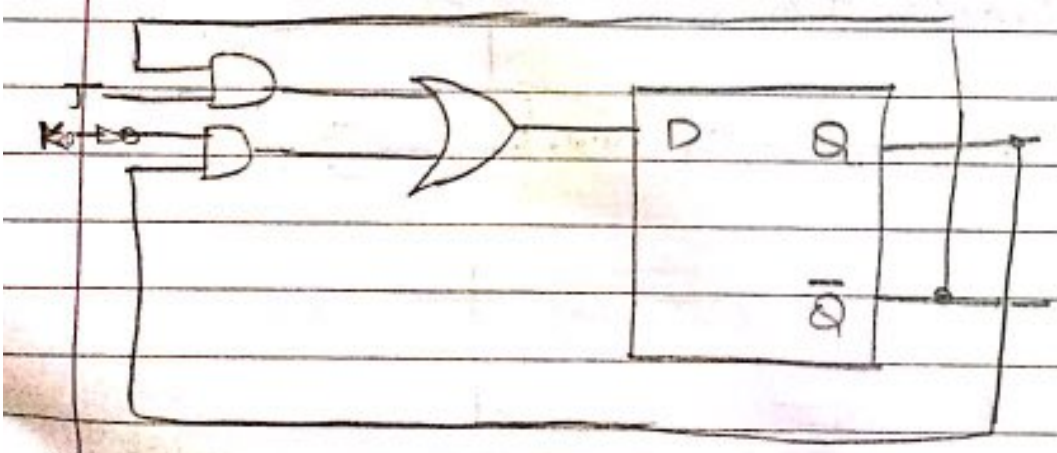
Conversions between flip flops :-



Convert D FF into JK FF

i/p of req F.F		Present state	Next state	i/p of exist FF
J	K	Q_n	Q_{n+1}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

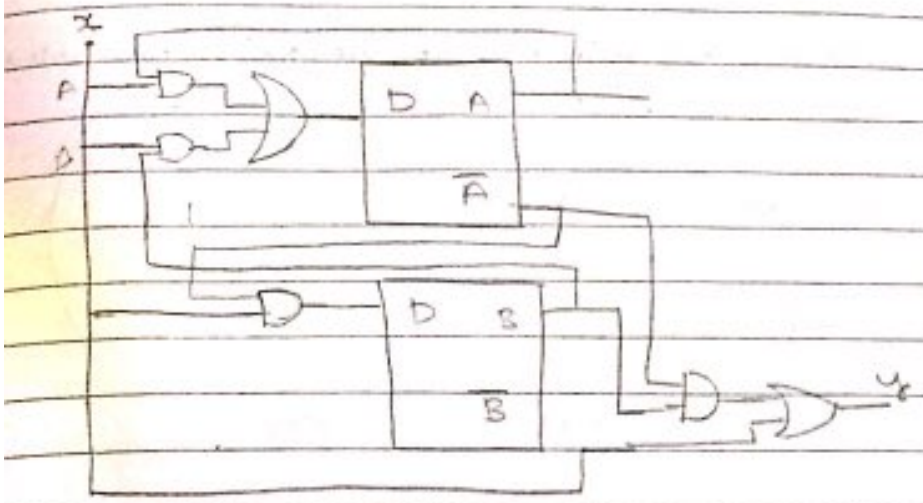
$$D = J\bar{Q}_n + \bar{K}Q_n$$



Camlin Page
Date / /

Sequential circuit Analysis Procedure :-

(i) D- Flip Flop :-



$A(t+1) = A(t)\,x(t) + B(t)\,x(t) = Ax + Bx$

$B(t+1) = \bar{A}(t)\,x(t) = \bar{A}x$

$y(t+1) = \bar{A}B + x(t)\,\bar{A}(t)\,B(t) + x(t)$

State Table :-

Present State		i/p	next state		o/p
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0	0	0	0
1	1	1	1	0	1

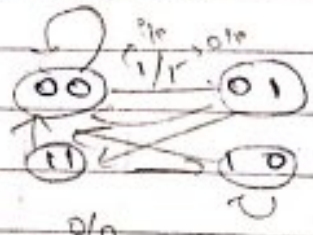
→ This circuit may exist in 4 states

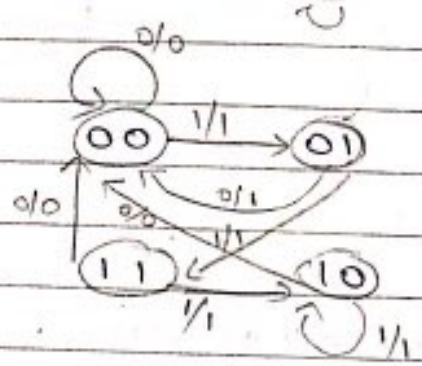
00 01

11 10

→ State diagrams represents the i/p, o/p and etc. for state table

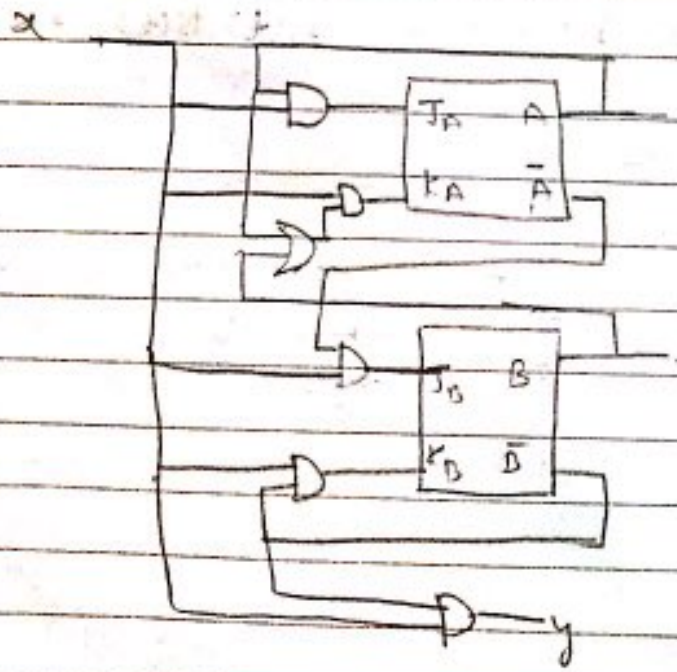
i/p — o/p — s/r





State diagram

(ii) JK - Flip Flop :-)



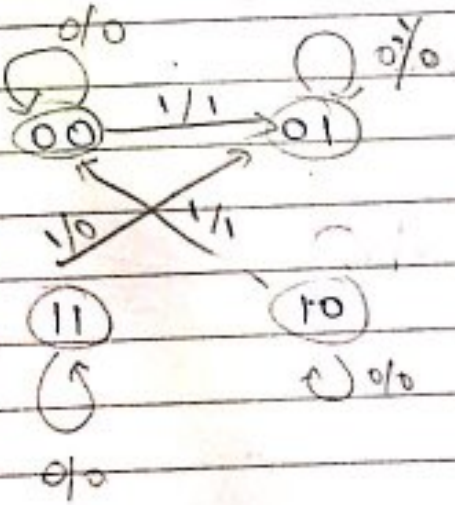
F F i/p eqn

J_A ~~$A(t+1)$~~ $= A(t)\,x(t)$ $\quad K_A = (\bar{A}(t) + B(t))\,x(t)$

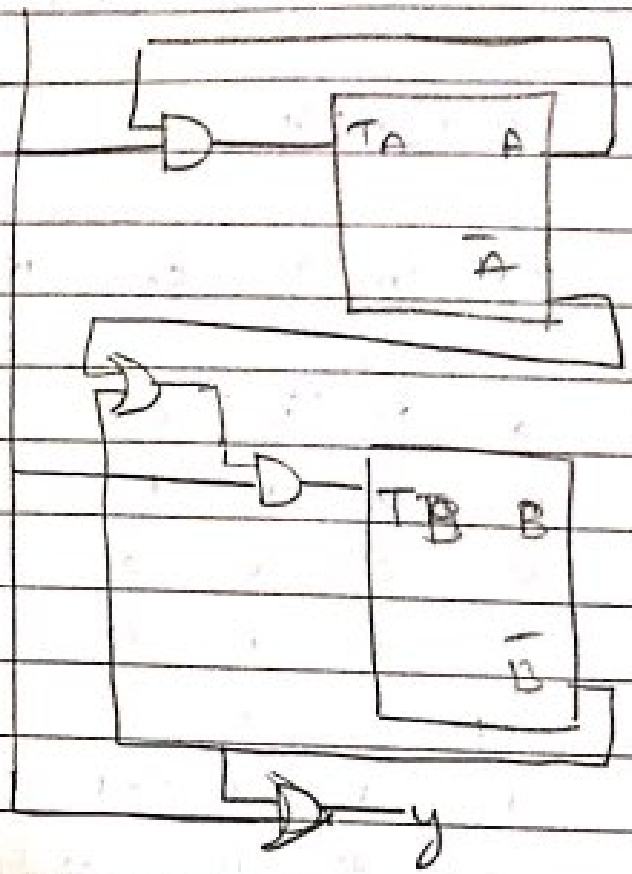
J_B ~~$B(t+1)$~~ $= \bar{A}(t)\,x(t)$ $\quad K_B = x(t)\cdot\bar{B}(t)$

$y = \bar{B}(t)\cdot x(t)$

Present state		i/p	J_A	K_A	J_B	K_B	Next state		o/p
A	B	x					A	B	y
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	1	0	1	0
0	1	0	0	0	0	0	0	1	0
0	1	1	0	1	1	0	0	1	0
1	0	0	0	0	0	0	1	0	0
1	0	1	1	1	0	1	0	0	0
1	1	0	0	0	0	0	1	1	0
1	1	1	1	1	0	0	0	1	0



(iii) T-Flip Flop:-

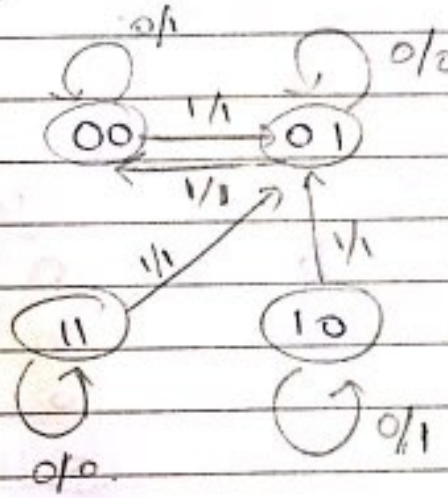


$$T_A = A(t) \cdot x(t)$$

$$T_B = \left[\bar{A}(t) + \bar{B}(t)\right] \cdot x(t)$$

$$y = \bar{B}(t) \cdot x$$

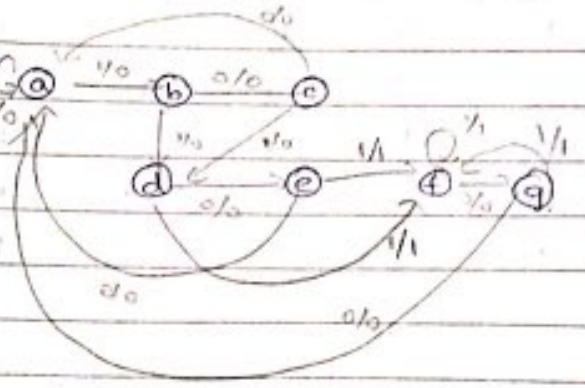
Present state		i/p	T_A	T_B	Next state		o/p
A	B	x			A	B	y
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	0	0	0	0	0
0	1	1	0	1	0	0	0
1	0	0	0	0	1	0	0
1	0	1	1	1	0	1	0
1	1	0	0	0	1	1	0
1	1	1	1	0	0	1	1



Sequential circuit design procedure :-

State Reduction and assignment :-
Choose the states and assignment variables for them

Ex:



→ Reduce the state i.e state reduction

State Table :->

Present state	Next state x=0	Next state x=1	o/p x=0	o/p x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

→ ~~for~~ some combination of i/p, we get same next state and same o/p ; then one of combination can be removed.

1st → e & g
2nd → d & f

Reduced state table :->

P.S	N.S x=0	N.S x=1	O/P x=0	O/P x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

→ The state should be assigned with binary values.
i.e state assignment

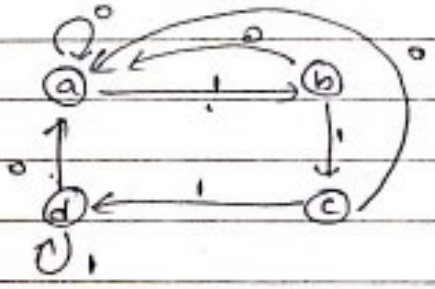
state	binary	gray code	One shot
a	0 0 0	0 0 0	0 0 0 0 1
b	0 0 1	0 0 1	0 0 0 1 0
c	0 1 0	0 1 1	0 0 1 0 0
d	0 1 1	0 1 0	0 1 0 0 0
e	1 0 0	1 1 0	1 0 0 0 0

any 3 these can be used for designing.

Synchronous seqy circuit design:-

(i) From the design objective, the state diagram has to be derived.
(ii) If possible, states are to be reduced.
(iii) For the final states, binary values are to be assigned.
(iv) Then required flip flops are to be chosen.
(v) The flip flop i/p and o/p equation are to be derived.
(vi) The logic diagram of the required synchronous seqy circuit is to be drawn.

Design a synchronous seqy circuit which detects 3 or more consecutive in the sequence of bits.



P.S	N.S		P.S.		i/p	N.S	
	x=0	x=1	A	B		A	B
a	a	b	0	0	0	0	0
					1	0	1
b	a	c	0	1	0	0	0
					1	1	0
c	a	d	1	0	0	0	0
					1	1	1
d	a	a	1	1	0	0	0
					1	0	0

The flip flop chosen is 'D'.

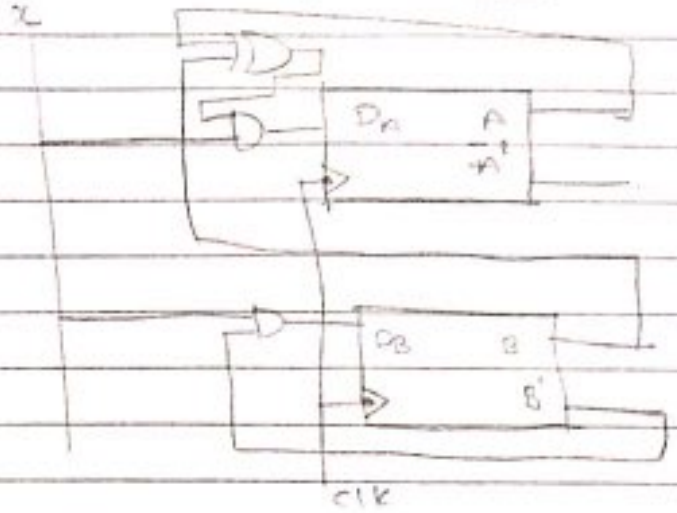
$D_A = AB'x + A'B_2$

$= x(A \oplus B)$

A \ B_2	00	01	11	10
0	0	0	1	0
1	0	1	0	0

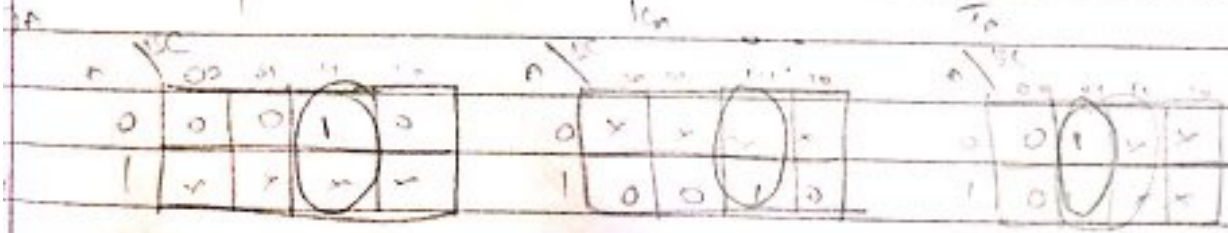
$DB = B'x$

A \ Bx	00	01	11	10
0	0	1	0	0
1	0	1	0	0



Design a synchronous sequential circuit which acts as a 3-bit binary counter.

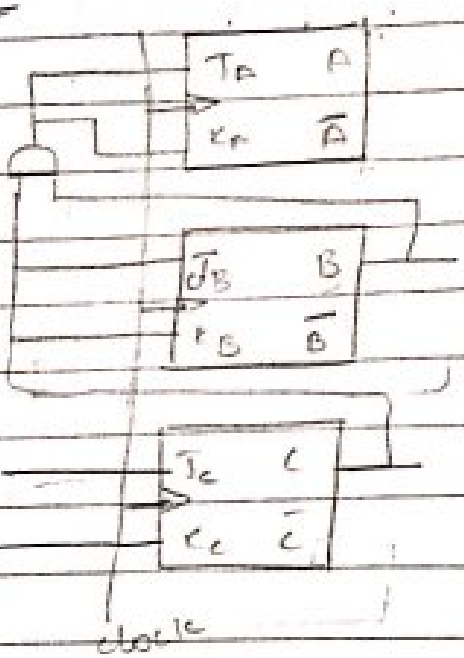
P.S.			N.S										
A	B	C	A	B	C	J_A	K_A	J_B	K_B	J_C	K_C	T_A	T_B	T_C
0	0	0	0	0	1	0	x	0	x	1	x	0	0	1
0	0	1	0	1	0	0	x	1	x	x	1	0	1	1
0	1	0	0	1	1	0	x	x	0	1	x	0	0	1
0	1	1	1	0	0	1	x	x	1	x	1	1	1	1
1	0	0	1	0	1	x	0	0	x	1	x	0	0	1
1	0	1	1	1	0	x	0	1	x	x	1	0	1	1
1	1	0	1	1	1	x	0	x	0	1	x	0	0	1
1	1	1	0	0	0	x	1	x	1	x	1	1	1	1



$J_A = BC$ $K_A = BC$ $J_B = C$

$K_B = C$ $J_C = 1$ $K_C = 1$

Using JK



Using T.

0	0	1	0	$T_A = BC$
0	0	1	0	

0	1	1	0	$T_B = C$
0	1	1	0	

1	1	1	1	$T_C = 1$
1	1	1	1