25/04/'17

**Digital:** Analog — Analog signal: range of values: continuous
Digital — Discrete / fixed value

**Digital logic:** Logic required to design digital ckts.

ON HIGH → (2–5 V) (5 V as symbol)
OFF LOW → (0–0.8 V) (0 V as symbol)

HIGH(5V) = Logic 1
LOW(0V) = Logic 0

All ckts will have input and output as 1 or 0.

DLO → All values in binary.

# Chapter–1

## Number Systems

- Binary, Octal, Decimal, Hexadecimal
- Conversion
- binary addition, subtraction, multiplication
- 1's complement
- 2's complement
  - Until first bit of (1) is reached, keep the bits as it is. Then flip the rest bits. (R → L)

- 1/0 - bit
- 1011 - 4 bits
- IC - Integrated ckt - many ckts implemented together
- SSI - Small scale integration
- MSI - Medium " "
- LSI - Large " "
- VLSI - Very large " "
- 0 - 15 : learn the binary numbers

Advantage of digital ckt - Transmission, processing, storing, of digital signal is easier. (Noise = error)
~~poor detecting~~

DLD
- i) Combinational ckt
- ii) Sequential ckt

Combinational ckt - Output depends on input (current) Eg: multiplication

Sequential ckt - Output depends on both current and previous input. Eg: goals in football

24/04/'17

# Introduction to Basic Logic Operations and Basic Digital Circuits

- **Logic ckt / digital ckt / switching ckt:**

A ckt whose input and output signals are 2 state, either low or high voltages. The basic logic ckt are: OR, AND, NOT gates.

- **Gate:** a logic ckt with one or more input signals but only one output signal.
- **Truth table:** A table that shows all input and output possibilities for a logic ckt. For n inputs, there are $2^n$ input combinations

i) **AND operation:**


Fig: 2 input AND gate

**Logic equation:**

$$Y = AB$$

**Truth table:**

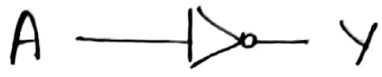
| Inputs | | Output |
|---|---|---|
| A | B | Y |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

ii) OR operation:


$Y = A + B$

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

iii) NOT operation:

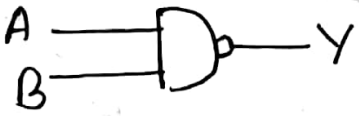



$Y = \bar{A}$

or, $Y = A'$

| A | Y |
|---|---|
| 0 | 1 |
| 1 | 0 |

iv) NAND operation:




$Y = \overline{AB}$

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

v) NOR operation:

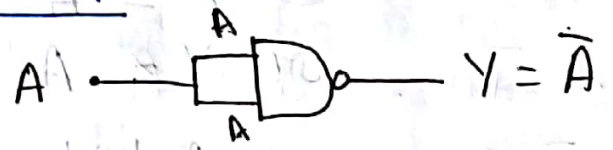

$Y = \overline{A+B}$

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

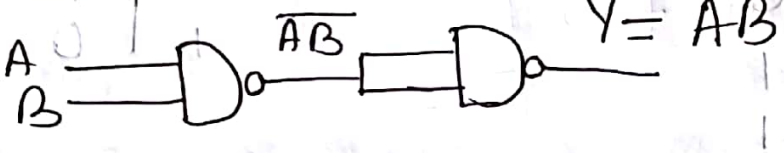
Being the only one type of gates either NAND or NOR are sufficient for the realisation of any logical expression, so they are known as universal gates

- **Realisation of basic logic operations using NAND gates:**

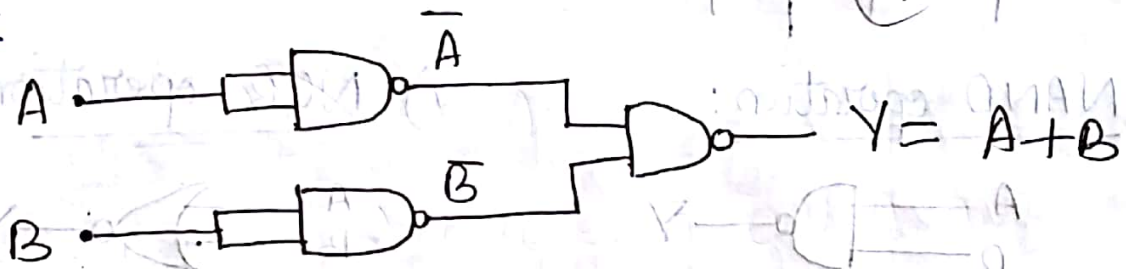
a) **NOT:**




b) **AND:**

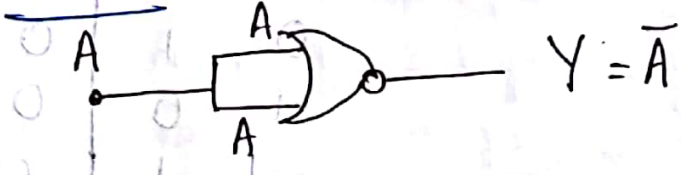



c) **OR:**

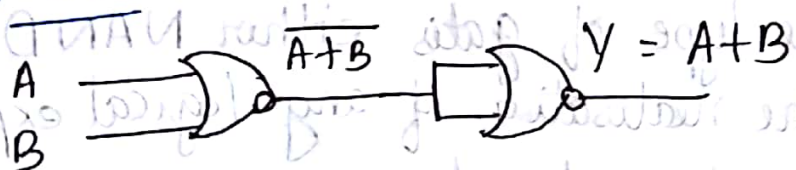



- **H/W:** Realisation of basic logic operations using NOR gates.

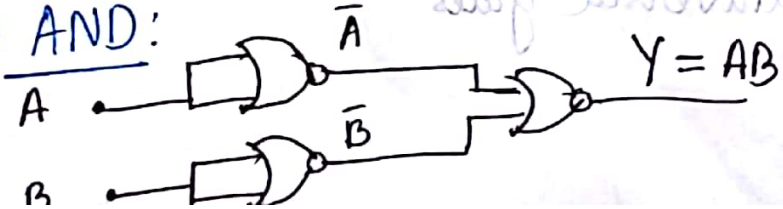
a) **NOT:**



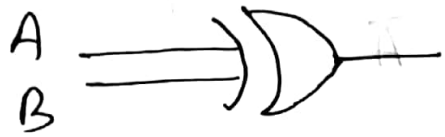

b) **OR:**




c) **AND:**



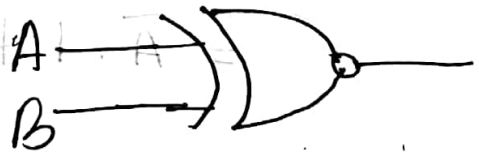
## vi) Exclusive OR (EX-OR / X-OR) operation:




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

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

## vii) Exclusive NOR (Ex-NOR / X-NOR) operation:

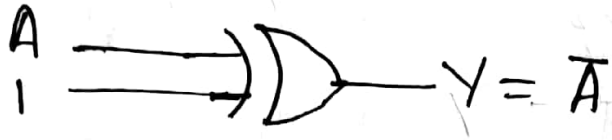



$$Y = \overline{A \oplus B} = A \odot B$$
$$= A'B' + AB$$

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

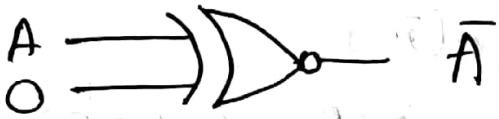
[No. of 1 in input if odd, output = 1
No. of 1 in input if even, output = 0]

- **Realisation of NOT gate using XOR gate:**




$$A \oplus B = A\bar{B} + \bar{A}B$$
$$= A.\bar{1} + \bar{A}.1$$
$$= A.0 + \bar{A}$$
$$= \bar{A}$$

- **Realisation of NOT gate using X-NOR gate:**




$$A \odot B = A'B' + AB = \bar{A}.\bar{0} + A.0$$
$$= \bar{A}.1 + 0 = \bar{A}$$

30/04/'1

# Boolean Algebra & Logic Simplification:

1854 – George Boole

Definitions: variable, boolean, complement, literal

## • Laws of Boolean Algebra:

① a) $A + B = B + A$ [Commutative laws]

b) $AB = BA$

② a) $A + (B + C) = (A + B) + C$ [Associative laws]

b) $A(BC) = (AB)C$

③ a) $A(B + C) = AB + AC$ [Distributive laws]

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

## • De Morgan's Theorems:

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

[See the proofs from book]

## * Rules of Boolean Algebra

1) $A + 0 = A$ 2) $A + 1 = 1$

3) $A \cdot 0 = 0$ 4) $A \cdot 1 = A$

5) $A + A = A$ 6) $A + \overline{A} = 1$

7) $A \cdot A = A$ 8) $A \cdot \overline{A} = 0$

9) $\overline{(\overline{A})} = A$ 10) $A + AB = A$

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

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

[See details from book]

## * Duality Principle:

It states that every algebraic expression deducible from the postulates of Boolean Algebra ~~contains~~ remains valid if the operators and the identity elements are interchanged.

Eg: if $A + \bar{A} = 1$,
then $A \cdot \bar{A} = 0$
if $A + 1 = 1$, then $A \cdot 0 = 0$

[AND $\longleftrightarrow$ OR
0 $\longleftrightarrow$ 1]

Prove ~~Proof~~ that:

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

L.H.S = $A + \bar{A}C$

$= (A + \bar{A})(A + C)$ [Distributive law]

$= 1 \cdot (A + C)$ [$\because A + \bar{A} = 1$]

$= A + C$

$= R.H.S.$

Alternate form:

| A | C | $\bar{A}C$ | $A + \bar{A}C$ | $A + C$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |

## * Simplification using Boolean Algebra:

i) $AB + A(B+C) + B(B+C)$

Solution:

$= AB + AB + AC + BB + BC$ (Distributive law)

$= AB + AC + B + BC$ [$\because BB = B$, $AB + AB = AB$]

$= AB + AC + B$ [$\because B + BC = B$]

$= B + AC$ [$\because AB + B = B$]

Ans: $B + AC$

ii) $[A\bar{B}(C+BD) + \bar{A}\bar{B}]C$

Solution:

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

$= (A\bar{B}C + A.0.D + \bar{A}\bar{B})C$ [$\because B\bar{B} = 0$]

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

$= A\bar{B}CC + \bar{A}\bar{B}C$ [Distributive law]

$= A\bar{B}C + \bar{A}\bar{B}C$ [$\because CC = C$]

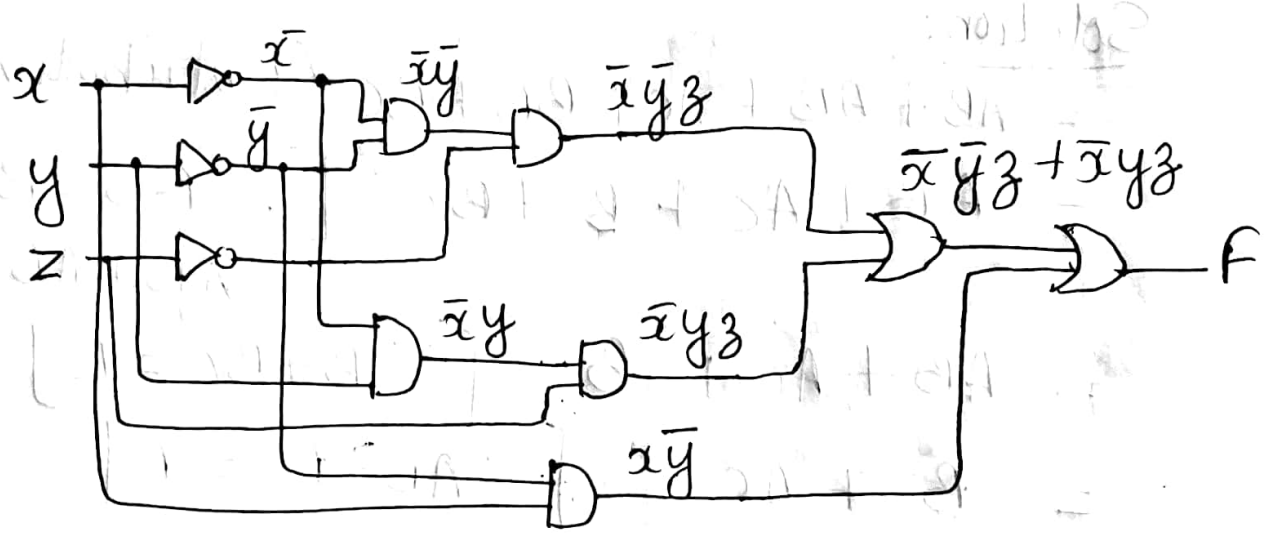
$= \bar{B}C(A + \bar{A})$

$= \bar{B}C.1$ [$\because A + \bar{A} = 1$]

Ans: $\bar{B}C$

- **Boolean Functions:**

Implement $f = \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}$




Truth table:

| x | y | z | F |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

$\bar{x}\bar{y}z$ = 1 only when 0 0 1.

$\bar{x}yz$ = 1 only when 0 1 1

$x\bar{y}$ = 1 only when 1 0

Since OR, thus checking only one term is sufficient.

Variable : symbol holding logic value (0 or 1)

Boolean : true or false holding of variables

Complement : inverted or / not form

Literal : variable that may be itself or its complement

01/05/17

- Binary codes for the decimal digits

i) BCD (Binary coded decimal) or 8-4-2-1 code

- In this code, decimal digits 0 through 9 are represented by their natural binary equivalents using 4 bits and each decimal digit of a decimal number is represented by this 4 bit code individually. For example:

| Decimal | Binary | BCD |
|---|---|---|
| 0 | 0000 | 0000 |
| 5 | 0101 | 0101 |
| 9 | 1001 | 1001 |
| 25 | 11001 | 00100101 |

ii) Excess-3 Code

The code for each decimal digit is obtained by adding decimal 3 to the natural BCD code of the digit.

Example:

| Decimal | Excess-3 code |
|---|---|
| 7 | 1010 |
| 6 | 1001 |

iii) 2, 4, 2, 1 Code:

| Decimal | 2-4-2-1 code |
|---|---|
| 7 | 0111 |
| 7 | 1101 |
| 9 | 1111 |
| 3 | 1001 |
| 3 | 0011 |

iv) 8, 4, -2, -1 code:

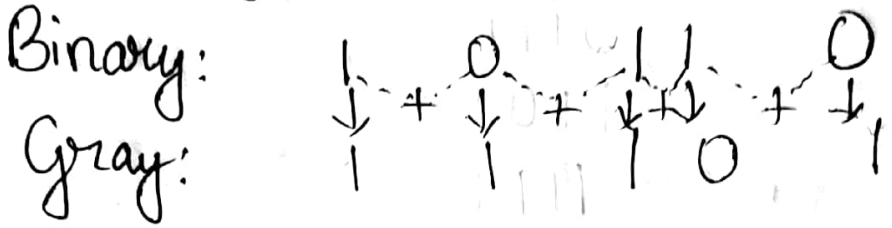
| Decimal | 8, 4, -2, -1 code |
|---|---|
| 1 | 0111 |
| 6 | 1010 |

- Coding Conversion

| Decimal | 2, 4, 2, 1 code | 8, 4, -2, -1 code |
|---|---|---|
| 0 | 0000 | 0000 |
| 1 | 0001 | 0111 |
| 2 | 0010 | 0110 |
| 3 | 0011 | 0101 |

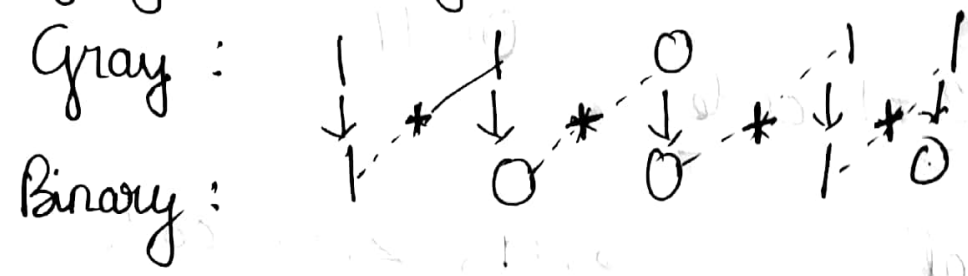
v) The Reflected Code or Gray Code

a) Binary to Gray Conversion.




0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0

b) Gray to binary conversion:




- Error – detection codes:

Parity bit: A parity bit is an extra bit included with a message to make the total number of 1's either odd (odd parity) or even (even parity)

Say,

| message: | odd parity | even parity: |
|---|---|---|
| 10110 | 0 | 1 |

Tues Quiz Chap 1 & 2

7/05/'17

- Hamming Code:

~~Hamming c~~

Hamming code is a simple error ~~dect~~ detection-correction technique. It can be used to detect & correct one bit ~~ca~~ change in an encoded code word.

Example: Determine the single error correcting code for '1011010'.

Solution:

Total no. of bits in the original message, $n = 9$.

Say, we require $p$ parity bits. We know,

$2^p \geq n + p + 1$ → $9 + 4 + 1 = 14$, $2^4 = 16$

$\therefore p = 4$

So, we will send

$n + p = 13$ bits

Sending end:

| bit position | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ~~$P_1$~~ | ~~$P_2$~~ | ~~$P_3$~~ | ~~$P_4$~~ | | | | | | | | | |
| $2^0$ → | $P_1$ | $P_2$ ← $2^1$ | 1 | $P_3$ ← $2^2$ | 0 | 1 | 1 | $P_4$ ← $2^3$ | 1 | 0 | 1 | 1 | 0 |

↳ parity bits are positioned in powers of 2

0 0 1 1 (3)
0 1 1 0 (6)
0 1 1 1 (7)

1 0 0 1 (9)
1 0 1 1 (11)
(XOR) 1 1 0 0 (12)

$P_4$ — 1 1 0 0 → $P_1$
$P_2$ $P_3$

So, the sending message (hamming coded message) will be:

0 0 11 0 111 0 11 0

Receiving End:

Case 1: If no error occurs during transmission, we will get: 0011 0111 0110 to check:

0011 (3)
0100 (4)
0110 (6)
0111 (7)
1000 (8)
1001 (9)
1011 (11)
(XOR) 1100 (12)
0000

So, there is no error.

Case 2: Say, 6th position bit is altered during transmission & we get:

0 0 11 0 0 111 0 11 0

to check:

0011 (3)
0100 (4)
0111 (7)
1000 (8)
1001 (9)
1011 (11)
(XOR) 1100 (12)
0110 (6)

as, we got $(0110)_2$, i.e. 6 after XOR operation, so there is an error in 6th position. So, the original sending message was: 0011011110110

8/05/'17

# Chapter - 2

## Standard forms of Boolean Expressions:

- Standardization makes the evaluation, simplification and implementation of Boolean expressions much more systematic and easier. All Boolean expressions can be converted into either of 2 standard forms

### 1. The SUM OF PRODUCTS (SOP) FORM:

Eg: $AB + A\bar{B}C$, $ABC + CDE + B'CD'$ etc.

### 2. THE PRODUCT OF SUM (POS) FORM:

$(A'+B)(A+\bar{B}+C)$,
$(A'+B'+C')(C+D'+E)(B'+C+D)$ etc.

- Canonical FORM:

Boolean functions expressed as a sum of minterms or products of maxterms are said to be in canonical form.

Eg:

| | A B C | F |
|---|---|---|
| Maxterm | 0 0 0 | 0 |
| $M_0$ $A+B+C$ | 0 0 1 | 1 |
| $M_1$ $A+B+C'$ | 0 1 0 | 0 |
| | 0 1 1 | 0 |
| $M_7 = A'+B'+C'$ | 1 0 0 | 1 |
| | 1 0 1 | 1 |
| | 1 1 0 | 1 |
| | 1 1 1 | 1 |

Minterms

$m_0$ $A'B'C'$
$m_1$ $A'B'C$
$m_7$ $ABC$

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

or,

$F = \Pi(0,2,3)$

- Determing standard expressions from a truth table:

Example 1: $F(A,B,C,) = \Sigma(1,4,5,6,7)$

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

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

$= A'B'C + AB' + AB$ $[\because C'+C = 1]$

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

$= A'B'C + A$ $[\because B+B' = 1]$

$= (A+A')(A+B'C)$ [Applying distributive law]

$= A + B'C$ $[\because A+A' = 1]$

Ans: $= A + B'C$

Example 2: $F(A,B,C) = \Pi(0,2,3)$

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

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

$= (A + AB' + AC + AB + 0 + BC + B'C + C)(A+B'+C')$

$[\because B\bar{B} = 0,\ A+A = 0,\ \eta A = \bar{A}]$

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

$= (A + AC + C)(A + \bar{B} + \bar{C})$ $[\because A + A = A]$

$= (A + C)(A + B' + C')$ $[\because A + AC = A]$

$= AA + AB' + AC' + AC + B'C + CC'$

$= A + AB' + A + B'C$ $[\because C + \bar{C} = 1,\ C\bar{C} = 0]$

$= A + A + B'C$ $[\because A + AB' = A]$

$= A + B'C$ $[\because A = A = A]$

Ans. $A + B'C$

[$\Sigma \rightarrow$ SOP, $\Pi$ – POS]

[m → minterm, M → maxterm]

09/05/17

- Conversion between minterm & maxterm:
- SOP to POS conversion:

Example 1: $F(A,B,C) = A'BC' + A'BC + ABC' + ABC$

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

$= \Pi(0,1,4,5)$

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

Ans: $(A+B+C)(A+B+C')(A'+B+C)(A'+B+C')$

Example 2: $F(A,B,C) = A + B'C$

$= A(B'C' + B'C + BC' + BC) + B'C(A+A')$ $[\because X + X' = 1]$

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

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

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

$= \Pi(0,2,3)$

$= (A+B+C)(A+B'+C)(A+B'+C')$ (Ans.)

* Whichever var. will be missing, all of their combinations have to be AND with the existing var.

$B'C' + BC' + B'C + BC$

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

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

$= 1 \cdot 1 = 1$

- Converting POS to SOP:

Example: $F(A,B,C) = (A+B'+C)(A'+B)$

$= AA' + AB + A'B' + BB' + A'C + BC$

$= AB + A'B' + A'C + BC \quad [\because XX' = 0]$

$= AB(C+C') + A'B'(C+C') + A'C(B+B') + BC(A+A') \quad [\because X+X' = 1]$

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

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

$= AB(C+C') + A'B'(C+C') + A'BC \quad [\because X+\bar{X}=1]$

$= AB + A'\{(B'+BC)\}$

$= AB + A'\{(B'+B)(B'+C)\} \quad [\because x+yz = (x+y)(x+z)]$

$= AB + A'(B'+C) \quad [\because B'+B = 1]$

$= AB + A'B' + A'C$ (Ans.)

* If $F(x,y,z) = 1$ that means,

$F = \Sigma(0,1,2,3,4,5,6,7)$

$= \pi(NIL)$

Quiz upto here (16/05/17)

* Whichever var. is missing, i.e. x, the $(x + x')$ is to be AND with existing var.

14/05/17

# Chapter - 3

## Karnaugh Map (K-Map):

A K-map is similar to a truth table because it presents all of the possible values of input variable and the resulting output for each value.

- **Two variable K-Map:**

No. of grids = $2^n$, where n = no. of variables

| | |
|---|---|
| $m_0$ | $m_1$ |
| $m_2$ | $m_3$ |

- **Three variable K-map:**

| | | | |
|---|---|---|---|
| $m_0$ | $m_1$ | $m_3$ | $m_2$ |
| $m_4$ | $m_5$ | $m_7$ | $m_6$ |

- **Four variable K-map:**

| | | | |
|---|---|---|---|
| $m_0$ | $m_1$ | $m_3$ | $m_2$ |
| $m_4$ | $m_5$ | $m_7$ | $m_6$ |
| $m_{12}$ | $m_{13}$ | $m_{15}$ | $m_{14}$ |
| $m_8$ | $m_9$ | $m_{11}$ | $m_{10}$ |

i)

| Input | | Output |
|---|---|---|
| x | y | F |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

- Simplify using K-map (SOP minimization):

i) $F(x,y) = \Sigma(1,2,3)$

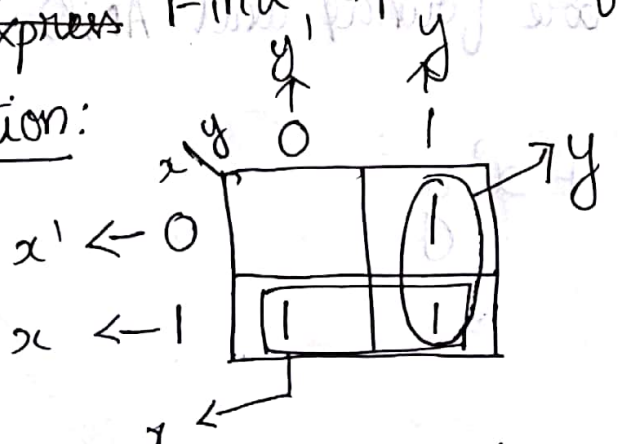
ii) $F(x,y,z) = \Sigma(2,3,4,5)$

iii) $F(A,B,C,D) = \Sigma(0,1,2,4,5,6,8,9,12,13,14)$

- Groups can be made horizontally or vertically.
- Groups can be of 2 or 4 (for two variable).
- Always form biggest group possible.
- Always try to put all 1s in a group.
- Denote each group with a different symbol.
- ~~Express~~ Find expression of each group and OR them

Solution:

i)

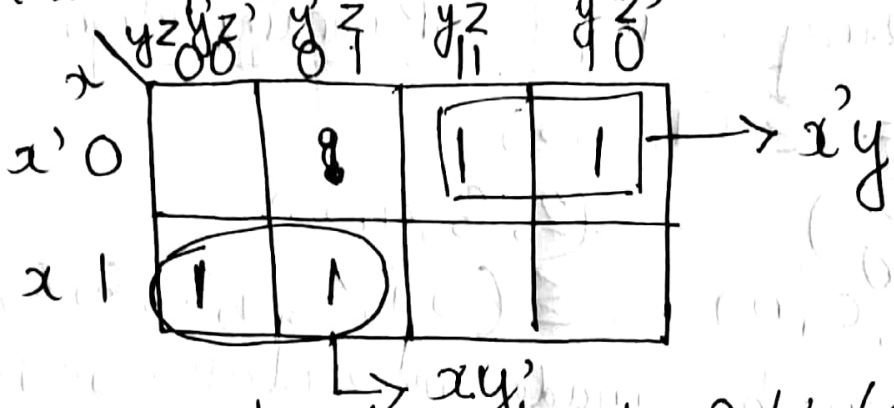



Ans: $x + y$

- K-map can be used to simplify boolean expressions.

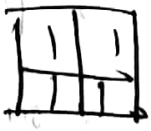
- If all outputs are 1, then the simplified expression is simply 1.

ii) - MSB (Most significant bit is placed at lower part, i.e. denotes, columns.

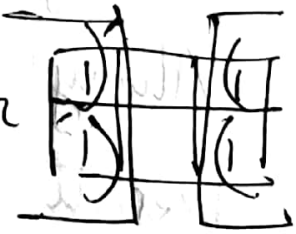



- Groups can be formed of 2/4/8.
- Horizontal / vertical groups, also in same if 1 is at two ends, they can be grouped as well.
- Groups of 4 can be:


or



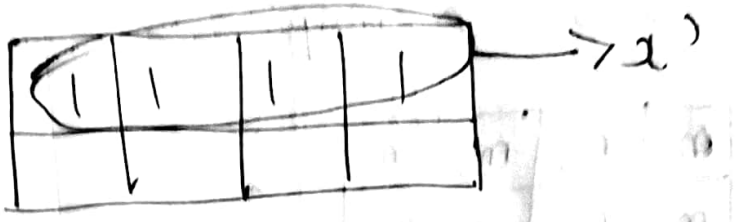

or



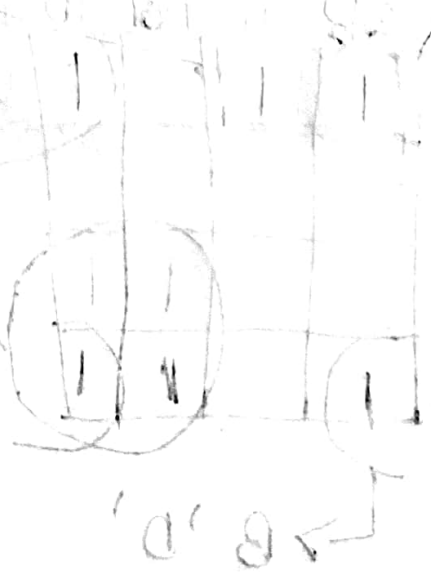

- Common variables are founded and ANDed.

∴ Ans: $xy' + x'y$

Eg:




- No common variables from columns.



15/05/'17

- **Four-variable K-map:**

| | | | |
|---|---|---|---|
| $m_0$ | $m_1$ | $m_3$ | $m_2$ |
| $m_4$ | $m_5$ | $m_7$ | $m_6$ |
| $m_{12}$ | $m_{13}$ | $m_{15}$ | $m_{14}$ |
| $m_8$ | $m_9$ | $m_{11}$ | $m_{10}$ |

- Simplify the following functions using K-map:

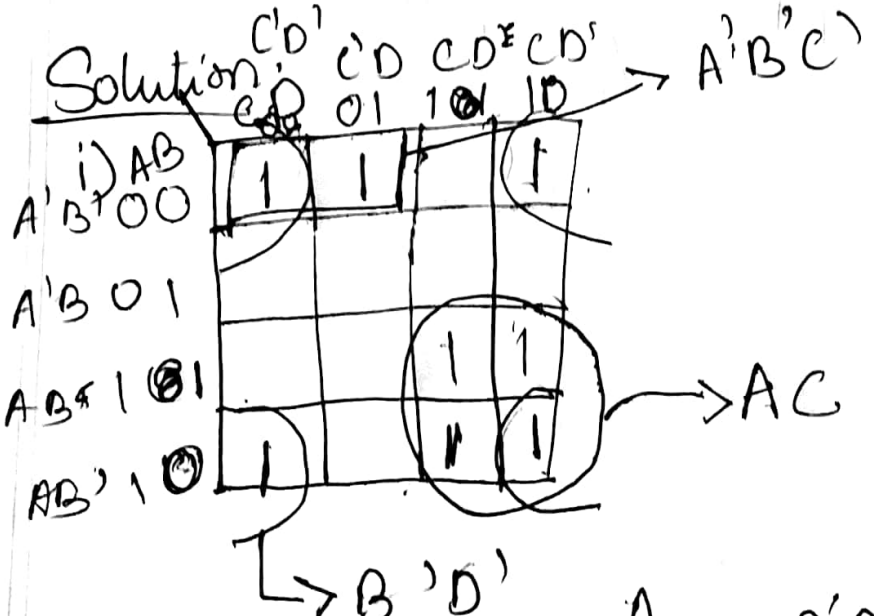
i) $F = \Sigma(0, 1, 2, 8, 10, 11, 14, 15)$

ii) $f = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

iii) $f = A'B'C' + B'CD' + A'BCD' + AB'C'$

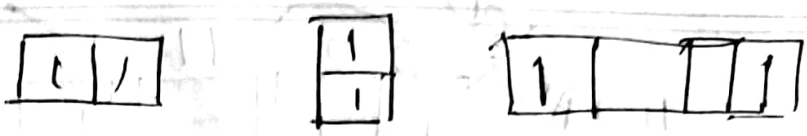
– Check max$^m$ minterm, eg. 15 — 4 bits needed ∴ 4 var

Solution:

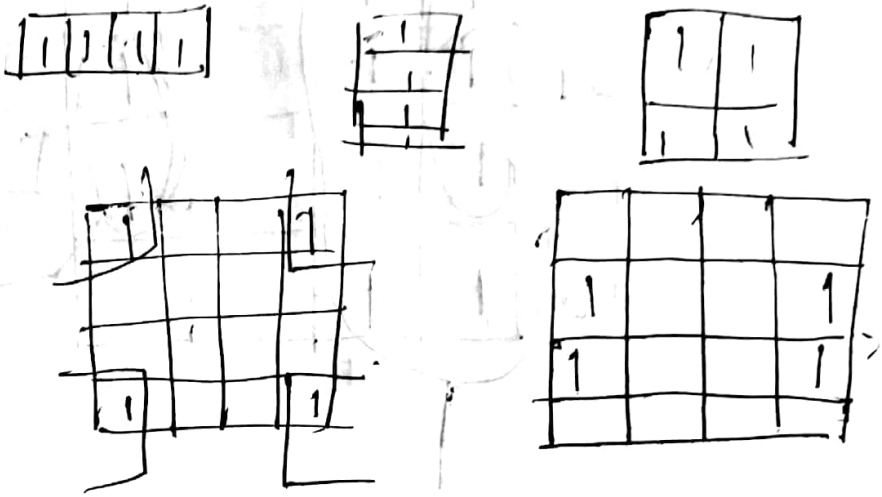



Ans: $B'D' + AC + A'B'C'$

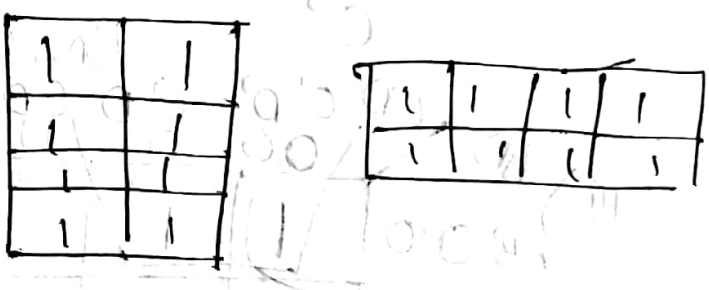
- Groups of 2 →



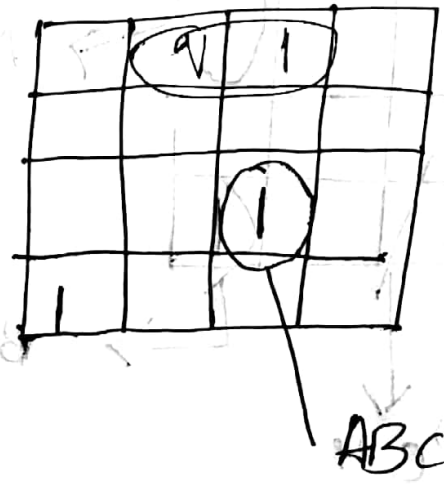
- Groups of 4 →



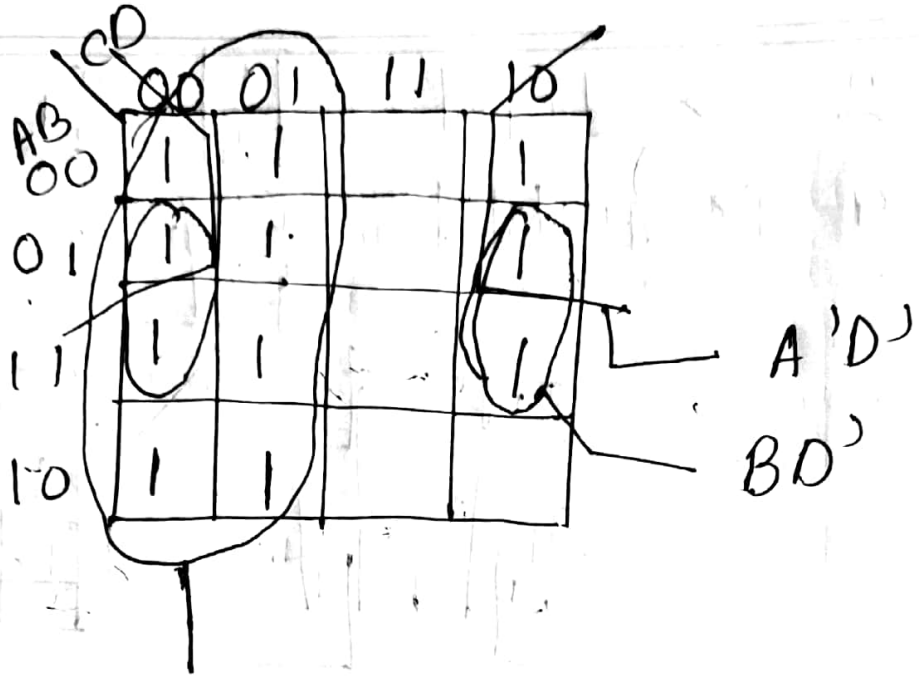
- Groups of 8 →



Eg:

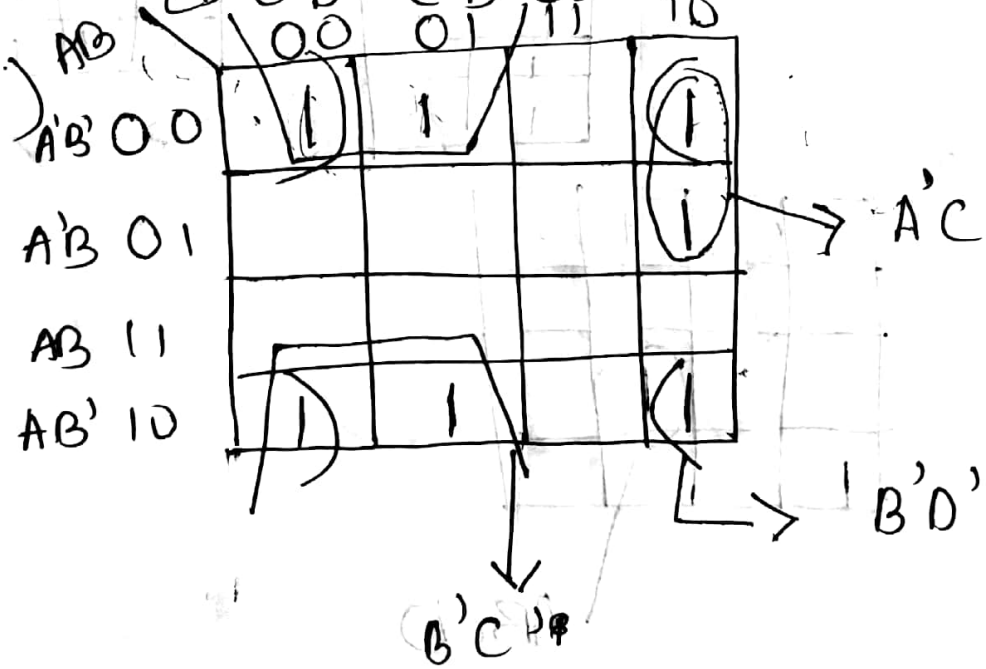


ii)




Ans: $C' + A'D' + BD'$

iii)

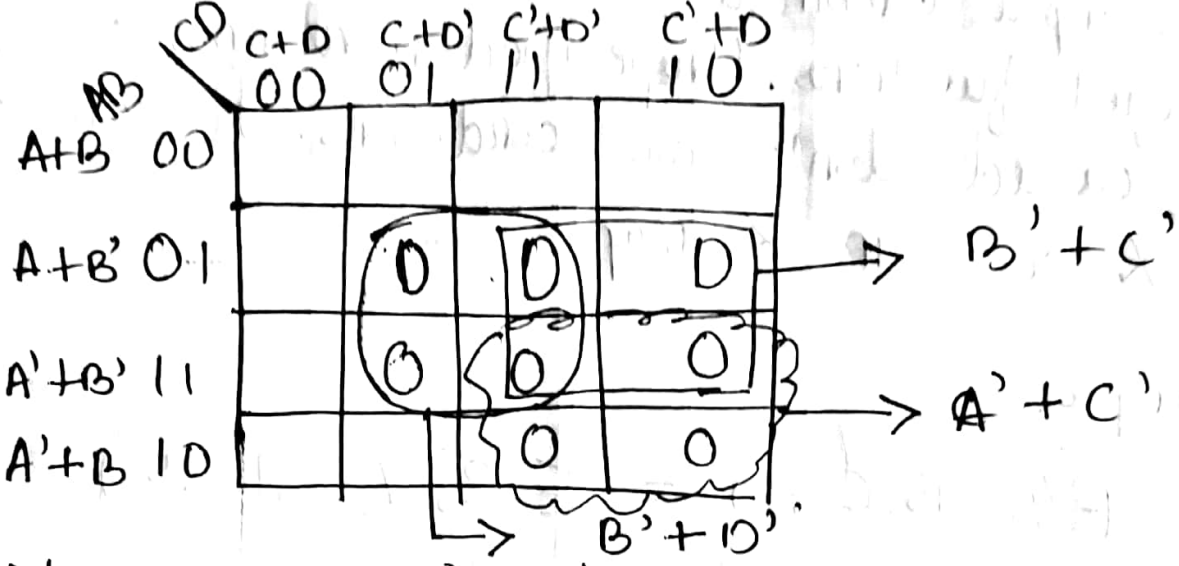



Ans: $A'CD' + B'C' + B'D'$

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

- **Karnaugh Map POS Minimization**

$F = \Pi(5, 6, 7, 10, 11, 13, 14, 15)$




- Maxterms (0) given.
- POS format. 0 → not prime, 1 → prime

∴ Ans: $(B'+D')(A'+C')(B'+C')$

21/05/'16

- **Don't care conditions:**

Functions that have unspecified output for some input combinations are called incompletely specified functions. Unspecified minterms of a function are called 'don't care' conditions. It can be represented by X or d in K-map.

Example:

i) $F(A, B, C, D) = \Sigma(3, 7, 8, 11, 15)$

$d = \Sigma(0, 2, 5)$

ii) $F(A, B, C, D) = \Sigma(1, 3, 7, 11, 15)$

$d = \Sigma(0, 2, 5)$

iii) $F = A'B'D' + A'CD + A'BC$

$d = A'BC'D + ACD + AB'D'$

Eg: BCD → Excess 3 code.

0 – 9

0000 → ——

⋮ ⋮

1001 → ——

∴ ~~for 10,~~ Using 4 var K-map, for 10, 11, 12, 13, 14, 15 and 16, use 'X' or 'd'

- 5 variable K-map:

| AB \ CDE | C'D'E' 000 | C'D'E 001 | C'DE 011 | C'DE' 010 | CDE' 110 | CDE 111 | CD'E 101 | CD'E' 100 |
|---|---|---|---|---|---|---|---|---|
| A'B' 00 | 0 | 1 | 3 | 2 | 6 | 7 | 5 | 4 |
| A'B 01 | 8 | 9 | 11 | 10 | 14 | 15 | 13 | 12 |
| AB 11 | 24 | 25 | 27 | 26 | 30 | 31 | 29 | 28 |
| AB' 10 | 16 | 17 | 19 | 18 | 22 | 23 | 21 | 20 |

- Example: Simplify using K-map.

$F = \Sigma(0, 2, 4, 6, 9, 11, 13, 15, 17, 21, 25, 27, 29, 31)$

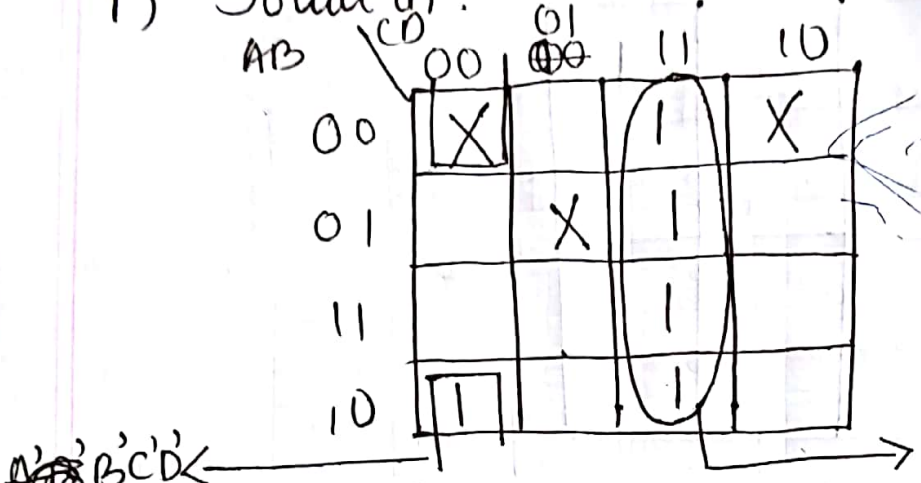
- 6 variable K-map:

| ABC \ DEF | 000 | 001 | 011 | 010 | 110 | 111 | 101 | 100 |
|---|---|---|---|---|---|---|---|---|
| 000 | 0 | 1 | 3 | 2 | 6 | 7 | 5 | 4 |
| 001 | 8 | 9 | 11 | 10 | 14 | 15 | 13 | 12 |
| 011 | 24 | 25 | 27 | 26 | 30 | 31 | 29 | 28 |
| 010 | 16 | 17 | 19 | 18 | 22 | 23 | 21 | 20 |
| 110 | 48 | 49 | 51 | 50 | 54 | 55 | 53 | 52 |
| 111 | 56 | 57 | 59 | 58 | 62 | 63 | 61 | 60 |
| 101 | 40 | 41 | 43 | 42 | 46 | 47 | 45 | 44 |
| 100 | 32 | 33 | 35 | 34 | 38 | 39 | 37 | 36 |

Simplify $F = \Sigma(6, 9, 13, 18, 19, 25, 27, 29, 41, 45, 57, 61)$

(H.W)

1) Solution:

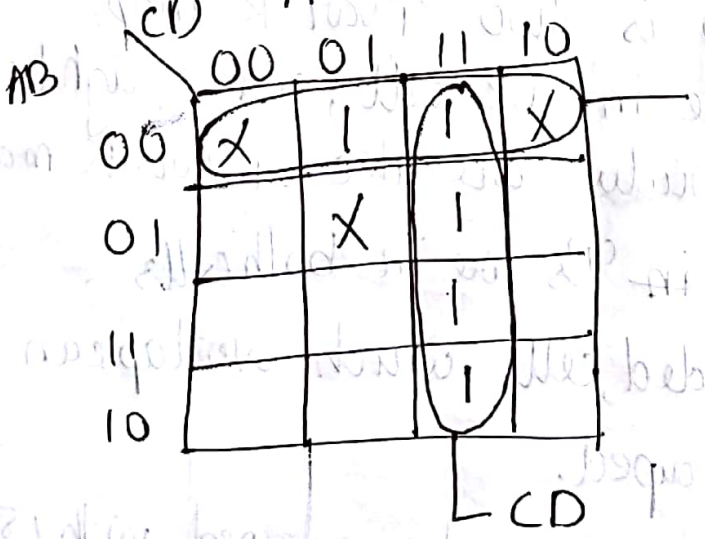



X – may be 0 or 1

- don't care may or may not be grouped. It will be grouped only if a '1' can be grouped with it.

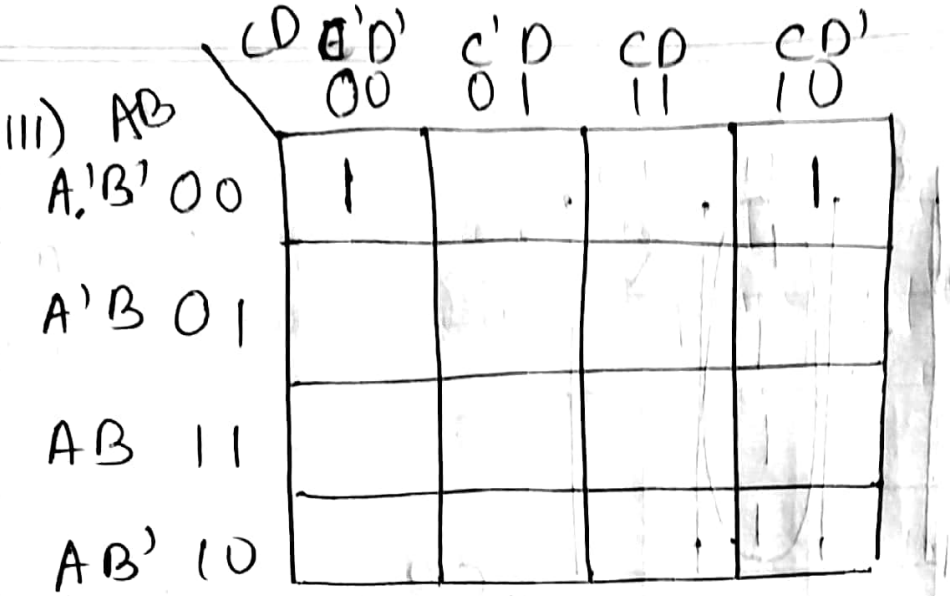
Ans: $CD + B'C'D'$

ii)




Ans: $A'B' + CD$

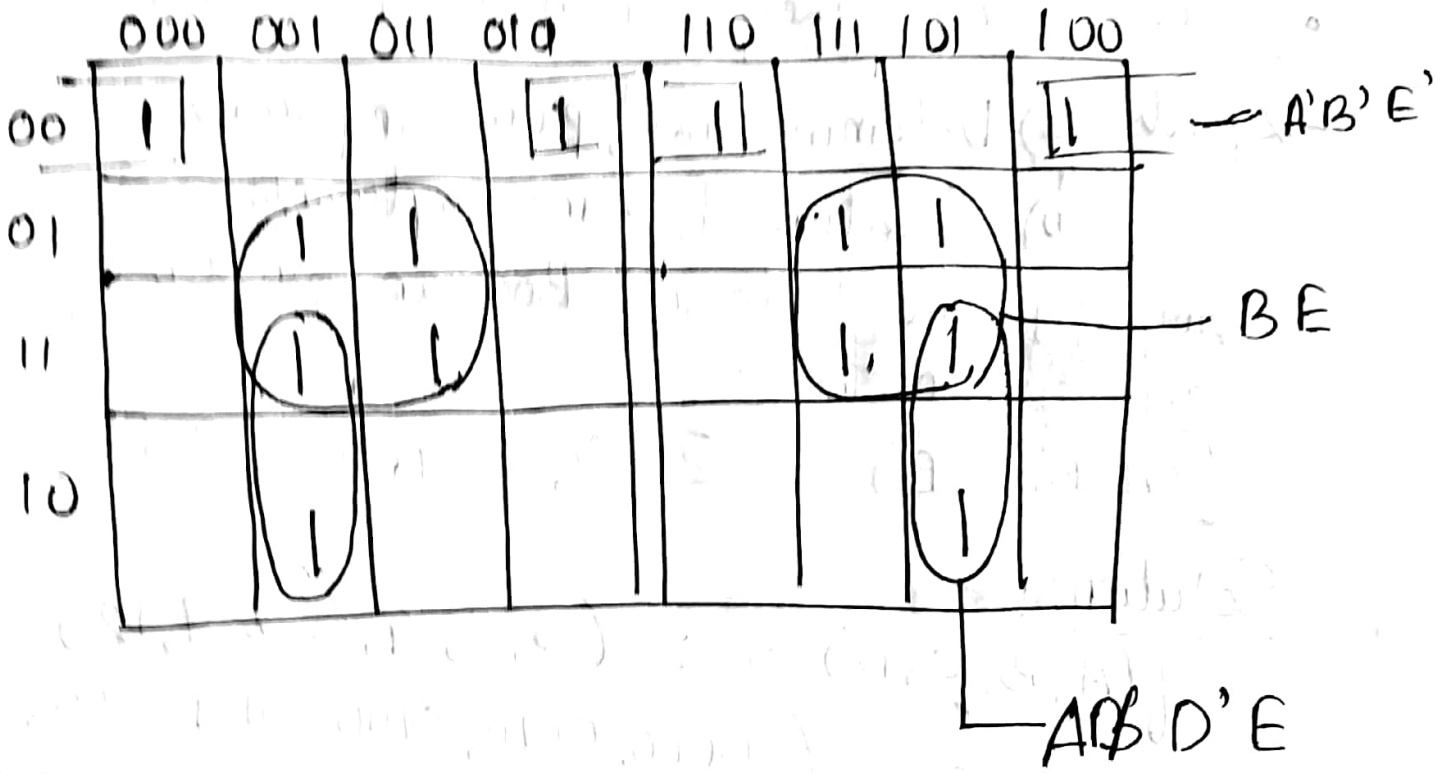
iii)




Solution:

- 5 var K-map is two 4 var K-map.
- If all 1's are in one cell, either right or left, grouping rules are like 4 var K-map.
- If mixed, ie. ~~in~~ 1's are in both cells. When folded, cells which overlap can be grouped.

Eg: 31 th pos can be grouped with 15, 30, 29, 23 and 27.




$$Ans = A'B'E + BE + BD'E$$

- For 6 var. K-map, take whichever ones are common for two folds.

N

22/05/'17

## • Quine - McCluskey / Tabulation Method:

- 2 parts: a) Determination of prime implicants,
b) Selection of prime implicants

Simplify the following Boolean function by using the tabulation method:

$f(A, B, C, D) = \Sigma(0, 2, 4, 5, 8, 9, 12)$

**Solution:**

$f(A, B, C, D) = \Sigma(0, 2, 4, 5, 8, 9, 12)$

$= \Sigma($ 0000, 0010, 0100, 0101, 1000, 1001, 1100) [binary equivalent]

**Step 1:** Determination of prime implicants.

(a)

| | A | B | C | D |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 |
| 8 | 1 | 0 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 |
| 12 | 1 | 1 | 0 | 0 |

0 is ← 0
1 is ← 4
2 is ← 8

(b)

| | A | B | C | D | |
|---|---|---|---|---|---|
| (0, 2) | 0 | 0 | _ | 0 | → No. of diff. bits = 1 |
| 0, 4 | 0 | _ | 0 | 0 | |
| 0, 8 | _ | 0 | 0 | 0 | |
| (4, 5) | 0 | 1 | 0 | _ | → No. of diff. bits = 1 |
| 4, 12 | _ | 1 | 0 | 0 | |
| (8, 9) | 1 | 0 | 0 | _ | |
| 8, 12 | 1 | _ | 0 | 0 | |

↳ compared with immediate above group.

Use those not used from (b) and that of (c)

(c)

| | A | B | C | D |
|---|---|---|---|---|
| 0, 4, 8, 12 | — | — | 0 | 0 |
| 0, 8, 4, 12 | — | — | 0 | 0 |

So, prime implicants:
$A'B'D' + A'BC' + AB'C' + C'D'$

↳ Check between groups for — in same position and with diff. of only 1 bit.

Step 2: Selection of prime implicants

Place — in place of diff. bit.

| ~~A'B'D'(0,2)~~ | 0 | 2 | 4 | 5 | 8 | 9 | 12 |
|---|---|---|---|---|---|---|---|
| A'B'D'(0,2) | X | X | | | | | |
| A'BC'(4,5) | | | X | X | | | |
| AB'C'(8,9) | | | | | X | X | |
| C'D'(0,4,8,12) | X | | X | | X | | X |

∴ Essential prime implicants: $A'B'D' + A'BC' + AB'C' + C'D'$

There is no unchecked minterm value, so the unsimplified answer is:

$F(A, B, C, D) = A'B'D' + A'BC' + AB'C' + C'D'$ (Ans)

↳ Choose the single crosses from each column.
- Final ans must have all essential prime implicants

# Chapter -4

## Combinational Logic

A combinational circuit consists of logic gates whose outputs at any time are determined directly from the present combination of inputs without regard to previous inputs.

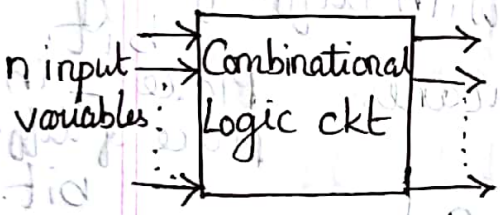



Fig: Block diagram of combinational ckt.

- **Design procedure of combinational logic ckt:**

① The problem is stated

② The no. of available input variables & required output variables is determined.

③ The input and output variables are assigned letter symbols.

④ The truth table is derived.

⑤ The simplified Boolean function for each output is obtained.

⑥ The logic diagram is drawn.

Example: Design a 3-bit square circuit.

Solution: Say input variables: $A, B, C$
output variables: $F_1, F_2, F_3, F_4, F_5, F_6$

- Truth table:

| Input | | | Output | | | | | |
|---|---|---|---|---|---|---|---|---|
| A | B | C | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

- ~~Boolean functions:~~

- Boolean functions

$$F_1 = \Sigma(6,7) = AB\bar{C} + ABC = AB$$

$$F_2 = \Sigma(4,5,7) = A(\bar{B}+C)$$

$$F_3 = \Sigma(3,5) = C(A \oplus B)$$

$$F_4 = \Sigma(2,6) = B\bar{C}$$

$$F_5 = 0, \quad F_6 = C$$

- **Block diagram:**

Do it yourself

[Quiz #2
13/06/17 – Tuesday]

29/05/'17

**Carry in:**

$$\begin{array}{r} {\scriptstyle +1} \\ 101 \\ +001 \\ \hline 110 \end{array}$$

**Carry out:**

$$\begin{array}{r} {\scriptstyle +1} \\ 110 \\ +011 \\ \hline 1001 \end{array}$$

- **Half adder**: adds 2 bits and produces a sum & a carry output.

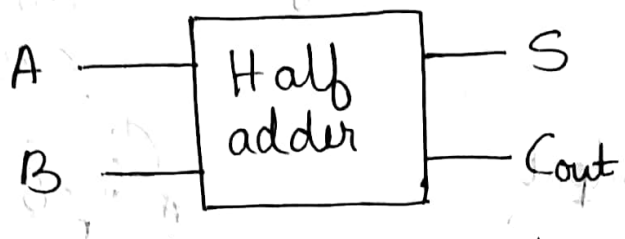
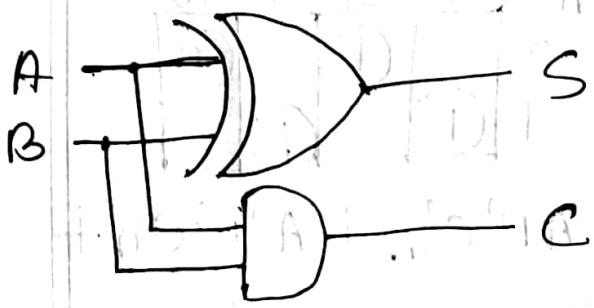



Fig: block diagram of a half adder

**Truth table**

| Input | | Output | |
|---|---|---|---|
| A | B | C | S |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

**Circuit diagram:**




**Output equations:**

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

$C = AB$

• Full adder : accepts 2 input bits & an input carry and generates a sum output & an output carry.

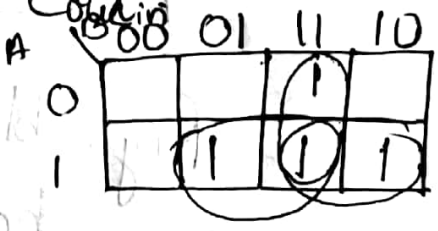

Fig: Block diagram

Truth table

| Input | | | Output | | |
|---|---|---|---|---|---|
| A | B | Cin | Cout | S | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 2 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 2 |
| 1 | 1 | 0 | 1 | 0 | 2 |
| 1 | 1 | 1 | 1 | 1 | 3 |

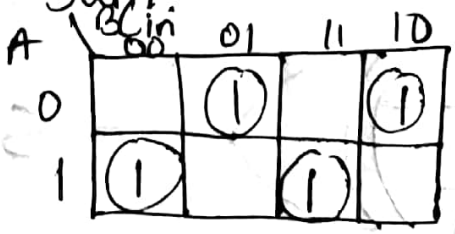
Output Equations:

Cout:




$C_{out} = AC_{in} + AB + BC_{in}$

Sum:




$S = AB'C'_{in} + A'B'C_{in} + ABC_{in} + A'BC'_{in}$

$= C'_{in}(AB' + A'B) + C_{in}(A'B' + AB)$

$= C'_{in}\,x + C_{in}\,x'$

$= C_{in} \oplus x$

## Ckt diagram:

Do it yourself

$= C_{in} \oplus (AB' + A'B)$

$= C_{in} \oplus (A \oplus B)$

Say,

$x = AB' + A'B$

$\therefore x' = A'B + AB$

## * Self study

i) Arrangement of 2 half adders to form a full adder

ii) half subtractor

iii) full subtractor

iv) carry look ahead adder

- **Binary parallel adder:** (definition)

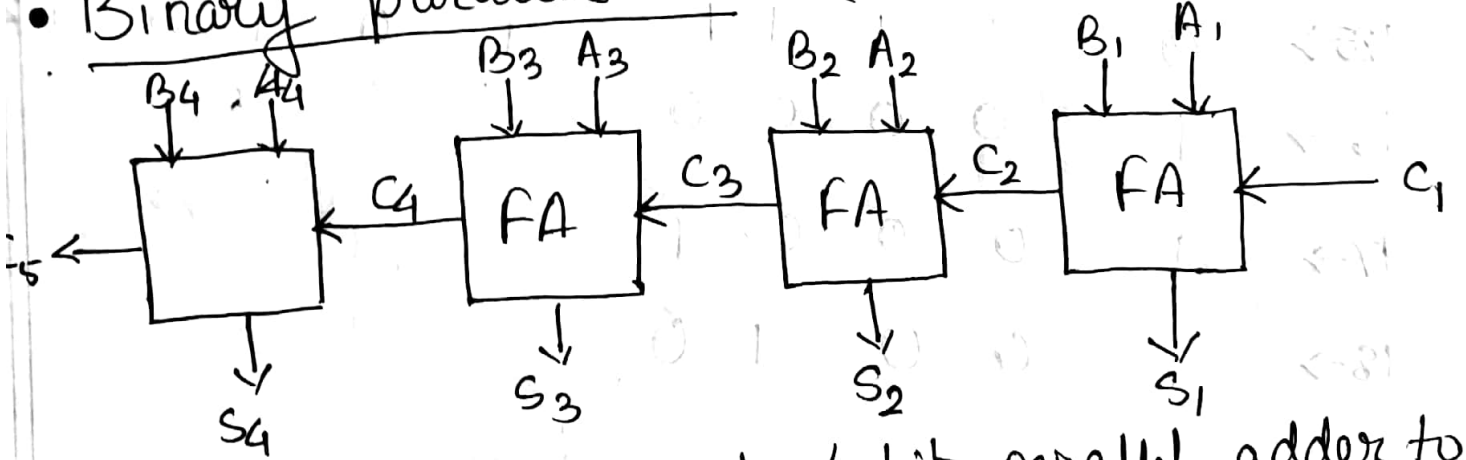



Fig: Block diagram of 4 bit parallel adder to add $(A_4 A_3 A_2 A_1)$ and $(B_4 B_3 B_2 B_1)$

30/05/17

• **BCD Adder** :- A BCD adder is a circuit that adds two BCD digits in parallel & produces a sum digit also in BCD.

| | Binary | | | | | BCD | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Cout | $S_4$ | $S_3$ | $S_2$ | $S_1$ | Cout | $S_4$ | $S_3$ | $S_2$ | $S_1$ |
| 0 → | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 → | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 10 → | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 (16) |
| 11 → | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 (17) |
| 12 → | 0 | 1 | 1 | 0 | 0 | 01 | 0 | 0 | 1 | 0 (18) |
| 15 → | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 (21) |
| 16 → | 1 | 0 | 0 | 0 | 0 | | | | | |
| 17 → | 1 | 0 | 0 | 0 | 1 | | | | | |
| 18 → | 1 | 0 | 0 | 1 | 0 | | | | | |

10 – 15

$$X = C_{out} + \overline{S_4 (S_3 + S_2)}$$

$$X = C_{out} + S_4 S_3 + S_4 S_2$$

16 – 18

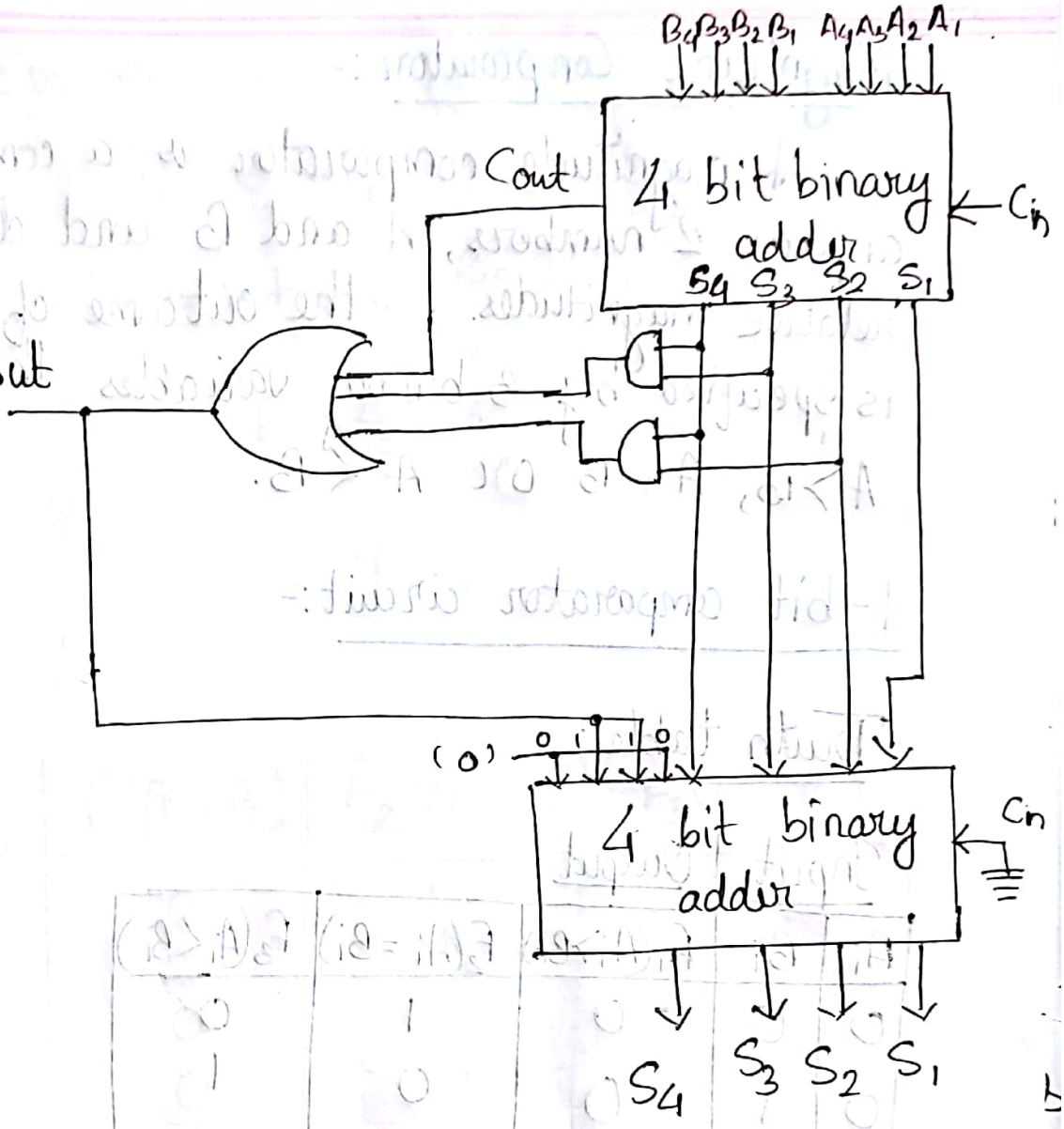



Fig: block diagram of a BCD adder.

* Maximum number that can be output in BCD $\Rightarrow$ 9 + 9.

4/06/17

## Magnitude Comparator :-

A magnitude comparator is a combinational ckt that compares 2 numbers, A and B and determines their relative magnitudes. — the outcome of the comparison is specified by 3 binary variables that indicate whether $A > B$, $A = B$ or $A < B$.

## 1-bit comparator circuit :-

### Truth table :

| Input | | Output | | |
|---|---|---|---|---|
| $A_i$ | $B_i$ | $F_1 (A_i > B_i)$ | $F_2 (A_i = B_i)$ | $F_3 (A_i < B_i)$ |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |

Output equations: $F_1 = A_i B_i'$

$F_2 = A_i' B_i' + A_i B_i$

$= A_i \odot B_i$

$F_3 = A_i' B_i$

Ckt diagram: Draw it yourself

## Two bit comparator:

$A = A_2 A_1 \qquad B = B_2 B_1$

$F_1 (A > B) = (A_2 > B_2) + (A_2 = B_2)(A_1 > B_1)$

| $A_2$ | $A_1$ | $B_2$ | $B_1$ | $F_1$ | $F_2$ | $F_3$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | | | |
| ⋮ | ⋮ | ⋮ | ⋮ | | | |
| 1 | 1 | 1 | 1 | | | |

| A | | B | | $F_1(A>B)$ | $F_2(A=B)$ | $F_3(A<B)$ |
|---|---|---|---|---|---|---|
| $A_2$ | $A_1$ | $B_2$ | $B_1$ | | | |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |

Use minterms to find equations

$F_2 (A \leq B) = (A_2 < B_2) + (A_2 = B_2)(A_1 < B_1)$

$F_3 (A < B) = A_2' B_2 + (A_2 \odot B_2)(A_1' B_1)$

$F_3 (A = B) = (A_2 = B_2)(A_1 = B_1) = (A_2 \odot B_2)(A_1 \odot B_1)$

5/06/17

## Multiplexer / MUX / Data Selector:

- A digital multiplexer is a combinational ckt that selects binary information from one of many input lines and directs it to a single output line.
- The selection of a particular input line is controlled by a set of selection lines. Normally, there are $2^n$ input lines & n selection lines.
- The size of a MUX is specified by the number $2^n$ of its inputs line & the single output line

## Uses:

## Example: 4X1 MUX

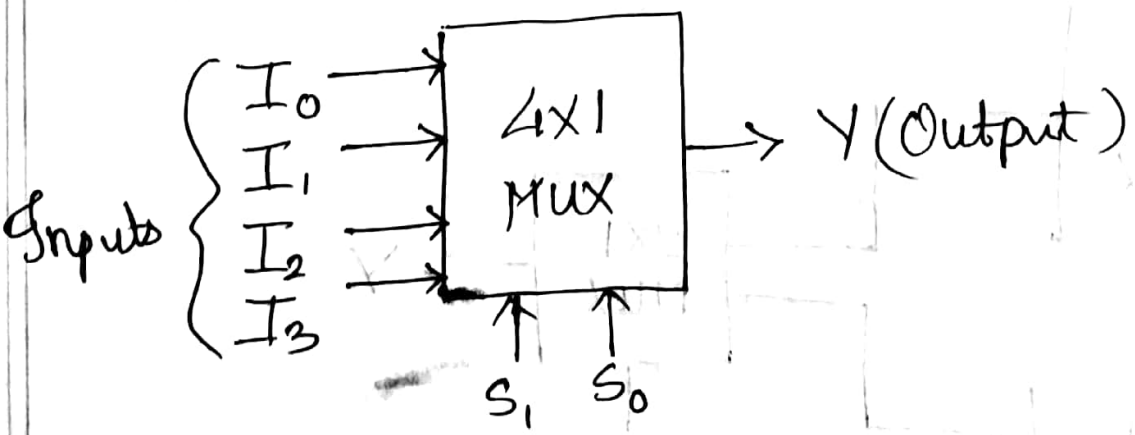



Fig: Block diagram of a 4X1 MUX

## Function table of a 4x1 MUX:

$S_1 S_0 = 00$ | $I_0 = 0, Y = 0$
$I_0 = 1, Y = 1$

| $S_1$ | $S_0$ | Y |
|---|---|---|
| 0 | 0 | $I_0$ |
| 0 | 1 | $I_1$ |
| 1 | 0 | $I_2$ |
| 1 | 1 | $I_3$ |

## Output Equation:

$$Y = S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_3$$

## MUX Expansion:

* Design a 4x1 MUX using 2x1 MUXs only.

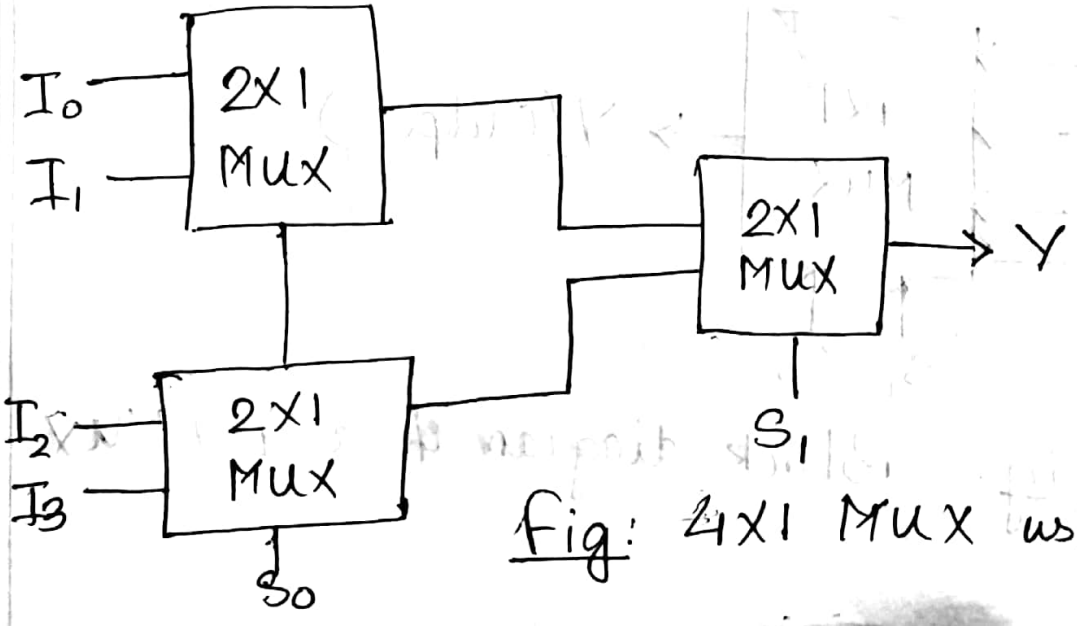



Fig: 4x1 MUX using 2x1 MUXs.

** 06/06/'17
End of N.B.

11/06/'17

- **Decoder :-**

A decoder is a combinational circuit that converts binary information from n input lines to a maximum of $2^n$ unique output lines.

Eg: 2×4, 3×8, 4×16 decoders etc.

- Active low enable ($\overline{E}$) — IC activates with enable pin
  Active high enable (E) — " " " " " "
  ∟ ICs have some extra pins. One of them is enable pin.
- **Example:** 2×4 decoder

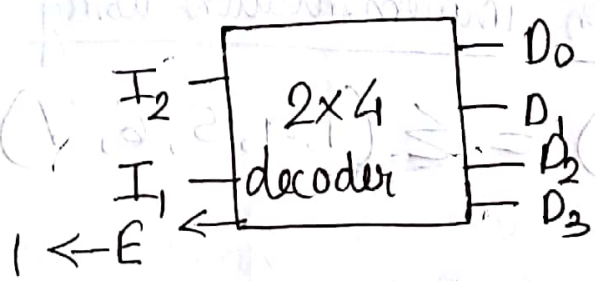



Fig: 2×4 decoder.

Truth Table:

| $I_2$ | $I_1$ | $D_0$ | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

Output equations:

$$D_0 = \overline{I_2}\,\overline{I_1}$$

$$D_1 = \overline{I_2}\,I_1$$

$$D_2 = I_2\,\overline{I_1}$$

$$D_3 = I_2\,I_1$$

Logic Diagram:

See from book.

- Boolean function implementation using decoder:

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

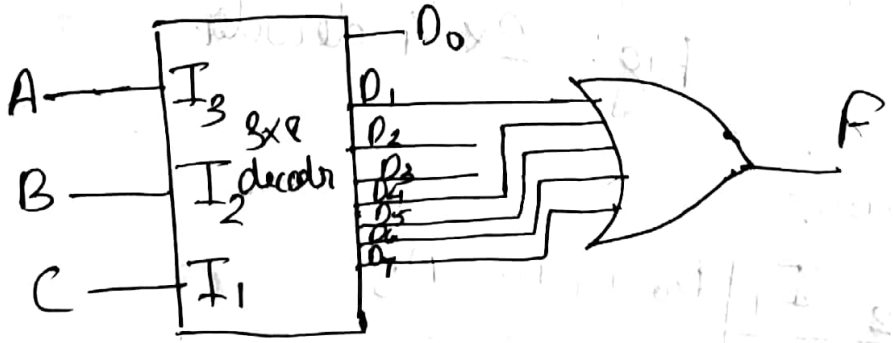
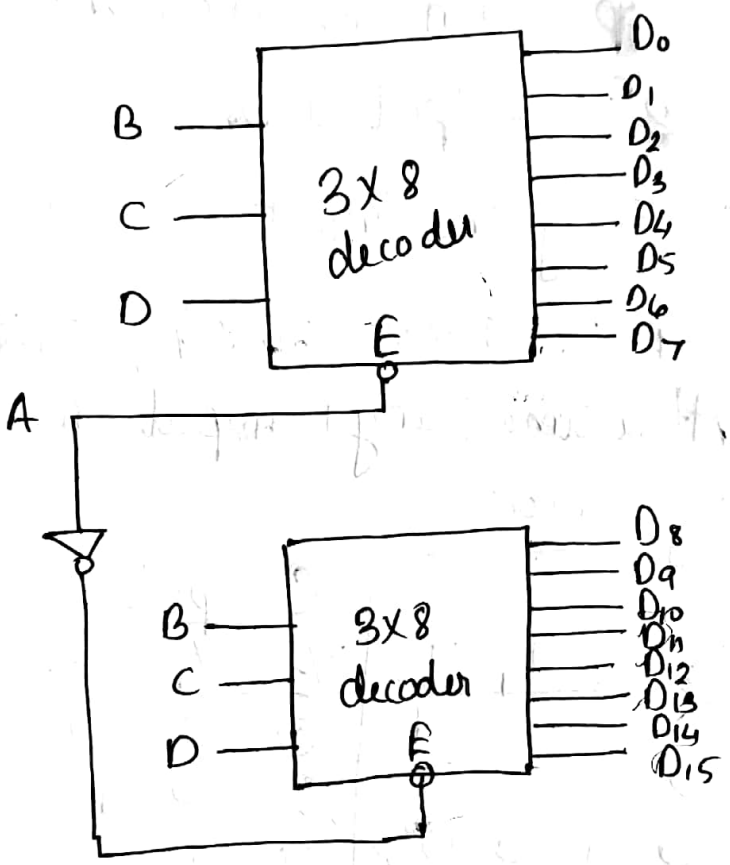



Fig: Implementation of F

• Design a 4 X16 line decoder using two 3x8 decoders:




- Use NAND gate when active low enable used.

## • Encoders:

- Does reverse operation to decoder
- Input lines: $2^n$, output lines: $n$
- Constraint: only one input is active at a time

Example: 4x2, 8x3, 16x4, 512 x 9 encoder etc. (At a time, any 1 input has to be 1, rest 0.)

## • 8x3 binary encoder:

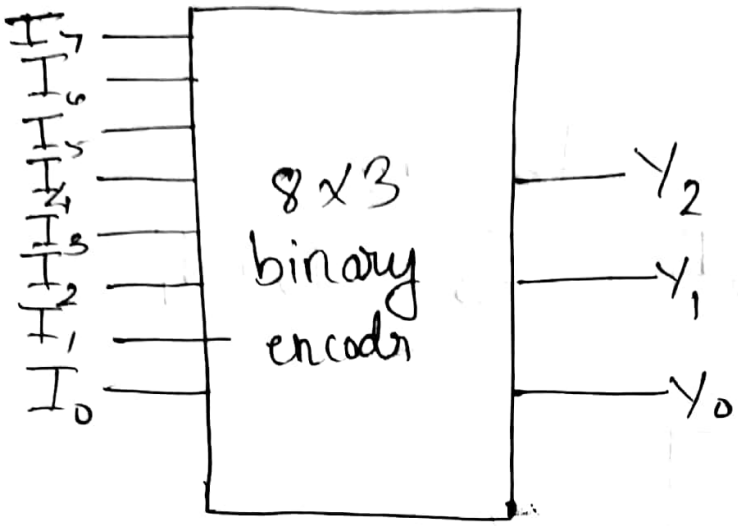
| Input | | | | | | | | Output | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $I_7$ | $I_6$ | $I_5$ | $I_4$ | $I_3$ | $I_2$ | $I_1$ | $I_0$ | $Y_2$ | $Y_1$ | $Y_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

## Equations:

$$Y_2 = I_7 + I_6 + I_5 + I_4$$

$$Y_1 = I_7 + I_6 + I_3 + I_2$$

$$Y_0 = I_7 + I_5 + I_3 + I_1$$




- Circuit diagram

See from book

- Priority encoders
  - Encoder with priority function
  - Multiple inputs may be true simultaneously.
  - Higher priority input gets the precedence
- 4-input Priority Encoder:

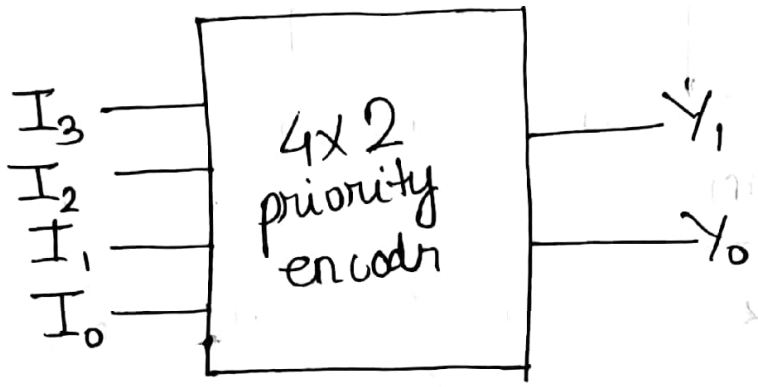
Priority order: $I_3 > I_2 > I_1 > I_0$

| Input | | | | Output | |
|---|---|---|---|---|---|
| $I_3$ | $I_2$ | $I_1$ | $I_0$ | $Y_1$ | $Y_0$ |
| 1 | X | X | X | 1 | 1 |
| 0 | 1 | X | X | 1 | 0 |
| 0 | 0 | 1 | X | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | X |

Equations:

$$Y_1 = I_3 + \overline{I}_3 I_2 = I_3 + I_2$$

$$Y_0 = I_3 + \overline{I}_3 \overline{I}_2 I_1 = I_3 + \overline{I}_2 + I_1$$

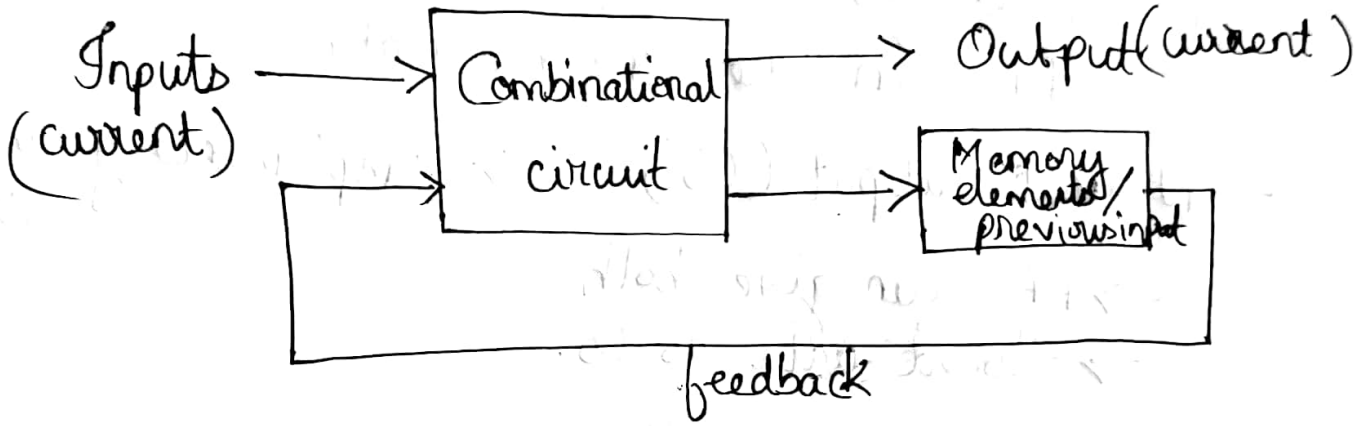



Quiz #3
11/07/17
Tuesday.
Upto here (Chapter-4 remaining)

04/07/'17

# Chapter - 5

## Sequential Circuit




Sequential circuit eg: football scoring.

Output = f(input, previous output)

- A memory element is required to store previous output.

### Classification of sequential circuit:

- Asynchronous sequential circuit: output changes as soon as input changed
- Synchronous sequential ckt: output does not change simultaneously. It is dependant on time. Clock pulse when pressed, output changes, or else output doesn't change.

# Flip-flops:

- Memory elements which store output are called flip-flops (FFs.)
- One FF can store one-bit data.
- Normal output (Q) → Complementary output (Q')
  - FFs can give both
    - → Stored data as Q.
- 4 types of FFs : i) SR
  ii) JK
  iii) T
  iv) D

## Basic flip-flop circuit:

### SR FF:

- using NOR gate
- using NAND gate

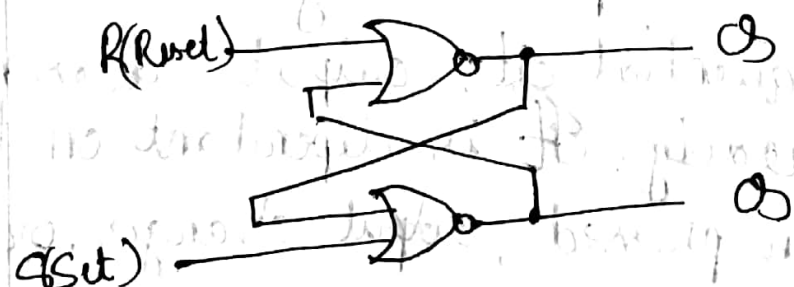



Fig: using NOR gate

| S | R | Q | Q' |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| * 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| * 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |

→ For same inputs, different outputs are obtained, due to different previous outputs.

Initially, Q & Q' 's values aren't known

$S \to 1$, $Q' = 0$

$R \to 0$ $\therefore Q \to 1$

If any one of the input is 1, output is 0.

Now the values of Q & Q' are known.

$Q \to 1$, $Q' \to 0$

Applying, $S \to 0$, $R \to 0$

$\therefore Q' \to 0$

$\therefore Q \to 1$

Applying $S \to 0$, $R \to 1$

$Q' \to 0$

$Q \to 0$

$\therefore Q' \to 1$

** We can never use S and R = 1 at the same time in SR FF, since both Q & Q' = 0 which is not possible logically.

The inputs should begin with at least one of them being 1.

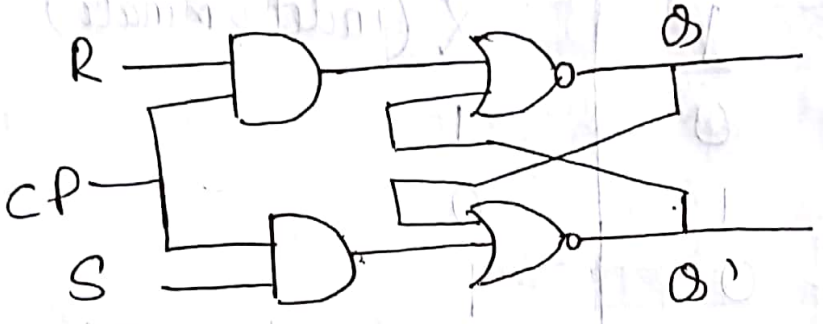
$S \rightarrow 1, Q \rightarrow 1$

$R \rightarrow 1, Q' \rightarrow 1$

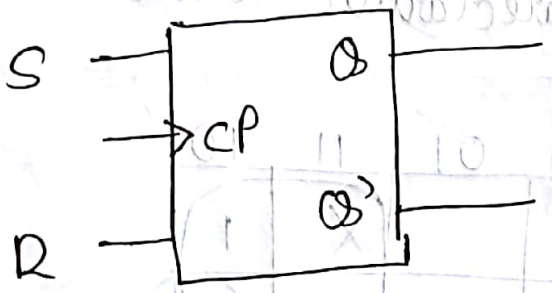
9/07/'17

## • Clocked SR flip-flop (FF):

The clocked SR FF in the following fig. consists of a basic NOR FF and 2 ø AND gates.




a) logic diagram




b) graphic symbol

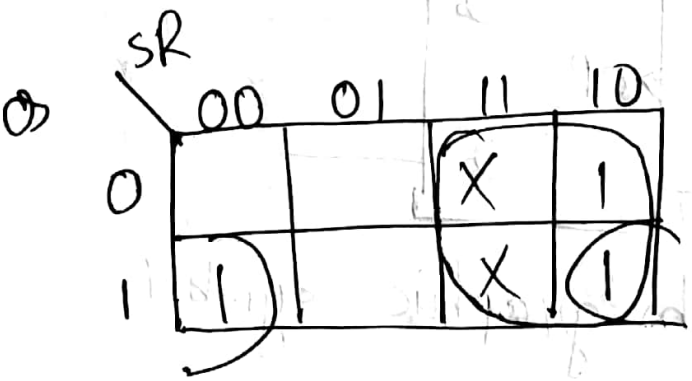
Q(t) - Previous o/p

current i/p

current o/p

| Q | S | R | Q(t+1) |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | X (indeterminate) |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | X (indeterminate) |

c) charectaristic table




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

d) Charectaristic equation

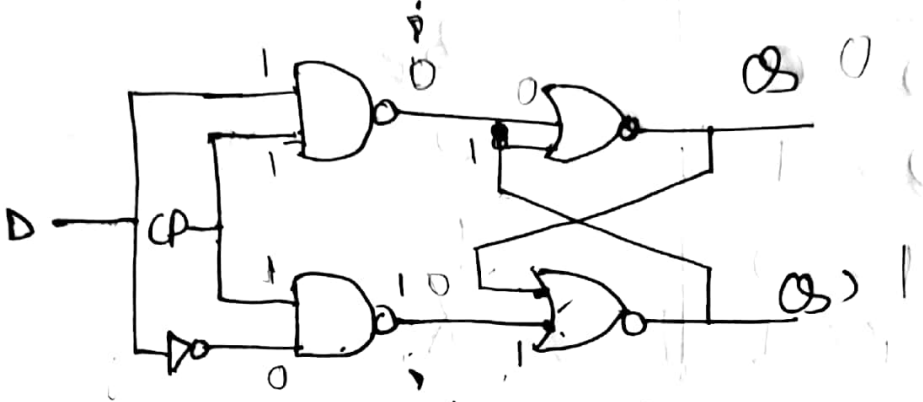
| S | R | Q(t+1) |
|---|---|---|
| 0 | 0 | $Q_t$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | X |

e) Truth table.

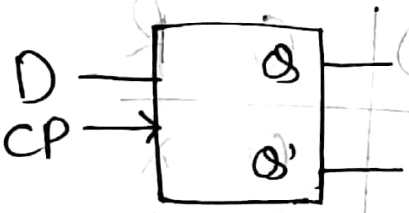
| Q(t) | Q(t+1) | S | R |
|---|---|---|---|
| 0 | 0 | 0 | X |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | X | 0 |

f) excitation table.

# • Clocked D Flip flop:




a) logic diagram




b) graphical symbol

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

c) charectaristic table

d) Charectaristic equation:

$$Q(t+1) = D$$

e) Truth table

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

f) Excitation table:

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

| $Q(t)$ | $Q(t+1)$ | T |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

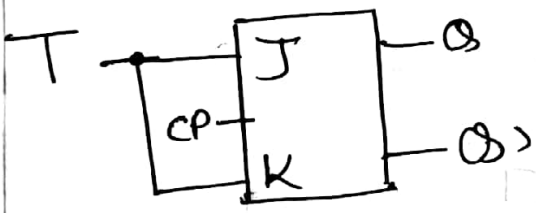
e) Excitation table

| T | $Q_{(t+1)}$ |
|---|---|
| 0 | $Q_t$ |
| 1 | $Q'_t$ |

Truth Table

- Conversions

From JK FF to TFF:



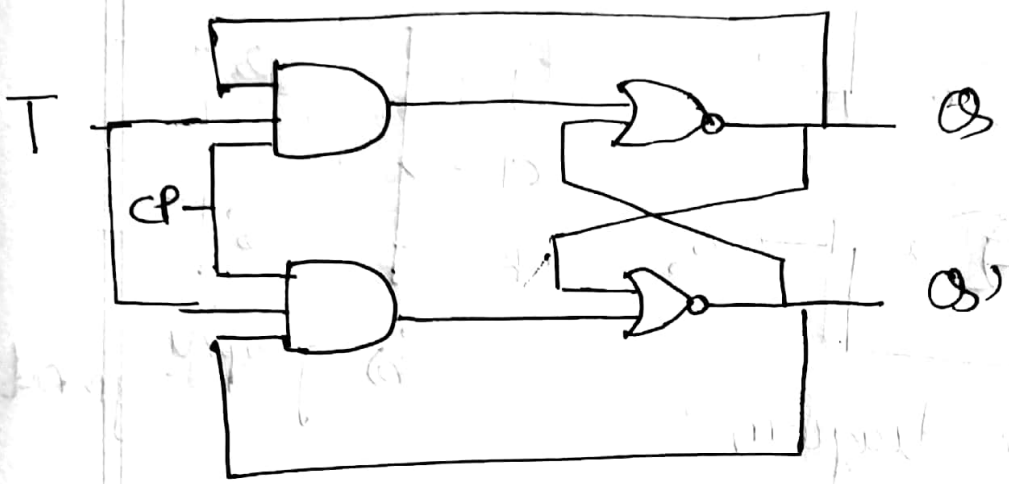

JK:

$$Q(t+1) = JQ' + K'Q$$
$$= JQ' + J'Q \quad (\because J=K)$$
$$= J \oplus Q$$

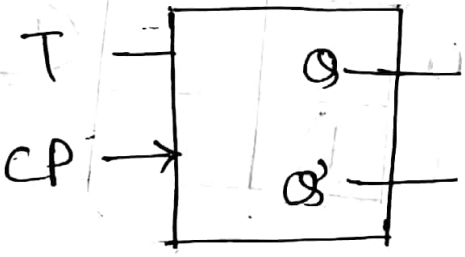
T:

$$Q(t+1) = T \oplus Q$$

# • Clocked T flip flop



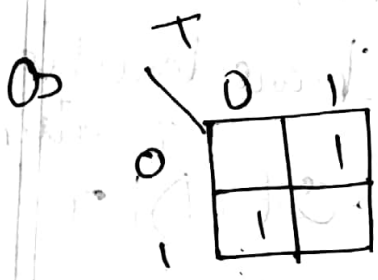

a) Logic diagram




b) Graphic symbol

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

c) Charectaristic Table.



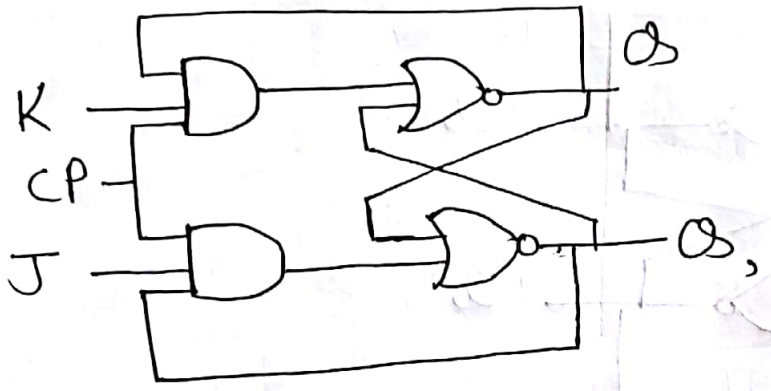

$Q(t+1) = T Q' + T' Q$

$= T \oplus Q$

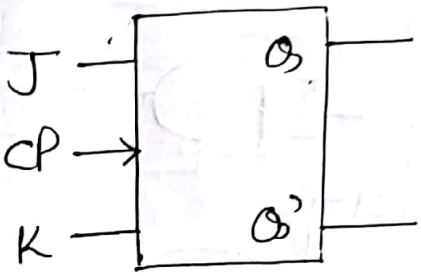
d) Charectaristic equation.

10/07/1

# • Clocked JK Flip flop:




a) logic diagram



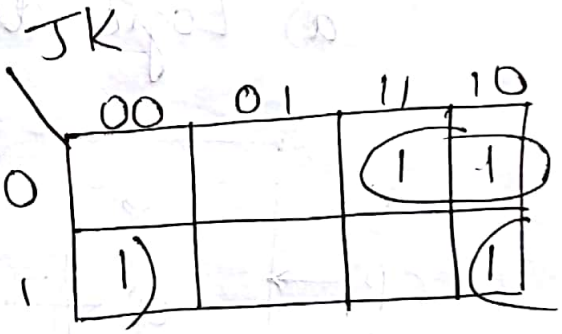

b) graphic symbol

| Q | J | K | Q(t+1) |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

c) Charectaristic table

| J | K | Q(t+1) |
|---|---|---|
| 0 | 0 | Q (unchanged) |
| 0 | 1 | 0 (reset) |
| 1 | 0 | 1 (set) |
| 1 | 1 | Q' (toggle) |

e) Truth table



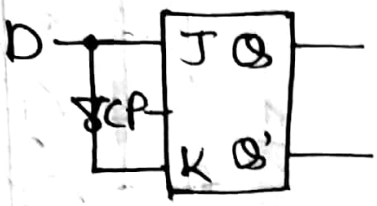

$Q(t+1) = JQ' + K'Q$

d) Charectaristic equation.

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

b) Truth table Excitation

# From JK FF to D FF:




**JK:**

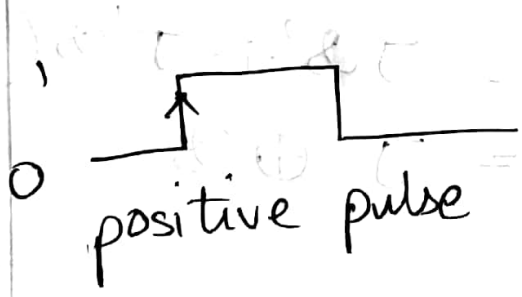
$$Q(t+1) = JQ' + K'Q$$
$$= JQ' + (J')'Q \quad [\because K = J']$$
$$= JQ' + JQ \quad [\because (J')' = J]$$
$$= J(Q + Q') \quad [\because Q + Q' = 1]$$
$$= J$$

**D:**

$$Q(t+1) = D$$

- **Triggering of flip flops:**

i) Positive edge triggering:



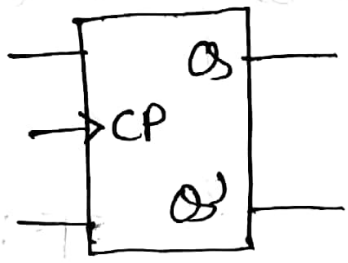




Fig: positive edge triggering

ii) Negative edge triggering:

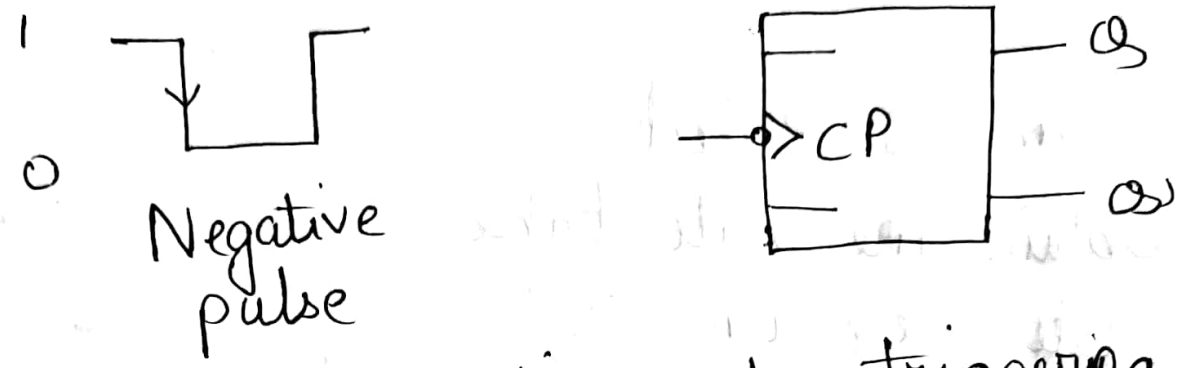



Fig: Negative edge triggering.

- Master slave Flip flop : (Self study)

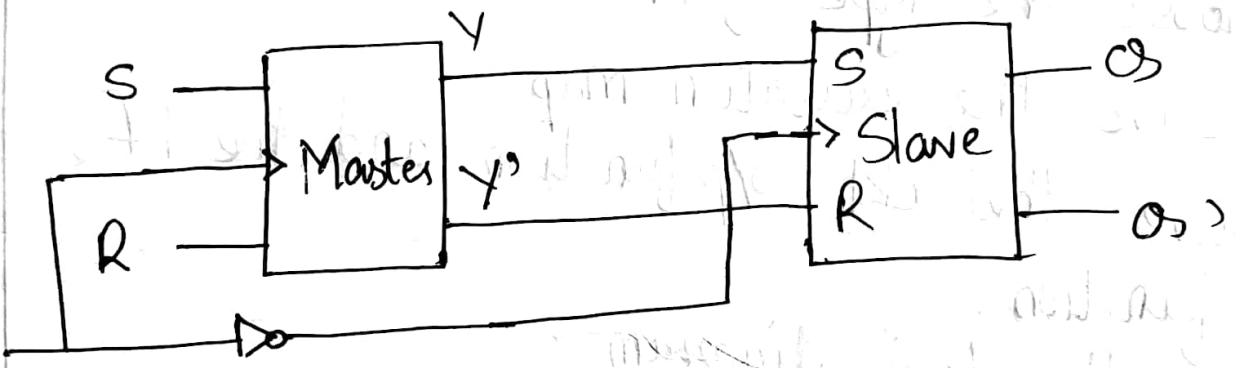


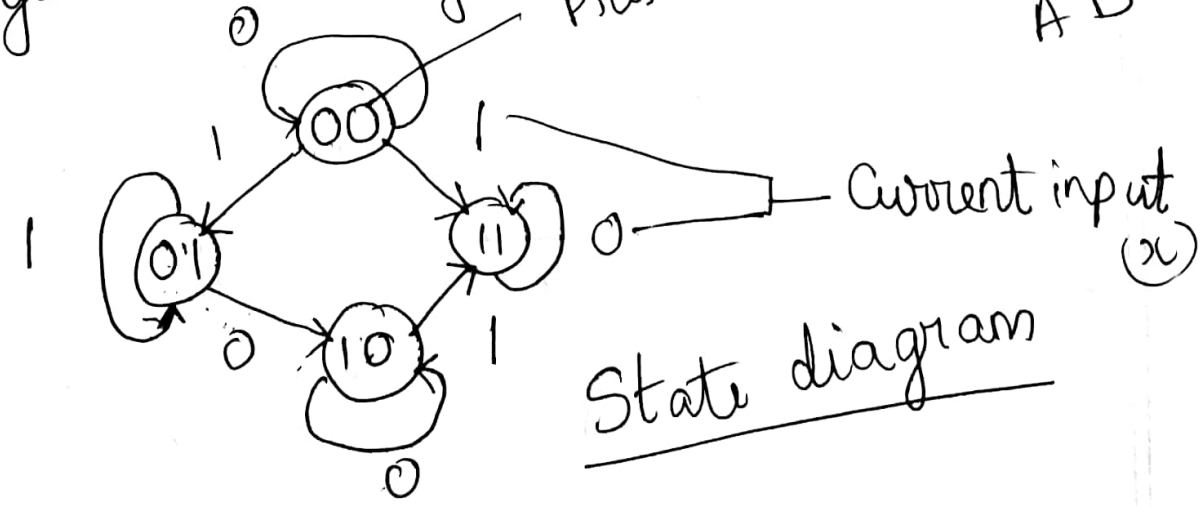

Fig: Master slave SR FF

16/07/'17

- **Design Procedure of Synchronous Sequential Circuit**

i) Problem is stated
ii) Obtain the state table
iii) State reduction (X)
iv) Assign binary values to each state
v) Determine the number of FFs needed and assign a letter symbol to each.
vi) Choose the type of FF to be used.
vii) Derive the excitation map.
viii) Derive the ckt o/p function and the FFs input function
ix) Draw the logic diagram.

Example: Design a synchronous sequential ckt for the given state diagram.




State diagram

Solution:

State table:

| Present state | | Next state | | | |
|---|---|---|---|---|---|
| | | $x = 0$ | | $x = 1$ | |
| A | B | A | B | A | B |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |

One FF can store 1-bit. ∴ For 2-bits, We need 2 FFs. We will use 2 JK FFs. (generally)

Excitation map:

| Present state | | Input | Next state | | FF's inputs | | | |
|---|---|---|---|---|---|---|---|---|
| A | B | x | A | B | $J_A$ | $K_A$ | $J_B$ | $K_B$ |
| 0 | 0 | 0 | 0 | 0 | 0 | x | 0 | x |
| 0 | 0 | 1 | 0 | 1 | 0 | x | 1 | x |
| 0 | 1 | 0 | 1 | 0 | 1 | x | x | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | x | x | 0 |
| 1 | 0 | 0 | 1 | 0 | x | 0 | 0 | x |
| 1 | 0 | 1 | 1 | 1 | x | 0 | 1 | x |
| 1 | 1 | 0 | 1 | 1 | x | 0 | x | 0 |
| 1 | 1 | 1 | 0 | 0 | x | 1 | x | 1 |

- FF's input equations: based on present state and input

| A \ Bx | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | | | | 1 |
| 1 | X | X | X | X |

$J_A = B\bar{x}$

| A \ Bx | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | X | X | X | X |
| 1 | | | 1 | |

$K_A = Bx$

| A \ Bx | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | | 1 | X | X |
| 1 | | 1 | X | X |

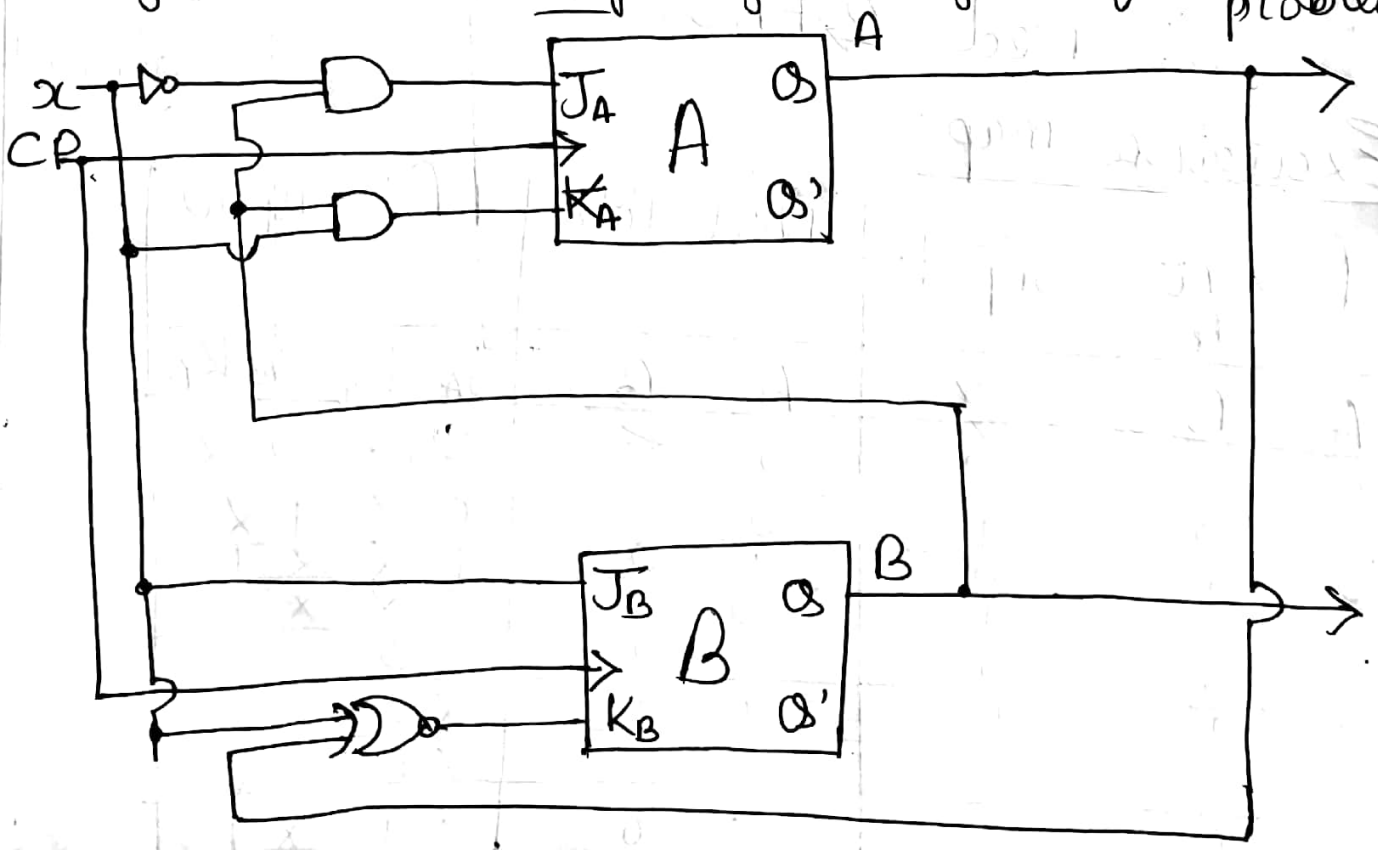
$J_B = x$

| A \ Bx | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | X | X | | 1 |
| 1 | X | X | 1 | |

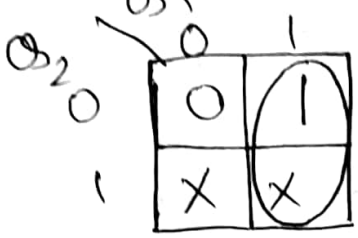
$K_B = \overline{A \oplus x}$

- Logic Diagram:

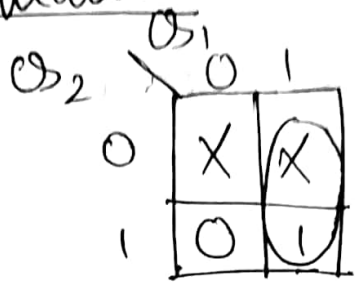
Fig: logic diagram for the given problem



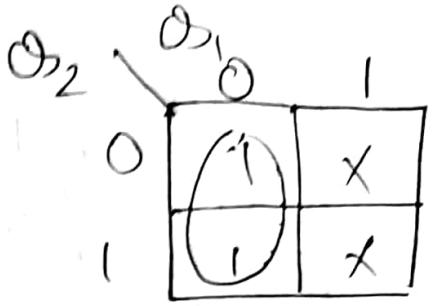
## FF's Input Equations: — based on present state



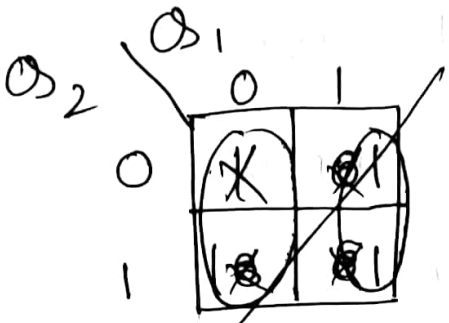

$J_2 = Q_1$



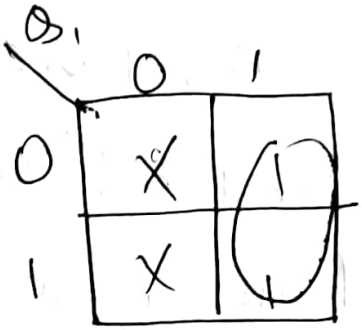

$K_2 = Q_1$




$J K_1 = 1$




$K_1 = 1$




$K_1 = 1$

## Logic Diagram:

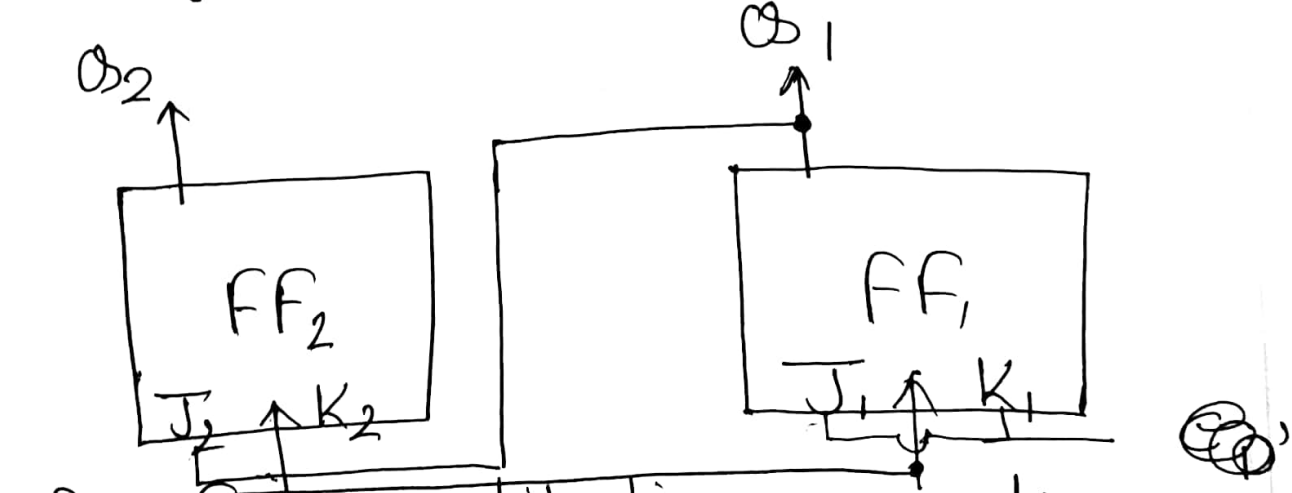



Fig: 2-bit ~~binary~~ synchronous binary up counter.

# Chapter-6

17/07/'17

## Design of Counters:

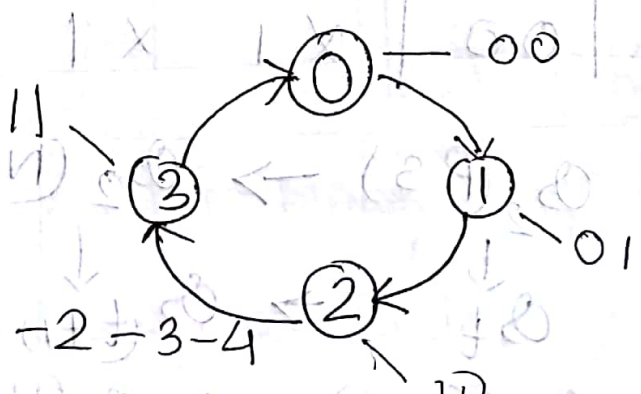
- **Counter:** A sequential ckt that goes through a prescribed sequence of states upon the application of input pulses is called a counter. A counter that follows the binary sequence is called a binary counter. An n bit binary counter consists of n FFs and can count in binary from 0 to $2^n - 1$.

- **Example:** Design a 2-bit synchronous binary up counter.

**Solution:**

**Counting sequence:** $0 \rightarrow 1 \rightarrow 2 \rightarrow 3$

**State diagram:**




* up → 1-2-3-4
down → 8-9-7-6

## State table:

| Present state (PS) | | Next State NS | |
|---|---|---|---|
| $Q_2$ | $Q_1$ | $Q_2$ | $Q_1$ |
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |

Say, we will use 2 JK Flip flops

## Excitation Map:

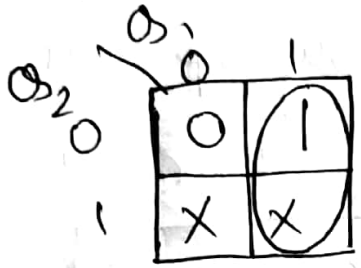
| PS | NS | FF's Input | | | |
|---|---|---|---|---|---|
| $Q_2$ $Q_1$ | $Q_2$ $Q_1$ | $J_2$ | $K_2$ | $J_1$ | $K_1$ |
| 0 0 | 0 1 | 0 | X | 1 | X |
| 0 1 | 1 0 | 1 | X | X | 1 |
| 1 0 | 1 1 | X | 0 | 1 | X |
| 1 1 | 0 0 | X | 1 | X | 1 |

$Q_2(PS) \rightarrow Q_2(NS) \Rightarrow J_2 K_2$

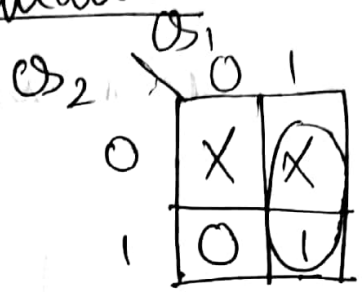
$Q_t \rightarrow Q_{(t+1)}$

$Q_1(PS) \rightarrow Q_1(NS) \Rightarrow J_1 K_1$

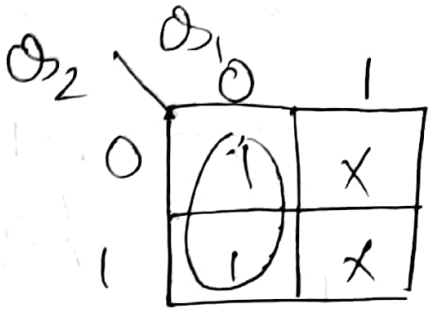
## FF's Input Equations: — based on present state



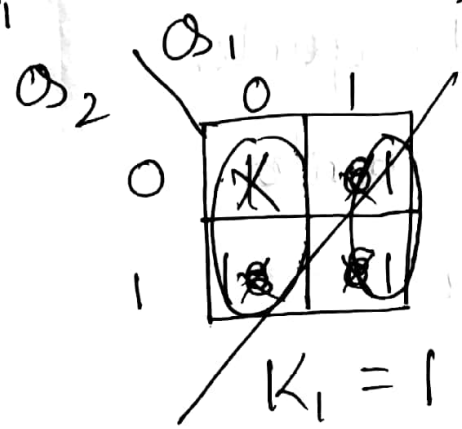

$J_2 = Q_1$



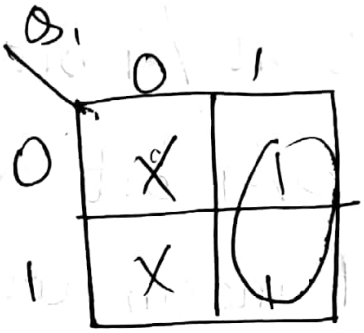

$K_2 = Q_1$




$J K_1 = 1$




$K_1 = 1$




$K_1 = 1$

## Logic Diagram:

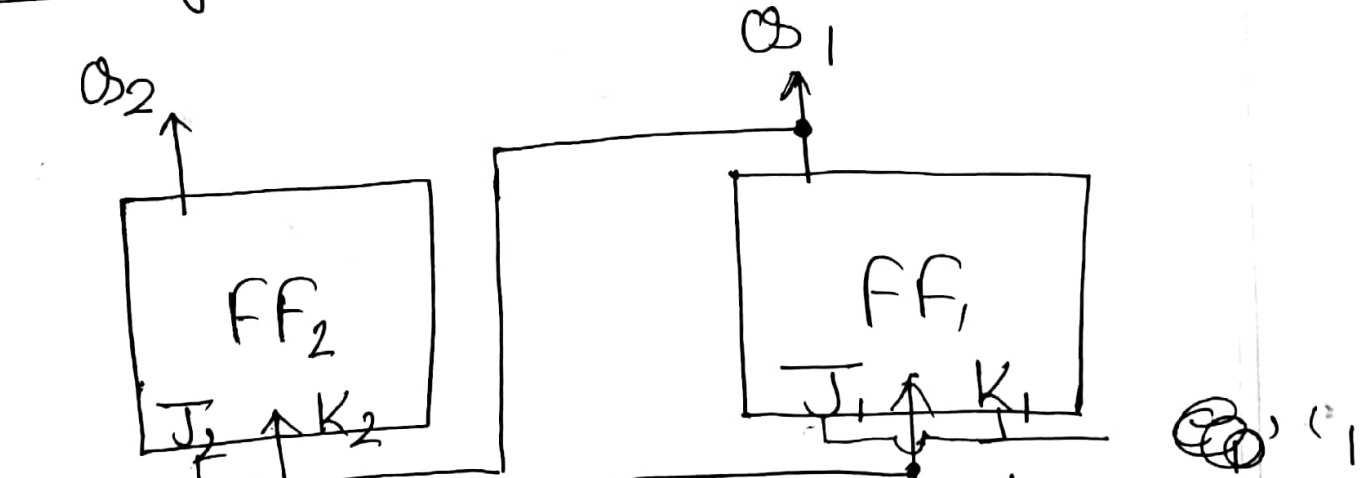



Fig: 2-bit ~~binary~~ synchronous binary up counter.

## Exercises:

1. 2/3/4 bit synchronous binary up counter
2. 2/3/4 bit " " down "
3. 3 bit prime counter [2-3-5-7]
4. 3 bit / 4 bit odd counter
5. 3/4 bit even counter
6. Random counter

18/07/17

- **Binary Up-down Counter:**

Design and implement 2-bit binary up-down counter.

**Solution:**

**Counting sequence:**

0 → 1 → 2 → 3 → 2 → 1

**Upward count sequence:**

0 → 1 → 2 → 3

**Downward count sequence:**

3 → 2 → 1 → 0

**State table for upward count sequence:**

**Excitation map for upward count sequence:**

**FF's input equations for upward counts sequence**

$J_2 = Q_1$, $K_2 = Q_1$, $J_1 = 1$, $K_1 = 1$

State table & excitation map for downward count sequence:

FF's input equations for downward counts:

$$J_2 = Q_1' ; K_2 = Q_1', \; J_1 = 1, \; K_1 = 1$$

Say, we have a control variable C and if C=0, our ckt will count upward and if C=1, our ckt will count downward. Then the final equations of our up-down counter willbe:

$$J_2 = \bar{C}Q_1 + C\bar{Q}_1 = C \oplus Q_1$$

$$K_2 = \bar{C}Q_1 + C\bar{Q}_1 = C \oplus Q_1$$

$$J_1 = \bar{C}.1 + C.1 = \bar{C} + C = 1$$

$$K_1 = \bar{C}.1 + C = \bar{C} + C = 1$$

Logic Diagram:

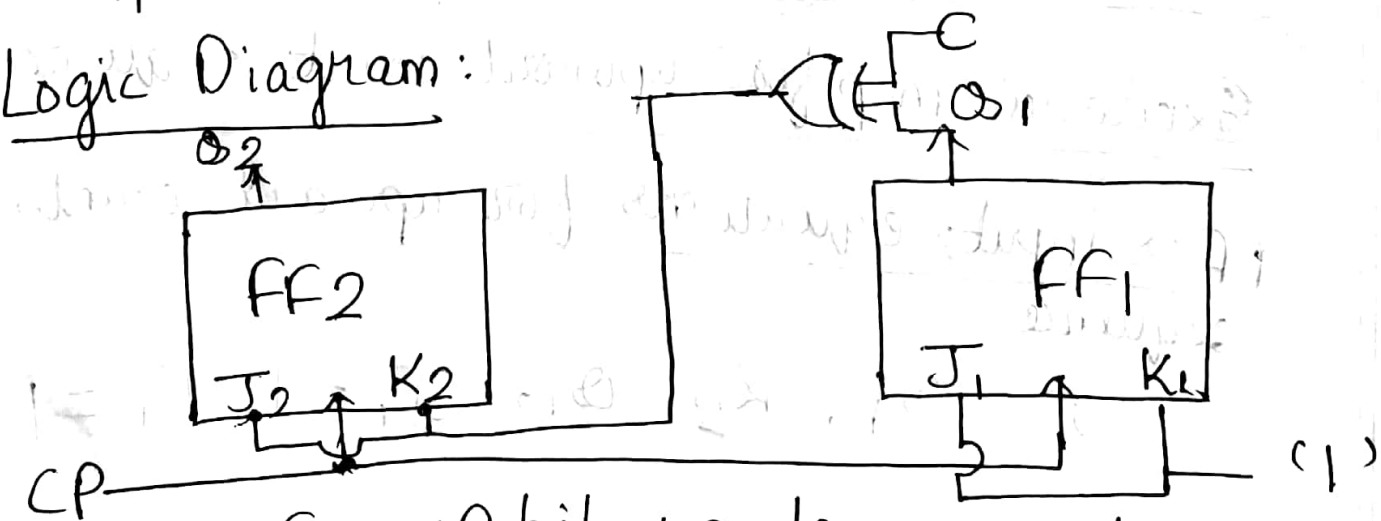
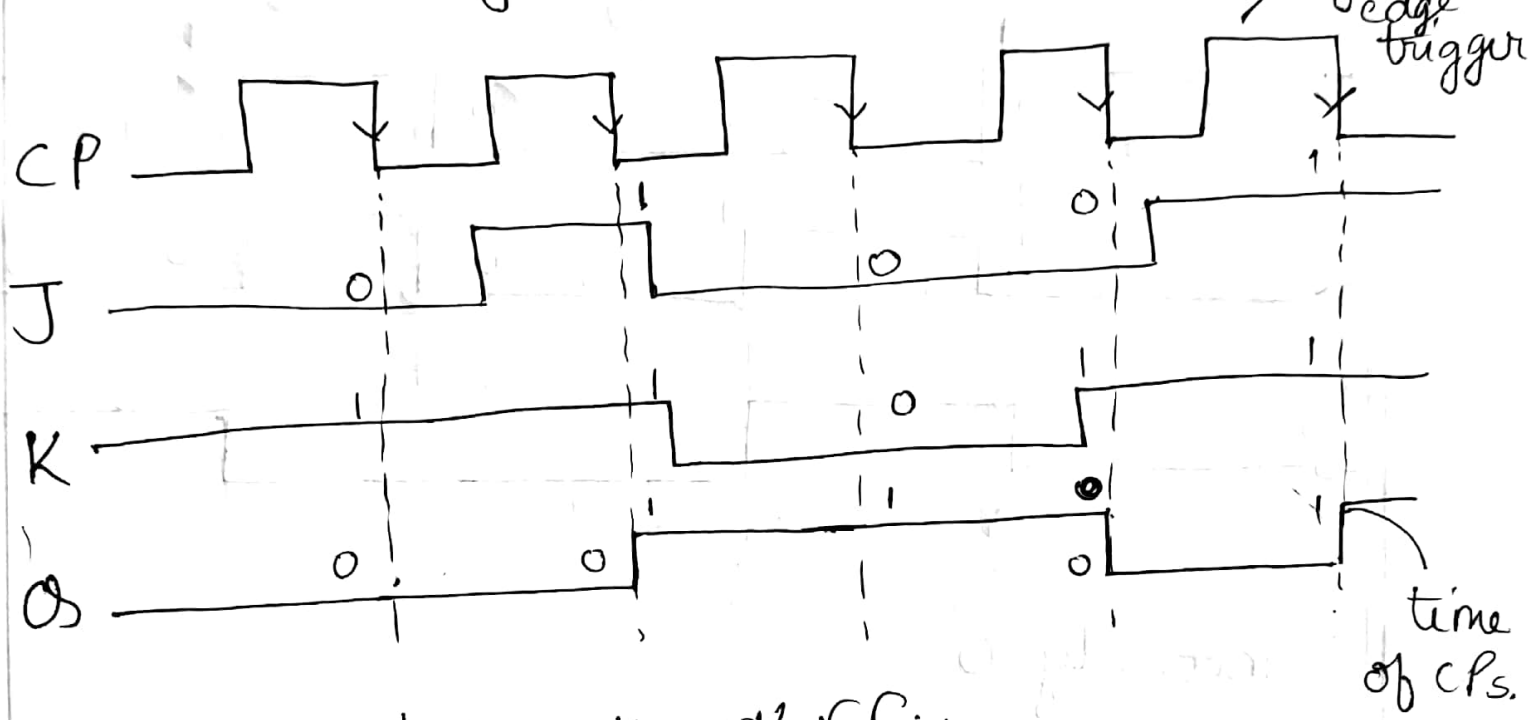



Fig: 2 bit up down counter

- **Timing diagram:**

Find the timing diagram for Qs. (5 clock pulses)




Truth table of JK ff:

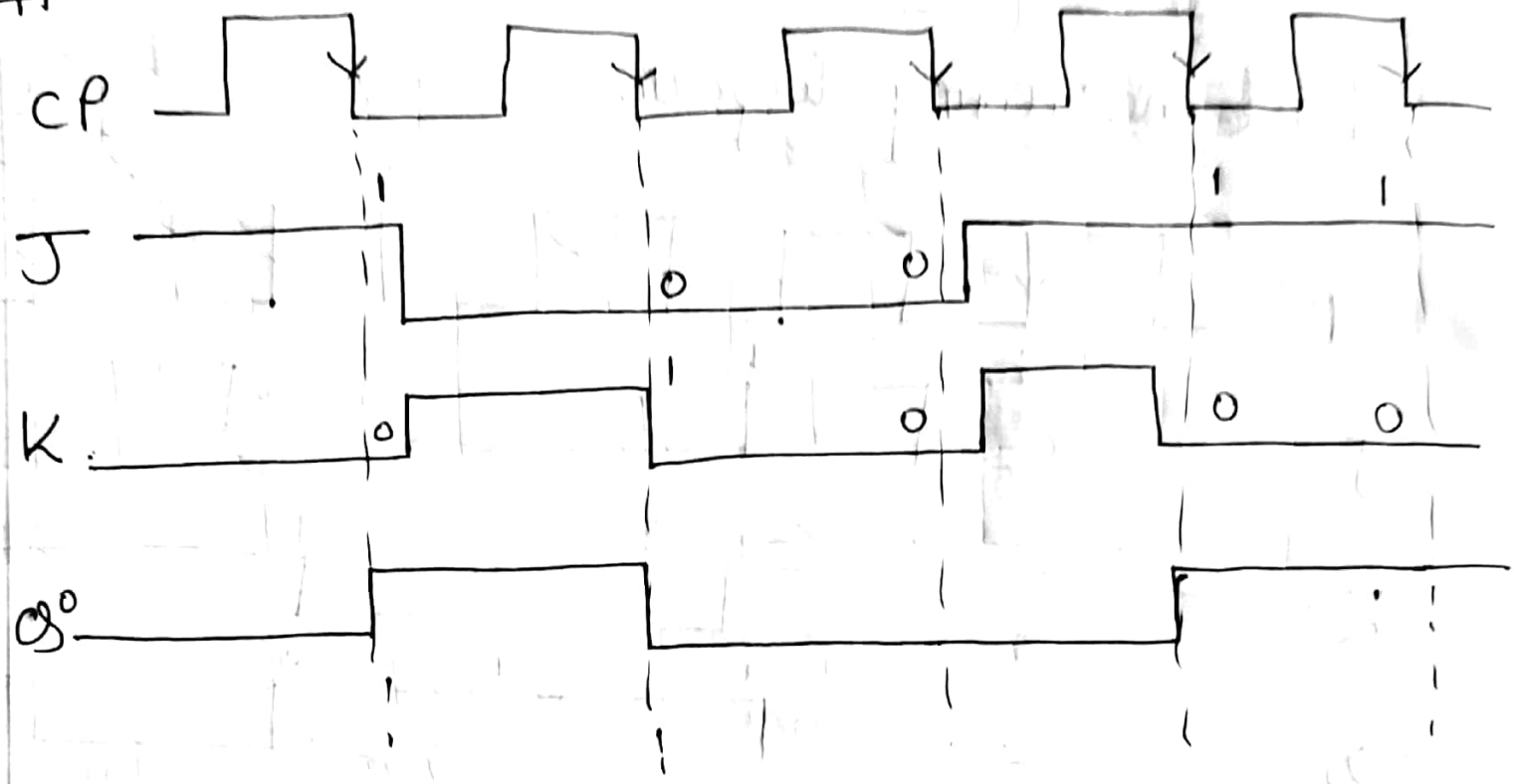
| J | K | Qt+1 |
|---|---|---|
| 0 | 0 | Qt |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | Q't |

Quiz #4 upto here
25/07/'17

CP: 1 → 0 ⇒ output changes

Let Qs's initial value = 0.

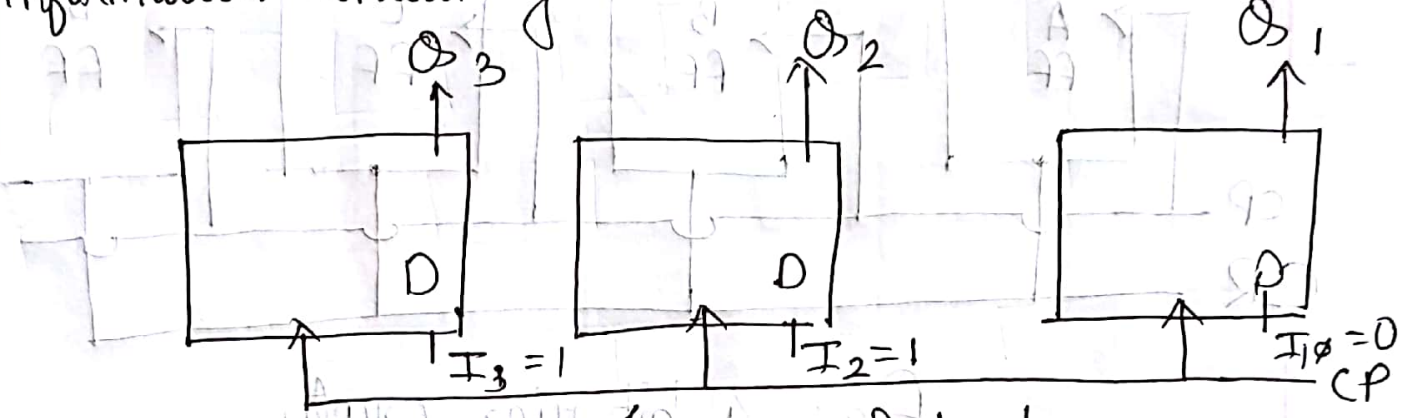
#




Q initially 0

23/07/'17

**Register** – A n-bit register has a group of n FFs and is capable of storing any binary information containing n-bits.




D FF, Input = Output

Fig: 3 bit register.

**Register with parallel load**: Self study

**Shift register**:

Blankspace 0/1 → 0 1 1 → 0 0 1 → 0 0 0

A register capable of shifting its binary information either to the right or to the left is called a shift register.

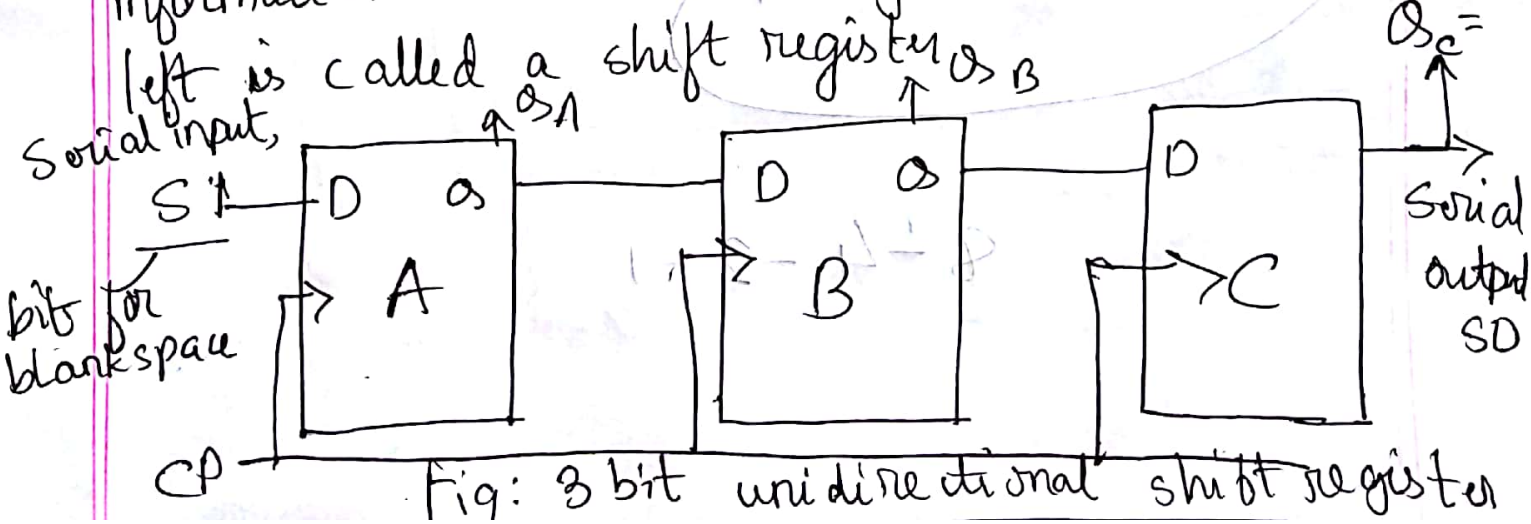



Fig: 3 bit unidirectional shift register

## Ring Counter :

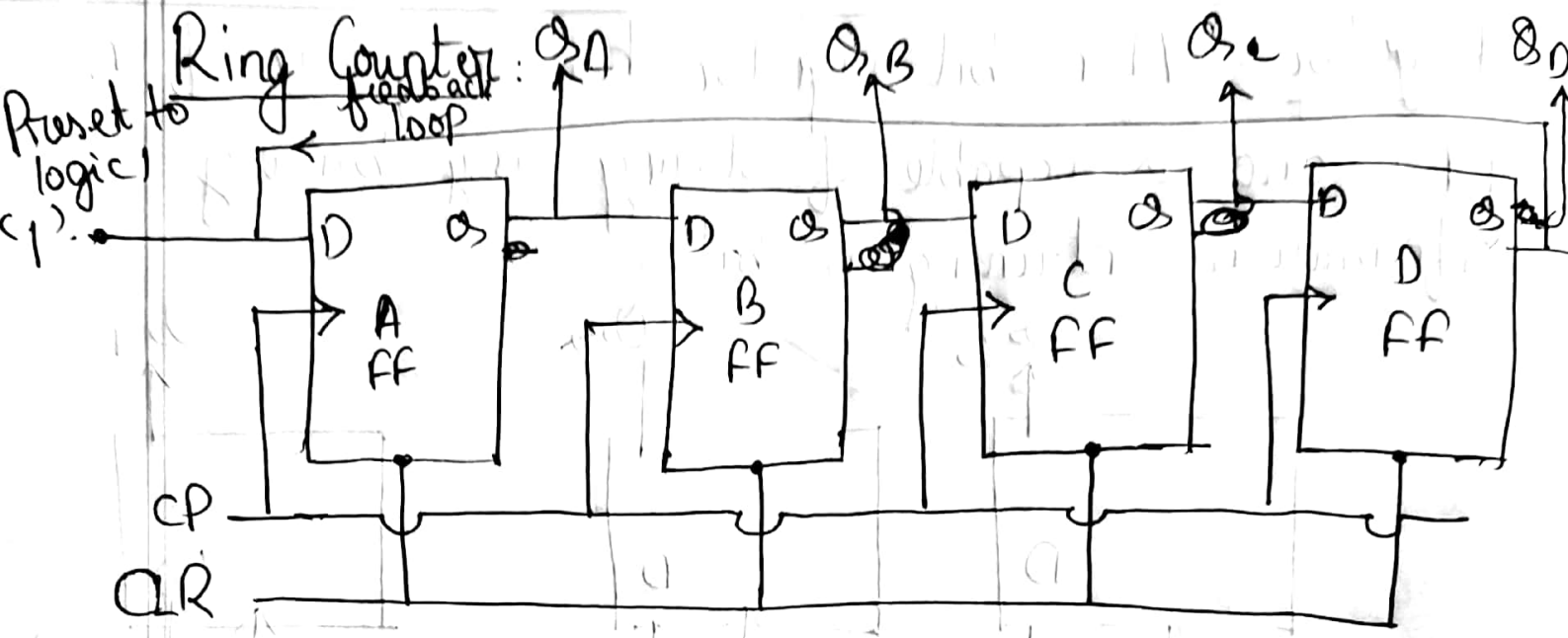



Fig: 4 bit ring counter

Counting sequence.

| $Q_A$ | $Q_B$ | $Q_C$ | $Q_D$ | |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | Initial |
| 0 | 1 | 0 | 0 | |
| 0 | 0 | 1 | 1 | |
| 0 | 0 | 0 | 1 | |

8-4-2-1

## Johnson Counter:

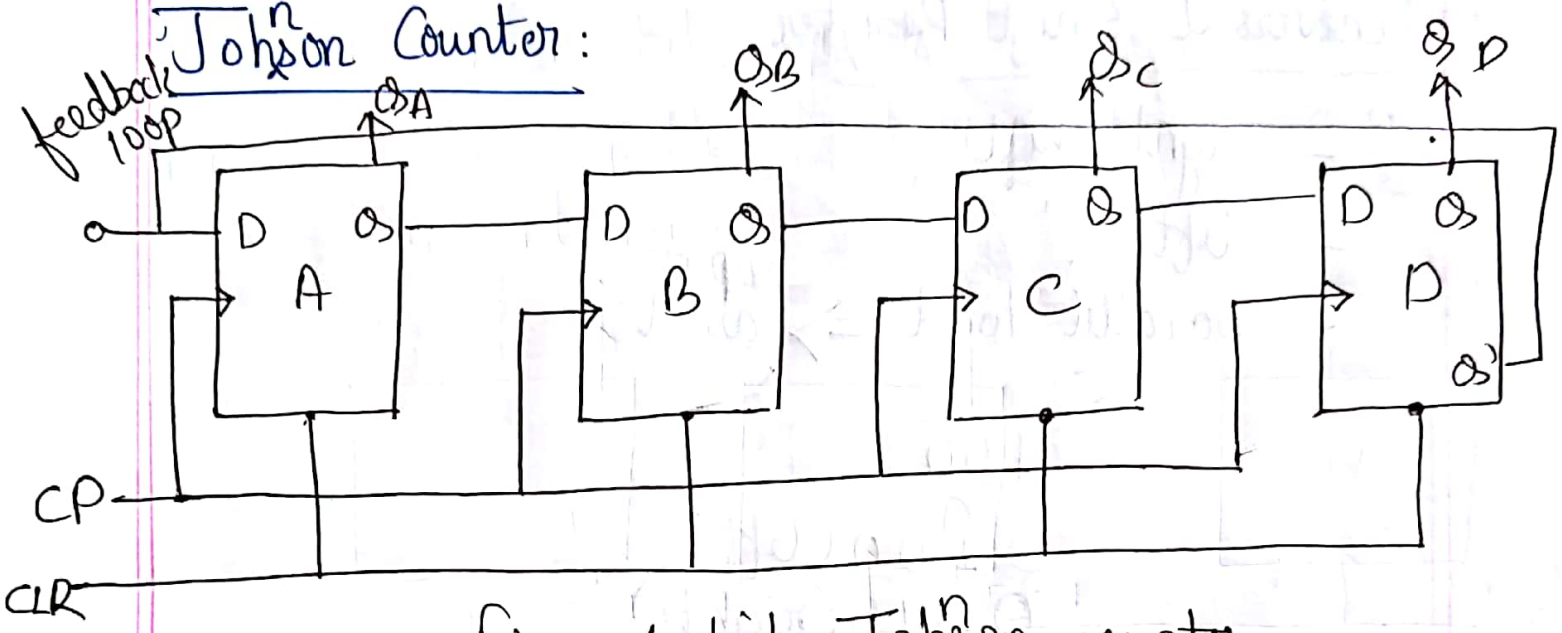



Fig: 4 bit Johnson counter

### Counting sequence:

| $Q_A$ | $Q_B$ | $Q_C$ | $Q_D$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 |
| 0 | | | |

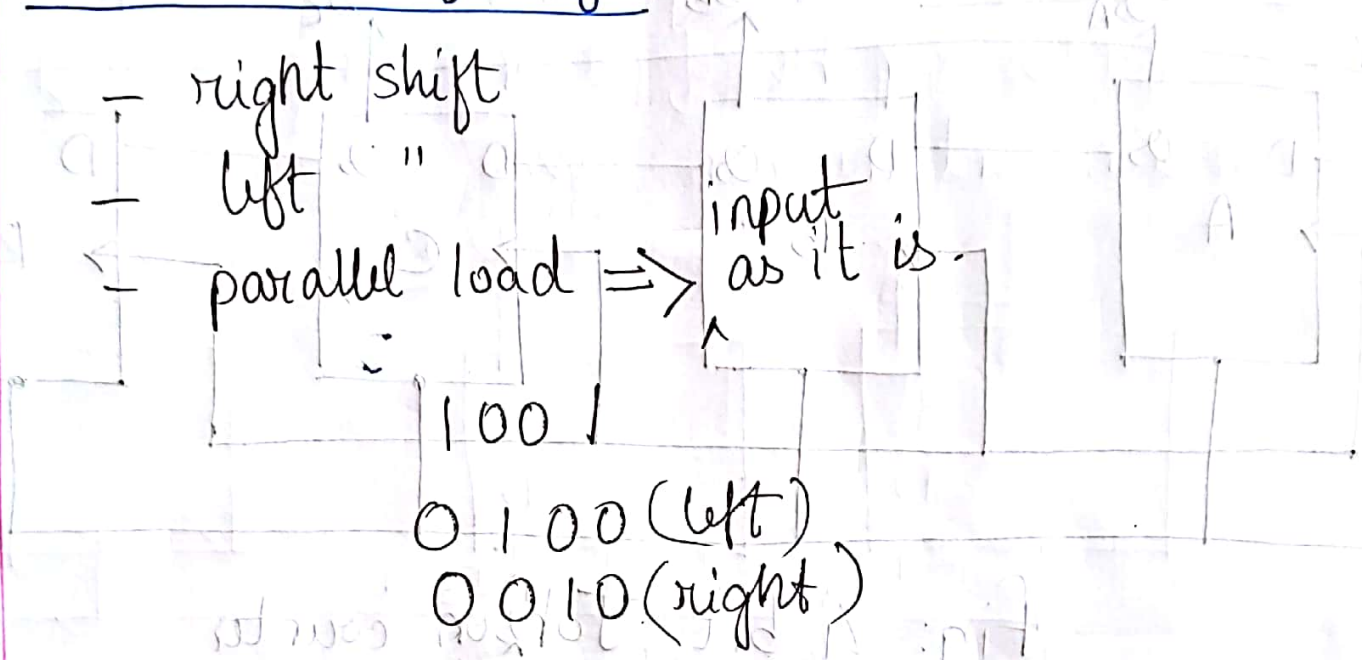
Difference of J counter from R counter :-
feedback is Q', not Q.

0 - 8 - 12 - 14 - 16 - 7 - 3 - 1

30/07/17

# Universal Shift Register

- right shift
- left "
- parallel load ⇒ input as it is.




## 4x1 MUX –

| $S_1$ | $S_0$ | |
|---|---|---|
| 0 | 0 | No change |
| 0 | 1 | Shift right |
| 1 | 0 | Shift left |
| 1 | 1 | Parallel load |

## 4-bit Universal Shift Register:

(L–R) all selectors short

00 — 00 — $I_0$ — whatever is given in input

⇒ output ($A_3$)

00 => $I_0$ — ($A_2$)

output — $A_2$

00 => $I_0$ — $A_1$

output — $A_1$

00 => $I_0$ — $A_0$

output — $A_0$

(L-R) :: All values same as before ($A_3$ $A_2$ $A_1$ $A_0$)

01 — $A_3$ $A_2$ $A_1$

01 => $I_1$ => serial input (bit for shifting) 0/1 Eg: 0

Output => 0 (instead of $A_3$)

01 => $I_1$ => $A_3$

Output => $A_3$ (instead of $A_2$)

01 => $I_1$ => $A_2$

O/P => $A_2$ (instead of $A_1$)

01 => $I_1$ => $A_1$

O/P => $A_1$ (instead of $A_0$)

$\therefore$ O/P $\Rightarrow$ O $A_3 A_2 A_1$

Q 10

$A_2 A_1 A_0$ —

to (R → L)

$I_2 \Rightarrow$ 0 (serial input)

O/P $\Rightarrow$ 0

$I_2 \Rightarrow A_0$

O/P $\Rightarrow A_0$

$I_2 \Rightarrow A_1$

O/P $\Rightarrow A_1$

$I_2 \Rightarrow A_2$

O/P $\Rightarrow A_2$

$\therefore$ $A_2 A_1 A_0$ 0

11

all $I_3$ => switch variable.

$I_3$ => whatever given in switch var.

$\therefore I_3 I_2 I_1 I_0 = A_3 A_2 A_1 A_0$

## Asynchronous / Ripple Counter:

Synchronous => common pulse

Asynchronous => one pulse, rest CP → output of prev.

Negative edge trigger — 1 → 0

Positive " " — 0 → 1

31/07/17

- MOD Counter:-

-It is a counter that skips a certain number of steps that we set. For this, we have to force counter to recycle before going through all of the states in the binary sequence. MOD-N counter can count 0 to N-1.

Example: MOD-6 (6 counted)

| J K | $Q_{t+1}$ |
|---|---|
| 1 1 | $\overline{Q}_t$ |

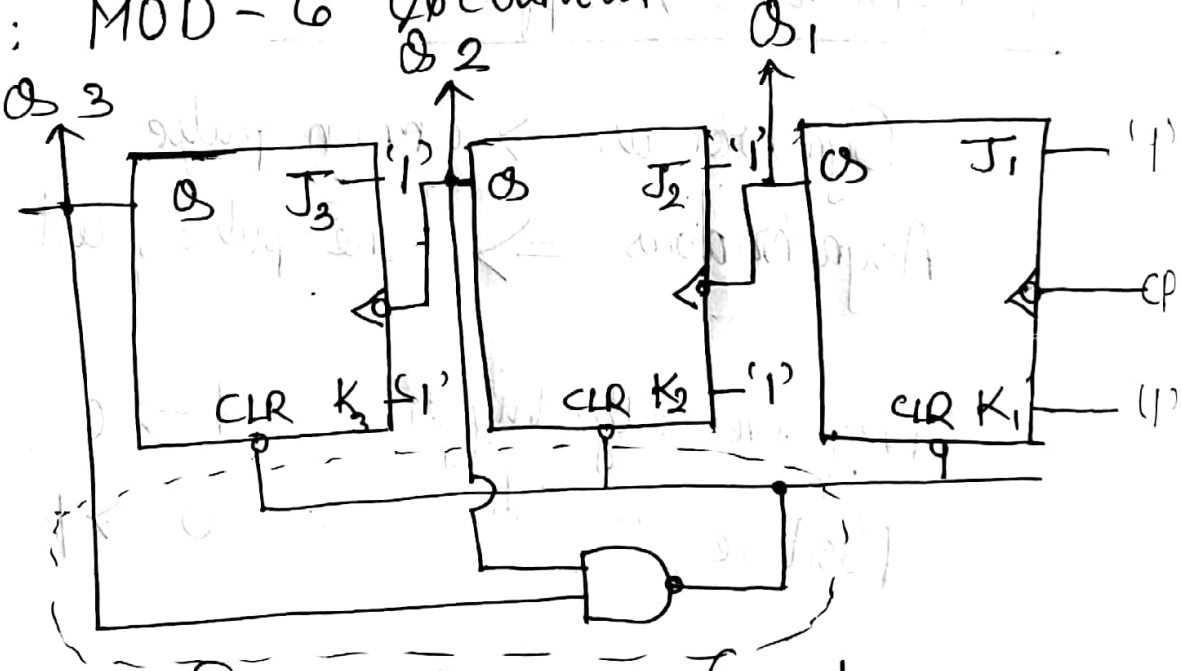



Fig: MOD-6 Counter

- BCD Ripple Counter:

↳ MOD-10 Counter

: can count 0 to 9

0 – 1 : No CP (since -ve
1 – 0 : CP edge)

* $Q_3$ $Q_2$ $Q_1$

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 0 | 1 |
| X 1 | 1 | 0 |
| X 1 | 1 | 1 |

CP → value changes

NO CP → value same as prev.

counts 0 – 7

without dotted part

- To make 0 – 5, dotted part is needed to recycle.
- ∴ After 5, FF has to be cleared. (i.e. when 6)
- 110 : $Q_3 = Q_2 = 1$ (first only when 6.) No combinations before this has both $Q_3$ and $Q_2$ as 1.

∴ NAND gate : both input 1, output 0.

⇓

CLR : clears the FF.

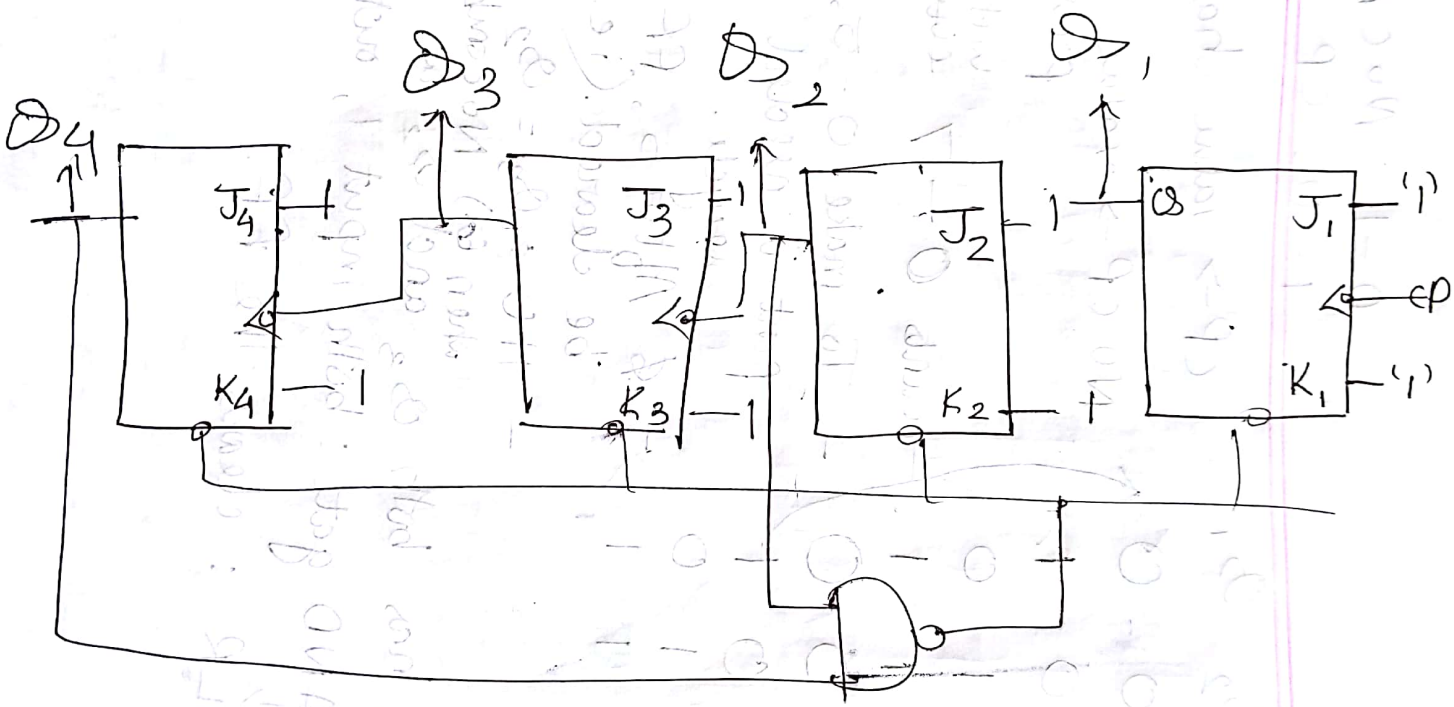
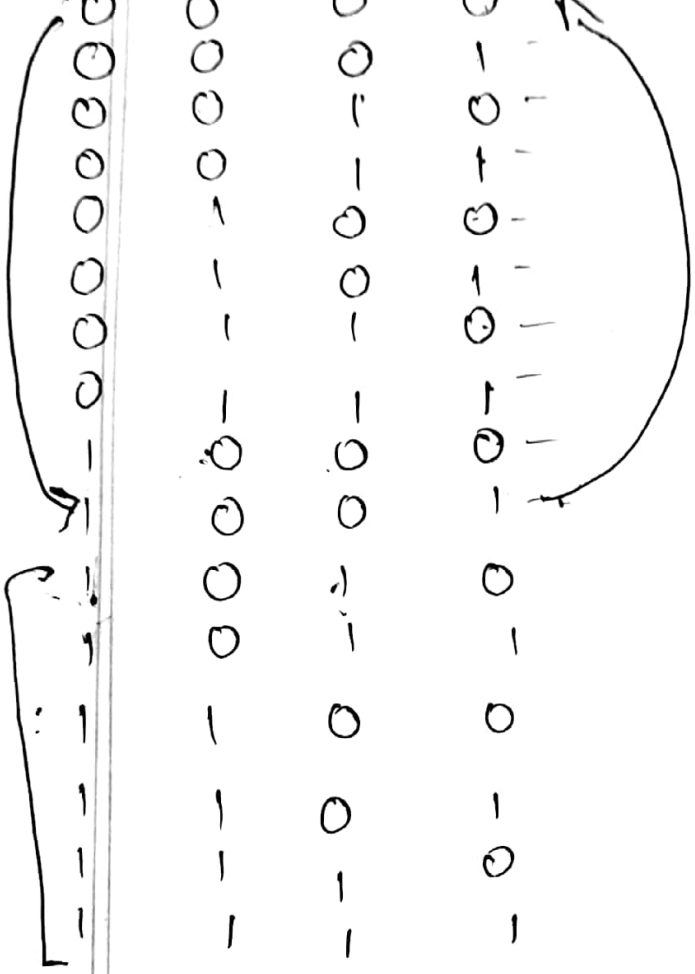



Fig: MOD-10 counter

| $Q_4$ | $Q_3$ | $Q_2$ | $Q_1$ |
|---|---|---|---|
| 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 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 |



BCD counter IC: decade counter.

1 can count 0 – 9

i.e. has 4 FF.

Practice:

MOD 100

MOD 75

b) using 4x1 Mux :-

Implementation table :-

| | $I_0$ | $I_1$ | $I_2$ | $I_3$ |
|---|---|---|---|---|
| A' | 0 | ① | 2 | ③ |
| A | 4 | ⑤ | ⑥ | 7 |
| | 0 | 1 | A | A' |

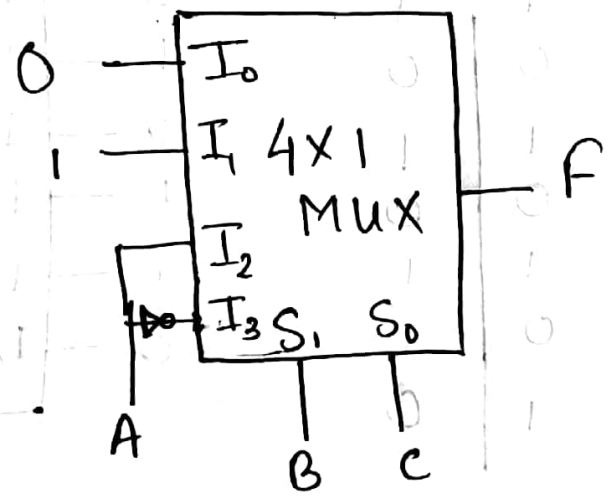



Fig :- using 4x1 MUX

* When a small sized MUX is used, then all inputs can't be connected in selectors. Then,

A B C

↓ └ selector

used in input line

## Boolean Function implementating using MUX:-

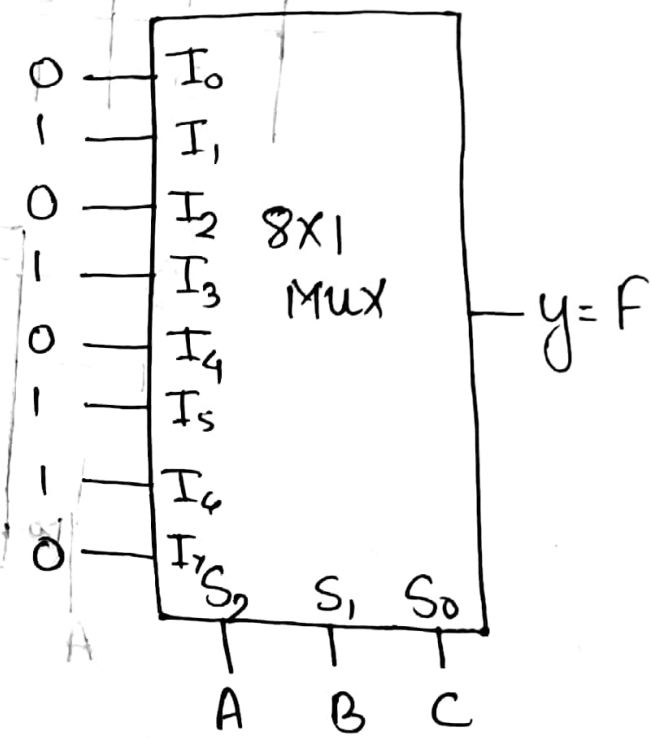
Implement $F(A,B,C) = \Sigma(1,3,5,6)$ using

a) 8×1 MUX

b) 4×1 MUX

Solution:-

| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |




* If no. of input variable = n+1, use $2^n \times 1$ MUX.

a) using 8×1

Eg: $n+1 = 3$

$n = 2$

$\therefore 2^n \times 1 = 4 \times 1$ MUX used

## Demultiplexer :-

Input line : 1

Output line : $2^n$

Selectors : n

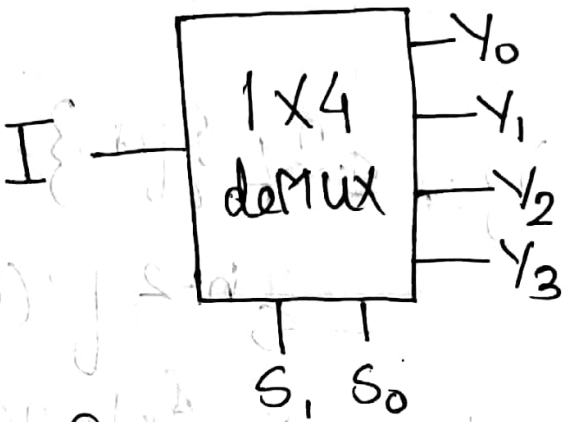



Fig : 1X4 deMux

## Function table:

| $S_1$ | $S_0$ | $Y_0$ | $Y_1$ | $Y_2$ | $Y_3$ |
|---|---|---|---|---|---|
| 0 | 0 | I | – | – | – |
| 0 | 1 | – | I | – | – |
| 1 | 0 | – | – | I | – |
| 1 | 1 | – | – | – | I |