2106

(Syeda Shabnam Mam)

(7A01/N)

Mark Distribution

Attendance ——— 10 %.

Mid (implementation) ——— 20 %. (individual exam)

Final (written) ——— 20 %.

Viva + report ——— 10 %.

Viva + Experiments ——— 40 %.

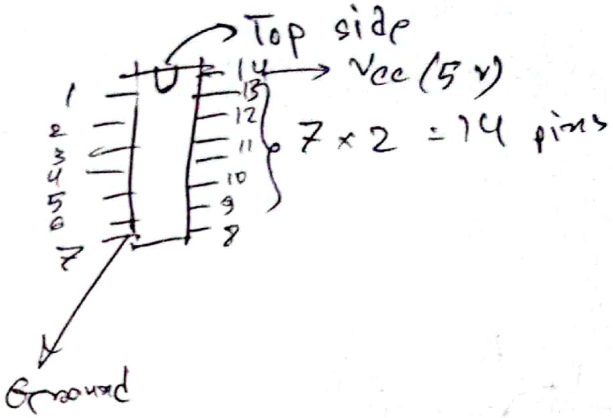
* Lab Report → Every week
  → has viva
  → will be submitted at the end

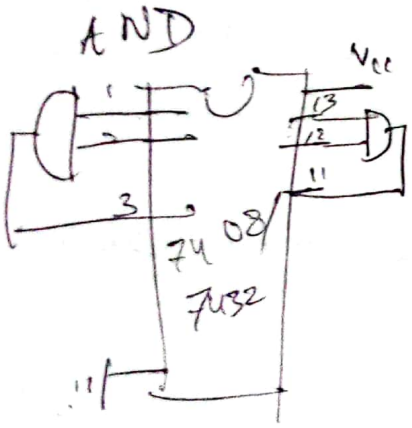
* Attendance → V. V. Imp

* No partial marking

Microprocessors Data handbook
- BCB Publication

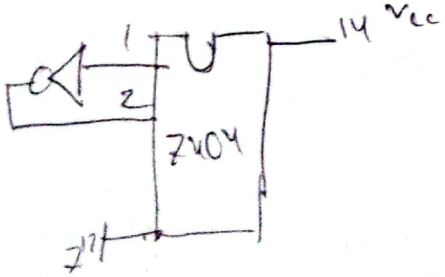
7408 - AND
7432 - OR
7404 - NOT



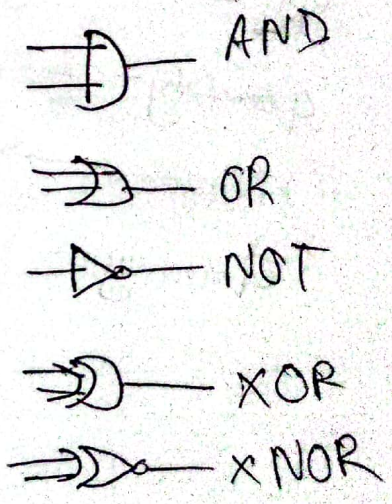


A	B	AND	OR
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

* For < 4, you need 1 IC



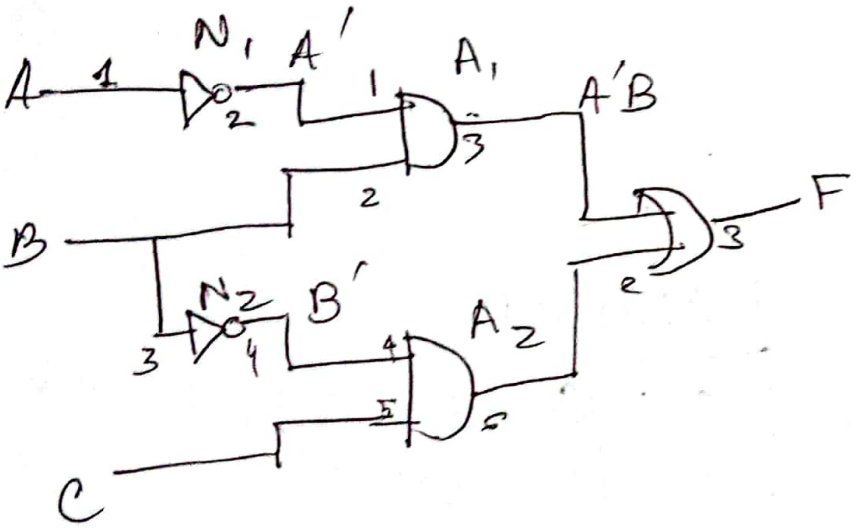
A	NOT
0	1
1	0



Simplification

i) $F = A'B + B'C$

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



Report:

- Expt No.:
- Expt Name
- Objective:
- Truth table
- Circuit Diagram
- Required ICs
- IC pins connections in the diagram
- Discussion (obj but rev)

Theory first → Prove Later

Require:

i) N_1 — NOT gate (7404)

quantity — 1

ii) Name → Gate Name (code)

Quantity —

2105 | 31/03/24

(Syeda Shabnam Miss)

~~Syeda~~

"Easiest one" ~ Miss

"Conceptual course"

"Either a lot of marks or none"

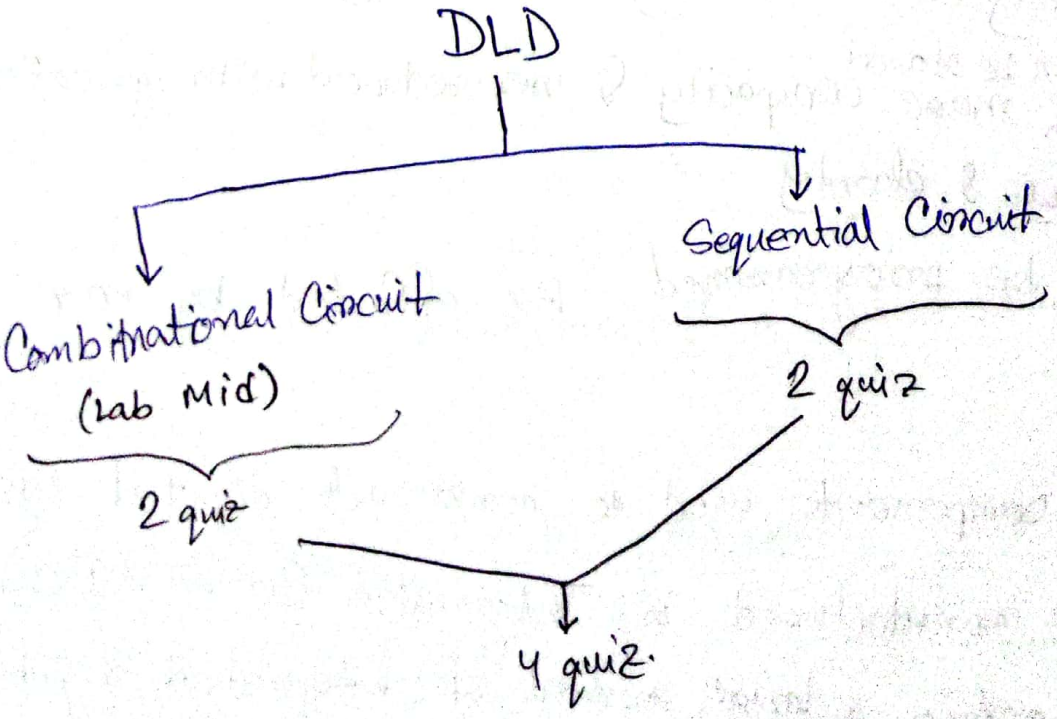
"Fully hardware related course"

"DLD is a core course"

"Book is essential. Slides are complementary"

Digital Design with an Introduction to the Verilog HDL (5th Edition).
~ M. Morris Mano, Michael D Ciletti

*Strategy: Class → Book → Slide
(Otw) (Home) (Before next class)



C-2(W-1) | DLD | 4/4/24

* Lecture - 1:

Electronic Circuits can be divided into two categories:

i) Digital electronics:
- involves quantities with discrete values

ii) Analog electronics:
- involves quantities with continuous values

The term 'digital' is derived from the way computers performs operations; by computing digits.

Advantages:
- Digital data can be processed & transmitted more efficiently and reliably than Analog data
- can be stored more compactly & reproduced with greater accuracy & clarity
- can be programmed, less effected by noise

IC:

The components used to construct digital systems are manufactured in Integrated Circuit (IC) form.
- IC contains a large amount of integrated digital circuit within a single package.

- SSI (Small scale Integration)
- MSI (Medium " ")
- LSI (Large " ")
- VLSI (Very " ")

Digital Logic is concerned with the interconnection among digital components and modules & is a term used to denote the design and analysis of digital systems.

* Digital electronics involves circuits & systems in which there are 2 possible states:

These states are presented by 2 different voltage levels;

A HIGH and ~~Low~~ LOW.

* In digital systems, combinations of 2 states called codes (Binary Numbers)

* The 2 state numbers system is called Binary & its two digits are 0 and 1. A binary digit is called a bit

* HIGH = 1 = ON
  LOW = 0 = OFF
  — V. V. Imp

The voltages used to represent a 1 & 0 are called logic gate.

* 0 - 15 in binary is ESSENTIAL

*Self Study (Important for i) Later || ii) Exams)

- Binary, Octal, Decimal, Hexadecimal number system
  - Conversion
  - Binary addition, subtraction, multiplication
  - 1's complement, 2's complement

C-3(W-2) 18/04/24

Lect - 2 - 3: Introduction to Basic Logic operations & basic digital circuits

*1 - 2 mark question

* Logic / Digital / switching circuit (ckt):

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

*Basic because all ~~gates can be divi~~ logic ends up here when broken down

* Gate:

A logic ckt with one or more input signals but only one output signal (logic 1 or 0)

* Truth Table:

Shows all inputs & output possibilities for a logic ckt. The input words are listed in binary progression.

* Word:

A string of bits that represent a coded instruction or data

* For n inputs, there are 2^n input combinations. Truth table contains one row for each one input combination.

* The AND, OR, NOT, NAND, NOR, XOR, XNOR, operations: Self study

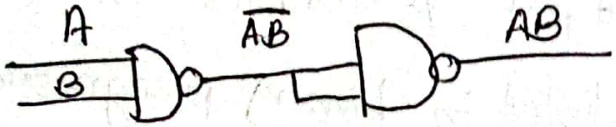
* Universal Gates:

The only one type gates, either NAND or NOR are sufficient for the realization of any logical expression. For this, NAND and NOR gates are known as universal gates.

Realization of basic logic operations using NAND gate:

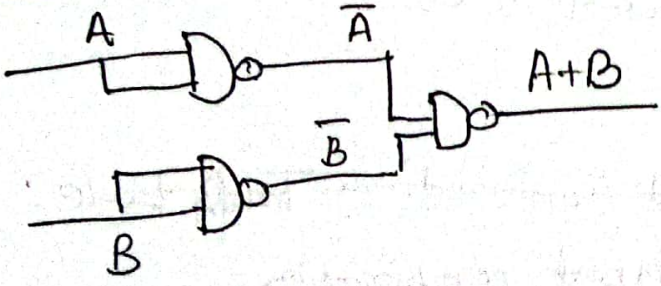
NOT operation:

AND operation:



AND operation:

OR operation:



Realization of basic logic operations using NOR: self study

same formation
juts AND and OR are flipped

* NOR : self study

* Make operation using NAND/NOR → just draw same figure

* XOR for more than two inputs:

- If total no. of '1' is odd, output will be 1
- If total no. of '1' is even, output will be 0

opposite for X-NOR

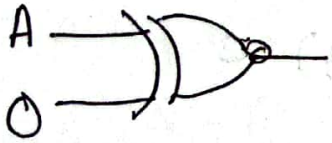
* Realization of NOT operation:

Using X-OR gate:

$\bar{A}$; [the 1 makes it either odd or even]

$A \oplus B = \bar{A}B + A\bar{B} = \bar{A} \cdot 1 + 1 \cdot A = \bar{A}$

Using x-NOR gate:



$\bar{A}$; [the 0 makes it odd or even]

$\overline{A \oplus B} = A \odot B = A'B' + AB = \bar{A} \cdot 1 + A \cdot 0 = \bar{A}$

Lecture 4: Boolean Algebra and Logic

Simplification:

* Introduced by George Boole in 1854

* Boolean: which is always either true or false.

* Variable:

A symbol used to represent a logical quantity.

Any single variable can have a 1 or 0 value.

* Complement:

The inverse of a variable which is indicated by a bar over the variable. If $A = 1$, then $\bar{A} = 0$

* Literal:

A variable or its complement

* Laws of Boolean Algebra: * Writing only once is enough

Commutative Law:

a) $A + B = B + A$

b) $AB = BA$

Associative Law:

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

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

Distributive Law:

a) $A(B+C) = AB+AC$

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

* De-Morgan's Law/Theorem:

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

* 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. $A+\bar{A} = 1$
7. $A\cdot A = A$
8. $A\cdot\bar{A} = 0$
9. $\bar{\bar{A}} = A$
10. $A+AB = A$
11. $A+\bar{A}B = A+B$
12. $(A+B)(A+C) = A+BC$

*See proofs

* Dual: Operators (AND/OR) & Fixed values (1/0) will be changed but variables and NOT operation will not be changed AND and OR will change into each others. For example:

$A+1 = 1$

$A\cdot 0 = 0$

* Duality principle: * 2 mark SQ

- that every algebraic expression deducible from the postulates of Boolean Algebra remains valid if the operators & identity elements are interchanged.

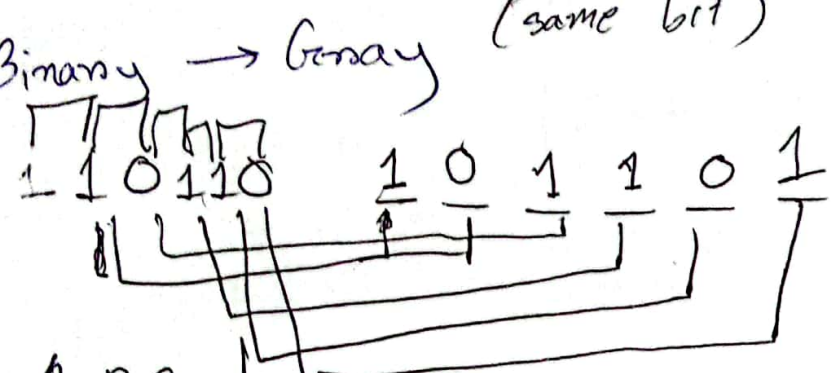
For example: if $A + \bar{A}$ is true, then
$A \cdot \bar{A} = 0$ is also true

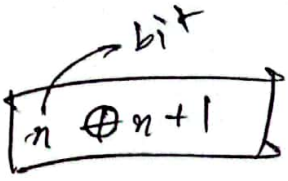
L-2 (w-3)

21/04/24

Gray Code:

Binary → Gray (same bit)





A	B	C	F_3	F_2	F_1
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

* At first place most significant bit

MSB

* Find for which combination,

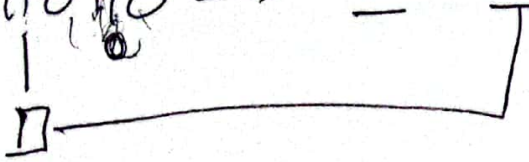
$F_3 = 1$

* Find func:

$F_3 = 1 = AB'C' + AB'C + ABC' + ABC$

Gray → Binary

$(1100)_{Gray} \rightarrow$ 1 0 0 1 0 0



C-4 (w-3) 21/04/24

Example Proof that $A+\bar{A}C = A+C$

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

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

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

$=$ R.H.S

$\therefore$ L.H.S $=$ R.H.S (proved)

HW: Proof that:

i) $x+xy = x(x+y)$

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

Lecture 5: Simplification using Boolean Algebra:

Example: Simplify $[A\bar{B}(C+BD)+\bar{A}\bar{B}]C$

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

$= (A\bar{B}C + A\bar{B}BD + \bar{A}\bar{B})C$; [distributive law]

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

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

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

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

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

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

$= BC \cdot 1$; $[\because A+\bar{A}=1]$

$= \bar{B}C$

H. W: Simplify:

$AB + A(B+C) + B(B+C)$ using Boolean Algebra

Boolean Functions:

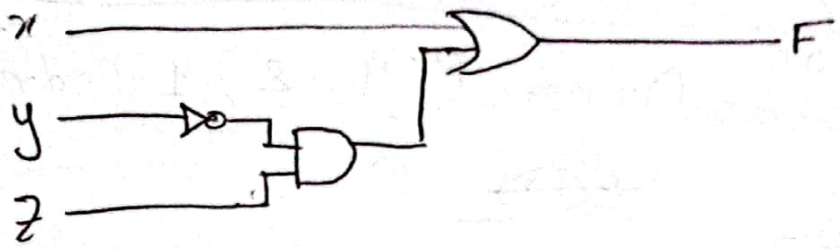
An expression formed with binary variables, AND, OR, NOT operators, parentheses and equal sign

Example: Implement $F_2 = x + y'z$ with basic gates.

↳ output will be either 1 or 0

CSE 2105
Quiz #1 (Sec: A)
28/04/24 SUNDAY
on chap-1 and 2

* One of the ways to simplify is Boolean Simplification
* Method of simplification will be mentioned in the exam



Input

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

*Short cut : Find the ones with 1

For $F = x + y'z$

1. if $x = 1$,
   $F = 1$

2. $y'z = 1$
   $\therefore z = 1$ & $y = 0$
   $F = 1$

Lecture 6: Binary codes for the decimal digits

* Mai Used in encryption

BCD (Binary Coded Decimal) or 8, 4, 2, 1 code

~~BCD~~

~~Binary~~

Decimal	Binary	BCD
0	0000	0000
1	0001	0001
⋮	⋮	⋮
9	1001	1001
⋮	⋮	⋮
~~12~~	1100	00010010
15	1111	00010101 (16, 5)

BCD ≡ Binary

change → separate each digit

1 2 3

0001 0010 0011

$(123)_{10} = (100100011)_{BCD}$

$(279)_{10}$ — 0010 0111 1001 (2, 7, 9)

Excess - 3:

Decimal	Binary	BCD
0	0000	0011
2	0010	0101
9	1001	1100

* Add 3 to decimal

2, 4, 2, 1

Decimal	2 4 2 1
0	0 0 0 0
1	0 0 0 1
5	0 1 0 1
5	1 0 1 1

* [same as BCD each bit has corresponding weight]
* [maximum can be 9]

8, 4, -2, -1

Decimal	8 4 -2 -1
2	0 1 1 0
6	1 0 1 0

* [yeah pretty much same]
* [maximum can be 12]

Coding conversion (Example):

Decimal	2 4 2 1	8 4 -2 -1
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 1 1 1

* (i) Convert to Decimal
* ~~Co~~ (ii) conven to the other

Binary to Gray Conversion:

Binary: 10110

Gray: 11101

→ Either $1 \oplus 0 \oplus 1 \oplus 1 \oplus 0$

→ Or $1+0+1+1+0$

Gray to Binary Conversion:

Gray: 11011

Binary: 10010

Bin: 1101101

Gray: 1011011

Gray: 1010110

Bin: 1100100

* Error Detection:

* Parity Bit: * For Lab * can be added anywhere

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: 10110 ——

* If we use odd parity, P(odd): 0

* " " " even " , P(even): 1

* Parity checkers → CHECKS parity in odd
111 → 1, because it has odd number of 1

* Parity generator → MAKES parity
111 → 0 in odd because it has odd number of 1

Hamming Code: * V. V. Imp for EXAM

A hamming code can be used to detect & correct one bit change in an encoded code word.

* detects and the error
* finds the position of error

Q. Determine the single error correcting code for 10111 0 110

Soln: The no. of data bits $n = 9$.

So, the required no. of parity bits (p) can be obtained from the following formula $2^p >= n + p + 1$

if $p = 1$, $2^1 >= 9 + 1 + 1$ (false)

if $p = 2$, $2^2 >= 9 + 2 + 1$ (false)

if $p = 3$, $2^3 >= 9 + 3 + 1$ (false)

if $p = 4$, $2^4 <= 9 + 4 + 1$ ~~(false)~~ (true)

So, $p = 4$ and we have to send total $n + p = 9 + 4 = 13$ bit message

Sending end: Original message: 101110110

* Use p in 2^n bits,
* Use bit positions that have 1

* As many bits as P_3

Bit position	1	2	3	4	5	6	7	8	9	10	11	12	13
values	P_1	P_2	1	P_3	0	1	1	P_4	1	0	1	1	0

To find the values of P_1, P_2, P_3, P_4, we have to XOR the binary values of bit positions holding 1's

0 0 1 1	= 1 1 0 0
0 1 1 0	$P_4\ P_3\ P_2\ P_1$
0 1 1 1	
1 0 0 1	
1 0 1 0	
1 0 1 1	
(xor) 1 0 0	

So, the sending message will be, 0 0 1 1 0 1 1 1 1 0 1 1 0

(cont)

Cont.

Receiving End :

Case 1: If there is no error in transmitting, the receiver will get 00110111011O.

to verify

* p will be 1,
* XOR positions with 1

```
  00 11  (3)
  0 100  (6)
  0 11 0 (7)
  1 000  (8)      1000 (8)
  1 001  (9)
  1 0 11 (11)
(XOR) 1100 (12)
  ------------
  0 000
```

So, there is no error in the received messages

Case (2): Say, there is an error in the 6th position transmitting the receiver will get: 00110011101 10.

to verify:

```
  00 11
 0 100
  0 111
  1000
  1001
  10 0 0
 (XOR) 1100
 ------------
  0 110
```

So, there is an error in the 6th position to get correct message, we have to flip the value of 6th position:

00110 1 1110110

Chap - 2

Lecture 7: Standard forms of Boolean Expressions

All Boolean expressions can be converted into either of 2 ~~states~~ standard forms:

i) The Sum (OR) of Products (SOP):

Example: $ABC + CDE + B'CD'$

ii) The Product (AND) of Sums (POS):

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

* Standard Form: either SOP or POS

* Non Standard Form: $AB + C(D+E)$ etc

* $AB + CD + CE$

Canonical Form:

Boolean functions expressed as a SUM of min terms or product of maxterms.

- We can derive truth table from standard form

Input	Output	Min terms	Max terms
x y z	F	x = 0 1 y = 1 z = 1	x = 0 y = 0 z = 0
0 0 0	0	$x'y'z'(m_0)$	$(x+y+z)(M_0)$
0 0 1	1	$x'y'z(m_1)$	$(x+y+z')(M_1)$
0 1 0	0	$x'yz'(m_2)$	$(x+y'+z)(M_2)$
0 1 1	1	$x'yz(m_3)$	$(x+y'+z')(M_3)$
1 0 0	1	$xy'z'(m_4)$	$(x'+y+z)(M_4)$
1 0 1	1	$xy'z(m_5)$	$(x'+y+z')(M_5)$
1 1 0	0	$xyz'(m_6)$	$(x'+y'+z)(M_6)$
1 1 1	0	xyz	$(x'+y'+z')(M_7$

By Minterms, $F = m_1 + m_3 + m_4 + m_5$

$= \sum(1, 3, 4, 5)$; [function gives 1]

SOP sign

$\therefore F = x'y'z + x'yz + xy'z + xy'z$

By Maxterms, $F = M_0 M_2 M_6 M_7$

$= \prod(0, 2, 6, 7)$; [function gives 0]

POS sign

$\therefore F = (x+y+z)(x+y'+z)(x'+y'+z')(x'+y'+z')$

* Express the Boolean Function

$F = A + B'C$ as sum of min terms.

Soln: $F = A + B'C$

$= A(B + B') + B'C(A + A')$; [$\because x + x' = 1$]

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

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

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

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

$= m7 + m6 + m5 + m4 + m1$

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

* Express the Boolean Function $F = xy + x'z$ as product of max terms

Soln: $F = xy + x'z$

$= (xy + x')(xy + z)$; [$\because x + yz = (x + y)(x + z)$]

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

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

$= (x + y + zz')(x + z + yy')(y + z + xx')$; [$A \cdot \bar{A} = 0$]

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

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

$= \pi M_0 M_2 M_4 M_5$

$= \pi (0, 2, 4, 5)$

* H.W; Simplify:

i) $F(A, B, C) = \pi(0, 2, 3)$

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

* Converting SOP to POS } self study (Complement each other)
* Converting POS to SOP }

[H] $F(x, y, z) = 1$ means?

↳ always 1

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

$F = \pi(N:D)$

28/04/24

Lecture 8:

Karnaugh Map (K-MAP)

↓

A table of n variables will have 2^n cells

Rule: (SOP)

1. Groups may not include any cell containing a zero.
2. ~~Cells~~ Groups may be horizontal or vertical
3. Groups must contain 1, 2, 4, 8 or 2^n cells
4. ~~Or~~ Each group should be as large as possible
5. Each cell containing a ~~one~~ 1 must be in at least one group.
6. Groups may overlap
7. Groups may wrap around the table
8. There should be as few groups as possible
9.

2-variable K-Map

- self study.

Three variable K-map: * cell address matters

A \ BC	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

Example: Simplify using K-map:

i) $F(A, B, C) = \Sigma(1, 2, 3, 5, 7)$ → can be done in SOP

ii) $F(x, y, z) = \Sigma(3, 4, 6, 7)$ → " " " POS

i)

A \ BC	B'C' (00)	B'C (01)	BC (11)	BC' (10)
A' (0)		1	1	1
A (1)		1	1	

* Groups should be visible
  - use colors if necessary

A'B

$F = A'B + C$

* if group covers all rows an column, it will not be counted

ii)

x \ yz	y'z' (00)	y'z (01)	yz (11)	yz' (10)
x' (0)			1	
x (1)	1		1	1

100 111 110

$yz + xz'$

4 - variable K-Map:

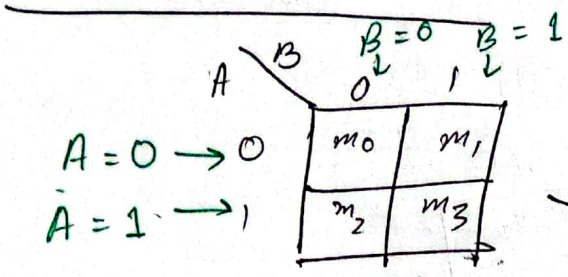
AB \ CD	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

Example: Simplify using K-Map:

i) $F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9)$

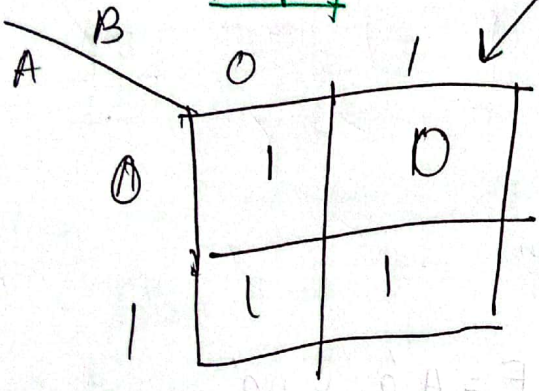
ii) $F(A, B, C, D) = \Sigma(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$

For 2 -variable



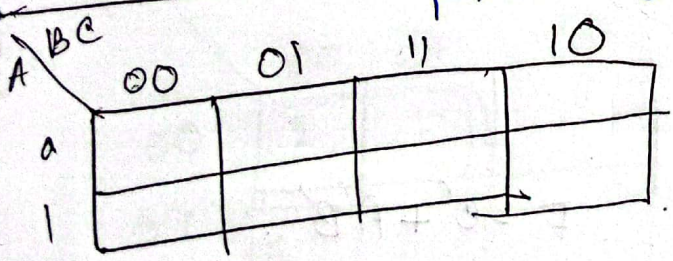
A	B	F
0	0	1
0	1	0
1	0	1
1	1	1

00	01
10	11



For 3 -variable:

→ we use gray code instead of binary order



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

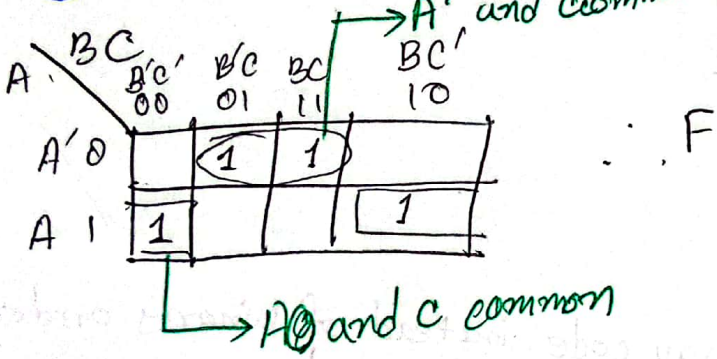
~~Q-5~~ Q-7(w-4) K-Map Practices 02/05/24 (online)

AB \ C	0	1
00	m_0	m_1
01	m_2	m_3
11	m_6	m_7
10	m_4	m_5

A \ BC	00	01	11	10
0	m_1	m_2	m_4	m_3
1	m_4	m_5	m_7	m_6

3 Variable Practice:

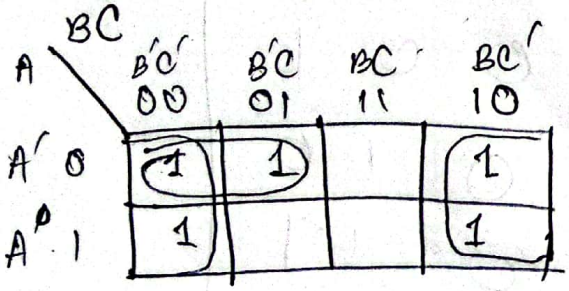
* Simplify $F = \Sigma m(1, 3, 4, 6)$



$\therefore F = AC' + A'C$

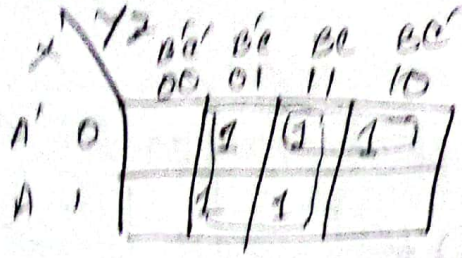
$= A \oplus C$

~~* Σm~~ * $F = \Sigma m(0, 1, 2, 4, 6)$



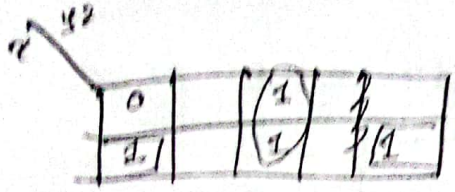
$\therefore F = C' + A'B'$

* $F = \Sigma(1, 2, 3, 5, 7)$



$F = z + x'y$

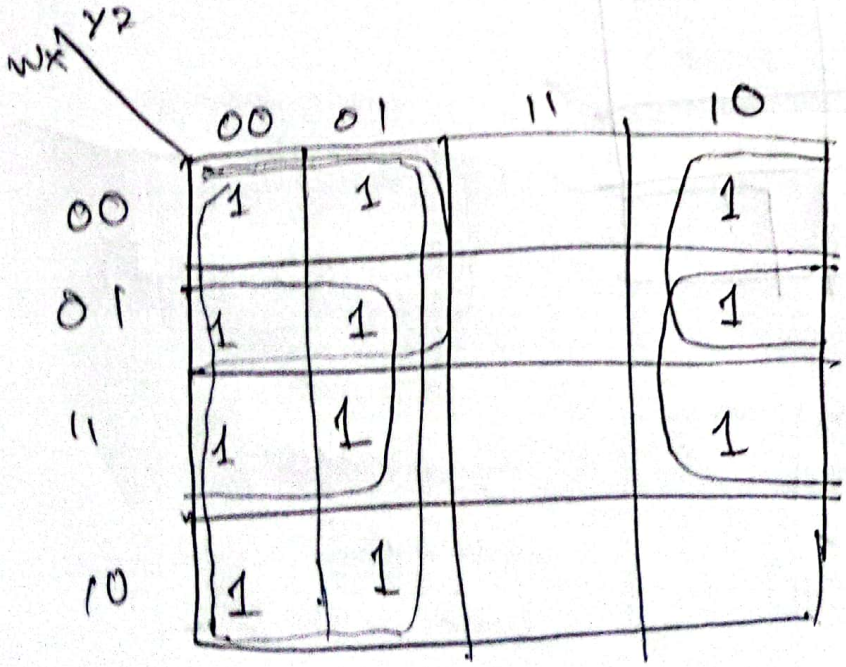
* $F(x, y, z) = \Sigma(3, 4, 6, 7)$



$F = yz + xz'$

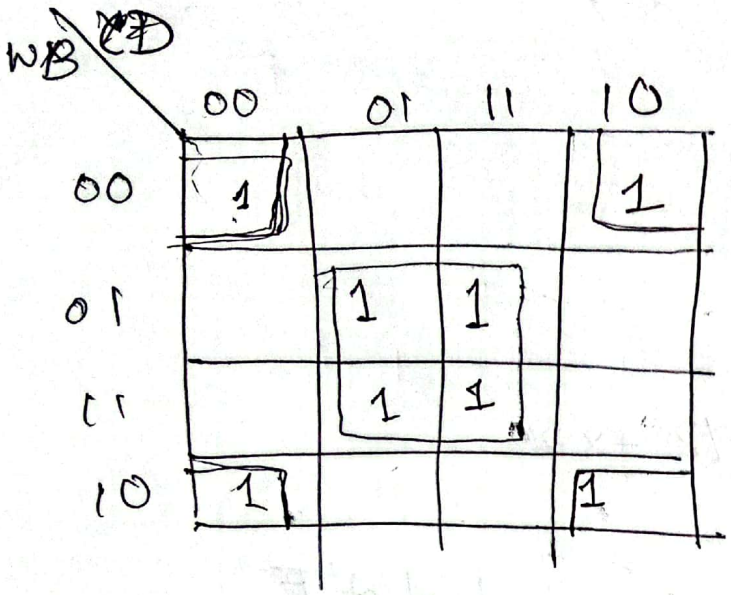
* Question may have Boolean expressions instead of F
  - → Place it directly OR
  - → Express in min terms

* $\Sigma F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$



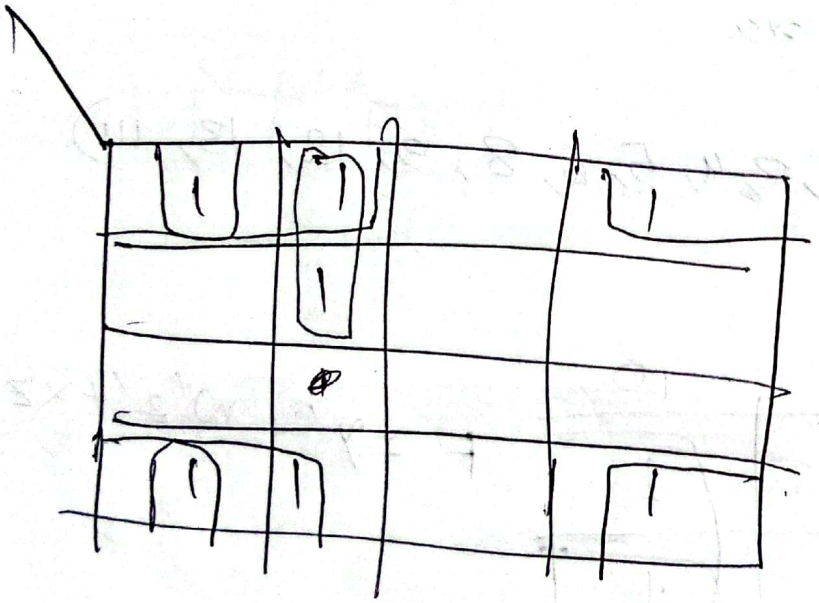
$F = y' + w'z' + xz'$

* $F(W, B, C, D) = \Sigma(0, 2, 5, 7, 8, 10, 13, 15)$



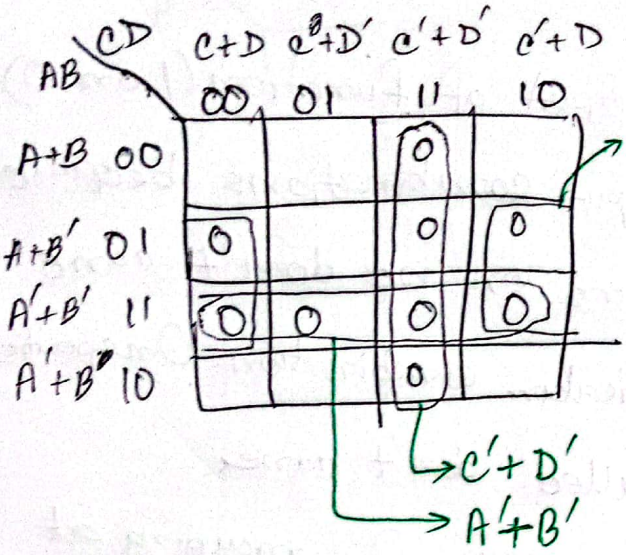
$F = \bar{B}D +$

* $F(A, B, C, D) = \Sigma m(0, 1, 2, 5, 8, 9, 10)$



$\therefore F = B'D' + B'C' + A\bar{C}'D$

* $F(A, B, C, D) = \Pi(3, 4, 6, 7, 11, 12, 13, 14, 15)$



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

C-8 (W-5) 05/05/24

Dont Care:

In some cases, the output of function (1 or 0) is not specified for certain input combinations because the input combination never occurs or we don't care about the outputs of this particular combination. Unspecified minterms for these functions are called don't cares.

Notation: 'X' or 'd' → Not necessary for grouping but can be used for convenience

Examples: Simplify using K-map:

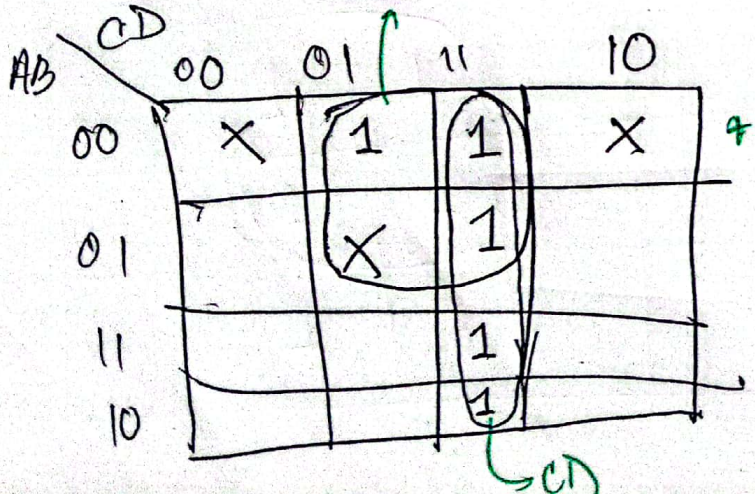
i) $F = \Sigma(1, 3, 7, 11, 15)$

$d = \Sigma(0, 2, 5)$ → d or X

ii) $F = \Sigma(1, 3, 7)$

$d = \Sigma(0, 5)$

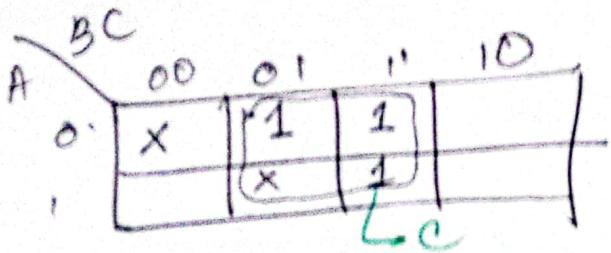
i)



∴ $F = CD + A'D$

Try to take minimum dont care

ii)

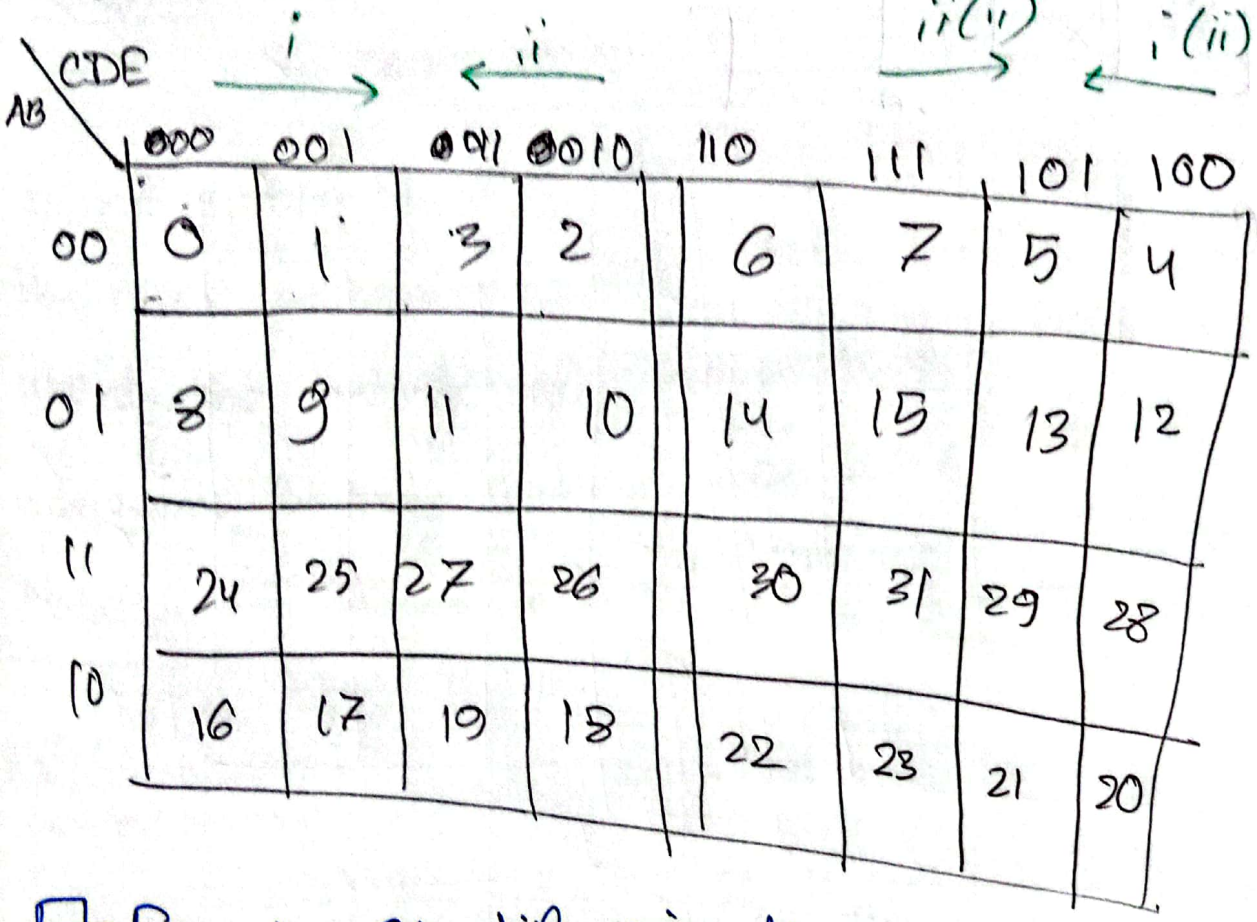


$\therefore F = C$

Lecture 10:

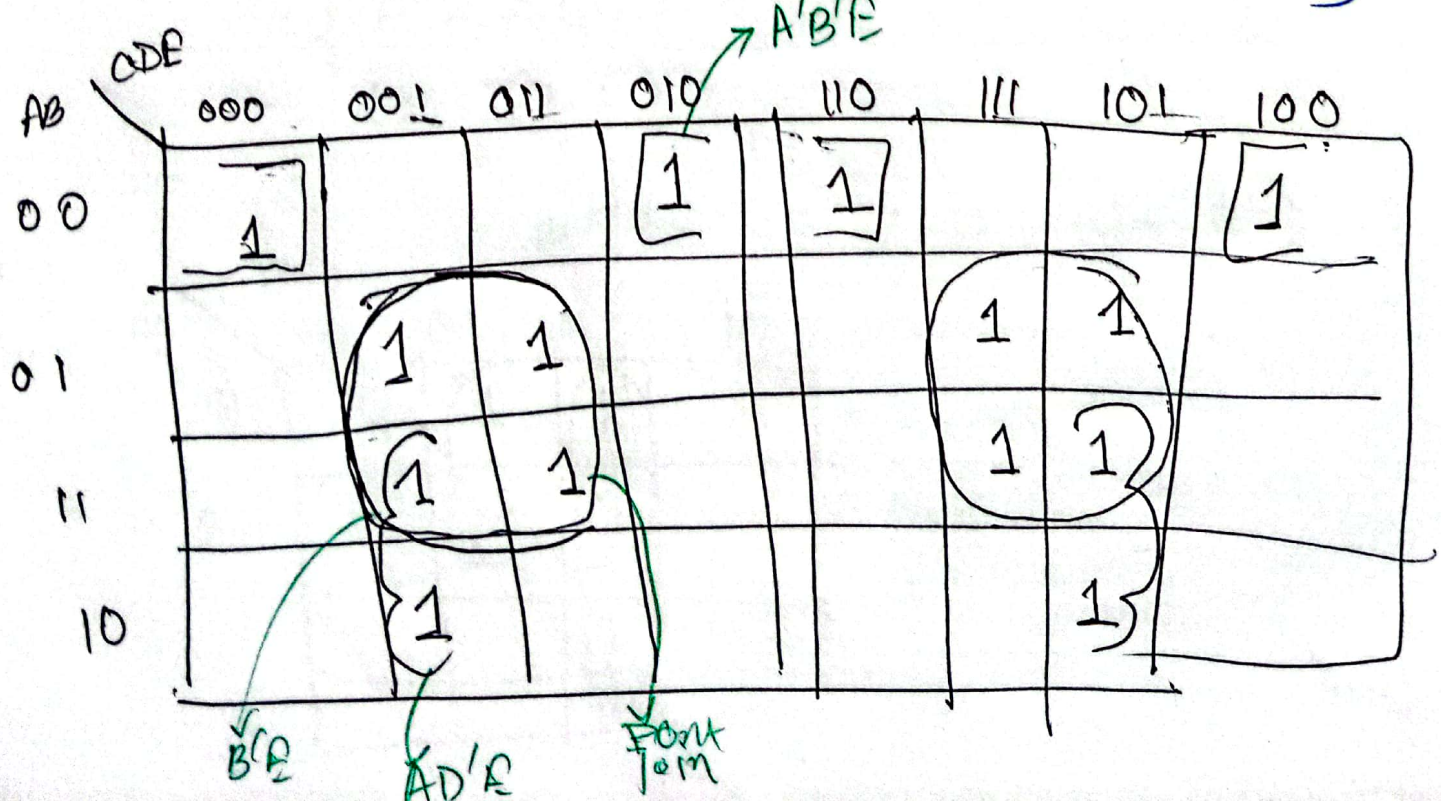
* Minmon Rule:

* 5 variable K-Map



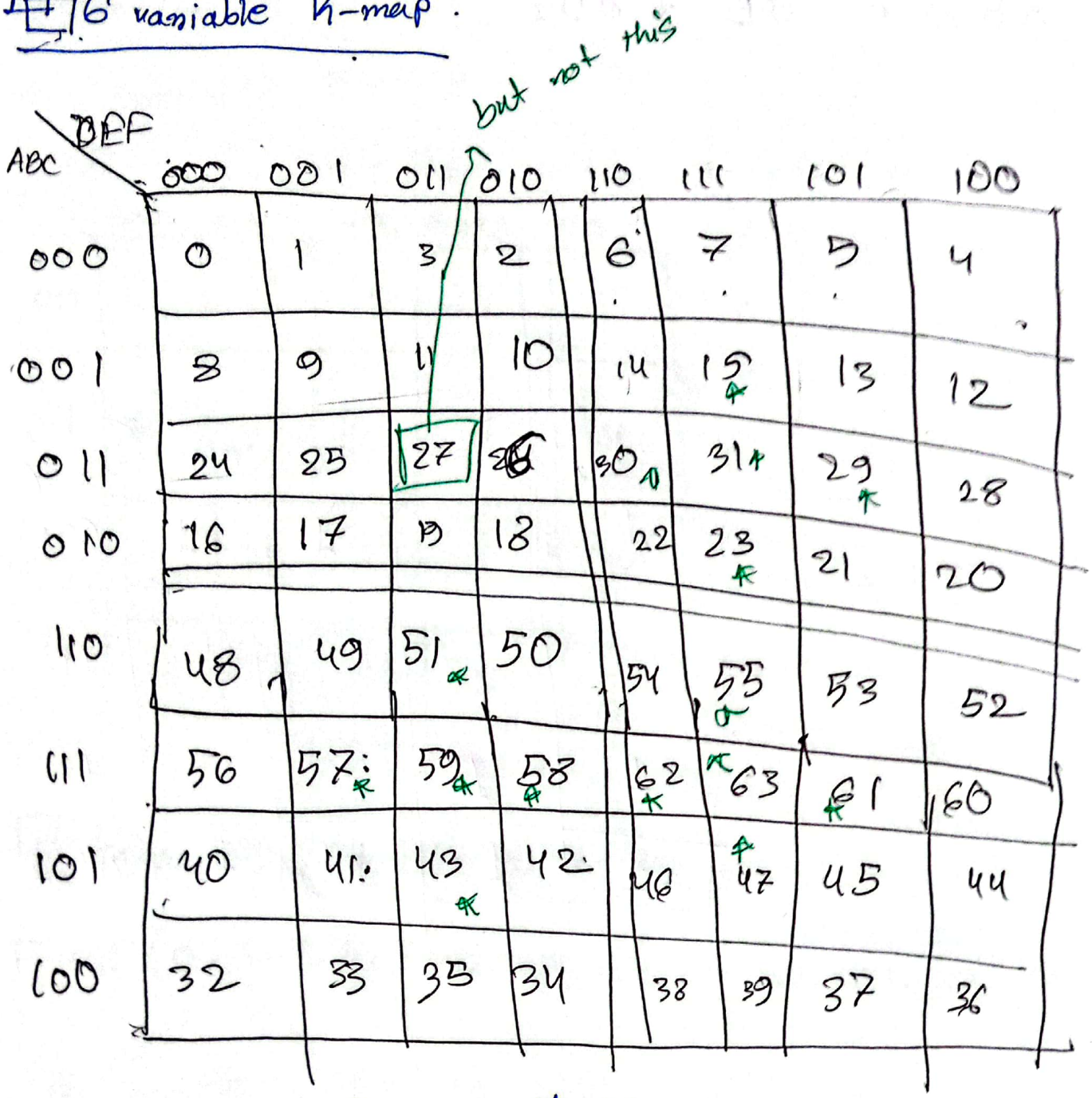
Example: Simplify using K-Map

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



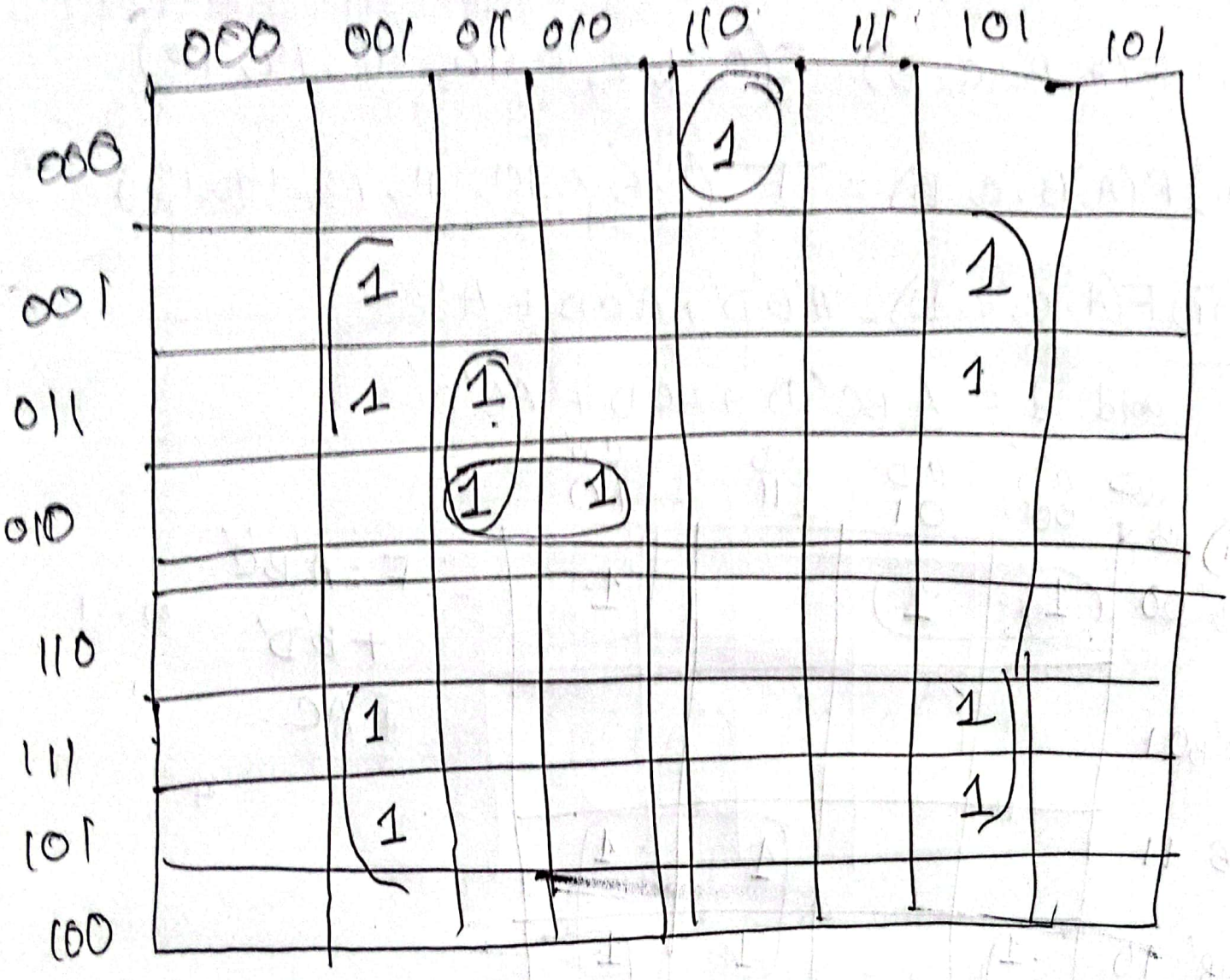
$\therefore F = A'B'E + B'E + AD'E$

6 variable K-map:



Example: Simplify using K-Map:

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



$$F = CE'F + A'B'C'DEF + A'BD'EF + A'BC'D'E$$

Q. W:

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

ii) $F(A, B, C, D) = \Pi(5, 6, 7, 10, 11, 13, 14, 15)$

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

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

i)

AB \ CD	C'D' 00	C'D 01	CD 11	CD' 10
A'B' 00	1	1		1
A'B 01				
AB 11			1	1
AB' 10	1		1	1

$\therefore F = A'B'D' + B'D' + AC$

$n - 1$

$1 \longrightarrow 4$

$2 \longrightarrow 3$

$4 \longrightarrow 2$

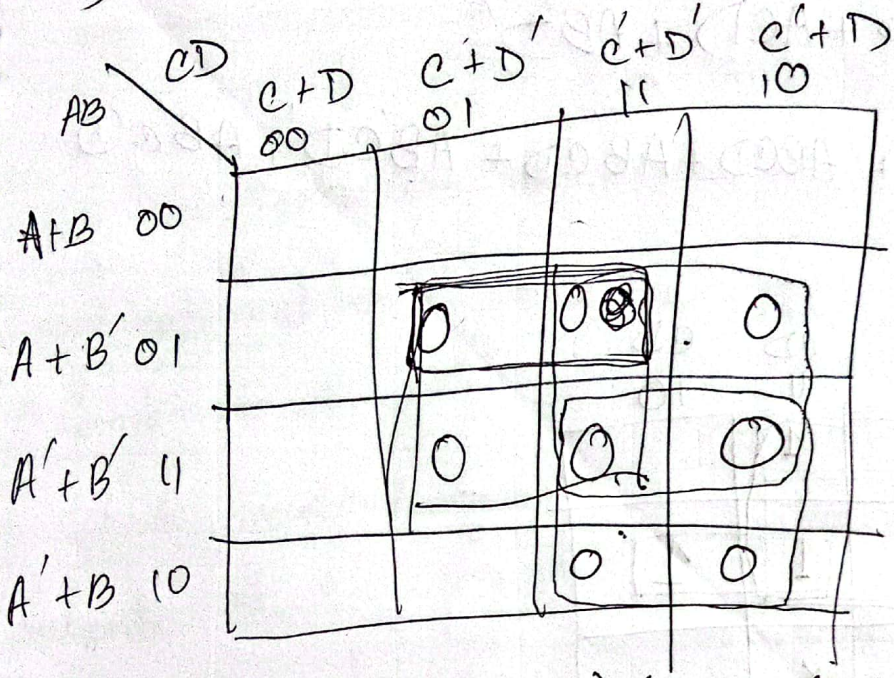
$8 \longrightarrow 1$

2^n

$n - g + 1$

$4 - 2 + 1$.

ii) $F(A, B, C, D) = \Pi(5, 6, 7, 10, 11, 13, 14, 15)$



$\therefore F = \cancel{A+B}\,(\cancel{A}+B'+D')(B'+C')(A'+C')$

110

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

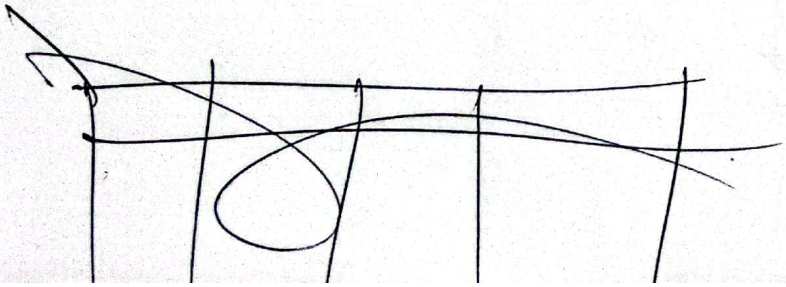
$= A'BC'D + ACD + A'B'D'$

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

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

$= A'B'CD + A'B'C'D + A'BCD + A'B'CD$

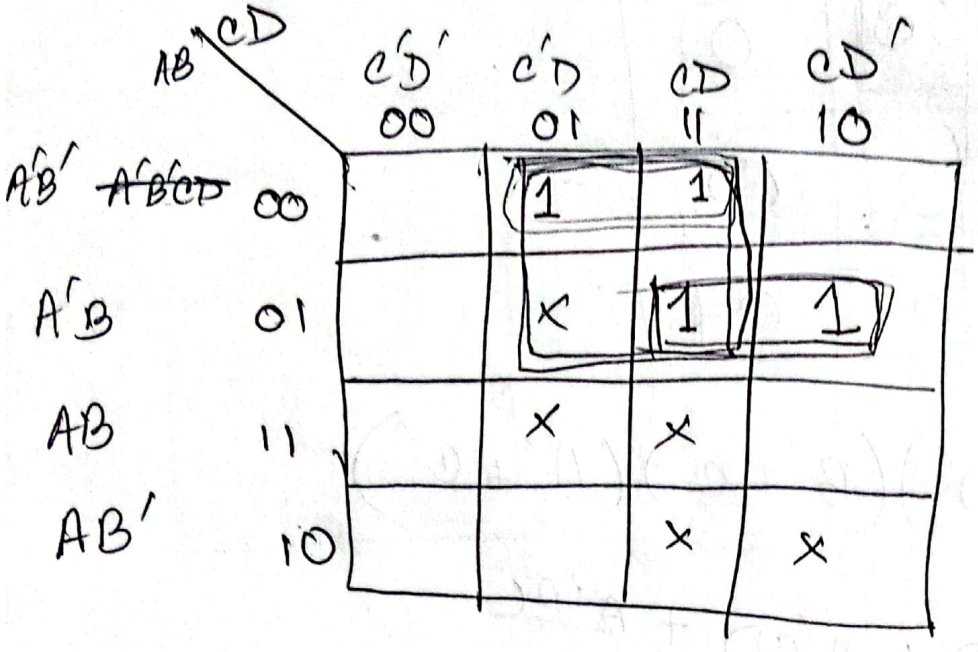
$+ A'BCD + A'BCD'$



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

$d(A, B, C, D) = A'BC'D + ACD + AB'D'$

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



$F = A'D + A'BC$

09/05/24

Lecture -11

Chapter-4 : Combinational Logic

Logic circuits for digital systems may be combinational or sequential circuits

upto now

* Combinational Circuit :

A combinational circuit consists of logic gates whose outputs at any time are ~~detect~~ determined directly from the present combination of inputs without regard to previous inputs. [outputs depend entirely on inputs at that exact moment]

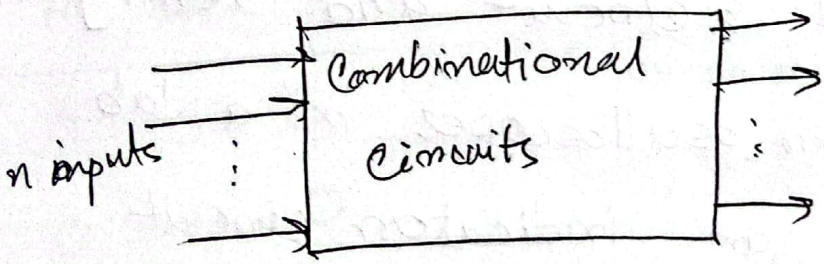


Fig: Block diagram of combinational circuit

* Diff betn seq comb and seq ckts → will be explained in seq quiz

Design Procedure of Combinational Logic Circuits:

Given a problem statement:

i) Determine the number of input variables and output variables. The input and output variables are assigned letter symbols

ii) Derive the truth table to define the required relationships between input and outputs.

iii) Simplify the boolean expressions for each outputs. (K-Map)

iv) Produce the required circuit and verify it. (Pin numbers are not needed in theory)

Example: There are four oscilloscopes in a lab. You have to ~~del~~ design an indicator circuit to ~~determin~~ detect the condition when two or more of the oscilloscopes are malfunctioning

Circuit Diagram

→ Things are drawn ~~#~~ with basic ~~names~~ mentions

→ Detailed

Block Diagram

→ Not mentioned. Makes a block with name

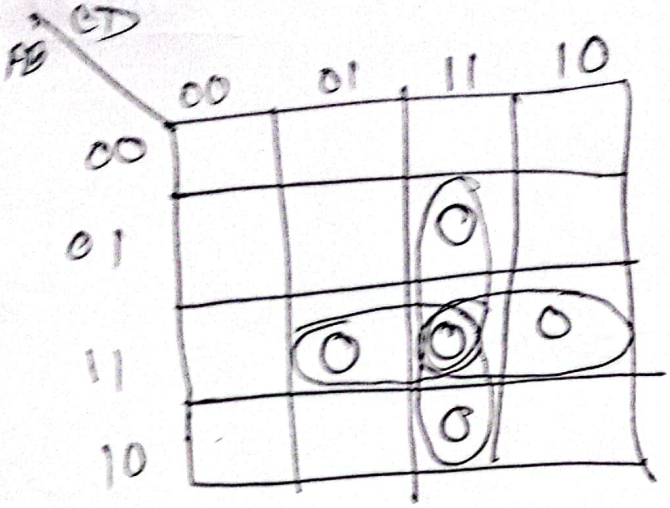
→ Not detailed

Ex →

Let, warning condition be 1
malfunctioning " " 0
∴ The indicator will be 1 when 2 of the bits will be 0

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

A	B	C	D	f=
0	0	0	0	1
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	0



AB

C-10 (W-6) 12/05/24

Binary Adder

* Half Adder :

- Adds 1 bit plus 1 bit
- Produces Sum & Carry

Input		Output	
x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$\therefore C = xy$

$\therefore S = xy' + x'y$

$= x \oplus y$

∴ Half Subtractor : B D
Borrow Diff

Cannot do: carry becomes borrow of 3 bit

1 1 1 0
A = 1 1 1 0
B = 1 1 1 1

*Full Adder:

The full adder accepts 2 input bits & an input carry & generates a sum and an output carry

X	Y	C_{in} (Cout of prev. bit)	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Carry = 1 1 1 . 0

A = 1 1 0 1 1 0 1

+ B = 1 1 1 1 1 0 1

} 7 Adders

[1] Arrangement of 2 half adders to form a full adder:

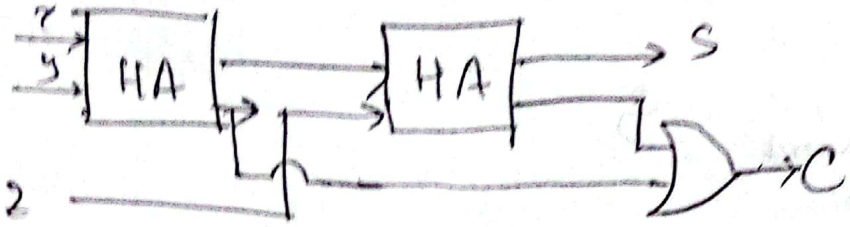


Fig: ~~Diagram~~ Block diagram of a full adder using 2 half adders.

[1] Self Study:

i) Half Subtractor [produce $x+y$]
ii) Full subtractor [produces $x-y-z$]
iii) Binary Subtractor

Magnitude Comparator

- A magnitude comparator is a combinational circuit that compares two numbers A and B and determines whether $A > B$, $A = B$ or $A < B$.

1 bit magnitude comparator:

Input		Outputs		
X	Y	$F_1 (A > B)$	$F_2 (A = B)$	$F_3 (A < B)$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0
		xy'	$x'y' + xy$	$x'y$

$= x \odot y$

4] 2-bit, 3-bit, 4-bit Comparators:

2 bit

X	Y	$F_1 (A>B)$	$F_2 (A=B)$	$F_3 (X<Y)$
$X_2 X_1$	$Y_2 Y_1$			
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
0 1	0 1	0	1	0
0 1	1 0	0	0	1
0 1	1 1	0	0	1
1 1	0 0	1	1	0
1 1	0 1	1	0	0
1 1	1 0	1	0	0
1 1	1 1	0	1	0

[Fix at home]

3-bit:

$X = X_3 X_2 X_1$

$Y = Y_3 Y_2 Y_1$

$F_1(X > Y)$: if $(X_3 > Y_3)$ +

if $(X_3 = Y_3)$ && $(X_2 > Y_2)$ +

if $(X_3 = Y_3)$ && $(X_2 = Y_2)$ && $(X_1 > Y_1)$

$= X_3 Y_3' + (X_3 \odot Y_3)(X_2 Y_2') + (X_3 \odot Y_3)(X_2 \odot Y_2)(X_1 Y_1')$

(from 1-bit)

$F_2(X = Y)$ = if $(X_3 = Y_3)$ && $(X_2 = Y_2)$ && $(X_1 = Y_1)$

$= (X_3 \odot Y_3)(X_2 \odot Y_2)(X_1 \odot Y_1)$

$F_3(X < Y)$ = if $(X_3 < Y_3)$ +

if $(X_3 = Y_3)(X_2 < Y_2)$ +

if $(X_3 = Y_3)(X_2 = Y_2)(X_1 < Y_1)$

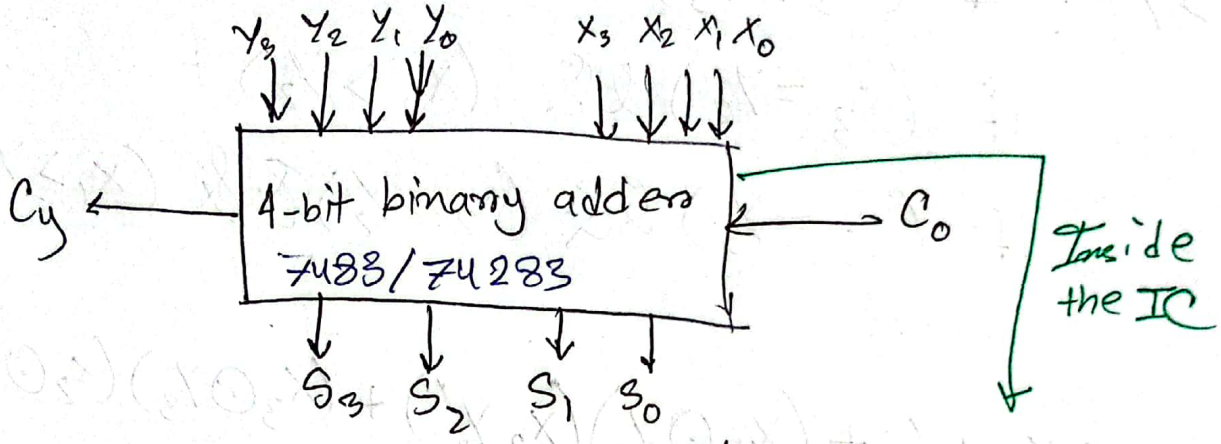
$= X_3' Y_3 + (X_3 \odot Y_3)(X_2' Y_2) + (X_3 \odot Y_2)(X_2 \odot Y_2)(X_1' Y_1)$

C-11 (W-6) 16/05/24

* Draw Diagrams. ALWAYS

Binary Parallel Adder:

- produces the arithmetic sum of two binary numbers in parallel

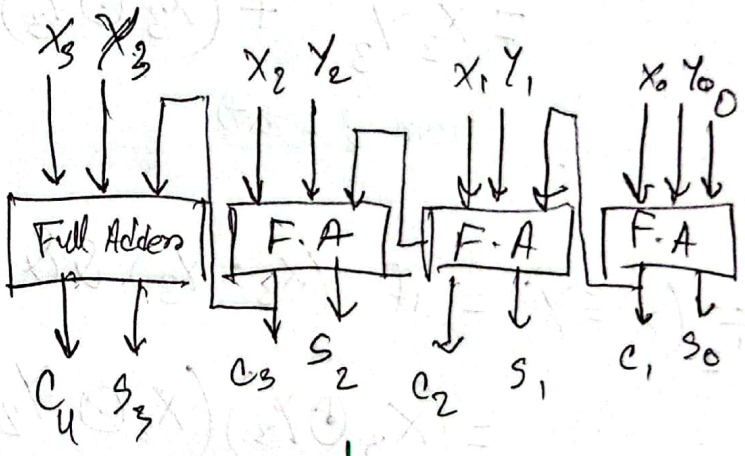


$X = X_3 X_2 X_1 X_0$

$Y = Y_3 Y_2 Y_1 Y_0$

$$\begin{array}{r} C_3\ C_2\ C_1 \\ X_3\ X_2\ X_1\ X_0 \\ +\ Y_3\ Y_2\ Y_1\ Y_0 \quad C_0 = 0 \\ \hline C_y\ \ S_3\ S_2\ S_1\ S_0 \end{array}$$

C_y and S_3 and ... C_2 and C_1



Time delay

(Unless Carry is completed, the next operation cannot happen)

↳ Ripple Effect

[pg] Carry look ahead Adder (CLA)

- A method of speeding up the addition process by eliminating the ripple carry delay is called look ahead carry addition. In CLA, we define 2 new binary variables P_i and G_i, where,

$$P_i = A_i \oplus B_i \quad , \quad G_i = A_i B_i$$

P_i → Carry propagate G_i → Carry generate

See, the output sum (S) and Carry (C) of full adder can be expressed as,

$$S_i = P_i \oplus C_i \quad , \quad C_{i+1} = G_i + C_i P_i$$

[pg] 3-bit CLA adder:

$$S_1 = P_1 \oplus C_1 = P_1 \oplus 0 = P_1$$

$$\therefore C_1 = P_1 = A \oplus B_1 \quad [\because P_i = A_i \oplus B_i]$$

$A = A_3 A_2 A_1$,
$B = B_3 B_2 B_1$.

$\therefore C_1$: initial carry $= 0$.

Similarly, $C_2 = G_1 + P_1 C_1 = A_1 B_1 + 0 \cdot A_1 B_1$

$$S_2 = P_2 \oplus C_2 = (A_2 \oplus B_2) \oplus C_2$$

$$C_3 = G_2 + P_2 = A_2 B_2 + (A_2 \oplus B_2) A_1 B_1$$

$$S_3 = P_3 \oplus C_3 = (A_3 \oplus B_3) \oplus [A_2 B_2 + (A_2 \oplus B_2) A_1 B_1]$$

$$C_4 = G_3 + P_3 C_3 = A_3 B_3 + (A_3 \oplus B_3)[A_2 B_2 + (A_2 + B_2) A_1 B_1]$$

Design a BCD adder using Binary adder

input : two BCD digits A & B

output : A + B (in BCD)

A : 0 - 9

B : 0 - 9

A + B = BCD Sum

C-12(W-Z)

19/05/24

26/05/24	Lecture 8-13
SUNDAY	
DLD Quiz-02	

#1. Design a BCD adder using binary adders & basic gates.

#2. Design a switch controlled 4-bit adder-subtractor circuit. When switch(s)=0, Your circuit will perform A+B, when s=1, your circuit will perform A-B.

*Working Procedure is must

*Questions without truth table needs working procedure

BCD Adder : A

Adds two BCD digits in parallel and produces a sum also in BCD.

- The output sum cannot be greater than $(9+9+1) = 19$

Sum (in decimal)	Sum (in binary)	Required Sum (in BCD) (when $C_{out} = 1$)
n	$C_y\ S_3\ S_2\ S_1\ S_0$	$C_y\ S_3\ S_2\ S_1\ S_0$
9	0 1 0 0 1	0 1 0 0 1 (9)
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	1 0 0 1 0 (18)
13	0 1 1 0 1	1 0 0 1 1 (19)
14	0 1 1 1 0	1 0 1 0 0 (20)
15	0 1 1 1 1	1 0 1 0 1 (21)
16	1 0 0 0 0	1 0 1 1 0 (22)
17	1 0 0 0 1	1 0 1 1 1 (23)
18	1 0 0 1 0	1 1 0 0 0 (24)
A	1 0 0 1 1	

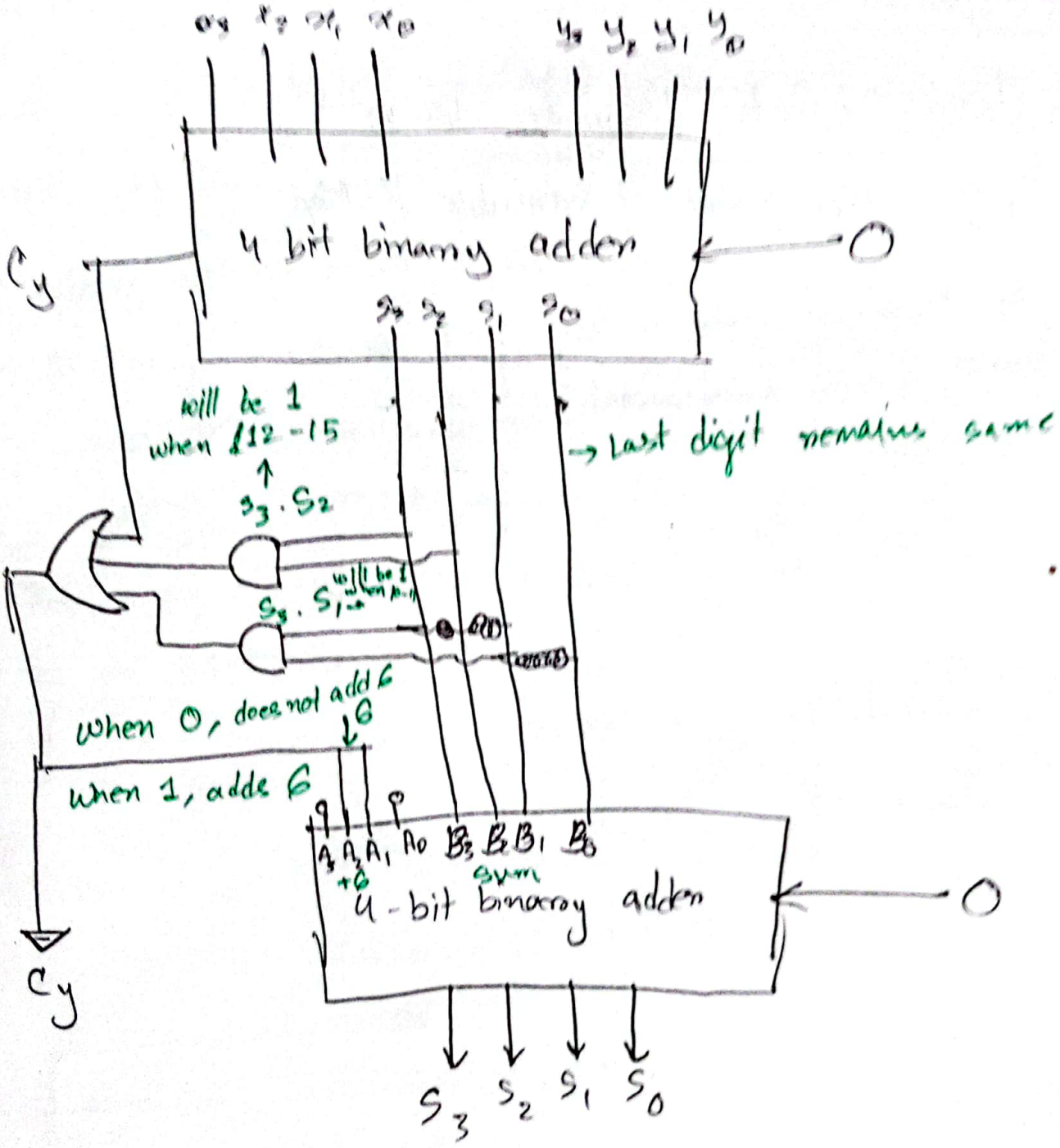


Fig: Block diagram of BCD adder using binary adders.

For Book:

Error Checker: Gets 1 when 10-15

then makes 4 variable K-Map

C-13 (W-7) 23/05/24

Multiplexers / MUX

Data Selector: → same

- Selects one of many input lines (2^n) and directs it to its single output line. There are n selection lines whose bit combinations determine which input is selected. input (2^n), output (1), selector (n) $n = 1 \ldots \infty$
- The size of a MUX is specified by the number 2^n of its input lines the single output line.
- It also contains n selectors.

Example: 2×1, 4×1, 16×2, 32×1, MUX etc.

↓ Input ↓ Output

4×1 MUX → 4 input, 2 selector, 1 output } 4×1

Function table:

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

Selectors determine what outcome is revealed

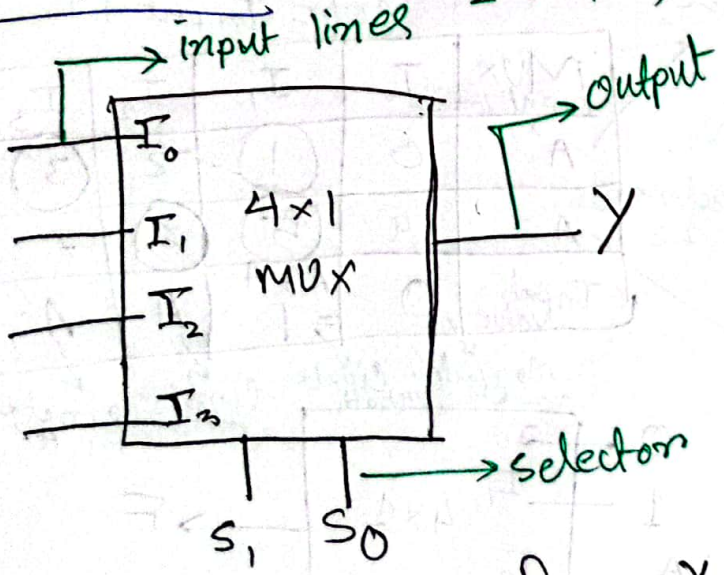


Fig: Block Diagram of a 4×1 MUX

Output equation:

$Y = S_1' S_0' I_0 + S_1' S_0 I_1 + S_1 S_0' I_2 + S_1 S_0 I_3$

Logic/Circuit Diagram: See yourself

Uses of MUX:

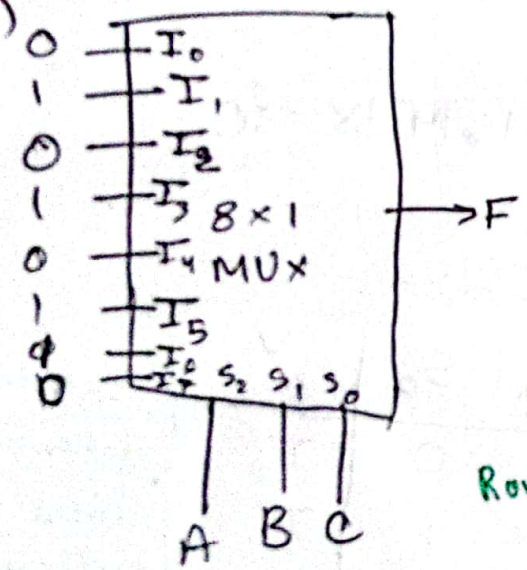
- Connecting two or more sources to a single destination
- It is useful for constructing a common bus system

Function Implementation Using MUX

Example: Implement $F(A, B, C) = \Sigma(1, 3, 5, 6)$ using: i) 8×1 MUX, ii) 4×1 MUX

Soln:

i)



ii) Using 4×1 MUX implementation table using B, C as selectors, A in the input line.

Implementation Table:

Col = Input + 1

MUX input line	I_0	I_1	I_2	I_3
A′	0	(1)	2	(3)
A	4	(5)	(6)	7
Input Value	0	A′+A = 1	A	A′

Row 0 = No. of selectors as input +2

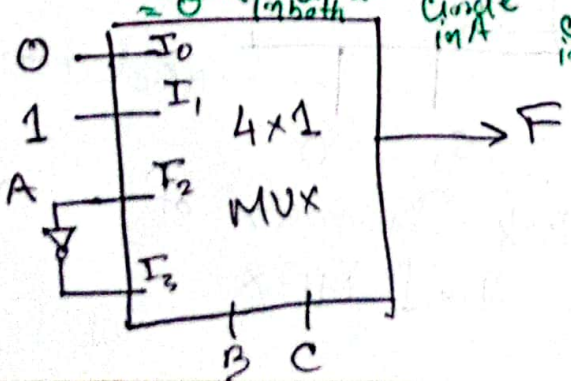
A = 0

A = 1

No circle in both

Circle in A′

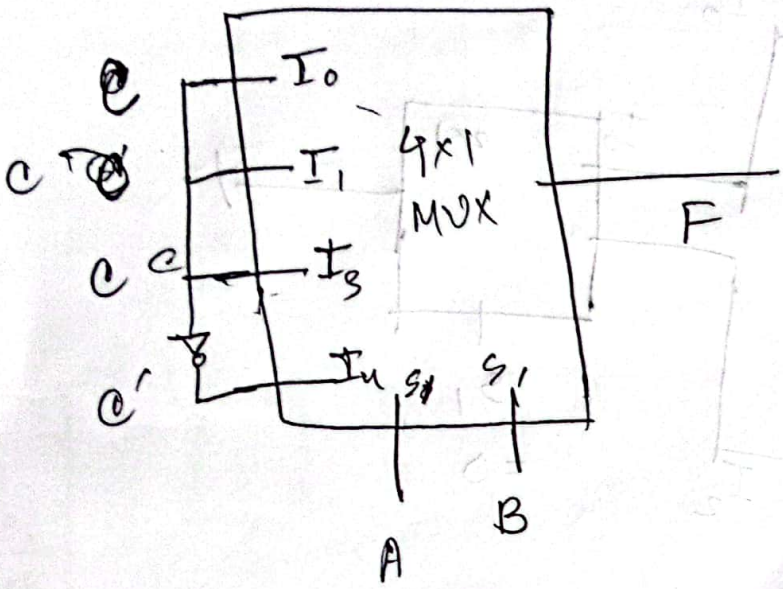
Circle in A



~~C.W~~ C.W :

Implement $F = \Sigma(1, 3, 5, 6)$ using 4×1 MUX considering A, B as selectors & C as input line

* Quiz - 3 and 4 on ~~Quiz~~ W = 13, 14.



MUX Line	I_0	I_1	I_2	I_3
C'	0	2	4	6
C	1	3	5	7
Input Value	C	C	C	C'

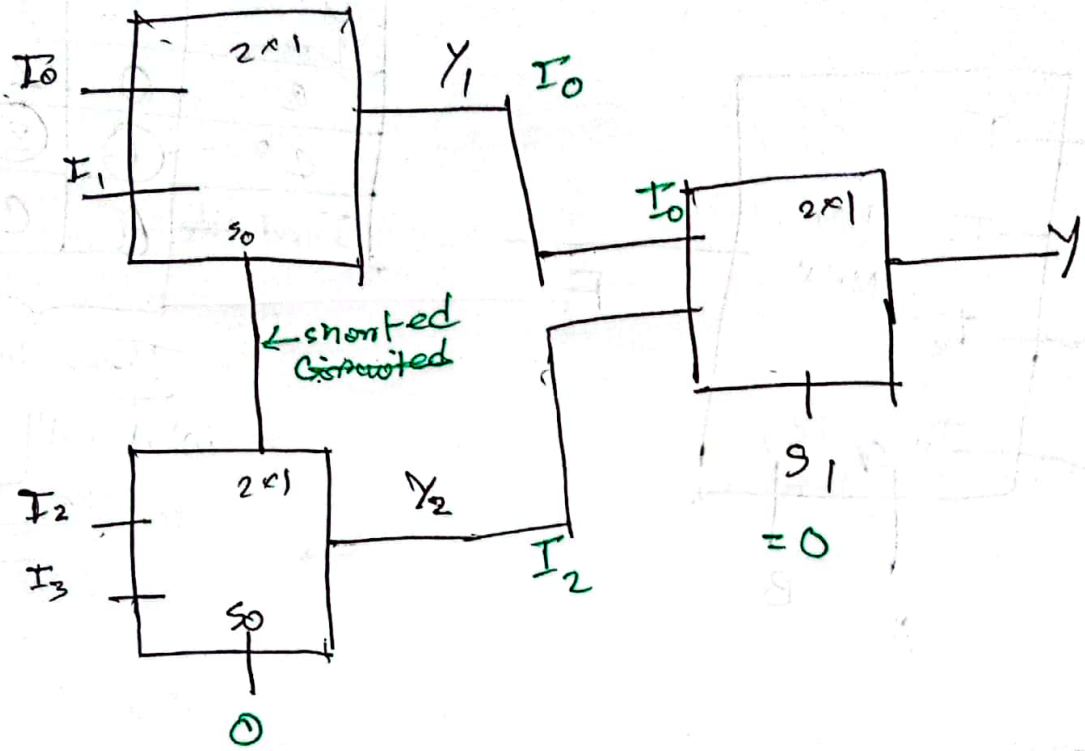
These determine what will be in the Input

C-14 (w-8)

30/05/24

Multiplexer Expansion:

* 4×1 MUX using 2×1 MUX



* Description: Say different values yield different values

— Not all values of have to be explained

* MUX

- Input line
- Output line
- Selectors
- Enable pin

Active High Enable:

The output is enabled when E = 1.

Notation: E (5V)

↳ E ——

Active Low Enable:

The output is enabled when E = 0.

Notation: $\overline{E}$ (Gnd)

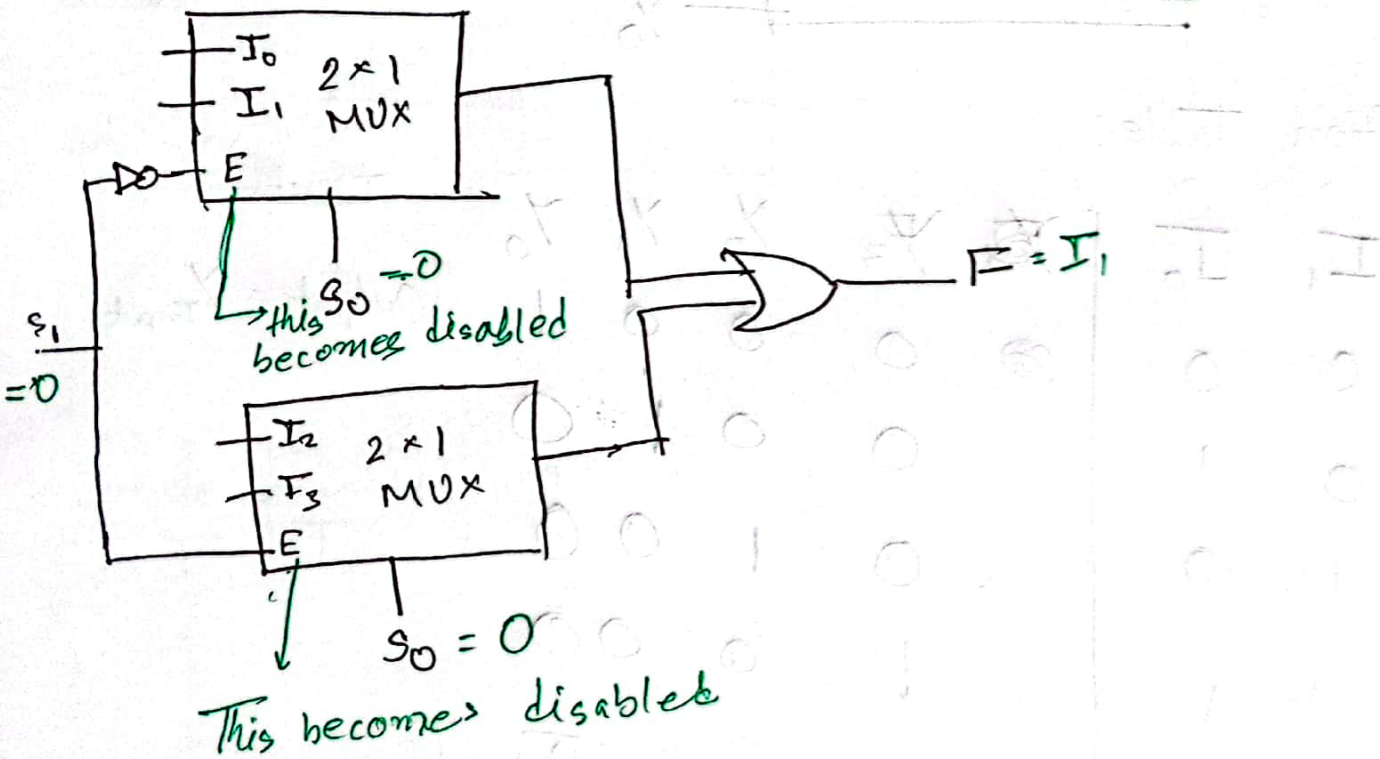
↳ E —o

Based on type of MUX

E_1 $\overline{E}_2$ E_3

↓ ↓ ↓

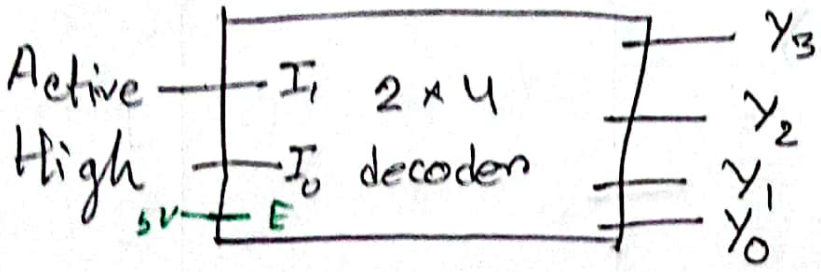
5V 0V 5V



Decoder:

Converts binary information from n input lines (starts from 2) to a maximum of 2^n unique output lines. [two selector]

* 2 to 4 lines Decoder:



Function Table:

I_1	I_0	Y_3	Y_2	Y_1	Y_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

Input

Output = Y_{Input}

$Y_3 = I_1 I_0$

$Y_2 = \overline{I_1} I_0$

$Y_1 = I_1 \overline{I_0}$

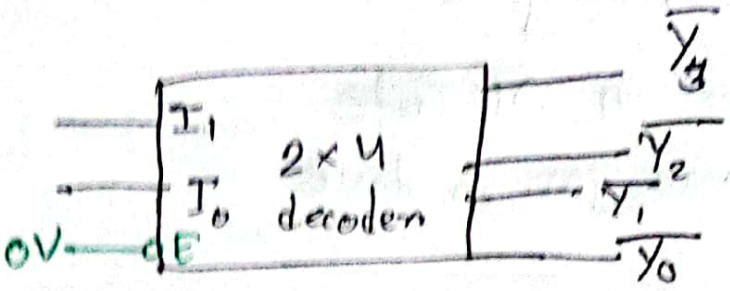
$Y_0 = \overline{I_1}\,\overline{I_0}$

Circuit diagram - draw yourself

I_1 I_0

* Active low

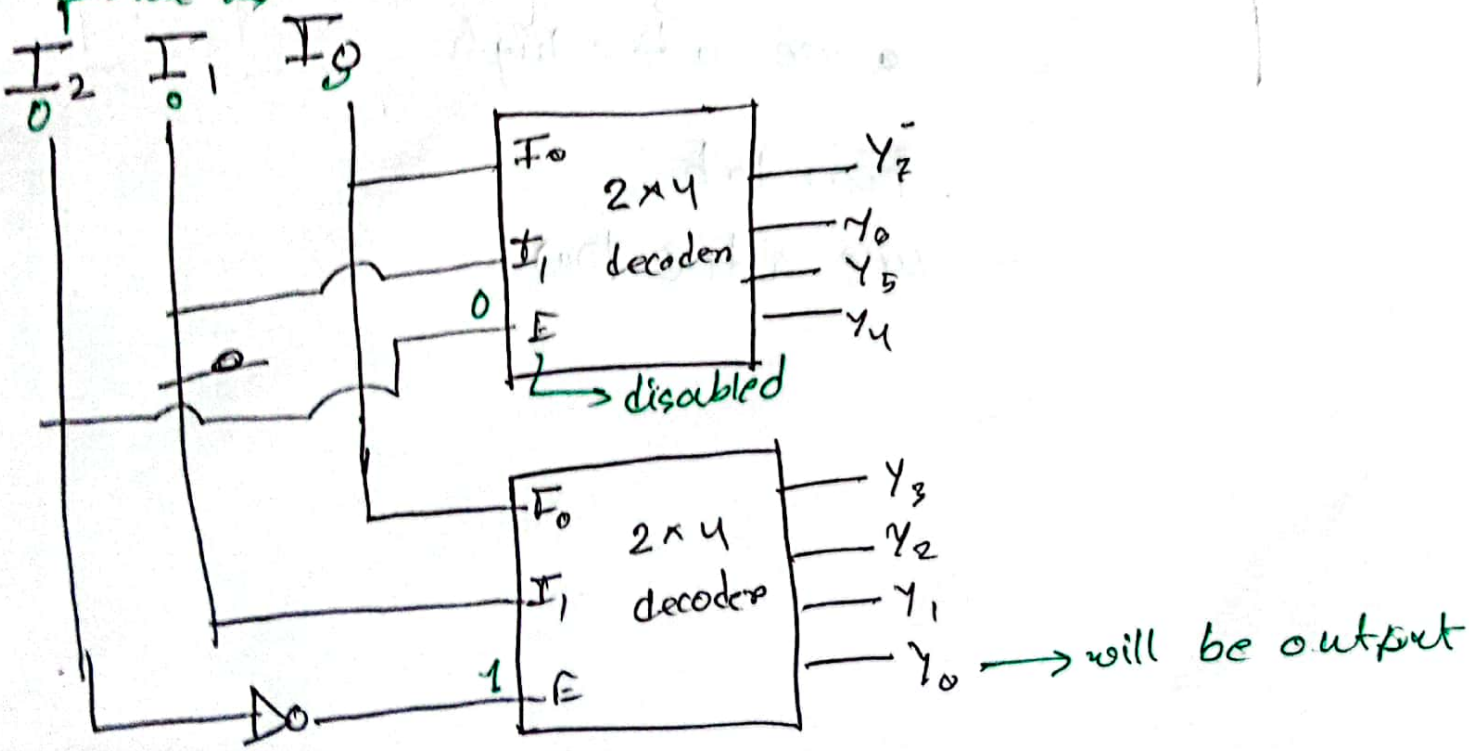
2×4 decoder



I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1

Design a 3×8 line decoder using two 2×4 decoders

→ use as enable pin

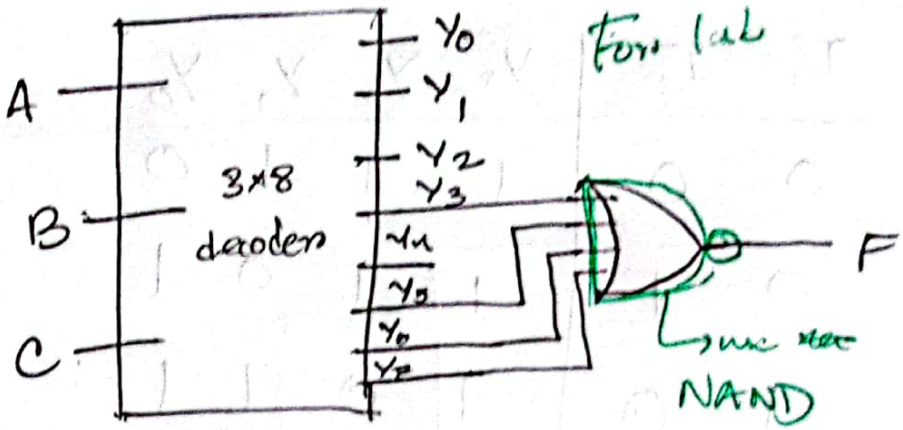


*Boolean Function implementation using Decoders.

#Implement $F = \sum(3, 5, 6, 7)$ using a 3×8 decoder & basic gate

↳ put in OR Gate and (or) is done

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

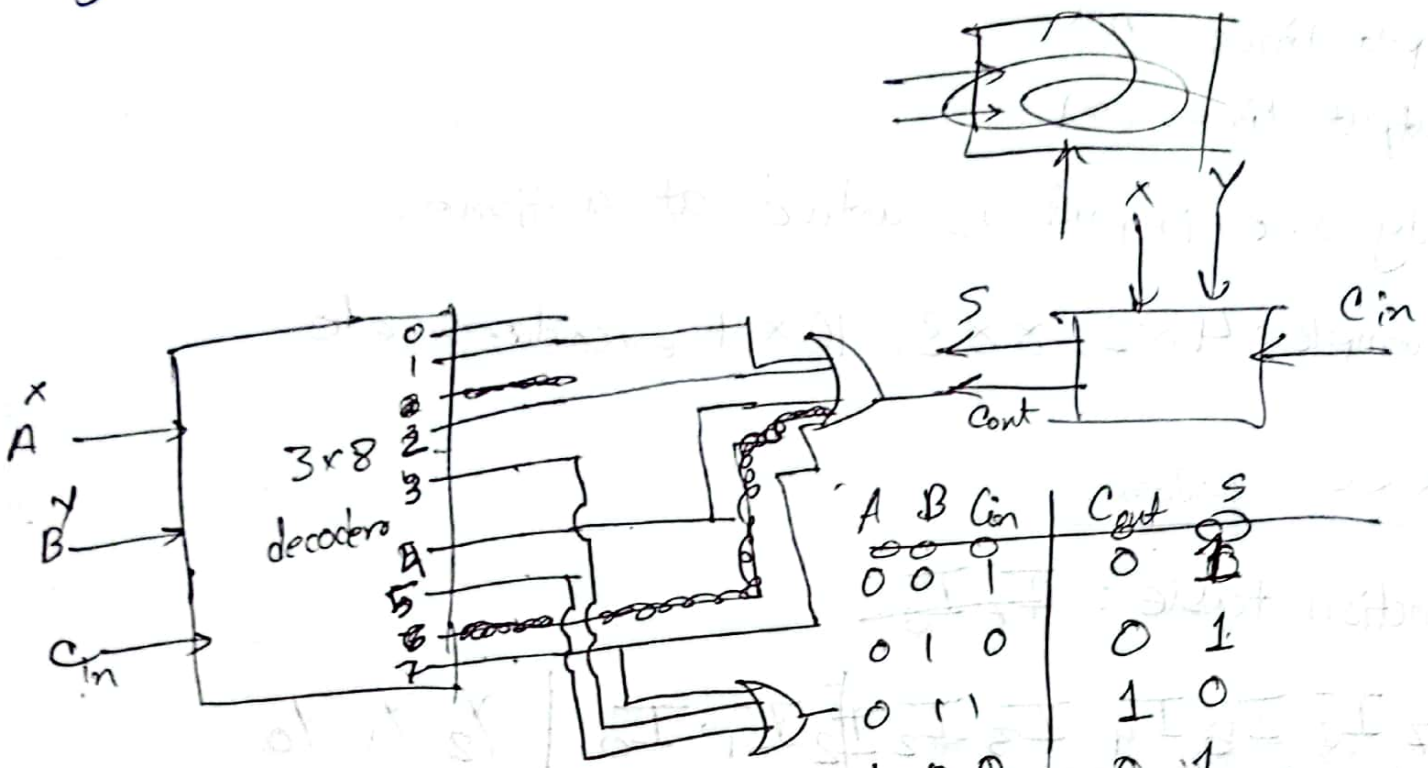


*For theory,
if nothing is presented,
@ use active high

For lab,
use active low

Pw: Implement 1 bit full adder using 3×8 decoder

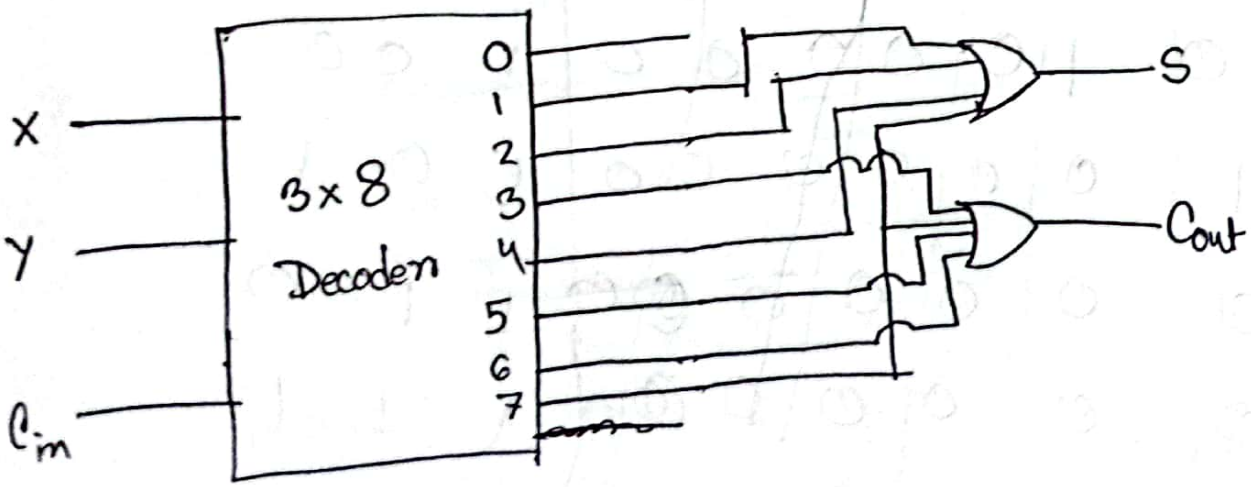
using 3×8 decoder & basic gates.



A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$S = m_1 + m_2 + m_4 + m_7$

$C_{out} = m_3 + m_5 + m_6 + m_7$



02/06/24

Encoder:

- Does reverse operation of decoder
- input lines: 2^n
  output lines: n
- Only one input is active at a time.
- Example: 4×2, 8×3, 16×4 encoders etc

* 8×3 encoder:

Function table: ~~I_7 I_6~~

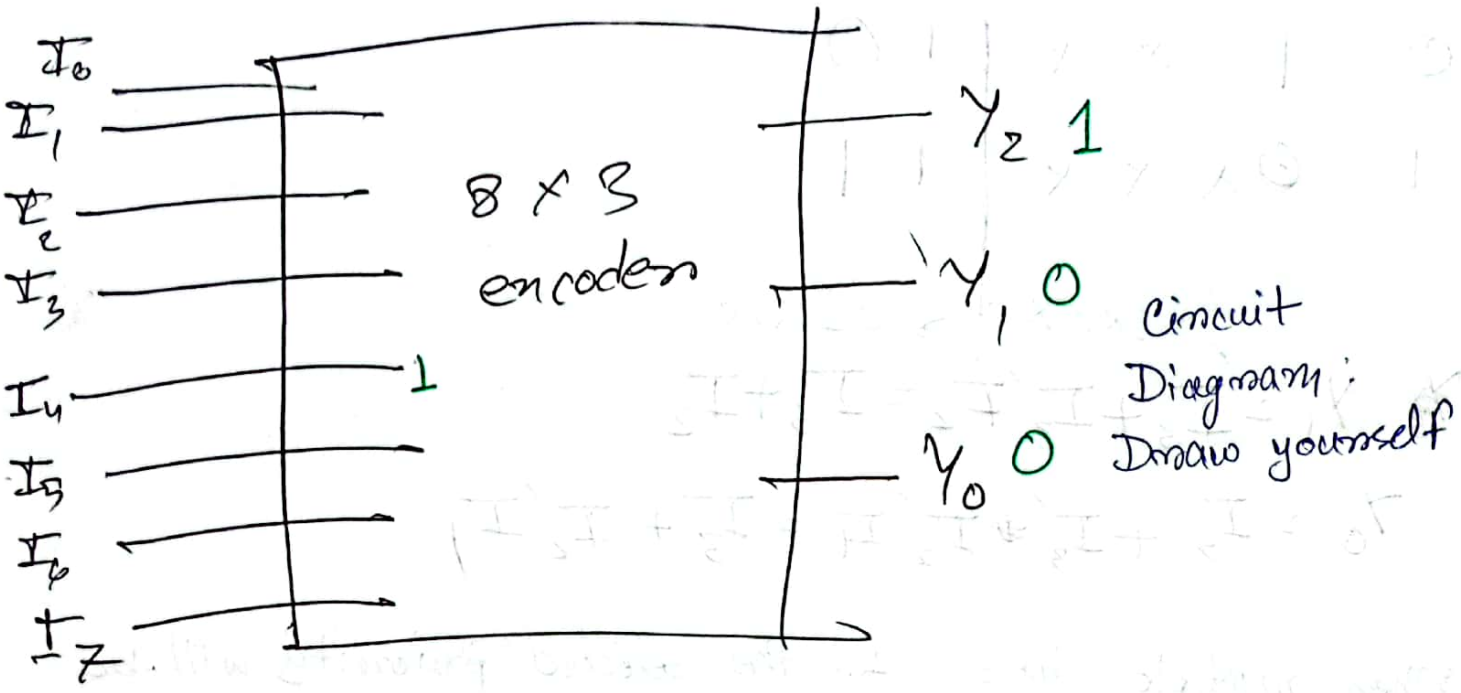
I_7	I_6	I_5	I_4	I_3	I_2	I_2	I_1	I_0	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	0	1	0	0	0			
0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	1	0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	0	1	1	1

Output 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: Draw yourself

Priority Encoder:

- Encoder with priority function. Multiple inputs may be true simultaneously, higher priority inputs gets the precedence.

* 4-input encoders:

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

Function table:

I_3	I_2	I_1	I_0	Y_1	Y_0
0	0	0	0	x	x
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	0	x	x	1	1

→ 1xxx → 01xx

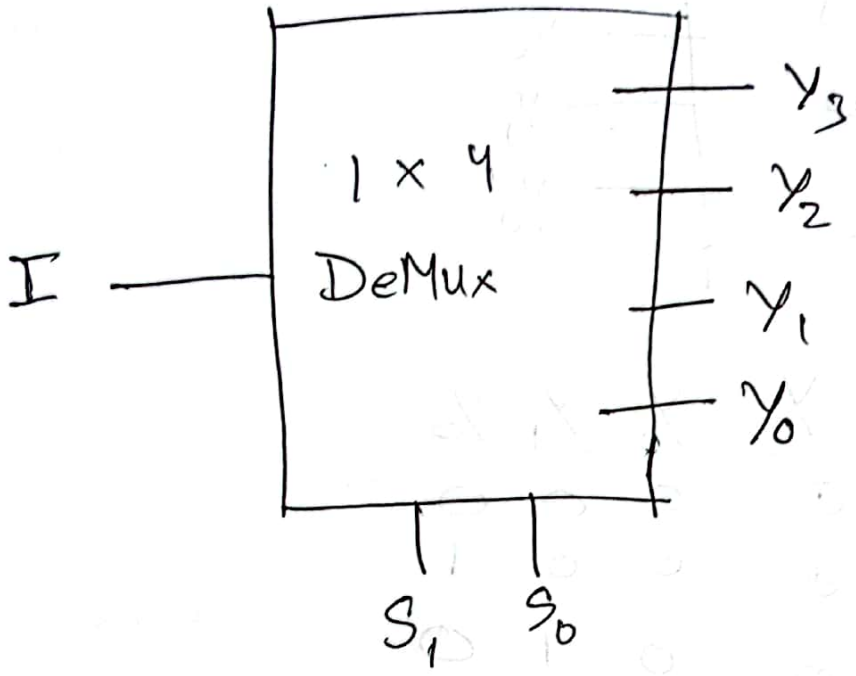
$Y_1 = I_3 + I_3' I_2 = I_3 + I_2$

$Y_0 = I_3 + I_3' I_2' I_1 = \overline{I_3} + \overline{I_2}' I_1$

* When multiple have 1, the selected priority will be selected.

* DeMultiplexer:

- A circuit receives information ~~for~~ from a single and directs it to one of 2^n possible output lines

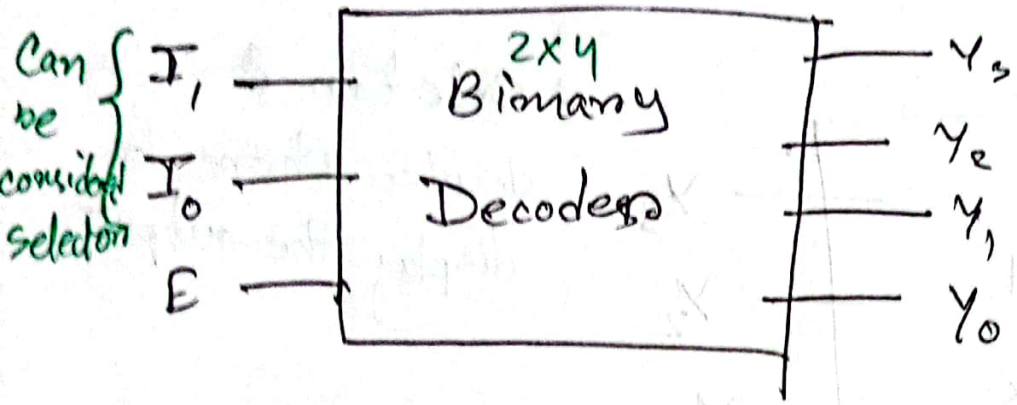


* Selection ② decides where to display the output

Function Table:

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

* A decoder with enable input can function as a demultiplexer



E	I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	0	0	0	0

* Decoder and DeMux have same IC

For final: Easy → Usual bs
Hard → Modify the Easy ones

C-16(w-9) 06/05/24

Quine-McCluskey/Tabulation Method:

Simplify the following Boolean Function by using Tabulation method:

$F(a,b,c,d) = \Sigma(0, 12, 3, 6, 7, 8, 9, 10, 14)$

Soln: $F(a,b,c,d) = \Sigma(0, 1, 2, 5, 6, 7, 8, 9, 10, 14)$

$= \Sigma(0000, 0001, 0010, 0101, 0110, 0111, 1000, 1001, 1010, 1110)$

Step 1: Determination of prime implicants (PIs):

Column I (→ No. of 1's)

	a b c d	
Group 0:		
Group 0: 0	0 0 0 0	✓
Group 1: 1	0 0 0 1	✓
2	0 0 1 0	✓
8	1 0 0 0	✓
Group 2: 5	0 1 0 1	✓
6	0 1 1 0	✓
9	1 0 0 1	✓
10	1 0 1 0	✓
Group 3: 7	0 1 1 1	✓
14	1 1 1 0	✓

Column II (→ 3 bits same with immediate lower group)

	a b c d	
(0,1)	0 0 0 -	✓
(0,2)	0 0 - 0	✓
(0,8)	- 0 0 0	✓
(1,5)	0 - 0 1	*
(1,9)	- 0 0 1	✓
(2,6)	0 - 1 0	✓
(2,10)	- 0 1 0	✓
(8,9)	1 0 0 -	✓
(8,10)	1 0 - 0	✓
(5,7)	0 1 - 1	*
(6,7)	0 1 1 -	*
(6,14)	- 1 1 0	✓
(10,14)	1 - 1 0	✓

Column III (→ same position mismatch + 1 bit)

	a b c d	
(0,1,8,9)	- 0 0 -	*
(0,2,8,10)	- 0 - 0	*
(2,6,10,14)	- - 1 0	*

Step-2: Selection of Prime Implicants:

∴ Prime implicants:

$a'c'd + a'bd + a'bc + b'c' + b'd' + cd'$

(1, 5) (0, 1, 8, 9) (0, 2, 8, 10) (0, 2, 8, 10, 14)

Step-2: Selection of Prime Implicants:

PI's	a b c d	0	1	2	5	6	7	8	10	14
(0, 1, 8, 9)	- 0 0 -	×	×					×	(×)	
(0, 2, 8, 10)	- 0 - 0 -	×		×				×	×	
(2, 6, 10, 14)	- - 1 0			×		×			×	(×)
(1, 5)	0 - 0 1		×		×					
(5, 7)	0 1 - 1				×		×			
(6, 7)	0 1 1 -					×	×			

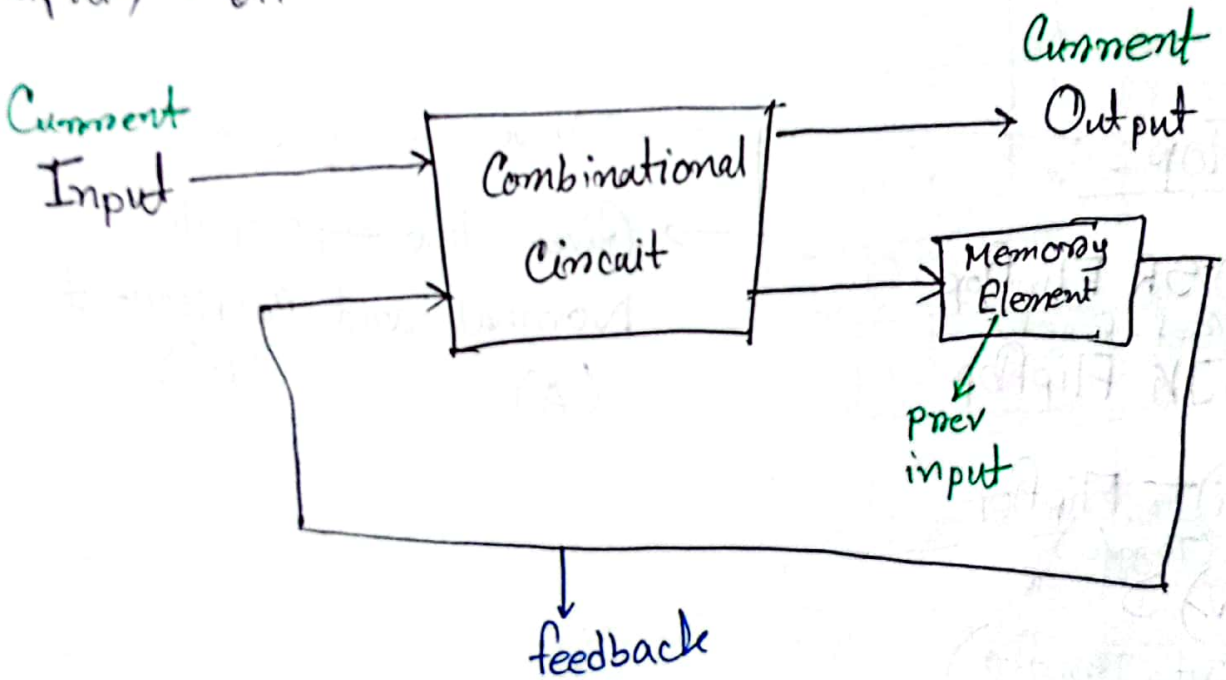
Ans. $b'c' + cd' + a'bd$

C-17(10-10) Ch - 5 : Sequential Circuit 23/06/24

Synchronous Sequential Logic

Current Output = f(previous output, & current input)

Next Output, Q_{t+1} = f(current output Q_t, input)



* Flip Flop (FF) : Memory element.

Sequential Circuit

Asynchronous

Changing sequence of input changes ~~order~~ of output

Synchronous

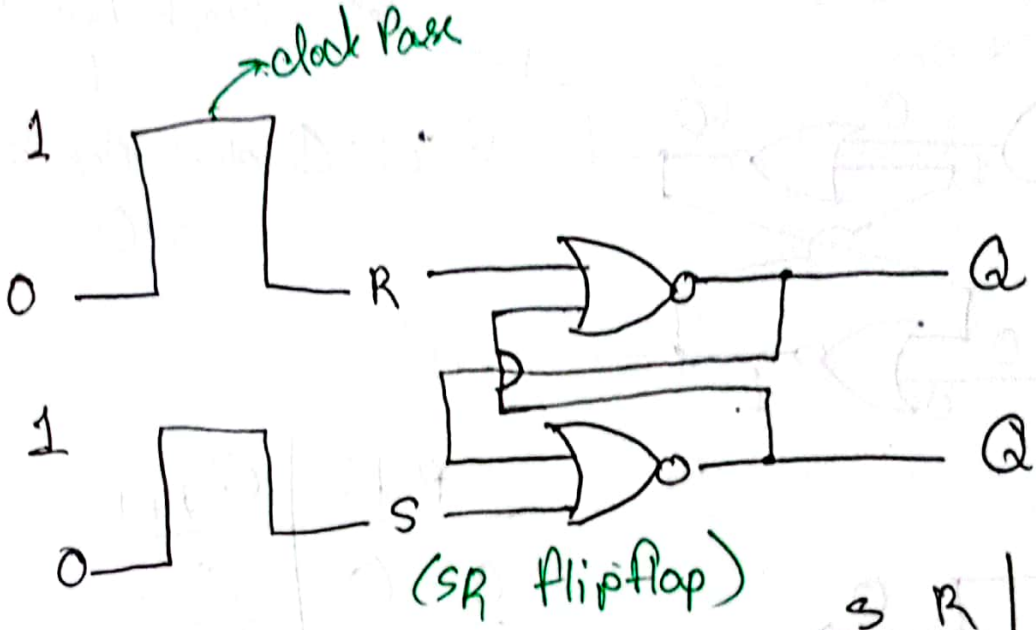
Changing ~~inp~~ time changes output

Flip Flop :-

- Stores only 1 bit
- Gives two ~~inp~~ outputs; Normal and Complement
  (A) (A')

i) SR Flipflop (Set-Reset)
ii) JK Flipflop
iii) T Flipflop (Toggle)
iv) D " (Data Transfer)

* Basic Flip Flop : * Self Study : Construct using NAND



R : Q :: S : Q'

(SR flipflop)

S	R	Q	Q'
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	0	0

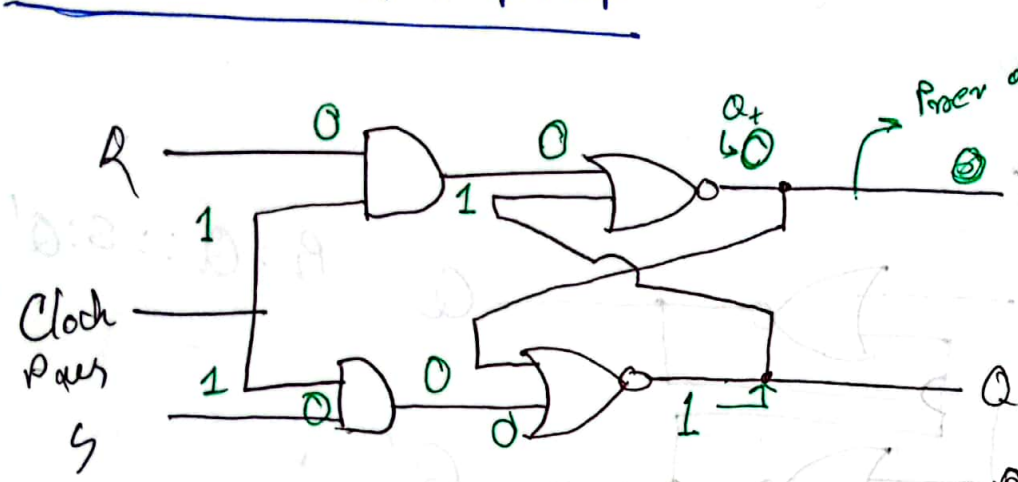
Rare case → This is responsible for the newer outputs

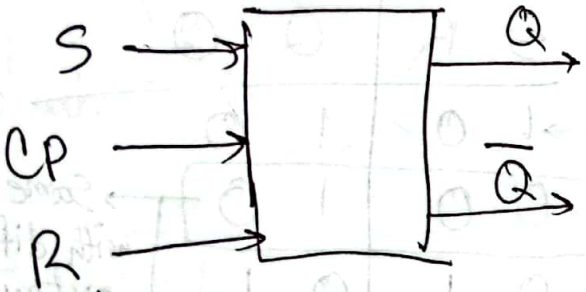
→ Same inp with diff. output ↓ Because of previous input

→ Invalid Condition

↳ For SR using ~~NAND~~, S and R NOR, both cannot be 1 ✌.

i. Clocked SR Flip Flop:





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

Characteristics Table

Q_t	S	R	Q_{t+1}
* 0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	Indeterminate
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	Indeterminate

you get the idea

→ ~~Characteristics~~ Table

$$Q_{t+1} = S + R'Q$$

Excitation Table: (SR) Map

* Using desired output to find the required input

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

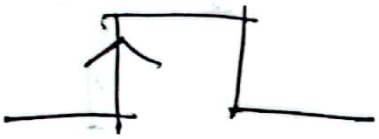
↳ Using Characteristics Table

* Self Study: Memorize all flipflop Excitation Table + the rest of FFs

Triggering Flipflops:

* Triggering Point: When output changes.

$\overrightarrow{CP}$ → positive edge



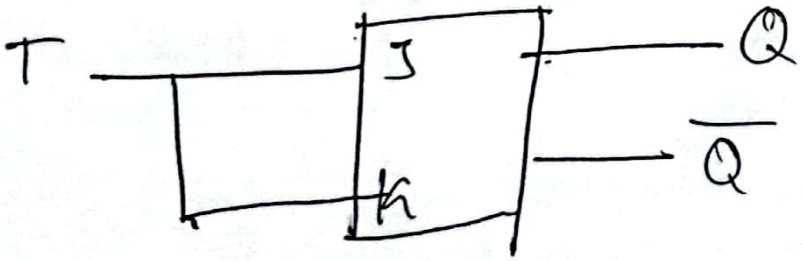
$\overrightarrow{\overline{CP}}$ → negative edge

Conversion:

* JK to T:

JK: $Q_{t+1} = JQ' + K'Q$

T: $Q_{t+1} = TQ' + T'Q = T \oplus Q$



Here,

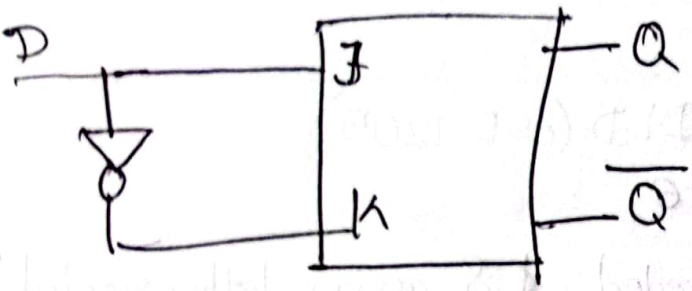
$Q_{t+1} = JQ' + K'Q$

$= TQ' + T'Q$

$= T \oplus Q$

*JK to D:

D : $Q_{t+1} = D$



$Q_{t+1} = JQ' + K'Q$

$= \cancel{JQ' +}$

$= DQ' + (D)'Q$

$= DQ' + DQ$

$= D(Q' + Q)$

$= D$

C-18 (w-10)

27/06/24

Design Procedure of Sequential Ckt:

i) Problem is stated → text/state diagram/state table

ii) Obtain the state table

iii) State reduction → Not required for DLD (CSE 1205)

iv) Assign binary values to each state.

v) Determine the no. of FFs needed & assign letter symbol to each
   - no. of bits = no. of FFs
   - each IC has 2 bits
   - 7476 → JK flipflop

vi) Choose the type of FF to be used.

vii) Derive the excitation map.

viii) Derive the ckt output function & FF input function.

ix) Draw the logic diagram.

Chap - 6

Registers & Counters

Counters: ⟶ keeps a count

A seqntl ckt that goes through a prescribed sequence of states upon the application of input pulses is called a counter.

- Counting occurrence of an event
- Generates timing sequence
- Tracks elaped time between events.

* Binary counter follows binary sequence
  - ↳ n-bit binary counter needs n FFs and counts 0 to $2^n - 1$

3 bit binary counter ⟶ 3 FFs ⟶ 0 – 7 ⟶ 0 → 1 → 2 → 3 → 4 → 5 → 6 → 7

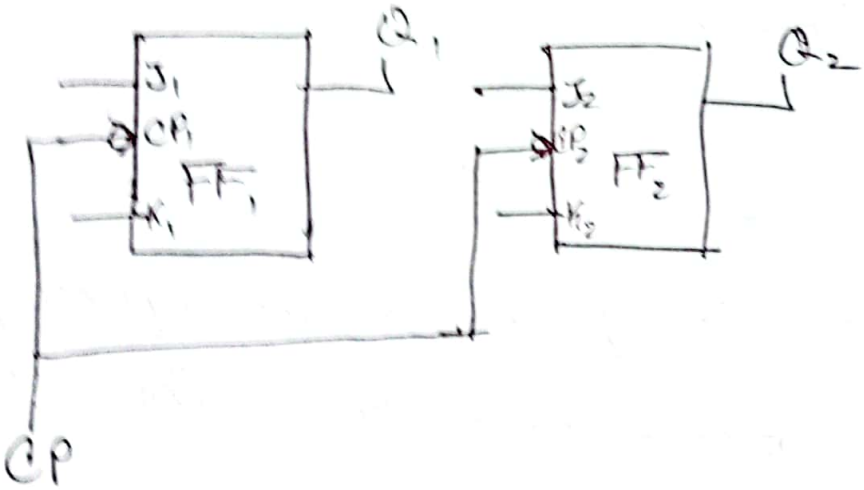
can start from anywhere but must follow pattern.

Counter is of 2 types:

i) Synchronous ⟶ Same clock pulse

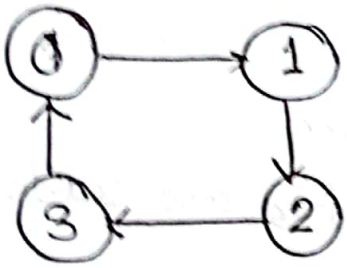
ii) Asynchronous ⟶ different clock pulse.

Design a 2-bit synchronous binary counter.



* $\overline{SD}$ ← 0
  - Sets output to 1
* $\overline{RD}$ ← 0
  - Sets output to 0

Keep them disabled by 5V to get different values



→ Draw using binary or decimal

State Diagram

Present State		Next State	
Q_2	Q_1	Q_2	Q_1
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

→ make table for every UNIQUE combination/state

Excitation Table: → When making input eq's ALWAYS consider present state

Present State		Next State		FF			
Q_2	Q_1	Q_2	Q_1	J_2	K_2	J_1	K_1
0	0	0	1	0	×	1	×
0	1	1	0	1	×	×	1
1	0	1	1	×	0	1	×
1	1	0	0	×	1	×	1

compare (present state vs next state)

compare (Q_2 vs Q_1)

FF2

For J_2,

Q_2 \ Q_1Q_1'	Q_1	Q_1'	Q_1
Q_2'	0	1	
Q_1	×	×	

$\therefore J_2 = Q_1$

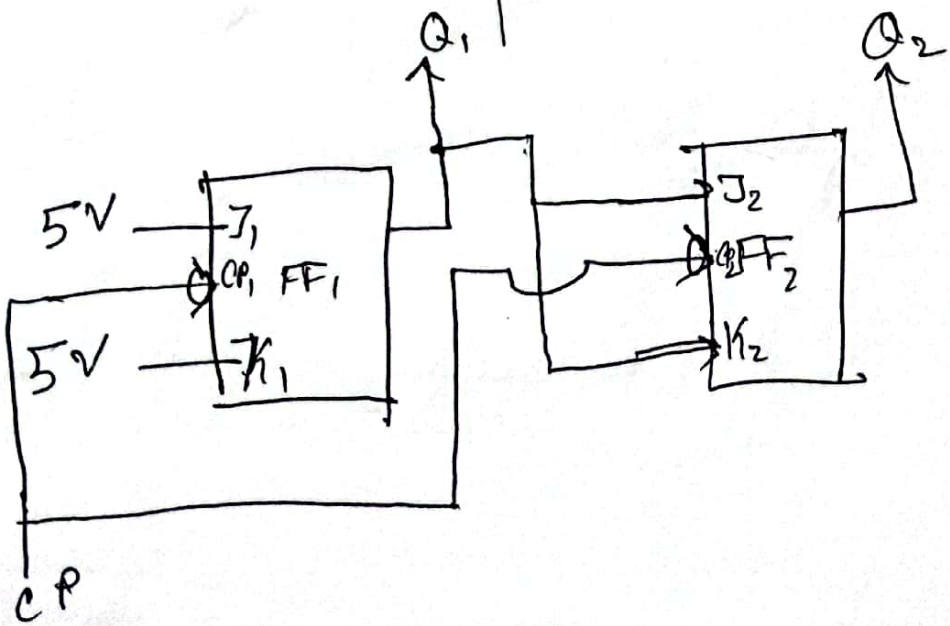
For K_2,

Q_2 \ Q_1Q_1'	Q_1	Q_1'	Q_1
Q_2'	×	×	
Q_2		1	

$\therefore K_2 = Q_1$

For J_1

Q_2 \ Q_1Q_1'	Q_1	Q_1'	Q_1
Q_2'	1	×	
Q_2	1	×	



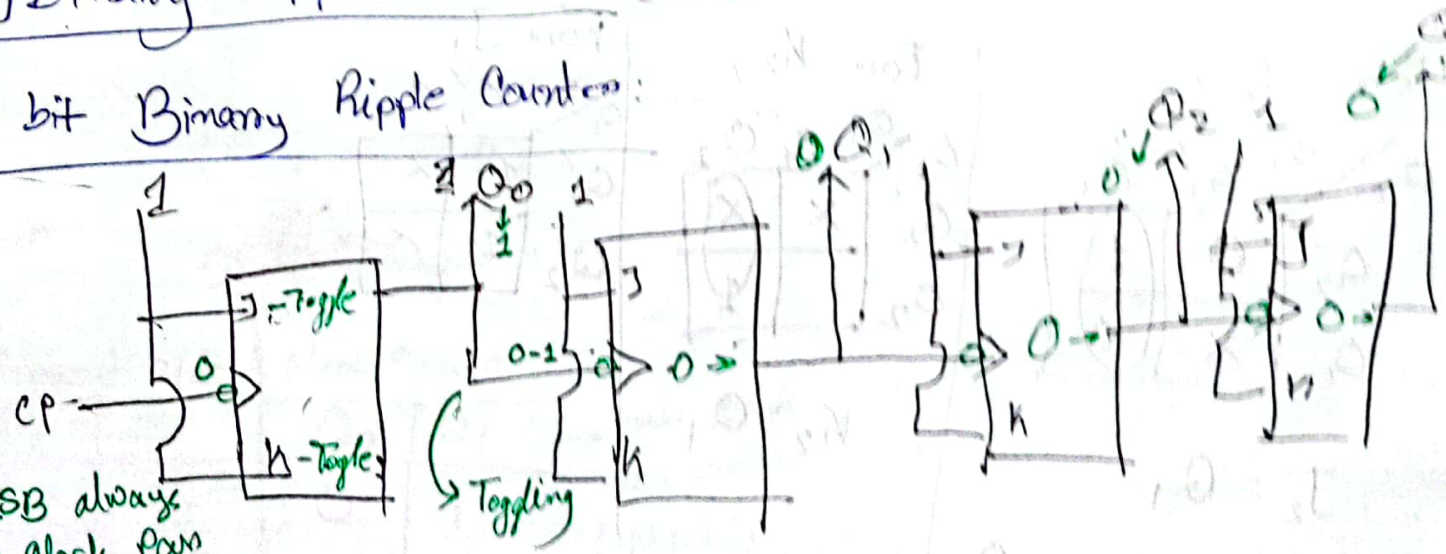
Asynchronous Counters

- Different Clock Pulse
  - Not necessarily by the same
  - Uses different method to solve
  - Working Procedure is necessary

* Also known as Ripple Counter
* Different Clock Pulse in FFs
→ Slower

Binary Ripple Counter : (Slide is incomplete)

4 bit Binary Ripple Counter:



* LSB always gets clock-pulse

J.K Timing

J	K	Q_+
0	0	Q_{prev}
0	1	0
1	0	1
1	1	$\bar{Q}_0$

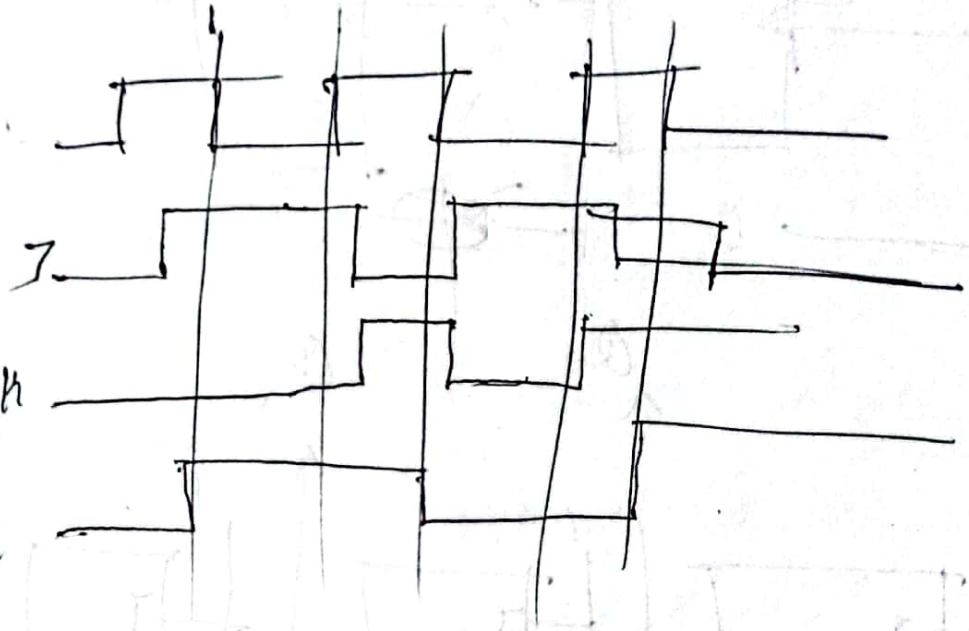
* Timing Diagram: *Use scale

→ Draw CP

→ Identify where changes

→ change accordingly

→ Check J and K

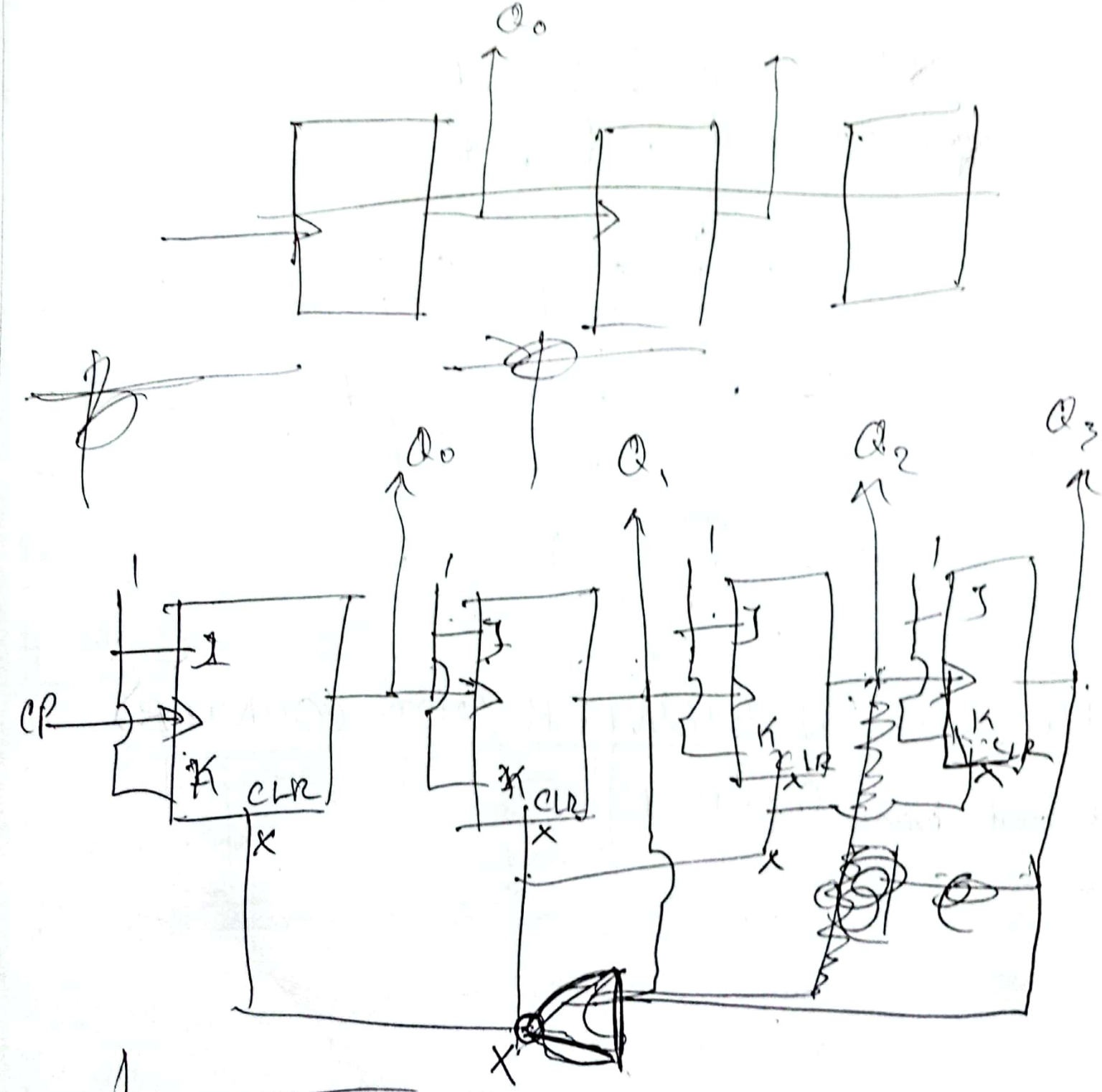


*MODULO N COUNTER (MOD COUNTER)

* Counts from 0 to N-1

* MOD - 10 counter

$Q_3 Q_2 Q_1 Q_0$



1000
1010
1011
1100

* Self Study: Do the decade counter maths

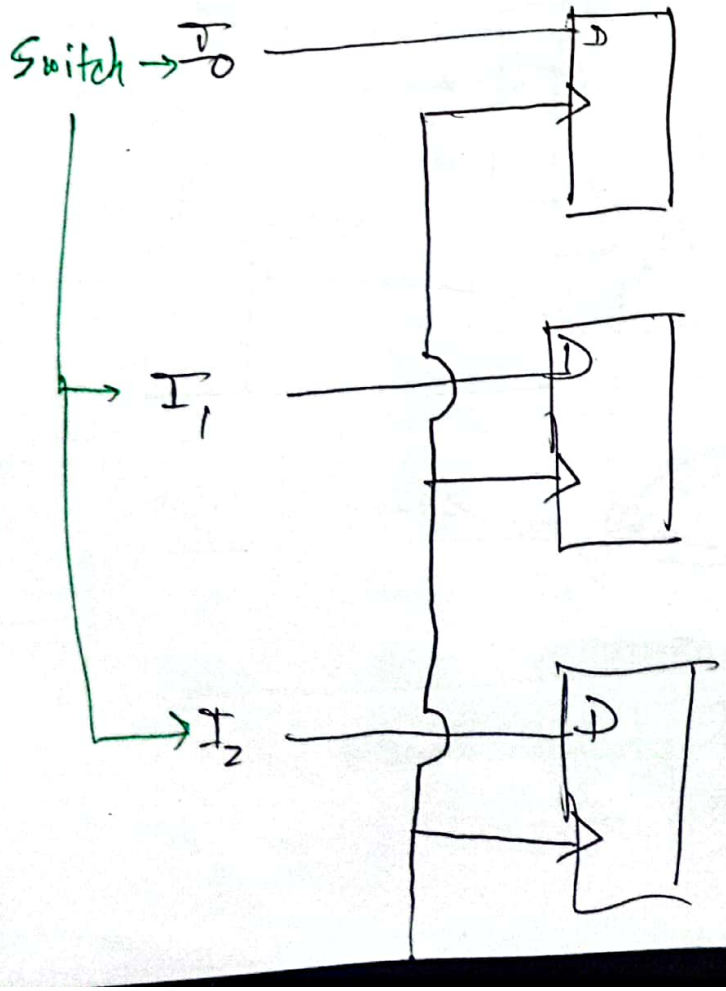
C-20(10-12) 07/07/24

Register → will be in quiz-4

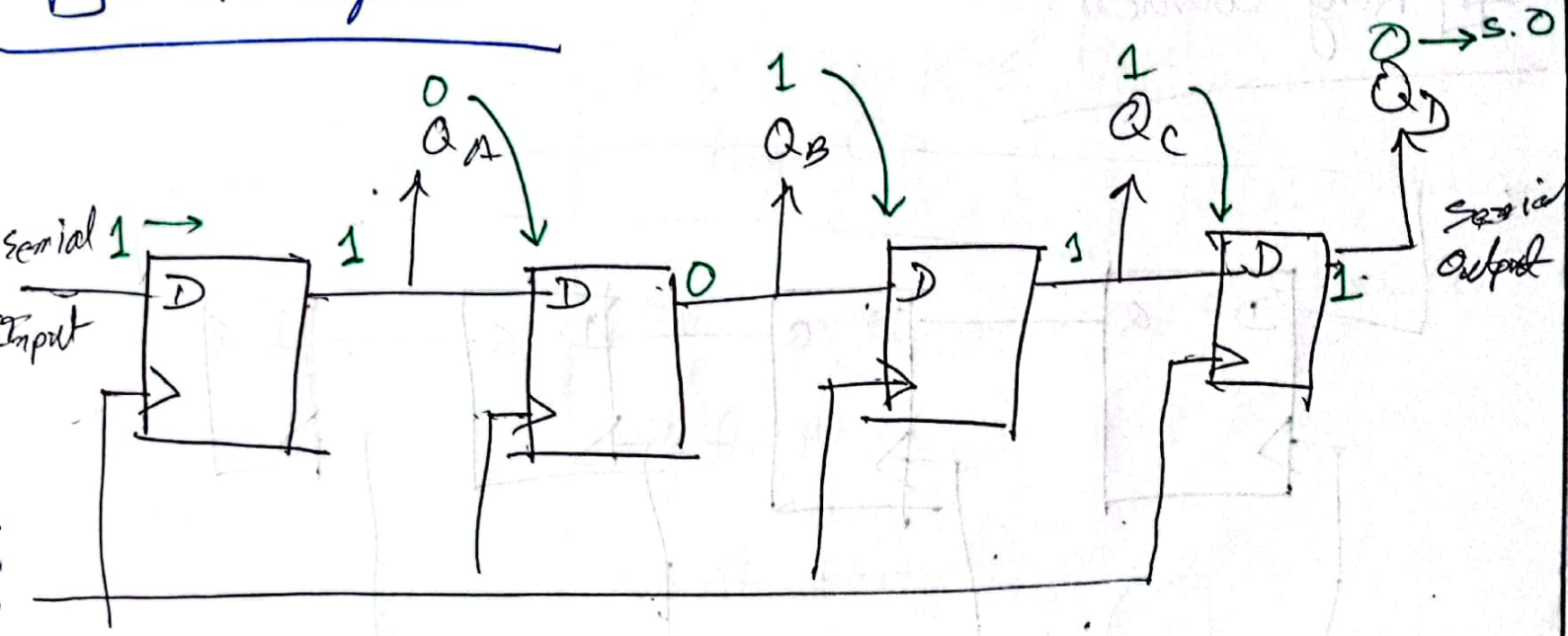
* Stores data
* Register is made using a group of flip flops

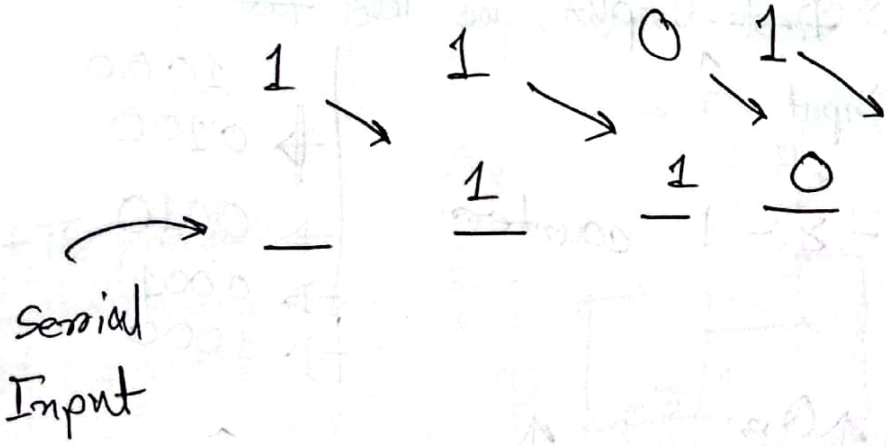
Register = Flip-Flop + Combinational Gate
(essential) (optional)

* N-bit registers require N flip flops
* Usually uses D-flip-flop



Shift Register :



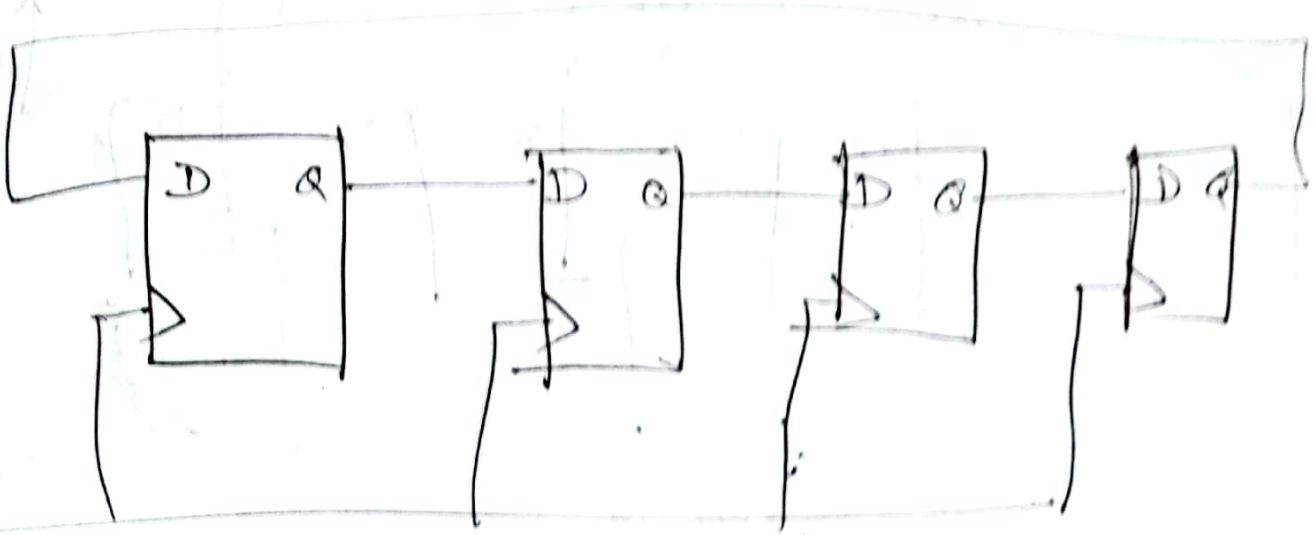


* Right shifting $x = \left\lfloor \frac{x}{2} \right\rfloor$

+ Left shifting $x = 2x$

* Serial Input can be whatever we want

Ring Counter:

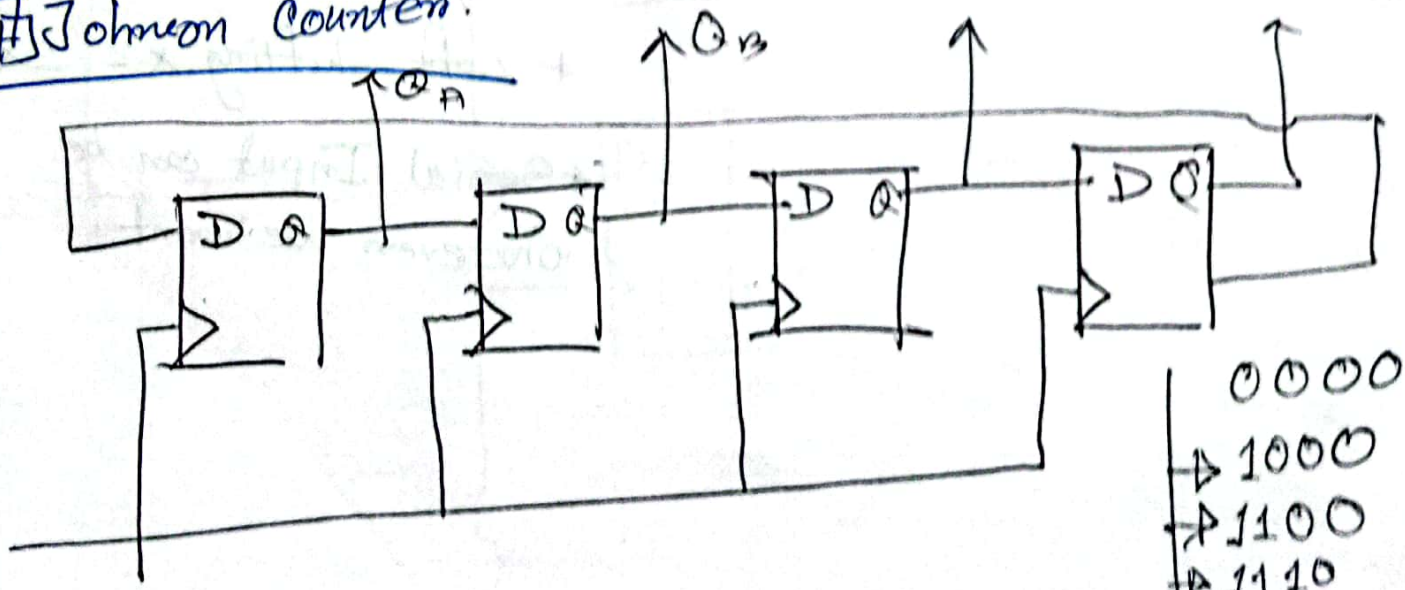


* Instead of switch Input as first serial input, we use last out serial output as serial input.

* Also known as 8-4-2-1 counter

1000
0100
0010
0001
1000

Johnson Counter:



0000
1000
1100
1110
1111
0111
0011
0001

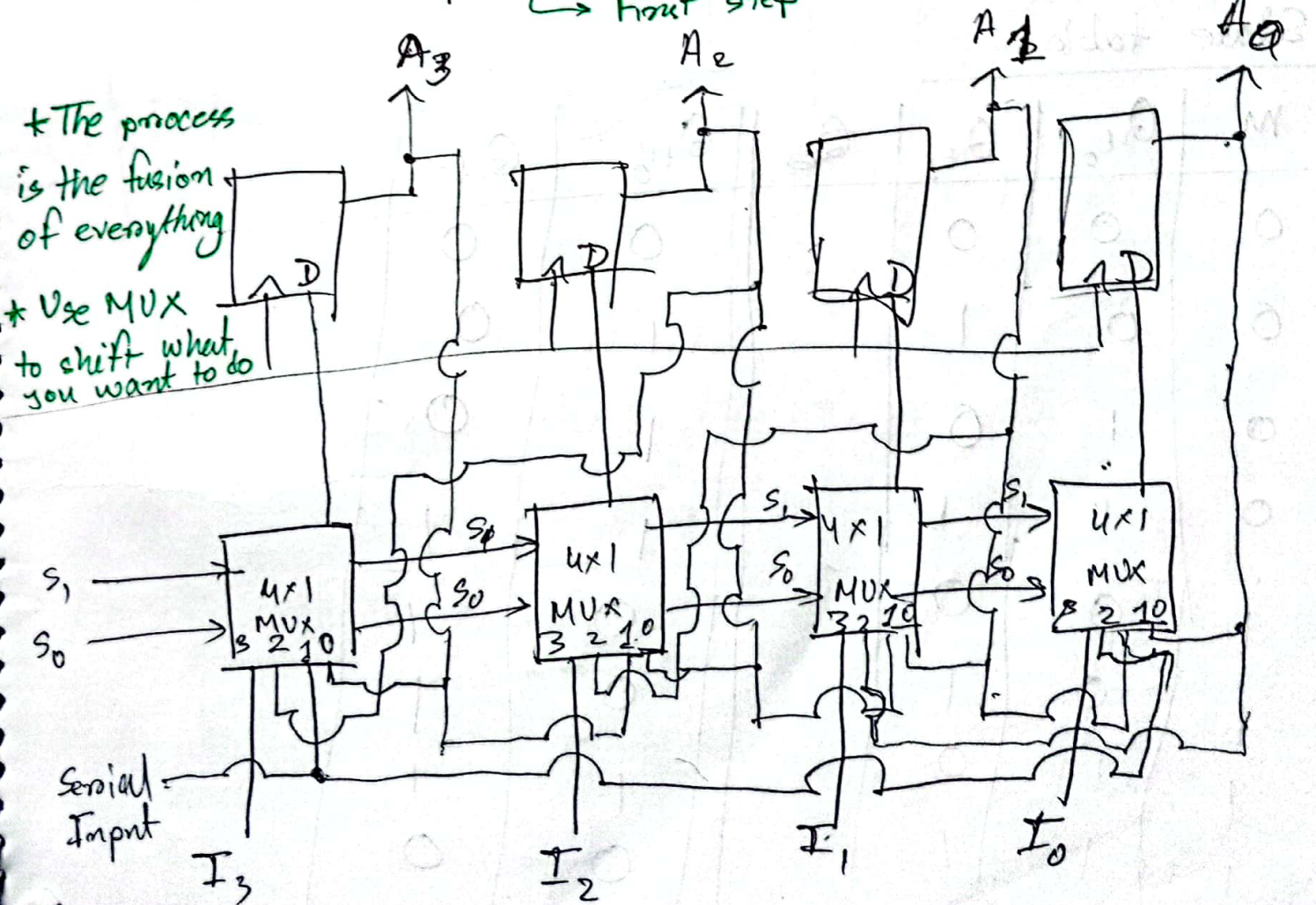
Universal Shift Register

Table not fixed

S_1	S_0	Reg Op
0	0	No change → Gives old result
0	1	Shift Right
1	0	Shift Left
1	1	Parallel Load → Stores new value

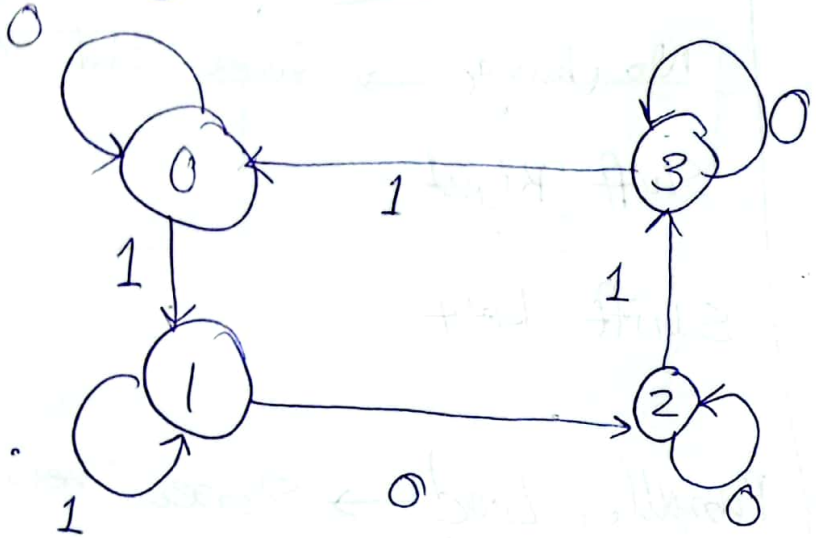
↳ First step in Lab

* The process is the fusion of everything
* Use MUX to shift what you want to do



C-21 (W-12) 10/07/24

Design a synchronous sequential circuit for the following problem (using JK flip-flop)



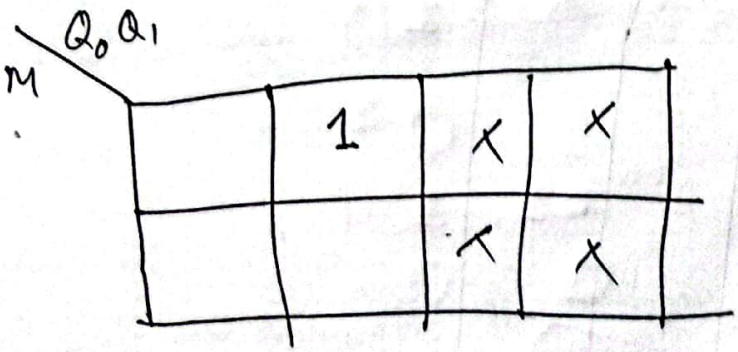
State table:

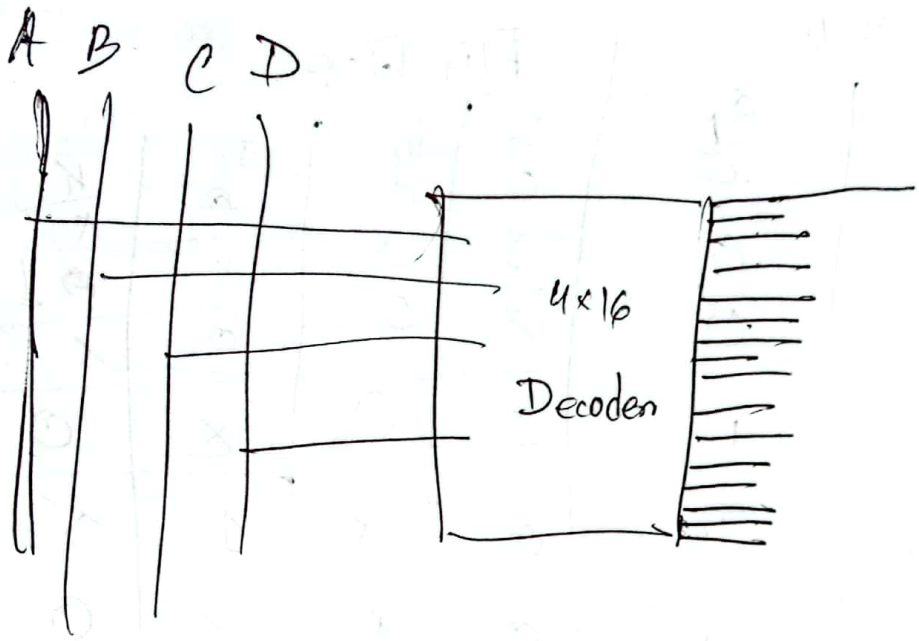
M	Q_{t_0}	Q_{t_1}	Q_{t+1_0}	Q_{t+1_1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	1
1	1	1	0	0

Excitation Table :

Guild	Present State		Next State		Flip Flop			
M	Q_0	Q_1	Q_0	Q_1	J_0	K_0	J_1	K_1
0	0	0	0	0	0	x	0	x
0	0	1	1	0	1	x	x	1
0	1	0	1	0	x	0	0	x
0	1	1	1	1	x	0	x	0
1	0	0	0	1	0	x	1	x
1	0	1	0	1	0	x	x	0
1	1	0	1	1	x	0	1	x
1	1	1	0	0	x	1	x	1

$\therefore J_0$ $\therefore K_0$



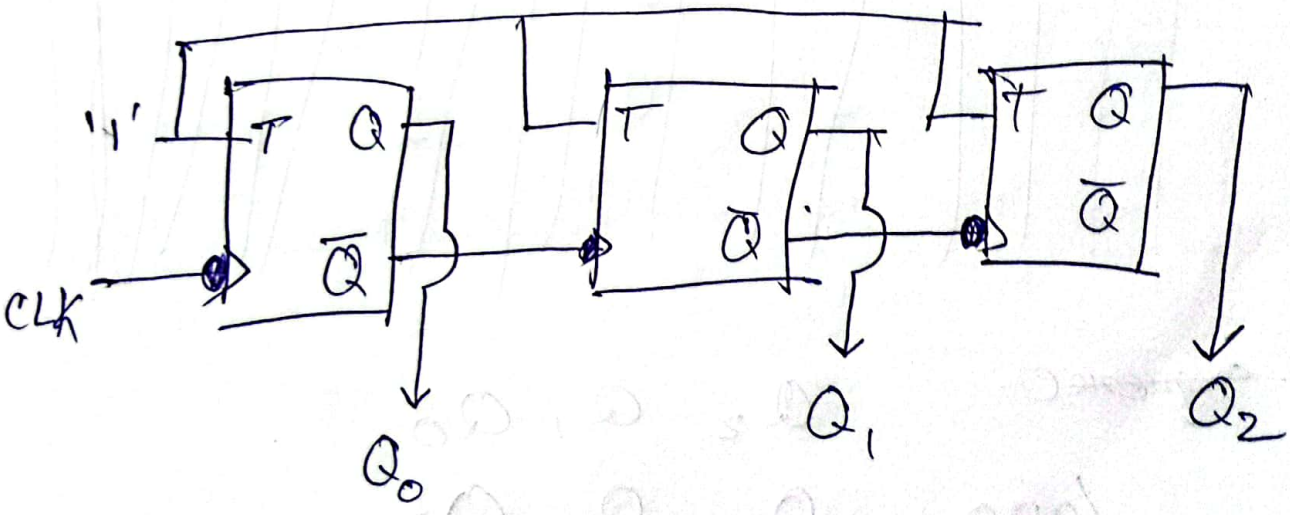


In				Out			
A	B	C	D	P	Q	R	S
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	0	1	0	0	1	0	0
0	1	0	0	0	1		
0	1	0	1	0			
0	1	1	0	0			
0	1	1	1	0			
0	1	0	0	0			
1	0	0	0	0			
1	0	0	1	0			
1	0	1	0	0			

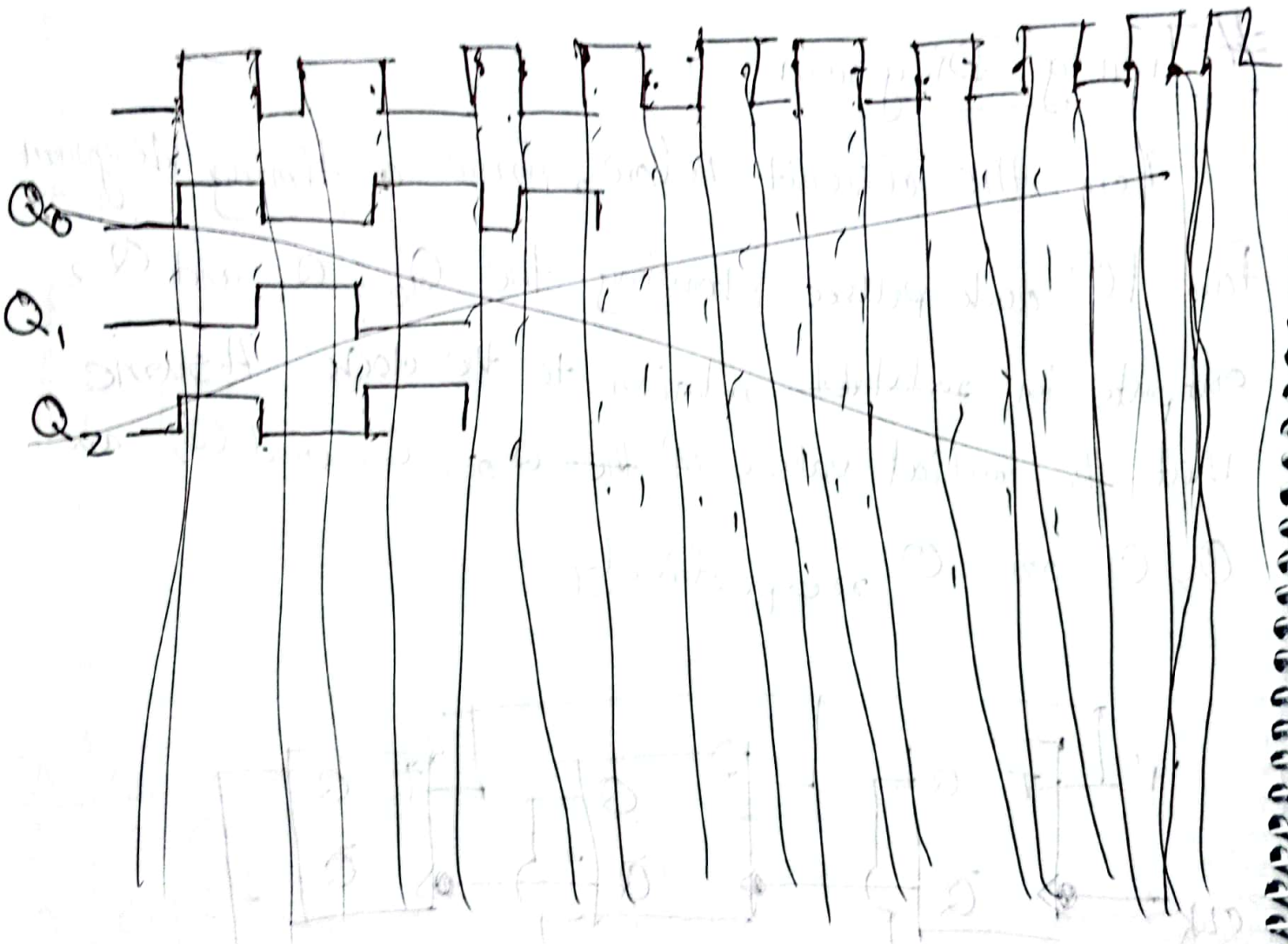
C-23 (W-13) 19/09/24

#. Timing Diagram

For the circuit below, draw a timing diagram for 10 clock pulses showing the Q_0, Q_1 and Q_2 outputs in ~~relatabl~~ relation to the clock. Assume that the initial values of ~~the~~ Q_0, Q_1 and Q_2 are 0, 0 and 0 respectively.

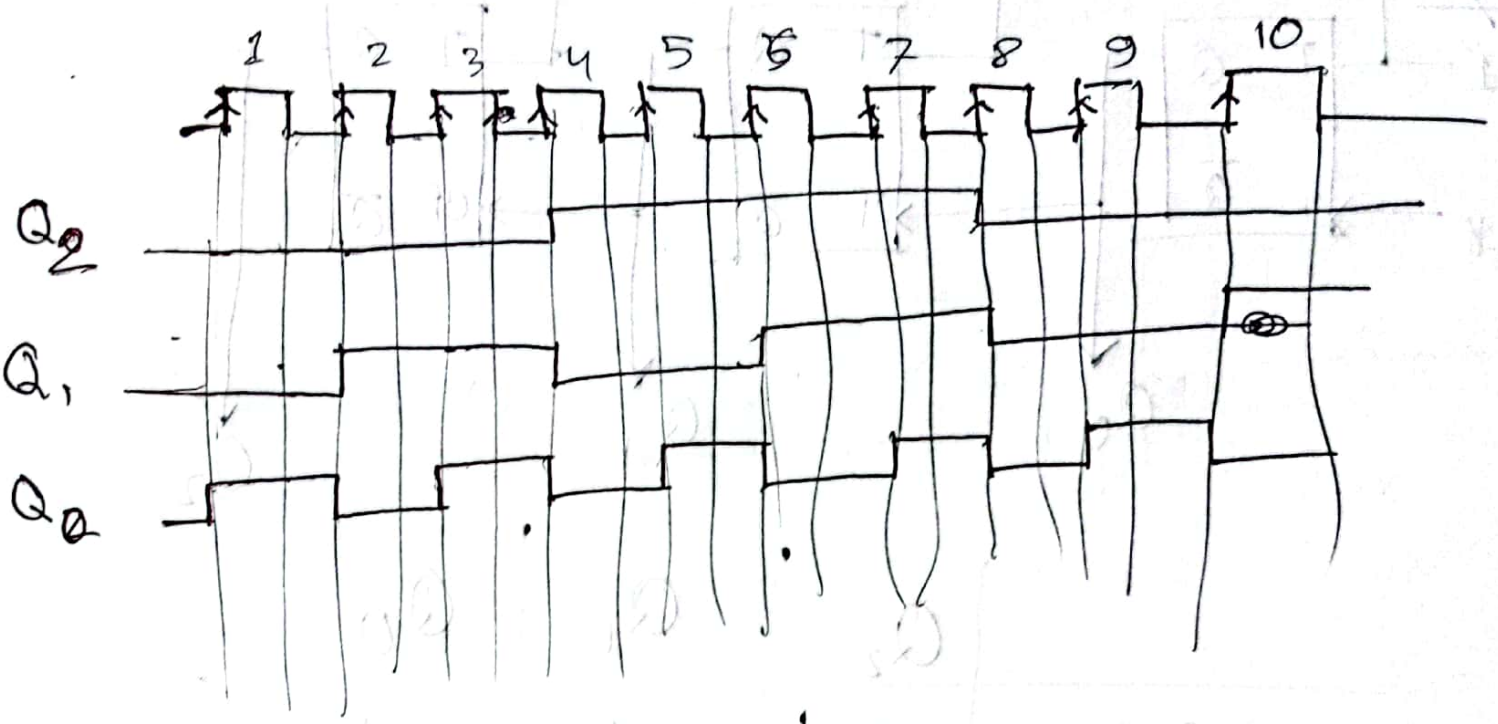


(i) Notice the CLK (negative/positive)
(ii) " " FF
(iii) " " MSB and LSB
(iv) Find the sequence first draw later.

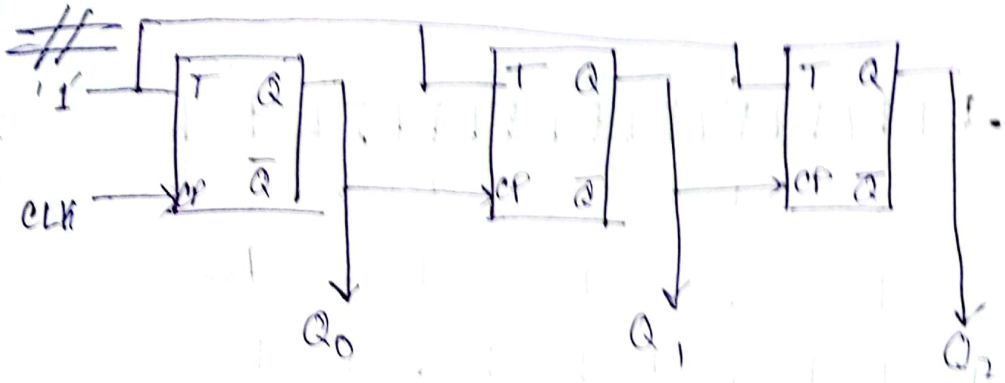


Output sequence

	Q_2	Q_1	Q_0
CLK Initial /CP.0 :	0	0	0
CP 1	0	0	1
CP 2	0	1	0
CP 3	0	1	1
CP 4	1	0	0
CP 5	1	0	1
CP 6	1	1	0
CP 7	1	1	1

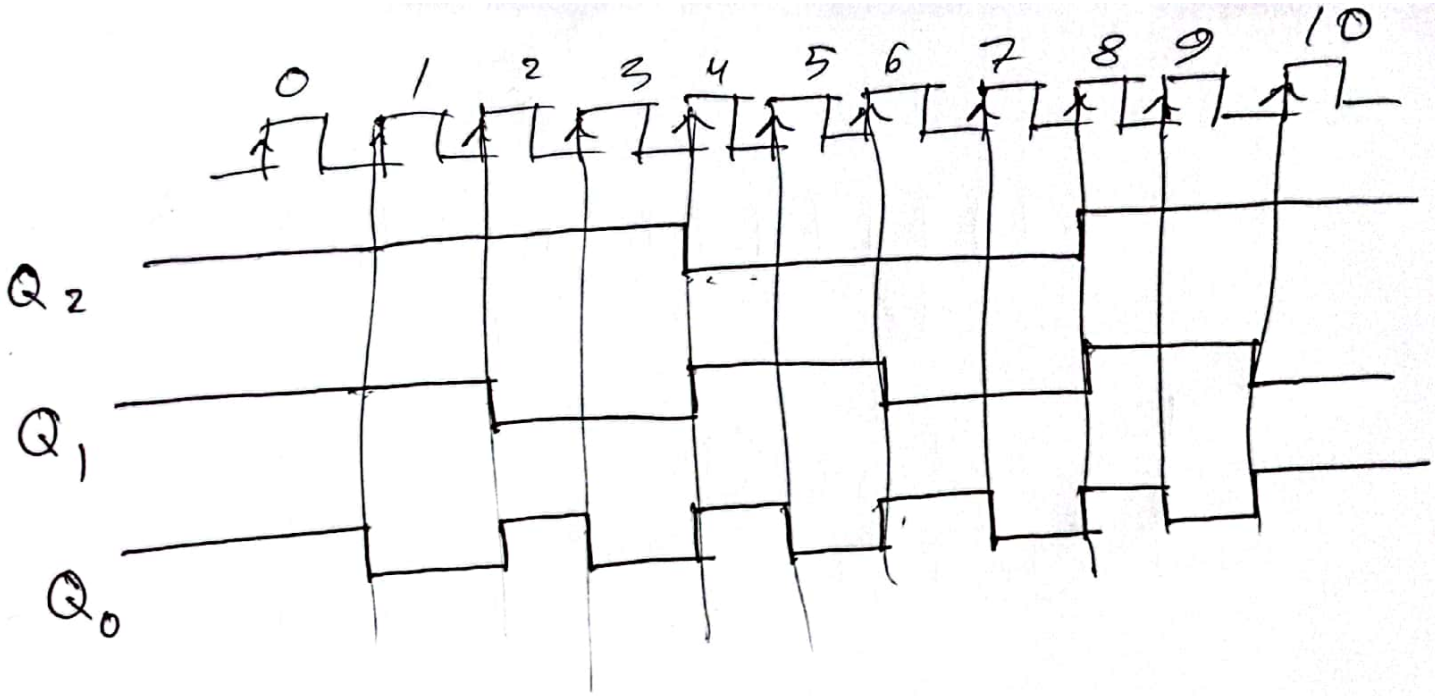


	Q_2	Q_1	Q_0
cp 8	0	0	0
cp 9	0	0	1
cp 10	0	1	0



	Q_2	Q_1	Q_0
C.P 0	1	1	1
" 1	1	1	0
" 2	1	0	1
" 3	1	0	0
" 4	0	1	1
" 5	0	1	0
" 6	0	0	1
" 7	0	0	0
" 8	1	1	1
" 9	1	1	0
" 10	1	0	1

Q_2
Q_1
Q_0


0
1
2
3
4
5
6
7
8
9
10
Q_2
Q_1
Q_0