

Mark Distribution

2106
(Syeeda Shabnam Miss)
(FA01/N)

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

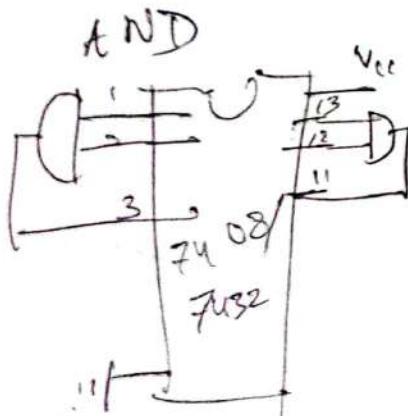
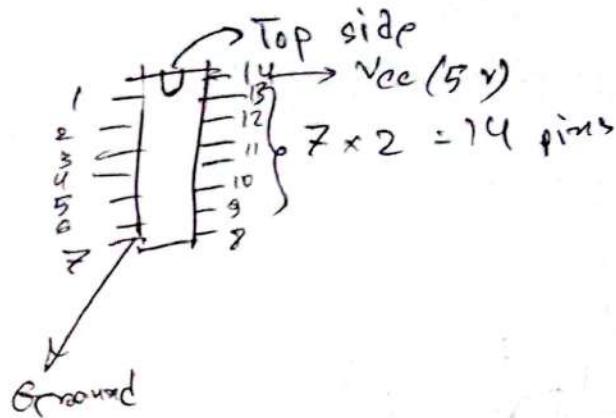
Microprocessors Data handbook

- BCB publication

 7408 - AND

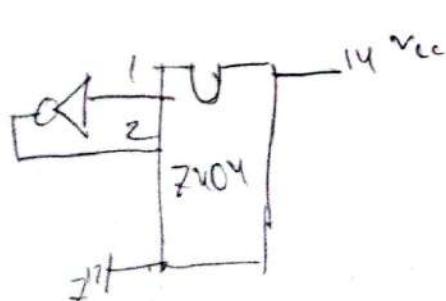
7432 - OR

7404 - NOT



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

+ Four < 4 , you need
1 IC



		NOT
A		
0		1
1		0

 AND

 OR

 NOT

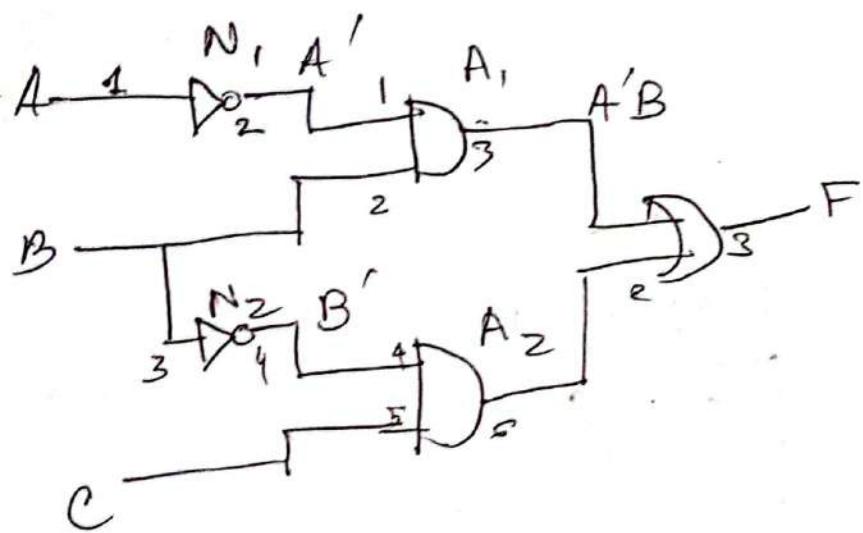
 XOR

 XNOR

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 & rev)

theory first → have later

Require: NOT gate (\neg icon)

i) N_1 — NOT gate (code)
Quantity — 1

ii) Name → Create Name (code)

Quantity —

(Syeda Shabnam Miss)

Syeda

"Easiest one" ~ Miss

"Either a lot of marks or none"

"Conceptual course"

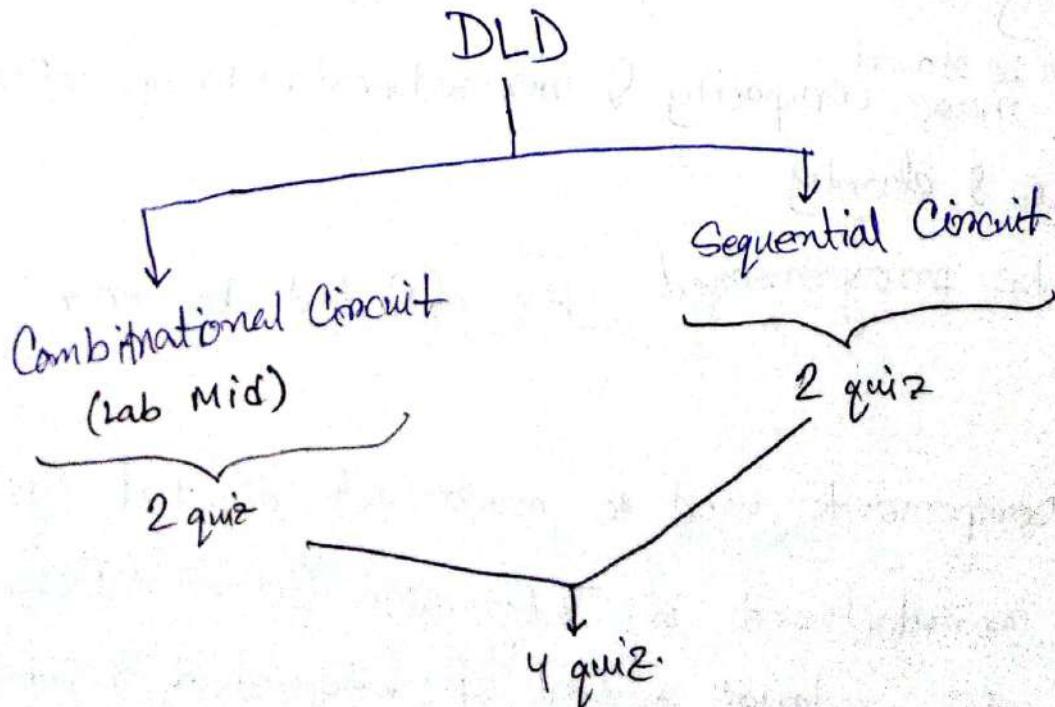
"Fully hardware related course"

"DLD is a core course"

"Book is essential. Slides are complementary"

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

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



* 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:

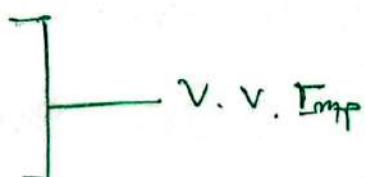
These states are presented by 2 different voltage levels;
A HIGH and Low LOW.

^{Binary}
Number System

* In digital systems, combinations of 2 states called codes

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

* HIGH = 1 = ON
LOW = 0 = OFF



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

* 0 - 15 in binary is ESSENTIAL

* Self Study (Important for i) Lab tests II ii) Exams)

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

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 ~~are~~ signals are 2 state, either low or high voltages. The basic logic ckt's are

OR, AND and NOT gates.

~~* Basic because all gates can be derived~~ 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:

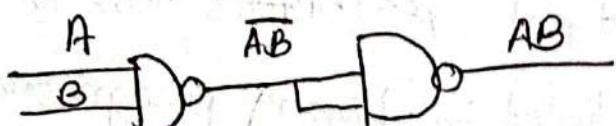
The only one type gates, either NAND or NOR are sufficient for the realization of any logical expression. For this, NAND and NOR gates are known as universal gates.

Realization of basic logic operations using NAND gate:

NOT operation:

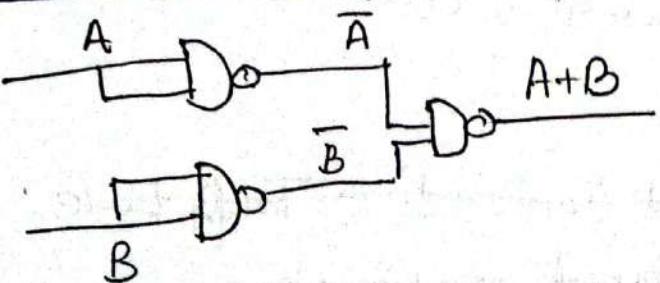


AND operation:



AND operation:

OR operation:



→ Realization of basic logic operations using NOR: self study

some formation

juts AND and OR are flipped

* NOR : Self study

* XOR for more than two inputs:

* Make operation ~~in~~ using
NAND/NOR → just draw
some figure

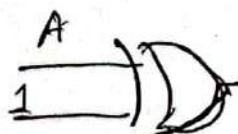
- If total no. of '1' is odd,
output will be 1

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

opposite for X-NOR

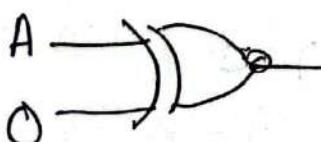
* Realization of NOT operation:

→ Using X-OR gate:



$A \oplus B = \bar{A}B + A\bar{B} = \bar{A} \cdot 1 + 0 \cdot A = \bar{A}$; [the 1 makes it either odd or even]

→ Using X-NOR gate:



$\bar{A} \oplus B = A \odot B = A'B' + AB = \bar{A} \cdot 0 \cdot 1 + \bar{A} \cdot 0 = \bar{A}$; [the 0 makes it odd or even]

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

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

* See the proofs from book

* Rules of Boolean Algebra:

1. $A+0 = A$

3. $A \cdot 0 = 0$

5. $A+A = A$

2. $A+1 = 1$

4. $A \cdot 1 = A$

6. $A+\overline{A} = 1$

7. $A \cdot A = A$

9. $\overline{\overline{A}} = A$

11. $A+\overline{A}B = A+B$

8. $A \cdot \overline{A} = 0$

10. $A+AB = A$

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

* See proofs

* Dual: Operators (AND/OR) & fixed values (1/0) will be changed but variables and NOT operator will ^{not} be changed AND and OR will change into each other. For example:

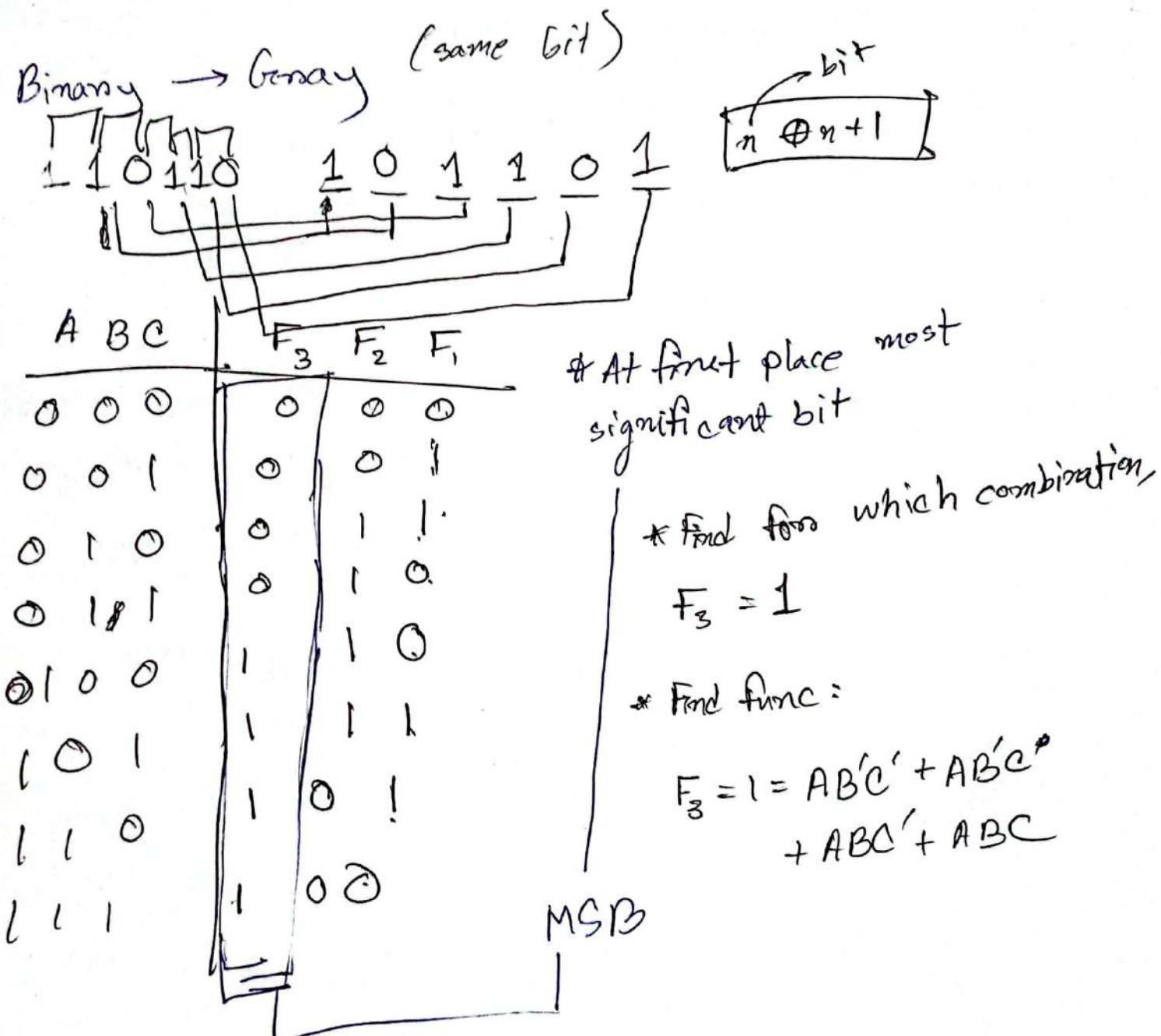
$$A+1 = 1$$

$$A \cdot 0 = 0$$

* Duality principle: + 2 mark SQ

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

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

Gray Code:

Gray \rightarrow Binary

$$\begin{array}{c} \text{Gray} \\ \begin{array}{r} 110110 \\ \hline 1 \end{array} \end{array} \rightarrow \begin{array}{c} \underline{1} \quad \underline{0} \quad \underline{1} \quad \underline{1} \quad \underline{0} \quad \underline{0} \end{array}$$

C-4 (w-3)

27/04/2024

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

Sol'n: 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 $[AB(C + BD) + \bar{A}\bar{B}]C$

Sol'n: $[AB(C + BD) + \bar{A}\bar{B}]C$
= $(ABC + A\bar{B}BD + \bar{A}\bar{B})C$; [distributive law]
= $(ABC + A \cdot 0 \cdot D + \bar{A}\bar{B})C$; [$\because BB = 0$]
= $(ABC + 0 + \bar{A}\bar{B})C$; [$\because x \cdot Q = 0$]
= $(ABC + \bar{A}\bar{B})C$

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

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

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

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

$$= \bar{B}C$$

Q1 H.W: Simplify:

$$AB + A(B+C) + B(B+C) \text{ using Boolean Algebra}$$

Q2 Boolean Functions:

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

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

↳ output will be either 1 or 0

CSE 2105

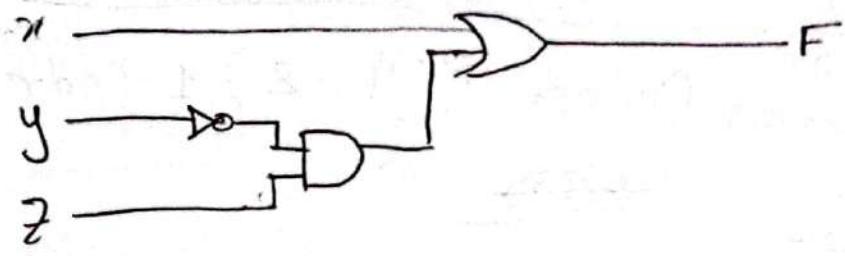
Quiz #1 (Sec: A)

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 $F = x + y^z$

- if $x = 1$,
 $F = 1$

- $y^z = 1$
 $\therefore z = 1 \& y = 0$
 $F = 1$

Lecture 6: Binary codes for the decimal digits

* Main Used in Encryption Coded Decimal on 8, 4, 2, 1 code
 BCD (Binary)

<u>Decimal</u>	<u>Binary</u>	<u>BCD</u>
0	0000	0000
1	0001	0001
:	:	:
9	1001	1001
:	:	:
12	1100	00010000
15	1111	00010011
(279) ₁₀	<u>0010 0111 0100 1</u>	<u>1 2 3</u> 1001 0001 0011 $(123)_{10} = (1001\ 0001\ 0011)_{BCD}$

Excess - 3:

<u>Decimal</u>	<u>Binary</u>	<u>BCD</u>
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

* [same as BCD
each bit has corresponding
weight]

0

1

5

5

0	0	0	0
0	0	0	1

0	1	0	1
0	1	0	1

1	0	1	1
1	0	1	1

* [maximum can be 9]

8, 4, -2, -1

Decimal

8	4	-2	-1
0	1	1	0

* [yeah pretty much
same]

* [maximum can be 12]

② 2

6

Coding conversion (Example):

Decimal

2	4	2	1
0	0	0	0

0

1

8	4	-2	-1
0	0	0	0

0	1	1	1
0	1	1	1

- * Convert to Decimal
- * Convert to the other

Binary to Gray Conversion:

Binary: 10110 → Either 1⊕0⊕1⊕1⊕0
 Gray: 11101 → On 1+0+1+1+0

Gray to Binary Conversion:

Gray: 11011
 Binary: 10010

Bin: 1101101
 Gray: 10100101
 Binary: 1010110
 Bin: 11001010

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, $P(\text{odd}) = 0$
 " " " even " , $P(\text{even}) = 1$

* Parity checker
 * Parity generator

> CHECKS parity
 $111 \rightarrow 1$, because it has odd number of 1's

MAKES parity

$111 \rightarrow 0$ is odd because it has odd number of 1's

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

Determine the single error, code for 101110110

Soln: The no. of data bits $n = 9$.
So, the required no. of parity bits (p) can be obtained from the following formula $2^p \geq n + p + 1$

if $p = 1$, $2^1 \geq 9 + 1 + 1$ (False)

if $p = 2$, $2^2 \geq 9 + 2 + 1$ (False)

if $p = 3$, $2^3 \geq 9 + 3 + 1$ (False)

if $p = 4$, $2^4 \geq 9 + 4 + 1$ (True)

if $p = 4$, we have to send total $n + p = 9 + 4 = 13$.

So, $p = 4$ and we have to send 13 bit message

Original message: 101110110

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

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 P₁, P₂, P₃, P₄, we have to

XOR the binary values of bit positions holding 1's

$$\begin{array}{l} 0011 \\ 0110 \\ 0111 \\ 1001 \\ 0100 \\ 0111 \\ (0)100 \end{array} = 1100$$

P₄ P₃ P₂ P₁

so, the sending message will be 001101110110
(cont)

Cont.

Receiving End :

Case 1: If there is no error in transmitting, the receiver will get 001101110110.
to verify

~~→ P will be 1~~

XOR positions with 1

$$\begin{array}{r} 00\ 11\ (5) \\ 0\ 100\ (6) \\ 0\ 11\ 0\ (7) \\ \hline \cancel{1}\ 000\ \cancel{(8)}\ 1000\ (8) \\ 1\ 001\ \cancel{(9)} \\ 10\ 11\ \cancel{(10)} \\ \hline (\text{XOR})\ 1100\ (12) \\ \hline 0\ 000 \end{array}$$

So, there is no error in the received message

Case (2): Say, there is an error in the 6th position
transmitting the receiver will get: 0011001110110.
to verify:

$$\begin{array}{r} 00\ 11 \\ 0\ 1\ 00 \\ 0\ 1\ 11 \\ 1\ 0\ 00 \\ 1\ 0\ 01 \\ 1\ 0\ \cancel{0}\cancel{0} \\ \hline (\text{XOR}) & 1\ 1\ 00 \\ \hline & 0\ 1\ 10 \end{array}$$

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

$$00110\underline{1}1110110$$

Chap - 2

Lecture 7: Standard forms of Boolean

Expressions

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

i) The Sum of Products (SOP): $\rightarrow \text{OR}$

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

$\rightarrow \text{AND}$

ii) The Product 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 minterms or product of maxterms.

- We can derive truth table from standard form

Input	Output	Minterms	Max terms
$x \ y \ z$	F	$x = 0 \ 1$ $y = 1 \ 0$ $z = 1 \ 1$	$x = 0$ $y = 0$ $z = 2$
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	$x'y.z (m_3)$	$(x+y'+z') (M_3)$
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, $F = m_1 + m_3 + m_4 + m_5$
 $\Rightarrow \sum (1, 3, 4, 5) ; [function gives 1]$

$$\therefore F = x'y'z + x'y.z + xy'z + xyz'$$

By Maxterms, $F = M_0 \cdot M_2 \cdot M_6 \cdot M_7$
 $\Rightarrow \prod (0, 2, 6, 7) ; [function gives 0]$

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

* Express the Boolean Function

$$F = A + B'C \text{ as sum of minterms}$$

Solⁿ: $F = A + B'C$

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

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

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

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

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

④

$$= \sum(m_7 + m_6 + m_5 + m_4 + m_1)$$

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

* Express the Boolean function $F = xy + z^2$ as product of minterms

Solⁿ: $F = xy + z^2$

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

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

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

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

$$\begin{aligned}
 &= (x' + y + z)(x' + y + z')(x + y + z) + (x + y' + z) \\
 &\quad (x + y + z)(x + y + z') \\
 &= (x' + y + z)(x' + y + z')(x + yz)(x + y' + z) \\
 &= M_0 M_2 M_4 M_5 \\
 &= \overline{\Pi}(0, 2, 4, 5)
 \end{aligned}$$

* 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 others)
 * Converting POS to SOP }

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

↳ always 1

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

$$F = \Pi(N:D)$$

Lecture 8:

Karnaugh Map (K-MAP)

A table of n variables will have 2^n cells

Rules: (SOP) 2^n cells

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

2-variable K-Map

- self study.

Three variable K-map: * cell address matters

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

Example: Simplify using K-map:

i) $F(A, B, C) = \Sigma(1, 2, 3, 5, 7)$ \rightarrow can be done in SOP
ii) $F(x, y, z) = \Sigma(3, 4, 6, 7)$ \rightarrow " "

		BC	B'C'	B'C	BC	BC'
		A'	0	1	1	1
		A	1	1	1	
0	0					
1	0					

* Groups should be visible
→ use colors if necessary

$F = A'B + C$ * if group covers all rows or columns
it will not be counted

		yz'		yz		yz'	
		00	01	11	10	00	11
		x'	0	1	1	1	1
0	0						
1	0						
0	1	100		111	110		
1	1						

$yz + xz'$

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_2	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) $F(x, y, z) = \sum(0, 1, 2, 4, 5, 6, 8, 9)$

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

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

	01	11	10	00
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

For 2-variable

A	B	$B=0$	$B=1$
$A=0 \rightarrow 0$		m_0	m_1
$A=1 \rightarrow 1$		m_2	m_3

III

00	01
10	11

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

A	B		
0	0	1	0
1	1	1	1

For 3-variable:

A	B	C	
0	0	0	
0	0	1	

→ we use gray code instead of binary order

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

~~Q5~~ Q7 ($w-y$)

K-Map Practices

02/03/24
(online)

		C	
AB		0	1
A	0	m_0	m_1
	1	m_2	m_3
1	00	m_6	m_7
1	10	m_4	m_5

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

3 Variable Practice:

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

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

$\rightarrow A'0$ and C common

$$\begin{aligned} F &= A' C + A C' \\ &= A \oplus C \end{aligned}$$

* $\sum m(0, 1, 2, 4, 6)$

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

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

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

	x'	y'	z'	w'
	00	01	11	10
x'	1	1	1	1
y'	1	1	1	1

$$F = z' + x'y$$

$$\begin{matrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \\ 0 & 0 \end{matrix} \Rightarrow \begin{matrix} 1 \\ 0 \\ 1 \\ 0 \end{matrix}$$

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

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

$$F = z' + x'z$$

* Question may have Boolean expressions instead of F

- Place it directly OR
- Express in min terms

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

	wx'	yz'		
	00	01	11	10
00	1	1		
01	1	1		
11	1	1		
10	1	1		

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

$$F(A, B, C, D) = \sum m(0, 2, 4, 5, 7, 8, 10, 12, 13, 15)$$

~~WB~~ ~~CD~~

	00	01	11	10
00	1			1
01		1 1		
11		1 1		
10	1 1			1

$$F = \bar{B}D +$$

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

~~A~~ ~~B~~ ~~C~~ ~~D~~

U	U	U	U
		1	
			0
0	1	1	1

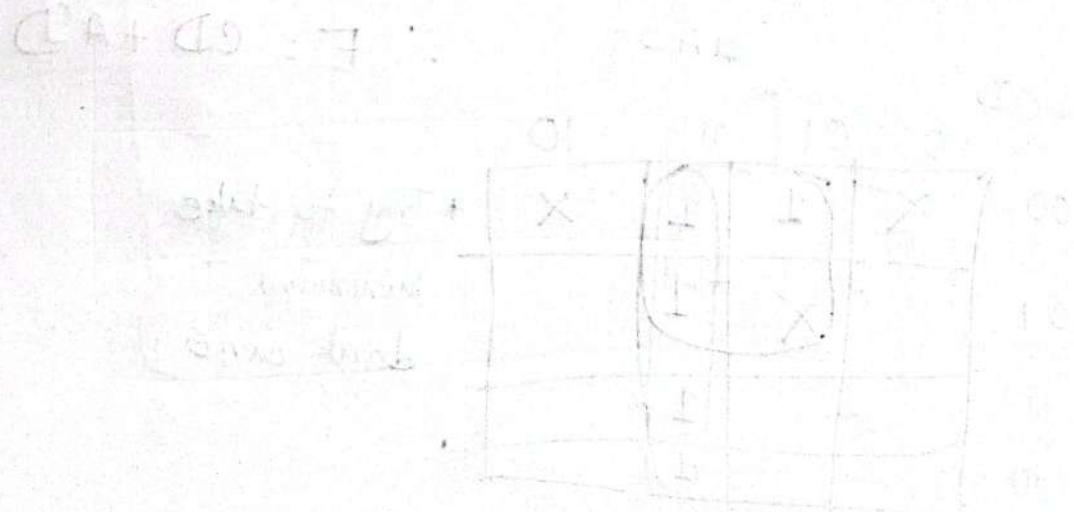
$$F = B'D' + B'C' + A'C'D$$

$$F(A, B, C, D) = \bar{T}(3, 4, 6, 7, 11, 12, 13, 14, 15)$$

	AB	CD	$C+D$	$C'+D'$	$C'+D'$	$C'+D$
	00	00	0	0	0	0
$A+B$	00					
$A+B'$	01	0	0	0	0	0
$A'+B'$	11	0	0	0	0	0
$A'+B$	10		0			

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

↓ $B'+D$
↓ $C'+D'$
↓ $A'+B'$



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. One point about the outputs of this particular combination is that the minterms for three functions are called don't care.

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

Example: Simplify using K-map:

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

$$d = \sum(0, 2, 5) \rightarrow d \text{ on } X$$

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

$$d = \sum(0, 5)$$

$$\therefore F = CD + A'D$$

AB \ CD	00	01	11	10
00	X	1	1	X
01	X	1	1	
11		1	1	
10			1	

A'D

+ Try to take
minimum
don't care

i)

A	BC	00	01	11	10
0	X	1	1		
1		X	1		

C

$$\therefore F = C$$

Lecture 10:

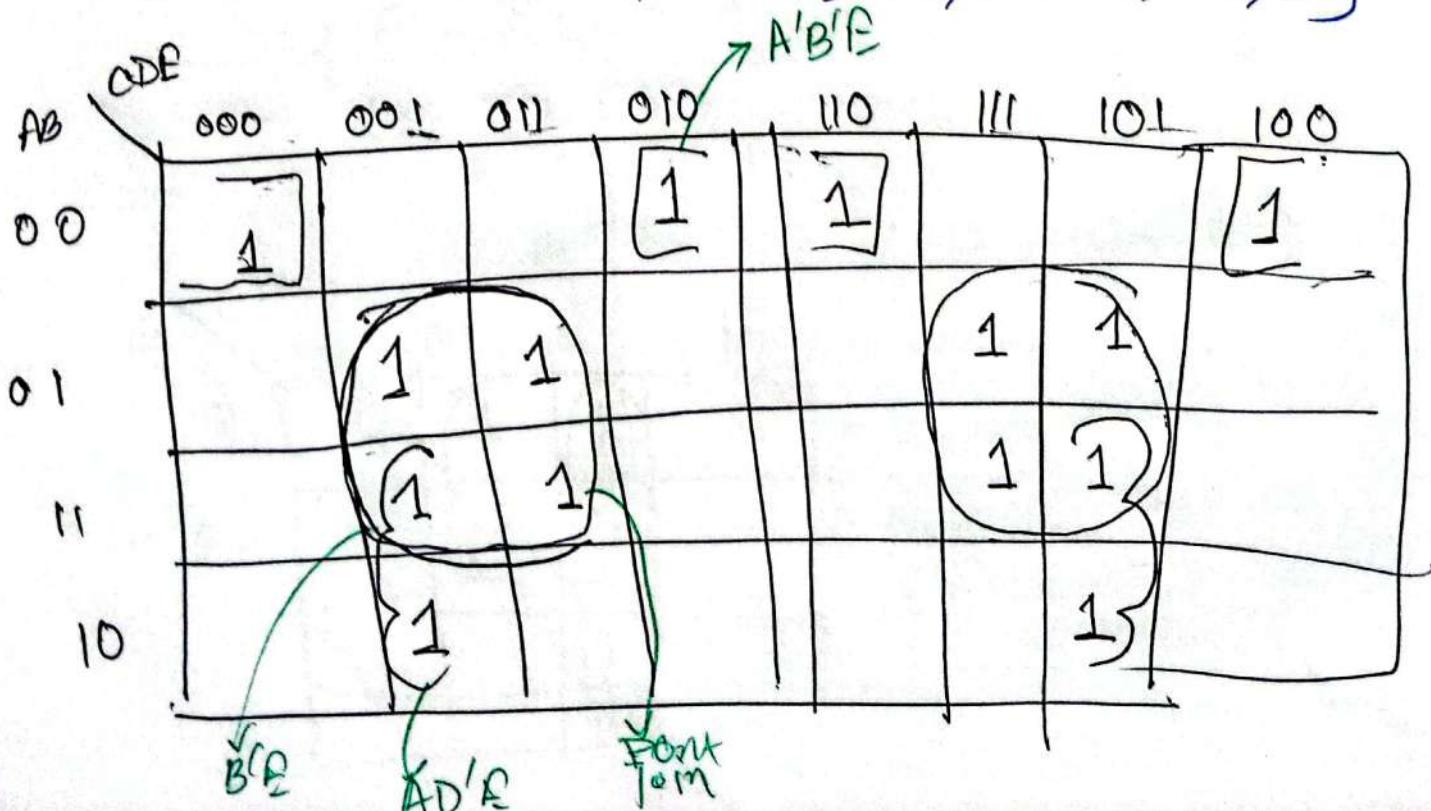
* Minimun Rule:

* 5 variable K-map

AB		CDE		<i>i</i>		<i>ii</i>				<i>ii(i)</i>		<i>i(ii)</i>	
		000	001	011	0010	110	111	101	100				
		00	0	1	3	2	6	7	5	4			
01	8	9	11	10	14	15	13	12					
11	24	25	27	26	30	31	29	28					
10	16	17	19	18	22	23	21	20					

Example: Simplify using K-map

$$F = \sum(0, 2, 4, 6, 9, 11, 13, 15, 17, 21, 23, 27, 29, 31)$$



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

7 variable K-map:

but not this

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	28	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

Example: Simplify using K-Map:

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

	000	001	011	010	110	111	101	101
000					1			
001		1					1	
011		1	(1)				1	
010			(1)	(1)				
110								
111		1					1	
101			1				1	
100								

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

A.W.

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

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

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

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

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

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

$$i \rightarrow 4$$

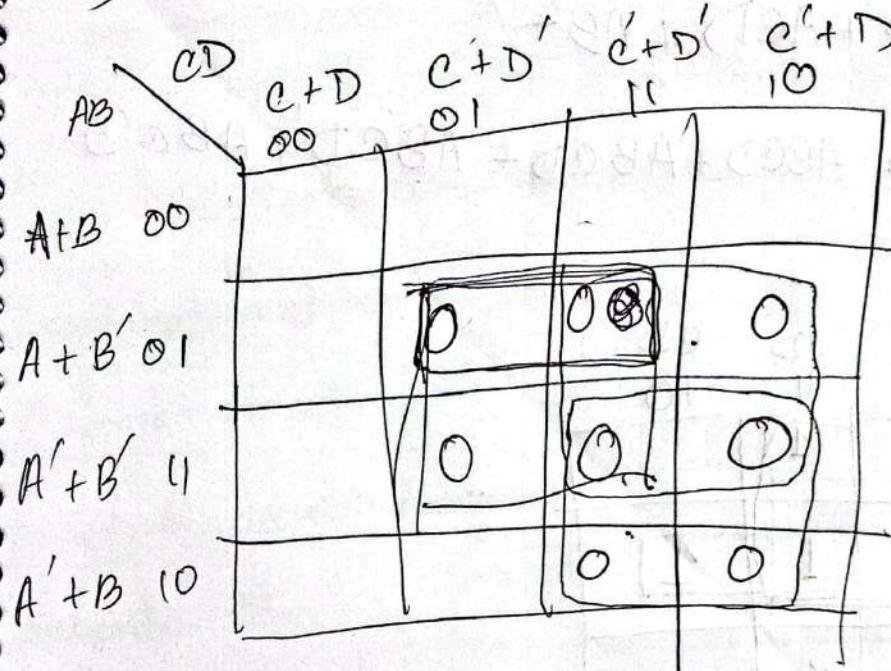
$$2 \rightarrow 3$$

$$4 \rightarrow 2$$

$$8 \rightarrow 1$$

$$Z^n \quad n - g + 1 \\ 4 - 2 + 1.$$

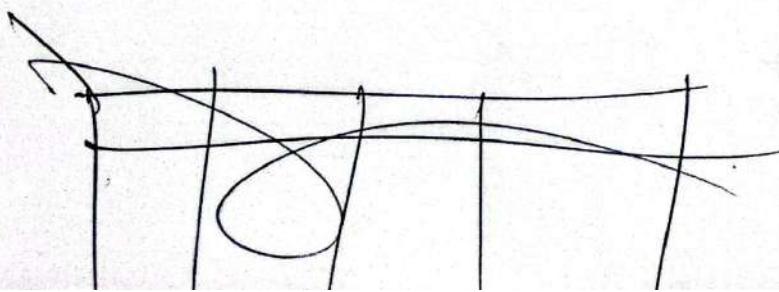
$$\text{ii) } F(A, B, C, D) = \prod (5, 6, 7, 10, 11, 13, 14, 15)$$



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

$$\text{iii) } F(A, B, C, D) = A'B'D + A'CD + A'BC \\ d = A'BC'D + ACD + A'B'D'$$

$$\begin{aligned} F(A, B, C, D) &= A'B'D + A'CD + A'BC \\ &= A'B'D(C+C') + A'CD(B+B') + A'BC(D+D') \\ &= A'B'CD + A'B'C'D + A'BCD + A'B'CD \\ &\quad + A'BCD + A'BCD' \end{aligned}$$



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

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

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

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

$$F = A'D + A'BC$$

Lecture - 11

Chapters - 4 : Combinational Logic

Logic circuits for digital systems may be
 sequential
combinational or, circuits
 upto now

Combinational Circuits:

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. [Outputs depend entirely on inputs at that exact moment]

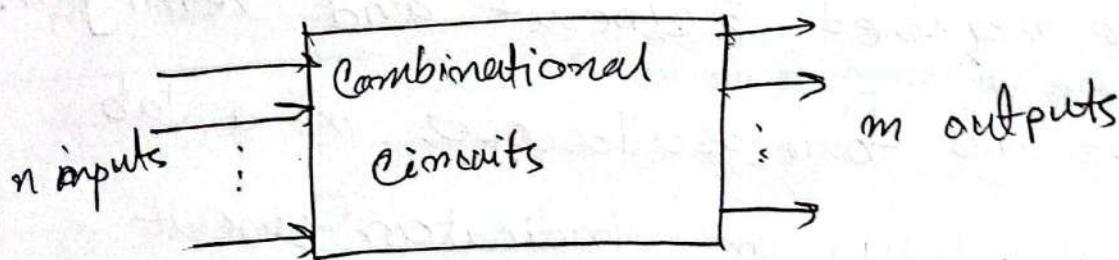


Fig: Block diagram of combinational circuit

*Diff betⁿ seq comb and seq circs → 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 design an indicator circuit to ~~determine~~ detect the condition when two or more of the oscilloscopes are malfunctioning

Circuit Diagram

- Things we draw ~~is~~ with basic ~~names~~ mentions
- Detailed

Ex →

Let, working condition be 1

non-functioning " 0 " 1 of the
when 2^n bits will be 0

∴ The indication will be 0

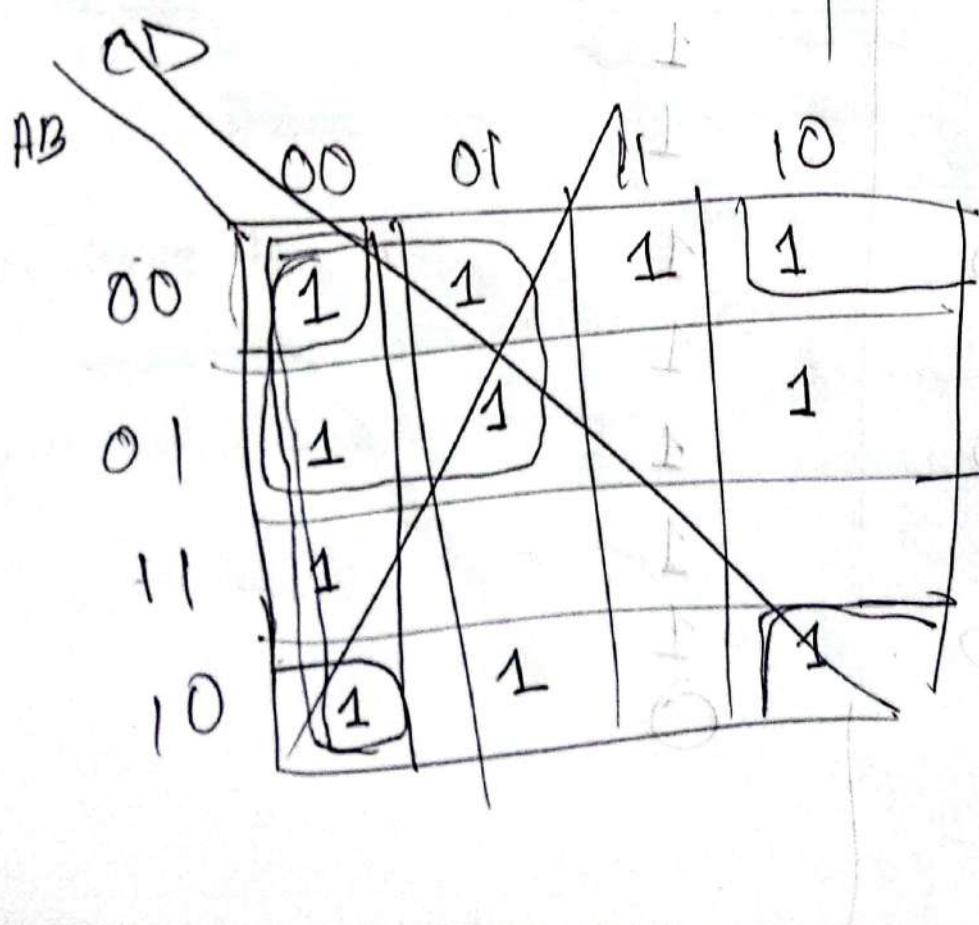
A	B	C	D	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	0	0	0	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0

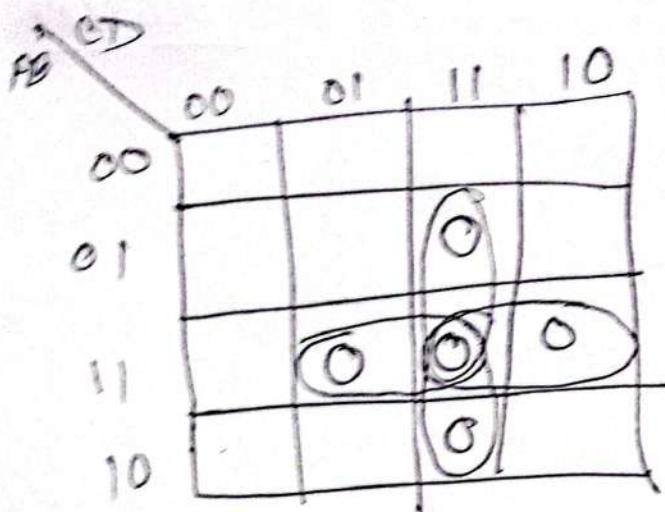
Block Diagram

- Not mentioned. Makes a block with name

- Not detailed

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





AB

Binary Adder

* Half Adder :

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

<u>Input</u>				
X	Y	\oplus	C	S
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	0	1	0

$$\therefore C = X \cdot Y$$

$$\therefore S = X \cdot Y' + X' \cdot Y$$

$$= X \oplus Y$$

Cannot do:
 $A = 1110$ $B = 1111$
 Cannot become
 borrow 3 bit

∴ Half Subtractor : B D
 Borrow Diff

*Full Adder:

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

X	Y	C _{in} (Carry 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	-	1
1	0	1	-	0
1	1	0	-	1
1	1	1	-	0

Carry = 1

A = 1101101

+ B = 1111101

{ 7 Addends

Q1) Arrangement of 2 half adders to form a full adder:

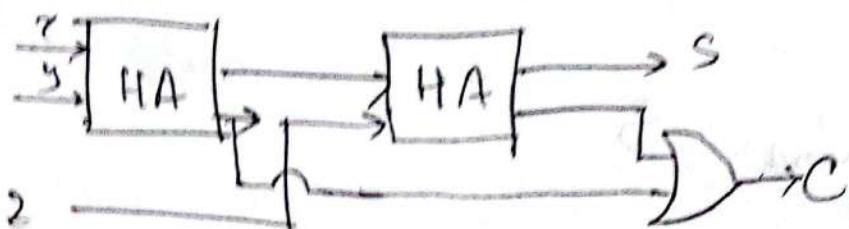


Fig: Diagram Block diagram of a full adder using 2 half adder.

Q2) Self Study:

- i) Half Subtraction [produces $x-y$]
- ii) Full subtraction [produces $x-y-z$]
- iii) Binary Subtractor

Magnitude Comparators

- A magnitude com 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 comparators:

Input		Outputs		
<u>x</u>	<u>y</u>	<u>$F_1 (A > B)$</u>	<u>$F_2 (A = B)$</u>	<u>$F_3 (A < B)$</u>
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	$x'y' + xy$	$x'y$
		xy'	$= x \oplus y$	

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

2 bit

$\frac{X}{x_2 x_1}$	$\frac{Y}{y_2 y_1}$	$F_1(A > B)$	$F_2(A = B)$	$F_3(A < B)$
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	0
0 1	0 0	0	1	0
0 0 1 0	0 0	0	1	0
0 1	0 0	1	0	0
1 1	0 1	1	0	0
1 1	1 0	1	0	0
1 1	1 1	0	1	0

[$F_{1,2}$ at home]

3-bit:

$$x = x_3 x_2 x_1$$

$$y = y_3 y_2 y_1$$

$$F_1(x > y) = \text{if } (x_3 > y_3) + 0$$

$$\text{if } (x_3 = y_3) \& \& (x_2 > y_2) +$$

$$\text{if } (x_3 = y_3) \& \& (x_2 = y_2) \& \& (x_1 > y_1)$$

$$= x_3 y_3' + (x_3 \odot y_3)(x_2 y_2') + (x_3 \odot y_3)(x_2 \odot y_2)$$

(from 0 1-bit)

$$F_2(x = y) = \text{if } (x_3 = y_3) \& \& (x_2 = y_2) \& \& (x_1 = y_1)$$

$$= (x_3 \odot y_3)(x_2 \odot y_2)(x_1 \odot y_1)$$

$$F_3(x < y) = \text{if } (x_3 < y_3) +$$

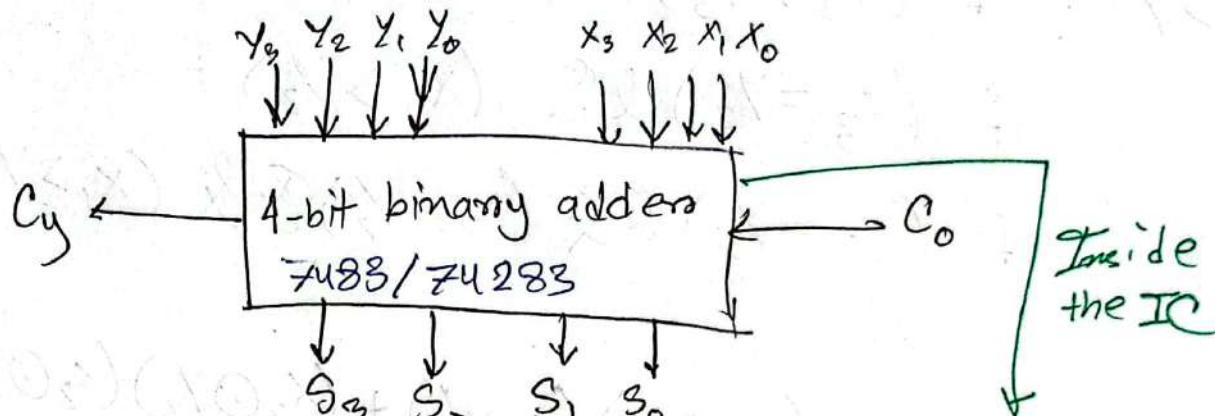
$$\text{if } (x_3 = y_3)(x_2 < y_2) +$$

$$\text{if } (x_3 = y_3) > (x_2 = y_2)(x_1 < y_1)$$

$$= x_3' y_3 + (x_3 \odot y_3)(x_2' y_2) + (x_3 \odot y_3)(x_2 \odot y_2)(x_1' y_1)$$

* Draw Diagrams. ALWAYS
Binary Parallel Adder:

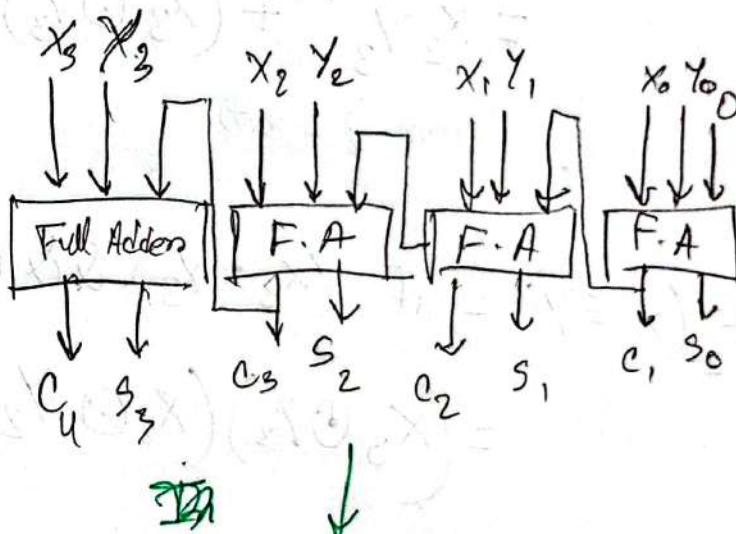
- produces the arithmetic sum of two binary numbers in parallel



$$X = x_3 \ x_2 \ x_1 \ x_0$$

$$Y = y_3 \ y_2 \ y_1 \ y_0$$

$$\begin{array}{r} C_3 \quad C_2 \quad C_1 \\ x_3 \quad x_2 \quad x_1 \quad x_0 \\ + \quad y_3 \quad y_2 \quad y_1 \quad y_0 \quad C_0 = 0 \end{array}$$



$C_y \quad S_3 \quad S_2 \quad S_1 \quad S_0$
and and
 $C_2 \quad C_1$

(Carry is completed,
the next operation cannot
happen)

→ Ripple Effect

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,

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

\rightarrow carry propagate \rightarrow carry generate
So, the output sum (S_i) and carry (C_{i+1}) of full adder can be expressed as,

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

3-bit CLA adder:

$$S_1 = P_1 \oplus C_0 = \cancel{P_1 \oplus 0} = \cancel{P_1} \quad A = A_1, A_2, A_3$$

$$B = B_1, B_2, B_3$$

$$\therefore C_1 = P_1 = A_1 \oplus B_1 \quad [\because P_1 = A_1 \oplus B_1]$$

$\therefore C_1$: initial carry = 0

$$\text{Similarly, } C_2 = G_1 + P_1 C_1 \Rightarrow A_1 B_1 + 0 = A_1 B_1$$

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

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

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

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

To 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

26/05/24	
SUNDAY	Lecture 8-13
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)
9	0 1 0 0 1	0 1 0 0 1 (9)
10	0 1 0 1 0	1 0 0 0 0 (10)
11	0 1 0 1 1	1 0 0 0 1 (11)
12	0 1 1 0 0	1 0 0 1 0 (12)
13	0 1 1 0 1	1 0 0 1 1 (13)
14	0 1 1 1 0	1 0 1 0 0 (14)
15	0 1 1 1 1	1 0 1 0 1 (15)
16	1 0 0 0 0	1 0 1 1 0 (16)
17	1 0 0 0 1	1 0 1 1 1 (17)
18	1 0 0 1 0	1 1 0 0 0 (18)
19	1 0 0 1 1	1 1 0 0 1 (19)

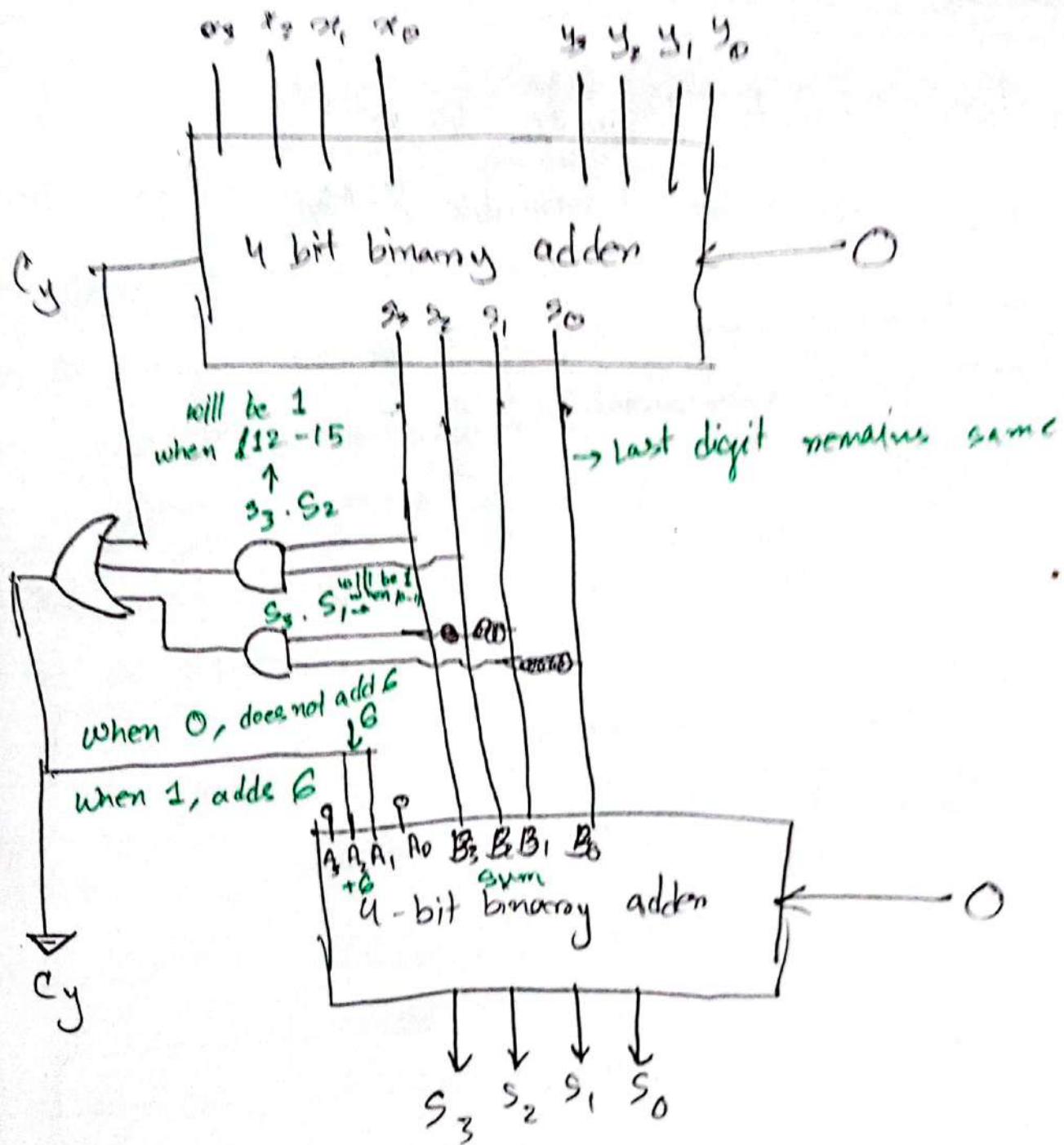
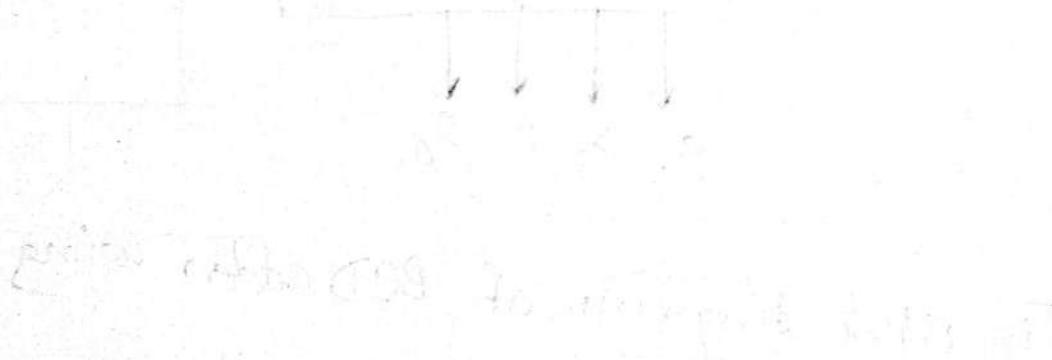


Fig: Block diagram of BCD adder using binary adder.

For Book:

Emraan Shekhar : Gets 1 when 10^{-15}

then makes 4 variable K-Map

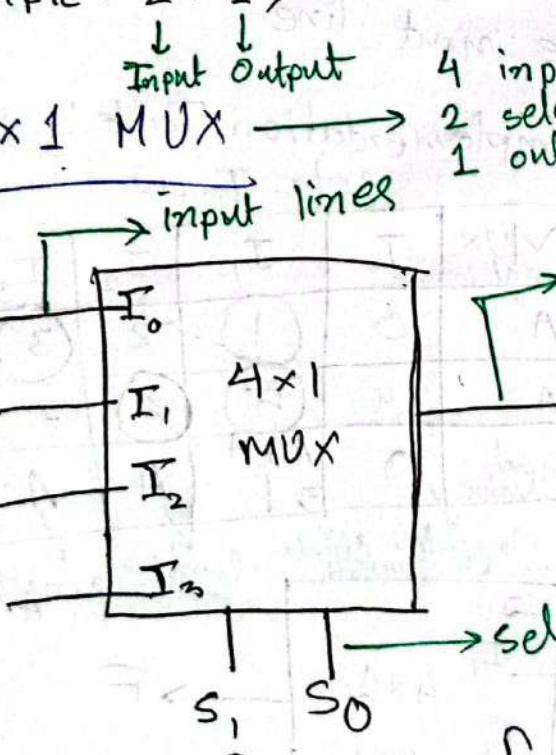


Multiplexers / MUX

Data Selection:  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. $\text{input } (2^n)$, $\text{output } (1)$, $\text{selector } (n)$
- The size of a MUX is specified by the number 2^n of its input lines the single output line.
- It also contains n selections.

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

4×1 MUX  Function table:

		S_1	S_0	Y	Selectors determine what outcome is selected
0	0	I ₀			
0	1		I ₁		
1	0		I ₂		
1	1		I ₃		

Fig : Block Diagram of a 4×1 MUX

$$\begin{aligned} Y = & S_1' S_0' I_0 + S_1' S_0 I_1 \\ & + S_1 S_0' I_2 + S_1 S_0 I_3 \end{aligned}$$

Logic/Circuit Diagram: See generated

Uses of MUX:

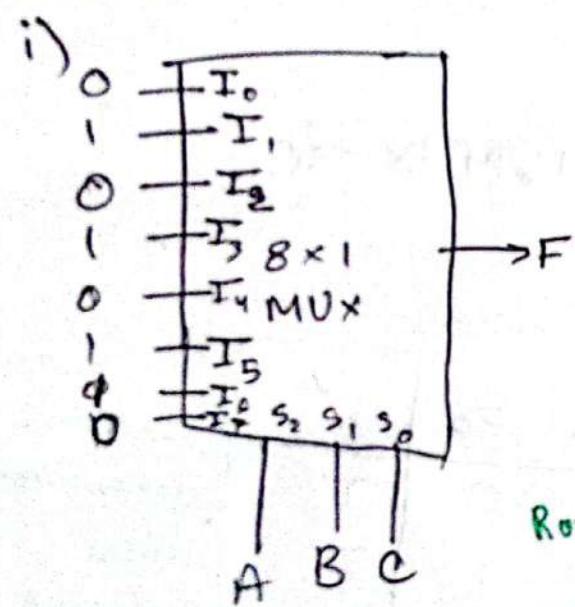
- Connecting two or more sources to a single destination
- It is useful for constructing a common bus system

Function Implementation Using MUX:

Example: Implement $F(A, B, C) = \sum(1, 3, 5, 6)$

using: i) 8×1 MUX, ii) 4×1 MUX

Solⁿ:



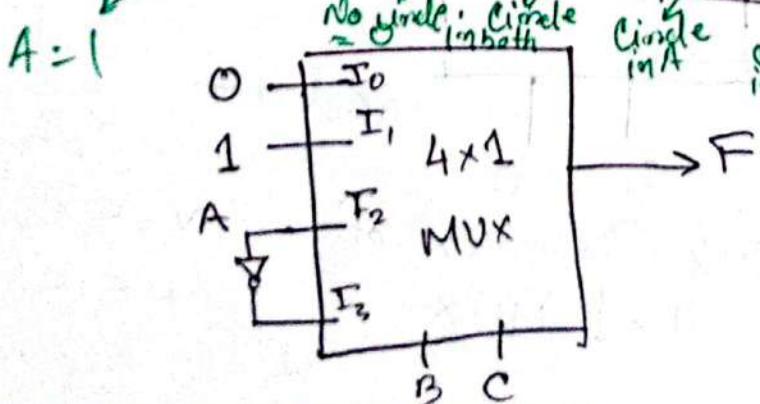
ii) Using 4×1 MUX implementation table using
implementation table using
 B, C as selections, A in
the input line

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	$A + A'$	A	A'

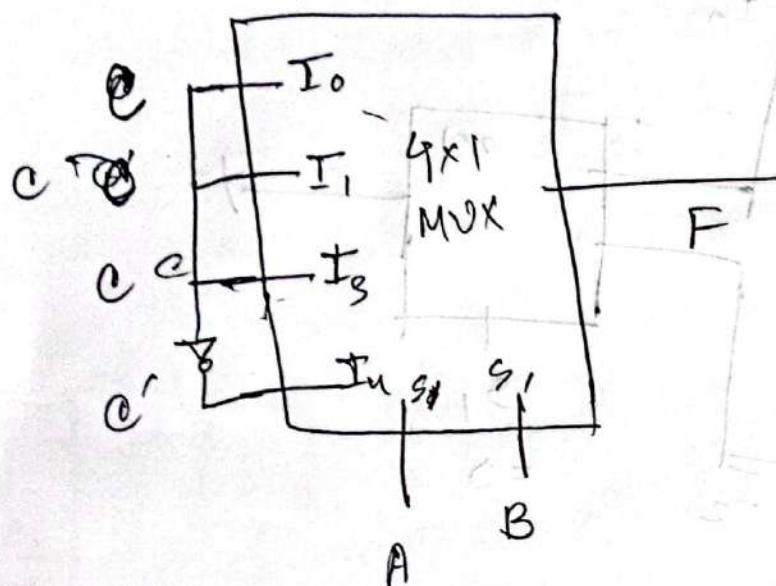
Row = No. of selectors as input + 2



C.W C.W:

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

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

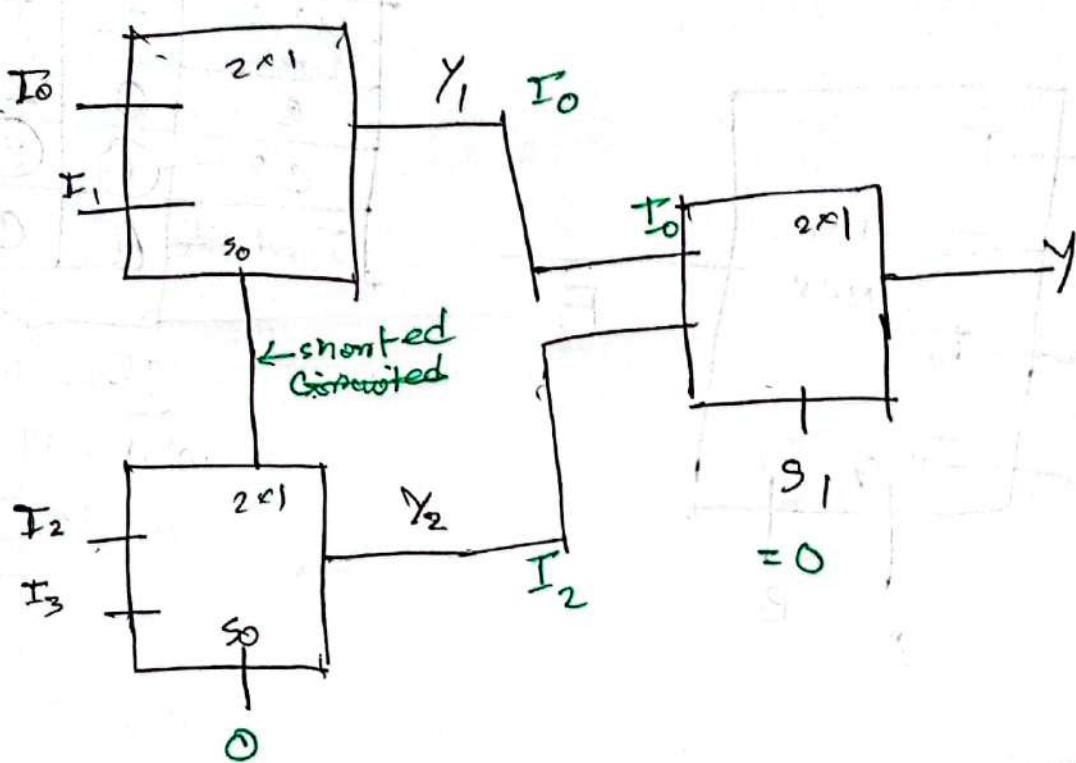


MUX Line	I_0	I_1	I_2	I_3
c'	0	2	4	6
c	1	3	5	7
Input Value	A	C	C	C'

These determine
what will be
in the Input

Multiplexer Expansion:

* 4x1 MUX using 2x1 MUX



* Description: Say different values yield different values

→ Not all values of have to be explained

_MUX

- Input line
- Output line
- Selectors
- Enable pin

⑩ Active High Enable:

The output is enabled when $E = 1$.

Notation: $E (5V)$

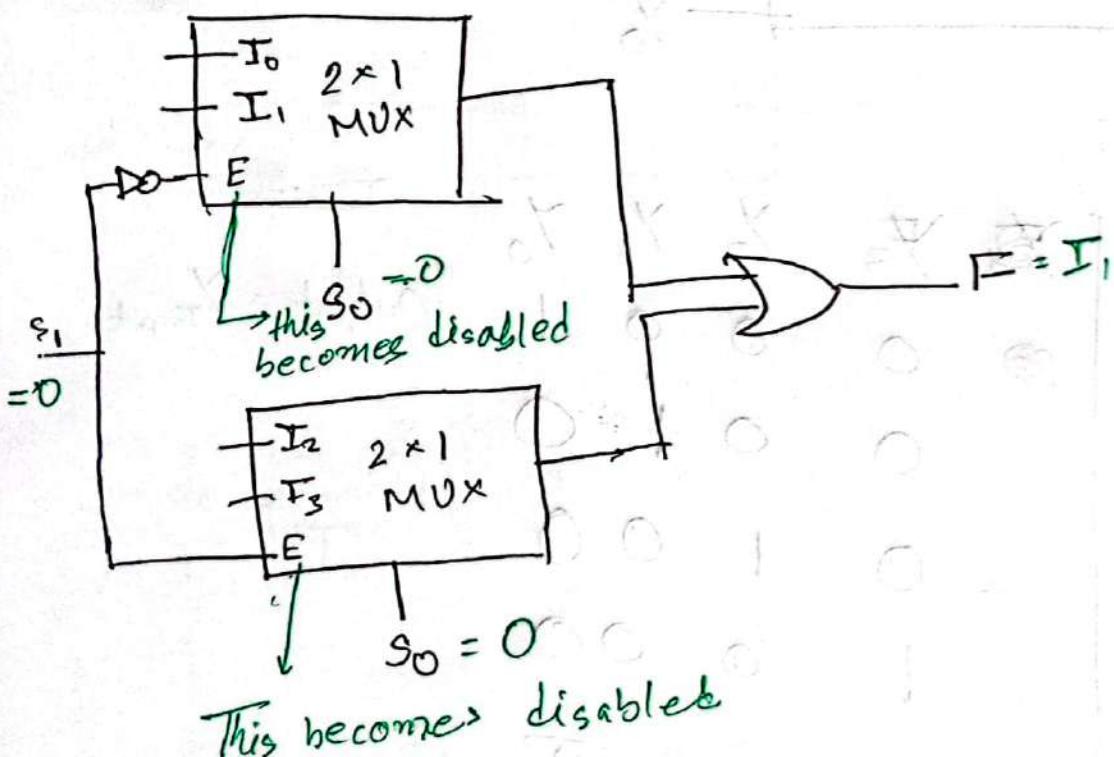
Active Low Enable:

The output is enabled when $E = 0$.

Notation: $\bar{E} (\text{Gnd})$

Based on
type of MUX

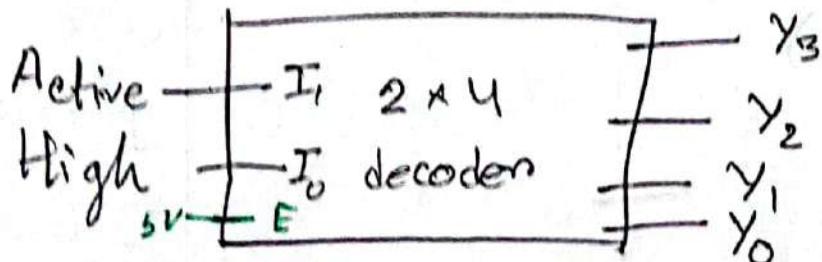
$E_1 \downarrow 5V$ $\bar{E}_2 \downarrow 0V$ $E_3 \downarrow 5V$



Decoder:

Converts binary information from n input lines to a maximum of 2^n unique output lines. [no selector]

* 2 to 4 lines Decoder:



Function Table:

I_1	I_0	Input	Y_3	Y_2	Y_1	Y_0	$\text{Output} = Y_{\text{Input}}$
0	0	0	0	0	0	1	1
0	1	0	0	0	1	0	0
1	0	0	0	1	0	0	0
1	1	1	0	0	0	0	0

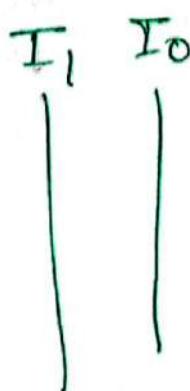
$$Y_3 = I_1 I_0$$

$$Y_2 = \overline{I}_1 I_0$$

$$Y_1 = I_1 \overline{I}_0$$

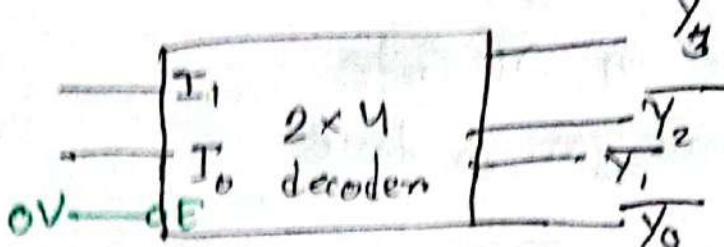
$$Y_0 = \overline{I}_1 \overline{I}_0$$

Circuit diagram
- draw yourself



Active low

2x4 decoder

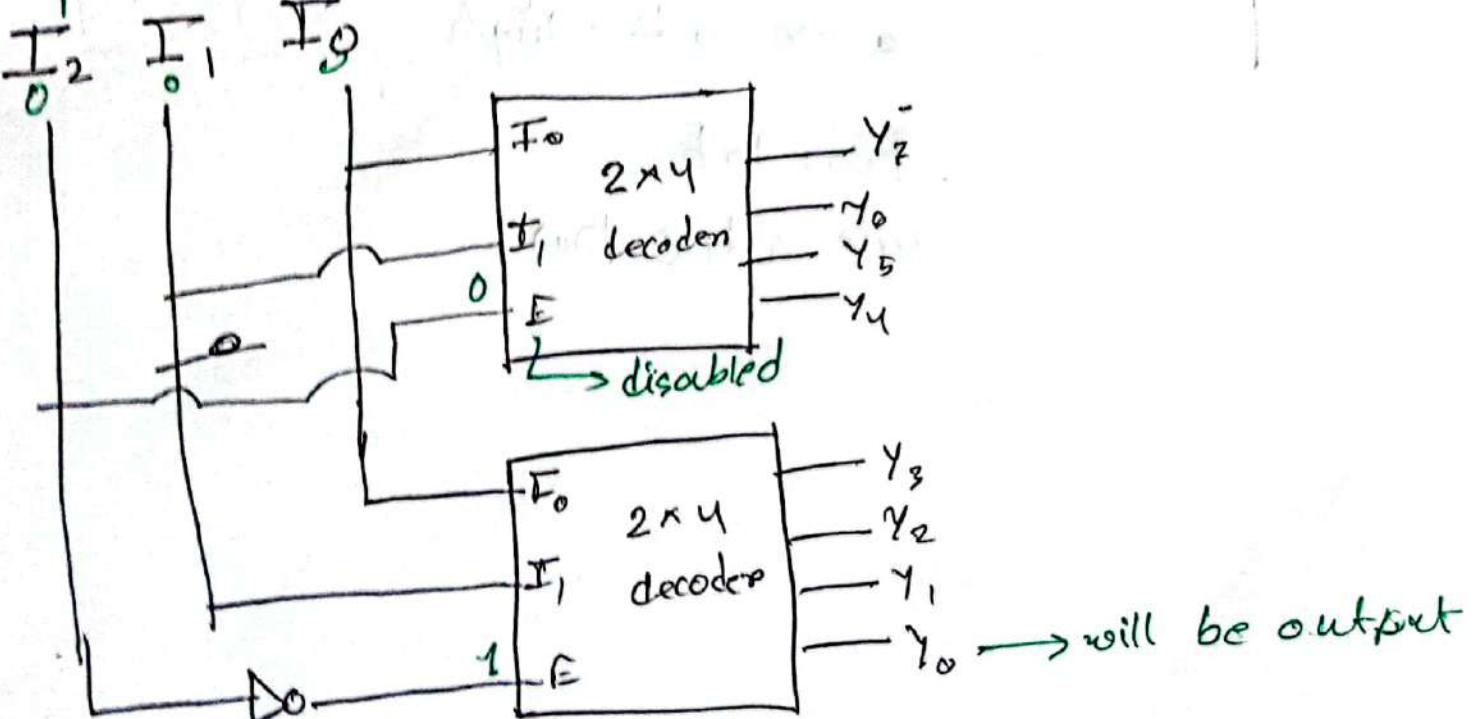


I_1	I_0	\overline{Y}_3	\overline{Y}_2	\overline{Y}_1	\overline{Y}_0
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1

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

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

use I_2 as enable pin

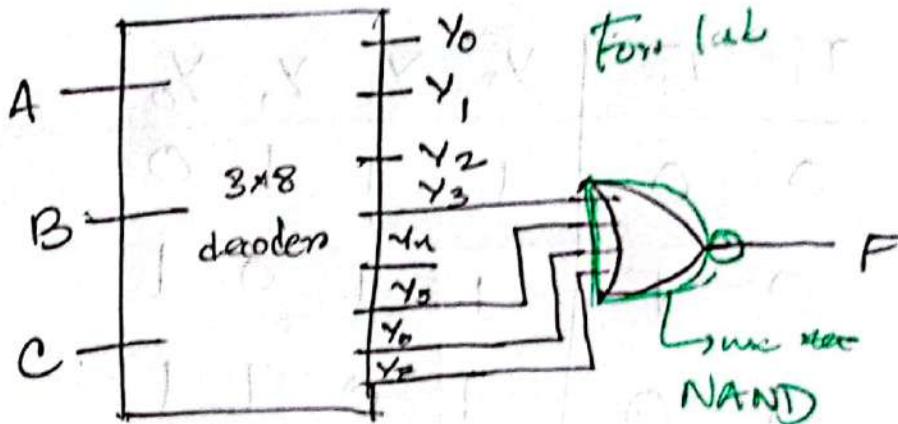


* Boolean Function implementation using Decoders

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

→ put in OR Gate
and job 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

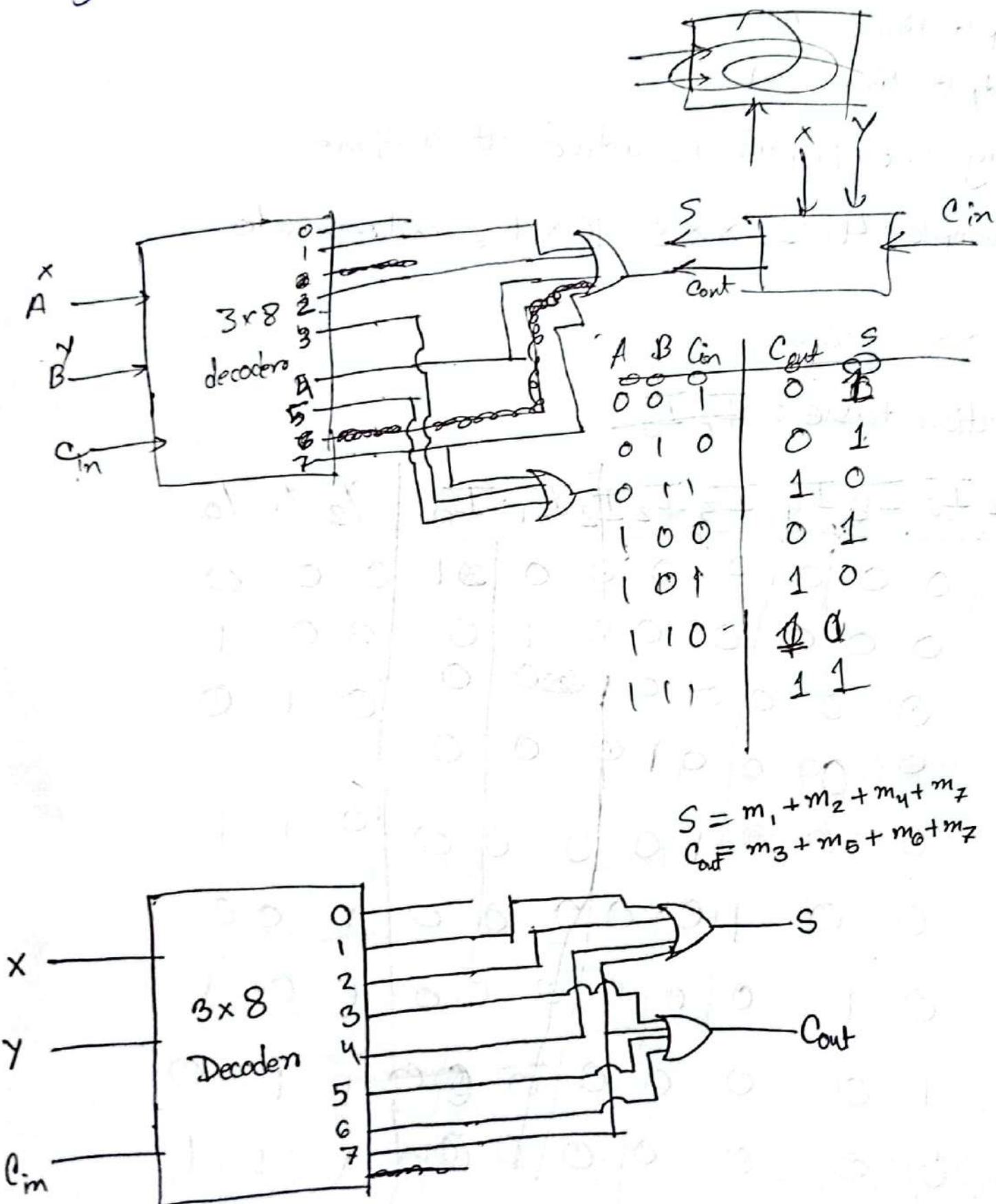


* For theory -

if nothing is presented,
use active high

For lab,
use active low

PW: Implement 1 bit full adder using 3×8 decoders
using 3×8 decoders & basic gates.



Encoder:

- Does reverse operation of decoders
- 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: ~~F₇ I₆~~

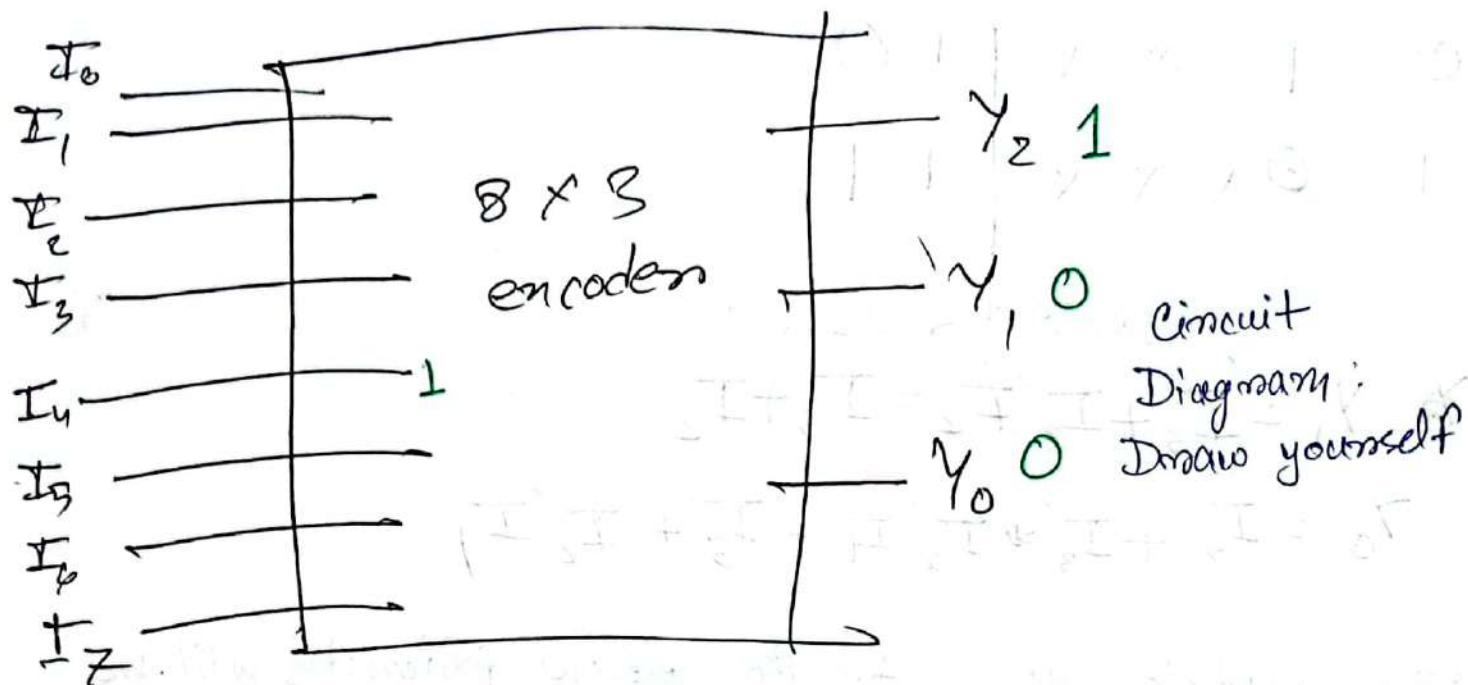
I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	y_2	y_1	y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	0	0	1
0	0	0	0	0	0	1	0	0	1	0
0	0	0	0	0	1	0	0	0	1	1
0	0	0	0	0	1	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Output Equations:

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

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

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



Priority Encoders:

- 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

$\rightarrow 1xxx \quad \rightarrow 01xx$

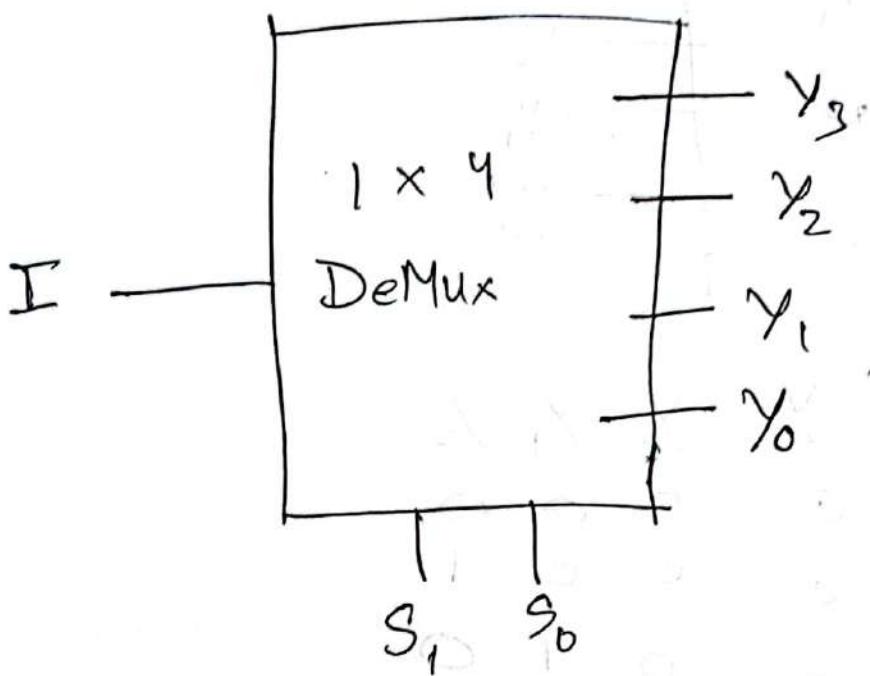
$$\therefore y_1 = I_3 + I_3' F_2 = I_3 + I_2$$

$$y_0 = I_3 + I_3' * I_2' F_1 = I_3 + I_2' I_1$$

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

* DeMultiplexen:

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



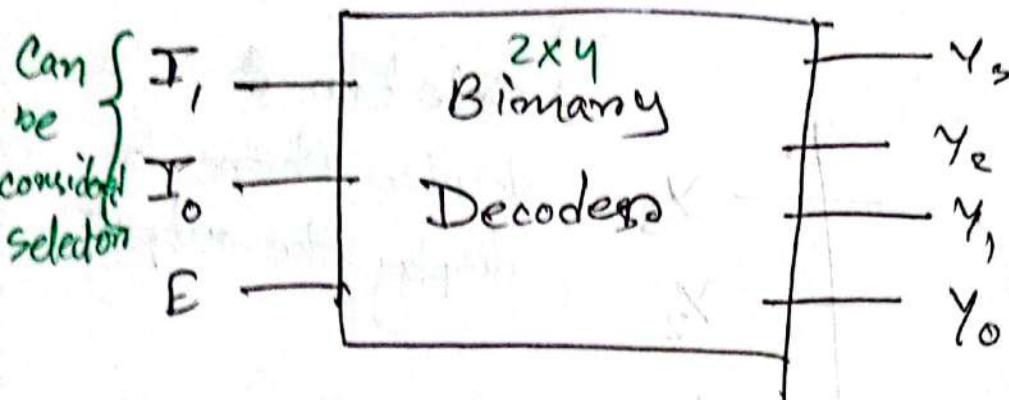
* Selection

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	I
0	1	0	0	I	0	0
1	0	0	I	0	0	0
1	1	F	0	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	1	0	0	0

* Decoders and DeMux have same IC

For fixed : Easy \rightarrow Use as
Hard \rightarrow Modify the Easy ones

Quine-McCluskey / Tabulation Method:

Simplify the following Boolean Function by using Tabulation method:

$$F(a, b, c, d) = \sum (0, 12, 3, 6, 2, 8, 9, 10, 14)$$

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

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

Step 1: Determination of prime implicants (PIs):

Column I → No. of 1's
 Column - II → 3 bits same
 with immediate group
 Column - III → Same portion
 in same row + 1 bit

	a b c d	000 -	00 - 0	0 - 000	a b c d
Group 0: 0	a b c d	(0, 1)			(0, 1, 8, 9) - 00 - *
Group 0: 0 0000		(0, 2)	00 - 0		(0, 2, 8, 10) - 0 - 0 *
Group 1: 1 0001	✓	(0, 8)	0 - 000		(2, 6, 10, 14) 0 - - 10
2 0010	✓		(1, 5) 00 - 01 *		
8 1000	✓			(21, 9) 0 - 001 ✓	
Group 2: 0 101	✓		(2, 6) 0 - 010 ✓		
6 0110	✓		(2, 10) - 0 010 ✓		
9 1000	✓		(8, 9) 100 - ✓		
10 1010	✓		(8, 10) 10 - 0 ✓		
Group 3: 7 0 111	✓			(5, 7) 01 - 1 *	
14 1 110	✓			(6, 7) 01 1 - *	
				(6, 14) - 1 1 0 ✓	
				(10, 14) 1 - 1 0 ✓	

Step - 2: Selection of Prime Implicants:

- Prime implicants:

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

(1, 5)

(0, 1, 8, 9) (0, 2, 8, 10) (0, 2, 6, 10, 14)

Step - 2: Selection of Prime Implicants:

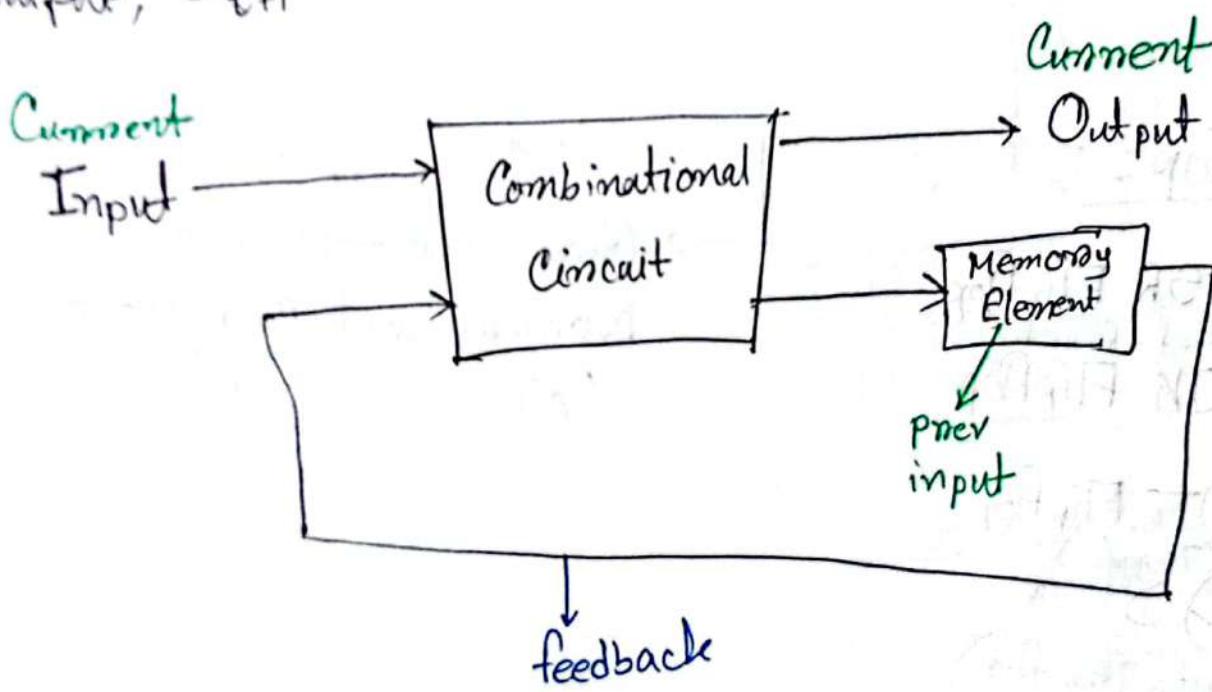
PIs com	PIs	a	b	c	d	0	1	2	5	6	8	8	10
(0, 1, 8, 9)	0 - 0 0 -	x	x								x	x	
(0, 2, 8, 10)	- 0 - 0 -	x					x				x	x	x
(2, 6, 10, 14)	- - 1 0			x				x			x	x	x
(1, 5)	0 - 0 1		x					x			x	x	x
(5, 7)	0 1 - 1				x			x		x	x	x	x
(1, 6, 7)	0 1 1 -					x	x	x	x	x	x	x	x

$$\text{Ans. } b'c' + cd' + a'b'd$$

Synchronous Sequential Logic

Current Output = $f(\text{previous output}, \text{current input})$

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



* Flip Flop(FF): Memory element.

Sequential Circuit

Asym
Asynchronous
changing sequence of
input changes order
of output

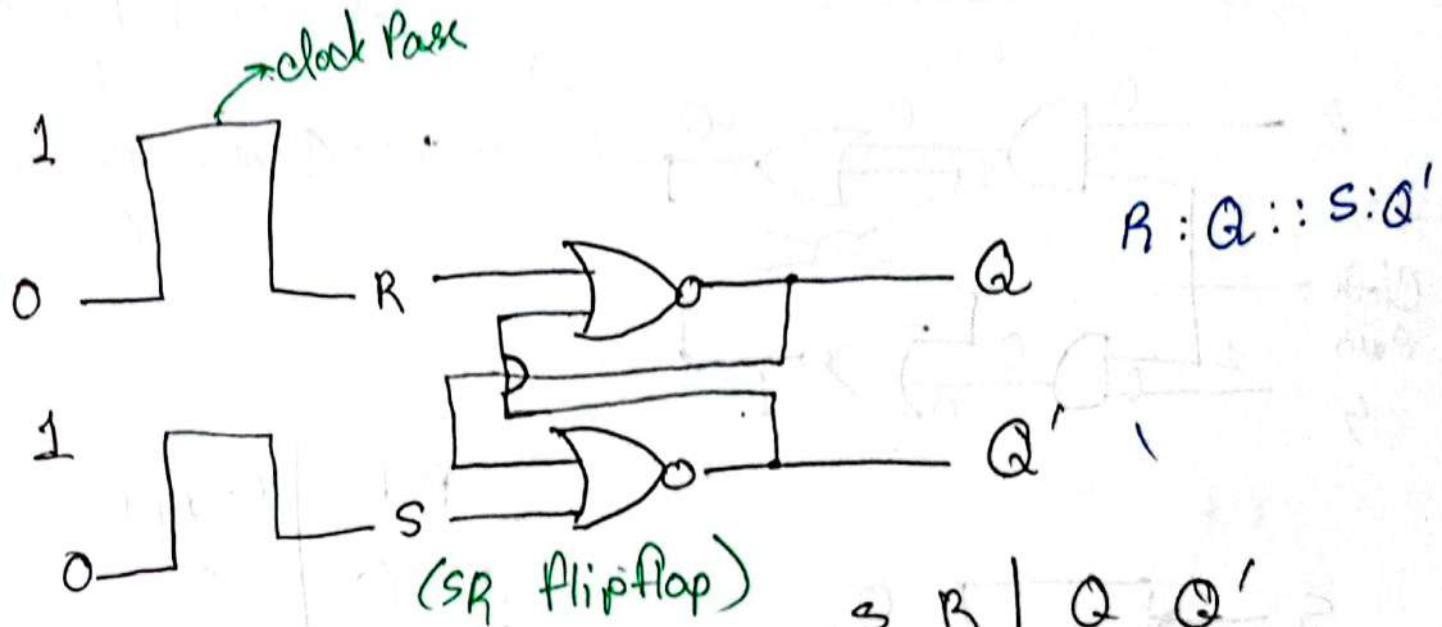
Synchronous
changing input time
changes output

Flip Flop

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

→ Stores only 1 bit
→ Gives two outputs;
Normal and Complement
(A) (A')

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



case
case
This is responsible
for the new
outputs

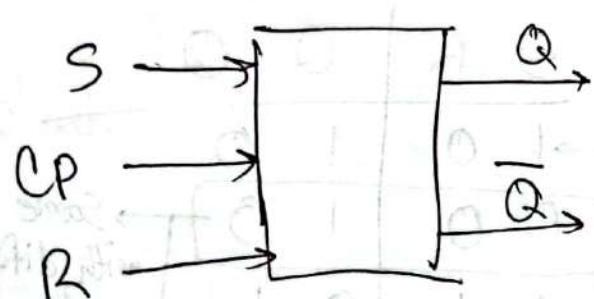
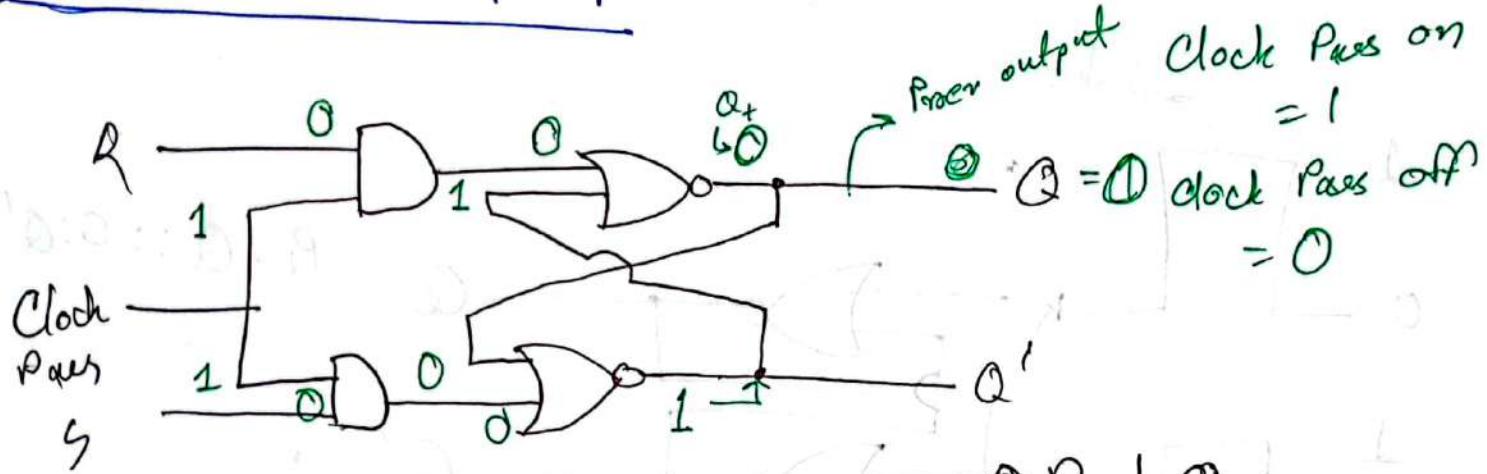
S	R	Q	Q'
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	0	0

Same in
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:



Characteristics Table

S	P	R	Q_{t+1}
0	0	0	Q
0	1	0	Q'
1	0	0	1
1	1	x	-

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

→ Characteristics Table

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

Excitation Table Map: (SR)

* Using desired output to find the required input

Q_+	Q_{++}	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 flip flop Excitation Table + the rest of FFs

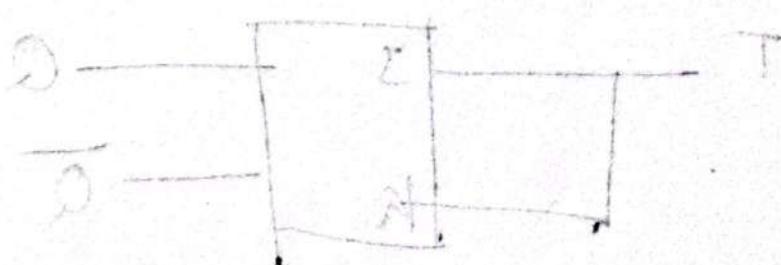
$$D'N + DC = , + D : NC$$

$$D \oplus T = DT + D'T = , , D : T$$

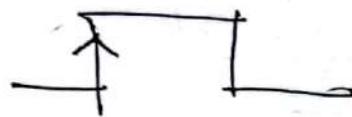
$$D'N + DC = , , D$$

$$D'T + D'T = , , D$$

$$D \oplus T = , , D$$



Triggering Flipflops:



* Triggering Point: When output changes.

\overrightarrow{CP} → positive edge

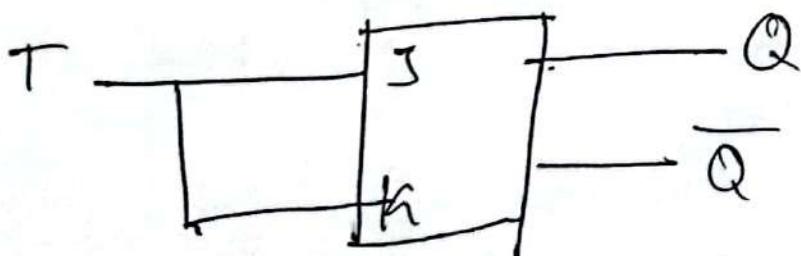
$\overrightarrow{\overline{CP}}$ → negative edge

Conversion:

* JK to T:

$$JK: Q_{++1} = JQ' + K'Q$$

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

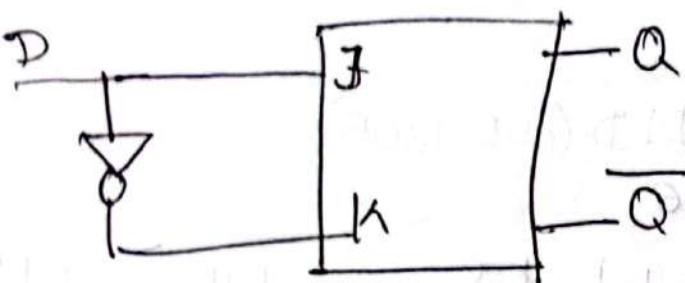


Hence,

$$\begin{aligned} Q_{++1} &= JQ' + K'Q \\ &= TQ' + T'Q \\ &= T \oplus Q \end{aligned}$$

* JK to D:

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



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

$$= \cancel{JQ'} +$$

$$= DQ' + (D')'Q$$

$$= DQ' + DQ$$

$$= D(Q' + Q)$$

$$= D\bar{Q}$$

Design Procedure of Sequential Ckt:

- i) Problem is stated \rightarrow text/state diagram/state table
- ii) Obtain the state table
- iii) State reduction \rightarrow 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

\rightarrow no. of bits = no. of FFs
 \rightarrow each IC has 2 bits
 \rightarrow 7470 \rightarrow JK flipflop
- vi) ~~Determine~~ 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

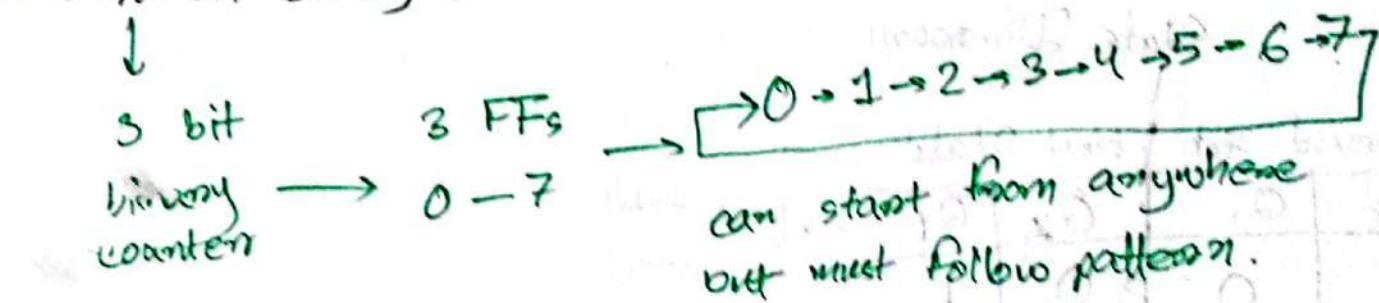
Registers & Counters

Counters: → keeps a count

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

- Counting occurrences of an event
- Generates timing sequence
- Tracks elapsed time between events.

→ Binary Counter follows binary sequence
 & Binary Counter needs n FFs and counts 0 to $2^n - 1$

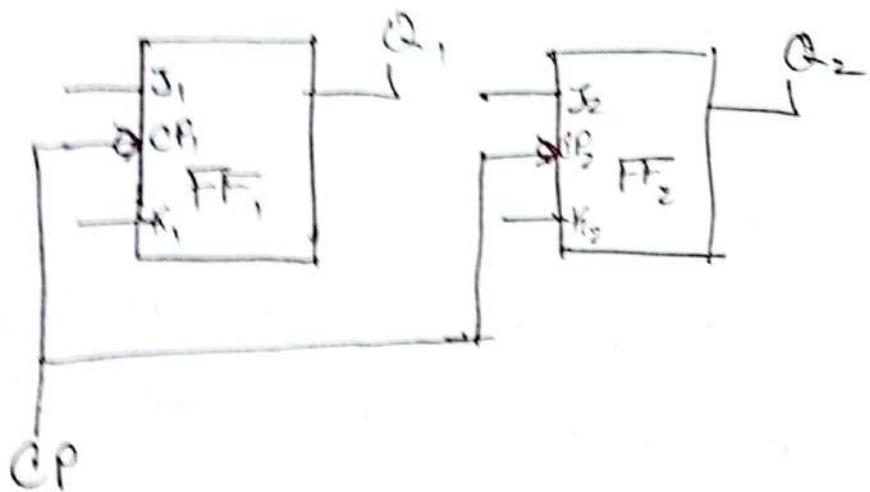


Counters is of 2 types:

i) Synchronous → same clock pulse

ii) Asynchronous → different clock pulse.

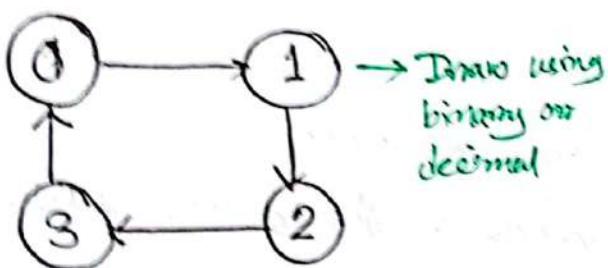
#P Design a 2-bit synchronous binary counter.



* $\overline{SD} \leftarrow 0$
Sets output to 1

* $\overline{RD} \leftarrow 0$
Sets output to 0

Keep them disabled by
5V to get different
values



State Diagram

Present State	Next State	CP	Q ₂	Q ₁
0	0	CP	0	1
0	1	CP	1	0
1	0	CP	1	1
1	1	CP	0	0

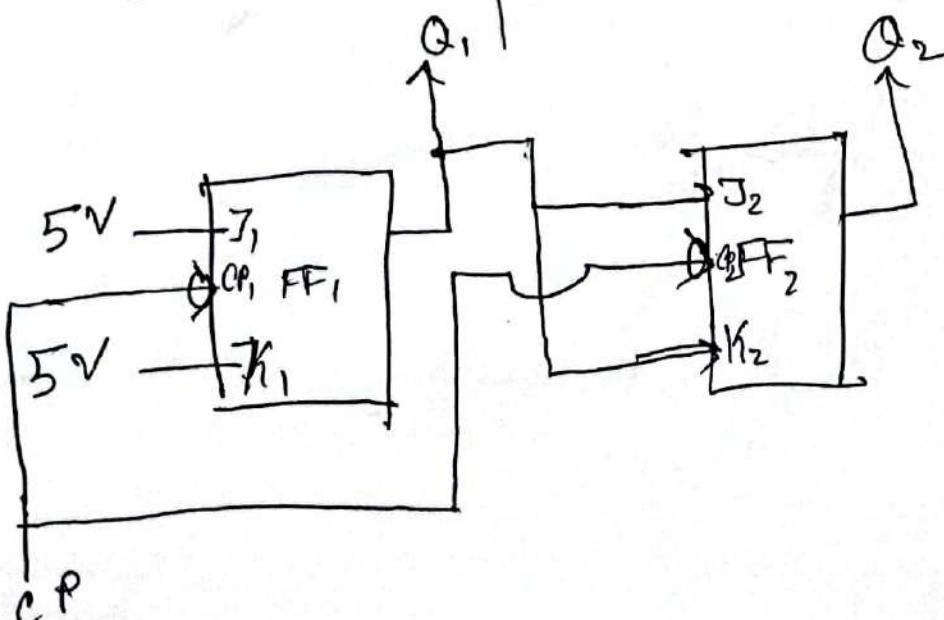
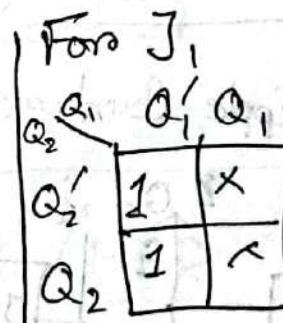
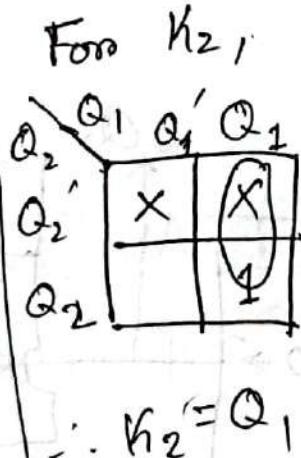
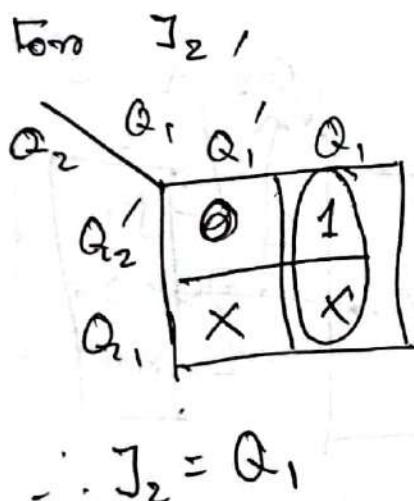
rule table
for every
UNIQUE
combination/state

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_F
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

FF2



C-19 (W-11)

Answers

Asynchronous Counters

Different Clock Pcs

* Also known as Ripple Counter

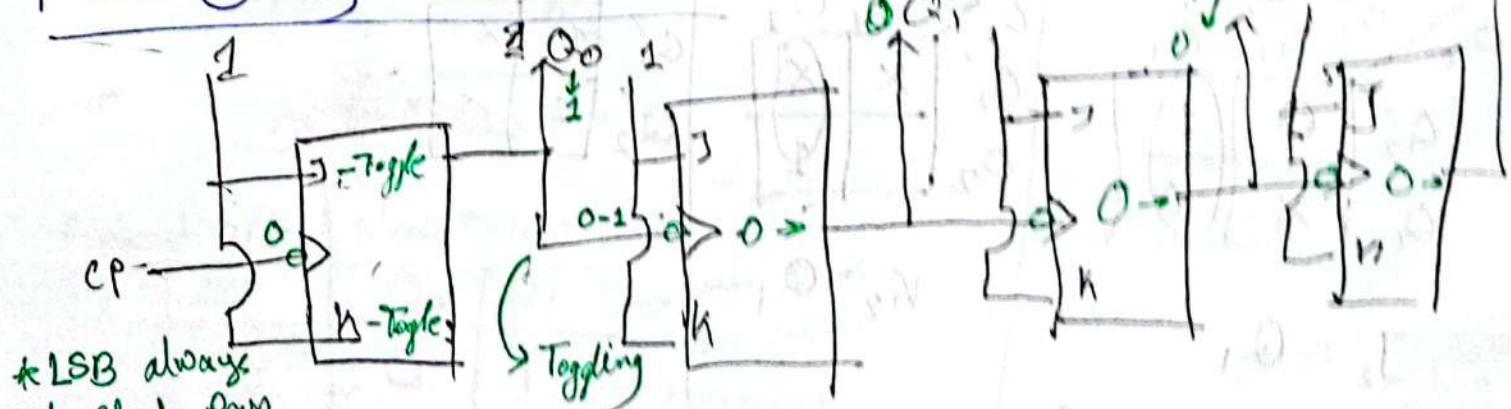
* Different Clock Pcs in FFs

→ Slower

- Not necessarily the same
- Use different method to solve
- Working Principle is necessary

Binary Ripple Counter : (slide is incomplete)

4 bit Binary Ripple Counter:

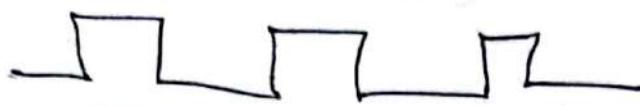


J.K Timing

J	K	Q _{t+1}
0	0	Q _{t+1}
0	1	0
1	0	1
1	1	Q _t

* Timing Diagram: use scale

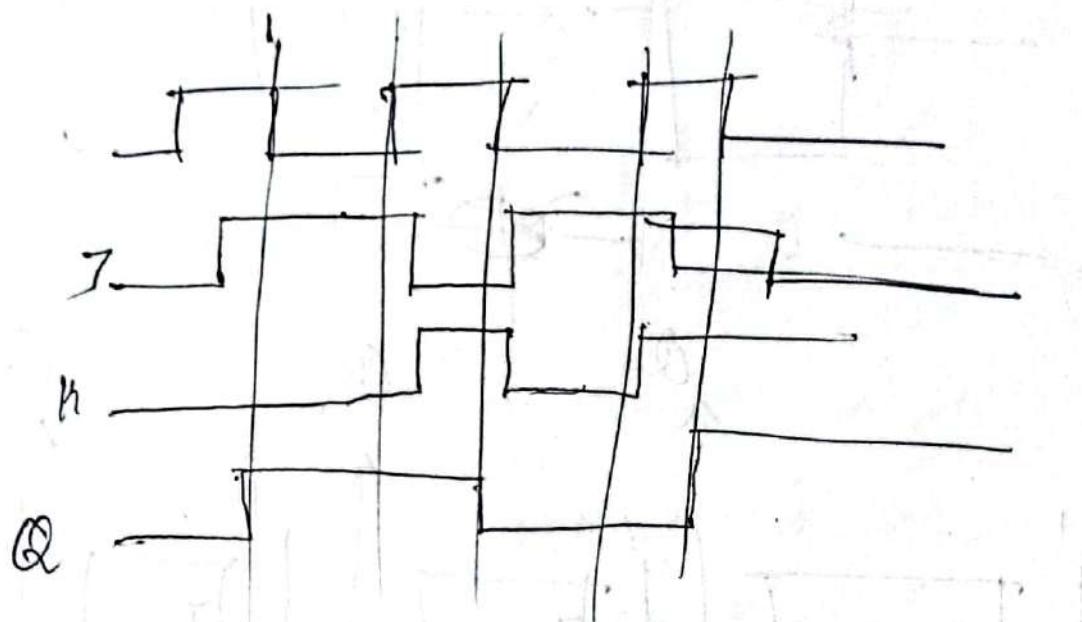
→ Draw CP



→ Identify where changes
changes

→ change
accordingly

→ Check
J and K

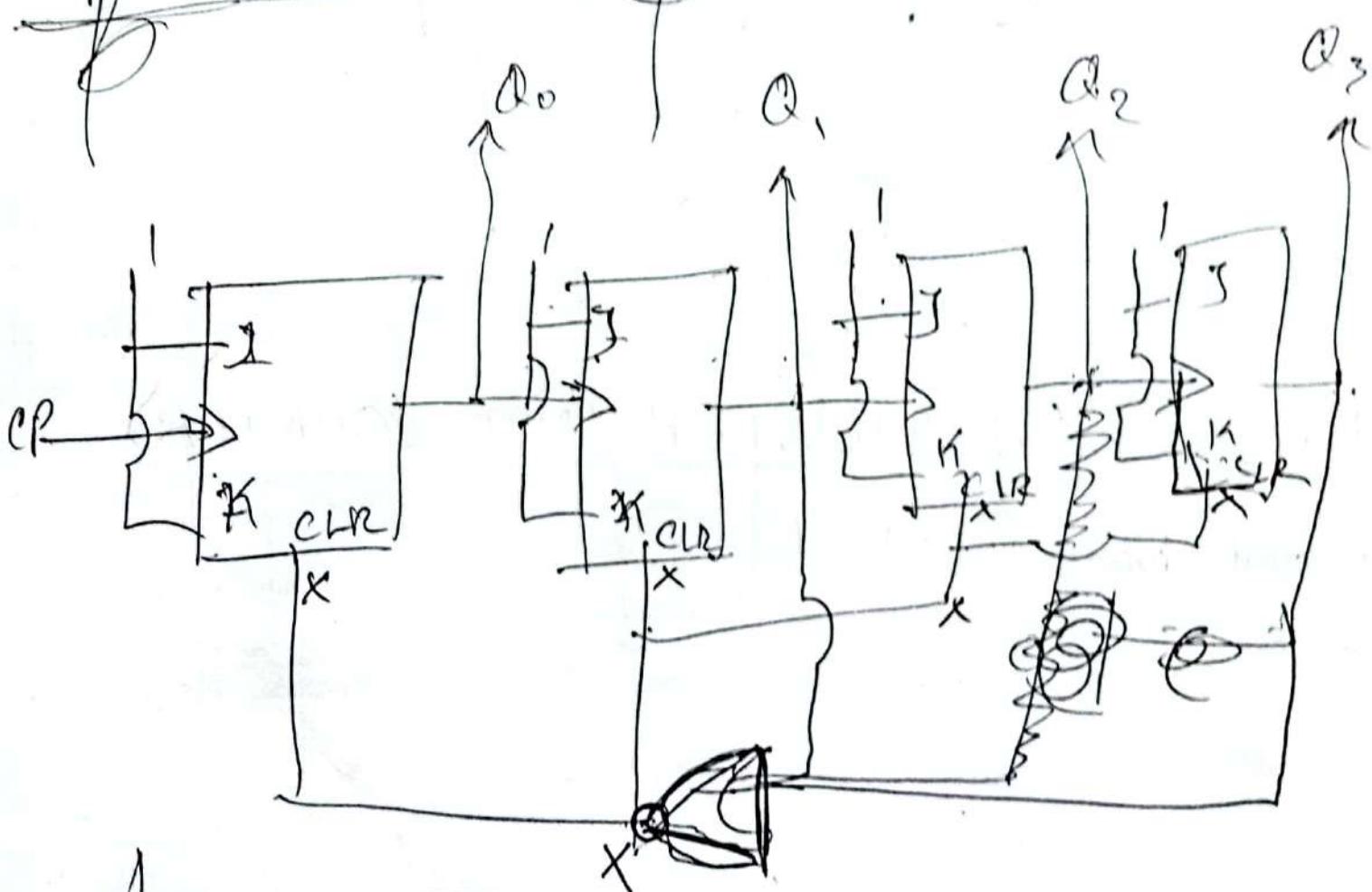
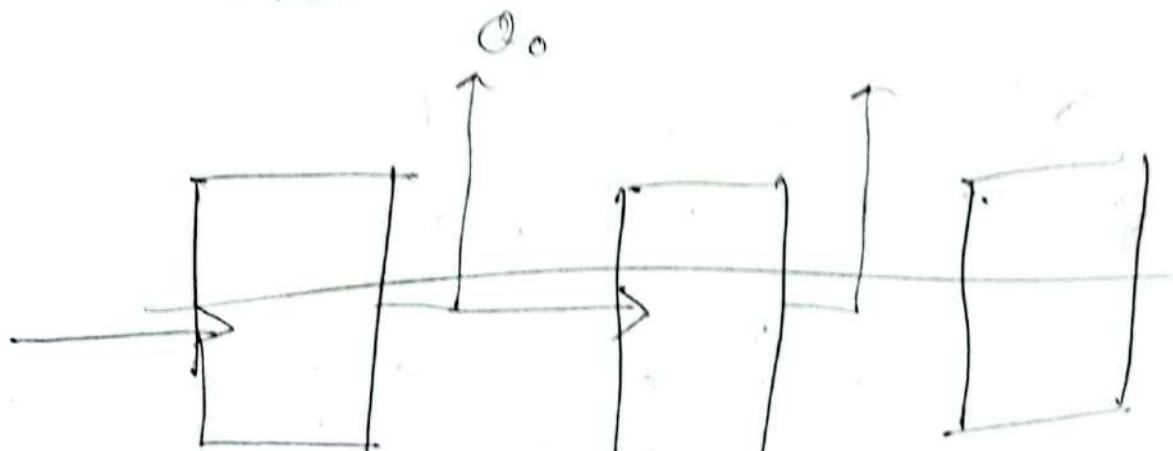


* MODULO N COUNTER (MOD COUNTER)

* Counts from 0 to $N - 1$

* MOD-10 Counter

$Q_3\ Q_2\ Q_1\ Q_0$



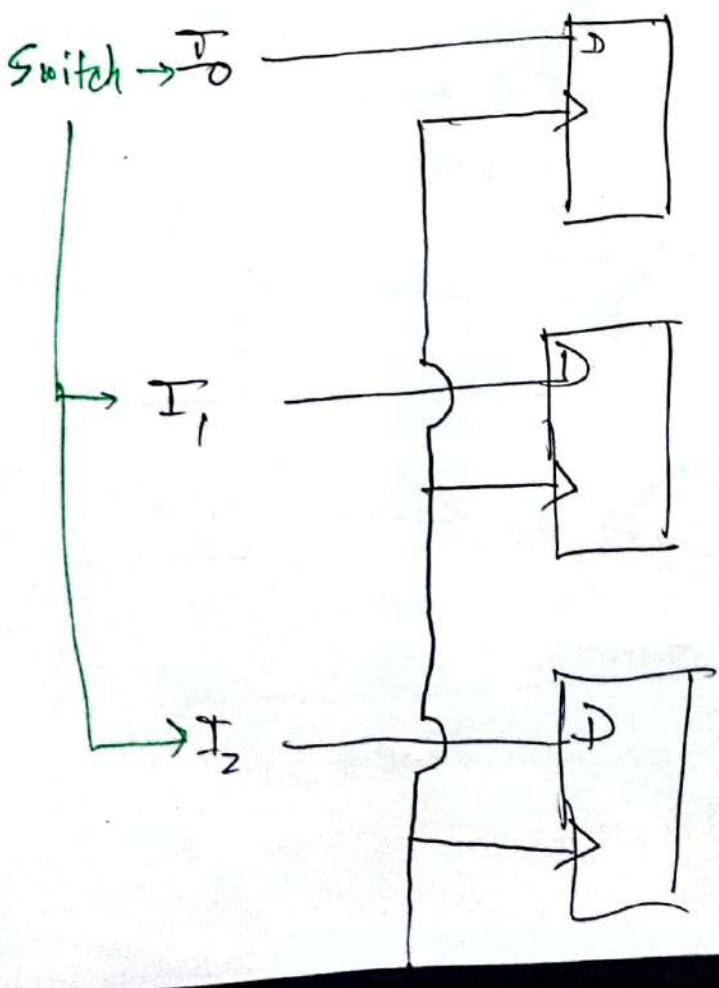
1000
1010
1011
10.

* Self Study: Do the decade counter maths

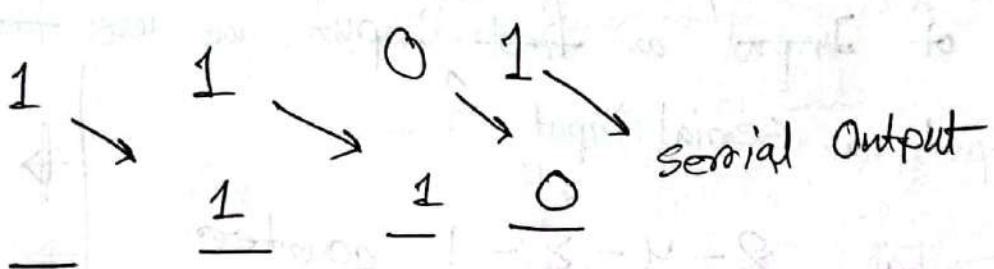
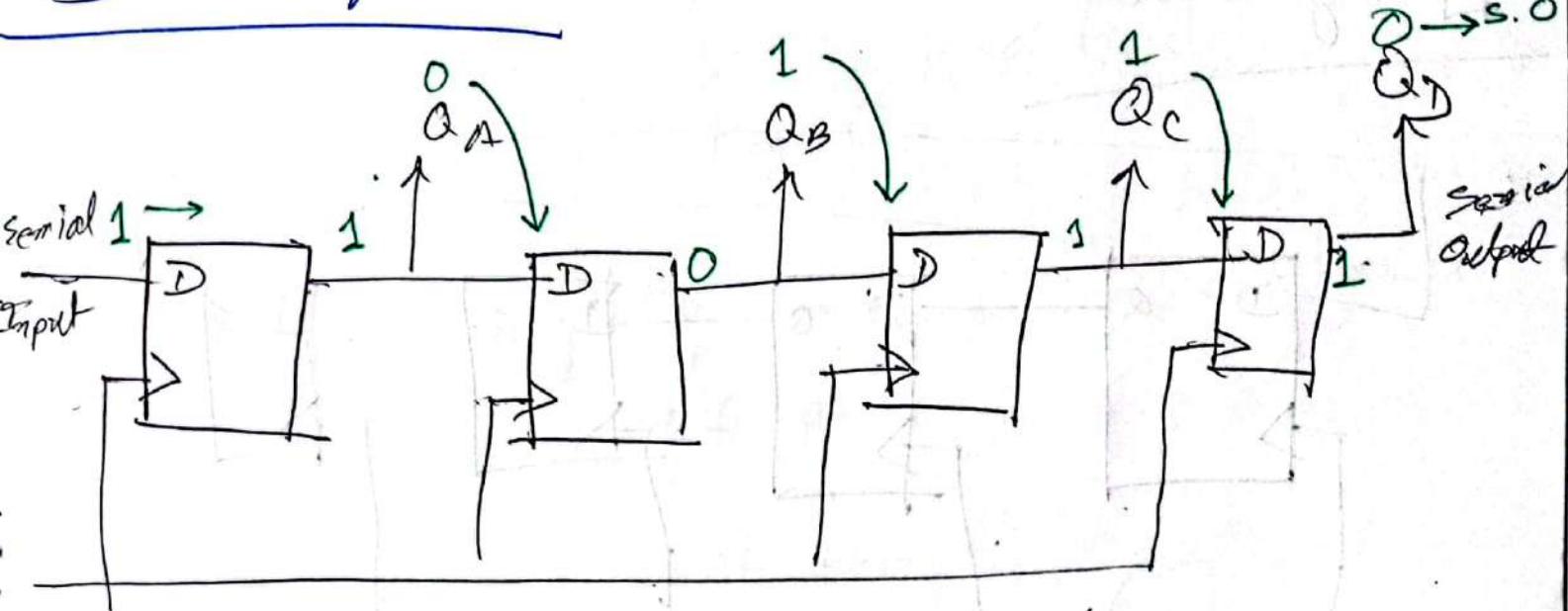
C-20(10-12)

07/07/24

Registers → will be in quiz-4



Shift Registers :



Serial Input

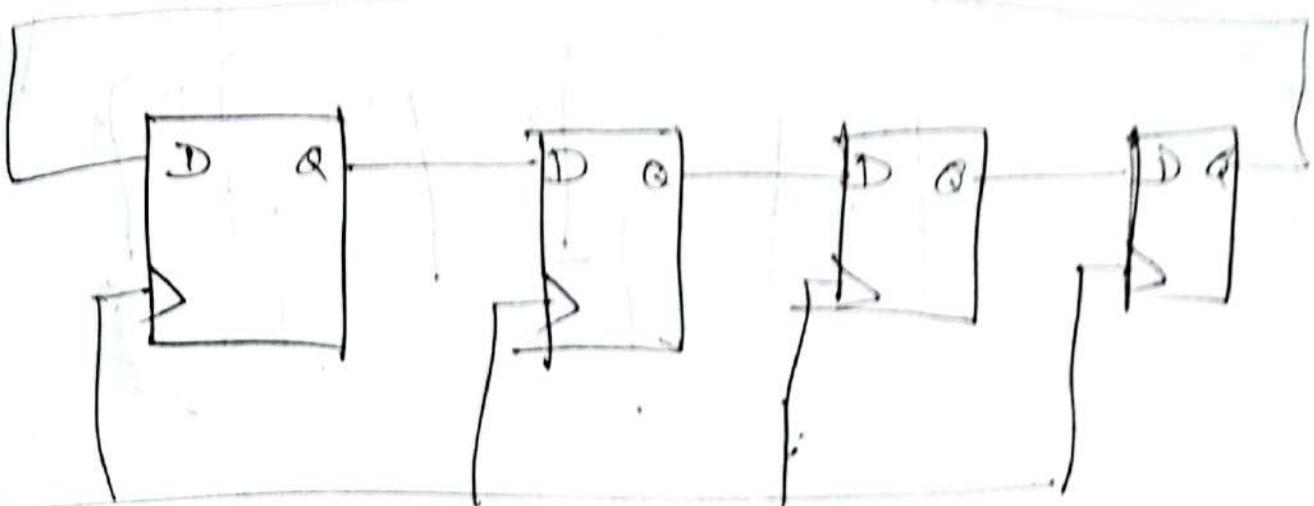
* Right shifting $x = \lfloor \frac{x}{2} \rfloor$

+ Left shifting $x = 2x$

* Serial Input can be whatever we want



Ring Counter:

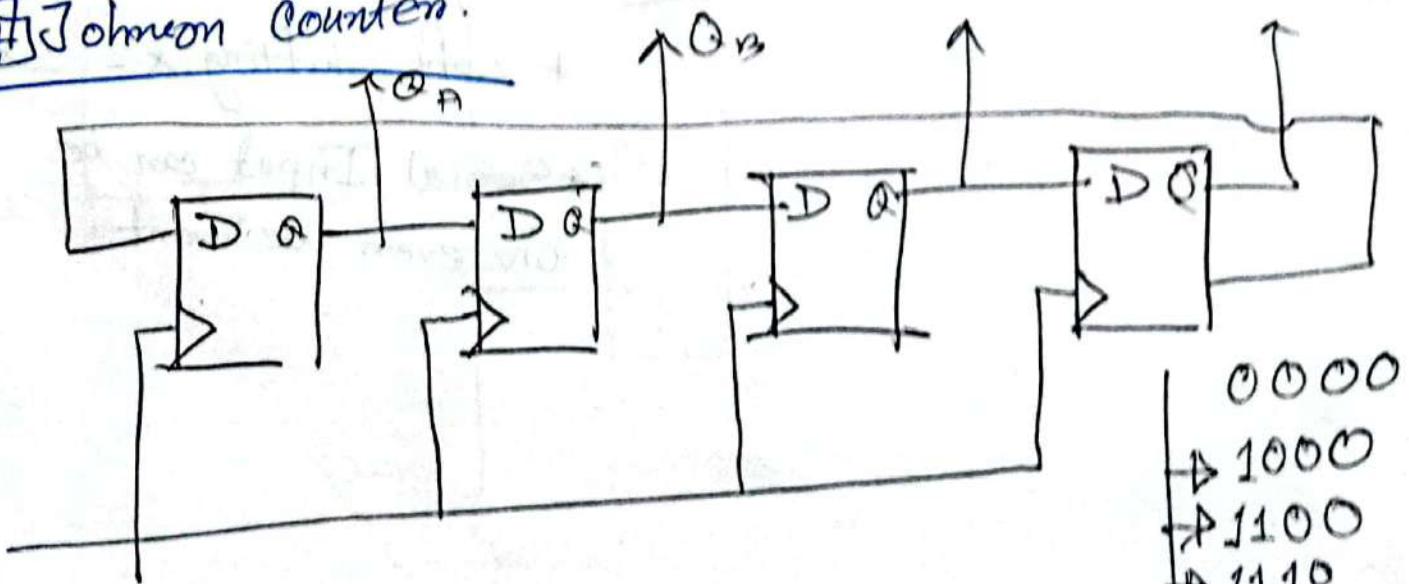


* Instead of ^{switch} Input as first ^{serial} input, we use last out serial output as serial input.

* 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

$S_1 \quad S_0$

Reg Op

Table not fixed

0 0

No change → Gives old result

0 1

Shift Right

1 0

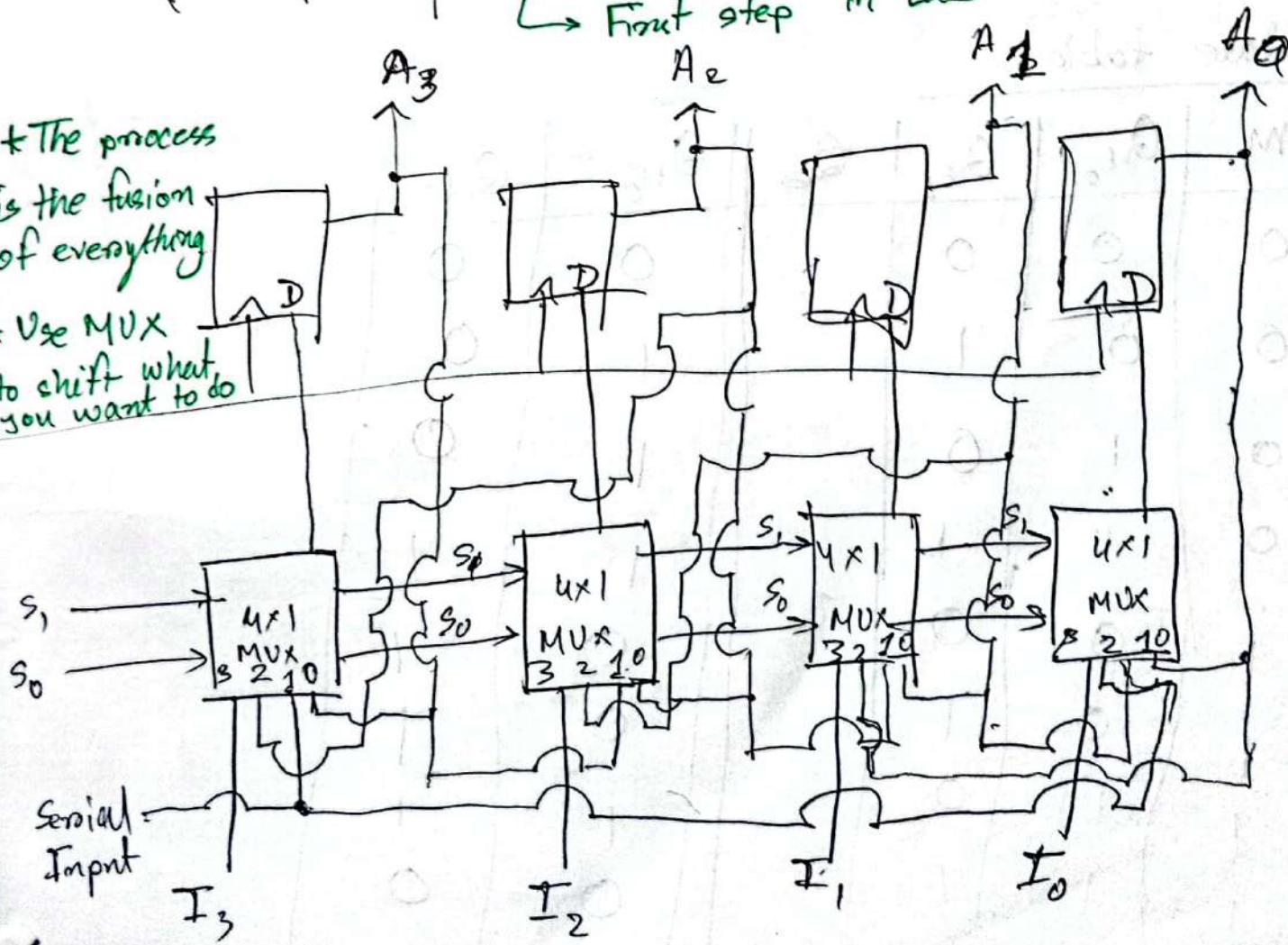
Shift Left

1 1

Parallel Load → Stores new value
↳ First step in Lab

* The process
is the fusion
of everything

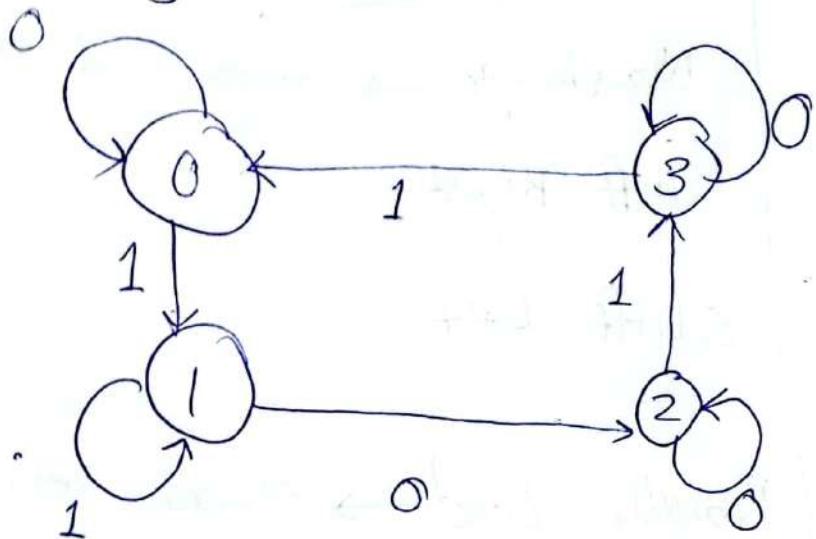
* Use MUX
to shift what
you want to do



C-21 (W-12)

10/07/24

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



State table:

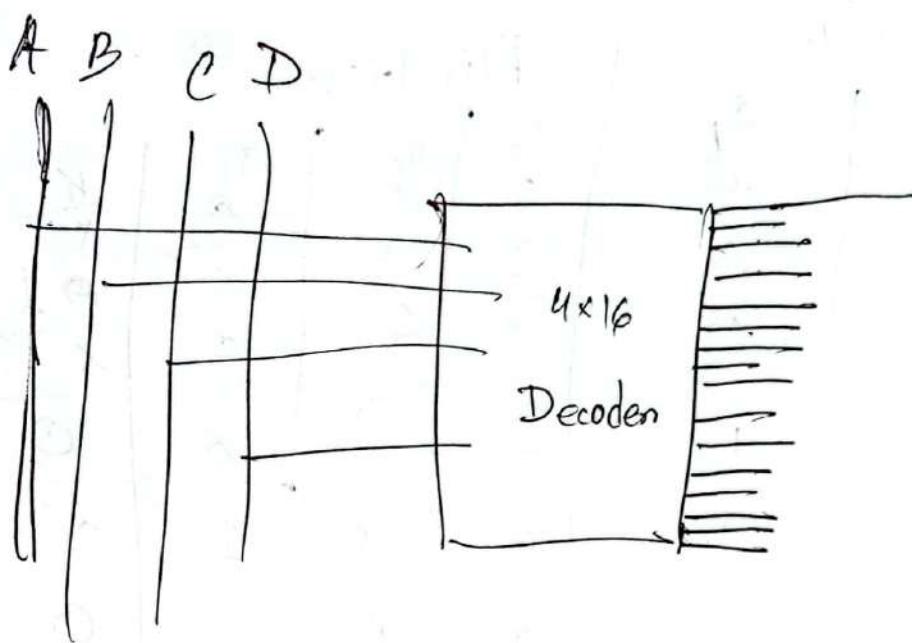
M	Q_{t_0}	Q_{t_1}	Q_t	Q_{t+0}	Q_{t+1}
0	0	0		0	0
0	0	1		1	0
0	1	0		1	1
0	1	1		1	1
i	0	0		0	1
i	0	1		0	1
1	1	0		1	0
1	1	1		0	1

Excitation Table :

Switch	Present State		Next State		Flip Flop	
	M	Q_0	Q_1	Q_0	Q_1	
0	0	0	0	0	0	0
0	0	0	1	1	0	1
0	1	0	0	1	0	0
0	1	1	1	1	1	1
1	0	0	0	1	0	1
1	0	1	0	1	0	0
1	1	0	1	1	1	1
1	1	1	0	0	1	1

Flip Flop

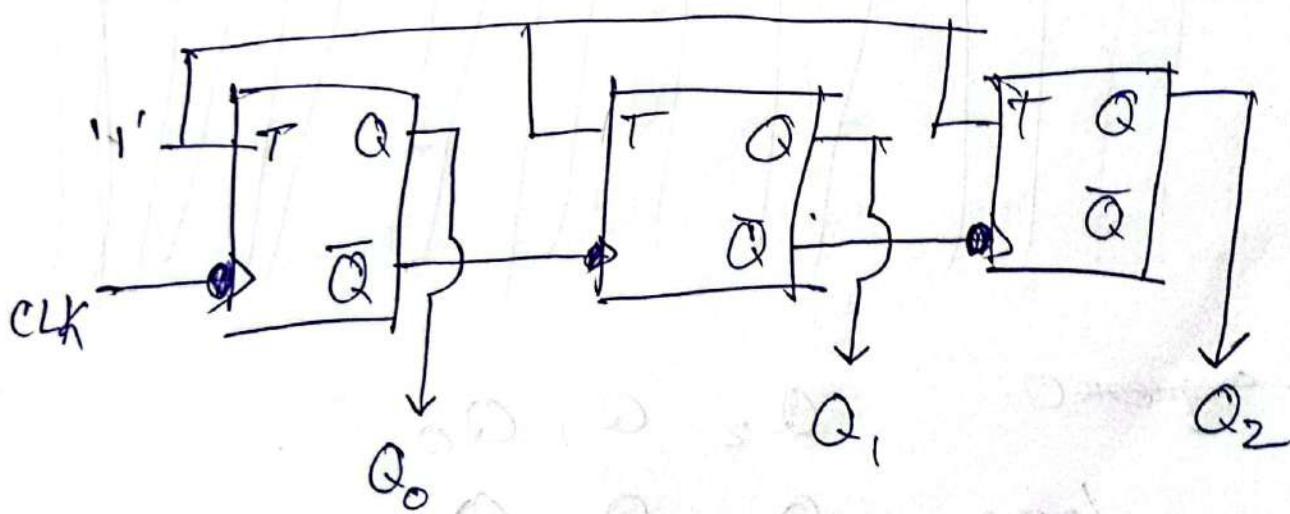
M	Q_0, Q_1	1	X	X
		X	X	



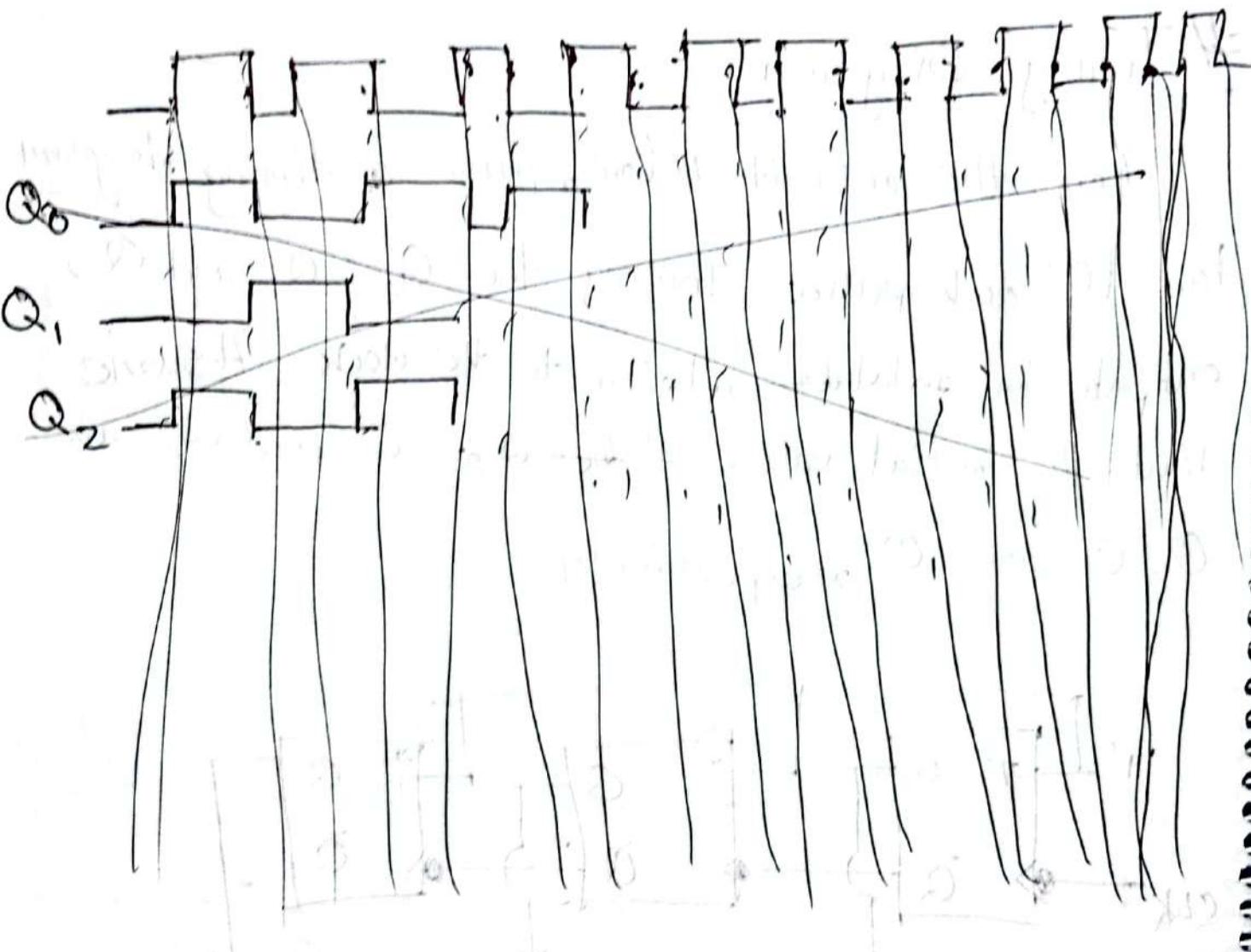
J_m	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}	J_{11}	J_{12}	J_{13}	J_{14}	J_{15}	J_{16}	out
A	B	C	D	P	Q	R	S										
0	0	0	0	0	0	0	0										
0	0	0	1	0	0	0	0										
0	0	1	0	0	0	0	0										
0	0	1	1	0	0	0	0										
0	0	1	1	0	0	1	0										
0	1	0	0	0	0	1	0										
0	1	0	1	0	0	1	0										
0	1	1	1	0	0	0	0										
0	1	1	1	1	0	0	0										
0	1	0	0	0	0	0	0										
1	0	0	0	1	0	0	0										
1	0	0	1	0	1	0	0										
1	1	0	0	0	0	0	0										
1	1	0	1	0	0	0	0										
1	1	1	0	0	0	0	0										
1	1	1	1	0	0	0	0										

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 ~~relative~~ 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.

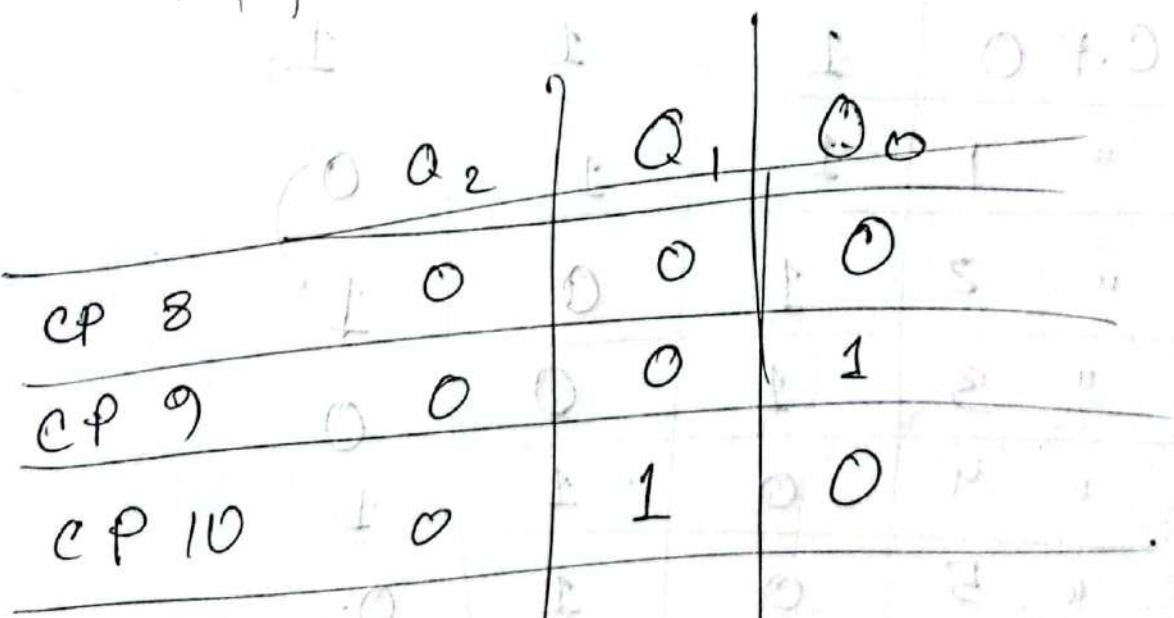
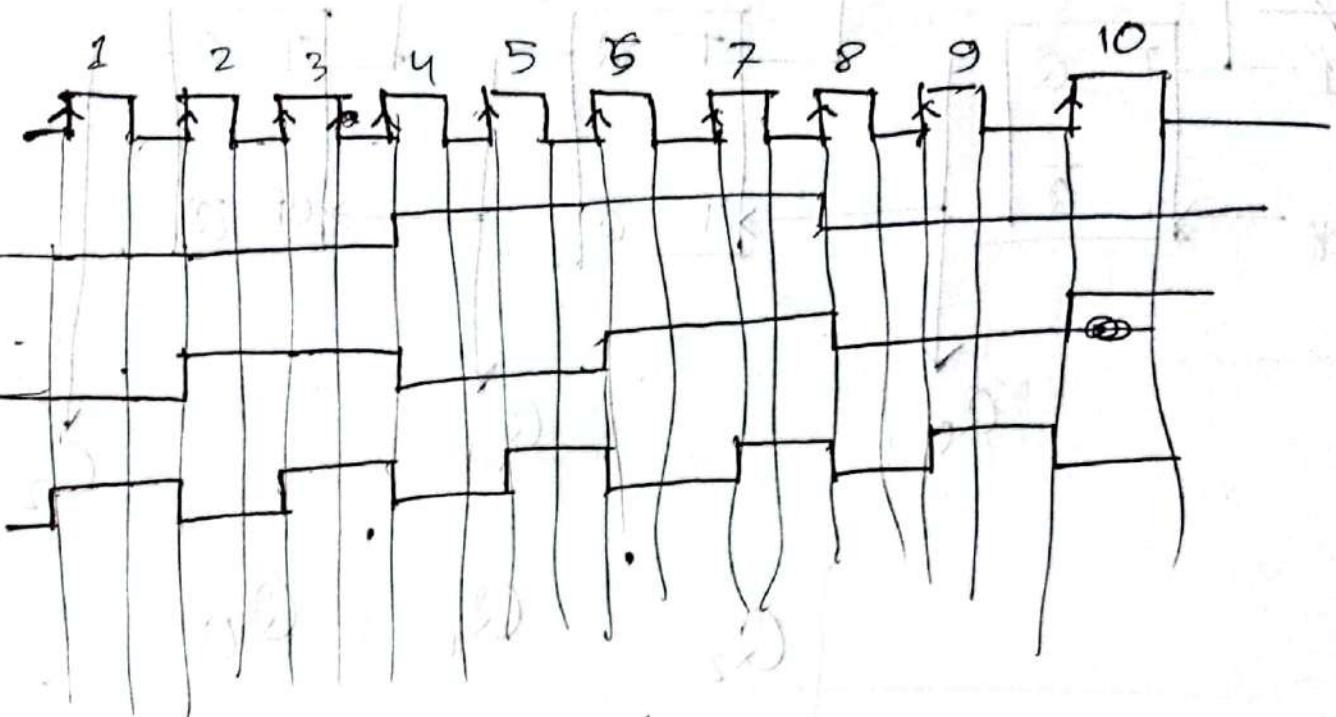


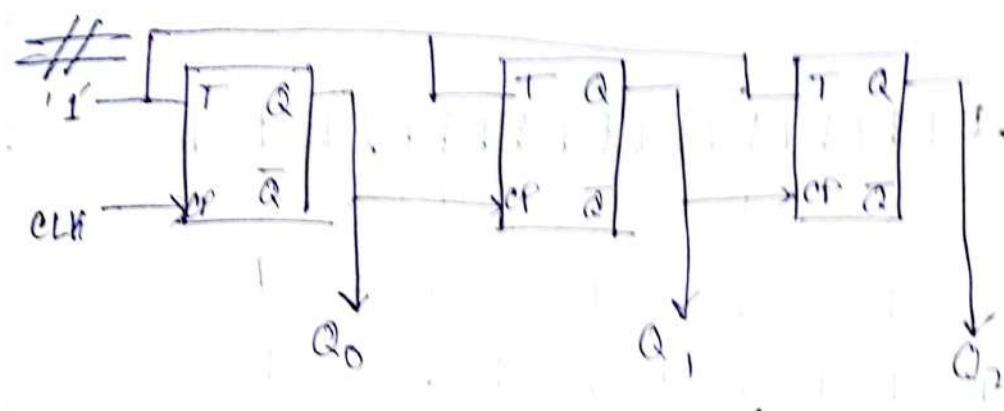
Output sequence

$Q_2 \quad Q_1 \quad Q_0$

CLK Initial /CP.O :

CP 1	0	0	0
CP 2	0	1	0
CP 3	0	0	1
CP 4	1	0	0
CP 5	1	0	1
CP 6	1	1	0
CP 7	0	1	1





C.P O	Q_2	Q_1	Q_0
" 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

