

# UNIT-IV

## Combinational Logic Circuits:

→ Basic Theorems and properties of Boolean Algebra:

Boolean Algebra is a set of rules, laws and Theorems by which logical operations can be mathematically expressed.

### Rules:

- 1) we will use capital letters for representing variables and functions of variables
- 2) Binary 1 will represent a HIGH level, and binary 0 will represent a Low level in Boolean equations.
- 3) The complement of a variable is represented by a "bar" over a letter.  
 ex:  $A = 1, \bar{A} = 0$  ; and if  $A = 0, \bar{A} = 1$   
 complement of A can be written as  $A'$ .
- 4) The logical AND function of two variables is represented as  $A \cdot B$  (or)  $AB$ .
- 5) The logical OR function of two variables is represented as  $A + B$  (read as A or B)
- 6) The basic rules for Boolean addition are

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 1$$

6. Basic rules for multiplication.

$$0 \cdot 0 = 0$$

$$0 \cdot 1 = 0$$

$$1 \cdot 0 = 0$$

$$1 \cdot 1 = 1$$

Several basic rules are

Rule No:	Rules:
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	$\overline{\bar{A}} = A$
10	$A + AB = A$
11	$A + \bar{A}B = A + B$
12	$(A + B)(A + C) = A + BC$

## Canonical and standard Forms!

(Minterms and maxterms)

→ Each individual term in standard sum of products (SOP) form is called minterm and.

→ each individual term in standard product of sum form is called maxterm.

→ The concept of minterms and maxterms allows us to introduce a very convenient shorthand notation to express logical functions.

Variables			minterms	max terms
A	B	C	$m_i$	$M_i$
0	0	0	$\bar{A}\bar{B}\bar{C} = m_0$	$A+B+C = M_0$
0	0	1	$\bar{A}\bar{B}C = m_1$	$A+B+\bar{C} = M_1$
0	1	0	$\bar{A}B\bar{C} = m_2$	$A+\bar{B}+C = M_2$
0	1	1	$\bar{A}BC = m_3$	$A+\bar{B}+\bar{C} = M_3$
1	0	0	$A\bar{B}\bar{C} = m_4$	$\bar{A}+B+C = M_4$
1	0	1	$A\bar{B}C = m_5$	$\bar{A}+B+\bar{C} = M_5$
1	1	0	$AB\bar{C} = m_6$	$\bar{A}+\bar{B}+C = M_6$
1	1	1	$ABC = m_7$	$\bar{A}+\bar{B}+\bar{C} = M_7$

Table: minterms and maxterms for three variables

## Distributive Law!

(2)

$$1) A(B+C) = AB+AC$$

$$2) A+BC = (A+B)(A+C)$$

## Associative laws!

$$1) A+(B+C) = (A+B)+C = A+B+C$$

$$2) A.(B.C) = (A.B).C = A.B.C$$

## Commutative law!

$$1) A+B = B+A$$

$$2) A.B = B.A$$

## AND laws!

$$A \cdot 0 = 0$$

$$A \cdot 1 = A$$

$$A \cdot A = A$$

$$A \cdot \bar{A} = 0$$

## OR Laws!

$$A+0 = A$$

$$A+1 = 1$$

$$A+A = A$$

$$A+\bar{A} = 1$$

## NOT Laws!

$$\overline{\bar{A}} = A$$

## DeMorgan's Theorems!

$$1) \overline{AB} = \bar{A} + \bar{B}$$

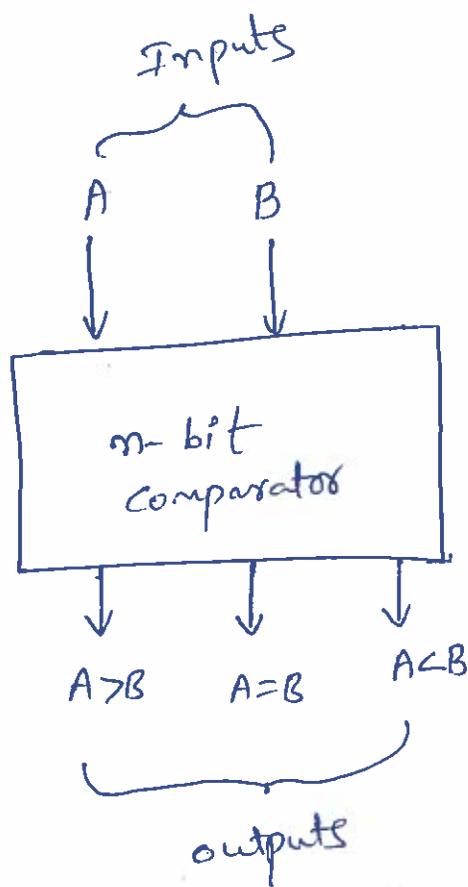
$$2) \overline{A+B} = \bar{A} \bar{B}$$

## UNIT-4

### Combinational Logic Circuits

### Magnitude Comparator (or) Digital Comparator:

A comparator is a special combinational circuit designed primarily to compare the relative magnitude of two binary numbers. Fig (1) shows the block diagram of an  $n$ -bit comparator. It receives two  $n$ -bit numbers  $A$  and  $B$  as inputs and the outputs are  $A > B$ ,  $A = B$  and  $A < B$ . Depending upon the relative magnitudes of the two numbers, one of the outputs will be high.



Fig(1): Block diagram of  $n$ -bit comparator.

Ex: Explain the working of 2-bit magnitude comparator?

The truth table for 2-bit comparator is given in table

Inputs				Outputs		
A <sub>1</sub>	A <sub>0</sub>	B <sub>1</sub>	B <sub>0</sub>	A > B	A = B	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	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	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	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

# K-map Simplification:

for  $A > B$

		$B_1 B_0$			
		00	01	11	10
$A_1 A_0$	00	0	0	0	0
	01	1	0	0	0
	11	1	1	0	1
	10	1	1	0	0

$$A > B = A_0 \bar{B}_1 \bar{B}_0 + A_1 \bar{B}_1 + A_1 A_0 \bar{B}_0$$

for  $A = B$

		$B_1 B_0$			
		00	01	11	10
$A_1 A_0$	00	1	0	0	0
	01	0	1	0	0
	11	0	0	1	0
	10	0	0	0	1

$A = B =$

$$\begin{aligned} & \bar{A}_1 \bar{A}_0 \bar{B}_1 \bar{B}_0 + \bar{A}_1 A_0 \bar{B}_1 B_0 + A_1 A_0 B_1 B_0 + A_1 \bar{A}_0 B_1 \bar{B}_0 \\ & = \bar{A}_1 \bar{B}_1 (\bar{A}_0 \bar{B}_0 + A_0 B_0) + A_1 B_1 (A_0 B_0 + \bar{A}_0 \bar{B}_0) \\ & = (A_0 \odot B_0) (A_1 \odot B_1) \end{aligned}$$

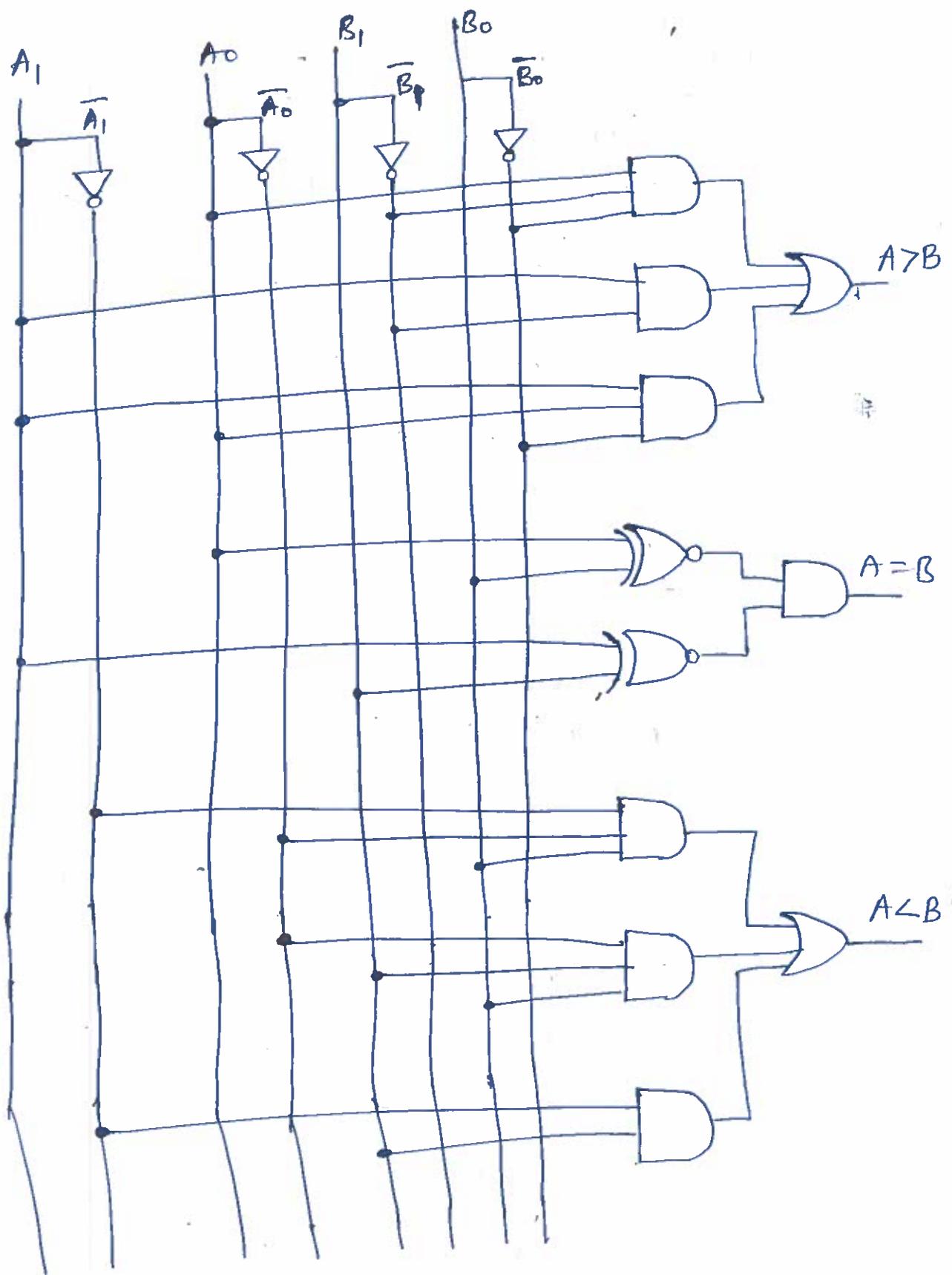
for  $A < B$

		$B_1 B_0$			
		00	01	11	10
$A_1 A_0$	00	0	1	1	1
	01	0	0	1	1
	11	0	0	0	0
	10	0	0	1	0

$A < B =$

$$\bar{A}_1 \bar{A}_0 B_0 + \bar{A}_0 B_1 B_0 + \bar{A}_1 B_1$$

Logic diagram:



## Decoders:

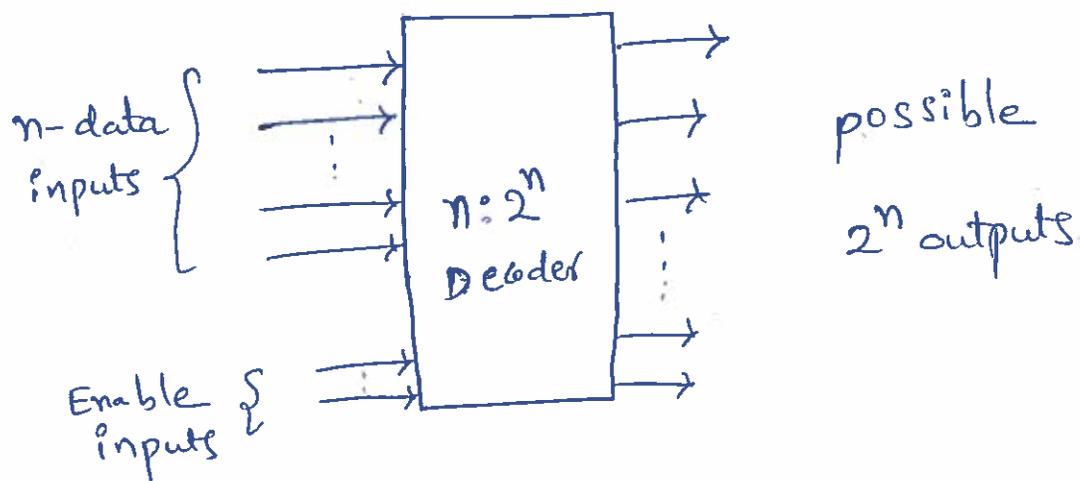


fig. General structure of decoder.

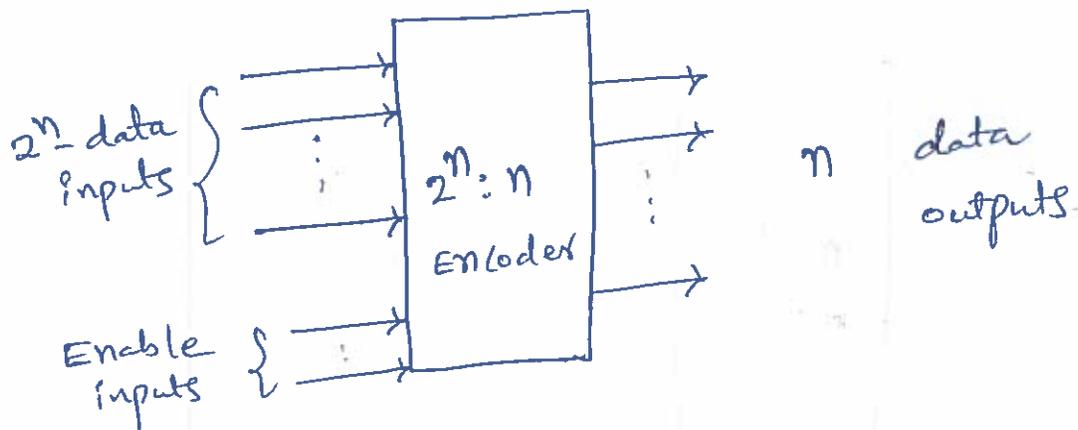
- The decoder is the opposite of the encoder
- This circuit converts binary data into other form i.e. decimal, octal, hexadecimal.
- Decoder is a combinational logic circuit
- Decoder is having a  $n$ -input lines and  $2^n$ -output lines.

ex: 2 input line gives  $2^2 = 4$  outputs

Similarly 3 input line gives  $2^3 = 8$  outputs

" 4 input line gives  $2^4 = 16$  outputs

## Encoders:



→ An encoder is a digital circuit that performs the inverse operation of a decoder.

→ Encoder has  $2^n$  input lines and  $n$  output lines.

→ The encoder circuit is used to convert one form of data into binary form.

### Encoder Applications:

1. Decimal to Binary Converters (BCD)
2. Octal to Binary Converters
3. Hexadecimal to Binary Converters
4. Used to drive 7 segment displays

### Decoder Applications:

1. BCD to Decimal Conversion
2. Binary to Octal Conversion
3. Hexadecimal to Binary Conversion
4. BCD to Gray or Gray to BCD Conversion.

Ex: Draw the logic circuit of a 3 to 8 decoder and explain its working.

Fig: Shows the 3-to-8 line decoder. Here 3 inputs are decoded into eight outputs, each output represent one of the minterms of the 3-input variables.

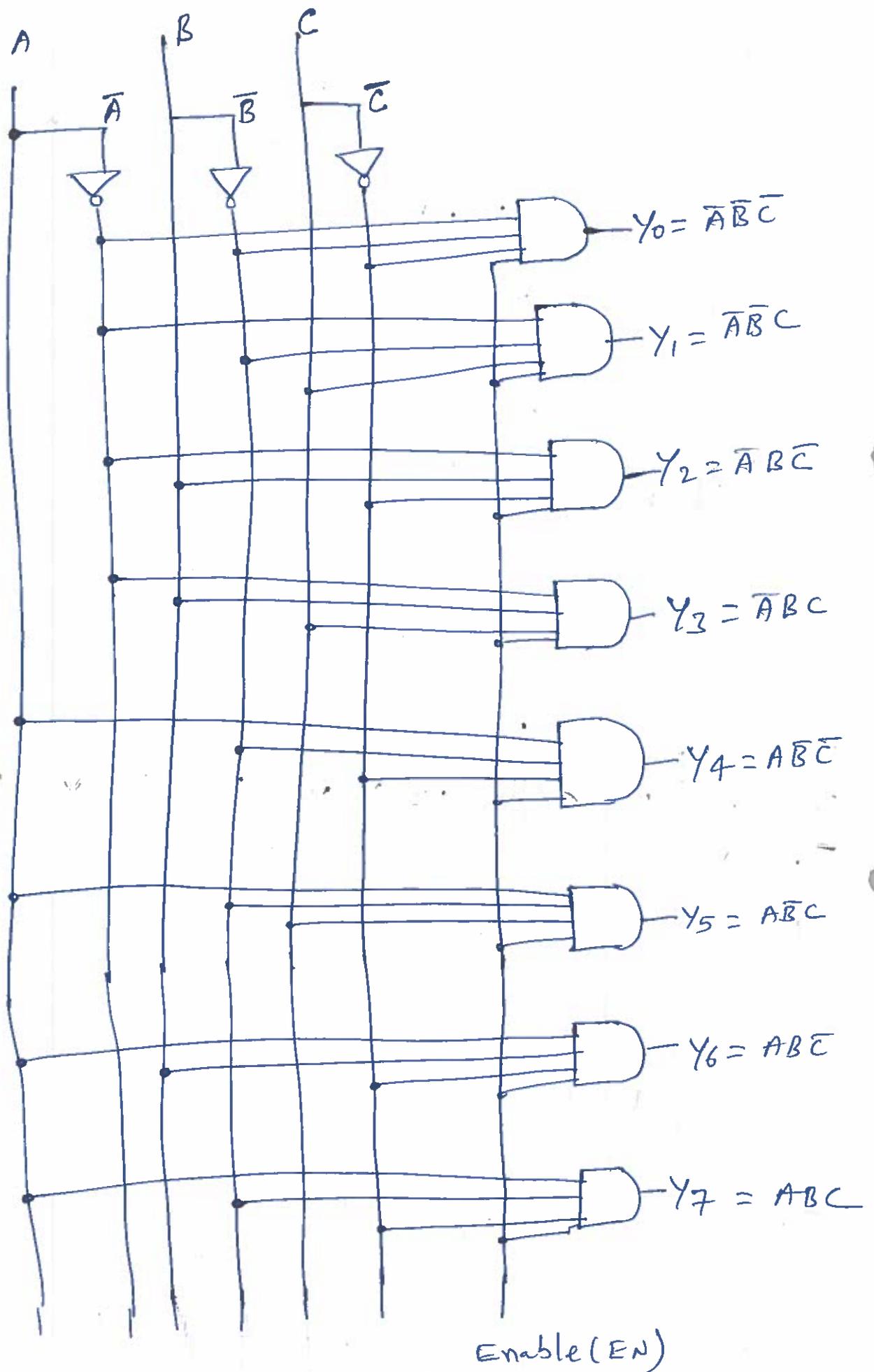
The three inverters provide the complement of the inputs, and each one of the eight AND gates generates one of the minterms. Enable input is provided to activate decoded outputs based on data inputs A, B, and C.

The table shows the truth table for 3 to 8 decoder.

Inputs				Outputs							
EN	A	B	C	Y <sub>7</sub>	Y <sub>6</sub>	Y <sub>5</sub>	Y <sub>4</sub>	Y <sub>3</sub>	Y <sub>2</sub>	Y <sub>1</sub>	Y <sub>0</sub>
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Truth table for a 3-to-8 decoder.

Logic diagram:



## Multiplexers:

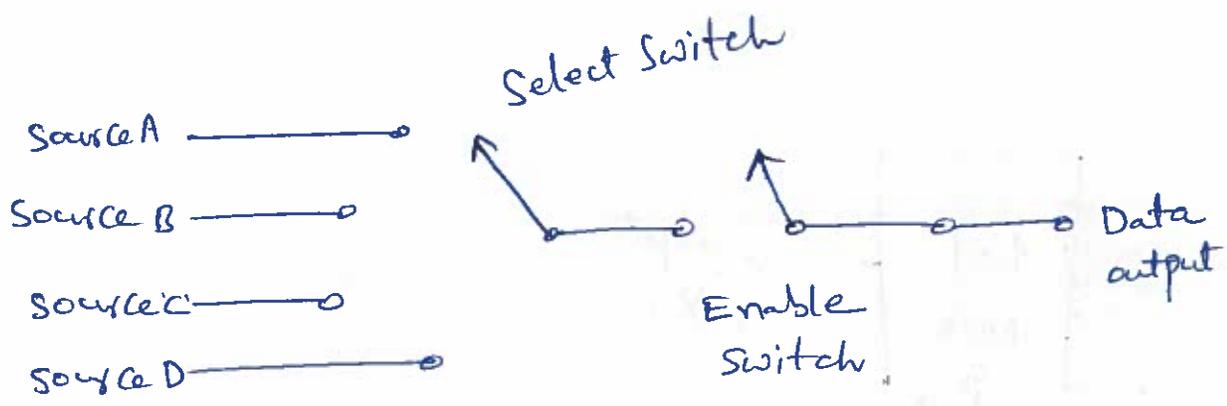


fig: ~~Analog~~ multiplexer switch.

→ multiplexer is a digital switch.

→ It allows digital information from several sources to be routed onto a single output line as shown in fig.

→ The basic multiplexer has several data-input lines and a single output line.

→ The selection of a particular input line is controlled by a set of selection lines.

→ There are  $2^n$  input lines and  $n$ -selection lines whose bit combinations determine which input is selected.

### Applications of multiplexer:

These are used to transmit a selected data at output.

Ex: 4-to-1 line multiplexer:

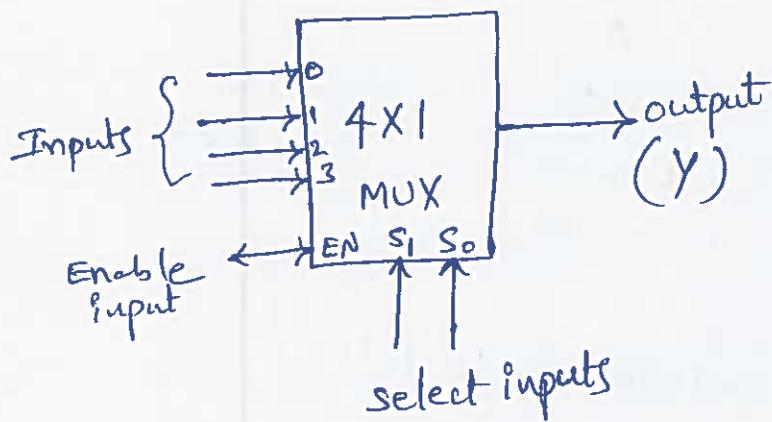


fig: Logic symbol

$S_1$	$S_0$	$Y$
0	0	$D_0$
0	1	$D_1$
1	0	$D_2$
1	1	$D_3$

fig: Function table

for example, when ~~0~~  $S_1, S_0 = 01$ , the AND gate associated with data input  $D_1$  has two of its inputs equal to 1 and the third input connected to  $D_1$ . The other three AND gates have at least one input equal to 0, which makes their outputs equal to 0. The OR gate output is now equal to the value of  $D_1$ . Thus we can say, data bit  $D_1$  is routed to the output when  $S_1, S_0 = 01$ .

Logic Diagram:

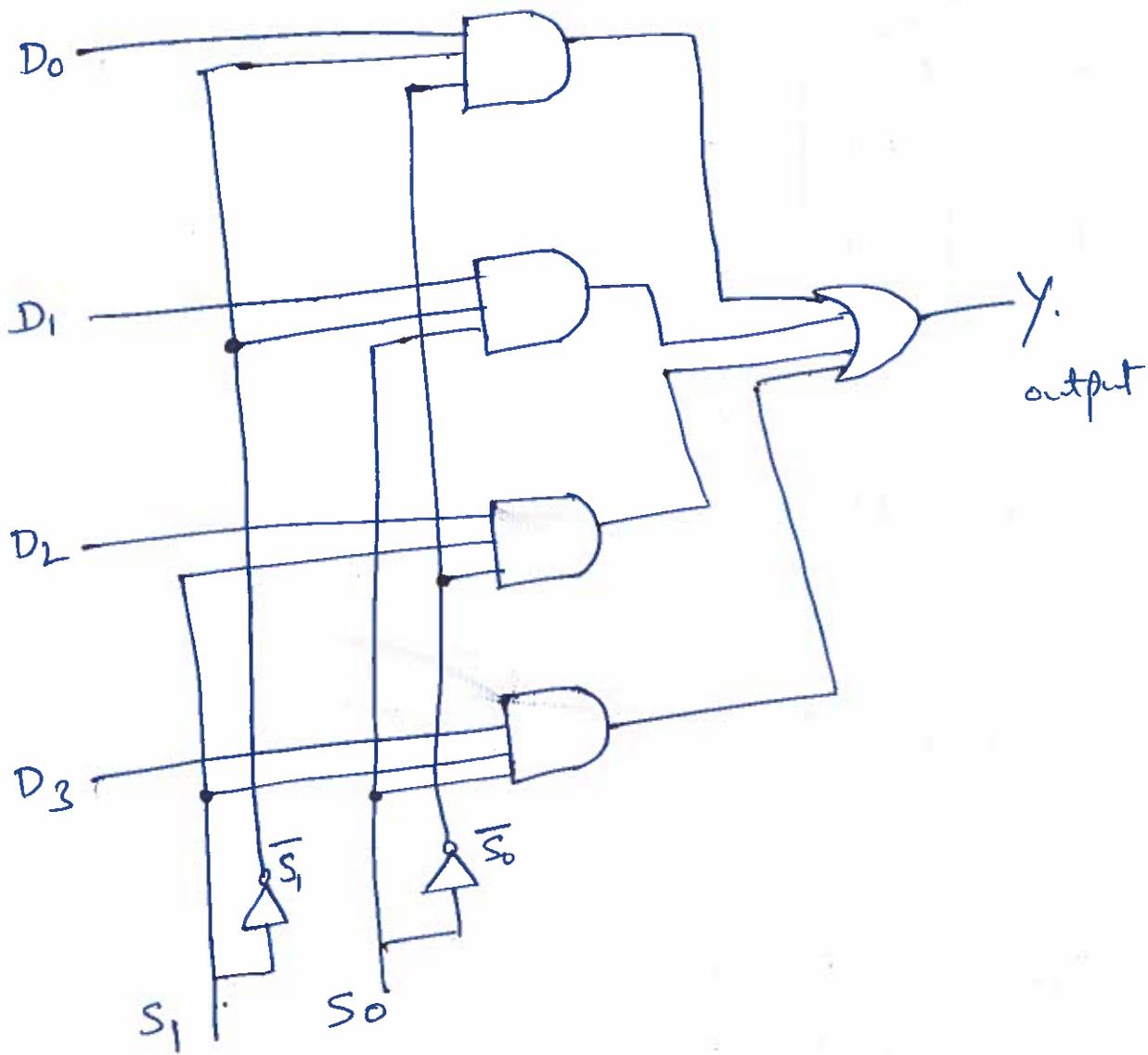


Fig: logic Diagram.

ex: Draw the  $2 \times 1$  multiplexer.

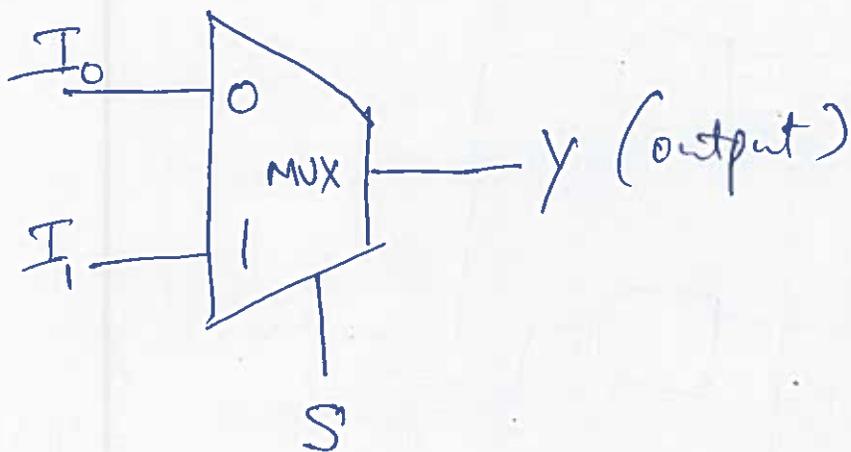
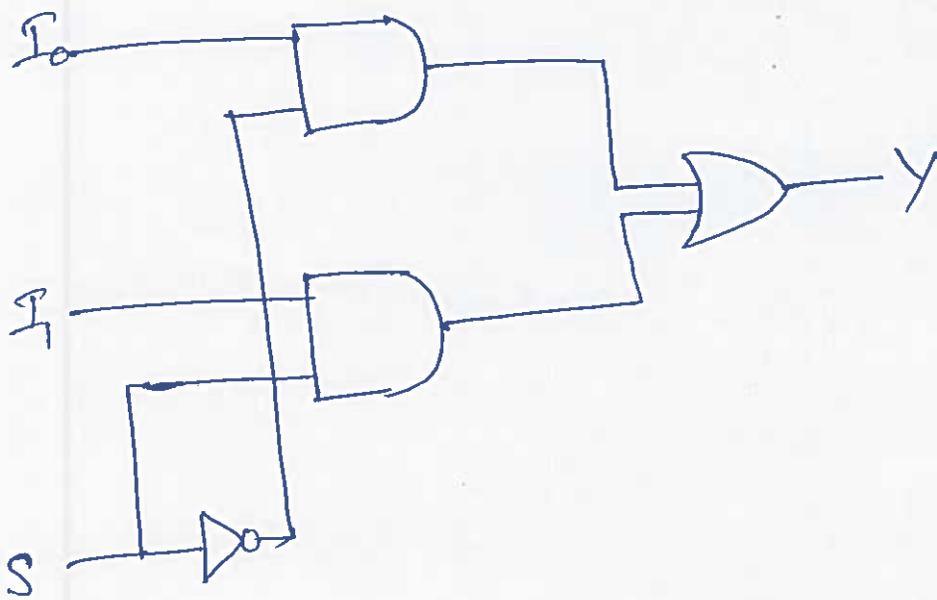


fig: block diagram

logic diagram:



$S \rightarrow$  selection line

## Address:

Digital Computers perform various arithmetic operations. The most basic operation, no doubt, is the addition of two binary digits. This simple addition consists of four possible elementary operations, namely

$$0+0 = 0$$

$$0+1 = 1$$

$$1+0 = 1$$

$$1+1 = 10_2$$

The first three operations produce a sum whose length is one digit, but when the last operation is performed sum is two digits.

The higher significant bit of this result is called a carry and lower significant bit is called sum. The logic circuit which performs

this operation is called a ~~half adder~~ half adder. The circuit which

performs addition of three bits is a full adder.

## Half adder:

The half-adder operation needs two binary inputs, and two binary outputs. Sum and Carry.

Inputs		Outputs	
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth table for half-adder

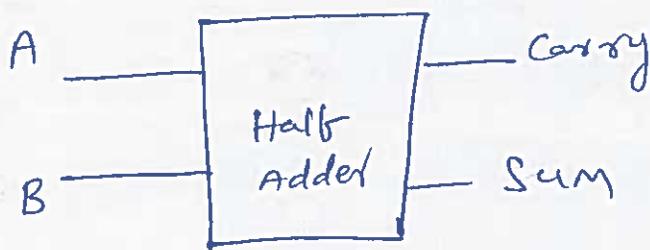
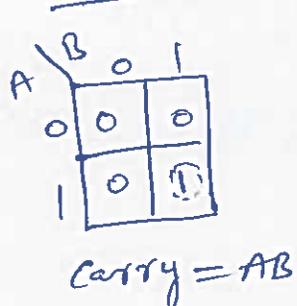


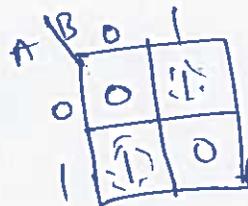
Fig: Block schematic of half-adder

K-map simplification for Carry and Sum

For Carry

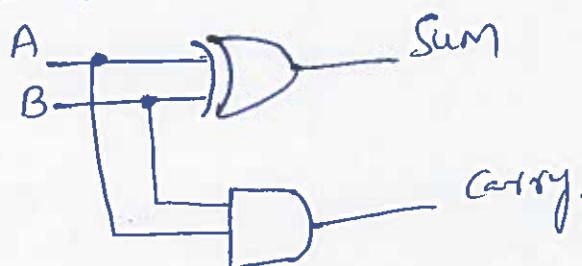


for Sum



$$\begin{aligned} \text{Sum} &= A\bar{B} + \bar{A}B \\ &= A \oplus B \end{aligned}$$

logic diagram:



## Full-Adder:

It consists of three inputs and two outputs. Two of the input variables, denoted by  $A$  and  $B$ , represent the two significant bits to be added. The third input  $C_{in}$ , represents the carry from the previous lower significant position. The truth table for full-adder is shown in Table

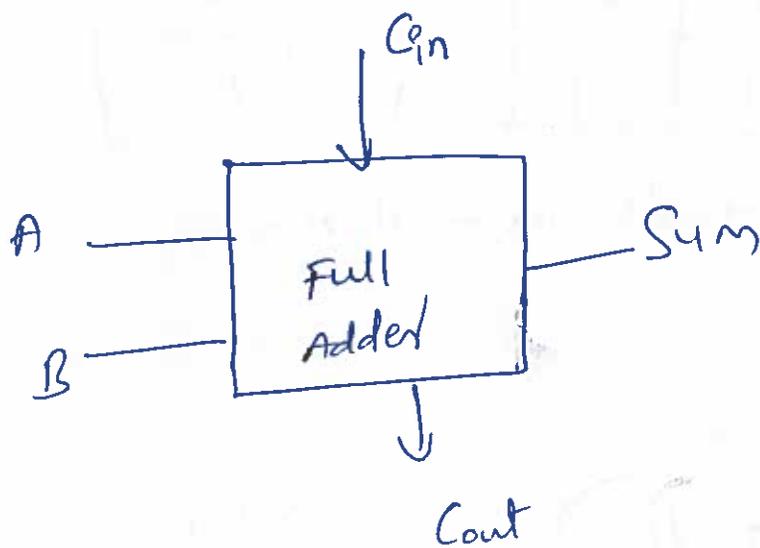


Fig: Block schematic of full adder.

K-map simplification for carry and sum.

For carry (Cout)

	$C_{in}$			
$A$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

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

For Sum

	$C_{in}$			
$A$	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$Sum = \bar{A}\bar{B}C_{in} + \bar{A}BC_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$

Inputs			Outputs	
A	B	C <sub>in</sub>	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Truth table for full Adder.

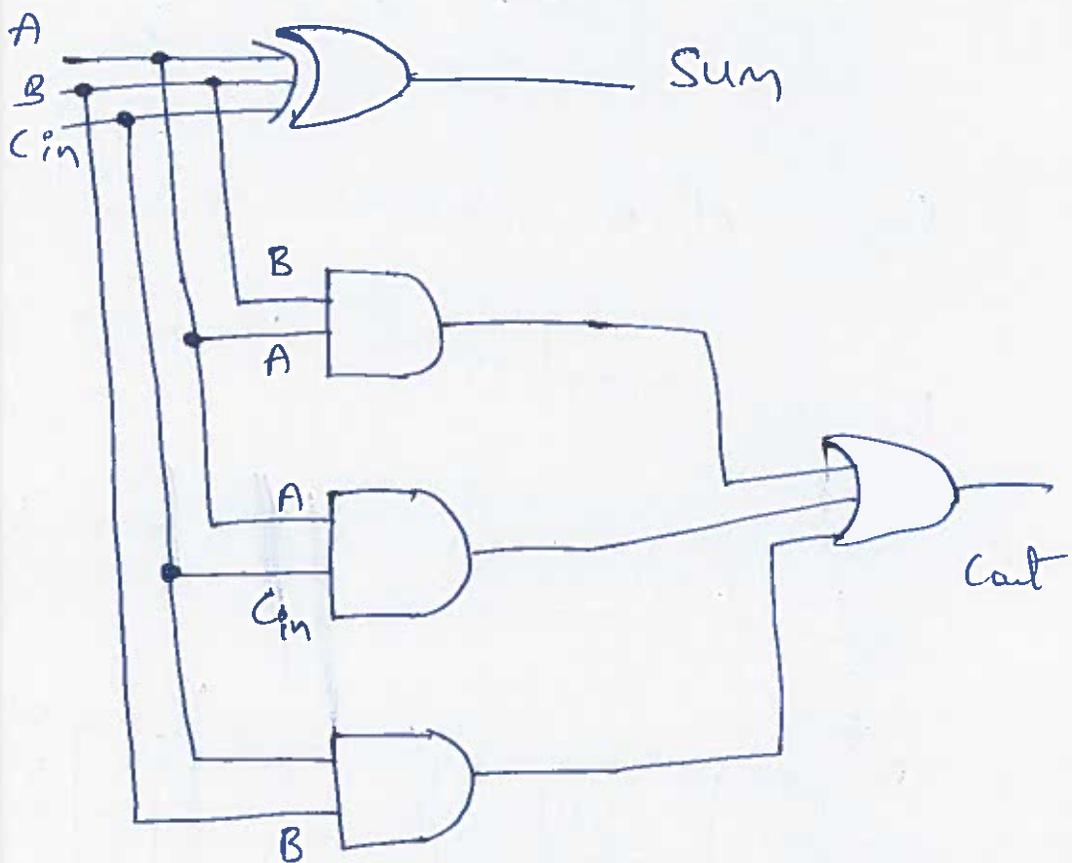


fig: logic diagram for full adder.

## Binary- Subtractor:

The subtraction consists of four possible elementary operations, namely.

$$0 - 0 = 0$$

$$0 - 1 = 1 \quad \text{with } 1 \text{ borrow}$$

$$1 - 0 = 1$$

$$1 - 1 = 0$$

In case of second operation the minuend bit is smaller than the subtrahend bit, hence 1 is borrowed.

## Half Subtractor:

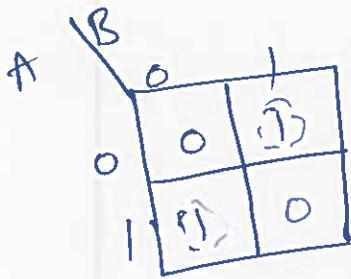
A half-subtractor is a combinational circuit that subtracts two bits and produces their difference.

Inputs		Outputs	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

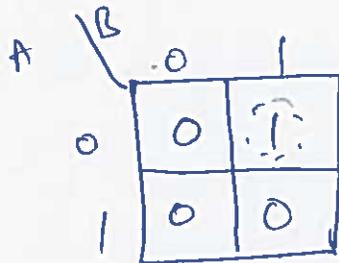
A Truth table for half-subtractor.

# K-map simplification for half-subtractor

for difference



$$\begin{aligned} \text{Difference} &= A\bar{B} + \bar{A}B \\ &= A \oplus B \end{aligned}$$



$$\text{Borrow} = \bar{A}B$$

Logic diagram:

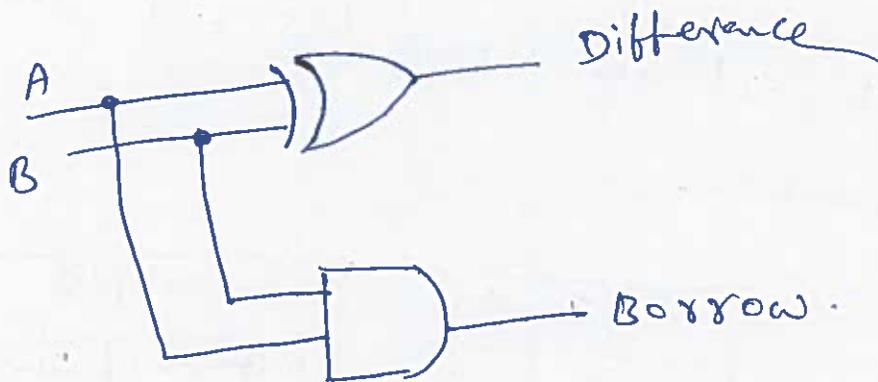


fig: logic diagram for Half-subtractor

## Full-Subtractor:

A full-subtractor is a combinational circuit that performs a subtraction between two bits, taking into account borrow of the lower significant stage.

The three inputs are  $A$ ,  $B$  and  $B_{in}$ .  
The two outputs are  $D$ , and  $B_{out}$  represent the difference and borrow.

Inputs			Outputs	
A	B	$B_{in}$	D	$B_{out}$
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Fig: Truth table for full-subtractor.

# K-Map Simplification of D and Bout

for D

	B <sub>in</sub>			
A	00	01	11	10
0	0	1	0	1
1	1	0	1	0

for Bout

	B <sub>in</sub>			
A	00	01	11	10
0	0	1	1	1
1	0	0	1	0

$$B_{out} = \bar{A}B_{in} + \bar{A}B + BB_{in}$$

$$D = \bar{A}\bar{B}B_{in} + \bar{A}B\bar{B}_{in} + A\bar{B}B_{in} + AB\bar{B}_{in}$$

Logic diagram:

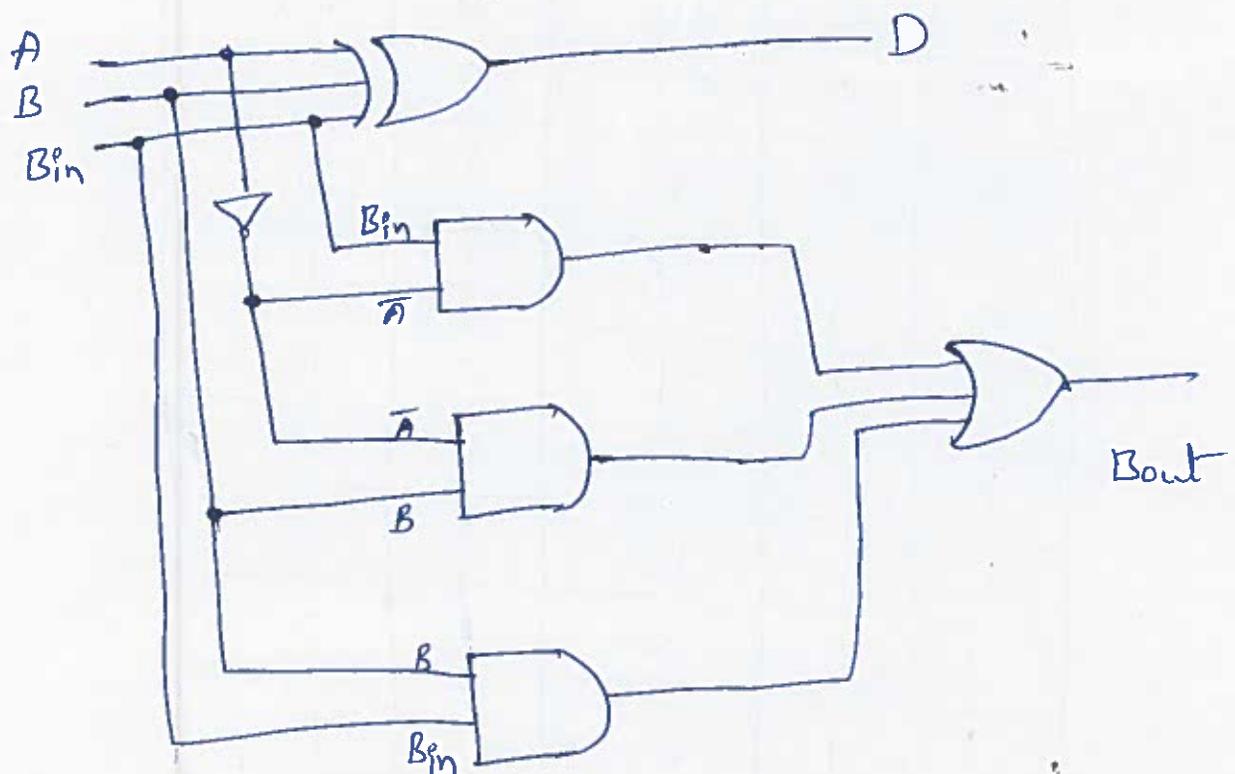


fig: logic diagram for full-subtractor.

① Design a BCD to Excess-3 Code Converter.

Excess-3 code is a modification of a BCD number. The Excess-3 Code can be derived from the natural BCD code by adding 3 to each coded number. For example, decimal 12 can be represented in BCD as 0001 0010. Now adding 3 to each digit we get Excess-3 code as 0100 0101. With this information the truth table for BCD to Excess-3 Code Converter can be determined as shown in table.

Decimal	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	E <sub>3</sub>	E <sub>2</sub>	E <sub>1</sub>	E <sub>0</sub>
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

# K-Map Simplification

	$B_1, B_0$	00	01	11	10
$B_3, B_2$	00	0	0	0	0
	01	0	1	1	1
	11	X	X	X	X
	10	1	1	X	X

$$\therefore E_3 = B_3 + B_2(B_0 + B_1)$$

for  $E_2$

	$B_1, B_0$	00	01	11	10
$B_3, B_2$	00	0	1	1	1
	01	1	0	0	0
	11	X	X	X	X
	10	1	X	X	X

$$E_2 = B_2 \overline{B_1} \overline{B_0} + \overline{B_2} (B_0 + B_1)$$

for  $E_1$

	$B_1, B_0$	00	01	11	10
$B_3, B_2$	00	1	0	1	0
	01	1	0	1	0
	11	X	X	X	X
	10	1	0	X	X

$$E_1 = \overline{B_1} \overline{B_0} + B_1 B_0$$

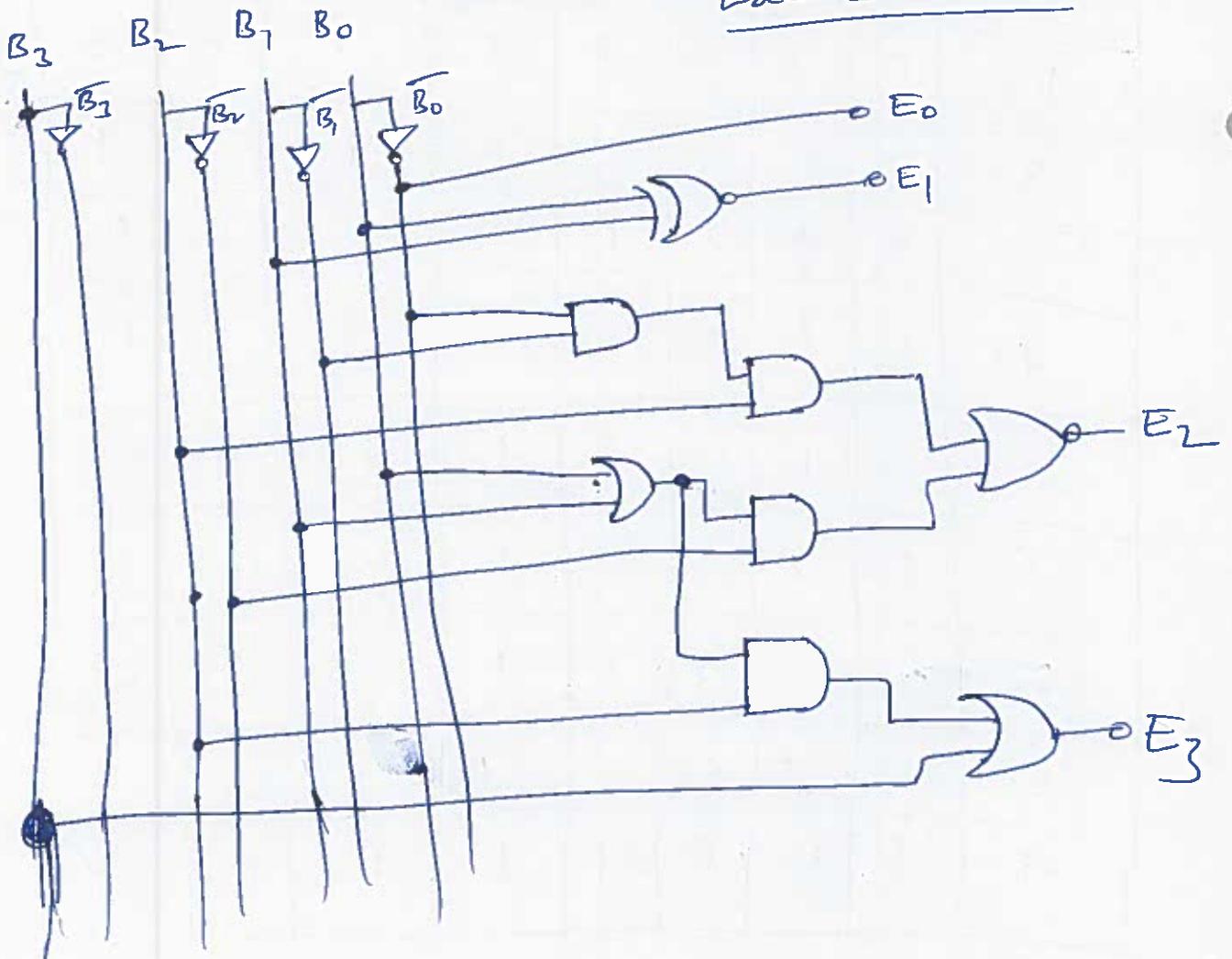
$$= B_1 \odot B_0$$

for  $E_0$

	$B_1, B_0$	00	01	11	10
$B_3, B_2$	00	1	0	0	1
	01	1	0	0	1
	11	X	X	X	X
	10	X	0	X	X

$$E_0 = \overline{B_0}$$

logic diagram:



Excess-3 Code

## BCD to 7 Segment Display decoders:

In most practical applications, seven segment displays are used to give a visual indication of the output states of digital ICs such as decade counters, latches. ---

These outputs are usually in four bit BCD (Binary Coded Decimal) form.

LED and LCD decoders/drivers for seven segment displays.

Digit	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	1	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	1	0	1

Truth table for BCD to 7 segment decoder

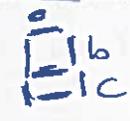
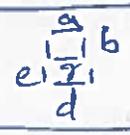
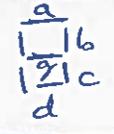
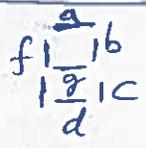
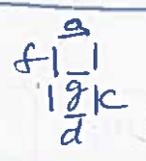
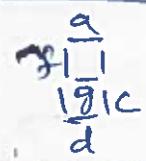
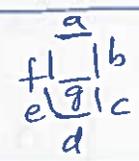
Digit	Segments Activated	Display
0	a, b, c, d, e, f	
1	b, c	
2	a, b, d, e, g	
3	a, b, c, d, g	
4	b, c, f, g	
5	a, c, d, f, g	
6	a, c, d, e, f, g	
7	a, b, c	
8	a, b, c, d, e, f, g	
9	a, b, c, d, f, g	

Table: Segments activated during each digit display.

## NAND and NOR Implementation:

### i) NAND-NAND implementation

The implementation of a ~~Boolean~~ Boolean function with NAND-NAND logic requires that the function be simplified in the sum of product form. The relationship between AND-OR logic and NAND-NAND logic is explained using following example

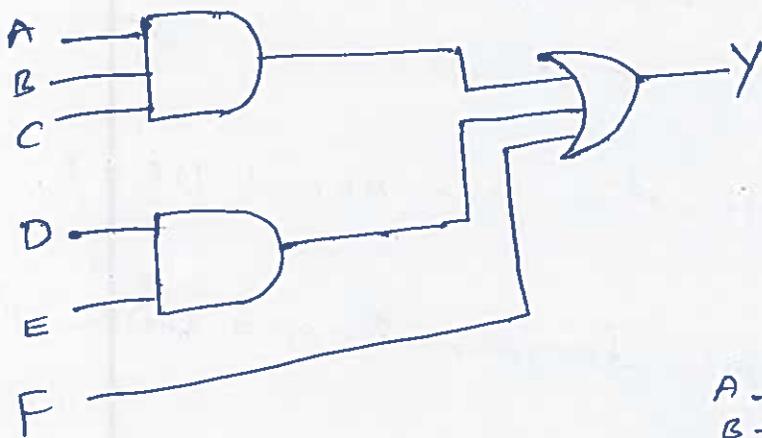
$$\text{Boolean function: } Y = ABC + DE + F.$$

NAND-NAND logic diagram from a Boolean function as follows:

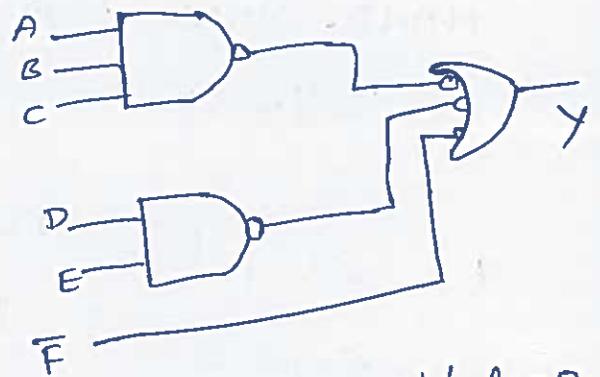
1. Simplify the given Boolean function and express it in sum of product form (SOP form)
2. Draw a NAND gate for each product term of the function that has two or more literals. The inputs to each NAND gate are the literals of the term. This constitutes a group of first-level gates.
3. If Boolean function includes any single literals or literals draw NAND gate for

each single literal and connect corresponding literal as an input to the NAND gate.

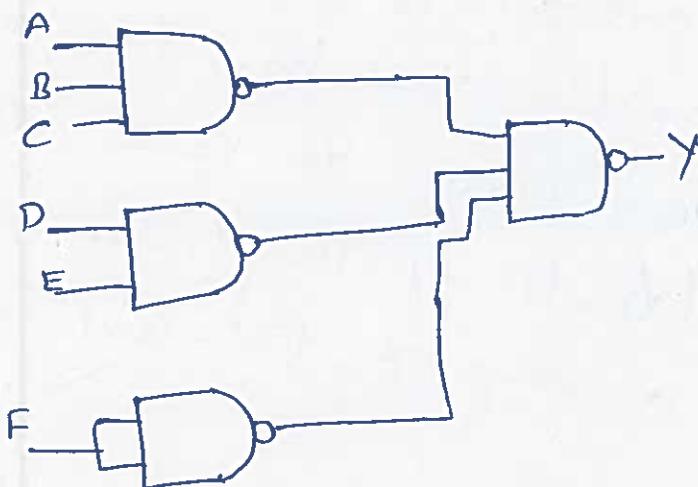
4. Draw a single NAND gate for the second level, with inputs coming from outputs of first level gates.



(a) AND-OR



(b) NAND-Bubbled OR



(c) NAND-NAND  
fig: NAND-NAND implementation.

## NOR-NOR Implementation:

The NOR function is a dual of the NAND function. For this reason, the implementation procedures and rules for NOR-NOR logic are the duals of the corresponding procedures and rules developed for NAND-NAND logic. The relationship between OR-AND logic and NOR-NOR is explained using following example.

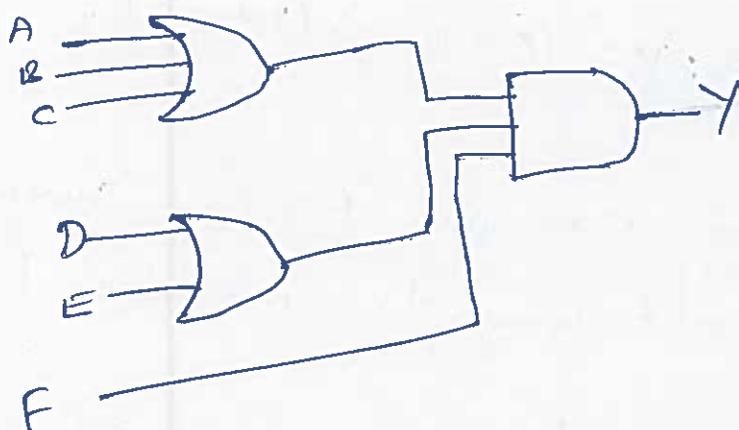
$$\text{Boolean function: } Y = (A+B+C)(D+E)F.$$

From the above example we can summarise the rules for obtaining the NOR-NOR logic diagram from a Boolean function as follows.

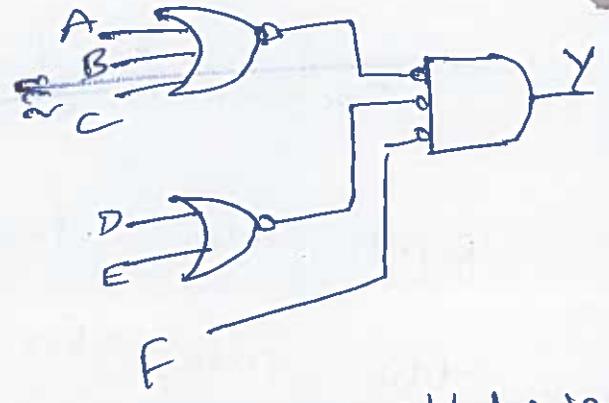
1. Simplify the given Boolean function and express it in product of sum form (POS)
2. Draw a NOR gate for each sum term of the function that has two or more literals. The inputs to each NOR gate are the literals of the term. This constitute a group of first level gates.

3. If Boolean function includes any single literal or literals, draw NOR gate for each single literal and connect for each single literal and connect corresponding literal as an input to the NOR gate.

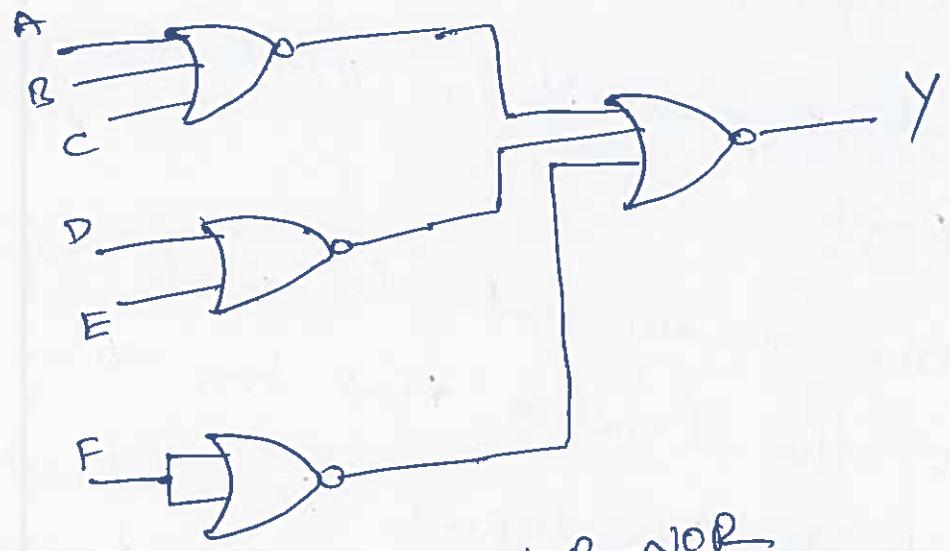
4. Draw a single NOR gate in the second level, with inputs coming from outputs of first level gates.



(a) OR-AND



(b) NOR-Bubbled AND



(c) - NOR-NOR

→ Simplify the following Boolean function using Quine - McCluskey method.

$$F(A, B, C, D) = \sum m(0, 2, 3, 6, 7, 8, 10, 12, 13)$$

Solution:

step 1: List all minterms in the binary form as shown in table (column(a)).

step 2: Arrange the minterms according to number of 1s, as shown in table (column(b)).

Minterm	Binary Representation	Minterm	Binary Representation
$m_0$	0000	$m_0$	0000 ✓
$m_2$	0010	$m_2$	0010 ✓
$m_3$	0011	$m_8$	1000 ✓
$m_6$	0110	$m_3$	0011 ✓
$m_7$	0111	$m_6$	0110 ✓
$m_8$	1000	$m_{10}$	1010 ✓
$m_{10}$	1010	$m_{12}$	1100 ✓
$m_{12}$	1100	$m_7$	0111 ✓
$m_{13}$	1101	$m_{13}$	1101 ✓
Column(a)		Column(b)	

Step 3: Compare each binary number with every term in the adjacent, next higher category and if they differ only by one position, put a check mark and copy the term in the next column in the position that they differed.

Step 4: Apply the same process described in step 3 for the resultant column and continue these cycles until a single pass through cycle yields no further elimination of literals.

Minterms	Binary Representation	Minterms	Binary Representation
0, 2	00-0v	0, 2, 8, 10	-0-0
0, 8	-000v	2, 3, 6, 7	0-1-
2, 3	001-v		
2, 6	0-10v		
2, 10	-010v		
8, 10	10-0v		
8, 12	1-00		
3, 7	0-11v		
6, 7	011-v		
12, 13	110-		
	column (c)		column (d)

step 5! List the prime implicants.

prime Implicants	Binary Representation
8, 12	1-00
12, 13	110-
0, 2, 8, 10	-0-0
2, 3, 6, 7	0-1-

step 6: select the minimum number of prime implicants which must cover the minterms.

prime Implicants	m <sub>0</sub> (col 1)	m <sub>2</sub> (col 2)	m <sub>3</sub> (col 3)	m <sub>6</sub> (col 4)	m <sub>7</sub> (col 5)	m <sub>8</sub> (col 6)	m <sub>10</sub> (col 7)	m <sub>12</sub> (col 8)	m <sub>13</sub> (col 9)
8, 12						•		•	
12, 13 ✓								•	•
0, 2, 8, 10 ✓	•	•				•	•		
2, 3, 6, 7 ✓		•	•	•	•				

Table: prime implicant selection chart

The final expression is

$$F(A, B, C, D) = (110-) + (-0-0) + (0-1-)$$

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

Unit 4, 36/44

<p>1. 100</p> <p>2. 100</p> <p>3. 100</p> <p>4. 100</p> <p>5. 100</p>	<p>1. 100</p> <p>2. 100</p> <p>3. 100</p> <p>4. 100</p> <p>5. 100</p>
---	---

Unit 4, 36/44

1	2	3	4	5	6	7	8	9	10
100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100

Unit 4, 36/44

$$(100 - 100) + (100 - 100) = 0 + 0 = 0$$

Unit 4, 36/44

→ Simplify the following function using map method.

$$F(A, B, C, D) = \sum(0, 1, 2, 3, 4, 6, 9, 10) + d(7, 11, 12, 13, 15)$$

Sol:

K-map for four variables and it is plotted according to expression in stand SOP form.

AB		CD			
		$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	$CD$ 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$	00	1	1	1	1
$\bar{A}B$	01	1		X	1
$AB$	11	X	X	X	
$A\bar{B}$	10		1	X	1

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

-> Simplify the following function using K-map

$$F(A, B, C, D) = \sum (1, 3, 4, 5, 6, 11, 13, 14, 15)$$

K-map for four variables and it is plotted according to given minterms.

		CD			
		$\bar{C}\bar{D}$	$\bar{C}D$	$CD$	$C\bar{D}$
$\bar{A}\bar{B}$	00		1	1	
$\bar{A}B$	01	1	1		1
$AB$	11		1	1	1
$A\bar{B}$	10			1	

$$F(A, B, C, D) = \bar{A}\bar{B}D + \bar{A}BC + \bar{A}\bar{C}D + ABD + ABC + ACD + BC\bar{D}$$

# Map method

The basis of this method is a chart known as Karnaugh map (K-map). It contains boxes called cells. Each of the cell represents one of the  $2^n$  possible products that can be formed from  $n$  variables.

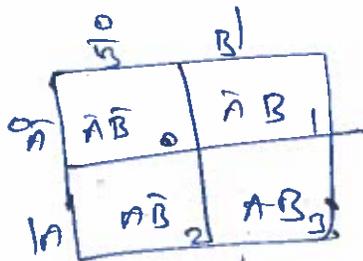
2 variable map contains  $2^2 = 4$  cells

3 variable map contains  $2^3 = 8$  cells

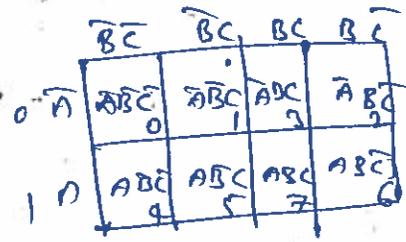
4 variable map contains  $2^4 = 16$  cells



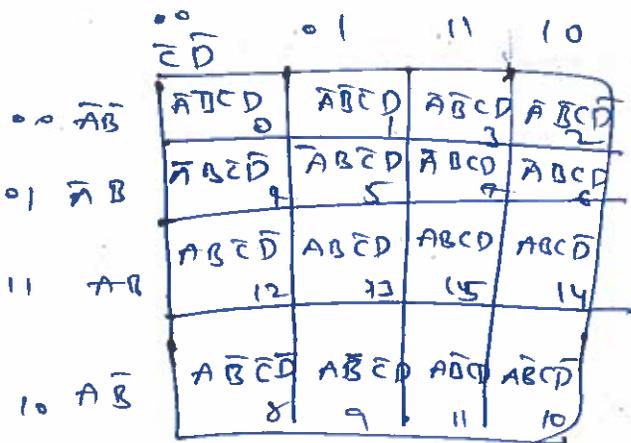
1-variable map  
(2 cells)



2-variable map  
(4 cells)



3-variable map  
(8 cells)



4-variable map  
(16 cells)

maps with product terms

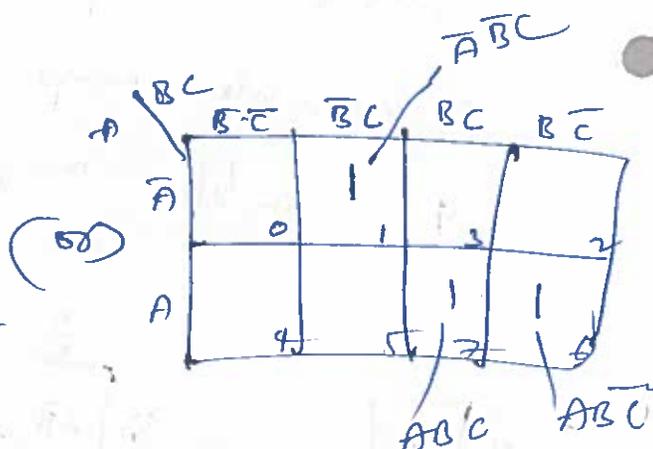
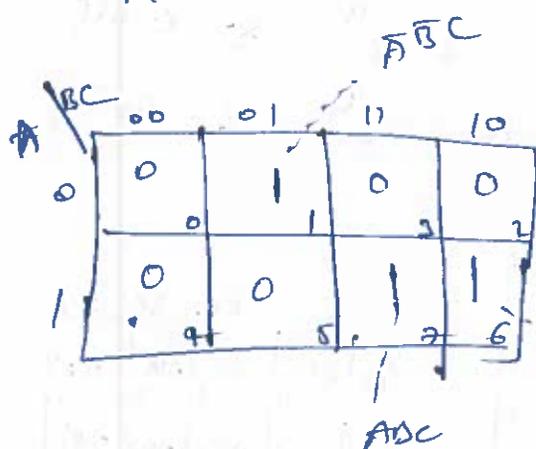
unit-4, 39/44

unit-4, 39/44

We know that Canonical formules can be represented in various forms such as truth table, sop boolean expression and pos boolean expression.

ex: plot Boolean expression  $Y = A\bar{B}\bar{C} + ABC + \bar{A}\bar{B}C$  on the Karnaugh map.

The expression has 3 variables and hence it can be plotted using 3-variable map.



product terms representation K-map

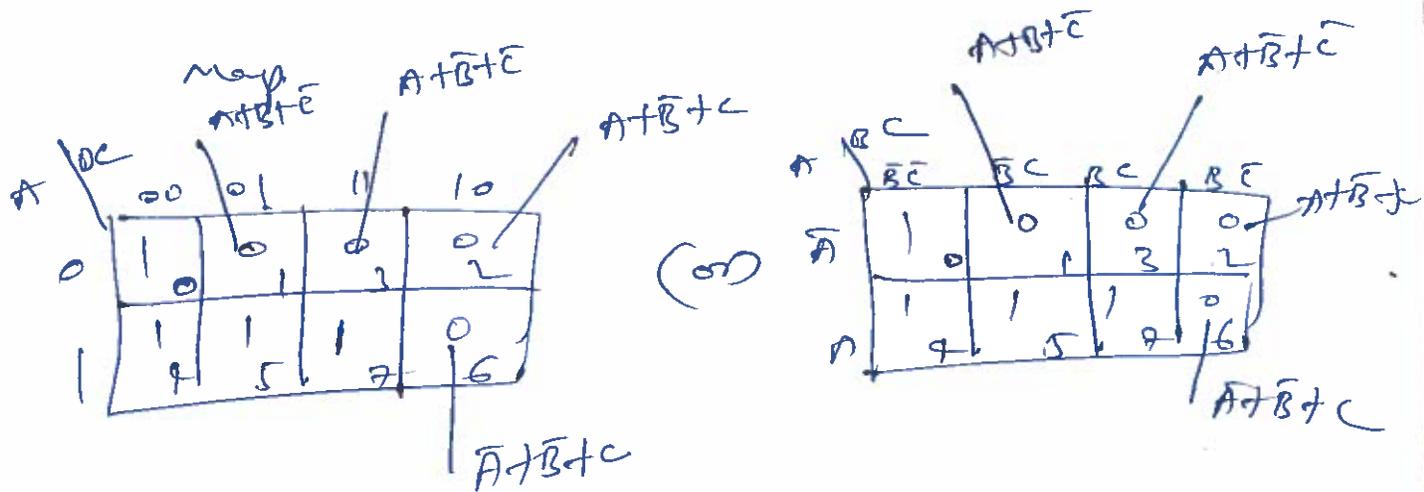
ex: A boolean expression with product terms (the product of sums) can be plotted on the K-map by

placing a 0 in each cell corresponding to a term (maxterm) in the expression.

Remaining cells are filled with ones.

Q1. Plot Boolean expression  $Y_2 = (A+B+C)(A+B+C)$   
 $(\bar{A}+\bar{B}+\bar{C})(A+B+C)$  on the K-map

The expression has 3 variables and hence it can be plotted using 3-variable



### Sum Terms Representation on K-map

A Boolean expression with sum terms (sum of products form) can be plotted on the K-map by using a 1 in each cell corresponding to a term. Counterterms in the sum of products expression. Remaining cells are filled with zeros.



ex: ① 
$$Y = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}\bar{C}$$

$$= m_0 + m_1 + m_3 + m_6$$

$$= \sum m(0, 1, 3, 6)$$

② 
$$Y = (A+B+\bar{C})(A+\bar{B}+\bar{C})(\bar{A}+B+C)$$

$$= M_1 + M_3 + M_6$$

$$= \pi M(1, 3, 6)$$

Here  $\sum$  denotes Sum of product

and  $\pi$  denotes product of sum.

we know that logical expression can be represented

in the truth table form. it is possible to

write logic expression in standard SOP or POS

form corresponding to a given truth table.

one product term for each input combination that produces an output of 1.

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

←  $\bar{A}B\bar{C}$

←  $ABC$

←  $A\bar{B}\bar{C}$

Sum of product form is

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

product of sums ~~from~~ form by writing the sum term for each output '0'.

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

←  $A + \bar{B} + C$

←  $\bar{A} + B + \bar{C}$

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