Digital Design and Computer Organization

MODULE - 2

Pothi Reddy . K

Combinational Logic Circuits

Introduction :

Definition :- when logic gates are connected together to produce a specified output for certain specified combinations of input variables, with no storage involved, the resulting circuit is called Combinational-Logic Circuit.

In Combinational logic circuit, the output variables are at all times dependent on the combination of input variables.

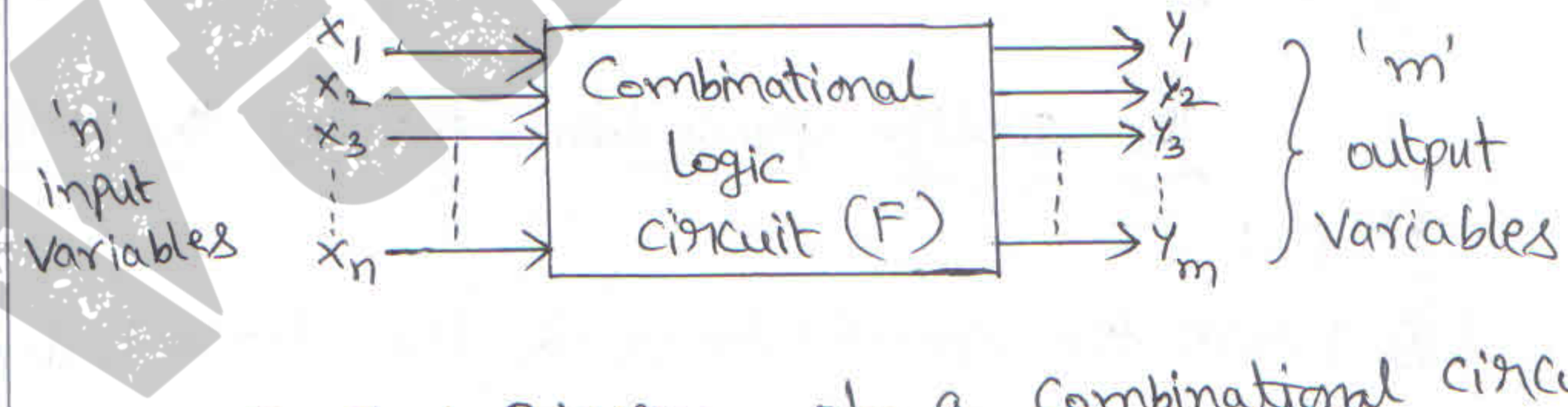


Fig: ① Block Diagram of a Combinational circuit

The relationship between input and output of a Combinational logic circuit is,

$$Y = F(X) \longrightarrow ①$$

where $X = \{x_1, x_2, x_3 \cdots x_n\}$ and $Y = \{Y_1, Y_2 \cdots Y_m\}$.

Design Procedure:

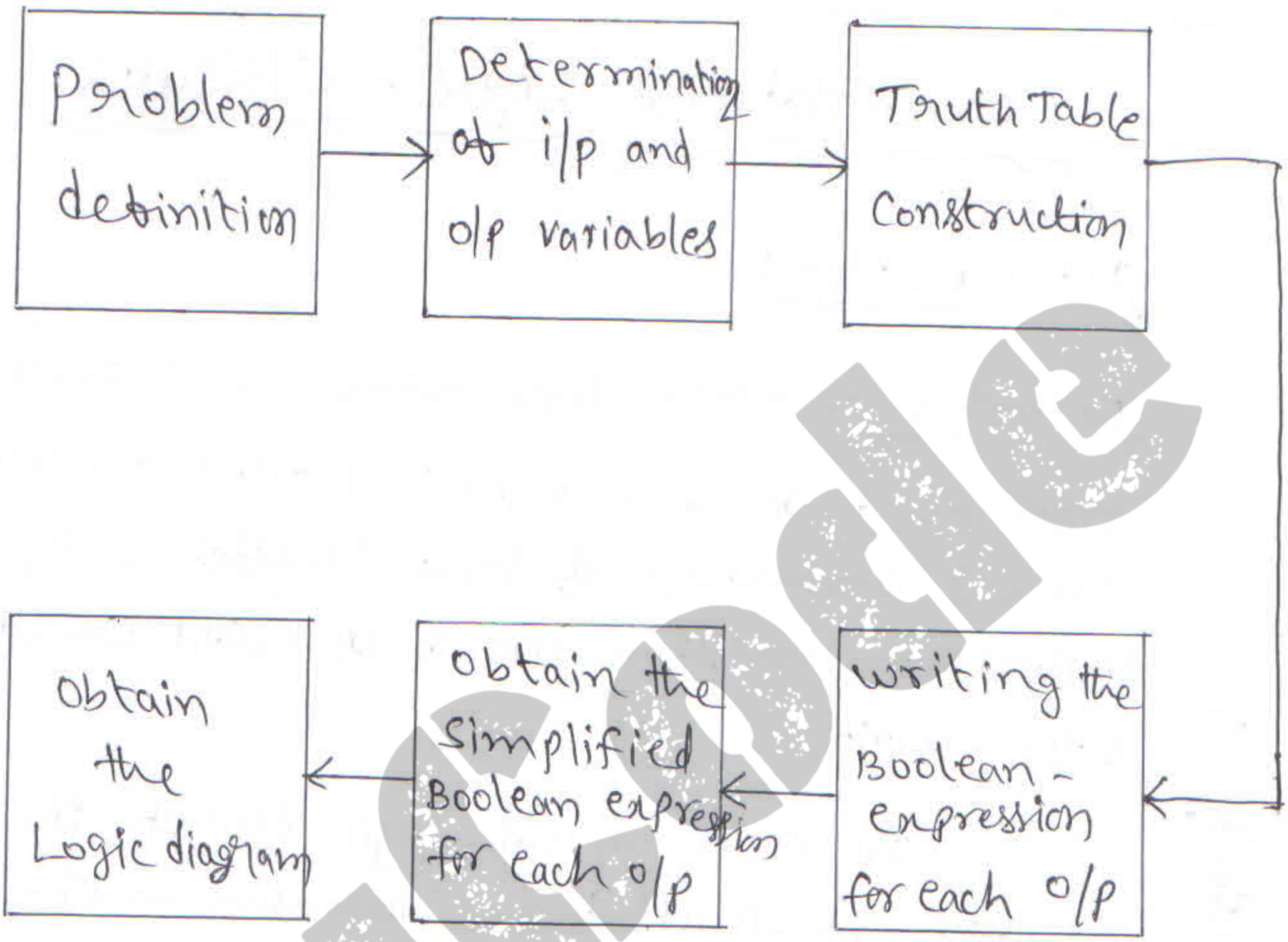


Fig: ② Typical Design sequence of Combinational circuit.

The Design procedure involves the following steps:

① From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.

② Derive the truth table that defines the required relationship between inputs and outputs.

③ obtain the simplified Boolean functions for each output as a function of the input variables.

④ Draw the logic diagram.

Examples :

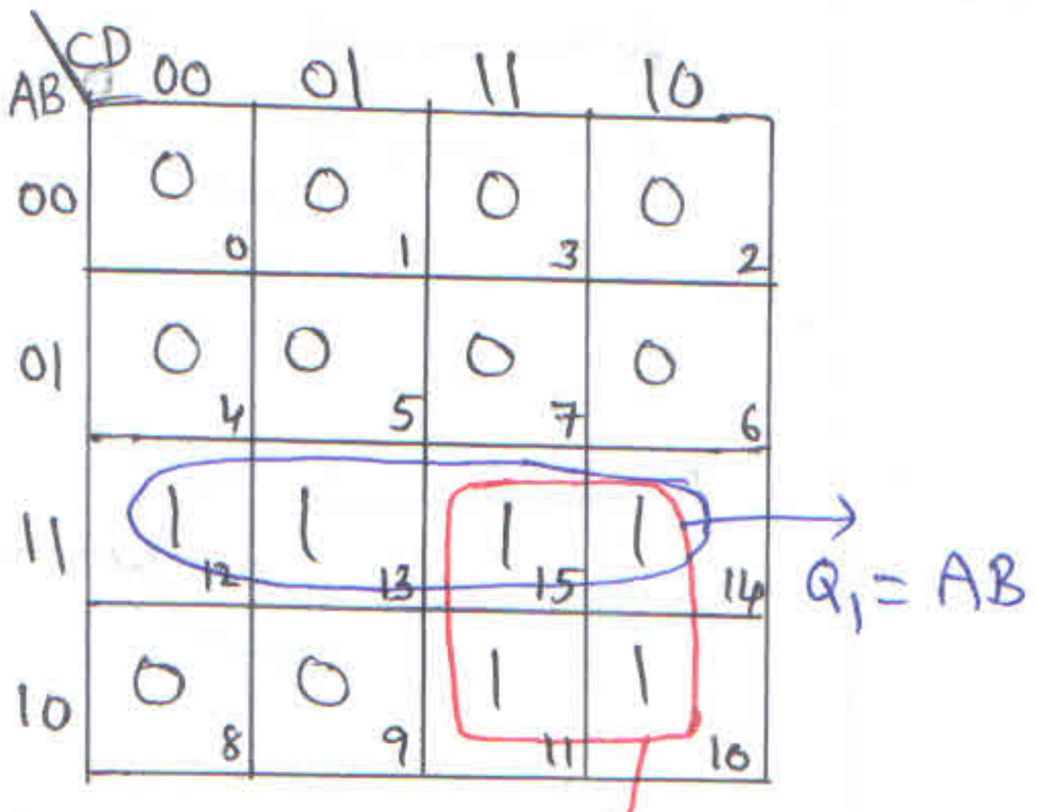
① Design a circuit to detect invalid BCD number and implement using NAND gate only.

Sol)

Truth Table :-

Inputs				output
A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

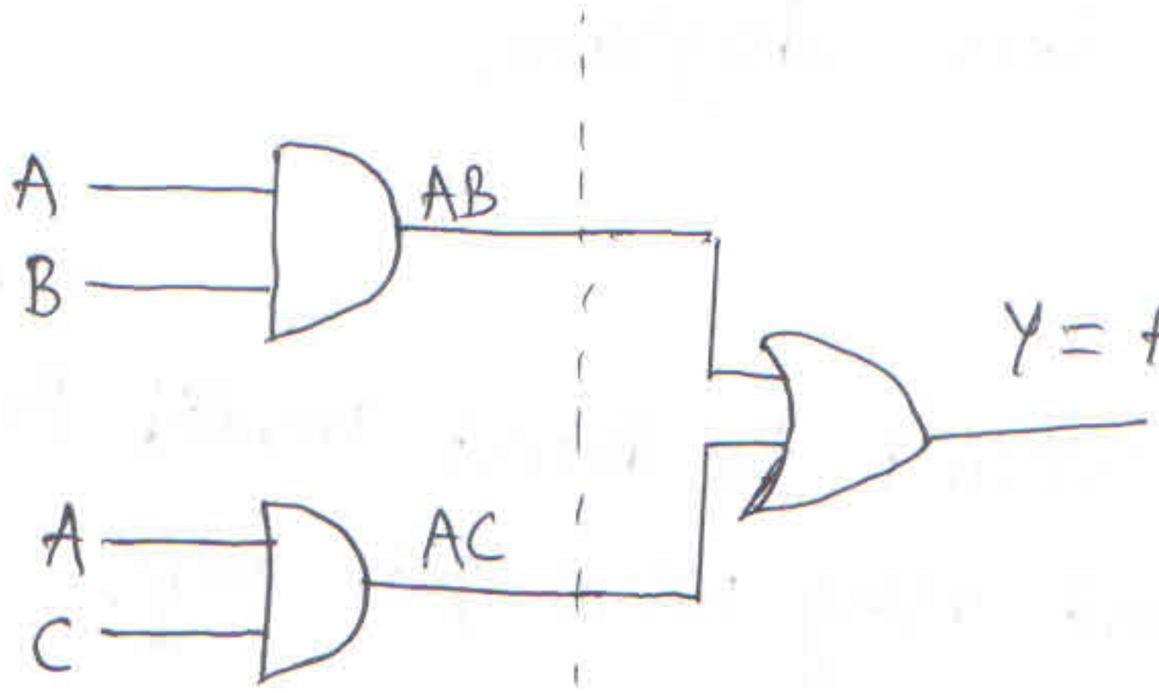
K-map Simplification :-



$Y = AB + AC$

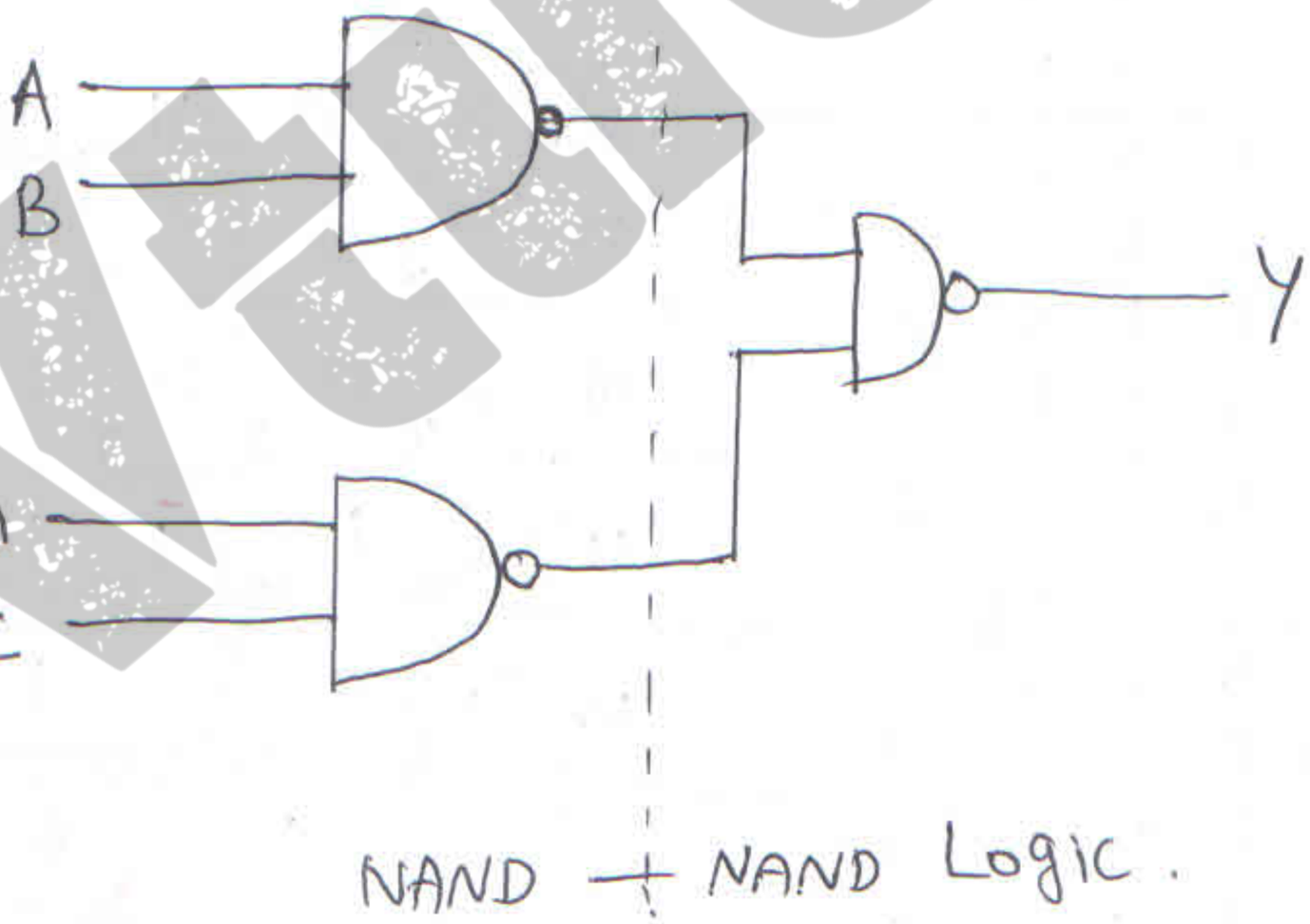
Logic Diagram :-

$Y = AB + AC$



AND – OR Logic

The AND – OR logic can be directly implemented by NAND – NAND logic.



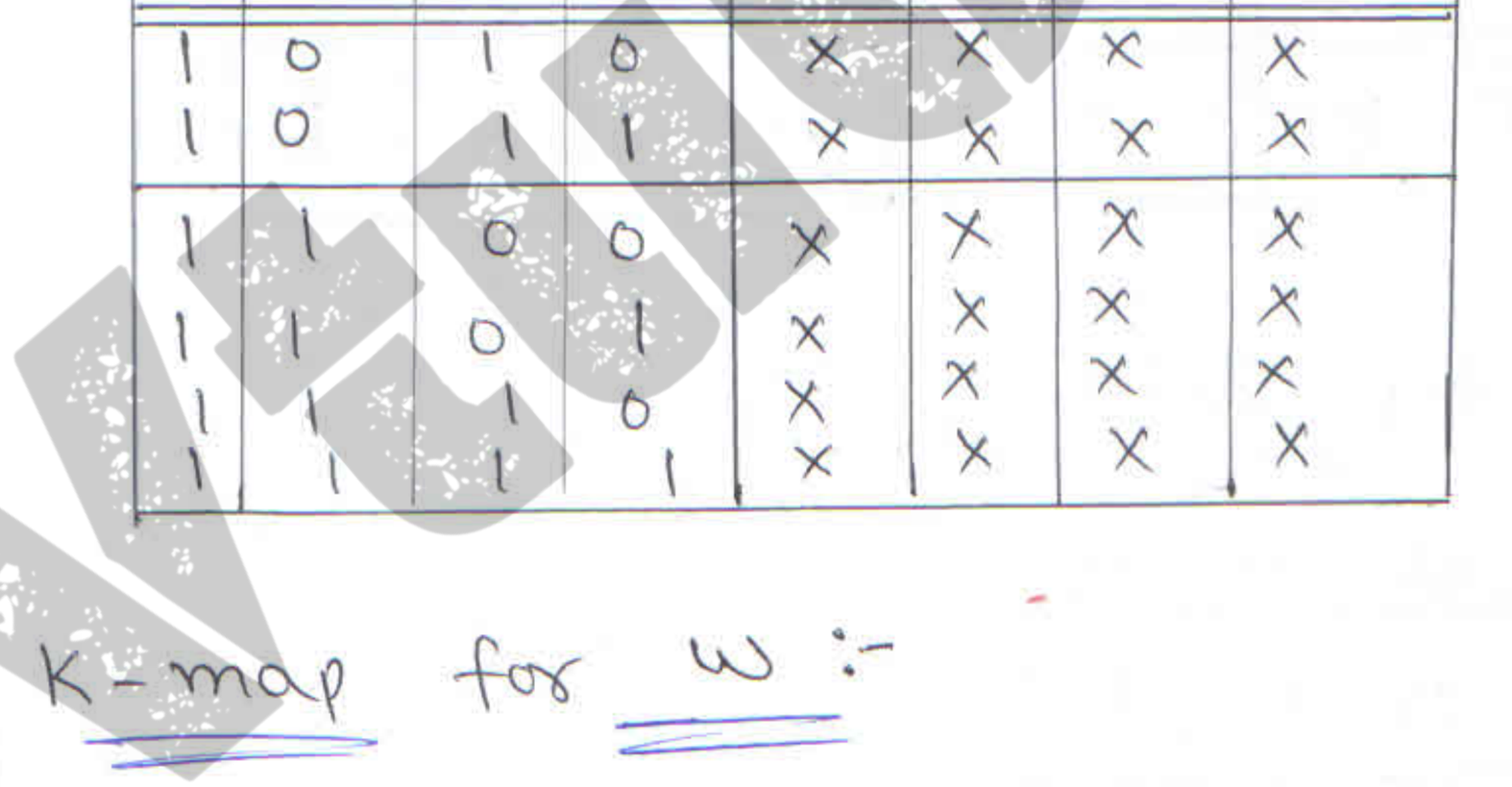
NAND – NAND Logic.

② Design a Combinational Logic circuit to Convert BCD to Excess-3 Code Conversion.

Sol) Truth Table :-

Inputs (BCD)				outputs Excess-3			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

K-map for w :-



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

K-map for X :-

AB \ CD	00	01	11	10
00	0	1	1	1
01	1	0	0	0
11	X	X	X	X
10	0	1	X	X

$Q_1 = \bar{B}D$

pair $= B\bar{C}\bar{D}$

$Q_2 = \bar{B}C$

$$X = \bar{B}D + \bar{B}C + B\bar{C}\bar{D}$$

K-map for Y :-

AB \ CD	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	X	X	X	X
10	1	0	X	X

$Q_1 = CD$

$Q_2 = \bar{C}\bar{D}$

$$Y = CD + \bar{C}\bar{D}$$

K-map for Z :-

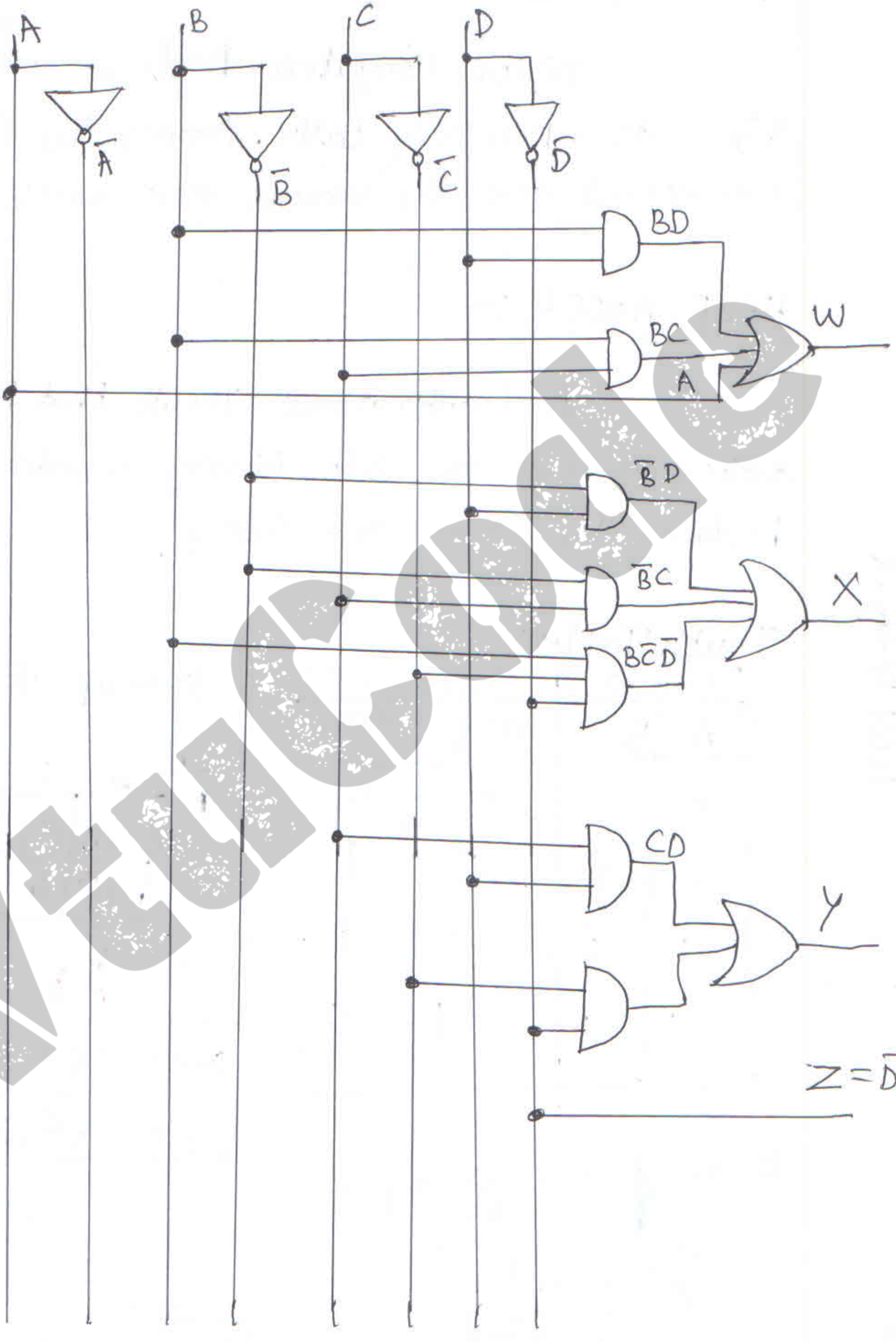
AB \ CD	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	X	X	X	X
10	1	0	X	X

octet $= \bar{D}$

$$Z = \bar{D}$$

Pothi Reddy. K

Logic diagram :- (BCD to Excess-3 Code Converter)



Binary Adder - Subtractor :

Digital Computers perform a variety of information-processing tasks. Among the functions encountered are the various arithmetic operations.

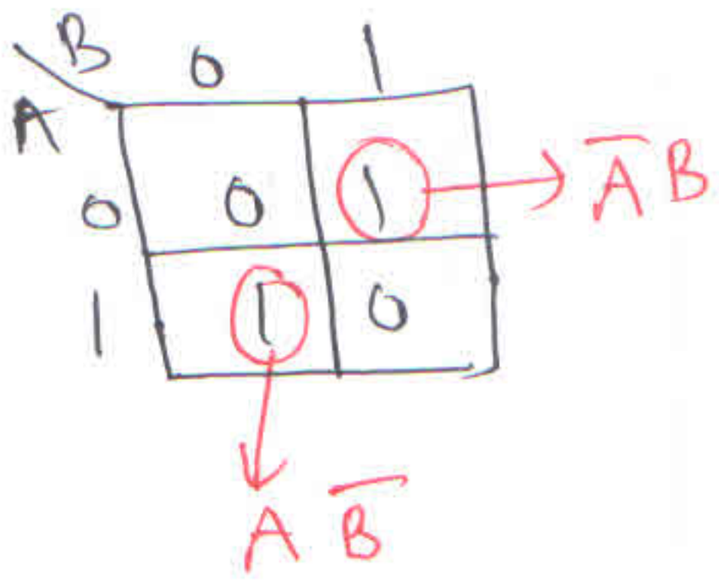
HALF ADDER :-

A Combinational Circuit that ~~performs~~ the addition of two-one bit binary numbers and produce the Sum and Carry.

Truth Table:

Inputs		outputs.	
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

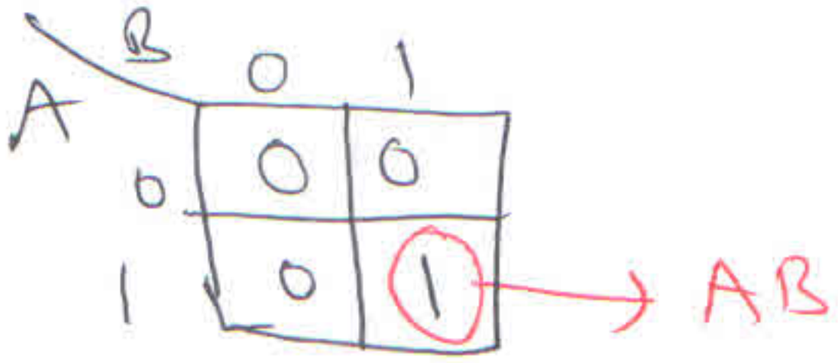
K-map for Sum:-



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

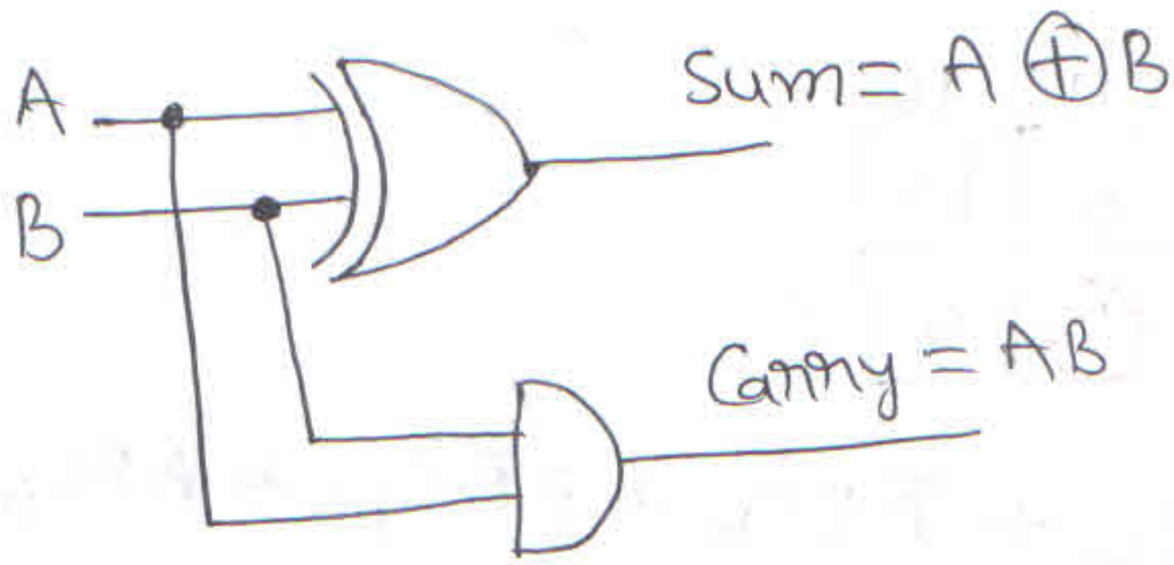
$Sum = A \oplus B$

K-map for Carry :-



$Carry = AB$

Logic Diagram :-



Full Adder:

A Full-adder is a combinational circuit that forms the arithmetic sum of three input bits and produces the sum and Carry.

Truth Table :-

Inputs			outputs	
A	B	C_{in}	C_{out}	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

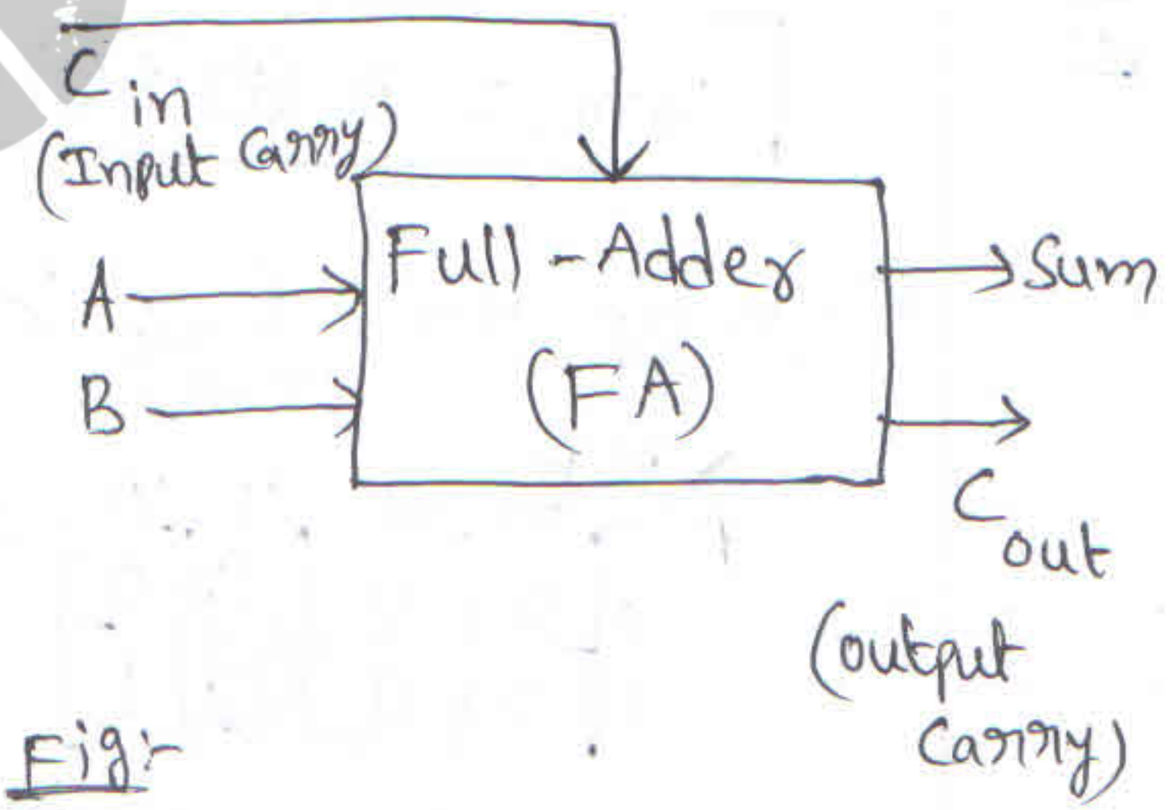


Fig:-
Block Diagram of Full Adder

K-map for Sum :-

A \ BCin	00	01	11	10
0	0	(1)	0	(1)
1	(1)	0	(1)	0

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

$$= \bar{A}(\bar{B}C_{in} + B\bar{C}_{in}) + A(\overline{\bar{B}\bar{C}_{in} + BC_{in}})$$

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

$$= \bar{A}.X + A.\bar{X} \quad (\because X = B \oplus C_{in})$$

$$= A \oplus X$$

$$\boxed{Sum = A \oplus B \oplus C}$$

K-map for Cout :-

A \ BCin	00	01	11	10
0	0	0	(1)	0
1	0	(1)	(1)	(1)

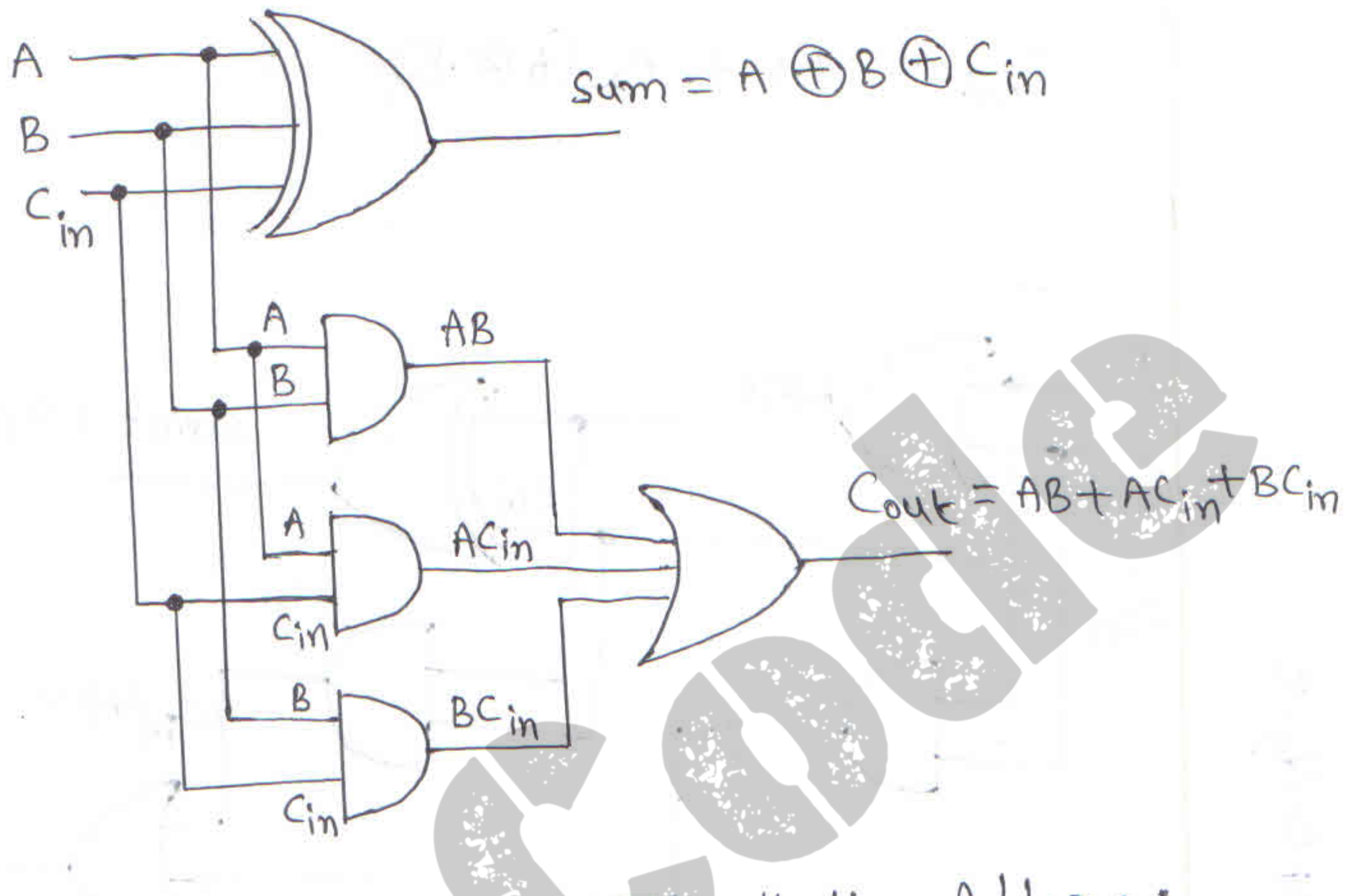
$P_1 = BC_{in}$

$P_2 = AC_{in}$

$P_3 = AB$

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

Logic Diagram :-



Full - Adder using Two Half - Adders :

A Full - adder can also be implemented with two Half - adders and one OR gate.

$$\boxed{Sum = A \oplus B \oplus C_{in}}$$

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

$$= AB + AC_{in}(B + \bar{B}) + BC_{in}(A + \bar{A})$$

$$= AB + ABC_{in} + A\bar{B}C_{in} + ABC_{in} + \bar{A}BC_{in}$$

$$= AB(1 + C_{in} + C_{in}) + A\bar{B}C_{in} + \bar{A}BC_{in}$$

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

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

$$Sum = A \oplus B \oplus C_{in}$$

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

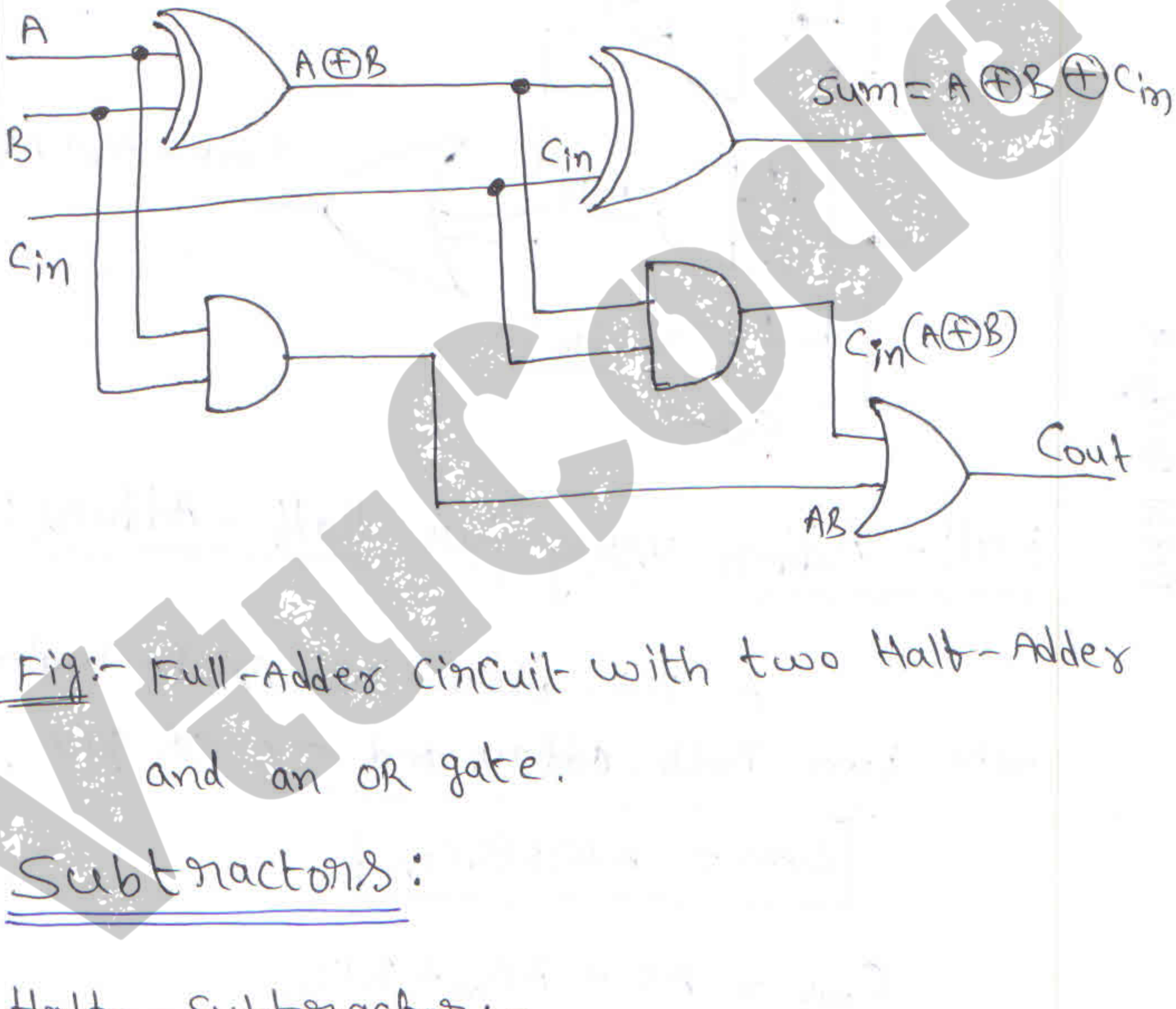


Fig:- Full-Adder circuit with two Half-Adder and an OR gate.

Subtractors:

Half-Subtractor:-

A Half-subtractor is a combinational circuit that subtracts two, 1-bit numbers and produces their difference and borrow.

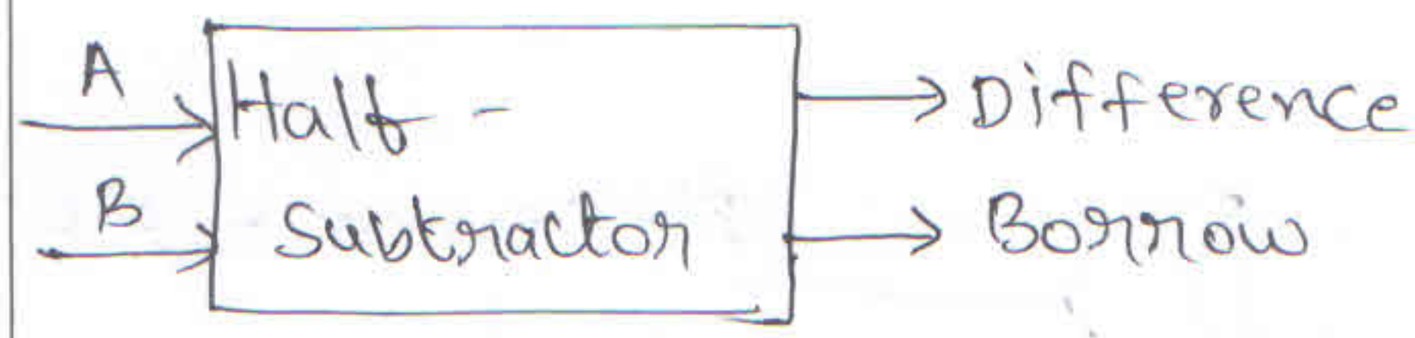


Fig. Block Diagram

Truth Table :-

Inputs		outputs	
A	B	Borrow	Difference
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

K-maps :-

For Difference:

A \ B	0	1
0	0	1
1	1	0

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

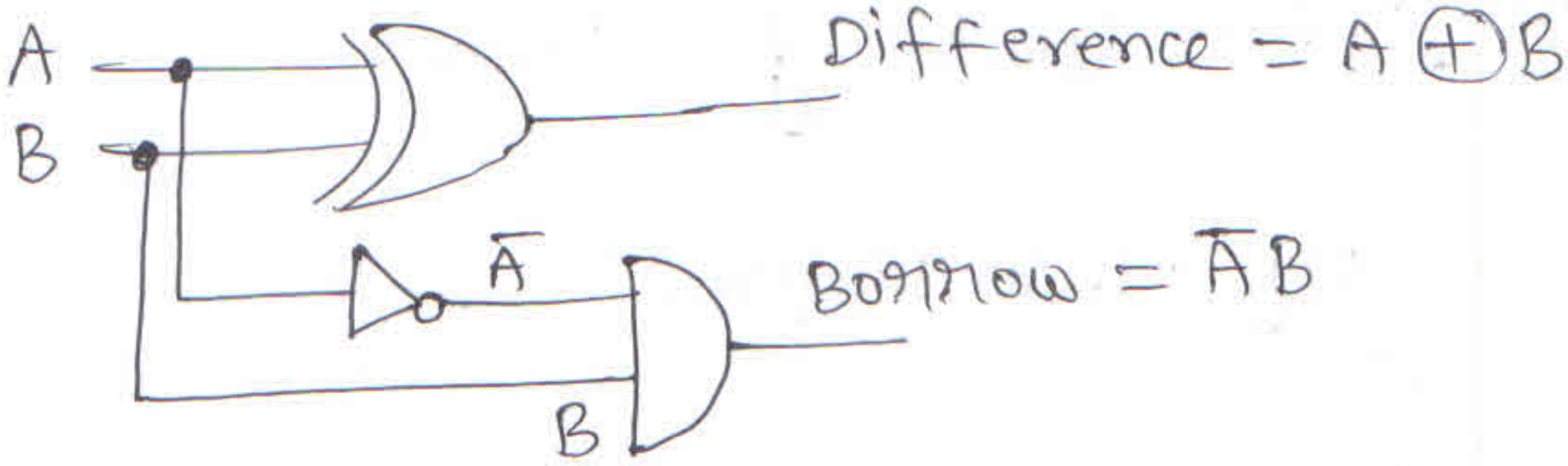
$$\boxed{Diff = A \oplus B}$$

For Borrow:

A \ B	0	1
0	0	1
1	0	0

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

Logic Diagram :-



Full - Subtractor :

A Full-Subtractor is a combinational-circuit that performs a subtraction between two - 1-bit numbers, taking into account borrow of the lower significant stage and produces the Difference and Borrow.

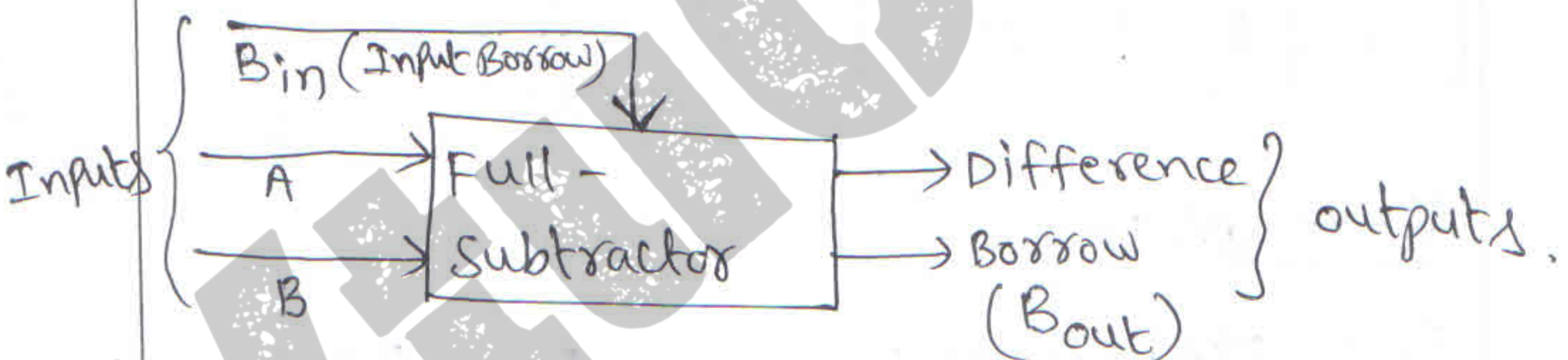


Fig: Block Diagram.

Truth Table :-

Inputs			outputs.	
A	B	B_{in}	B_{out}	Difference
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-maps :-

For Difference :

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

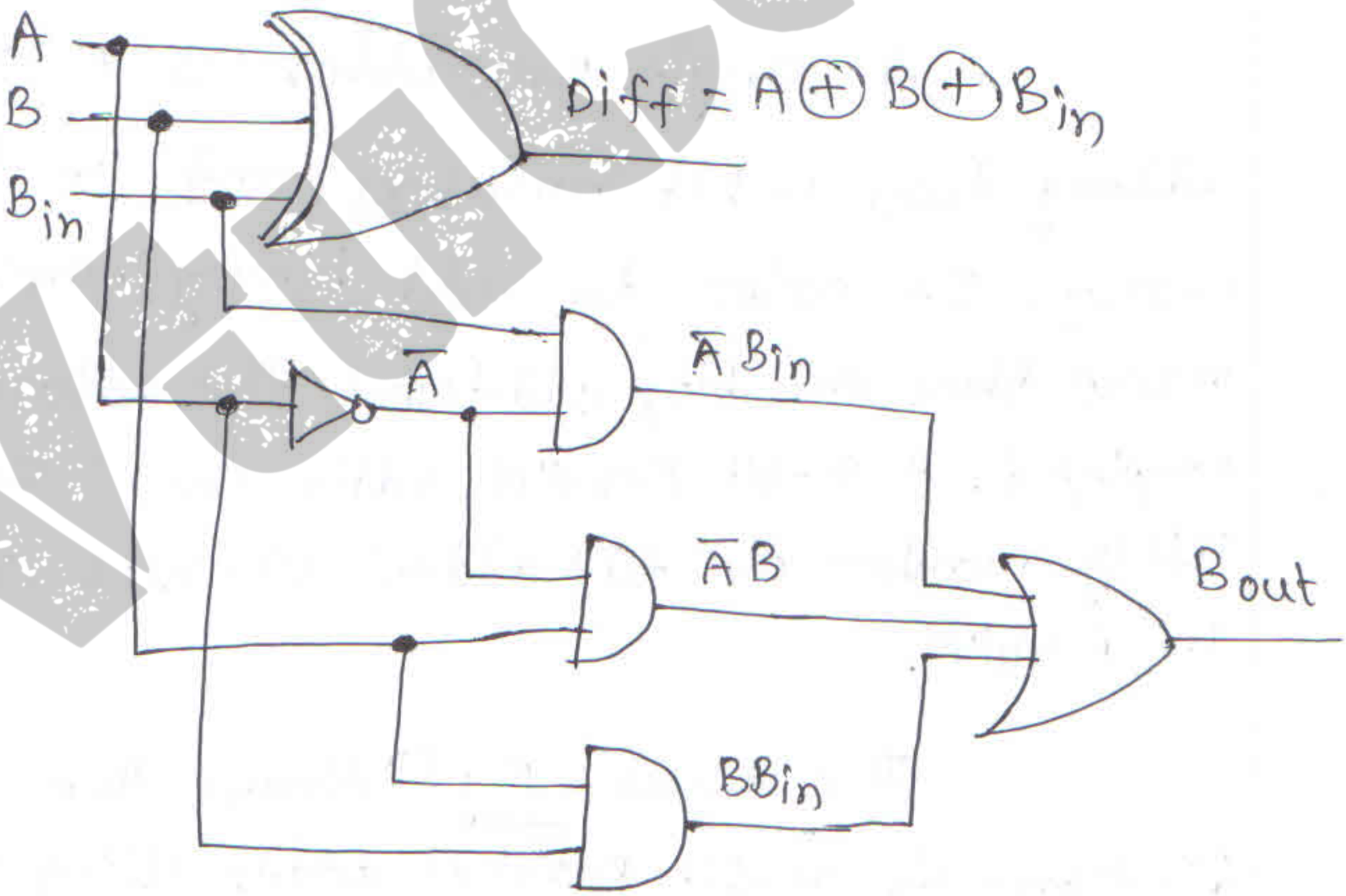
$Diff = A \oplus B \oplus B_{in}$

For B_{out} :

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

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

Logic Diagram :-



Cascading Full - Adders :

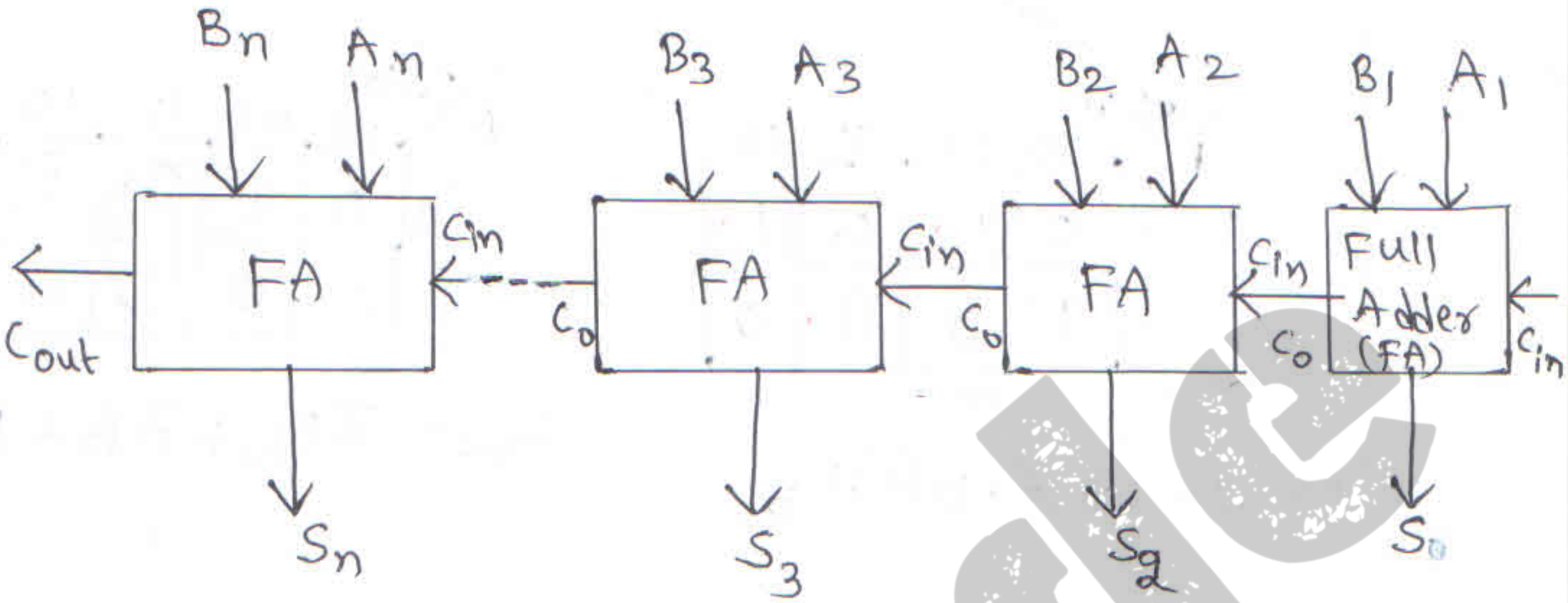


Fig:- ① Block Diagram of n-bit Parallel Adder

A single Full adder is a capable of adding two, 1-bit numbers and an input Carry. In order to add binary numbers with more than one bit, additional full adders must be employed. A n-bit Parallel adder can be constructed using number of full adder circuits connected in parallel.

The above Fig.① shows the block-diagram of n-bit parallel adder using number of Full adder circuits connected in Cascade, i.e, the Carry output of each adder is connected to the Carry input of the next higher-order adder.

4-bit Parallel Adder using Full adders:

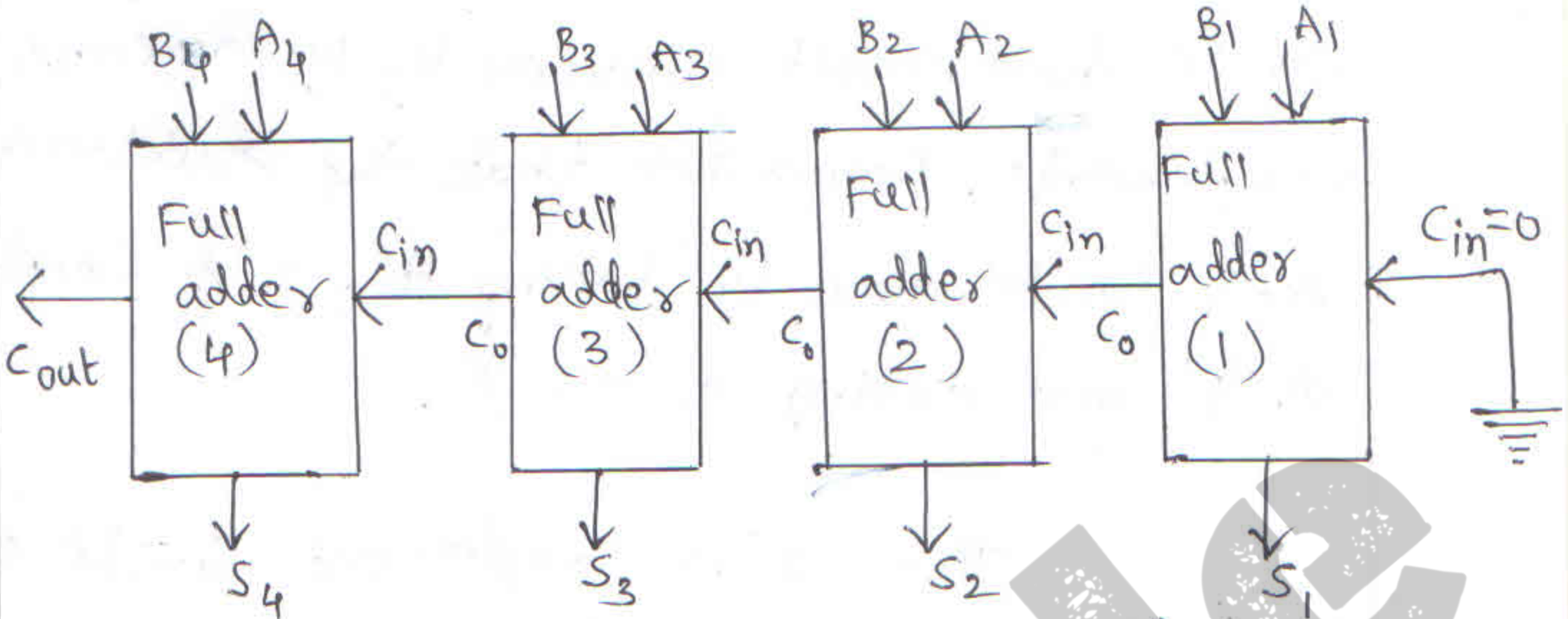


Fig.(2): Block diagram of 4-bit Parallel Adder.

The Fig(2) shows the block diagram and here, the LSB position, input Carry of Full-adder is made '0'.

Parallel Subtractor :

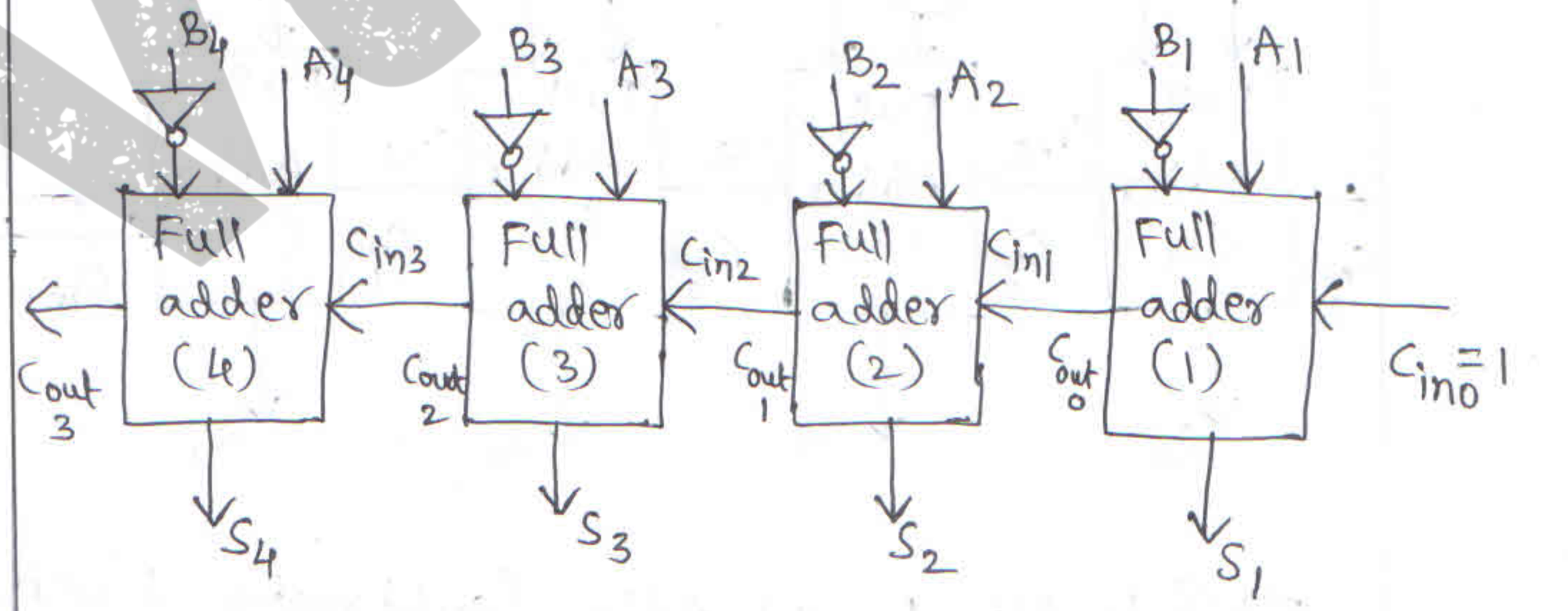


Fig.(3): Block diagram of 4-bit Parallel Subtractor

The Subtraction of binary numbers can be done most conveniently by means of Complements. Remember that the subtraction A-B can be done by taking the 2's Complement of 'B' and adding it to 'A'.

The 2's Complement can be obtained by taking the 1's Complement and adding '1' to the Least Significant pair of bits. The 1's Complement can be implemented with inverters (NOT Gate) and a '1' can be added to the sum through the input Carry as shown in the Fig. (3).

Parallel Adder/Subtractor :

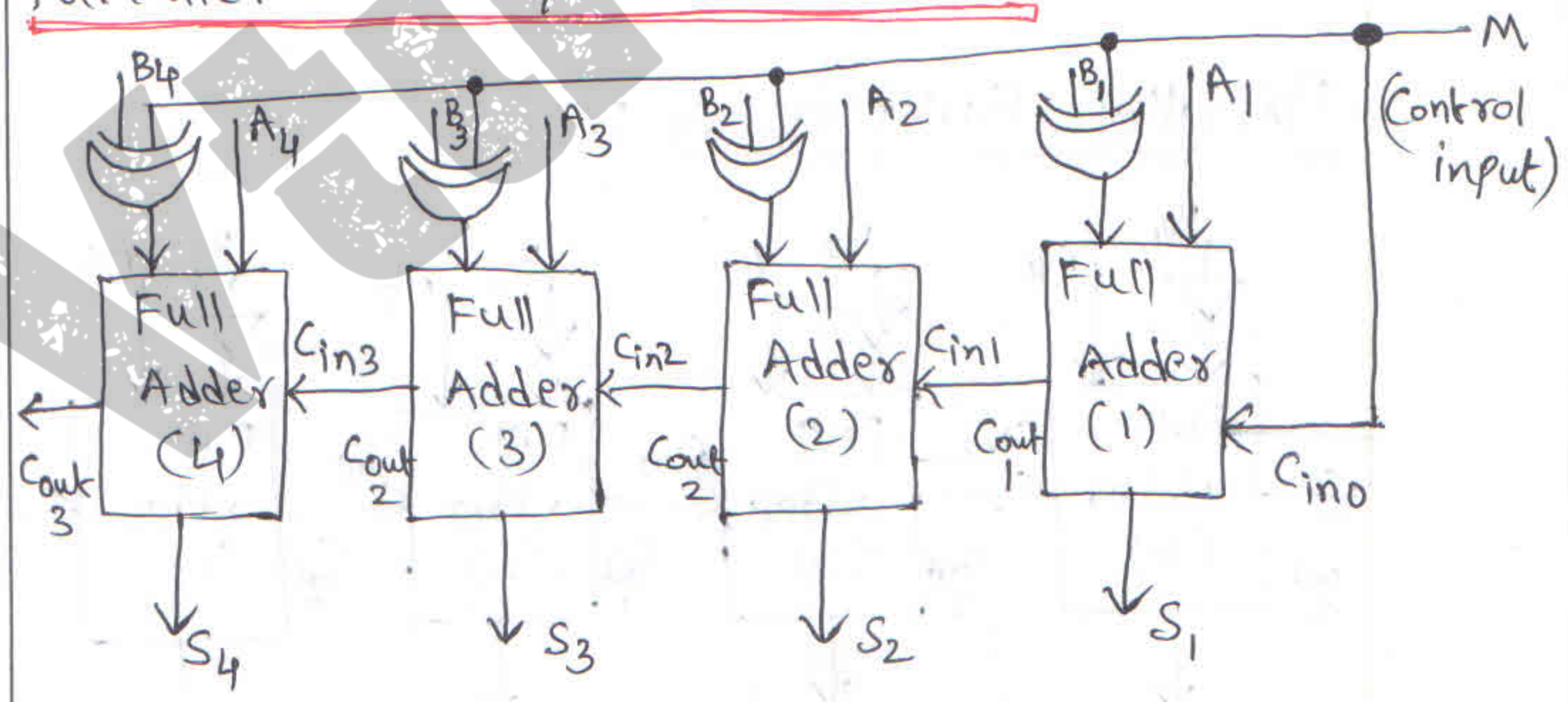


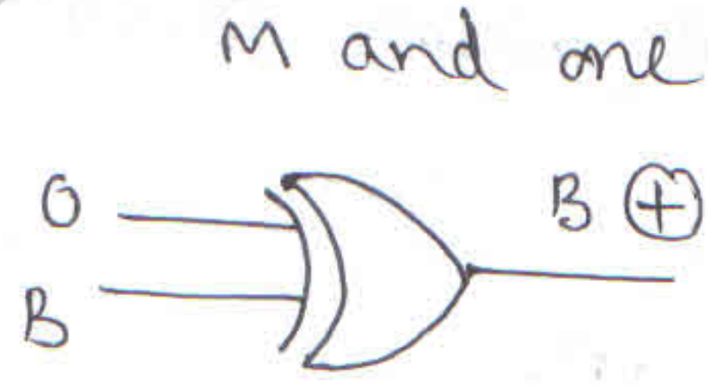
Fig. (4) 4-bit Parallel Adder/Subtractor Block diagram.

The addition and subtraction operations can be combined into one circuit with common binary-adder. This is done by including an Ex-OR gate with each Full adder as shown in the Fig. (4). The mode input (M) controls the operation of the circuit

Case (i): when M = 1, The circuit becomes a subtractor. Each Ex-OR gate receives input M and one of the inputs of B.

The Ex-OR gate output ($\bar{B}$), gives the 1's complement and the input Carry (C_{in_0}) is 1, and circuit performs A−B.

Case (ii): when M = 0, The circuit becomes an adder. Each Ex-OR gate receives input M and one of the inputs of B.



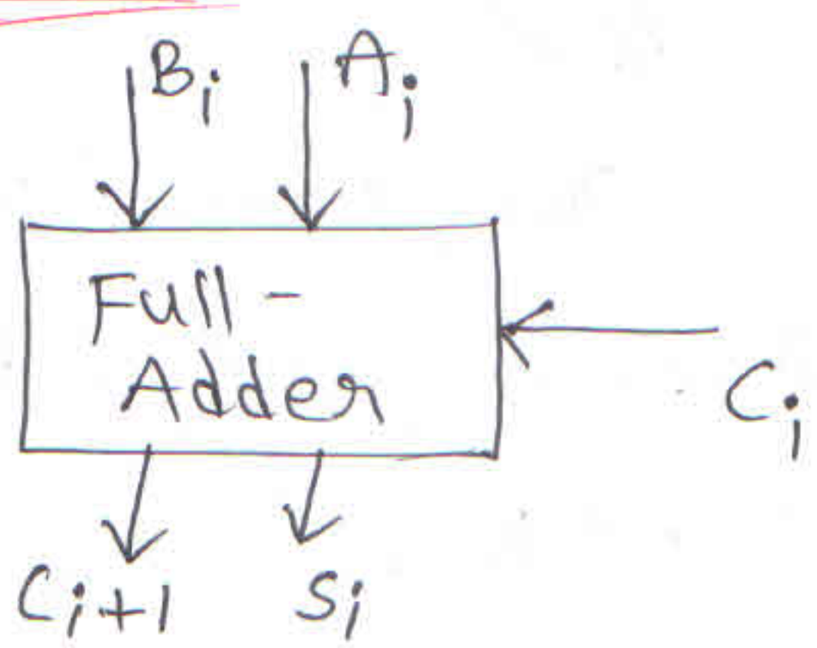
The Ex-OR gate output (B) and the input Carry (C_{in_0}) is 0, and circuit performs A+B.

The parallel adder is ripple Carry adder in which the Carry output of each Full-Adder stage is connected to the Carry input of the next higher-order stage. Therefore, the Sum and Carry outputs of any stage Cannot be produced until Carry occurs, this leads to a time delay in the addition process. This delay is Known as Carry propagation delay.

Look-Ahead Carry Adder:

One method of speeding up this process by eliminating inter stage Carry delay is called Look-Ahead Carry addition. It uses two functions: Carry generate and Carry Propagate.

Consider the full adder circuit is shown in the below figure.



The "Sum" expression is,

$$S_i = A_i \oplus B_i \oplus C_i \longrightarrow (1)$$

and Carry expression is,

$$C_{i+1} = A_i B_i + A_i C_i + B_i C_i \longrightarrow (2)$$

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

The term $A_i B_i$ relates to the Carry formed at the i^{th} stage and is referred to as the Carry generate function,

$$g_i = A_i B_i \longrightarrow (3)$$

The term $(A_i + B_i)$ relates to the carry 'C_i' generated at the previous stage and thus $(A_i + B_i)$ is referred to as the Carry Propagate function,

$$P_i = A_i + B_i \longrightarrow (4)$$

the equation (2), becomes

$$\therefore C_{i+1} = g_i + C_i P_i \longrightarrow (5)$$

Now, let us look at the Carry generated at every stage of the 4-bit Parallel Adder, by setting $i=0$ in equation (5),

$$\boxed{C_1 = g_0 + C_0 P_0} \longrightarrow (6)$$

and setting $i=1$, in equation (2), we get

$$C_2 = g_1 + C_1 P_1$$

$$C_2 = g_1 + (g_0 + C_0 P_0) P_1$$

$$\boxed{C_2 = g_1 + g_0 P_1 + C_0 P_0 P_1} \longrightarrow (7)$$

The C_1 and C_2 are functions of only the parallel inputs and C_0.

Similarly, $i=2$, in equation (5), we get

$$C_3 = g_2 + C_2 P_2$$

$$C_3 = g_2 + (g_1 + g_0 P_1 + C_0 P_0 P_1) P_2$$

$$\boxed{C_3 = g_2 + g_1 P_2 + g_0 P_1 P_2 + C_0 P_0 P_1 P_2} \longrightarrow (8)$$

and ($\because i=3$) $\rightarrow$

$$C_4 = g_3 + C_3 P_3$$

$$C_4 = g_3 + (g_2 + g_1 P_2 + g_0 P_1 P_2 + C_0 P_0 P_1 P_2) P_3$$

$$\boxed{C_4 = g_3 + g_2 P_3 + g_1 P_2 P_3 + g_0 P_1 P_2 P_3 + C_0 P_0 P_1 P_2 P_3} \longrightarrow (9)$$

Pothi Reddy.K

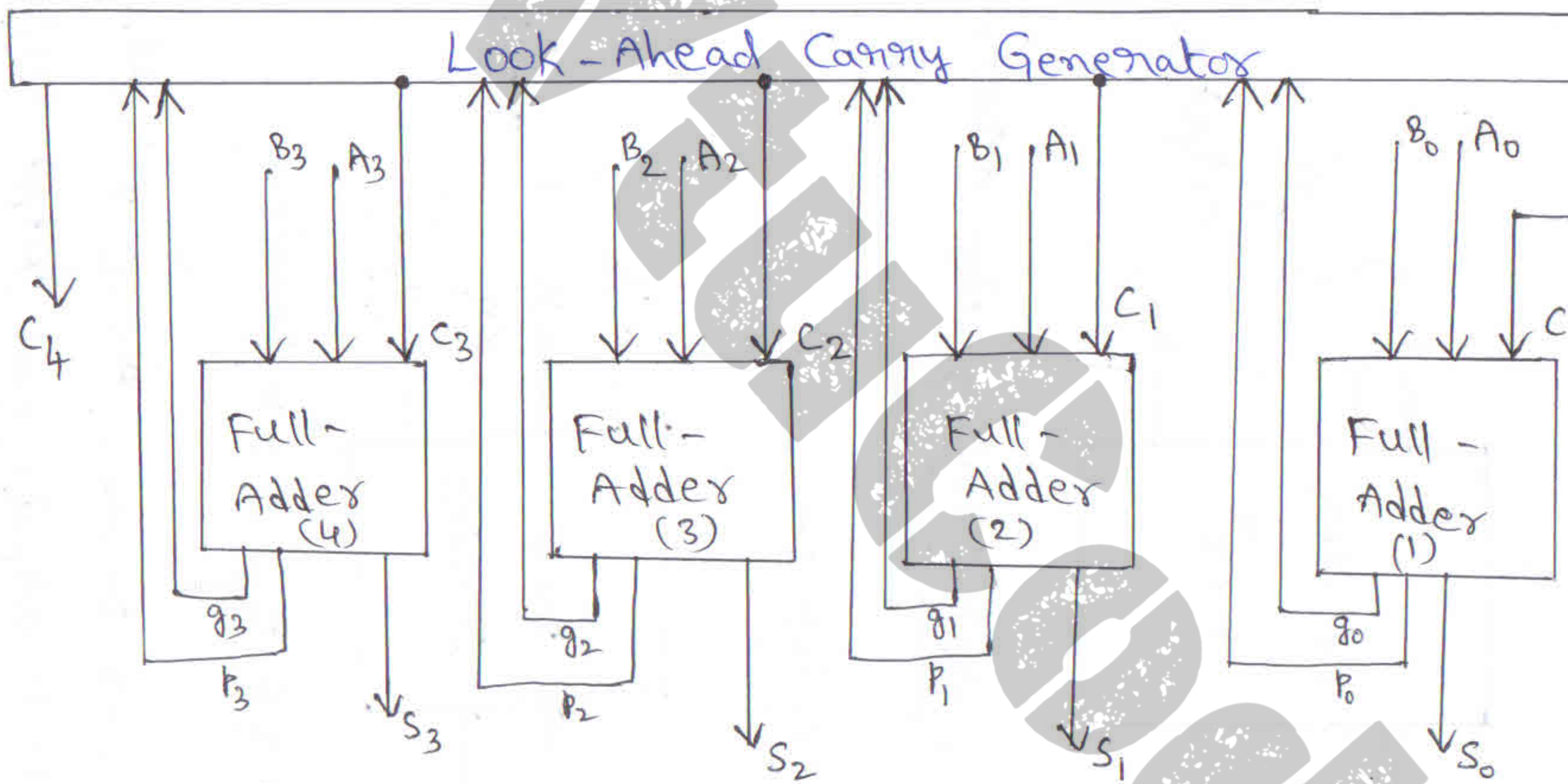


Fig:(5). The 4-bit Parallel Fast Look-Ahead Carry-Adder.

The C_3 and C_4 are also functions of only the parallel inputs and C_0.

Decoders :

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

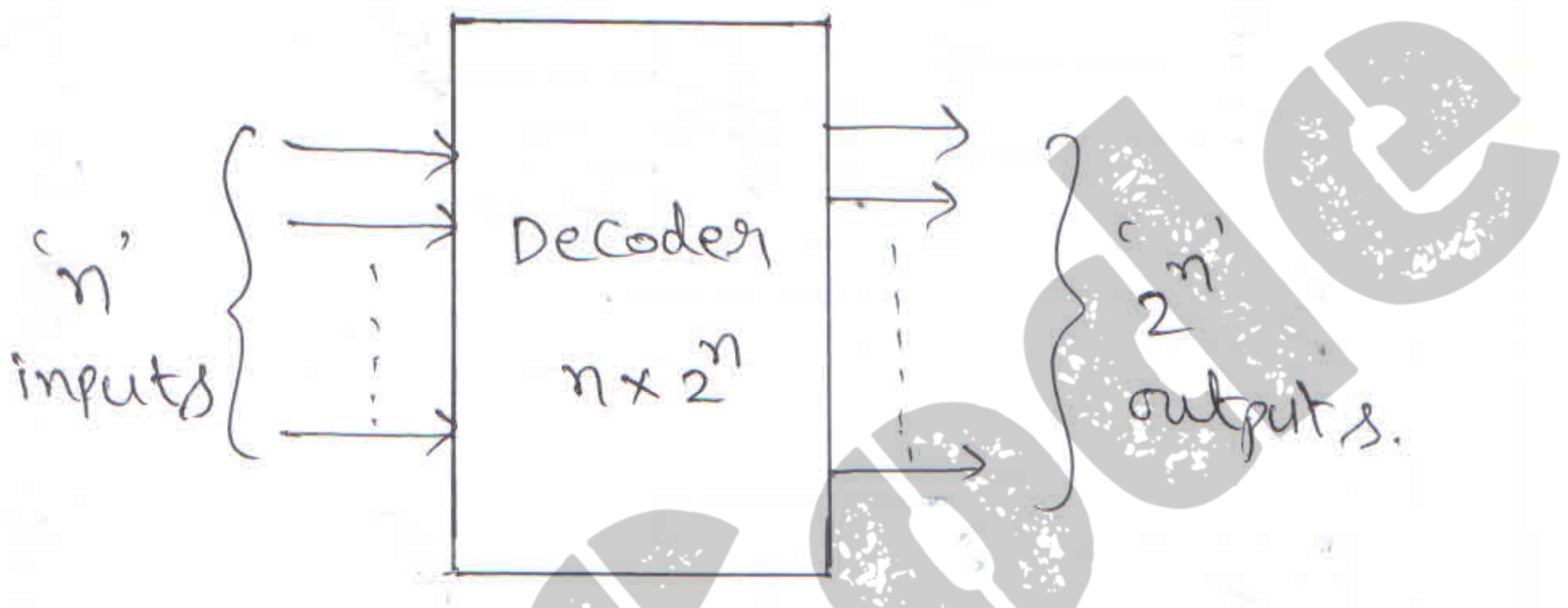


Fig (1). Block Diagram of a Decoder.

2 × 4 Decoder :-

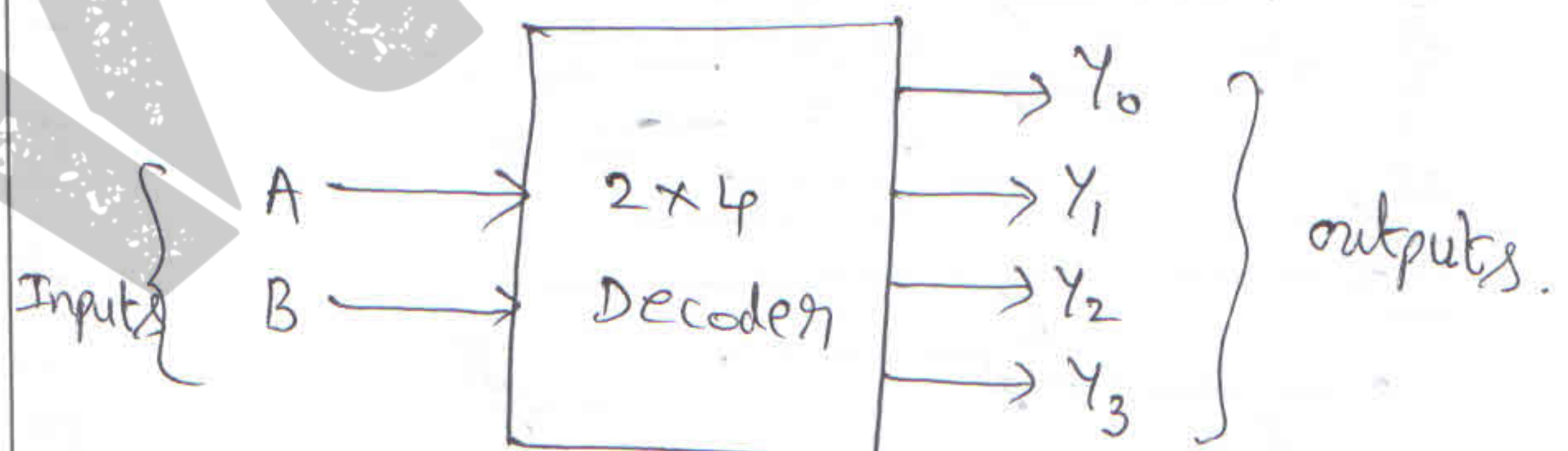


Fig. (2) The 2 × 4 Decoder.

the 2 × 4 Decoder, Here 2 inputs are decoded into 4 outputs, each output represent

one of the minterms of the 2-input variables.

Truth Table for a 2 to 4 Decoder :-

Inputs		outputs.			
A	B	Y_0	Y_1	Y_2	Y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

The output expressions are,

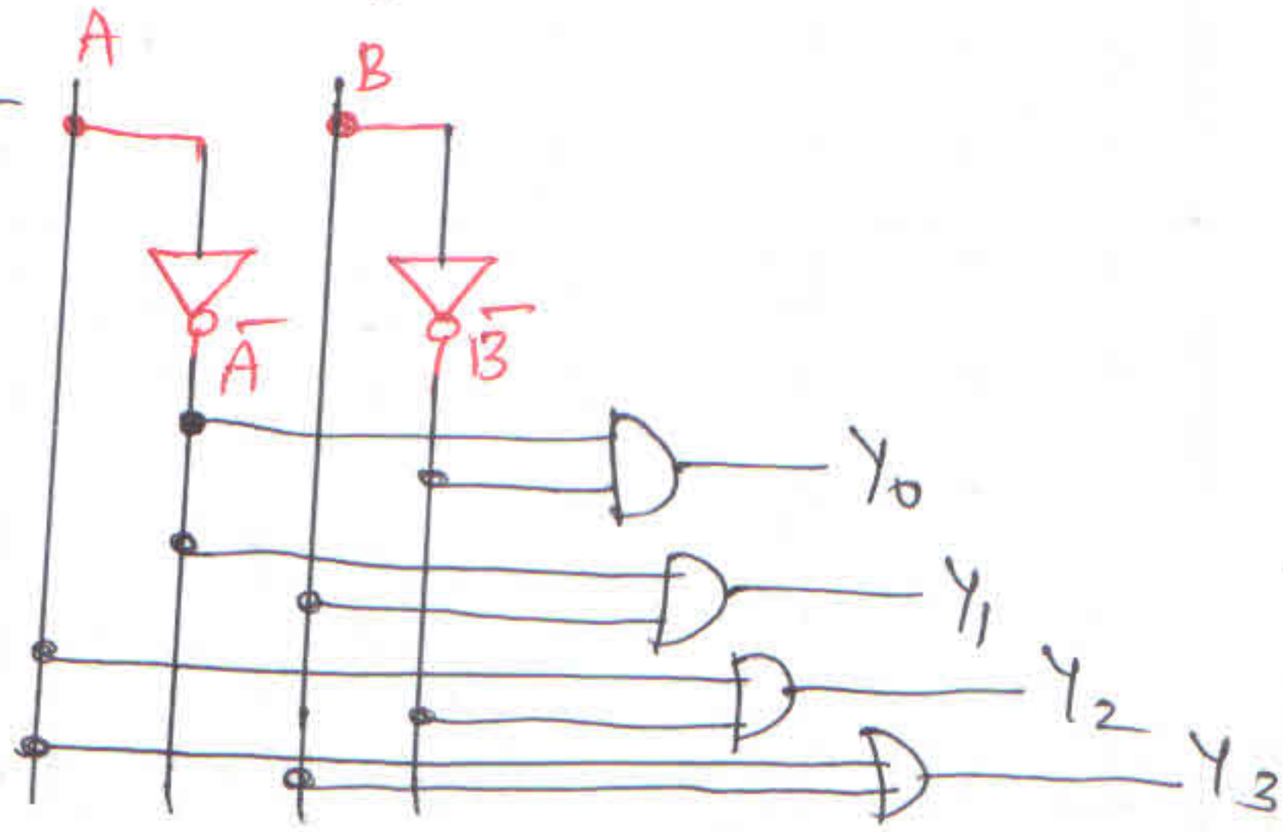
$Y_0 = \bar{A}\bar{B}$,

$Y_1 = \bar{A}B$,

$Y_2 = A\bar{B}$,

and $Y_3 = AB$

Logic Diagram :-



3 to 8 Decoder :

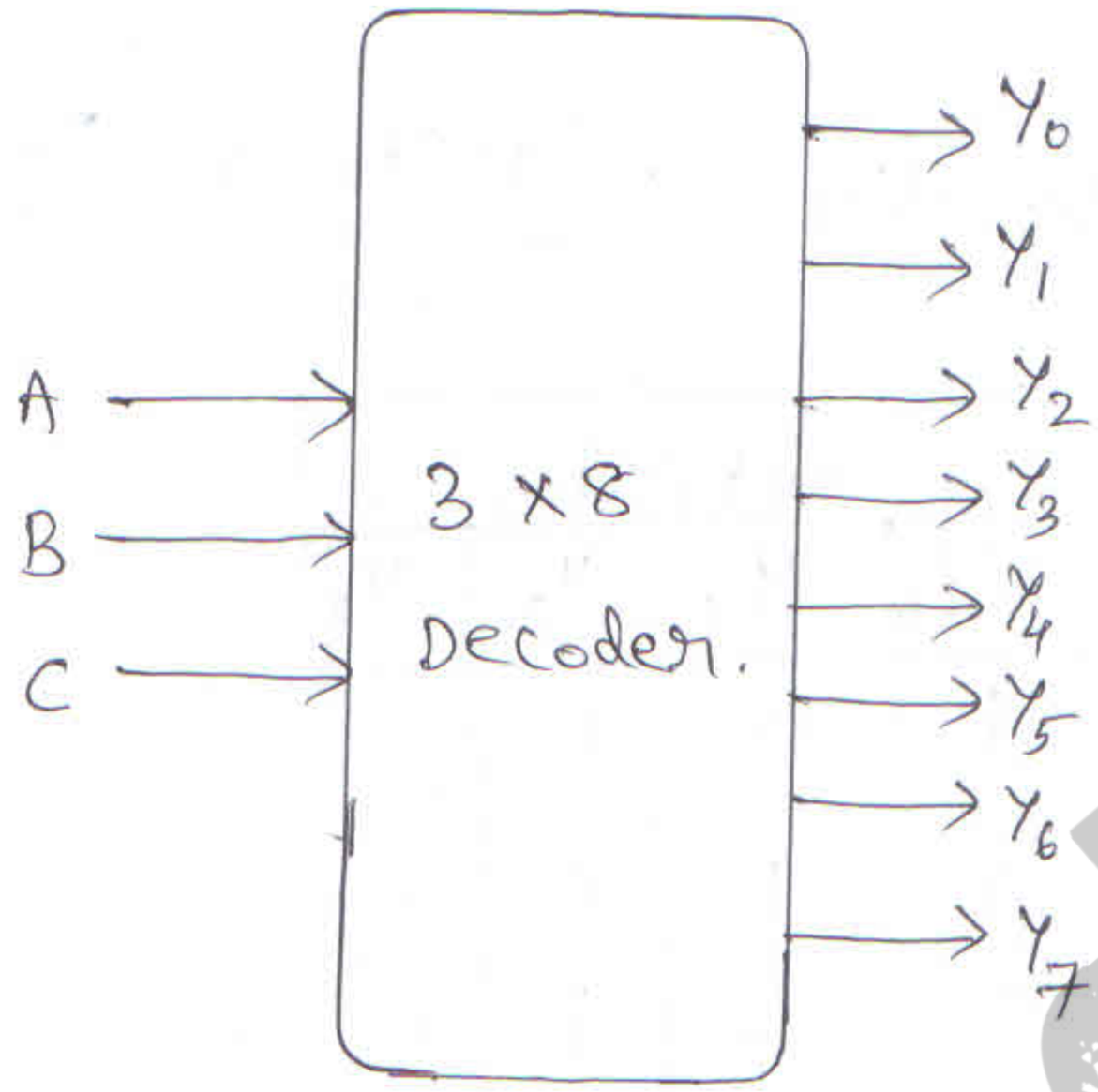


Fig.①. the 3x8 Decoder

The 3 x 8 decoder, here 3 inputs are decoded into 8 outputs, each output represent one of the minterms of the 3-input variables.

Inputs			outputs.							
A	B	C	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

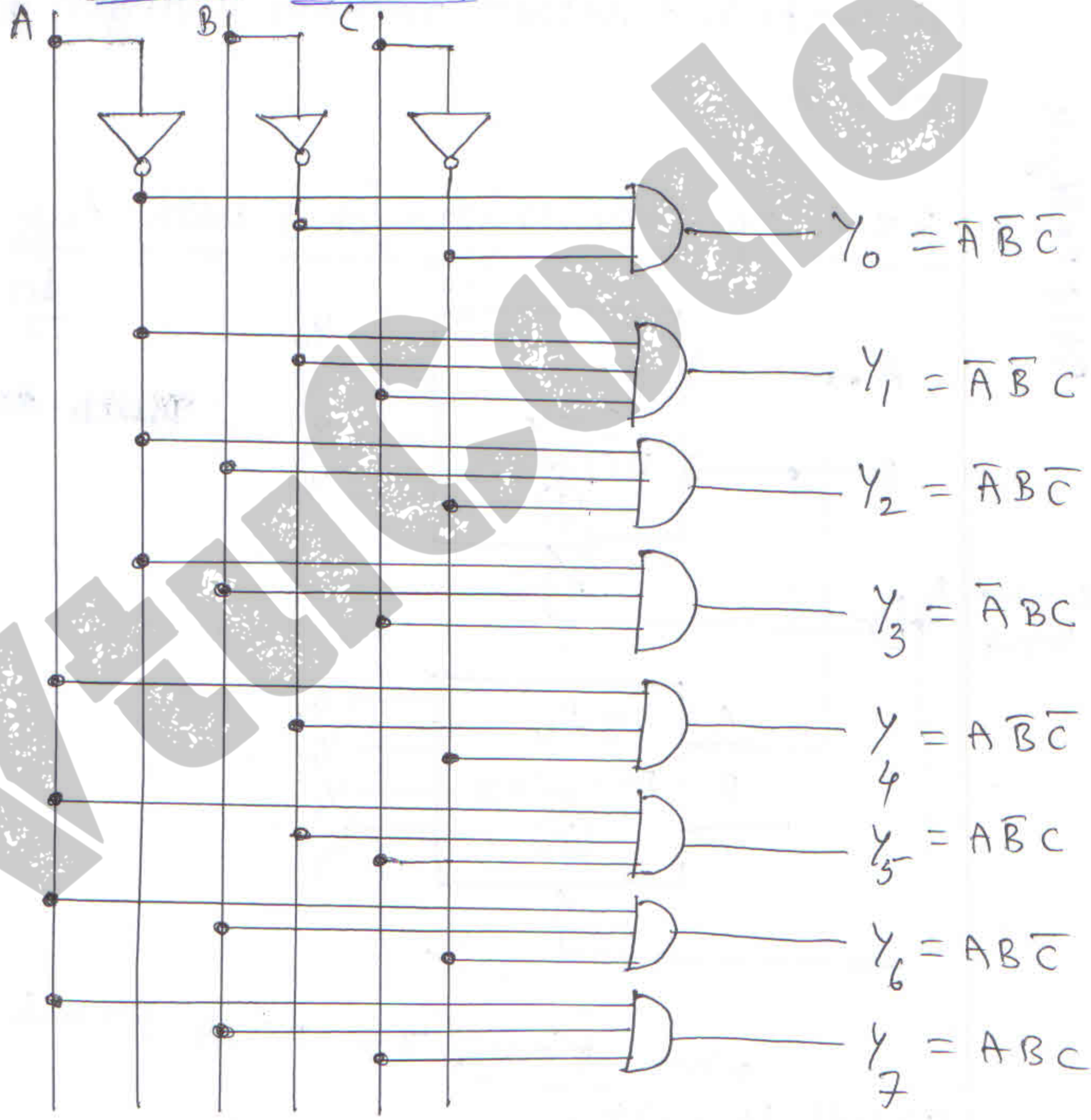
Truth Table for 3x8 Decoder

The output expressions are,

$Y_0 = \bar{A}\bar{B}\bar{C}$, $Y_1 = \bar{A}\bar{B}C$, $Y_2 = \bar{A}B\bar{C}$, $Y_3 = \bar{A}BC$

$Y_4 = A\bar{B}\bar{C}$, $Y_5 = A\bar{B}C$, $Y_6 = AB\bar{C}$ & $Y_7 = ABC$

The Logic Diagram :-

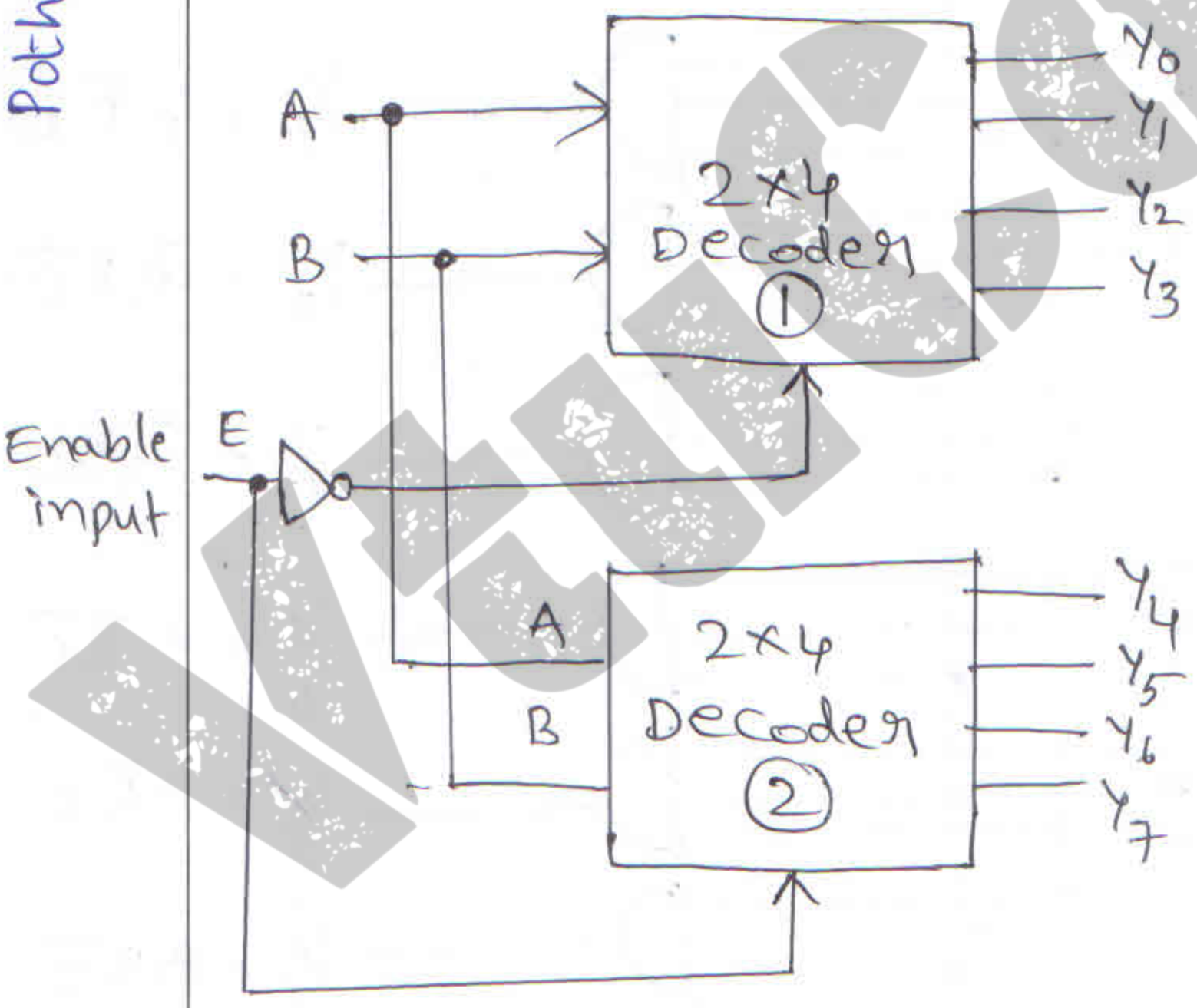


Cascading Binary Decoders:

Binary decoder circuits can be connected together to form a larger decoder-circuit.

Decoders with Enable (E) inputs can be connected together to form a larger decoder circuit.

3 x 8 Decoder Constructed with two 2x4 decoders:-



when E = 0, The First 2x4 decoder circuit is Active.

when E = 1, The second 2x4 decoder circuit is Active.

4x16 Decoder Constructed with two 3x8 decoders:-

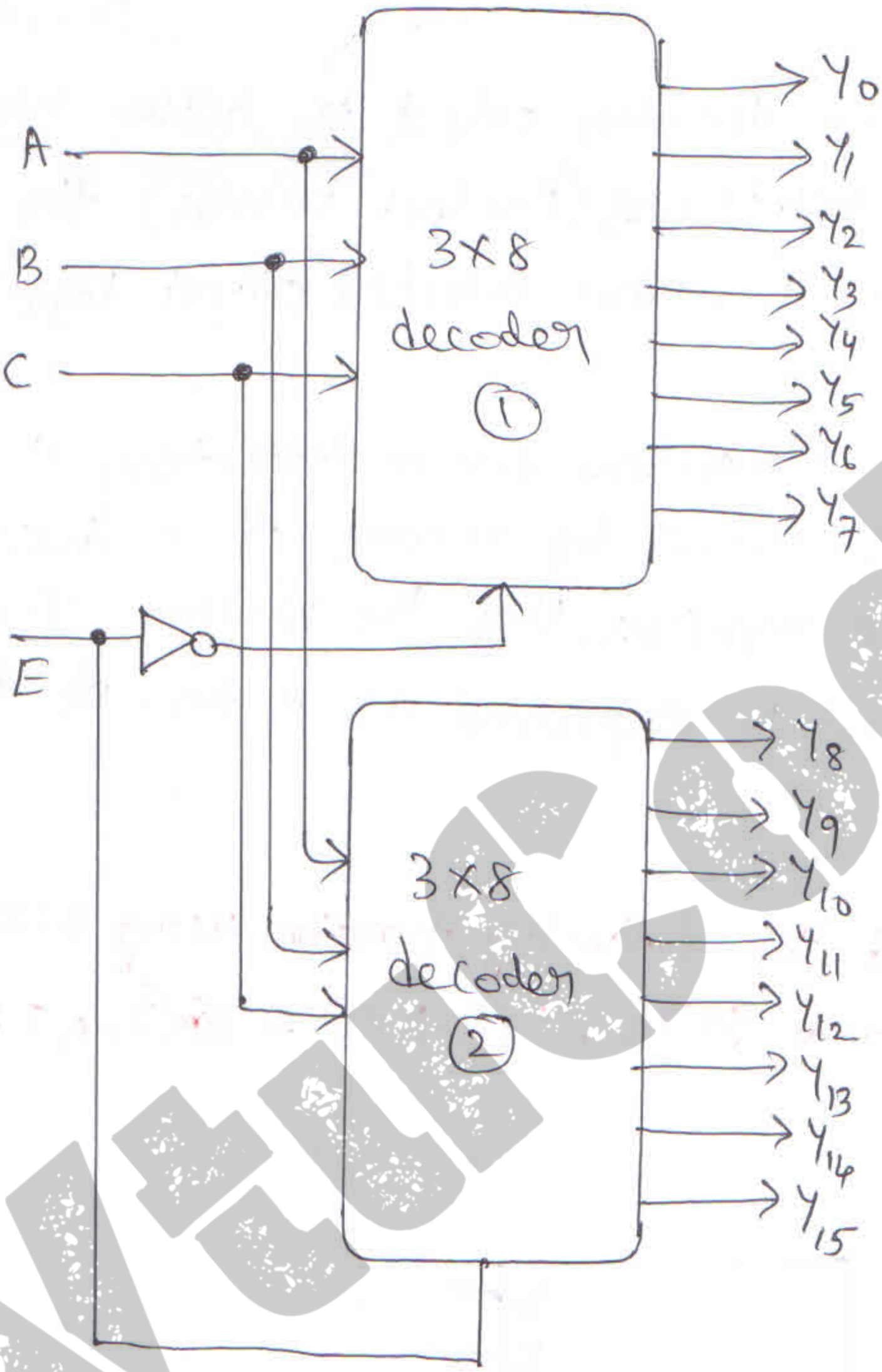


Fig:- The 4x16 Decoder circuit.

Implementation of Combinational Logic using Decoder :-

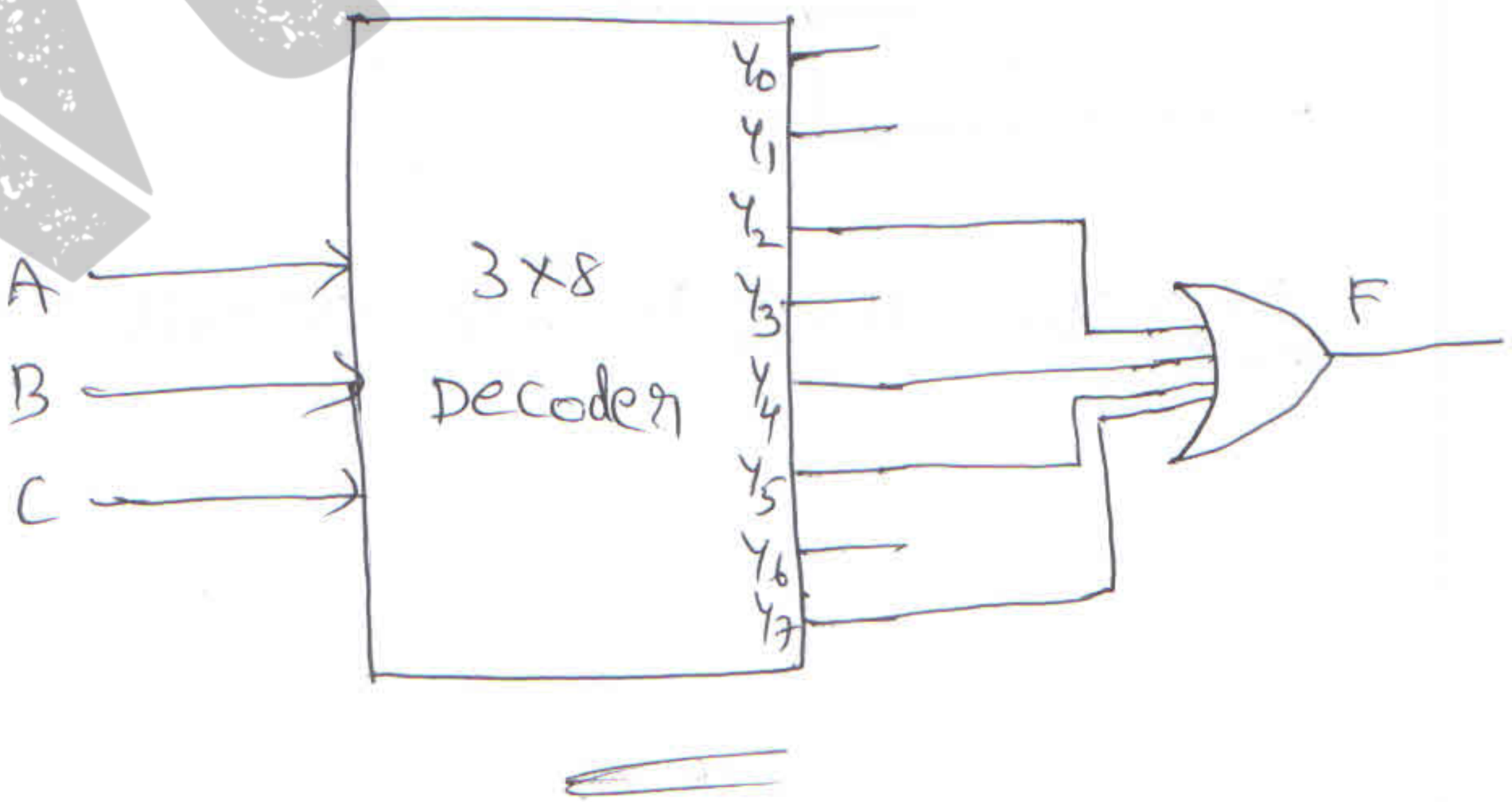
when decoder output is Active high, it generates minterms (Product terms) for input variables; i.e. it makes selected output logic '1'.

The procedure for implementing a combinational circuit by means of a decoder and OR gates requires that the Boolean function for the circuit be expressed as a sum of minterms.

Examples:

① Implement the following function using 3:8 decoder and external gates: $F(A,B,C) = \Sigma m(2,4,5,7)$.

Sol(n)



(2) Implement the Full-adder using a decoder.

Sol) Truth Table (Full-Adder) :-

Inputs			outputs	
A	B	C_{in}	Cout	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

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

$$C_{out} = \Sigma m(3, 5, 6, 7)$$

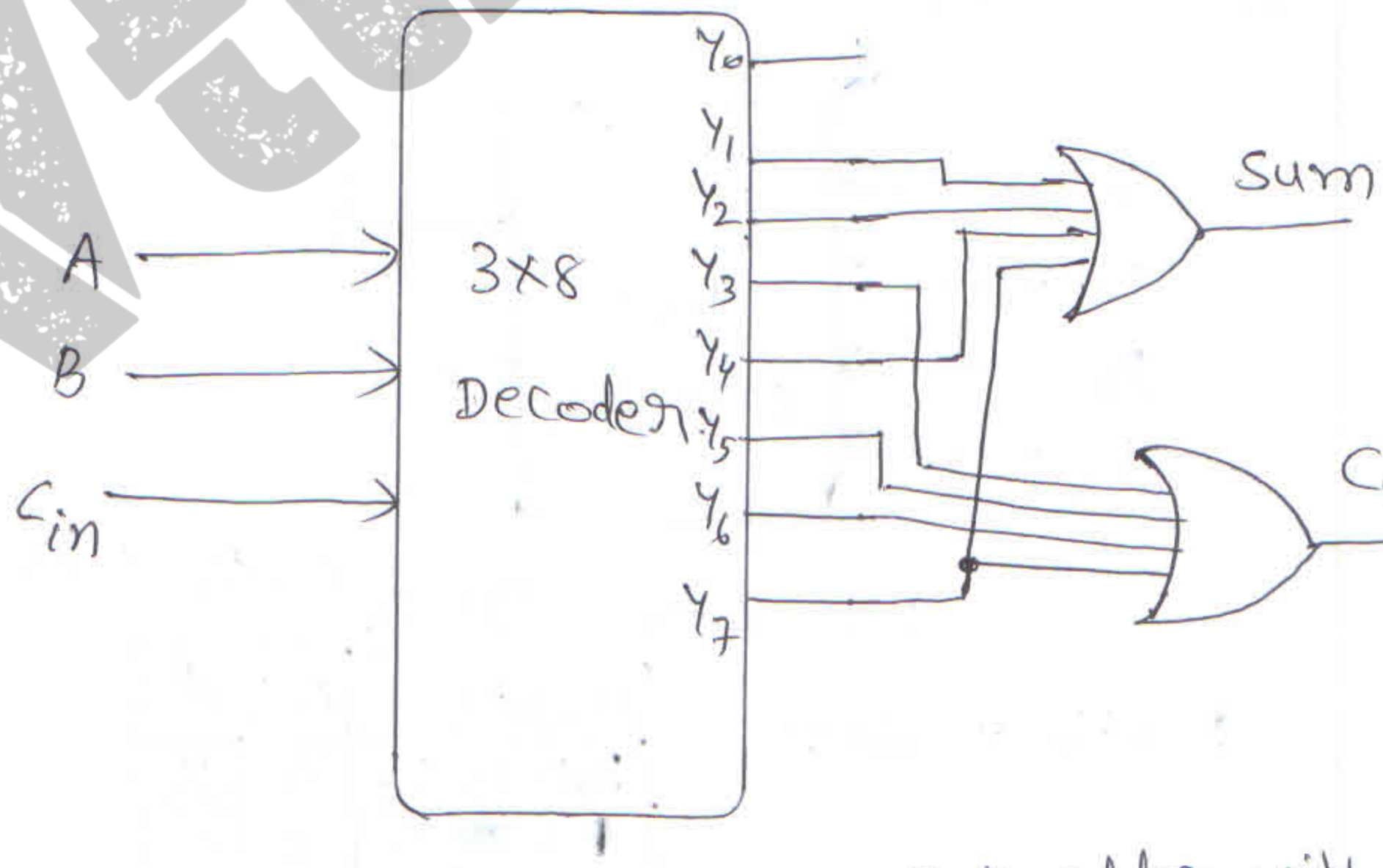


Fig:- Implementation of a Full adder with a decoder.

Encoders :-

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

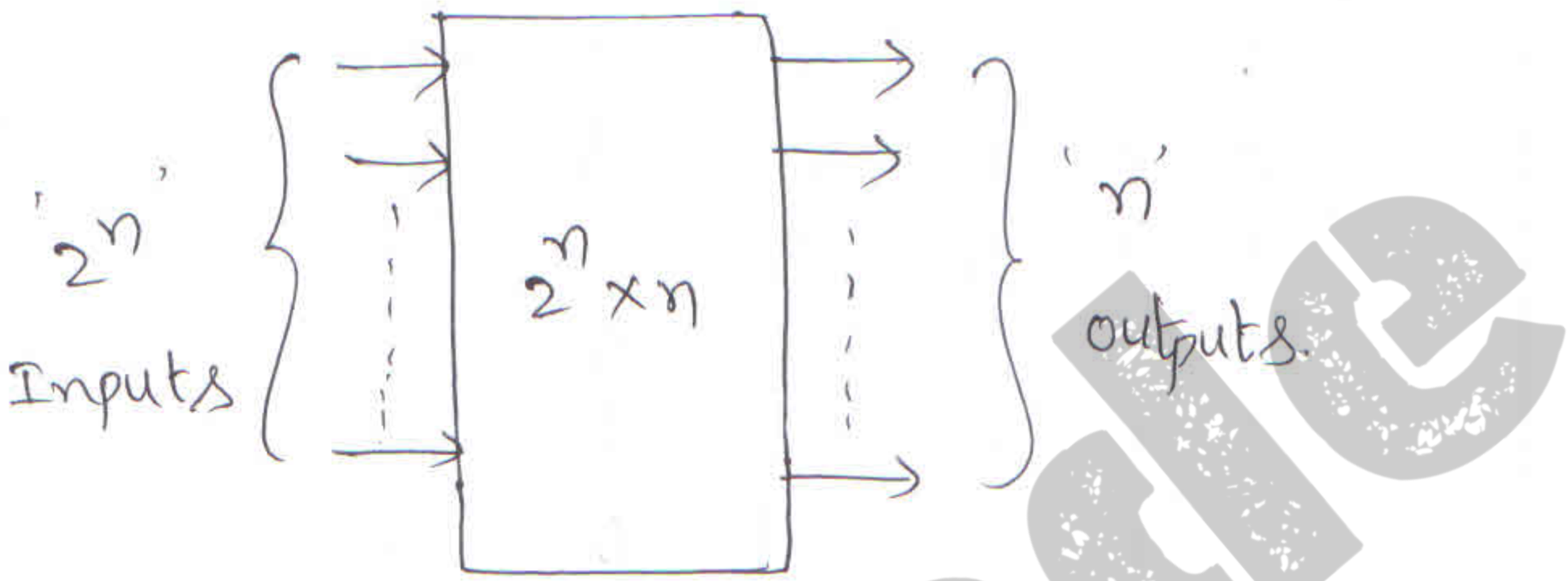


Fig. (1) Block diagram of an Encoder.

An Encoder has 2^n (or fewer) inputs and 'n' output.

4 x 2 Encoder :

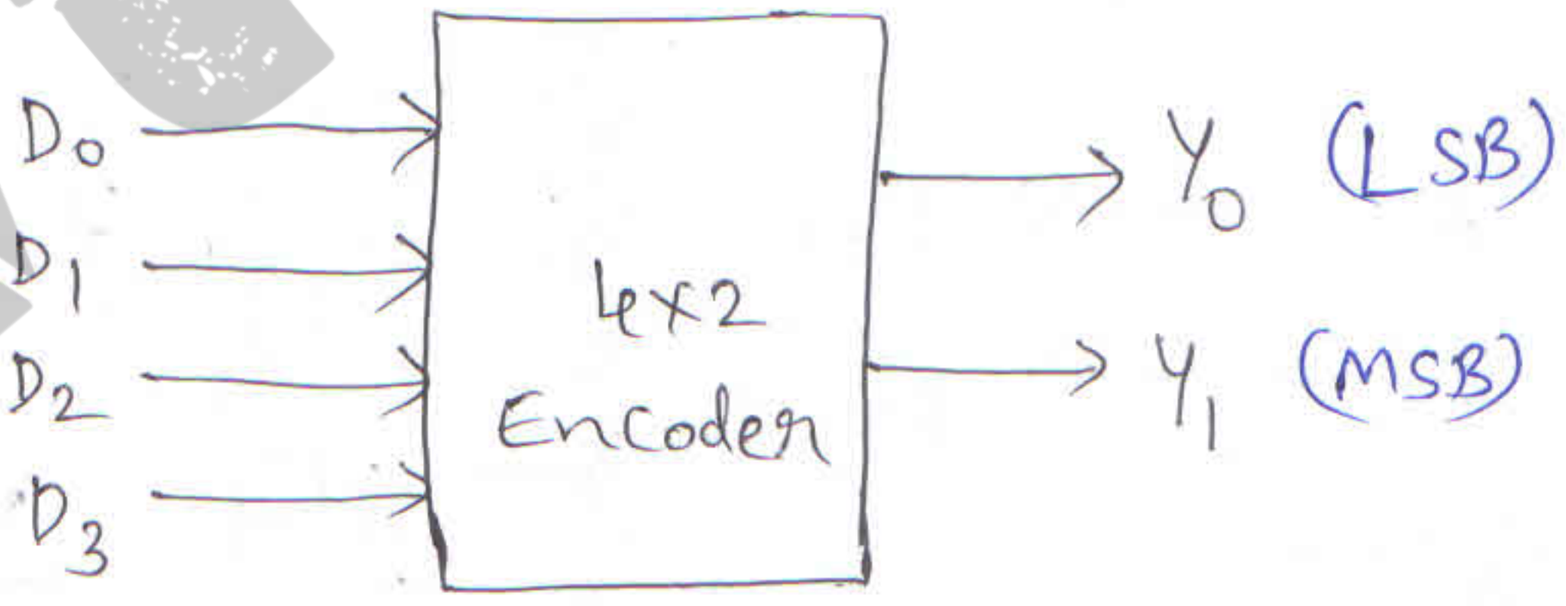


Fig:- Logic Symbol for 4 x 2 Encoder

Truth Table :-

Inputs				outputs	
D_0	D_1	D_2	D_3	Y_1	Y_0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

Octal to Binary Encoder : (8×3 Encoder)

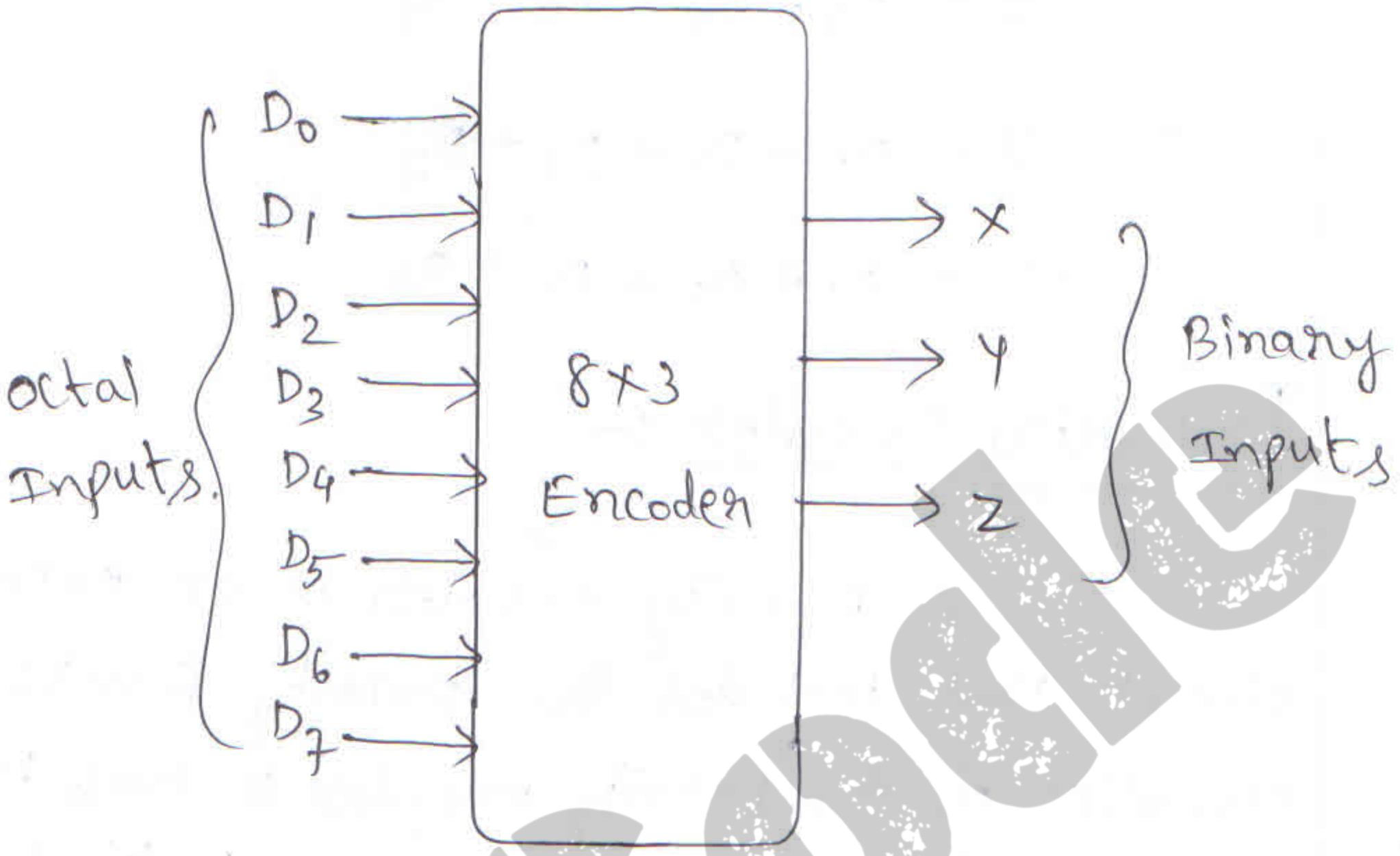


Fig. the 8×3 Encoder.

Truth Table :

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

The Boolean output functions,

$$X = D_4 + D_5 + D_6 + D_7$$

$$Y = D_2 + D_3 + D_6 + D_7$$

$$Z = D_1 + D_3 + D_5 + D_7$$

Priority Encoder :-

A priority encoder is an encoder circuit that includes the priority function. The operation of the priority encoder is such that if two or more inputs are equal to '1' at the same time, the input having the highest - priority will take precedence.

For Example :-

① write the Condensed truth table for a 4x2 - line priority encoder with a Valid output, where the highest priority is given to the highest bit position or input with highest index and obtain the minimal Sum expressions for the outputs.

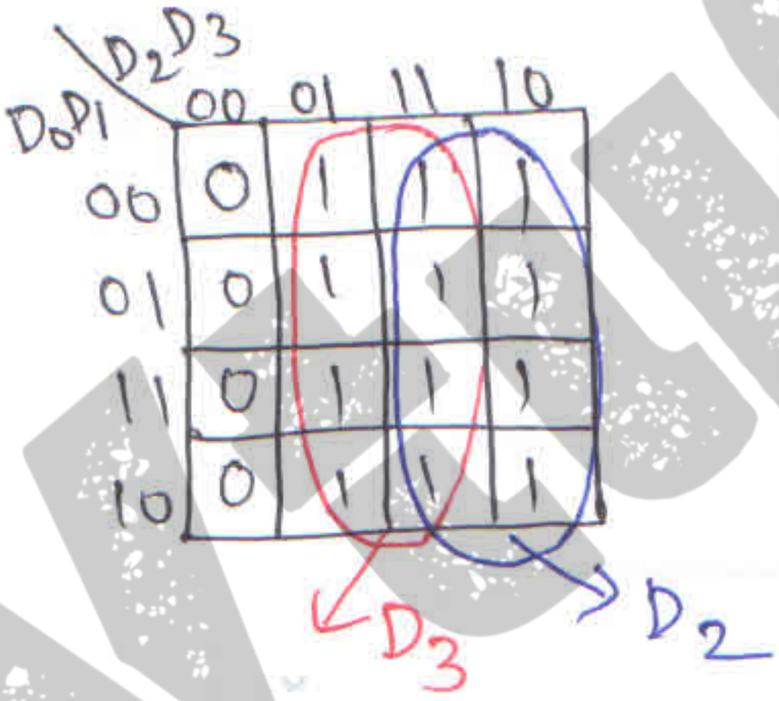
Sol)) the higher the subscript (index) number, the higher the priority of the input. Input D_3 has the highest priority.

Truth Table :-

Cell. no.	Inputs				outputs		
	D_0	D_1	D_2	D_3 (MP)	Y_1	Y_0	V (Valid o/p)
0	0	0	0	0	0	0	0
8	1	0	0	0	0	0	1
4, 12	X	1	0	0	0	1	1
2, 6, 10, 14	X	X	1	0	1	0	1
1, 3, 5, 7, 9, 11, 13, 15.	X	X	X	1	1	1	1

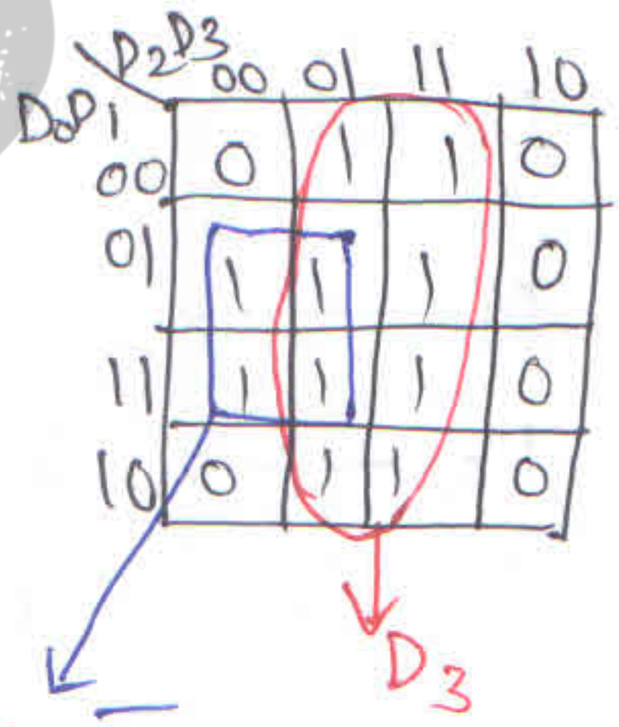
K-maps for outputs :-

For Y_1 :



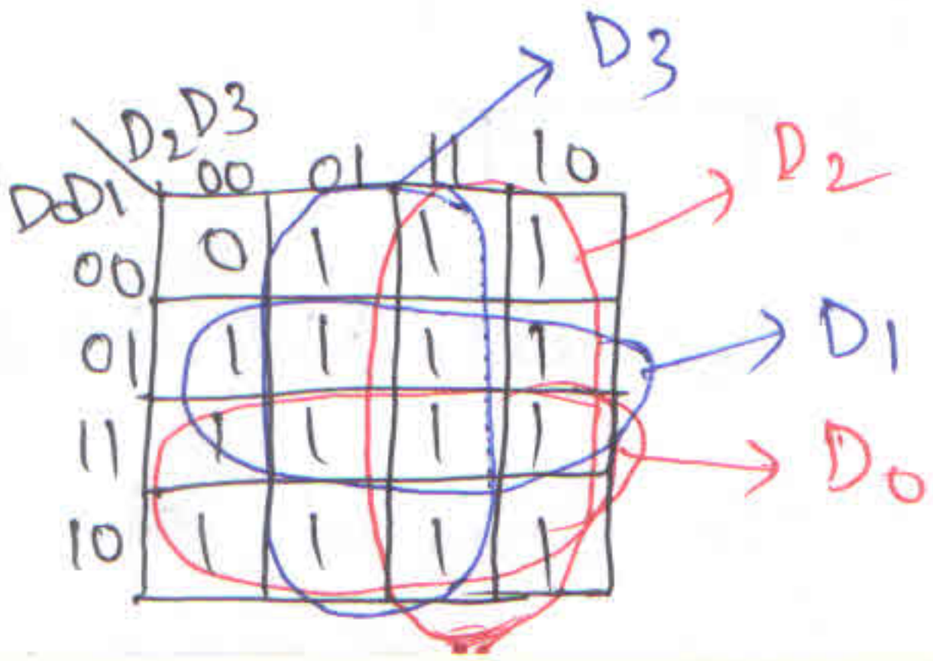
$$Y_1 = D_2 + D_3$$

For Y_0 :



$$Y_0 = D_3 + D_1 \bar{D}_2$$

For V :



$$V = D_0 + D_1 + D_2 + D_3$$

Multiplexers (MUX):

A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally, there are '2^n' input lines and 'n' selection lines, whose bit combinations determine which input is selected.

A multiplexer is also called a data-selector, since it selects one of many inputs and steers the binary information to the output line.

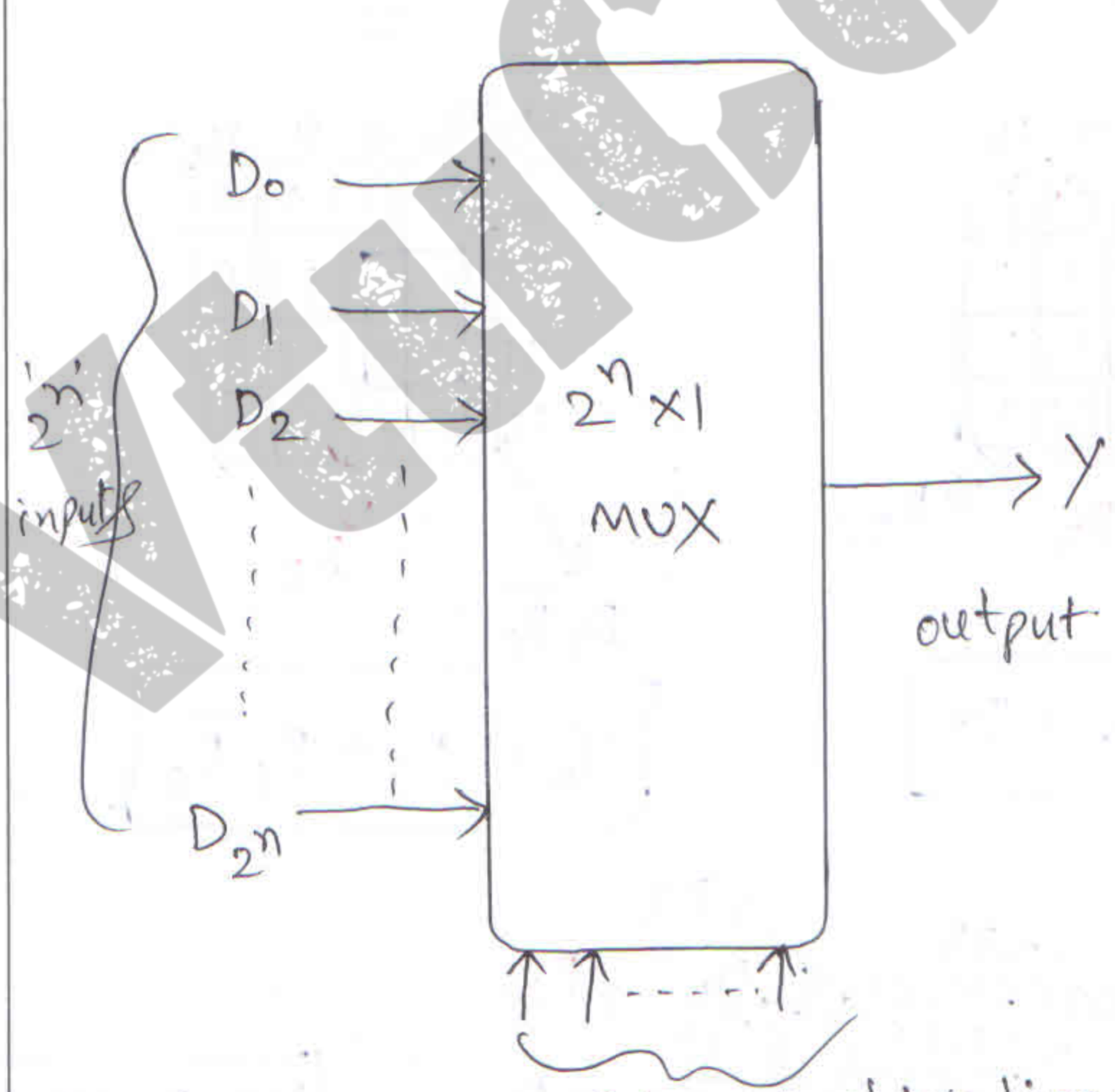


Fig:- Logic symbol of MUX.

2 x 1 MUX :

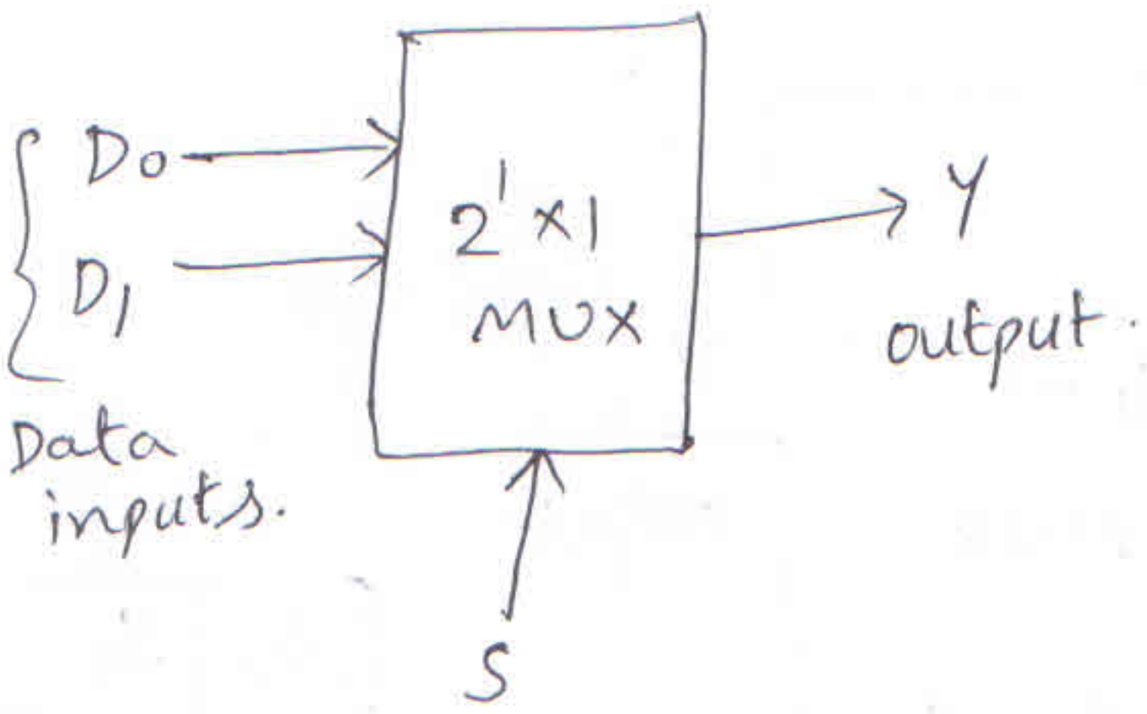


Fig. (1) Logic Symbol.

S	Y
0	D_0
1	D_1

Fig (2) Functional Table.

From the Functional Table, The output,

$$Y = \bar{S} D_0 + S D_1$$

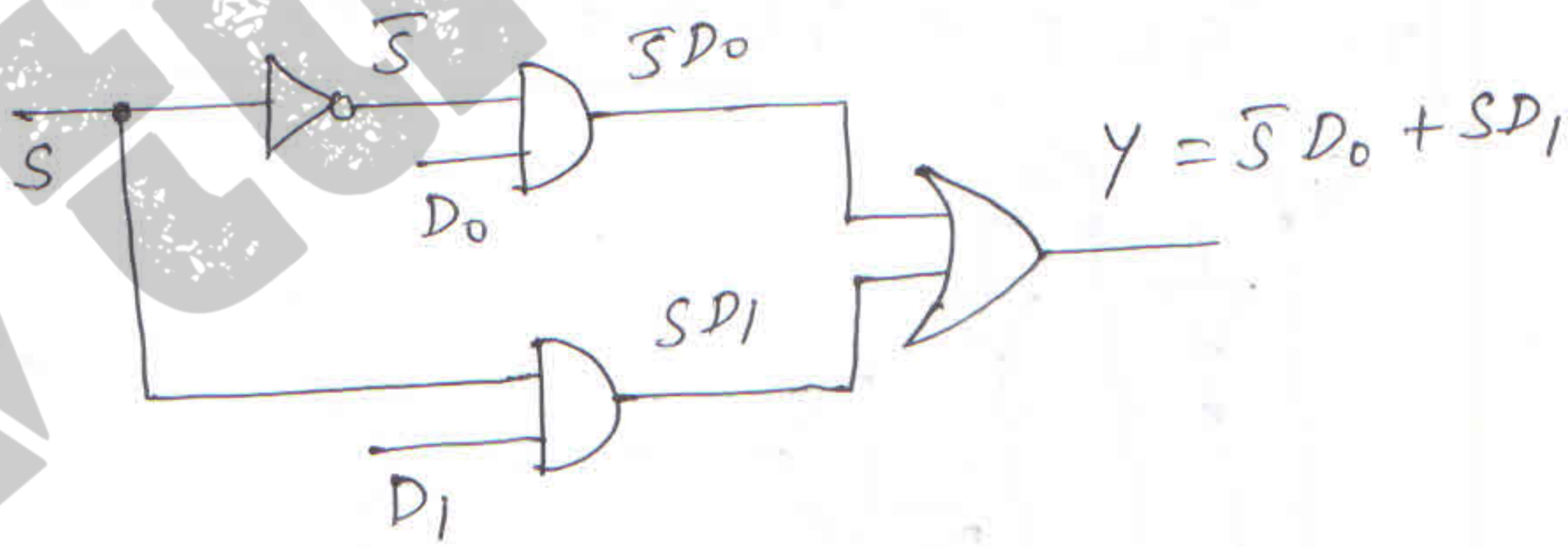


Fig (3) Logic diagram for 2 x 1 MUX.

Truth Table for 2 x 1 MUX.

S	D	Y
0	0	0
0	1	1
1	0	0
1	1	1

4 x 1 MUX :

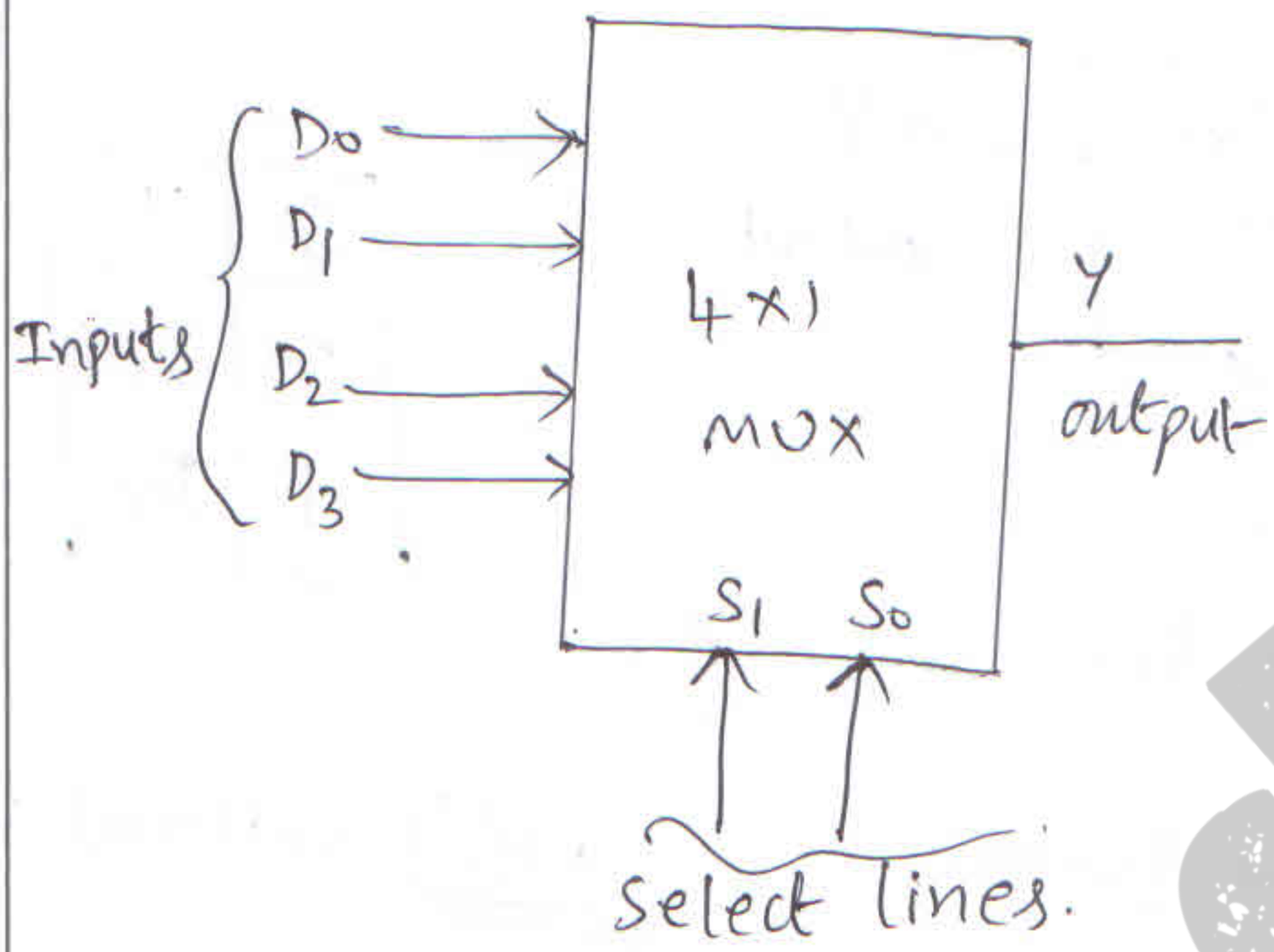


Fig. Logic Symbol.

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

Fig. Functional Table.

From the Functional Table, the output

$$Y = \bar{S}_1 \bar{S}_0 D_0 + \bar{S}_1 S_0 D_1 + S_1 \bar{S}_0 D_2 + S_1 S_0 D_3$$

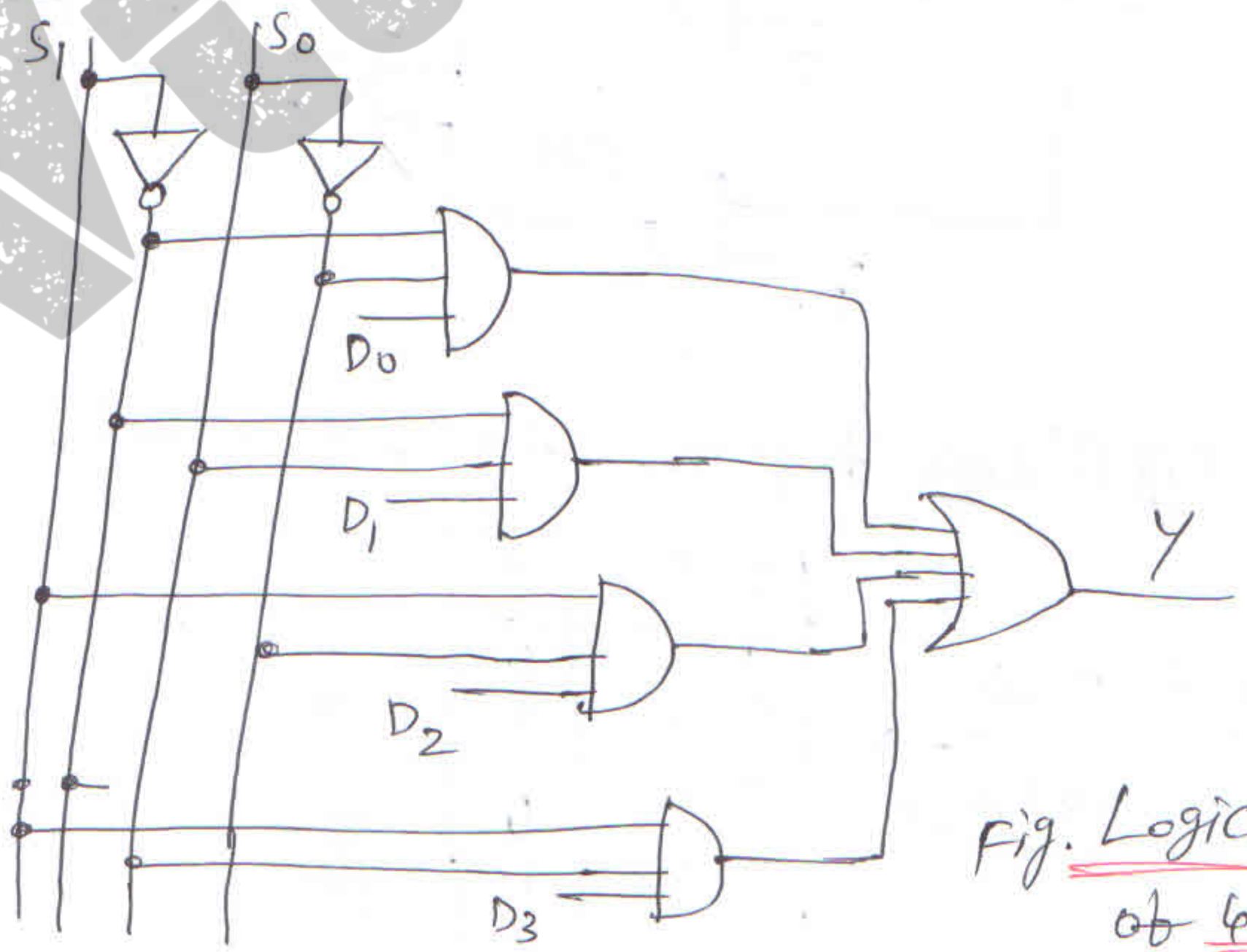


Fig. Logic diagram of 4x1 MUX.

Boolean Function Implementation (MUX):

A Multiplexer consists of a set of AND gates whose outputs are Connected to single OR gate. Because of this Construction ~~only~~ any Boolean function in a SOP form can be easily realized using multiplexer. Each AND gate in the MUX represents a minterm.

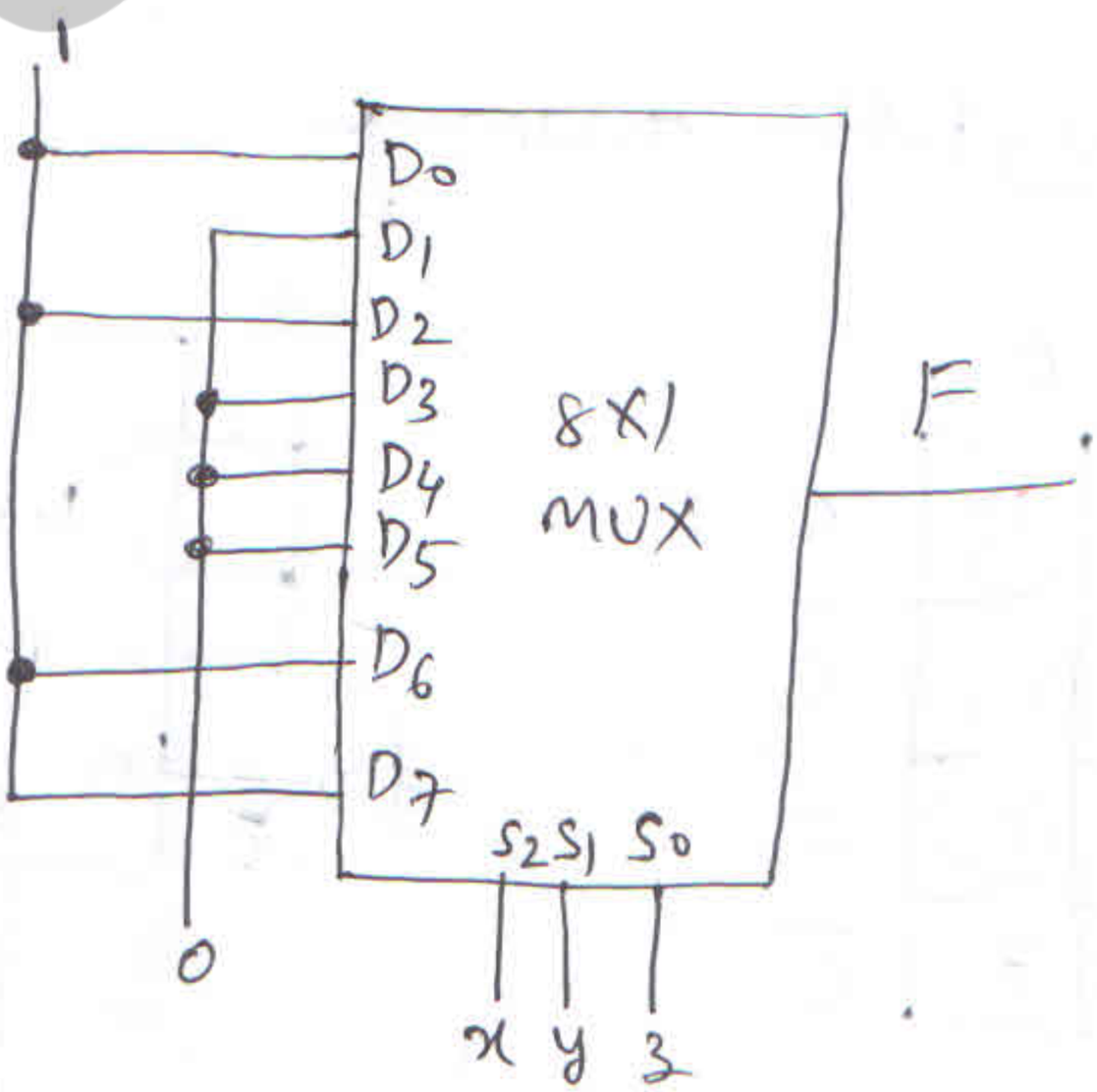
If a minterm exists in a function, we have to connect the AND gate data input to Logic '1'; otherwise we have to connect, it to Logic '0'.

Examples:

① Implement the given function using multiplexer.

$F(x,y,z) = \Sigma m(0,2,6,7)$.

Sol))



//.

(2) Implement the following Boolean function using 4 x 1 MUX.

$F(A,B,C) = \Sigma m\,(1,3,5,6)$.

Sol))

Implementation Table :-

	D_0	D_1	D_2	D_3
$\bar{A}$	0	(1)	2	(3)
A	4	(5)	(6)	7
	0	1	A	$\bar{A}$

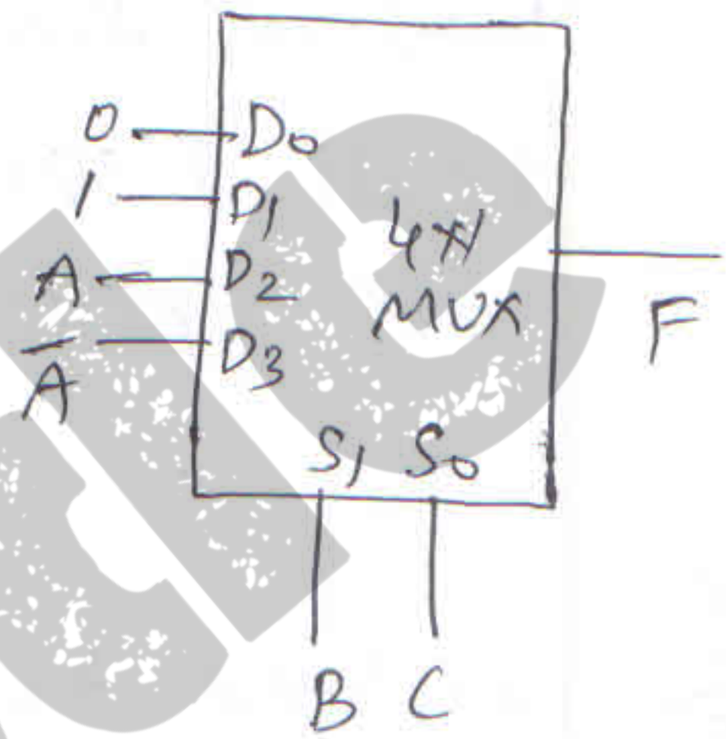


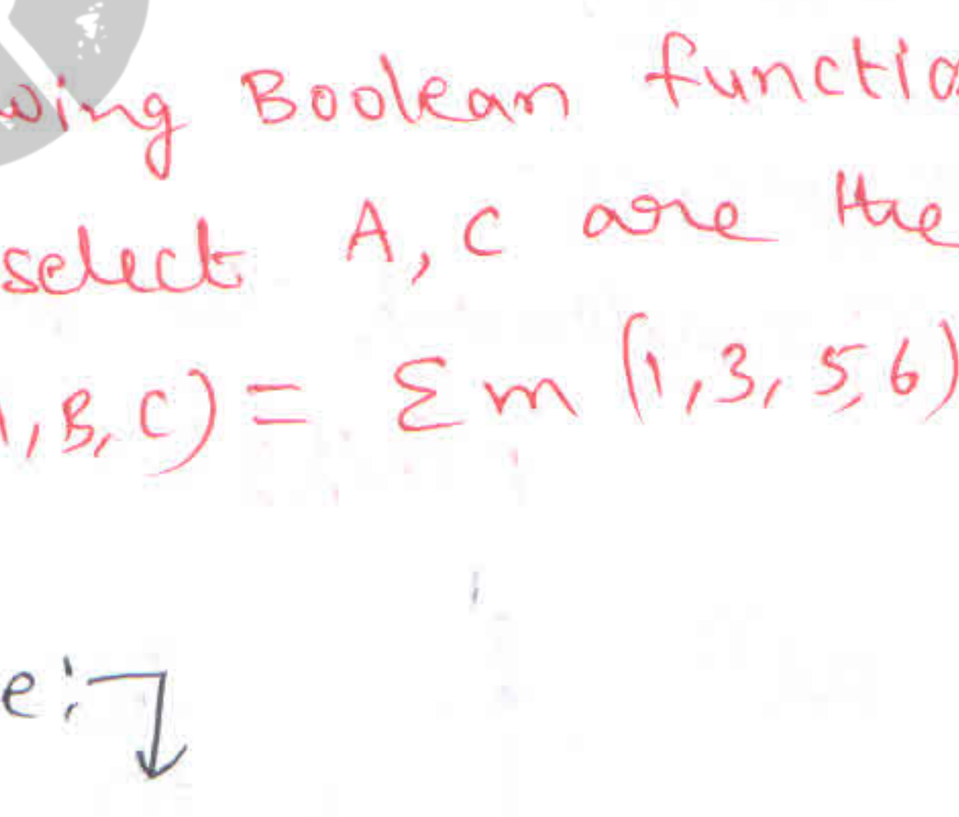
Fig:- Logic Symbol.

(3) Implement the following Boolean function using 4 x 1 MUX. select lines. $F(A,B,C) = \Sigma m\,(1,3,5,6)$ select A, C are the

Sol))

Implementation Table :-

	$\bar{C}$	C	
D_0	0	(1)	C
D_1	2	(3)	C
D_2	4	(5)	C
D_3	(6)	7	$\bar{C}$



(40)

HDL Models of Combinational Circuits :-

The module is the basic building block for modeling hardware with the Verilog HDL. The logic of a module can be described in any one of the following modeling styles:

① Gate - Level (structural) modeling using instantiations of predefined and user-defined primitive gates.

② Dataflow modeling using Continuous assignment statements with the keyword assign.

③ Behavioral modeling using procedural assignment statements with the keyword always.

→ Gate-level (structural) modeling describes a circuit by specifying its gates and how they are connected with each other.

→ Dataflow modeling is used mostly for describing the Boolean equations of Combinational logic.

→ Behavioral modeling that is used to describe Combinational and Sequential circuits at a higher level of abstraction.

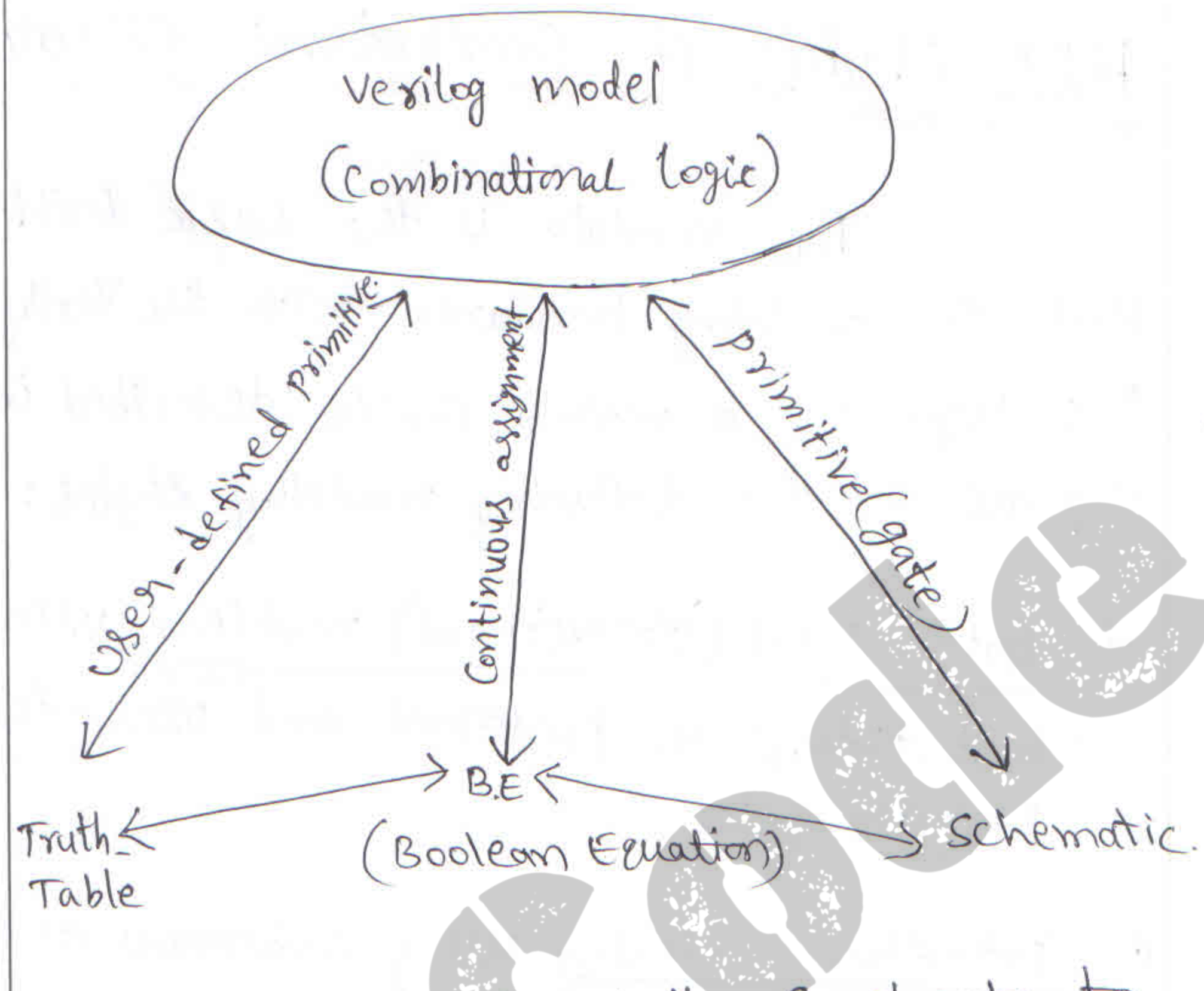


Fig. Relationship of Verilog Constructs to truth tables, Boolean equations, and schematics

① Dataflow Modeling :

Dataflow modeling of Combinational logic uses a number of operators that act on binary operands to produce a binary result.

Data flow modeling uses Continuous assignments and the keyword assign.

Some verilog HDL operators :-

Symbol	operation
&	bitwise AND
|	bitwise OR
^	bitwise X-OR
~	bitwise NOT

Behavioral Modeling :

Behavioral modeling represents digital circuits at a functional and algorithmic level. It is used mostly to describe sequential circuits, but can also be used to describe combinational circuits.

Behavioral descriptions use the keyword always, followed by an optional event control expression and a list of procedural assignment statements. The event control expression specifies when the statements will execute. The target output of a procedural assignment statement

must be of the reg data type. Contrary to the wire data type, whereby the target output of an assignment may be continuously updated, a reg data type retains its value until a new value is assigned.

Writing a Simple Test Bench :-

A test bench is an HDL program used for describing and applying a stimulus to an HDL model of a circuit in order to test it and observe its response during simulation. Test benches can be quite complex and lengthy and may take longer to develop than the design that is tested. The results of a test are only as good as the test bench that is used to test a circuit.

The initial statement executes only once, starting from simulation time '0', and may continue with any operations that are delayed by a given number of time units, as specified by the symbol #. For example, consider the initial block

```
initial
begin
    A=0; B=0;
    #10 A=1; B=0;
    #20 A=0; B=1
end
```

Sequential Logic Circuits

Introduction :-

There are many applications in which digital outputs are required to be generated in accordance with the sequence in which the input signals are received. This requirement can not be satisfied using a combinational logic-system. These applications require outputs to be generated that are not only dependent on the present input conditions but they also depend upon the past history of these inputs. The past history is provided by feedback from the output back to the input.

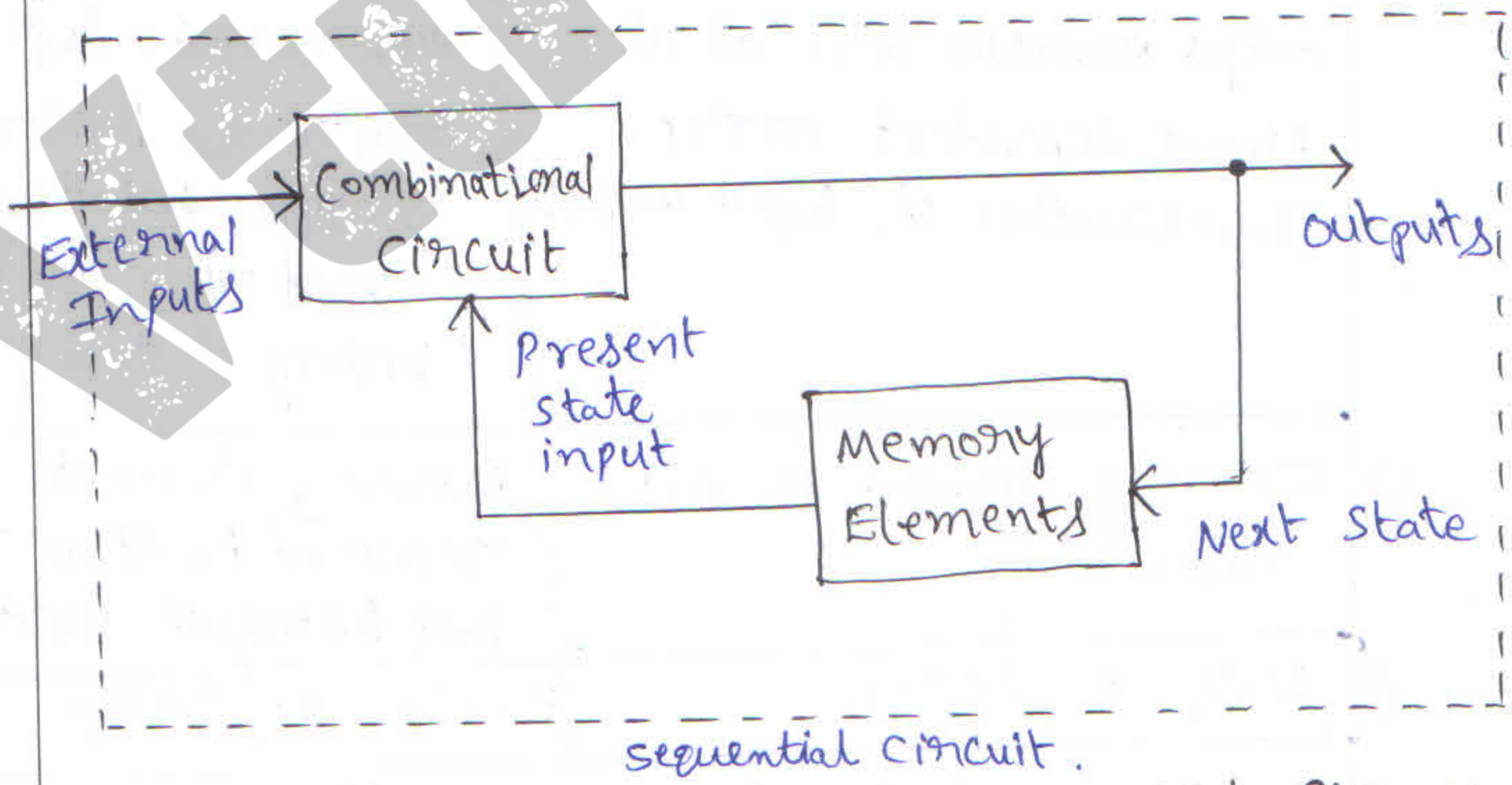


Fig. (1) The Block diagram of Sequential Circuit.

The Figure (1), shows the block diagram of Sequential Circuit. The memory elements are connected to the combinational circuit as a feedback path.

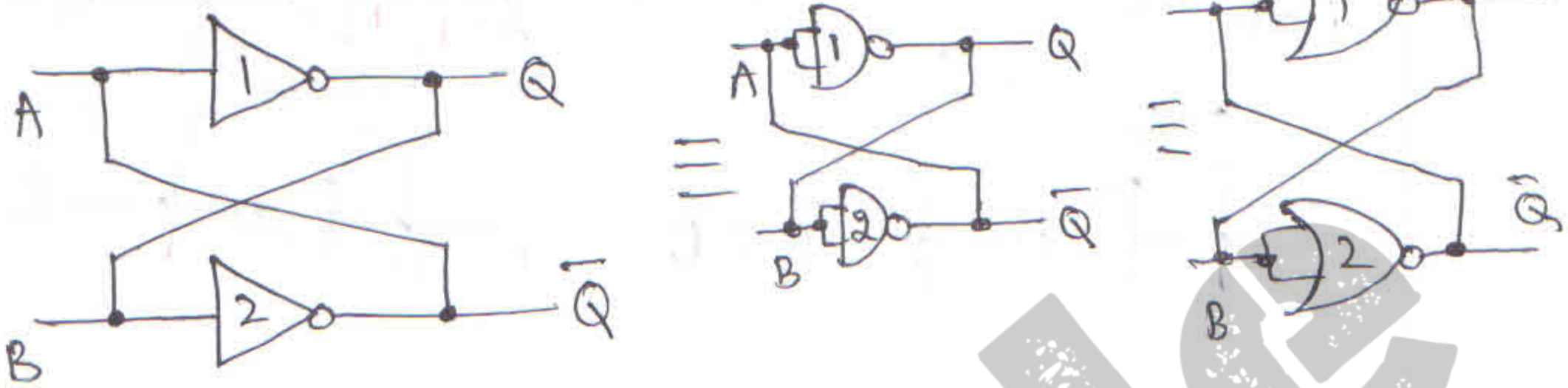
The information stored in the memory elements at any given time defines the present-state of the sequential circuit. The present state and the external inputs determine the outputs and the next state of the sequential circuit.

Comparison between Combinational and sequential Logic circuits :-

Combinational circuits	sequential circuits
(1) In Combinational circuits, the output variables are at all times dependent on the combination of input variables.	In sequential circuits, the outputs depend not only on the present i/p variables but they also depend upon the past history of these inputs
(2) Memory element is not required.	Memory elements is required to store the past history of inputs.
(3) Easy to Design	Harder to Design.
(4) Combinational circuits are faster.	sequential circuits are slower than Combinational circuits.

Latches :

Introduction :-



(a) Using Invertes (b) NAND gates (c) NOR gates

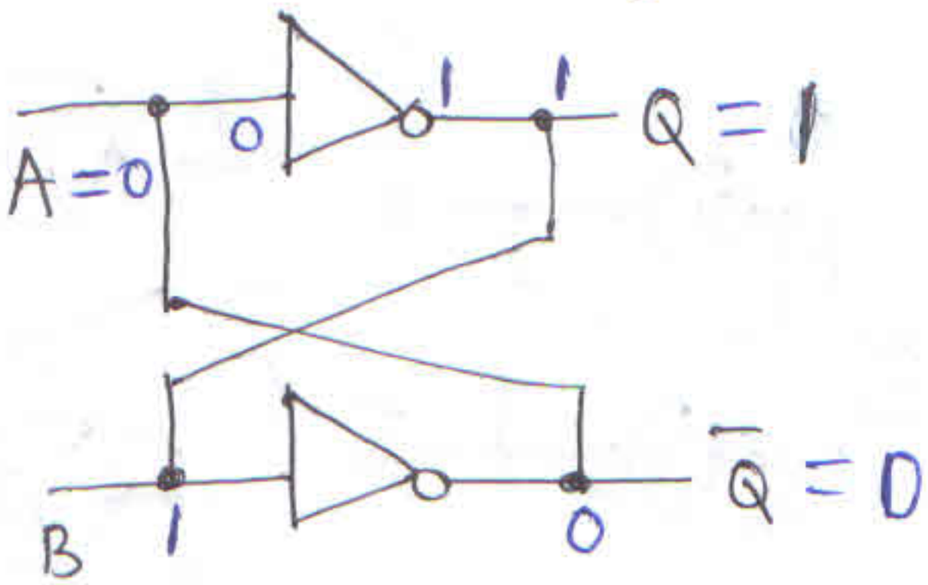
Fig. (2), The Basic Bistable element

the Fig. (2), shows the basic Bi-stable element used in latches and Flip-flops . It has two outputs Q and $\bar{Q}$. It has two cross-coupled inverters, i.e, the output of the first inverter is connected as an input to the second inverter and the output of second inverter is connected as an input to the first inverter .

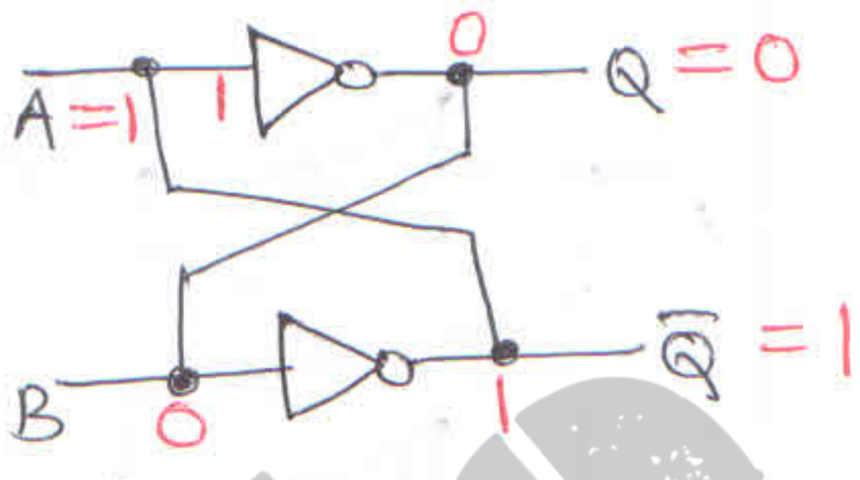
The basic bistable element circuit has two stable states logic '0' and logic '1', hence the name "Bistable" .

operation of Basic Bistable Element :-

i) when A = 0 ↘



(ii) when A = 1 ↘



(1) → The outputs Q and $\bar{Q}$ are always Complementary.

(2) → The circuit has two Stable States. when Q = 1 is referred to as Set State and Q = 0 is referred to as Reset State.

(3) → If the circuit is in the Set state, it will remain in the Set state and if the circuit is in the Reset state, it will remain in the Reset state. This property of the circuit shows that it can store 1-bit information. therefore, the circuit is called a 1-bit memory cell.

(4) → The 1-bit information stored in the circuit is locked (or) Latched in the circuit. therefore this circuit is also referred to as a Latch.

SR Latch :

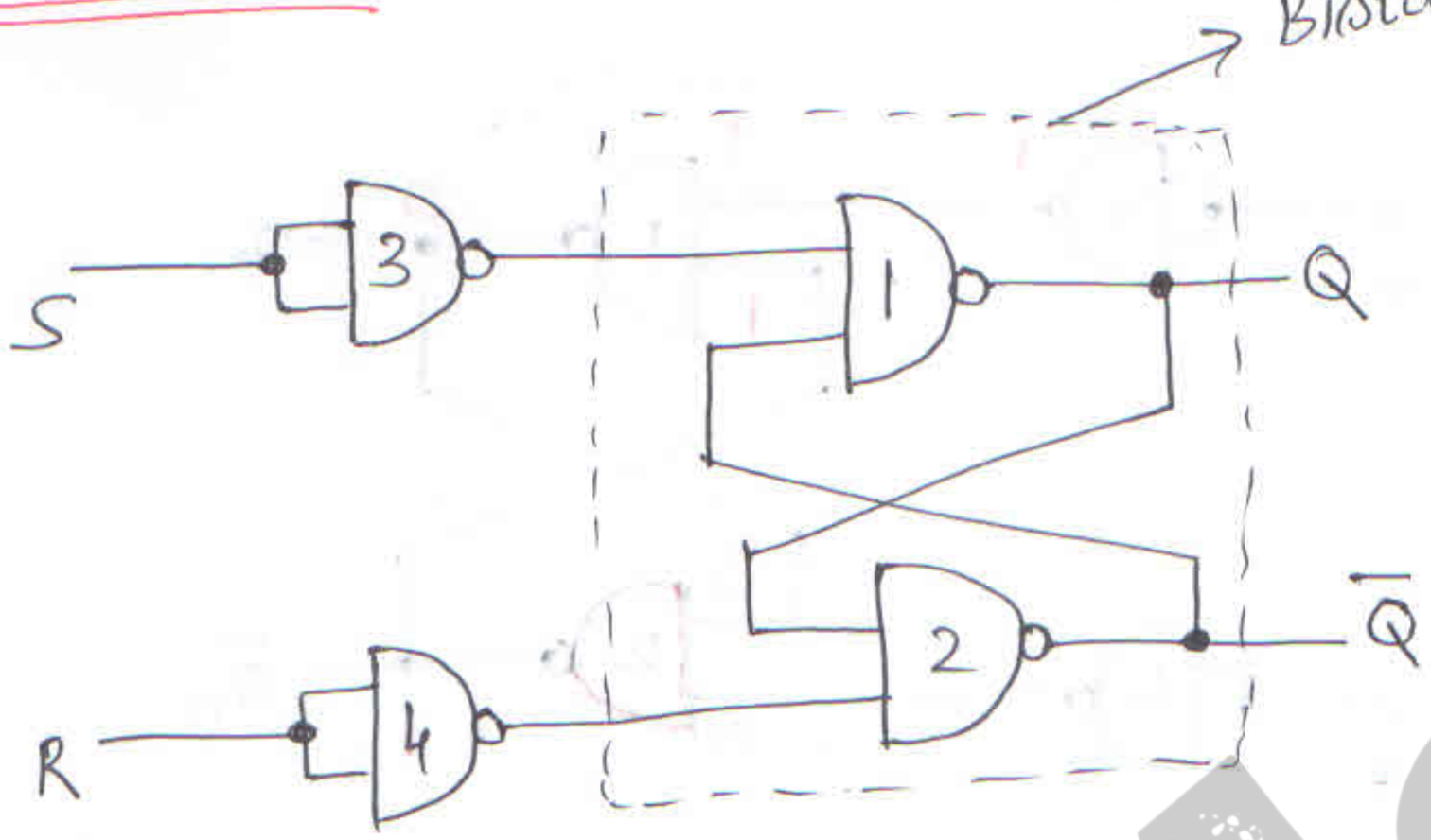


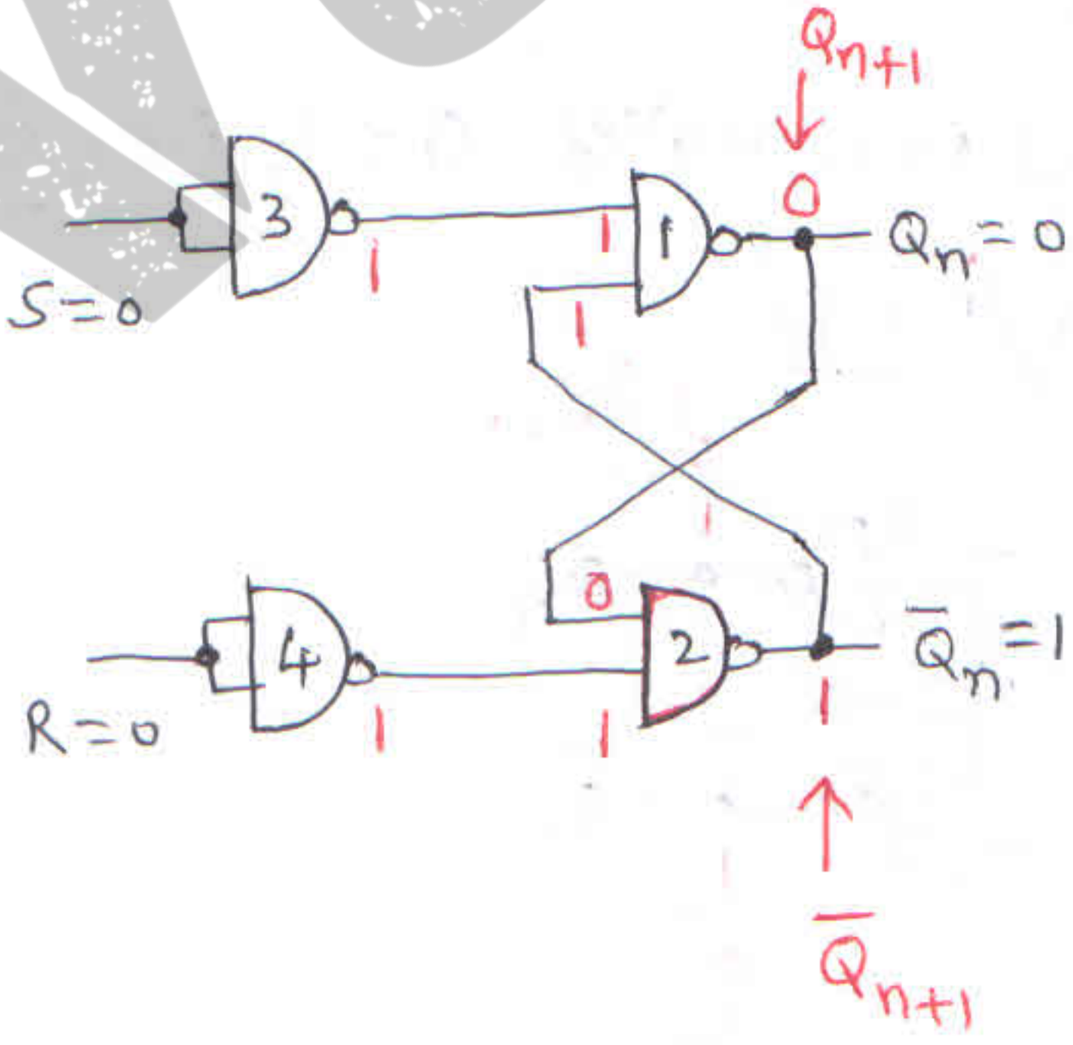
Fig. ③ SR Latch using NAND gates.

operation :-

Case ① : $S = 0$ and $R = 0$

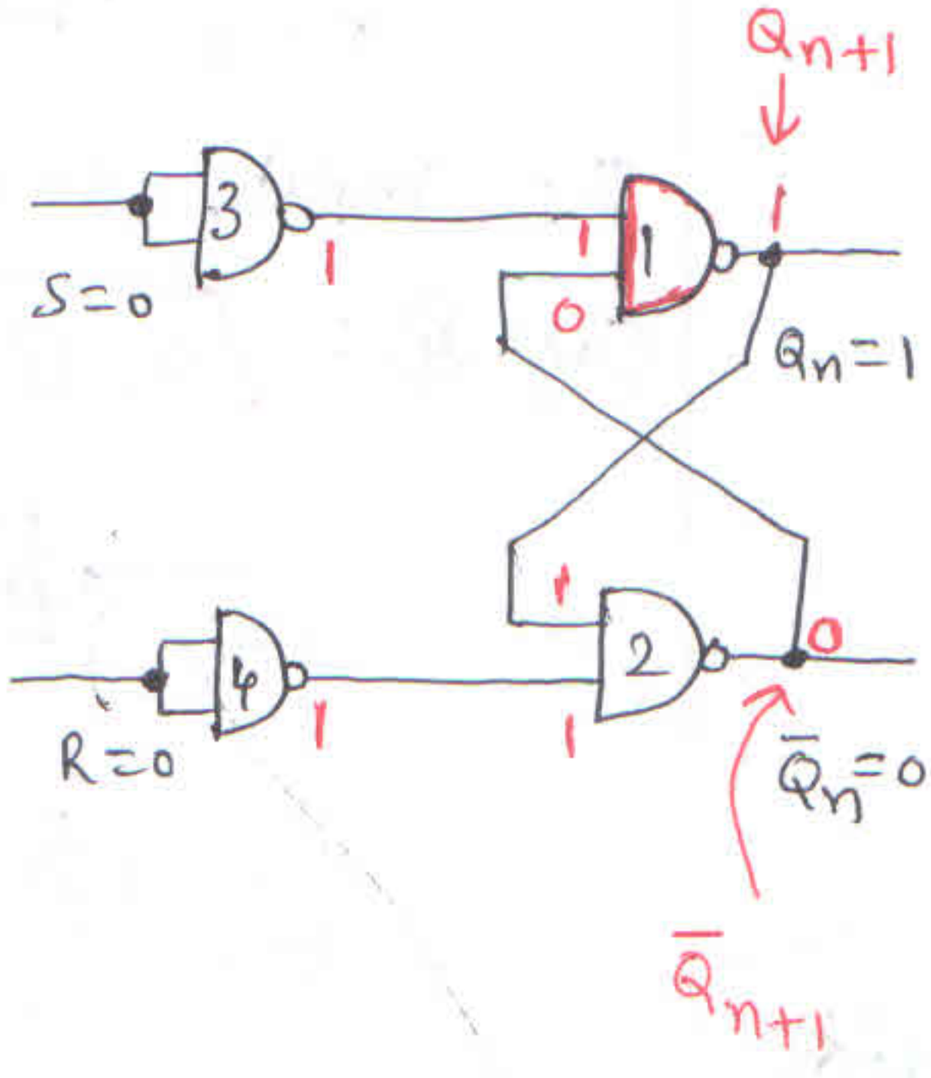
Initial state :-

$Q_n = 0$, $\bar{Q}_n = 1$



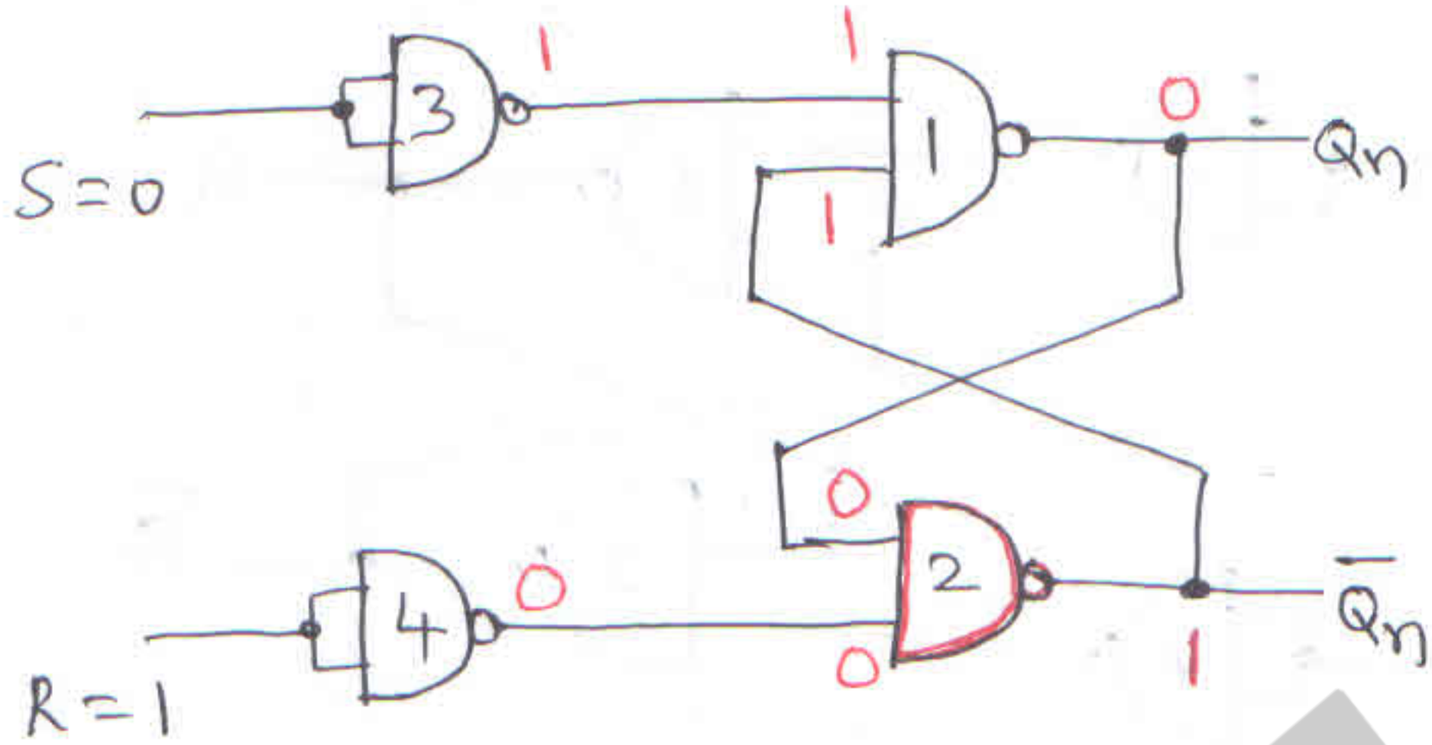
Initial state :-

$Q_n = 1$, $\bar{Q}_n = 0$



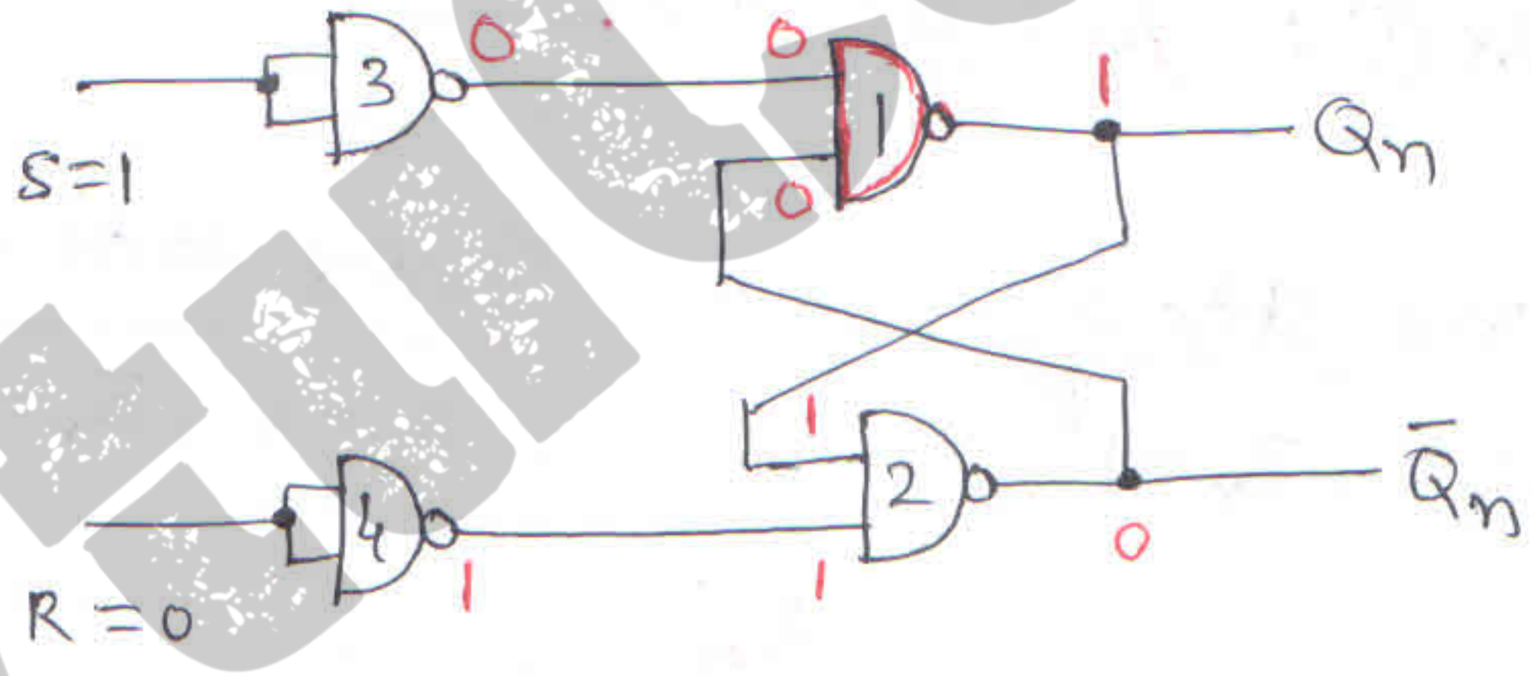
Pothi Reddy · K

Case ②: $S=0$ and $R=1$



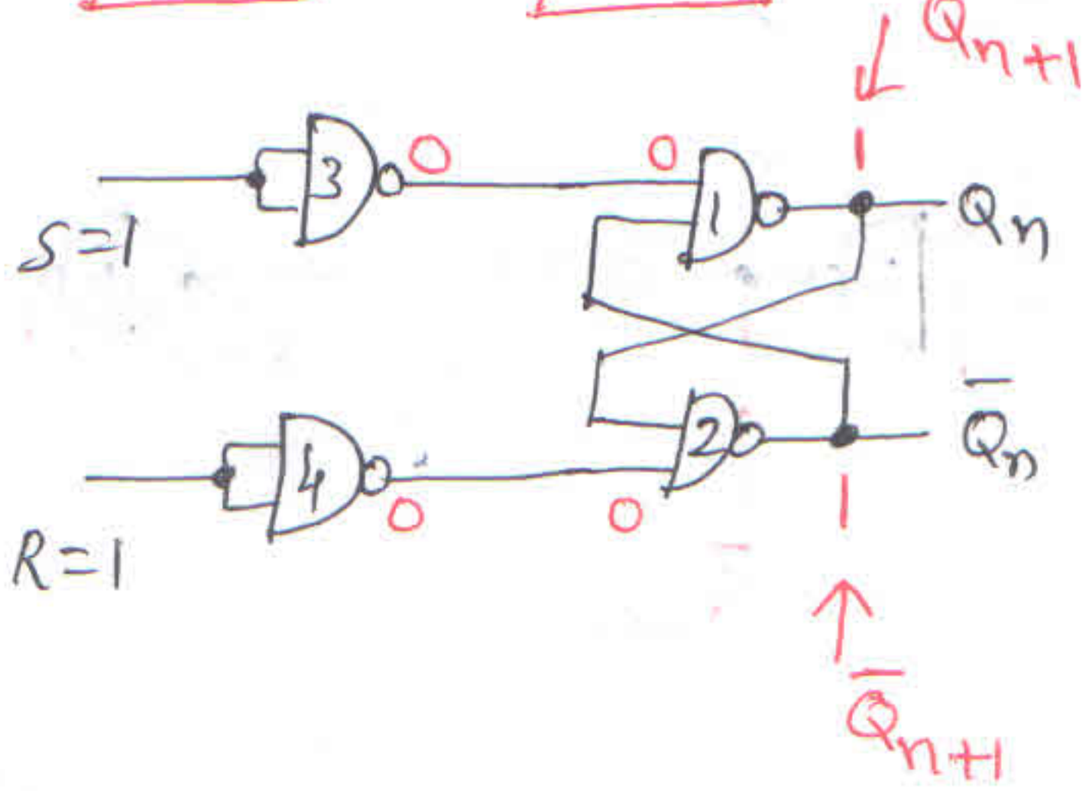
The inputs $S=0$ and $R=1$, makes $Q=0$; i.e., Reset state.

Case ③: $S=1$ and $R=0$



The inputs $S=1$ and $R=0$, makes $Q=1$, i.e., Set State.

Case ④: $S=1$ and $R=1$

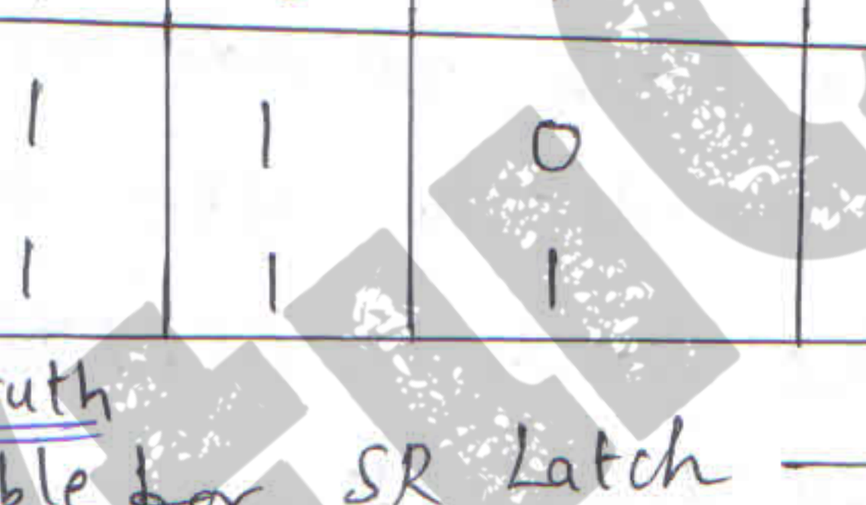


when S=1 and R=1, both outputs Q_{n+1} and $\overline{Q}_{n+1}$ to become "1". which is not allowed and i.e, this condition is Invalid.

Inputs		Present State o/p	Next state o/p	
S	R	Q_n	Q_{n+1}	State
0	0	0	0	No change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	X	Invalid
1	1	1	X	

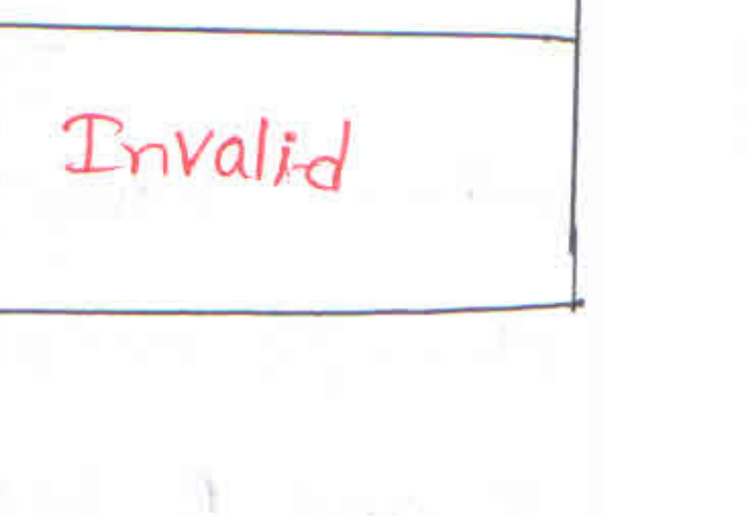
Truth Table for SR Latch

Characteristic Equation:



$$Q_{n+1} = S + \overline{R} Q_n$$

Logic Symbol:



SR Latch using NOR gates :

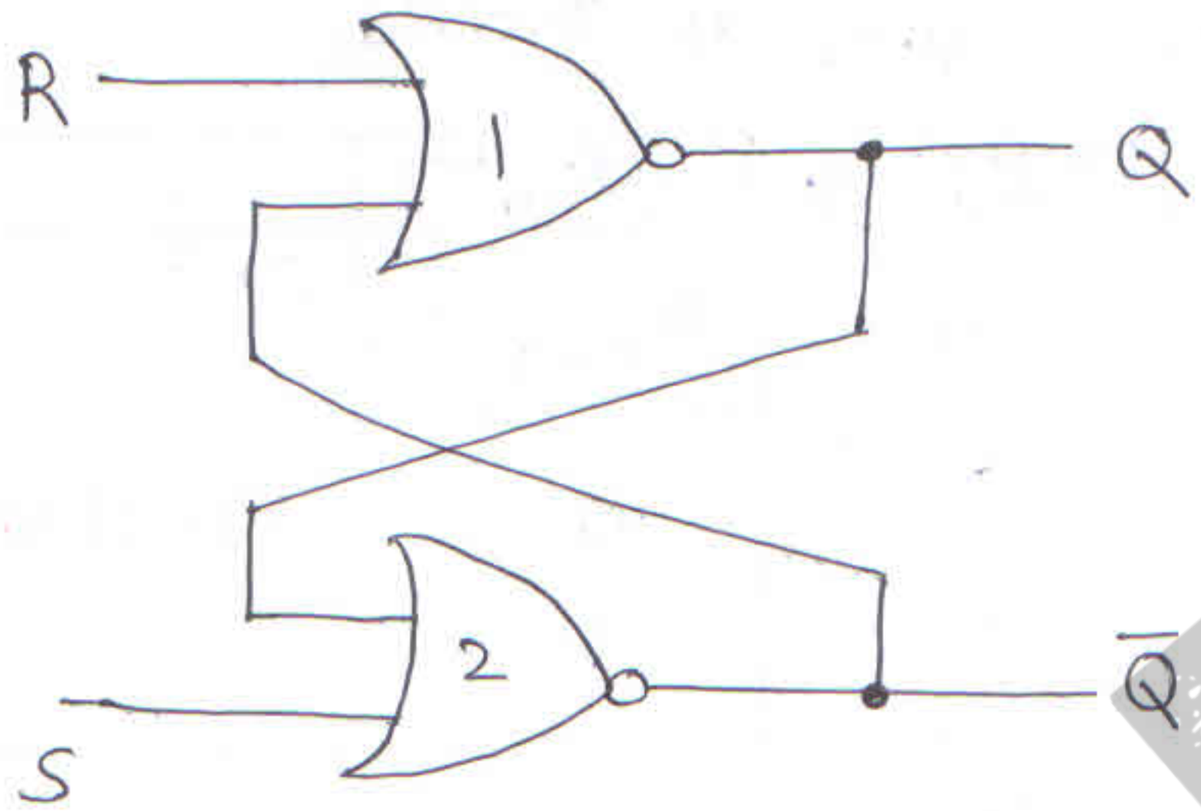


Fig: (4) Logic Diagram

Gated SR Latch :

In the SR Latch, we have seen that output changes occur immediately after the input changes occur. i.e. the latch is sensitive to its S and R inputs at all times. However, it can easily be modified to create a latch that is sensitive to these inputs only when an Enable(EN) is Active. Such a latch with enable input is known as Gated SR Latch.

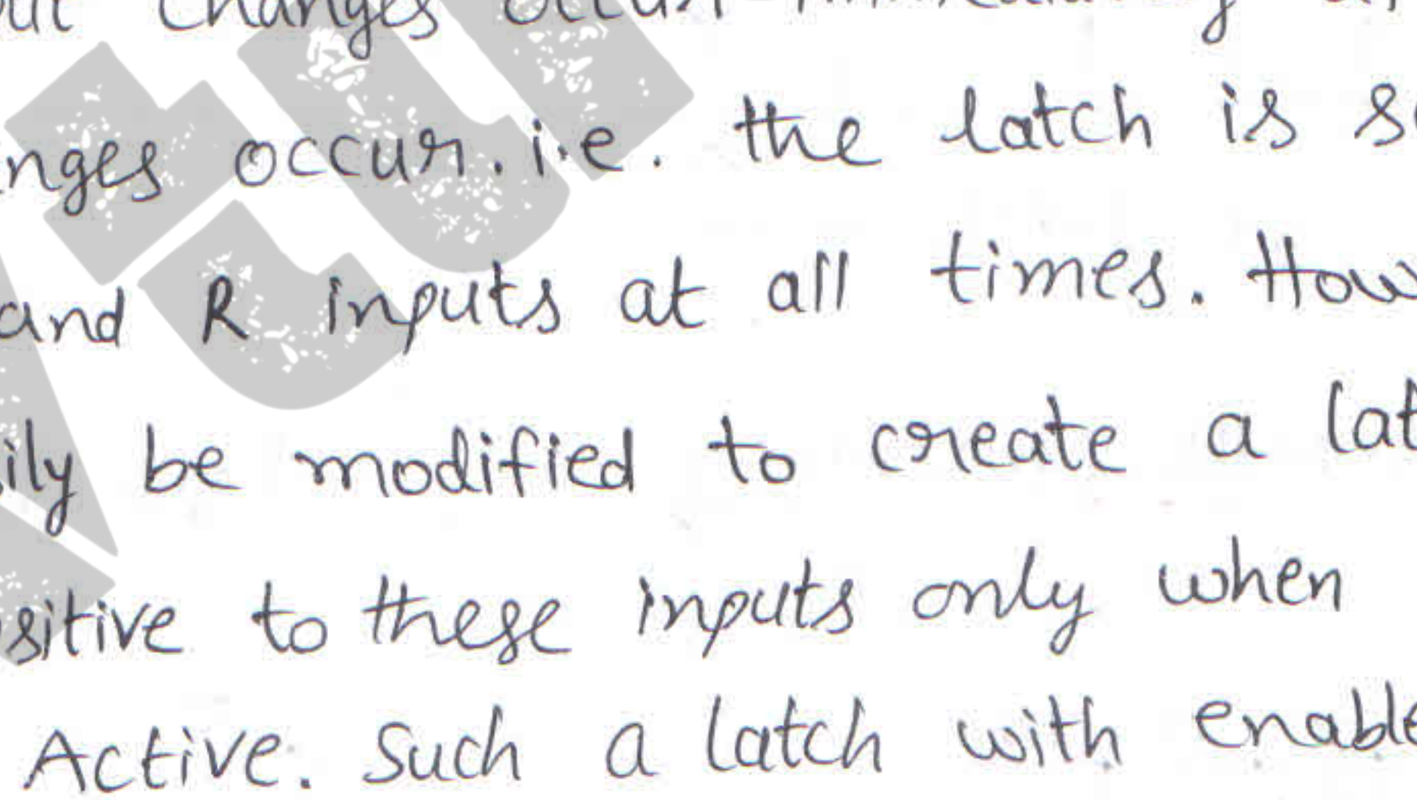


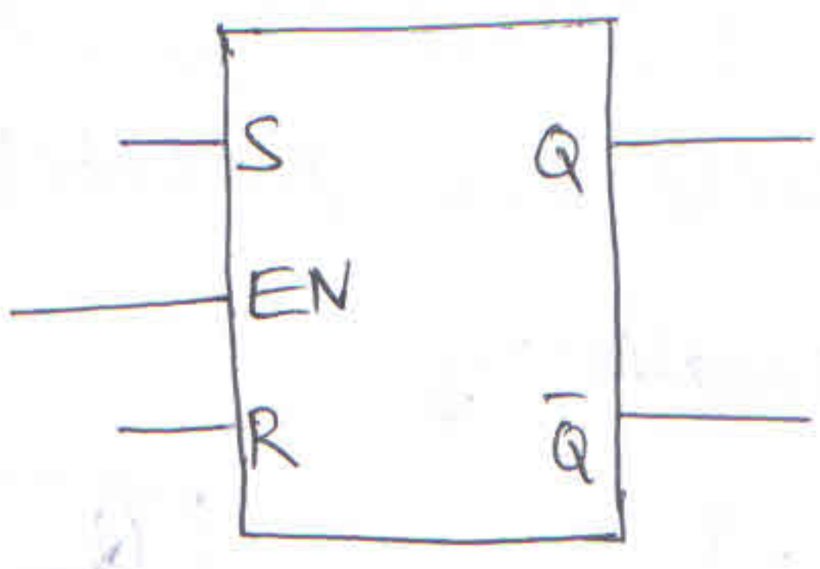
Fig (5) Gated SR Latch

EN	S	R	Q_n	Q_{n+1}	State
1	0	0	0	0	No change
1	0	0	1	1	
1	0	1	0	0	Reset
1	0	1	1	0	
1	1	0	0	1	Set
1	1	0	1	1	
1	1	1	0	x	Invalid
1	1	1	1	x	
0	x	x	0	0	No change
0	x	x	1	1	

Truth Table for SR Latch with Enable input

As shown by Truth Table, the circuit behaves like a SR Latch when EN = 1, and retains its previous state when EN = 0.

Logic Symbol :



D - Latch (Transparent Latch) :

One way to eliminate the undesirable condition of the indeterminate state in the SR latch is to ensure that inputs S and R are never equal to "1" at the same time. This is done in the D latch.

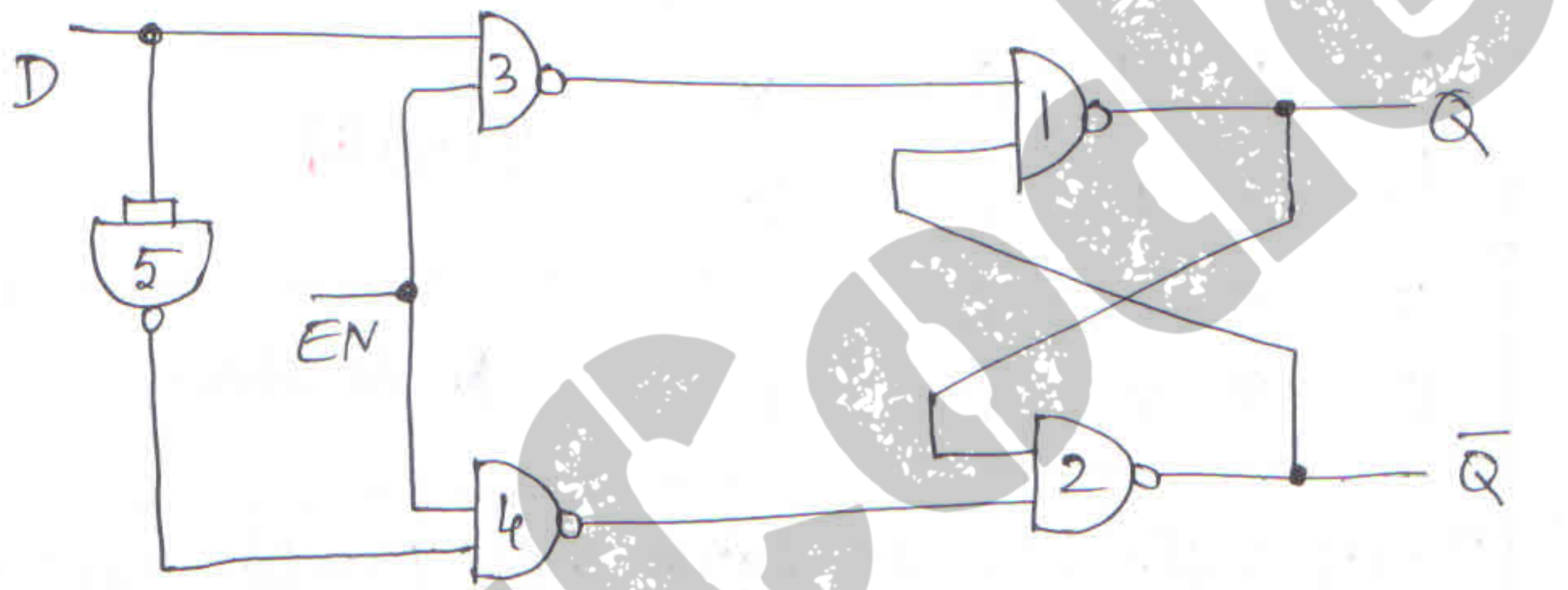
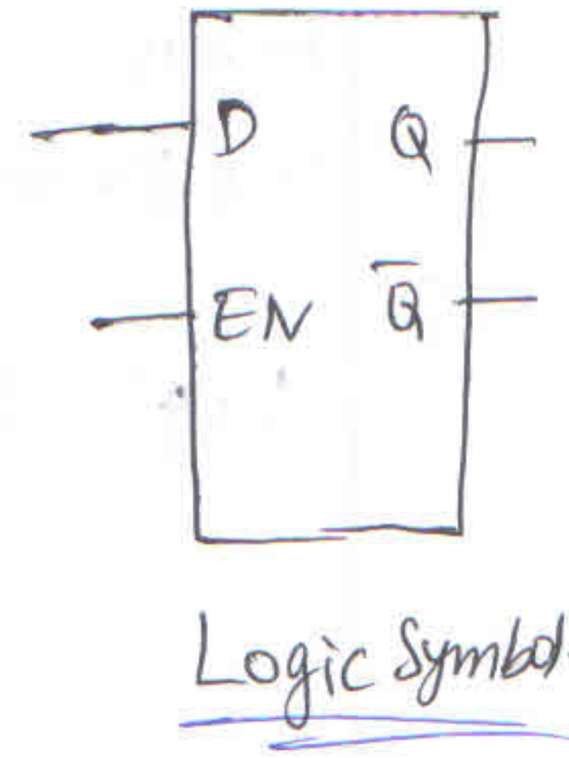


Fig.⑥ Logic Diagram of a D-Latch.

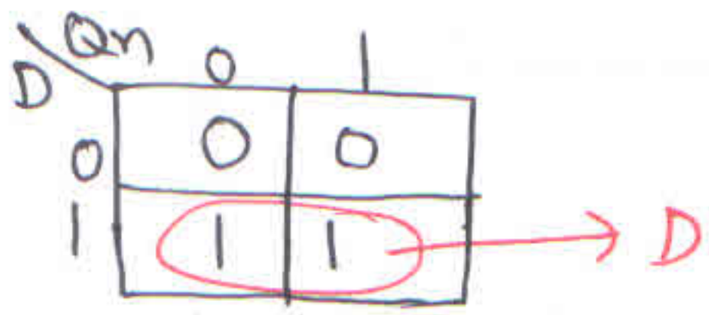
EN	D	Q_n	Q_{n+1}	State
1	0	X	0	Reset
1	1	X	1	Set
0	X	X	Q_n	No change



Logic Symbol

Fig.⑦ Truth Table for D-Latch.

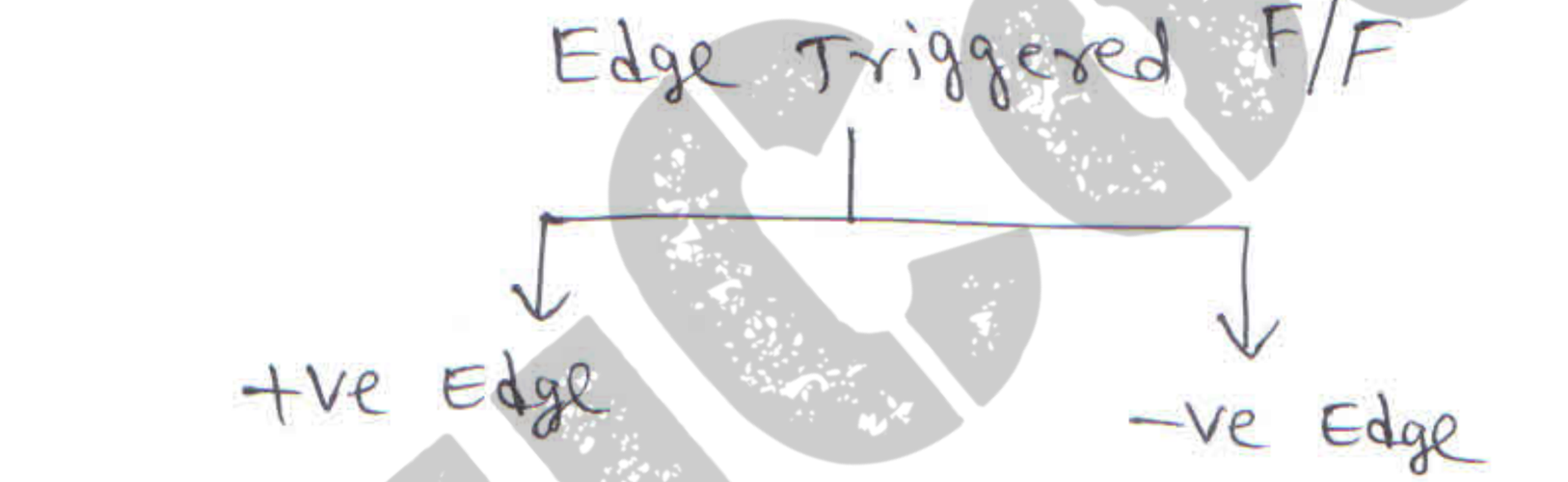
Characteristic Equation: ↴



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

FLIP-FLOPS : (F/F)

A simple latch forms the basis for the flip-flop. Latches are Controlled by enable signal, and they are level triggered, either +ve or -ve level triggered. the output state is free to change according to the S and R input values, when active level is maintained at the enable input. Flip-flops are different from latches. Flip-flops are pulse or clock edge triggered instead of Level triggered.



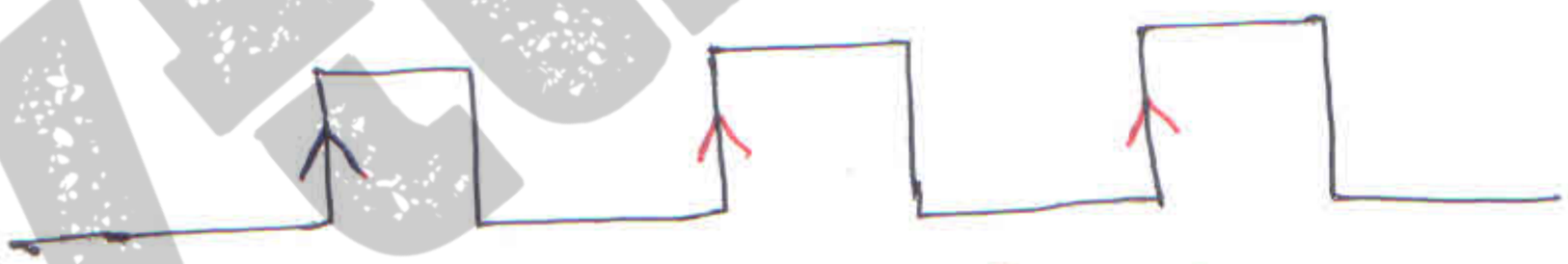


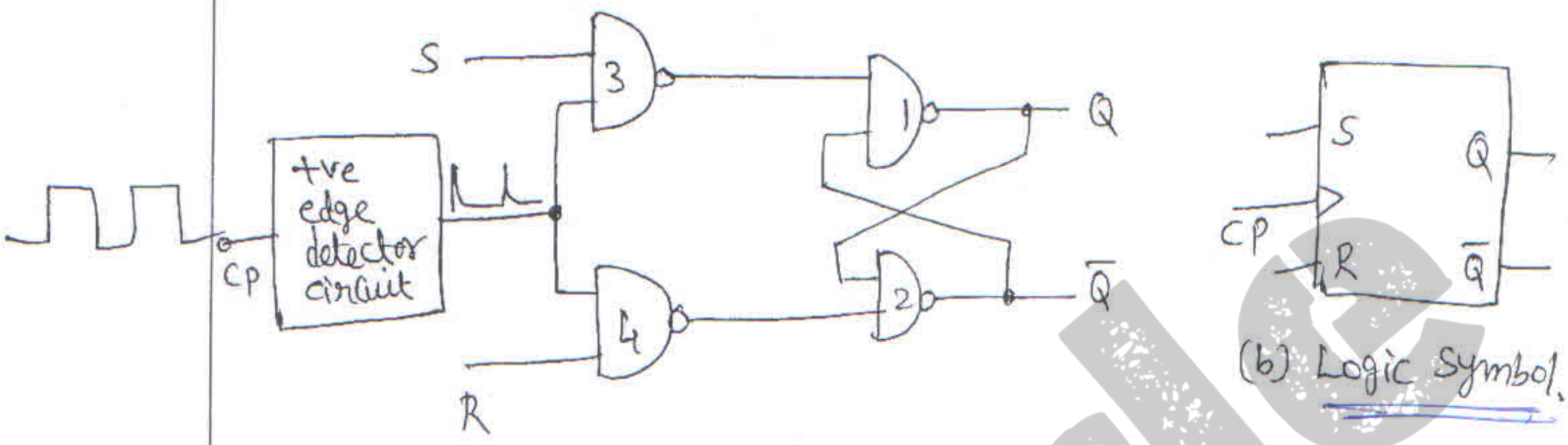
Fig. (a) Positive Edge Response

Fig. (b) Negative Edge Response

Fig. (8) Clock response in Flip-Flops.

Clocked SR - Flip-Flop :

(i) Positive Edge Triggered SR Flip-Flop :-



(b) Logic Symbol.

Fig ① (a) SR Flip-Flop using NAND gates.

The figure 1(a) shows the +ve edge triggered-SR flip-flop. The circuit is similar to the SR latch (Gated) except enable signal is replaced by the clock pulse (CP) followed by the positive edge detector circuit. The edge detector circuit is a differentiator.

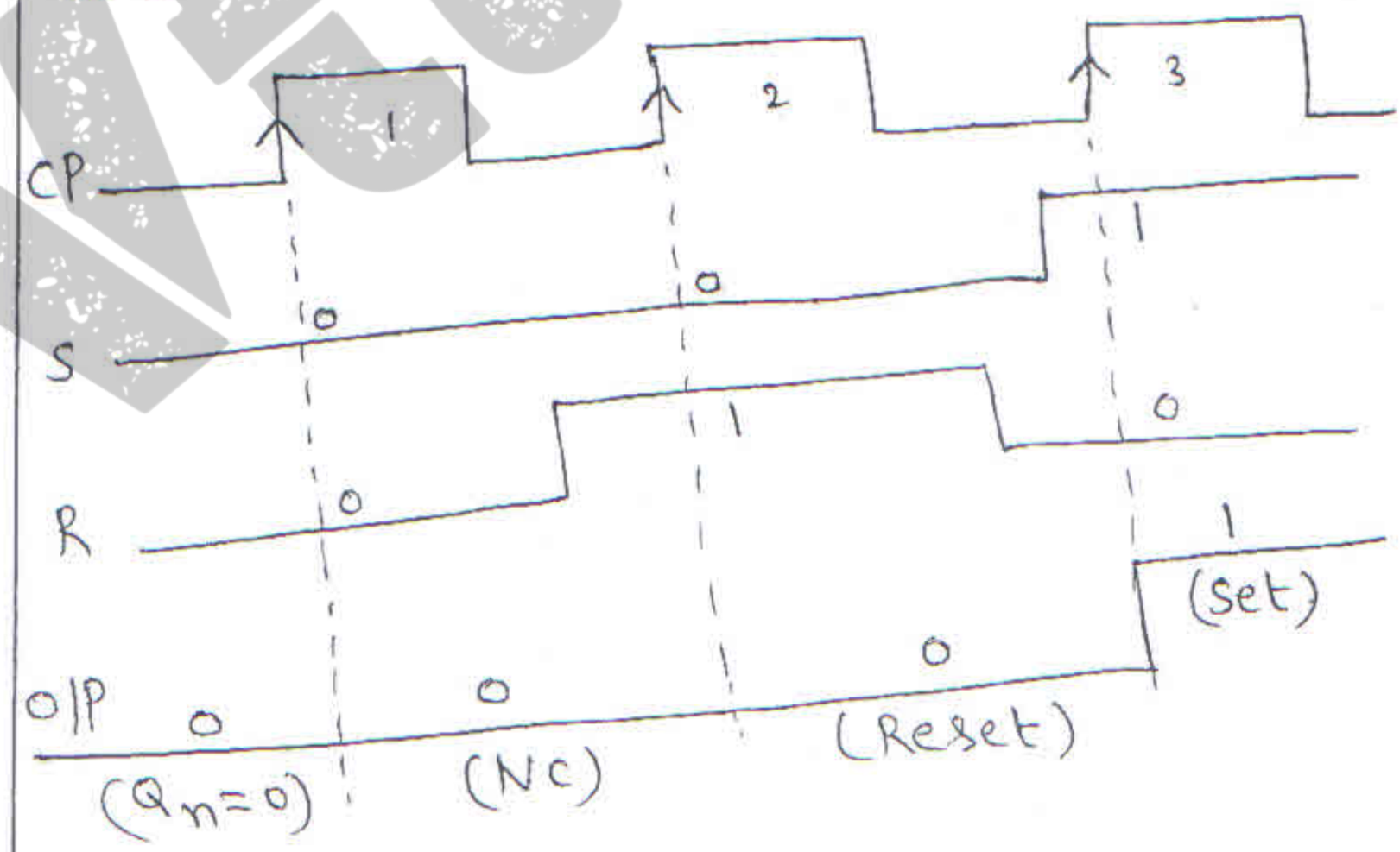


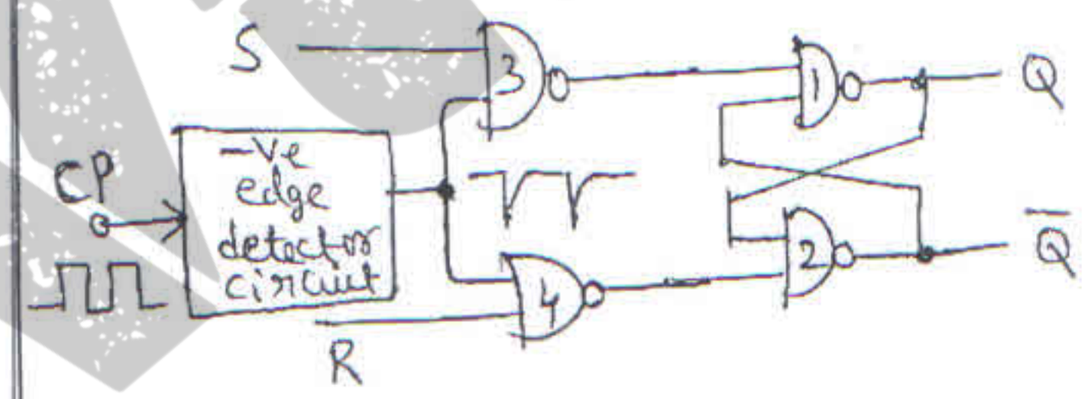
Fig ①(c) :- Input and output waveforms (or) Timing diagrams.

CP	S	R	Q_n	Q_{n+1}	State
↑	0	0	0	0	No change
↑	0	0	1	1	
↑	0	1	0	0	Reset
↑	0	1	1	0	
↑	1	0	0	1	Set
↑	1	0	1	1	
↑	1	1	0	X	Invalid
↑	1	1	1	X	
0	X	X	0	0	No change
0	X	X	1	1	

Table①: Truth Table for positive edge clocked SR Flip-Flop

(ii) Negative Edge Triggered SR Flip-Flop:

In the -ve edge triggered SR flip-flop, the negative edge detector circuit is used and the circuit output responds at the negative edges of the clock pulse.



Fig(2) a: Negative edge triggered SR Flip-flop using NAND gates.

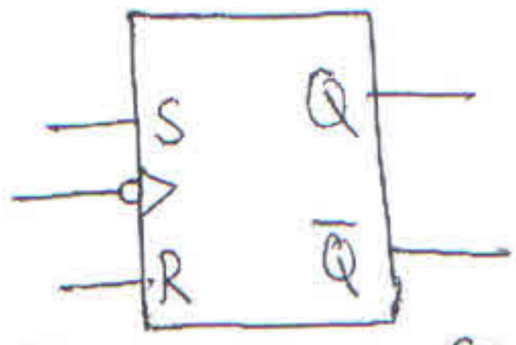


fig 2(b): Logic Symbol

the fig 2(a) shows the logic diagram and fig 2(b) shows the logic symbol. The bubble at the clock input indicates that the flip-flop is negative edge triggered.

CP	S	R	Q_n	Q_{n+1}	state
↓	0	0	0	0	No change (N.C)
↓	0	0	1	1	
↓	0	1	0	0	Reset
↓	0	1	1	0	
↓	1	0	0	1	Set
↓	1	0	1	1	
↓	1	1	0	X	Invalid
↓	1	1	1	X	
0	X	X	0	0	No change
0	X	X	1	1	

Table ②: Truth Table for -ve edge clocked SR Flip-Flop.

Fig 2(c): Input and output waveforms for -ve edge triggered clocked SR Flip-Flop.

Clocked D - Flip-Flop:

In D - flip-flop the basic SR flip-flop is used with complemented inputs. The D flip-flop is similar to D-Latch except clock pulse followed by edge detector is used instead of enable input. such an edge triggered D flip-flop can be of two types: (1) positive edge D-F/F (2) Negative edge D-F/F.

(1) Positive edge triggered D flip-flop:

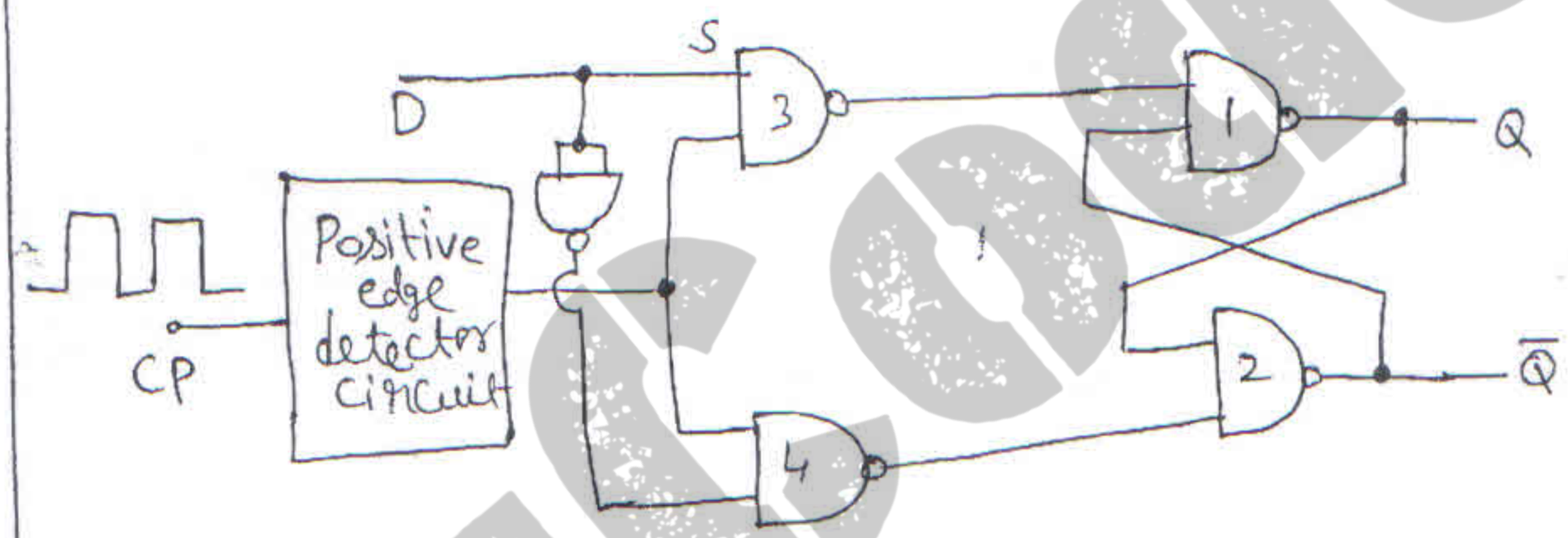


Fig 1(a):- Positive edge triggered D-FlipFlop.

The figure 1(a), shows the +ve edge D-F/F. It Consists of a gated D-Latch and a positive edge-detector circuit.

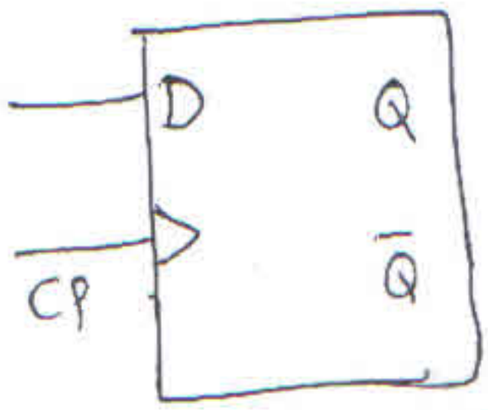


Fig 2(a): Logic Symbol

Fig 2(b):- Truth Table of

CP	D	Qn	Qn+1	state
↑	0	0	0	Reset
↑	0	1	0	
↑	1	0	1	Set
↑	1	1	1	
0	X	0	0	No change
0	X	1	1	

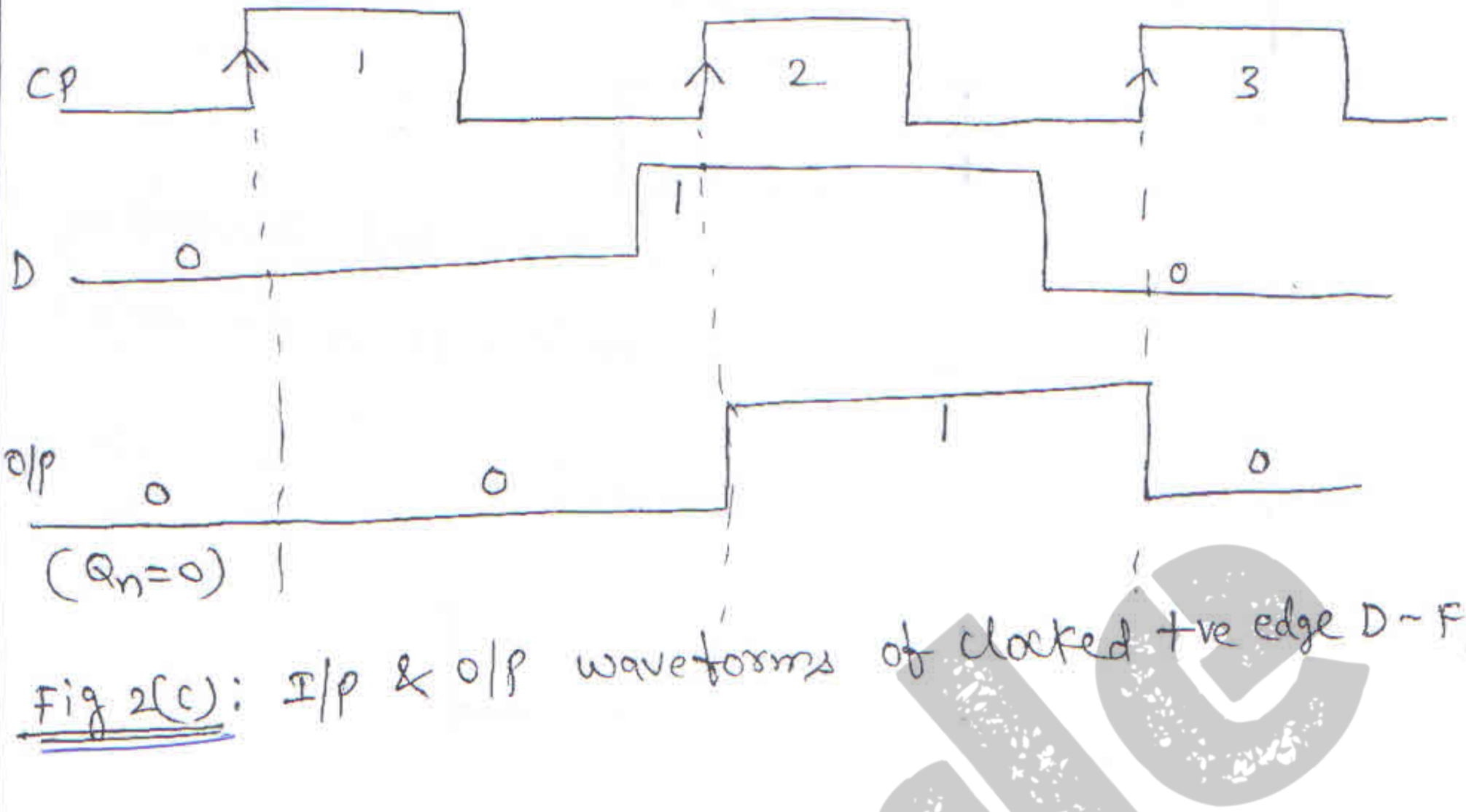


Fig 2(c): I/p & o/p waveforms of clocked +ve edge D-F/F

Clocked JK-Flip-Flop:

The uncertainty in the state of an SR flip-flop, when S=R=1 can be eliminated by converting it into a JK flipflop. The Data inputs are J and K which are ANDed with $\bar{Q}$ and Q respectively, to obtain S and R inputs, as shown in the Fig (1)a. Thus $S = J\bar{Q}$ and $R = KQ$.

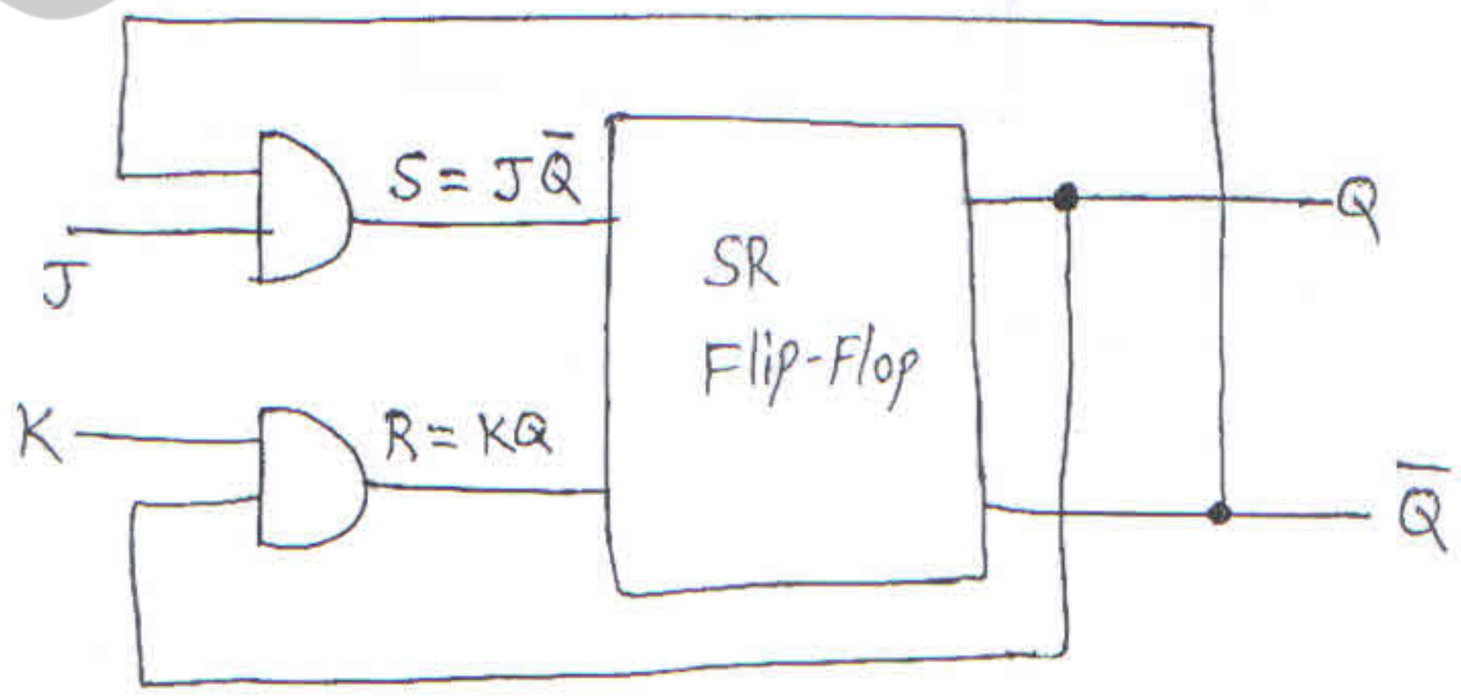


Fig (1)(a): JK flipflop using SR flipflop.

Let us see the operation of JK flip-flop.

Case ①: $J = K = 0$

when $J = K = 0$, $S = R = 0$ and according to truth table of SR flip-flop there is no change in the output.

when inputs $J = K = 0$, output does not change.

Case ②: $J = 1$ and $K = 0$

$Q = 0, \bar{Q} = 1$: when $J = 1, K = 0$ and $Q = 0$, $S = 1$ and $R = 0$. According to truth table of SR flipflop it is Set State and the output Q will be 1.

$Q = 1, \bar{Q} = 0$: when $J = 1, K = 0$ and $Q = 1$, $S = 0$ and $R = 0$ since $SR = 00$, there is no change in the output and therefore, $Q = 1$ and $\bar{Q} = 0$.

The inputs $J = 1$ and $K = 0$, makes $Q = 1$, i.e. set state.

Case ③: $J = 0$ and $K = 1$

$Q = 0, \bar{Q} = 1$: when $J = 0, K = 1$ and $Q = 0$, $S = 0$ and $R = 0$. since $SR = 00$, there is no change in the output

$\therefore Q = 0$ and $\bar{Q} = 1$.

$Q = 1, \bar{Q} = 0$: when $J = 0, K = 1$ and $Q = 1$, $S = 0$ and $R = 1$. According to truth table of SR-F/F it is a Reset State and the output Q will be 0.

The input $J = 0$ and $K = 1$ makes $Q = 0$, i.e. Reset state.

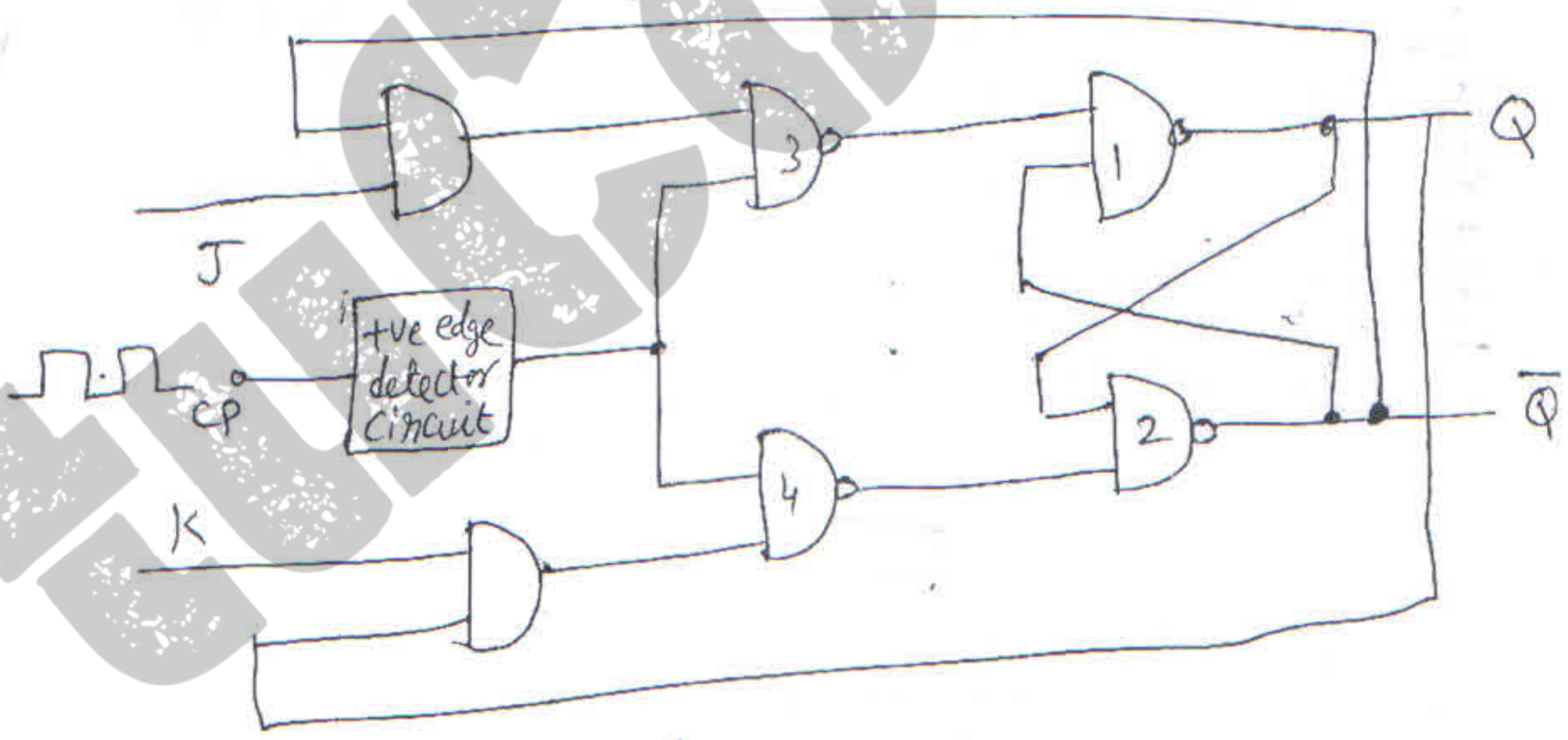
Case ④: J = K = 1

Q = 0, $\bar{Q}$ = 1 : when J = K = 1 and Q = 0, S = 1 and R = 0.
According to truth table of SR-F/F it is Set State and the output Q will be 1.

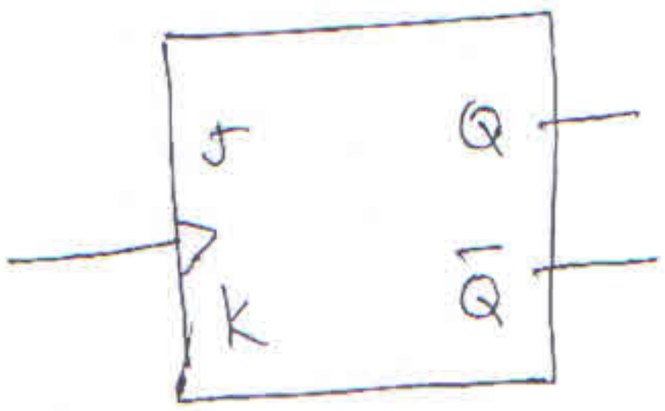
Q = 1, $\bar{Q}$ = 0 : when J = K = 1 and Q = 1, S = 0 and R = 1.
According to truth table of SR-F/F it is a Reset State and the output Q will be 0.

the inputs J = K = 1, toggles the flip flop output.

The fig 2(a),b,c. shows the logic symbol, truth tab



Logic diagram

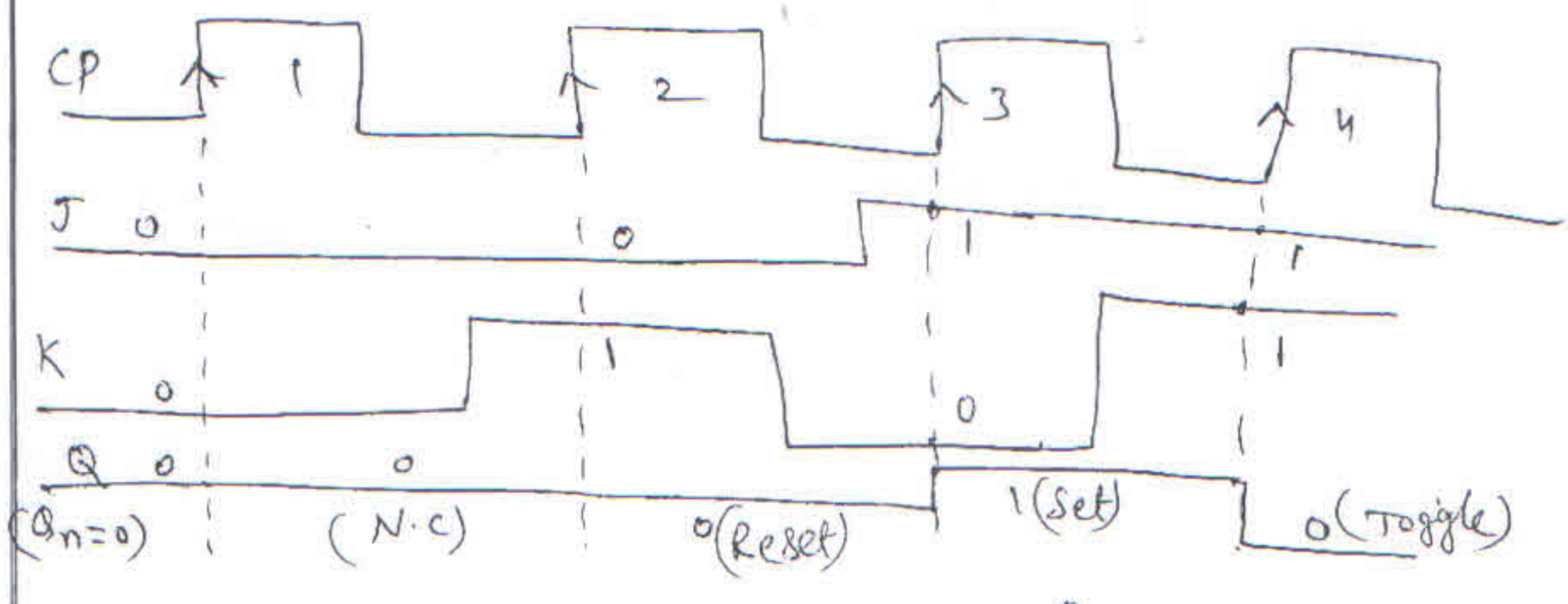


Logic Symbol

J	K	Qn	Qn+1	State
0	0	0	0	No change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	1	Toggle
1	1	1	0	

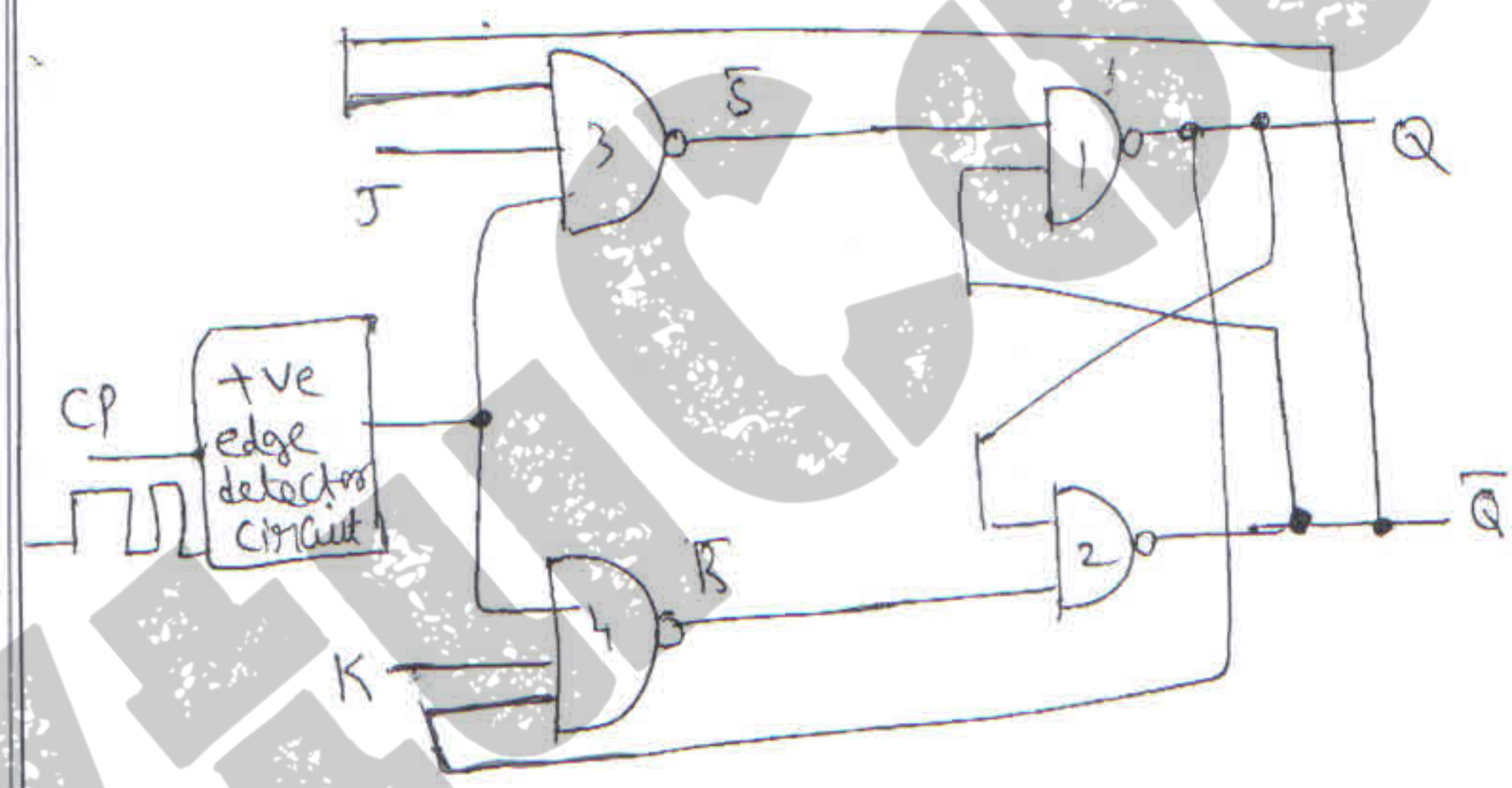
Truth Table

Pothi Reddy. K



Timing diagram for JK F/F

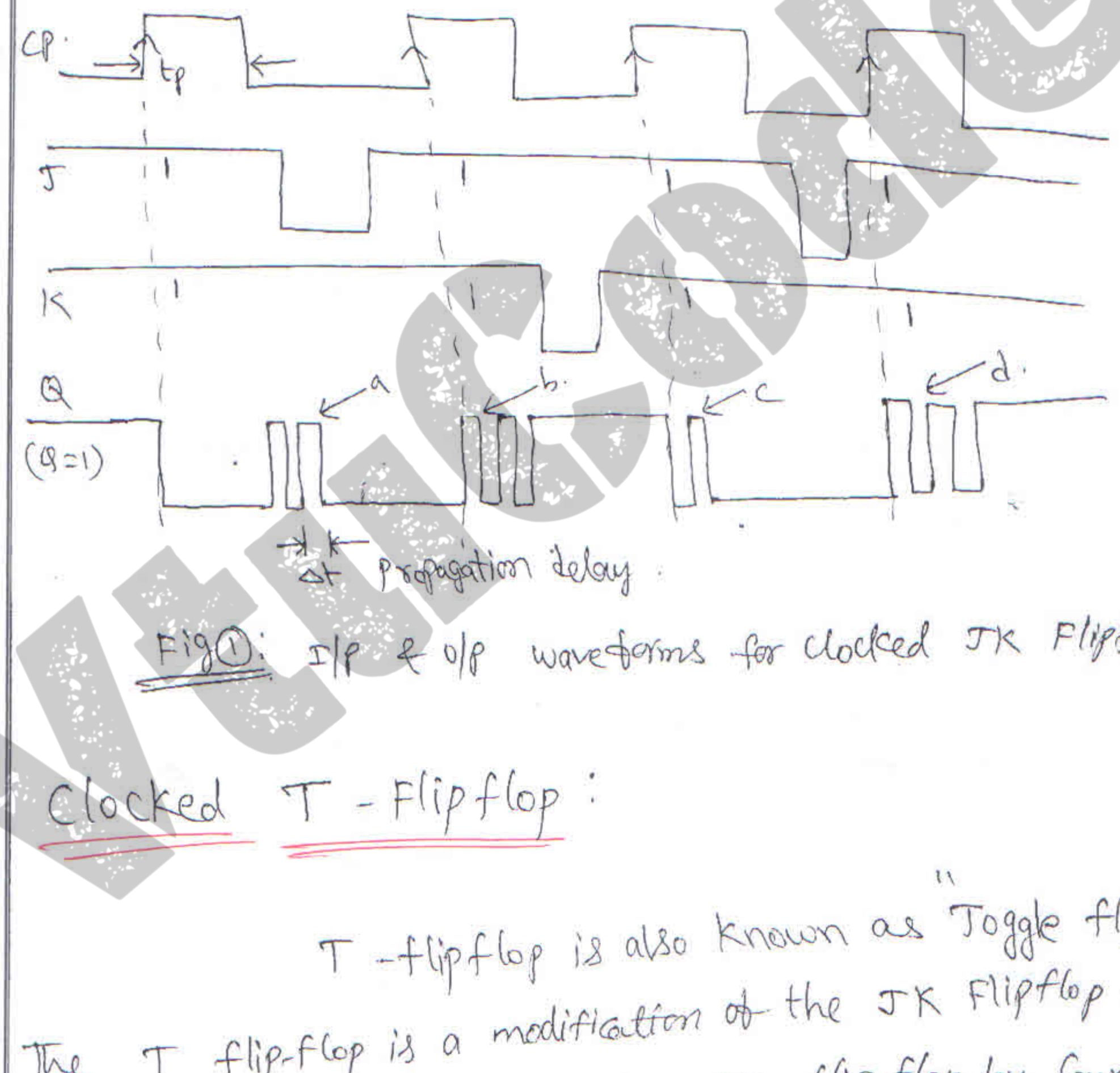
JK - Flip flop using NAND gates :



Race - around Condition :

In JK Flip-flop, when J=K=1, the output toggles (o/p changes either from 0 to 1 or from 1 to 0). Consider that initially Q=0 and J=K=1. After a time interval Δt equal to the propagation delay through two NAND gates in series, the o/p will change to Q=1 and after another time interval Δt the output will change back to Q=0. This toggling will continue until the flip-flop is enabled and J=K=1. At the end

of clock pulse the flipflop is disabled and the value of Q is uncertain. This situation is referred to as the race-around Condition. This is illustrated in Fig ①. This Condition exits when $t_p \geq \Delta t$. Thus by keeping $t_p < \Delta t$, we can avoid race around condition. A more practical method for overcoming this difficulty is the use of the master-slave (MS) Configuration.



Fig①: I/p & o/p waveforms for clocked JK Flipflop.

Clocked T-Flipflop:

T-flipflop is also known as Toggle flipflop. The T flip-flop is a modification of the JK Flipflop. The T-flipflop is obtained from a JK flip-flop by Connecting both inputs, J and K together.

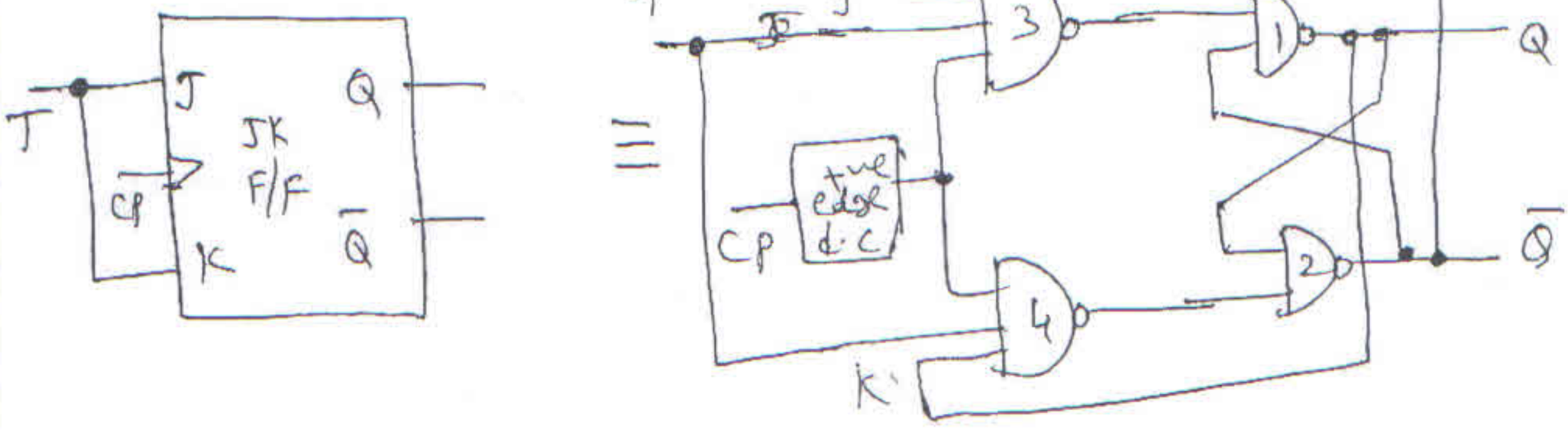
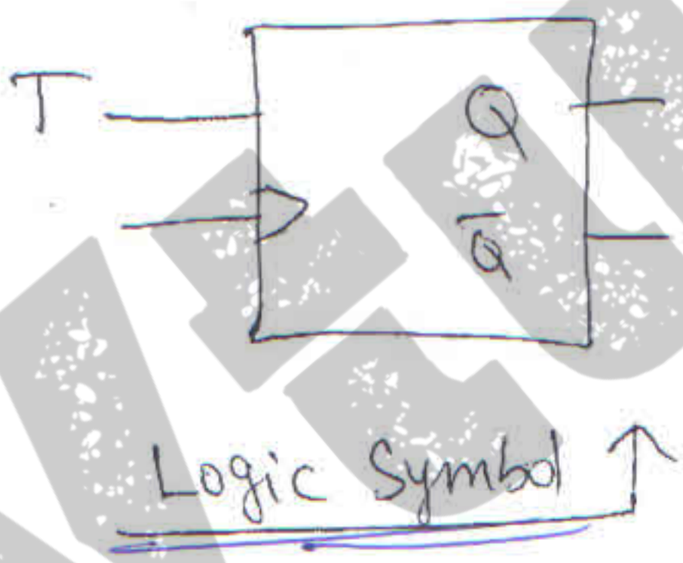


Fig. T-flip flop using NAND gates.

operation:

when T = 0, J = K = 0 and hence there is no change in the output.

when T = 1, J = K = 1 and hence output toggles.



Logic Symbol

T	Q_n	Q_{n+1}	State
0	0	0	No change
0	1	1	
1	0	1	Toggle
1	1	0	

Truth Table

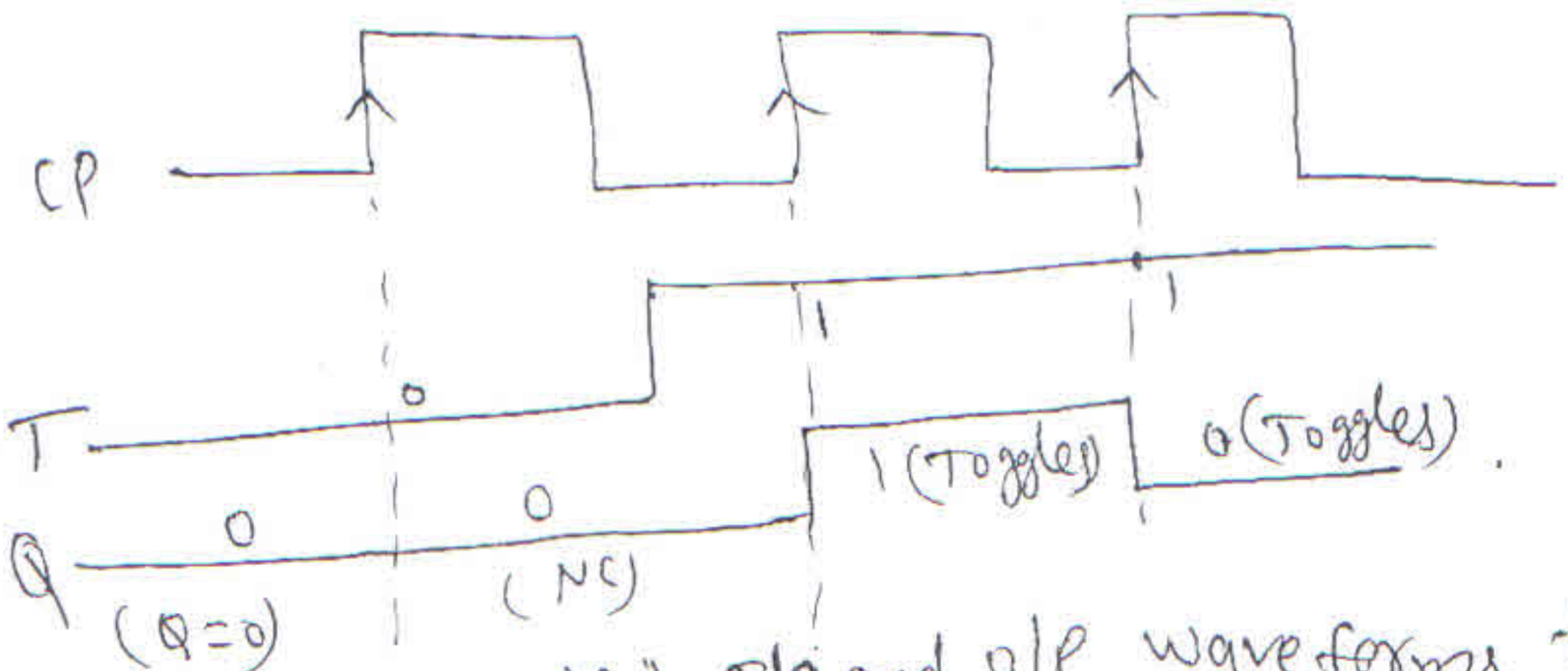
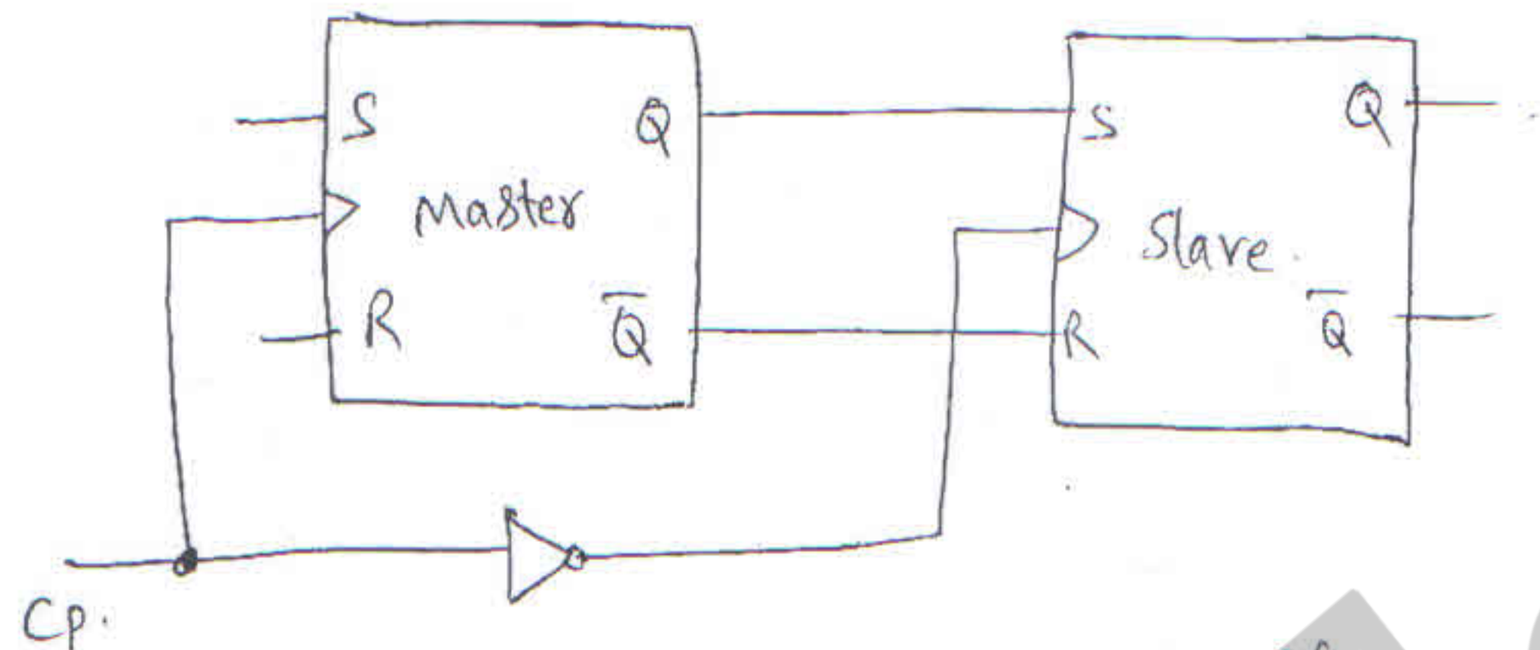


Fig:- I/p and o/p waveforms for T-F/F.

Pothi Reddy. K

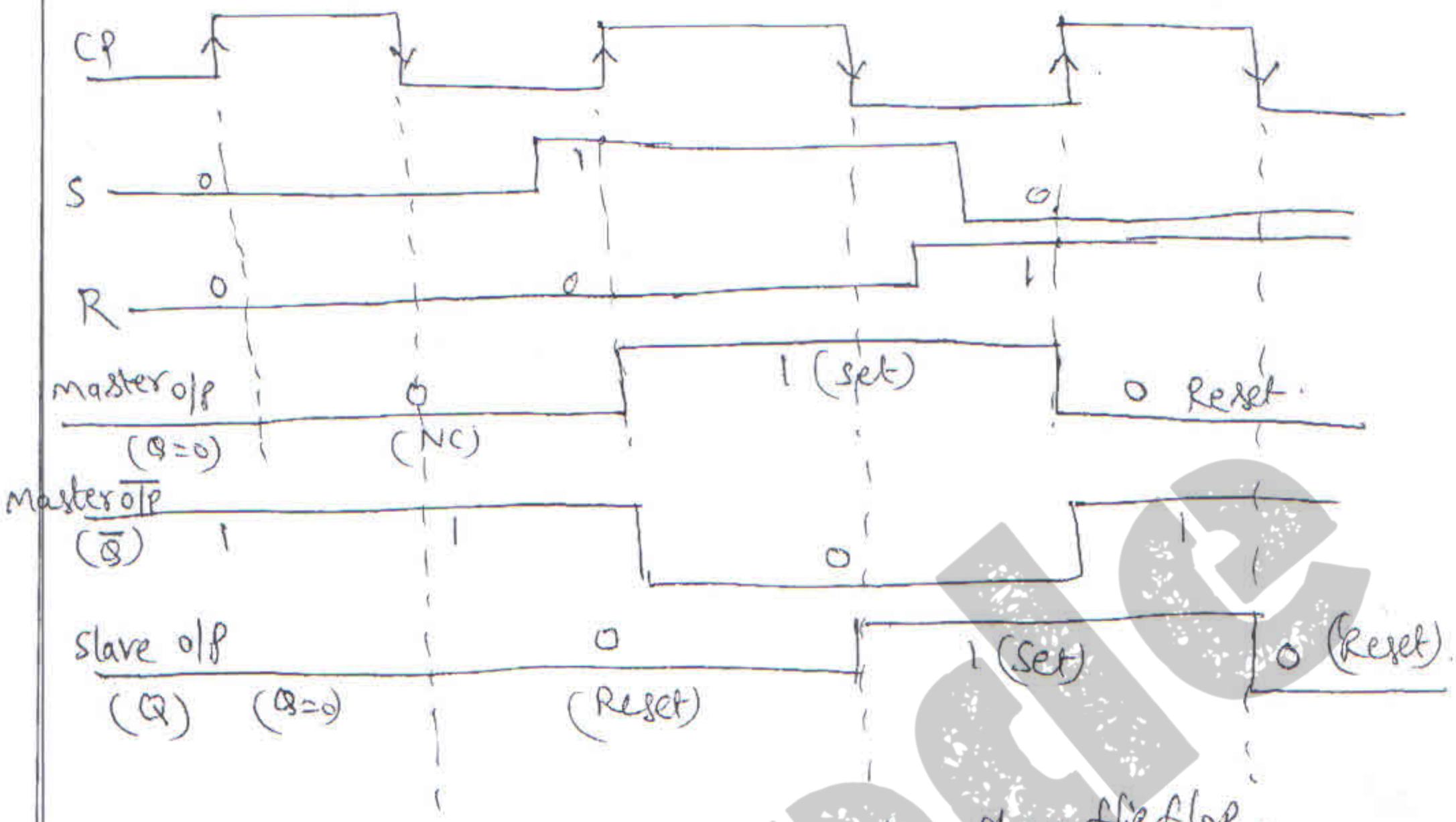
Master - Slave SR - Flip - Flop :



Fig①: Master - Slave SR flip-flop.

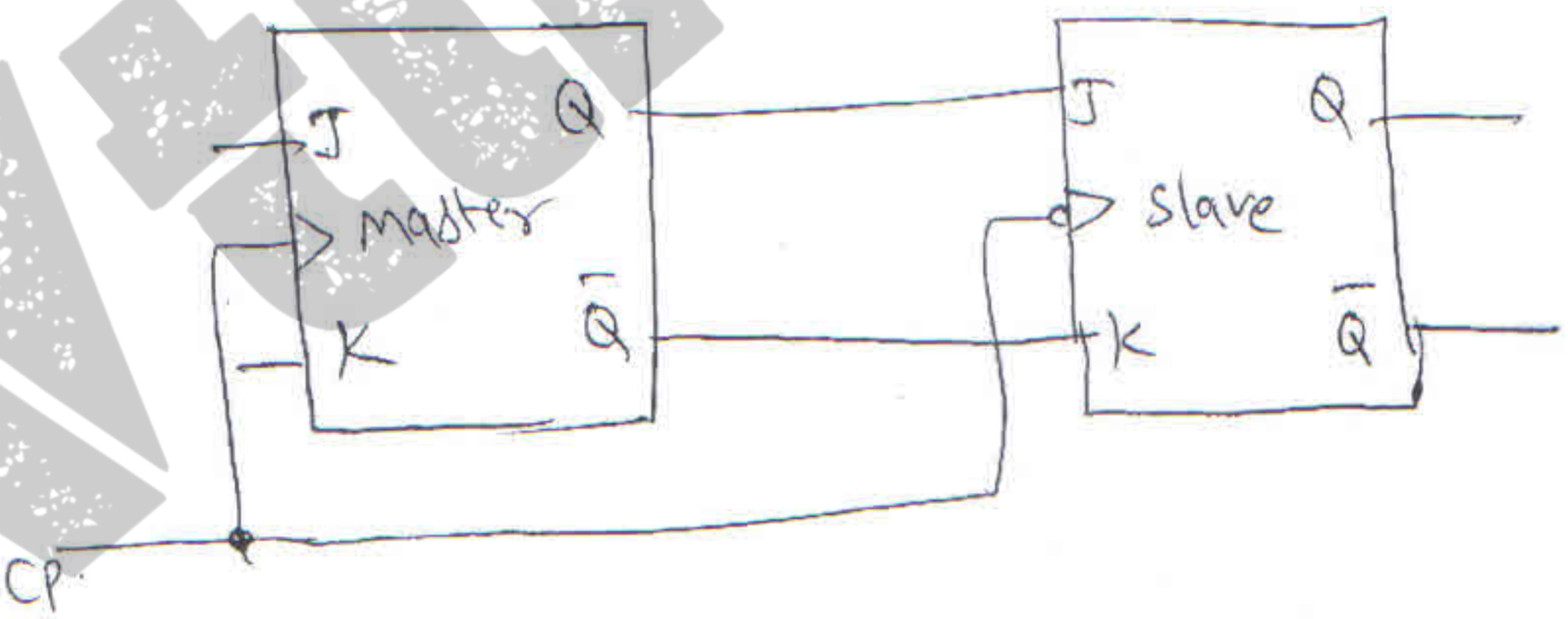
A master - slave flip-flop is constructed from two flip-flops. one circuit serves as a master and the other as a slave, and the overall circuit is referred to as a master-slave flipflop as shown in fig①. It consists of a master flipflop, a slave flipflop and an inverter. Both the flip-flops are positive level - triggered, but inverter connected at the clock input of the slave flip-flop forces it to trigger at the -ve - level.

The output state of the master flipflop is determined by the S and R inputs at the positive CP. The output state of the master is then transferred as an input to the slave flip-flop. The slave flip-flop uses this input at the -ve clock pulse to determine its output state.



Fig(2): I/P & o/P waveforms for master-slave flip flop.

Master - Slave JK - Flip-Flop :-



operation: similar to master-slave SR Flip-flop & waveforms.

Characteristic Equations :

For SR - Flipflop :

S	R	Q_n	Q_{n+1}
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	X
1	1	1	X

Truth Table ↑

K-map

Q_{n+1}

S \ RQ_n	00	01	11	10
0	0	1	0	0
1	1	1	X	X

$\therefore Q_{n+1} = S + \bar{R} Q_n$

For JK - Flipflop :

k-map

J \ KQ_n	00	01	11	10
0	0	1	0	0
1	1	1	0	1

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

Truth Table ↓

J	K	Q_n	Q_{n+1}
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	0

For D - Flip-flop:

Truth Table ↓

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

K-map

D \ Q_n	0	1
0	0	0
1	1	1

D.

$$Q_{n+1} = D$$ ✓

For T- Flip-flop:

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

Truth Table ↑

K-map.

T \ Q_n	0	1
0	0	1
1	1	0

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

$$Q_{n+1} = T \oplus Q_n$$ ✓

ALL THE BEST →

Pothi Reddy. K