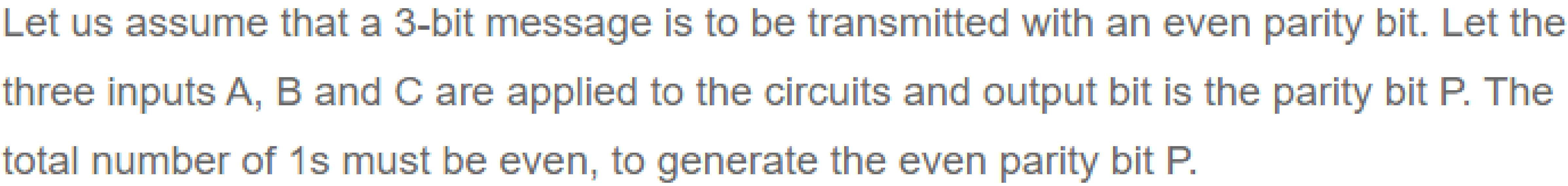
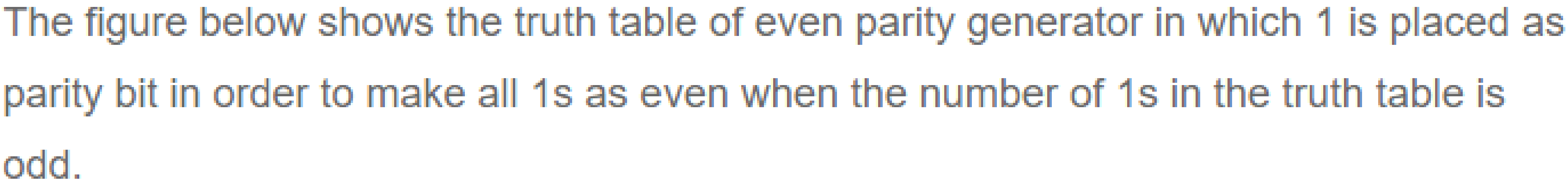
Even Parity Generator

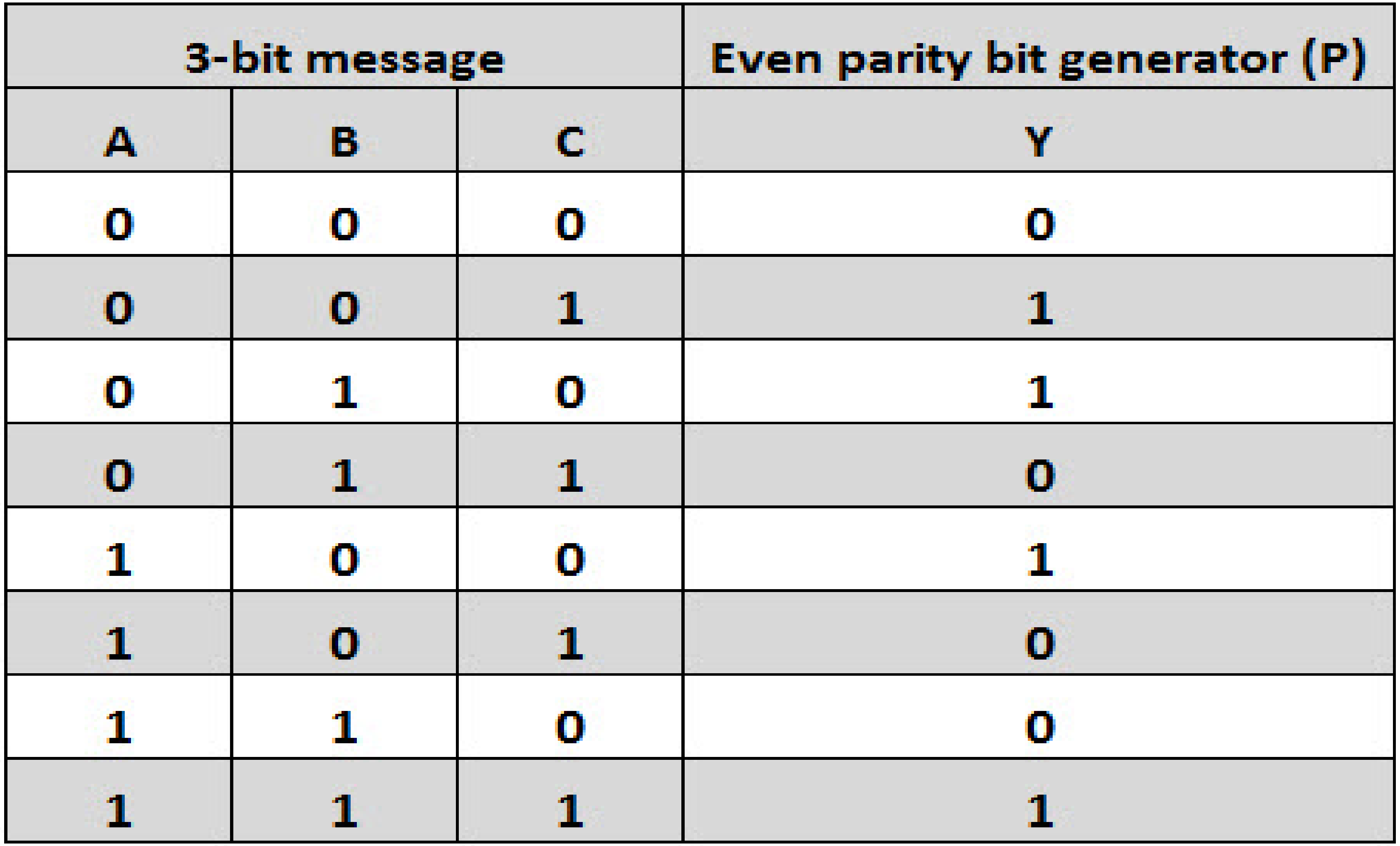


Let us assume that a 3-bit message is to be transmitted with an even parity bit. Let the three inputs A, B and C are applied to the circuits and output bit is the parity bit P. The total number of 1s must be even, to generate the even parity bit P.



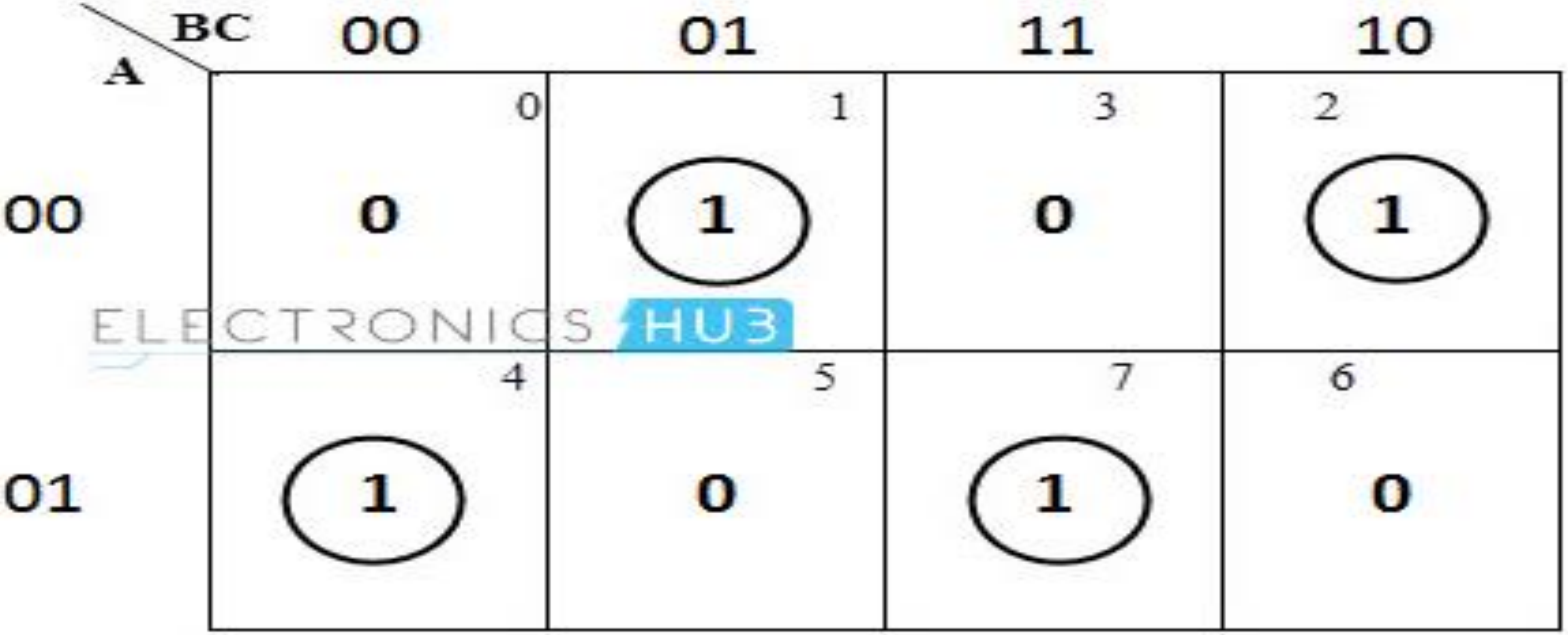
The figure below shows the truth table of even parity generator in which 1 is placed as parity bit in order to make all 1s as even when the number of 1s in the truth table is odd.

The figure below shows the truth table of even parity generator in which 1 is placed as parity bit in order to make all 1s as even when the number of 1s in the truth table is odd.



3-bit message			Even parity bit generator (P)
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

The K-map simplification for 3-bit message even parity generator is



A \ BC	00	01	11	10
00	0 0	1 1	3 0	2 1
01	4 1	5 0	7 1	6 0

BC A	00	01	11	10
00	0 0	1 1	3 0	2 1
01	4 1	5 0	7 1	6 0

From the above truth table, the simplified expression of the parity bit can be written as

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

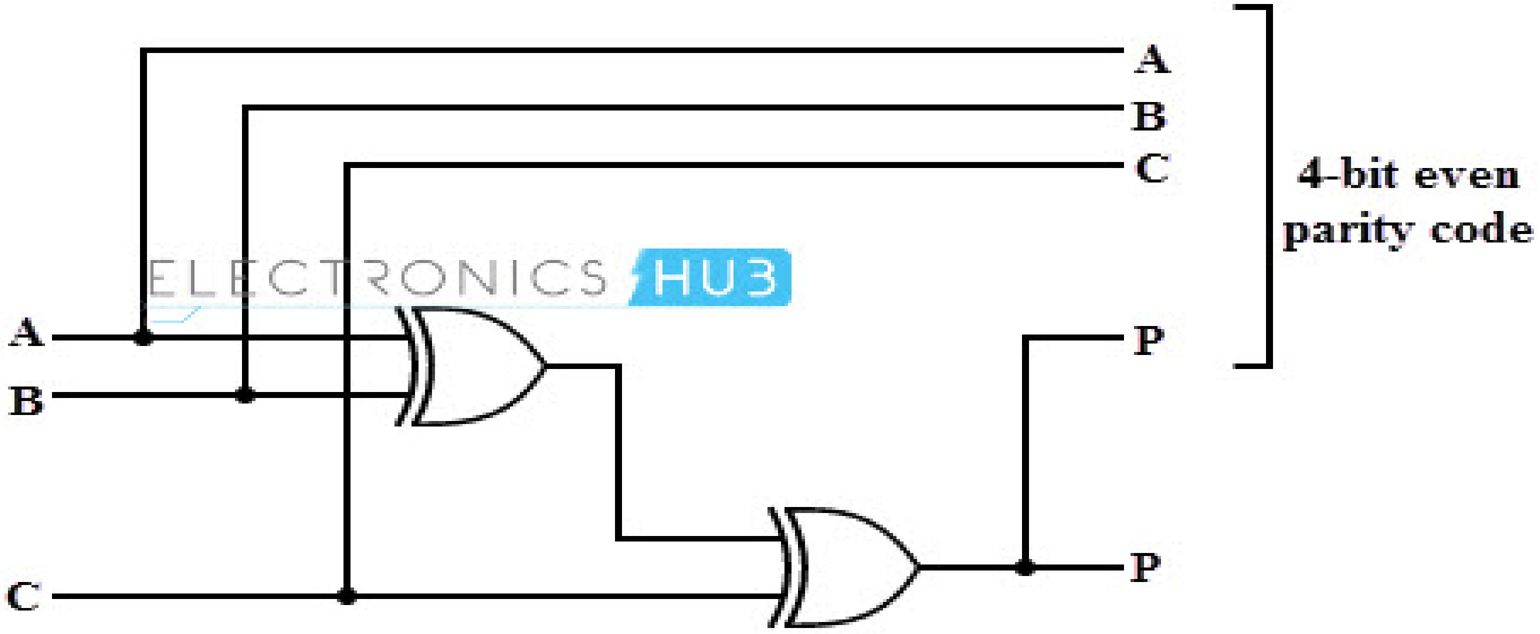
$$= \bar{A}\,(\bar{B}\,C + B\,\bar{C}) + A\,(\bar{B}\,\bar{C} + B\,C)$$

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

$$P = A \oplus B \oplus C$$

The above expression can be implemented by using two Ex-OR gates. The logic diagram of even parity generator with two Ex – OR gates is shown below. The three bit message along with the parity generated by this circuit which is transmitted to the receiving end where parity checker circuit checks whether any error is present or not.

To generate the even parity bit for a 4-bit data, three Ex-OR gates are required to add the 4-bits and their sum will be the parity bit.



Q:- Design a parity generator using basic gates to produce digital words with odd parity. Assume the input to be three bits binary words?

Soln:-

B_2	B_1	B_0	Odd parity bit P
0	0	0	1
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	0

Kmap:-

$\overline{B_2}B_1B_0$

$B_2 \backslash B_1B_0$	00	01	11	10
0	1	0	1	0
1	0	1	0	1

$\overline{B_2}\,\overline{B_1}\,\overline{B_0}$ $B_2B_1\overline{B_0}$

$B_2\overline{B_1}B_0$

$\Rightarrow \overline{B_2}\,\overline{B_1}\,\overline{B_0} + \overline{B_2}B_1B_0 + B_2\overline{B_1}B_0 + B_2B_1\overline{B_0}$

$\Rightarrow \overline{B_2}(\overline{B_1}\,\overline{B_0} + B_1B_0) + B_2(\overline{B_1}B_0 + B_1\overline{B_0})$

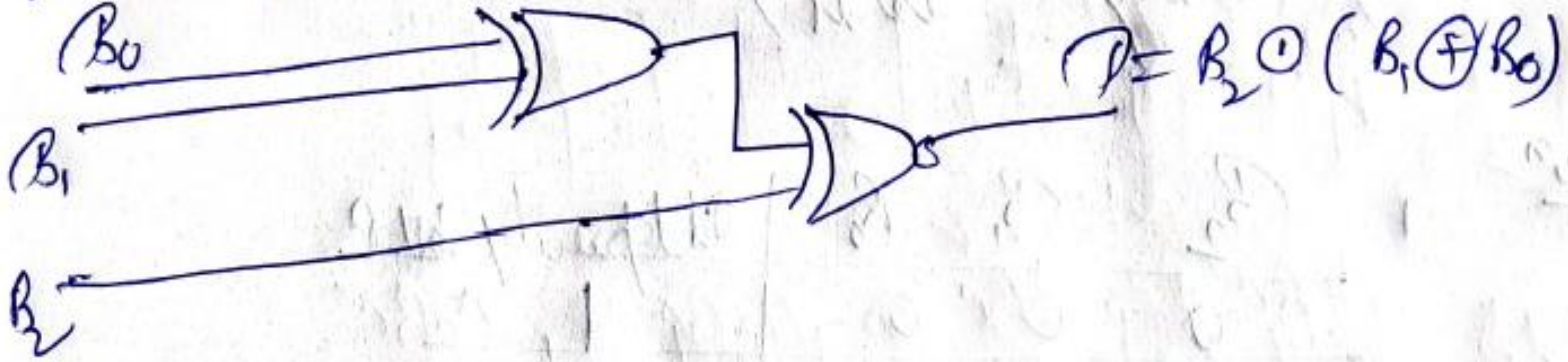
$\Rightarrow \overline{B_2}(\overline{B_1 \oplus B_0}) + B_2(B_1 \oplus B_0)$ $\{\overline{B_2}\,\bar{x} + B_2x\}$

$\{B_2 \odot x\}$

$\Rightarrow B_2 \odot B_1 \oplus B_0$

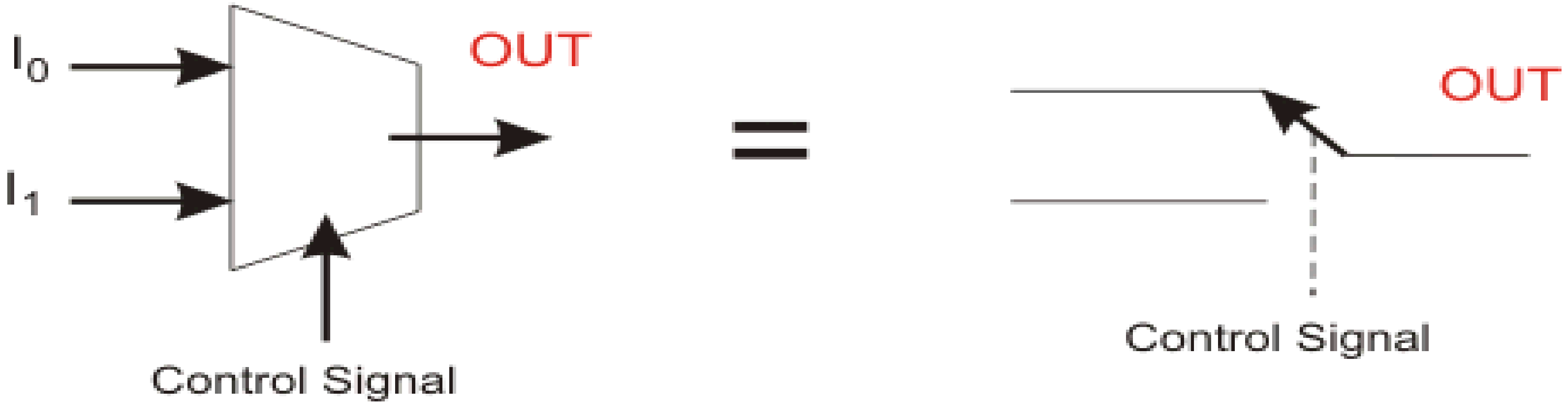
$$P = B_2 \odot (B_1 \oplus B_0)$$

Logic diagram:-

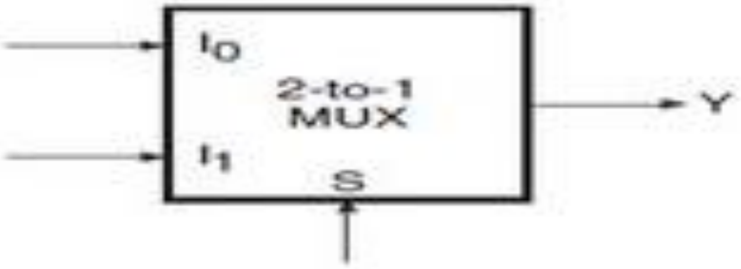


MUX

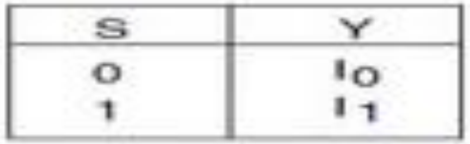
A *multiplexer* is best defined as a combinational logic circuit that acts as a switcher for multiple inputs to a single common output line. Also known as "MUX" it delivers either digital or analog signals at a higher speed on a single line and in one shared device but then recovers the separate signals at the receiving end. An MUX has a maximum of 2^n (two raised to n) data inputs. One of the inputs is connected to the output based on the value of the selection lines. There will be 2^n possible combinations of 1s and 0s since there are 'n' selection lines.



2:1 Mux



(a)

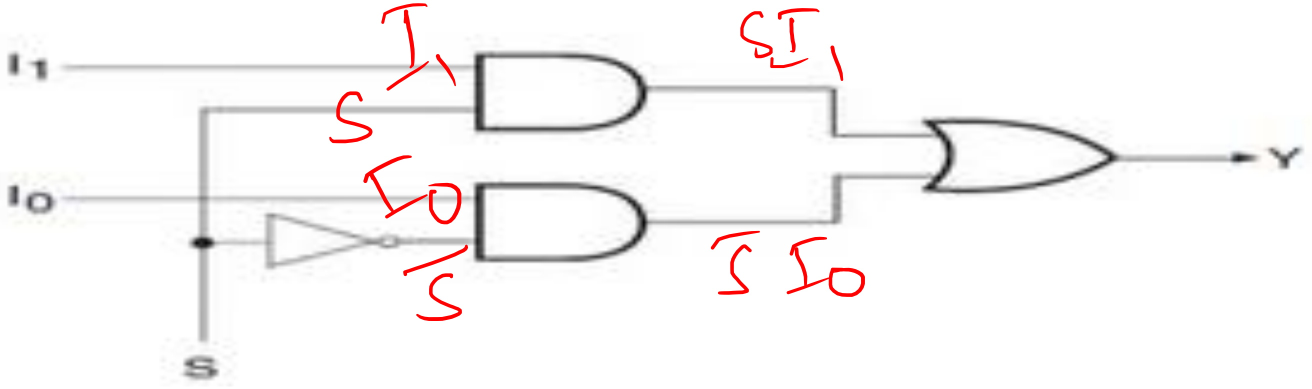


S	Y
0	I$_0$
1	I$_1$

(b)

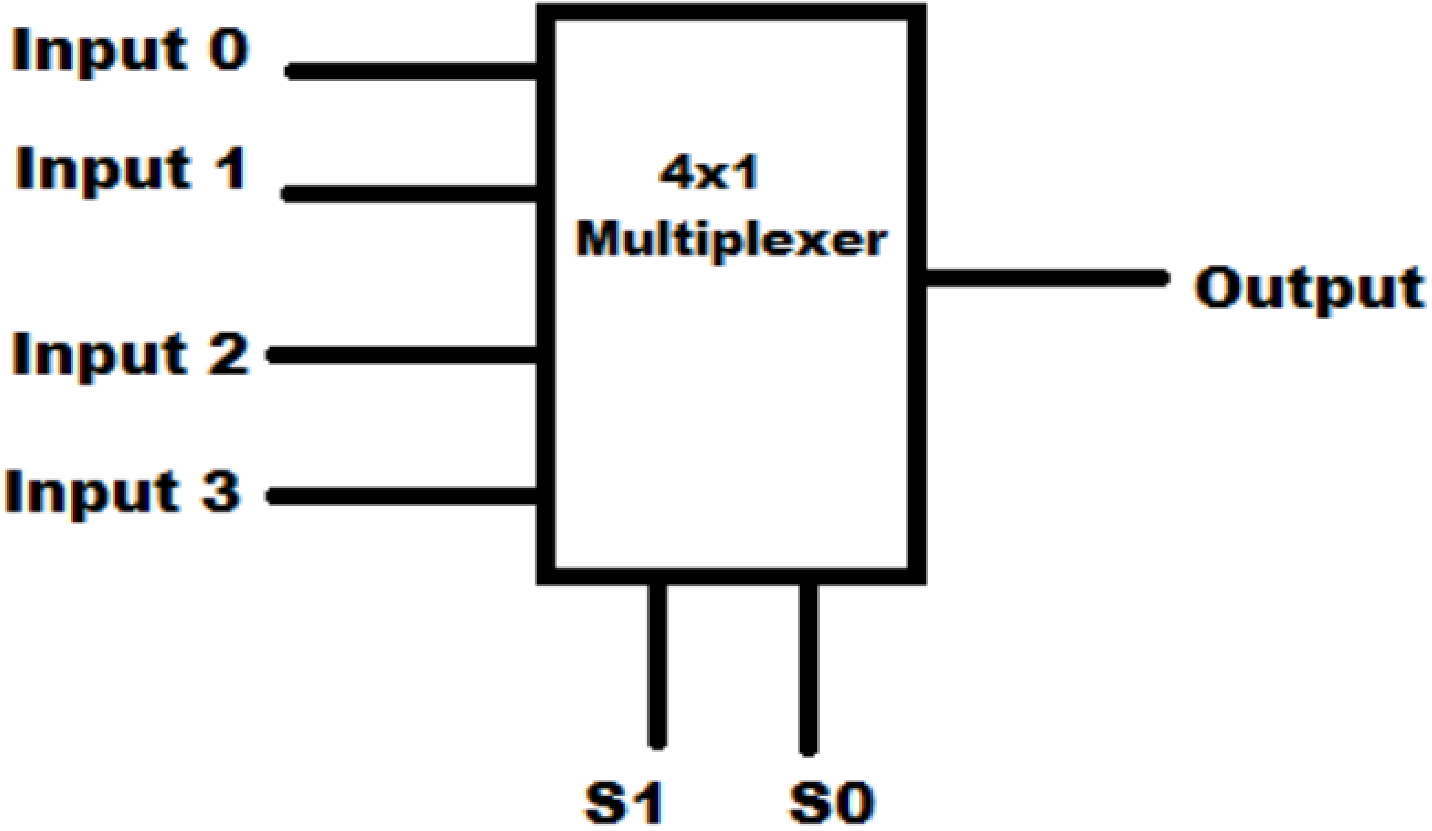
Equation (2:1) Mux

$$Y = \bar{S}I_0 + SI_1$$



$= \bar{S}I_0 + SI_1$

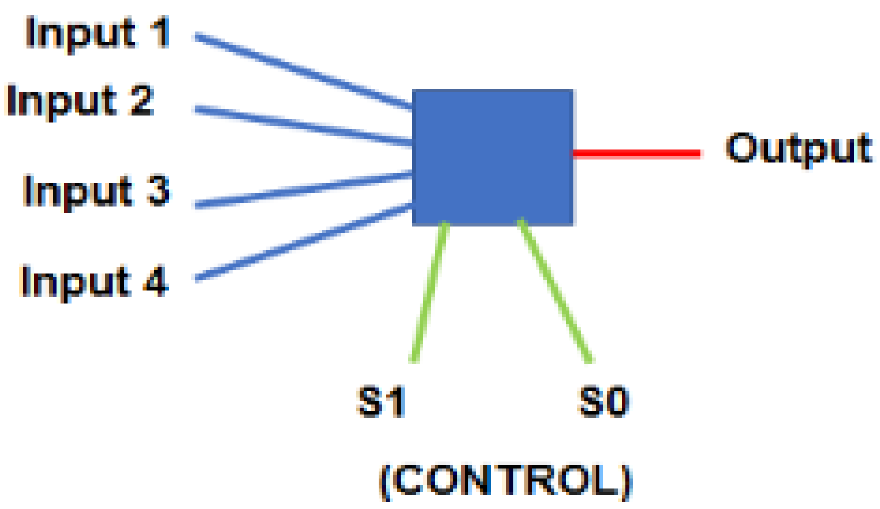
4:1 Mux



2^n :1

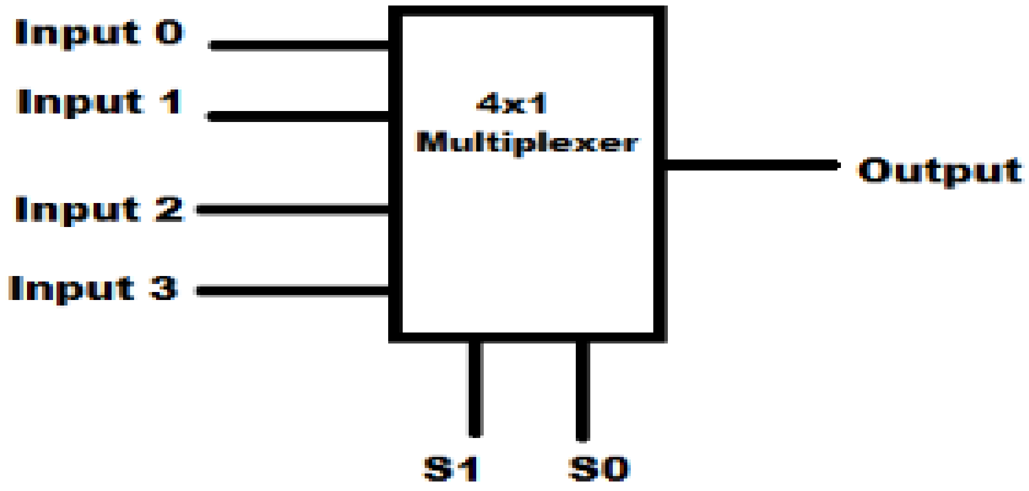
2^2 :1

HOW DOES A MULTIPLEXER WORK?

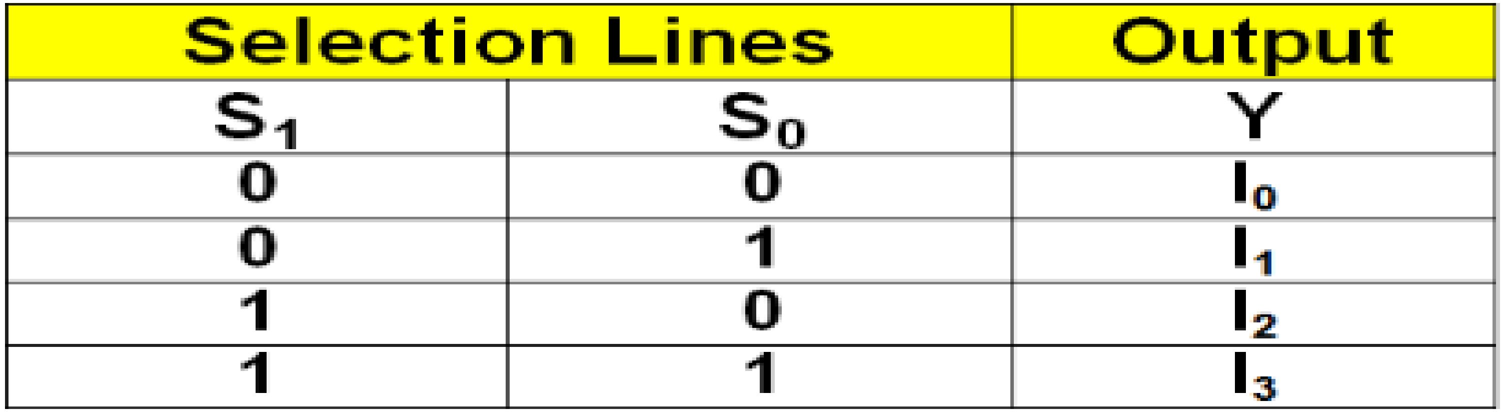


In this figure, the switch has four inputs with one output pin and you can select based on the signal given. It demonstrates the three basic parts of any multiplexer namely the input pins, output pins, and control signals.

- Input Pins – these are all the available input signals from which the best required signal has to be selected. It can be analog or digital.
- Output Pin – the chosen input signal will be provided by the output pin.
- Control/Selection Pin – this selects the input pin signal. The number of control pins depends on the number of input pins.



Any of the four inputs will be connected to the output based on the combination present at these two selection lines.

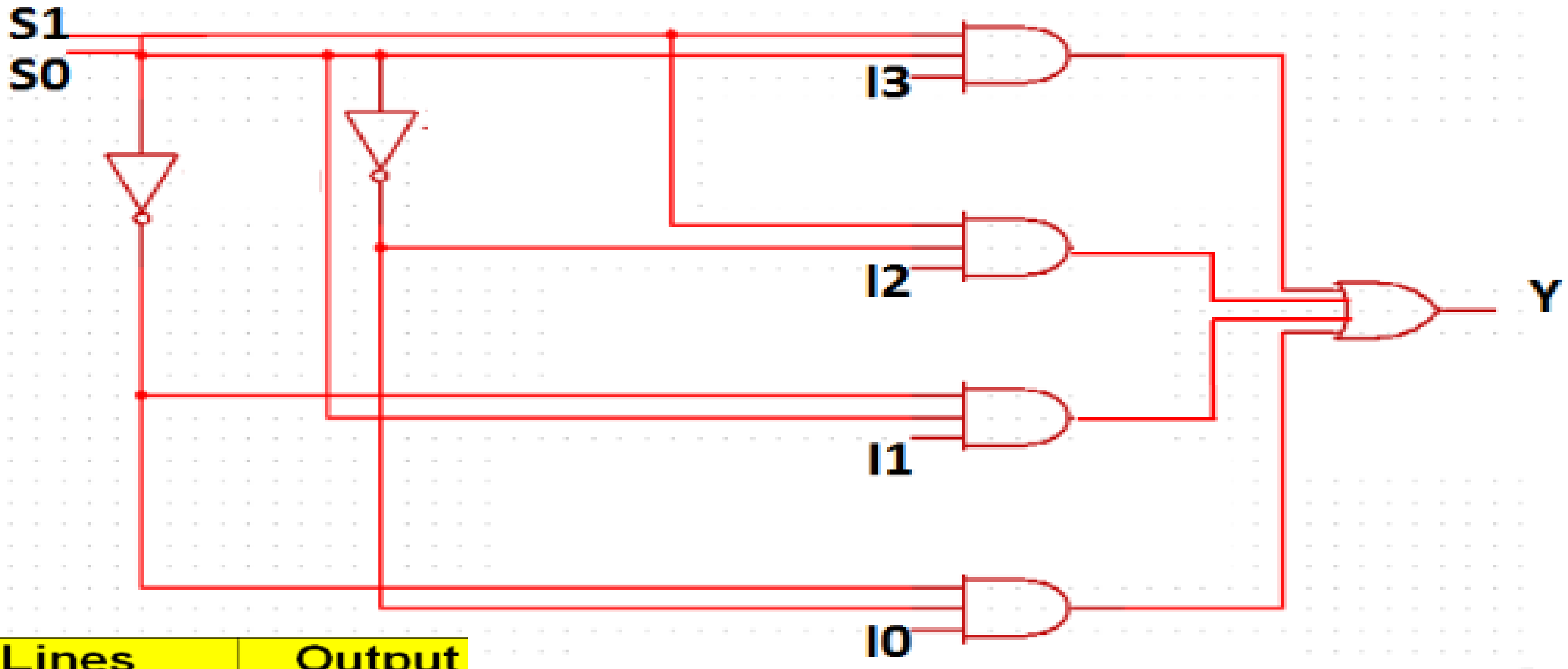


Selection Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Truth Table of 4×1 Multiplexer

$$Y=S1'S0'I0+S1'S0I1+S1S0'I2+S1S0I3$$

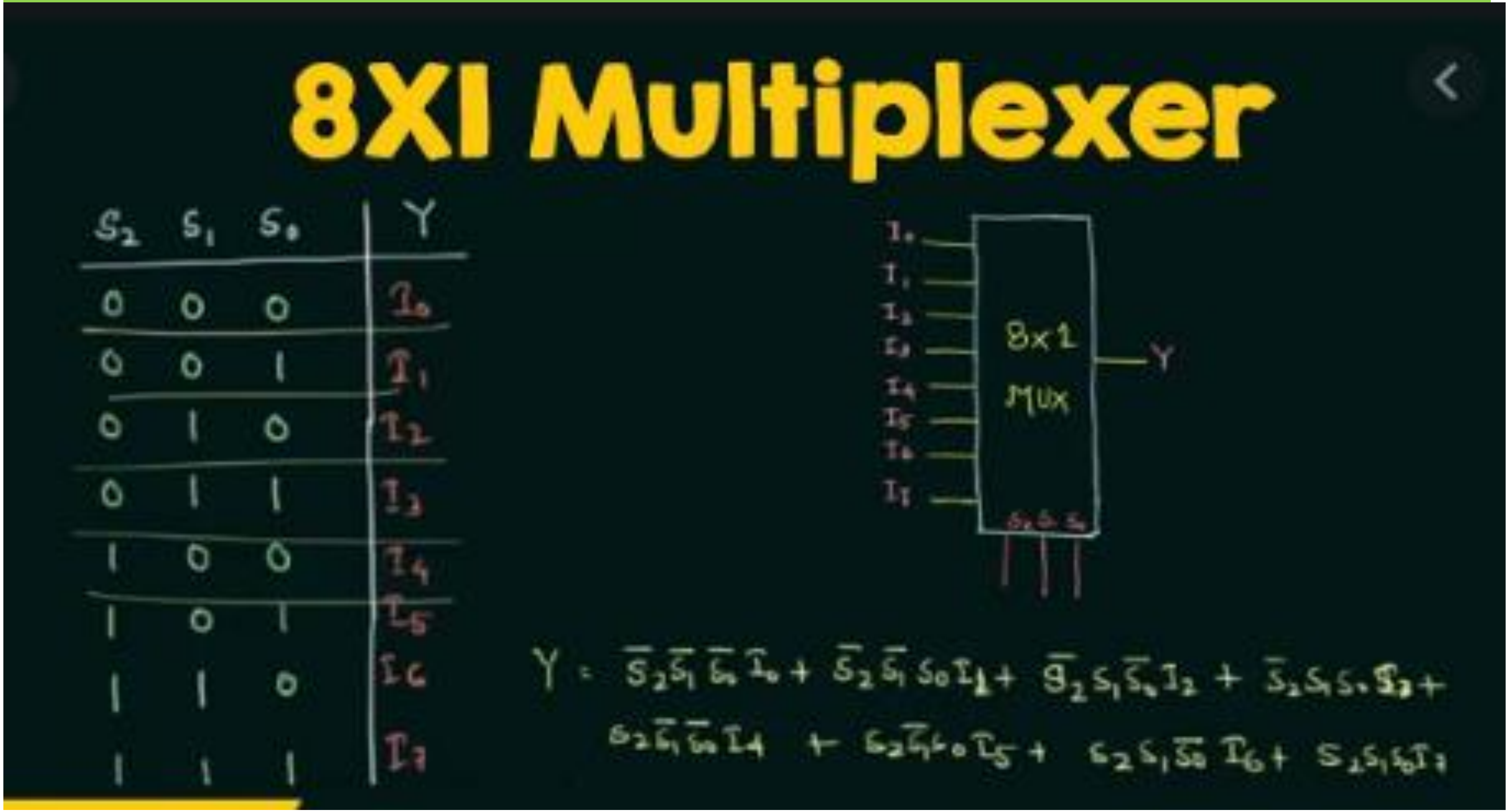
Given the Boolean function, we can implement the 4×1 multiplexer using inverters in this circuit diagram.



Circuit Diagram of 4×1 Multiplexer

Selection Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

8:1 Mux



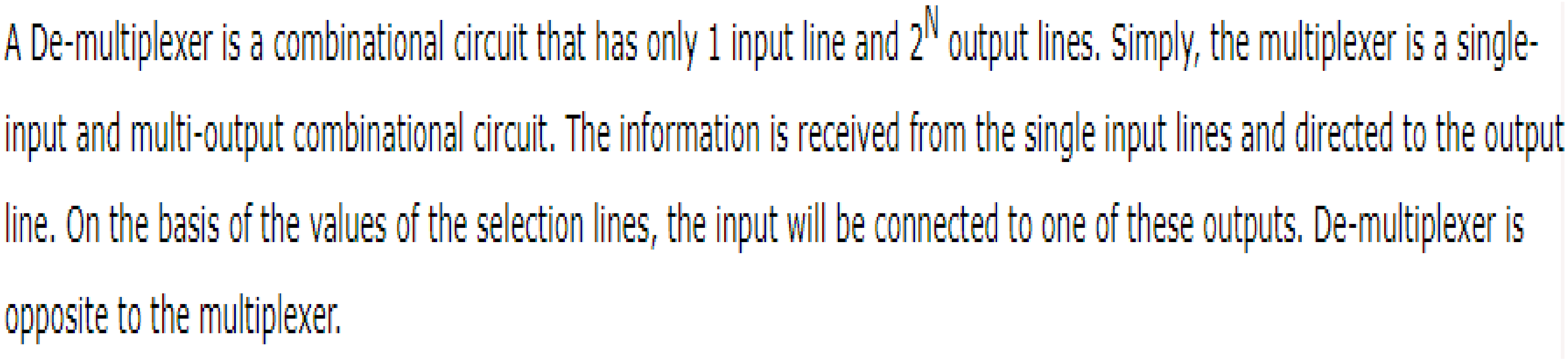
8XI Multiplexer

S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

$$Y = \bar{S}_2\bar{S}_1\bar{S}_0 I_0 + \bar{S}_2\bar{S}_1 S_0 I_1 + \bar{S}_2 S_1\bar{S}_0 I_2 + \bar{S}_2 S_1 S_0 I_3 +$$

$$S_2\bar{S}_1\bar{S}_0 I_4 + S_2\bar{S}_1 S_0 I_5 + S_2 S_1\bar{S}_0 I_6 + S_2 S_1 S_0 I_7$$

De-multiplexer(De-Mux)

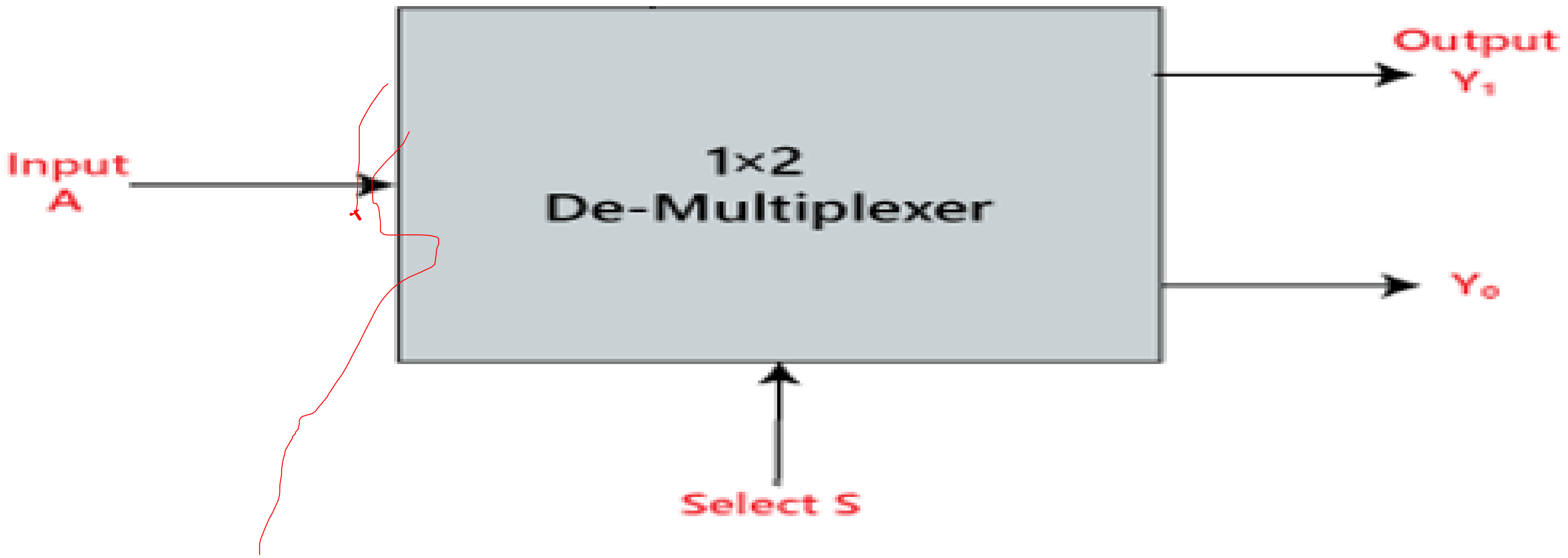


A De-multiplexer is a combinational circuit that has only 1 input line and 2^N output lines. Simply, the multiplexer is a single-input and multi-output combinational circuit. The information is received from the single input lines and directed to the output line. On the basis of the values of the selection lines, the input will be connected to one of these outputs. De-multiplexer is opposite to the multiplexer.

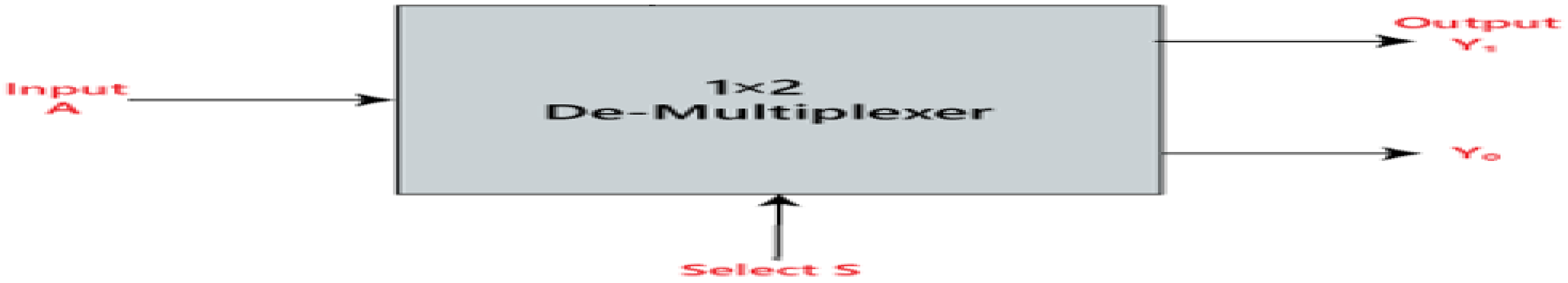
1×2 De-multiplexer:

In the 1 to 2 De-multiplexer, there are only two outputs, i.e., Y_0, and Y_1, 1 selection lines, i.e., S_0, and single input, i.e., A. On the basis of the selection value, the input will be connected to one of the outputs. The block diagram and the truth table of the 1×2 multiplexer are given below.

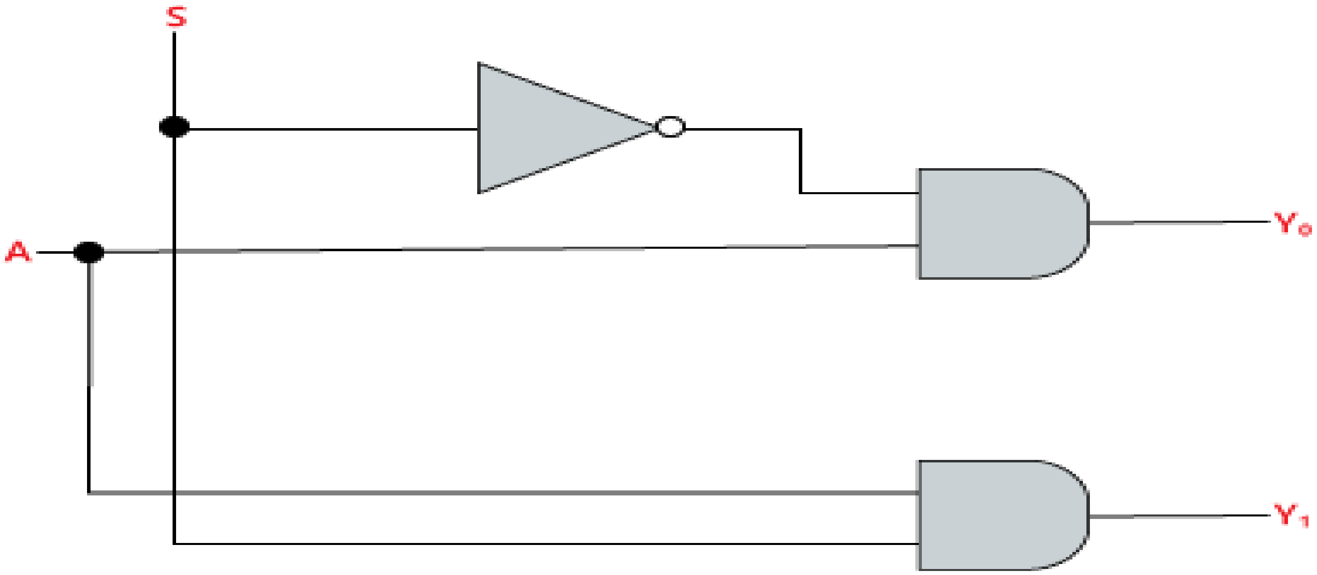
Block Diagram:



Block Diagram:



Logical circuit of the above expressions is given below:



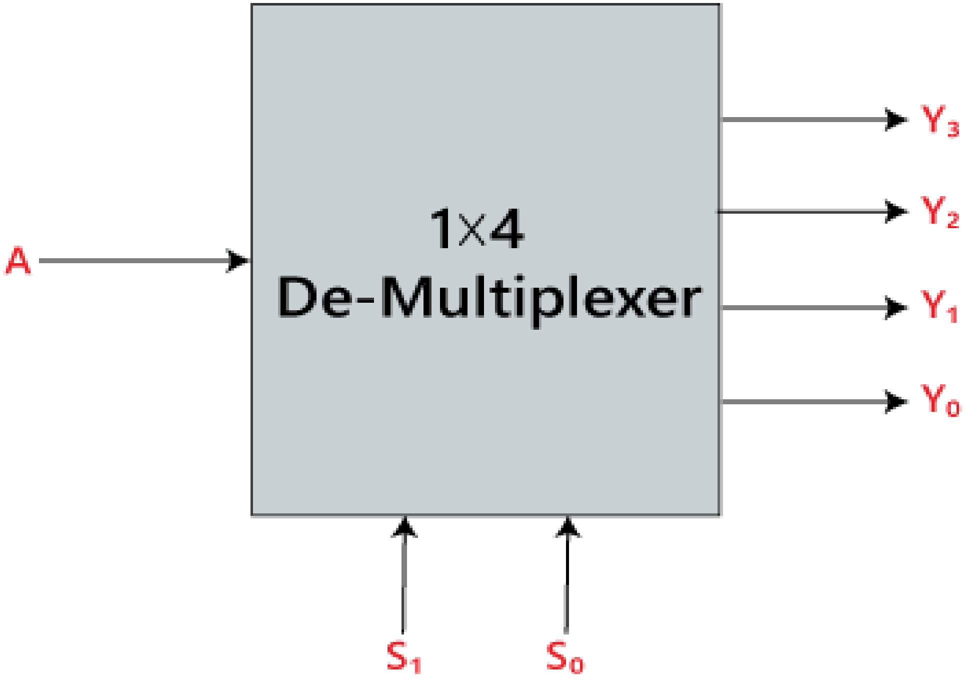
Truth Table:

INPUTS	Output	
S_0	Y_1	Y_0
0	0	A
1	A	0

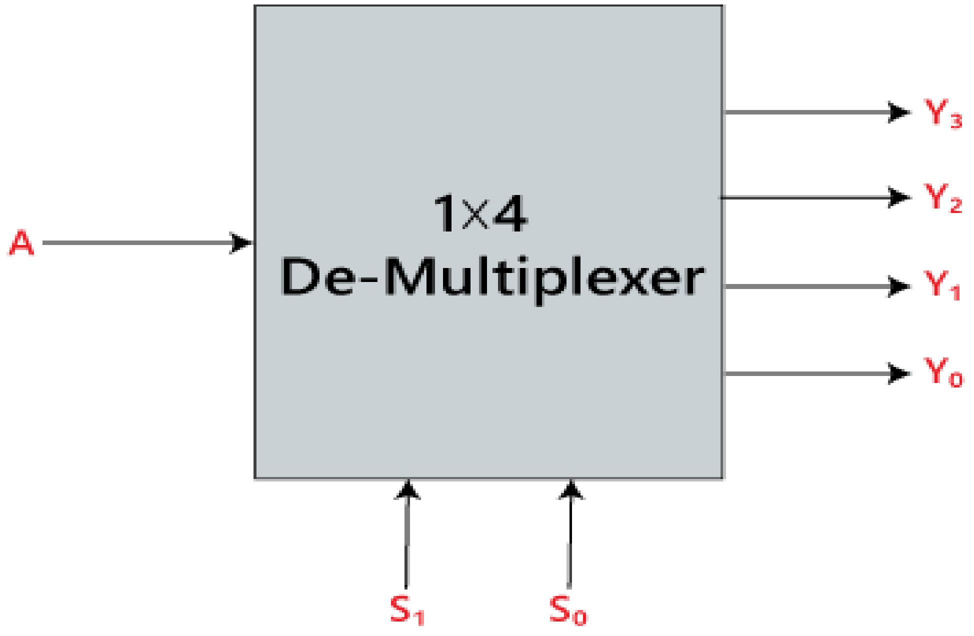
1×4 De-multiplexer:

In 1 to 4 De-multiplexer, there are total of four outputs, i.e., Y_0, Y_1, Y_2, and Y_3, 2 selection lines, i.e., S_0 and S_1 and single input, i.e., A. On the basis of the combination of inputs which are present at the selection lines S_0 and S_1, the input be connected to one of the outputs. The block diagram and the truth table of the 1×4 multiplexer are given below.

Block Diagram:



Block Diagram:



Truth Table:

INPUTS		Output			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	A
0	1	0	0	A	0
1	0	0	A	0	0
1	1	A	0	0	0

The logical expression of the term Y is as follows:

$Y_0=S_1' S_0' A$

$y_1=S_1' S_0 A$

$y_2=S_1 S_0' A$

$y_3=S_1 S_0 A$

The logical expression of the term Y is as follows:

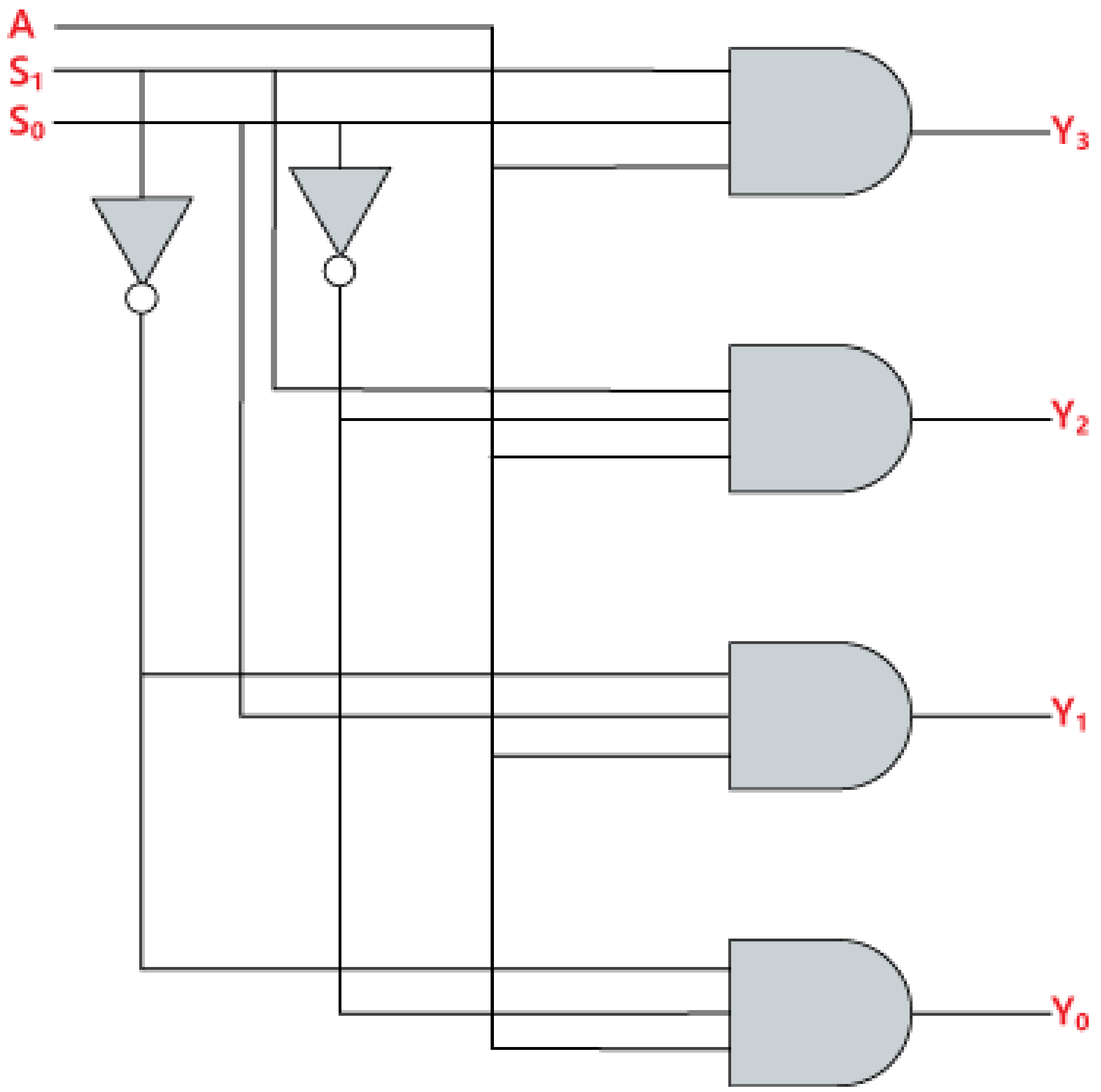
$Y_0=S_1' S_0' A$

$Y_1=S_1' S_0 A$

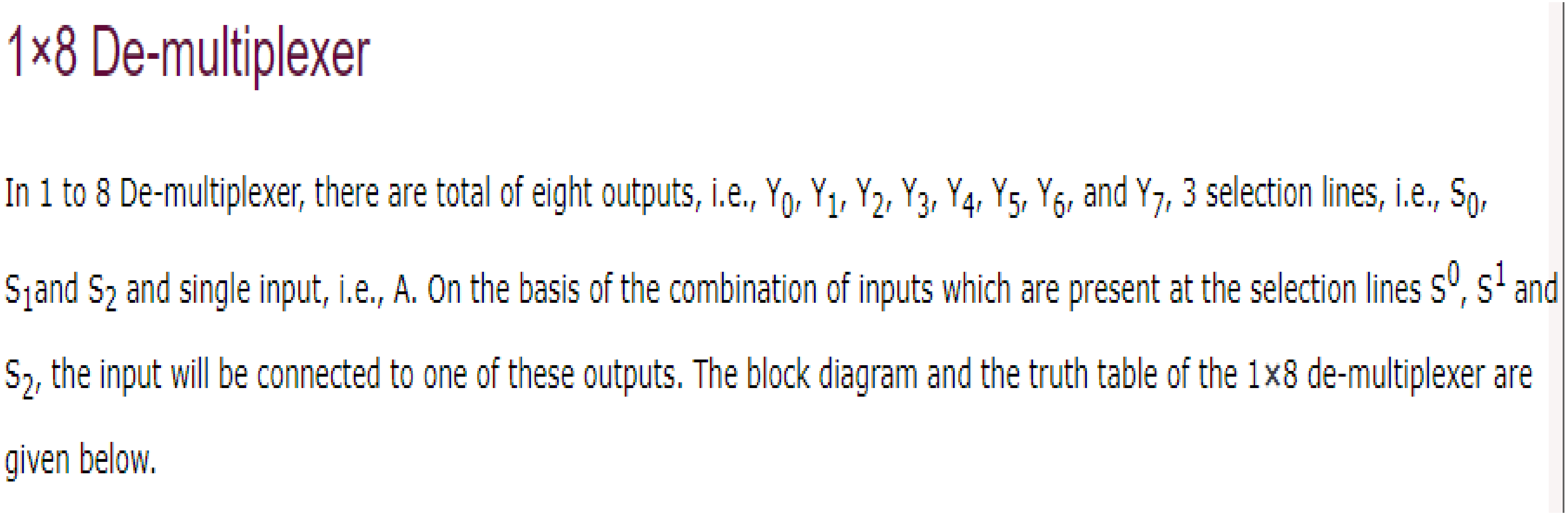
$Y_2=S_1 S_0' A$

$Y_3=S_1 S_0 A$

Logical circuit of the above expressions is given below:



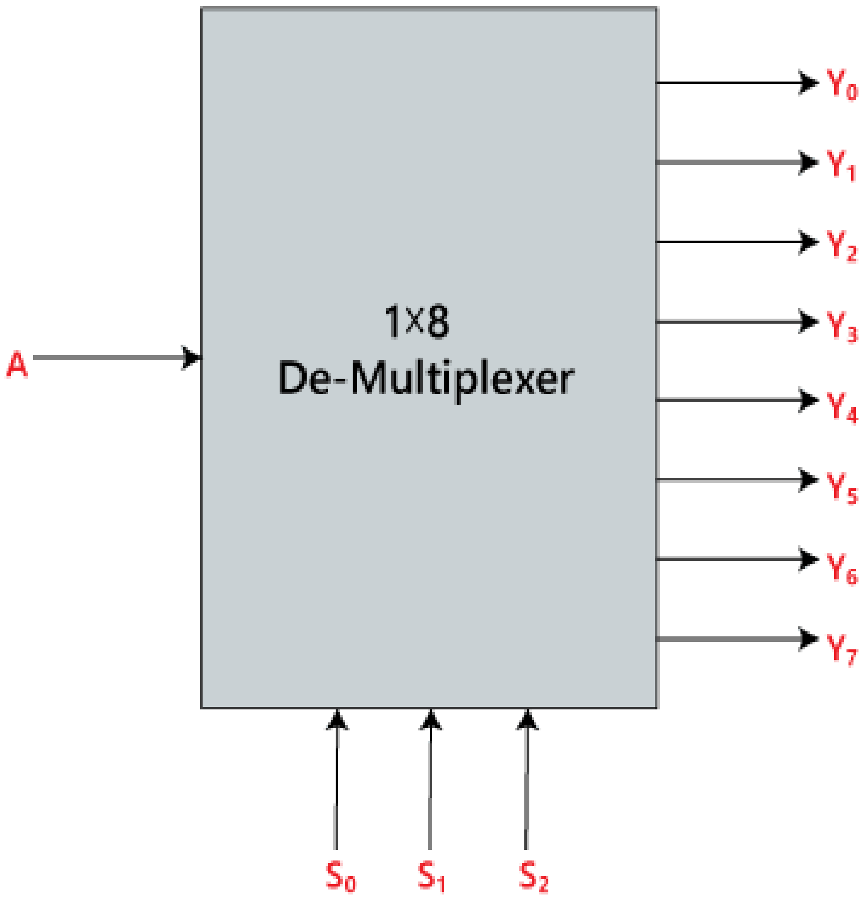
1:8 Demux



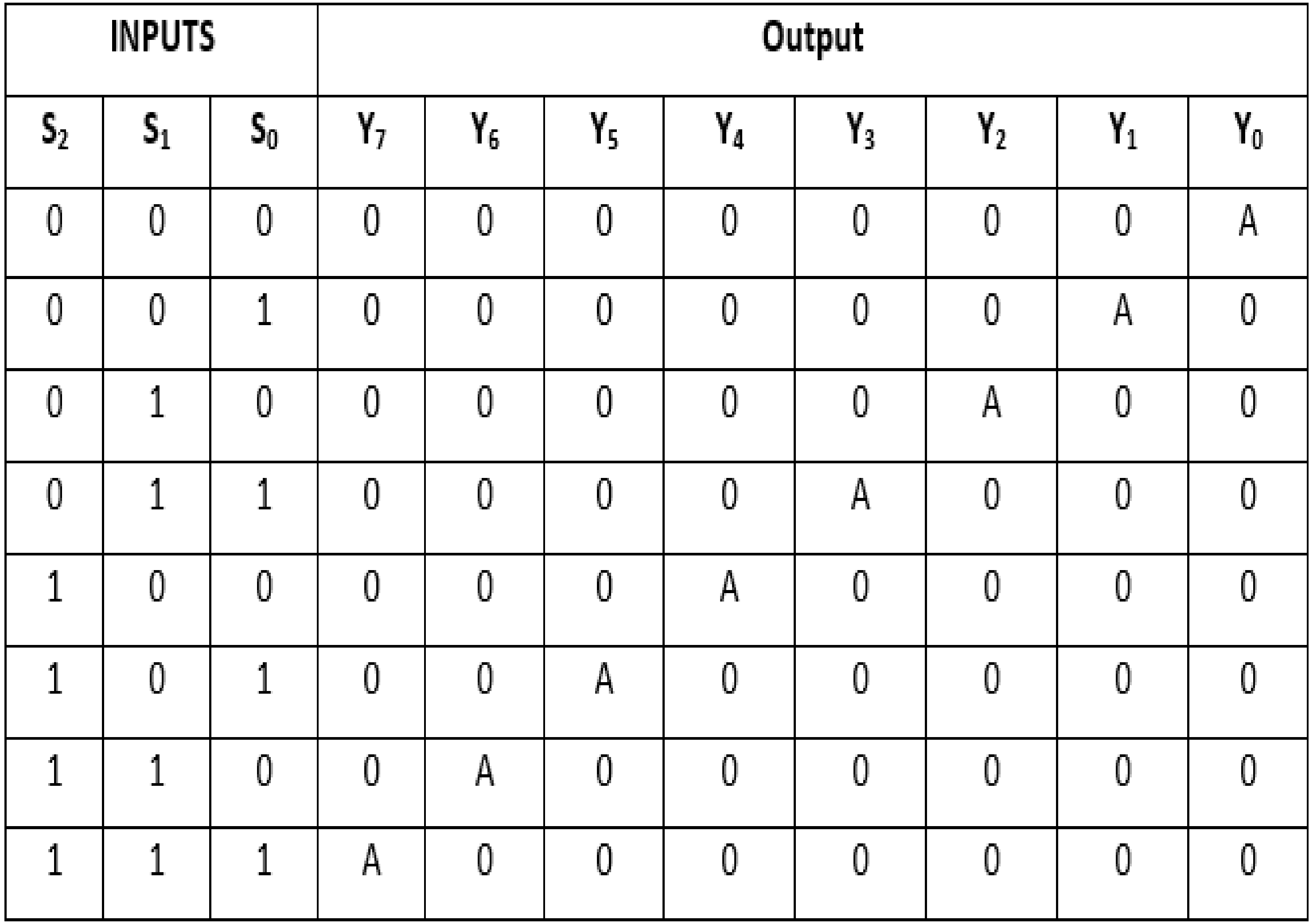
1×8 De-multiplexer

In 1 to 8 De-multiplexer, there are total of eight outputs, i.e., Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, and Y_7, 3 selection lines, i.e., S_0, $S_1$and S_2 and single input, i.e., A. On the basis of the combination of inputs which are present at the selection lines S^0, S^1 and S_2, the input will be connected to one of these outputs. The block diagram and the truth table of the 1×8 de-multiplexer are given below.

Block Diagram:



Truth Table:



INPUTS			Output							
S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	A
0	0	1	0	0	0	0	0	0	A	0
0	1	0	0	0	0	0	0	A	0	0
0	1	1	0	0	0	0	A	0	0	0
1	0	0	0	0	0	A	0	0	0	0
1	0	1	0	0	A	0	0	0	0	0
1	1	0	0	A	0	0	0	0	0	0
1	1	1	A	0	0	0	0	0	0	0

Truth Table:

INPUTS			Output							
S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	A
0	0	1	0	0	0	0	0	0	A	0
0	1	0	0	0	0	0	0	A	0	0
0	1	1	0	0	0	0	A	0	0	0
1	0	0	0	0	0	A	0	0	0	0
1	0	1	0	0	A	0	0	0	0	0
1	1	0	0	A	0	0	0	0	0	0
1	1	1	A	0	0	0	0	0	0	0

The logical expression of the term Y is as follows:

$Y_0=S_0'.S_1'.S_2'.A$

$Y_1=S_0.S_1'.S_2'.A$

$Y_2=S_0'.S_1.S_2'.A$

$Y_3=S_0.S_1.S_2'.A$

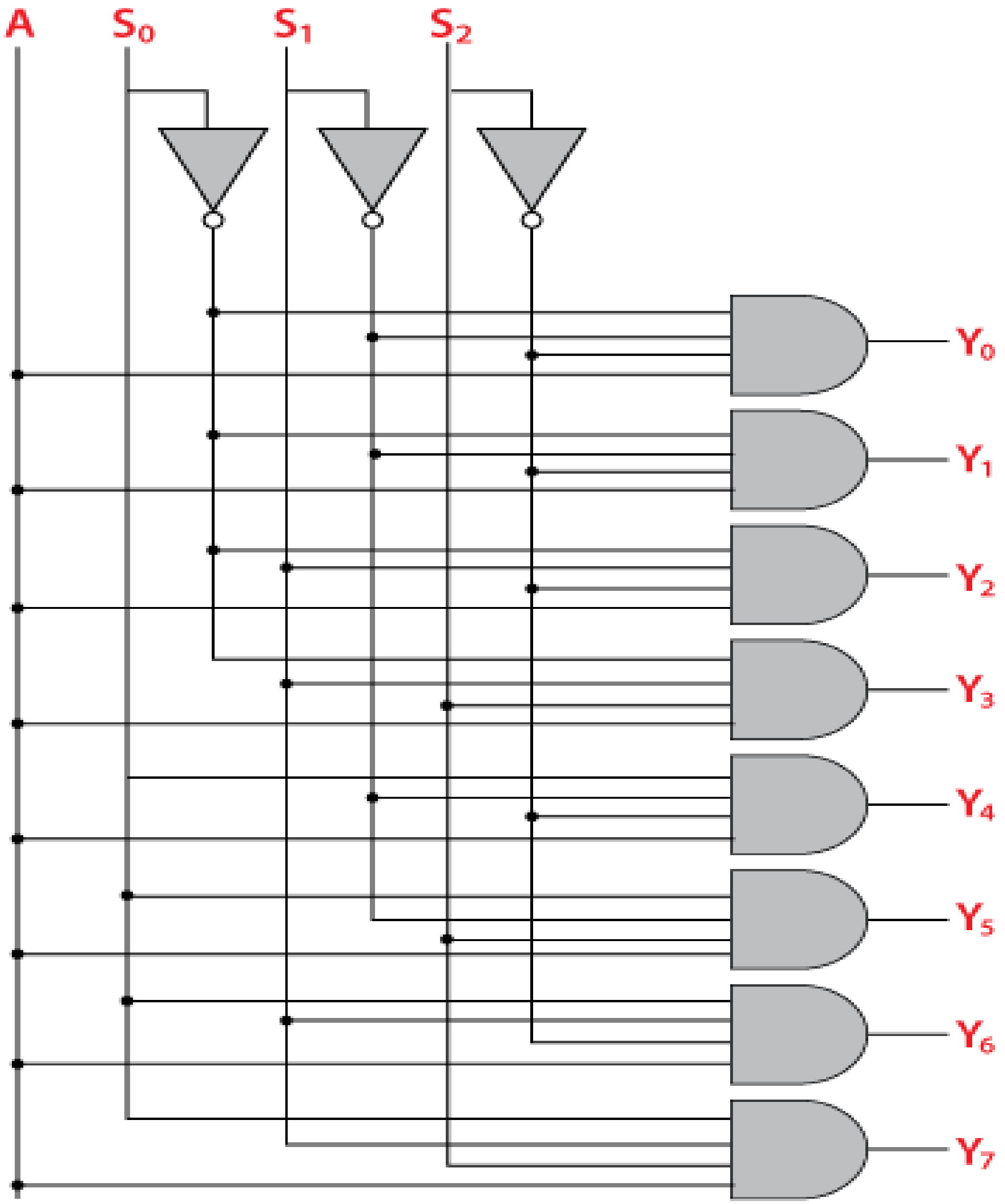
$Y_4=S_0'.S_1'.S_2\ A$

$Y_5=S_0.S_1'.S_2\ A$

$Y_6=S_0'.S_1.S_2\ A$

$Y_7=S_0.S_1.S_3.A$

Logical circuit of the above expressions is given below:



The logical expression of the term Y is as follows:

$Y_0=S_0'.S_1'.S_2'.A$

$Y_1=S_0.S_1'.S_2'.A$

$Y_2=S_0'.S_1.S_2'.A$

$Y_3=S_0.S_1.S_2'.A$

$Y_4=S_0'.S_1'.S_2\ A$

$Y_5=S_0.S_1'.S_2\ A$

$Y_6=S_0'.S_1.S_2\ A$

$Y_7=S_0.S_1.S_{\underset{2}{3}}.A$

function implementation using Mux

Implement the logic function $Y(A,B,C) = \Sigma m\,(0,1,3,5,7)$ using

a) 8:1 Mux

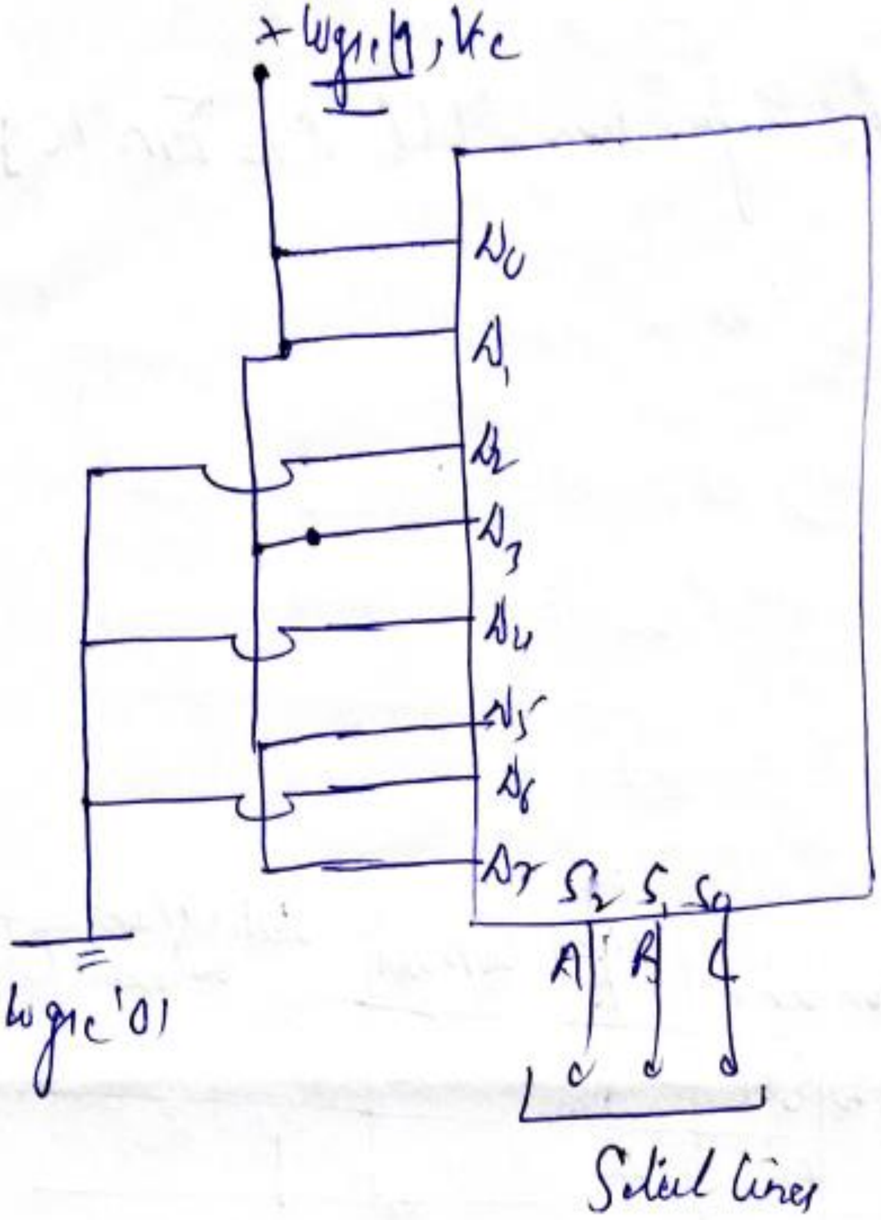
b) 4:1 Mux

c) 2:1 Mux

Solution:-

Decimal	Selection Line			Input to mux	Select Minterm for output Y
	$S_2 = A$	$S_1 = B$	$S_0 = C$		
0	0	0	0	D_0	1
1	0	0	1	D_1	1
2	0	1	0	D_2	0
3	0	1	1	D_3	1
4	1	0	0	D_4	0
5	1	0	1	D_5	1
6	1	1	0	D_6	0
7	1	1	1	D_7	1

Implementation using 8:1 mux



Solution:-

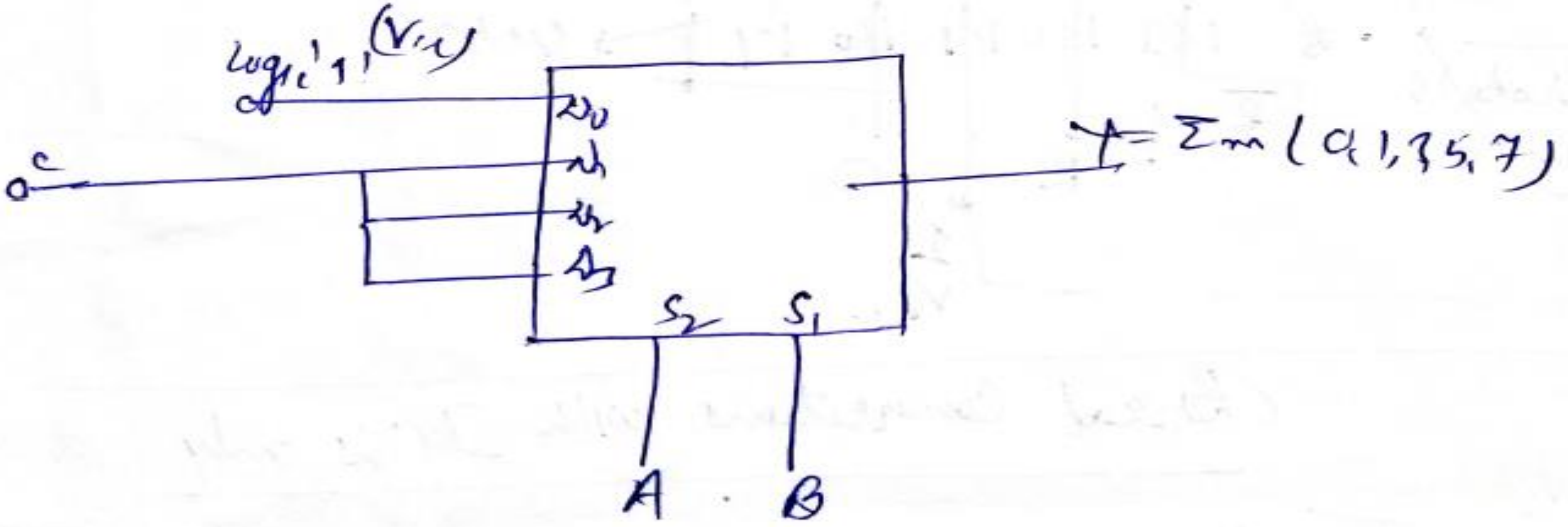
Decimal	Selection line			Input to mux	Select Minterm for output Y
	$S_2=A$	$S_1=B$	$S_0=C$		
0	0	0	0	D_0	1
1	0	0	1	D_1	1
2	0	1	0	D_2	0
3	0	1	1	D_3	1
4	1	0	0	D_4	0
5	1	0	1	D_5	1
6	1	1	0	D_6	0
7	1	1	1	D_7	1

Using 4:1 Mux

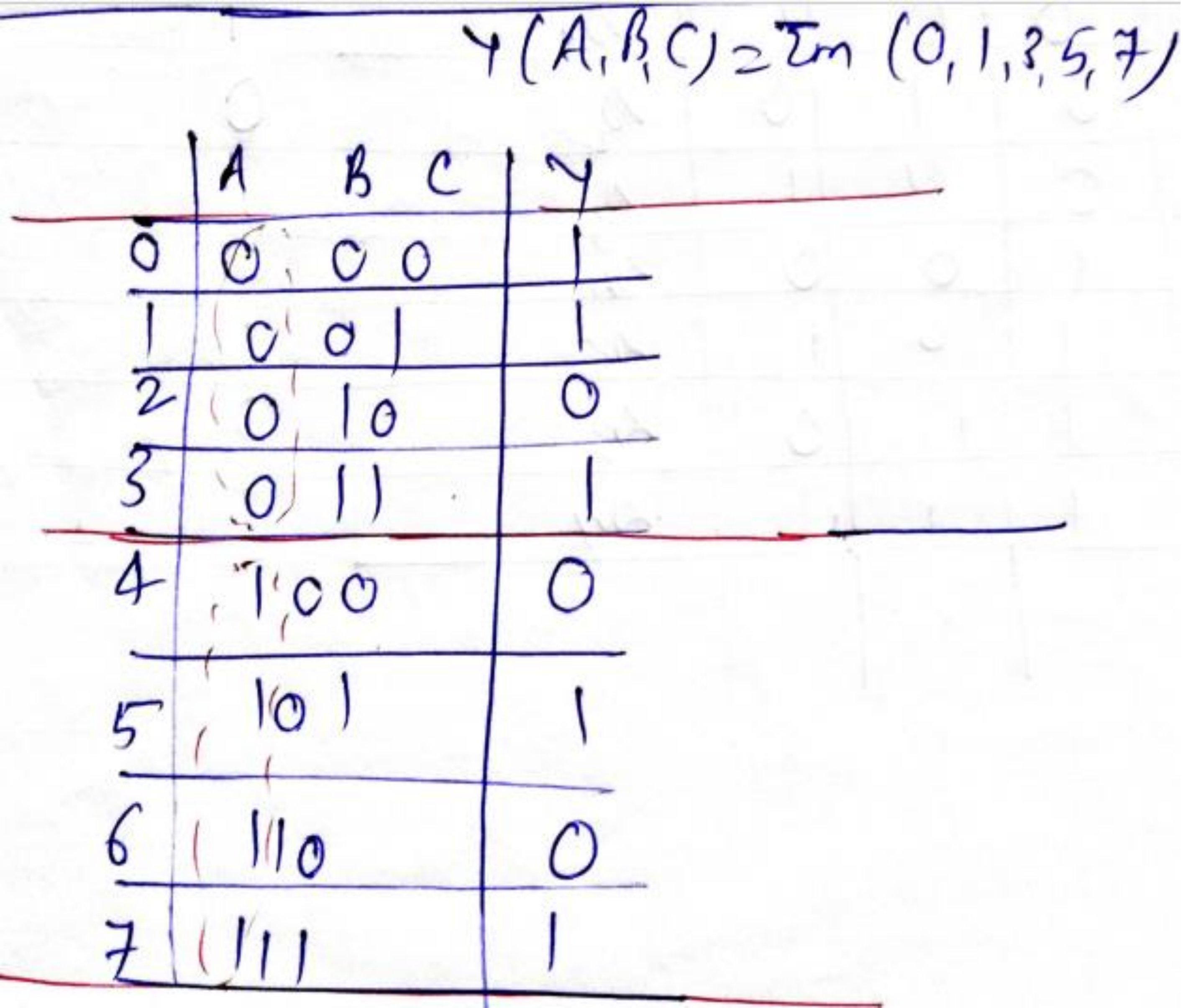
(1)

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

$Y(A,B,C) = \Sigma m(0,1,3,5,7)$

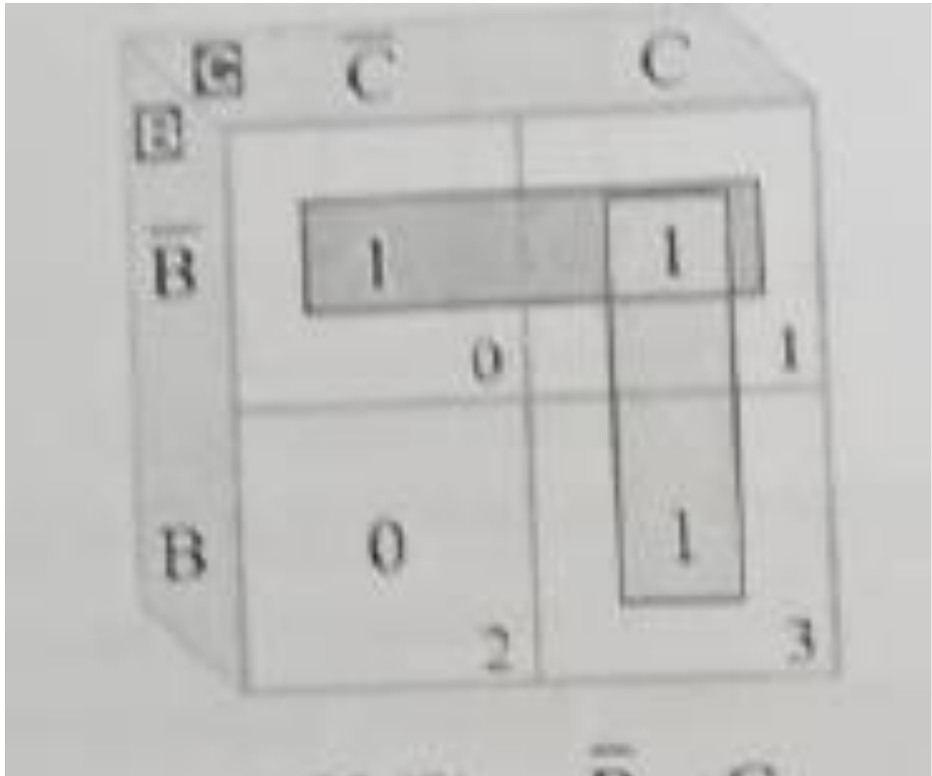


Using 2:1 Mux



$Y(A,B,C) = \Sigma m(0,1,3,5,7)$

	A B C	Y
0	0 0 0	1
1	0 0 1	1
2	0 1 0	0
3	0 1 1	1
4	1 0 0	0
5	1 0 1	1
6	1 1 0	0
7	1 1 1	1

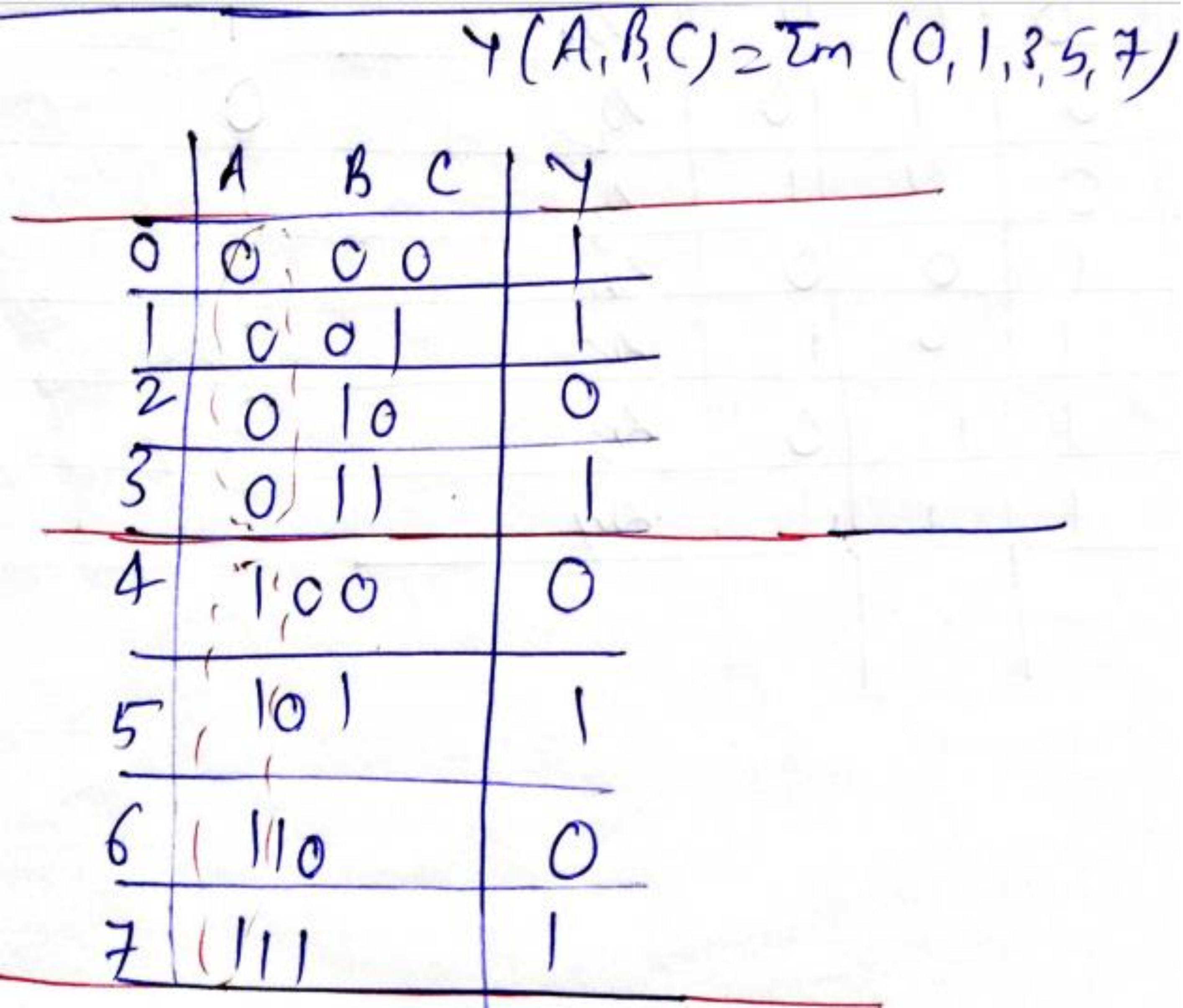


B \ C	$\overline{C}$	C
$\overline{B}$	1 0	1 1
B	0 2	1 3

$\therefore\ Y(0) = \overline{B} + C$

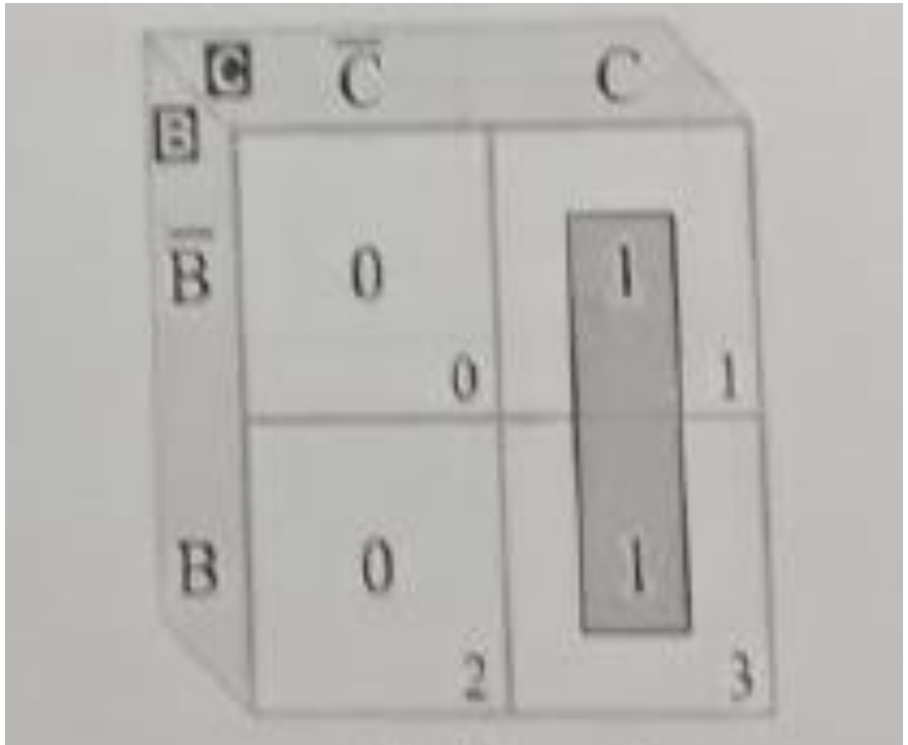
Fig. (*a*)

Using 2:1 Mux



$Y(A,B,C) = \Sigma m\,(0,1,3,5,7)$

	A B C	Y
0	0 0 0	1
1	0 0 1	1
2	0 1 0	0
3	0 1 1	1
4	1 0 0	0
5	1 0 1	1
6	1 1 0	0
7	1 1 1	1

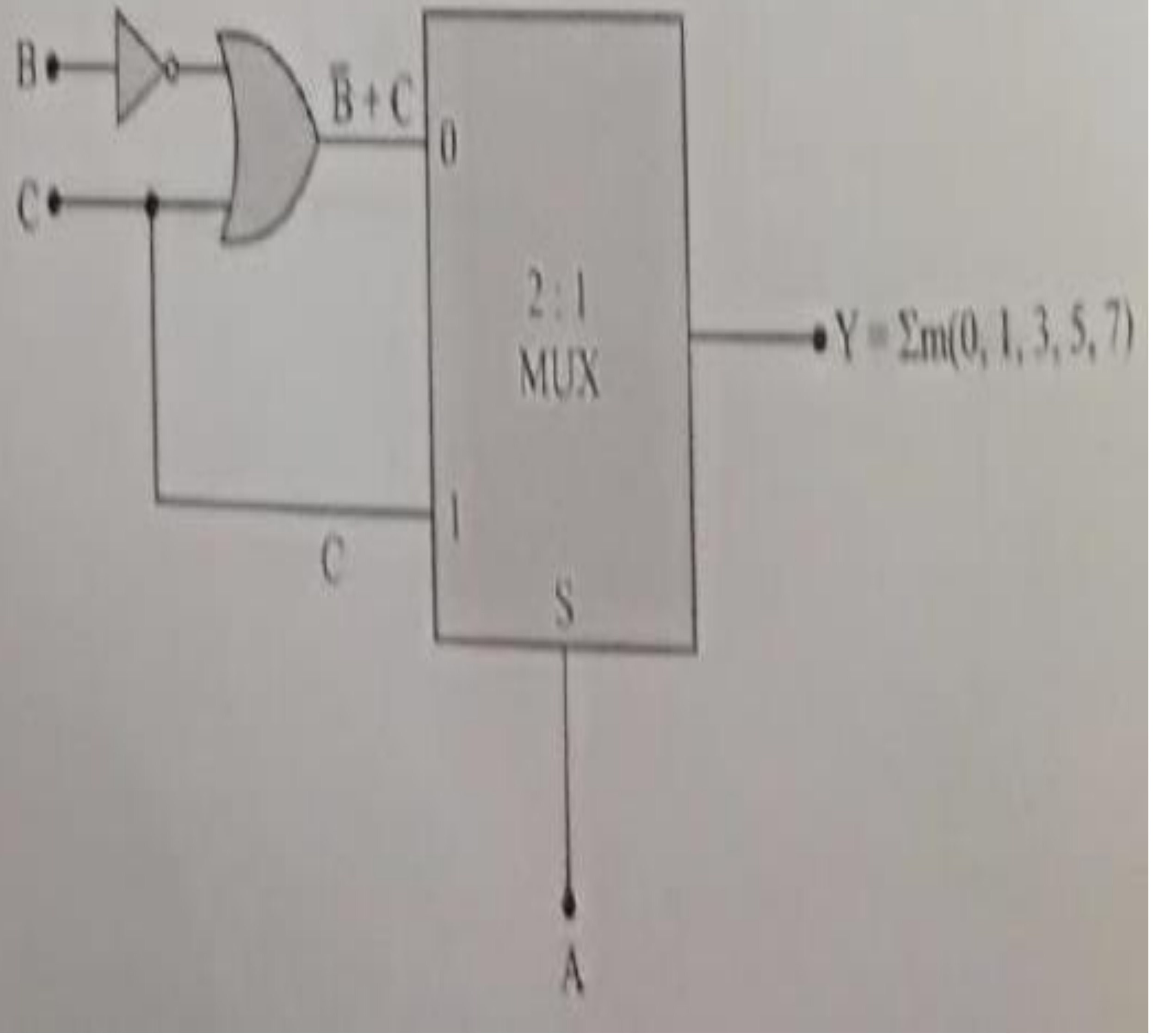


B \ C	$\overline{C}$	C
$\overline{B}$	0 (0)	1 (1)
B	0 (2)	1 (3)

$\therefore Y(1) = C$

Fig. (*b*)

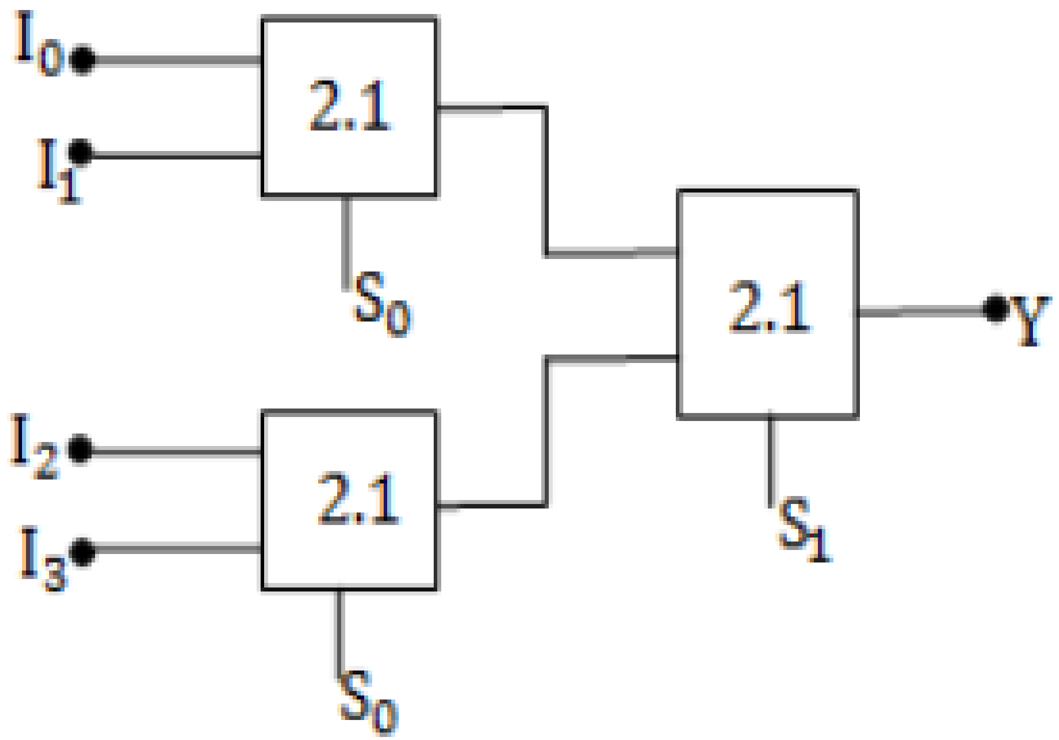
Implementation using 2 : 1 MUX.



Multiplexer Design

Implementation of Higher Order MUX using Lower Order MUX:

4:1 MUX by 2:1 MUX



S1	S0	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

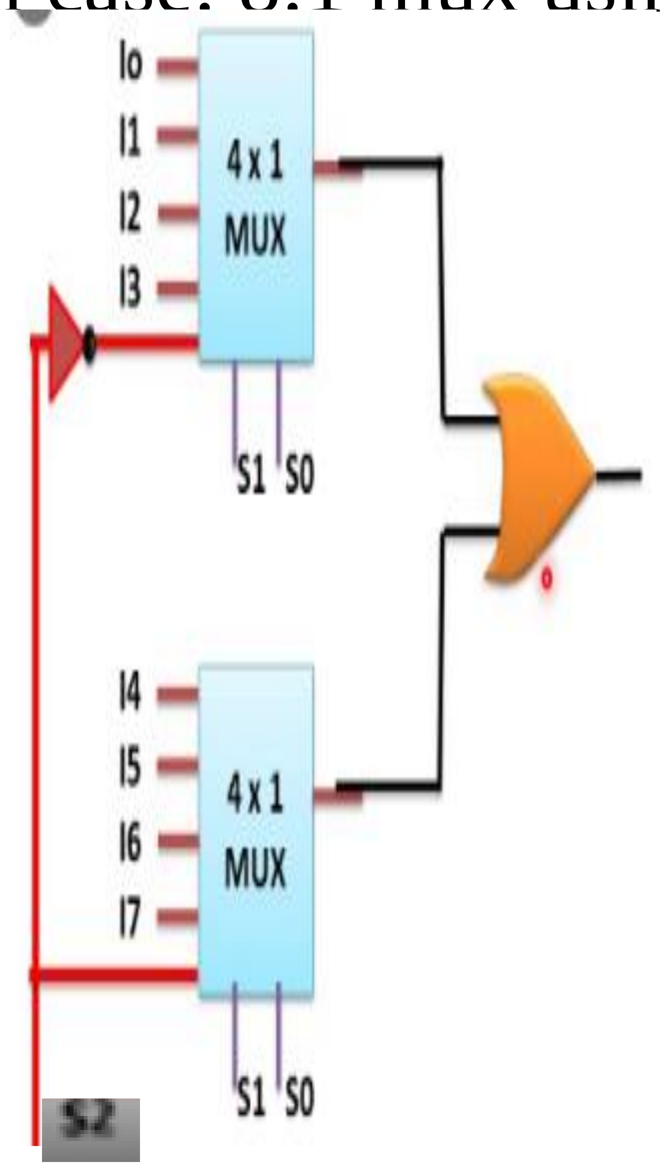
n = 4

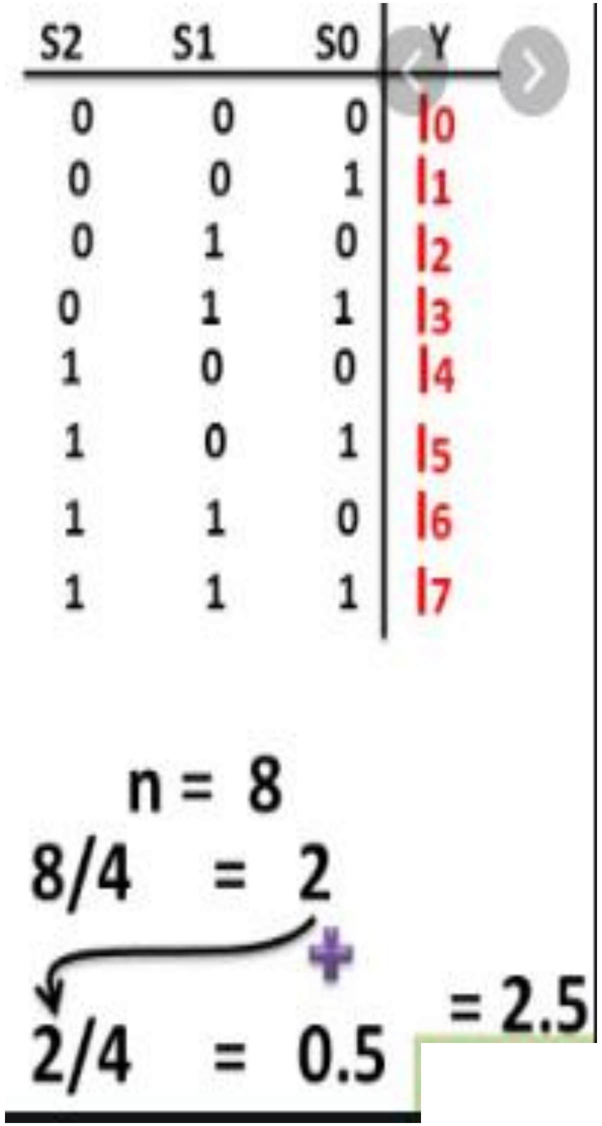
4/2 = 2

+ = 3

2/2 = 1

Special case: 8:1 mux using 4:1 mux





S2	S1	S0	Y
0	0	0	I0
0	0	1	I1
0	1	0	I2
0	1	1	I3
1	0	0	I4
1	0	1	I5
1	1	0	I6
1	1	1	I7

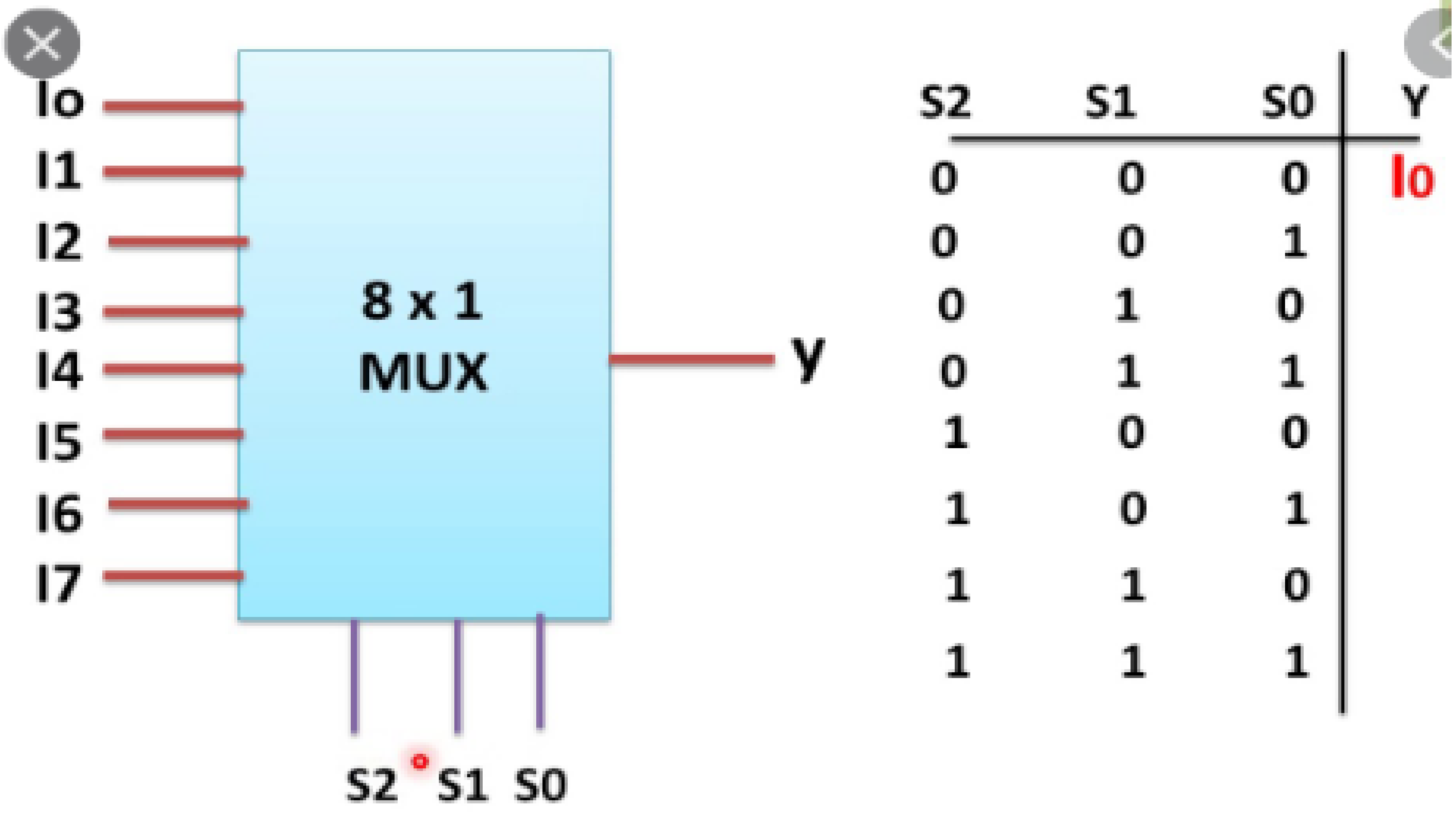
n = 8

8/4 = 2

2/4 = 0.5

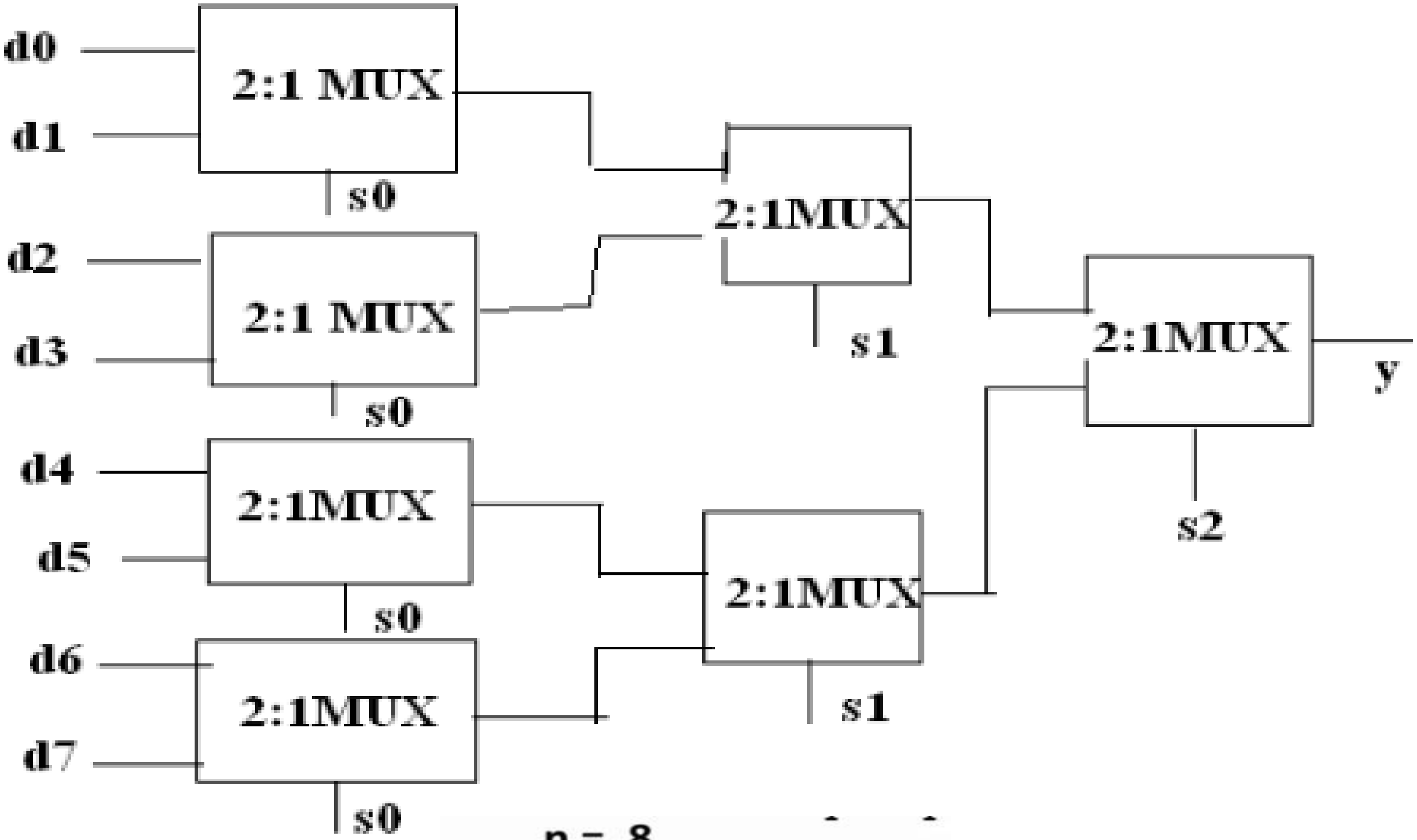
= 2.5

8:1 mux

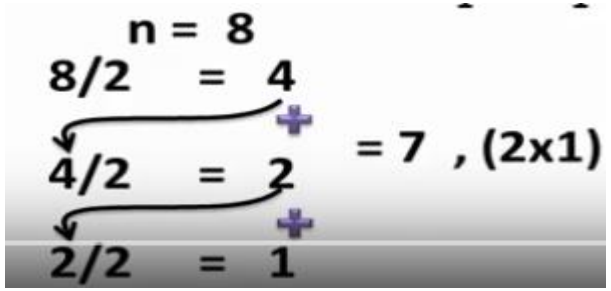


S2	S1	S0	Y
0	0	0	Io
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

8:1 mux using 2:1 mux

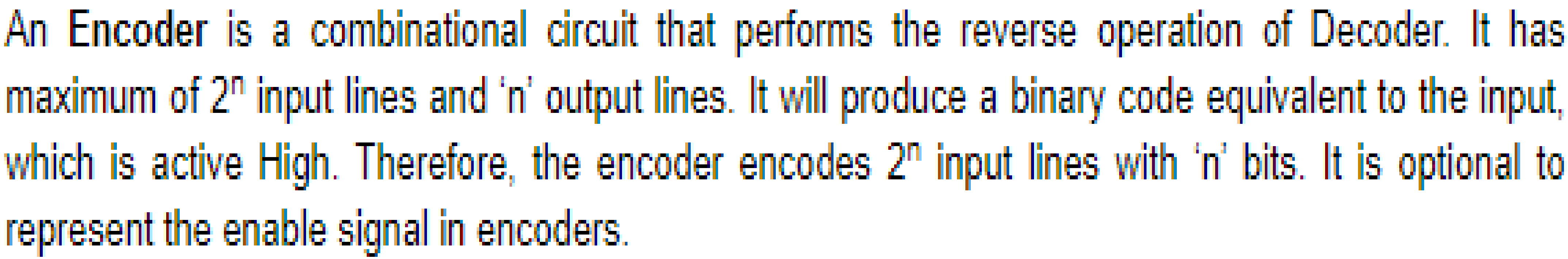


S2	S1	S0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1



$$\begin{aligned} n &= 8 \\ 8/2 &= 4 \; + \\ 4/2 &= 2 \; + \quad = 7\ , (2x1) \\ 2/2 &= 1 \end{aligned}$$

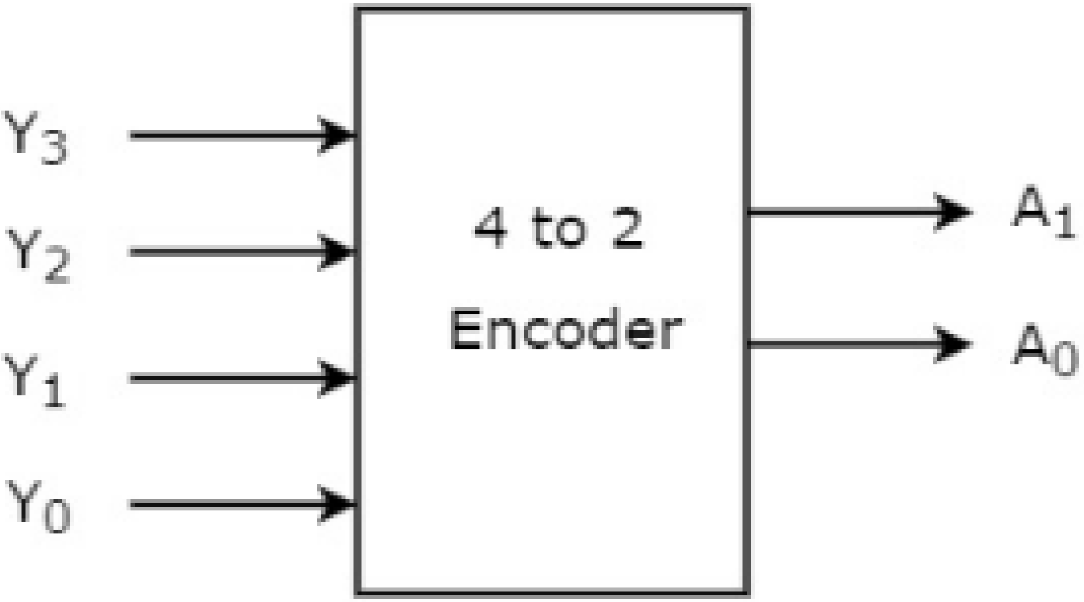
Encoders



An Encoder is a combinational circuit that performs the reverse operation of Decoder. It has maximum of 2^n input lines and 'n' output lines. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes 2^n input lines with 'n' bits. It is optional to represent the enable signal in encoders.

4 to 2 Encoder

Let 4 to 2 Encoder has four inputs Y_3, Y_2, Y_1 & Y_0 and two outputs A_1 & A_0. The **block diagram** of 4 to 2 Encoder is shown in the following figure.



At any time, only one of these 4 inputs can be ‘1’ in order to get the respective binary code at the output. The **Truth table** of 4 to 2 encoder is shown below.

At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output. The **Truth table** of 4 to 2 encoder is shown below.

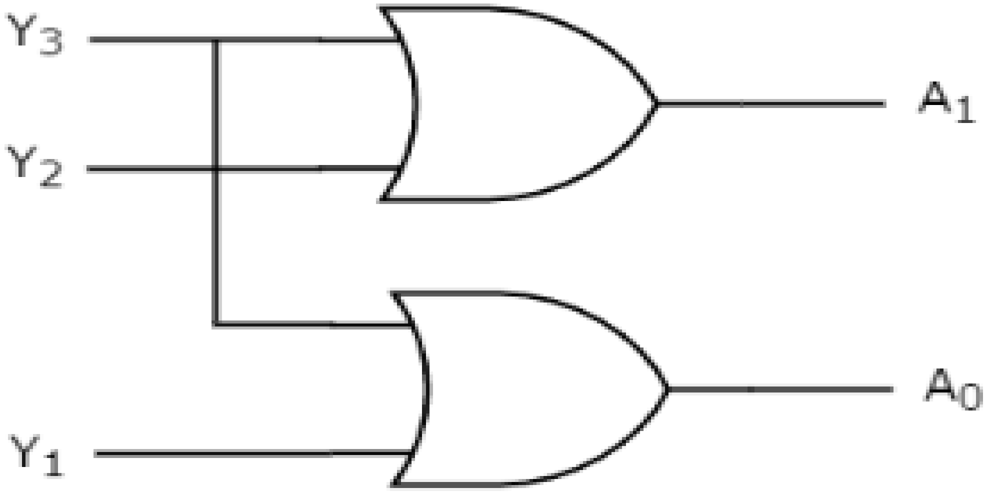
Inputs				Outputs	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

From Truth table, we can write the **Boolean functions** for each output as

$$A_1 = Y_3 + Y_2$$

$$A_0 = Y_3 + Y_1$$

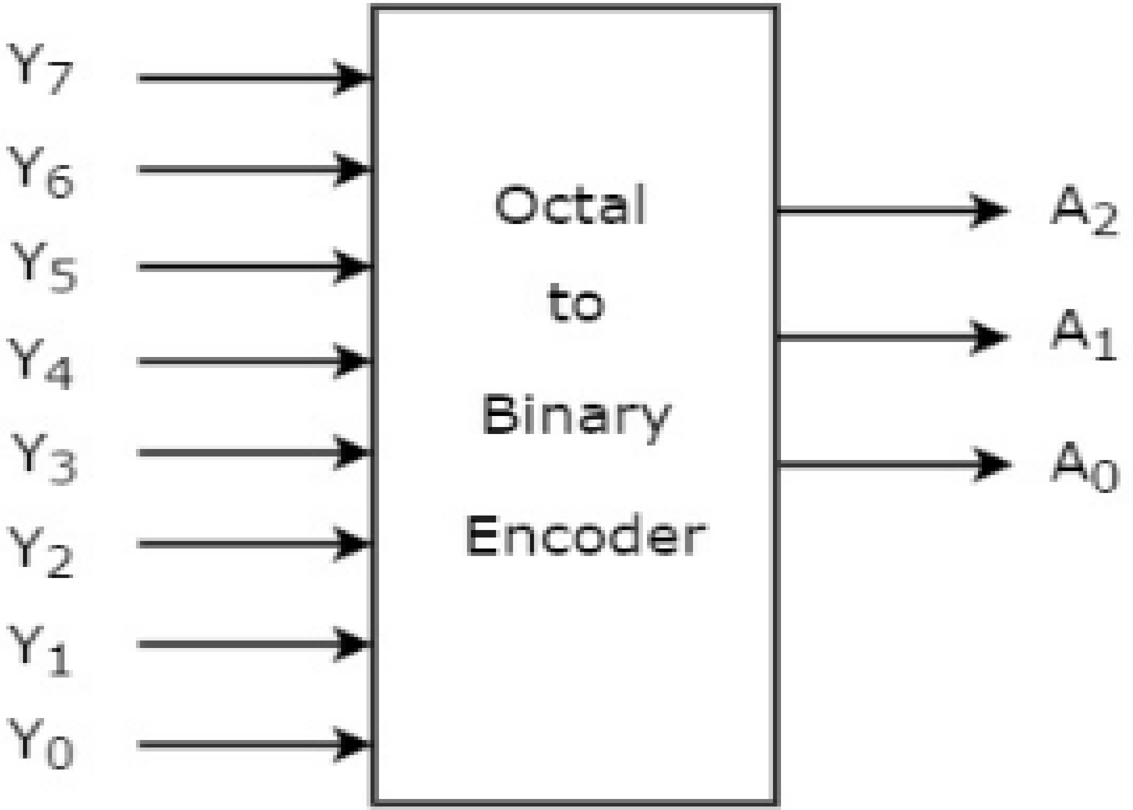
We can implement the above two Boolean functions by using two input OR gates. The **circuit diagram** of 4 to 2 encoder is shown in the following figure.



The above circuit diagram contains two OR gates. These OR gates encode the four inputs with two bits

Octal to Binary Encoder

Octal to binary Encoder has eight inputs, Y_7 to Y_0 and three outputs A_2, A_1 & A_0. Octal to binary encoder is nothing but 8 to 3 encoder. The **block diagram** of octal to binary Encoder is shown in the following figure.



At any time, only one of these eight inputs can be '1' in order to get the respective binary code. The **Truth table** of octal to binary encoder is shown below.

At any time, only one of these eight inputs can be '1' in order to get the respective binary code. The **Truth table** of octal to binary encoder is shown below.

Inputs								Outputs		
Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

From Truth table, we can write the **Boolean functions** for each output as

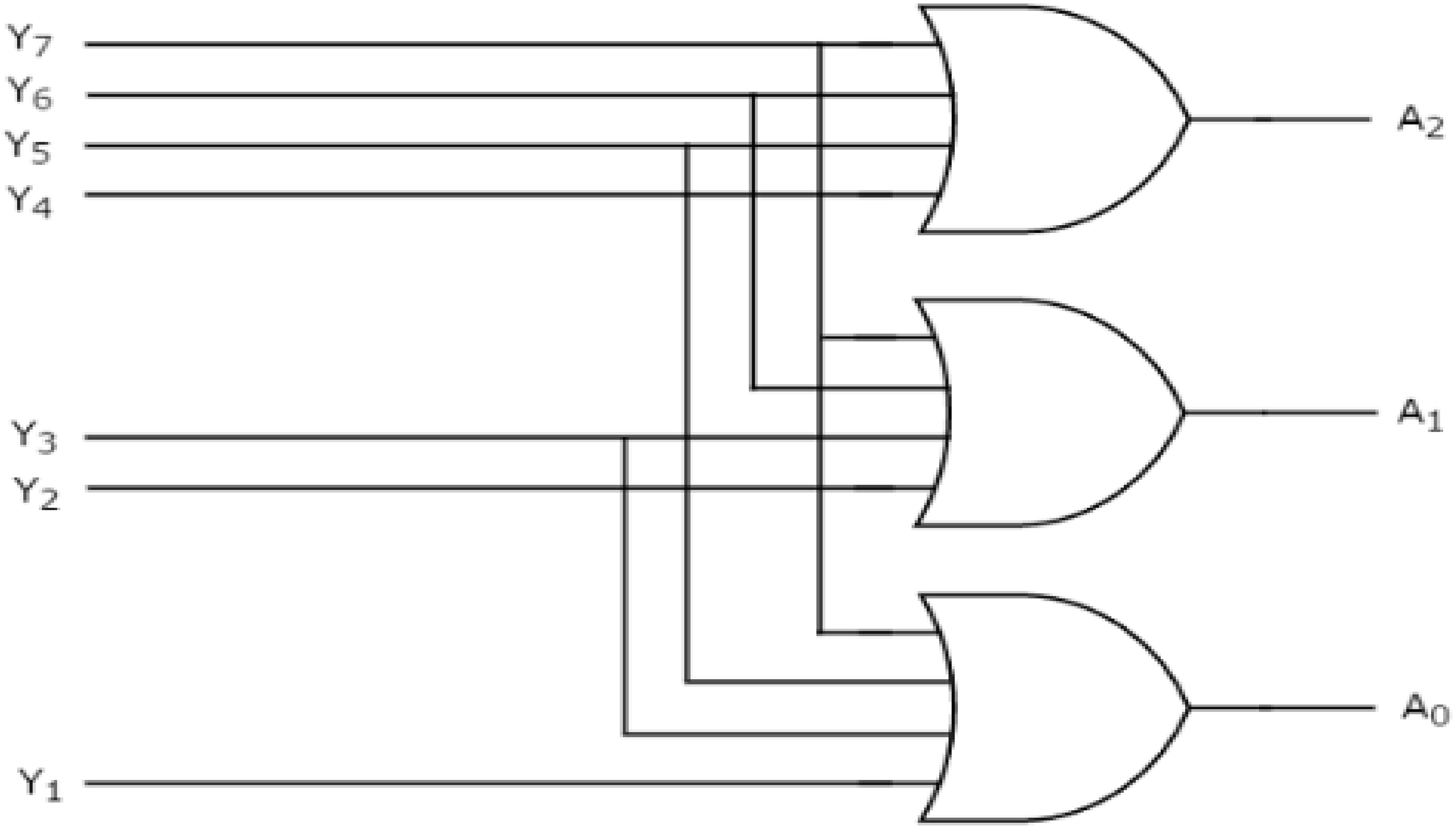
$$A_2 = Y_7 + Y_6 + Y_5 + Y_4$$

$$A_1 = Y_7 + Y_6 + Y_3 + Y_2$$

$$A_0 = Y_7 + Y_5 + Y_3 + Y_1$$

We can implement the above Boolean functions by using four input OR gates. The **circuit diagram** of octal to binary encoder is shown in the following figure.

We can implement the above Boolean functions by using four input OR gates. The **circuit diagram** of octal to binary encoder is shown in the following figure.

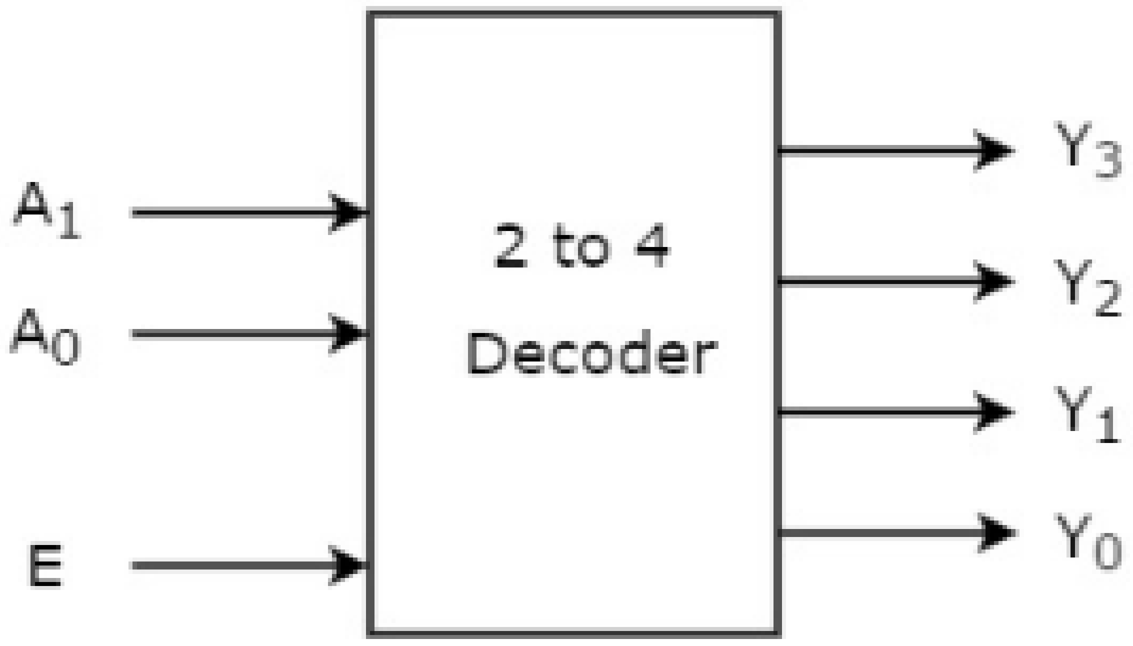


The above circuit diagram contains three 4-input OR gates. These OR gates encode the eight inputs with three bits.

Decoder is a combinational circuit that has 'n' input lines and maximum of 2^n output lines. One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code. The outputs of the decoder are nothing but the min terms of 'n' input variables $lines$, when it is enabled.

2 to 4 Decoder

Let 2 to 4 Decoder has two inputs A_1 & A_0 and four outputs Y_3, Y_2, Y_1 & Y_0. The **block diagram** of 2 to 4 decoder is shown in the following figure.



One of these four outputs will be '1' for each combination of inputs when enable, E is '1'. The **Truth table** of 2 to 4 decoder is shown below.

One of these four outputs will be '1' for each combination of inputs when enable, E is '1'. The **Truth table** of 2 to 4 decoder is shown below.

Enable	Inputs		Outputs			
E	A_1	A_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

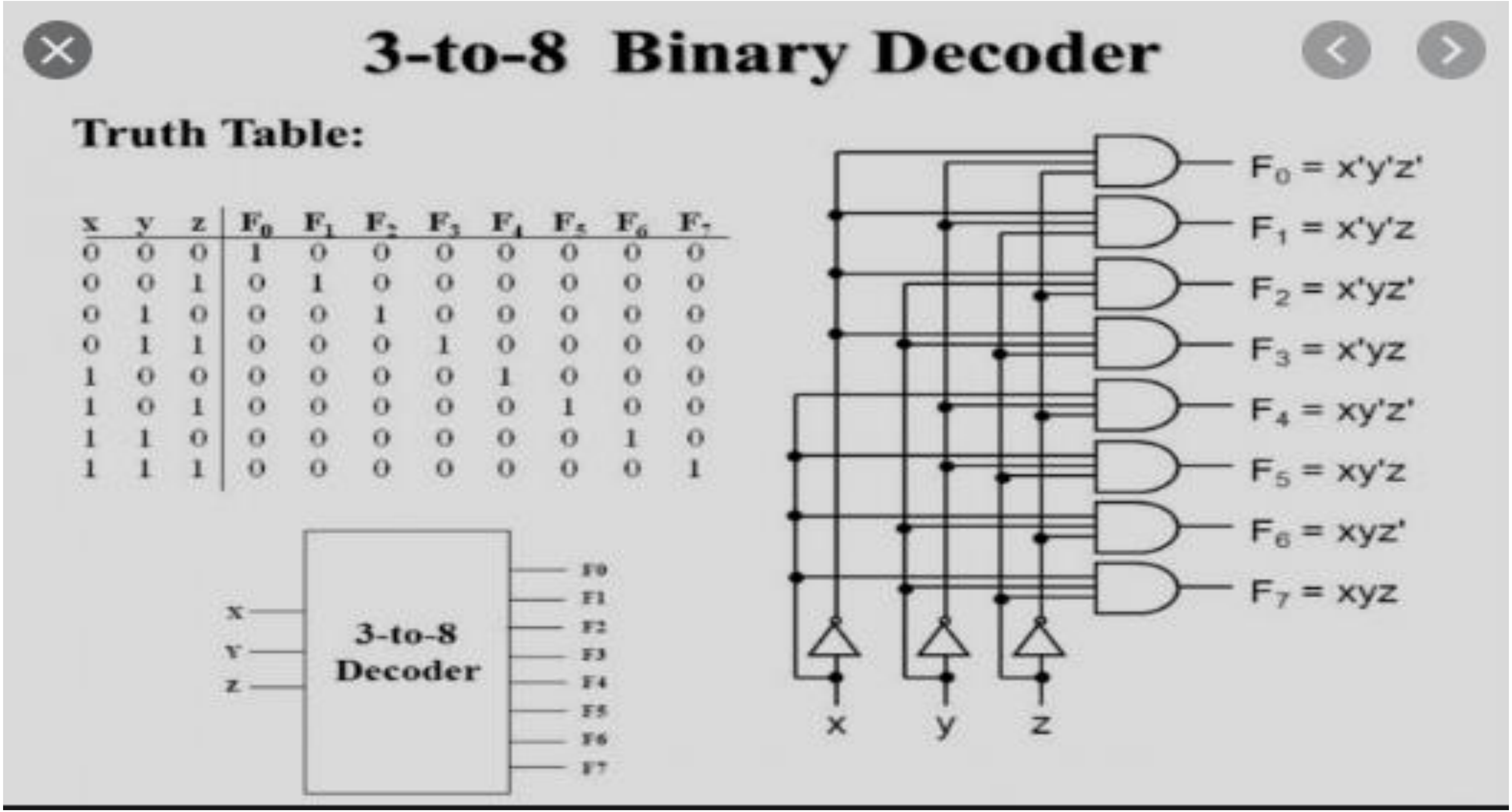
From Truth table, we can write the Boolean functions for each output as

$$Y_3 = E.A_1.A_0$$

$$Y_2 = E.A_1.A_0'$$

$$Y_1 = E.A_1'.A_0$$

$$Y_0 = E.A_1'.A_0'$$



3-to-8 Binary Decoder

Truth Table:

x	y	z	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_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

Implementation of Higher-order Decoders

Now, let us implement the following two higher-order decoders using lower-order decoders.

- 3 to 8 decoder

3 to 8 Decoder

In this section, let us implement **3 to 8 decoder using 2 to 4 decoders**. We know that 2 to 4 Decoder has two inputs, A_1 & A_0 and four outputs, Y_3 to Y_0. Whereas, 3 to 8 Decoder has three inputs A_2, A_1 & A_0 and eight outputs, Y_7 to Y_0.

We can find the number of lower order decoders required for implementing higher order decoder using the following formula.

$$Required\, number\, of\, lower\, order\, decoders = \frac{m_2}{m_1}$$

Where,

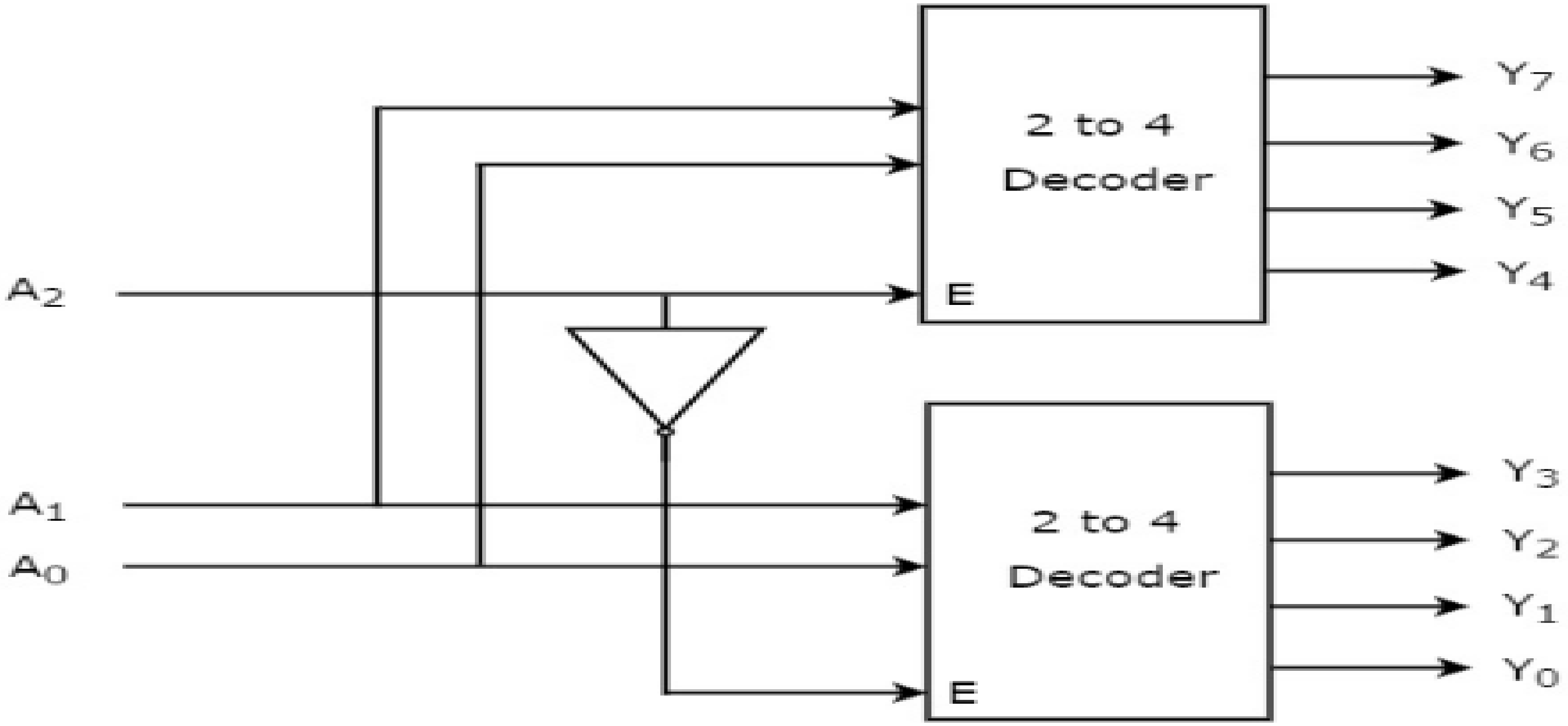
m_1 is the number of outputs of lower order decoder.

m_2 is the number of outputs of higher order decoder.

Here, m_1 = 4 and m_2 = 8. Substitute, these two values in the above formula.

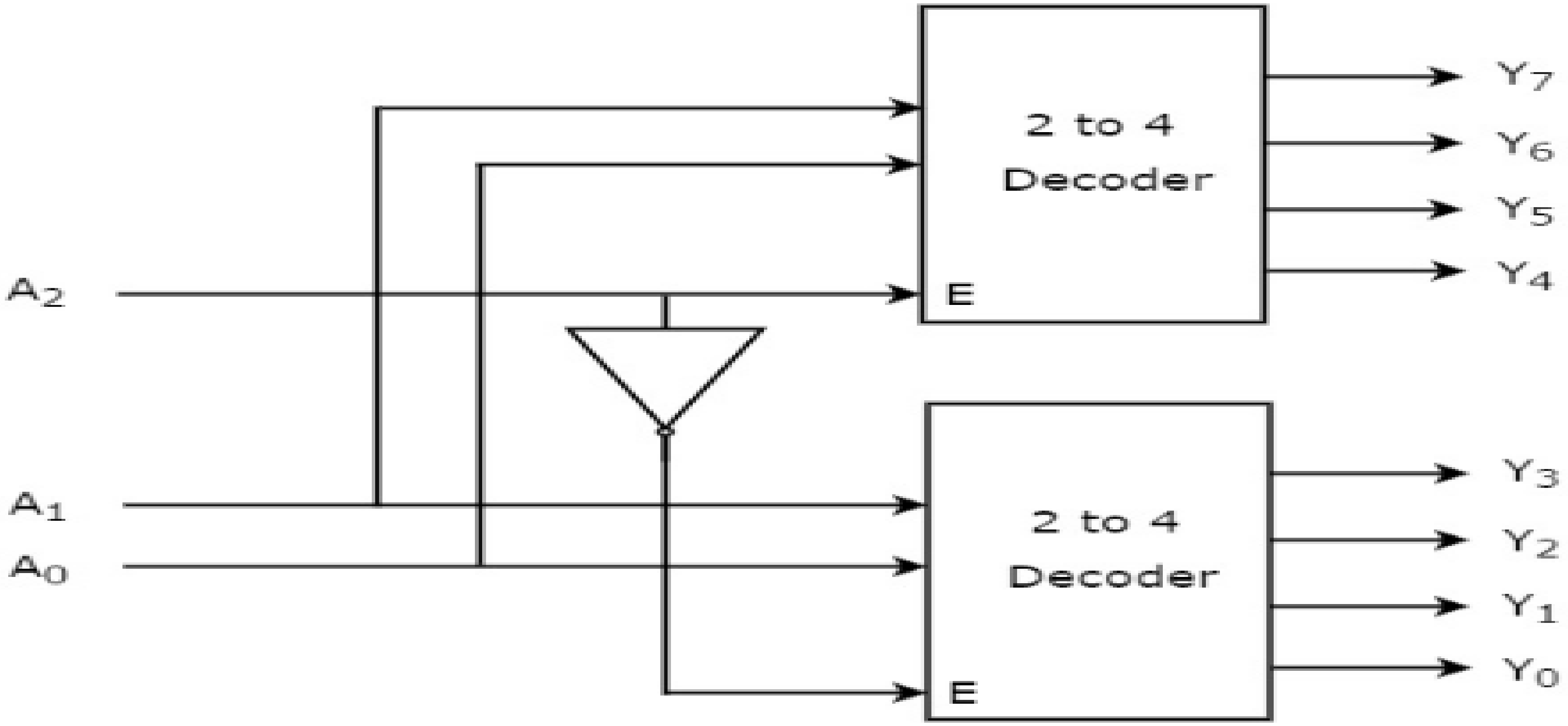
$$Required\ number\ of\ 2\ to\ 4\ decoders = \frac{8}{4} = 2$$

Therefore, we require two 2 to 4 decoders for implementing one 3 to 8 decoder. The **block diagram** of 3 to 8 decoder using 2 to 4 decoders is shown in the following figure.



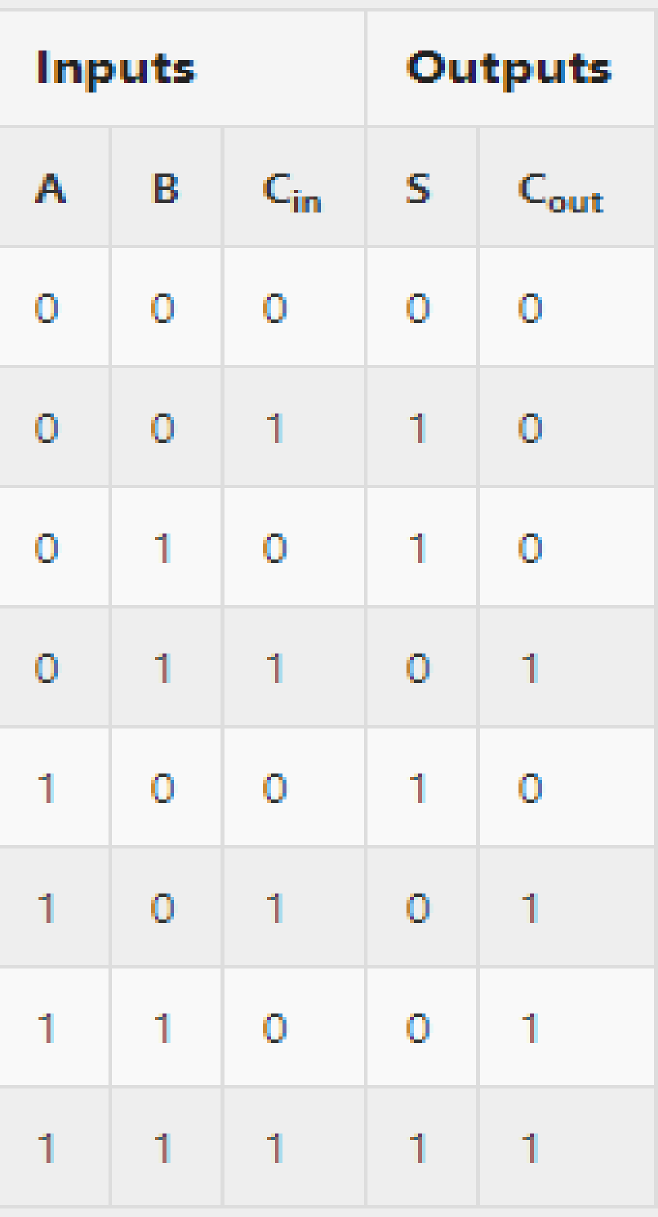
The parallel inputs A_1 & A_0 are applied to each 2 to 4 decoder. The complement of input A_2 is connected to Enable, E of lower 2 to 4 decoder in order to get the outputs, Y_3 to Y_0. These are the **lower four min terms**. The input, A_2 is directly connected to Enable, E of upper 2 to 4 decoder in order to get the outputs, Y_7 to Y_4. These are the **higher four min terms**.

Therefore, we require two 2 to 4 decoders for implementing one 3 to 8 decoder. The **block diagram** of 3 to 8 decoder using 2 to 4 decoders is shown in the following figure.



The parallel inputs A_1 & A_0 are applied to each 2 to 4 decoder. The complement of input A_2 is connected to Enable, E of lower 2 to 4 decoder in order to get the outputs, Y_3 to Y_0. These are the **lower four min terms**. The input, A_2 is directly connected to Enable, E of upper 2 to 4 decoder in order to get the outputs, Y_7 to Y_4. These are the **higher four min terms**.

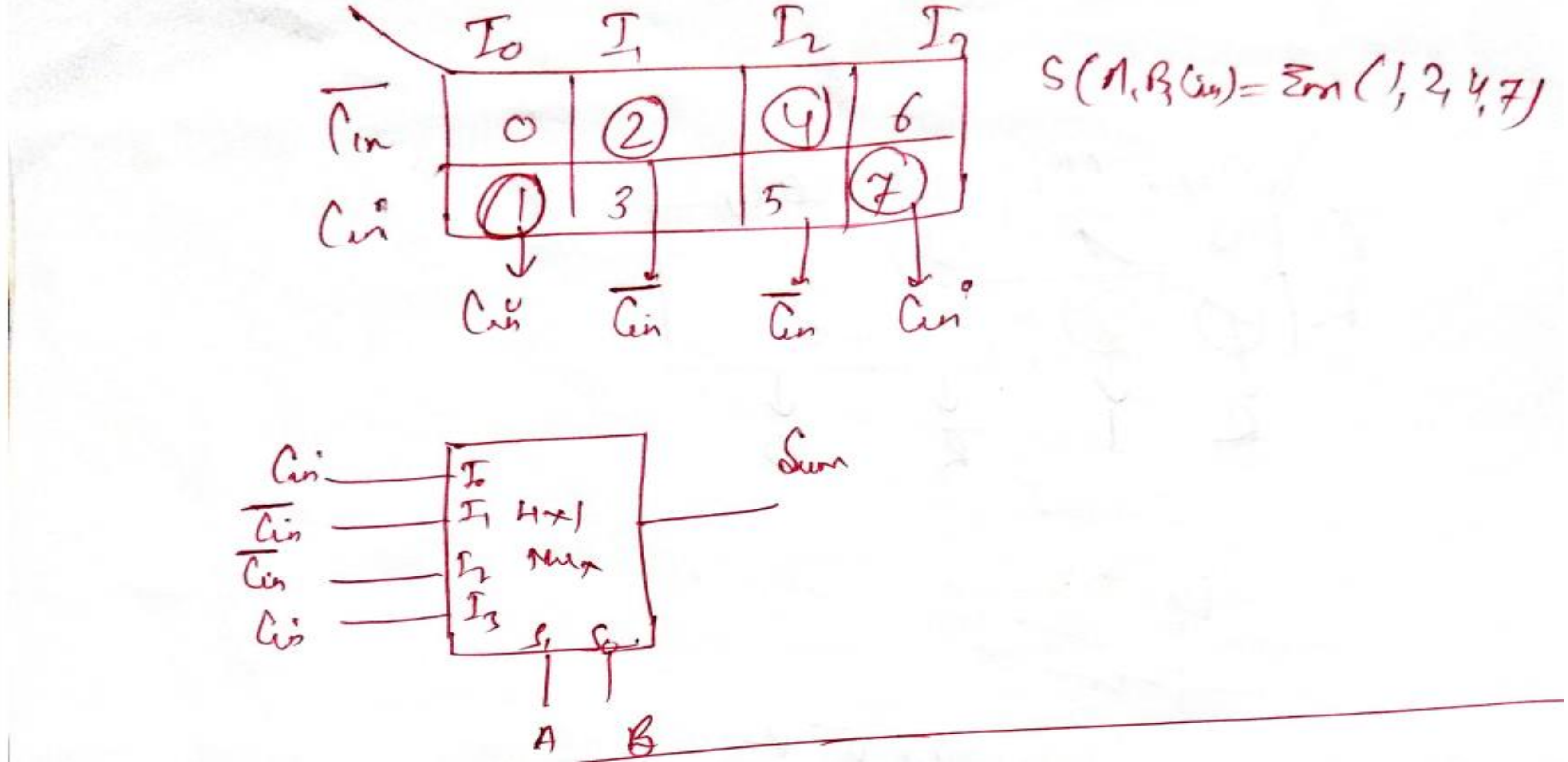
Full Adder using Multiplexer



Inputs			Outputs	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

S(A,B,Cin)= ∑(1,2,4,7)

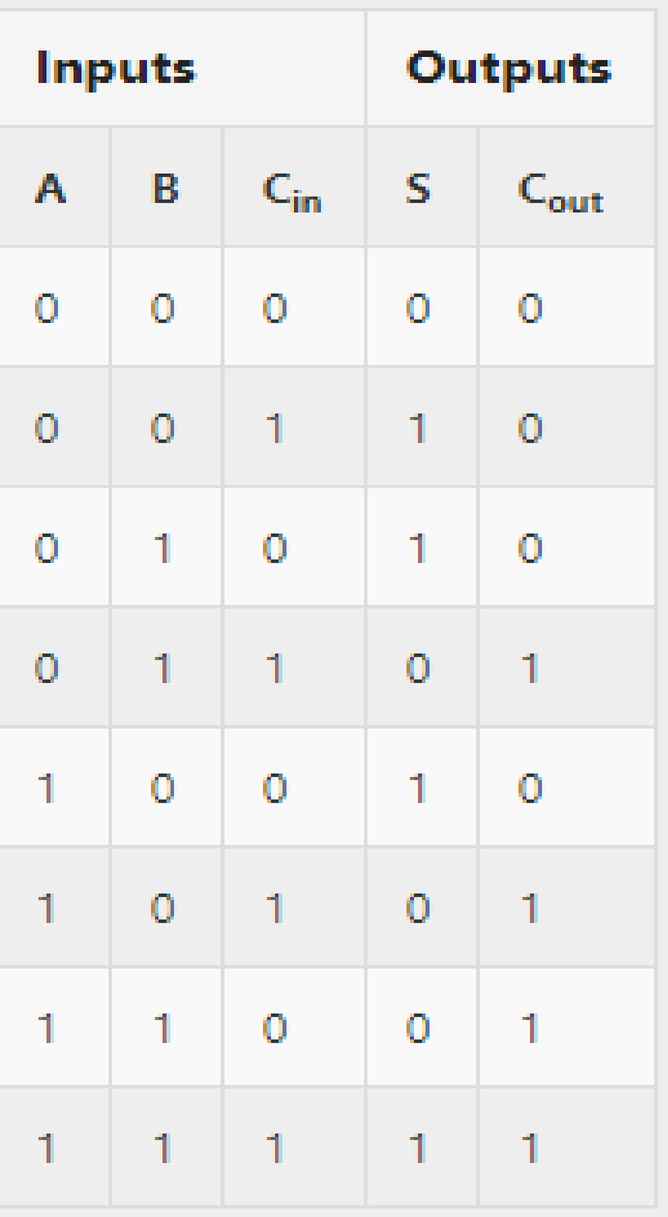
Cout(A,B,Cin)= ∑(3,5,6,7)



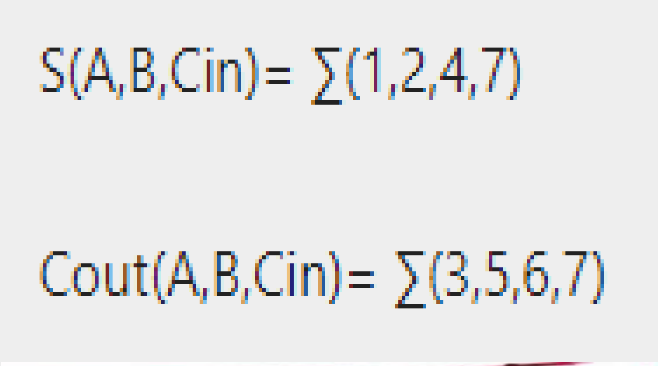
	I_0	I_1	I_2	I_3
$\overline{C_{in}}$	0	2	4	6
C_{in}	1	3	5	7
	C_{in}	$\overline{C_{in}}$	$\overline{C_{in}}$	C_{in}

$S(A,B,C_{in}) = \Sigma m(1,2,4,7)$

Full Adder using Multiplexer

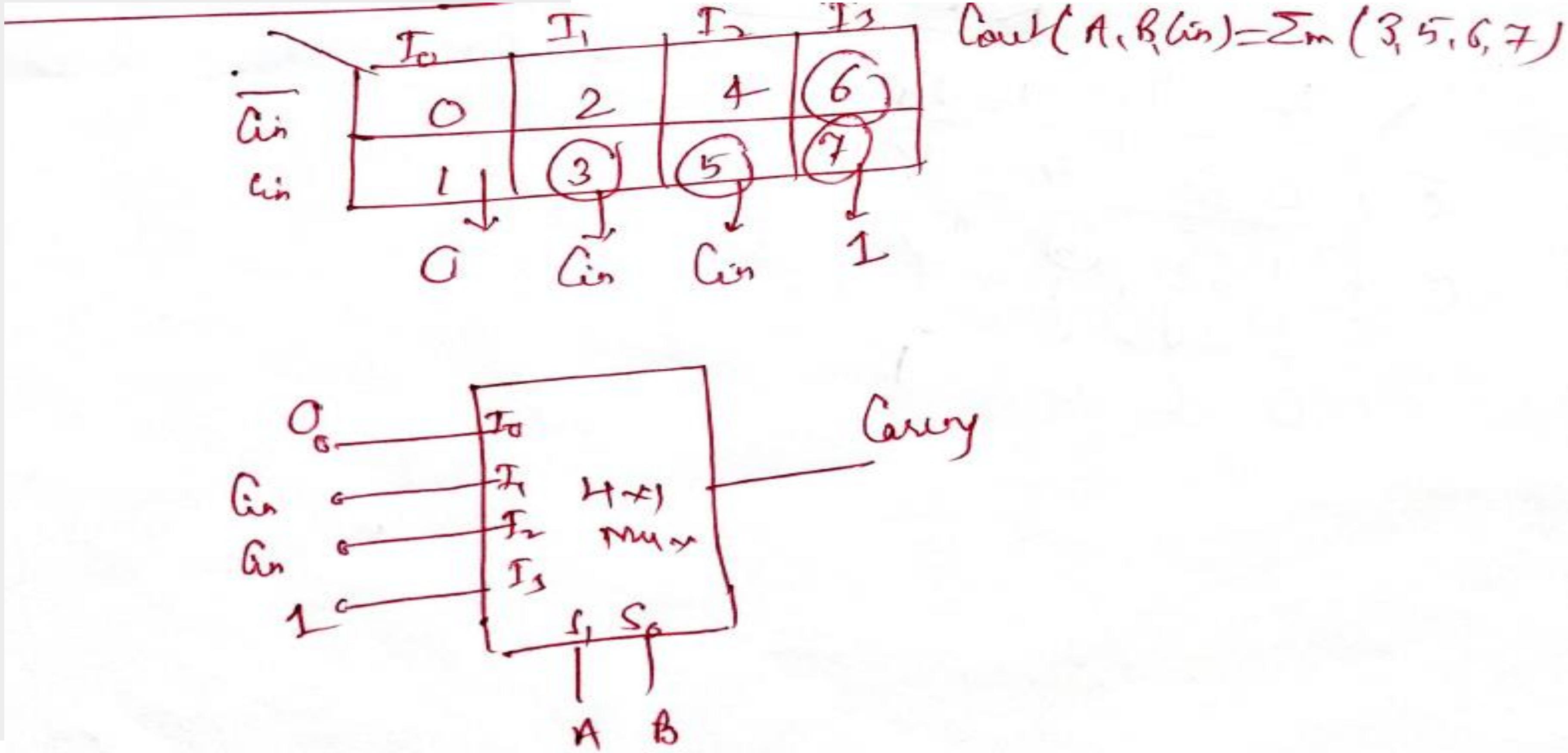


Inputs			Outputs	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



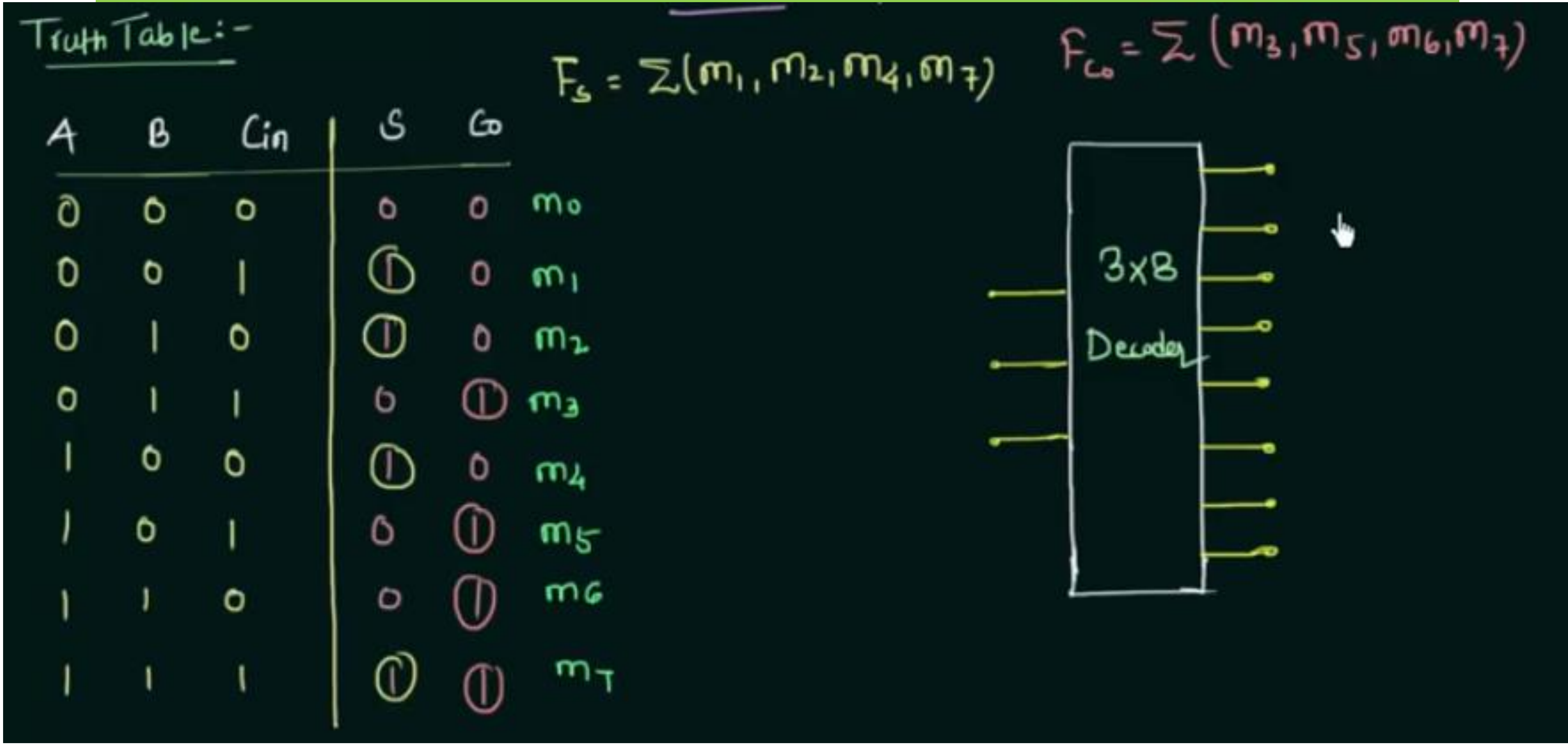
$S(A,B,Cin)= \sum(1,2,4,7)$

$Cout(A,B,Cin)= \sum(3,5,6,7)$



$Cout(A,B,Cin)=\sum m(3,5,6,7)$

Full adder using Decoder



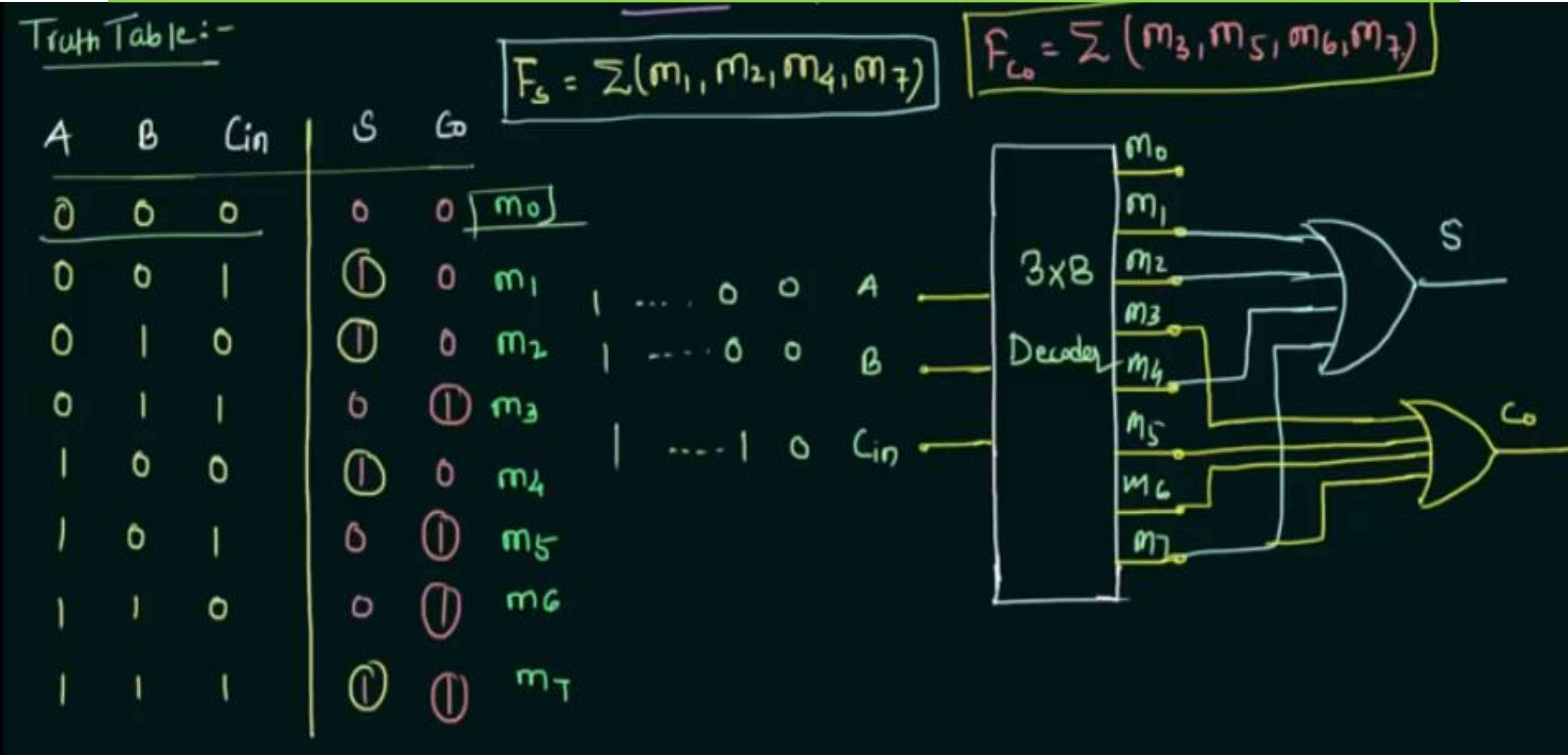
Truth Table:-

$F_s = \Sigma(m_1, m_2, m_4, m_7)$

$F_{co} = \Sigma(m_3, m_5, m_6, m_7)$

A	B	Cin	S	Co	
0	0	0	0	0	m_0
0	0	1	1	0	m_1
0	1	0	1	0	m_2
0	1	1	0	1	m_3
1	0	0	1	0	m_4
1	0	1	0	1	m_5
1	1	0	0	1	m_6
1	1	1	1	1	m_7

Full adder using Decoder



Truth Table:-

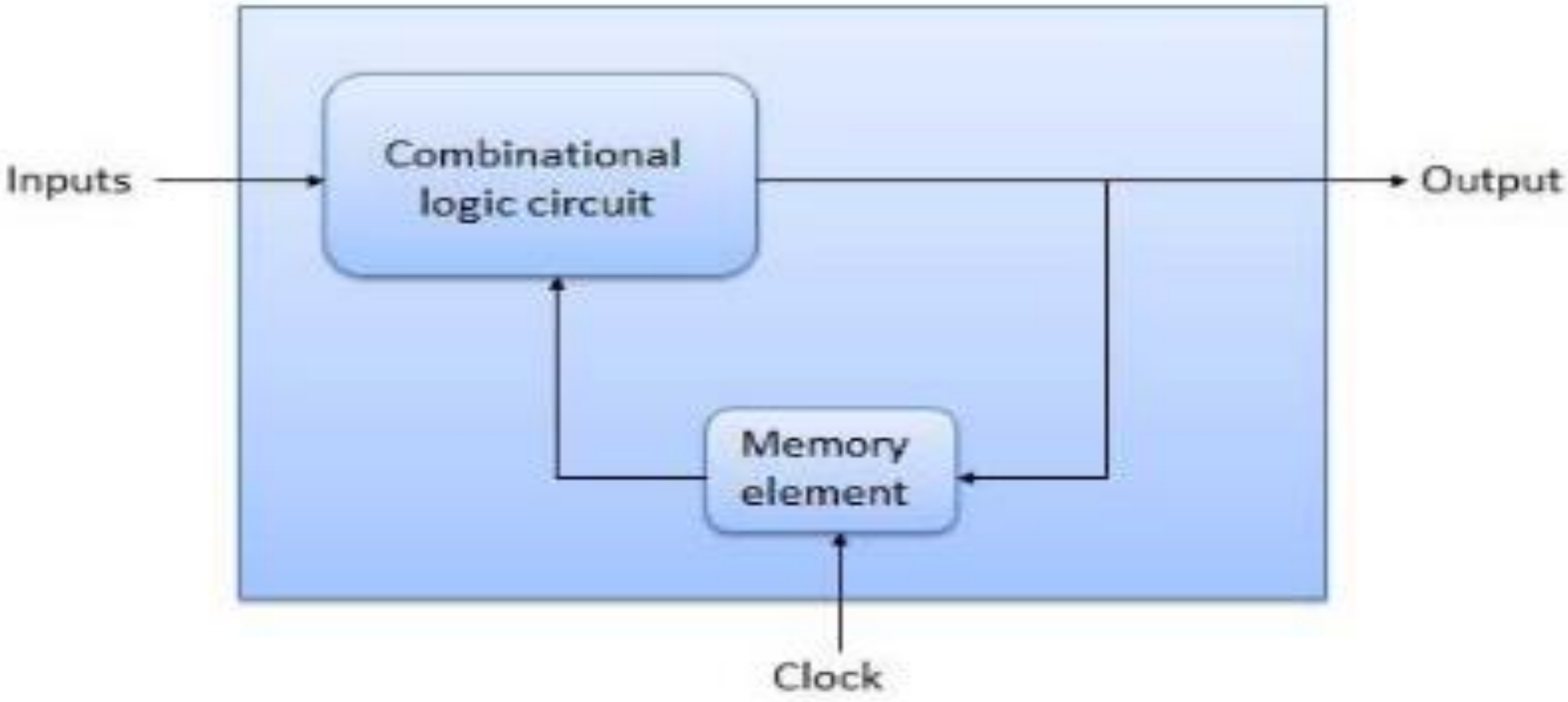
A	B	Cin	S	Co	
0	0	0	0	0	m_0
0	0	1	1	0	m_1
0	1	0	1	0	m_2
0	1	1	0	1	m_3
1	0	0	1	0	m_4
1	0	1	0	1	m_5
1	1	0	0	1	m_6
1	1	1	1	1	m_T

$F_s = \Sigma(m_1, m_2, m_4, m_7)$

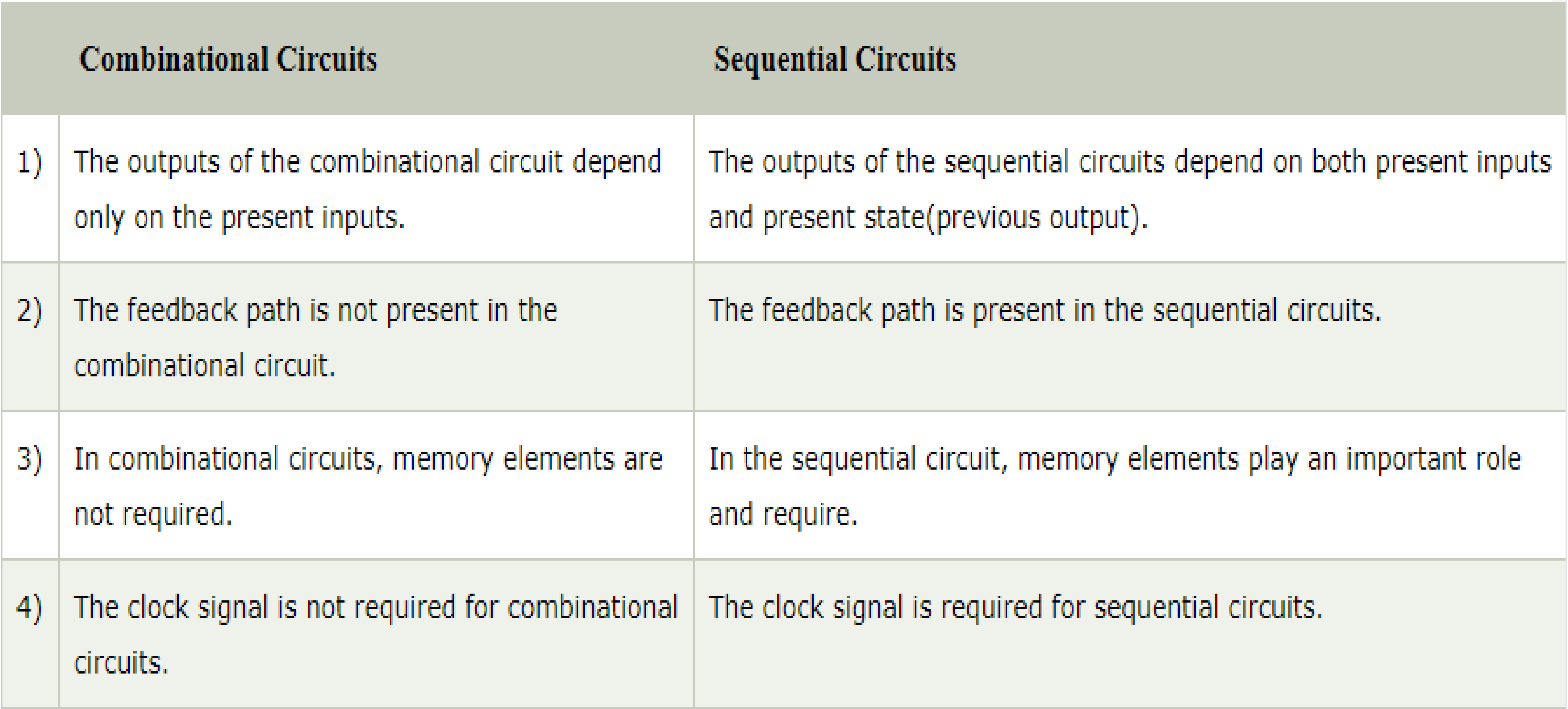
$F_{Co} = \Sigma(m_3, m_5, m_6, m_7)$

The combinational circuit does not use any memory. Hence the previous state of input does not have any effect on the present state of the circuit. But sequential circuit has memory so output can vary based on input. This type of circuits uses previous input, output, clock and a memory element.

Block diagram



Difference between the combinational circuits and sequential circuits are given below

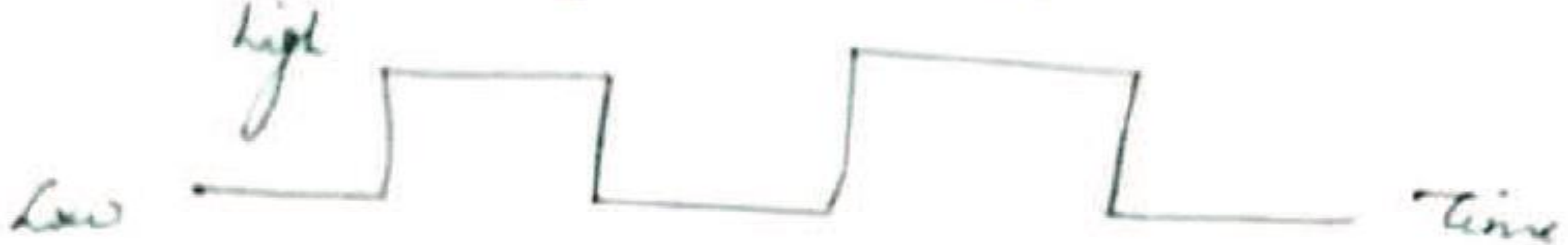


	Combinational Circuits	Sequential Circuits
1)	The outputs of the combinational circuit depend only on the present inputs.	The outputs of the sequential circuits depend on both present inputs and present state(previous output).
2)	The feedback path is not present in the combinational circuit.	The feedback path is present in the sequential circuits.
3)	In combinational circuits, memory elements are not required.	In the sequential circuit, memory elements play an important role and require.
4)	The clock signal is not required for combinational circuits.	The clock signal is required for sequential circuits.

Clock Signal

The clock signal is a timing sig. Every sequential ~~signal~~ ckt will have this timing signal applied.

[2] Clock is a rectangular signal & it repeats i
after every T seconds.



Triggering :-

Types of Triggering

- Level triggering
- Edge Triggering
  - positive edge trigger
  - negative edge trigger

Types of Triggering

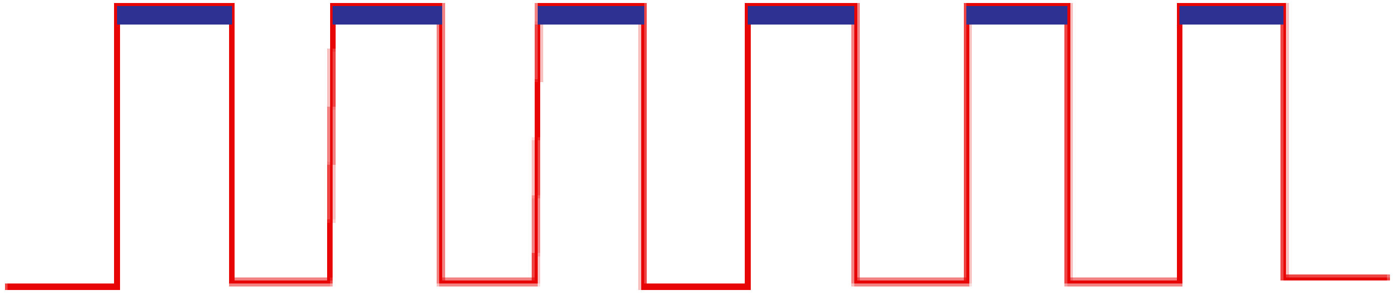
These are two types of triggering in sequential circuits:

Level triggering

The logic High and logic Low are the two levels in the clock signal. In level triggering, when the clock pulse is at a particular level, only then the circuit is activated. There are the following types of level triggering:

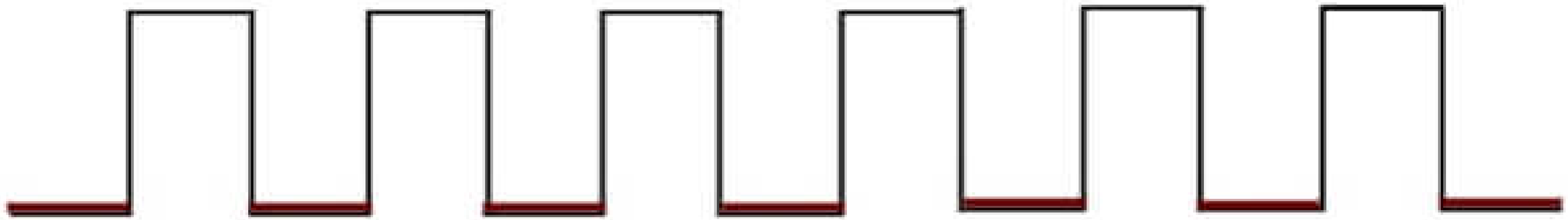
Positive level triggering

In a positive level triggering, the signal with Logic High occurs. So, in this triggering, the circuit is operated with such type of clock signal. Below is the diagram of positive level triggering:



Negative level triggering

In negative level triggering, the signal with Logic Low occurs. So, in this triggering, the circuit is operated with such type of clock signal. Below is the diagram of Negative level triggering:



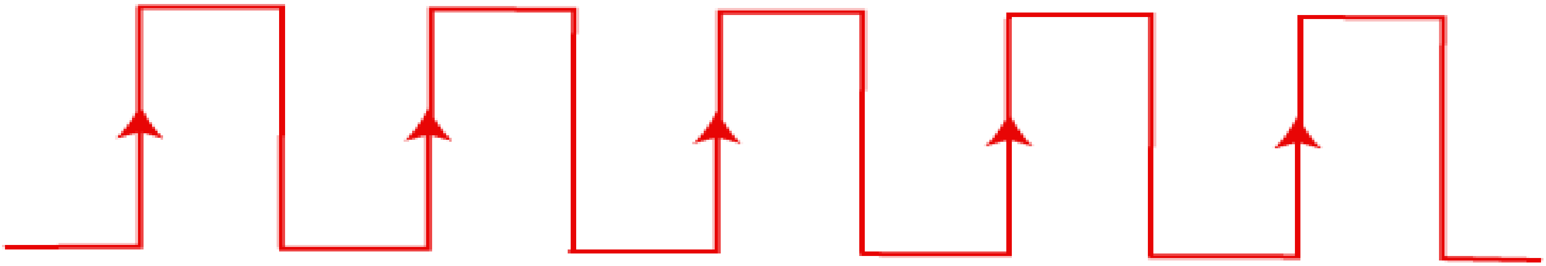
Edge triggering

In clock signal of edge triggering, two types of transitions occur, i.e., transition either from Logic Low to Logic High or Logic High to Logic Low.

Based on the transitions of the clock signal, there are the following types of edge triggering:

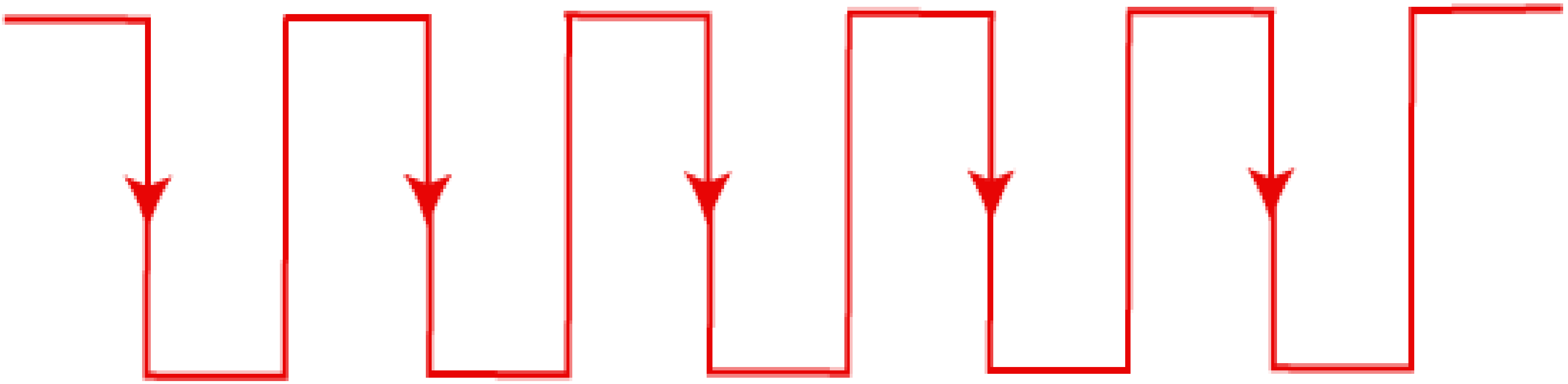
Positive edge triggering

The transition from Logic Low to Logic High occurs in the clock signal of positive edge triggering. So, in positive edge triggering, the circuit is operated with such type of clock signal. The diagram of positive edge triggering is given below.

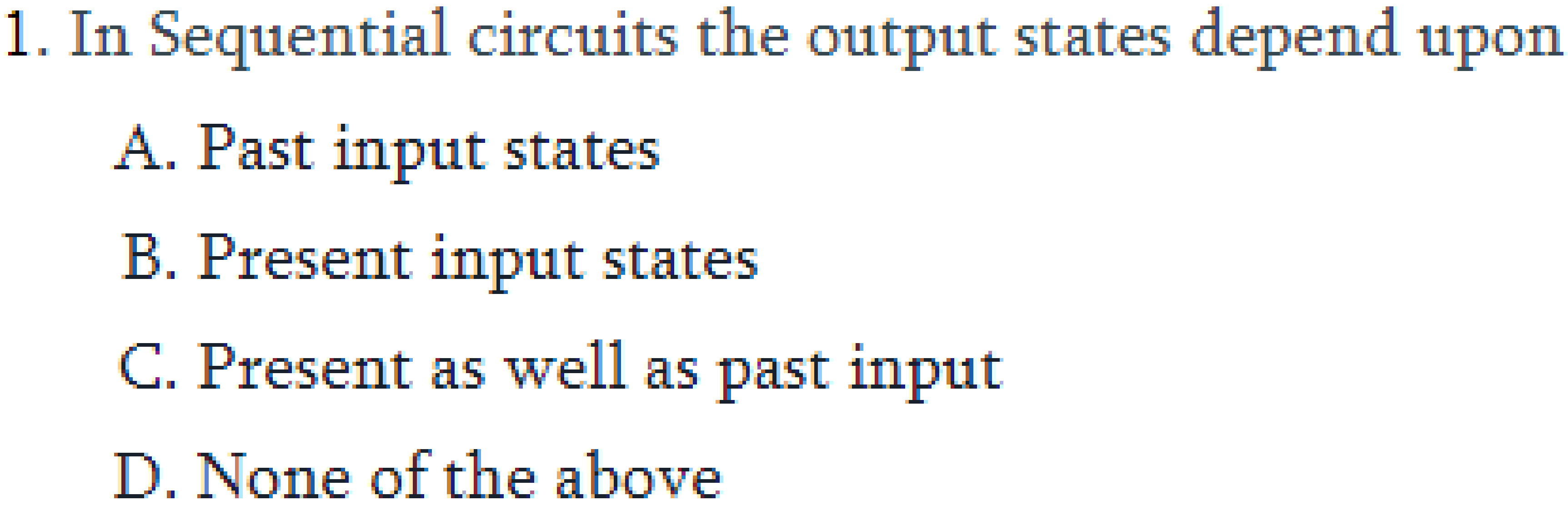


Negative edge triggering

The transition from Logic High to Logic low occurs in the clock signal of negative edge triggering. So, in negative edge triggering, the circuit is operated with such type of clock signal. The diagram of negative edge triggering is given below.



Poll



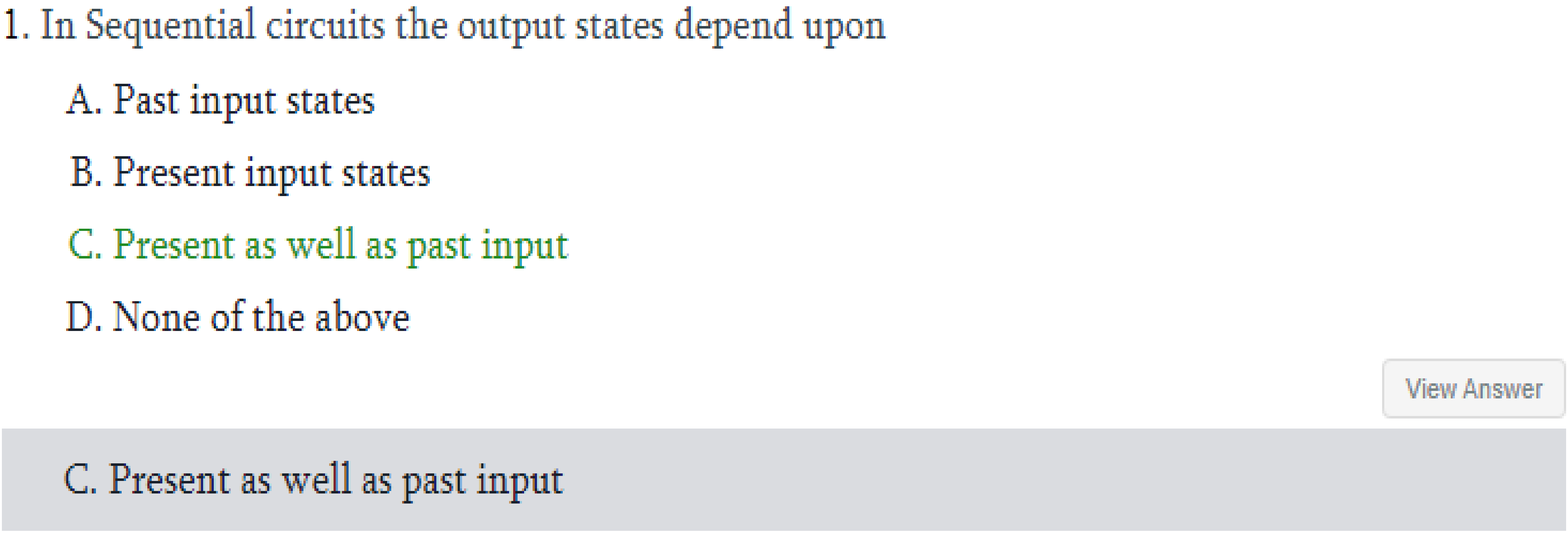
1. In Sequential circuits the output states depend upon

   A. Past input states

   B. Present input states

   C. Present as well as past input

   D. None of the above



1. In Sequential circuits the output states depend upon

A. Past input states

B. Present input states

C. Present as well as past input

D. None of the above

C. Present as well as past input

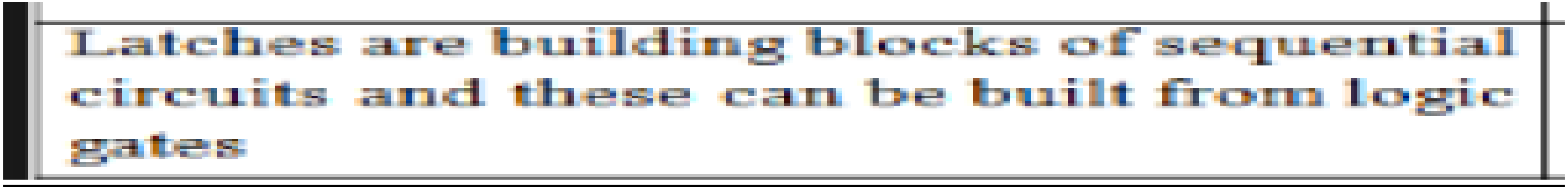
Latch

1) Latch is a sequential logic ckt which takes all it's input continuously & will change its o/p as soon as the input changes without waiting for clock s/g.

2) It is capable of locking (latching) the information.

3) (It has two output Q & $\bar{Q}$ which are complement of each other)

NOR LATCH



Latches are building blocks of sequential circuits and these can be built from logic gates

SR NoR latch

S R	Q $\bar{Q}$
00	
01	
10	

Idle mode

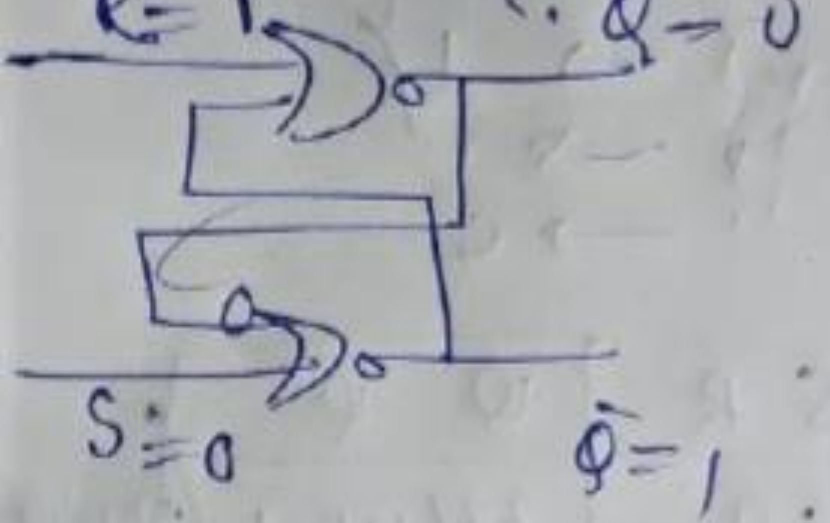
NOR

00 → 1

10 → 0

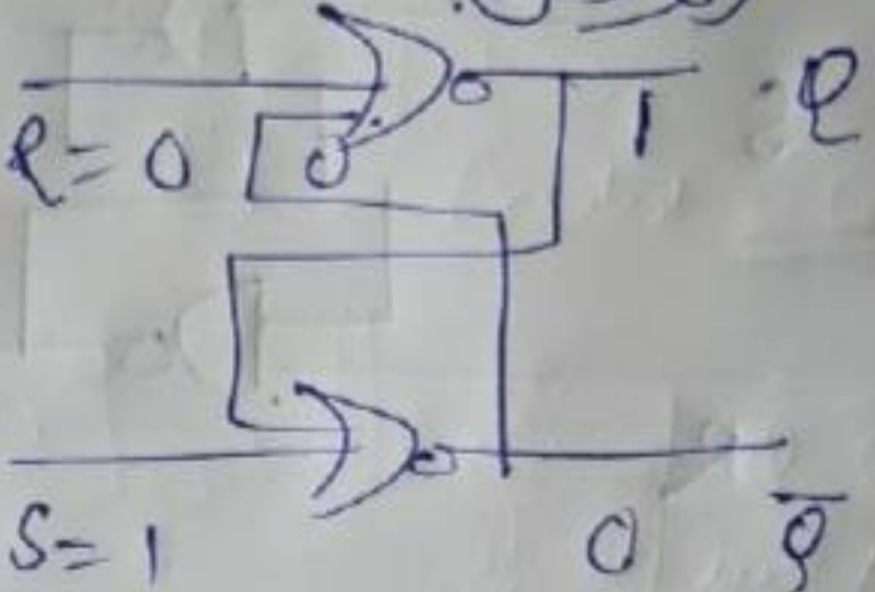
01 → 0

11 → 0



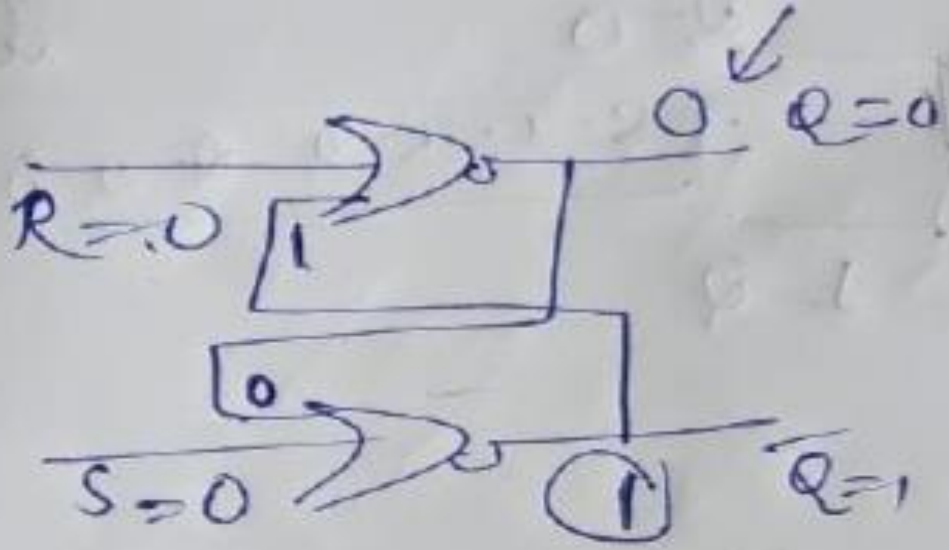
$S=0$; $R=1$; $Q=0$; $\bar{Q}=1$

Reset



$S=1$ $R=0$; $Q=1$; $\bar{Q}=0$

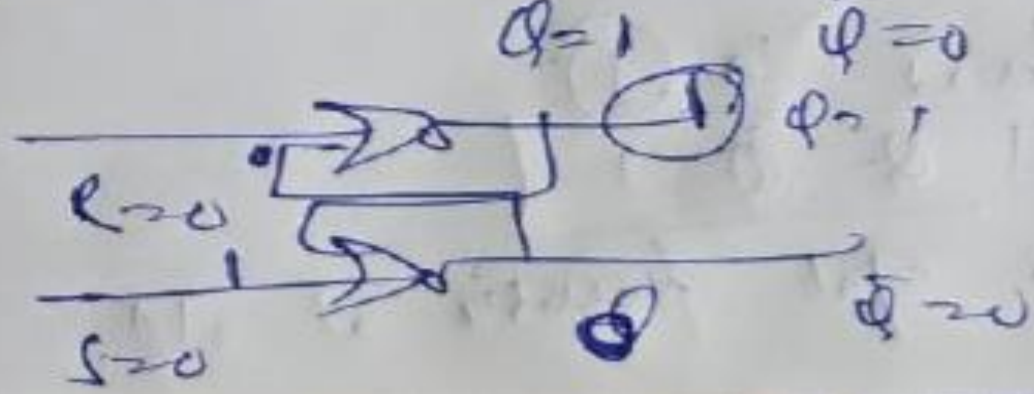
Set

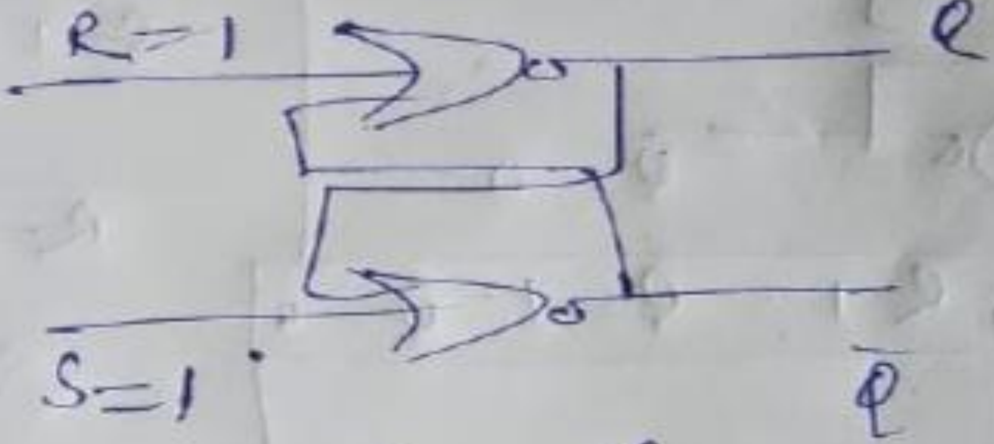


we assume previous state

$Q=0 ; \bar{Q}=1$

$Q=0 ; \bar{Q}=1$





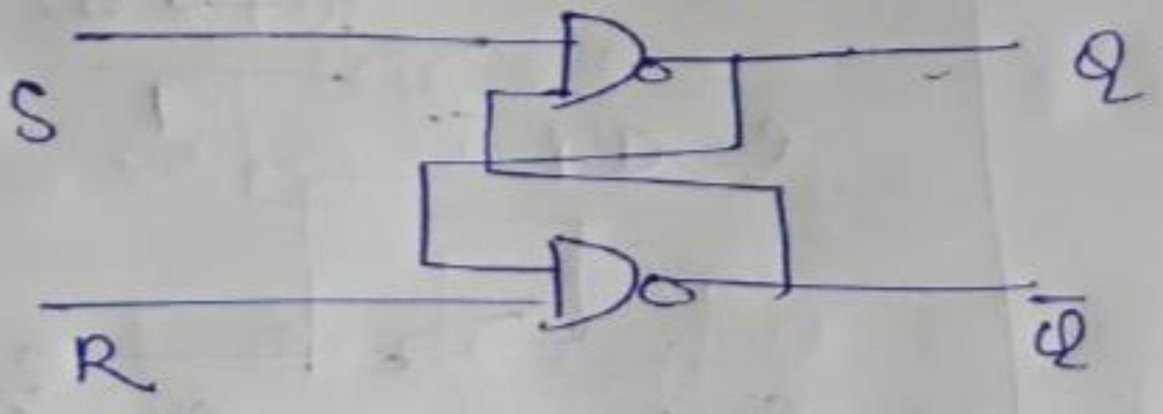
$Q=0$; ~~$\bar{Q}=0$~~

not possible

S	R	Q	$\bar{Q}$
0	0	Memory	
0	1	0	
1	0	1	0
1	1	Invalid (Not used)	

SR NAND latch FF

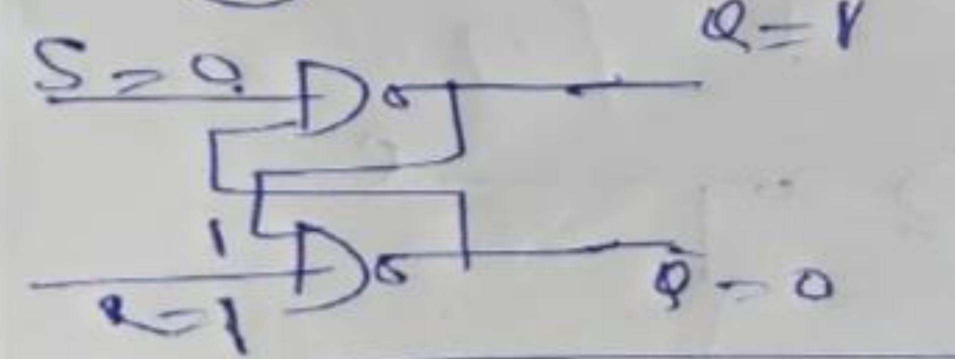
'0' → bre Imove



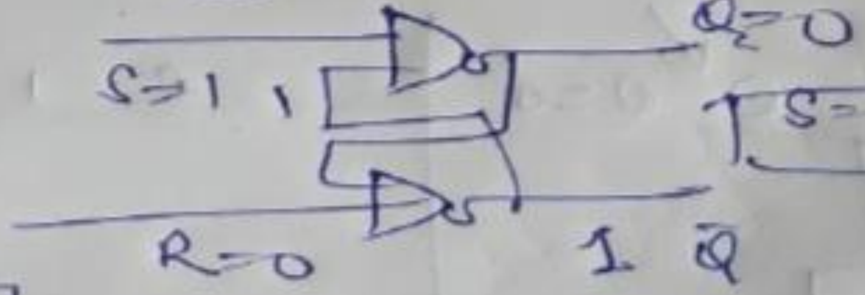
NAND

0 0 →	1
0 1 →	1
1 0 →	1
1 1 →	0

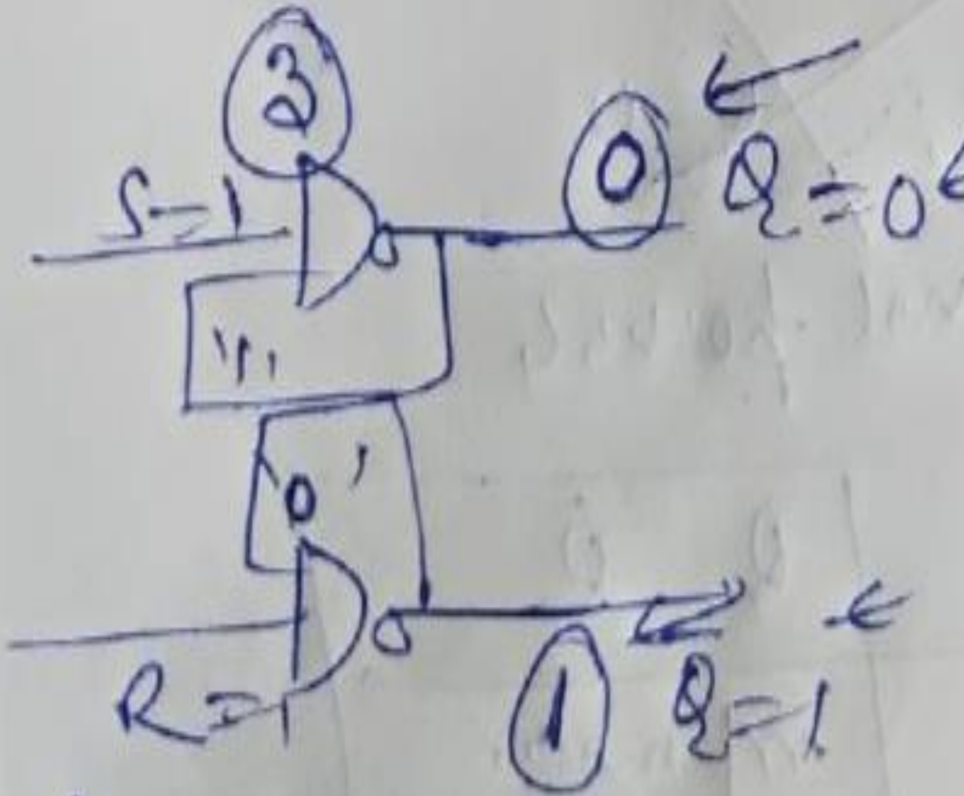
S R	Q $\bar{Q}$
0 0	Not used / invalid
0 1	1 0
1 0	0 1
1 1	Memory / No change

① 

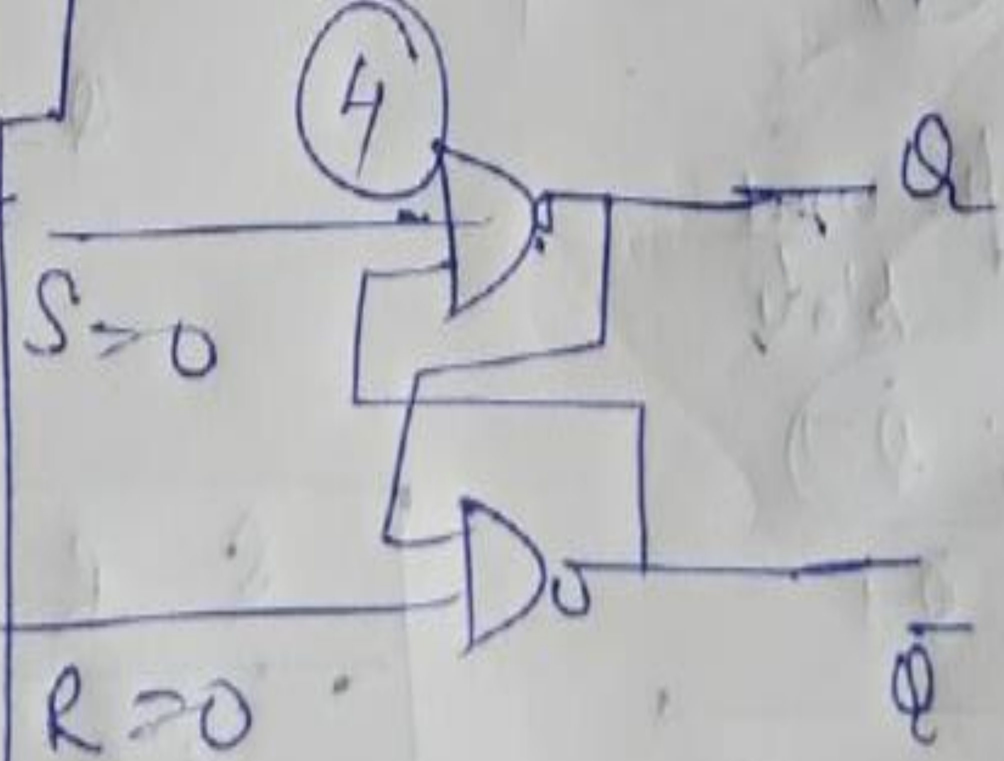
S=0; R=1; Q=1; $\bar{Q}$=0

② 

S=1; R=0; Q=0; $\bar{Q}$=



Since both '1' so assume previous state-
Because nothing is forced to 1.



Both forced to 1

$\boxed{Q = \bar{Q} = 1}$

Which is not possible at all.

Flip Flop

- Flip Flop is a sequential circuit which is used to store single bit of information at a time i.e., 0 or 1.
- Used:-Registers

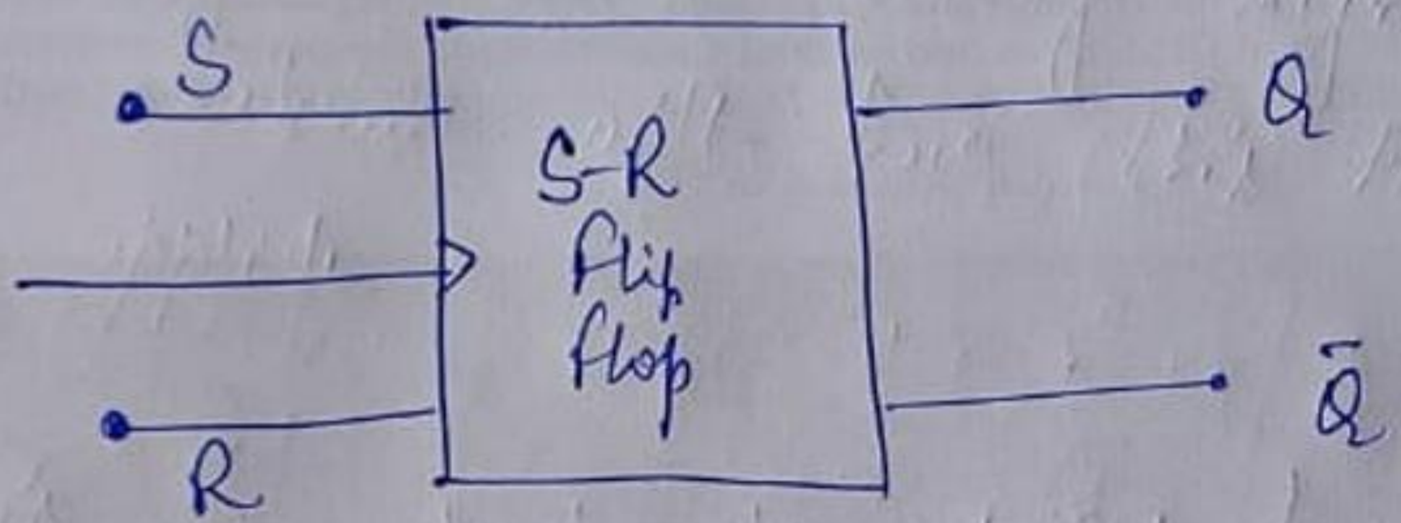
SR FLIP FLOP

- SR Flip Flop is the set reset flip flop. It consist of SR latch with clock circuit.

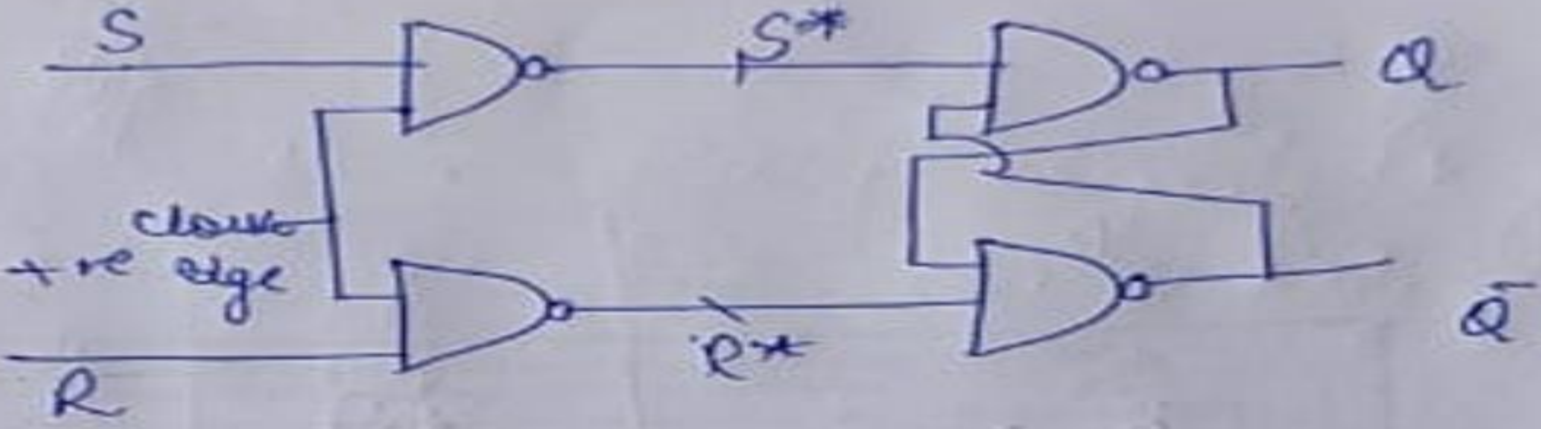
- It may be positive edge triggered or negative edge triggered.

- Triggering is the process of change of state of flop by applying input signal.

1 S-R flip flop



positive edge triggered

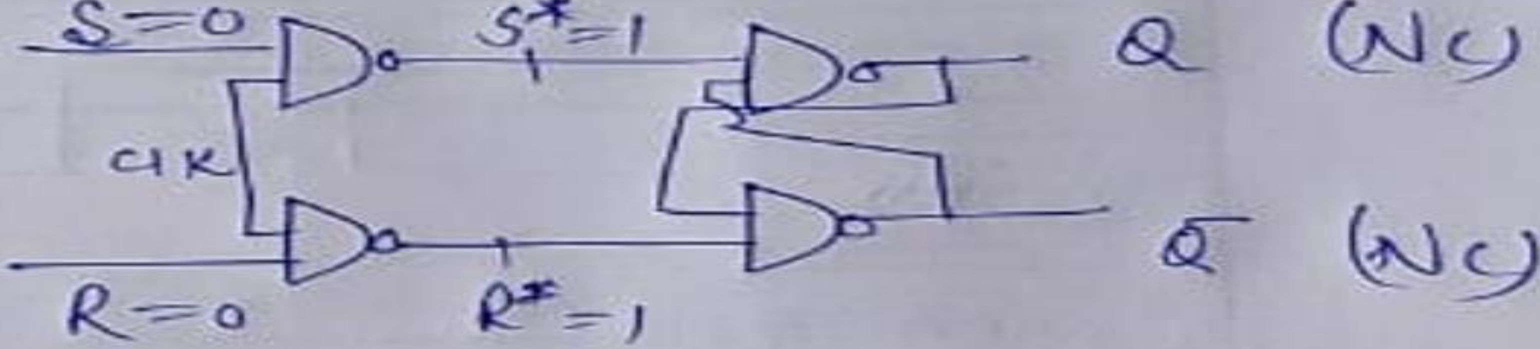


(SR NAND Latch)

S^*	R^*	Q $\bar{Q}$
0	0	Invalid
0	1	1 0
1	0	0 1
1	1	NC No change

Case I :-

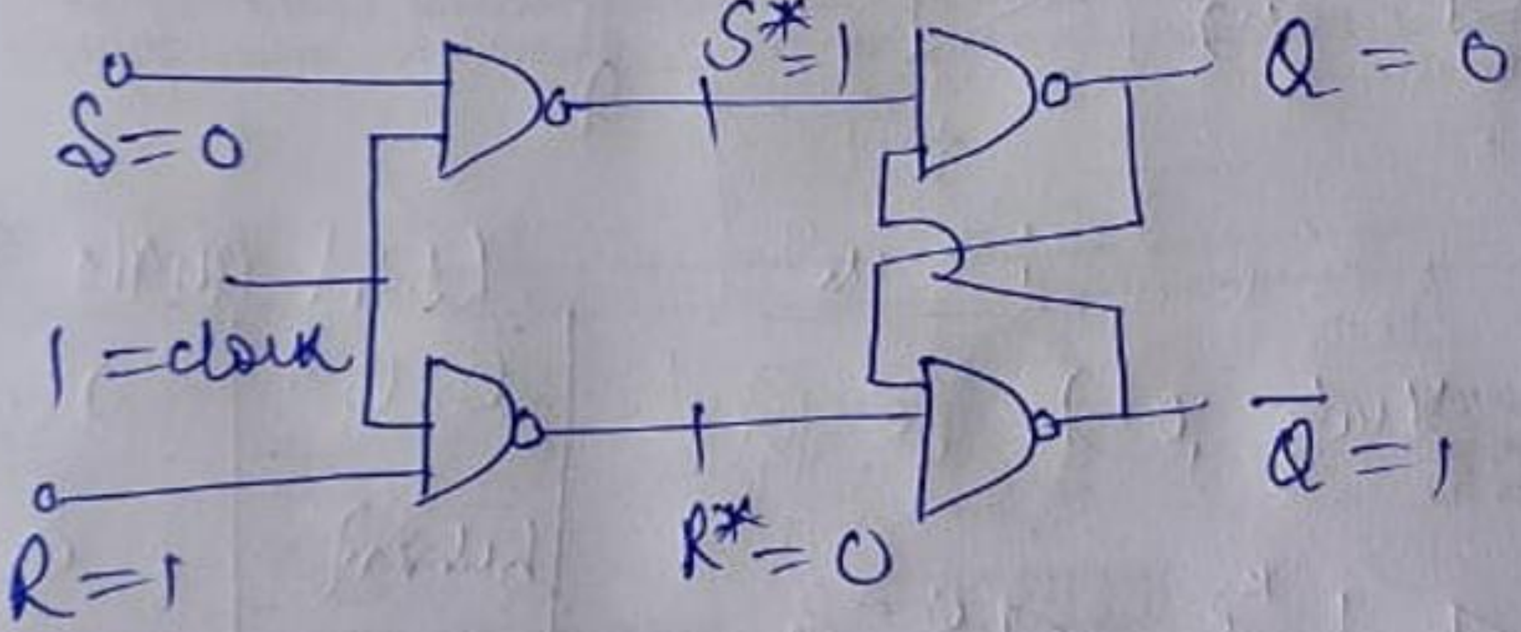
S=0, R=0, clock = ↑



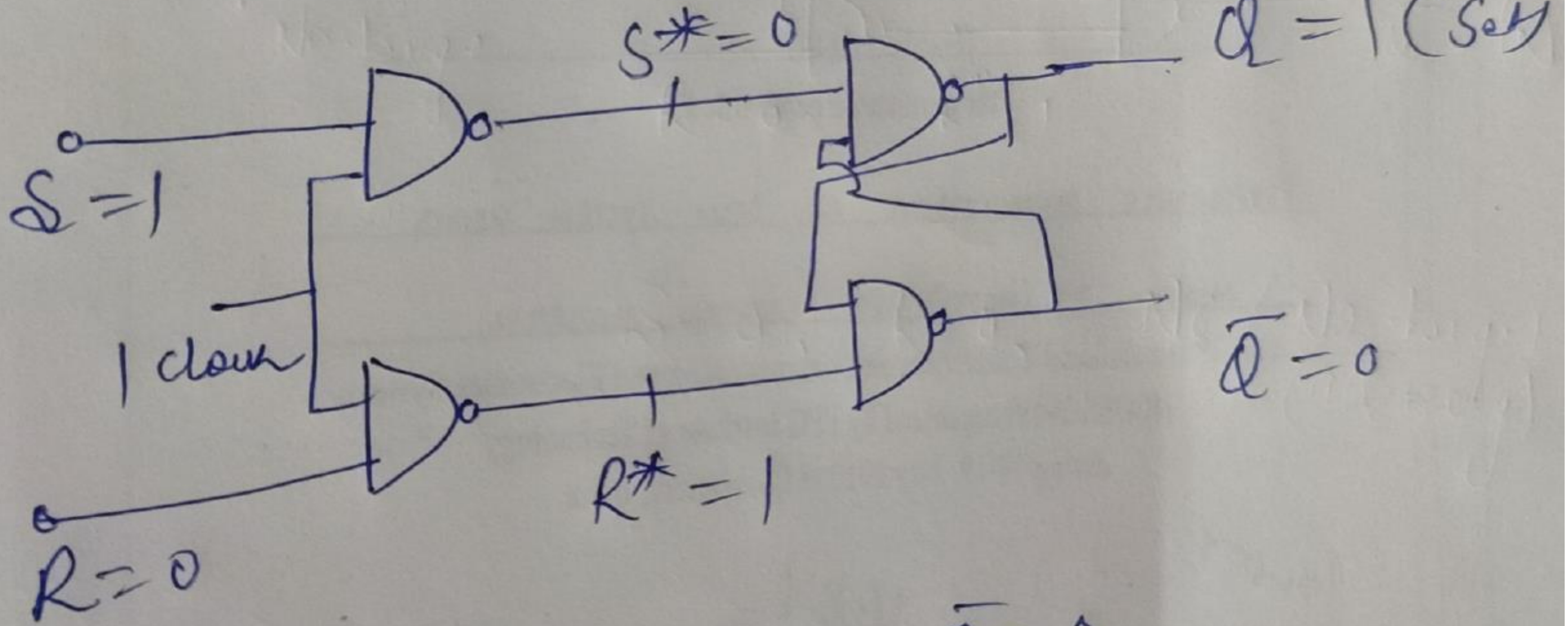
S=0, R=0, NC (No change)
(Memory)

Case II :-

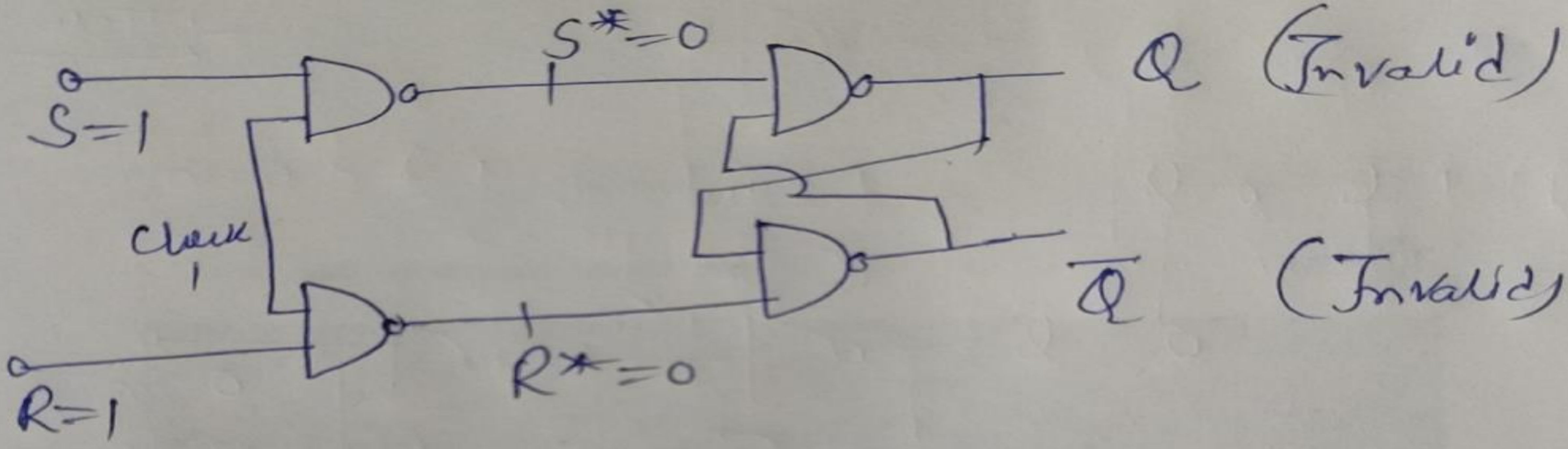
S=0 & R=1 , clock = 1 Q=0, $\bar{Q}$=1 (Reset)



Case III :-

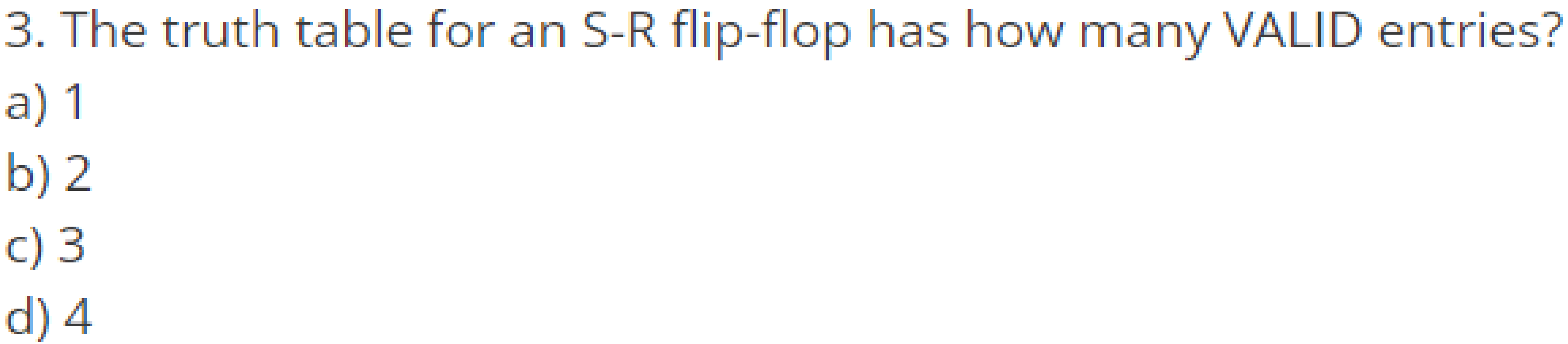


$S = 1; \; R = 0; \quad Q = 1; \; \bar{Q} = 0$



clock	S	R	Q	$\overline{Q}$	
↑	0	0	NC	NC	
↑	0	1	0	1	Reset
↑	1	0	1	0	Set
↑	1	1	Invalid		

MCQ



3. The truth table for an S-R flip-flop has how many VALID entries?

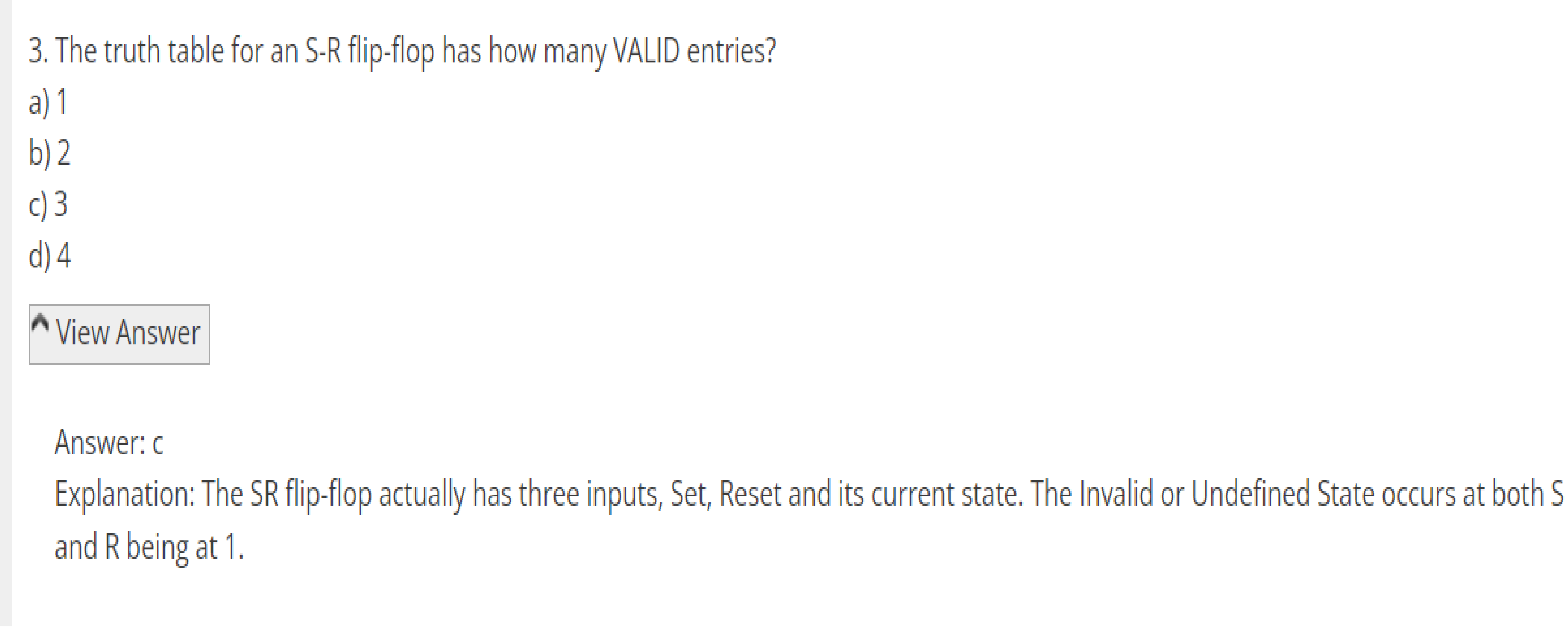
a) 1

b) 2

c) 3

d) 4

MCQ



3. The truth table for an S-R flip-flop has how many VALID entries?

a) 1

b) 2

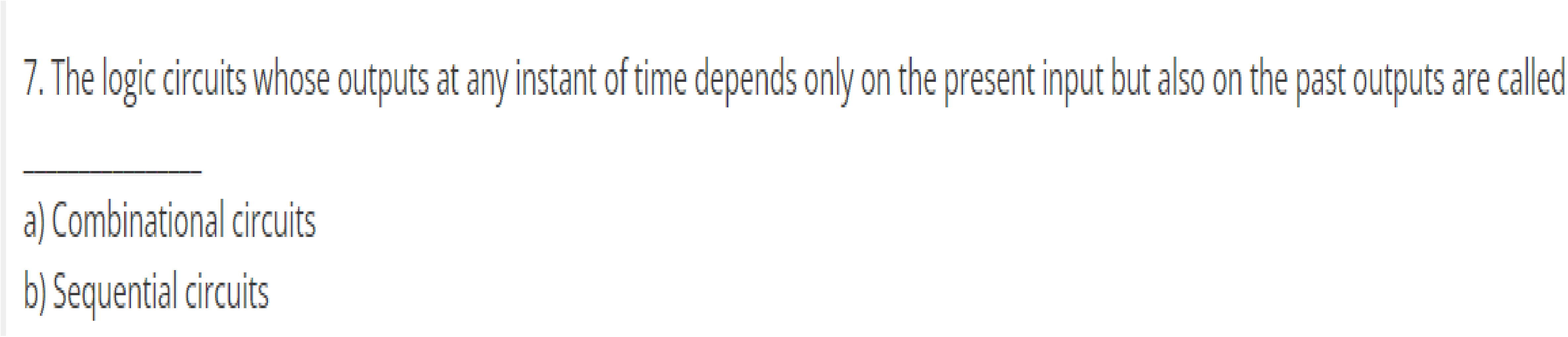
c) 3

d) 4

Answer: c

Explanation: The SR flip-flop actually has three inputs, Set, Reset and its current state. The Invalid or Undefined State occurs at both S and R being at 1.

MCQ



7. The logic circuits whose outputs at any instant of time depends only on the present input but also on the past outputs are called

____

a) Combinational circuits

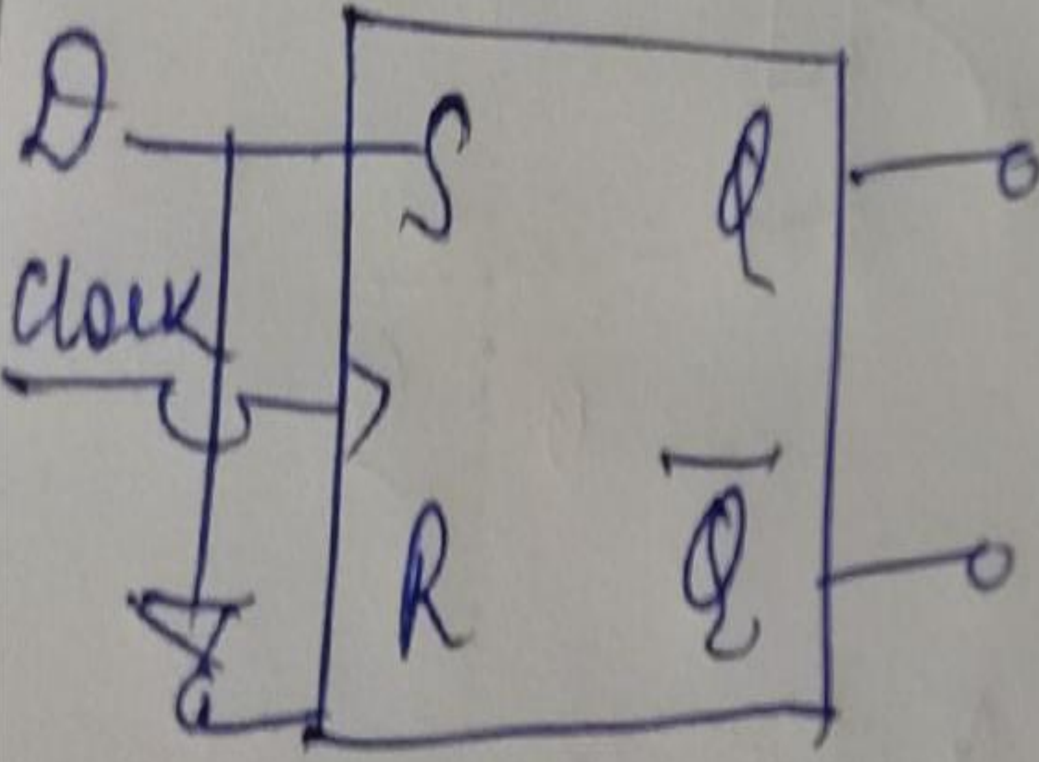
b) Sequential circuits

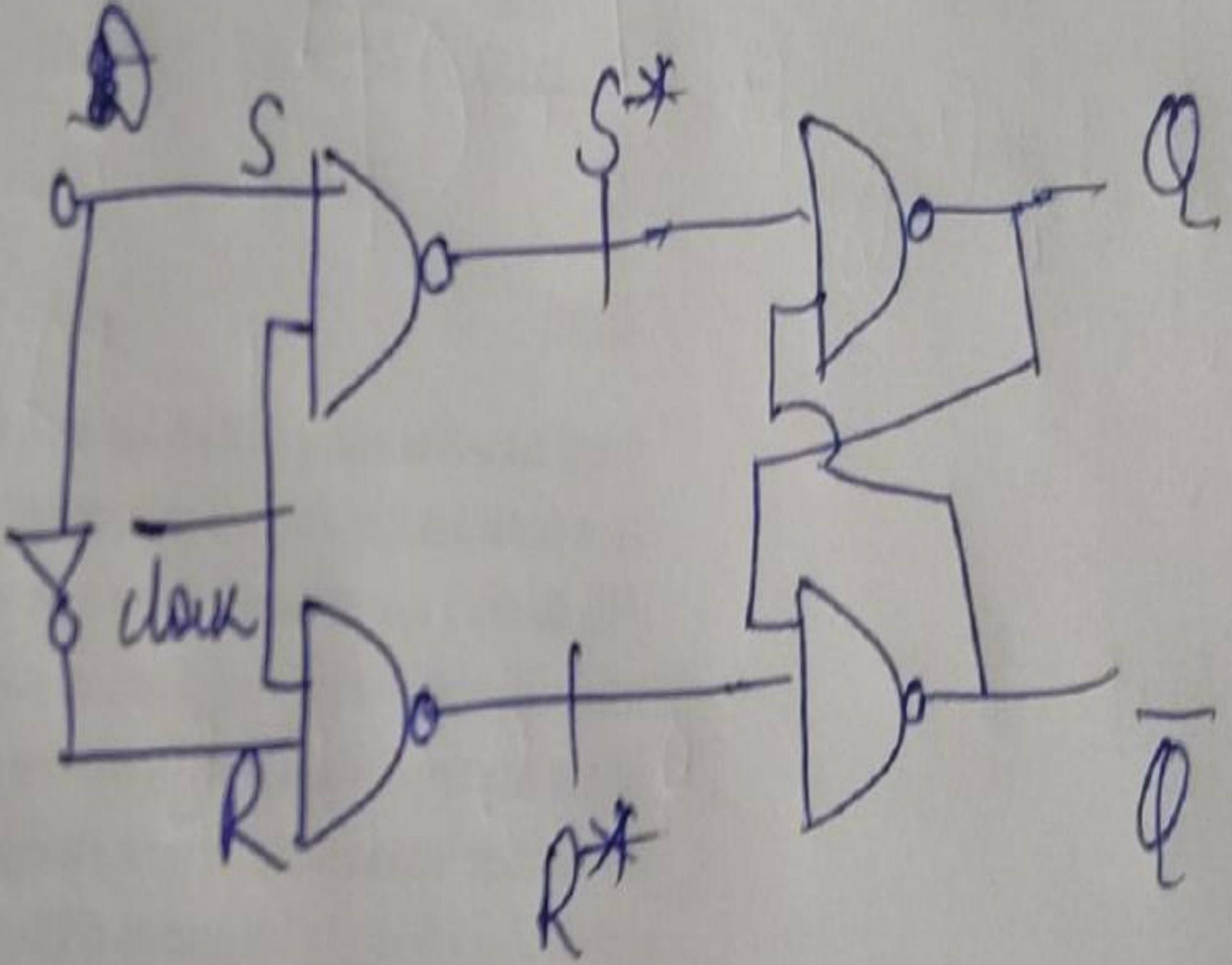
MCQ

- Answer: b
  Explanation: In sequential circuits, the output signals are fed back to the input side.

D Flip Flop (Delay Flip Flop)

- It can be designed from S-R Flip Flop by putting an inverter or NOT gate in S- R Flip Flop.


D
Clock
S
Q
R
$\overline{Q}$


D
S
S*
Q
clock
R
R*
$\overline{Q}$

Case I → $D = 1 \Rightarrow S = 1$; $R = 0$

$Q = 1$; $\overline{Q} = 0$

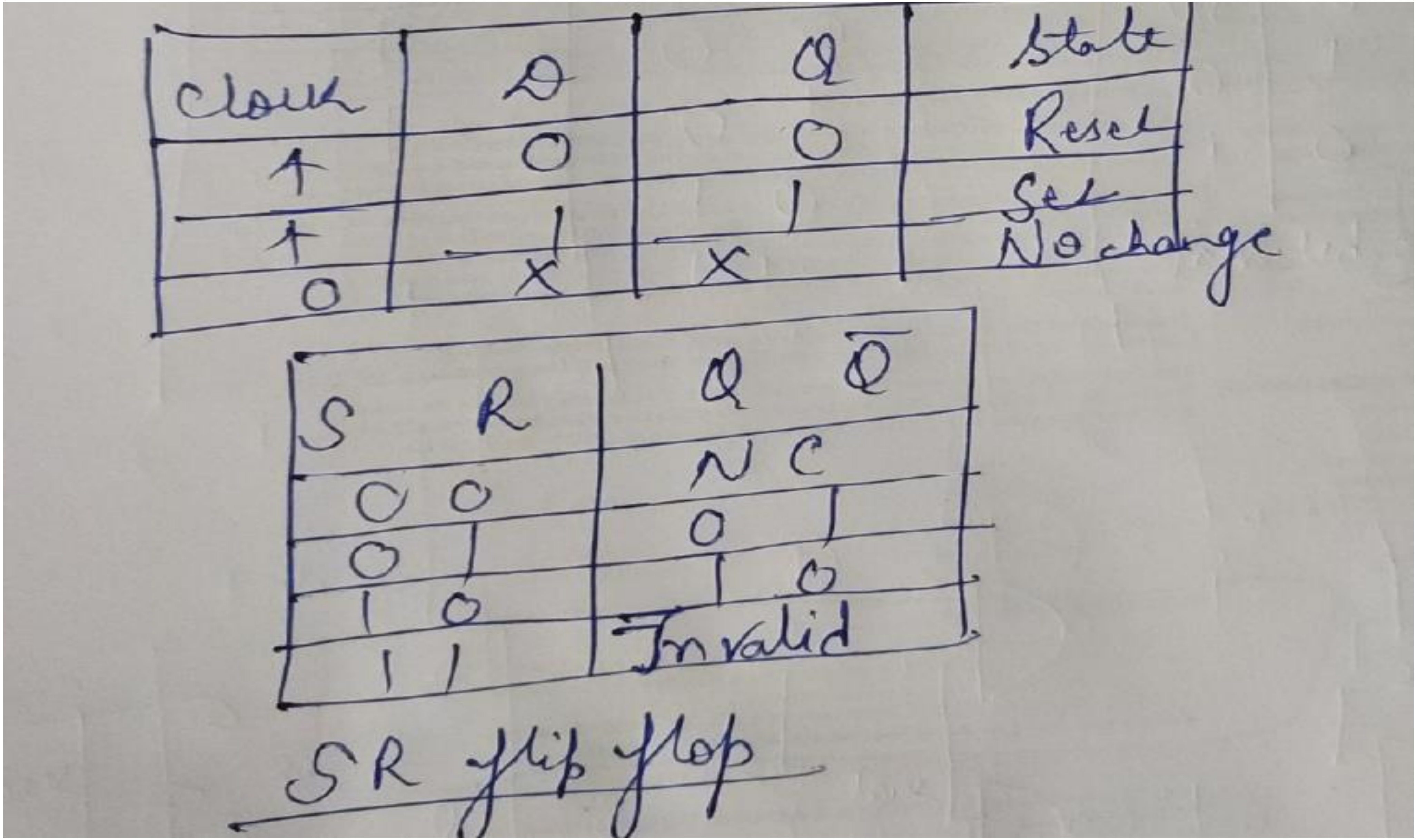
(Set)

Case II → $D = 0 \Rightarrow S = 0$; $R = 1$

$Q = 0$; $\overline{Q} = 1$

(Reset)

clock	D	Q	State
↑	0	0	Reset
↑	1	1	Set
0	X	X	No change

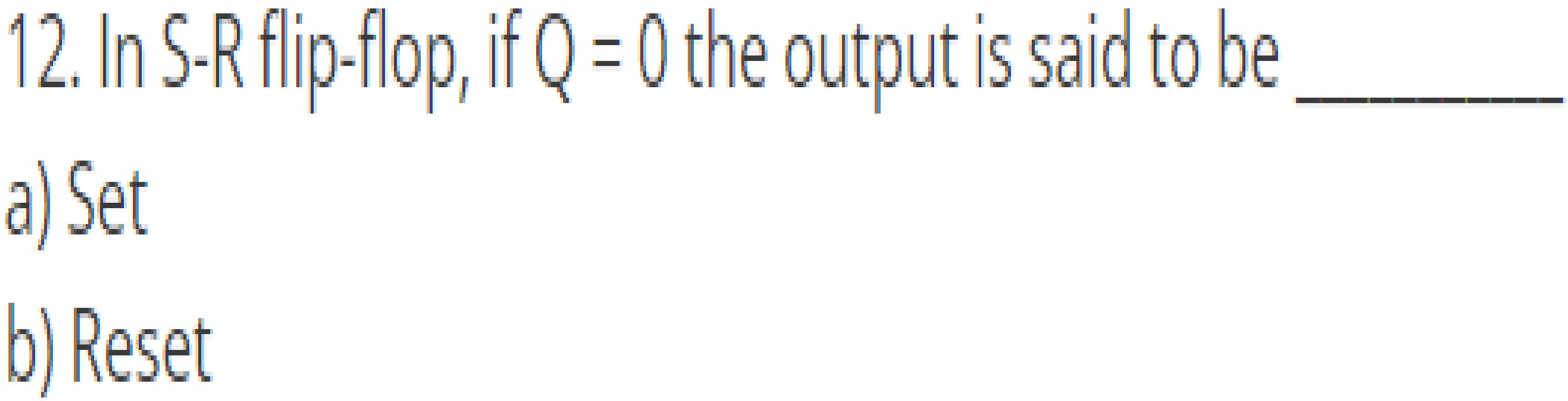


Clock	D	Q	State
↑	0	0	Reset
↑	1	1	Set
0	X	X	No change

S	R	Q	$\overline{Q}$
0	0	N C	
0	1	0	1
1	0	1	0
1	1	Invalid	

SR flip flop

MCQ



12. In S-R flip-flop, if Q = 0 the output is said to be ________

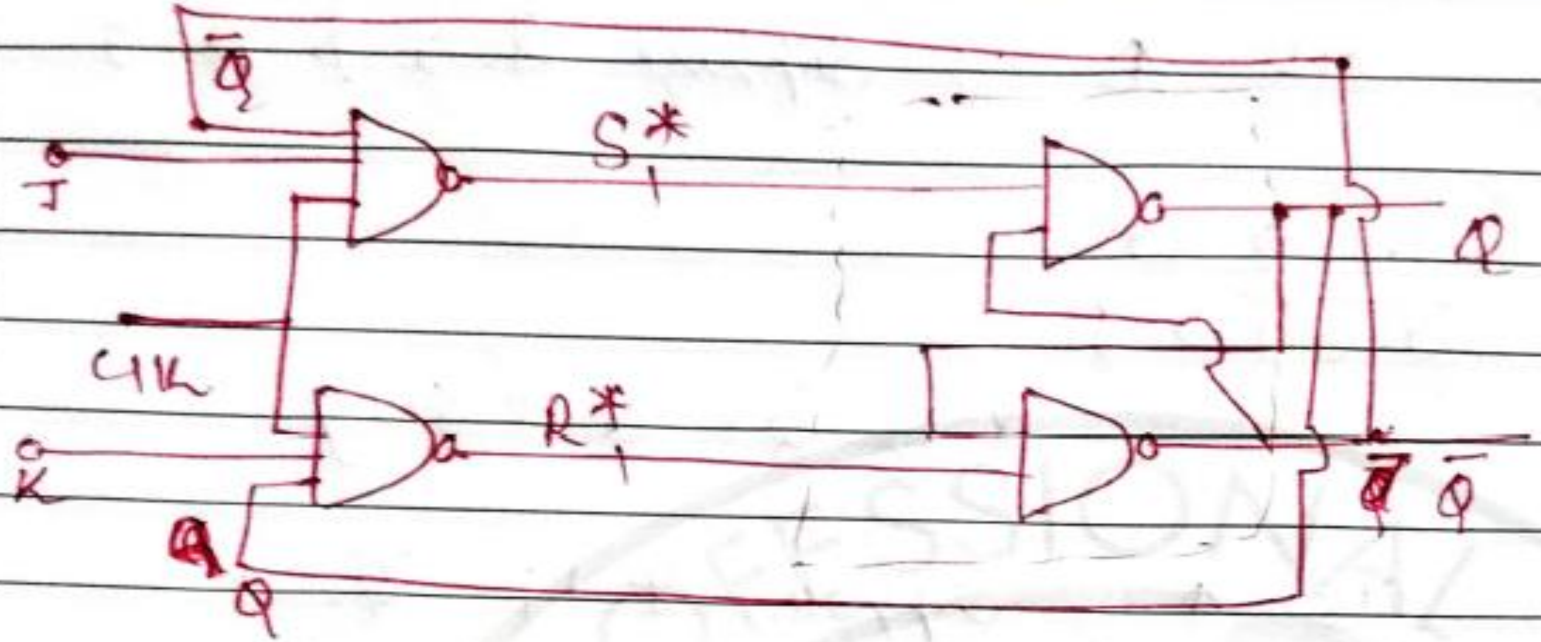
a) Set

b) Reset

MCQ

- Answer: b
- Explanation: In S-R flip-flop, if Q = 0 the output is said to be reset and set for Q = 1

J-K flip flop



SR NAND Latch

S^*	R^*	Q	$\bar{Q}$
0	0	Invalid	
0	1	1	0
1	0	0	1
1	1	NC/memory	

Case 1:- Clk → ↑ ; $J=0$; $K=0$

$R^*=1$ & $S^*=1$

Case2:- $Clk \rightarrow \uparrow$; $J=0$; $K=1$; $Q=0$; $\bar{Q}=1$ (Reset

a) If suppose $Q=0$ & $\bar{Q}=1$

$S^* = \overline{\bar{Q}.J.clk} = 1$

$R^* = \overline{Q \cdot K \cdot clk} \Rightarrow 1$ } NC state

so $Q=0$ & $\bar{Q}=1$ (Reset

(b) If suppose $Q=1$ & $\bar{Q}=0$

$S^* = \overline{0.0.1} \Rightarrow 1$

$R^* = \overline{1.1.1} \Rightarrow 0 \Rightarrow Q=0$ & $\bar{Q}=1$ (Reset

Case III :- $S^* = \overline{\bar{Q} \cdot J \cdot Clk}$

$R^* = \overline{Q \cdot K \cdot Clk}$

previous state

A) $J=1$; $K=0$; $clk = \uparrow$; Suppose $Q=0$ & $\bar{Q}=1$

$S^* = \overline{1 \cdot 1 \cdot 1} \Rightarrow 0$

$R^* = \overline{0 \cdot 0 \cdot 1} \Rightarrow 1$

$Q=1$; $\bar{Q} = 0$ { Set

(B) $J=1$; $K=0$, $clk = \uparrow$; Suppose $Q=1$; $\bar{Q}=0$

$S^* = \overline{0 \cdot 1 \cdot 1} \Rightarrow 1$ } NC (No change)

$R^* = \overline{1 \cdot 0 \cdot 1} \Rightarrow 1$ } means

$Q=1$; $\bar{Q}=0$ (Remain same)

Case IV ; $Clk = \uparrow$; $J=1$, $K=1$

$S^* = \overline{\bar{Q}\cdot J\cdot Clk}$

$R^* = \overline{Q\cdot K\cdot Clk}$.

Previous state assume

$Q=1$; $\bar{Q}=0$

$\left.\begin{array}{l} S^* = \overline{0.1.1} \Rightarrow 1 \\ R^* = \overline{1.1.1} \Rightarrow 0 \end{array}\right\}$ $Q=0$ & $\bar{Q}=1$ Reset

$Q_{n+1} = 0$ & $\overline{Q_{n+1}} = 1$

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

$Q=0$ & $\bar{Q}=1$

$\left.\begin{array}{l} S^* = \overline{1} = 0 \\ R^* = 1 \end{array}\right\}$ SET

$Q=1$; $\bar{Q}=0$

$Q_{n+1} = 1$ ($\overline{Q_n}$ }

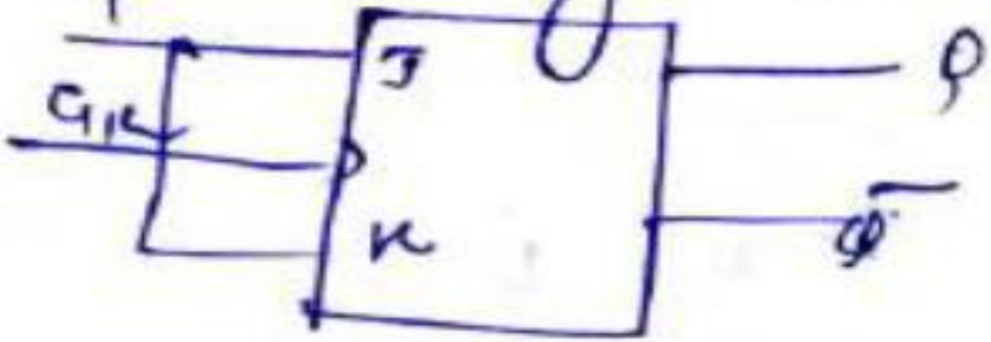
(Toggling)

Clock	J	K	Q_{n+1}, O/P	$\bar{Q}_{n+1}$,	State
↑	0	0	NC	NC	Hold
↑	0	1	0	1	Reset
↑	1	0	1	0	Set
↑	1	1	$\bar{Q}_n$	Q_n	Toggle

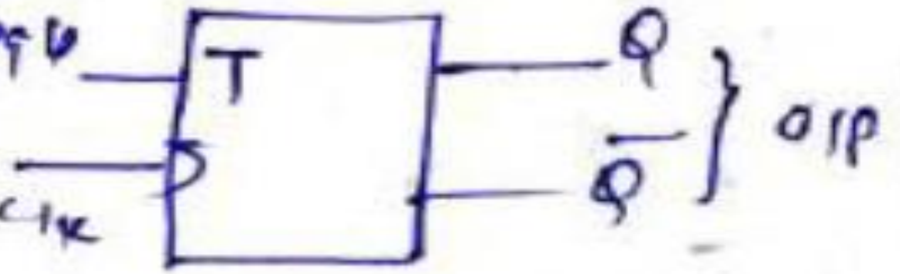
Thus as long as J=K=1 ; the o/p will keep toggling indefinitely.
This multiple toggling in JK is called Race around
condition.

T-ff (toggle flip flop)

Toggle flip flop is basically a JK flip flop with J & K terminals permanently connected together. It has only one input T.



JK ff is convto T flipf.



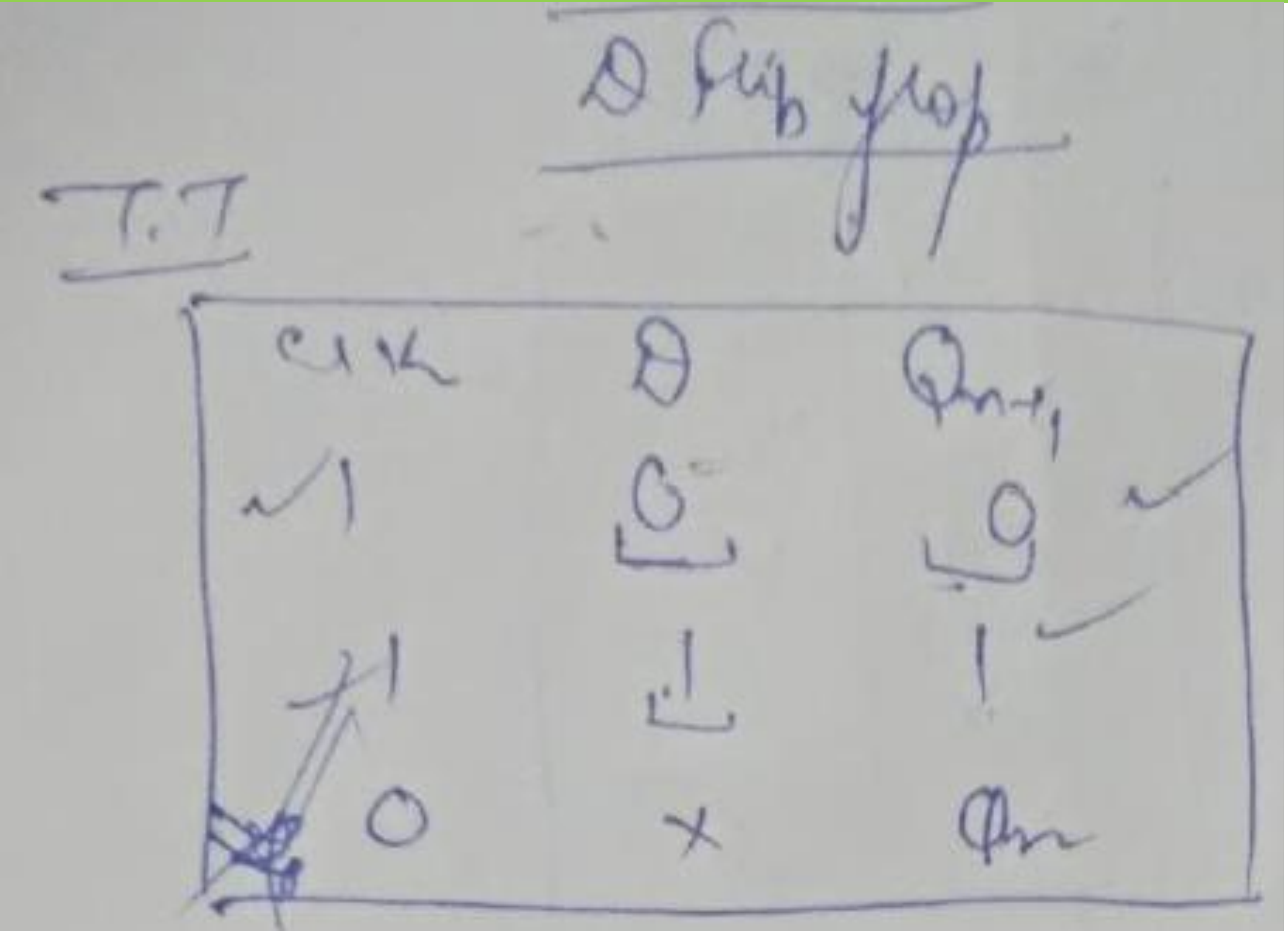
Logi symbol of positive edge triggers T flipflop triggered T flipflop

Case 1: if T=0; J=0; K=0 NC

if T=1; J=1; K=1 o/p will toggle corresponding to every leading edge of clock signal.

Clk	T	Q_{n+1}	$\bar{Q}_{n+1}$	
↑	1	$\bar{Q}_n$	Q_n	Toggle
↓	0	Q_n	$\bar{Q}_n$	NC

Truth Table, Characteristic Table and Excitation Table(D-Flip Flop)

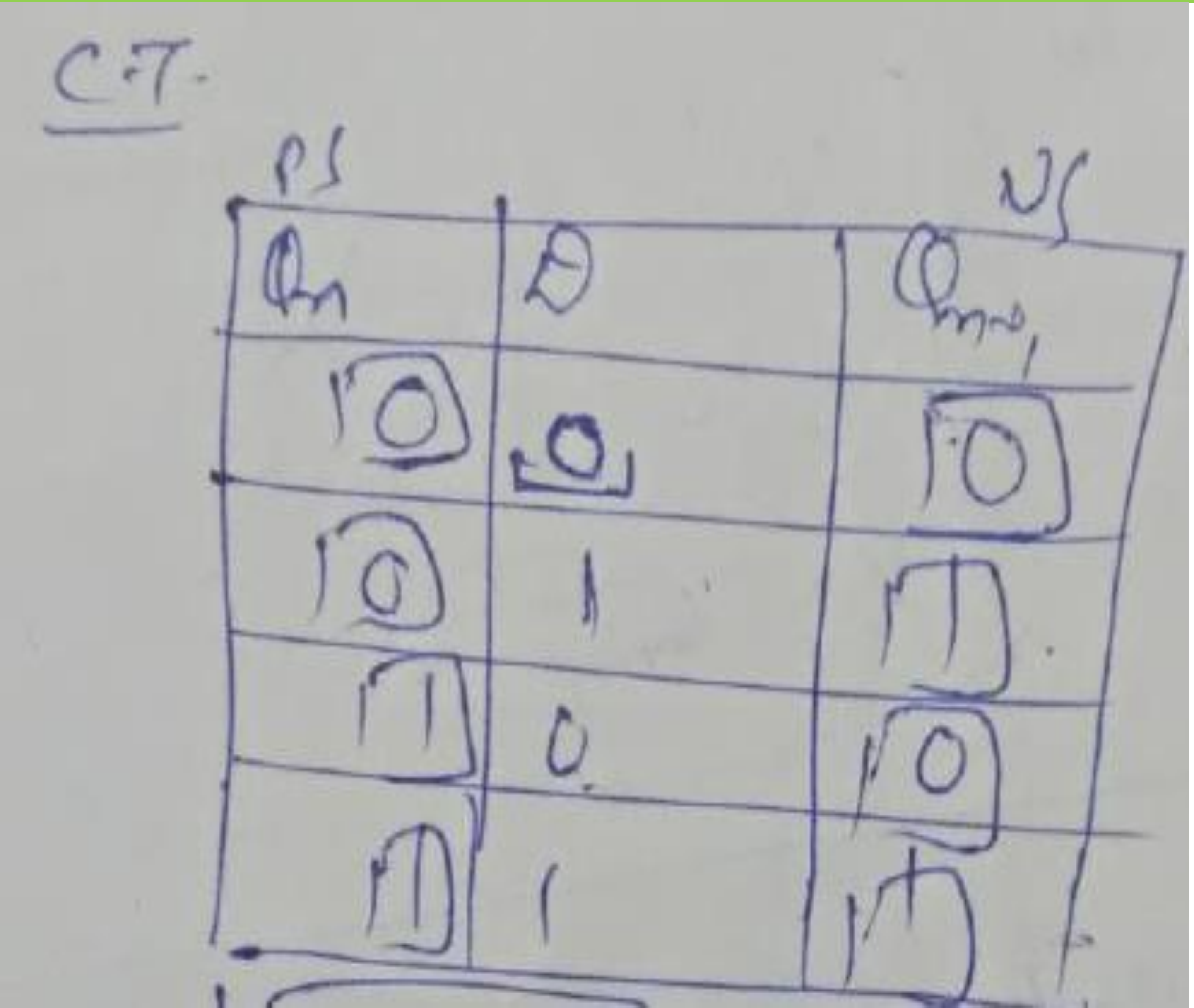


D flip flop

T.T

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

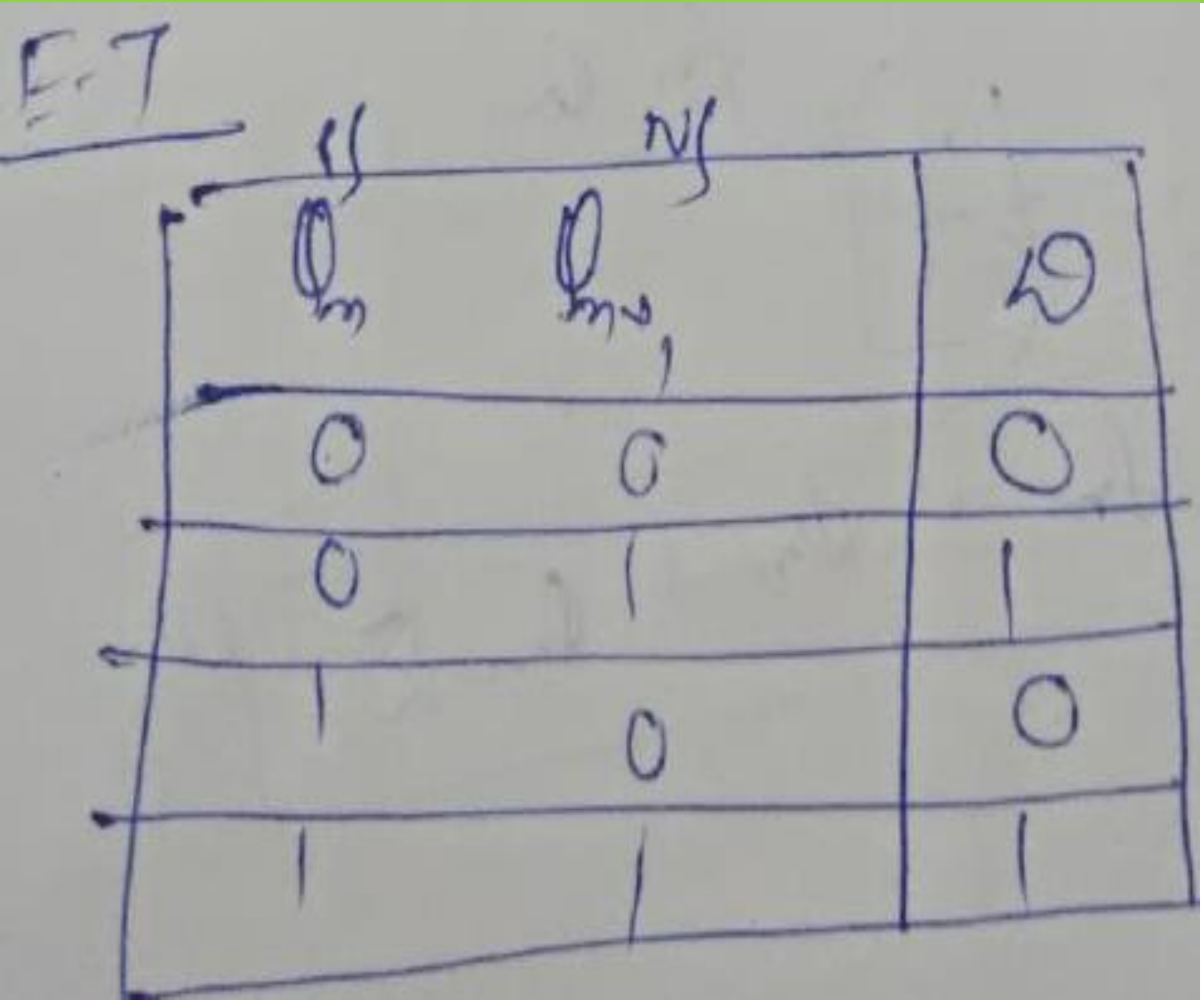
Characteristic Table(D Flip Flop)



C.T.

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

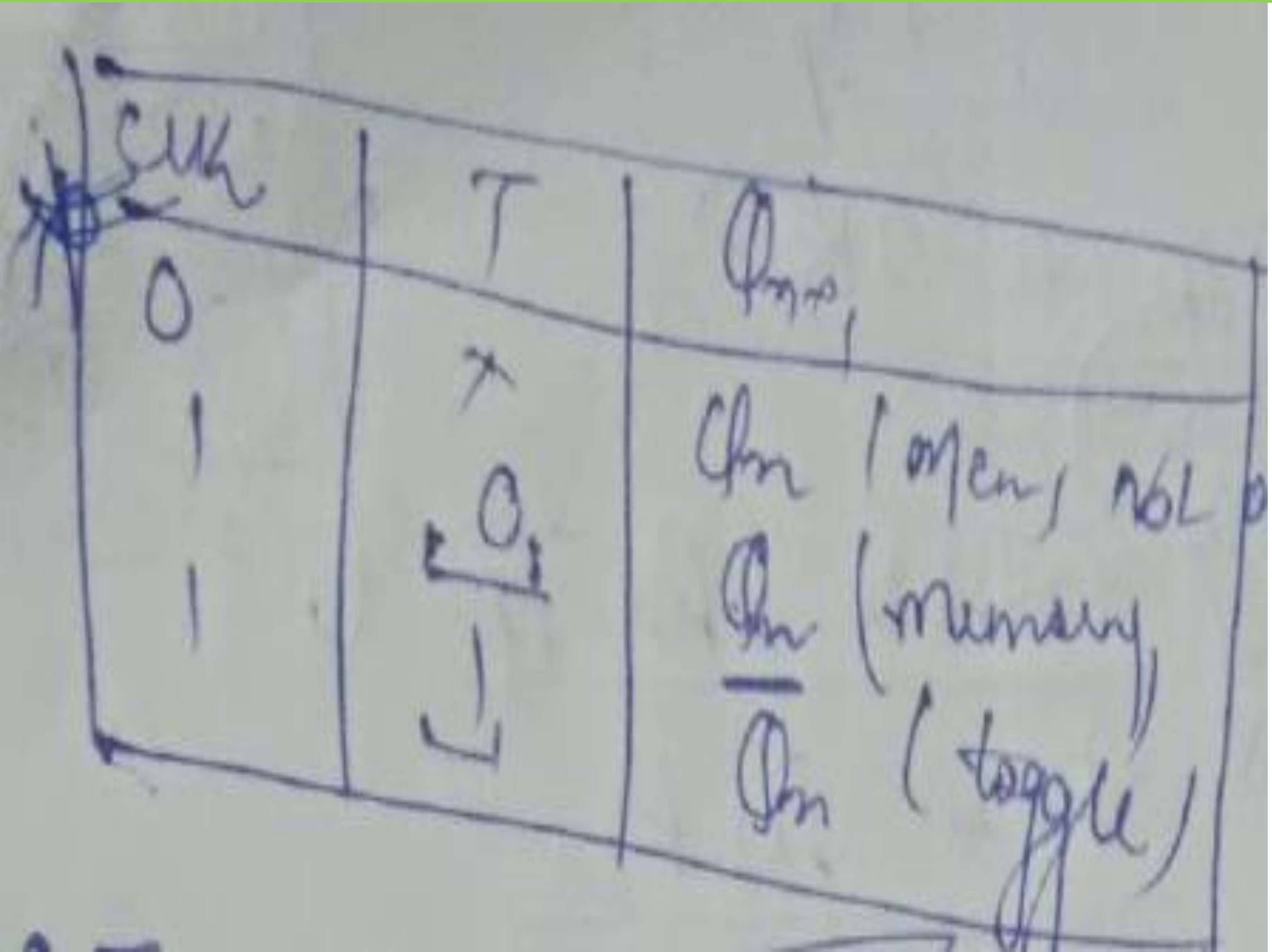
Excitation Table(D Flip Flop)



E.T

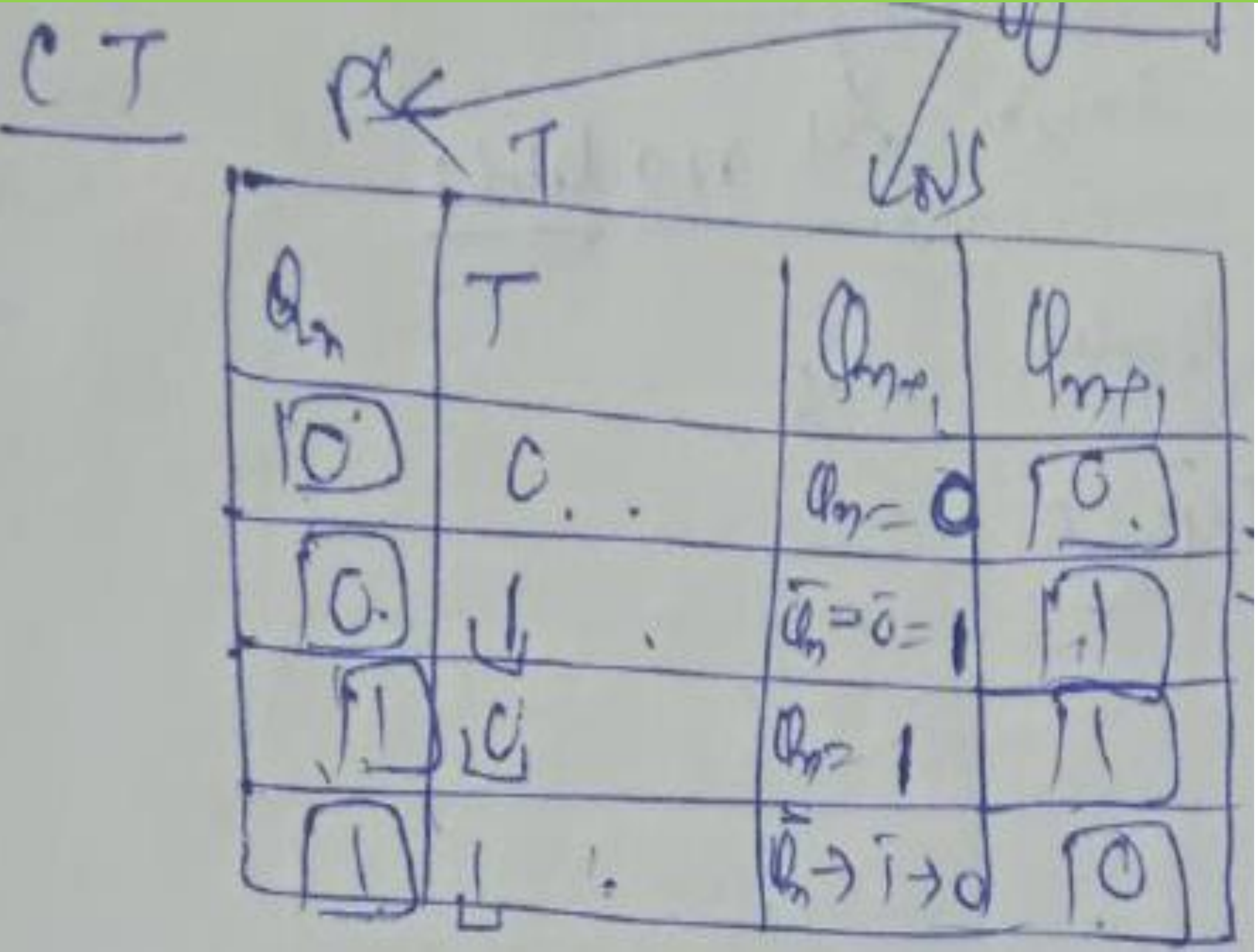
PS	NS	
Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Truth Table, Characteristic Table and Excitation Table(T-Flip Flop)



Clk	T	Q_{n+1}
0	x	Q_n (means not
1	0	Q_n (memory)
1	1	$\overline{Q}_n$ (toggle)

Characteristic Table(T Flip Flop)



CT

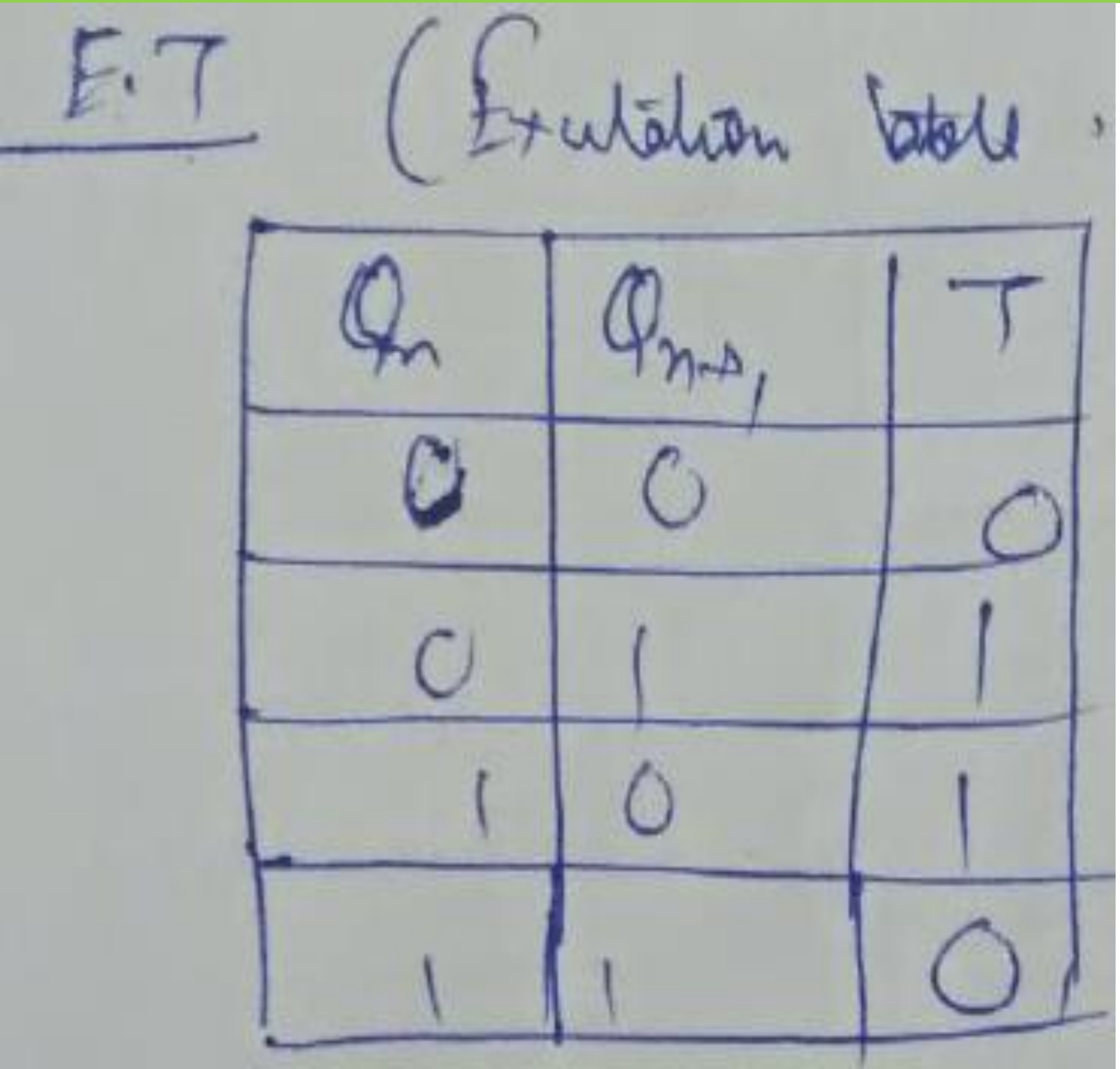
PS

T

NS

Q_n	T	Q_{n+1}	Q_{n+1}
0	0	$Q_n = 0$	0
0	1	$\bar{Q}_n = \bar{0} = 1$	1
1	0	$Q_n = 1$	1
1	1	$\bar{Q}_n \rightarrow \bar{1} \rightarrow 0$	0

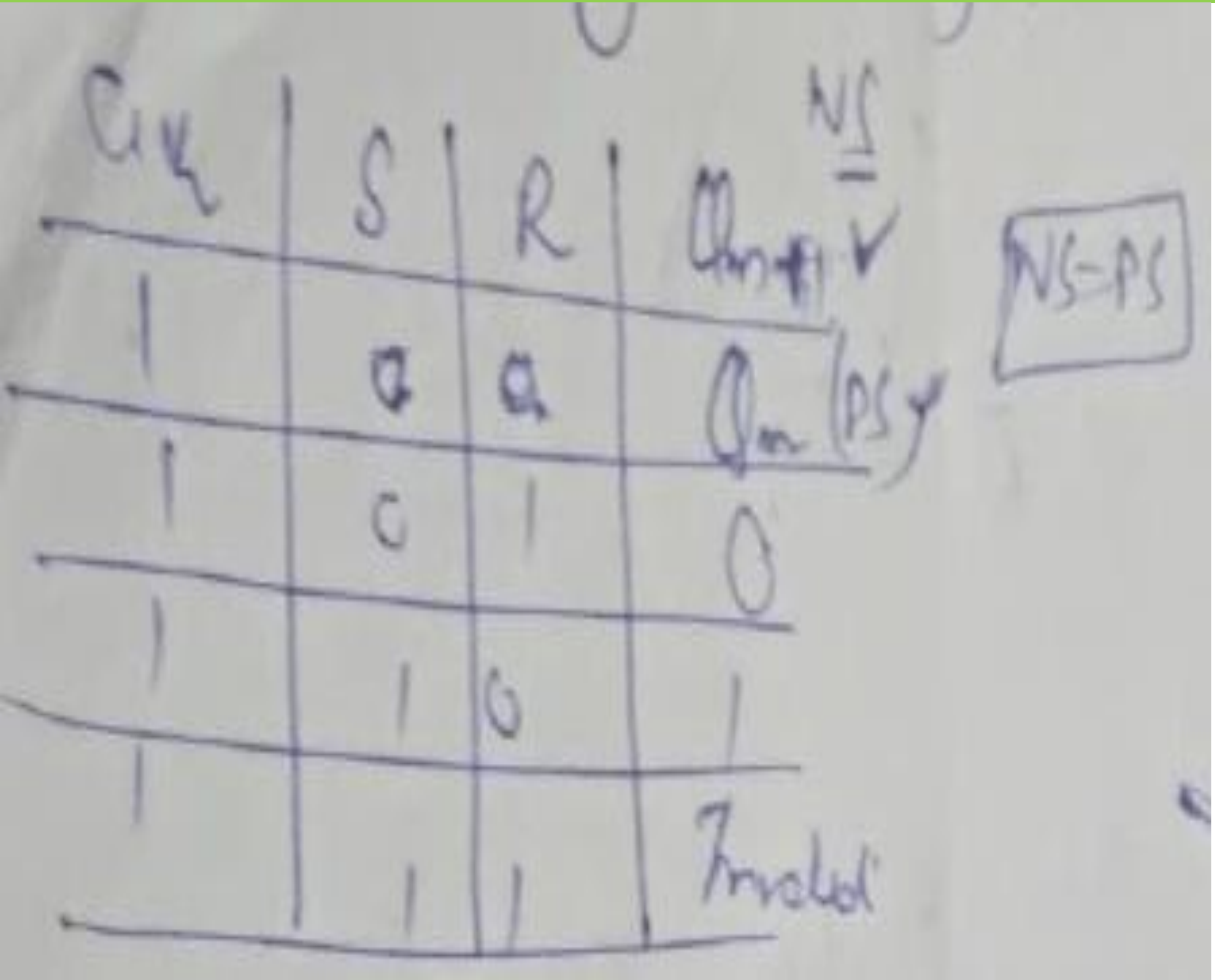
Excitation Table(T-Flip Flop)



E.T (Excitation table

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

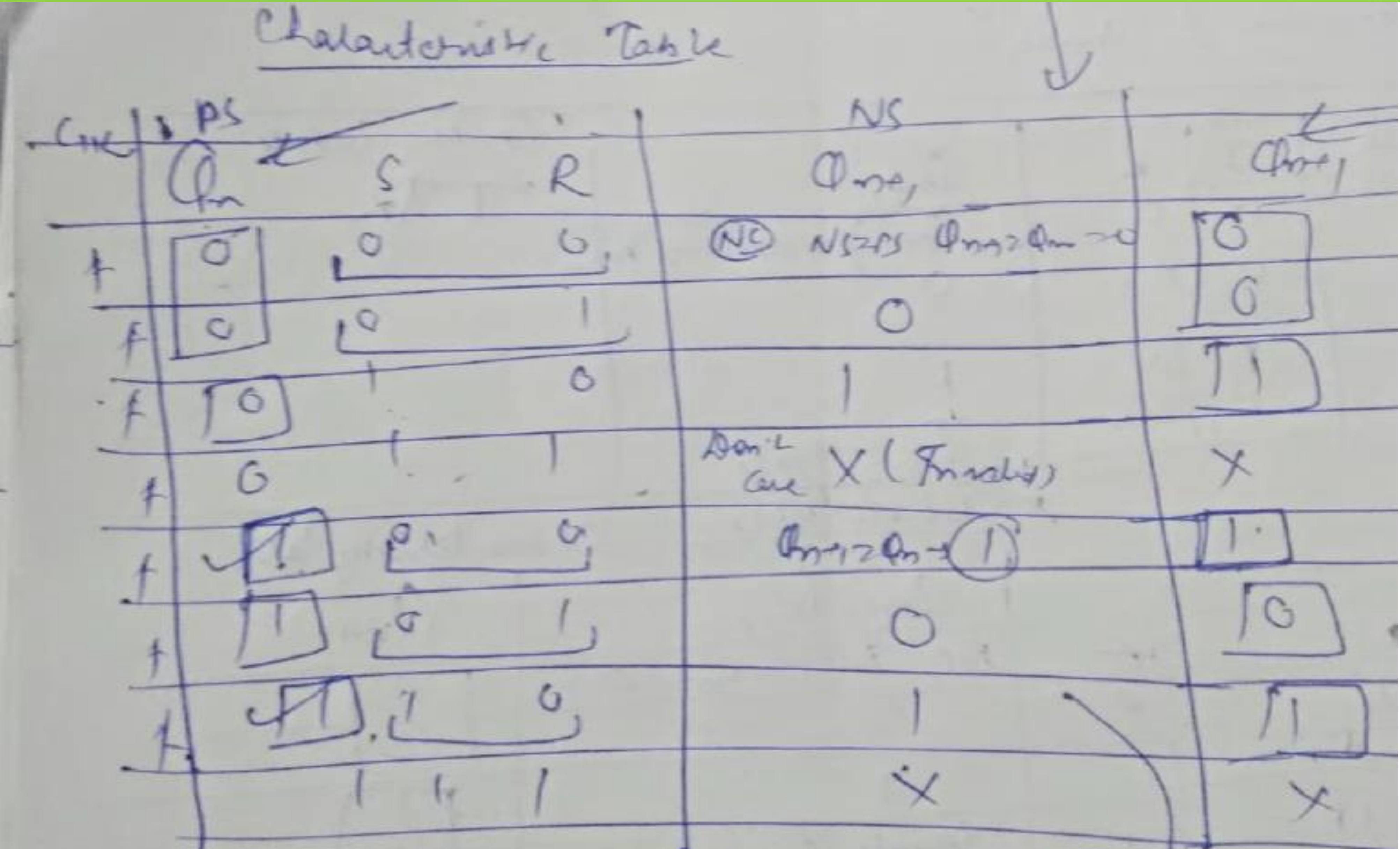
Truth Table, Characteristic Table and Excitation Table(SR Flip Flop)



Clk	S	R	NS Q_{n+1}
1	0	0	Q_n (PS)
1	0	1	0
1	1	0	1
1	1	1	Invalid

$\boxed{NS = PS}$

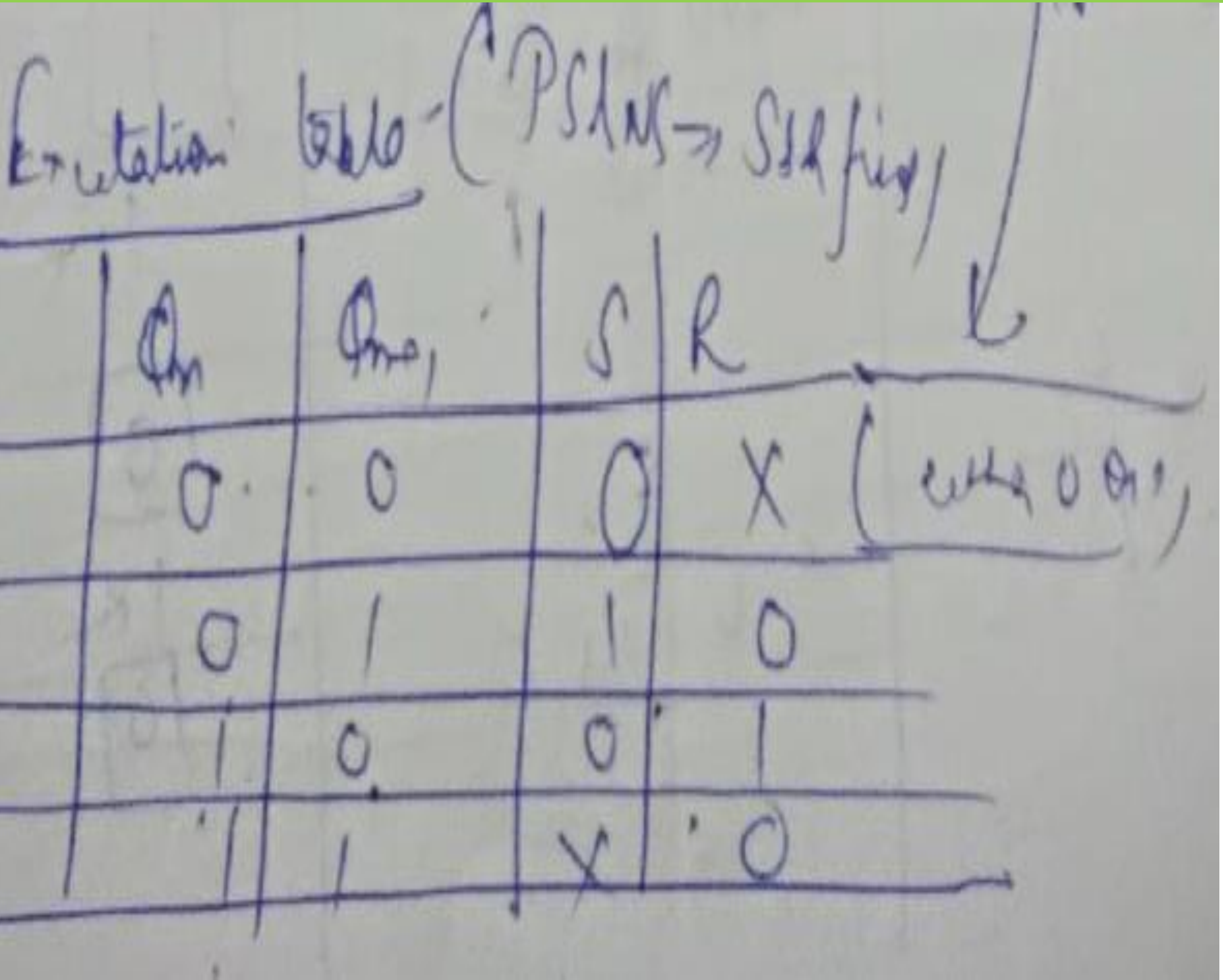
Characteristic Table(SR Flip-Flop)



Characteristic Table

Clk	PS Q_n	S	R	NS Q_{n+1}	Q_{n+1}
↑	0	0	0	NC NS=PS $Q_{n+1}=Q_n \to 0$	0
↑	0	0	1	0	0
↑	0	1	0	1	1
↑	0	1	1	Don't Care X (Invalid)	X
↑	1	0	0	$Q_{n+1}=Q_n \to 1$	1
↑	1	0	1	0	0
↑	1	1	0	1	1
	1	1	1	X	X

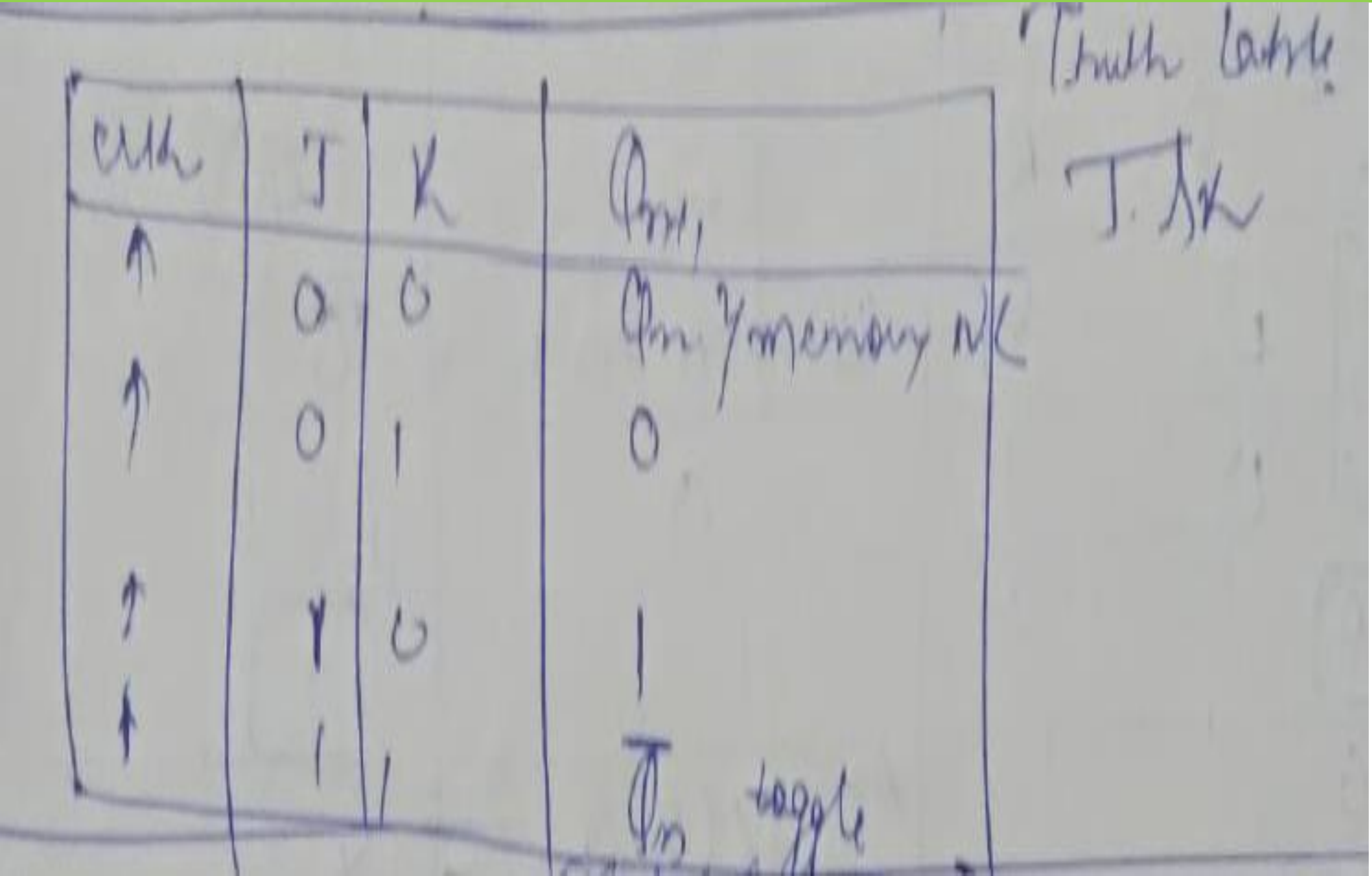
Excitation Table(SR Flip-Flop)



Excitation table (PS & NS ⇒ SR flip)

Q_n	Q_{n+1}	S	R	
0	0	0	X	(when 0 Q_n,
0	1	1	0	
1	0	0	1	
1	1	X	0	

Truth Table, Characteristic Table and Excitation Table(JK Flip Flop)

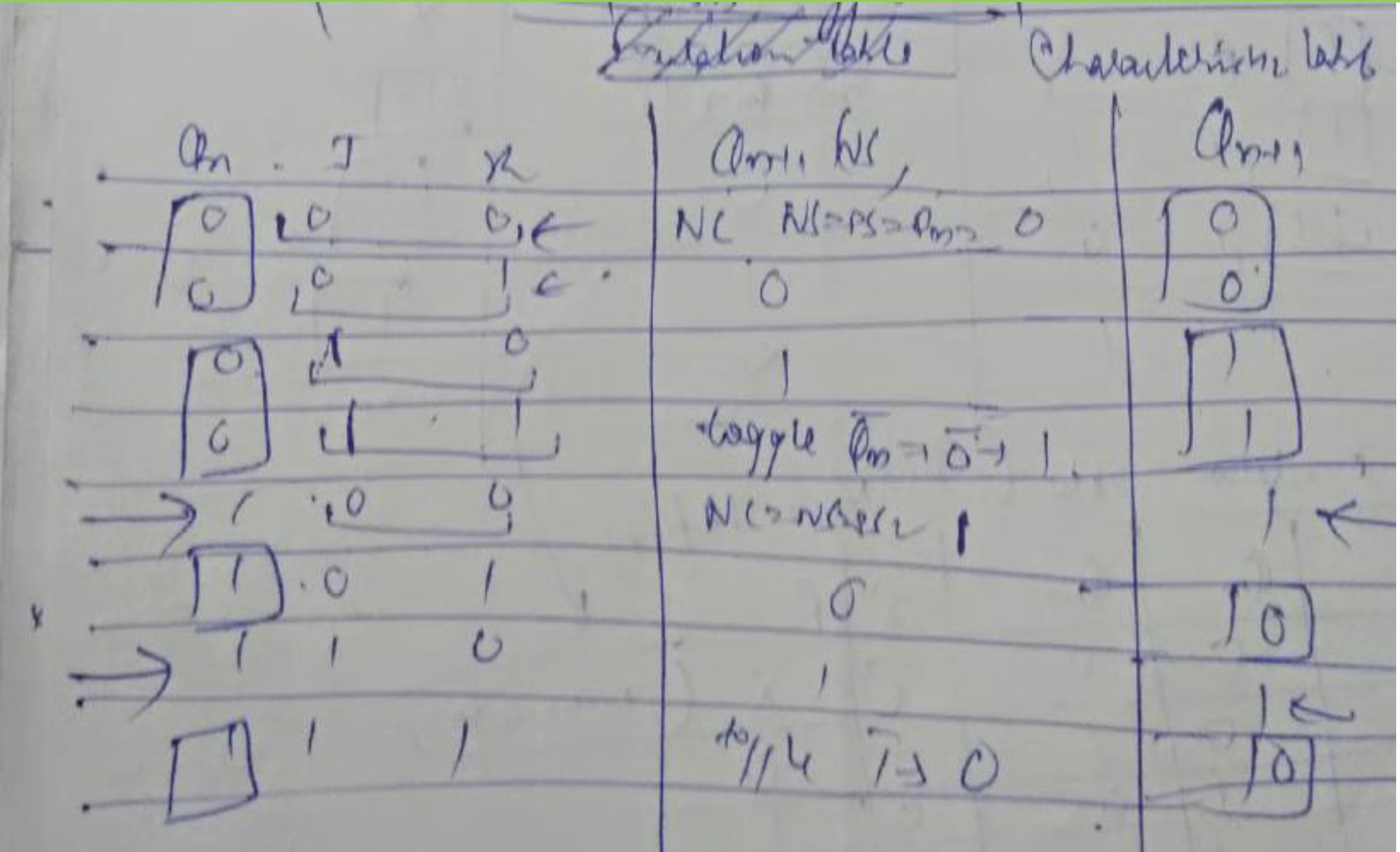


Truth table:

J.K

Clk	J	K	Q_{n+1}
↑	0	0	Q_n memory NC
↑	0	1	0
↑	1	0	1
↑	1	1	$\bar{Q}_n$ toggle

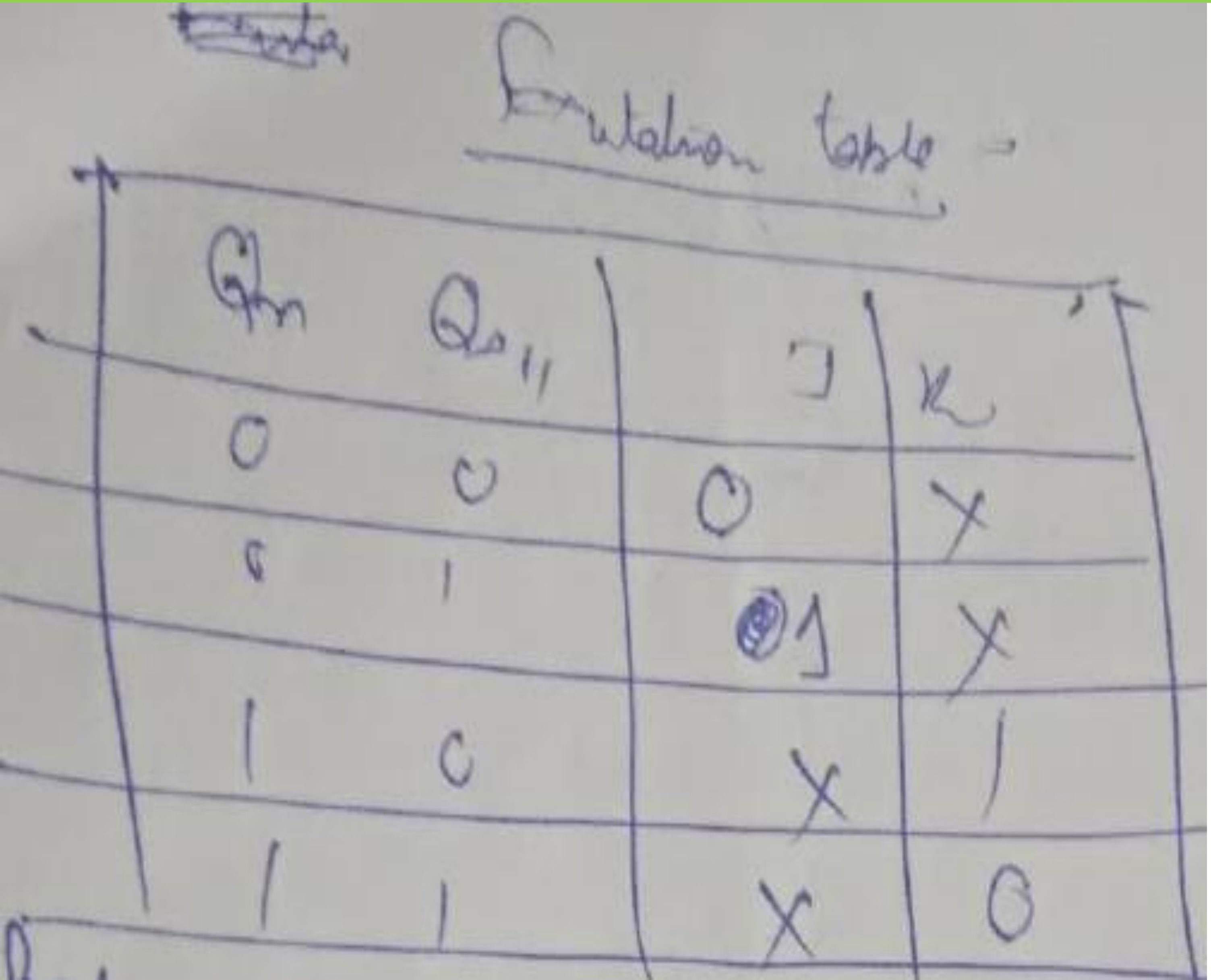
Characteristeristic Table(JK Flip Flop)



Excitation table | Characteristic table

Q_n	J	K	Q_{n+1} NS	Q_{n+1}
0	0	0	NC NS=PS=Q_n= 0	0
0	0	1	0	0
0	1	0	1	1
0	1	1	toggle $\bar{Q}_n \to \bar{0} \to 1$	1
1	0	0	NC⇒NS=PS 1	1
1	0	1	0	0
1	1	0	1	1
1	1	1	toggle $\bar{1} \to 0$	0

Excitation Table(JK Flip Flop)

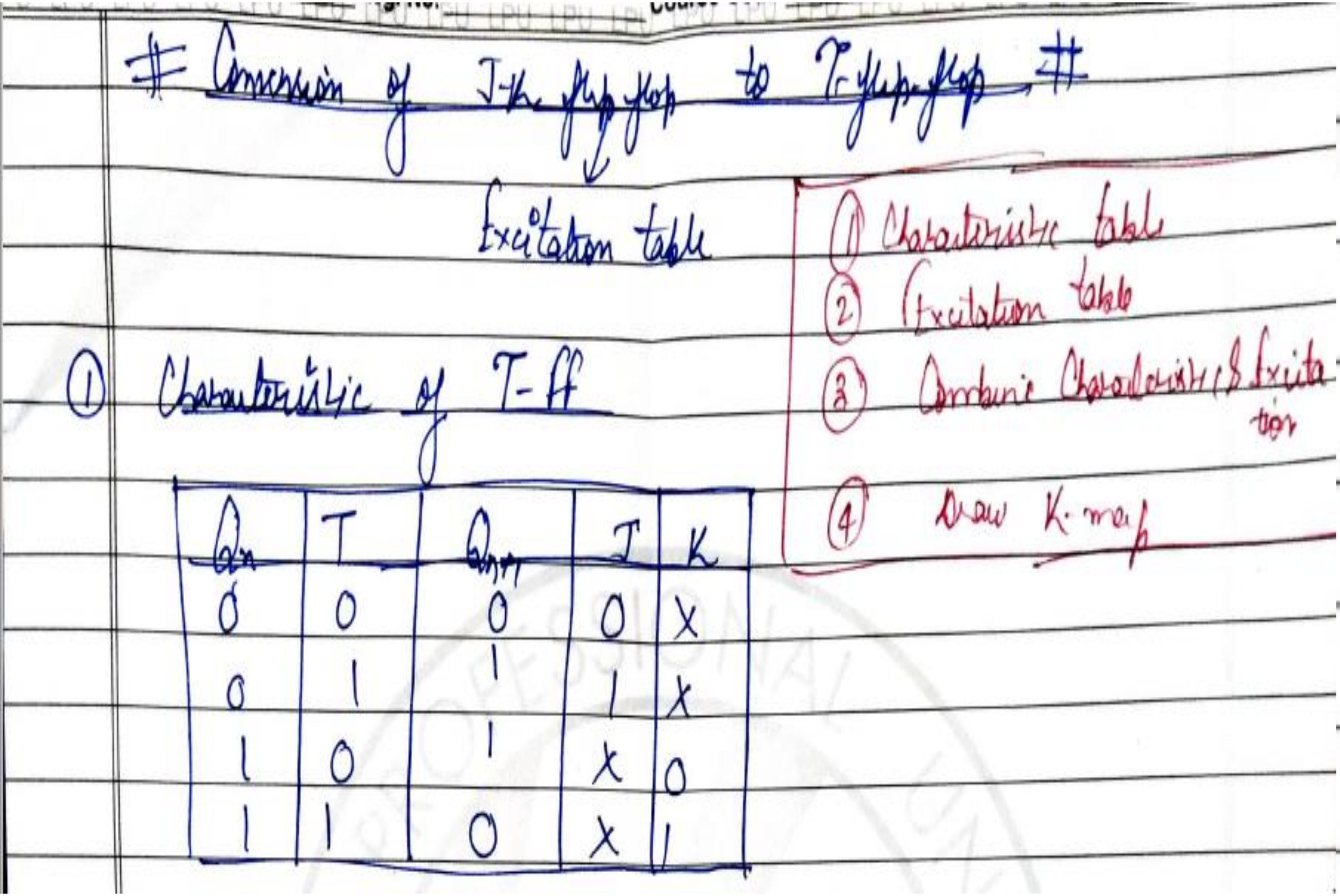


Excitation table →

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Conversion of Flip-Flops(JK Flip Flop to T Flip Flop)

J-K FLIP FLOP===EXCITATION TABLE ; T FLIP FLOP==CHARACTERISTIC TABLE



Conversion of J-K flip flop to T-flip-flop

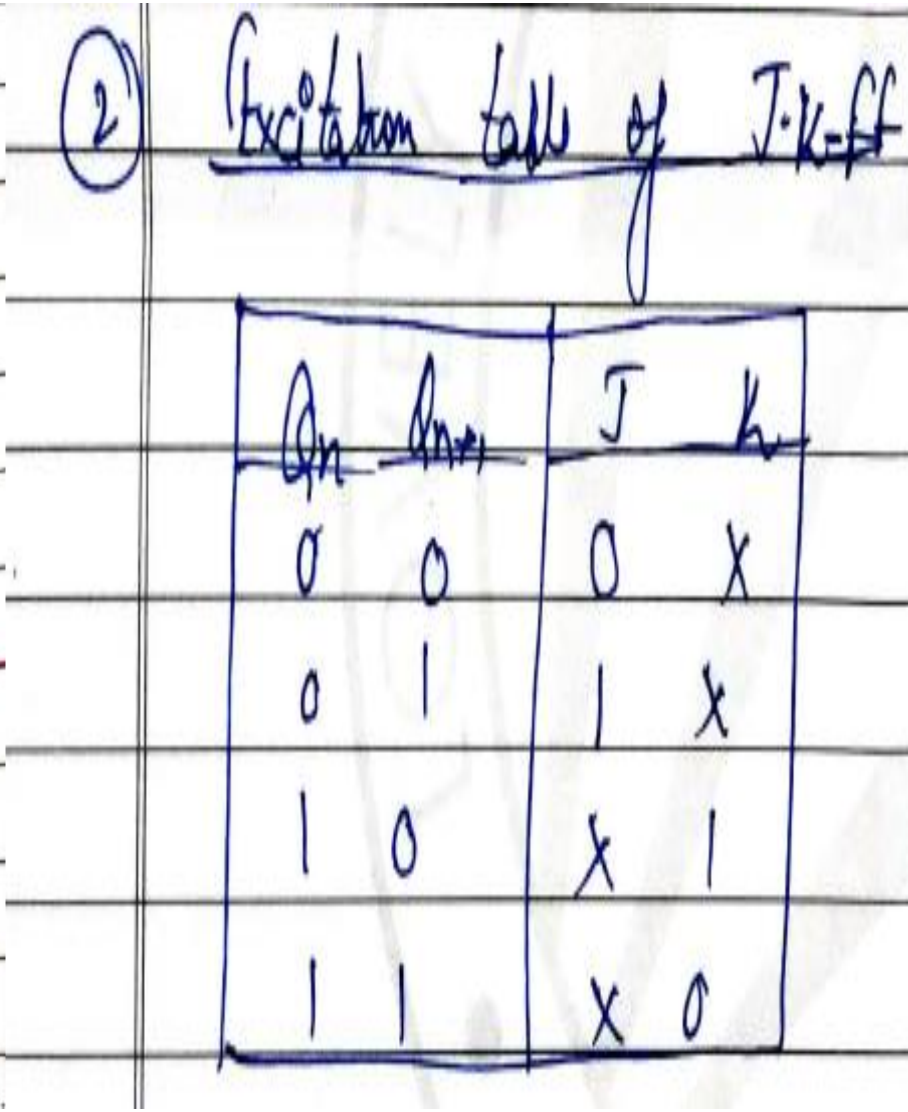
↓

Excitation table

① Characteristic table
② Excitation table
③ Combine Characteristic & Excitation
④ Draw K-map

① Characteristic of T-ff

Q_n	T	Q_{n+1}	J	K
0	0	0	0	X
0	1	1	1	X
1	0	1	X	0
1	1	0	X	1



② Excitation table of J-K-ff

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

(3) Draw K-map

Q_n \ T	0	1
0	0	1
1	X	X

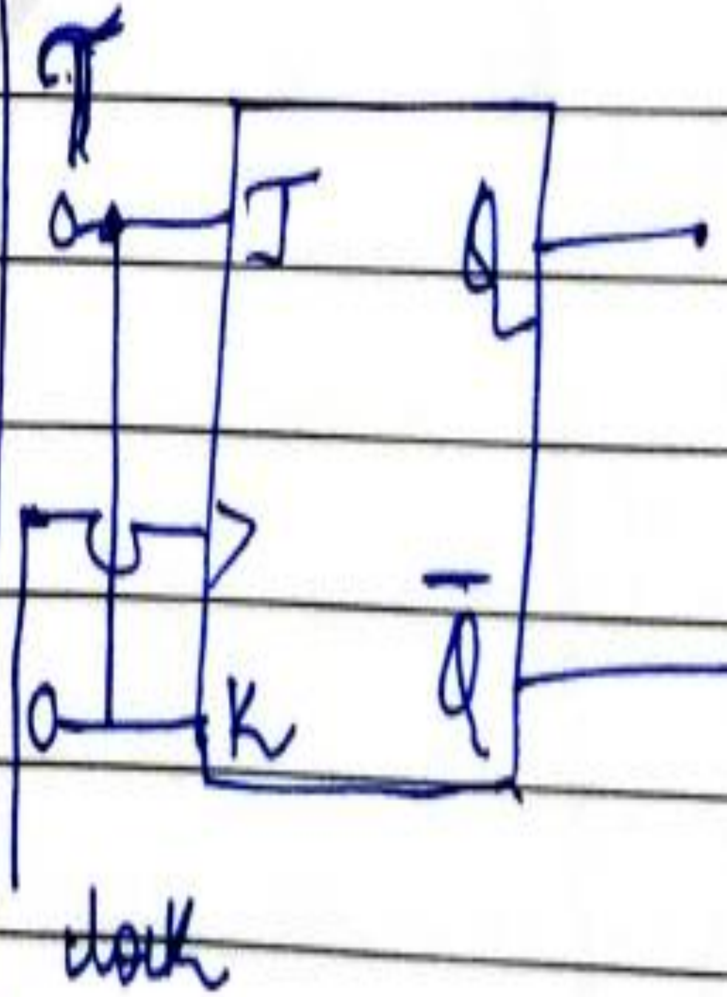
for J

J=T

For K

Q_n \ T	0	1
0	X	X
1	0	1

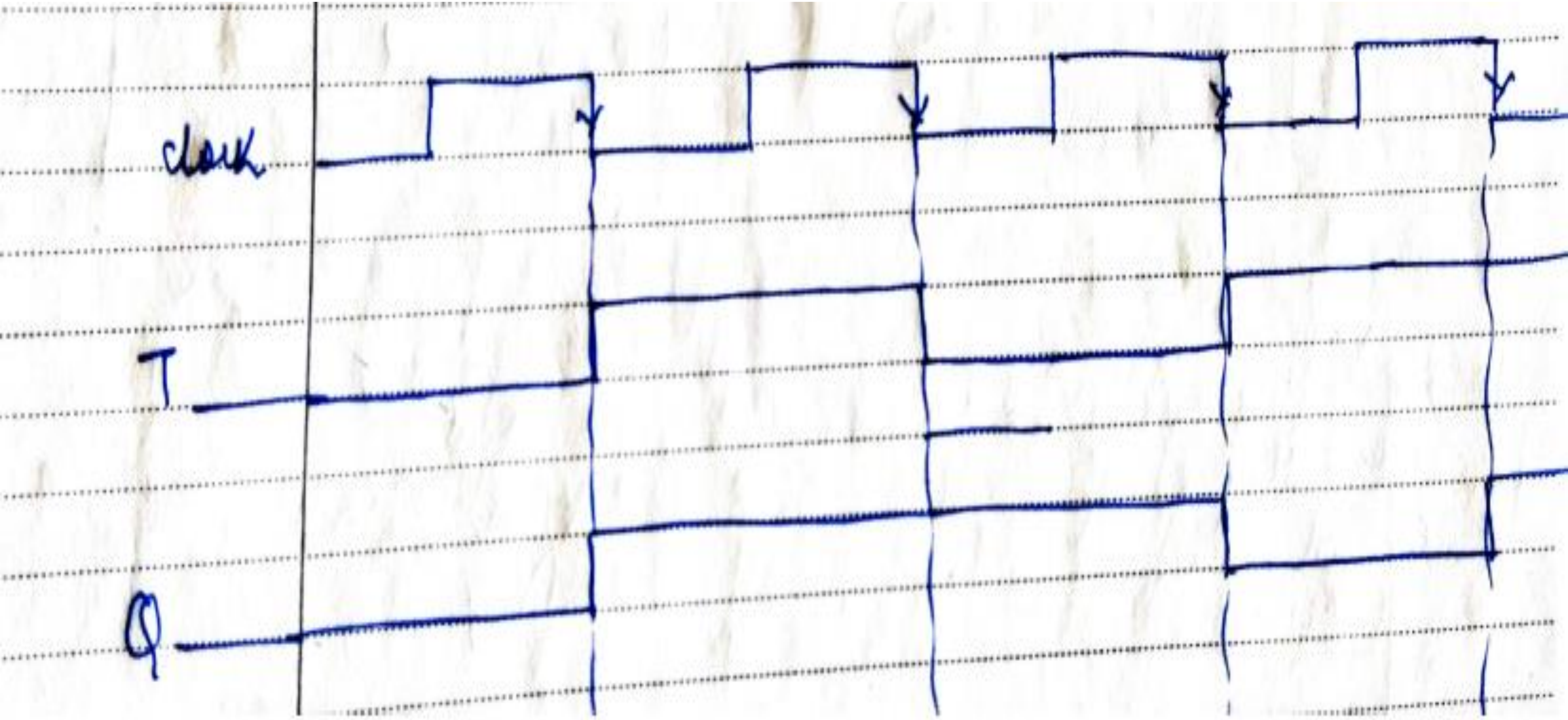
K=T



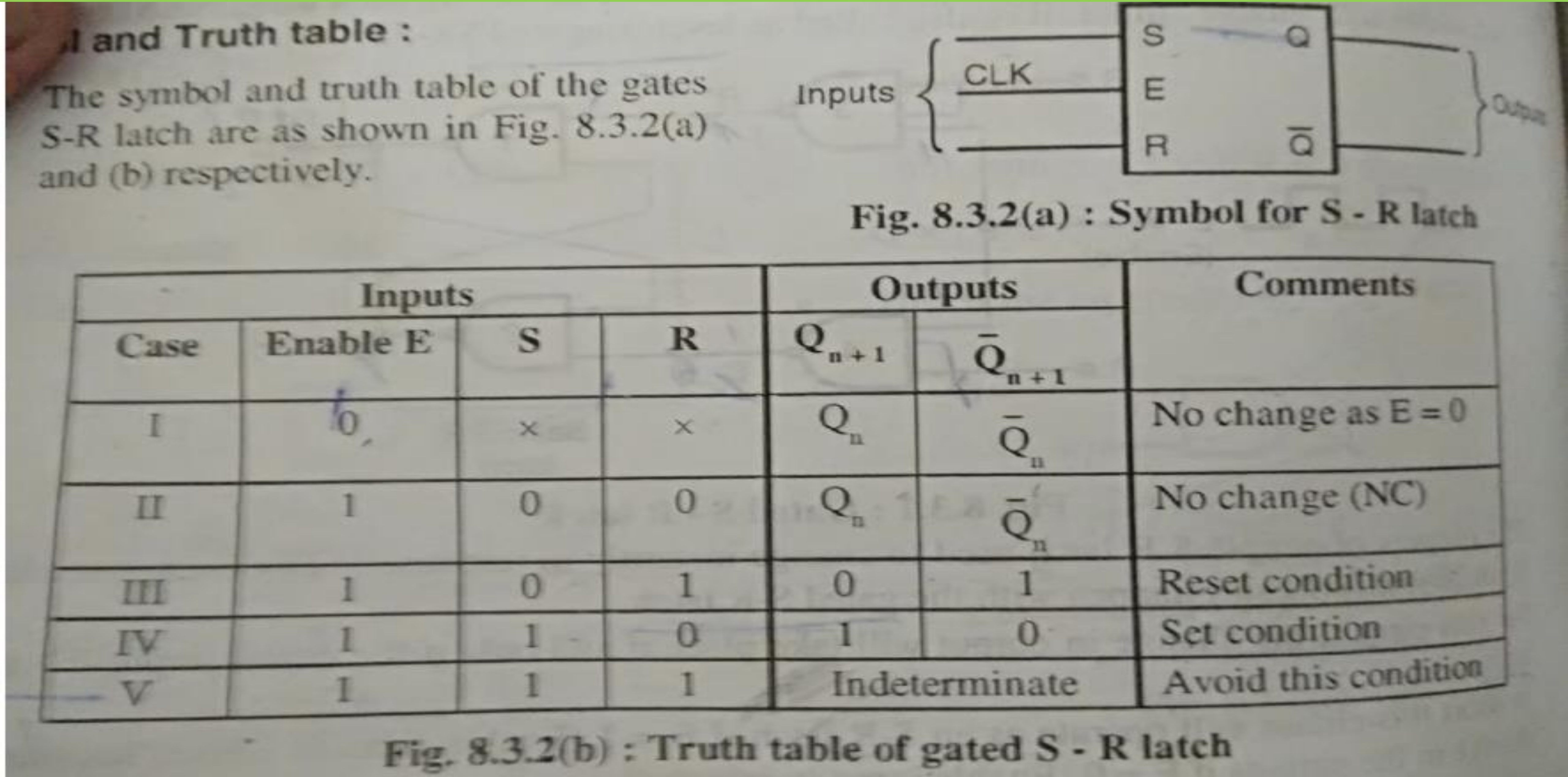
Diagram

*To be used fo

Draw the output waveform for the negative edge triggered T flip flop, if the clock and T input input waveforms are follows:



Gated SR Latch



l and Truth table :

The symbol and truth table of the gates S-R latch are as shown in Fig. 8.3.2(a) and (b) respectively.

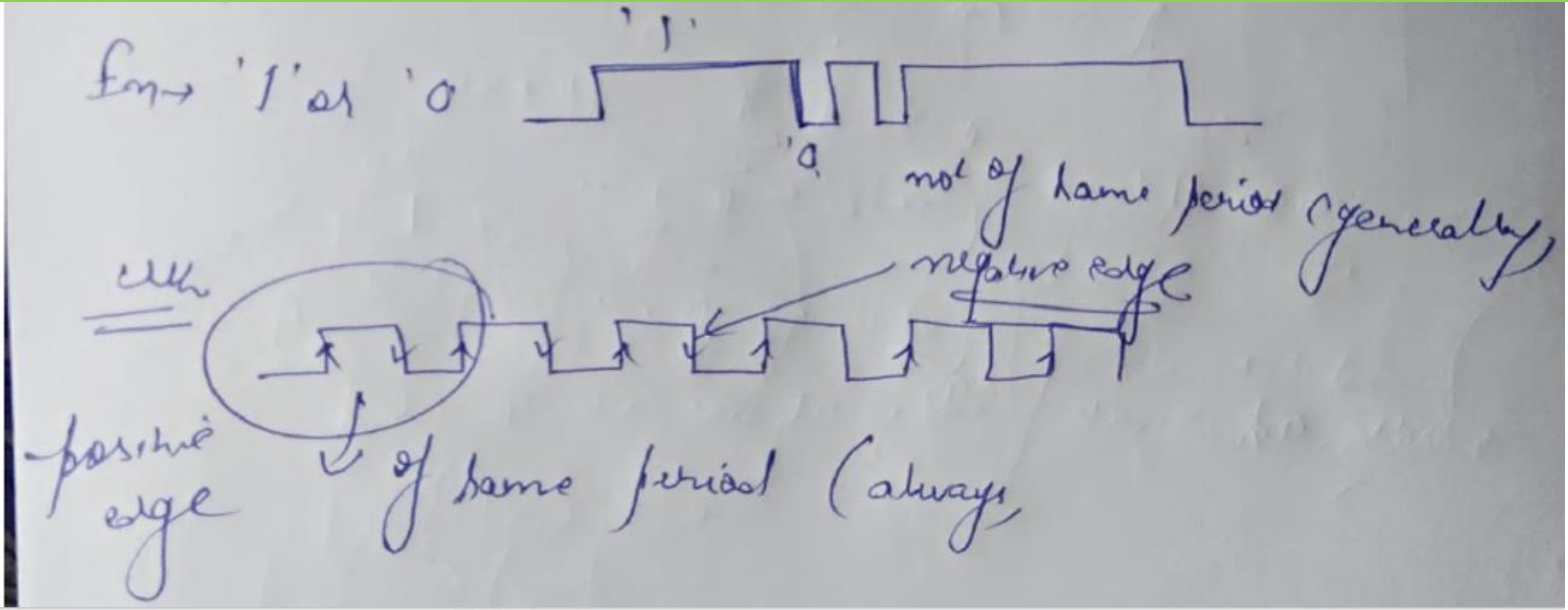
Inputs
CLK
S
E
R
Q
$\bar{Q}$
Outputs

Fig. 8.3.2(a) : Symbol for S - R latch

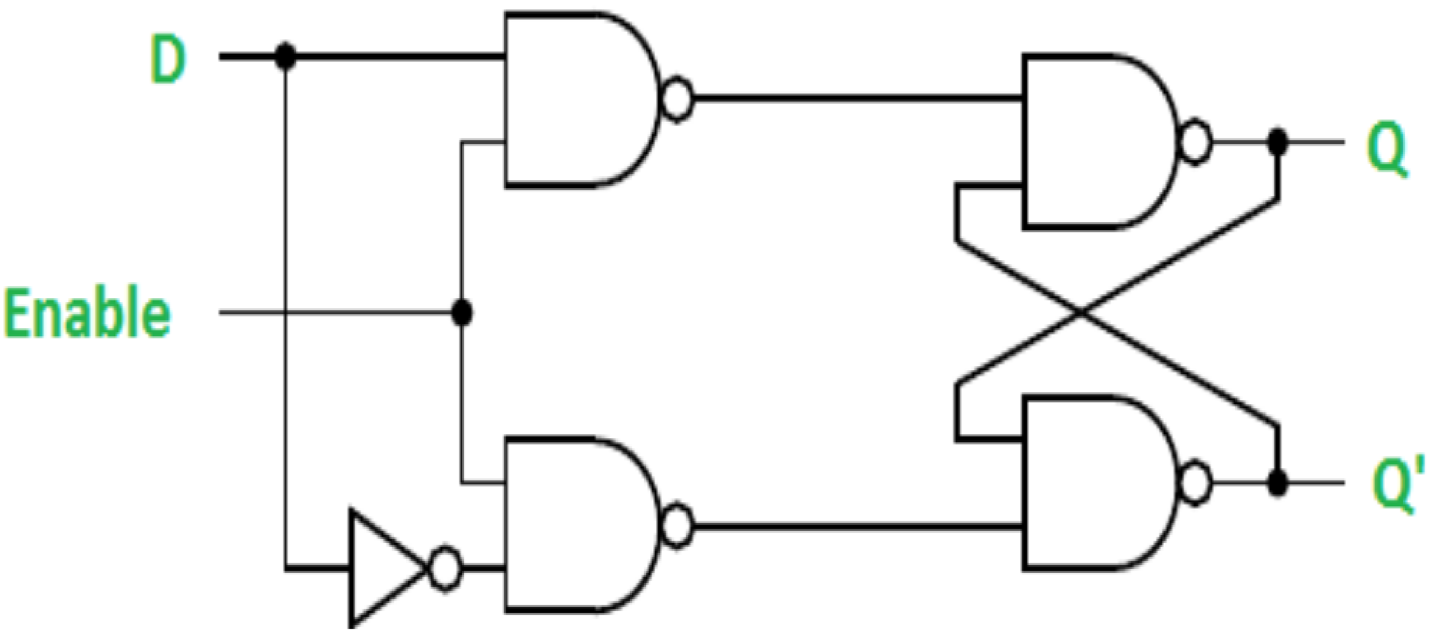
Inputs				Outputs		Comments
Case	Enable E	S	R	Q_{n+1}	$\bar{Q}_{n+1}$	
I	0	×	×	Q_n	$\bar{Q}_n$	No change as E = 0
II	1	0	0	Q_n	$\bar{Q}_n$	No change (NC)
III	1	0	1	0	1	Reset condition
IV	1	1	0	1	0	Set condition
V	1	1	1	Indeterminate		Avoid this condition

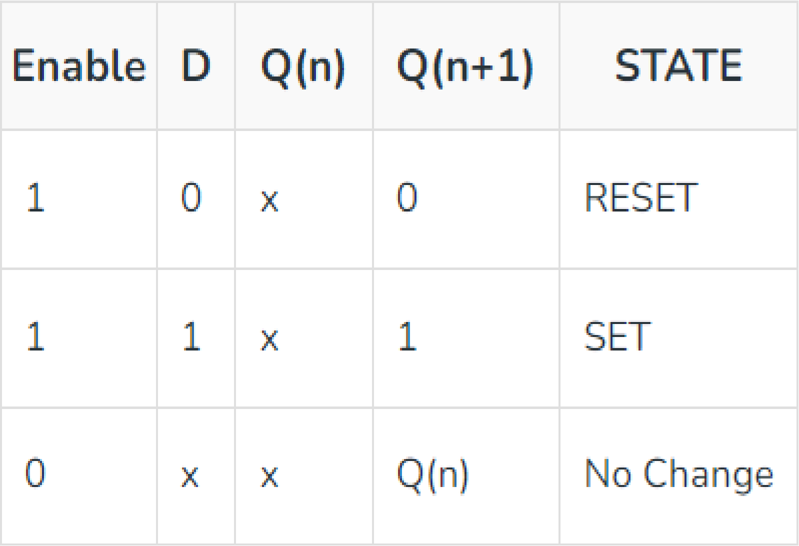
Fig. 8.3.2(b) : Truth table of gated S - R latch

Gated D Latch



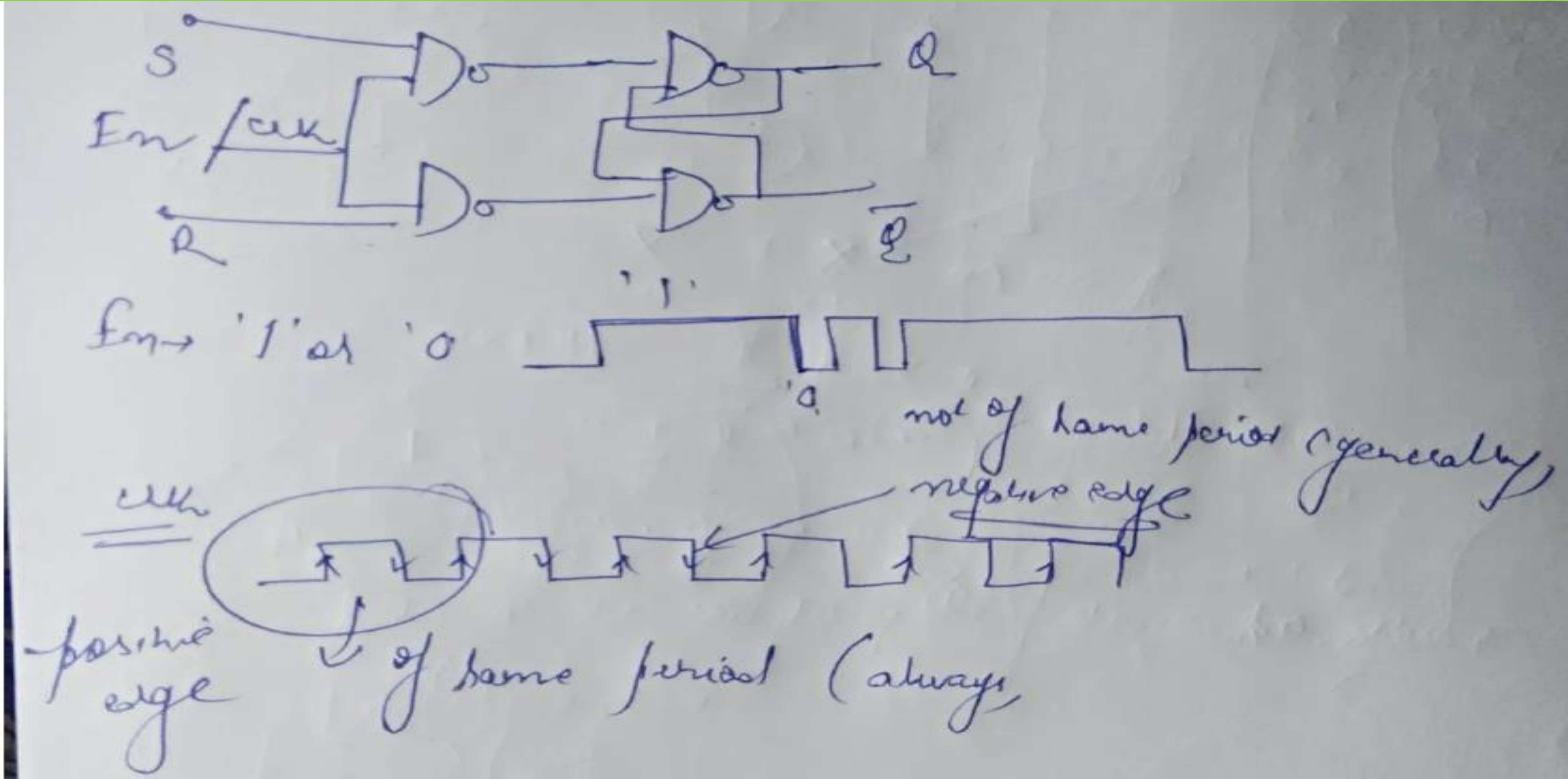
Gated D Latch



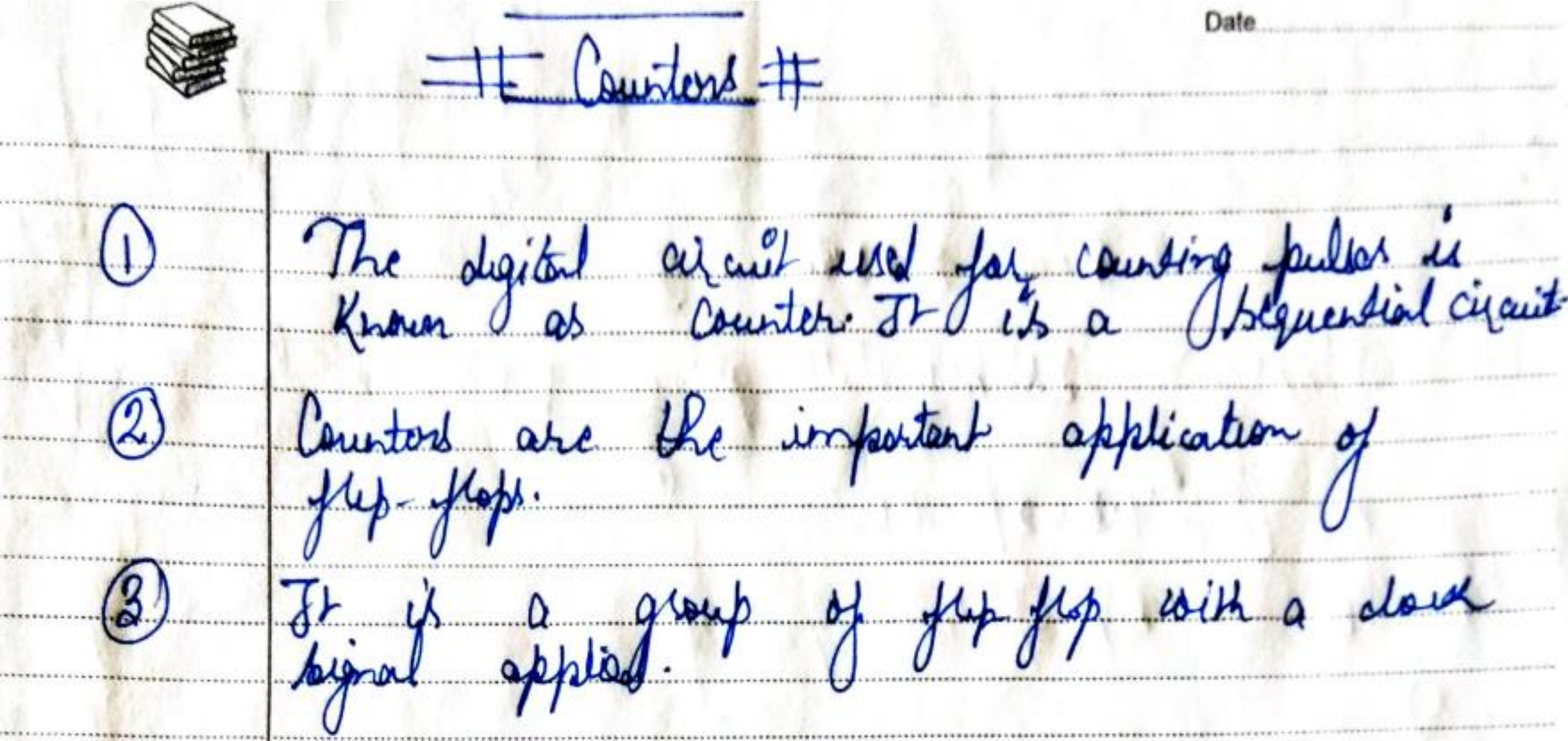


Enable	D	Q(n)	Q(n+1)	STATE
1	0	x	0	RESET
1	1	x	1	SET
0	x	x	Q(n)	No Change

Gated SR Latch



Counter

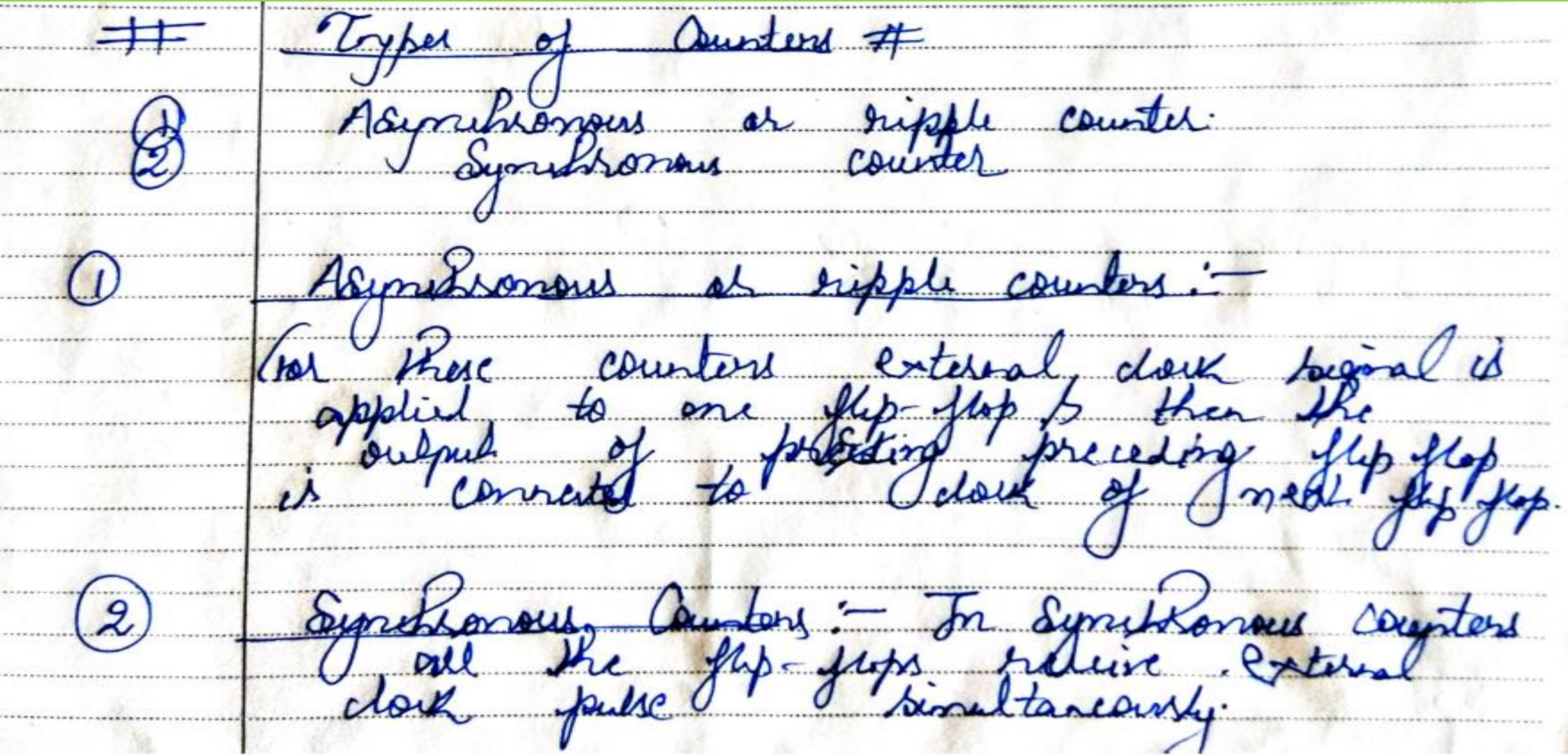


Counters

Date

1. The digital circuit used for counting pulses is Known as Counter. It is a sequential circuit.
2. Counters are the important application of flip-flops.
3. It is a group of flip flop with a clock signal applied.

Counter



# Types of Counters #

(1) Asynchronous or ripple counter

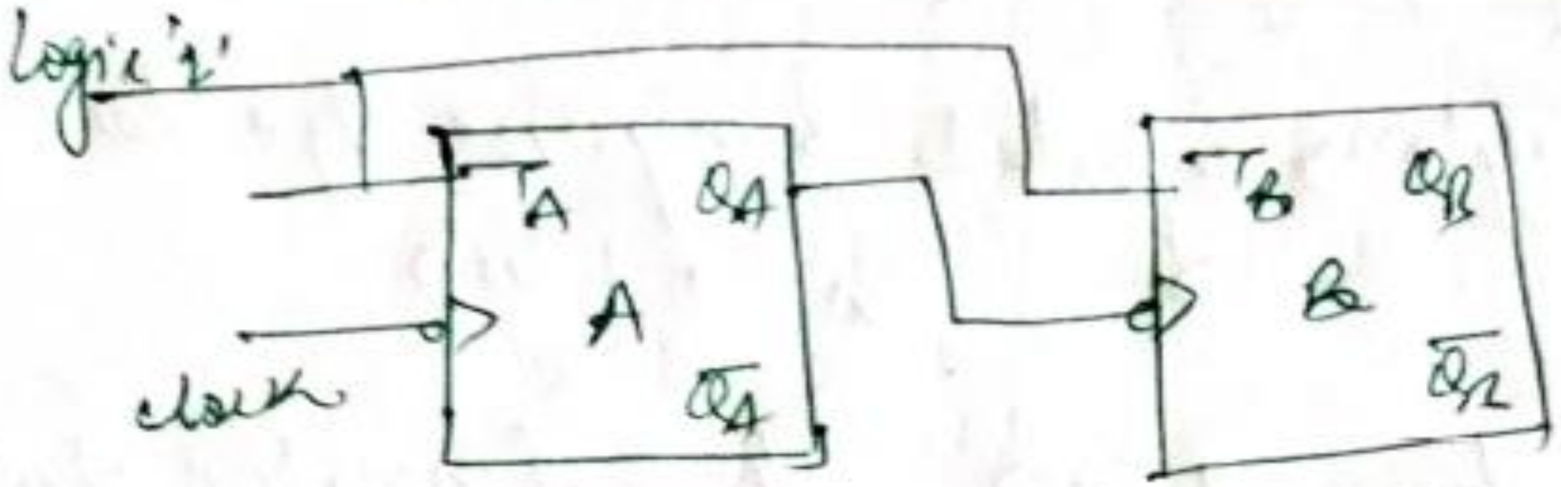
(2) Synchronous counter

(1) Asynchronous or ripple counters:-

In these counters external clock signal is applied to one flip-flop & then the output of ~~presiding~~ preceding flip flop is connected to clock of next flip flop.

(2) Synchronous Counters:- In synchronous counters all the flip-flops receive external clock pulse simultaneously.

2 Bit Asynchronous Counters (Ripple Counter) # (1)



A two bit synchronous binary counter

(1) In the given fig, the no. of flip-flops are 2.
(2) Thus, the no. of bits will always be equal to no of flip flop.

(1) In the given fig, the no. of flip-flops are 2.

(2) Thus, the no. of bits will always be equal to no. of flip-flop.

(3) External clock is applied to the clock input of first flip flop & Q_A (output) is applied to clock input of next flip-flop.

Operation:- (1) Initially both flip-flops are in reset condition

$\boxed{Q_B Q_A = 00}$

(1) On first negative edge

(1) FF-A → input 1 → Toggle → Q_A from 0 to 1.

(2) Q_A is connected as clock input to FF-B & Q_A is changing from 0 to 1

positive edge. So no change in output of $Q_B = 0$

$\boxed{Q_B Q_A = 01}$

On 2nd falling edge of clock

(1) a, ff-A again toggle → input 1 & negative edge clock
→ $Q_A \rightarrow$ 1 to 0

b) Q_A changes from 1 to 0 ‾|_ {negative edge}

c) Hence Q_B changes (toggle) ⇒ $Q_B \Rightarrow$ 0 to 1

$$\boxed{Q_B Q_A = 10}$$

On 3rd falling edge of clock

① a) On 3rd neg edge of clock → ff-A toggle from 0 to 1 $\boxed{Q_A = 1}$

b) $Q_A \; 0 \nearrow 1$ positive edge

c) No change in $Q_B = 1$

$$\boxed{Q_B Q_A = 11}$$

On 4th negative clock edge

① ff-A toggle (input 1) & Q_A changes from 1 to 0

② $Q_A \to 1 \searrow 0$ negative edge clock $\underline{Q_A = 0}$

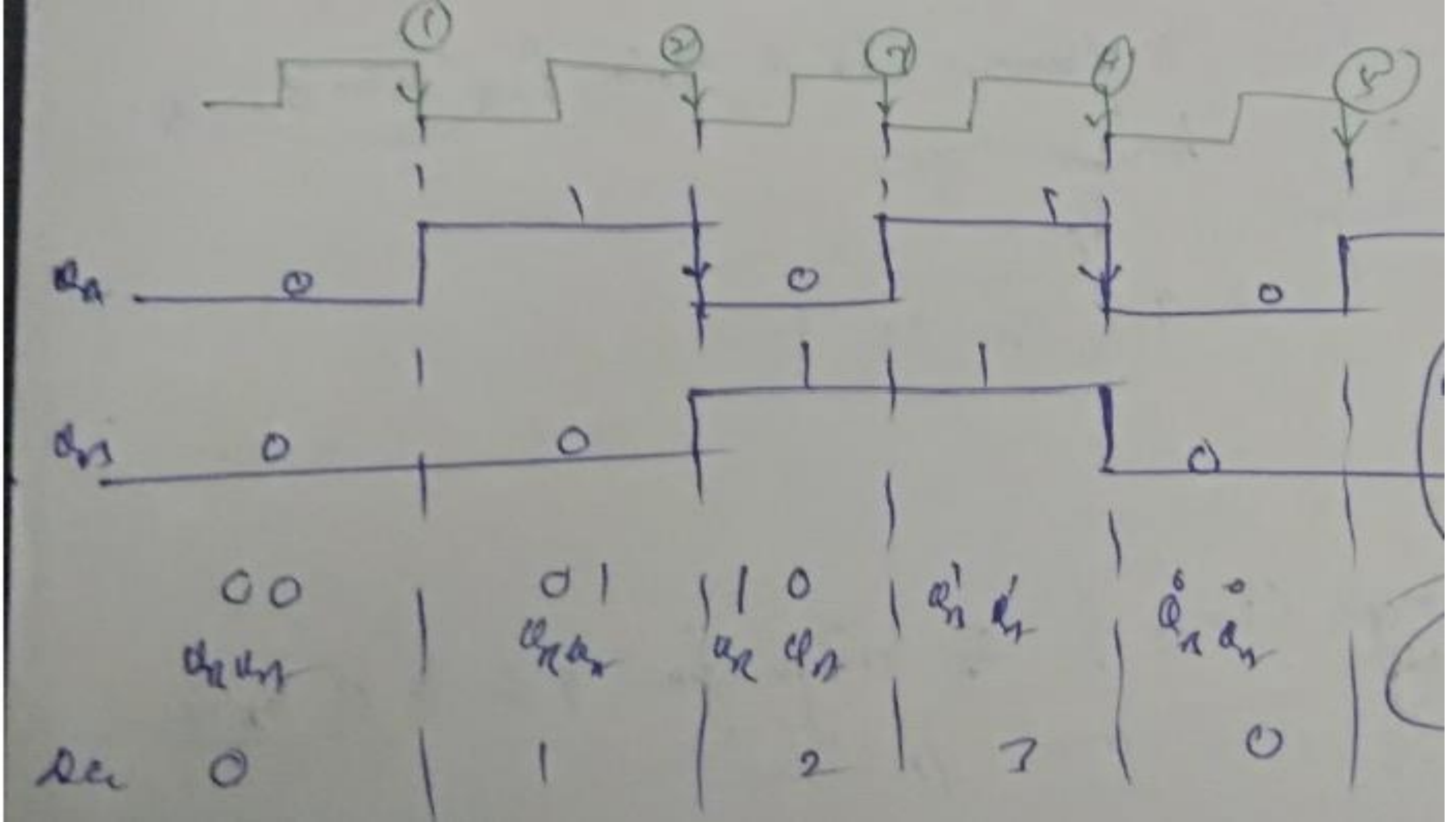
③ Q_B toggle from 1 to 0 $\boxed{Q_B = 0}$

$$\boxed{Q_B Q_A = 00}$$

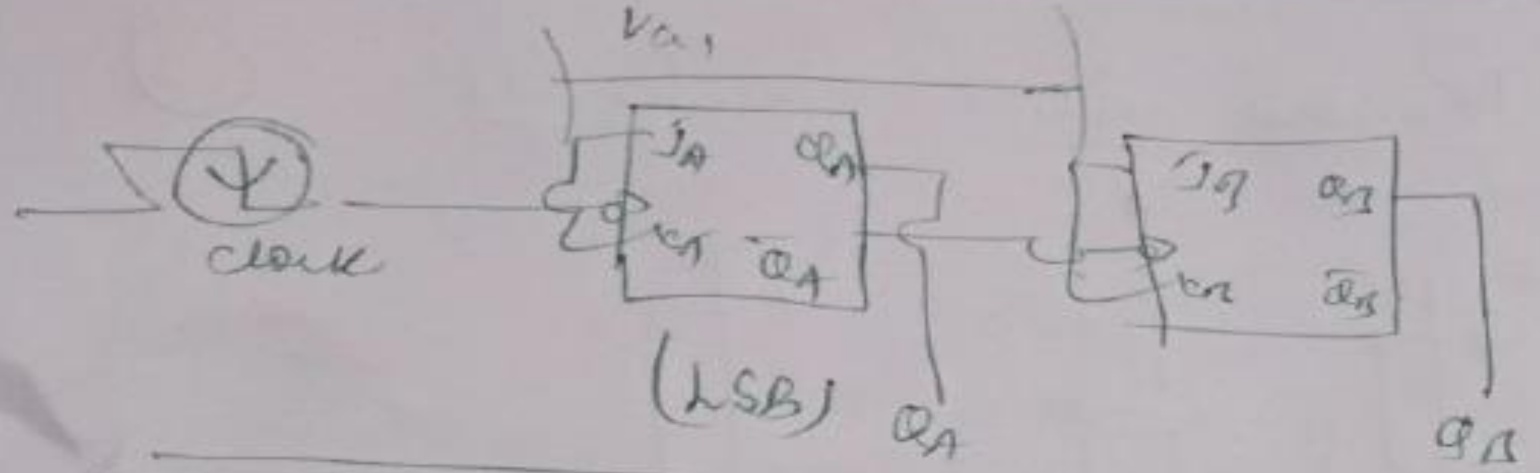
No. of States = 2^n (n no. of flip flops)

Maximum count = $2^n - 1$ ($2^2 - 1 = 3$)

Clock	Q_B	Q_A (LSB)	Dec
initially	0	0	0
1st ↓	0	1	1
2nd ↓	1	0	2
3rd ↓	1	1	3
4th ↓	0	0	0



1
2
3
4
5
QA
0
1
0
1
0
QB
0
0
1
1
0
0 0
QA QB
0 1
QA QB
1 0
QA QB
1 1
QA QB
0 0
QA QB
Dec
0
1
2
3
0

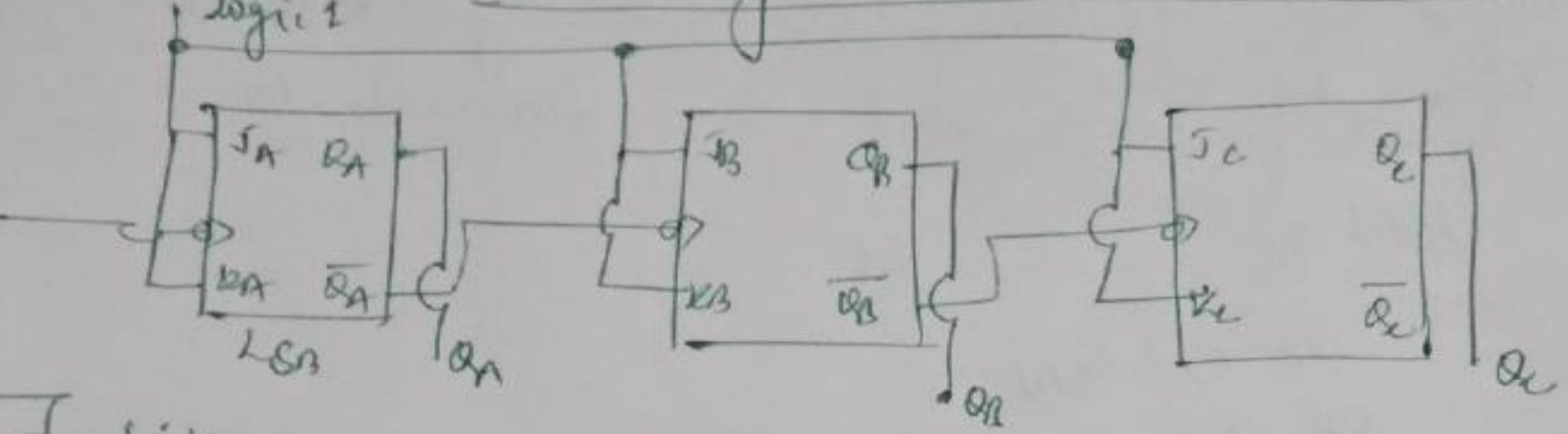


2 Bit asynchronous down Counter

Truth

	Q_B	Q_A (LSB)		
0	0	0		
Clock 1 →	1	1	3	{ Q_A → 0 to 1 ↑ $\bar{Q}_A$ → 1 to 0 ↓ neg }
clock → 2 →	1	0	2	{ Q_A → 1 to 0 ↓ $\bar{Q}_A$ → 0 to 1 ↑
clock → 3 →	0	1	1	{ Q_A → 0 to 1 ↑ $\bar{Q}_A$ → 1 to 0 ↓ }
clock → 4 →	0	0	0	{ Q_A → 1 to 0 ↓ $\bar{Q}_A$ → 0 to 1 ↑ }

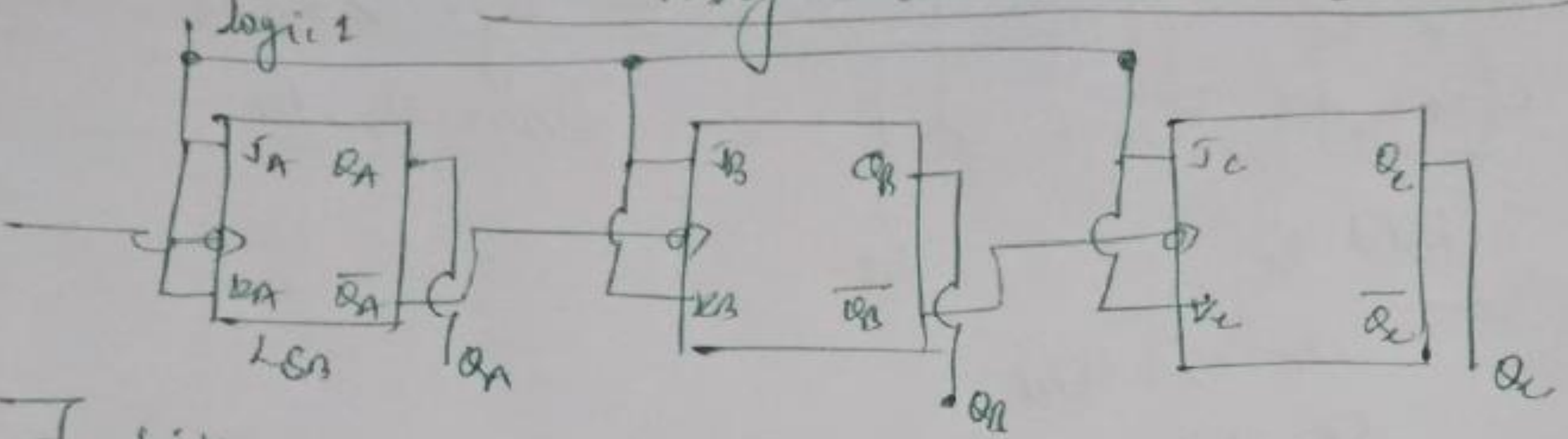
3 bit Asynchronous down Counter



Initially

Q_C	Q_B	Q_A
0	0.	0

3 bit Asynchronous down Counter



Initially

Q_C Q_B Q_A

0 0 0

clock 1 →
(7)

Q_C Q_B Q_A

1 1 1

{ Q_A → 0 to 1 ↑
{ $\overline{Q_A}$ → 1 to 0 ↓

{ Q_B → 0 to 1 ↑
{ $\overline{Q_B}$ → 1 to 0 ↓

{ Q_C → 0 to 1 ↑
{ $\overline{Q_C}$ → 1 to 0 ↓

clock 2 → (6, Q_C Q_B Q_A
1 1 0

{ $Q_A \to$ 1 to 0 ; $\overline{Q_A} \to$ 0 to 1 NC }
$Q_B \to$ NC
$Q_C \to$ NC

clock 3 → (5, Q_C Q_B Q_A
1 0 1

{ $Q_A \to$ 0 to 1 ; $\overline{Q_A} \to$ 1 to 0 .
$Q_B \to$ 1 to 0 , $\overline{Q_B} \to$ 0 to 1. ⌐ }
$Q_C \to$ NC

clock → 4 (4, Q_C Q_B Q_A
1 0 0

{ $Q_A \to$ 1 to 0 ; $\overline{Q_A} \to$ 0 to 1 ⌐ NC
Q_B NC ; $Q_C \to$ NC

clock 5 → (3, Q_C Q_B Q_A
0 1 1.

{ $Q_A \to$ 0 to 1 $\overline{Q_A} \to$ 1 to 0 ⌐
$Q_B \to$ 0 to 1 , $\overline{Q_B} \to$ 1 to 0 ⌐
$Q_C \to$ 1 to 0

clock 6 → (2, Q_C Q_B Q_A
0 1 0

{ Q_A 1 to 0 $\overline{Q_A} \to$ 0 to 1
Q_B NC | Q_C = NC

clock 5 → 13, Q_C Q_B Q_A 0 1 1. { Q_A → 0 to 1 $\overline{Q_A}$ → 1 to 0 ₹ ; Q_B → 0 to 1; $\overline{Q_B}$ → 1 to 0 ₹ ; Q_C → 1 to 0

clock 6 → 12, Q_C Q_B Q_A 0 1 0 { Q_A 1 to 0 $\overline{Q_A}$ → 0 to 1 ; Q_B NC | Q_C = NC

clock 7 11, Q_C Q_B Q_A 0 0 1.

clock 8 10 Q_C Q_B Q_A 0 0 0

Modulus of the Counter # (1)

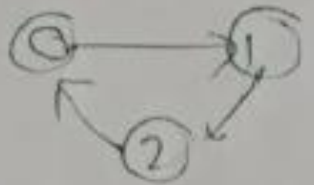
(1) The 2 bit ripple counter is called mod-4 & 3 bit ripple counter is called mod-8.

So n bit ripple counter is called modulo N counter.

$$\boxed{\text{mod no} = 2^n}$$

Q1. Design a mod-3 asynchronous counter using a 2 bit ripple counter ?

Soln:- mod-3 counter is a counter having three states 00, 01, 10. After 10 it will be: reduced back to 00.



(*) As soon as flip flop reaches state 2, the reset logic should produce 0 at o/p & clear all the ff so jump back to 0 state.

Q_1	Q_0	Y o/p
0	0	1
0	1	1
1	0	1
1.	1	0

} valid state

→ clear all ff

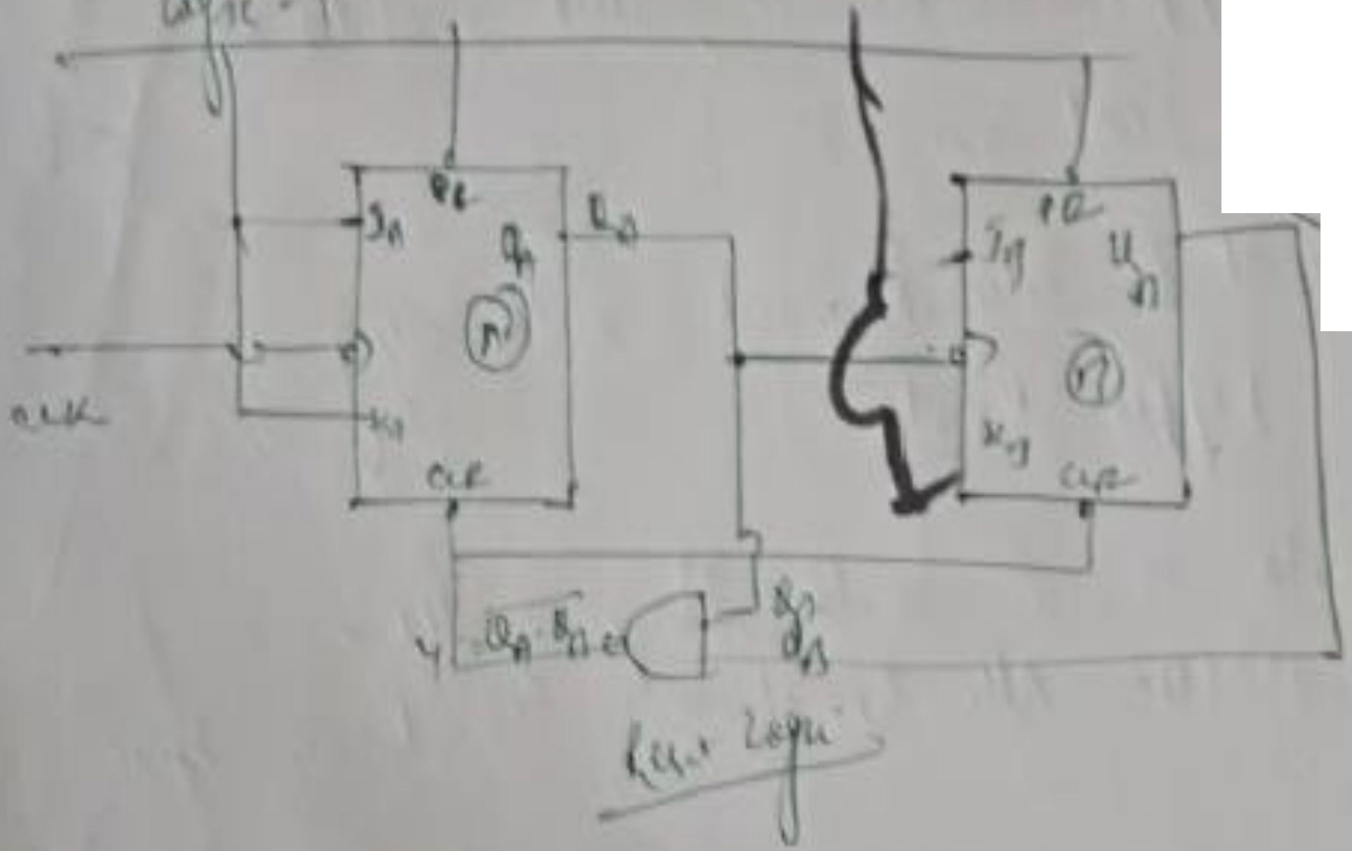
$Q_A \backslash Q_B$	0	1
0	1	1
1	1	0

$\overline{Q_B}$

$\overline{Q_A}$

$Y = \overline{Q_B} + \overline{Q_A}$

$$\boxed{Y = \overline{Q_A \cdot Q_B}}$$ NAND



Mod 6 Asynchronous Counter

Clock	Q_A	Q_B	Q_C	Count	Y
↓	0	0	0	0	1
↓	0	0	1	1	1
↓	0	1	0	2	1
↓	0	1	1	3	1
↓	1	0	0	4	1
↓	1	0	1	5	1
↓	1	1	0	6	0
↓	1	1	1	7	0
↓					

Q_A \ Q_BQ_C	00	01	11	10
0	1	1	1	1
1	1	1	0	0

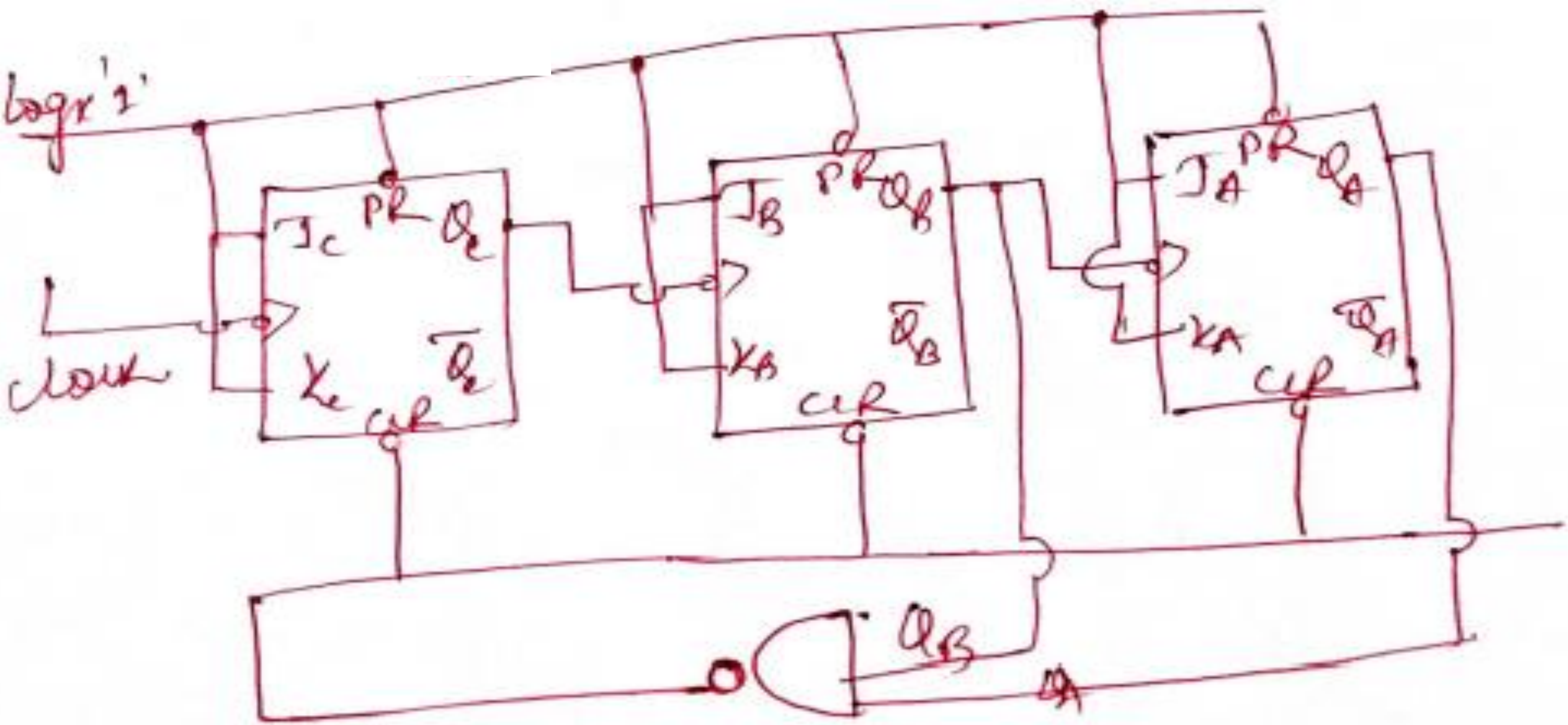
→ $\bar{Q}_A$

↓ $\bar{Q}_B$

$$Y = \bar{Q}_A + \bar{Q}_B$$

$$\boxed{Y \Rightarrow \overline{Q_A \cdot Q_B}}$$

NAND gate


logic '1'
clock
J_C
PR
Q_C
K_C
$\overline{Q_C}$
CLR
J_B
PR
Q_B
K_B
$\overline{Q_B}$
CLR
J_A
PR
Q_A
K_A
$\overline{Q_A}$
CLR
Q_B
Q_A