

University of Technology
Computer Science Department



Computer Organization and Logic Design

تركيب الحاسوب والتصميم المنطقي
(عملي)

Stage: 1st Class

Year: 2024-2025

م.د أحمد عبد الزهرة شكارة



cs.uotechnology.edu.iq

Part Three: Logic Design

(عملي)

ملاحظة: يبدأ العملي بعد الانتهاء من شرح محاضرة النظري (أنواع الأنظمة الرقمية)
(Types of number systems)

برنامج CEDAR Logic Simulator

هو البرنامج المستخدم للتدريب على بناء الدوائر المنطقية

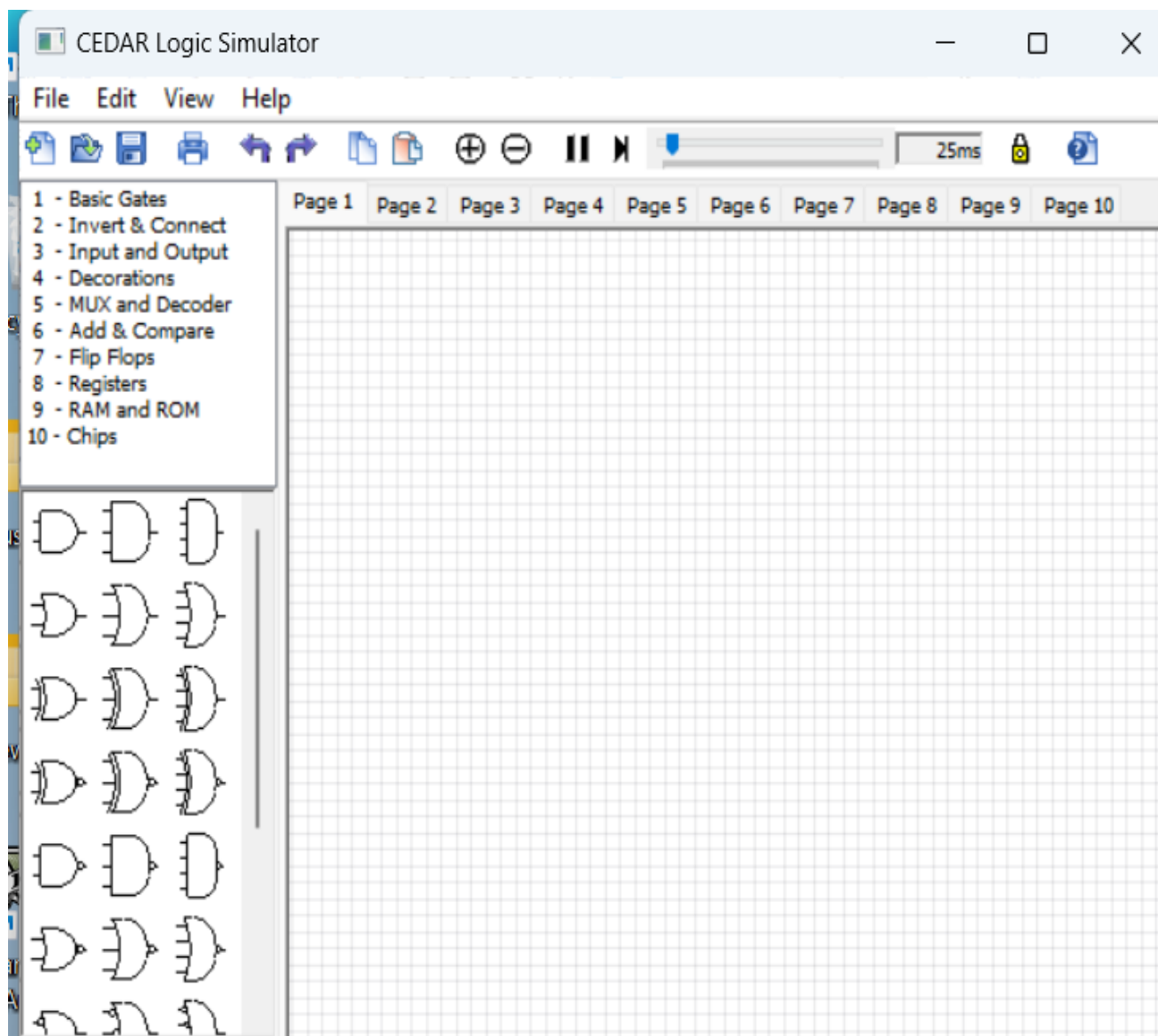


Figure 1: CEDAR Logic Simulator

Electronic and Logic Circuits

Modern devices contain two types of circuits:

Electronic circuits

The basic components of electronic circuits are transistors, resistors, capacitors, etc. Electronic circuits operate on a wide range of voltages such as (1V, 2.1V, 3.3V, 12V) positive or negative and deal with analog signals.

Logic Circuits

The basic component of digital circuits are logic gates such as AND, OR, NAND, NOR, XOR, XNOR, NOT... which deal with digital signals.

Integrated Circuits (IC)

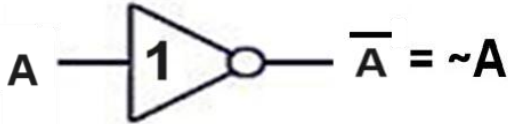
Integrated circuits consist of logic and electronic circuits built on a single small block or chip of semiconductor that all work together to perform a specific task. The IC is easily breakable, so to be attached to a circuit board, it is often housed in a plastic package with metal pins.

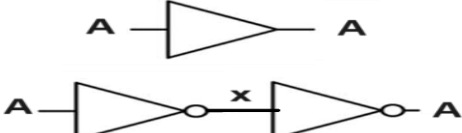

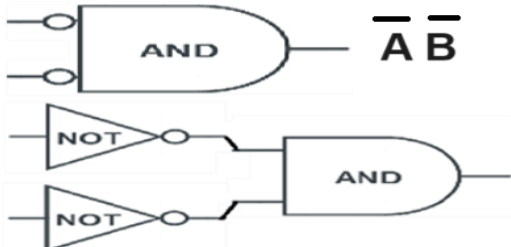
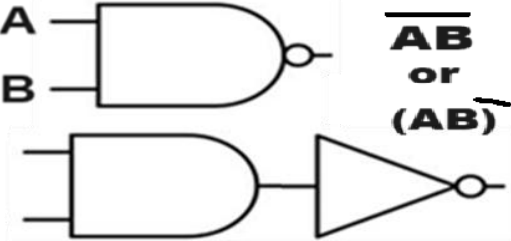
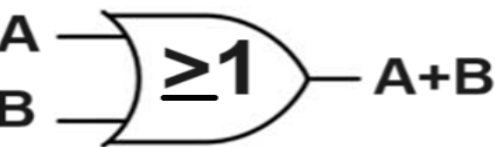
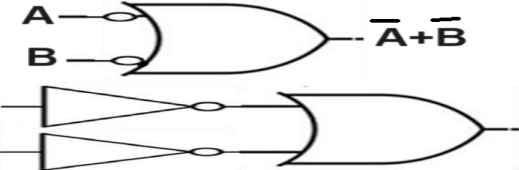
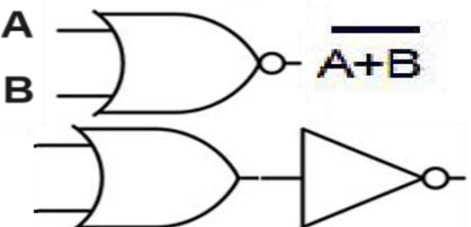


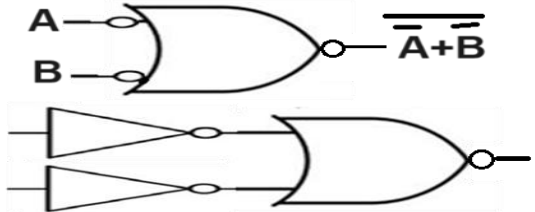
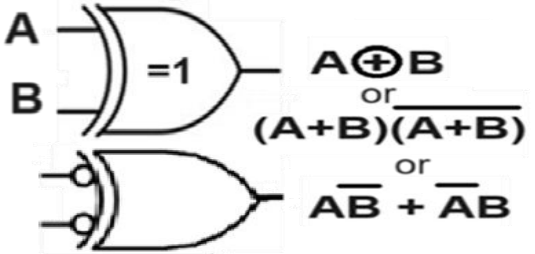
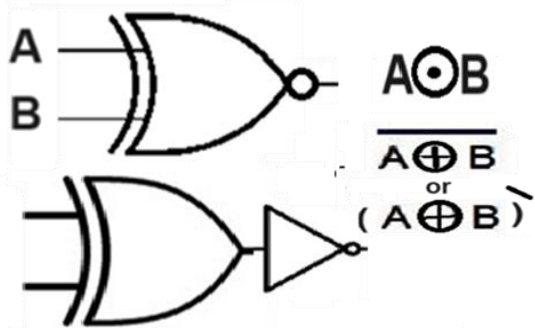
Figure 2: Integrated Circuits (IC)

Logic Gates

The set of logic gates is:

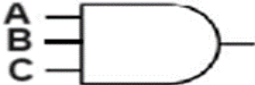

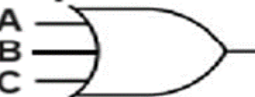
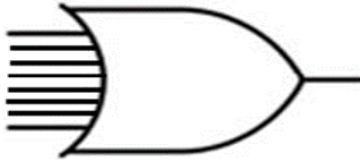
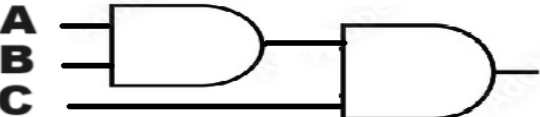

Name	Graphic symbols	Truth Tables						
NOT (Inverter) Complement		<table border="1"><thead><tr><th>A</th><th>\bar{A}</th></tr></thead><tbody><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></tbody></table>	A	\bar{A}	0	1	1	0
A	\bar{A}							
0	1							
1	0							

Buffer		<table border="1" data-bbox="1045 199 1302 331"> <thead> <tr> <th>A</th> <th>A</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	A	0	0	1	1									
A	A																
0	0																
1	1																
AND		<table border="1" data-bbox="1045 346 1404 504"> <thead> <tr> <th>A</th> <th>B</th> <th>AB</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	AB	0	0	0	1	0	0	0	1	0	1	1	1
A	B	AB															
0	0	0															
1	0	0															
0	1	0															
1	1	1															
Negative AND		<table border="1" data-bbox="1045 520 1396 777"> <thead> <tr> <th>A</th> <th>B</th> <th>$\overline{A} \overline{B}$</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	$\overline{A} \overline{B}$	0	0	1	1	0	0	0	1	0	1	1	0
A	B	$\overline{A} \overline{B}$															
0	0	1															
1	0	0															
0	1	0															
1	1	0															
NAND		<table border="1" data-bbox="1045 802 1372 1050"> <thead> <tr> <th>A</th> <th>B</th> <th>\overline{AB}</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	\overline{AB}	0	0	1	1	0	1	0	1	1	1	1	0
A	B	\overline{AB}															
0	0	1															
1	0	1															
0	1	1															
1	1	0															
OR		<table border="1" data-bbox="1045 1071 1404 1234"> <thead> <tr> <th>A</th> <th>B</th> <th>A+B</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	A+B	0	0	0	0	1	1	1	0	1	1	1	1
A	B	A+B															
0	0	0															
0	1	1															
1	0	1															
1	1	1															
Negative OR		<p data-bbox="1045 1255 1380 1417">Same NAND (De Morgan's theorem)</p>															
NOR		<p data-bbox="1045 1455 1380 1654">Same Negative AND (De Morgan's theorem)</p>															

Negative NOR		Same AND AND (De Morgan's theorem)															
XOR Exclusive OR Ex-OR		<table border="1" data-bbox="1040 438 1409 690"> <thead> <tr> <th>A</th> <th>B</th> <th>$A \oplus B$</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	$A \oplus B$	0	0	0	0	1	1	1	0	1	1	1	0
A	B	$A \oplus B$															
0	0	0															
0	1	1															
1	0	1															
1	1	0															
XNOR Exclusive NOR		<table border="1" data-bbox="1040 711 1409 1037"> <thead> <tr> <th>A</th> <th>B</th> <th>$A \odot B$</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	$A \odot B$	0	0	1	0	1	0	1	0	0	1	1	1
A	B	$A \odot B$															
0	0	1															
0	1	0															
1	0	0															
1	1	1															

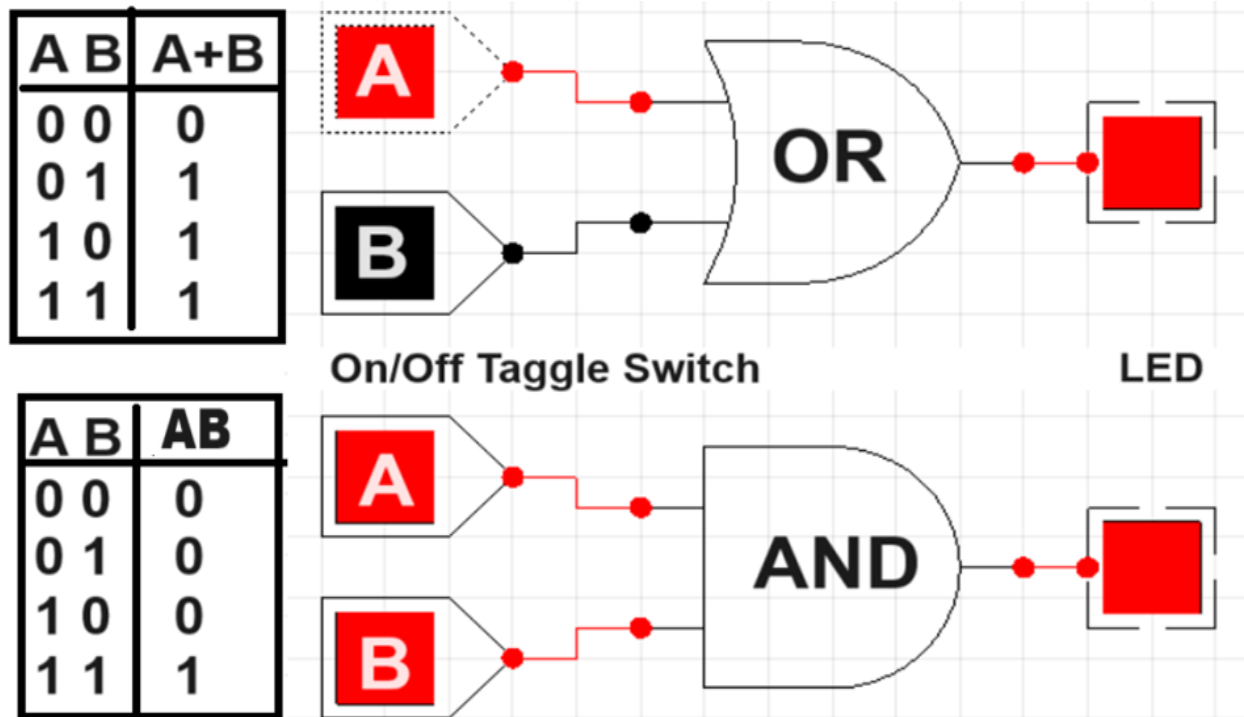
Note:

The number of inputs bits for gates ranged from 1 to 8-bits (1byte).

3-input AND 	<table border="1" data-bbox="548 1213 784 1388"> <thead> <tr> <th>A</th> <th>B</th> <th>C</th> <th>ABC</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>:</td> <td>:</td> <td>:</td> <td>:</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	C	ABC	0	0	0	0	0	0	1	0	:	:	:	:	1	1	1	1	8-input (1-byte) 
A	B	C	ABC																			
0	0	0	0																			
0	0	1	0																			
:	:	:	:																			
1	1	1	1																			
3-input OR 	<table border="1" data-bbox="548 1413 800 1598"> <thead> <tr> <th>A</th> <th>B</th> <th>C</th> <th>A+B+C</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>:</td> <td>:</td> <td>:</td> <td>:</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	C	A+B+C	0	0	0	0	0	0	1	1	:	:	:	:	1	1	1	1	
A	B	C	A+B+C																			
0	0	0	0																			
0	0	1	1																			
:	:	:	:																			
1	1	1	1																			
3-input AND 	3-input OR 																					

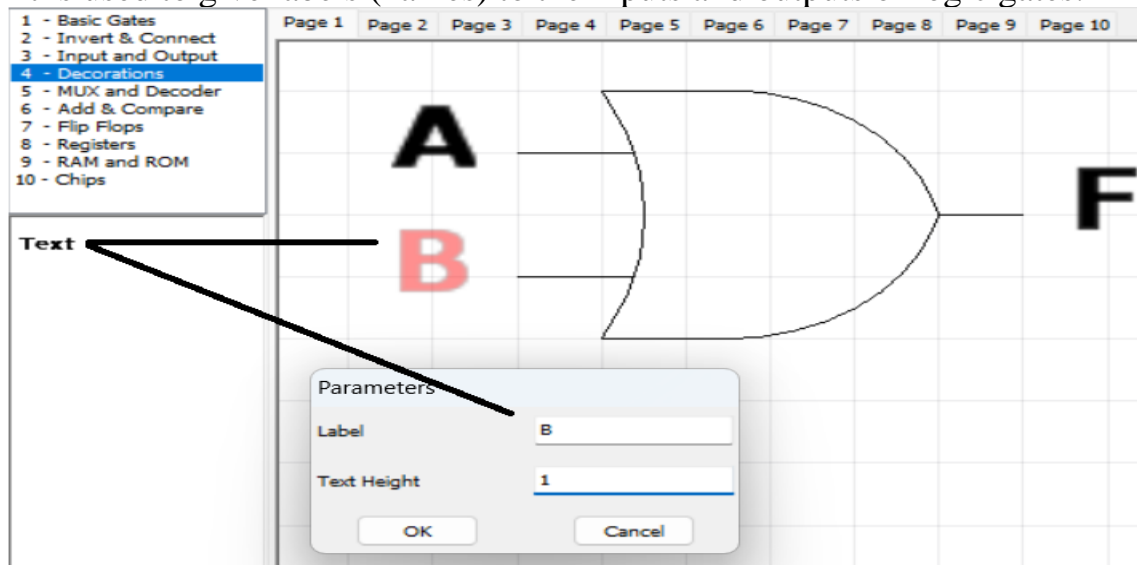
ملاحظة: للتأكد من بناء أي دائرة منطقية (logic circuit) بشكل صحيح يتم تنفيذ الجدول المنطقي عليها (Truth Table)

Example 1: Design logic circuit diagrams for OR & AND gates.

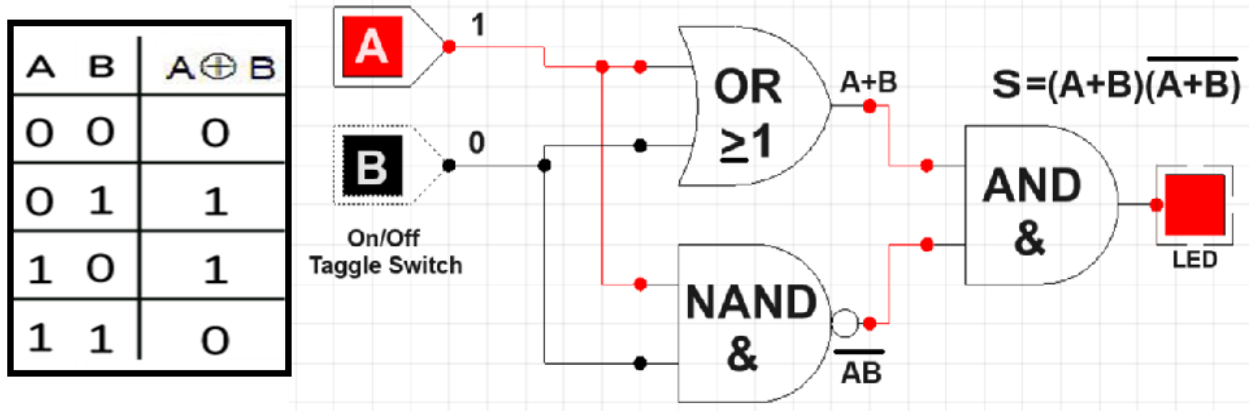


Text (in Decorations):

It is used to give labels (names) to the inputs and outputs of logic gates.



Example 2: Design the logic circuit diagram of the XOR gate.

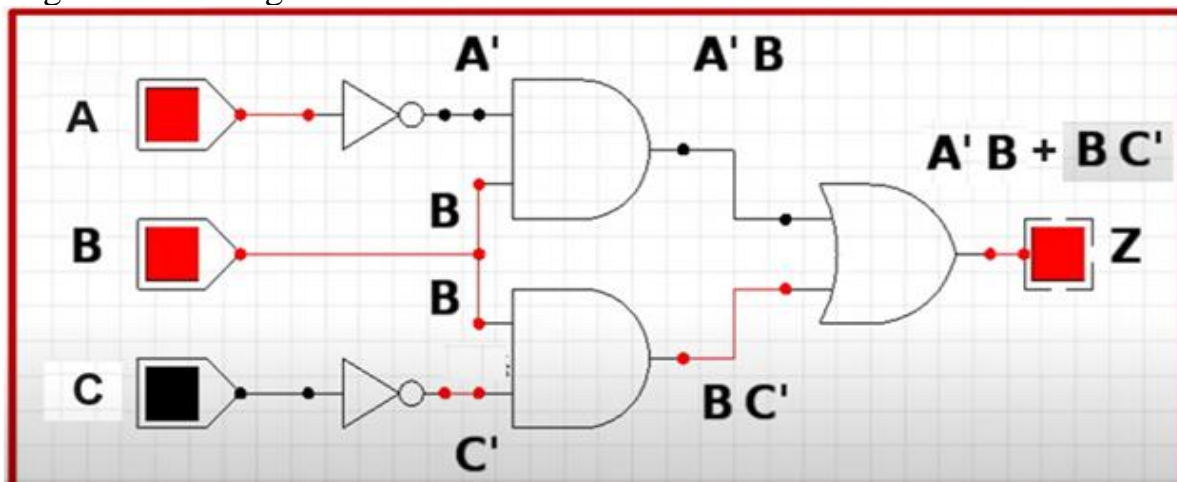


Example 3: Derive the logic circuit diagram and truth table from the following Boolean expression (logical expression).

$$Z = \overline{A} B + B \overline{C}$$

Sol:

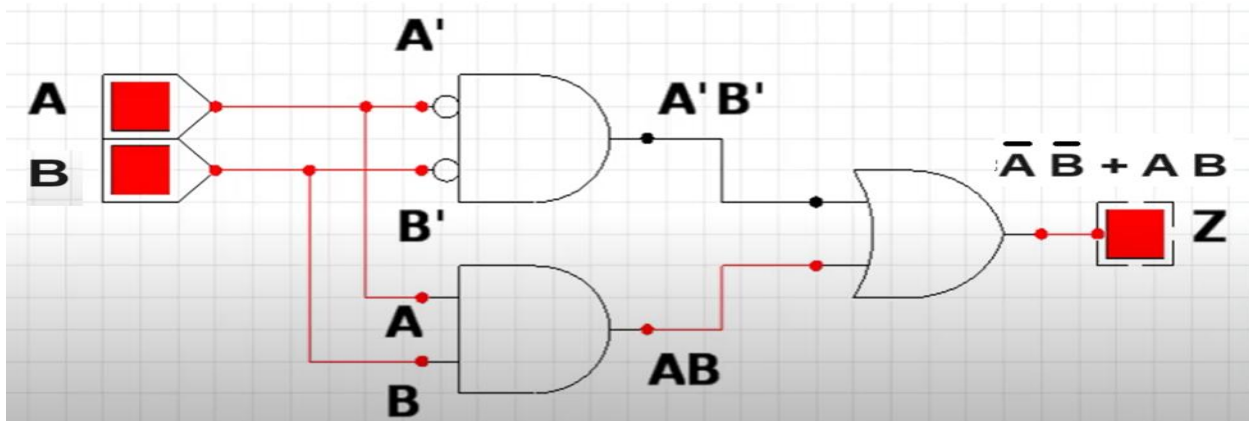
Logic Circuit Diagram



The truth table

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

Example 4: Derive the Boolean expression and truth table from the following logic circuit.



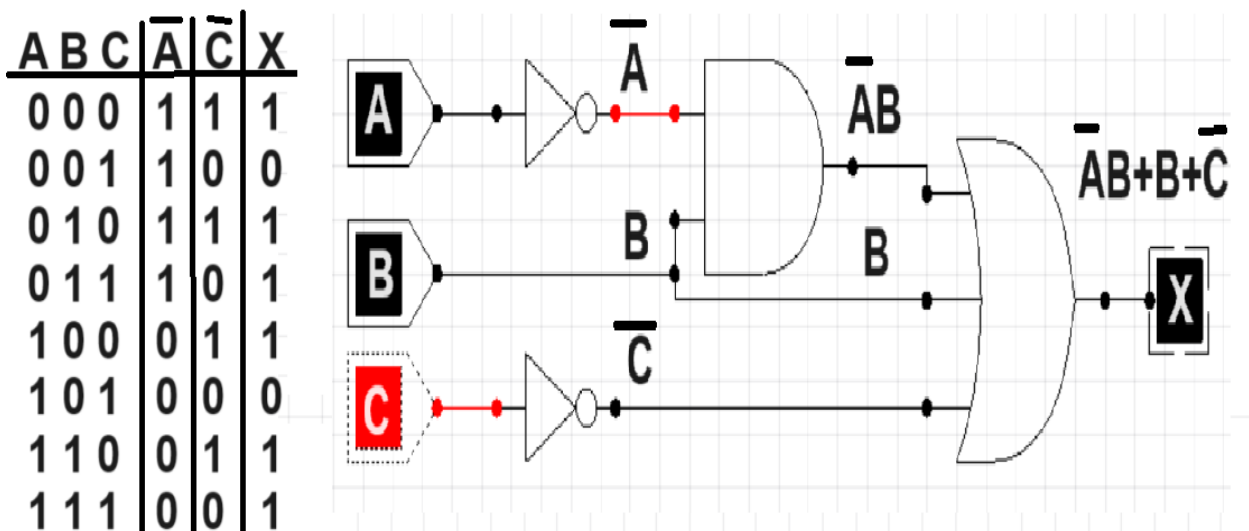
The Boolean expression:

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

The truth table:

A	B	A'	B'	A' B'	A B	A' B' + A B
0	0	1	1	1	0	1
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	1	0	0	0	1	1

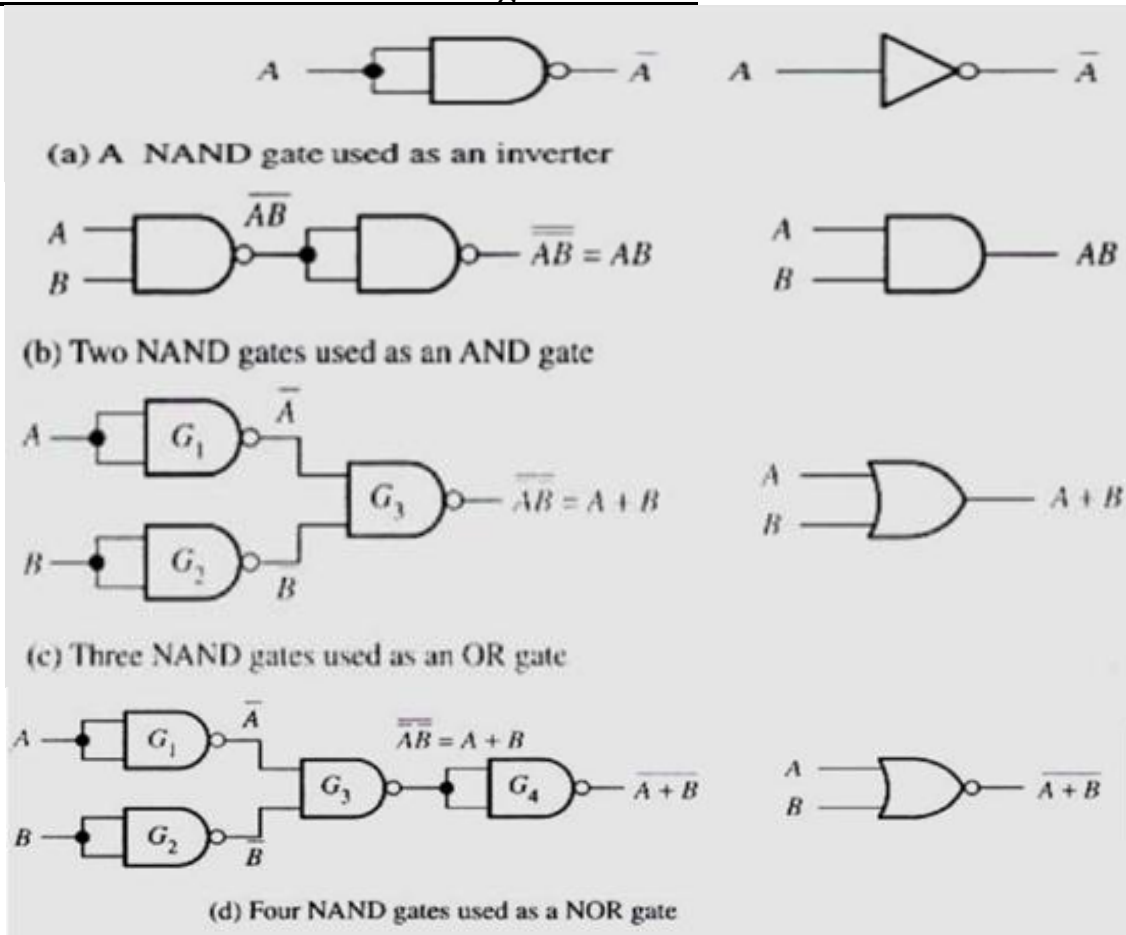
Example 5: Derive the Boolean expression and truth table from the following logic circuit.



The Boolean expression:

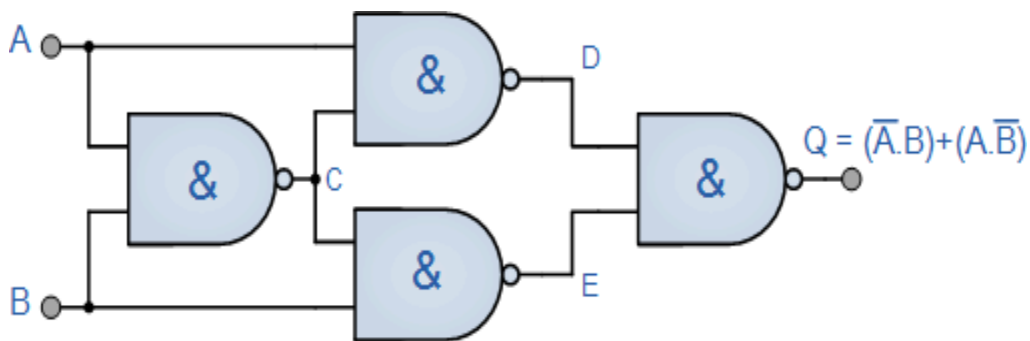
$$X = \bar{A} B + B + \bar{C}$$

The NAND Gate as a Universal Logic Element:



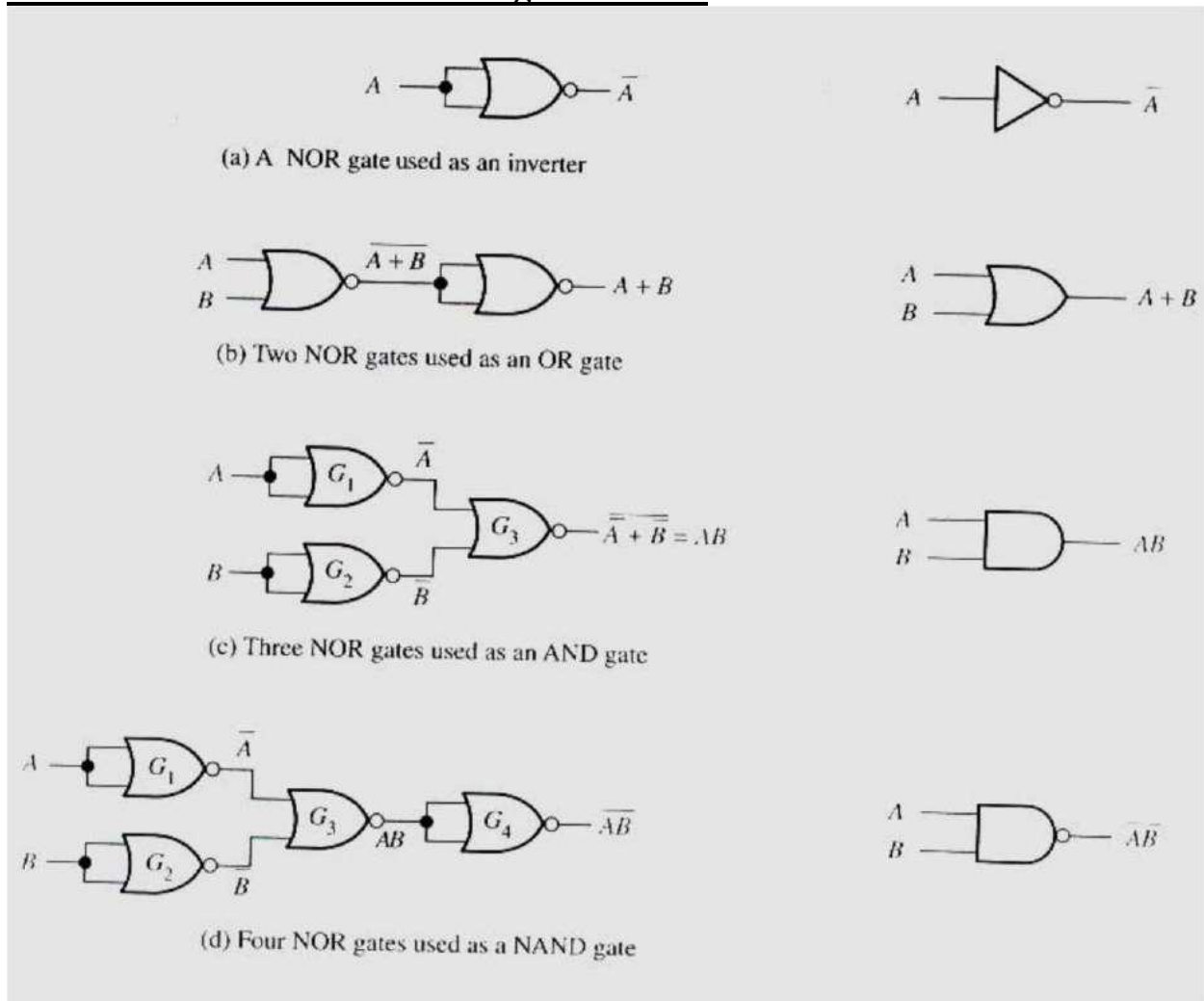
Ex-OR Function using NAND gates

$$S = A \oplus B = \overline{A}B + A\overline{B} = (A+B)\overline{(A+B)}$$



NAND Gate Realisation

The NOR Gate as a Universal Logic Element:



Types of logic circuits (Integrated Circuits)

Digital logic circuits are divided into:

1- Combinational Logic Circuits.

Its main component is the gates.

Such as: Adder and Subtractor circuits

2- Sequential Logic Circuits.

Its main component is the Flip Flops

Such as: Counter and Processor circuits

The difference:

Sequential in which the output change occurs only at the edge of the clock pulse (CLK), example of CLK impulses: Traffic signal, Horse racing start signal. While the combinational changes its output at any point (any time) according to the state of input (according to the user).

Combinational Logic Circuits.

Comparator [2x3]

Logic circuit used to compare two digital numbers

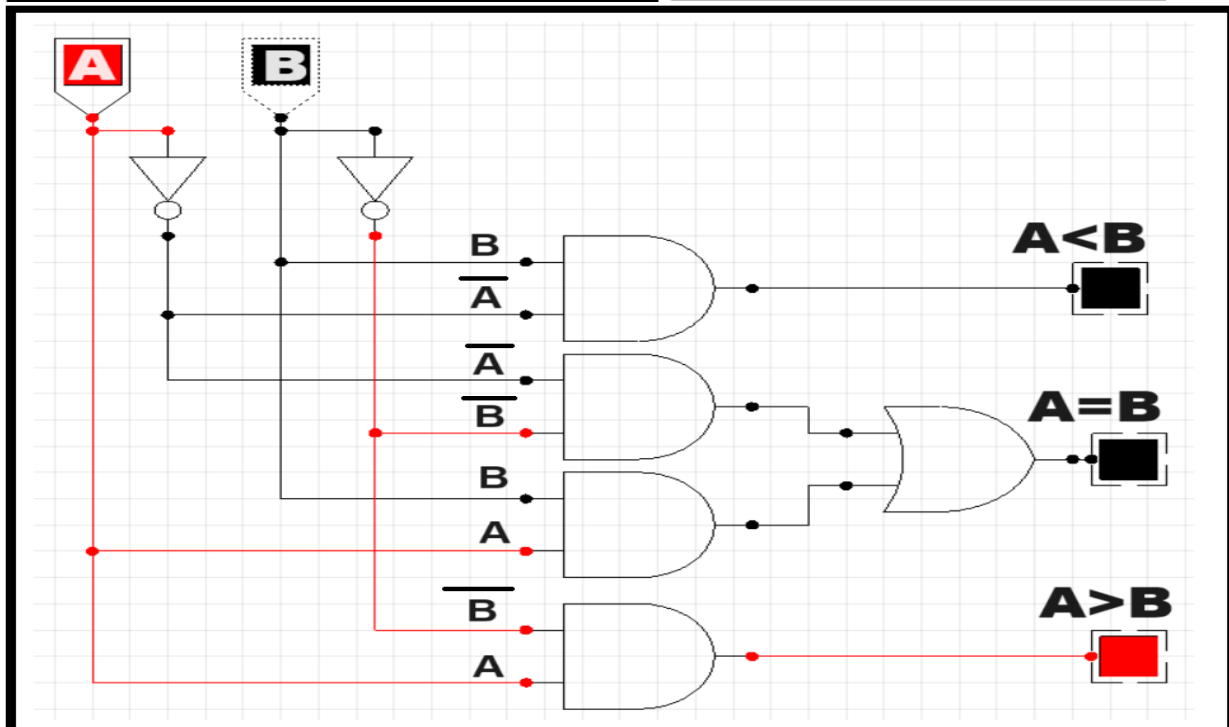
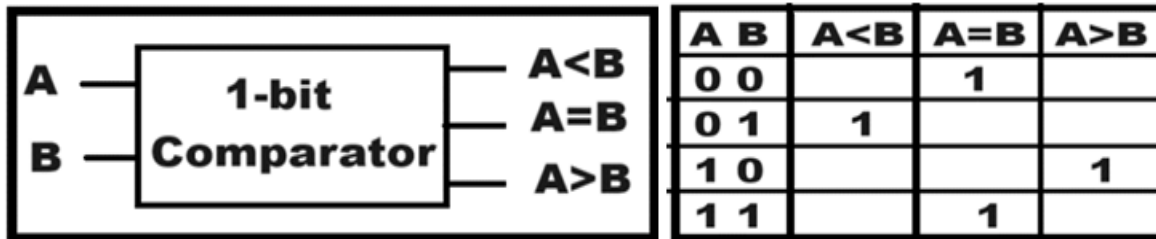


Figure 3: Comparator [2x3]

Adder and Subtractor circuits

Adder circuits

Used to perform addition on binary numbers.

Half - Adder circuit:

It is used to perform addition on 2 bits. This circuit takes two bits A and B as inputs and produces two outputs S (sum) and C (carry). If both A and B are equal to 1, then the value of the C (carry) will be 1, otherwise the carry will be 0.

Half- Adder questions:

$$S = A \oplus B = \overline{A}B + A\overline{B} = (A+B)\overline{(A+B)}$$

$$C = AB$$

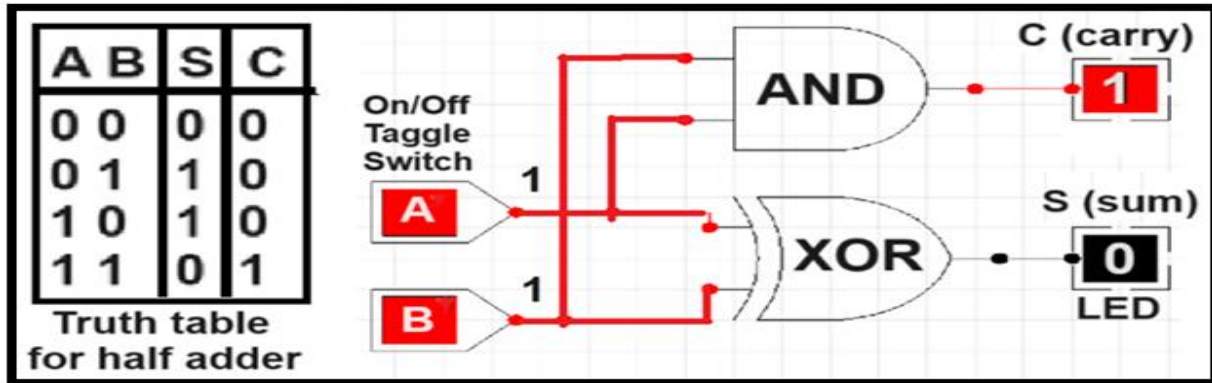


Figure 4: Half Adder Logic diagram

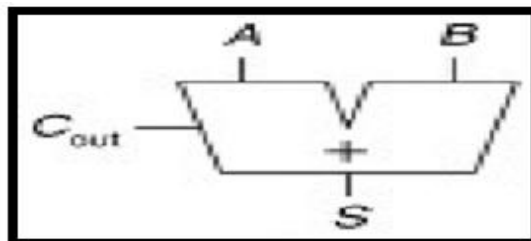


Figure 5: Half Adder Block diagram

Full-Adder circuit:

The full adder is used to add 3-bit. The Full Adder has three input A, B, and carry (C in) and two output sum and carry (C out).

Full - Adder questions:

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

$$C = AB \oplus AC_{in} \oplus BC_{in} = AB \oplus C_{in}(A + B)$$

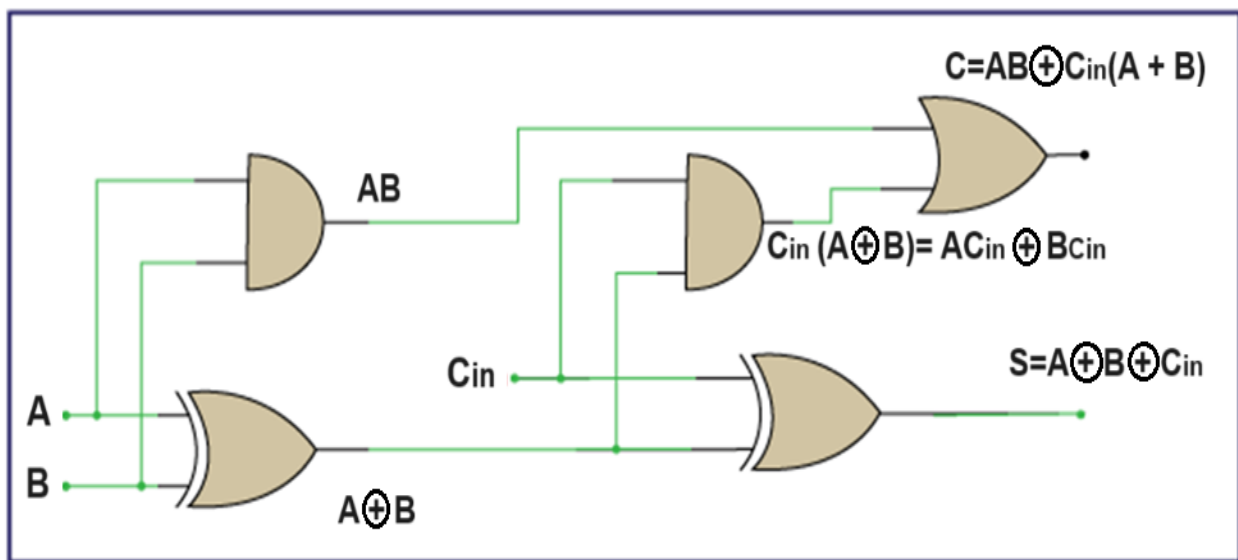


Figure 6: Full Adder Logic diagram

Truth table for Full Adder

Inputs			Outputs	
A	B	C _{in}	Sum	Carry
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

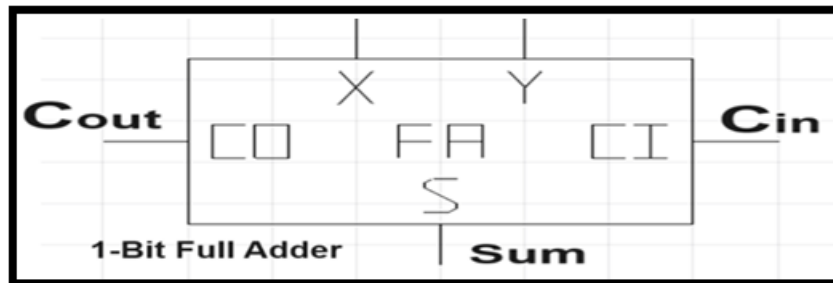


Figure 7: Full Adder Block diagram

4- Bit Full Adder:

The 4-bit full Adder is used to add 4 bits.

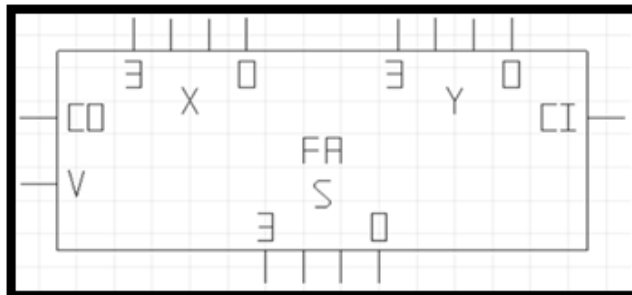


Figure 8:4-bit Full Adder Block diagram

4- Bit Parallel Adder:

Two 4-bit digital circuits are connected together in parallel to add two binary (4-bit) numbers as shown in the following figure.

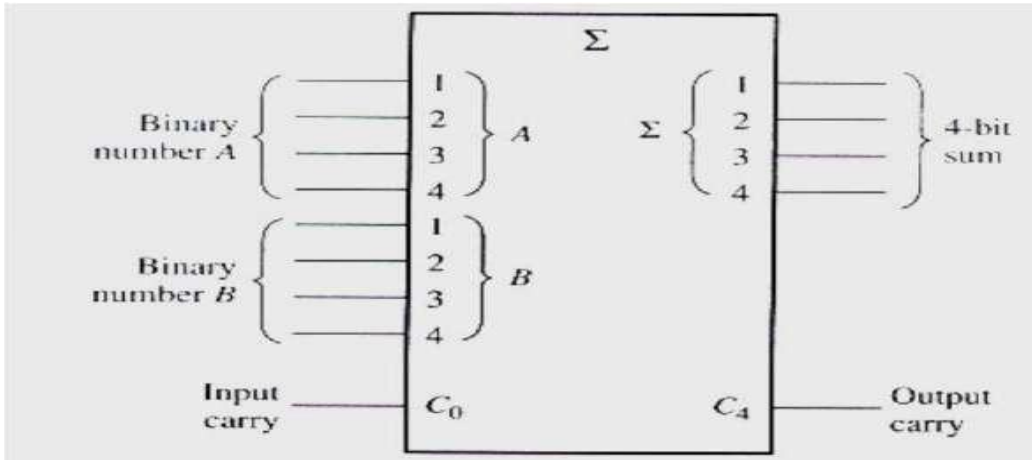


Figure 9: 4- Bit Parallel Adder:

Example:

Draw the 4-bit parallel Adder and find the sum and carry of the following two 4-bit numbers, knowing that the input carry (C_{n-1}) = 0.

$A=1010$, $B=1011$

Sol:

For n=1

$A_1=0, B_1=1, C_0=0 \rightarrow \Sigma = 1, \text{ and } C_1=0$

For n=2

$A_2=1, B_2=1, C_1=0 \rightarrow \Sigma=0, \text{ and } C_2=1$

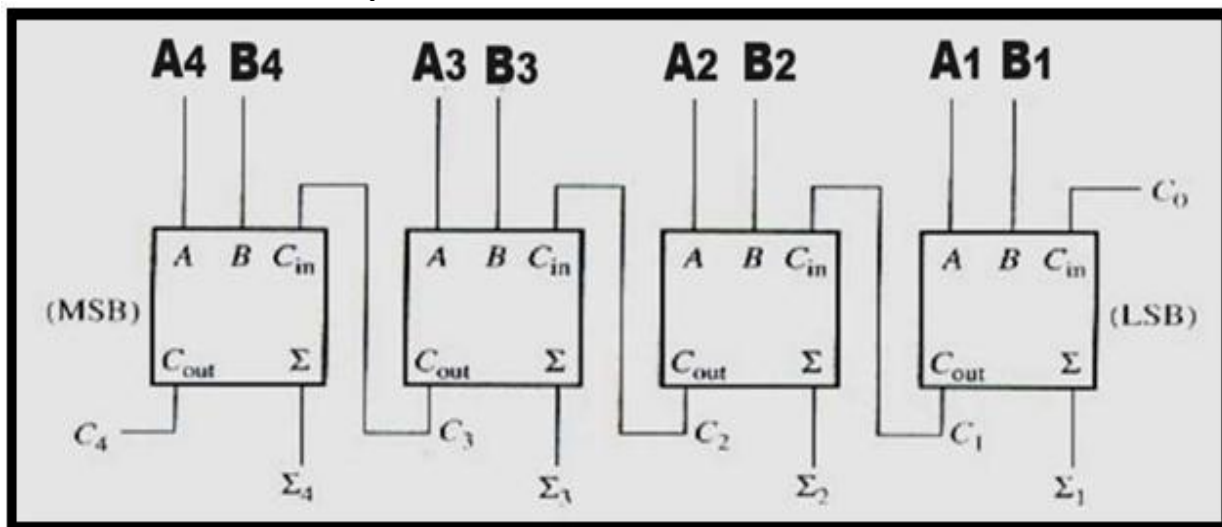
For n=3

$A_3=0, B_3=0, C_2=1 \rightarrow \Sigma=1, \text{ and } C_3=0$

For n=4

$A_4=1, B_4=1, C_3=0 \rightarrow \Sigma=0, \text{ and } C_4=1$

Results = 0100 with carry (1) $\rightarrow (10100)_2$



Subtractor circuits

Used to perform Subtraction on binary numbers.

Half-Subtractor circuit

A half-subtractor circuit is used to perform a binary subtraction of 2-bits of data. It is based on the following Truth Table.

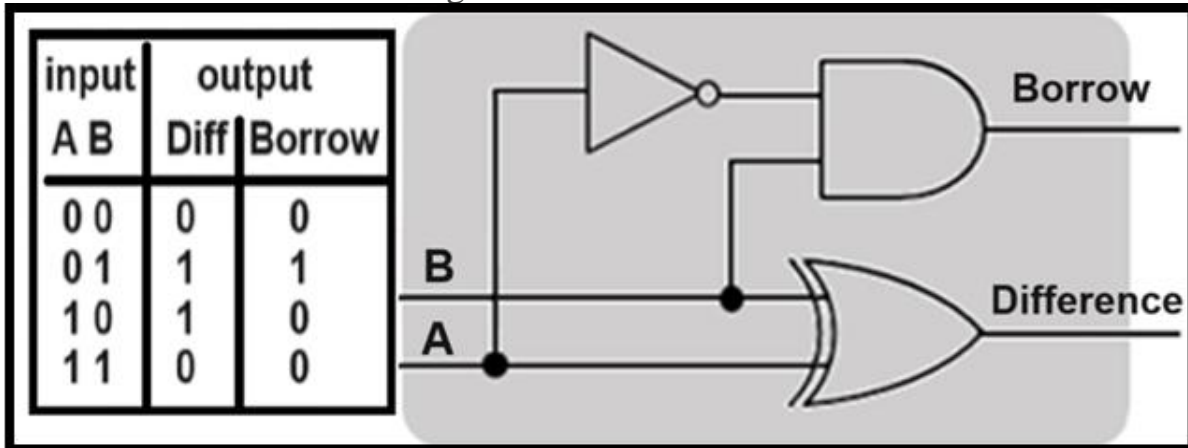


Figure 10: Logic diagram of half subtractor

Full-Subtractor circuit

A full-subtractor circuit is used to perform a binary subtraction using three bits of data.

Full - Subtractor equations:

$$D_{out} = B_{in} \oplus (A \oplus B)$$

$$B_{out} = B_{in} \overline{(A \oplus B)} \oplus \overline{AB}$$

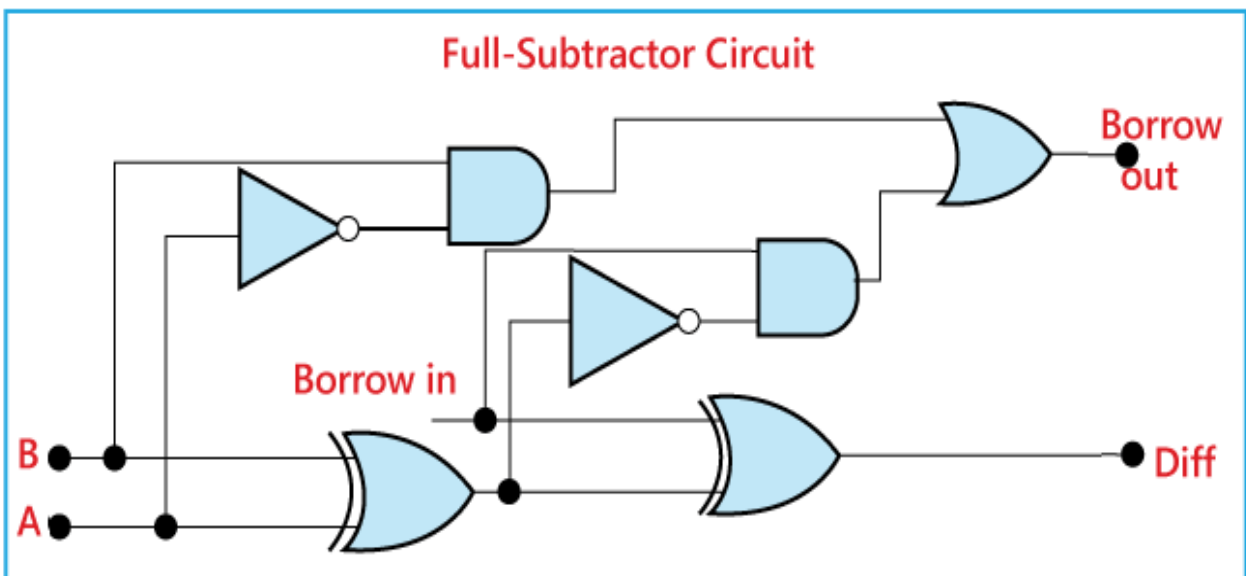


Figure 11: Full subtractor Logic diagram

Truth table for Full subtractor

Inputs			Outputs	
A	B	Borrow _{in}	Diff	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

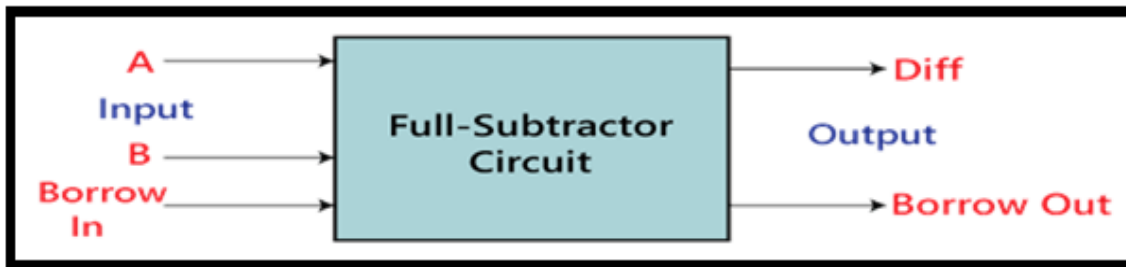


Figure 12: Full subtractor Block diagram

4-Bit binary Adder-Subtractor

For n-bit binary subtraction, full adders are used with the complement applied to the number being subtracted.

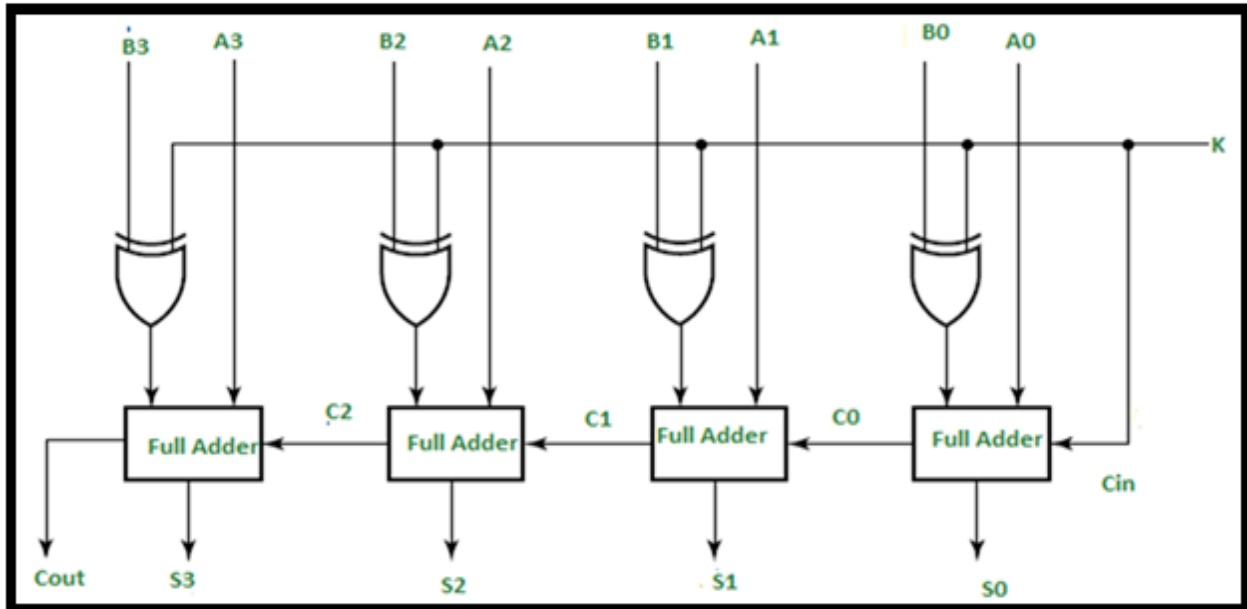


Figure 13: Bit binary Adder-Subtractor

Decoder and Encoder circuits

Decoders:

A combinational logic circuit that converts an N-bit binary input code into 2^N output channels in such a way that only one output channel is activated for single binary number.

Types of Decoders

- **2x4 Decoder** (2-binary input & $2^2 = 4$ output channels)
- **3x8 Decoder** (3- binary input & $2^3 = 8$ output channels)
- **4x16 Decoder** (4- binary input & $2^4 = 16$ output channels)

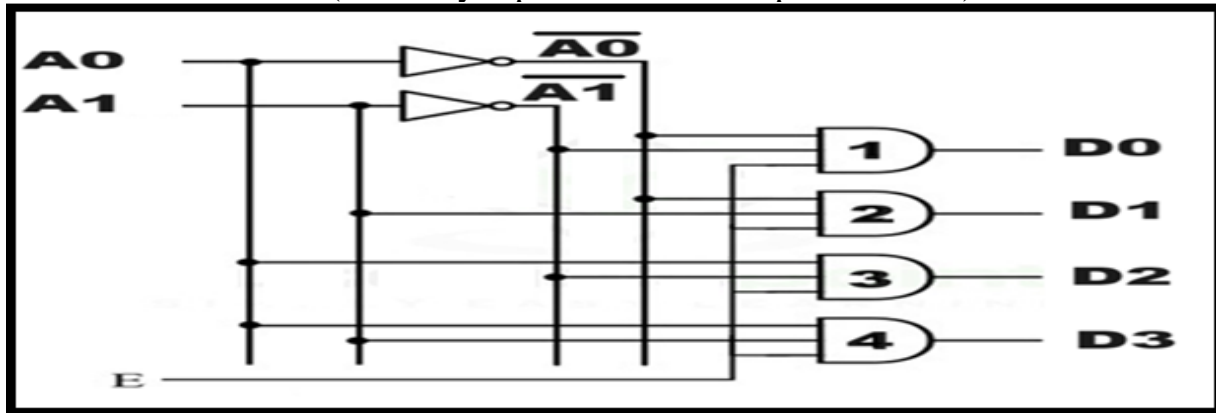


Figure 14: Logic Diagram of 2x4 Decoder

Note: (E) used to Enable (1) / Disable (0) the Decoder, default value (1).

Truth table 2x4 Decoder

	Inputs		Outputs			
	A1	A0	D3	D2	D1	D0
0	0	0	0	0	0	1
1	0	1	0	0	1	0
2	1	0	0	1	0	0
3	1	1	1	0	0	0

The Boolean expression for each output is:

$$\begin{array}{l}
 \mathbf{D0 = \bar{A} \bar{B}} \quad \mathbf{D2 = \bar{A} B} \\
 \mathbf{D1 = \bar{A} B} \quad \mathbf{D3 = A B}
 \end{array}$$

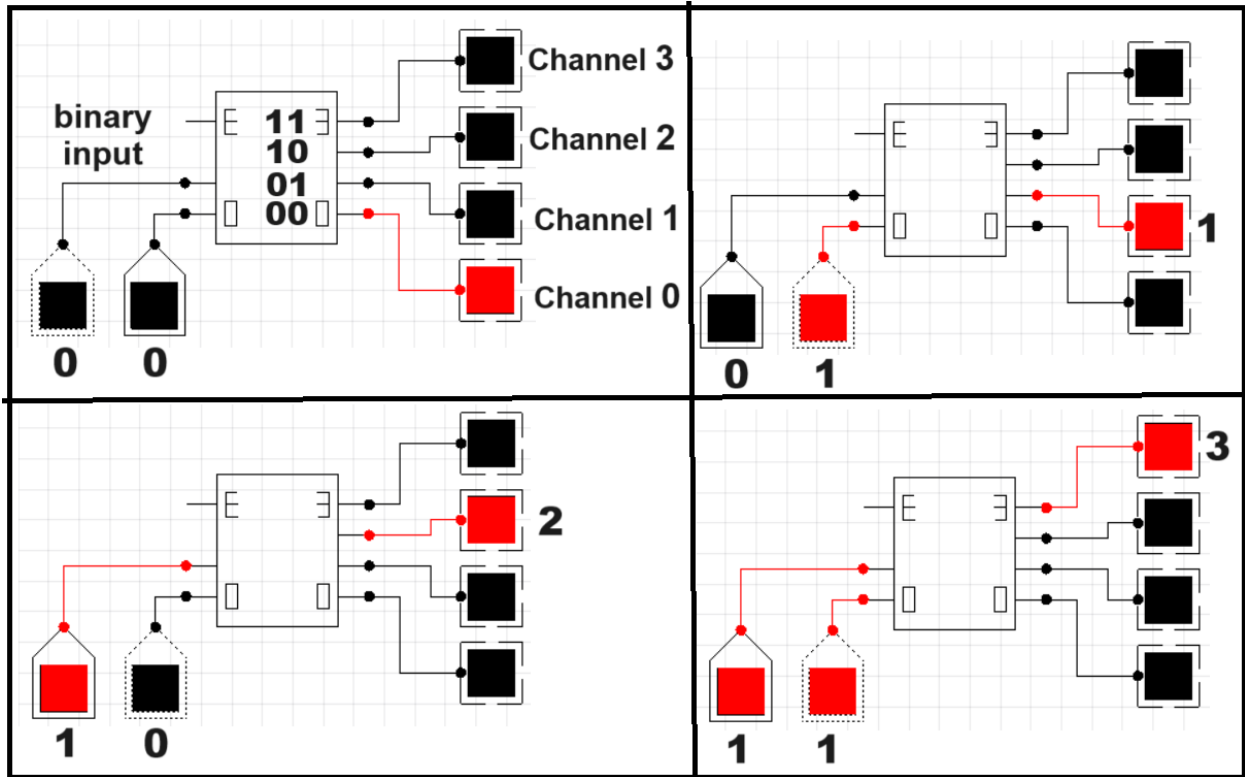


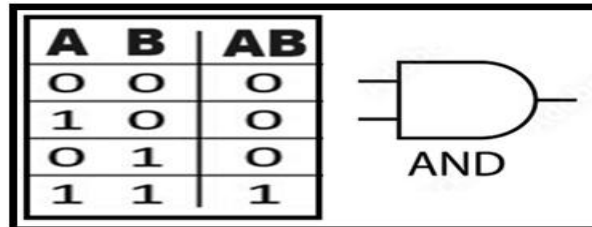
Figure 15: 2x4 Decoder Block diagram

ملاحظة: عند التطبيق العملي انتبه الى ان الجدول (Truth Table) مقلوب من أسفل الى اعلى.

Decoder implementation

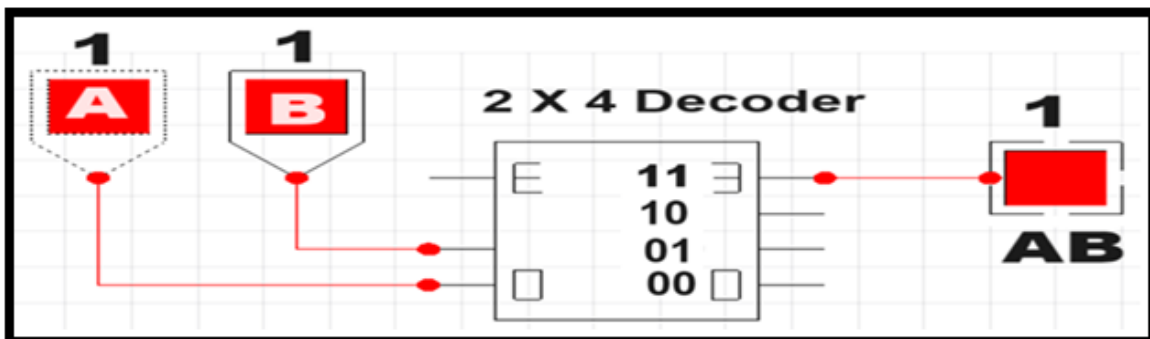
Example 1: Use 2x4 Decoder as 2- input AND gate.

Sol:



2- Input AND gate

$$Z=AB$$



ملاحظة: تسلسل جدول (Truth table) في ال (decoder) مقلوب

Example 2: Use 2x4 Decoder as 3-input AND gate.

Note: (E) used to Enable (1) / Disable (0) the Decoder.

Sol:

Decoder	E	A	B	AB
	0	0	0	0
	0	0	1	0
	0	1	0	0
	0	1	1	0
	1	0	0	0
	1	0	1	0
	1	1	0	0
	1	1	1	1

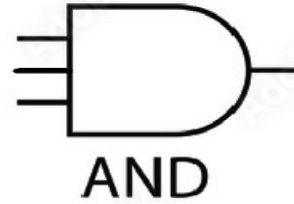
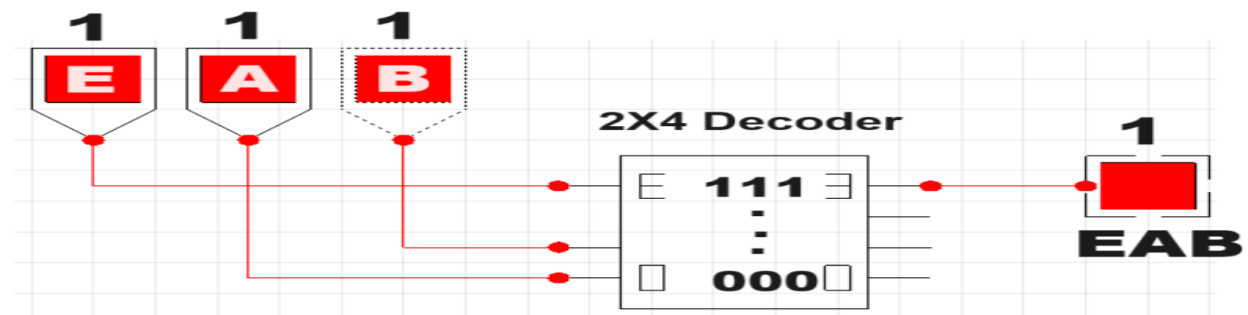


Figure 15: 3-Input AND Truth Table

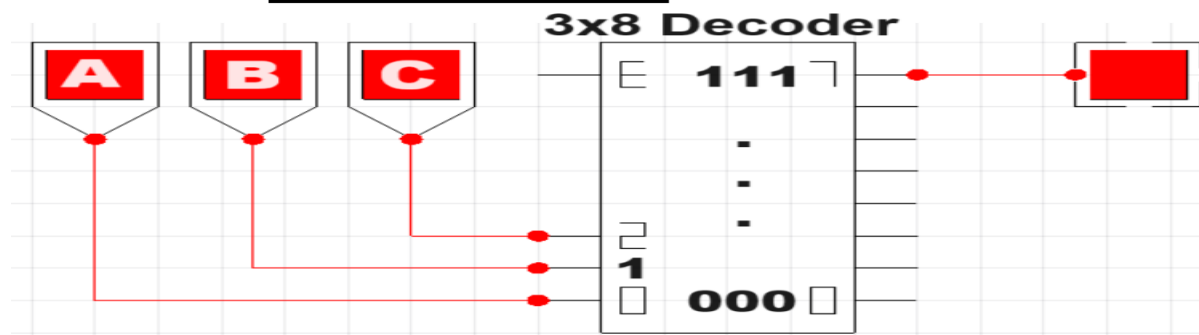
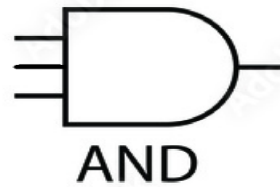
$Z = EAB$



Example 3: Use 3x8 Decoder as 3-input AND gate.

Sol:

A	B	C	AB
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

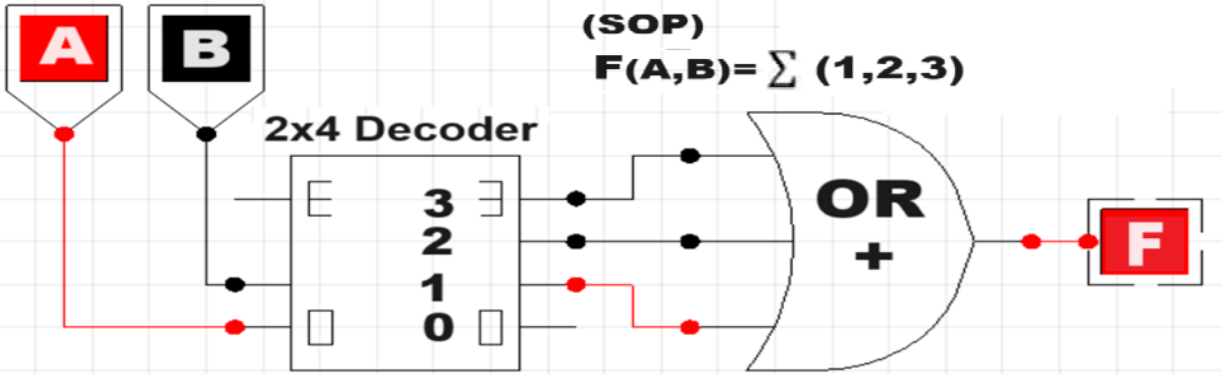


Example 4: Use the Decoder as 2-input OR gate.

Sol:

	A	B	F
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

(SOP)
 $F(A,B) = \sum (1,2,3)$
 Decoder Output



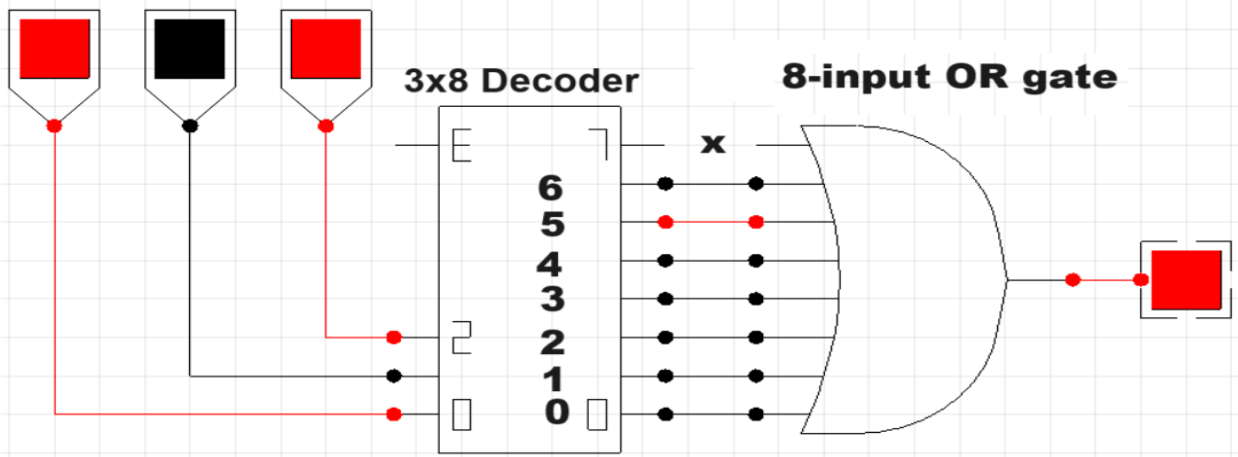
Example 5: Use 3x8 Decoder as 3-input NAND gate.

Sol:

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



(SOP)
 $F(A,B,C) = \sum (0,1,2,3,4,5,6)$



Example 6: Use the Decoder as Full Adder.

Sol:

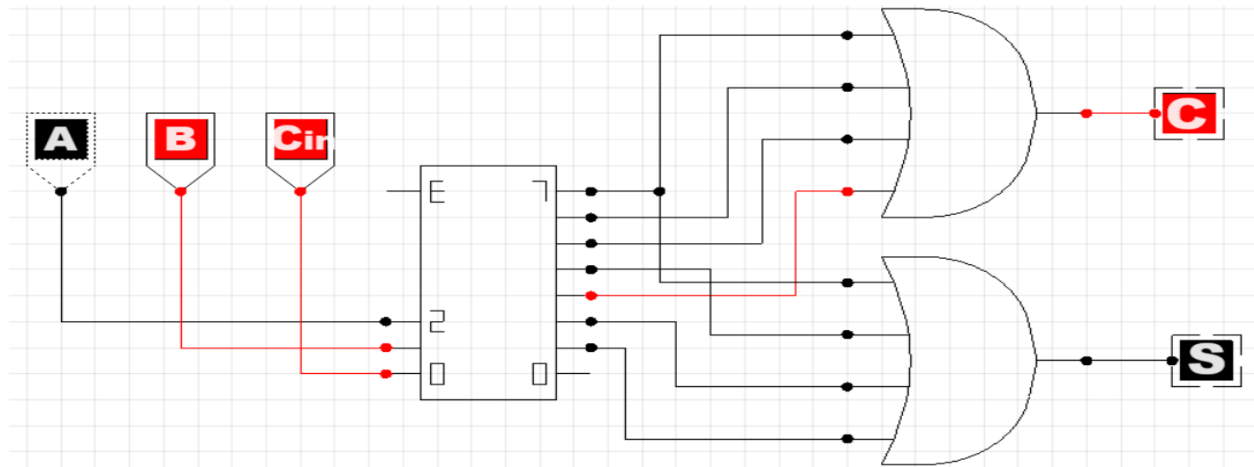
The truth table of Full Adder

	Inputs			Outputs	
	A	B	C _{in}	Sum	Carry
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1

(SOP)

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

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



Example 7:

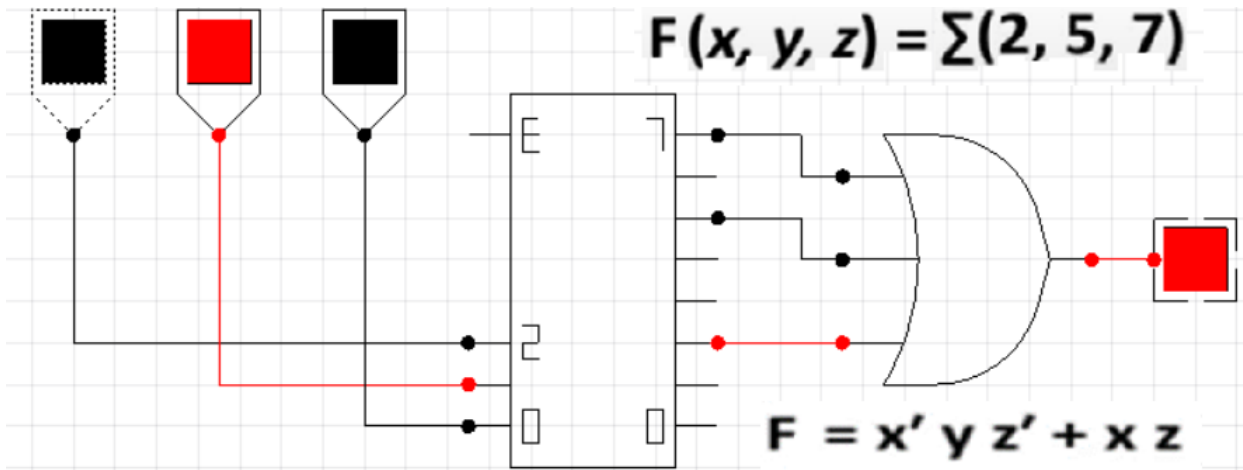
Use a decoder to design a logic circuit defined by the following Boolean function.

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

Sol:

	x	y	z	\bar{x}	\bar{z}	$\bar{x}y\bar{z}$	xz	F
0	0	0	0	1	1	0	0	0
1	0	0	1	1	0	0	0	0
2	0	1	0	1	1	1	0	1
3	0	1	1	1	0	0	0	0
4	1	0	0	0	1	0	0	0
5	1	0	1	0	0	0	1	1
6	1	1	0	0	1	0	0	0
7	1	1	1	0	0	0	1	1

$$F(x, y, z) = \sum (2, 5, 7)$$



Example 8: Use a single decoder to design a logic circuit defined by the following Boolean functions.

$$F_1 = x y' z' + x' y$$

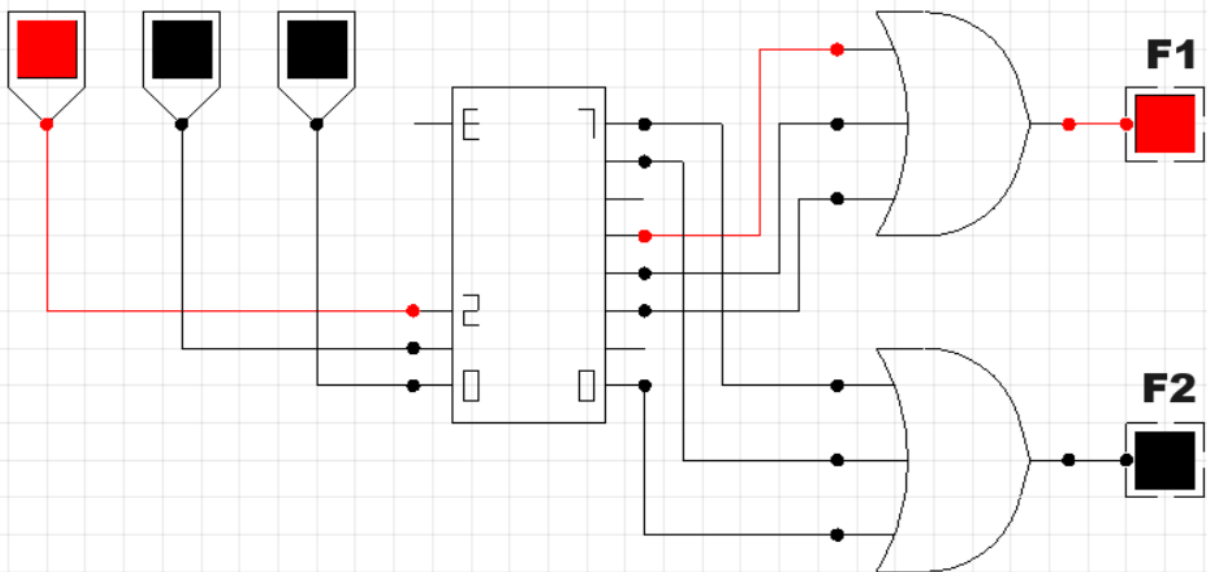
$$F_2 = x' y' z' + x y$$

Sol:

	X	Y	Z	F ₁	F ₂
0	0	0	0	0	1
1	0	0	1	0	0
2	0	1	0	1	0
3	0	1	1	1	0
4	1	0	0	1	0
5	1	0	1	0	0
6	1	1	0	0	1
7	1	1	1	0	1

$$F_1(x, y, z) = \Sigma(2, 3, 4)$$

$$F_2(x, y, z) = \Sigma(0, 6, 7)$$



Example 9: Use a single decoder to design a combinational circuit defined by the following Boolean functions.

$$F_1 = (y' + x) z$$

$$F_2 = y' z' + x' y + y z'$$

$$F_3 = (x + y) z$$

Sol:

$$F_1 = (y' + x) z = y' z + x z$$

$$F_2 = y' z' + x' y + y z'$$

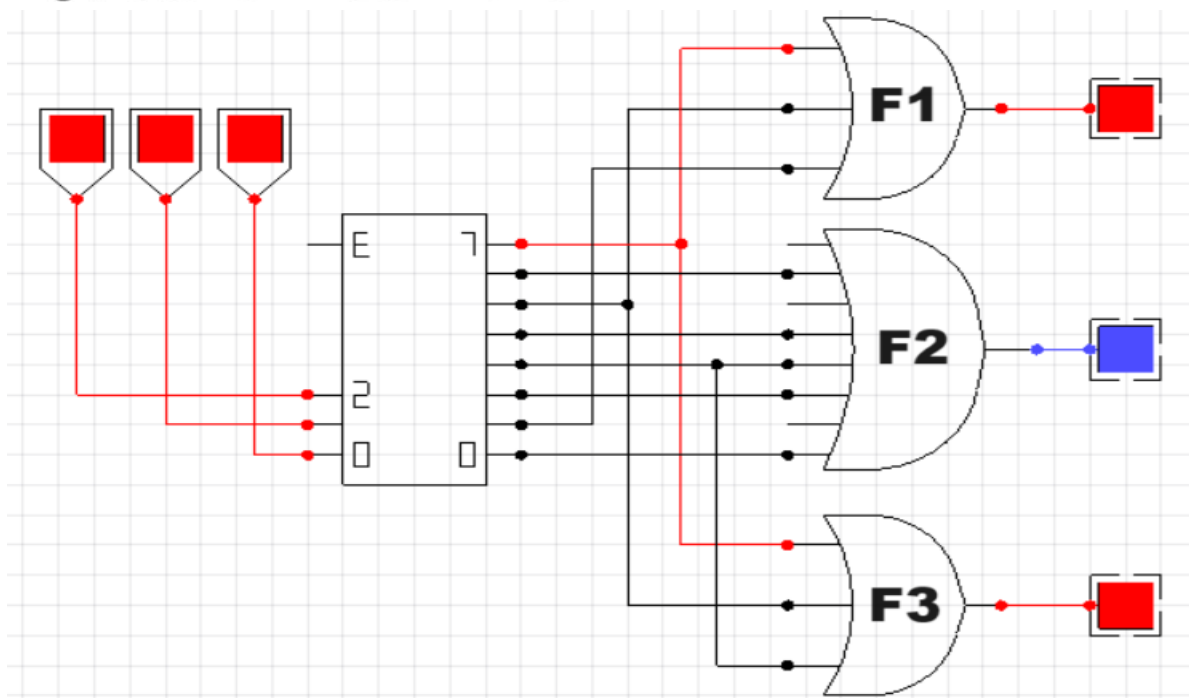
$$F_3 = (x + y) z = x z + y z$$

	X	Y	Z	F ₁	F ₂	F ₃
0	0	0	0	0	1	0
1	0	0	1	1	0	0
2	0	1	0	0	1	0
3	0	1	1	0	1	1
4	1	0	0	0	1	0
5	1	0	1	1	0	1
6	1	1	0	0	1	0
7	1	1	1	1	0	1

$$F_1(x, y, z) = \Sigma(1, 5, 7)$$

$$F_2(x, y, z) = \Sigma(0, 2, 3, 4, 6)$$

$$F_3(x, y, z) = \Sigma(3, 5, 7)$$

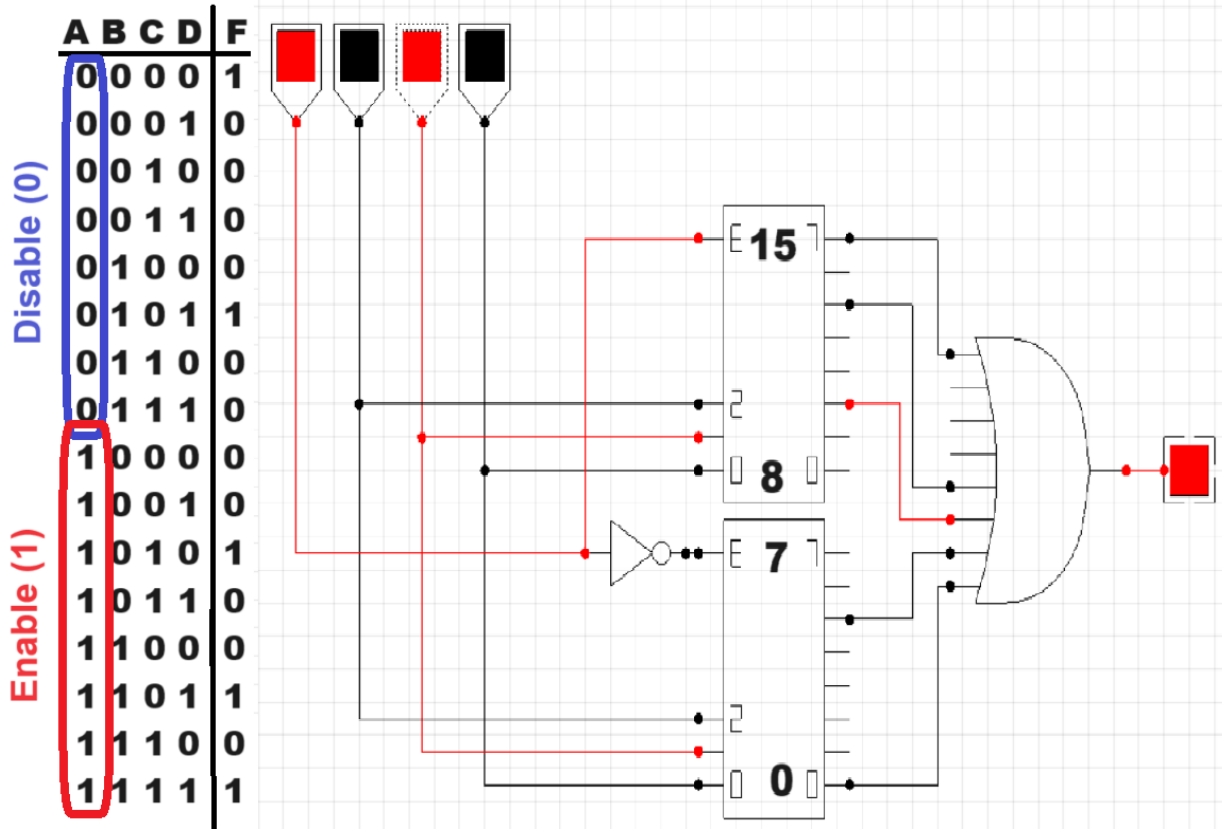


Example 10: Use 3x8 Decoder to design a logic circuit defined by the following canonical (SOP).

$$F(A,B,C,D) = \sum (0,5,10,13,15)$$

Sol:

Note: If 4 values are required to be input to a 3-input decoder, two 3X8 decoders are used and the E value is used to input the fourth most significant bit (MSB).



Encoder:

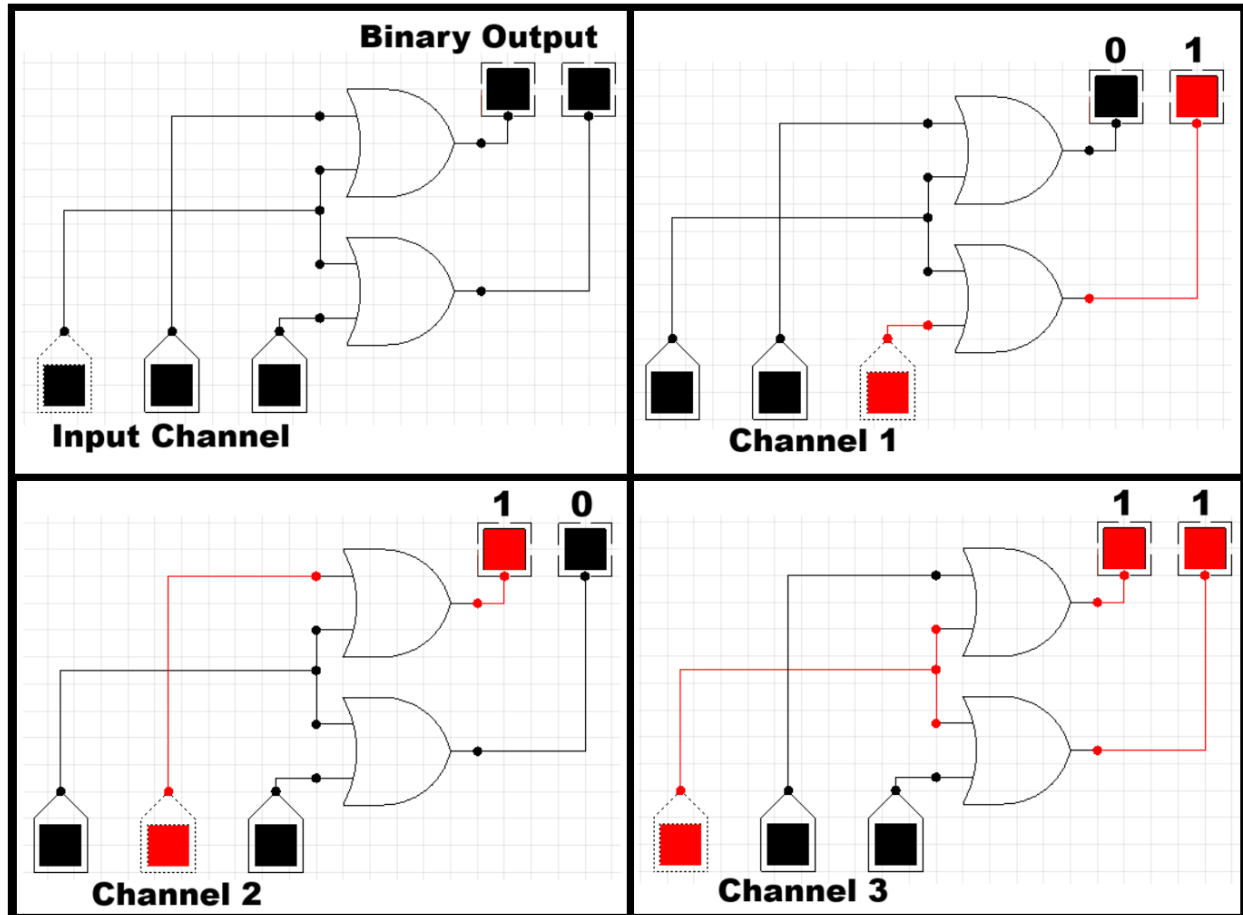
An encoder is a digit circuit that performs the inverse operation of a decoder. An encoder has 2^N input channels and N output binary numbers.

Types of Encoders

- **4x2 Encoder** (4-input channels & 2-digit binary number output)
- **8x3 Encoder** (8-input channels & 3-digit binary number output)
- **16x4 Encoder** (16-input channels & 4-digit binary number output)

4x2 Encoder

4x2 Encoder logic circuit



Truth Table of 4 x 2 Encoder

Inputs				Outputs	
D3	D2	D1	D0	A1	A0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

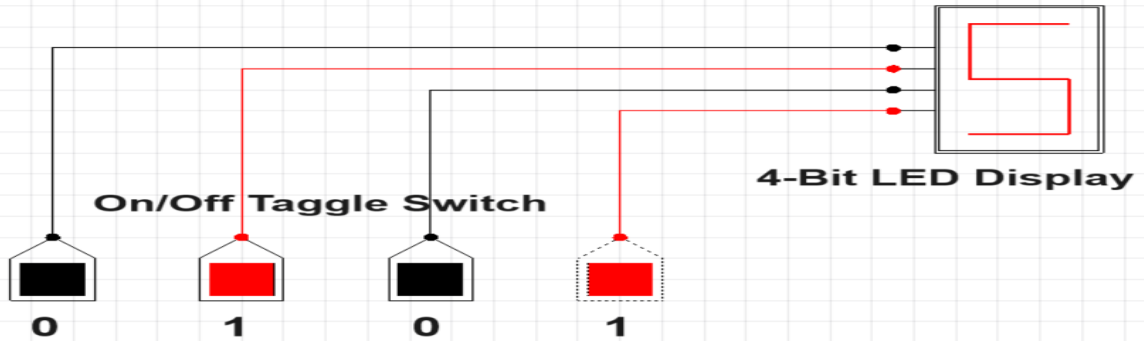
The Boolean expression for each output (A1 and A0) is.

$$A0 = D1 + D3$$

$$A1 = D2 + D3$$

Binary Coded Decimal (BCD):

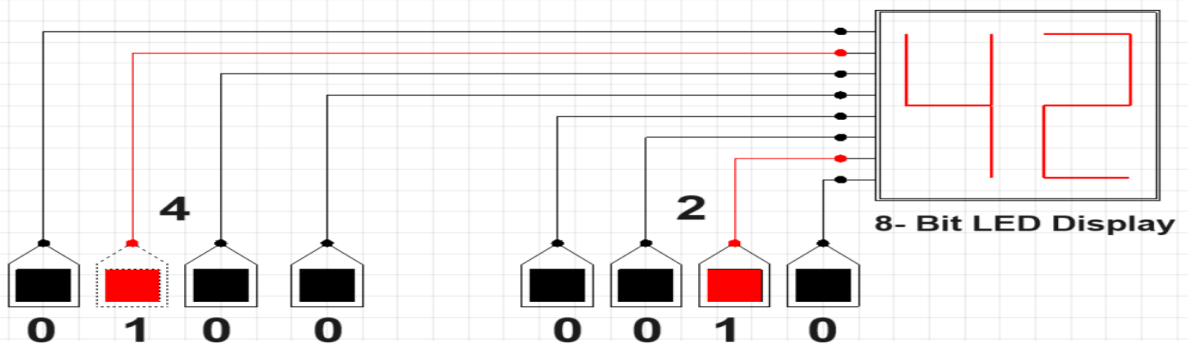
Binary coded decimal means that each decimal digit, 0 through 9, is represented by a binary code of 4-Bits. This system is used in LED displays in front of stores.



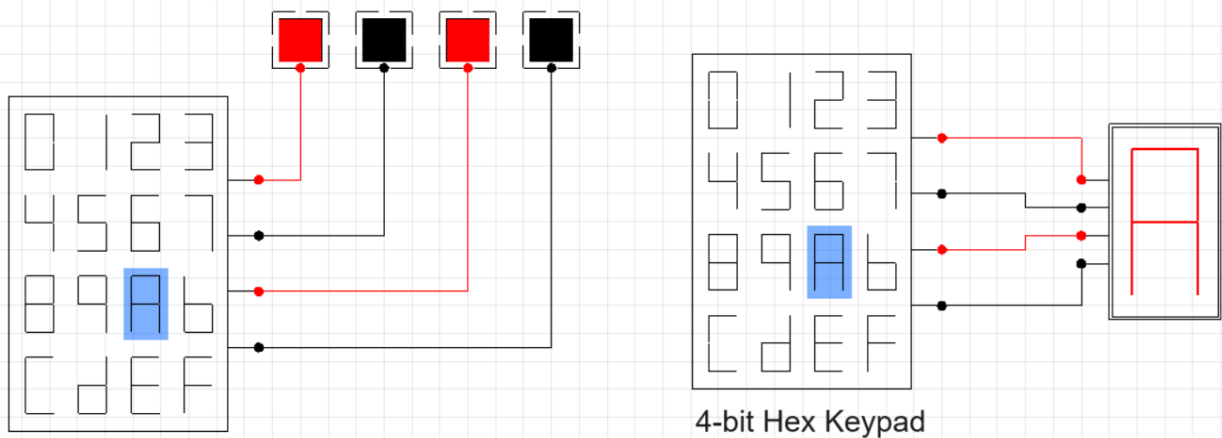
To represent two-digit decimal numbers from 00 to 99, two LED displays are placed side by side (8-bit LED display), and to represent 3-digit digit, three LED displays are used, and so on. This means that the number of LED Display used will be equal to the number of digits.

Example:

Use the LED display to represent the number 42 in BCD.



Note: LED displays can be used to represent hexadecimal digits from 0 to F.



BCD Conversion:

BCD weight = (2³ 2² 2¹ 2⁰)

weight value = (8 4 2 1)

Base = BCD

Weight value 8 4 2 1	Convert to decimal	decimal
0 0 0 0	0	0
0 0 0 1	1	1
0 0 1 0	2	2
0 0 1 1	2+1=3	3
0 1 0 0	4	4
0 1 0 1	4+1=5	5
0 1 1 0	4+2=6	6
0 1 1 1	4+2+1=7	7
1 0 0 0	8	8
1 0 0 1	8+1=9	9

Example: Convert the following numbers.

1- (1263)₁₀ to BCD

Sol: 1 2 6 3

0001 0010 0110 0011

(1263)₁₀ = (00010010 0110 0011)_{BCD}

2- (010101110101)_{BCD} to decimal

Sol: 0 1 0 1 0 1 1 1 0 1 0 1

Weight 8 4 2 1 8 4 2 1 8 4 2 1

4+1=5 4+2+1=7 4+1=5

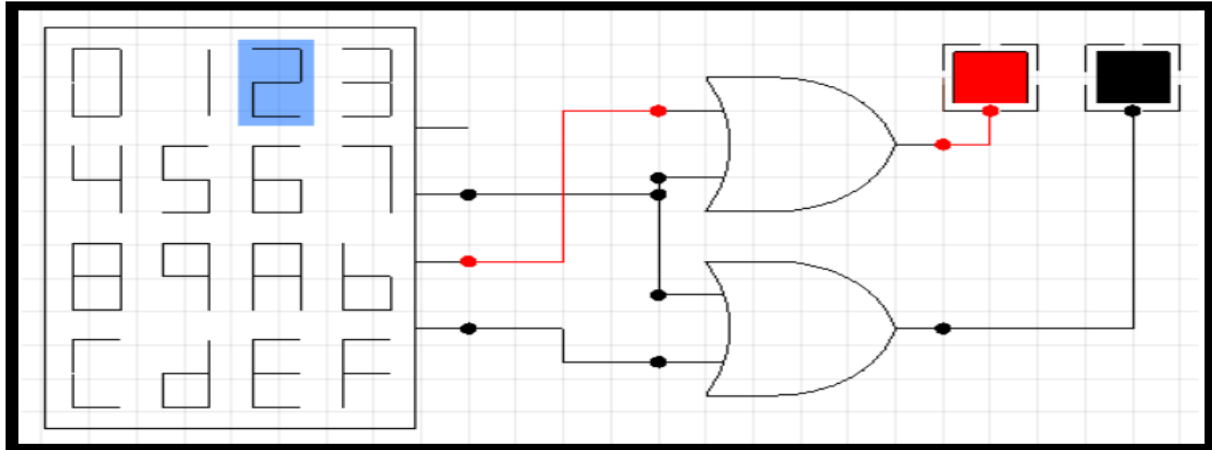
(010101110101)_{BCD} = (575)₁₀

Decimal Coded Binary (DCB):

The Encoder can be used to convert numbers from decimal to binary.

Example:

Use a 4x2 Encoder to design a logic circuit that converts numbers from decimal to binary.



Read Only Memory (ROM)

Stores binary information permanently. it's a decoder connected to OR gates.

- Number of decoder output lines \geq Number of numbers in the set
- Number of OR gets = Number of bits in great number

Example 1:

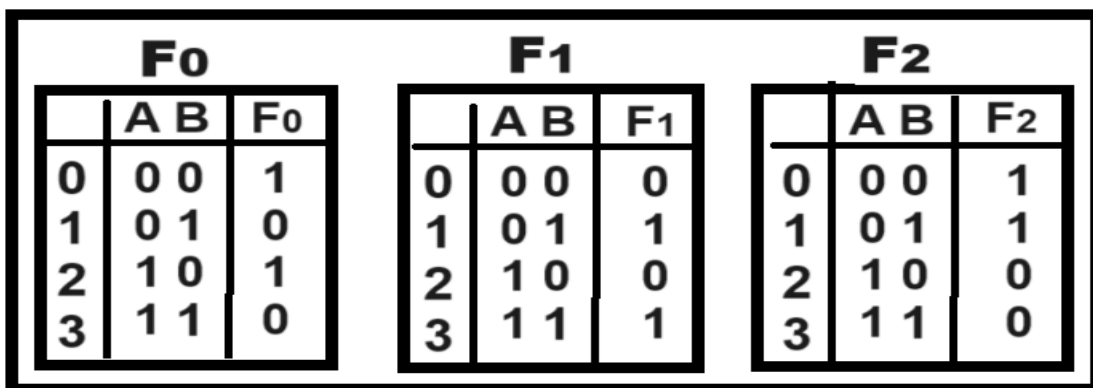
Design a ROM to store the following data (101 ,011 ,100 ,010)

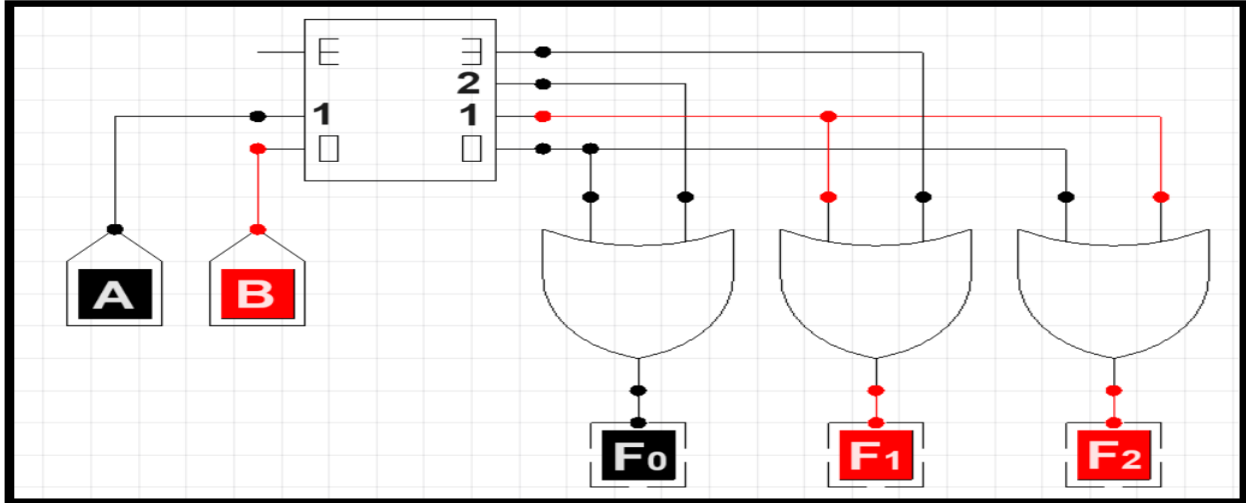
Sol:

- 4 number in the set = 4 decoder output lines = 2 decoder input lines
- 3 Bits in great number = 3 OR gets.

Decoder input lines		Output Data		
	A B	F ₀	F ₁	F ₂
0	0 0	1	0	1
1	0 1	0	1	1
2	1 0	1	0	0
3	1 1	0	1	0

State Table





Example 2:

Design a ROM to store the following data

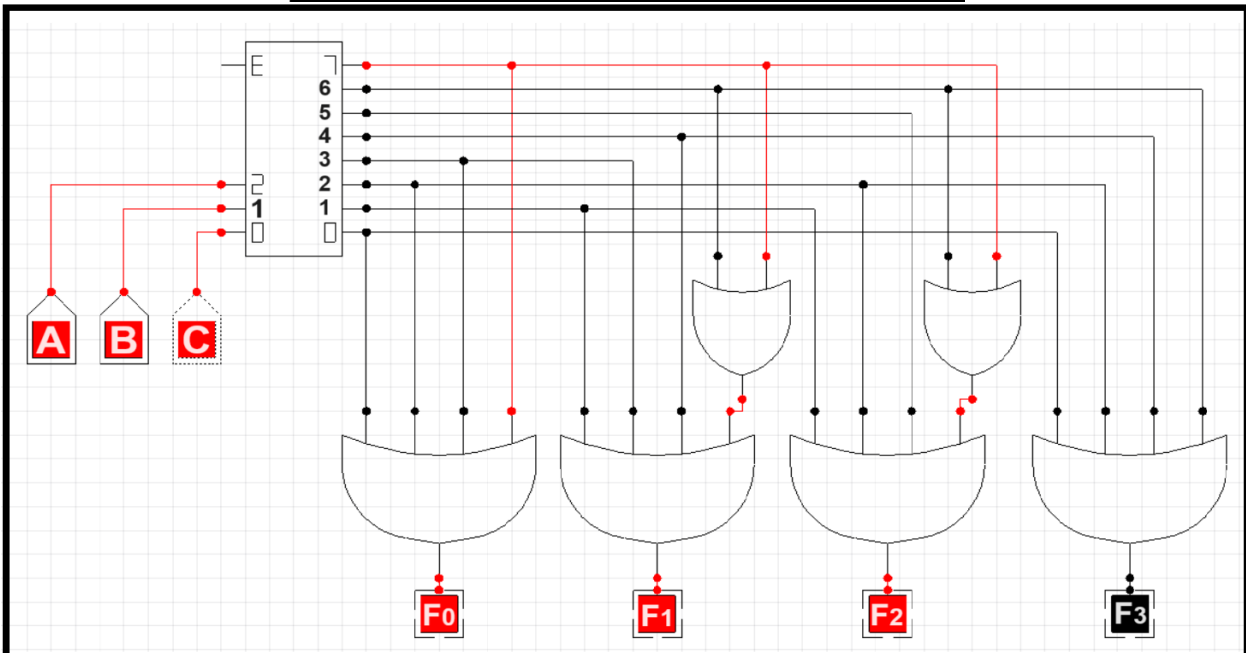
(1001 ,0110 ,1011 ,1100 ,0101 ,1010 ,0111 ,1110)

Sol:

- 8 number in set = 8 decoder output lines = 3 decoder input lines
- 4 Bits in great number = 4 OR gets.

State Table

	A	B	C	F0	F1	F2	F3
0	0	0	0	1	0	0	1
1	0	0	1	0	1	1	0
2	0	1	0	1	0	1	1
3	0	1	1	1	1	0	0
4	1	0	0	0	1	0	1
5	1	0	1	1	0	1	0
6	1	1	0	0	1	1	1
7	1	1	1	1	1	1	0



Multiplexer and Demultiplexer circuits

Multiplexer (MUX) or Data Selector:

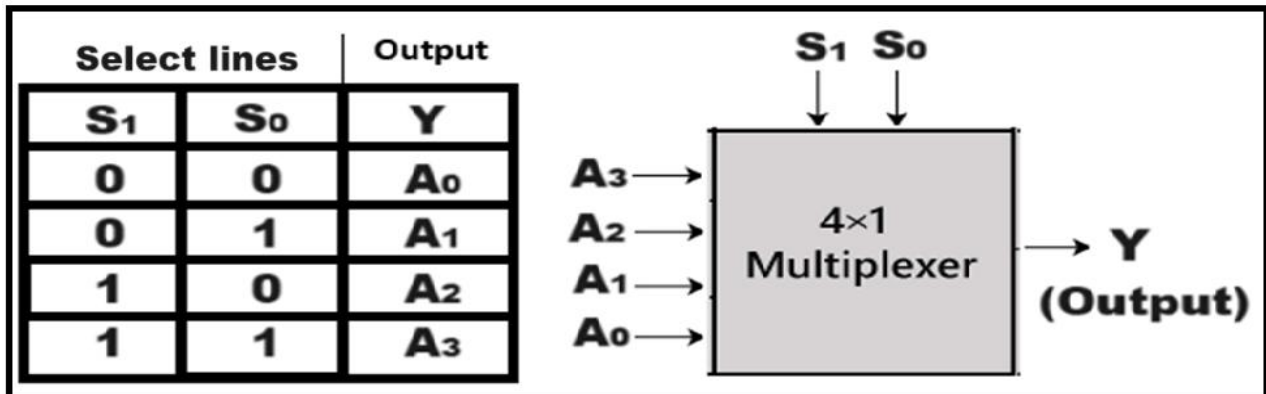
It is a combinational circuit that selects between several digital input signals (or analog signals) and forwards the selected signal to the output channel. The selection of the input signal is controlled by a separate set of digital values known as select lines (Control lines). A multiplexer of 2^N inputs has N select lines, which are used to select which input line to send to the output channel.

Types of Multiplexers

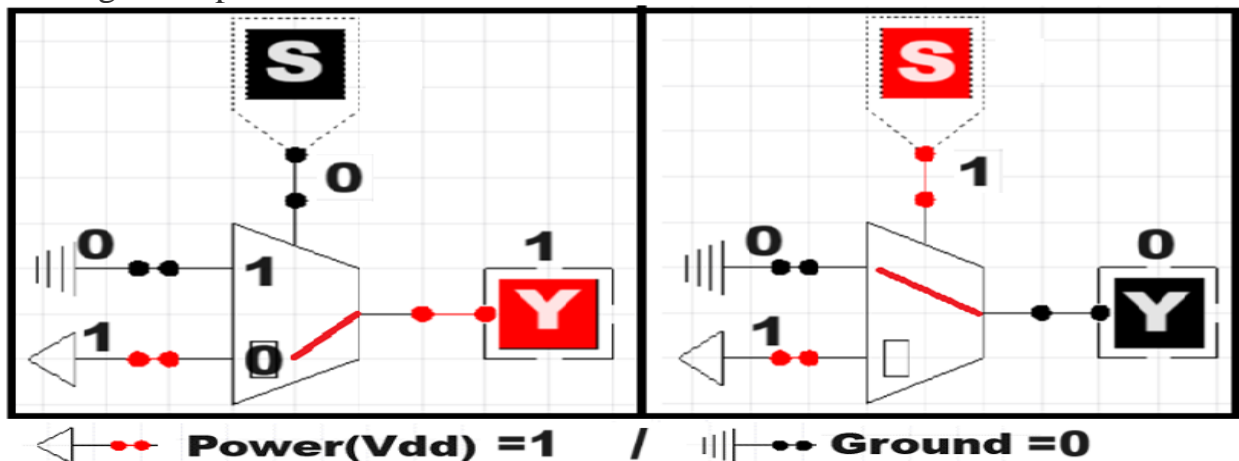
1) 2x1 Multiplexer

- 1- $2^N = 2^1 = 2$ Input signals.
- 2- $N=1$ select lines (Control lines).
- 3- One output channel.

Note: The number of selection lines is equal to the power N.



The logical expression of the term Y is: $Y = S_1' \cdot A_0 + S_1 \cdot A_1$



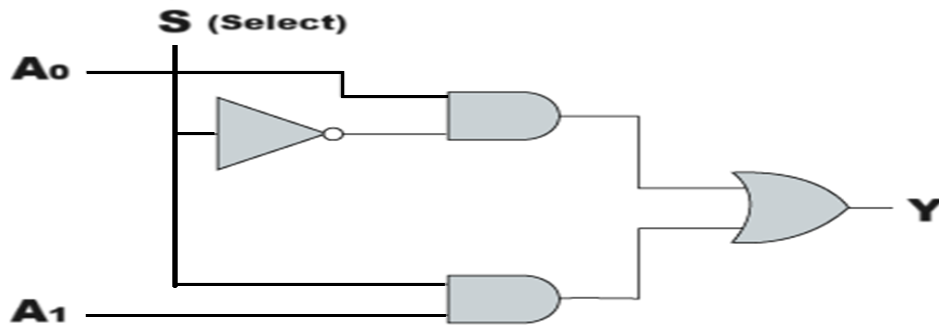
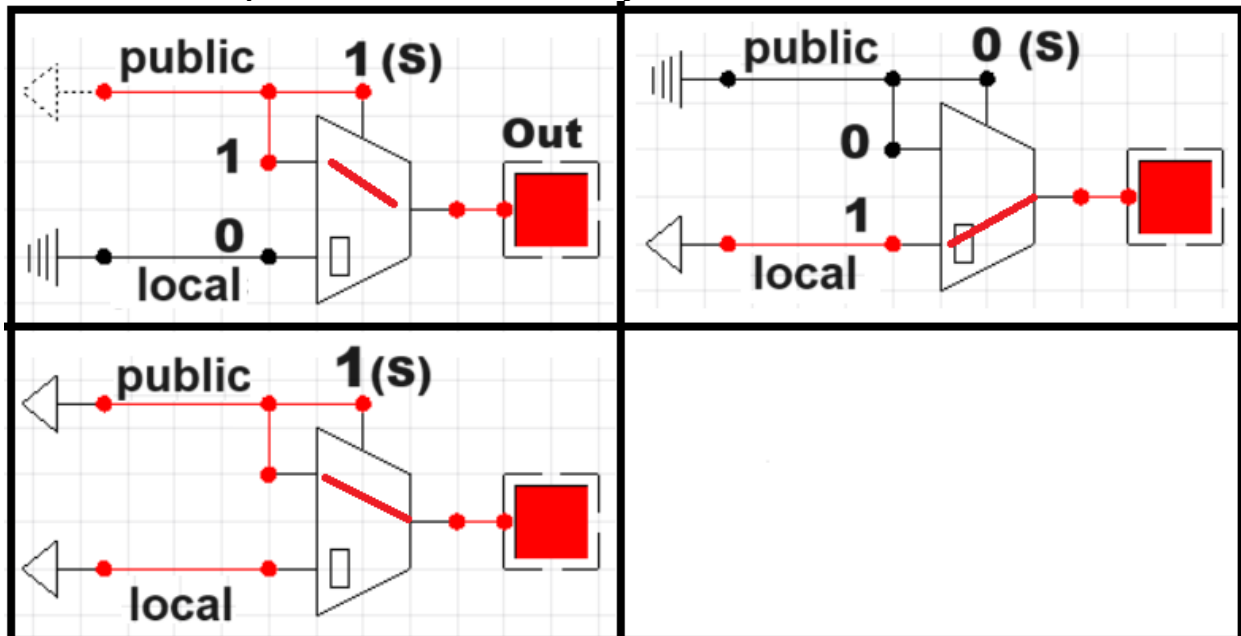


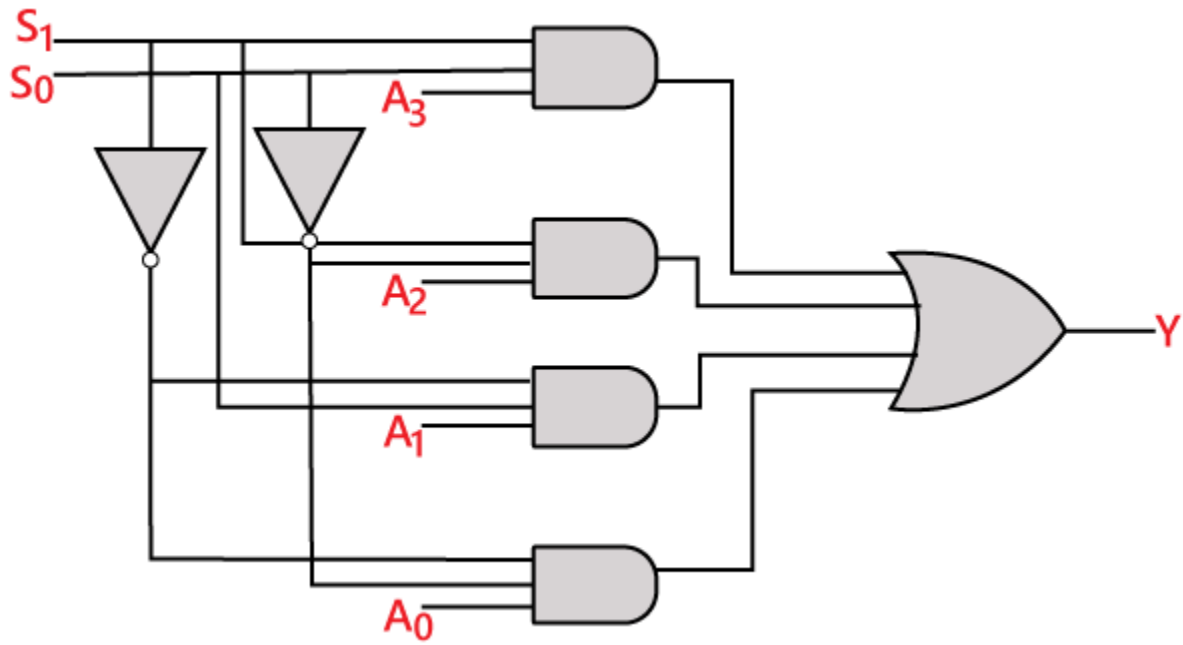
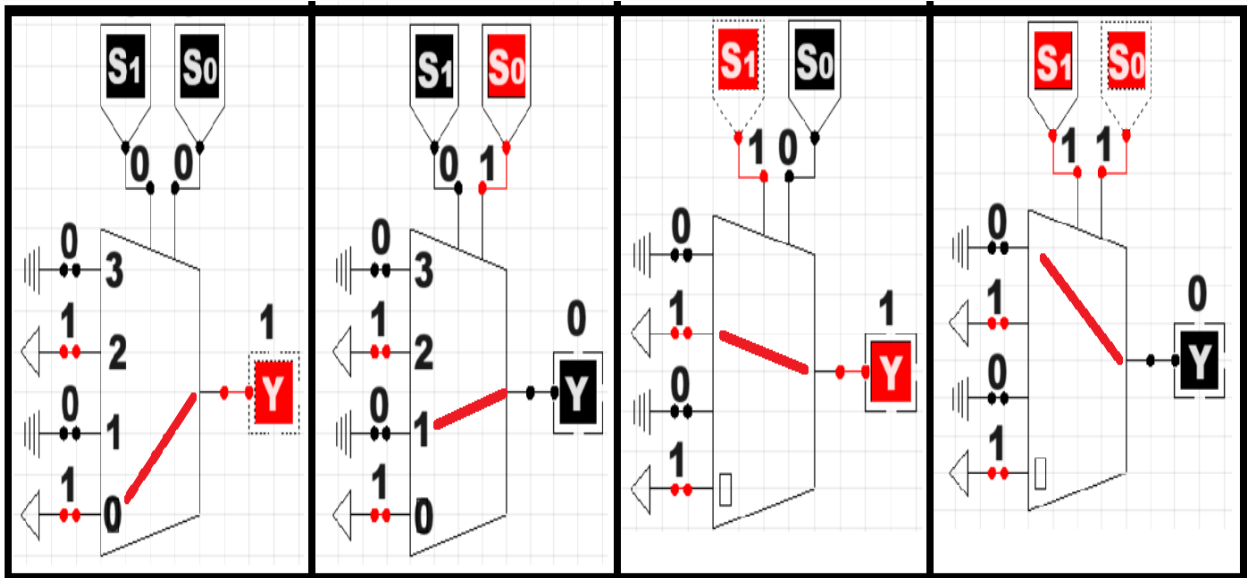
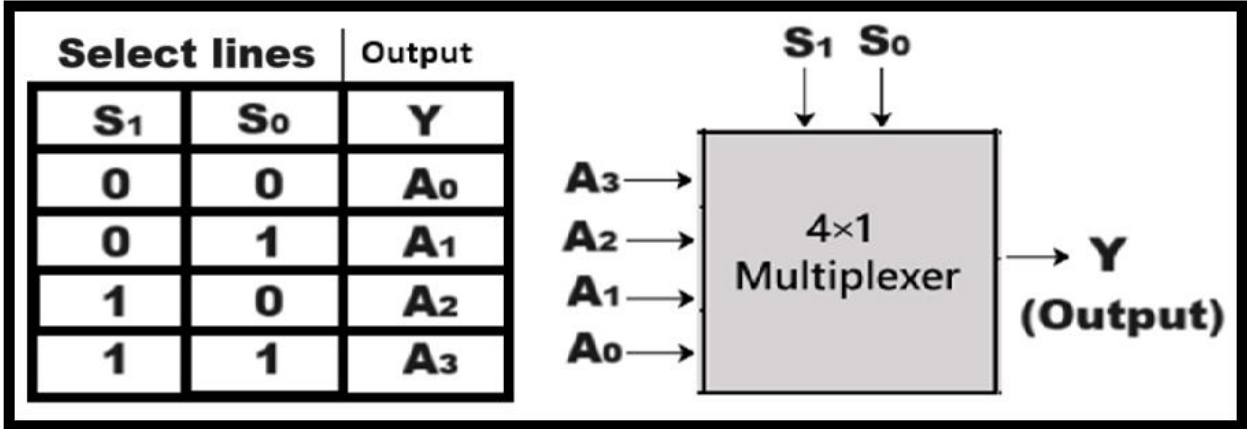
Figure 17: Logic Diagram of 2x1 Multiplexer

Example: Design a logic circuit in which a multiplexer is used as a Changeover switch between public and local electricity.



2) 4x1 Multiplexer

- 4- $2^N = 2^2 = 4$ Input signals.
- 5- $N=2$ select lines (Control lines).
- 6- One output channel.



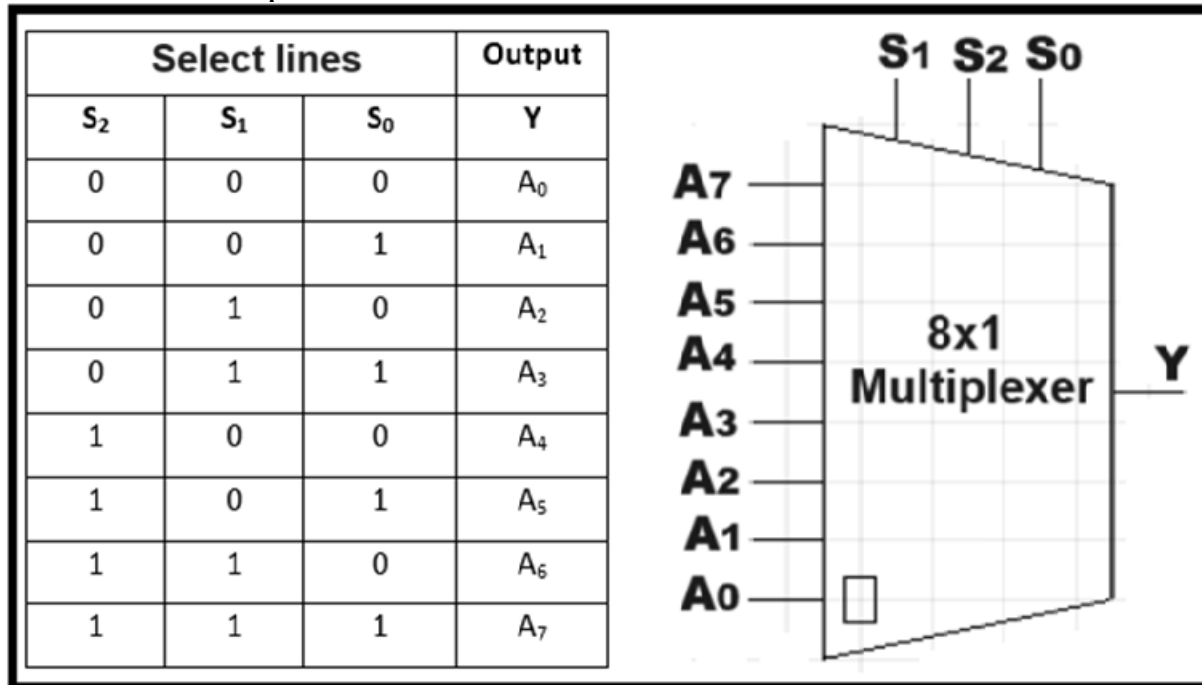
Logic Diagram of 4x1 Multiplexer

3) 8x1 Multiplexer

7- $2^N = 2^3 = 8$ Input signals.

8- $N=3$ select lines (Control lines).

9- One output channel.



Block Diagram of 8x1 Multiplexer

4) 16x1 Multiplexer

10- $2^N = 2^4 = 16$ Input signals.

11- $N=4$ select lines (Control lines).

12- One output channel.

De-multiplexer (DE-MUX) or Data Distributor

A Demultiplexer reverses the function of a multiplexer, taking the digital signal from one channel and distributing it over a certain number of output lines (single-input, multiple-output switch). For this reason, a multiplexer is also known as a data distributor.

Types of Demultiplexer:

- **1x2 Demultiplexer**
- **1x4 Demultiplexer**
- **1x8 Demultiplexer**
- **1x16 Demultiplexer**

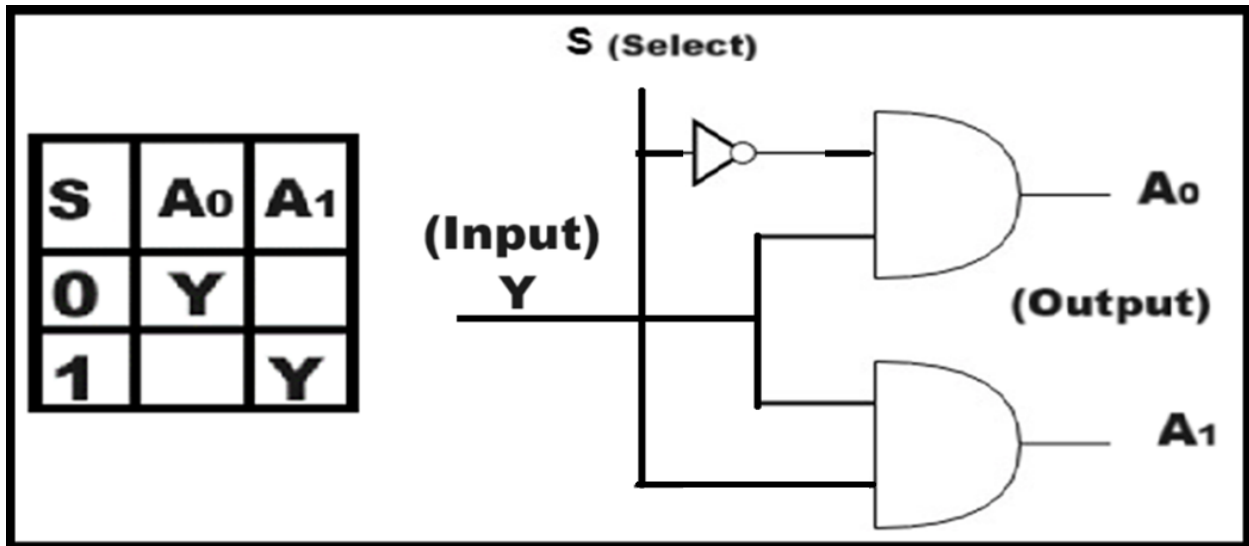


Figure 18: Logical Circuit of 1x2 Demultiplexer.

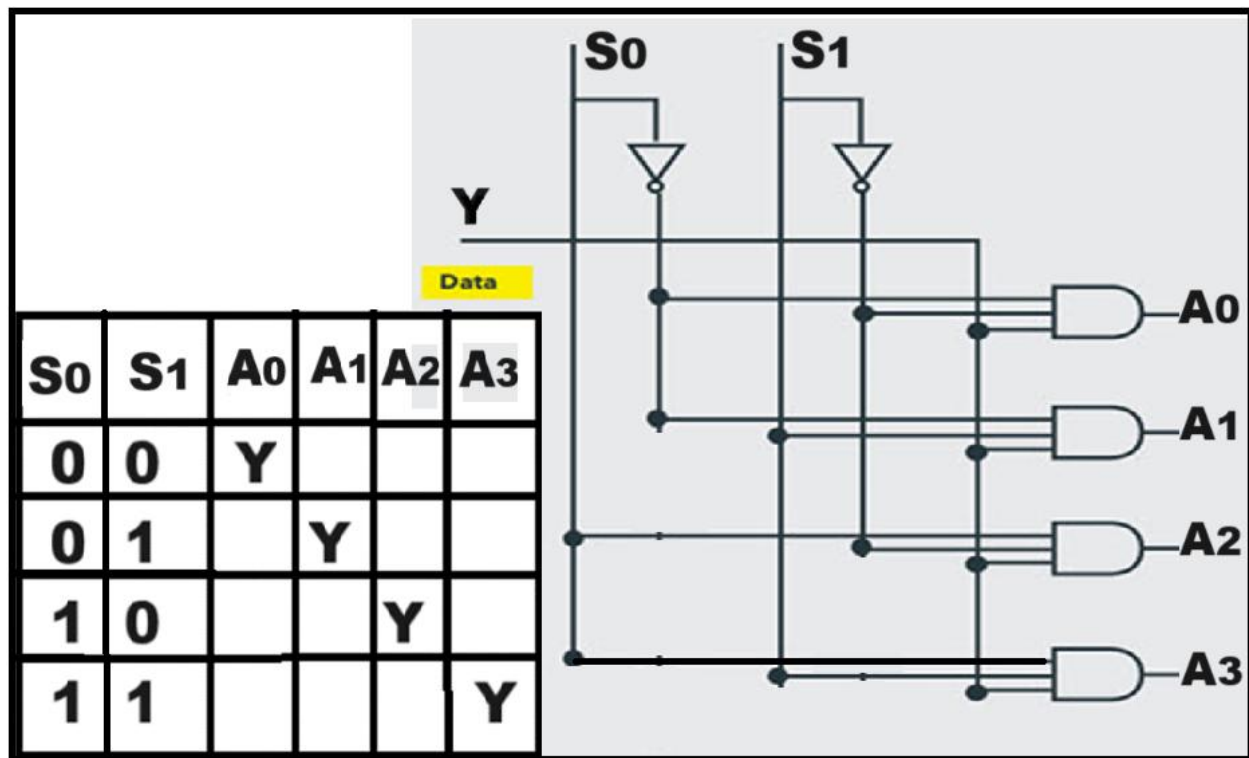
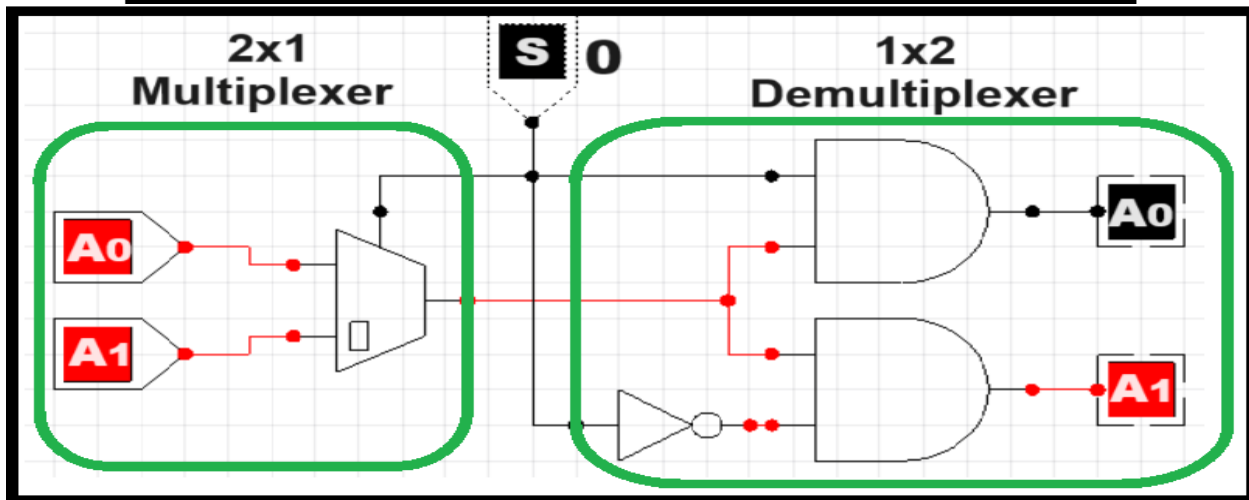
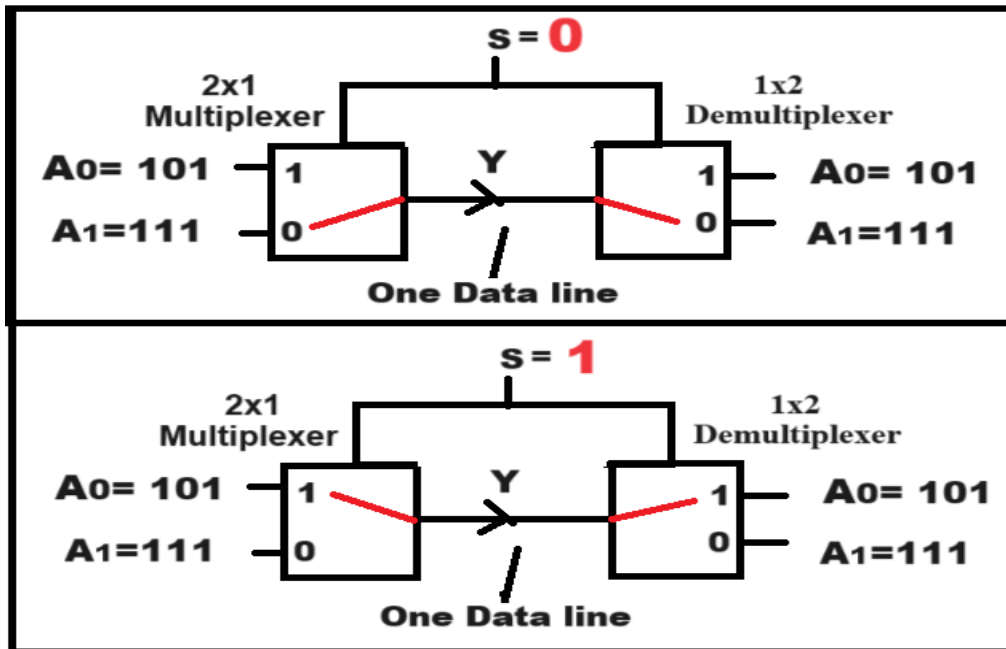


Figure 19: Logical Circuit of 1x4 Demultiplexer.

The basic function of a Multiplexer and Demultiplexer:

A multiplexer combining multiple inputs into a single data stream. On the receiving side, a demultiplexer splits the single data stream into the original multiple signals.



Multiplexer implementation

In MUX, Implementation tables are used to build a logic circuit that fits the number of multiplexer inputs.

Example 1: Use 2x1 MUX to design a logic circuit defined by the following canonical (SOP).

$$F(X,Y) = \Sigma(1,2)$$

Sol:

In 2x1 Multiplexer:

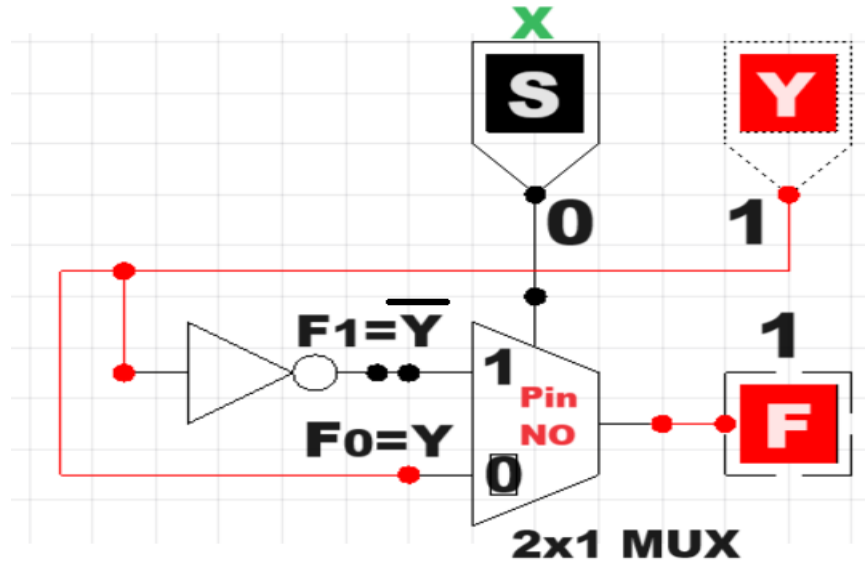
- 13- 1 select line \rightarrow The First one left column (X) is select line.
- 14- 2 Input \rightarrow The table is divided horizontally into 2 sections.

	S In		Out
	X	Y	F
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

Implementation table

	\bar{Y}	Y	
0		1	1
1	1		3

$F_0 = Y$
 $F_1 = \bar{Y}$



Example 2: Use 2x1 MUX to design a logic circuit defined by the following canonical (SOP).

$$F(X,Y,Z) = \sum (0,1,3,6)$$

Sol:

	S input			Out
	X	Y	Z	F
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	0
6	1	1	0	1
7	1	1	1	0

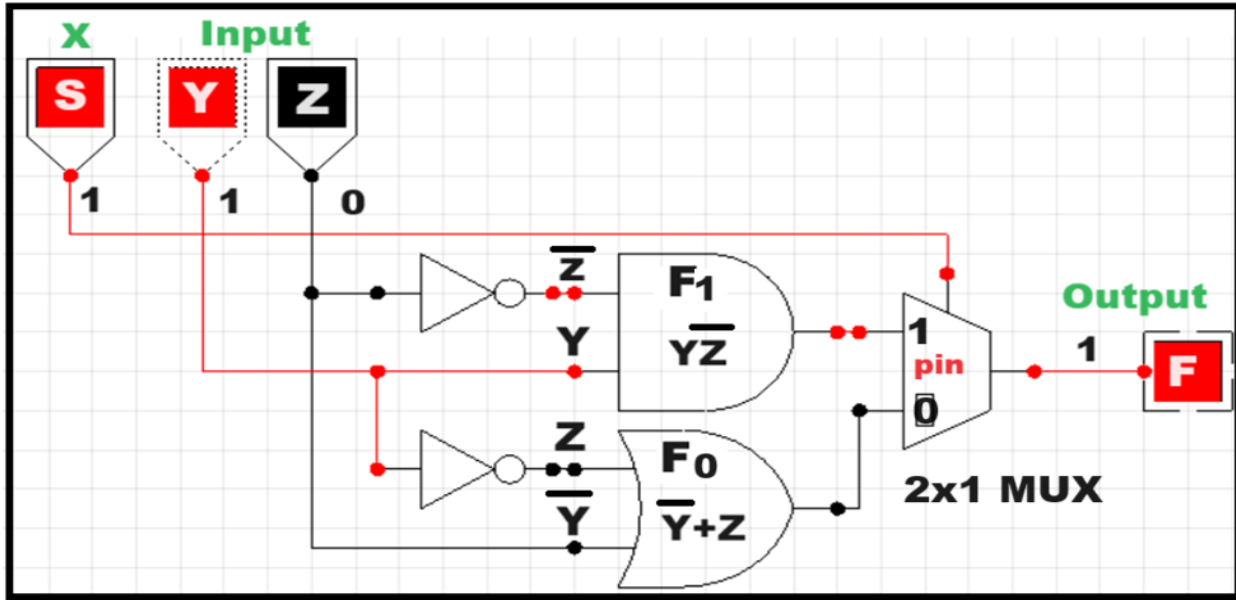
Implementation table

	$\bar{Y}\bar{Z}$	$\bar{Y}Z$	$Y\bar{Z}$	YZ
0	1	1		1
1			1	

$$\begin{aligned}
 F_0 &= \bar{Y}\bar{Z} + \bar{Y}Z + YZ \\
 &= \bar{Y}(\bar{Z} + Z) + YZ \\
 &= \bar{Y}(1) + YZ \\
 &= \bar{Y} + YZ \quad \text{قانون} \\
 &= (\bar{Y} + Y)(\bar{Y} + Z) \\
 &= (1)(\bar{Y} + Z)
 \end{aligned}$$

Pin (1) $F_1 = \bar{Y}\bar{Z}$

Pin (0) $F_0 = \bar{Y} + Z$



Example 3: Use 2x1 MUX to design a logic circuit defined by the following canonical (SOP).

$$F(A,B,C,D) = \sum (0,1,5,6,7,9,10,15)$$

Sol:

	S	Input			Out								
	A	B	C	D	F		\overline{BCD}	$\overline{BC}D$	$B\overline{CD}$		$BC\overline{D}$	BCD	
							000	001	010		110	111	
0	0	0	0	0	1								
1	0	0	0	1	1								
2	0	0	1	0	0	0	1						
3	0	0	1	1	0								
4	0	1	0	0	0								
5	0	1	0	1	1	1							
6	0	1	1	0	1								
7	0	1	1	1	1								
8	1	0	0	0	0								
9	1	0	0	1	1								
10	1	0	1	0	1								
11	1	0	1	1	0								
12	1	1	0	0	0								
13	1	1	0	1	0								
14	1	1	1	0	1								
15	1	1	1	1	0								

	\overline{BCD}	$\overline{BC}D$	$B\overline{CD}$	$BC\overline{D}$	BCD
0	1	1			
1	0	1			
2			1		
3				1	
4					1
5					
6					
7					
8					
9		1	1		
10					
11					
12					
13					
14					
15					

$$F_0 = \overline{BCD} + \overline{BC}D + B\overline{CD} + BC\overline{D}$$

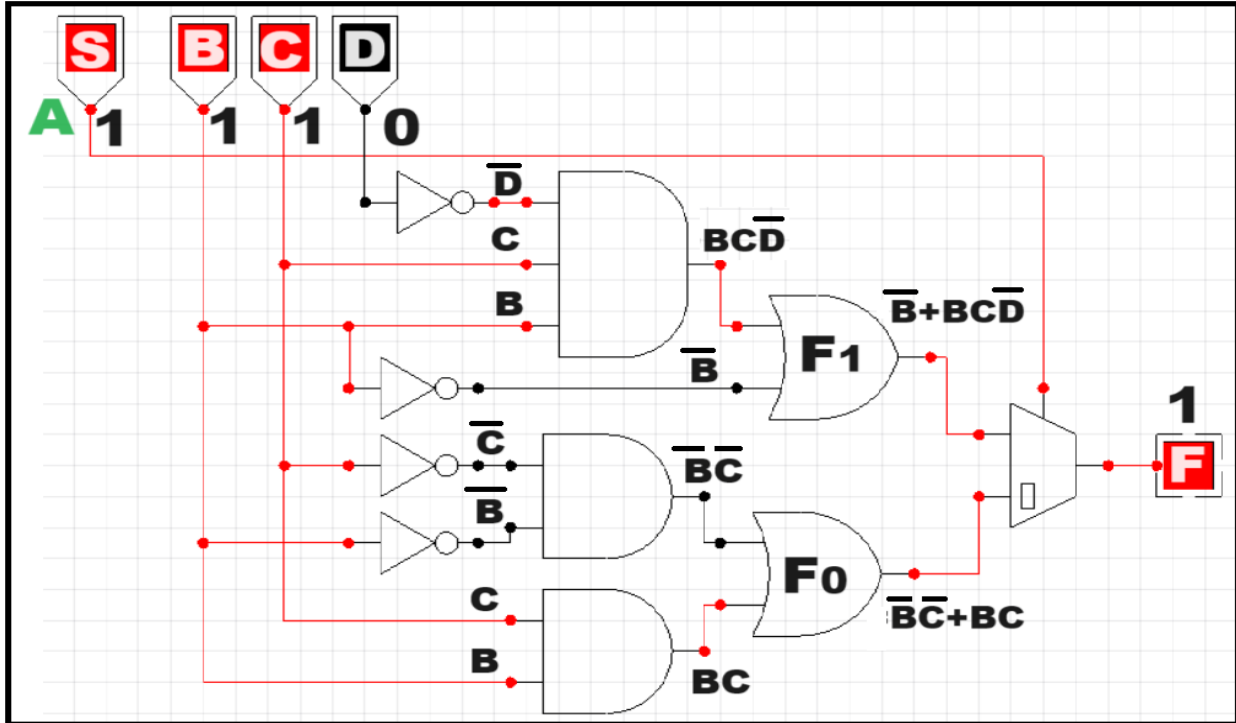
$$= \overline{BC}(\overline{D} + D) + BC(\overline{D} + D)$$

$$F_0 = \overline{BC} + BC$$

$$F_1 = \overline{BCD} + \overline{BC}D + B\overline{CD}$$

$$= \overline{B}(\overline{CD} + C\overline{D}) + B\overline{CD}$$

$$F_1 = \overline{B} + B\overline{CD}$$



Example 4: Use 4x1 MUX to design a logic circuit defined by the following canonical (SOP).

$$F(A,B,C) = \sum(0,1,3,6)$$

Sol:

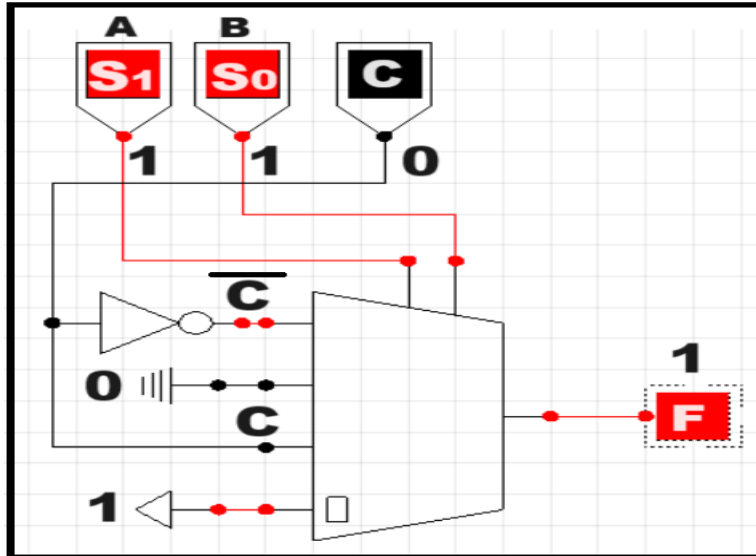
In 8x1 Multiplexer:

15- 2 select line → The first 2 left columns (A, B,) are select line.

16- 4 Input → The table is divided horizontally into 4 sections.

	S1	S0	in	out
	A	B	C	F
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	0
6	1	1	0	1
7	1	1	1	0

Pin NO	\bar{C} 0	C 1	
0	1	1	$\bar{C} + C = 1$
1	0	1	C
2	2	3	0
3	4	5	\bar{C}
	6	7	



Example 5: Use 4x1 MUX to design a logic circuit defined by the following canonical (SOP). $F(A,B,C,D) = \sum (0,1,5,6,7,9,10,15)$

Sol:

	S1	S0	in		out
	A	B	C	D	F
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	0

	$\bar{C}\bar{D}$	$\bar{C}D$	$C\bar{D}$	CD
	00	01	10	11
0	1	1		
1		1	1	1
2		1	1	
3				1
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				

$$F_0 = \bar{C}\bar{D} + \bar{C}D = \bar{C}(\bar{D} + D) = \bar{C}$$

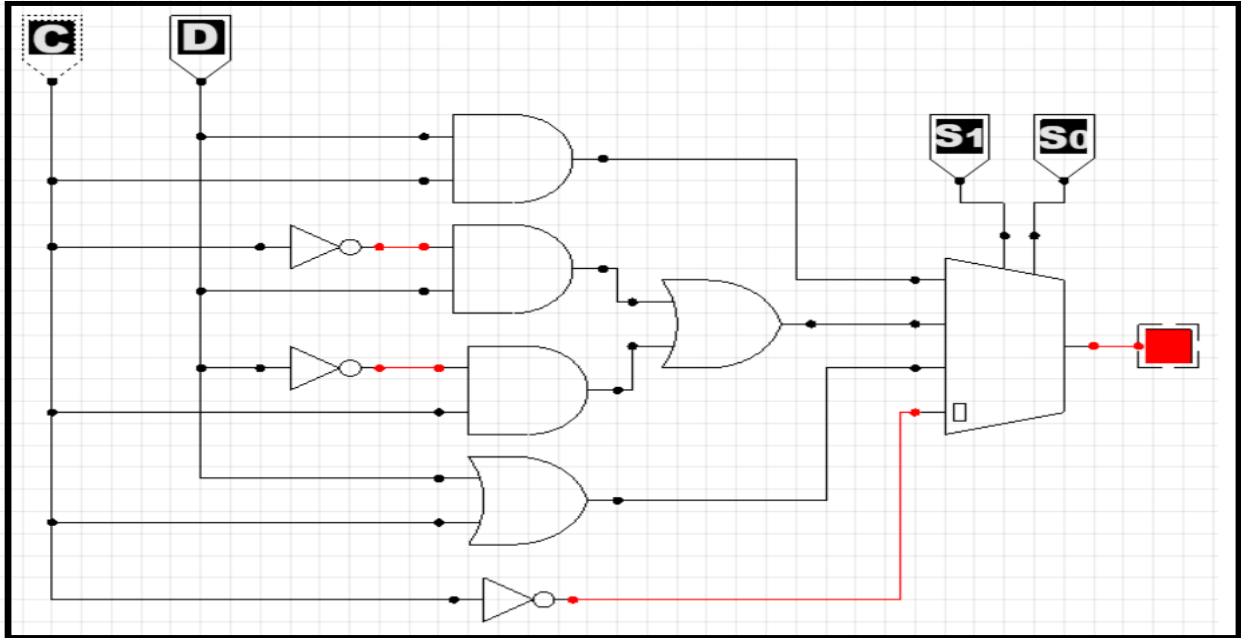
$$F_1 = \bar{C}D + C\bar{D} + CD$$

$$= \bar{C}D + C(\bar{D} + D) = \bar{C}D + C$$

$$= (\bar{C} + C)(C + D) = C + D$$

$$F_2 = \bar{C}D + C\bar{D}$$

$$F_3 = CD$$



Example 6: Use 8x1 MUX to design a logic circuit defined by the following canonical (SOP).

$$F(A,B,C,D) = \sum (0,1,5,6,7,9,10,15)$$

Sol:

In 8x1 Multiplexer:

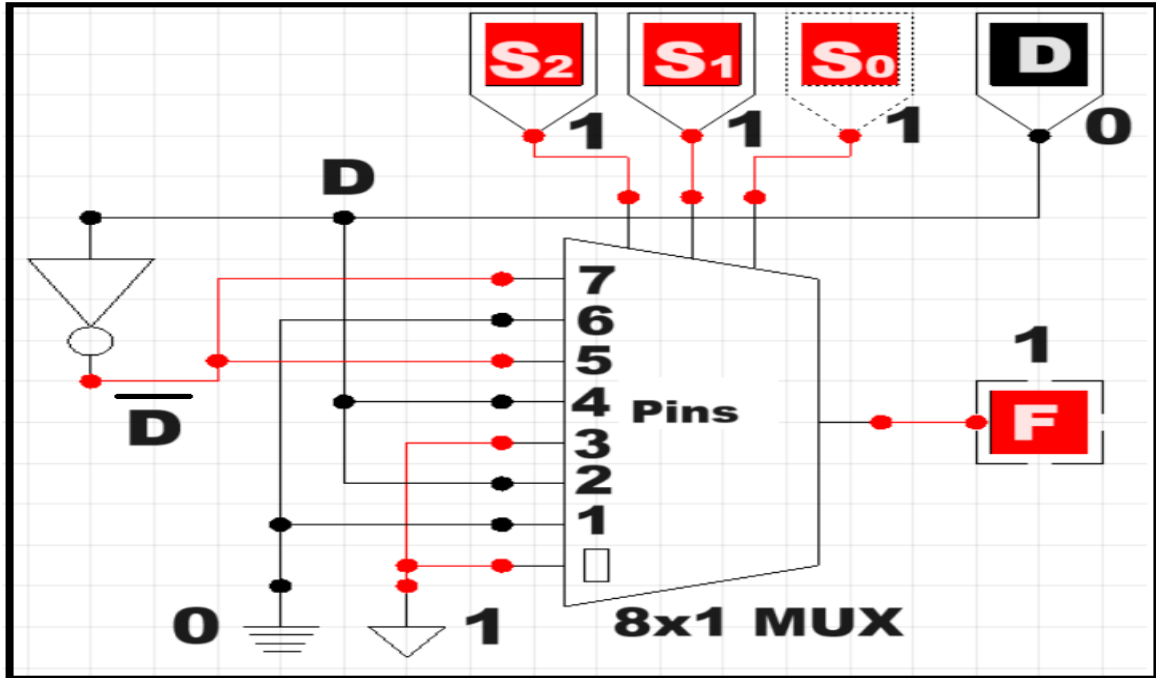
17- 3 select line → The first 3 left columns (A, B, C) are select line.

18- 8 Input → The table is divided horizontally into 8 sections.

	select			D	F
	S2	S1	S0		
	A	B	C		
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	0

Implementation Table

input pin	\overline{D}	D	
	0	1	
1	0	1	0
2	0	0	0
3	1	1	$\overline{D}+D=1$
4	0	1	D
5	1	0	D
6	0	0	0
7	1	0	D

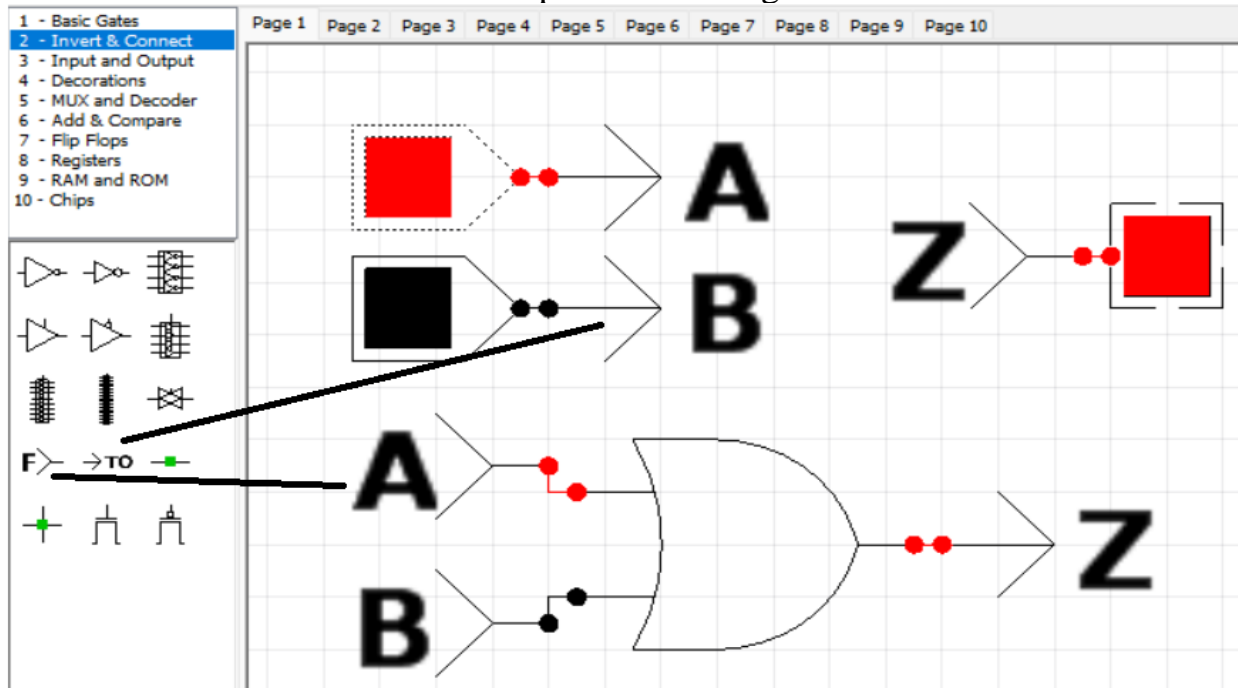


A Block diagram:

is a diagram of a system in which the principal parts or functions are represented by blocks connected by lines that show the relationships of the blocks.

"TO" Named Link & "From" Named Link

It is used to connect the lines of the parts of the diagram to each other.



Time Interval:

In the context of oscilloscopes(O-Scope), is the time difference between two specific events captured during waveform analysis. These events could be voltage transitions, current changes, or any other signal characteristic that you are observing.

The screenshot shows the CEDAR Logic Simulator interface. The 'View' menu is open, and the 'Oscope' option is selected. The circuit diagram shows two logic gates: an AND gate and an OR gate. The AND gate has inputs A and B, and output Z. The OR gate has inputs A and B, and output Z. The circuit is connected to an O-Scope window.

The O-Scope window displays the following signal lines:

A	A - Time signal line	0
B	B - Time signal line	0
Z	Z - Time signal line	0

Arrows point to the dropdown menus for A, B, and Z, with a label "Selecting inputs and outputs".

Below the O-Scope window, two circuit diagrams are shown with their corresponding truth tables:

Left Diagram: A and B are both 1. The AND gate output Z is 1.

A	1
B	0
Z	1

Right Diagram: A is 0, B is 1. The OR gate output Z is 1.

A	0
B	1
Z	1

Latches:

A latch is a digital memory circuit that stores one bit of data. Data remains in memory as long as the power is not turned off.

Type of Latches:

There are basically four main types of latches.

- 1- SR-latch
- 2- D-Latch
- 3- JK-Latch
- 4- T- Latch

SR-latch (Set-Reset Latch):

There are two types of SR-latches, the first is Low Active SR-latch and the second is High Active SR-latch.

Low Active SR-latch

It consists of two **NAND** gates coupled with feedback arrangement where the outputs are connected back to the opposite inputs (Bistable Multivibrator).

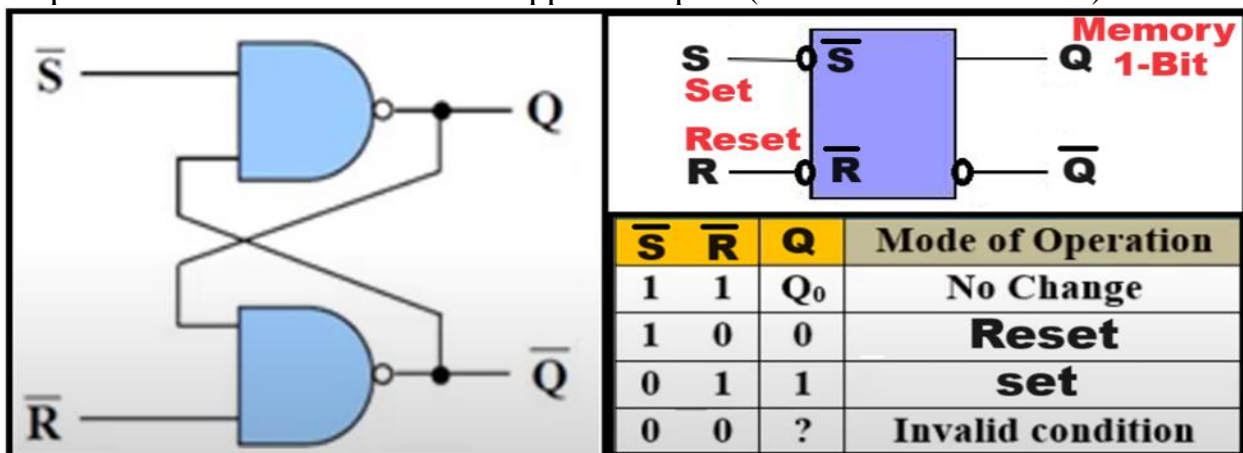
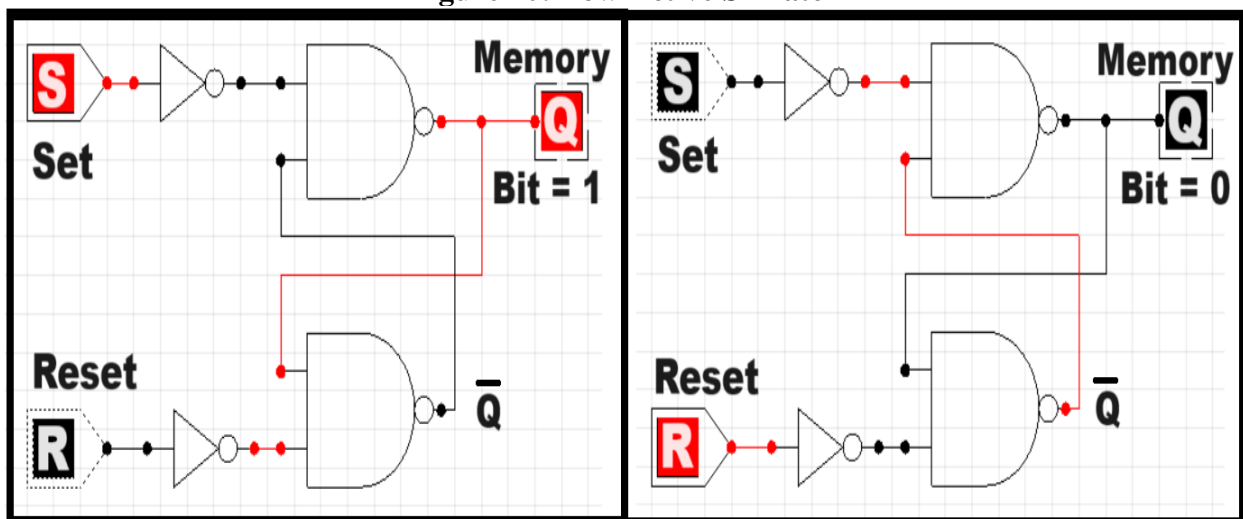


Figure 20: Low Active SR-latch



High Active SR-latch

It consists of two **NOR** gates coupled with feedback arrangement where the outputs are connected back to the opposite inputs (Bistable Multivibrator).

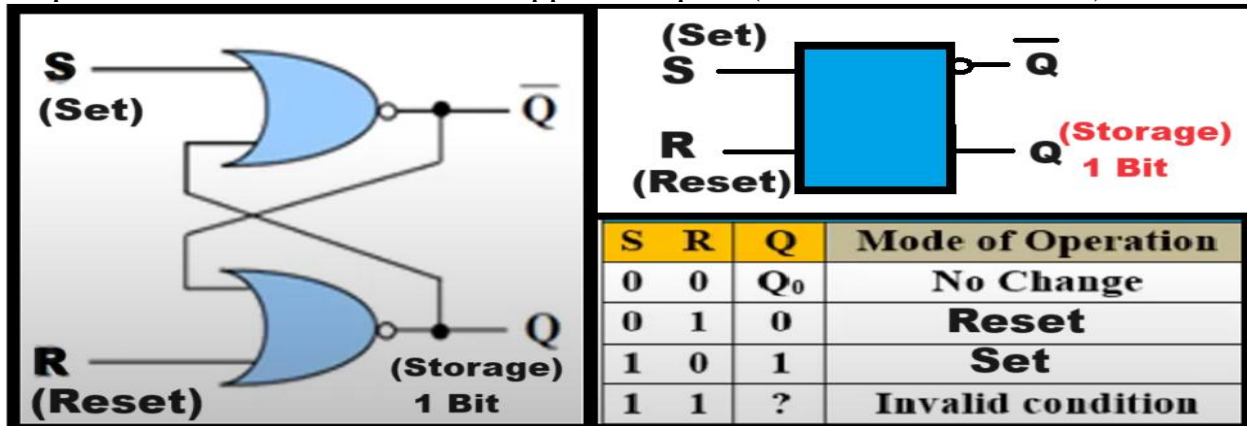
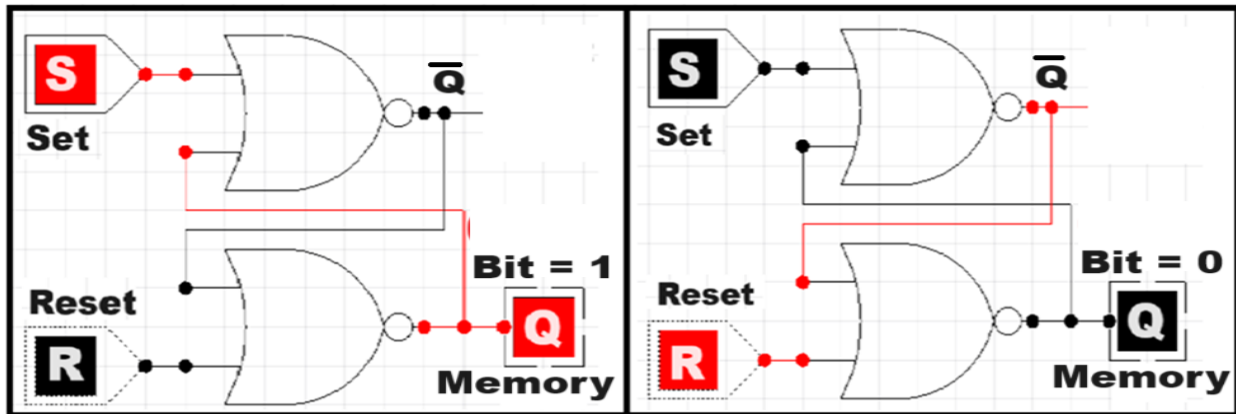
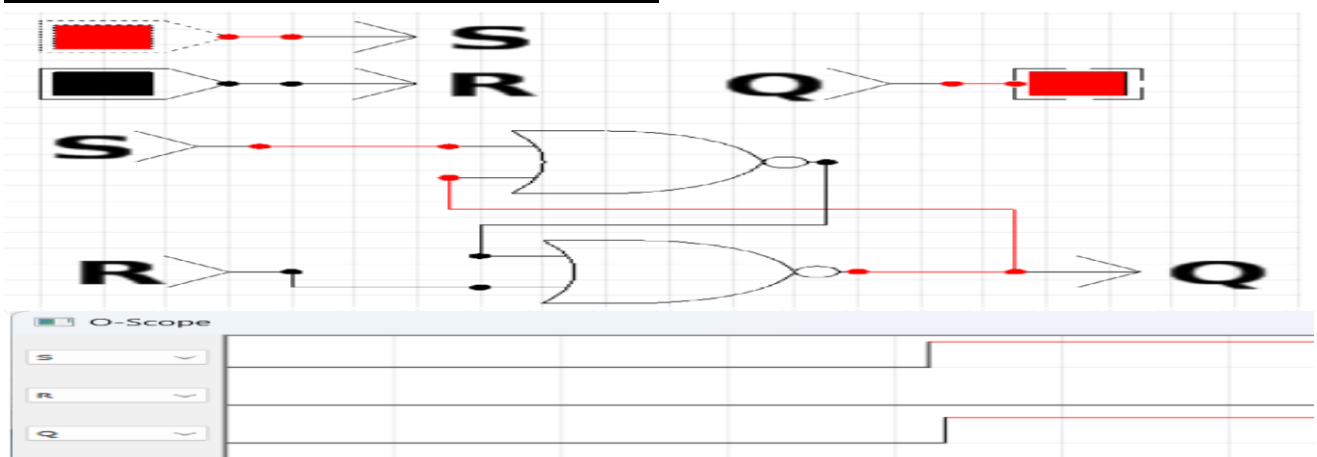


Figure 21: Active High Active SR-latch



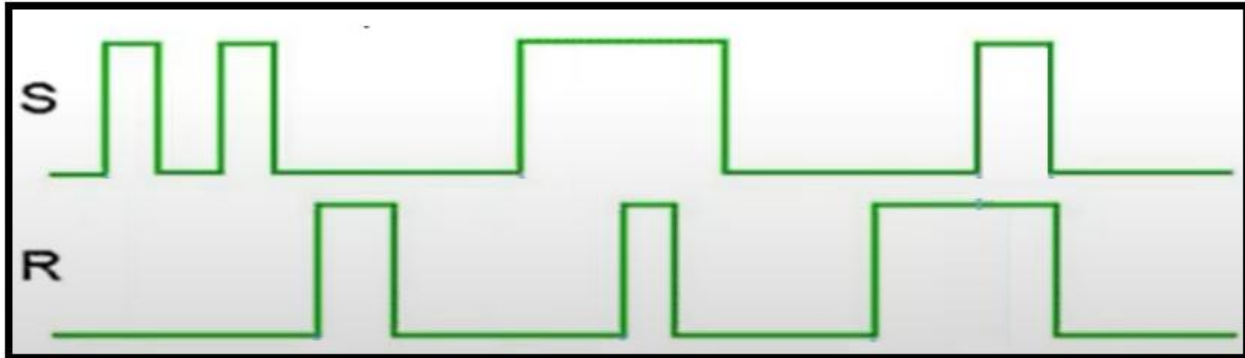
Timing Diagram of High Active SR-latch



Timing Diagram

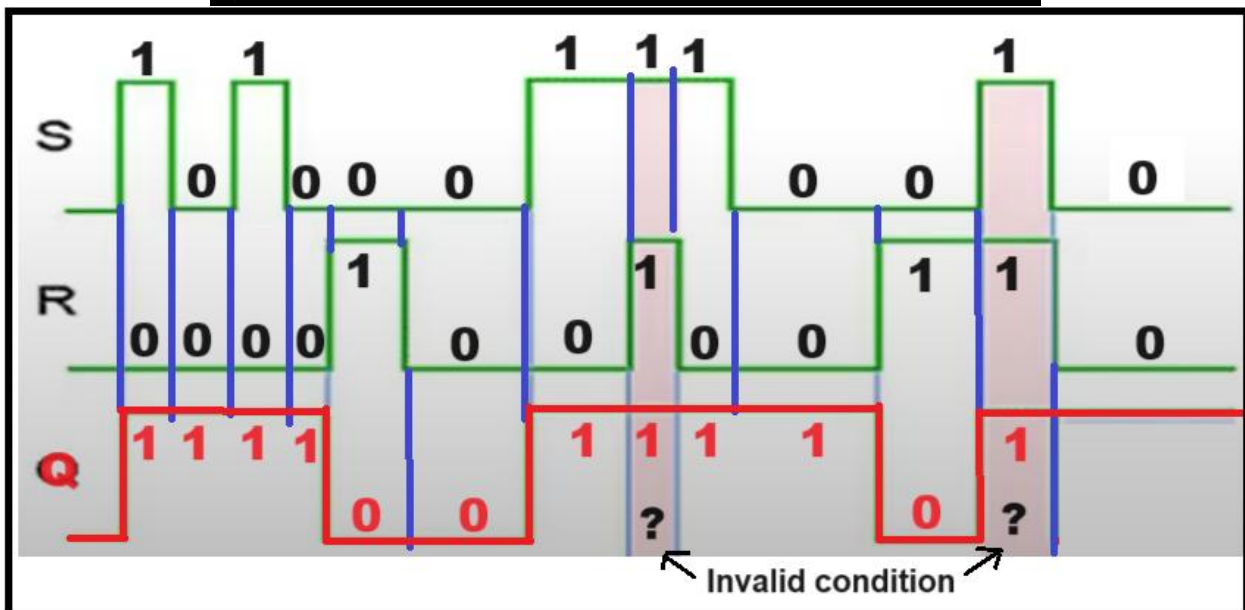
Example 1:

If the waveforms in the given timing diagram are applied to a High Active SR-latch, draw the resulting Q output waveform in relation to the inputs.



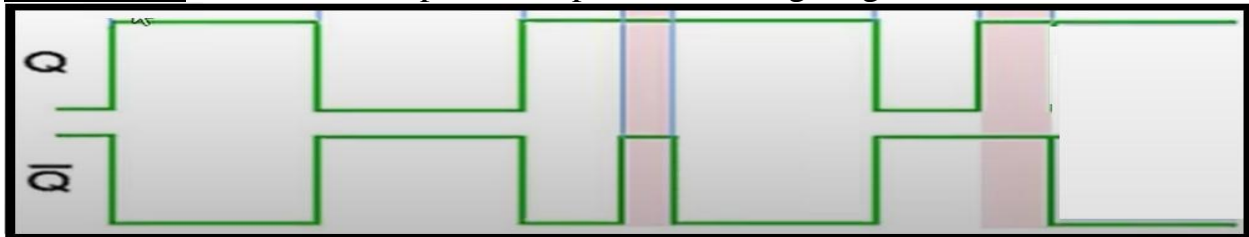
Sol:

S	R	Q	Mode of Operation
0	0	Q_0	No Change
0	1	0	Reset
1	0	1	Set
1	1	?	Invalid condition



Note: In timing diagram of High Active SR-latch start, the S, R inputs start from the low level (0) and the Q output start from the high level (1).

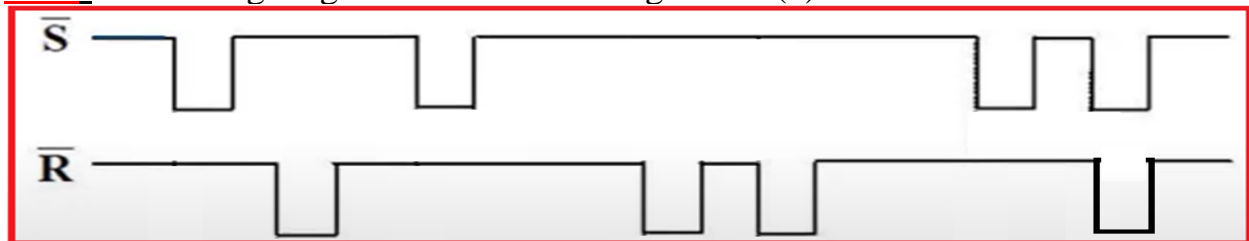
Example 2: Draw \overline{Q} output of the previous timing diagram.



Example 3:

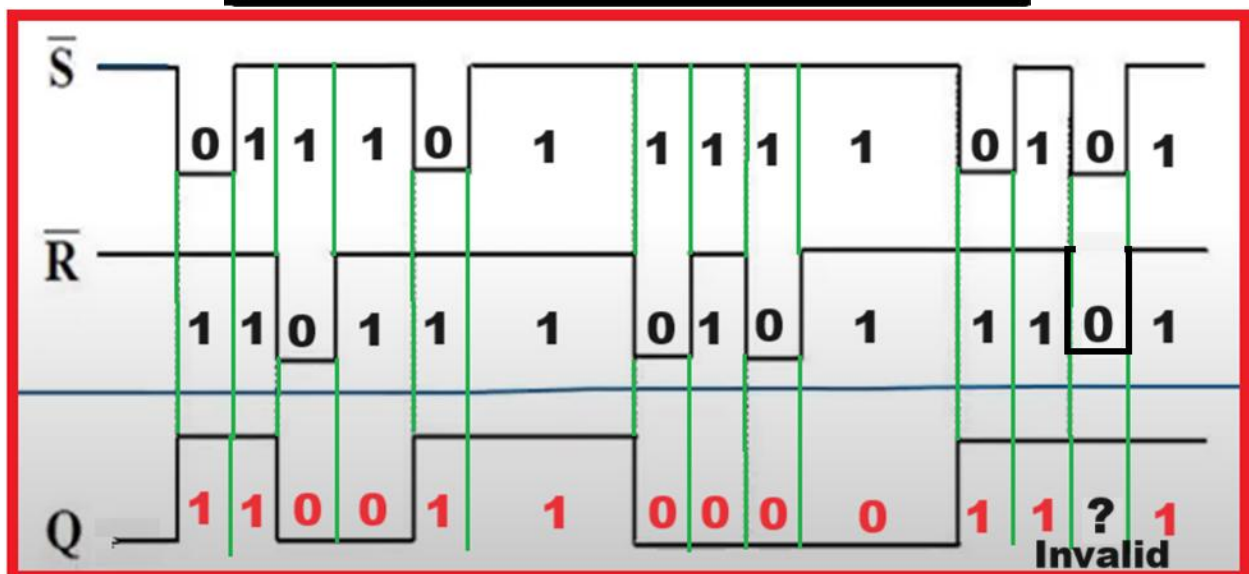
If the waveforms in the given timing diagram are applied to a Low Active SR-latch, draw the resulting Q output waveform in relation to the inputs.

Note: The timing diagram start from the high level (1).



Sol:

\overline{S}	\overline{R}	Q	Mode of Operation
1	1	Q_0	No Change
1	0	0	Reset
0	1	1	set
0	0	?	Invalid condition

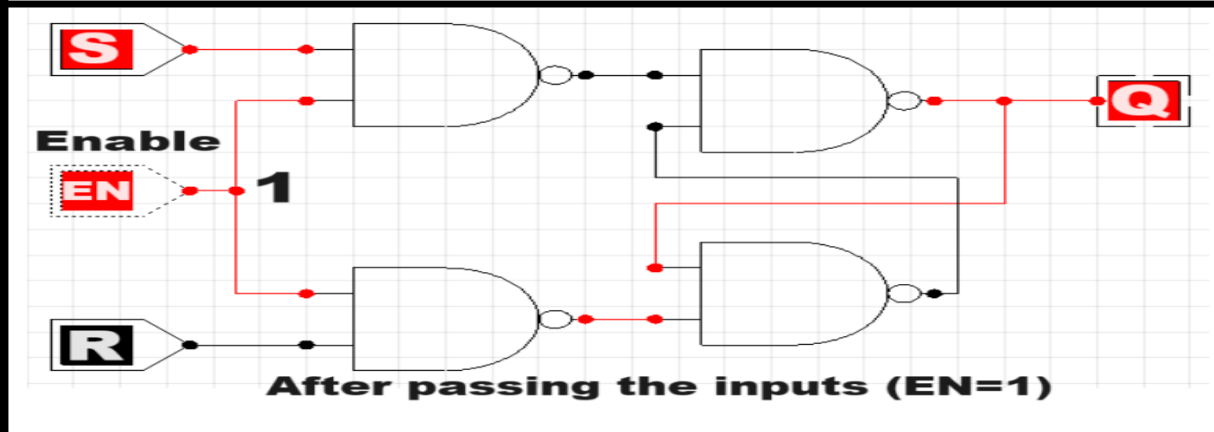
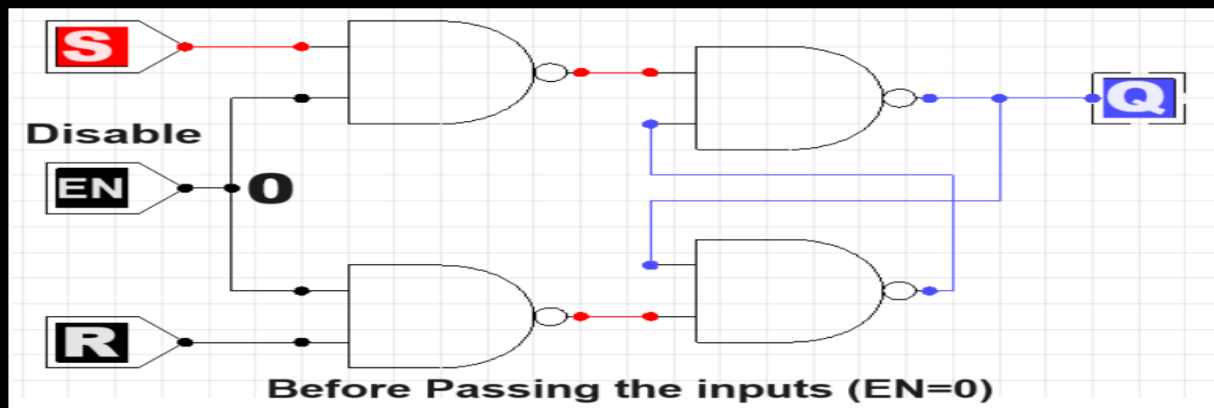
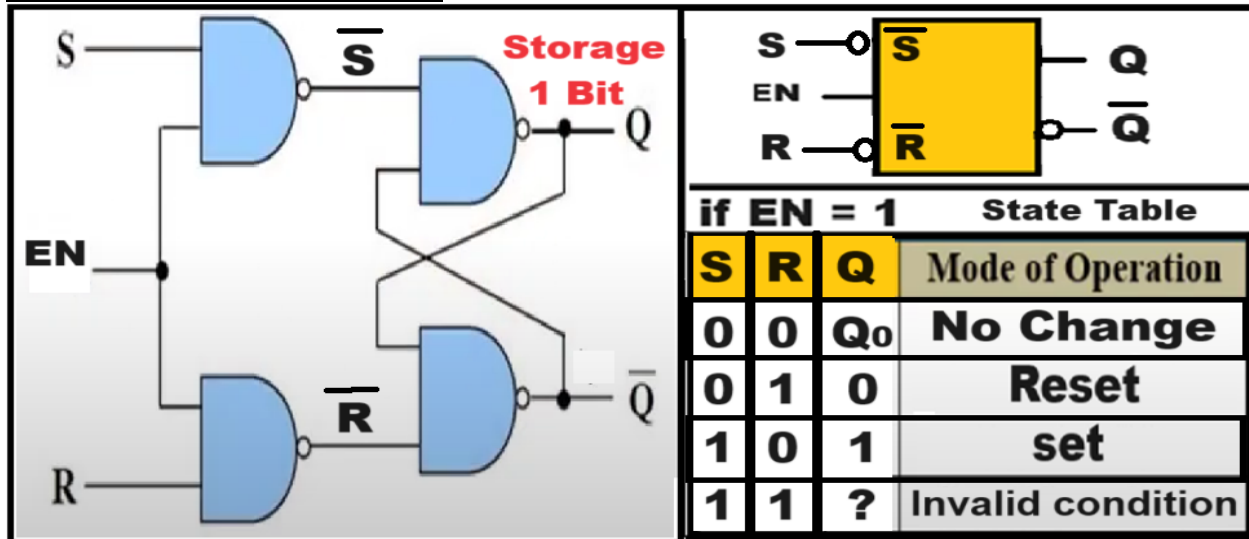


Note: In timing diagram of Low Active SR-latch start, the S, R inputs start from the high level (1) and the Q output start from the low level (0).

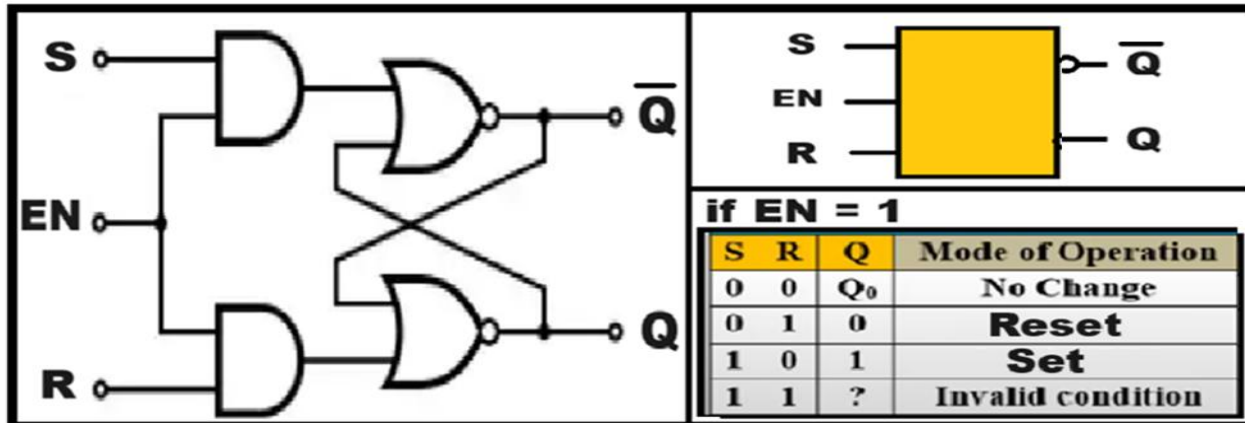
Gated SR-Latch (مقيد):

A gated SR latch requires an Enable (EN) signal to pass the input forward. (Enable "EN=1", Disable "EN=0"). There are two types of Gated SR-latches, the first is Low Active Gated SR-latch and the second is High Active Gated SR-latch.

Low Active Gated SR-latch



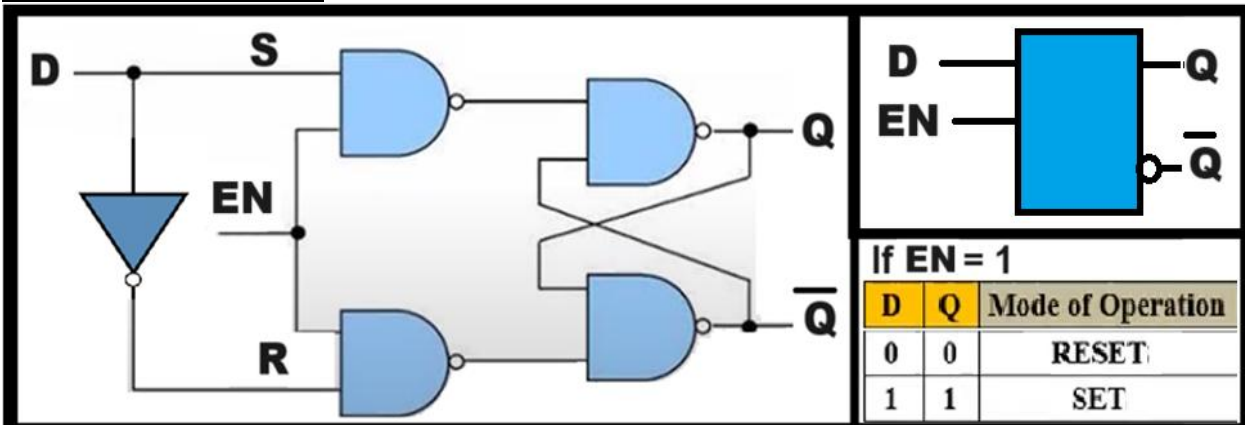
High Active Gated SR-latch



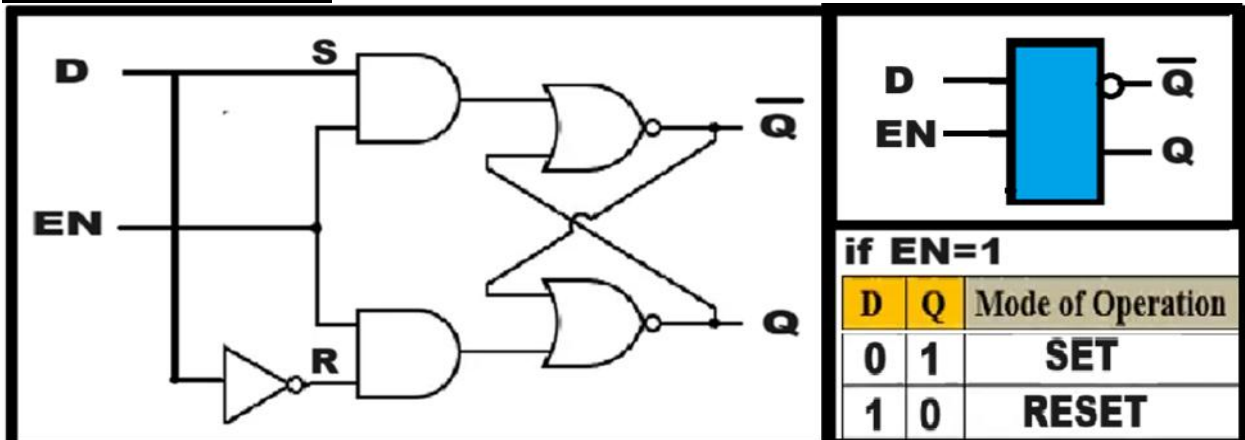
D-Latch (Data Latch):

It is a Gated SR-latch that has only one Data input (D) with EN. This Data input is called D input. When the D input is High (1) and the EN is High (1), the latch will set. When the D input is Low (0) and EN is High (1), the latch will reset.

Low Active D-latch:



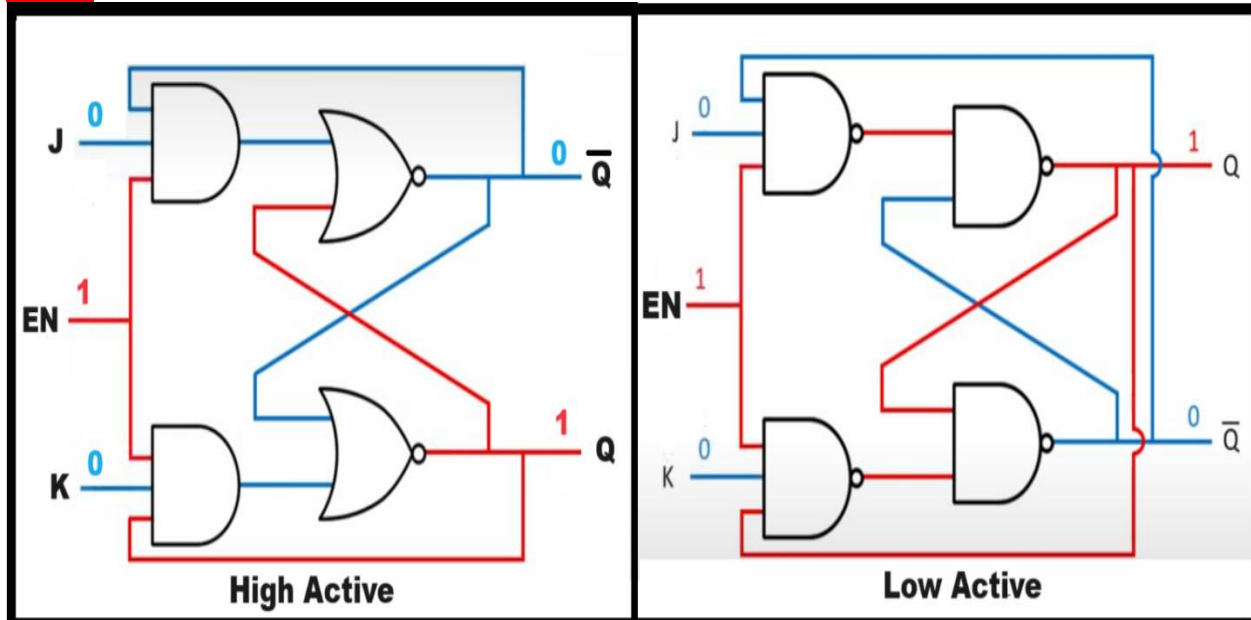
High Active D-latch:



JK-Latch

This latch is designed to solve the "invalid condition " problem of the SR latch which occurs when $S=1$ and $R=1$. The latch complements the output in state of "invalid condition " ($J=1, K=1$)

Note: $J=S$ and $K=R$.



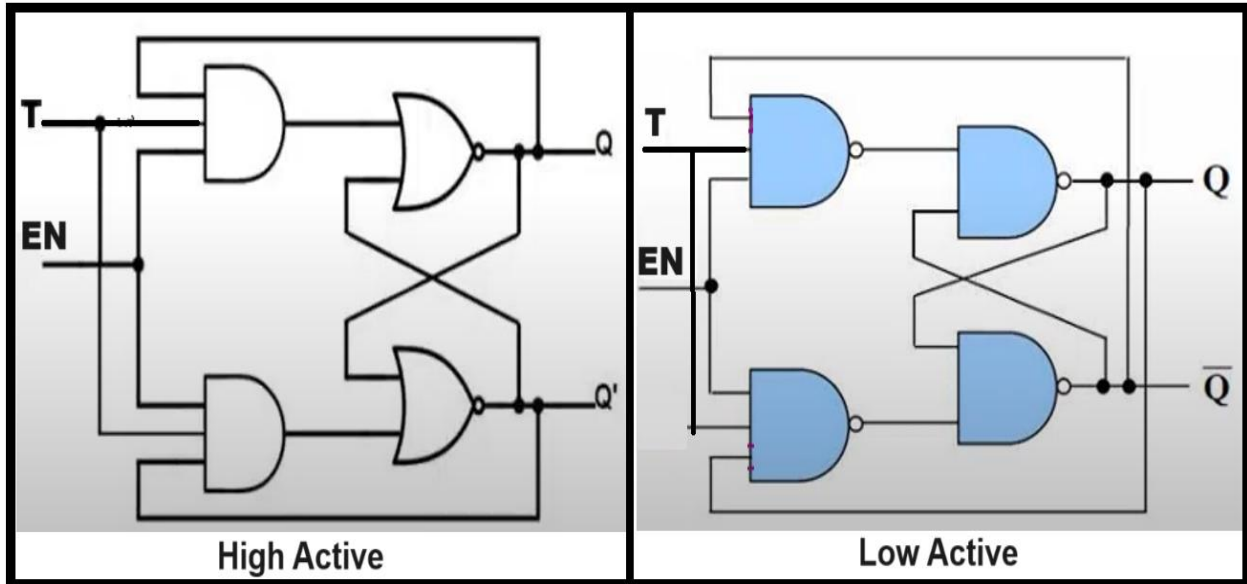
If $EN = 1$

Inputs		Outputs	
J	K	Q	Comments
0	0	Q	No change
0	1	0	Reset
1	0	1	Set
1	1	\overline{Q}	Output change

State Table of Low Active JK-Latch.

T-Latch (Toggle Lath)

It is a JK-latch that has only one input (T) with EN. This latch is called a Toggle latch because of its ability to complement the output of its state.



State Table of Low Active T-Latch.

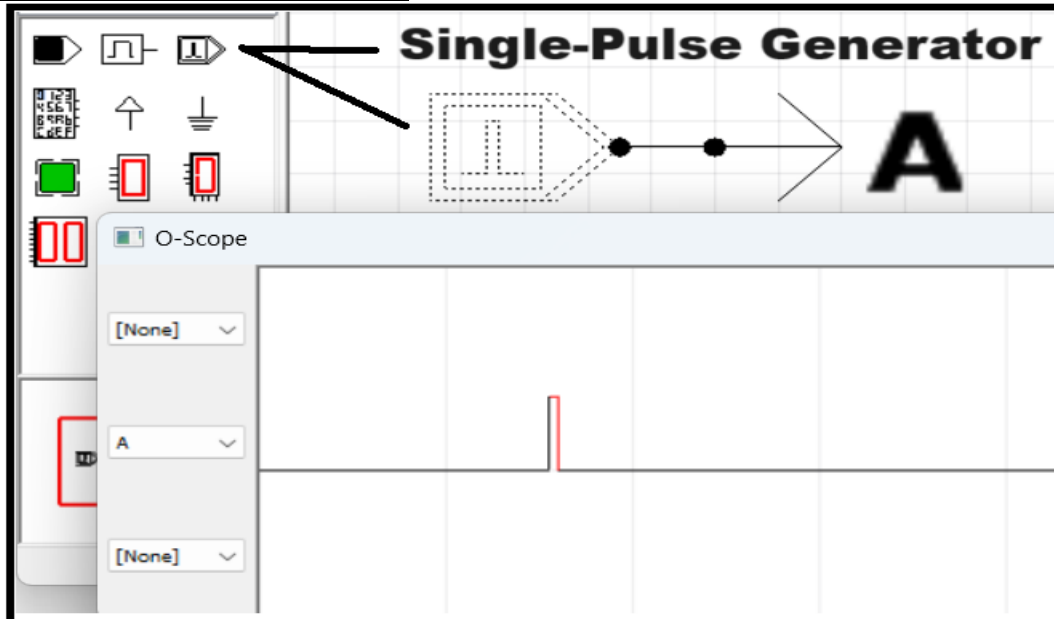
if EN = 1

T	Q	Mode of Operation
0	Q	No Change
1	\overline{Q}	Output change

Low Active T-Latch.

Sequential Logic Circuits.

Single-Pulse Generator (CLK):

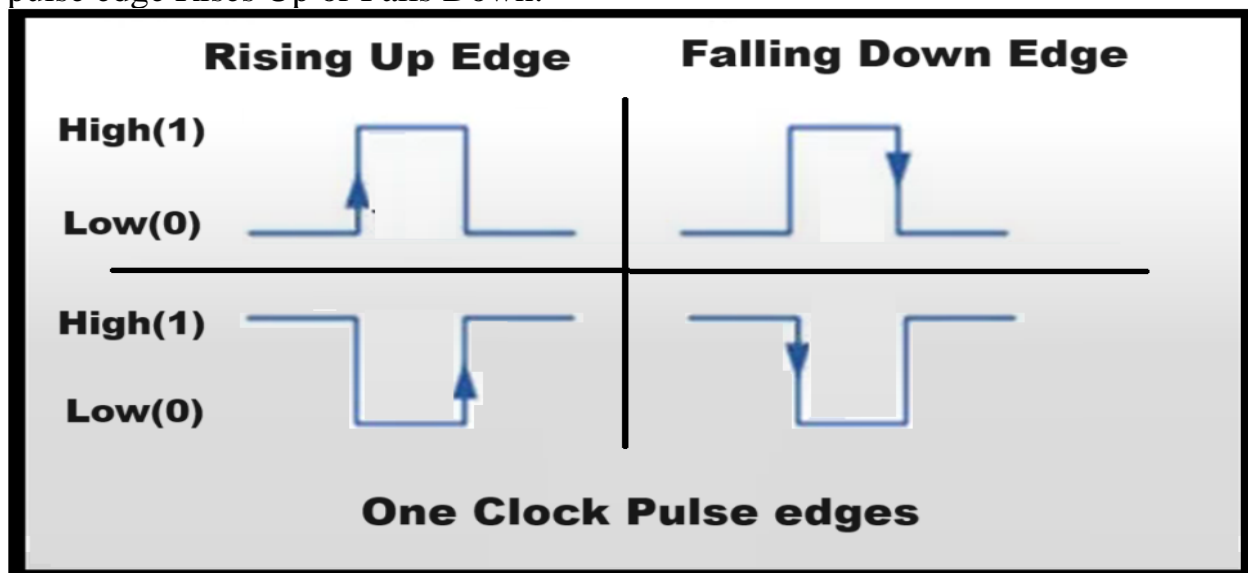


Flip Flops

The Flip Flop is a latch with a Clock (CLK or CK) instead of Enable (EN). The clock is used to synchronize the forward pass of inputs with each clock pulse.

Clock Pulse:

A single clock (CK) pulse contains two edges, a Rising Up edge (positive) and a Falling Down edge (negative). The input is passed forward either when the pulse edge Rises Up or Falls Down.



Type of Flip Flops:

- SR-Flip Flop (SR-FF)
- D-Flip Flop (D-FF)
- JK-Flip Flop (JK-FF)
- T-Flip Flop (T-FF)

SR-Flip Flop (SR-FF)

It is a Low Active Gated SR-latch with a Clock (CK).

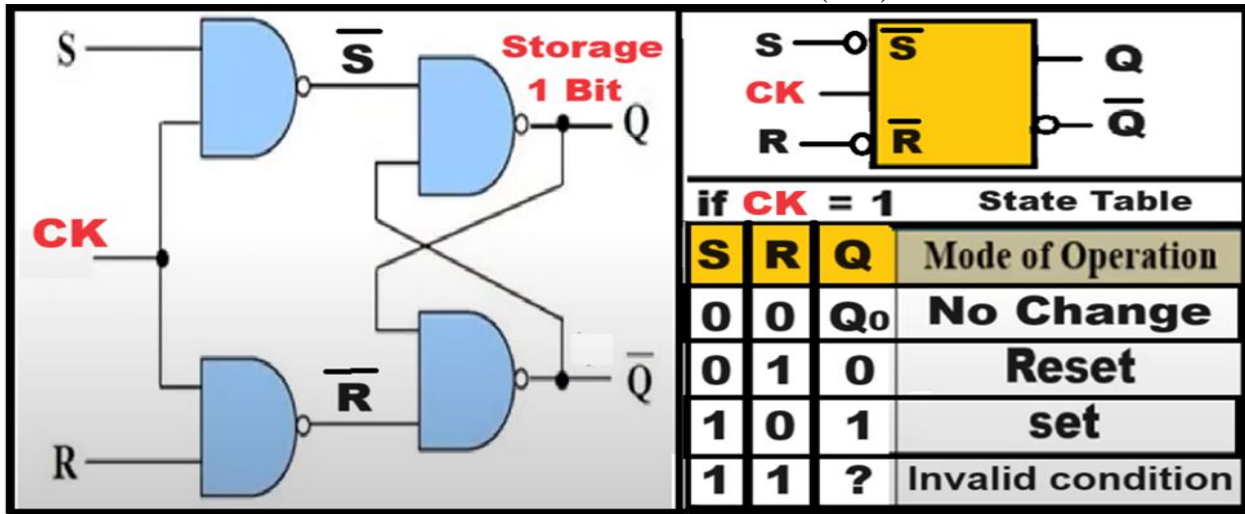
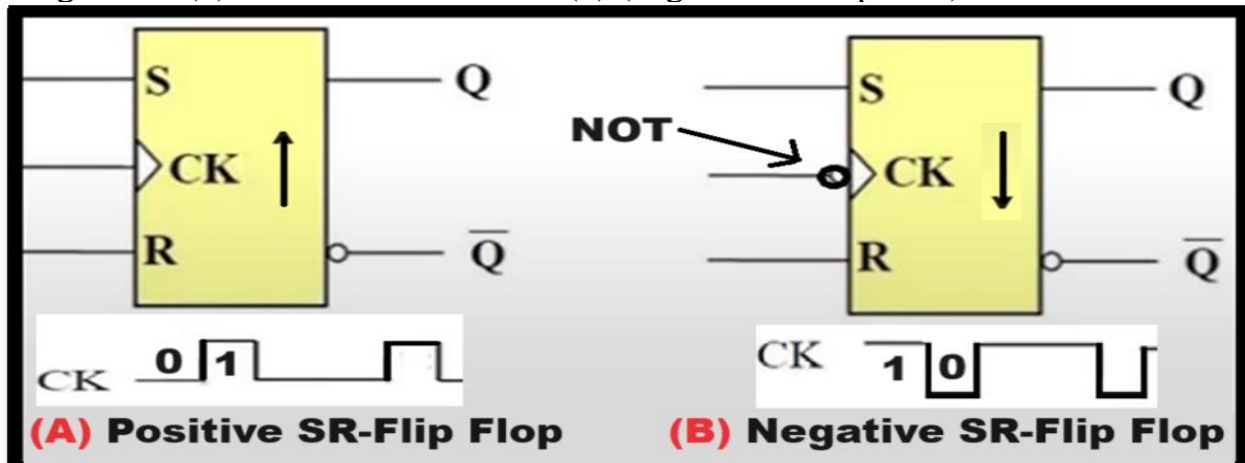


Figure 22: Logical Circuit of SR-Flip Flop

Logic symbol of SR-Flip Flops:

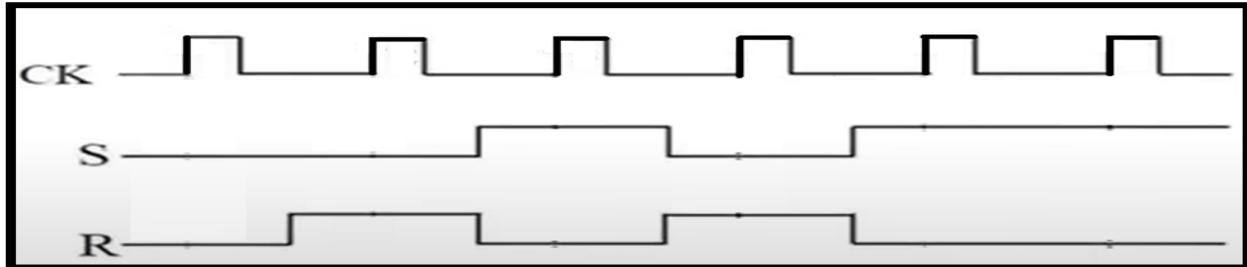
In Figure (A) Positive SR-FF, the clock pulse (CK) starts at a low level (0) and then rises to a higher level (1) (positive clock pulses). In Figure (b) Negative SR-FF, the complement of the clock pulses is taken, where the clock pulse starts at a high level (1) and then decreases to (0) (negative clock pulses).



Timing Diagram

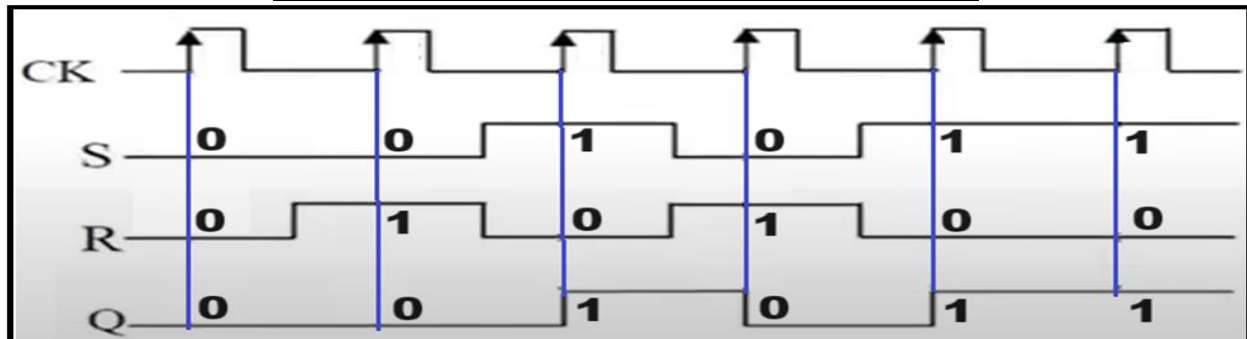
Example 1:

Draw the output shape of SR-FF in case the edge is up.



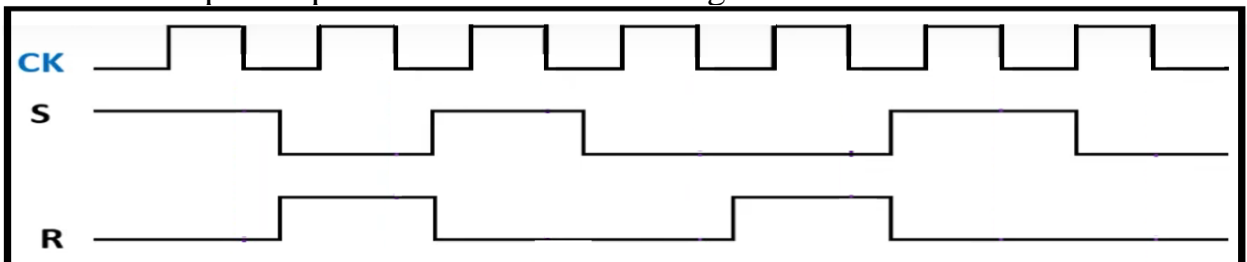
Sol:

S	R	Q	Mode of Operation
0	0	Q ₀	No Change
0	1	0	Reset
1	0	1	set
1	1	?	Invalid condition



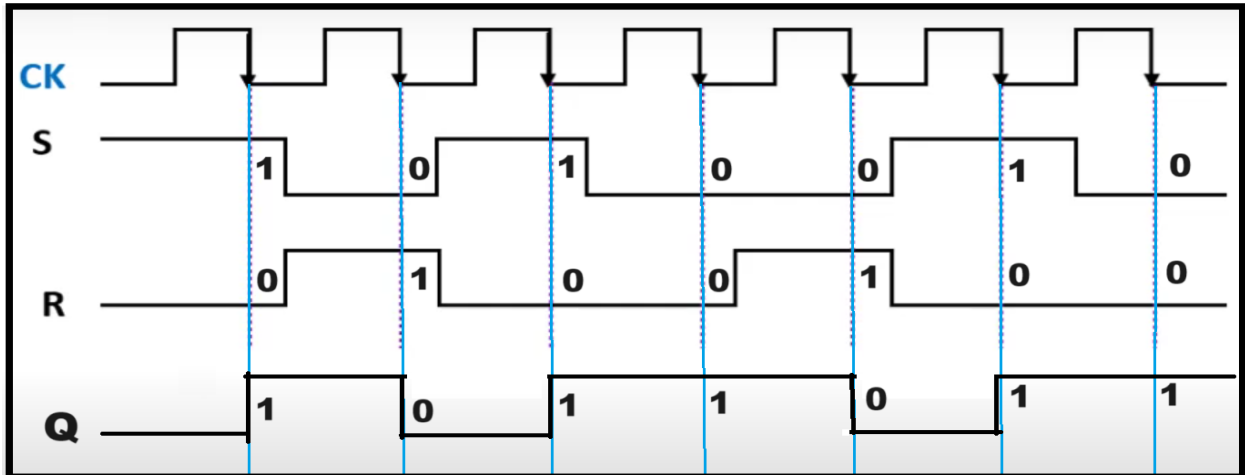
Example 2:

Draw the output shape of SR-FF in case the edge is down.



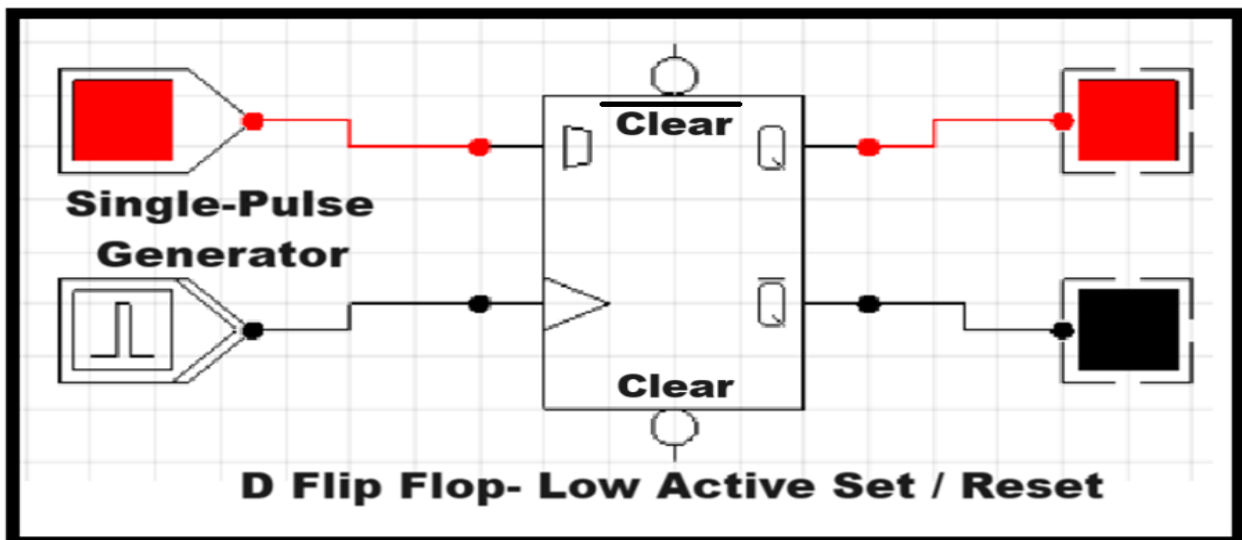
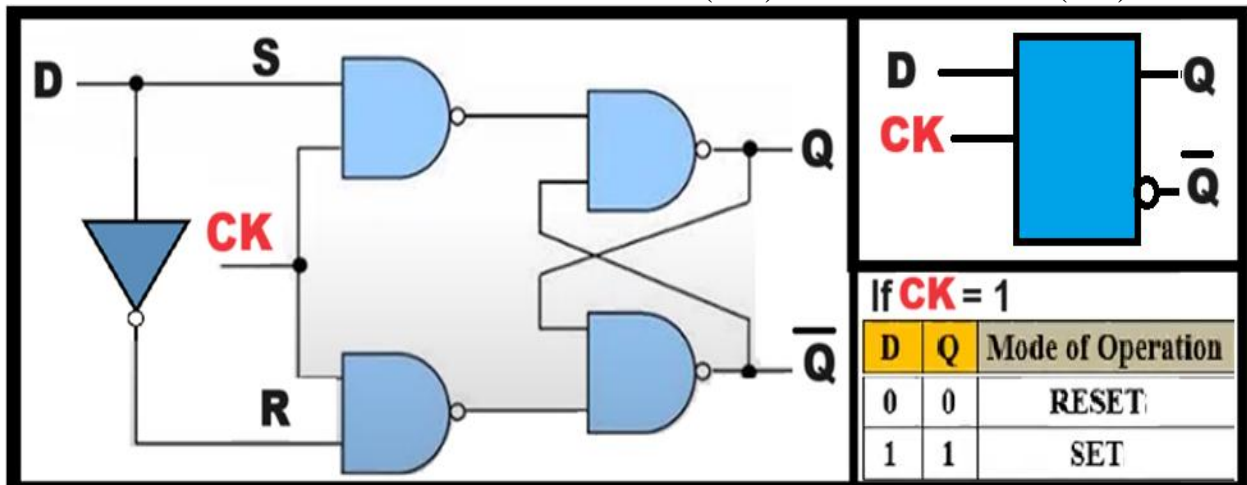
Sol:

S	R	Q	Mode of Operation
0	0	Q ₀	No Change
0	1	0	Reset
1	0	1	set
1	1	?	Invalid condition



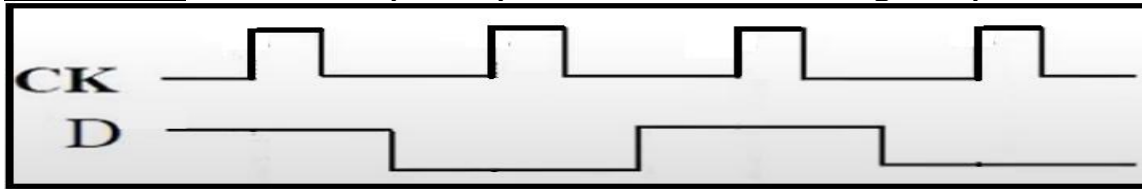
D- Flip Flop (D-FF)

It is a Low Active D-latch with a Clock (CK) instead of Enable (EN).



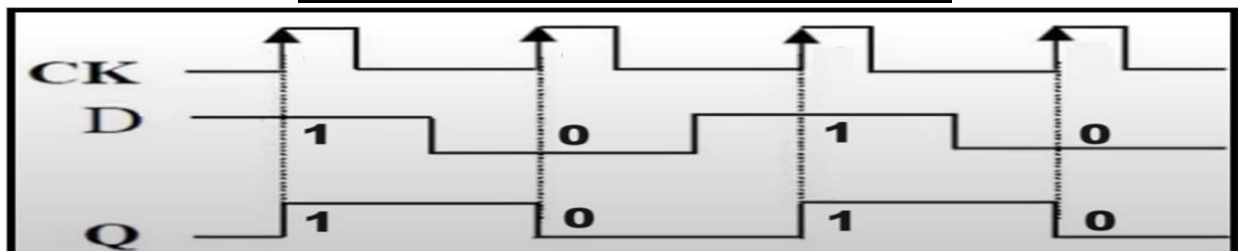
Timing Diagram

Example 1: Draw the output shape of D-FF in case the edge is up.



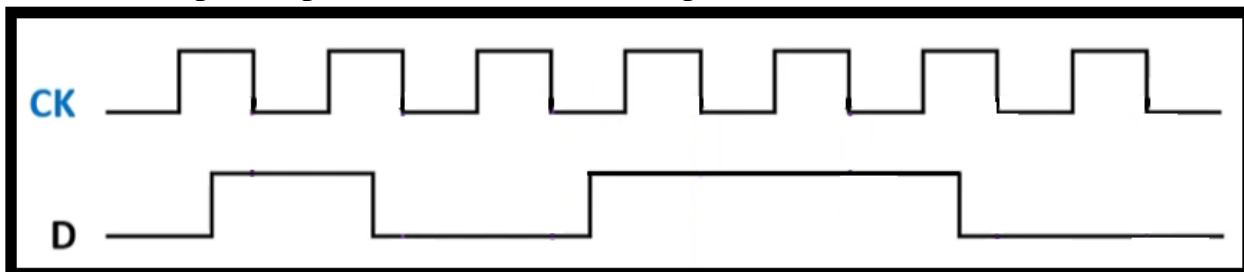
Sol:

D	Q	Mode of Operation
0	0	RESET
1	1	SET



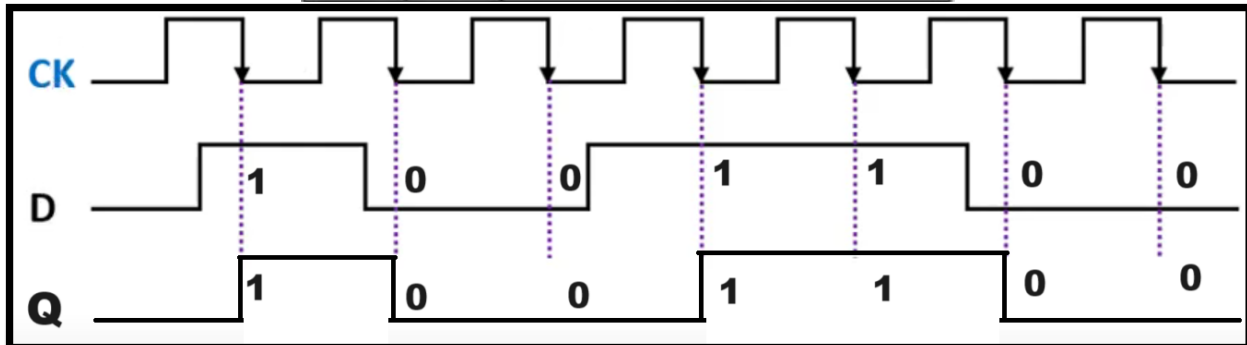
Example 2:

Draw the output shape of D-FF in case the edge is down.



Sol:

D	Q	Mode of Operation
0	0	RESET
1	1	SET



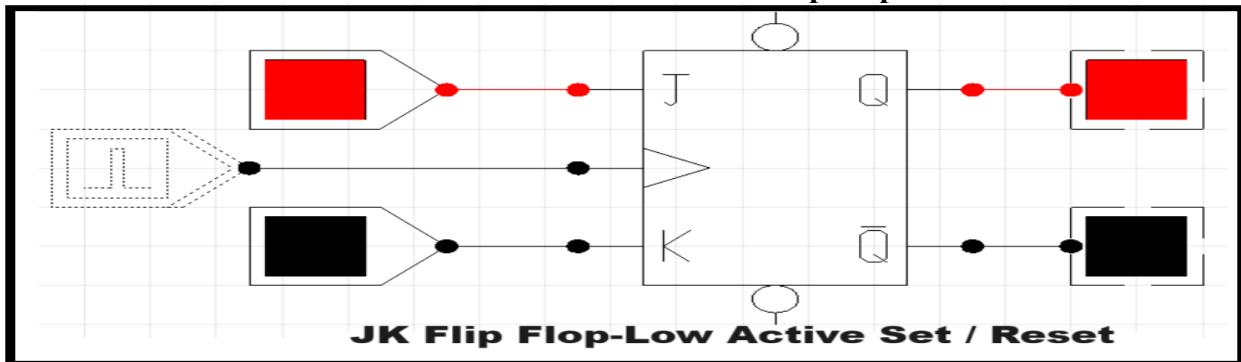
JK-Flip Flop (JK-FF)

It is a Low Active JK-latch with a Clock (CK).

If CK = 1

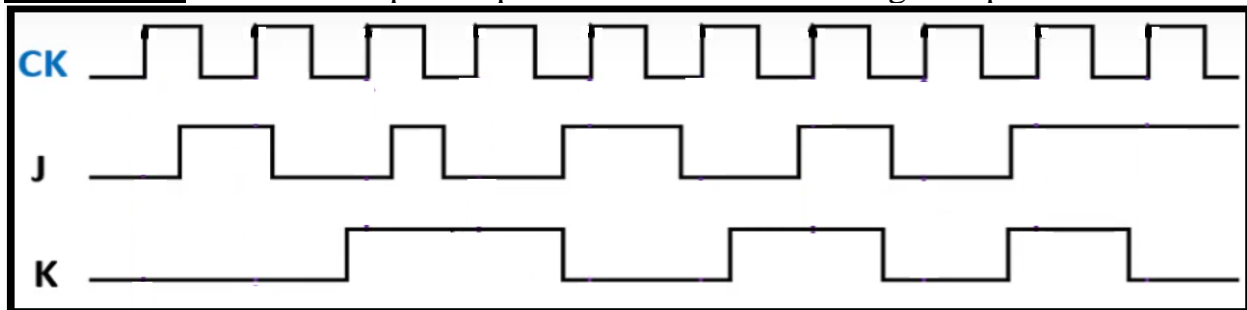
Inputs		Outputs	
J	K	Q	Comments
0	0	Q	No change
0	1	0	Reset
1	0	1	Set
1	1	\overline{Q}	Output change

State Table of Low Active JK-Flip Flop



Timing Diagram

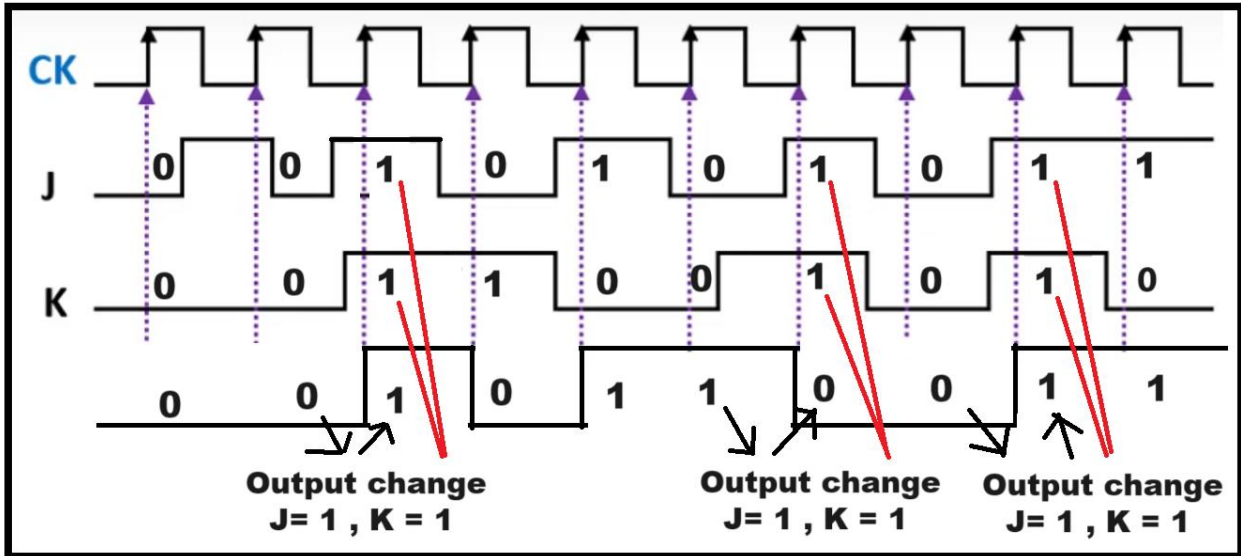
Example 1: Draw the output shape of JK-FF in case the edge is up.



Sol:

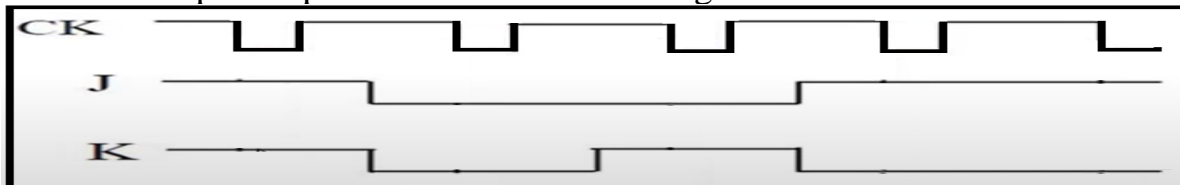
If CK = 1

Inputs		Outputs	
J	K	Q	Comments
0	0	Q	No change
0	1	0	Reset
1	0	1	Set
1	1	\overline{Q}	Output change



Example 2:

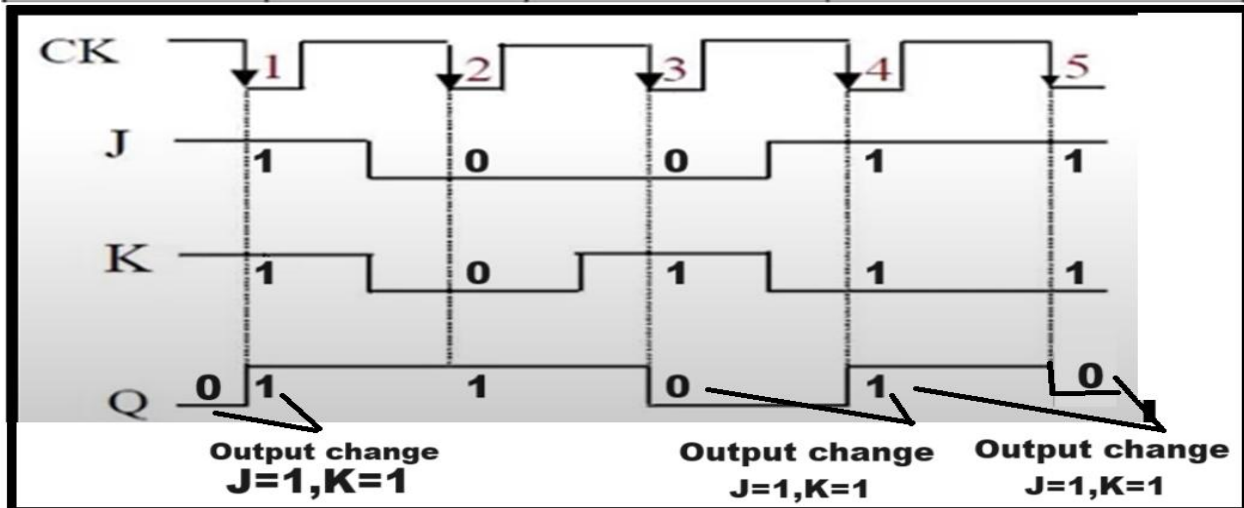
Draw the output shape of JK-FF in case the edge is down.



Sol:

If CK = 1

Inputs		Outputs	Comments
J	K	Q	
0	0	Q	No change
0	1	0	Reset
1	0	1	Set
1	1	\overline{Q}	Output change



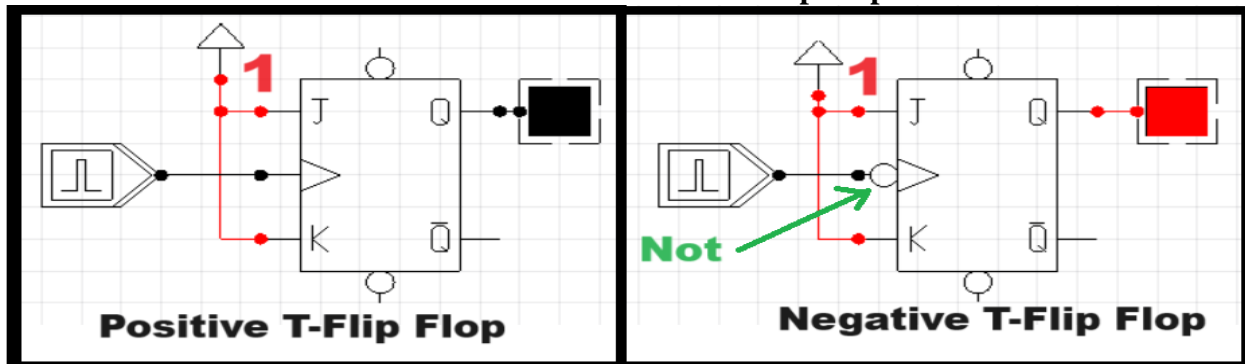
T-Flip Flop (TFF)

It is a Low Active JK-Flip Flop that has only one input (T) with Clock (CK). This Flip Flop is called a toggle Flip Flop because of its ability to complement the output of its state.

if CK = 1

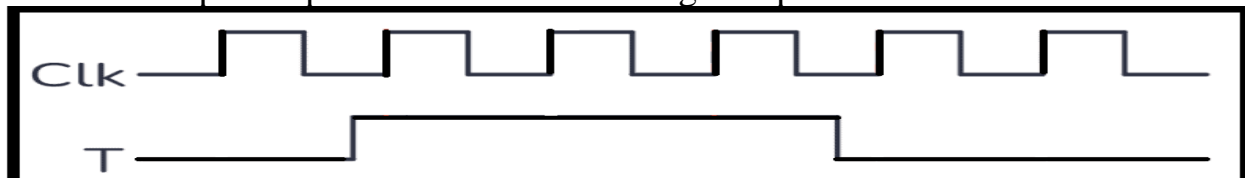
T	Q	Mode of Operation
0	Q	No Change
1	\overline{Q}	Output change

State Table of Low Active T-Flip Flop



Example1:

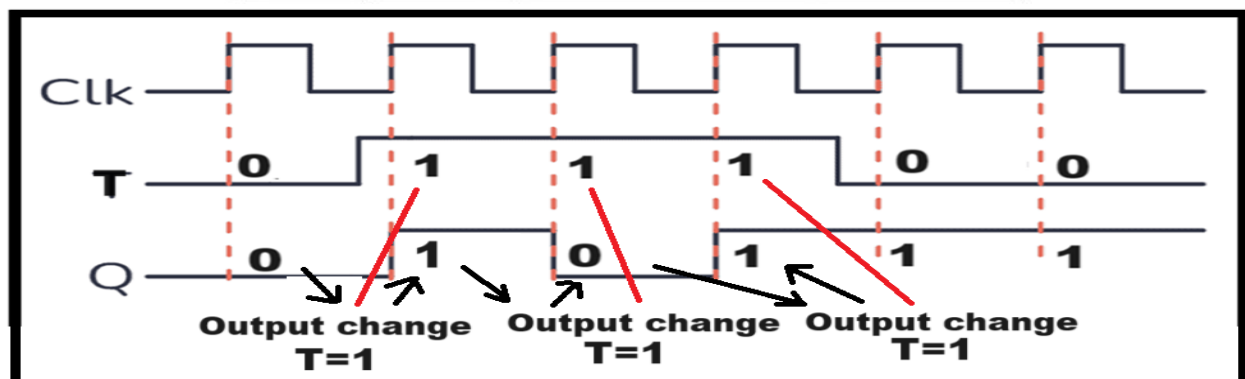
Draw the output shape of T-FF in case the edge is up.



Sol:

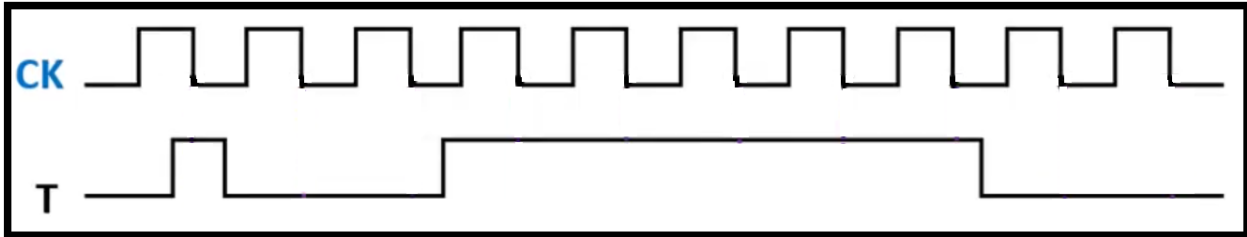
if CK = 1

T	Q	Mode of Operation
0	Q	No Change
1	\overline{Q}	Output change



Example 2:

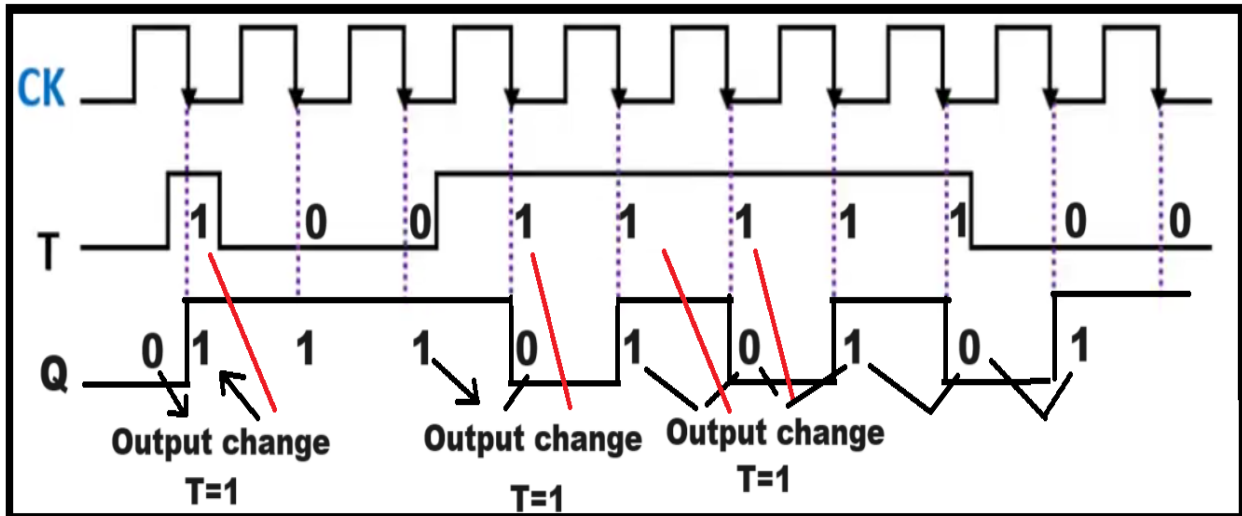
Draw the output shape of T-FF in case the edge is Down.



Sol:

if CK = 1

T	Q	Mode of Operation
0	Q	No Change
1	\overline{Q}	Change



Shift Register:

It is a small memory built from a set of Flip Flops connected together and has two main functions:

- 1- data storage.
- 2- data movement.

Shift Register Operations:

There are 4 main operations that can be performed on SISO registries:

- 1- Shift Right Operation
- 2- Shift Left Operation
- 3- Rotate Right Operation (ROR)
- 4- Rotate Left Operation (ROL)

Type of Shift Registers:

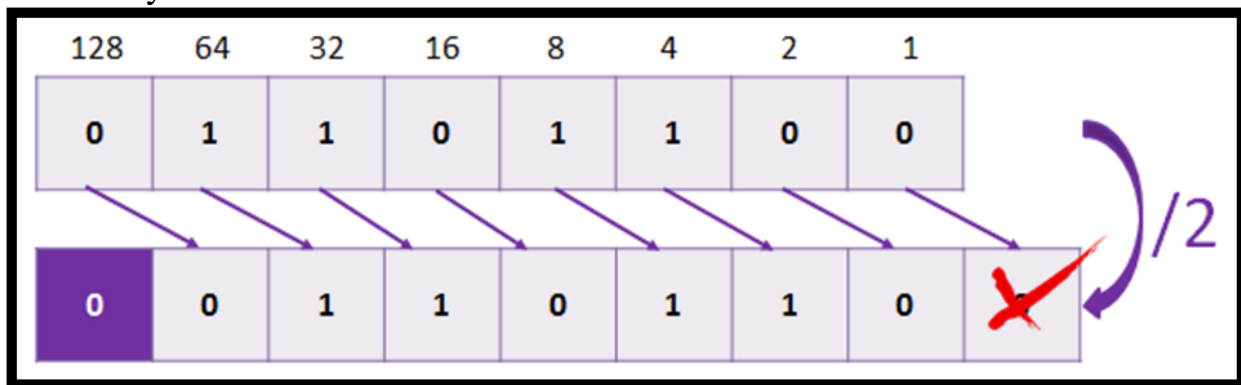
- 1- Serial In / Parallel Out Shift Register (SIPO).
- 2- Serial In / Serial Out Shift Register (SISO).
- 3- Parallel In / Serial Out Shift Register (PISO).
- 4- Parallel in / Parallel Out Shift Register (PIPO).

Serial In / Parallel Out Shift Register (SIPO):

In this type, data is entered in serial mode and retrieved in parallel mode.

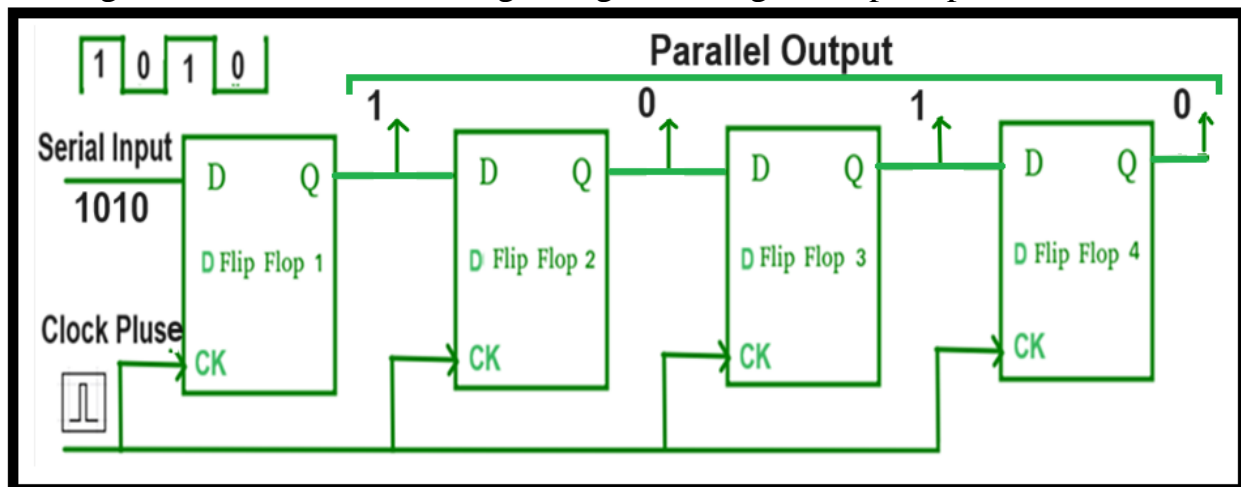
Shift Right Operation

It consists of shifting all the binary digits to the right by 1 digit and adding (0) at the beginning digit (to the left). A binary shift right is used to divide a binary number by two.



The logic circuit of Shift Right Register

To build a Shift Right Register, a set of JK/D Flip Flops can be connected in series mode. The digits of number are entered from right to left until it is complete. The logic circuit of 4-bit Shift Right Register using D- Flip Flop.

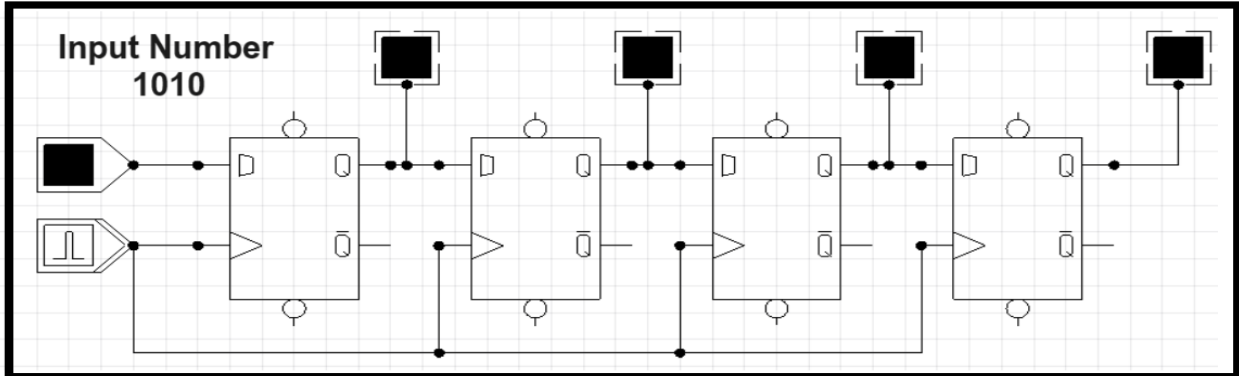


Working Principle:

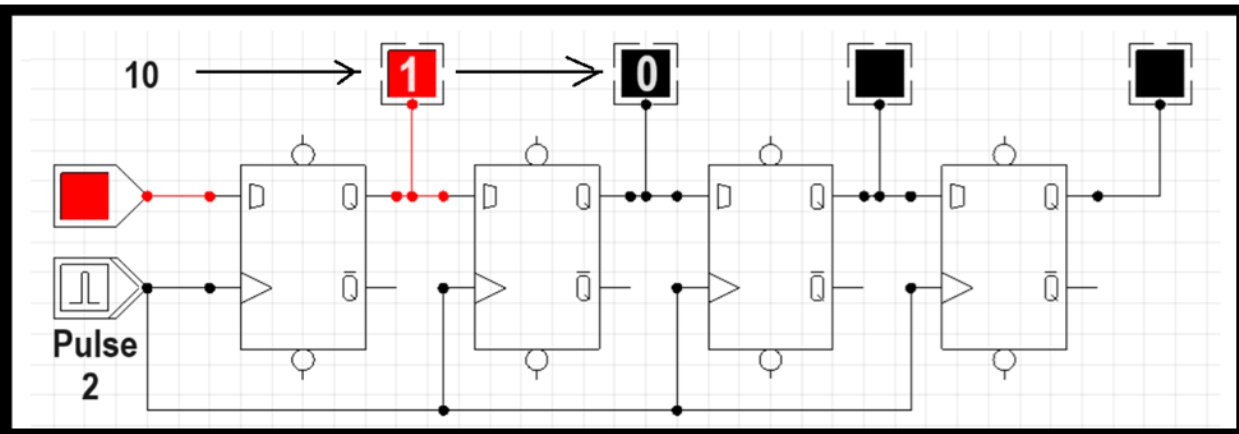
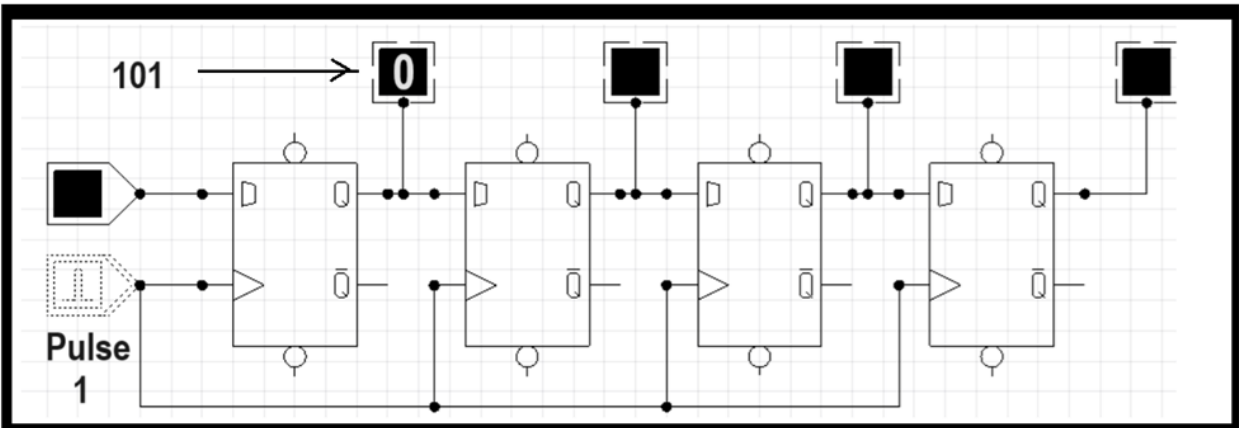
The working principle of SIPO shift registers involves enter the digits of the number bit by bit from right to left with one clock pulse per bit until the number is complete.

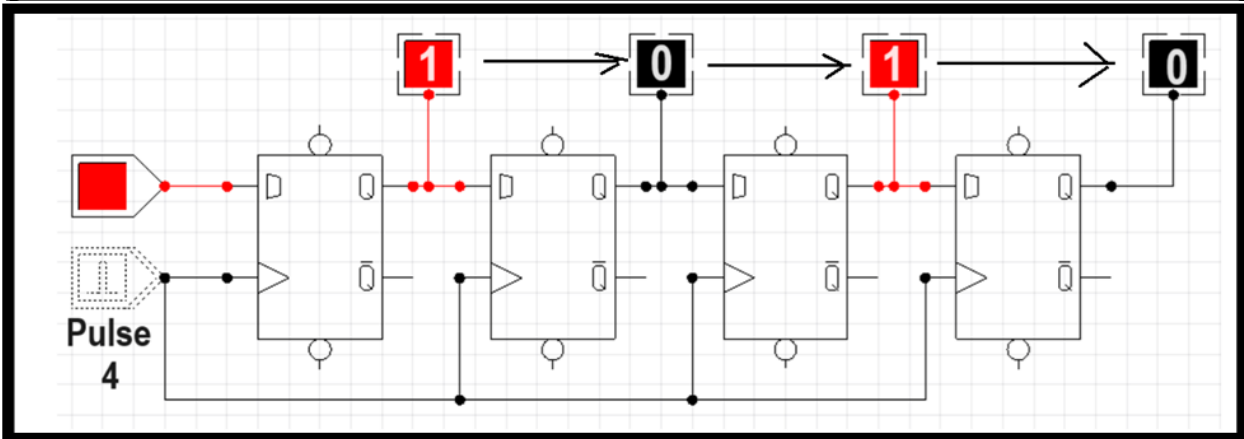
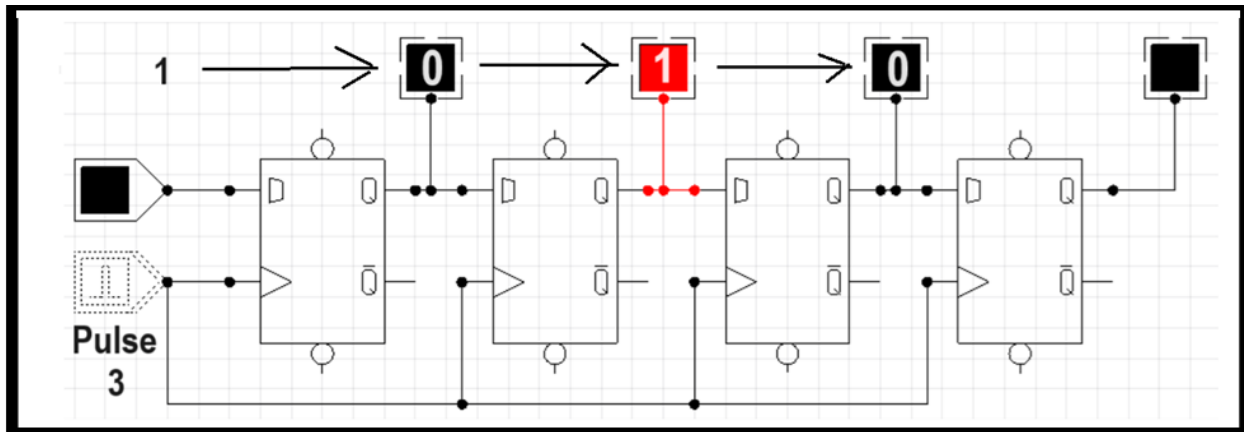
Example: Design SIPO Shift register to store 4-bit numbers using Shift Right Operation.

Sol:



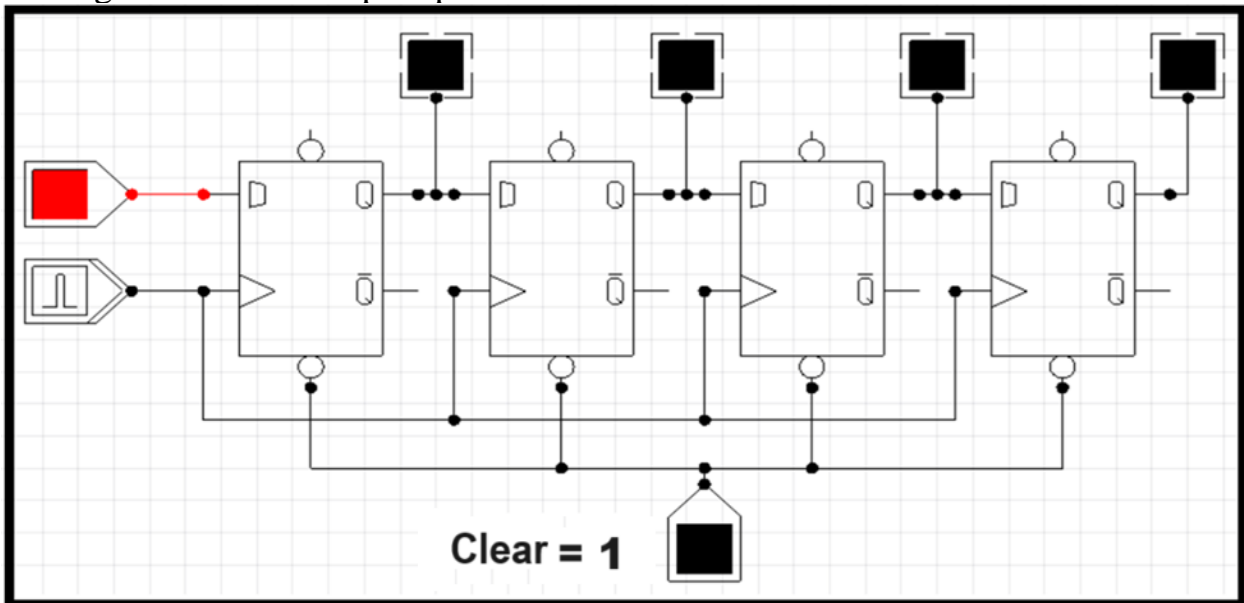
Ex: To enter number (1010)





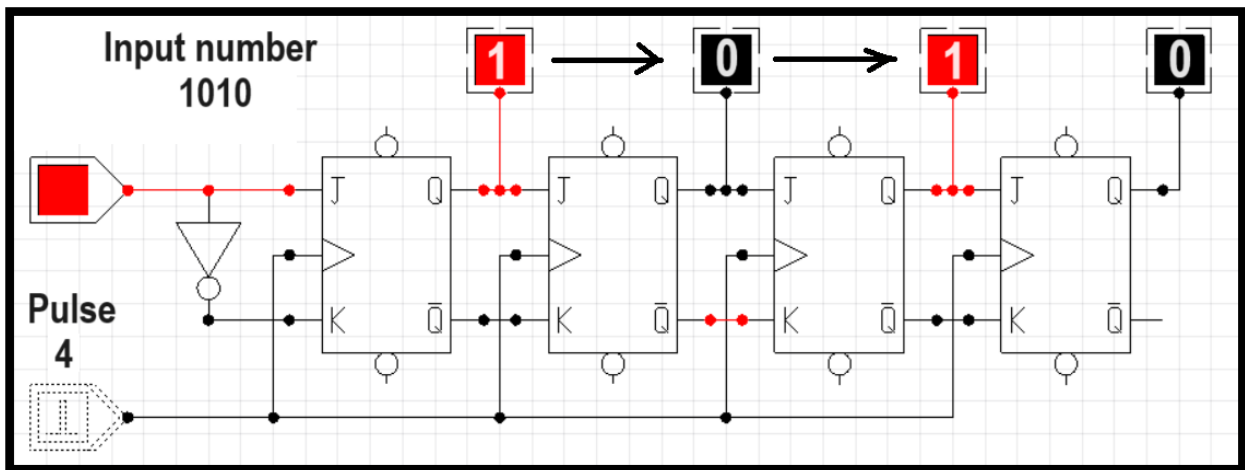
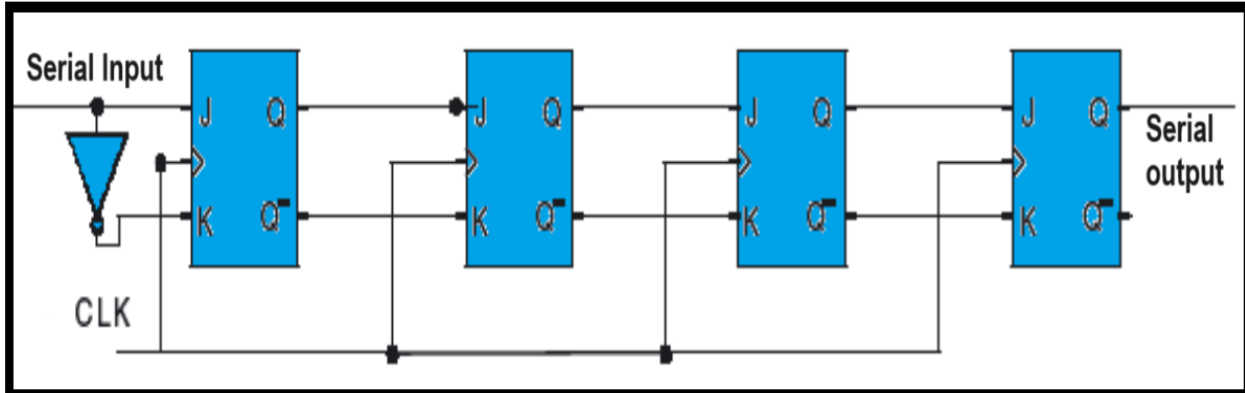
Initial Setup (Clear)

All flip-flops are cleared initially (Clear) to insert new data sequentially, starting from the first flip-flop.



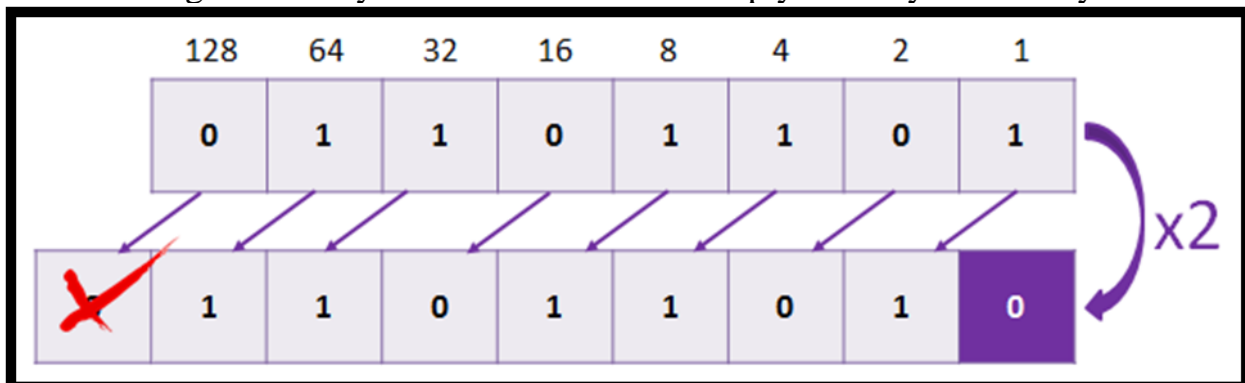
Shift Right Register using JK Flip Flop

The Logic circuit is:



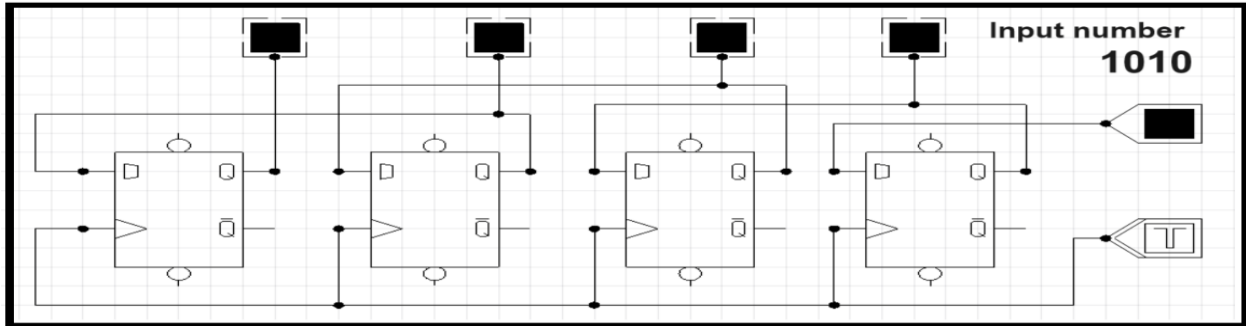
Shift Left Operation

It consists of shifting all the binary digits to the left by 1 digit and adding (0) at the end digit. A binary shift left is used to multiply a binary number by two.

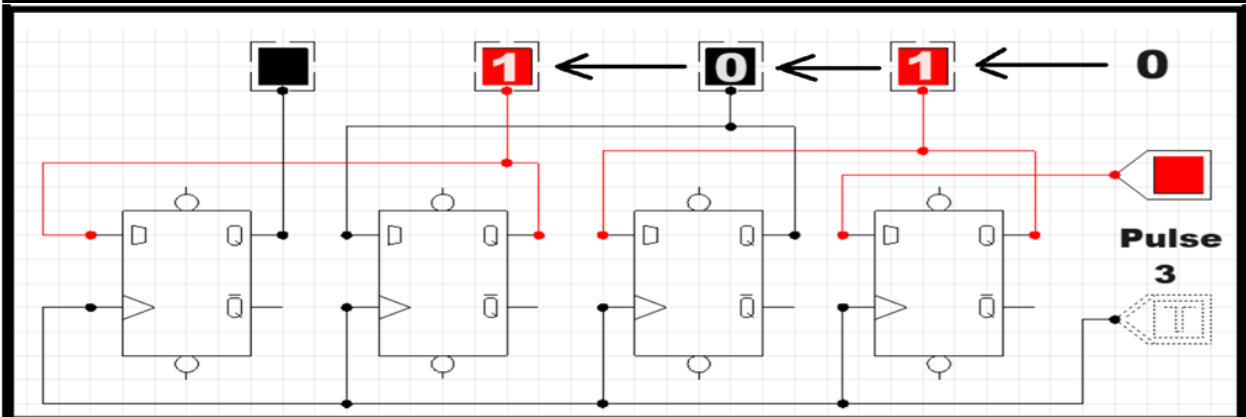
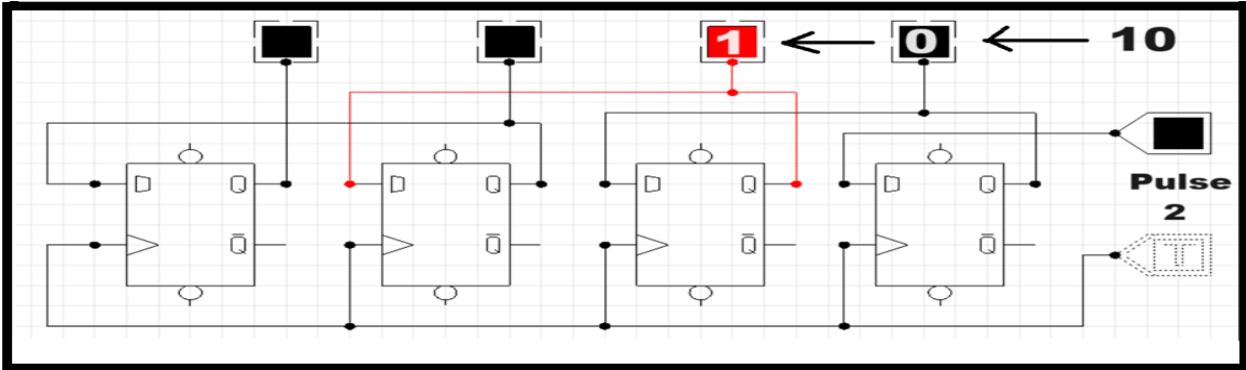
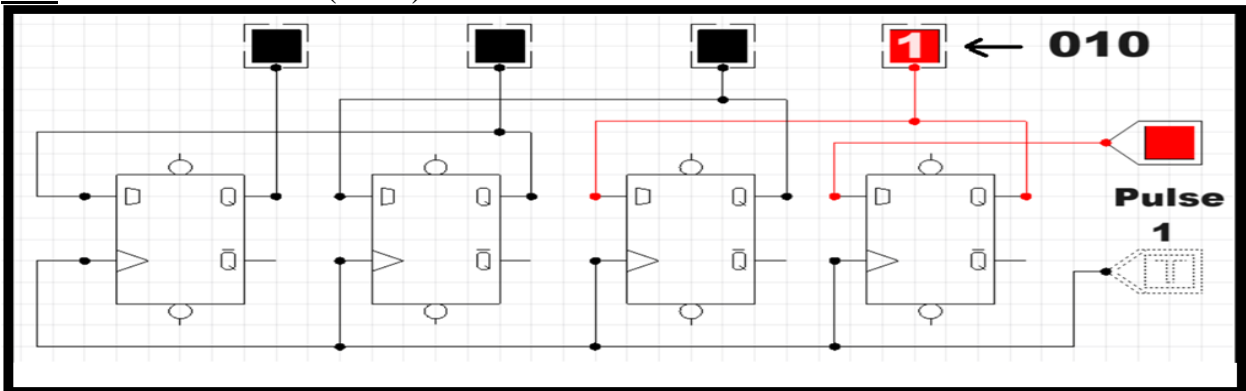


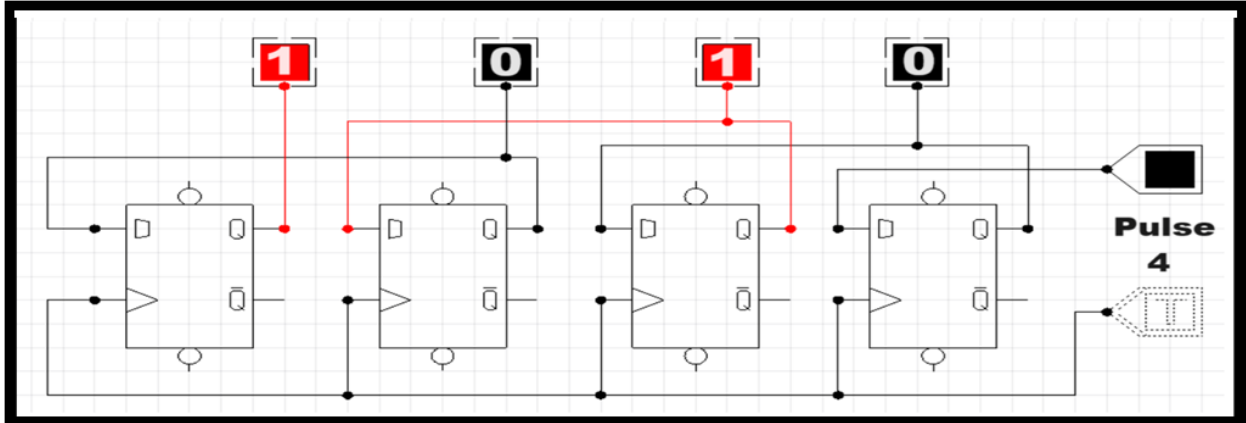
Example: Design SIPO Shift register to store 4-bit numbers using Shift Left Operation.

Sol:



Ex: To enter number (1010)





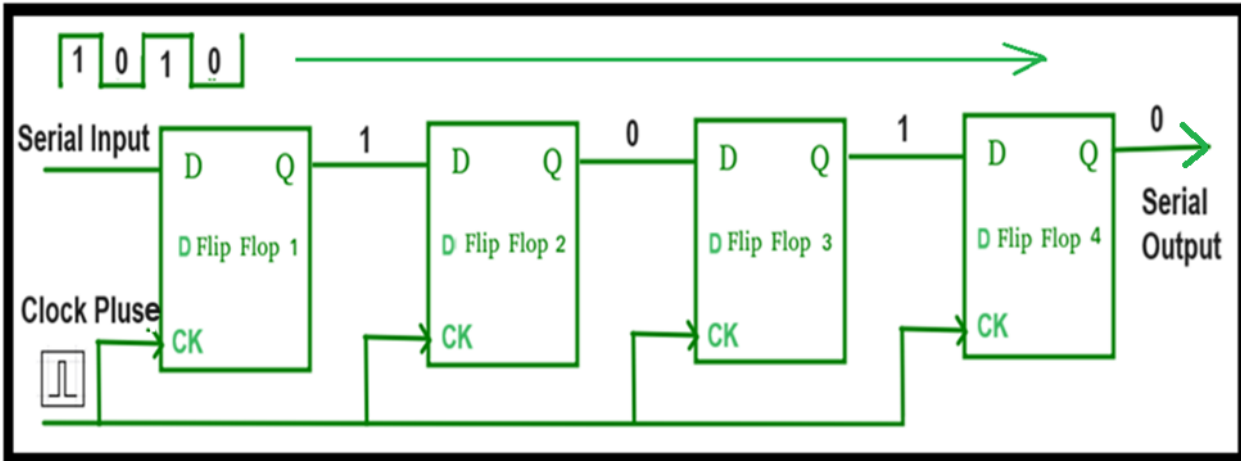
Serial In / Serial Out Shift Register (SISO).

In this type, data is entered and retrieved in serial mode. The entered number is output in full after 4 zeros.

Example:

Design SISO Shift register to store 4-bit numbers using Shift Right Operation.

Sol:

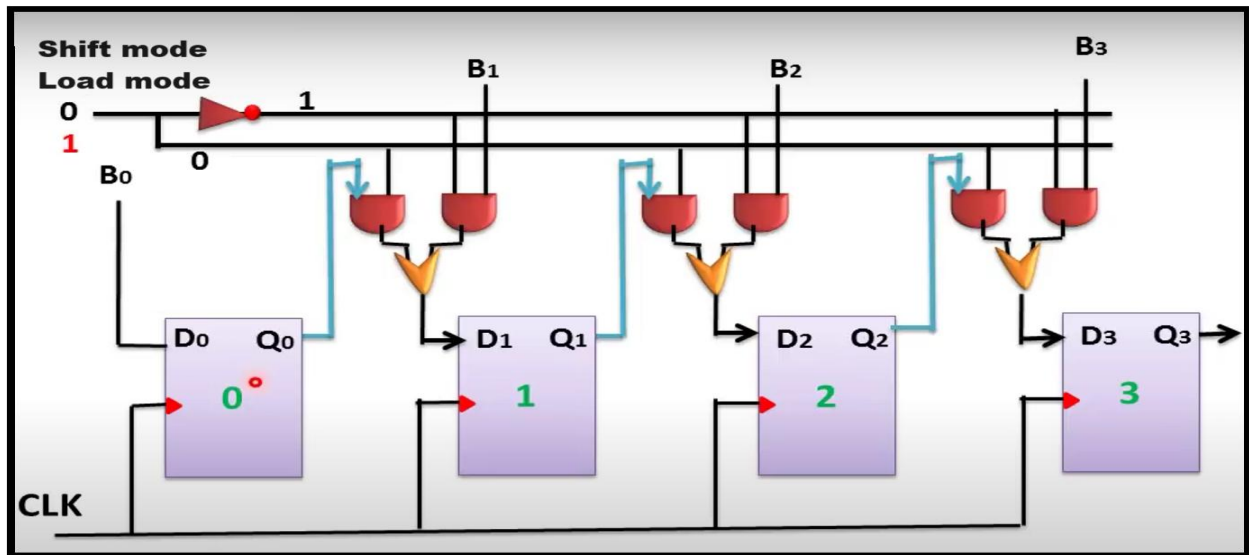


Parallel In / Serial Out Shift Register (PISO).

In this type, data is entered in parallel mode and retrieved in serial mode

Example: Design PISO Shift register to store 4-bit numbers using Shift Right Operation.

Sol:



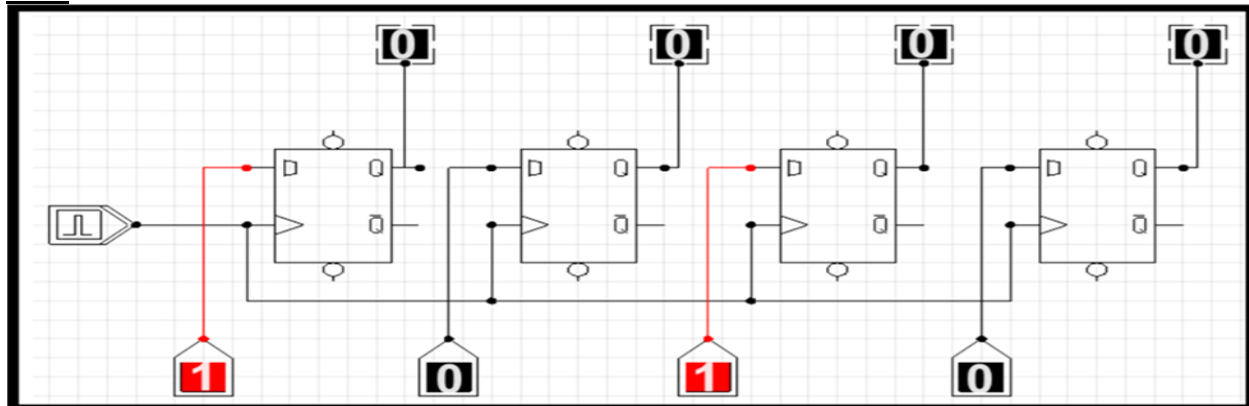
Parallel in / Parallel Out Shift Register (PIPO).

In this type, data is entered and retrieved in Parallel mode.

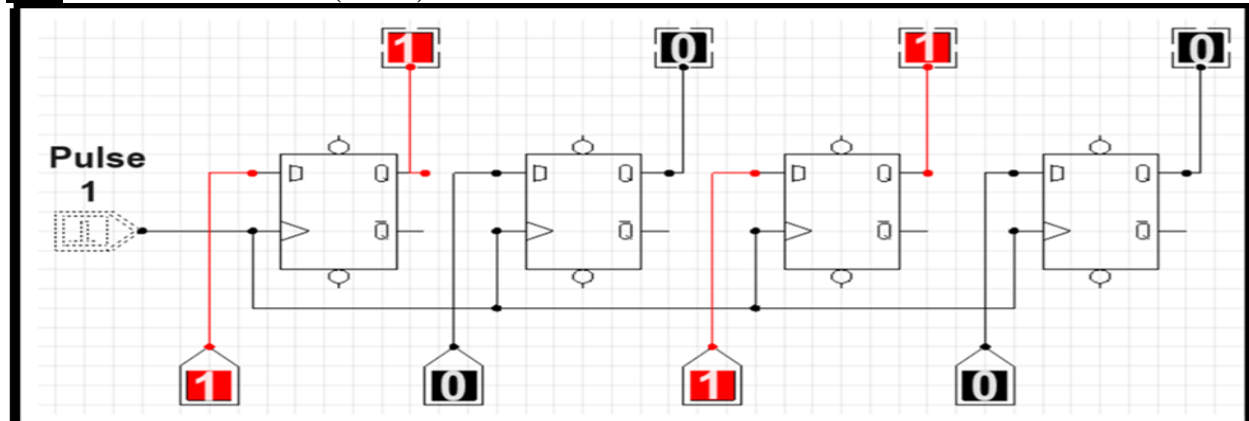
Example:

Design PIPO Shift register to store 4-bit numbers using Shift Right Operation.

Sol:

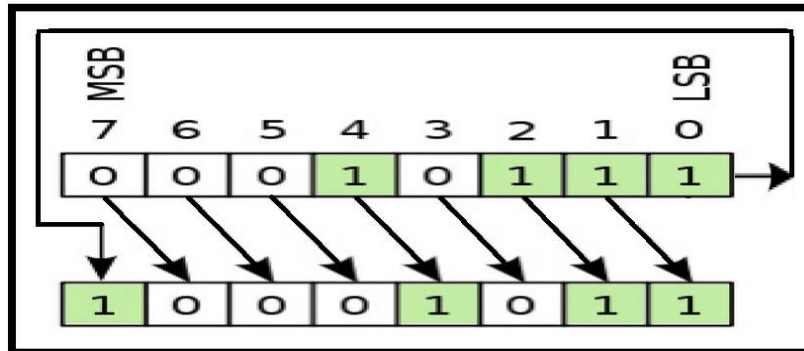


Ex: To enter number (1010)



Rotate Right Operation (ROR)

It is a shift to the Right but the bit that falls off at the Right side is put back on the left side.



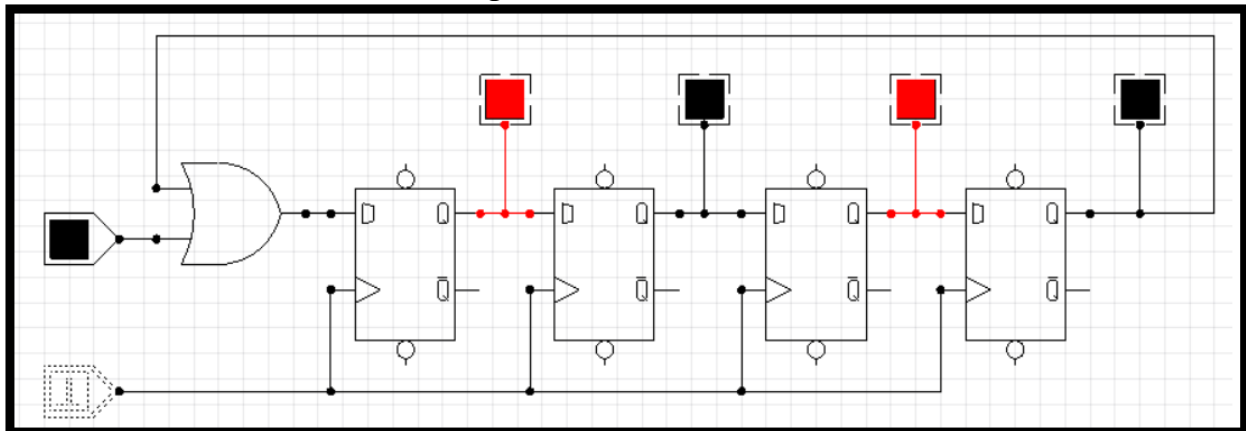
Example:

Design Rotate Right logic circuit to store 4-bit numbers using Shift Right Operation.

Sol:

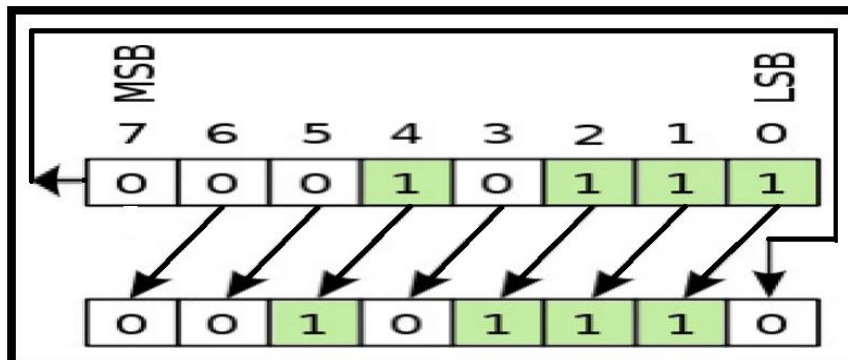
The working principle is:

- 1- Enter the number (Ex\ 1010)
- 2- For a full rotation to the right, enter four zeros (0000).

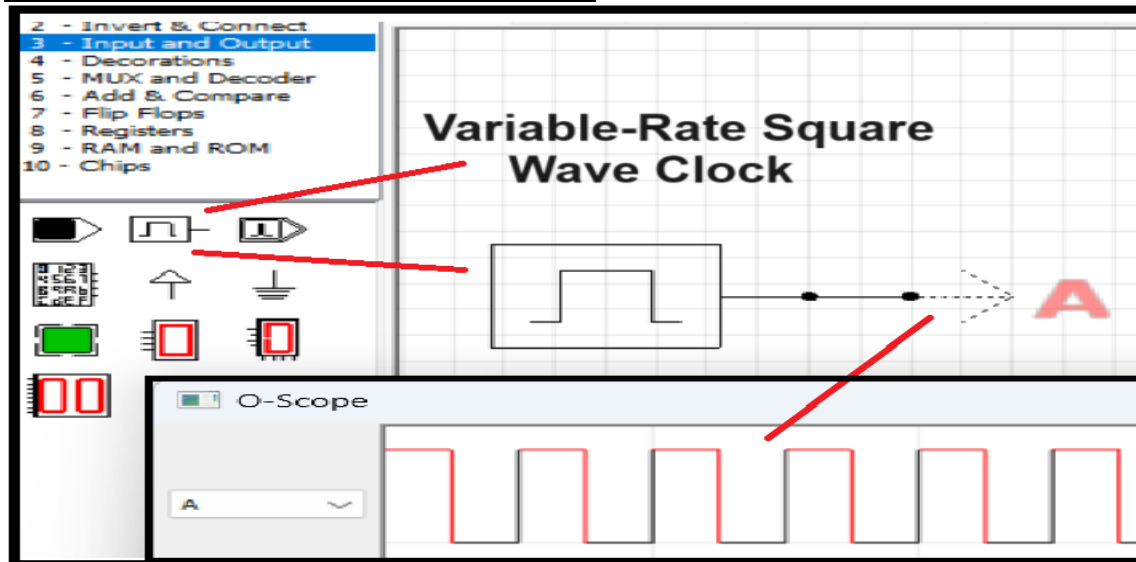


Rotate Left Operation (ROL)

It is a shift to the left but the bit that falls off at the left side is put back on the right side.



Wave of Clock pulses (Wave Clock) :



Binary Counter:

It is a set of flip-flops with clock pulses (CLK) used for counting.

Types of Counters:

The binary counter is consisting two types.

- 1- Synchronous counter.
- 2- Asynchronous counter.

Counting direction

- 1- Count UP (Incrementing)
- 2- Count Down (Decrementing)

Counting method

- 1- Normal counting
- 2- Decade counting
- 3- Random counting

Asynchronous counter.

In this type there is no clock directly connected to all the flip-flops, only the first flip-flop is timed by the external clock while the rest of the flip-flops are timed by the output of the flip-flop before it.

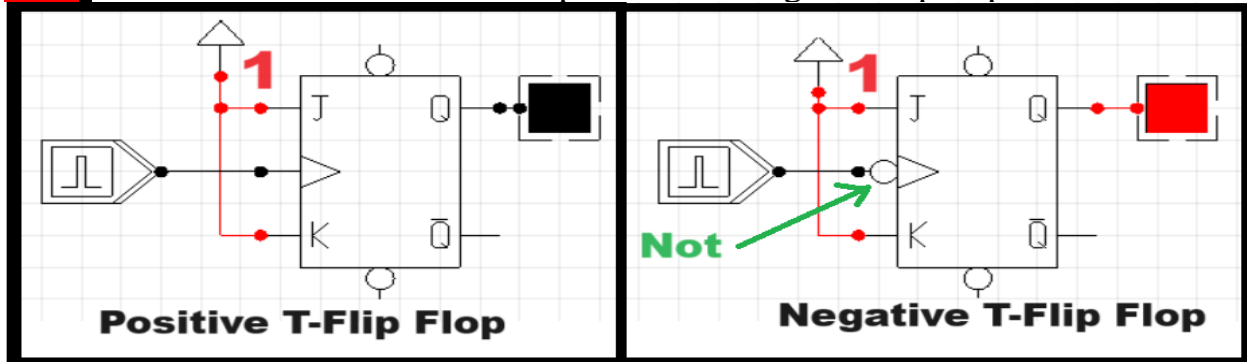
Logic Circuit of Asynchronous counter

To build an Asynchronous counter, a set of **Negative** T-Flip Flops can be used.

- 1- The number of T-FF = the number of bits in the count number.
- 2- State table

Count UP	Q Output
Count Down	\overline{Q} Output

Note: Notice the difference between positive and negative flip flops.



Normal counting

Use full digits storage space.

- Ex:** (0 to 11)₂
- (0 to 111)₂
- (0 to 1111)₂
- (111 to 0)₂

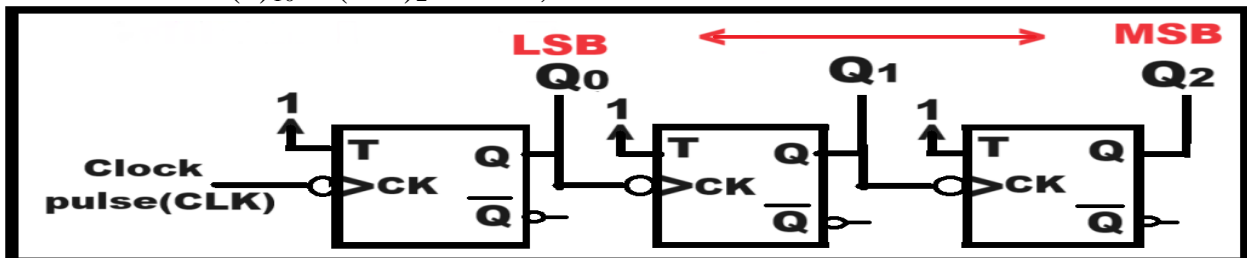
Example 1:

Design (0 to 7)₁₀ Asynchronous counter.

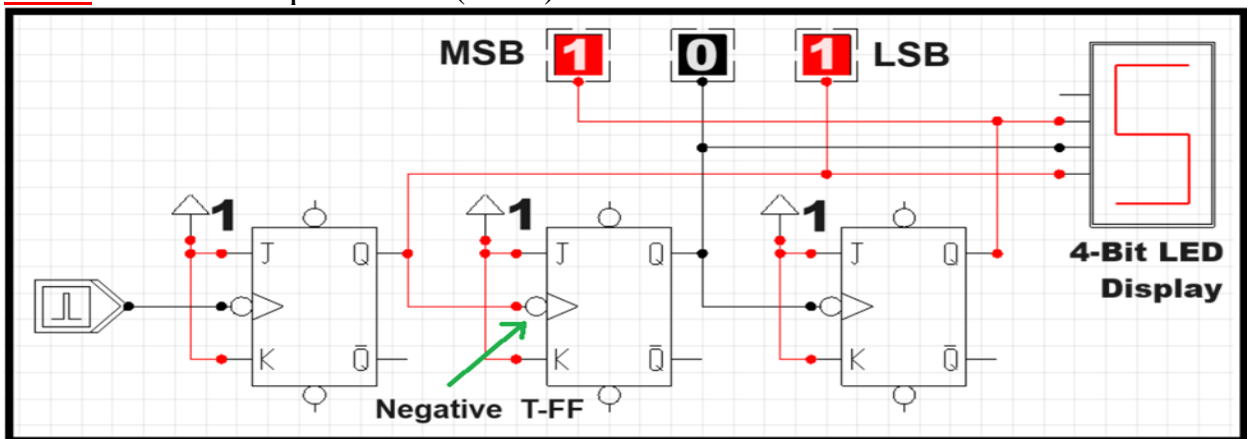
Sol:

The number of Flip Flops = the number of bits in the count number.

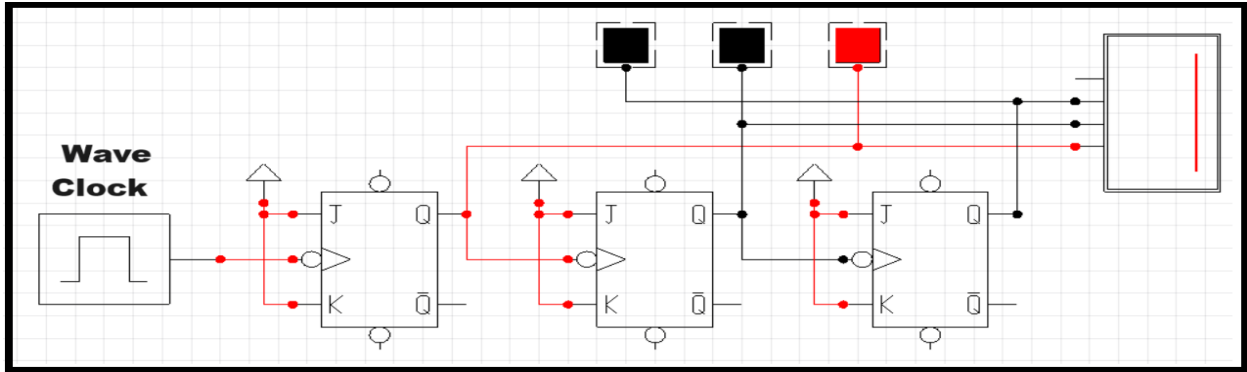
Count number (7)₁₀ = (111)₂ = 3-bit, So the number of T-FF = 3



Note: The most important bit (MSB) should be on the left.



In the previous example, the Wave Clock (Wave of Clock pulses) can be used to make the counter count continuously.



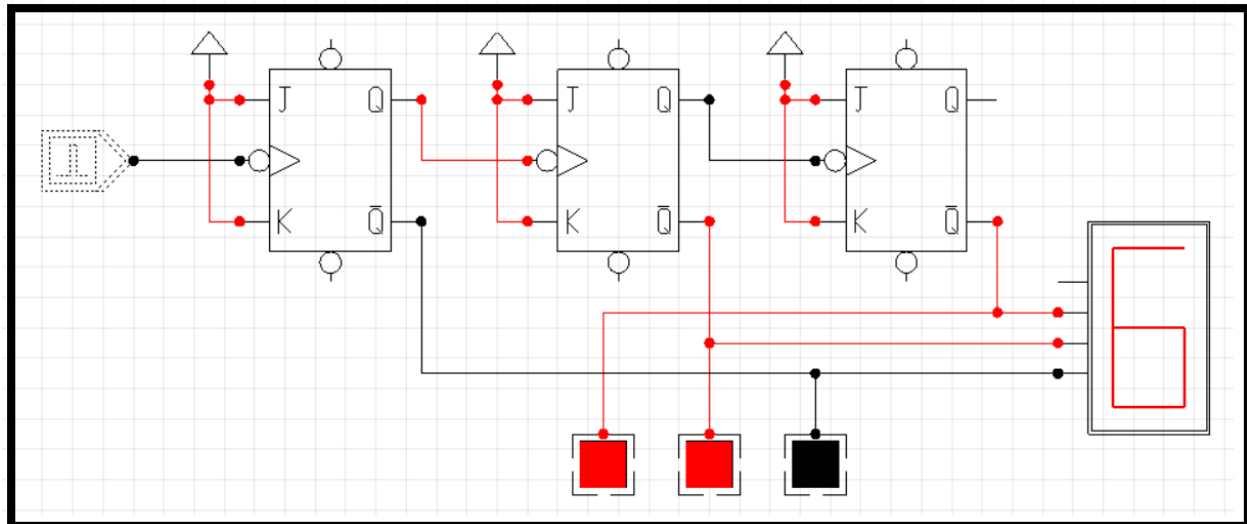
Example 2:

Design $(7 \text{ to } 0)_{10}$ Asynchronous counter

Sol:

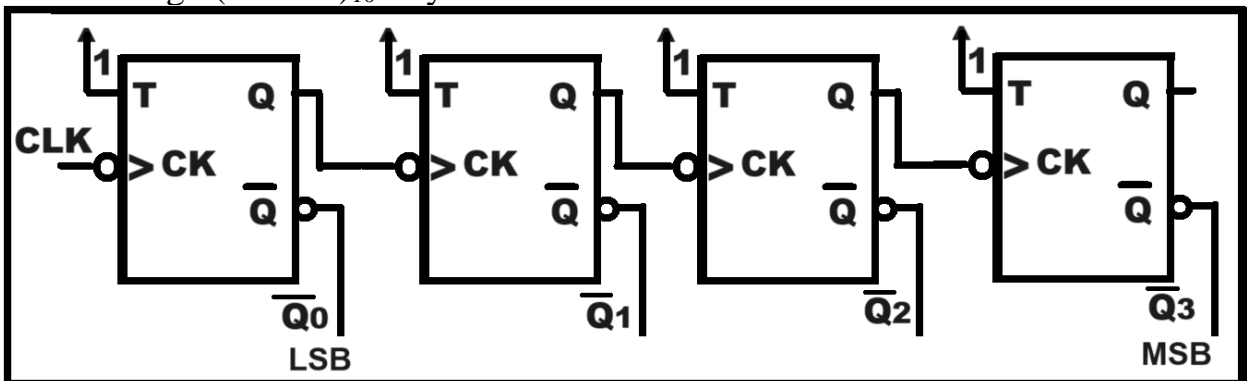
Count UP	Q Output
Count Down	\bar{Q} Output

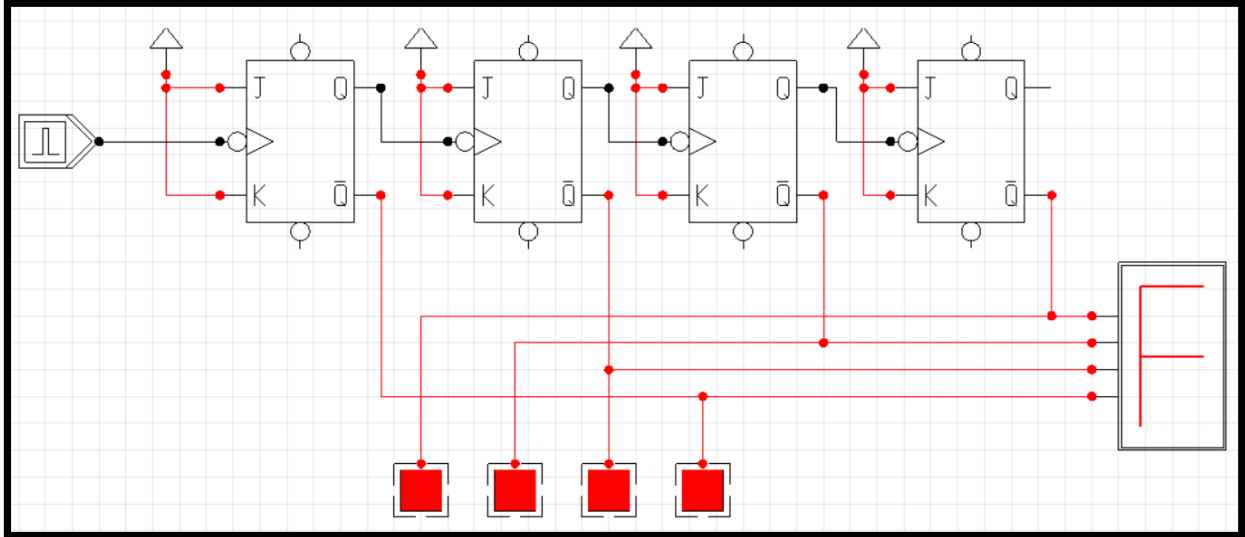
State table



Example 3:

Design $(15 \text{ to } 0)_{10}$ Asynchronous counter.





Decade counting

Counting from-to specific numbers

- Ex:** (0 to 100)₂
 (10 to 100)₂
 (10 to 111)₂
 (110 to 0)₂
 (101 to 11011)₂

Example 1:

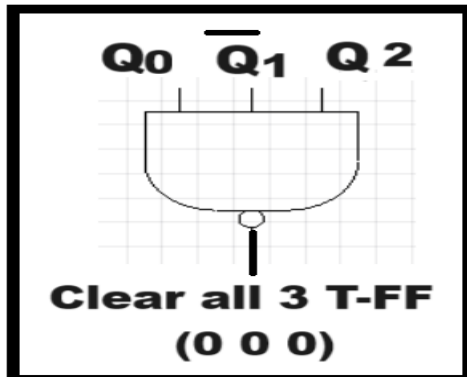
Design (0 to 4)₁₀ Asynchronous counter.

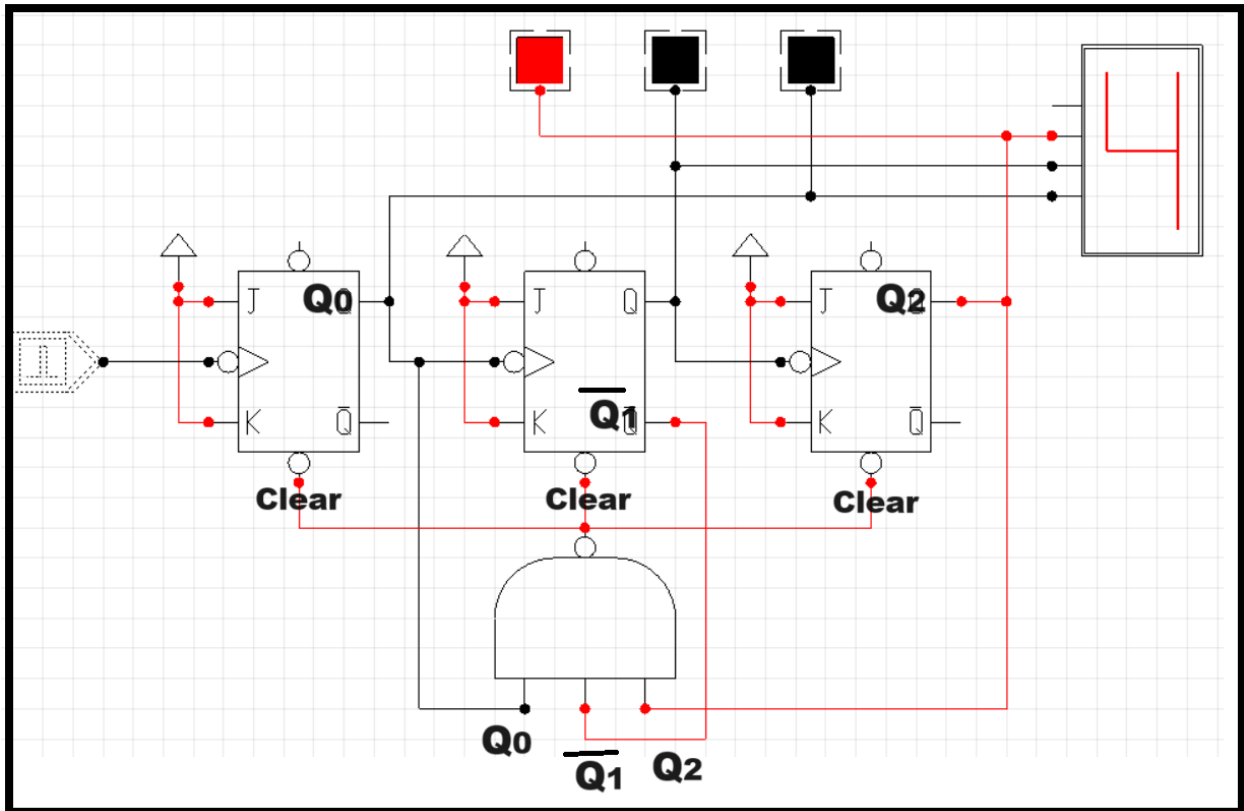
Sol:

- 1- The number after (4)₁₀ is (5)₁₀ = (101)₂

$$\begin{array}{ccc} \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{Q_0} & \overline{\mathbf{Q_1}} & \mathbf{Q_2} \end{array}$$

- 2- Use the NAND gate and connect it to the circuit as shown in the following figure.





Example 2:

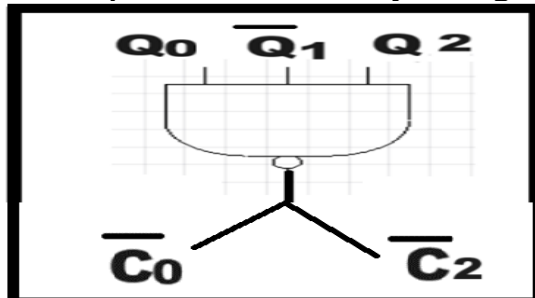
Design $(0 \text{ to } 4)_{10}$ Asynchronous counter.

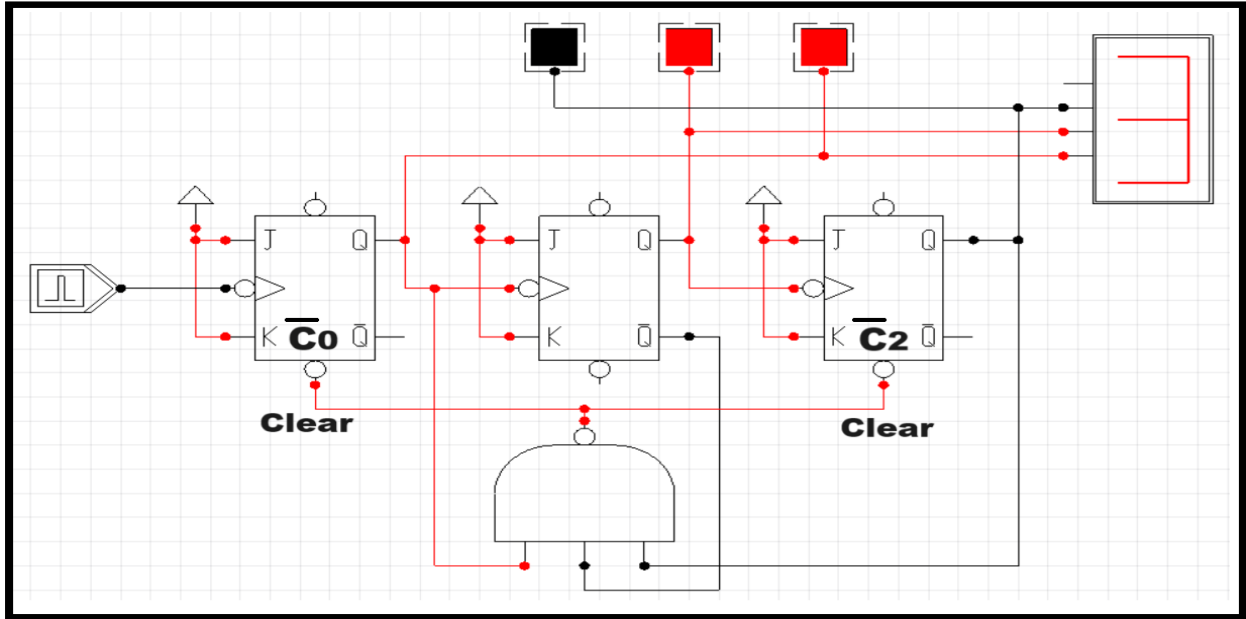
Steps 1 and 2 as in the previous example.

3- The low number is $(2)_{10} = (010)_2$

$$\begin{array}{ccc} 0 & 1 & 0 \\ \hline C_0 & C_1 & C_2 \end{array}$$

4- Connect the NAND Outputs with zeros only and ignores the ones.





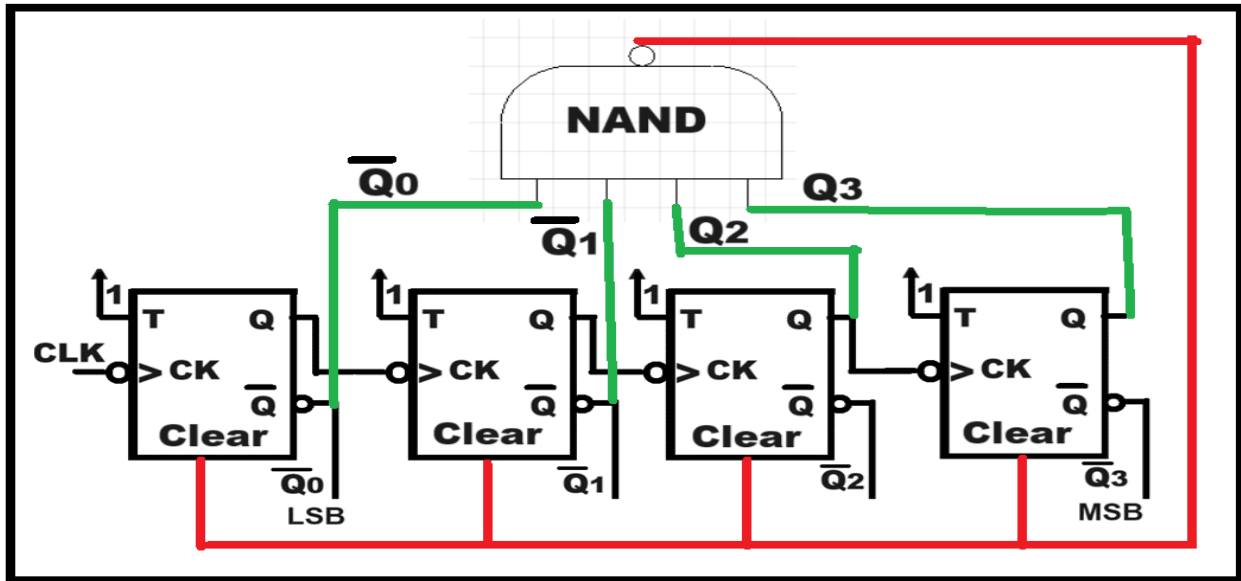
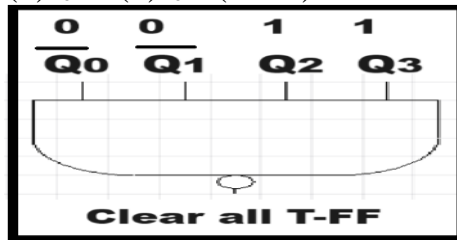
Example 1:

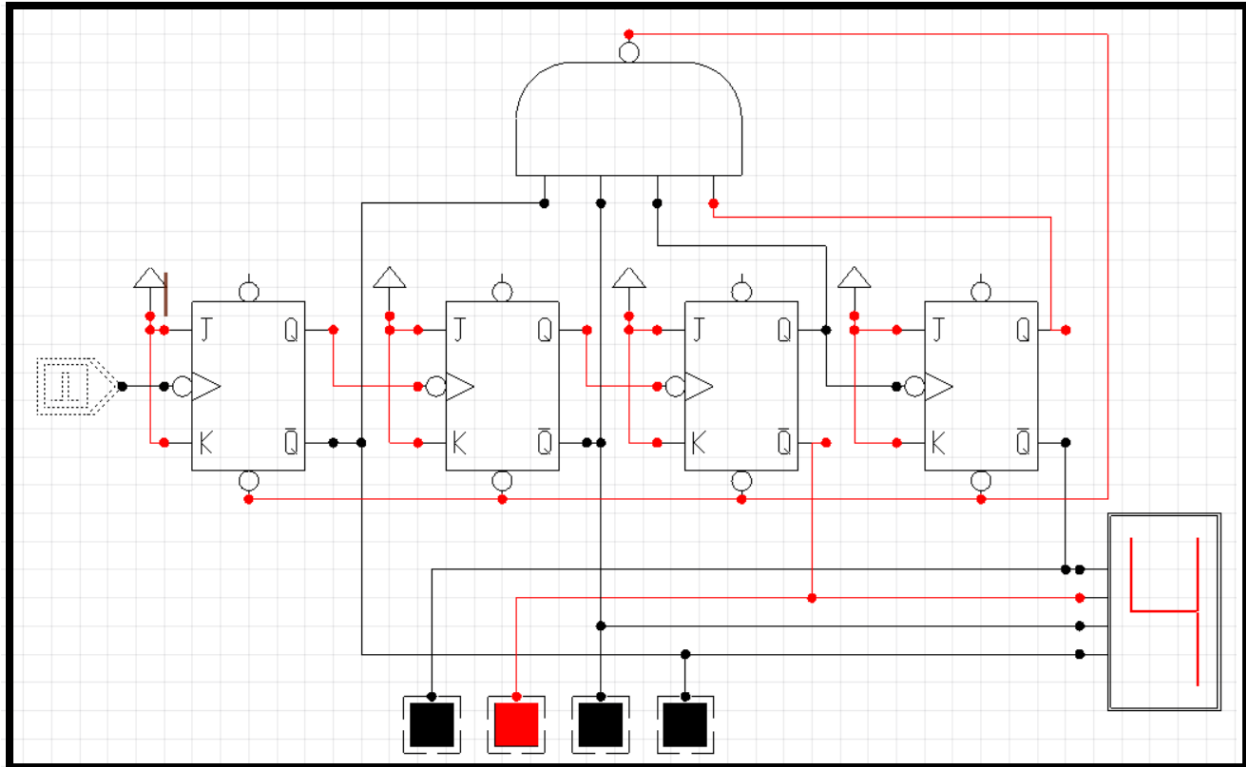
Design $(15 \text{ to } 4)_{10}$ Asynchronous counter.

Note: In countdown, the smallest number is the input to the NAND gate.

Sol:

The number before $(4)_{10}$ is $(3)_{10} = (0011)_2$.





Synchronous counter

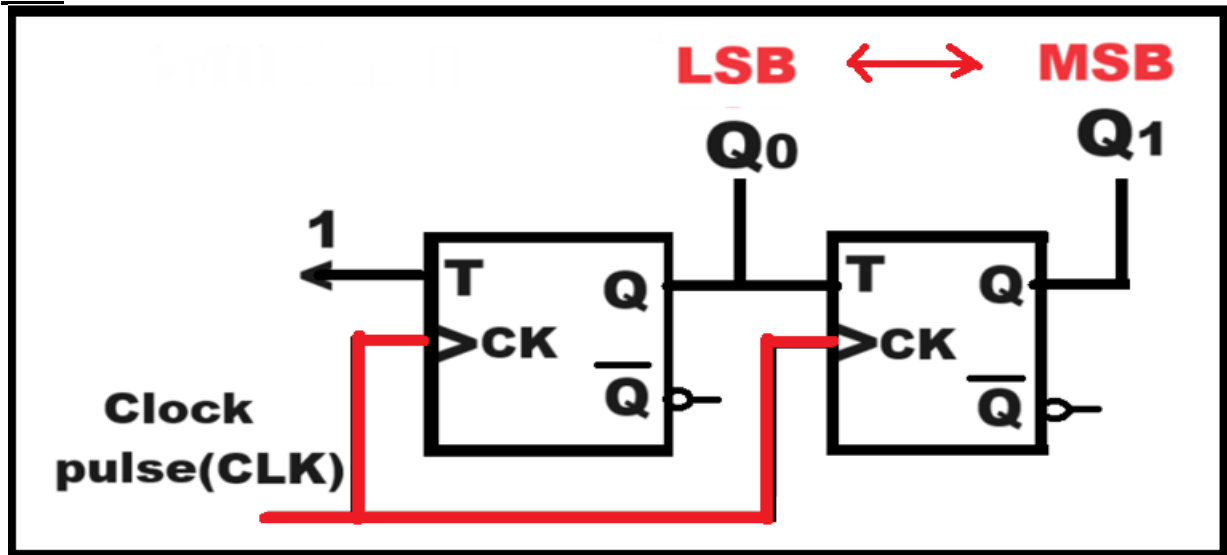
In synchronous counters, the clock is connected to all **Positive** T-Flip-Flops in order to transmit the clock pulse synchronously to all T-FF at the same time.

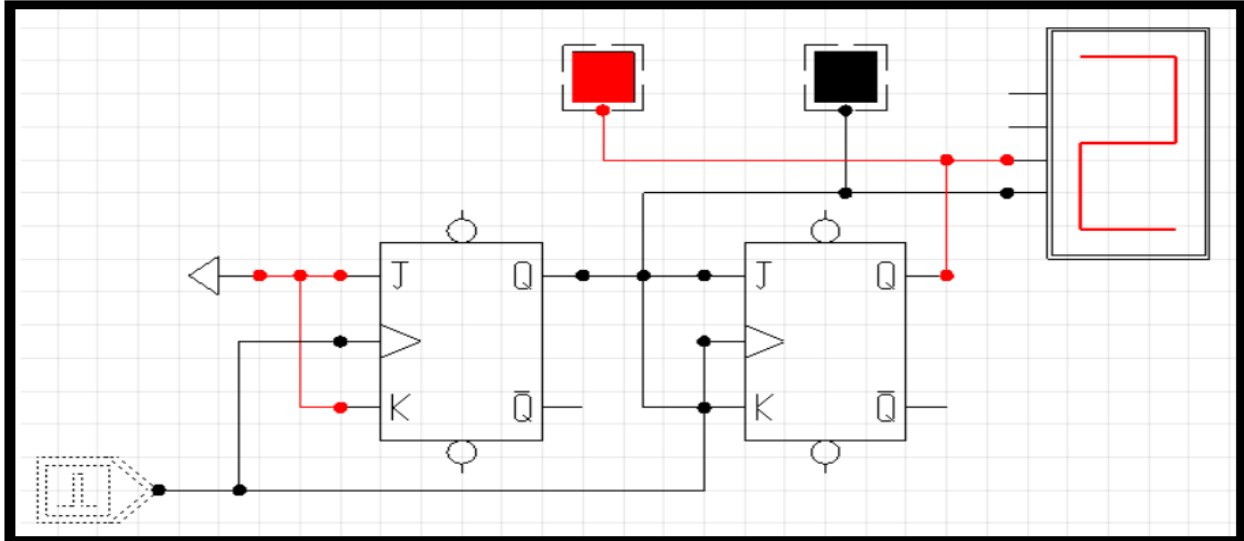
Note: In synchronous counter a **Positive** T- flip flop is used.

Example 1: (2-bit T-FF)

Design $(0 \text{ to } 3)_{10}$ Asynchronous counter.

Sol:

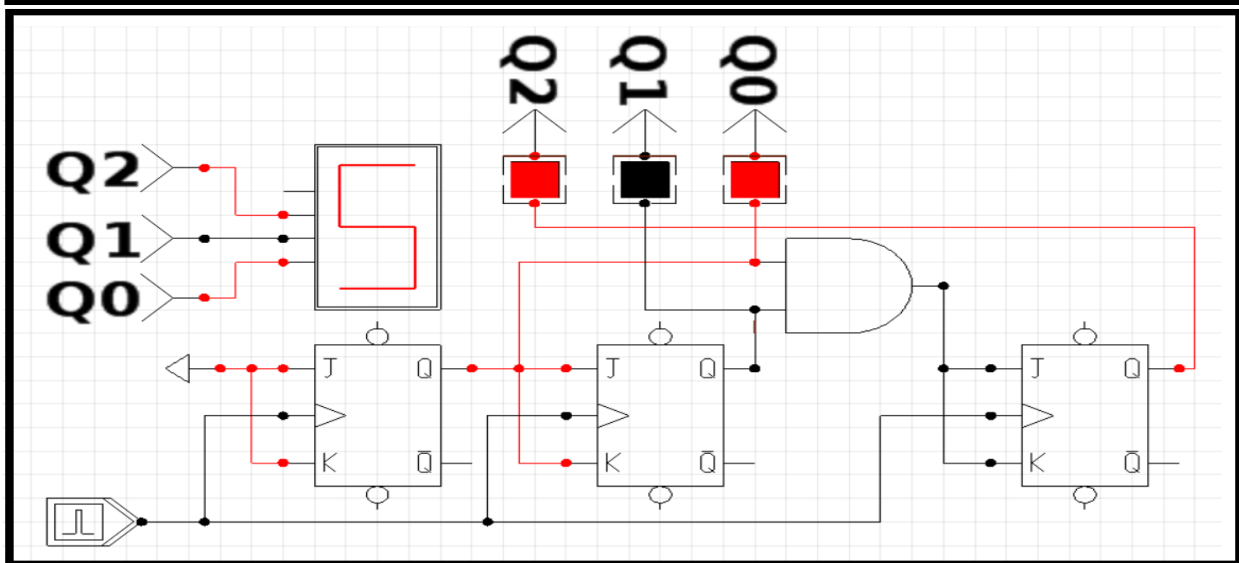
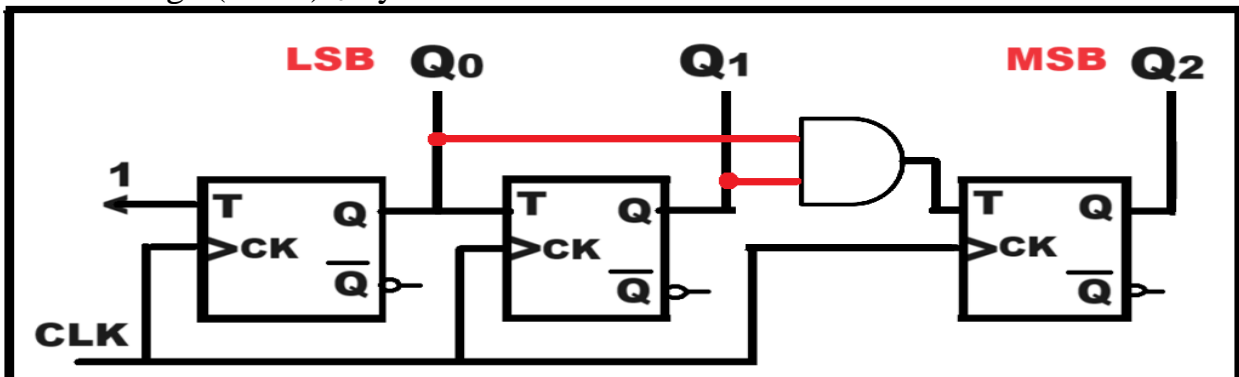




Note: When using more than 2 T-Flip Flops, the design will be changed.

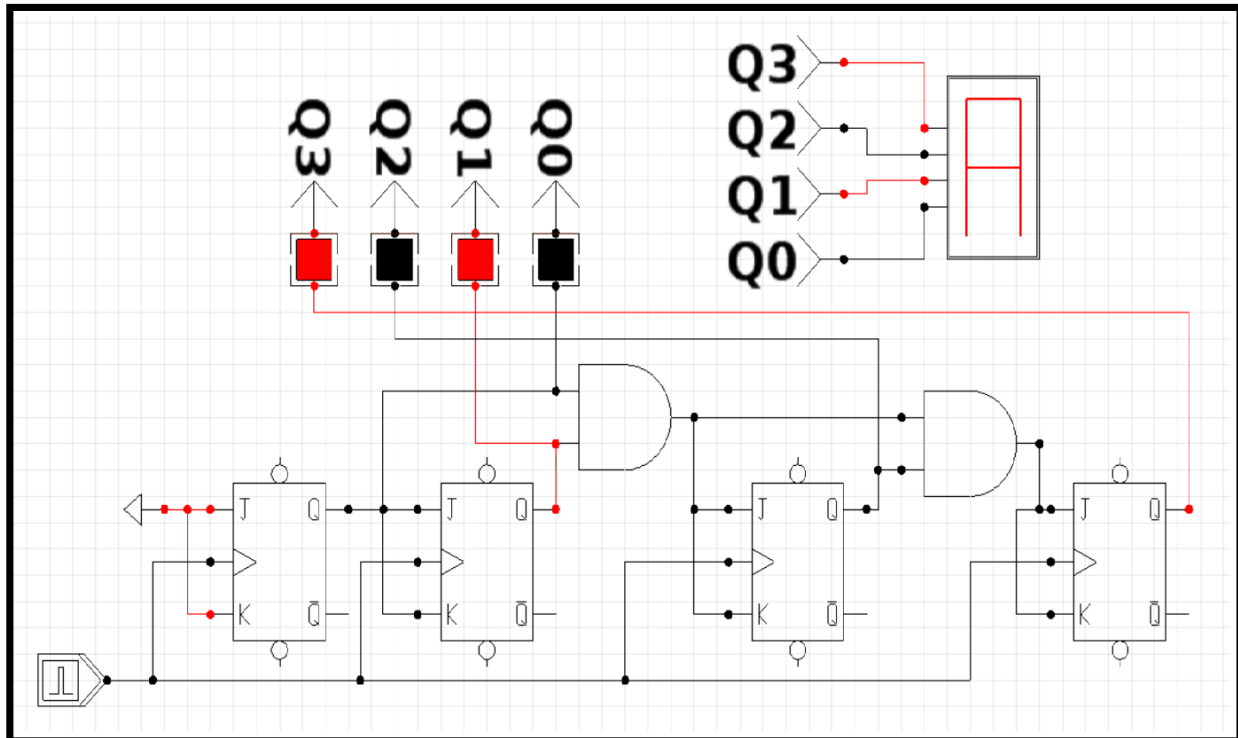
Example 2:

Design $(0 \text{ to } 7)_{10}$ Synchronous counter



Example 3:

Design (0 to 15)₁₀ Synchronous counter.



Random counting

It is used to count any random set of numbers.

Ex: (3,1,2,0)

(7,5,3,6,1)

(1,7,5,3,6,8)

The logic circuit of the Random counter

The logic circuit consists of the following:

- 1- D-Flip-Flops
- 2- Controller
- 3- Number of D-FF = number of bits in the highest number in the set

Example 1: Design $(3,1,2,0)_{10}$ counter.

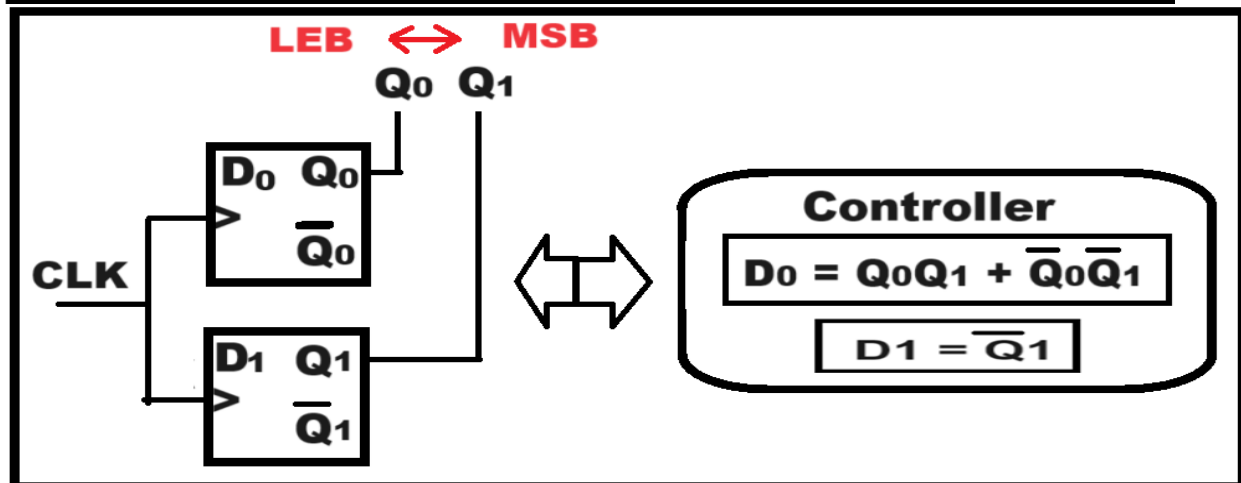
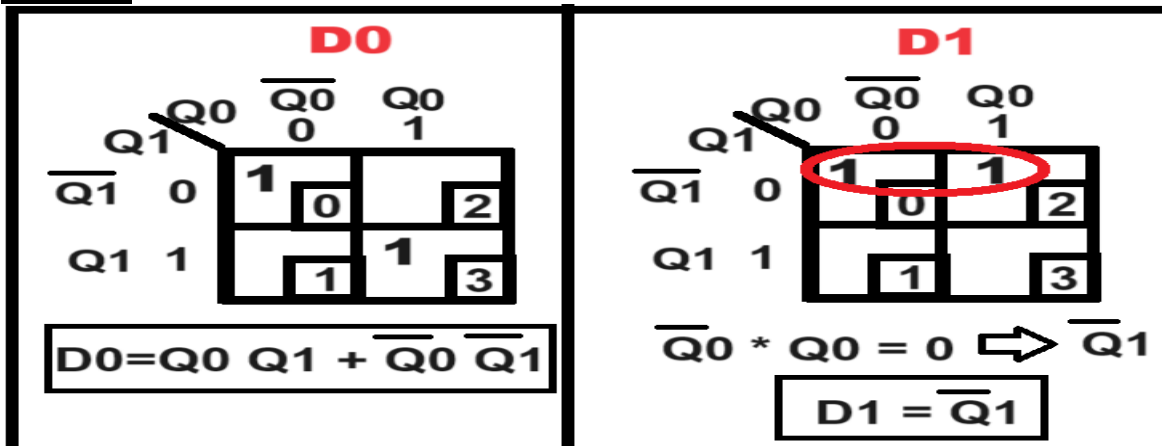
Sol:

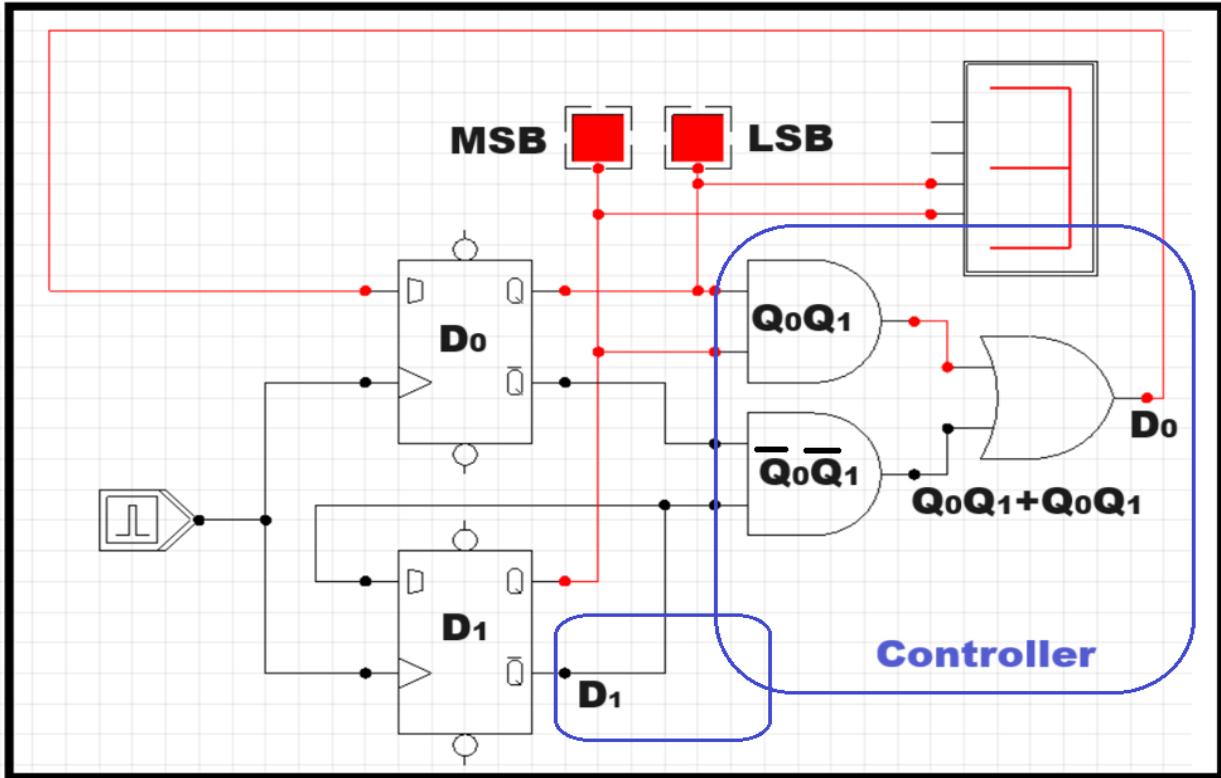
- 1- The largest number is $(3)_{10} = (11)_2 \rightarrow 2D\text{-FF}$
- 2- Controller design

Set	Outputs		inputs	
	Q1	Q0	D1	D0
3	1	1	0	1
1	0	1	1	0
2	1	0	0	0
0	0	0	1	1

D0				D1			
Set	Q1	Q0	D0	Set	Q1	Q0	D1
3	1	1	1	3	1	1	0
1	0	1	0	1	0	1	1
2	1	0	0	2	1	0	0
0	0	0	1	0	0	0	1

K-Maps





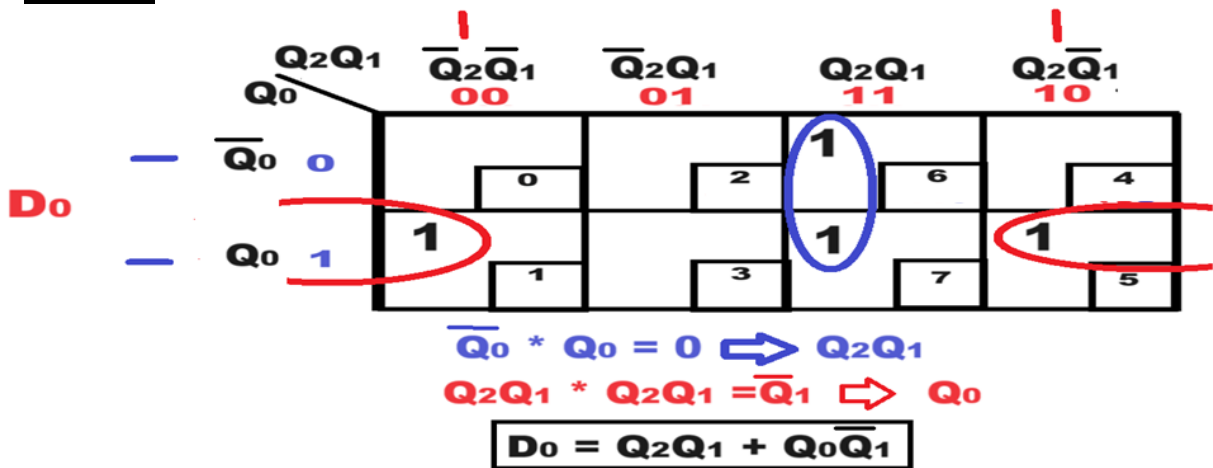
Example 2: Design $(7,5,3,6,1)_{10}$ counter.

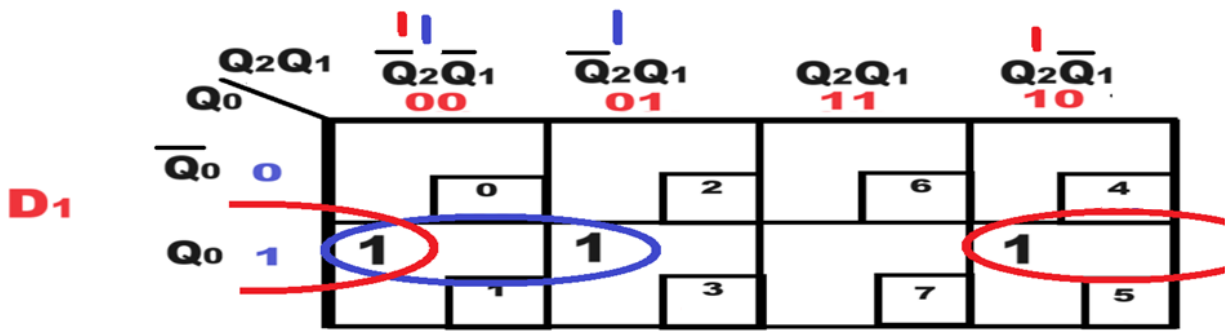
Sol:

- 1- The largest number is $(7)_{10} = (111)_2 \rightarrow 3D-FF$
- 2- Controller design

Set	Q_2	Q_1	Q_0	D_2	D_1	D_0
7	1	1	1	1	0	1
5	1	0	1	0	1	1
3	0	1	1	1	1	0
6	1	1	0	0	0	1
1	0	0	1	1	1	1

K-Maps

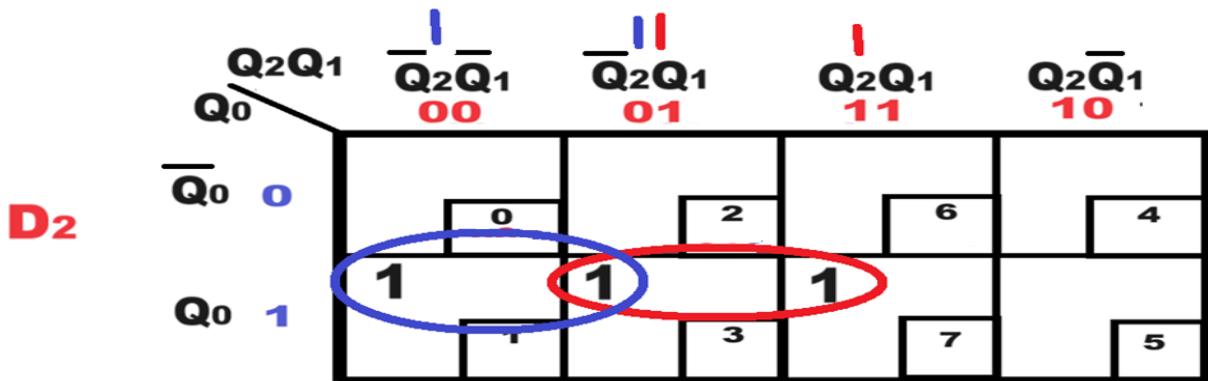




$$\bar{Q}_2\bar{Q}_1 * \bar{Q}_2Q_1 = \bar{Q}_2 \Rightarrow Q_0$$

$$\bar{Q}_2\bar{Q}_1 * Q_2\bar{Q}_1 = \bar{Q}_1 \Rightarrow Q_0$$

$$D_1 = Q_0\bar{Q}_1 + Q_0\bar{Q}_2$$



$$\bar{Q}_2\bar{Q}_1 * \bar{Q}_2Q_1 = \bar{Q}_2 \Rightarrow Q_0$$

$$\bar{Q}_2Q_1 * Q_2Q_1 = Q_1 \Rightarrow Q_0$$

$$D_2 = Q_0Q_1 + Q_0\bar{Q}_2$$

