

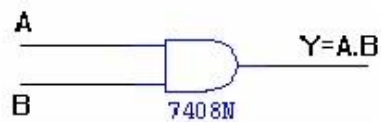
PRACTICAL-1

Aim: Study of Logic gates and their ICs and universal gates:

1. To verify the truth tables of OR, AND, NOR, NAND, X-OR, X-NOR gates
- b. To study IC 7400, 7402, 7404, 7408, 7432, 7486, 74266
- c. To implement and verify NAND and NOR as Universal gates

AND GATE:

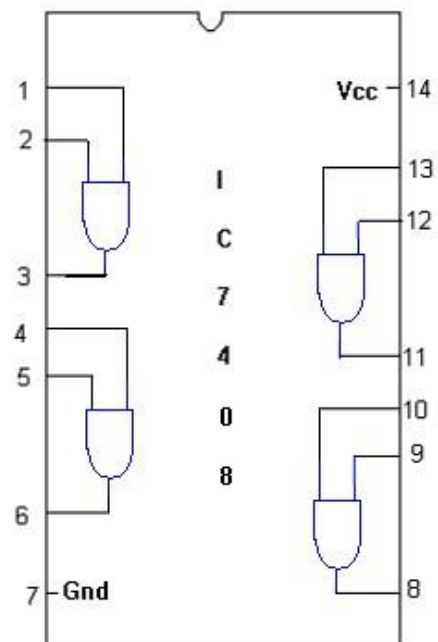
SYMBOL:



TRUTH TABLE

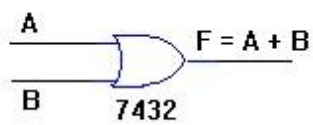
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

PIN DIAGRAM:



OR GATE:

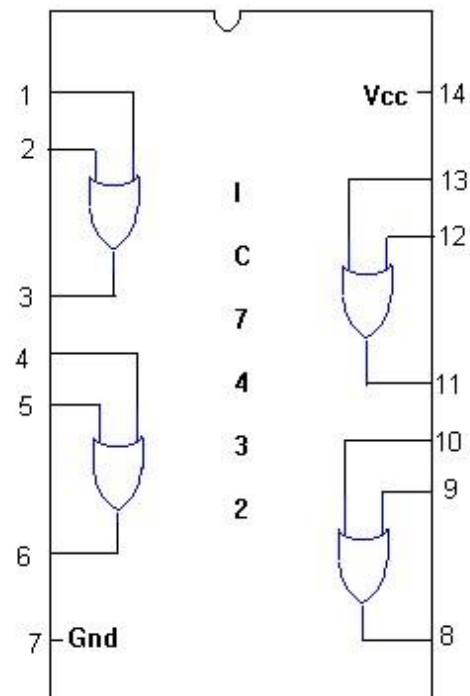
SYMBOL :



TRUTH TABLE

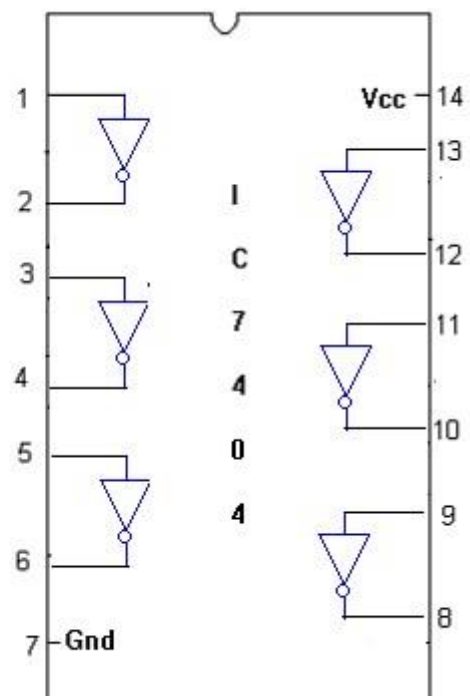
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

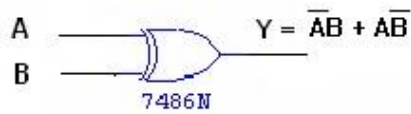
PIN DIAGRAM :

**NOT GATE:****SYMBOL:**

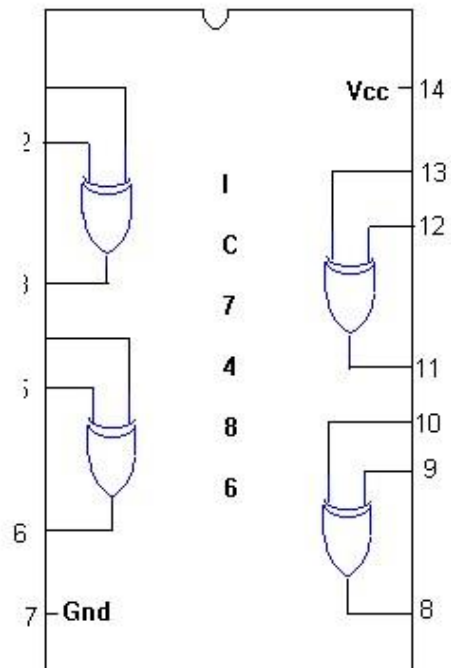
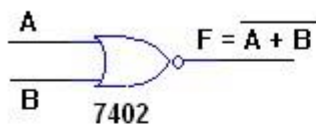
TRUTH TABLE :

A	\overline{A}
0	1
1	0

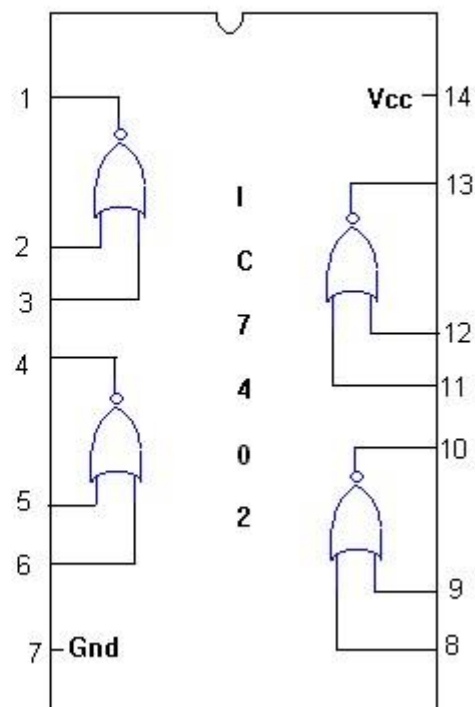
PIN DIAGRAM

X-OR GATE :**SYMBOL :****TRUTH TABLE :**

A	B	$\bar{A}B + A\bar{B}$
0	0	0
0	1	1
1	0	1
1	1	0


PIN DIAGRAM :**NOR GATE:****SYMBOL :****TRUTH TABLE**

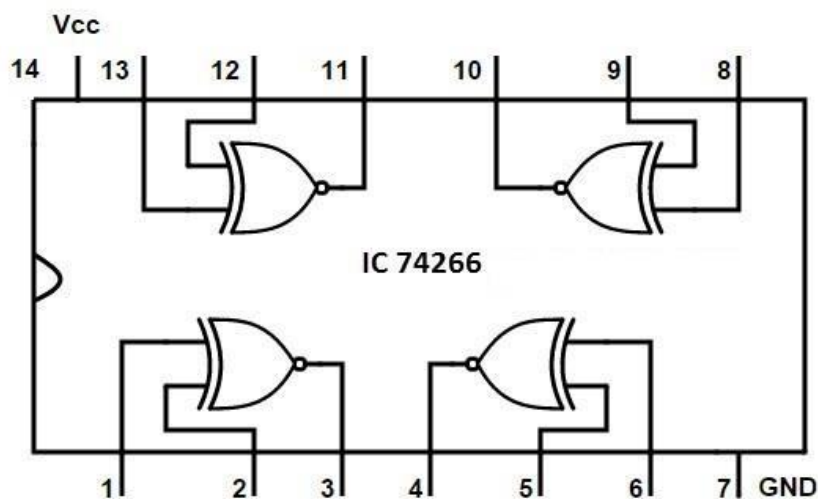
A	B	$\overline{A+B}$
0	0	1
0	1	1
1	0	1
1	1	0

PIN DIAGRAM :

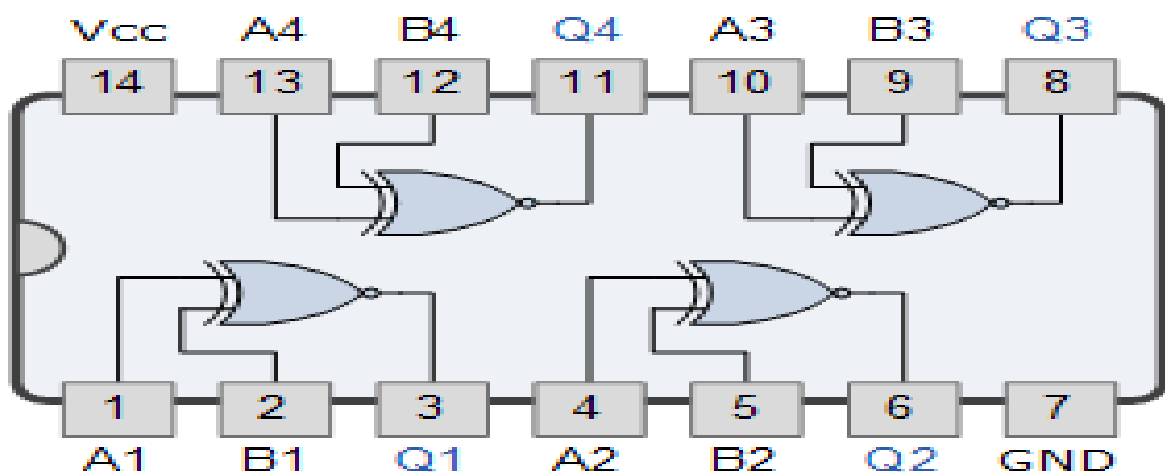
IC 74266

2-input "Ex-OR" gate plus a "NOT" gate

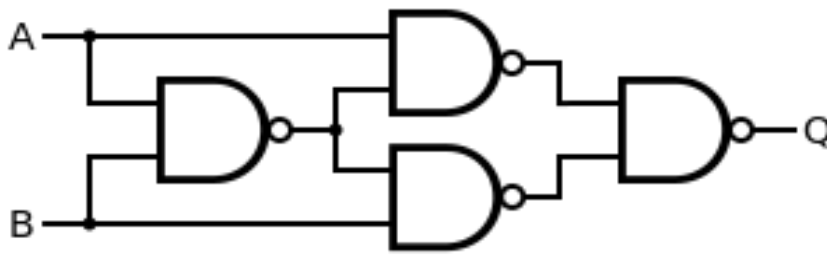
Symbol	Truth Table		
 2-input Ex-NOR Gate	B	A	Q
	0	0	1
	0	1	0
	1	0	0
	1	1	1
Boolean Expression $Q = \overline{A \oplus B}$		Read if A AND B the SAME gives Q	



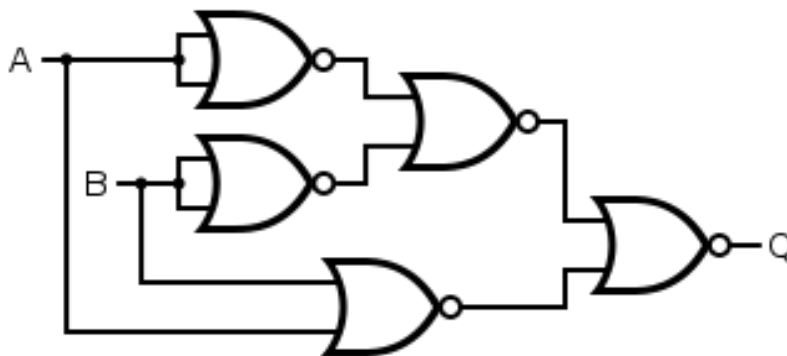
74266 Quad 2-input Ex-NOR Gate



XOR gate from NAND gates:



XOR gate from NOR gates:



PRACTICAL-2

Aim: Study of Boolean expressions

a. To verify De Morgan's laws

b. Implement the given expression using a minimum number of gates.

c. Implement the given expression using a minimum number of ICs

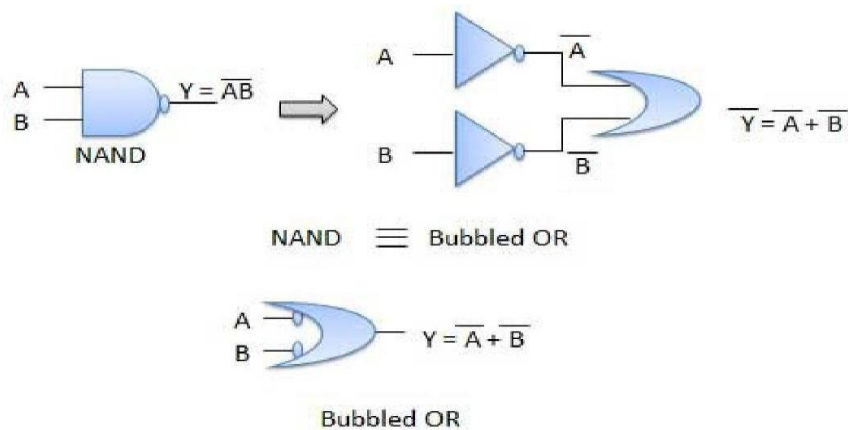
Theory: De Morgan has suggested two theorems which are extremely useful in Boolean Algebra. The two theorems are discussed below.

Theorem 1:

$$\overline{A \cdot B} = \overline{A} + \overline{B}$$

NAND = Bubbled OR

- The left hand side *LHS* of this theorem represents a NAND gate with inputs A and B, whereas the right hand side *RHS* of the theorem represents an OR gate with inverted inputs.
- This OR gate is called as Bubbled OR.



A	B	$\overline{A \cdot B}$	\overline{A}	\overline{B}	$\overline{A} + \overline{B}$
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

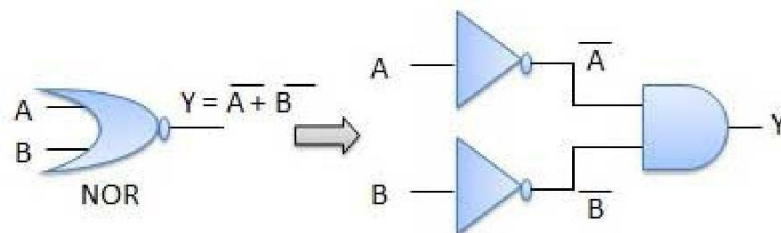
Theorem 2:

$$\overline{A + B} = \overline{A} \cdot \overline{B}$$

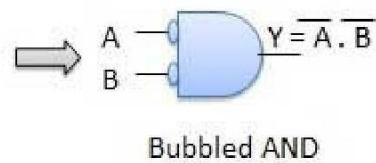
NOR = Bubbled AND

The LHS of this theorem represents a NOR gate with inputs A and B, whereas the RHS represents an AND gate with inverted inputs.

This AND gate is called as **Bubbled AND**.



NOR \equiv Bubbled AND



A	B	$\overline{A + B}$	\overline{A}	\overline{B}	$\overline{A} \cdot \overline{B}$
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	0

PRACTICAL-3

Aim: Design of Combinational Circuits using K-maps.

a. Design and implement combinational circuits for the given problem/problems using minimization techniques of K-maps.

Theory: Let us simplify the following Boolean function, $f(W,X,Y,Z) = WX'Y' + WY + W'YZ'$ using K-map.

The given Boolean function is in sum of products form. It is having 4 variables W, X, Y & Z. So, we require 4 variable K-map. The 4 variable K-map with ones corresponding to the given product terms is shown in the following figure.

WX \ YZ	00	01	11	10
00				1
01				1
11			1	1
10	1	1	1	1

Here, 1s are placed in the following cells of K-map.

- The cells, which are common to the intersection of Row 4 and columns 1 & 2 are corresponding to the product term, $WX'Y'$.
- The cells, which are common to the intersection of Rows 3 & 4 and columns 3 & 4 are corresponding to the product term, WY .
- The cells, which are common to the intersection of Rows 1 & 2 and column 4 are corresponding to the product term, $W'YZ'$.

There are no possibilities of grouping either 16 adjacent ones or 8 adjacent ones. There are three possibilities of grouping 4 adjacent ones. After these three groupings, there is no single one left as ungrouped. So, we no need to check for grouping of 2 adjacent ones. The 4 variable K-map with these three groupings is shown in the following figure.

WX \ YZ	00	01	11	10
00				1
01				1
11			1	1
10	1	1	1	1

WX'
 YZ'
 WY

Here, we got three prime implicants WX' , WY & YZ' . All these prime implicants are essential because of following reasons.

- Two ones (m_8 & m_9) of fourth row grouping are not covered by any other groupings. Only fourth row grouping covers those two ones.
- Single one (m_{15}) of square shape grouping is not covered by any other groupings. Only the square shape grouping covers that one.
- Two ones (m_2 & m_6) of fourth column grouping are not covered by any other groupings. Only fourth column grouping covers those two ones.

Therefore, the simplified Boolean function is

$$f = WX' + WY + YZ'$$

Follow these rules for simplifying K-maps in order to get standard product of sums form.

- Select the respective K-map based on the number of variables present in the Boolean function.
- If the Boolean function is given as product of Max terms form, then place the zeroes at respective Max term cells in the K-map. If the Boolean function is given as product of sums form, then place the zeroes in all possible cells of K-map for which the given sum terms are valid.
- Check for the possibilities of grouping maximum number of adjacent zeroes. It should be powers of two. Start from highest power of two and upto least power of two. Highest power is equal to the number of variables considered in K-map and least power is zero.
- Each grouping will give either a literal or one sum term. It is known as prime implicant. The prime implicant is said to be essential prime implicant, if atleast single '0' is not covered with any other groupings but only that grouping covers.
- Note down all the prime implicants and essential prime implicants. The simplified Boolean function contains all essential prime implicants and only the required prime implicants.

Note – If don't care terms also present, then place don't cares 'x' in the respective cells of K-map. Consider only the don't cares 'x' that are helpful for grouping maximum number of adjacent zeroes. In those cases, treat the don't care value as '0'.

Example

Let us simplify the following Boolean

function, $f(X,Y,Z)=\prod M(0,1,2,4)$ using K-map.

The given Boolean function is in product of Max terms form. It is having 3 variables X, Y & Z. So, we require 3 variable K-map. The given Max terms are M_0, M_1, M_2 & M_4 . The 3 variable K-map with zeroes corresponding to the given Max terms is shown in the following figure.

X \ YZ	00	01	11	10
	0	0		0
1	0			

There are no possibilities of grouping either 8 adjacent zeroes or 4 adjacent zeroes. There are three possibilities of grouping 2 adjacent zeroes. After these three groupings, there is no single zero left as ungrouped. The 3 variable K-map with these three groupings is shown in the following figure.

X \ YZ	00	01	11	10
	0	0		0
1	0			

Groupings shown in the K-map:

- Red box: $X + Y$ (covers cells (0,00) and (1,00))
- Blue box: $Y + Z$ (covers cells (0,00) and (0,01))
- Green box: $Z + X$ (covers cells (0,01) and (1,10))

Here, we got three prime implicants $X + Y, Y + Z$ & $Z + X$. All these prime implicants are essential because one zero in each grouping is not covered by any other groupings except with their individual groupings.

Therefore, the simplified Boolean function is

$$f = X + Y + Z$$

In this way, we can easily simplify the Boolean functions up to 5 variables using K-map method. For more than 5 variables, it is difficult to simplify the functions using K-Maps. Because, the number of cells in K-map gets doubled by including a new variable.

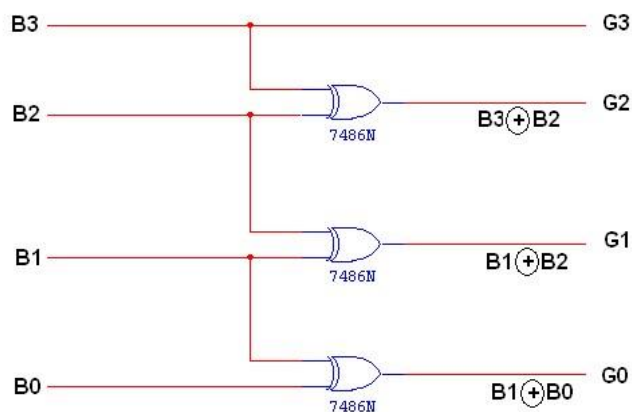
PRACTICAL-4

Aim: Design and implement code converters

- a. Design the circuit and implement Binary to gray code converter
- b. Design the circuit and implement Gray to Binary code converter
- c. Design the circuit and implement Binary to BCD code converter
- d. Design the circuit and implement Binary to XS-3 code converter

LOGIC DIAGRAM:

BINARY TO GRAY CODE CONVERTOR



K-Map for G_3 :

B3B2 \ B1B0				
	00	01	11	10
00				
01				
11	1	1	1	1
10	1	1	1	1

$$G_3 = B_3$$

K-Map for G_1 :

K-Map for G_2 :

B3B2 \ B1B0				
	00	01	11	10
00				
01	1	1	1	1
11				
10	1	1	1	1

$$G_2 = B_3 \oplus B_2$$

K-Map for G_0 :

B3B2	B1B0			
	00	01	11	10
00			1	1
01	1	1		
11	1	1		
10			1	1

$$G1 = B1 \oplus B2$$

B3B2	B1B0			
	00	01	11	10
00		1		1
01		1		1
11		1		1
10		1		1

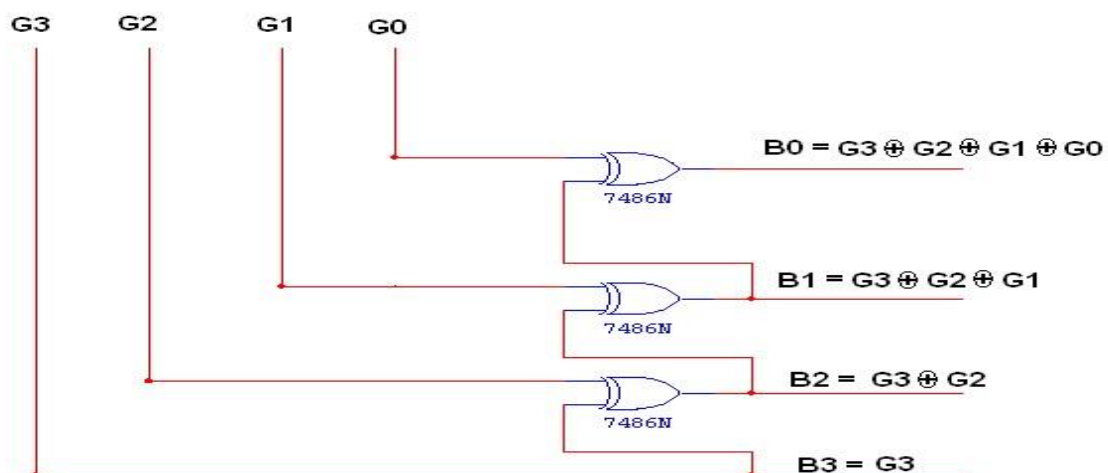
$$G0 = B1 \oplus B0$$

TRUTH TABLE:

B3	B2	B1	B0	G3	G2	G1	G0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

LOGIC DIAGRAM:

GRAY CODE TO BINARY CONVERTOR



TRUTH TABLE:

G3	G2	G1	G0	B3	B2	B1	B0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

K-Map for B₃:

G3G2 \ G1G0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

$$B_3 = G_3$$

G3G2 \ G1G0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

$$B_2 = G_3 \oplus G_2$$

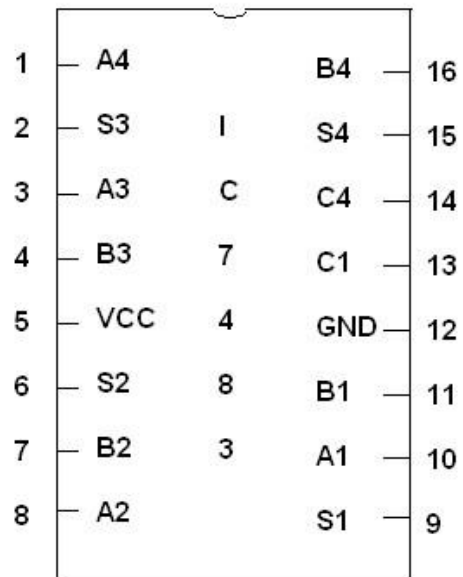
K-Map for B₁:

G3G2 \ G1G0	00	01	11	10
00	0	①	0	①
01	①	0	①	0
11	0	①	0	①
10	①	0	①	0

$$B_0 = G_3 \oplus G_2 \oplus G_1 \oplus G_0$$

G3G2 \ G1G0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	0	0	1	1
10	1	1	0	0

$$B_1 = G_3 \oplus G_2 \oplus G_1$$

PIN DIAGRAM FOR IC 7483:

PRACTICAL-5

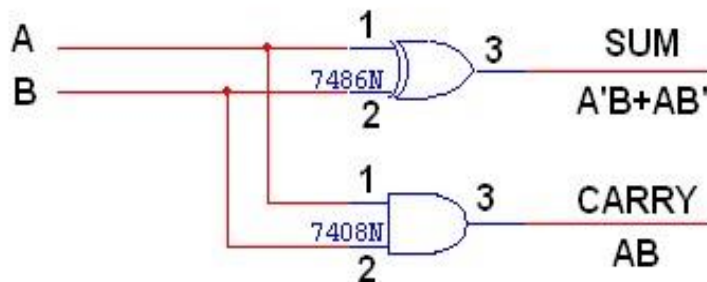
Aim: Implement Adder and Subtractor circuits

a. Design the circuit and implement Half Adder and Full Adder

b. Design the circuit and implement BCD Adder, XS-3 Adder , Binary Subtractor

LOGIC DIAGRAM:

HALF ADDER



TRUTH TABLE:

A	B	CARRY	SUM
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

K-Map for SUM:

B	00	01
A 00		1
A 01	1	

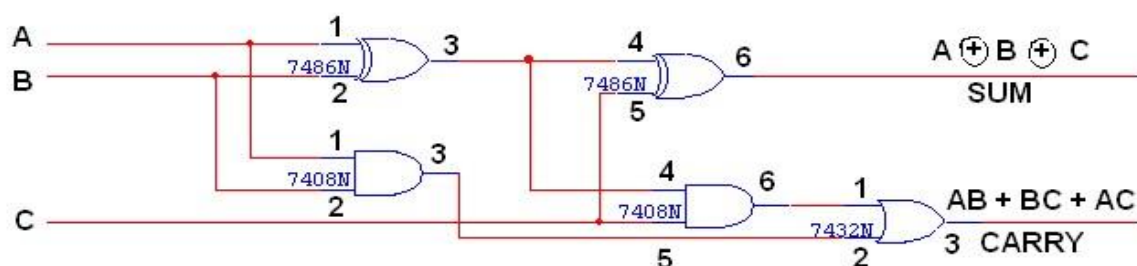
K-Map for CARRY:

B	00	01
A 00		
A 01		1

$$\text{SUM} = A'B + AB'$$

$$\text{CARRY} = AB$$

FULL ADDER USING TWO HALF ADDER



TRUTH TABLE:

A	B	C	CARRY	SUM
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

K-Map for SUM:

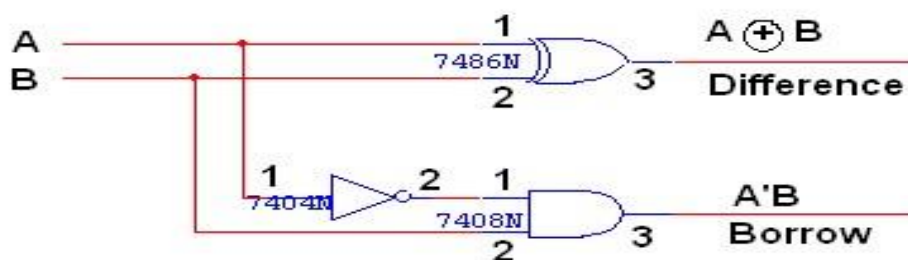
BC	00	01	11	10
A=0		1		1
A=1	1		1	

K-Map for CARRY:

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

$$\text{SUM} = A'B'C + A'BC' + ABC' + ABC$$

$$\text{CARRY} = AB + BC + AC$$

LOGIC DIAGRAM:**HALF SUBTRACTOR****TRUTH TABLE:**

A	B	BORROW	DIFFERENCE
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

K-Map for DIFFERENCE:

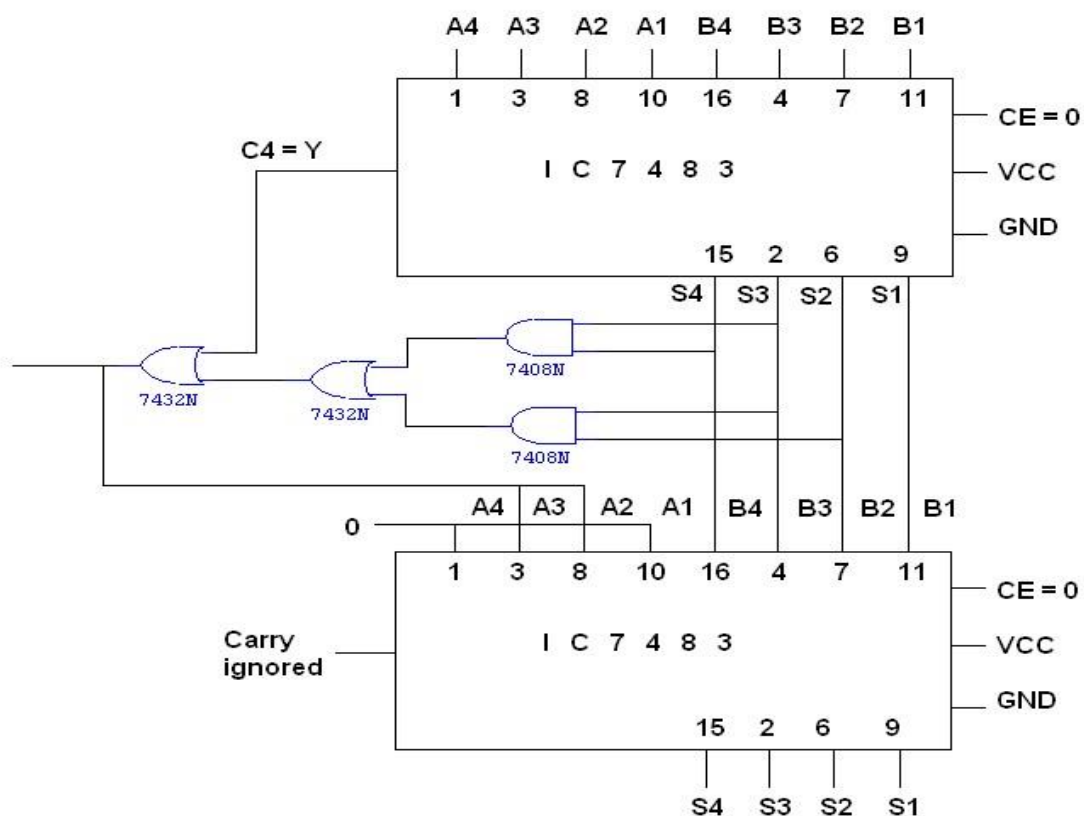
A \ B	00	01
00		1
01	1	

$$\text{DIFFERENCE} = A'B + AB'$$

K-Map for BORROW:

A \ B	00	01
00		1
01		

$$\text{BORROW} = A'B$$

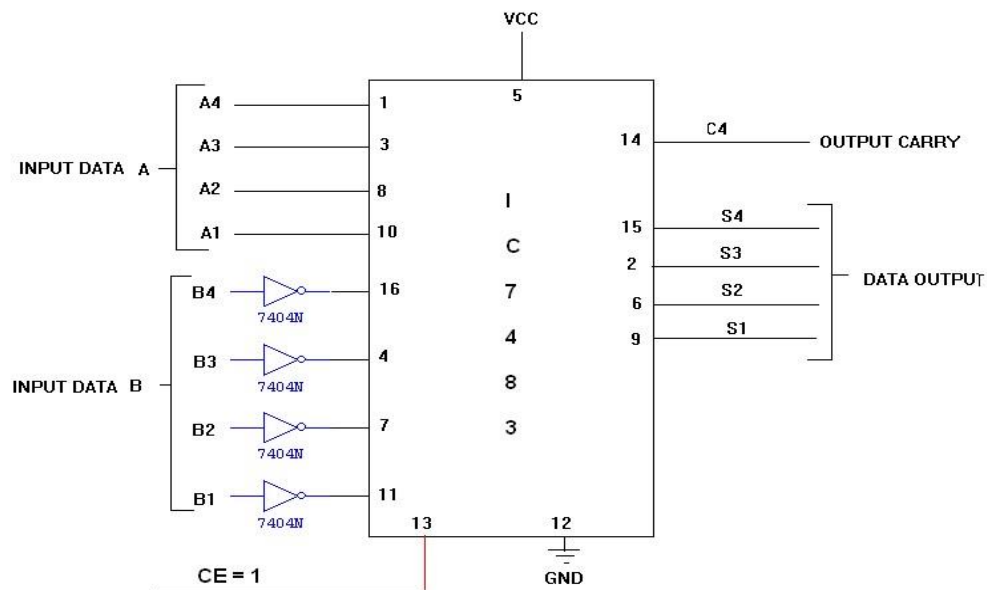
BCD ADDER

TRUTH TABLE:

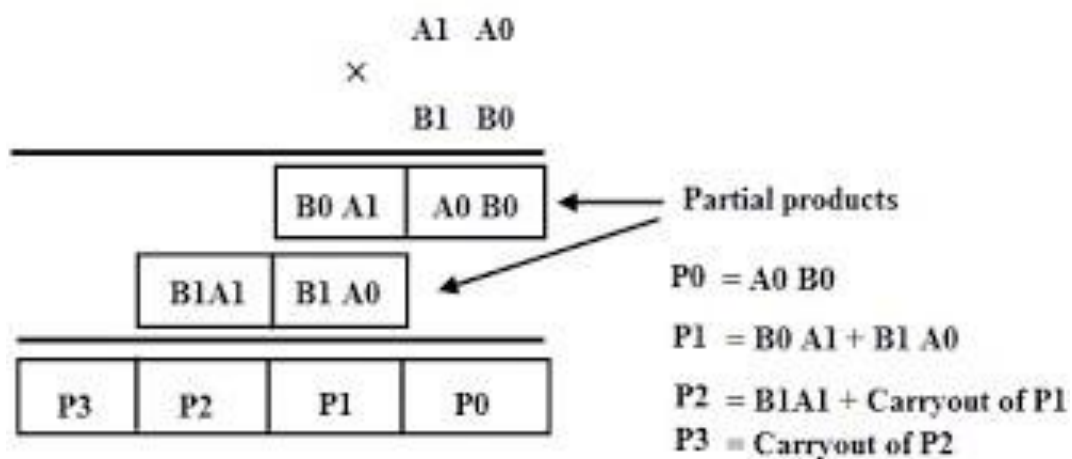
BCD SUM				CARRY
S4	S3	S2	S1	C
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

K MAP

		S1 S2			
S3 S4		00	01	11	10
	00	0	0	0	0
	01	0	0	0	0
	11	1	1	1	1
	10	0	0	1	1

LOGIC DIAGRAM:**BINARY SUBTRACTOR**

Input Data A				Input Data B				Addition					Subtraction				
A4	A3	A2	A1	B4	B3	B2	B1	C	S4	S3	S2	S1	B	D4	D3	D2	D1
1	0	0	0	0	0	1	0	0	1	0	1	0	1	0	1	1	0
1	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0
0	0	1	0	1	0	0	0	0	1	0	1	0	0	1	0	1	0
0	0	0	1	0	1	1	1	0	1	0	0	0	0	1	0	1	0
1	0	1	0	1	0	1	1	1	0	0	1	0	0	1	1	1	1
1	1	1	0	1	1	1	1	1	1	0	1	0	0	1	1	1	1
1	0	1	0	1	1	0	1	1	0	1	1	1	0	1	1	0	1

PRACTICAL-6**Aim: Design and implement Arithmetic circuits****a. Design and implement 2-by-2 bit multiplier.****Design and implement a 2-bit comparator**

$$\begin{array}{r}
 A_1 A_0 \\
 \times B_1 B_0 \\
 \hline
 B_1A_1 B_0A_0 \\
 B_1A_1 B_1A_0 \\
 x
 \end{array}$$

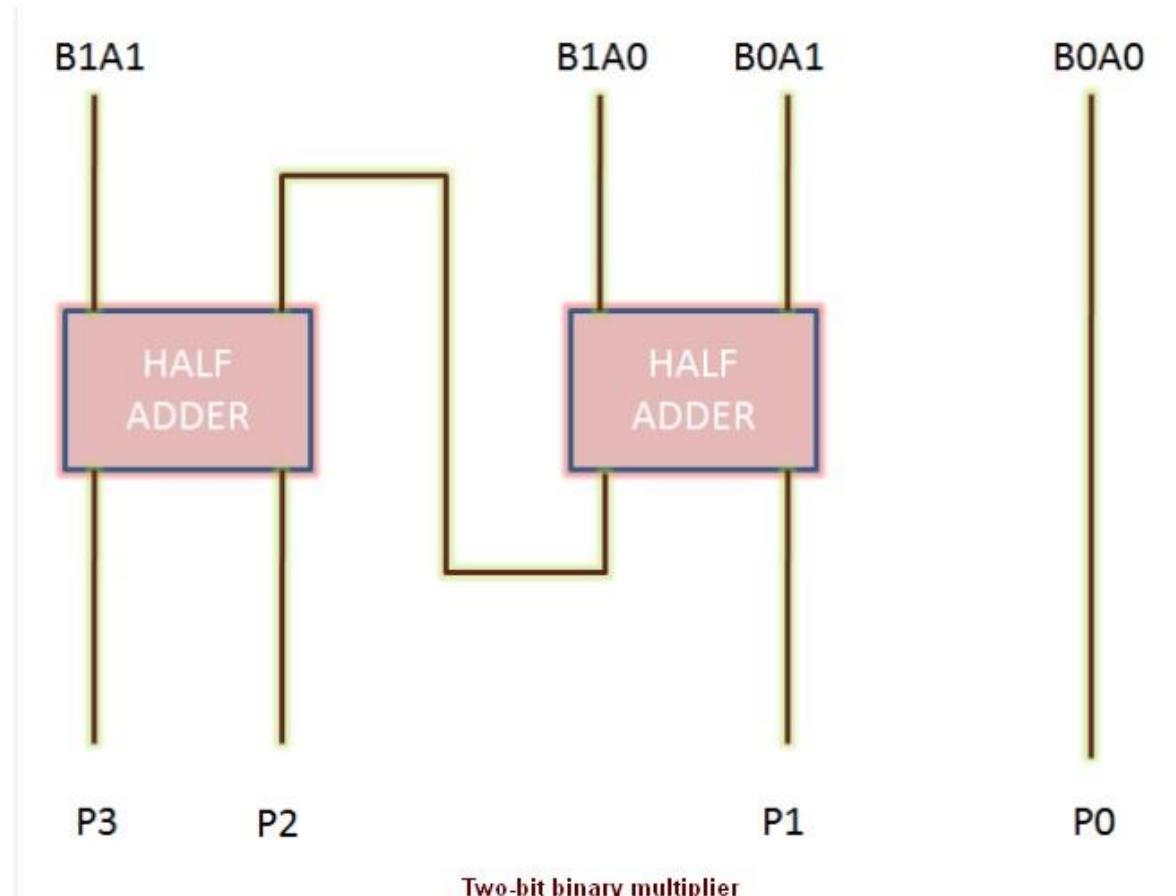
we get the partial products as:

$$P0 = A0 * B0$$

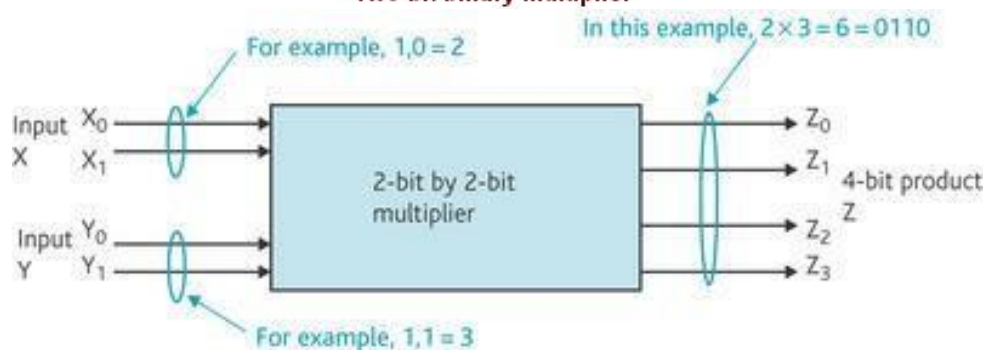
$$P1 = A0 * B1 \text{ xor } A1 * B0 \quad ; \text{ carry generated here goes to next stage}$$

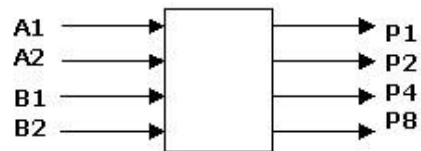
$$P2 = A1 * B1 \text{ xor } (A0 * B1) * (A1 * B0)$$

$$P3 = A1 * B1 \text{ and } (A0 * B1) * (A1 * B0)$$



Two-bit binary multiplier





block diagram
and
truth table

A2	A1	B2	B1	P8	P4	P2	P1
0	0	0	0	0	0	0	0
		0	1	0	0	0	0
		1	0	0	0	0	0
		1	1	0	0	0	0
0	1	0	0	0	0	0	0
		0	1	0	0	0	1
		1	0	0	0	1	0
		1	1	0	0	1	1
1	0	0	0	0	0	0	0
		0	1	0	0	1	0
		1	0	0	1	0	0
		1	1	0	1	1	0
1	1	0	0	0	0	0	0
		0	1	0	0	1	1
		1	0	0	1	1	0
		1	1	1	0	0	1

4-variable K-map
for each of the 4
output functions

PRACTICAL-7

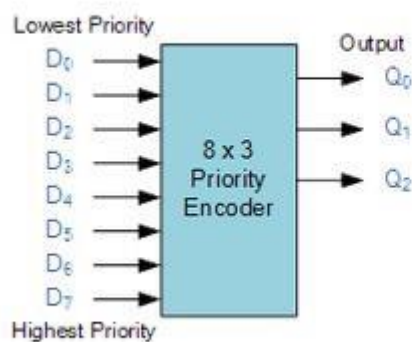
Aim: Implement Encoders and Decoders

a. Design and implement 8: 3 encoder

b. Design and implement 3:8 decoder

Design and implement 8:3 encoder :

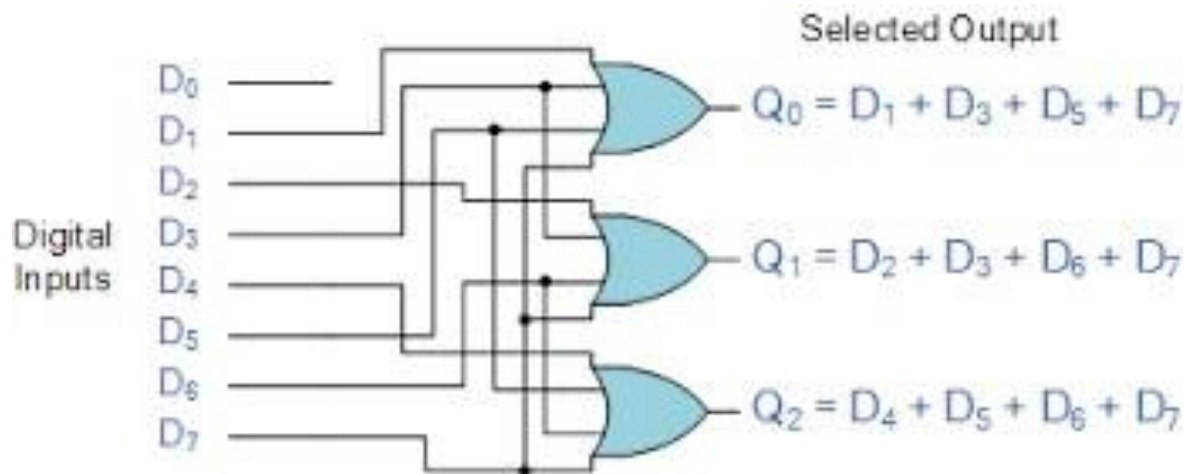
8-to-3 Bit Priority Encoder



Inputs								Outputs		
D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	Q ₂	Q ₁	Q ₀
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

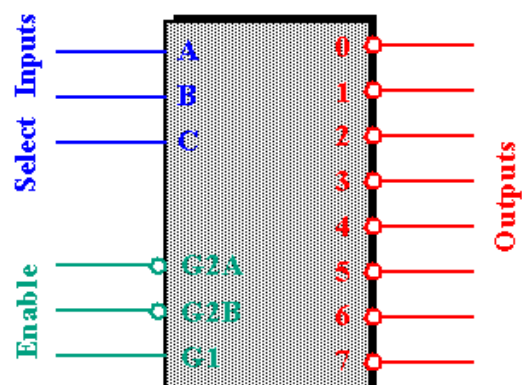
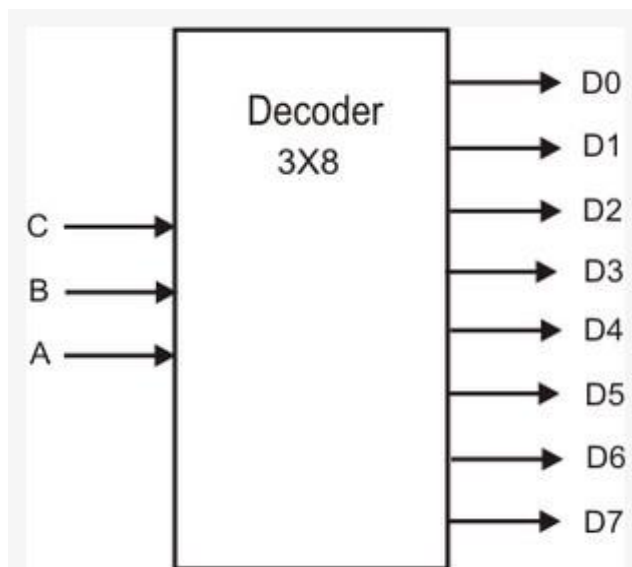
X = don't care

Digital Inputs								Binary Output		
D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	Q ₂	Q ₁	Q ₀
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	1	X	X	0	1	0
0	0	0	0	1	X	X	X	0	1	1
0	0	0	1	X	X	X	X	1	0	0
0	0	1	X	X	X	X	X	1	0	1
0	1	X	X	X	X	X	X	1	1	0
1	X	X	X	X	X	X	X	1	1	1



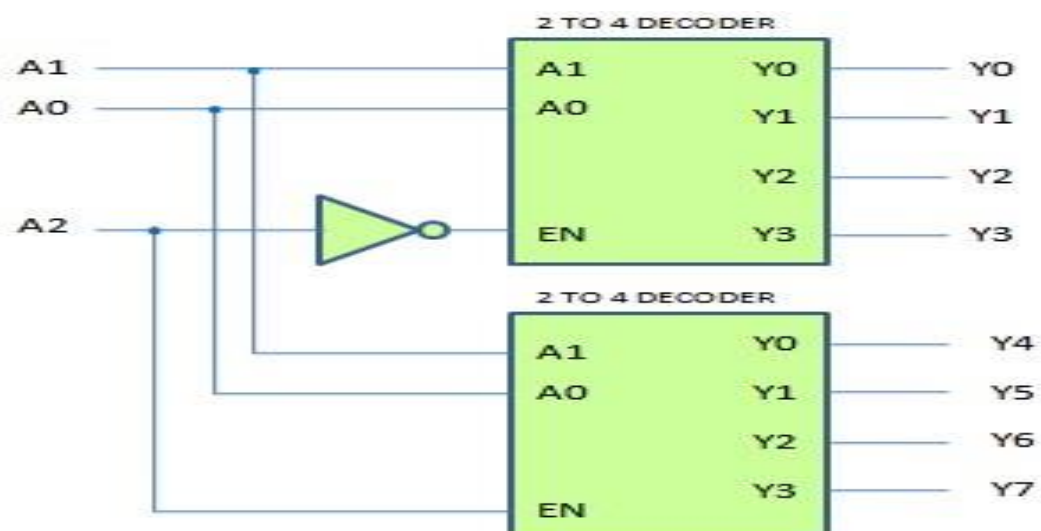
Digital Inputs								Binary Output		
D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	Q ₂	Q ₁	Q ₀
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	1	X	X	0	1	0
0	0	0	0	1	X	X	X	0	1	1
0	0	0	1	X	X	X	X	1	0	0
0	0	1	X	X	X	X	X	1	0	1
0	1	X	X	X	X	X	X	1	1	0
1	X	X	X	X	X	X	X	1	1	1

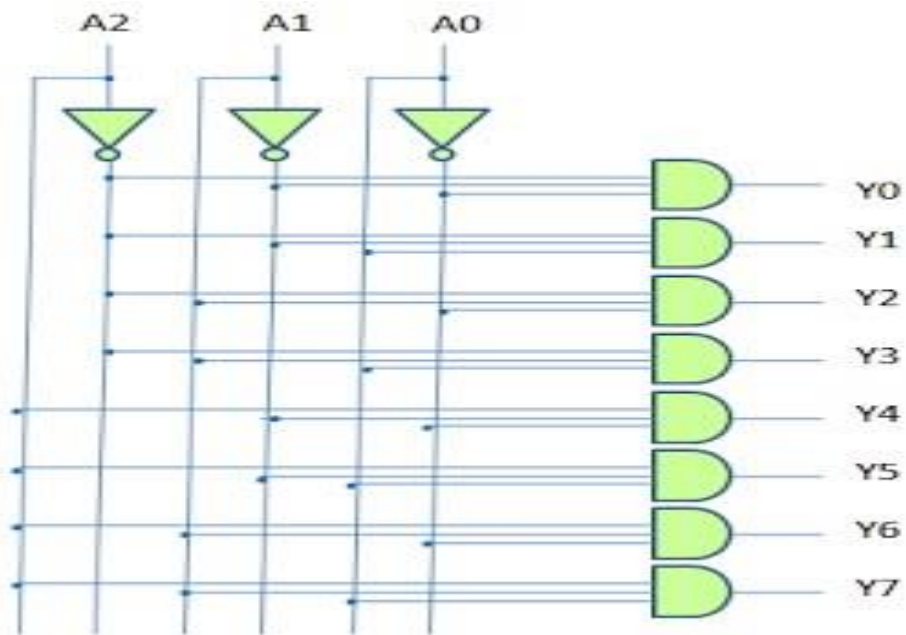
Design and implement 3:8 decoder :



Inputs						Output							
Enable			Select										
G2A	G2B	G1	C	B	A	0	1	2	3	4	5	6	7
1	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	0	X	X	X	1	1	1	1	1	1	1	1
0	0	1	0	0	0	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	0	1	1	1	1	1	1
0	0	1	0	1	0	1	1	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1	0	1	1	1	1
0	0	1	1	0	0	1	1	1	1	0	1	1	1
0	0	1	1	0	1	1	1	1	1	1	0	1	1
0	0	1	1	1	0	1	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	1	1	1	1	1	0

3 to 8 decoder using 2 to 4 decoders:



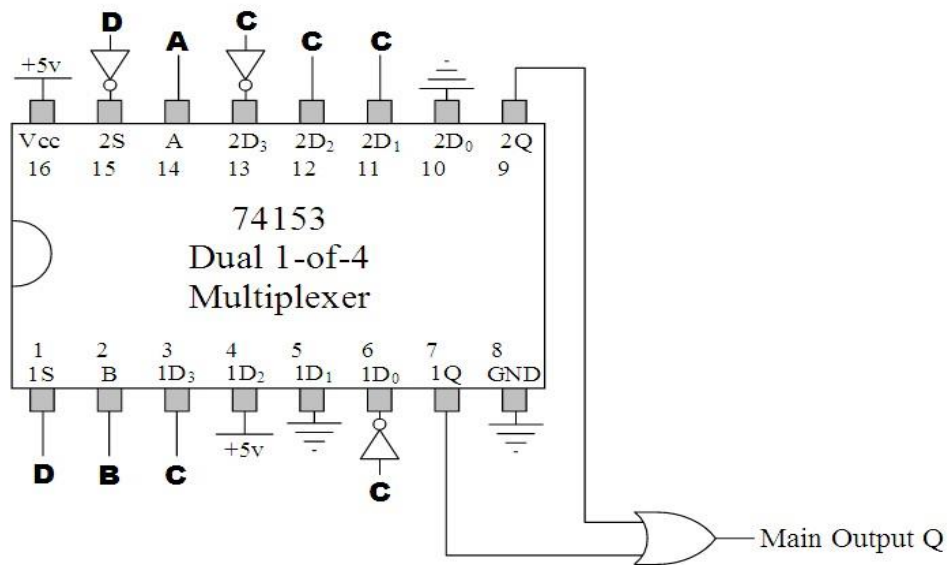
3 to 8 decoder using gates:

PRACTICAL-8

Aim: Multiplexers and Demultiplexers

- a. Design and Implement 4:1 multiplexer
- b. Design and Implement 1:4 demultiplexer
- c. Study IC 74151 8: 1 multiplexer and implement the expression
- d. Study IC 74138 3: 8 decoder and implement the expression

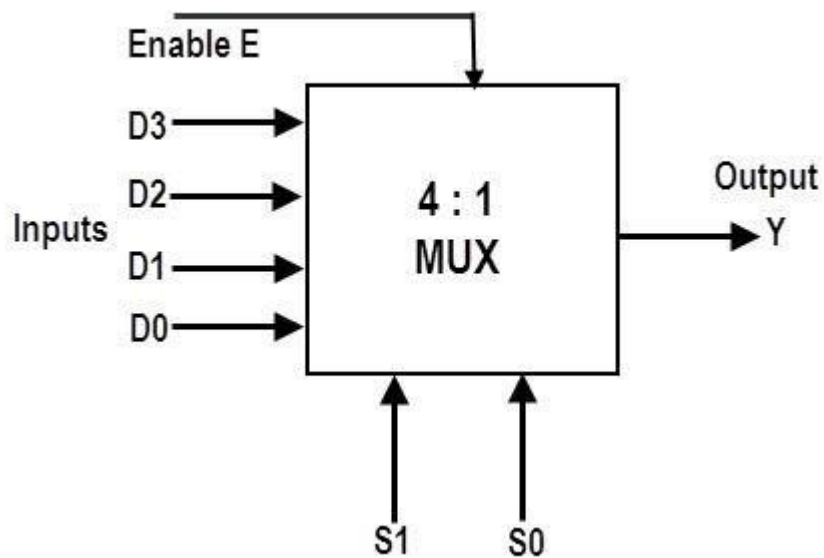
Design and implement 4:1 multiplexer. Study of IC 74153



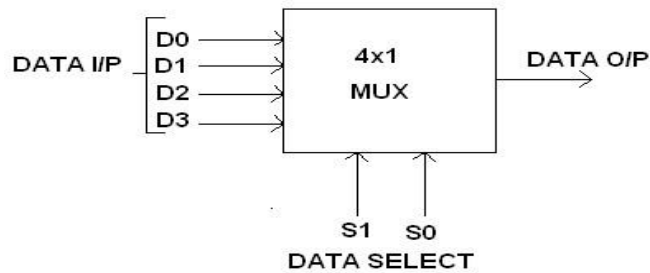
D	C	B	A	Q	D _i
0	0	0	0	1	1D ₀
0	0	0	1	0	1D ₁
0	0	1	0	1	1D ₂
0	0	1	1	0	1D ₃
0	1	0	0	0	1D ₀
0	1	0	1	0	1D ₁
0	1	1	0	1	1D ₂
0	1	1	1	1	1D ₃
1	0	0	0	0	2D ₀
1	0	0	1	0	2D ₁
1	0	1	0	0	2D ₂
1	0	1	1	1	2D ₃
1	1	0	0	0	2D ₀
1	1	0	1	1	2D ₁
1	1	1	0	1	2D ₂
1	1	1	1	0	2D ₃

Design and implement 4:1 multiplexer.

Study of IC 74157

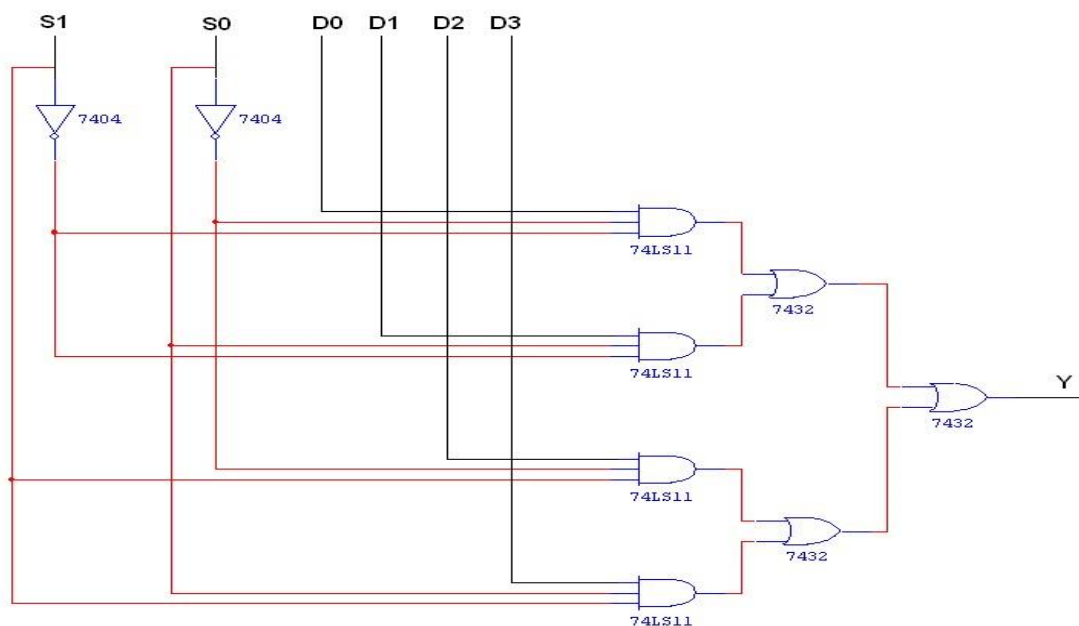


Select Data Inputs		Output
S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

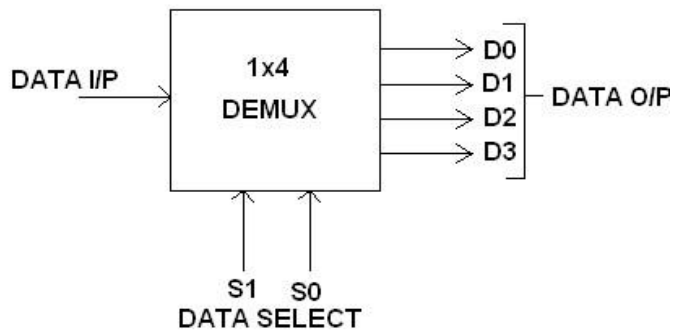
BLOCK DIAGRAM FOR 4:1 MULTIPLEXER:**FUNCTION TABLE:**

S1	S0	INPUTS Y
0	0	$D0 \rightarrow D0 S1' S0'$
0	1	$D1 \rightarrow D1 S1' S0$
1	0	$D2 \rightarrow D2 S1 S0'$
1	1	$D3 \rightarrow D3 S1 S0$

$$Y = D0 S1' S0' + D1 S1' S0 + D2 S1 S0' + D3 S1 S0$$

CIRCUIT DIAGRAM FOR MULTIPLEXER:**TRUTH TABLE:**

S1	S0	Y = OUTPUT
0	0	D0
0	1	D1
1	0	D2
1	1	D3

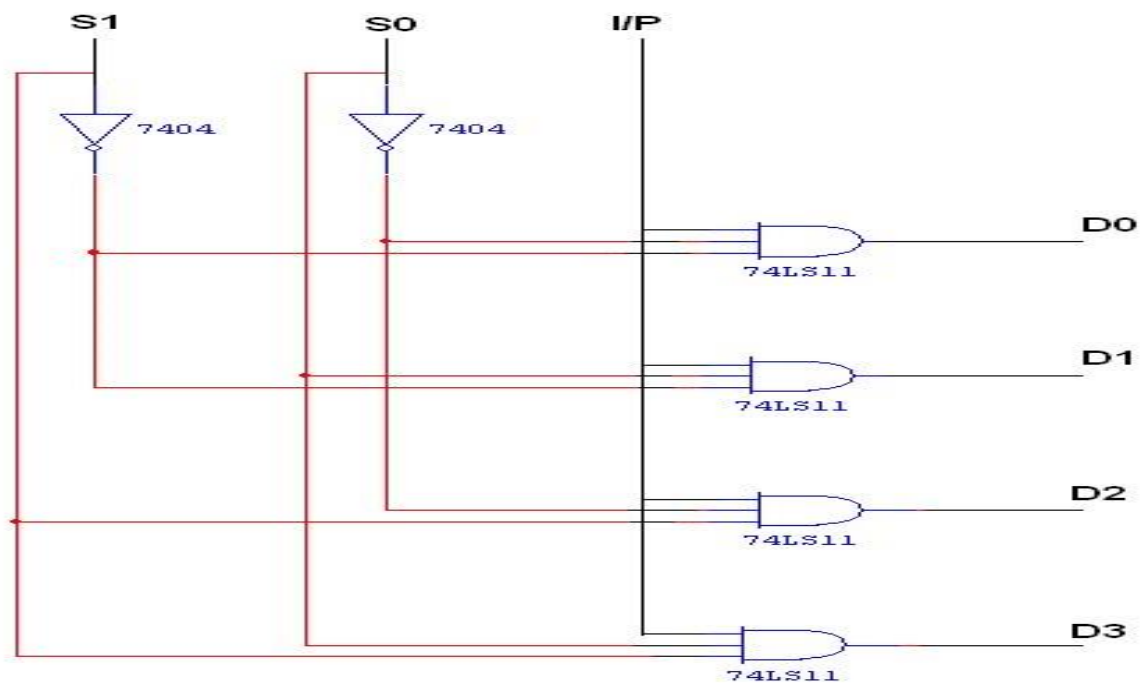
BLOCK DIAGRAM FOR 1:4 DEMULTIPLEXER:**FUNCTION TABLE:**

S1	S0	INPUT
0	0	$X \rightarrow D0 = X S1' S0'$
0	1	$X \rightarrow D1 = X S1' S0$
1	0	$X \rightarrow D2 = X S1 S0'$
1	1	$X \rightarrow D3 = X S1 S0$

$$Y = X S1' S0' + X S1' S0 + X S1 S0' + X S1 S0$$

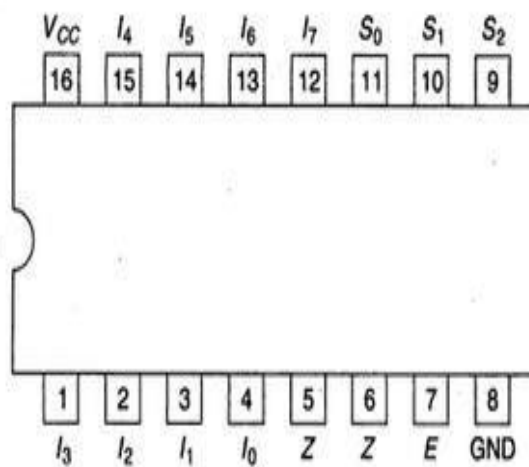
TRUTH TABLE:

INPUT			OUTPUT			
S1	S0	I/P	D0	D1	D2	D3
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	1	0
1	1	0	0	0	0	0
1	1	1	0	0	0	1

LOGIC DIAGRAM FOR DEMULTIPLEXER:**TRUTH TABLE:**

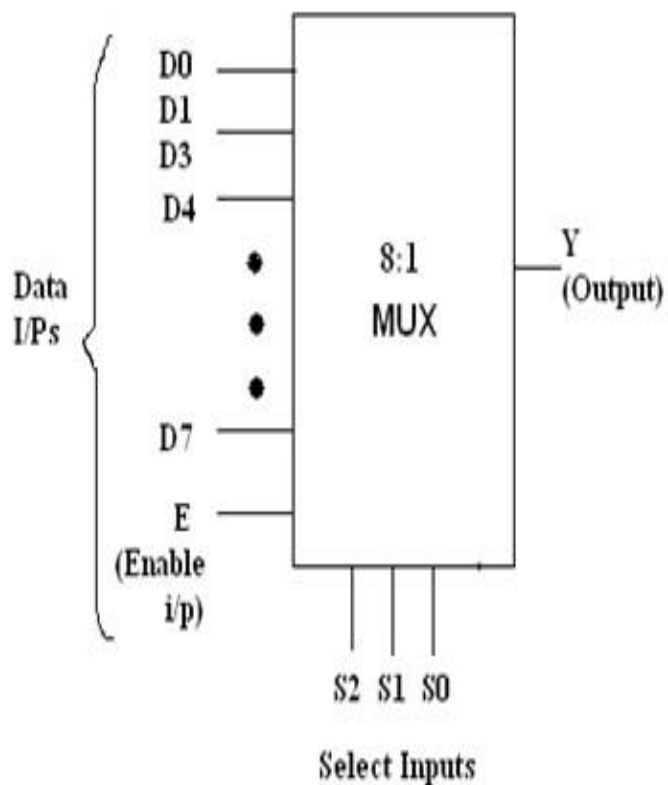
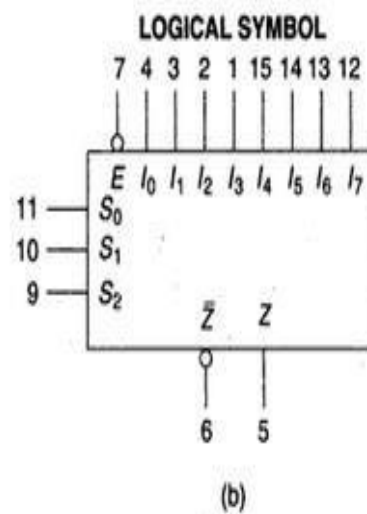
INPUT			OUTPUT			
S1	S0	I/P	D0	D1	D2	D3
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	1	0
1	1	0	0	0	0	0
1	1	1	0	0	0	1

Implement the given expression using IC 74151 8:1 multiplexer



PIN NAMES

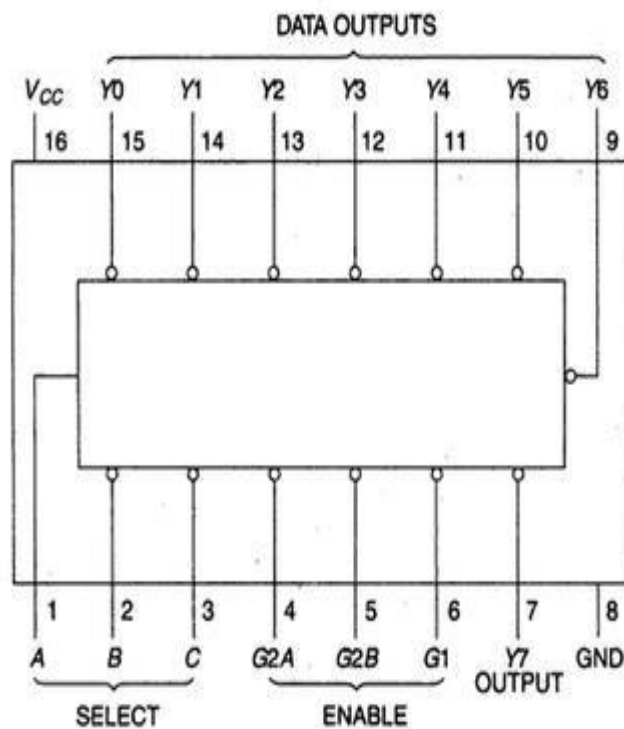
- $S_0 - S_2$ Select inputs
 E Enable (Active LOW) inputs
 $I_0 - I_7$ Multiplexer inputs
 Z Multiplexer outputs (note b)
 \bar{Z} Complementary multiplexer output



Enable	Select Inputs			Output
E	S2	S1	S0	Y
0	X	X	X	0
1	0	0	0	D0
1	0	0	1	D1
1	0	1	0	D2
1	0	1	1	D3
1	0	0	0	D4
1	0	0	1	D5
1	0	1	0	D6
1	0	1	1	D7

Implement the given expression using

IC 74138 3:8 decoder



Inputs				Outputs								
Enable		Select										
G1	G2	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

- a. Equipment : Bread Board, Power Supply, Resistors, LEDs or Digital IC Trainer Kit
- b. Discrete Components - IC7473 Dual JK Flip-flop

Theory

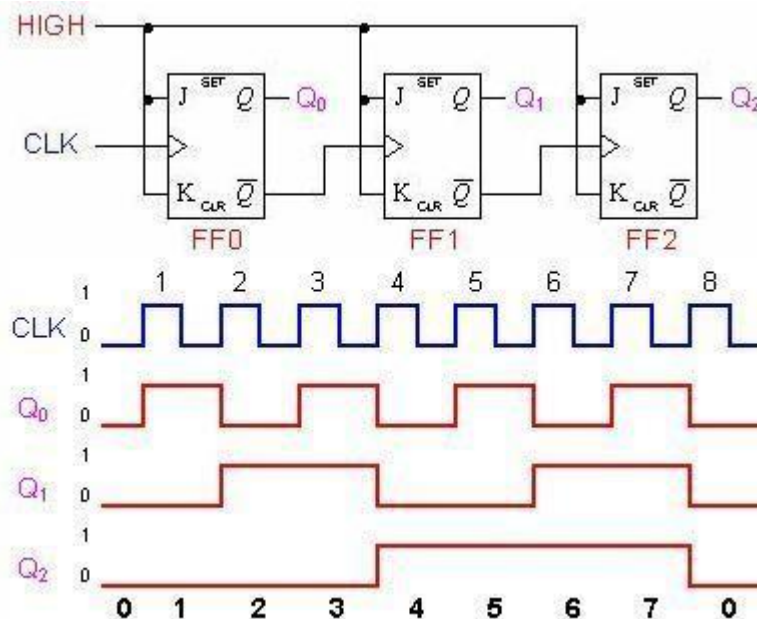
Asynchronous Counter is sequential circuit that is used to count the number of clock input signal. The output of one flip flop is given as a clock input to another flip-flop, so it is called as Serial Counter.

A ripple counter is an asynchronous counter where only the first flip-flop is clocked by an external clock. All subsequent flip-flops are clocked by the output of the preceding flip-flop. Asynchronous counters are also called ripple-counters because of the way the clock pulse ripples it way through the flip-flops.

The MOD of the ripple counter or asynchronous counter is 2^n if n flip-flops are used. A three-bit asynchronous counter is shown on the below figure . The external clock is connected to the clock input of the first flip-flop (FF0) only. So, FF0 changes state at the falling edge of each clock pulse, but FF1 changes only when triggered by the falling edge of the Q output of FF0 similarly FF2 changes only when triggered by the falling edge of the Q output of FF1. Because of the inherent propagation delay through a flip-flop, the transition of the input clock pulse and a transition of the Q output of FF0 can never occur at exactly the same time. Therefore, the flip-flops cannot be triggered simultaneously, producing an asynchronous operation.

Usually, all the CLEAR inputs are connected together, so that a single pulse can clear all the flip-flops before counting starts. The clock pulse fed into FF0 is rippled through the other counters after propagation delays, like a ripple on water, hence the name Ripple Counter.

Logic Diagram with Timing Diagram:



Truth table:

FF2	FF1	FF0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1



Result: Thus the timing diagram and state diagram of 3 bit asynchronous Ripple counter was verified.