PRACTICAL-1

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

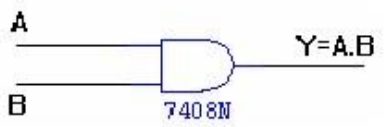
1. To verify the truth tables of OR, AND, NOR, NAND, X-OR, X-NOR gates

b. To study IC 7400, 7402, 7404, 7408, 7432, 7486, 74266

c. To implement and verify NAND and NOR as Universal gates

<u>AND GATE:</u>

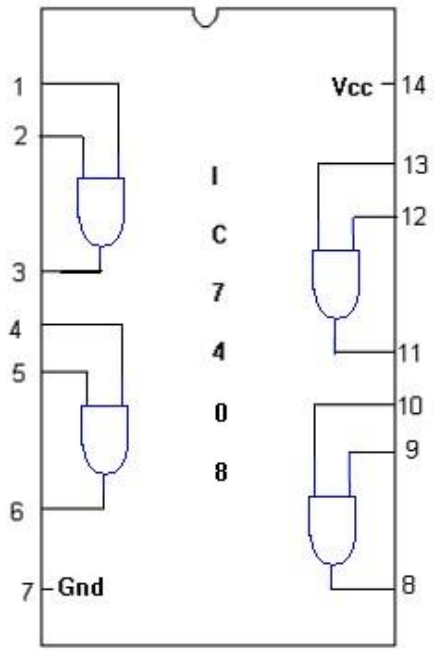
<u>SYMBOL:</u>



TRUTH TABLE

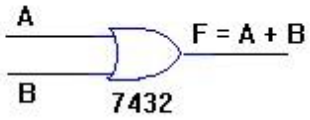
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

<u>PIN DIAGRAM:</u>

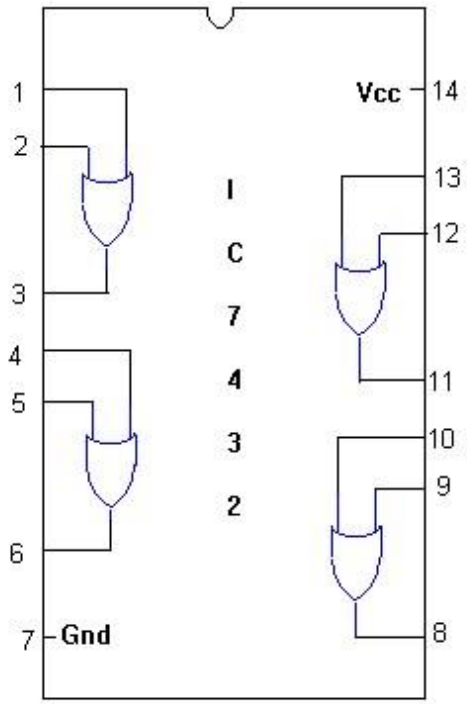


OR GATE:

SYMBOL :



PIN DIAGRAM :

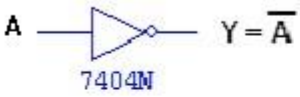


TRUTH TABLE

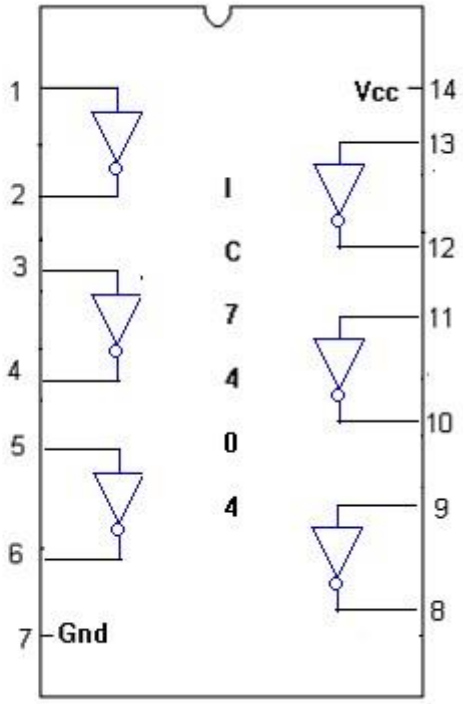
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

NOT GATE:

SYMBOL:



PIN DIAGRAM

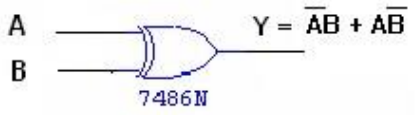


TRUTH TABLE :

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

X-OR GATE :

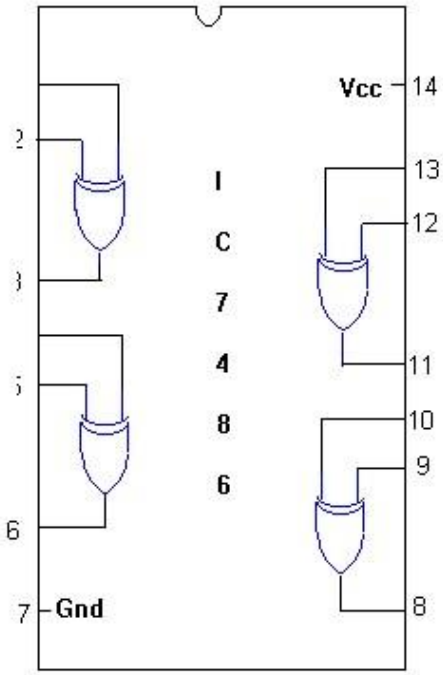
SYMBOL :



TRUTH TABLE :

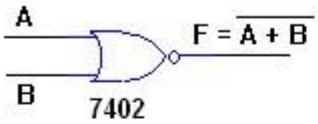
A	B	$\overline{A}B + A\overline{B}$
0	0	0
0	1	1
1	0	1
1	1	0

PIN DIAGRAM :



NOR GATE:

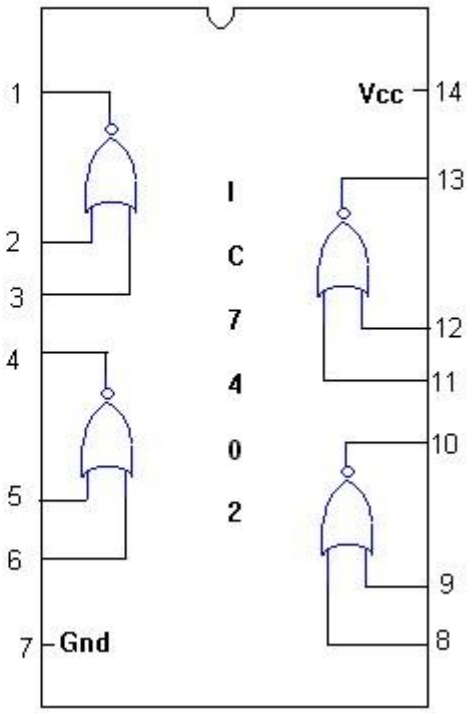
SYMBOL :



TRUTH TABLE

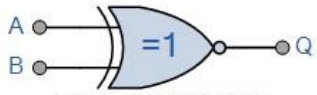
A	B	$\overline{A+B}$
0	0	1
0	1	1
1	0	1
1	1	0

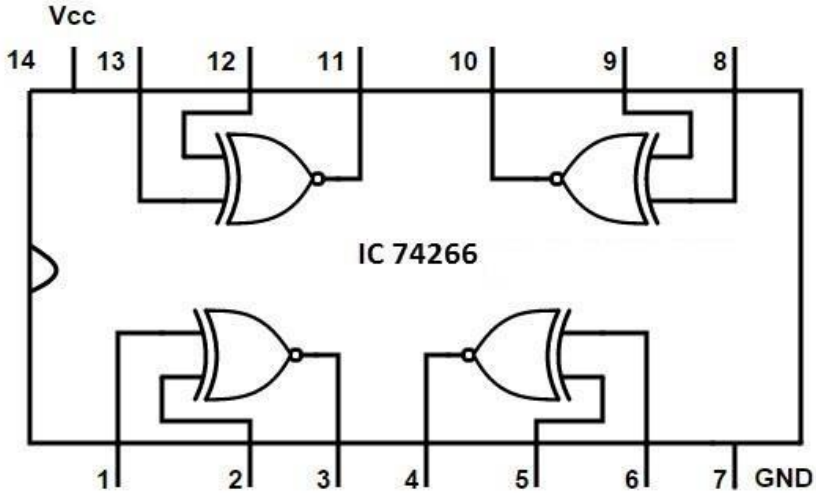
PIN DIAGRAM :



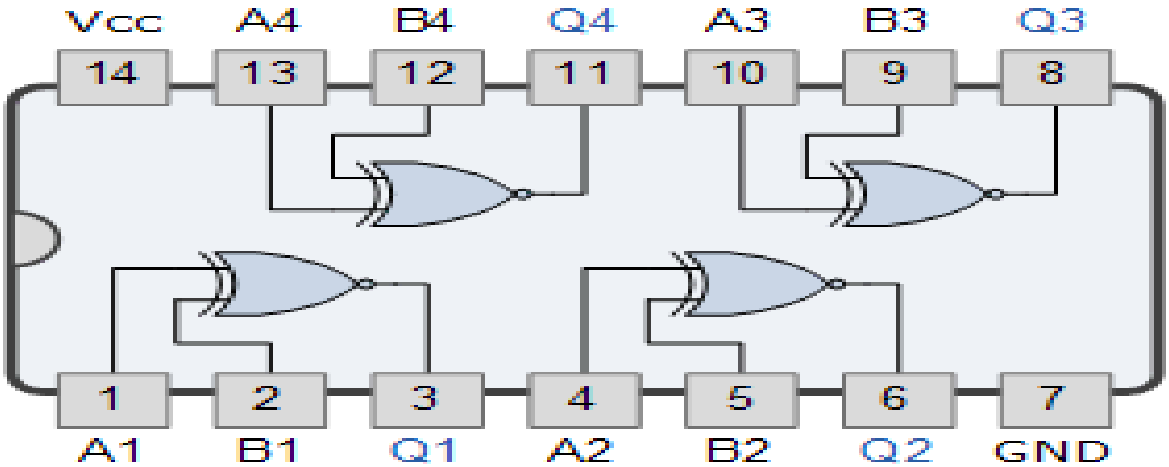
IC 74266

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

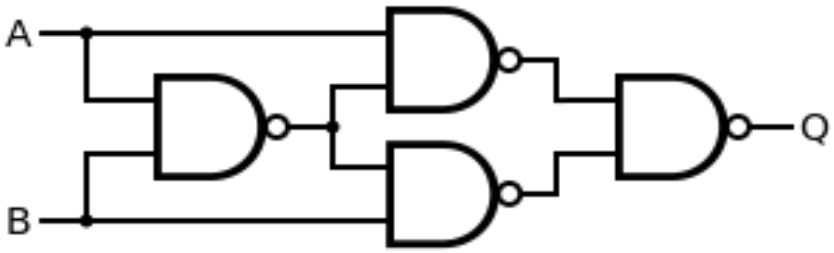
Symbol	Truth Table		
 2-input Ex-NOR Gate	B	A	Q
	0	0	1
	0	1	0
	1	0	0
	1	1	1
Boolean Expression Q = $\overline{A \oplus B}$	Read if A AND B the SAME gives Q		



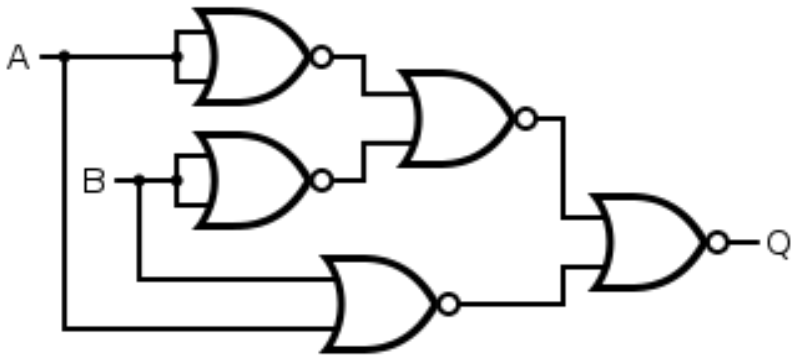
74266 Quad 2-input Ex-NOR Gate



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**.**

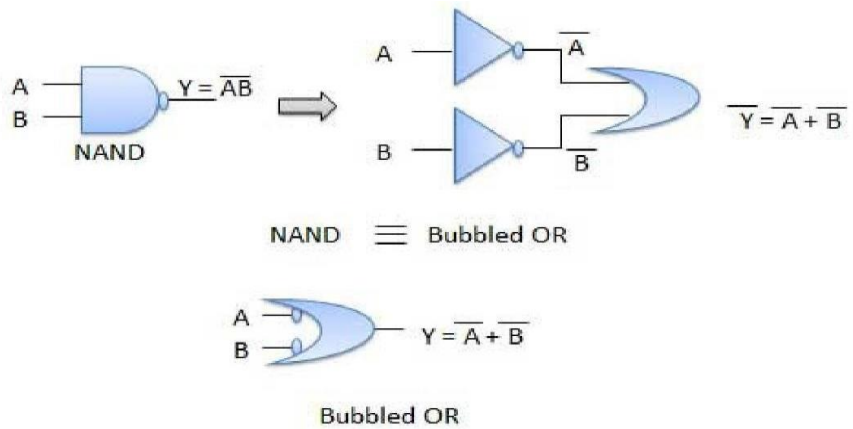
Theorem 1:

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

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

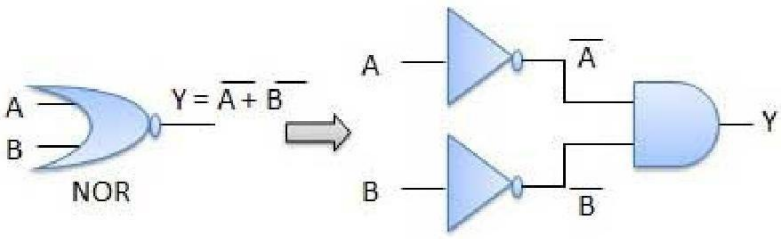
Theorem 2:

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

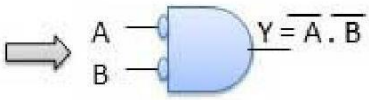
NOR = Bubbled AND

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 ≡ Bubbled AND



Bubbled AND

A	B	$\overline{A+B}$	$\overline{A}$	$\overline{B}$	$\overline{A}.\overline{B}$
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, fW,X,Y,ZW,X,Y,Z= WX'Y' + WY + W'YZ' using K-map.

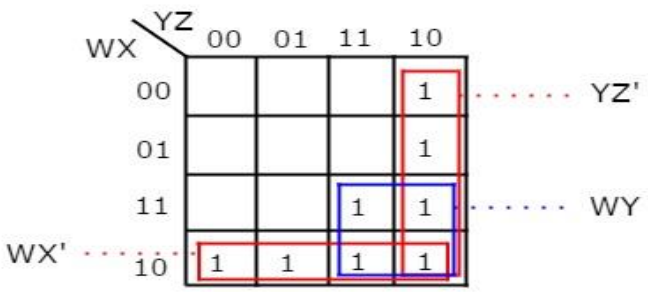
The given Boolean function is in sum of products form. It is having 4 variables W, X, Y & Z. So, we require 4 variable K-map. The 4 variable K-map with ones corresponding to the given product terms is shown in the following figure.

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.



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.

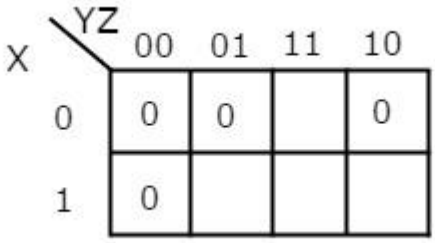
- Select the respective K-map based on the number of variables present in the Boolean function.
- If the Boolean function is given as product of Max terms form, then place the zeroes at respective Max term cells in the K-map. If the Boolean function is given as product of sums form, then place the zeroes 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'.

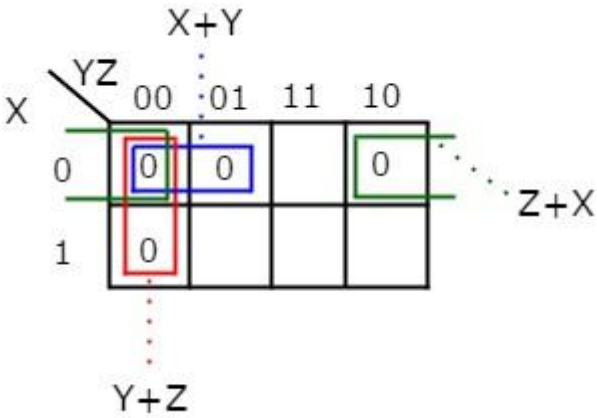
Example

Let us simplify the following Boolean function, f(X,Y,Z)=∏M(0,1,2,4)f(X,Y,Z)=∏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.



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.



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+YX+Y.Y+ZY+Z.Z+XZ+X

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.

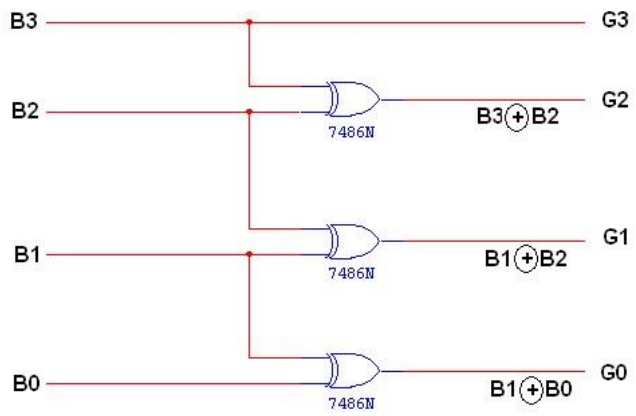
PRACTICAL-4

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:

<u>BINARY TO GRAY CODE CONVERTOR</u>



K-Map for G_3:

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

$G_3 = B_3$

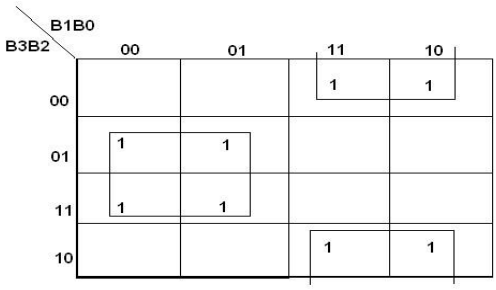
K-Map for G_2:

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

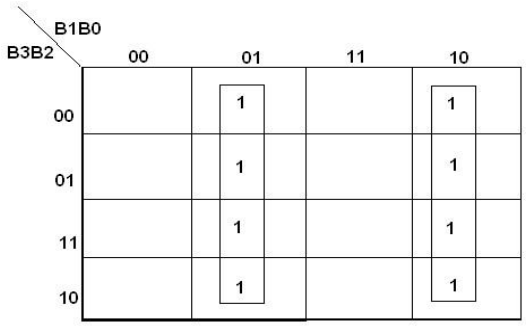
$G2 = B3 \oplus B2$

K-Map for G_1:

K-Map for G_0:



G1 = B1⊕B2



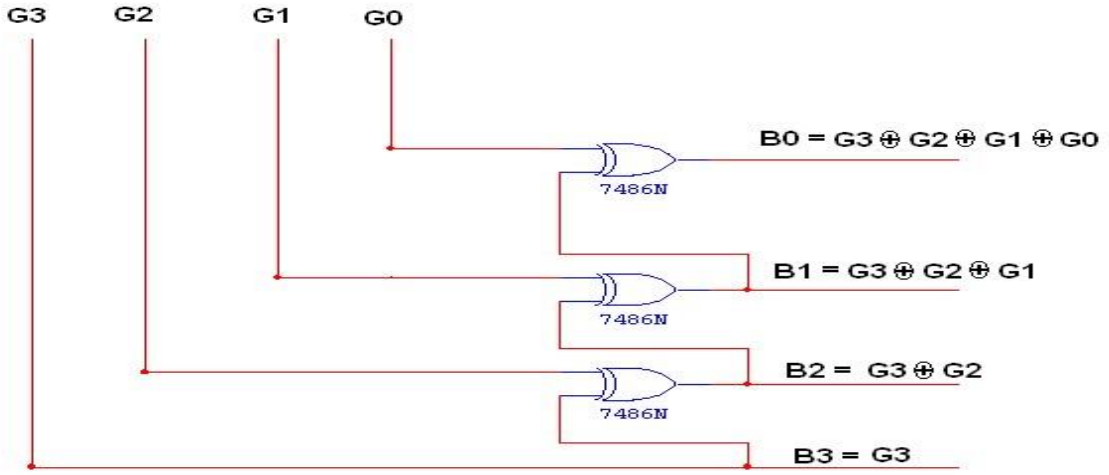
G0 = B1⊕B0

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:

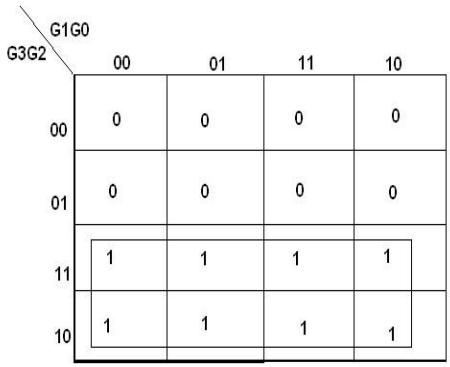
<u>GRAY CODE TO BINARY CONVERTOR</u>

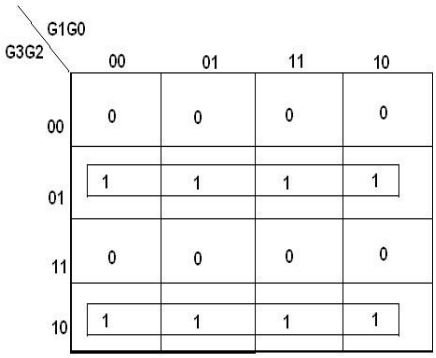


TRUTH TABLE:

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 B_3:

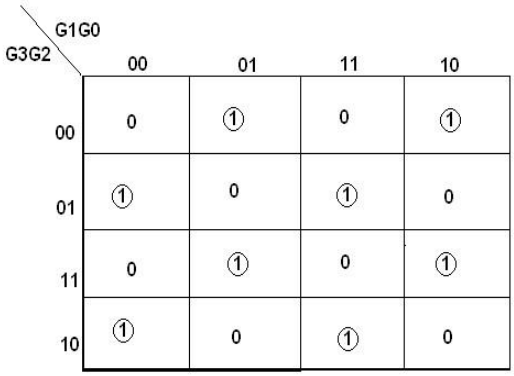


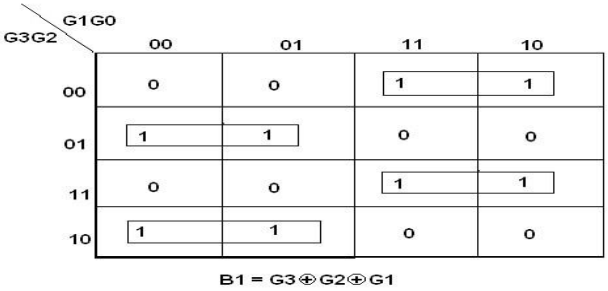


B3 = G3

B2 = G3⊕G2

K-Map for B_1:

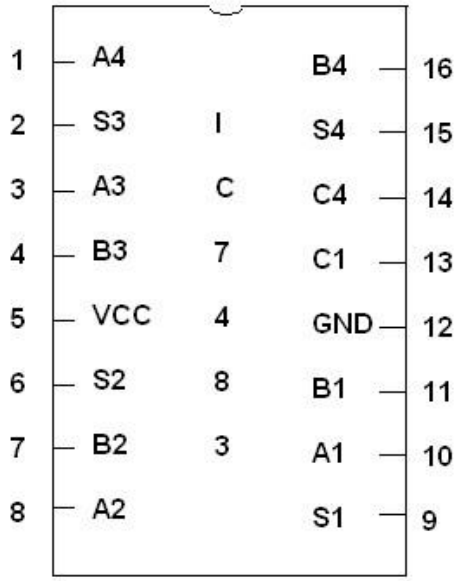




B1 = G3⊕G2⊕G1

B0 = G3⊕G2⊕G1⊕G0

PIN DIAGRAM FOR IC 7483:



PRACTICAL-5

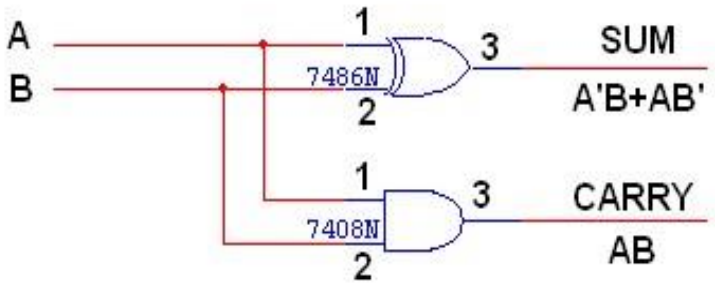
Aim: Implement Adder and Subtractor circuits

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:

<u>HALF ADDER</u>



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:

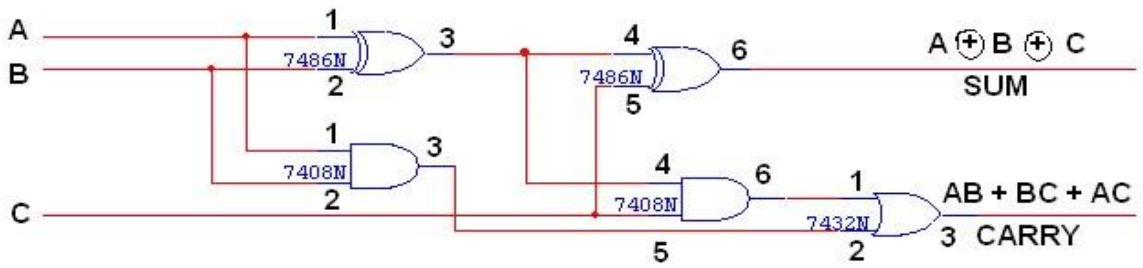
A \ B	00	01
00		1
01	1	

K-Map for CARRY:

A \ B	00	01
00		
01		1

SUM = A'B + AB' **CARRY = AB LOGIC DIAGRAM:**

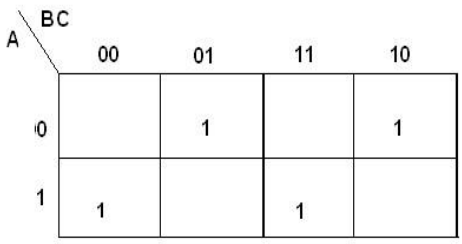
FULL ADDER USING TWO HALF ADDER

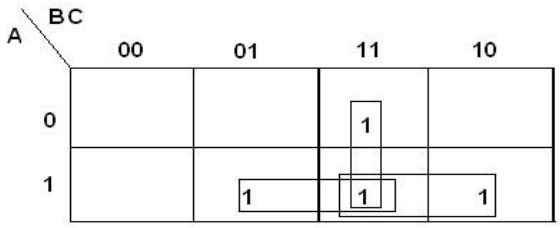


TRUTH TABLE:

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

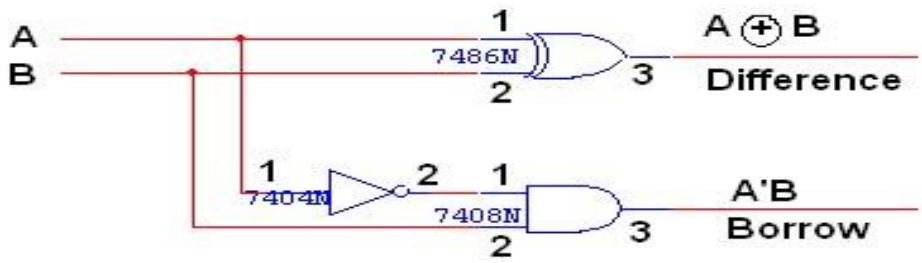
K-Map for SUM: **K-Map for CARRY:**





SUM = A’B’C + A’BC’ + ABC’ + ABC 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	

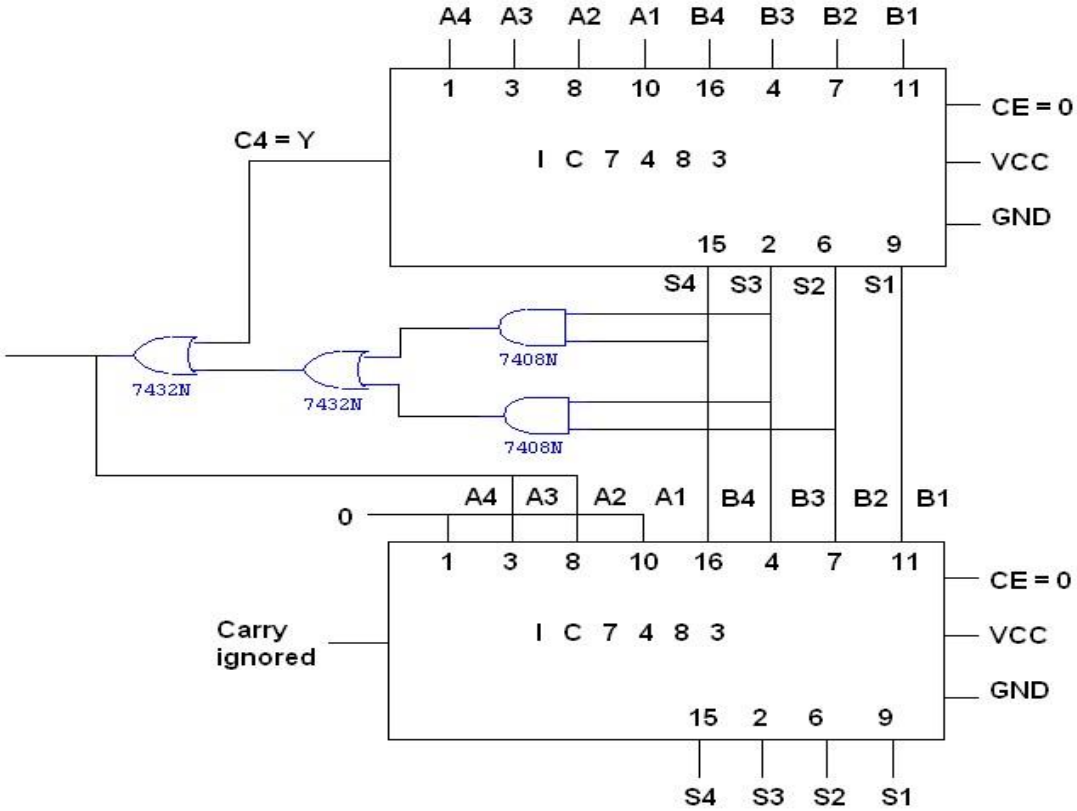
K-Map for BORROW:

A \ B	00	01
00		1
01		

DIFFERENCE = A'B + AB'

BORROW = A'B

BCD ADDER



TRUTH TABLE:

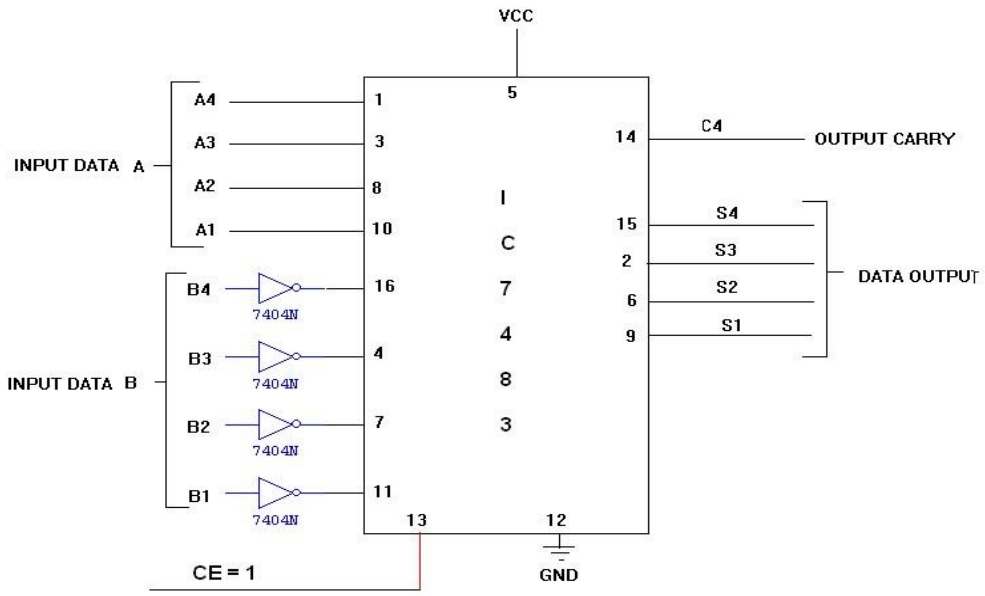
BCD SUM				CARRY
S4	**S3**	**S2**	**S1**	**C**
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

K MAP

S3 S4 \ S1 S2	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



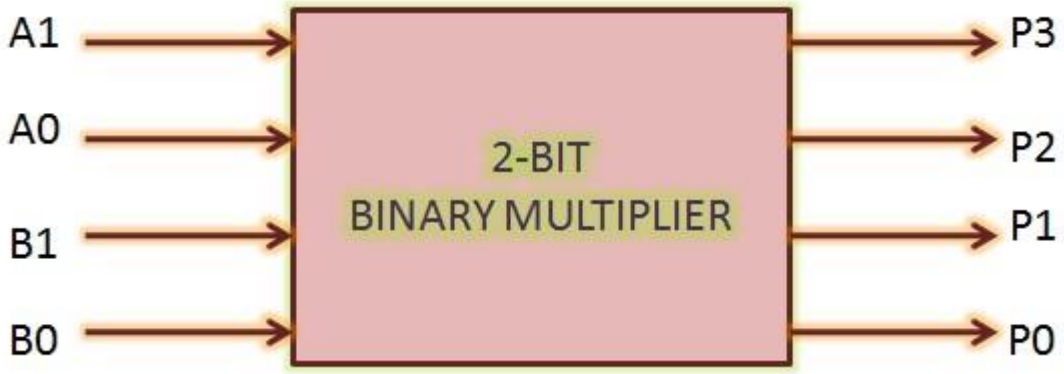
Input Data A				Input Data B				Addition					Subtraction				
A4	A3	A2	A1	B4	B3	B2	B1	C	S4	S3	S2	S1	B	D4	D3	D2	D1
1	0	0	0	0	0	1	0	0	1	0	1	0	1	0	1	1	0
1	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0
0	0	1	0	1	0	0	0	0	1	0	1	0	0	1	0	1	0
0	0	0	1	0	1	1	1	0	1	0	0	0	0	1	0	1	0
1	0	1	0	1	0	1	1	1	0	0	1	0	0	1	1	1	1
1	1	1	0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

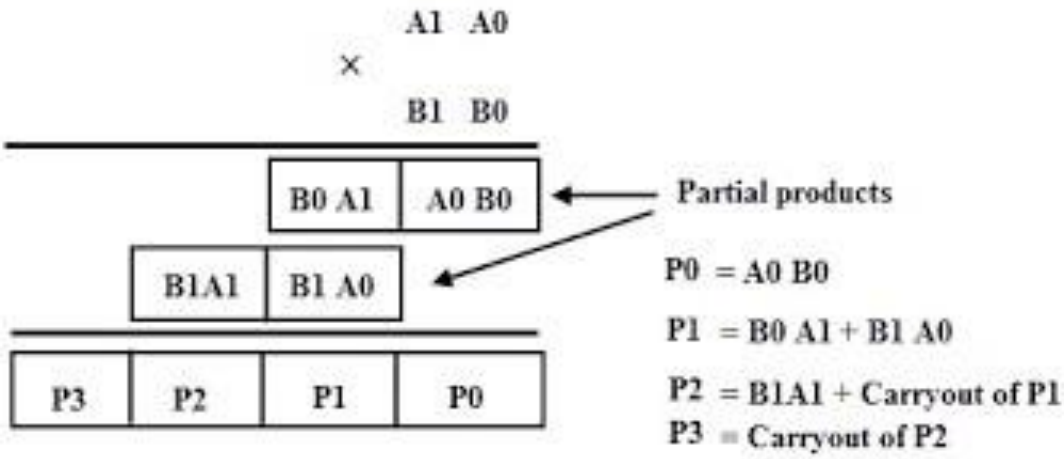
PRACTICAL-6

Aim: Design and implement Arithmetic circuits

a. **Design and implement 2-by-2 bit multiplier.**

Design and implement a 2-bit comparator





$$\begin{array}{ccc} & A_1 & A_0 \\ X & B_1 & B_0 \\ \hline & B_1A_1 & B_0A_0 \\ B_1A_1 & B_1A_0 & x \end{array}$$

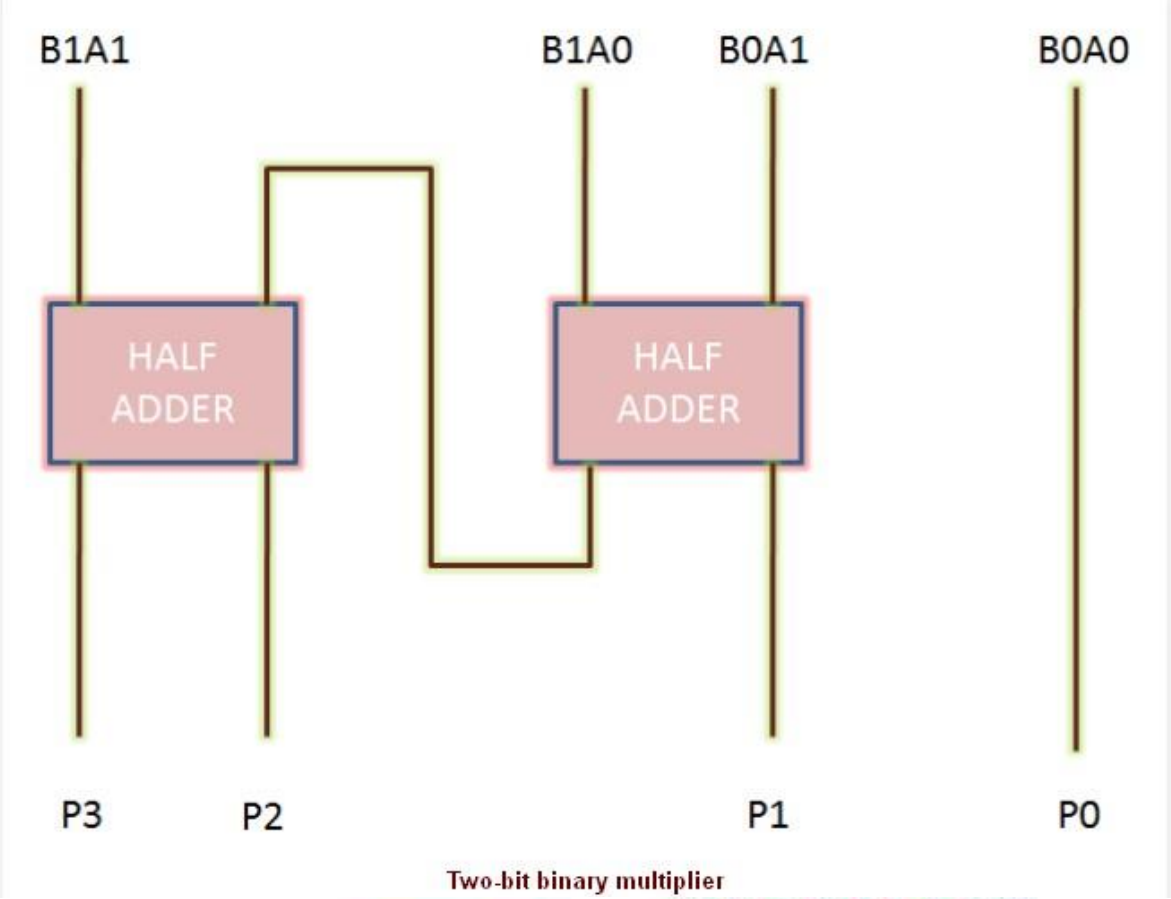
we get the partial products as:

P0 = A0*B0

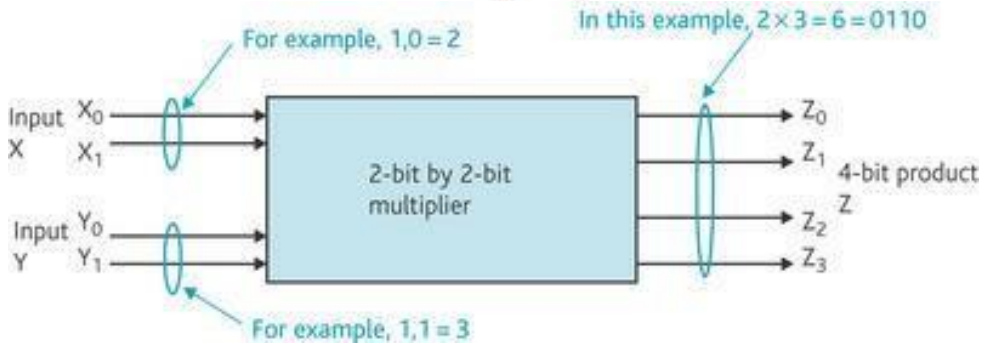
P1 = A0*B1 xor A1 * B0 ; carry generated here goes to next stage

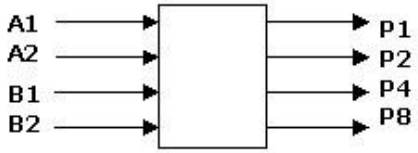
P2 = A1*B1 xor (A0*B1) * (A1*B0)

P3 = A1*B1 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

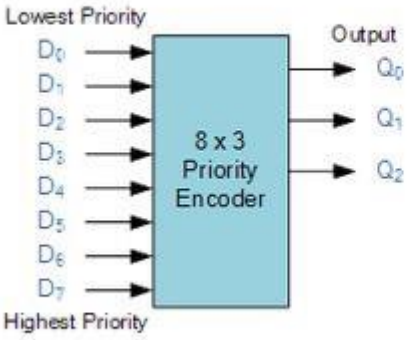
Aim: Implement Encoders and Decoders

a. Design and implement 8: 3 encoder

b. Design and implement 3:8 decoder

Design and implement 8:3 encoder :

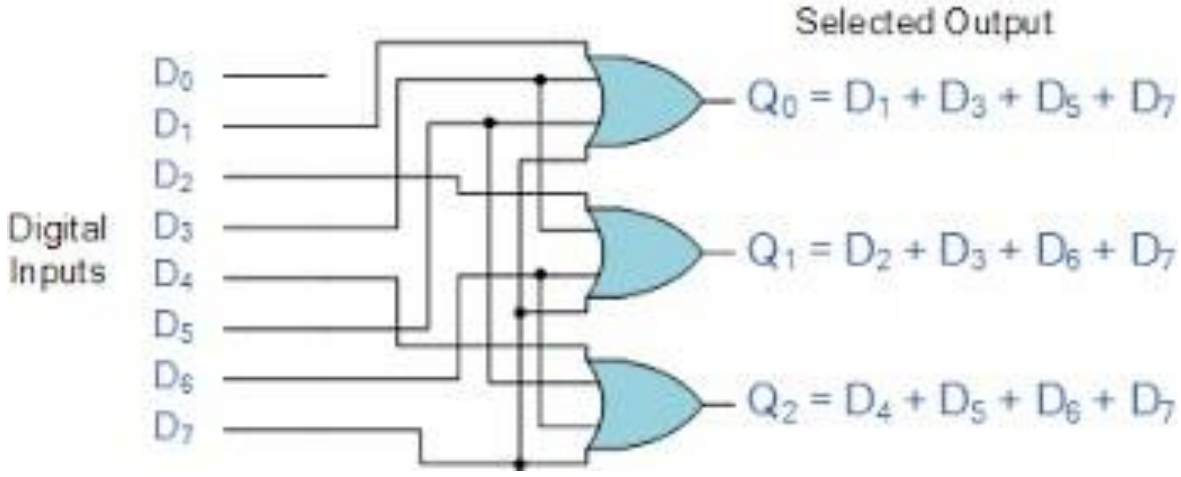
8-to-3 Bit Priority Encoder



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

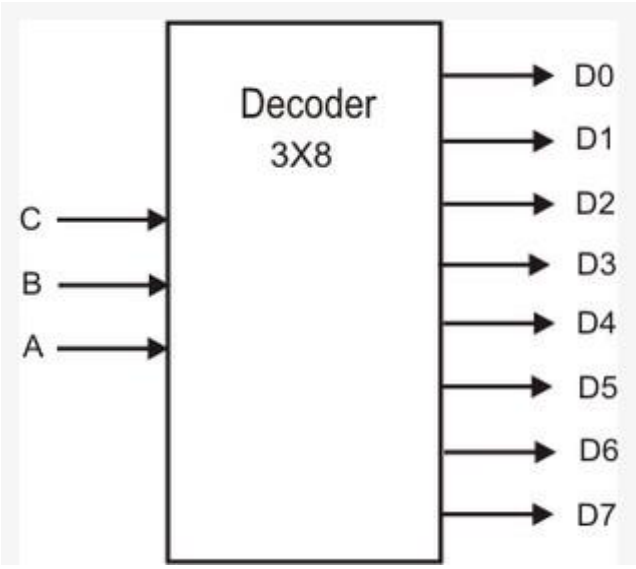
X = dont care

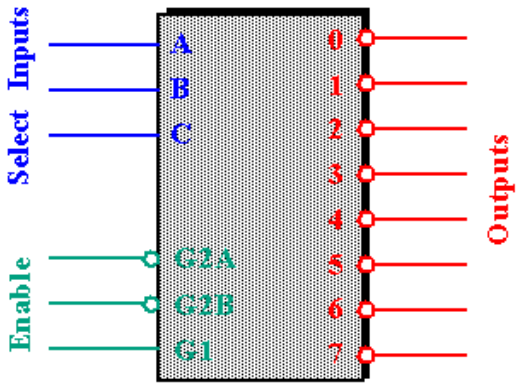
Digital Inputs								Binary Output		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	1	X	X	0	1	0
0	0	0	0	1	X	X	X	0	1	1
0	0	0	1	X	X	X	X	1	0	0
0	0	1	X	X	X	X	X	1	0	1
0	1	X	X	X	X	X	X	1	1	0
1	X	X	X	X	X	X	X	1	1	1



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

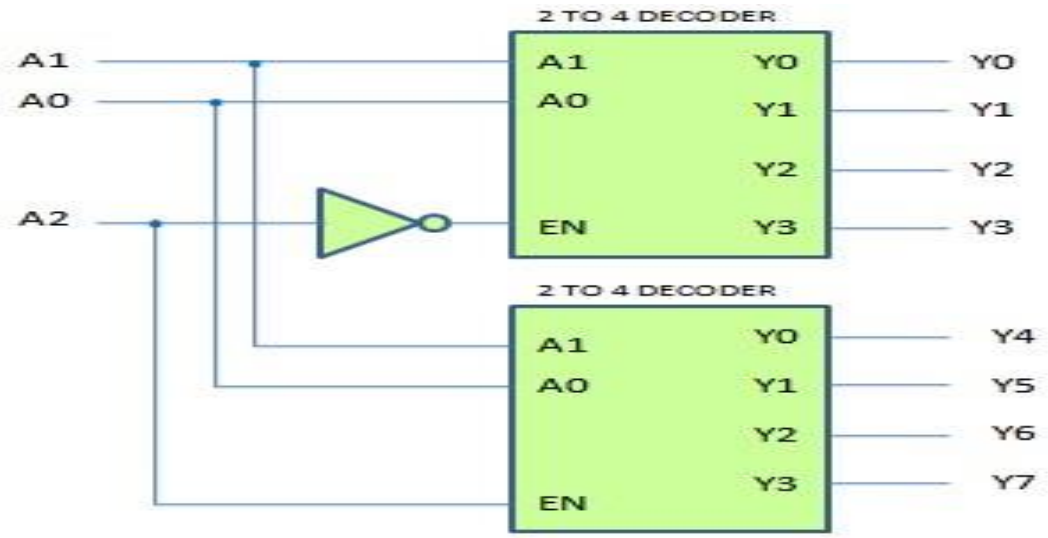
Design and implement 3:8 decoder :



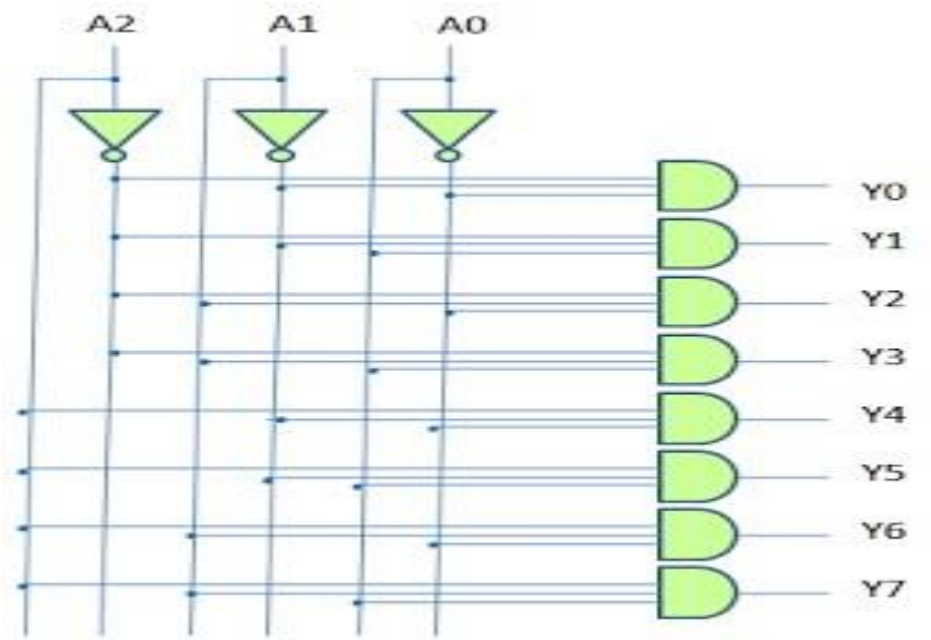


Inputs						Output							
Enable			Select										
G2A	G2B	G1	C	B	A	0	1	2	3	4	5	6	7
1	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	0	X	X	X	1	1	1	1	1	1	1	1
0	0	1	0	0	0	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	0	1	1	1	1	1	1
0	0	1	0	1	0	1	1	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1	0	1	1	1	1
0	0	1	1	0	0	1	1	1	1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

<u>3 to 8 decoder using 2 to 4 decoders:</u>



3 to 8 decoder using gates:



PRACTICAL-8

Aim: Multiplexers and Demultiplexers

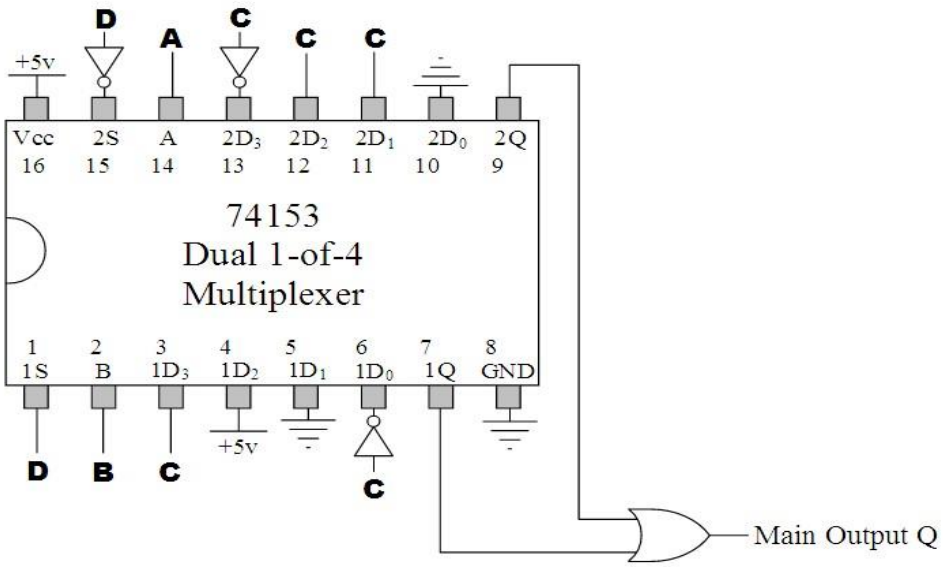
a. Design and Implement 4:1 multiplexer

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

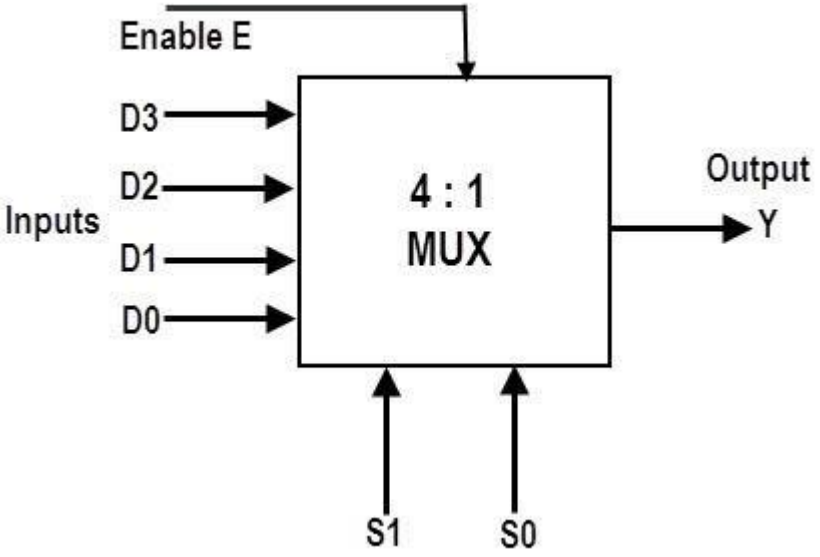
Design and implement 4:1 multiplexer. Study of IC 74153



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	$2D_3$

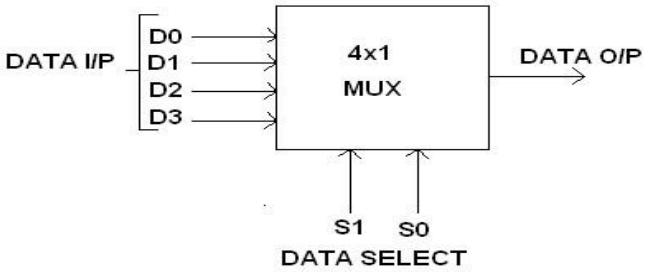
Design and implement 4:1 multiplexer.

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:

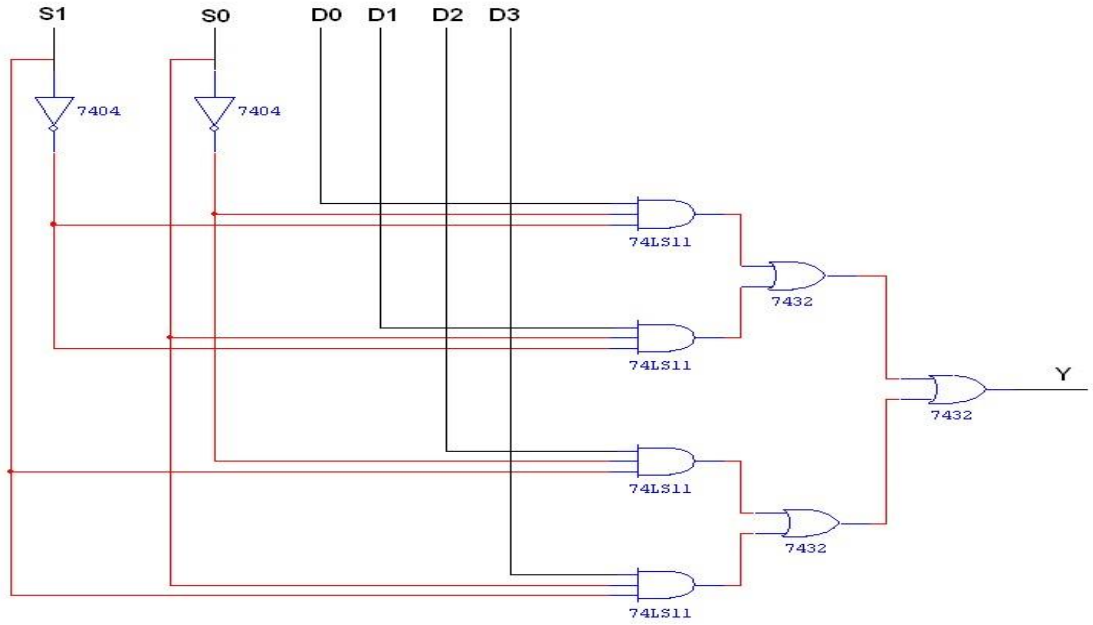


FUNCTION TABLE:

S1	S0	INPUTS Y
0	0	D0 → D0 S1' S0'
0	1	D1 → D1 S1' S0
1	0	D2 → D2 S1 S0'
1	1	D3 → D3 S1 S0

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

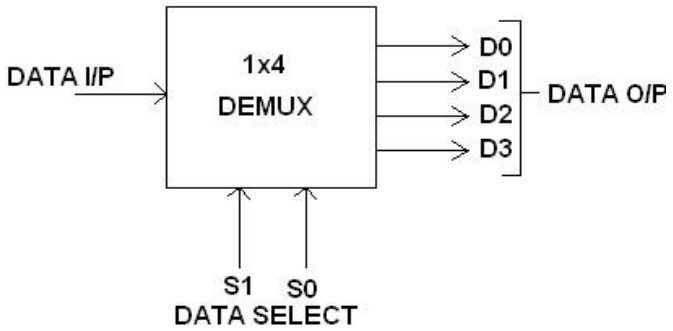
CIRCUIT DIAGRAM FOR MULTIPLEXER:



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:

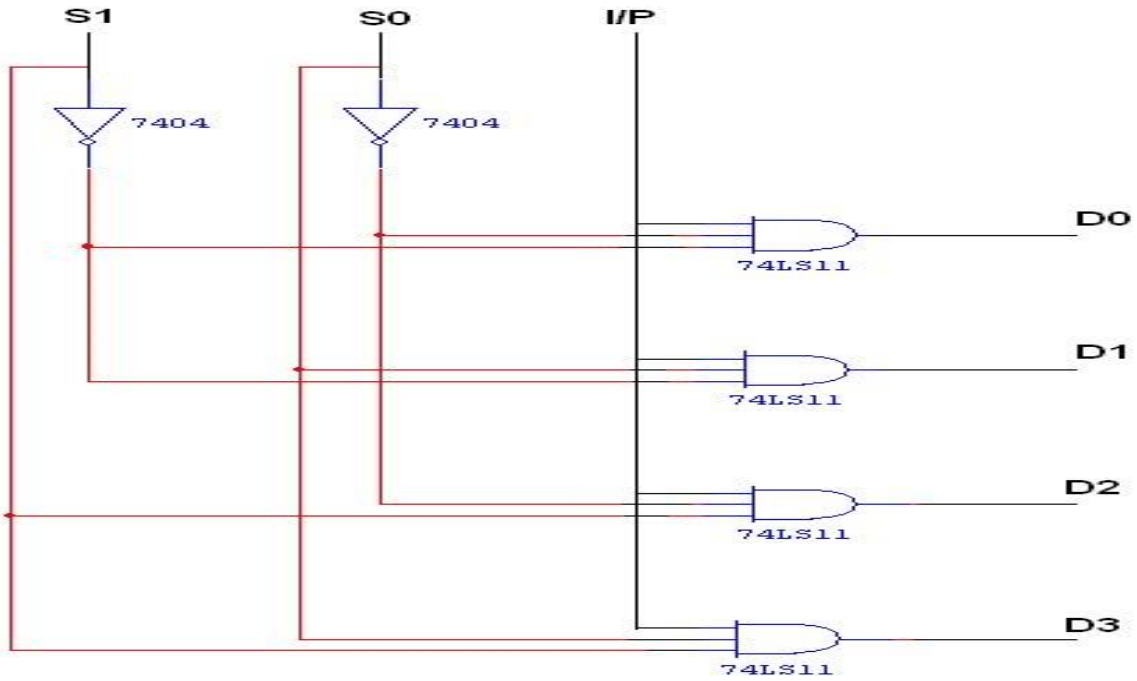
S1	S0	INPUT
0	0	X → D0 = X S1' S0'
0	1	X → D1 = X S1' S0
1	0	X → D2 = X S1 S0'
1	1	X → D3 = X S1 S0

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

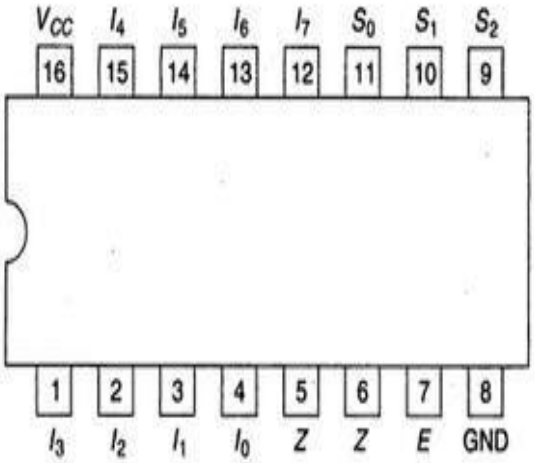
LOGIC DIAGRAM FOR DEMULTIPLEXER:

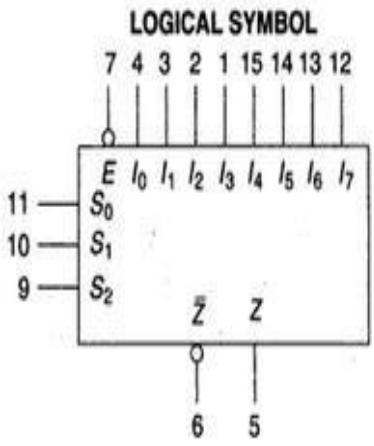


TRUTH TABLE:

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

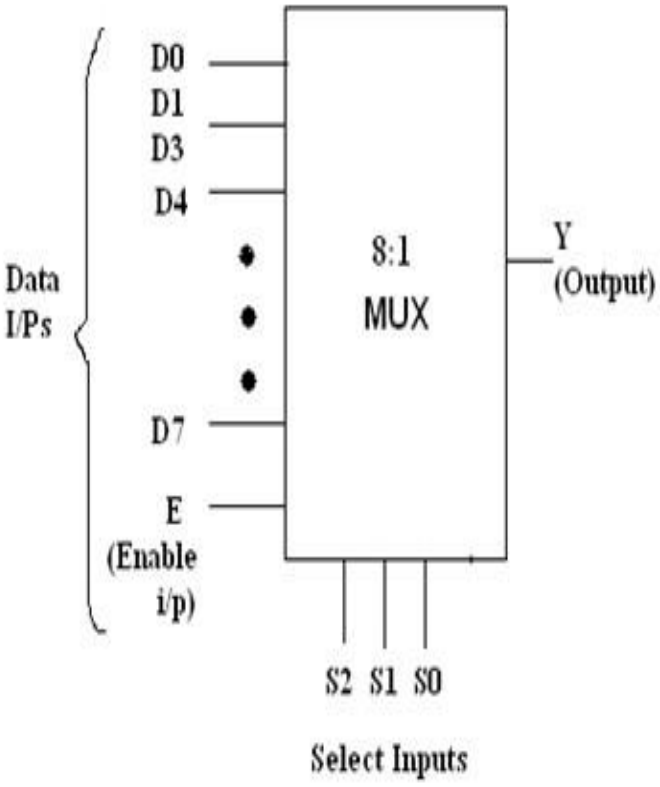




(b)

PIN NAMES

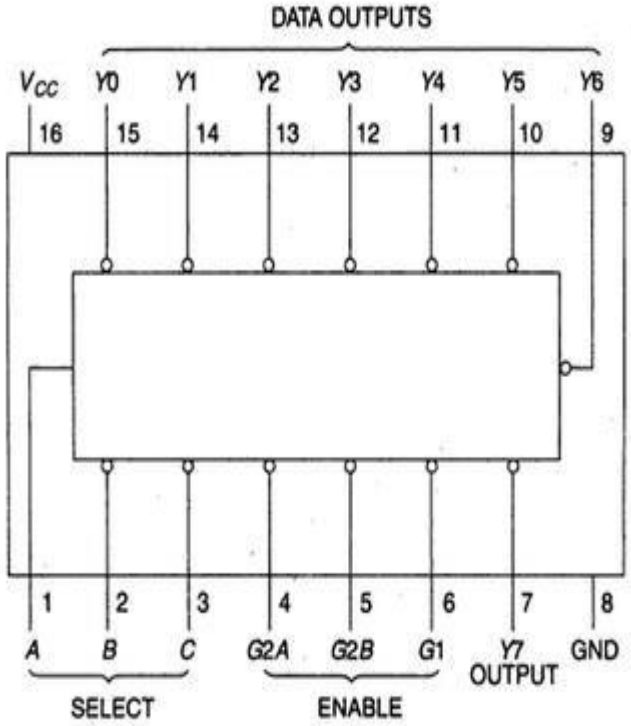
$S_0 - S_2$	Select inputs
E	Enable (Active LOW) inputs
$I_0 - I_7$	Multiplexer inputs
Z	Multiplexer outputs (note *b*)
$\bar{Z}$	Complementary multiplexer output



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	YO	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

PRACTICAL-9

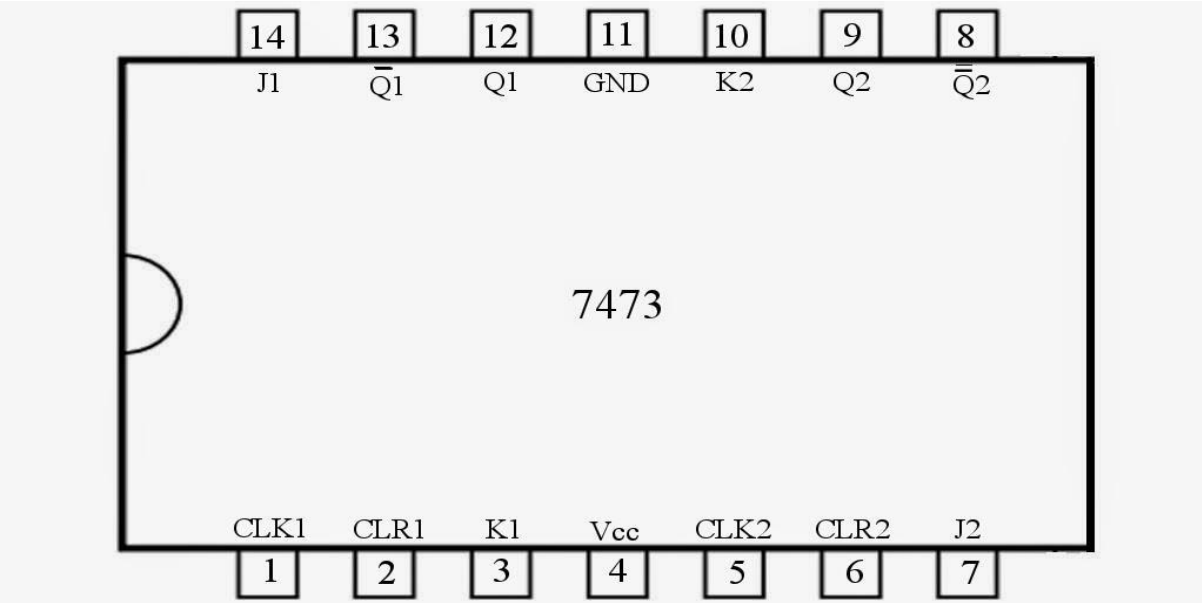
Aim: Study of Flipflops and Counters

a. Study of IC's 7473, 7474, and 7476

b. Design a 3-bit ripple/ synchronous counter using IC 7473 and required gates

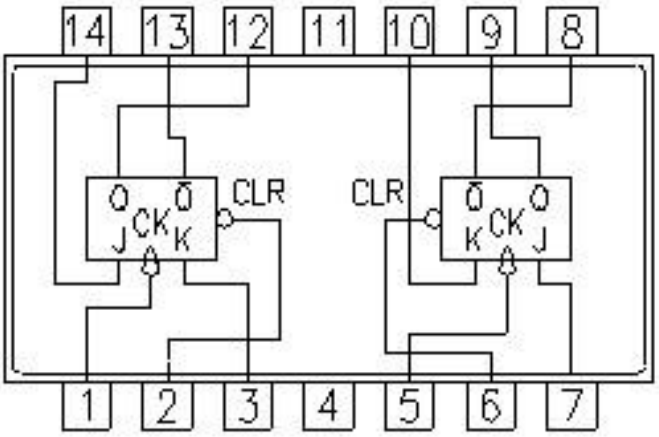
Study of IC 7473 :

Each 7473 has two master-slave J K flip-flips. Half portion of IC, above VCC and ground constitutes the first flip-flop and the half portion below VCC and Ground constitutes the second master-slave flip-flop.



TRUTH TABLE:

CLR	CLK	J	K	Q	$\overline{Q}$
L	X	X	X	L	H
H	↓	L	H	L	H
H	↓	H	L	H	L
H	↓	L	L	Retains previous state	
H	↓	H	H	Toggle	

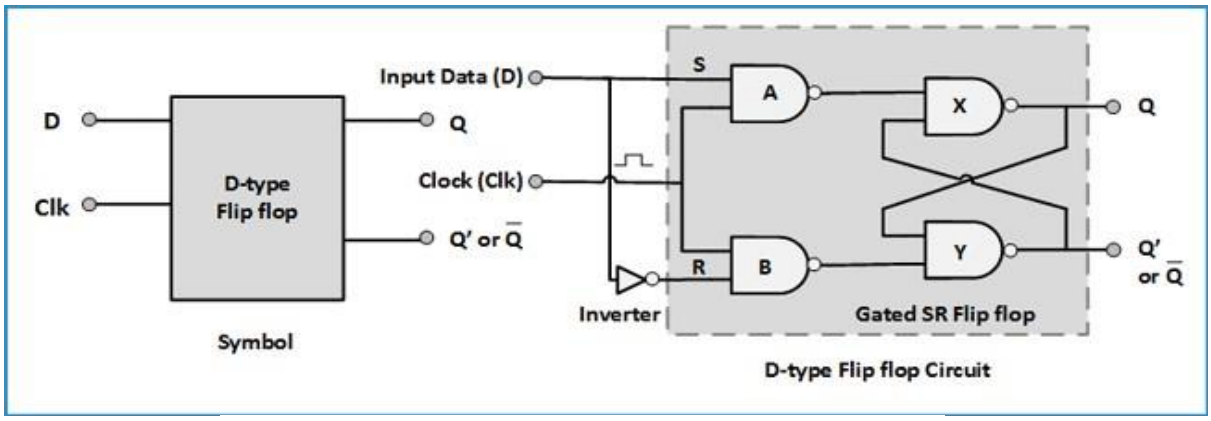


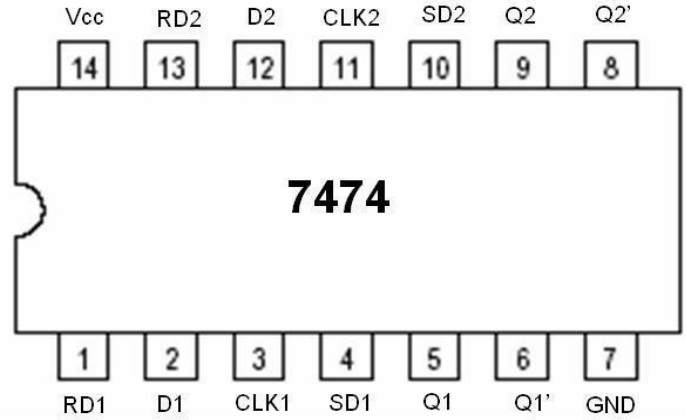
7473
Dual J-K M/S Flip-Flop
with Clear

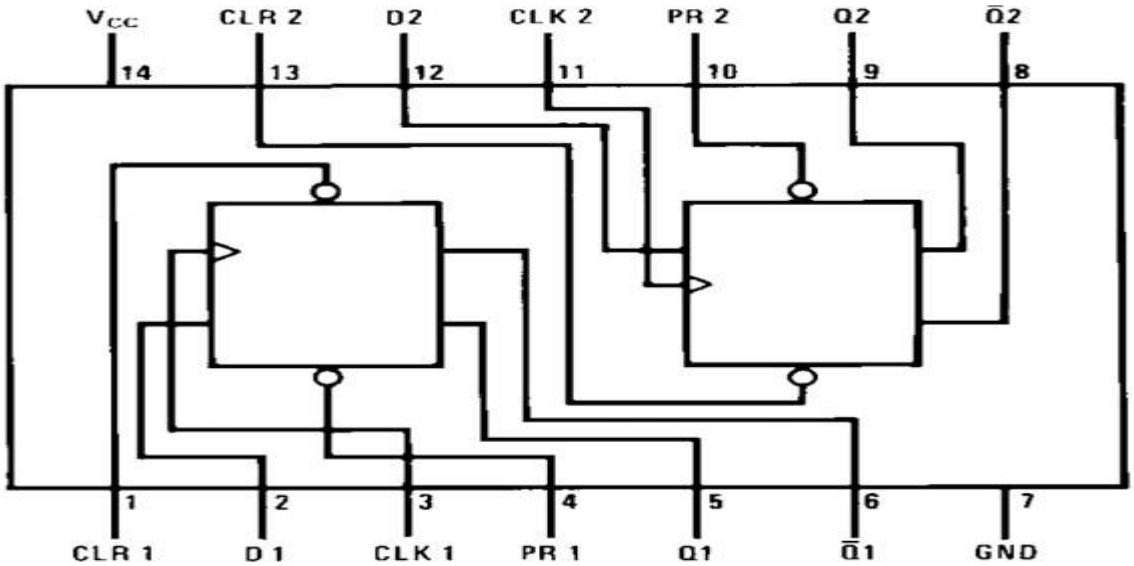
Pin Number	Description
1	Clock 1
2	Clear 1
3	K1 Input
4	Vcc - Positive Supply
5	Clock 2
6	Clear 2
7	J2 Input
8	Complement Q2 Output
9	Q2 Output
10	K2 Input
11	Ground
12	Q1 Output
13	Complement Q1 Output
14	J1 Input

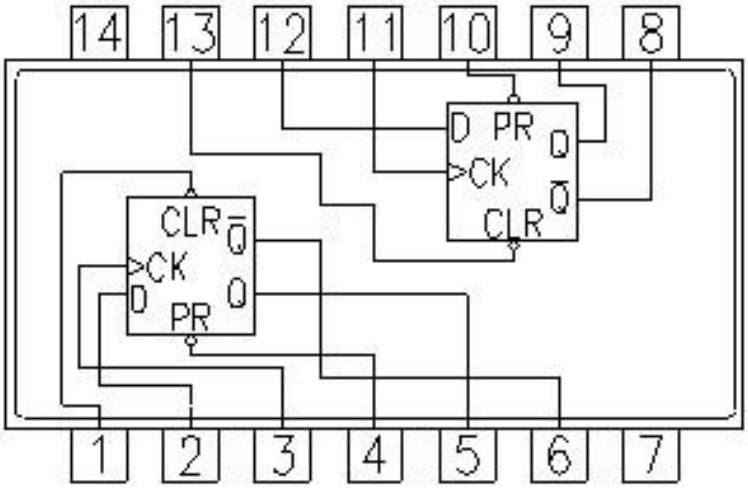
Study of IC 7474.

IC DM74S74N is the Dual D-type Flip-flop IC, in which there are two D-type Flip-flops, which can be either used individually or as a master-slave toggle combination Pins for first D flip-flop are the left side and for second flip flop are at right side. Also there are PRE and CLR pins for both the D-type Flip-flops which are active-low pins. These pin used to SET or RESET the D-type Flip-flop respectively, regardless of INPUT D and Clock. We have connected both to Vcc to make them inactive.









7474

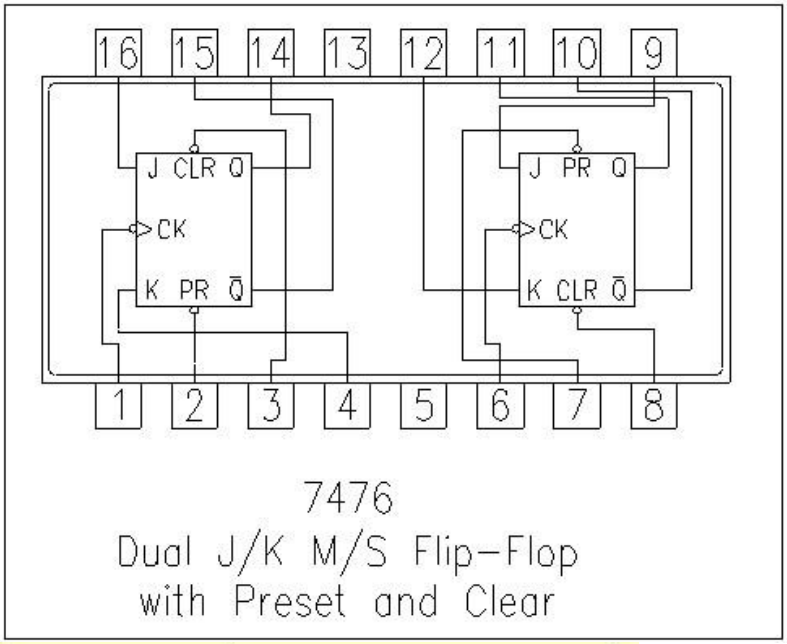
Dual D Flip-Flop with Preset and Clear

Pin Number	Description
1	Clear 1 Input
2	D1 Input
3	Clock 1 Input
4	Preset 1 Input
5	Q1 Output
6	Complement Q1 Output
7	Ground
8	Complement Q2 Output
9	Q2 Output
10	Preset 2 Input
11	Clock 2 Input
12	D2 Input
13	Clear 2 Input
14	Positive Supply

Study of IC 7476

7476 Dual JK Flip-Flop with Preset and Clear

Two JK Type Master/Slave Flip-Flops with Preset and Clear



Pin Number	Description
1	Clock 1 Input
2	Preset 1 Input
3	Clear 1 Input
4	J1 Input
5	Vcc - Positive Supply
6	Clock 2 Input
7	Preset 2 Input
8	Clear 2 Input
9	J2 Input
10	Complement Q2 Output
11	Q2 Output
12	K2 Input
13	Ground
14	Complement Q1 Output
15	Q1 Output
16	K1 Input

Aim

To design and verify the truth table for 3-bit synchronous up/down counter.

Hardware Requirement

Equipment : Equipment : Bread Board, Power Supply, Resistors, LEDs or Digital IC Trainer Kit

Discrete Components :

IC 7473 Dual JK Flip Flop 74LS08 Quad 2 input AND gate 74LS32Quad 2 input OR gate 74LS04 Hex 1 input NOT gate

Theory

Circuits for counting events are frequently used in computers and other digital systems. Since a counter circuit must remember its past states, it has to possess memory. The number of flip flops used and how they are connected determine the number of states and the sequence of the states that the counter goes through in each complete cycle.

Counters can be classified into two broad categories according to the way they are clocked:

a. Asynchronous (Ripple) Counters - the first flip-flop is clocked by the external clock pulse, and then each successive flip -flop is clocked by the Q or Q' output of the previous flip -flop.
b. Synchronous Counters - all memory elements are simultaneously triggered by the same clock.

Synchronous Counters

In *synchronous counters*, the clock inputs of all the flip-flops are connected together and are triggered by the input pulses. Thus, all the flip-flops change state simultaneously (in parallel). The circuit below is a 3-bit synchronous counter. The J and K inputs of FF0 are connected to HIGH. FF1 has its J and K inputs connected to the output of FF0, and the J and K inputs of FF2 are connected to the output of an AND gate that is fed by the outputs of FF0 and FF1. After the 3rd clock pulse both outputs of FF0 and FF1 are HIGH. The positive edge of the 4th clock pulse will cause FF2 to change its state due to the AND gate.

\

To design and verify the timing diagram of 3 bit Ripple Counter :

Apparatus Required

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

Theory

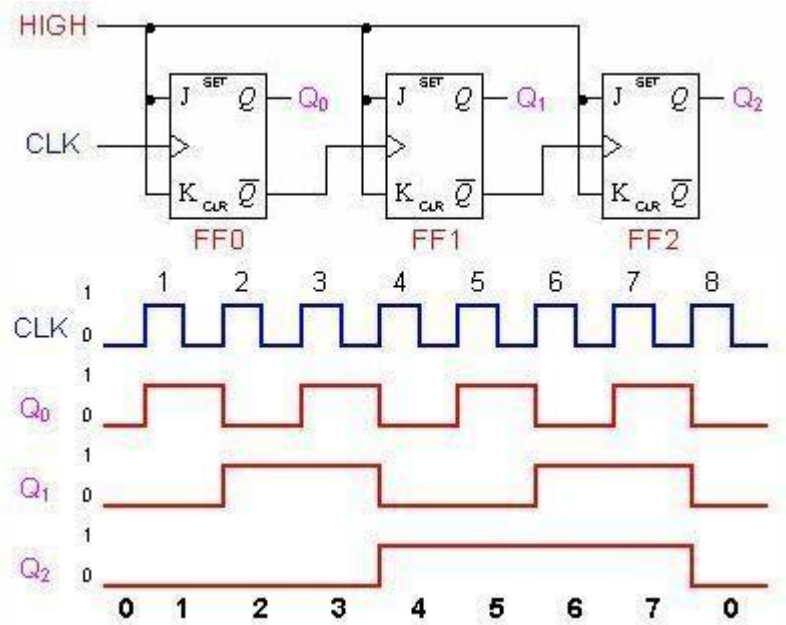
Asynchronous Counter is sequential circuit that is used to count the number of clock input signal. The output of one flip flop is given as a clock input to another flip-flop, so it is called as Serial Counter.

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.

PRACTICAL-10

Aim: Design of Shift Registers

a. Design of Shift registers using IC 7474

b. Implementation of digits using seven segment displays

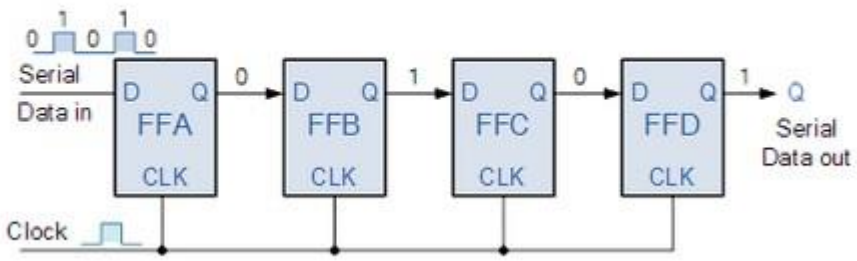
Theory:

The Shift Register

- **The Shift Register is another type of sequential logic circuit that can be used for the storage or the transfer of data in the form of binary numbers.**

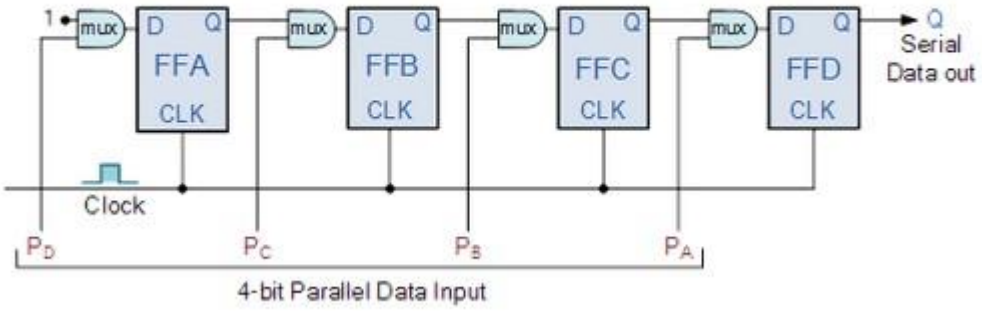
This sequential device loads the data present on its inputs and then moves or "shifts" it to its output once every clock cycle, hence the name "shift register".

4-bit Serial-in to Serial-out Shift Register



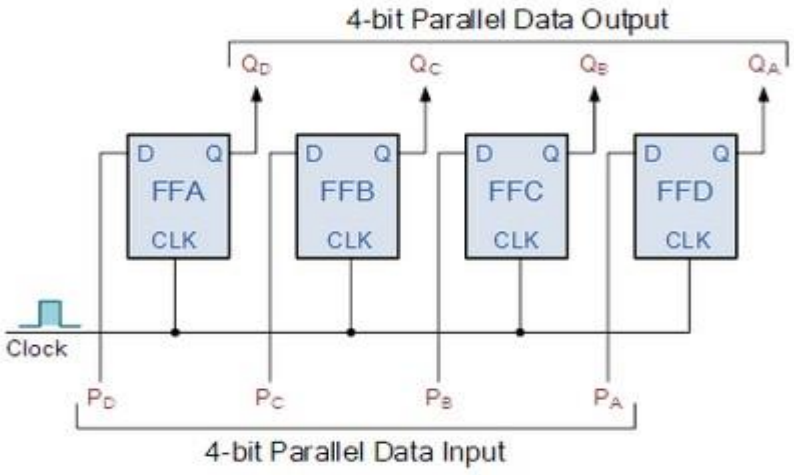
CLK	Serial in	Serial out
1	1	0
2	0	0
3	0	0
4	1	1
5	X	0
6	X	0
7	X	1

4-bit Parallel-in to Serial-out Shift Register



CLK	DATA INPUT				OUTPUT			
	D_A	D_B	D_C	D_D	Q_A	Q_B	Q_C	Q_D
1	1	0	0	1	1	0	0	1
2	1	0	1	0	1	0	1	0

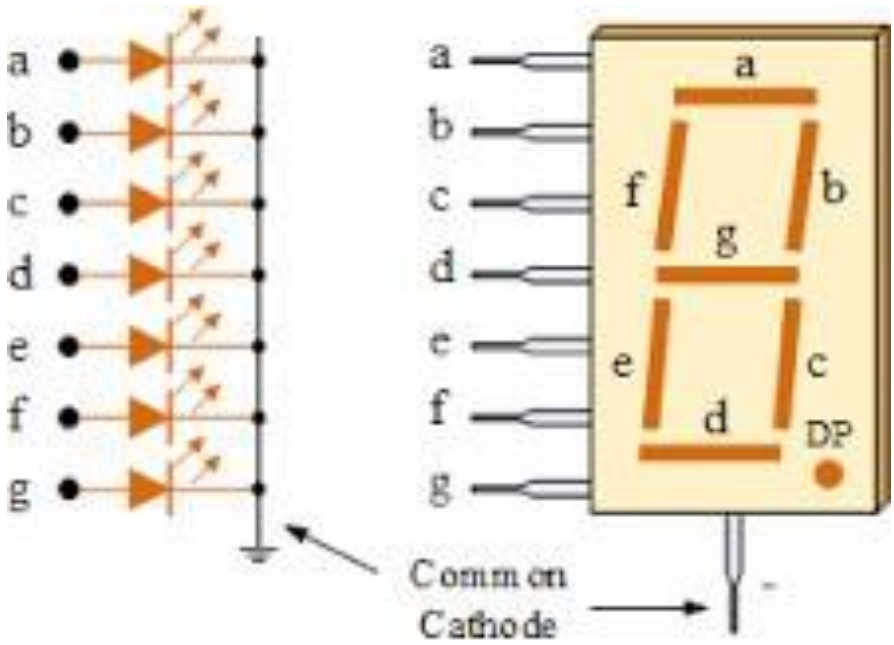
4-bit Parallel-in to Parallel-out Shift Register



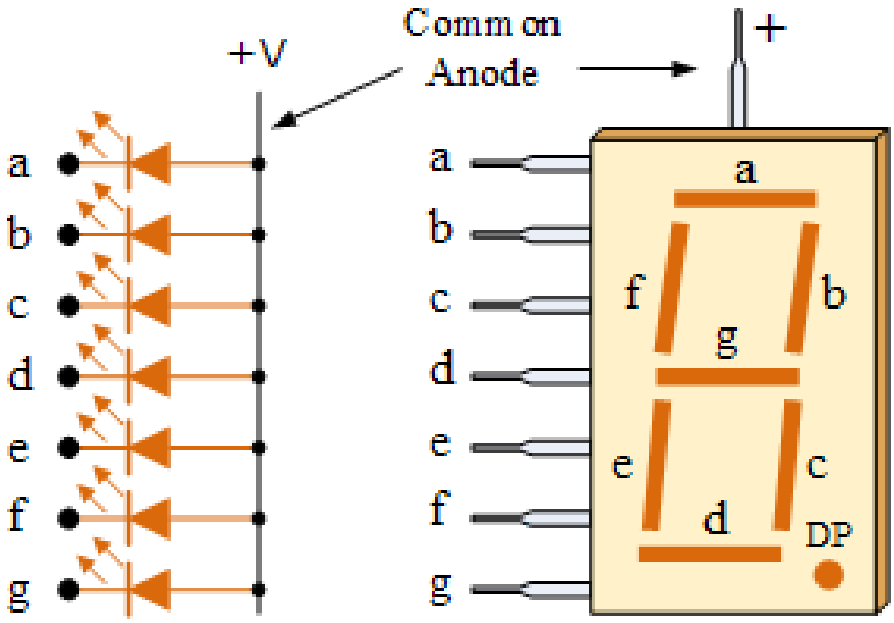
CLK	DATA INPUT				OUTPUT			
	D_A	D_B	D_C	D_D	Q_A	Q_B	Q_C	Q_D
1	1	0	0	1	1	0	0	1
2	1	0	1	0	1	0	1	0

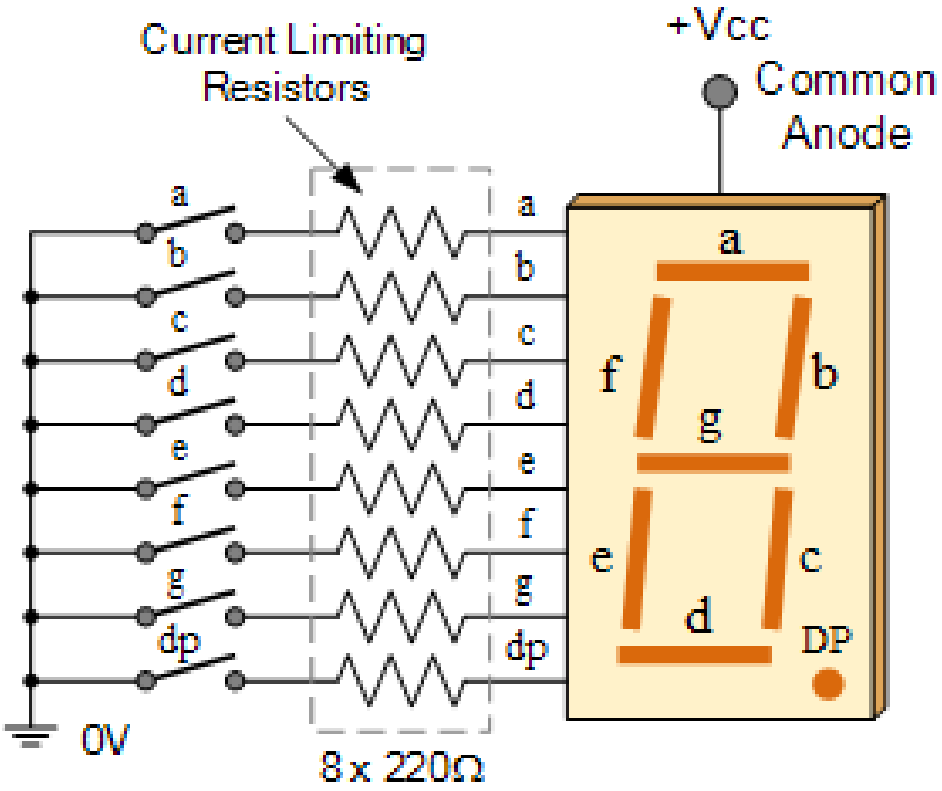
Implementation of digits using seven segment displays

Common Cathode 7-segment Display

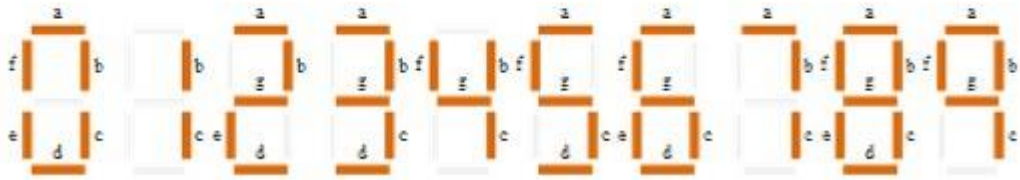


<u>Common Anode 7-segment Display</u>



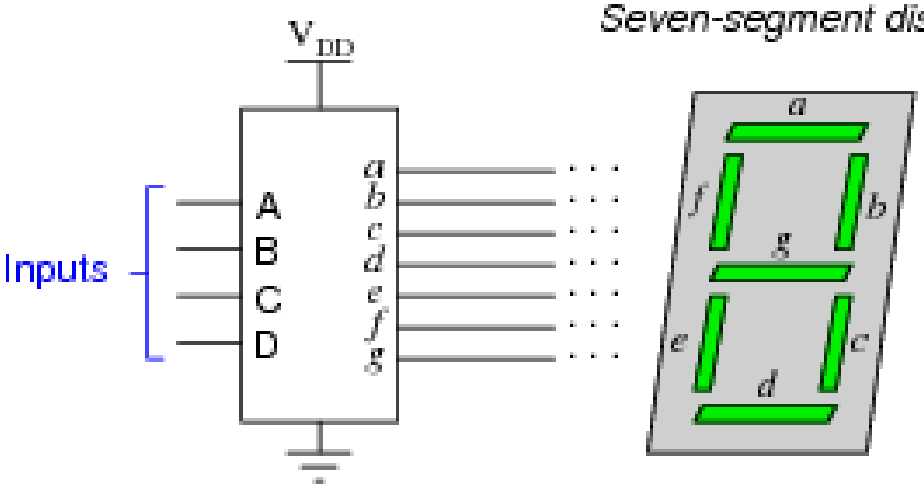


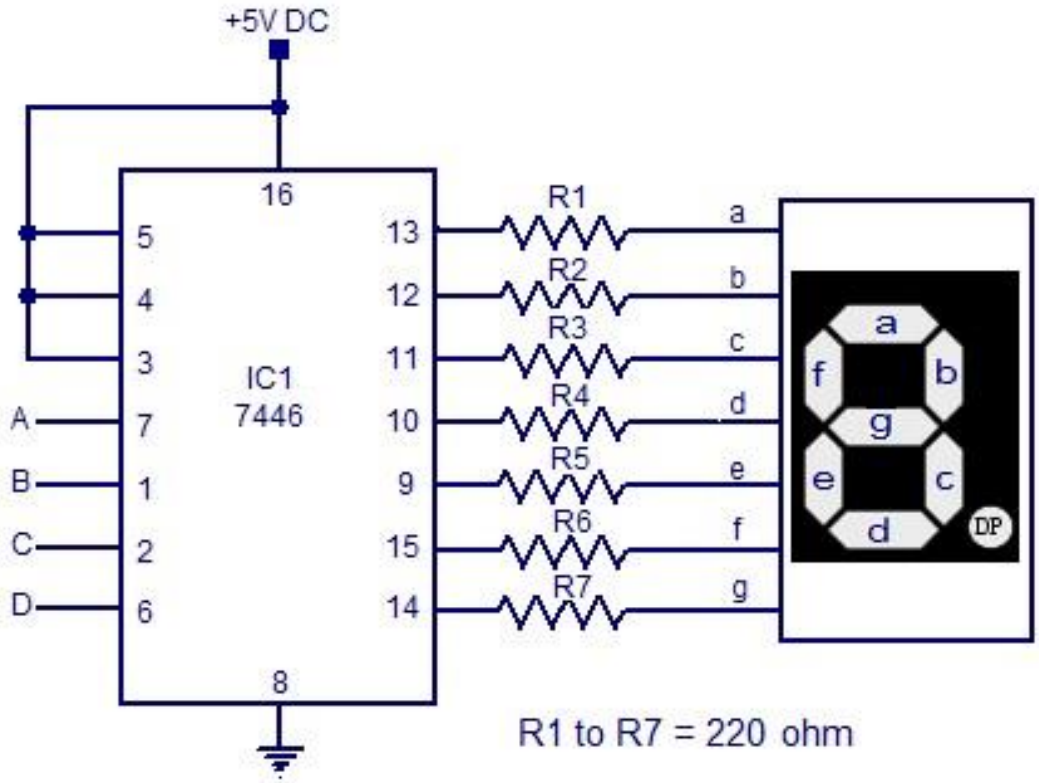
7-Segment Display Segments for all Numbers.



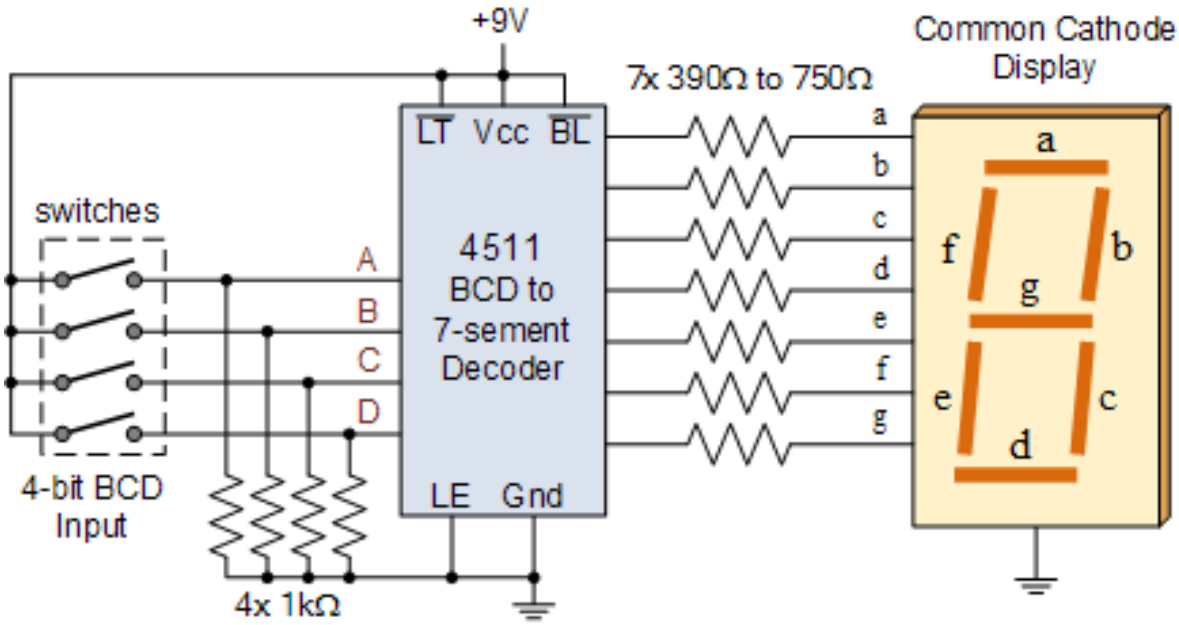
Display driver IC

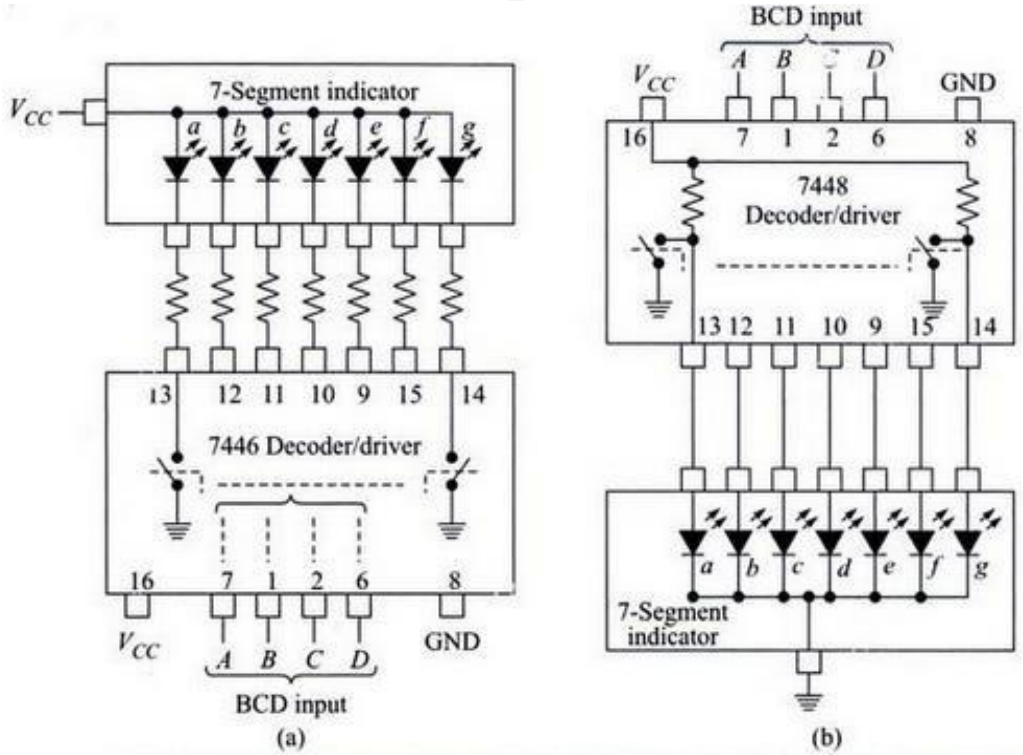
Seven-segment display





Segments Inputs							7 Segment Display Output
a	b	c	d	e	f	g	
0	0	0	0	0	0	1	0
1	0	0	1	1	1	1	1
0	0	1	0	0	1	0	2
0	0	0	0	1	1	0	3
1	0	0	1	1	0	0	4
0	1	0	0	1	0	0	5
0	1	0	0	0	0	0	6
0	0	0	1	1	1	1	7
0	0	0	0	0	0	0	8
0	0	0	0	1	1	0	9





(a) 7446 decoder-driver. (b) 7448 decoder-driver.