DIGITAL ELECTRONICS

F.Y.B.Sc.I.T

SEM-I

B. Sc (Information Technology)		**Semester – I**	
Course Name: Digital Electronics Practical		**Course Code: USIT1P2**	
Periods per week (1 Period is 50 minutes)		**3**	
Credits		**2**	
		Hours	**Marks**
Evaluation System	**Practical Examination**	**2½**	**50**
	Internal	**--**	**--**

List of Practical

1.	**Study of Logic gates and their ICs and universal gates:**
a.	Study of AND, OR, NOT, XOR, XNOR, NAND and NOR gates
b.	IC 7400, 7402, 7404, 7408, 7432, 7486, 74266
c.	Implement AND, OR, NOT, XOR, XNOR using NAND gates.
d.	Implement AND, OR, NOT, XOR, XNOR using NOR gates.
2.	**Implement the given Boolean expressions using minimum number of gates.**
a.	Verifying De Morgan’s laws.
b.	Implement other given expressions using minimum number of gates.
c.	Implement other given expressions using minimum number of ICs.
3.	**Implement combinational circuits.**
a.	Design and implement combinational circuit based on the problem given and minimizing using K-maps.
4.	**Implement code converters.**
a.	Design and implement Binary – to – Gray code converter.
b.	Design and implement Gray – to – Binary code converter.
c.	Design and implement Binary – to – BCD code converter
d.	Design and implement Binary – to – XS-3 code converter
5.	**Implement Adder and Subtractor Arithmetic circuits.**
a.	Design and implement Half adder and Full adder.
b.	Design and implement BCD adder.
c.	Design and implement XS – 3 adder.
d.	Design and implement binary subtractor.
e.	Design and implement BCD subtractor.
f.	Design and implement XS – 3 subtractor.
6.	**Implement Arithmetic circuits.**
a.	Design and implement a 2-bit by 2-bit multiplier.

b.	Design and implement a 2-bit comparator.
7.	**Implement Encode and Decoder and Multiplexer and Demultiplexers.**
a.	Design and implement 8:3 encoder.
b.	Design and implement 3:8 decoder.
c.	Design and implement 4:1 multiplexer. Study of IC 74153, 74157
d.	Design and implement 1:4 demultiplexer. Study of IC 74139
e.	Implement the given expression using IC 74151 8:1 multiplexer.
f.	Implement the given expression using IC 74138 3:8 decoder.
8.	**Study of flip-flops and counters.**
a.	Study of IC 7473.
b.	Study of IC 7474.
c.	Study of IC 7476.
d.	Conversion of Flip-flops.
e.	Design of 3-bit synchronous counter using 7473 and required gates.
f.	Design of 3-bit ripple counter using IC 7473.
9.	**Study of counter ICs and designing Mod-N counters.**
a.	Study of IC 7490, 7492, 7493 and designing mod-n counters using these.
b.	Designing mod-n counters using IC 7473 and 7400 (NAND gates)
10.	**Design of shift registers and shift register counters.**
a.	Design serial – in serial – out, serial – in parallel – out, parallel – in serial – out, parallel – in parallel – out and bidirectional shift registers using IC 7474.
b.	Study of ID 7495.
c.	Implementation of digits using seven segment displays.

Books and References:

Sr. No.	**Title**	**Author/s**	**Publisher**	**Edition**	**Year**
1.	Digital Electronics and Logic Design	N. G. Palan	Technova		
2.	Digital Principles and Applications	Malvino and Leach	Tata McGraw Hill		

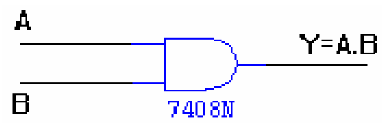
PRACTICAL-1

Study of Logic gates and their ICs and universal gates:

1. Study of AND, OR, NOT, XOR, XNOR, NAND and NOR gates
2. IC 7400, 7402, 7404, 7408, 7432, 7486, 74266
3. Implement AND, OR, NOT, XOR, XNOR using NAND gates.
4. Implement AND, OR, NOT, XOR, XNOR using NOR gates.

AND GATE:

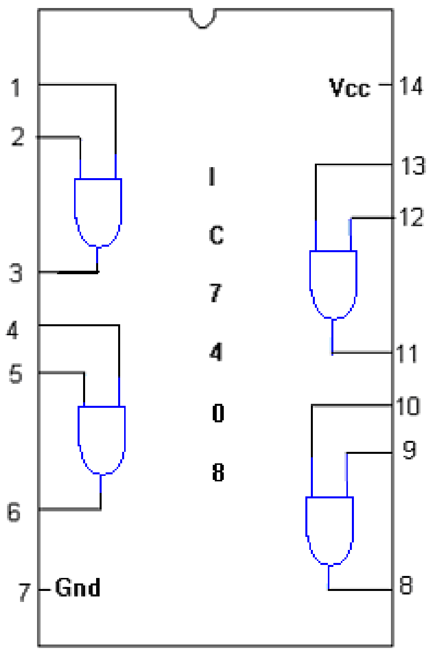
SYMBOL:



TRUTH TABLE

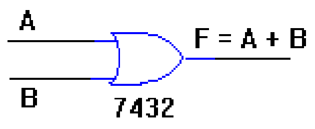
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

PIN DIAGRAM:



OR GATE:

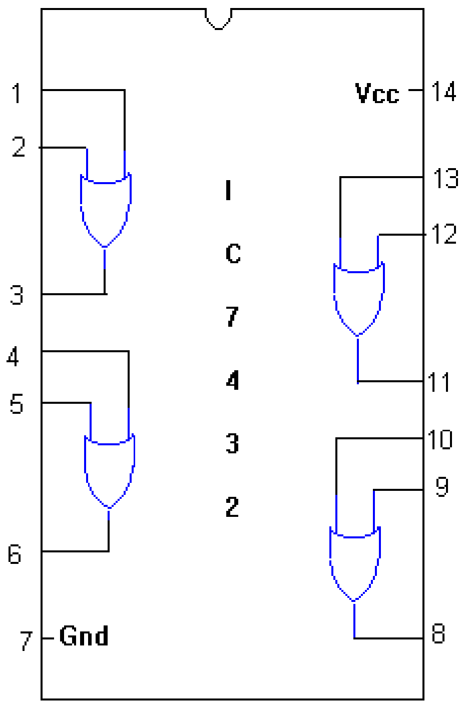
SYMBOL :



TRUTH TABLE

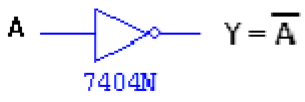
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

PIN DIAGRAM :

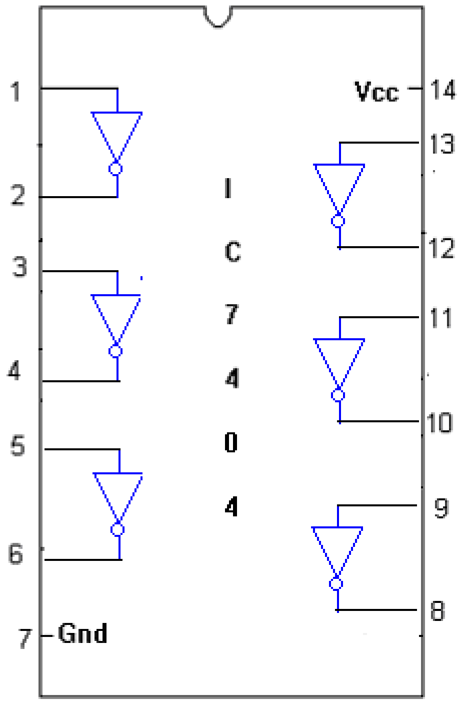


NOT GATE:

SYMBOL:



PIN DIAGRAM:

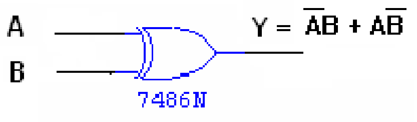


TRUTH TABLE :

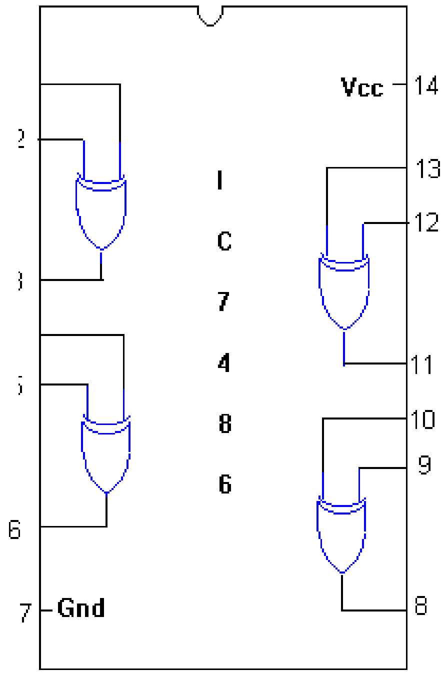
A	$\overline{A}$
0	1
1	0

X-OR GATE :

SYMBOL :



PIN DIAGRAM :

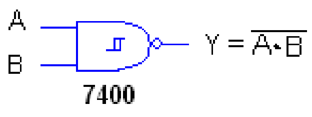


TRUTH TABLE :

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

2-INPUT NAND GATE:

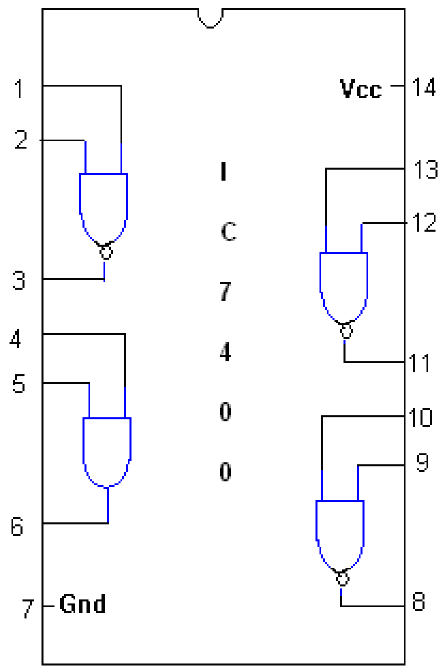
SYMBOL:



TRUTH TABLE

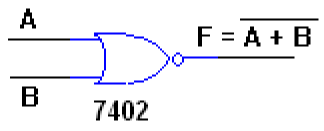
A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

PIN DIAGRAM:

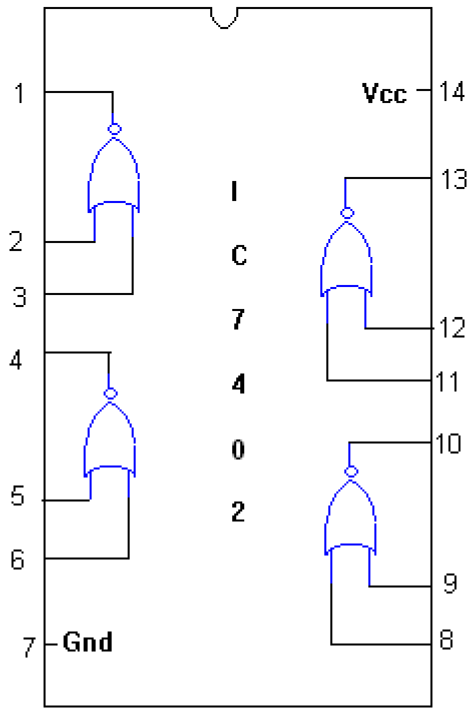


NOR GATE:

SYMBOL :



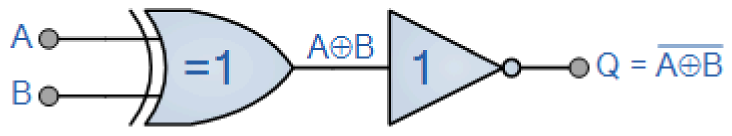
PIN DIAGRAM :



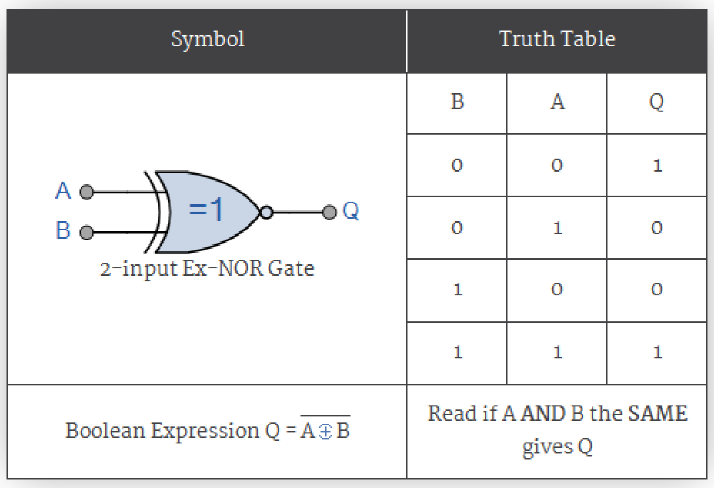
TRUTH TABLE

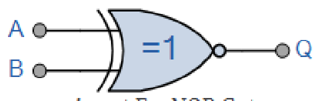
A	B	$\overline{A+B}$
0	0	1
0	1	1
1	0	1
1	1	0

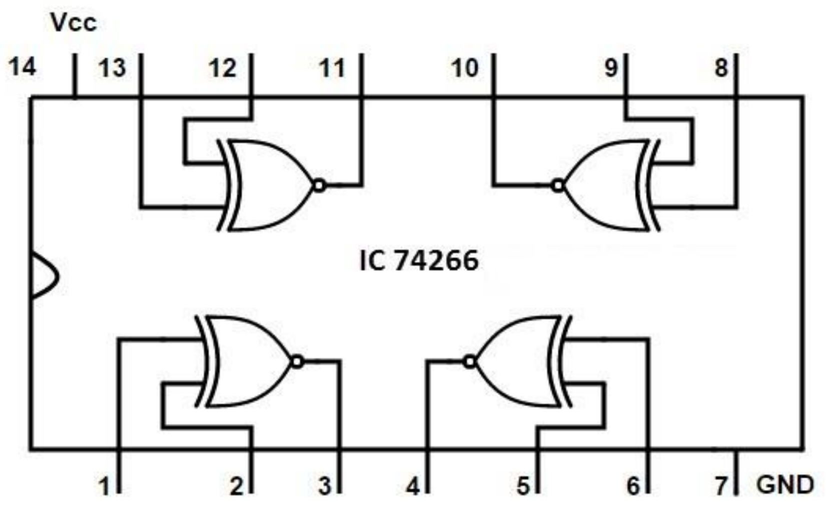
IC 74266



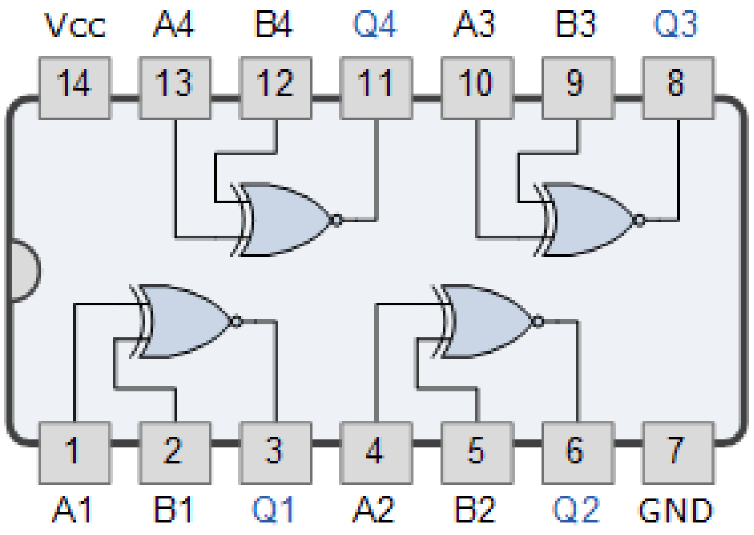
2-input "Ex-OR" gate plus a "NOT" gate



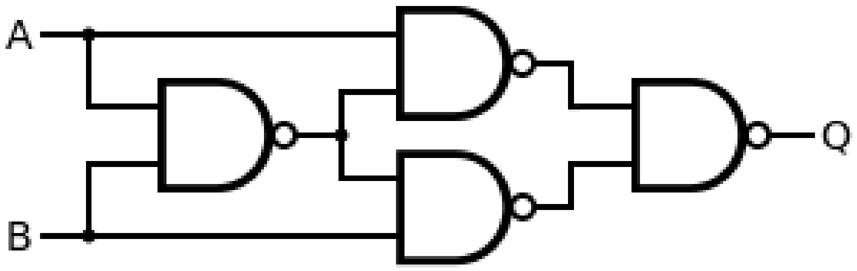
Symbol	Truth Table		
	B	A	Q
 2-input Ex-NOR Gate	0	0	1
	0	1	0
	1	0	0
	1	1	1
Boolean Expression $Q = \overline{A \oplus B}$	Read if A AND B the SAME gives Q		



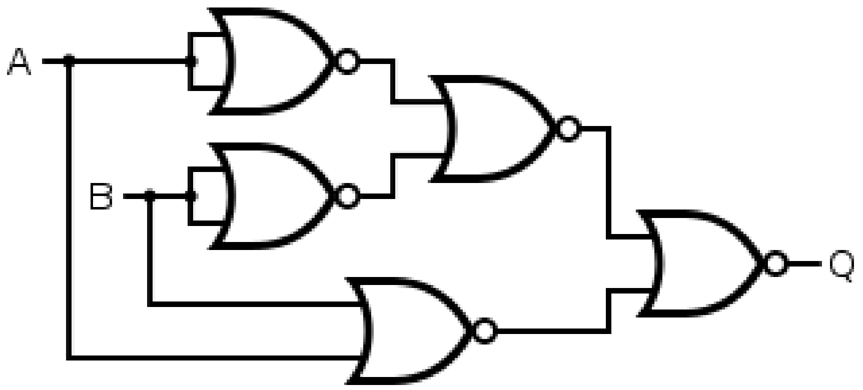
74266 Quad 2-input Ex-NOR Gate



XOR gate from NAND gates:



XOR gate from NOR gates:



PRACTICAL-2
Implement the given Boolean expressions using minimum number of gates

1. Verifying De Morgan’s laws.
2. Implement other given expressions using minimum number of gates. (any expression from Chapter 3, Reference 1)
3. Implement other given expressions using minimum number of ICs. (any expression from Chapter 3, Reference 1)

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

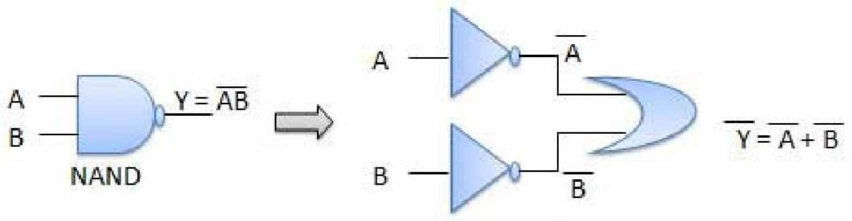
Theorem 1

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

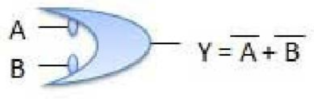
NAND = Bubbled OR

- 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.

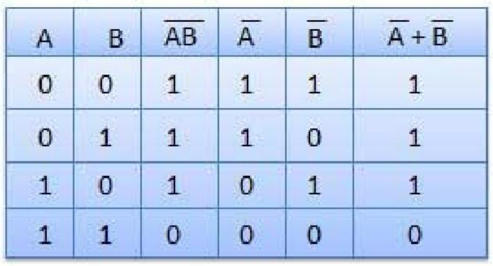
- This OR gate is called as **Bubbled OR**.



NAND $\equiv$ Bubbled OR



Bubbled OR



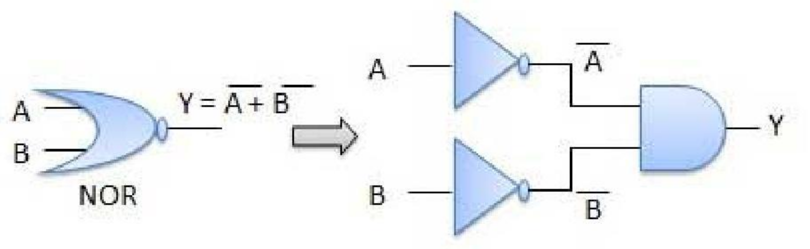
A	B	$\overline{AB}$	$\overline{A}$	$\overline{B}$	$\overline{A} + \overline{B}$
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

Theorem 2

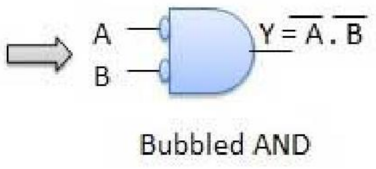
$$\overline{A + B} = \overline{A}.\overline{B}$$

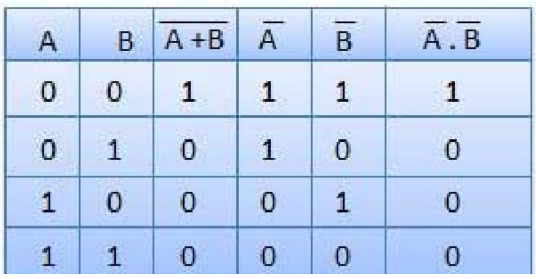
NOR = Bubbled AND

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 $\equiv$ 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
Implement combinational circuits

Design and implement combinational circuit based on the problem given and minimizing using K-maps.

(any question from Chapter 5, Reference 1)

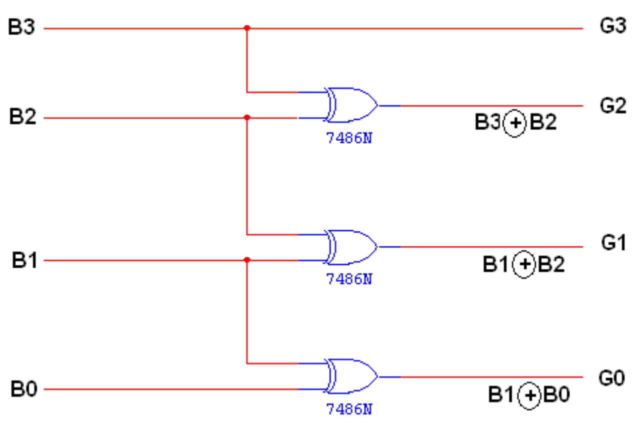
PRACTICAL-4

Implement code converters

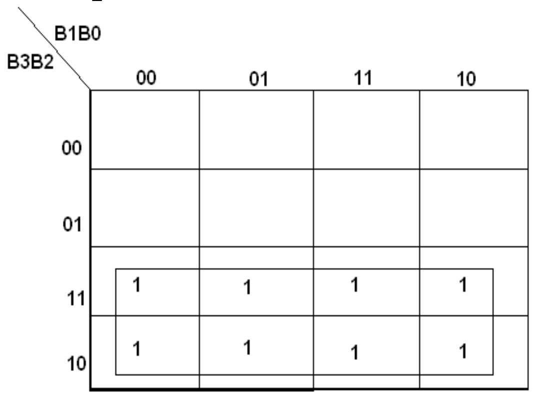
1. **Design and implement Binary – to – Gray code converter**
2. **Design and implement Gray – to – Binary code converter**
3. Design and implement Binary – to – BCD code converter
4. Design and implement Binary – to – XS-3 code converter

LOGIC DIAGRAM:

BINARY TO GRAY CODE CONVERTOR



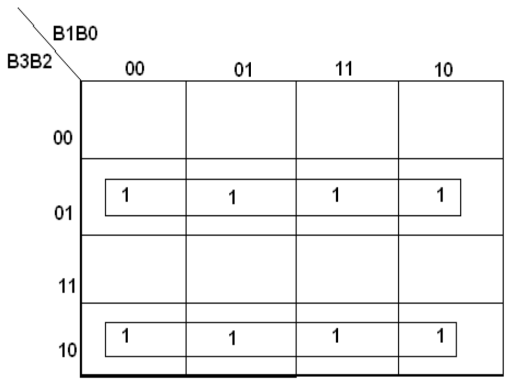
K-Map for G_3:



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

$G_3 = B_3$

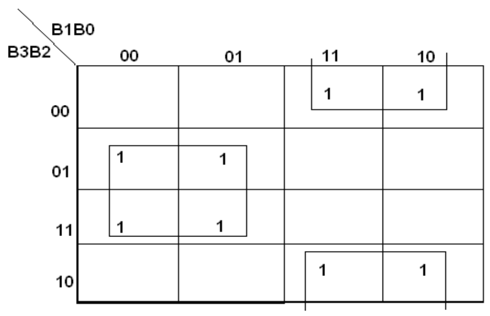
K-Map for G_2:



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

$G2 = B3 \oplus B2$

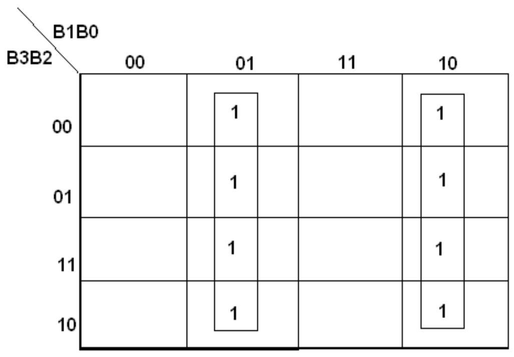
K-Map for G_1:



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

$G1 = B1 \oplus B2$

K-Map for G_0:



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

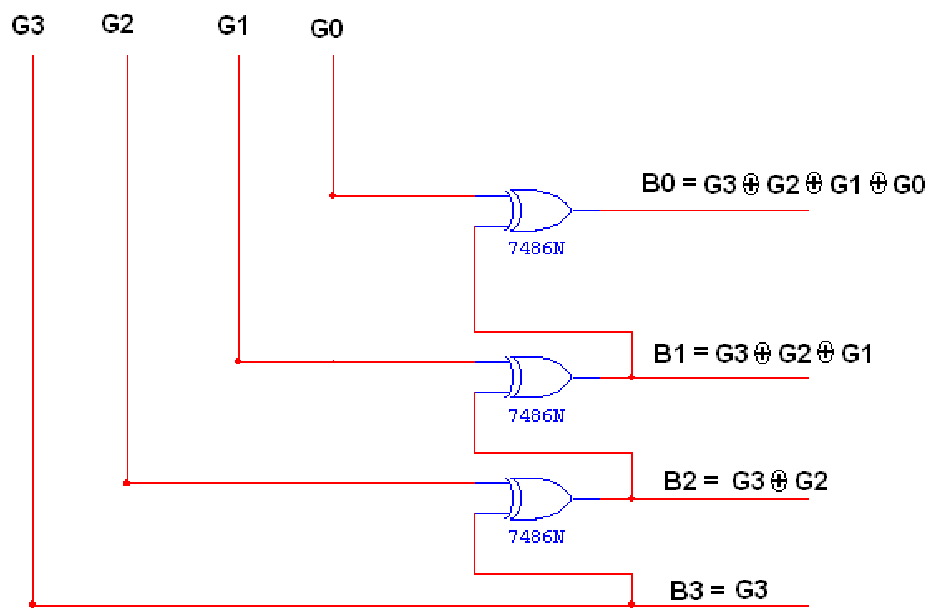
$G0 = B1 \oplus B0$

TRUTH TABLE:

Binary input				Gray code output			
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

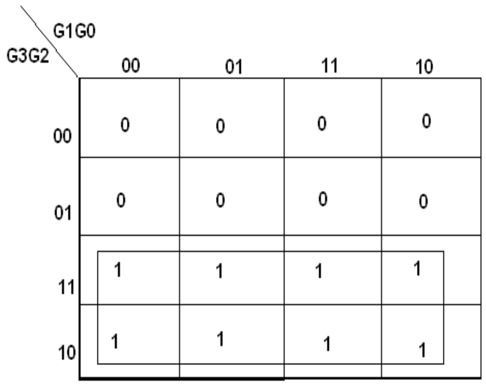


TRUTH TABLE:

| Gray Code | Binary Code |

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

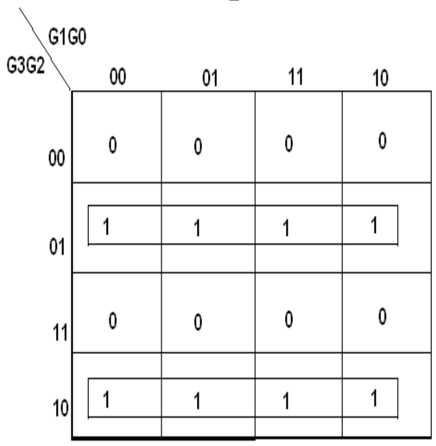
K-Map for B_3:



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

B3 = G3

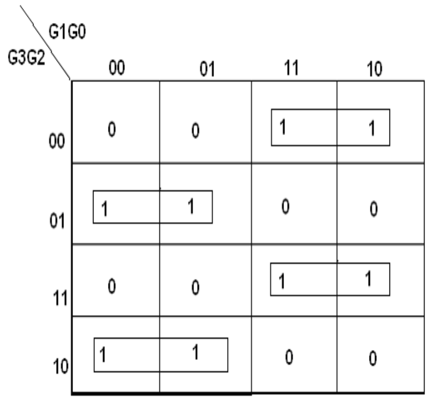
K-Map for B_2:



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

$B2 = G3 \oplus G2$

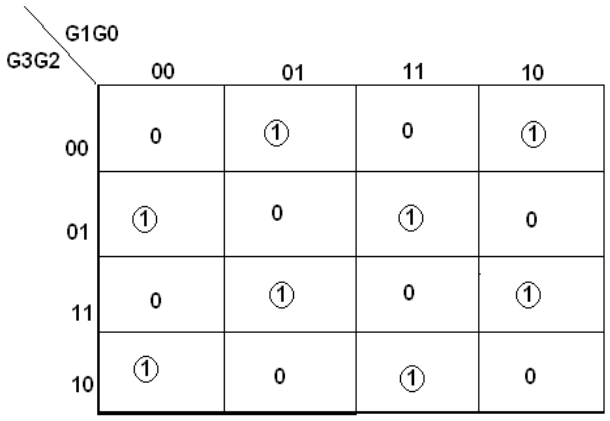
K-Map for B_1:



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

$B1 = G3 \oplus G2 \oplus G1$

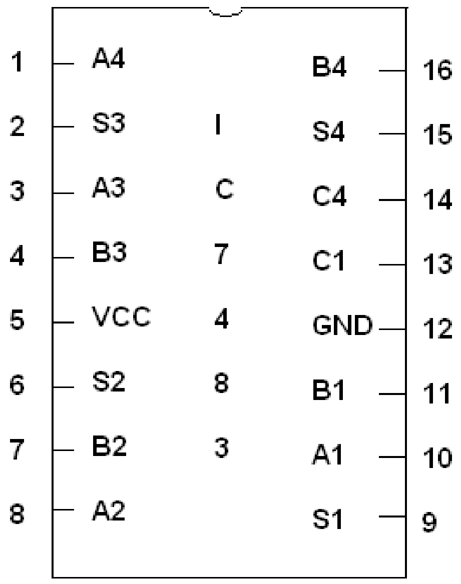
K-Map for B_0:



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

$B0 = G3 \oplus G2 \oplus G1 \oplus G0$

PIN DIAGRAM FOR IC 7483:



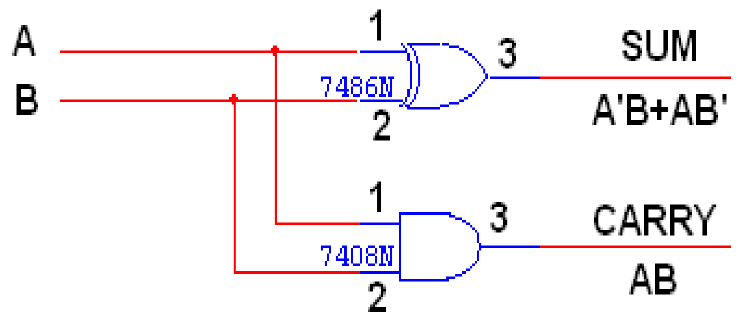
PRACTICAL-5

Implement Adder and Subtractor Arithmetic circuits

1. **Design and implement half adder and Full adder.**
2. **Design and implement BCD adder.**
3. Design and implement XS – 3 adder.
4. **Design and implement binary subtractor.**
5. Design and implement BCD subtractor.
6. Design and implement XS – 3 subtractor.

LOGIC DIAGRAM:

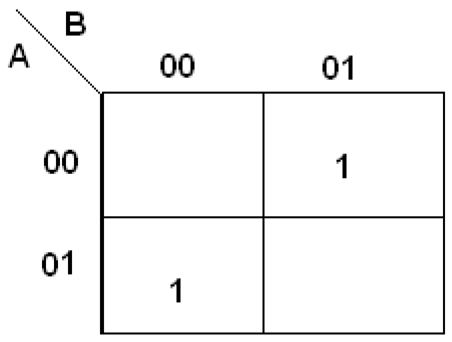
HALF ADDER

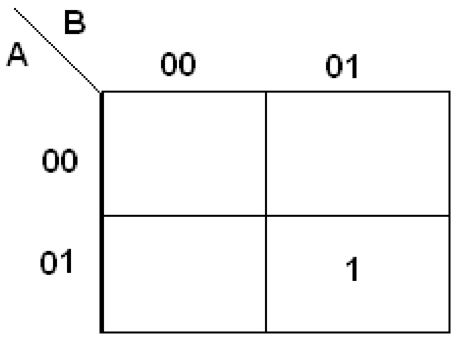


TRUTH TABLE:

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

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

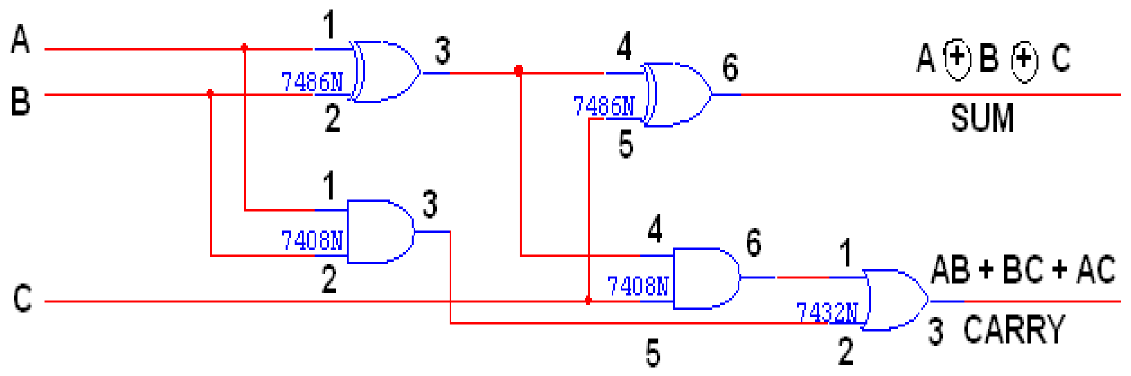




SUM = A'B + AB' **CARRY = AB**

LOGIC DIAGRAM:

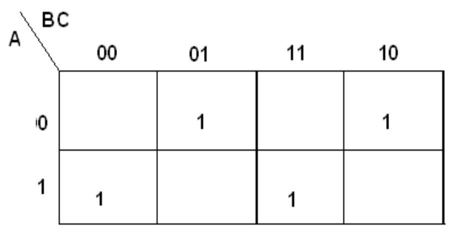
FULL ADDER USING TWO HALF ADDER



TRUTH TABLE:

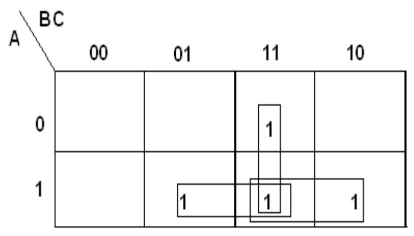
A	B	C	CARRY	SUM
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

K-Map for SUM:



SUM = A’B’C + A’BC’ + ABC’ + ABC

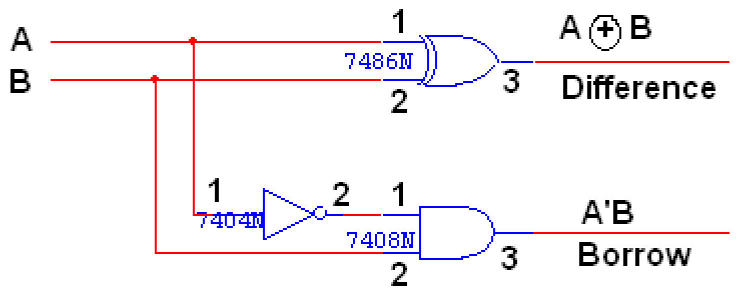
K-Map for CARRY:



CARRY = AB + BC + AC

LOGIC DIAGRAM:

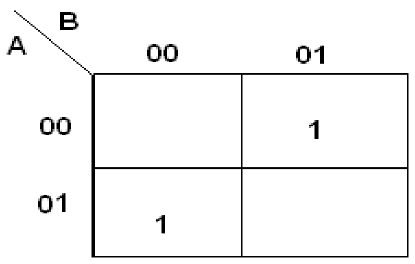
HALF SUBTRACTOR



TRUTH TABLE:

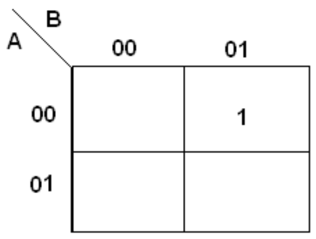
A	B	BORROW	DIFFERENCE
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

K-Map for DIFFERENCE:



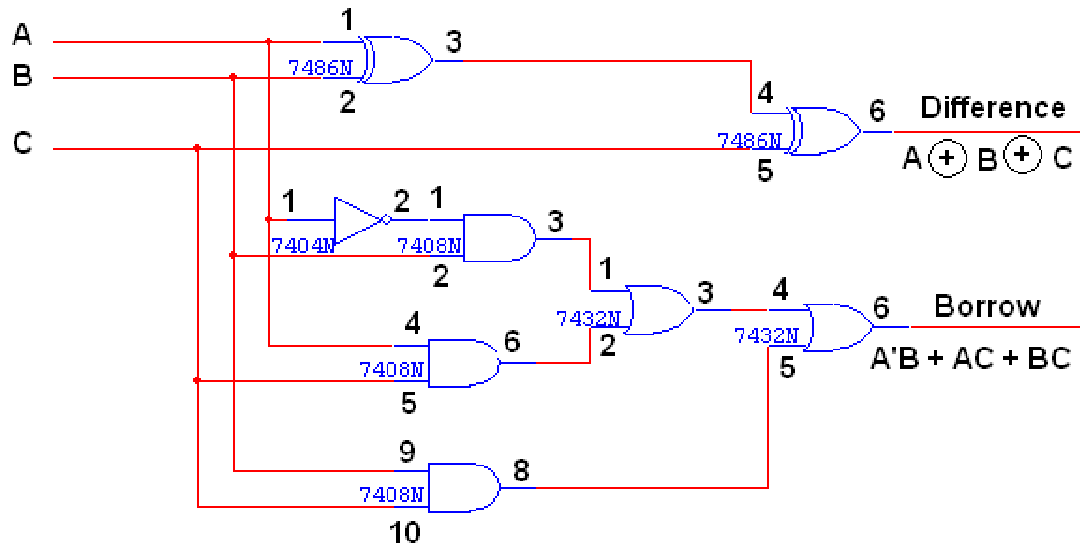
DIFFERENCE = A’B + AB’

K-Map for BORROW:

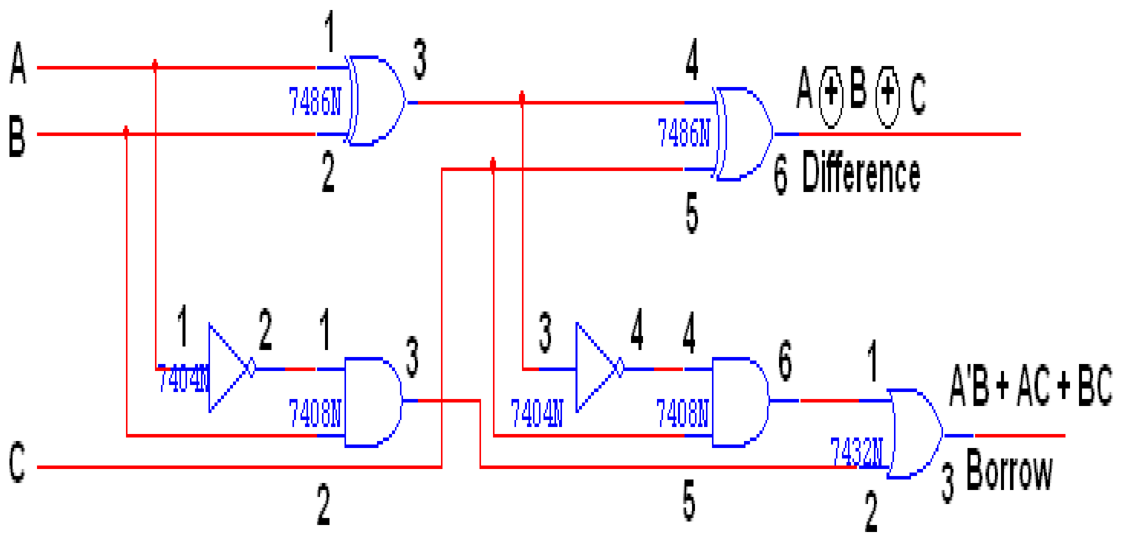


BORROW = A’B

FULL SUBTRACTOR



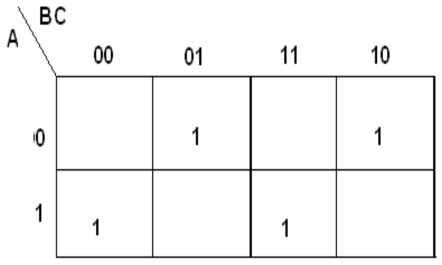
FULL SUBTRACTOR USING TWO HALF SUBTRACTOR:

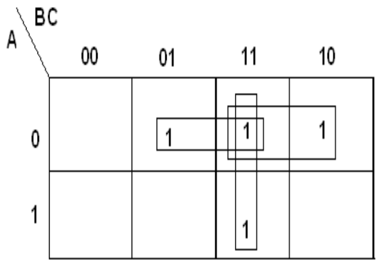


TRUTH TABLE:

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

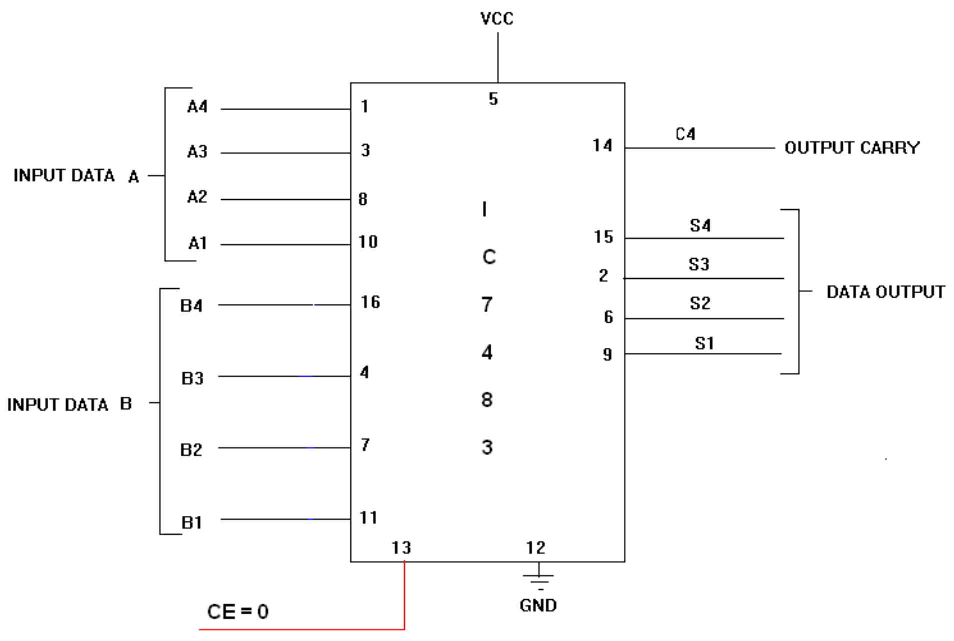
K-Map for Difference: **K-Map for Borrow:**



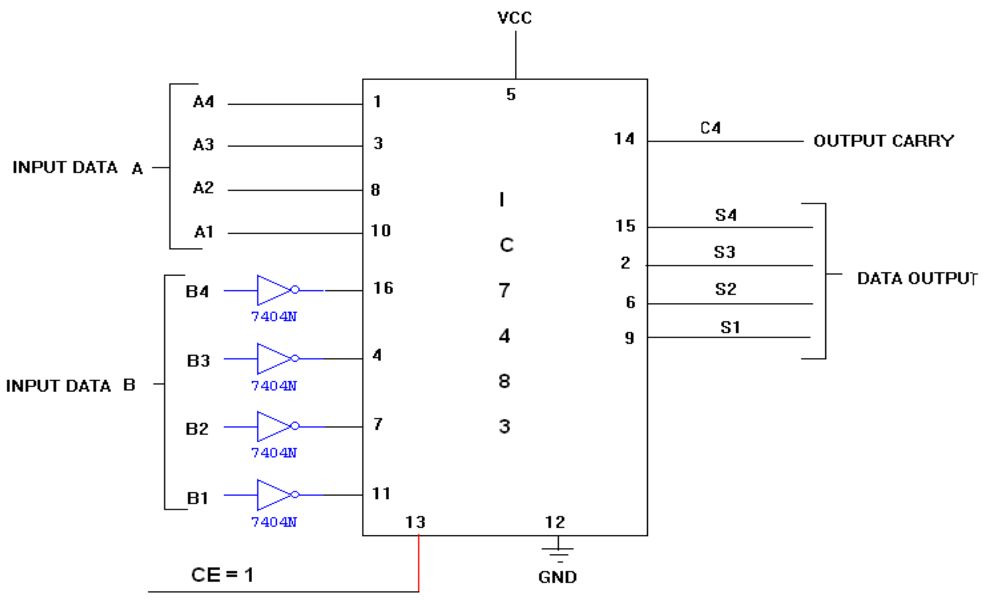


Difference = A’B’C + A’BC’ + AB’C’ + ABC **Borrow = A’B + BC + A’C**

LOGIC DIAGRAM:
4-BIT BINARY ADDER

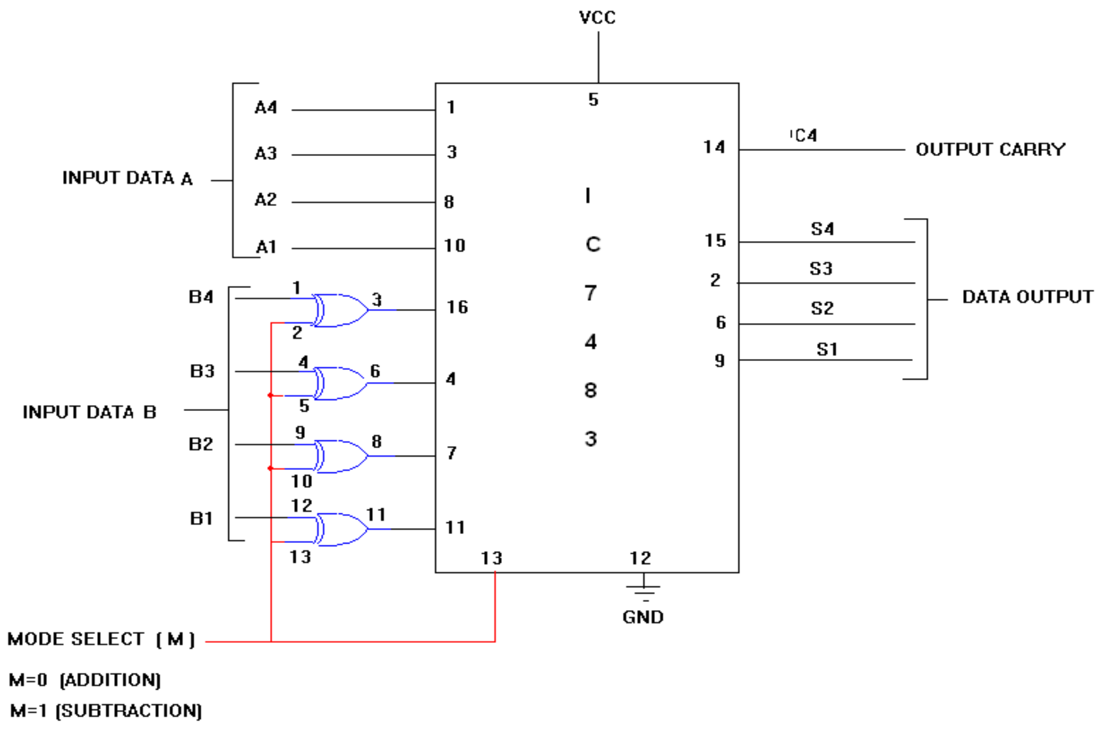


LOGIC DIAGRAM:
4-BIT BINARY SUBTRACTOR



LOGIC DIAGRAM:

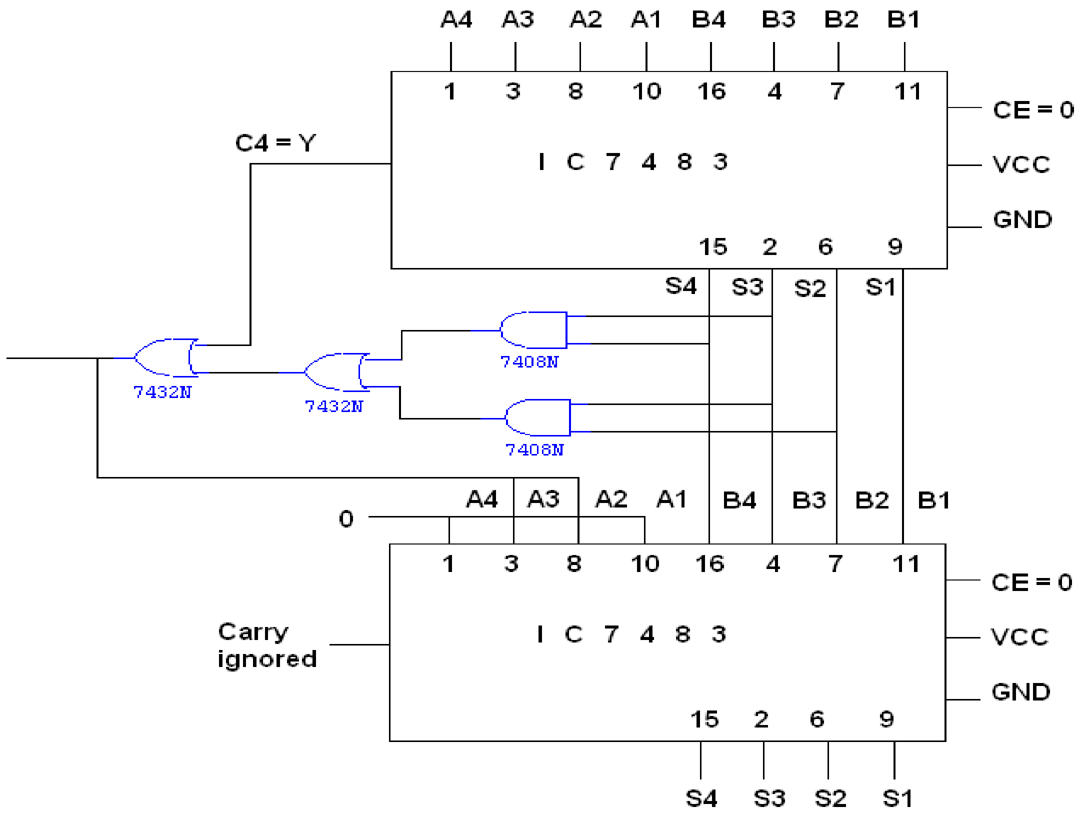
4-BIT BINARY ADDER/SUBTRACTOR



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

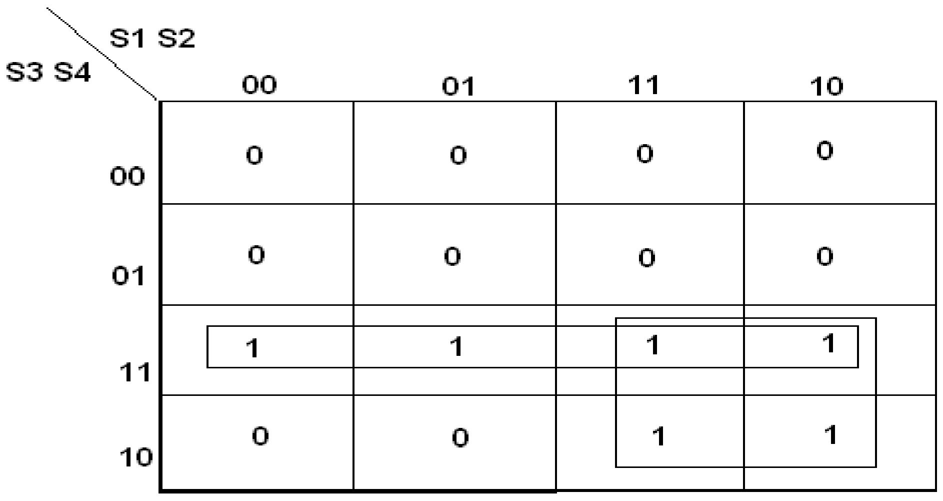
LOGIC DIAGRAM:

BCD ADDER



TRUTH TABLE:

K MAP



Y = S4 (S3 + S2)

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

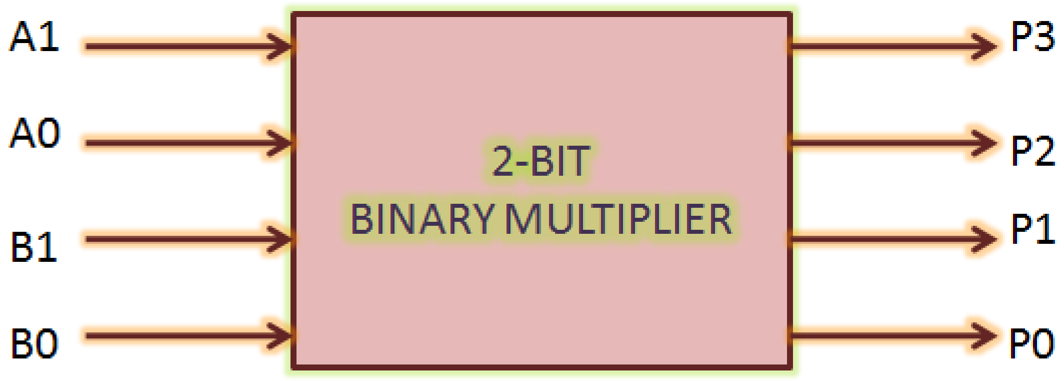
PRACTICAL-6

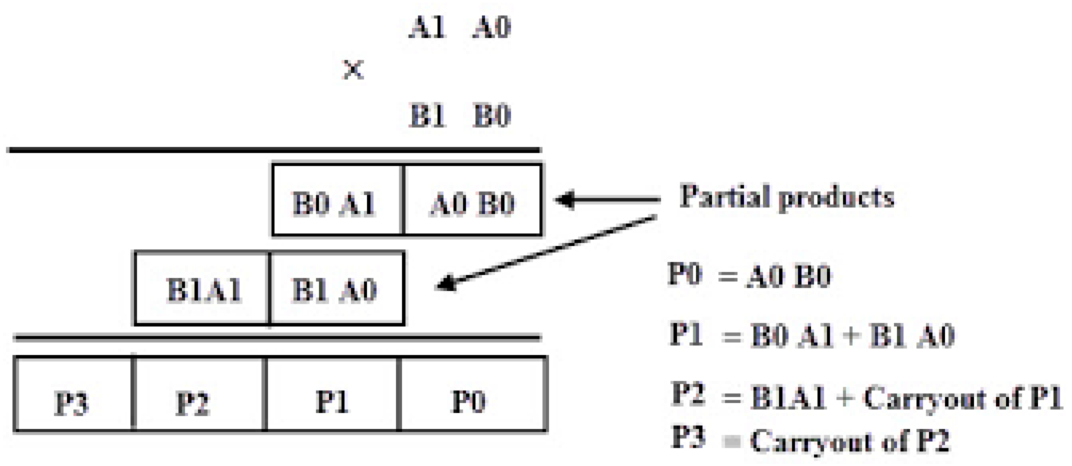
Implement Arithmetic circuits

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

2. Design and implement a 2-bit comparator

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





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

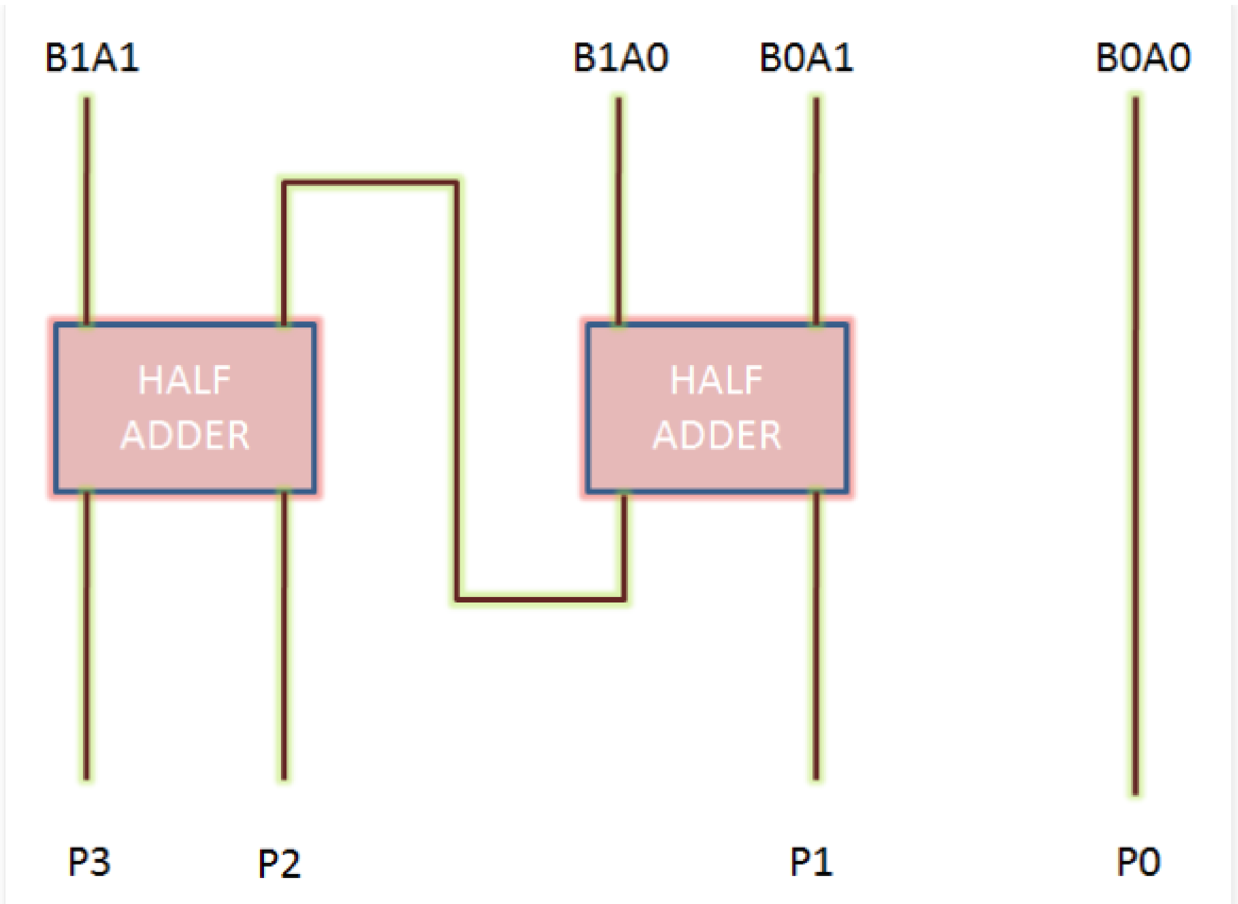
we get the partial products as:

P0 = A0*B0

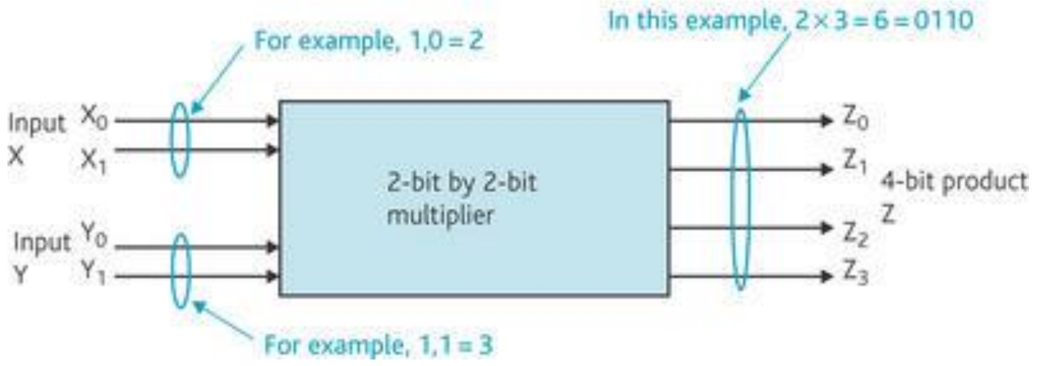
P1 = A0*B1 xor A1 * B0 ; carry generated here goes to next stage

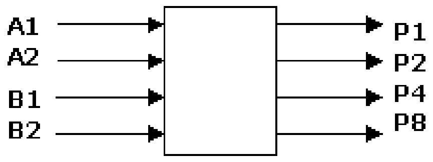
P2 = A1*B1 xor (A0*B1) * (A1*B0)

P3 = A1*B1 and (A0*B1) * (A1*B0)



Two-bit binary multiplier





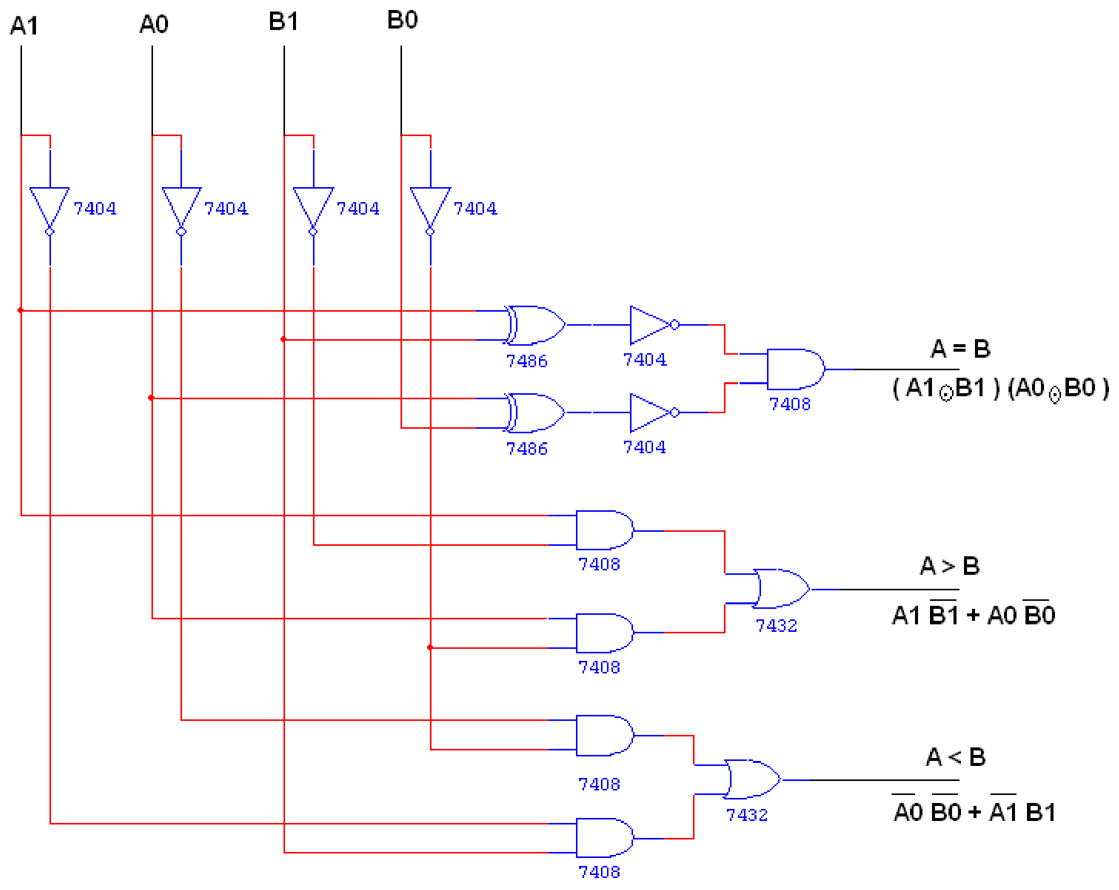
block diagram
and
truth table

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

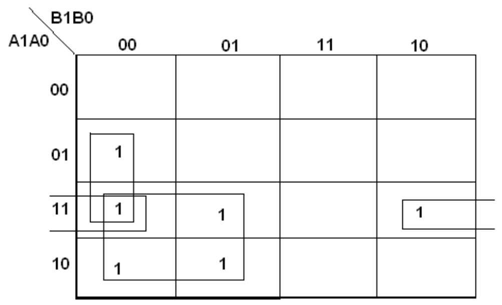
4-variable K-map
for each of the 4
output functions

LOGIC DIAGRAM:

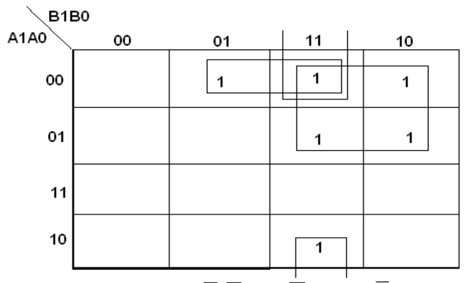
2 BIT MAGNITUDE COMPARATOR



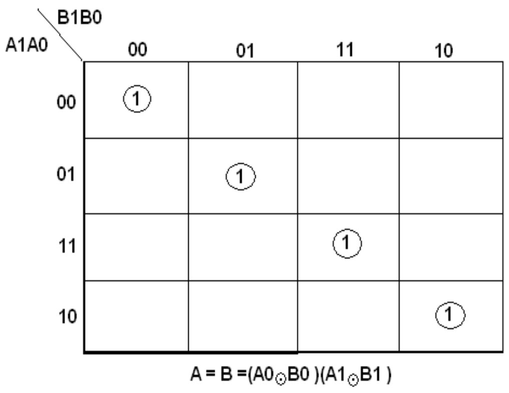
K MAP



$A > B = A0\,\overline{B0}\,B1 + A1\,\overline{B1} + A1\,A0\,\overline{B0}$



$A < B = \overline{A1}\,\overline{A0}\,B0 + \overline{A0}\,B0\,B1 + \overline{A1}\,B1$

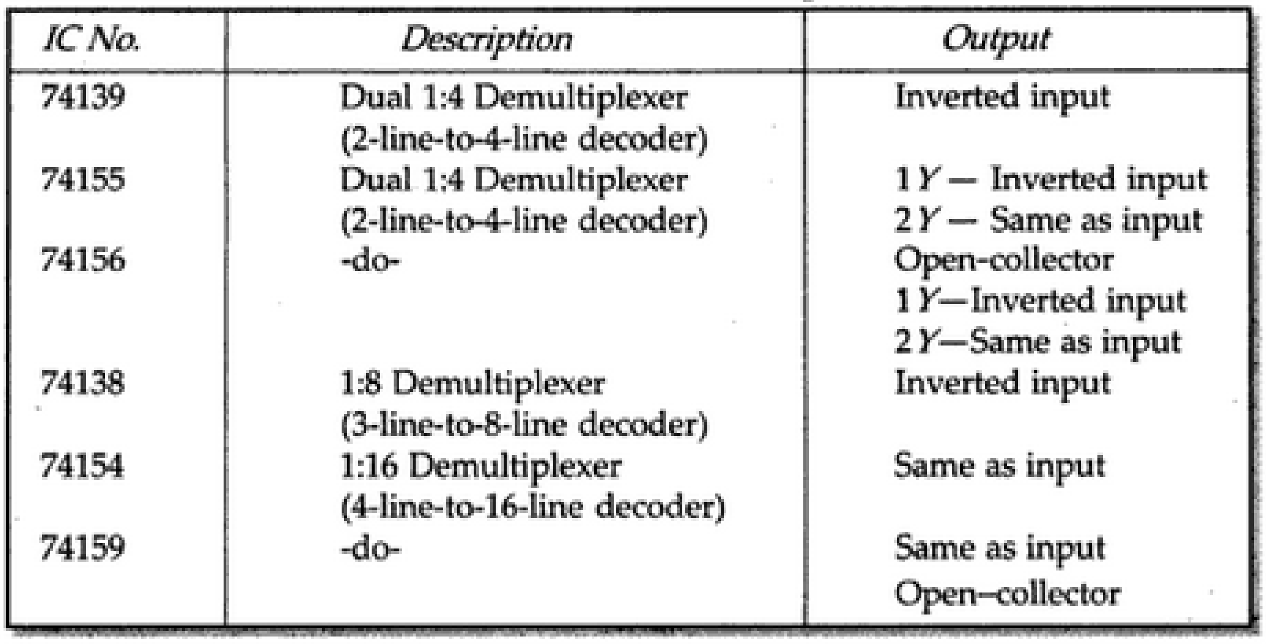


TRUTH TABLE

A1	A0	B1	B0	A > B	A = B	A < B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

PRACTICAL-7
Implement Encode and Decoder and Multiplexer and De-multiplexers

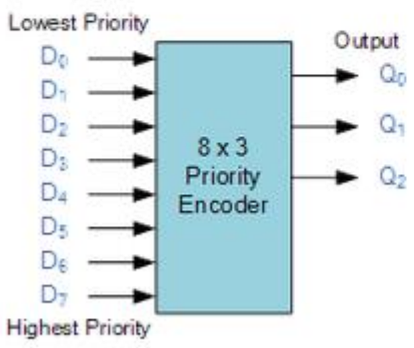
1. Design and implement 8:3 encoder.
2. Design and implement 3:8 decoder.
3. **Design and implement 4:1 multiplexer**. Study of IC 74153, 74157
4. **Design and implement 1:4 demultiplexer.** Study of IC 74139
5. Implement the given expression using IC 74151 8:1 multiplexer
6. Implement the given expression using IC 74138 3:8 decoder

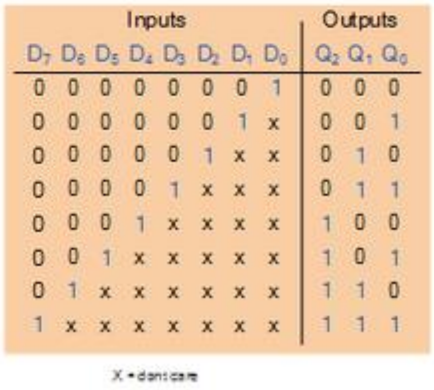


IC No.	*Description*	*Output*
74139	Dual 1:4 Demultiplexer (2-line-to-4-line decoder)	Inverted input
74155	Dual 1:4 Demultiplexer (2-line-to-4-line decoder)	1 *Y* — Inverted input 2 *Y* — Same as input
74156	-do-	Open-collector 1 *Y*—Inverted input 2 *Y*—Same as input
74138	1:8 Demultiplexer (3-line-to-8-line decoder)	Inverted input
74154	1:16 Demultiplexer (4-line-to-16-line decoder)	Same as input
74159	-do-	Same as input Open–collector

Design and implement 8:3 encoder

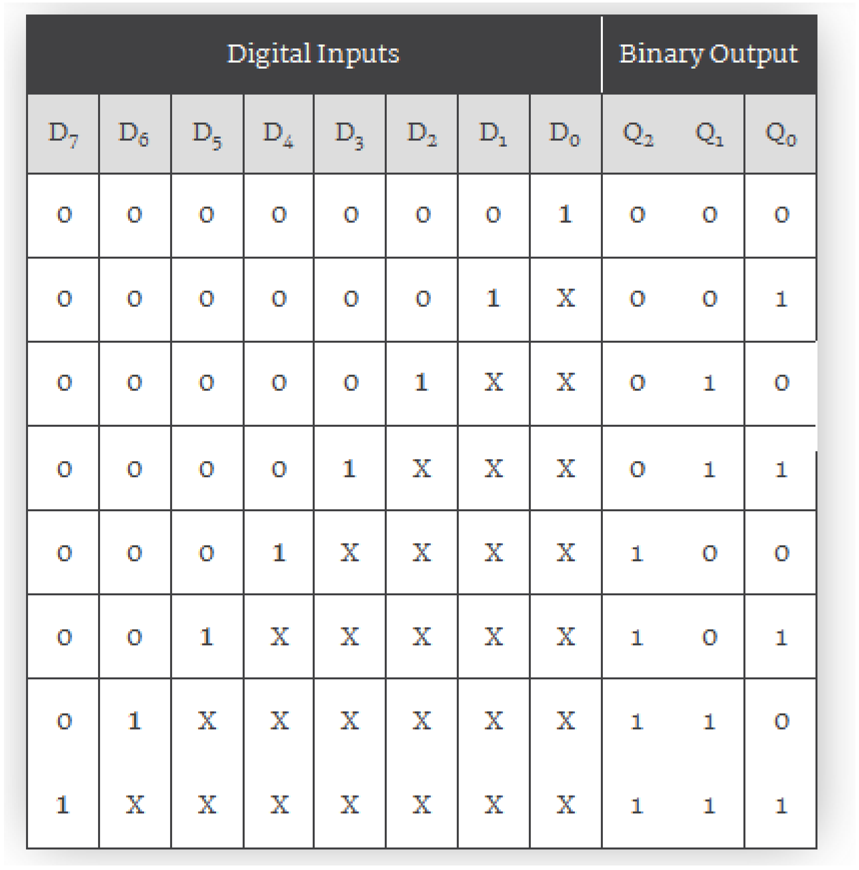
8-to-3 Bit Priority Encoder



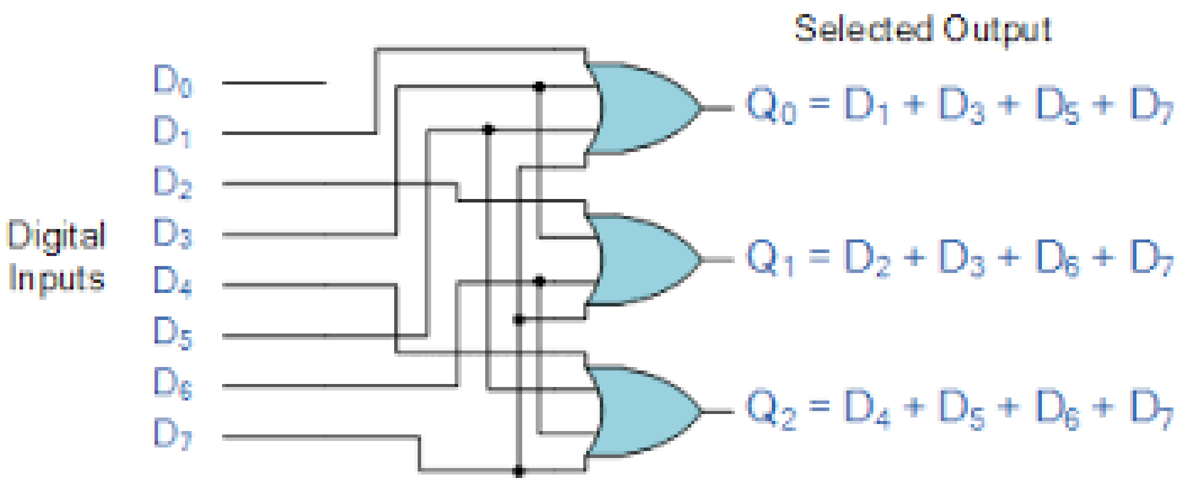


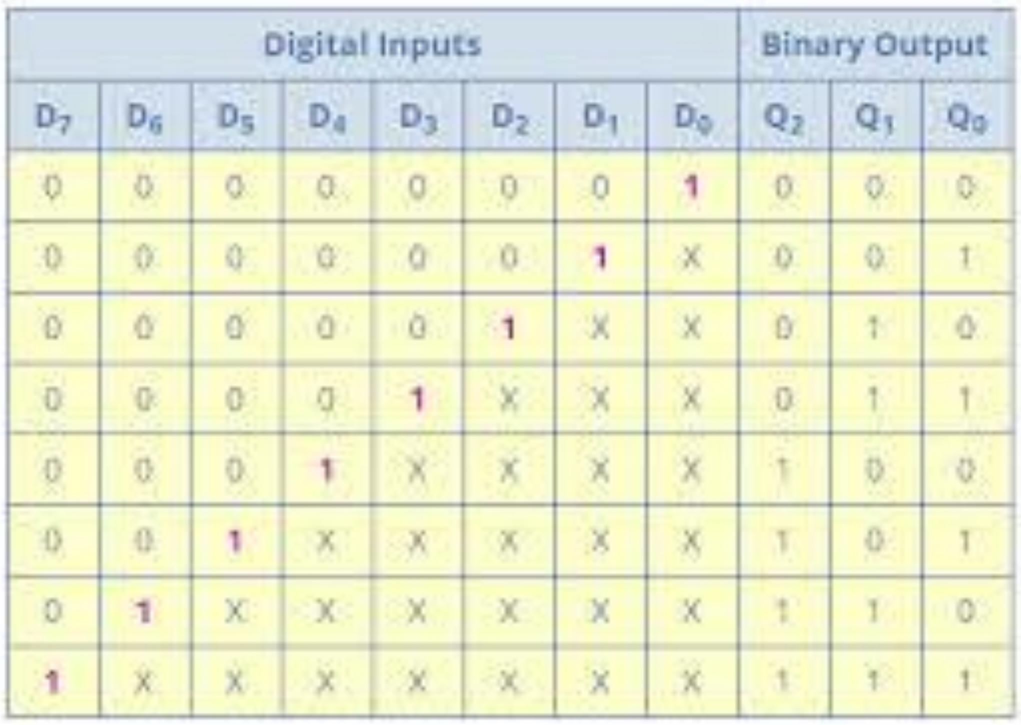
Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

X = don't care



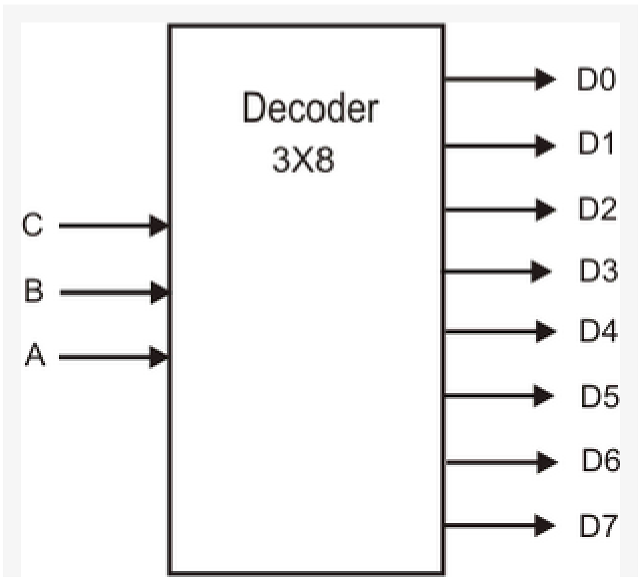
Digital Inputs								Binary Output		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	1	X	X	0	1	0
0	0	0	0	1	X	X	X	0	1	1
0	0	0	1	X	X	X	X	1	0	0
0	0	1	X	X	X	X	X	1	0	1
0	1	X	X	X	X	X	X	1	1	0
1	X	X	X	X	X	X	X	1	1	1

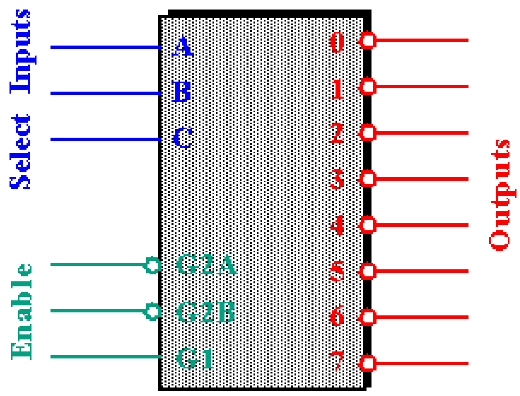

Selected Output
Digital Inputs
D_0
D_1
D_2
D_3
D_4
D_5
D_6
D_7
$Q_0 = D_1 + D_3 + D_5 + D_7$
$Q_1 = D_2 + D_3 + D_6 + D_7$
$Q_2 = D_4 + D_5 + D_6 + D_7$

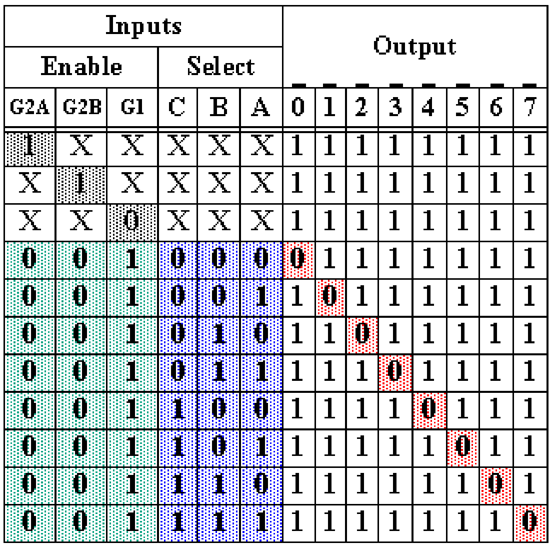


Digital Inputs								Binary Output		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	1	X	X	0	1	0
0	0	0	0	1	X	X	X	0	1	1
0	0	0	1	X	X	X	X	1	0	0
0	0	1	X	X	X	X	X	1	0	1
0	1	X	X	X	X	X	X	1	1	0
1	X	X	X	X	X	X	X	1	1	1

Design and implement 3:8 decoder

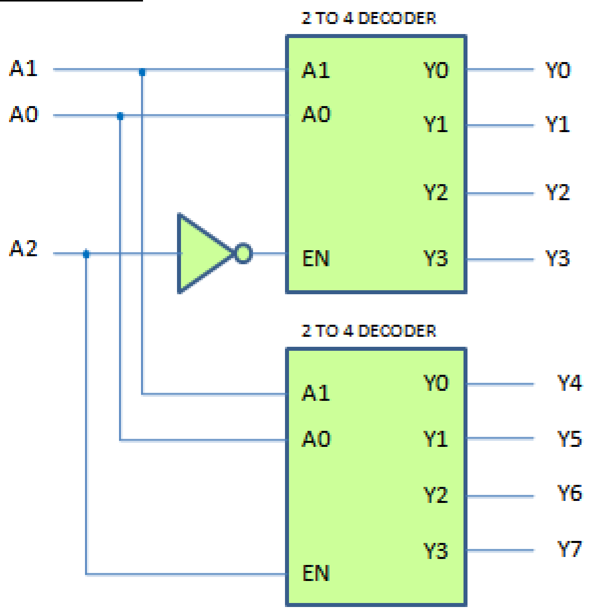




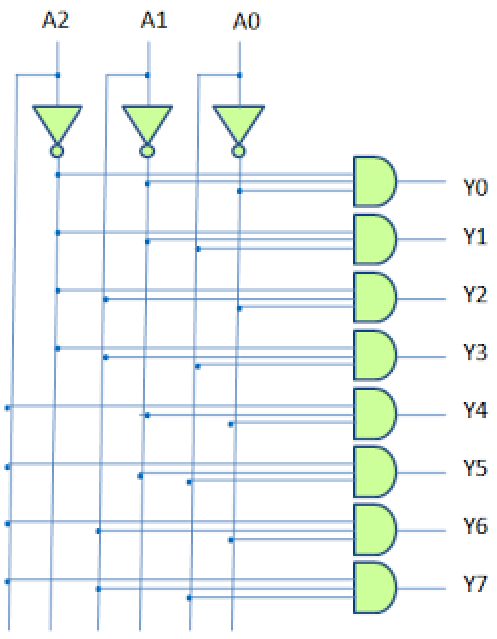


Inputs						Output							
Enable			Select										
G2A	G2B	G1	C	B	A	$\overline{0}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	$\overline{5}$	$\overline{6}$	$\overline{7}$
1	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	0	X	X	X	1	1	1	1	1	1	1	1
0	0	1	0	0	0	0	1	1	1	1	1	1	1
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:

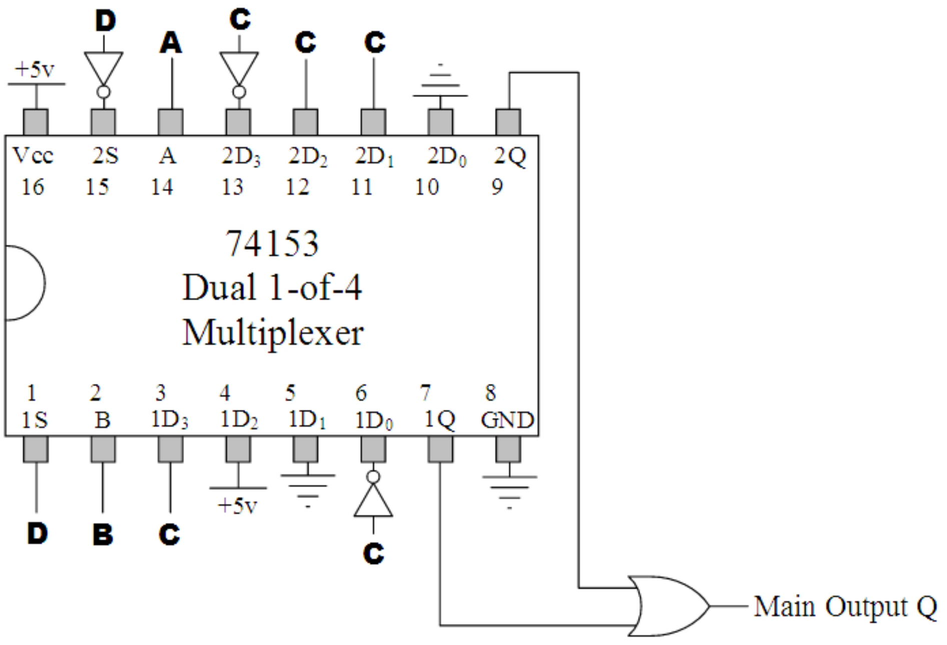


3 to 8 decoder using gates:



Design and implement 4:1 multiplexer.

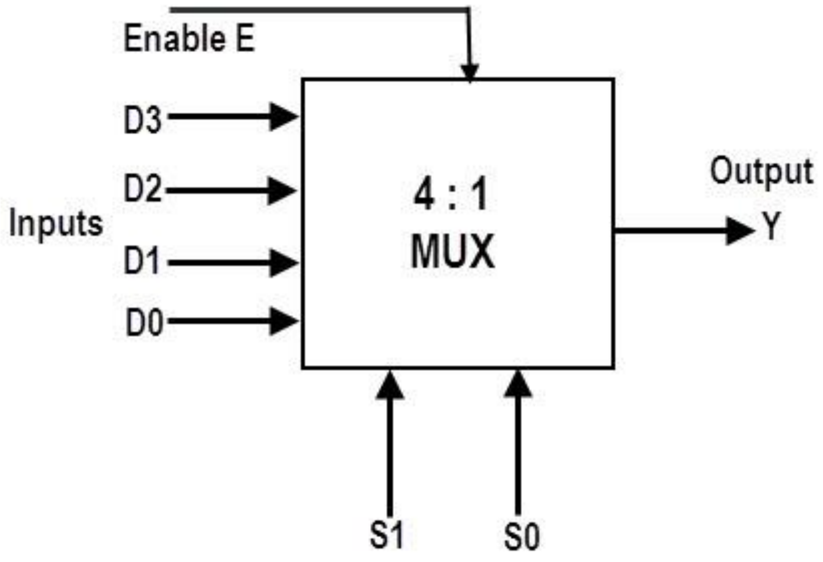
Study of IC 74153



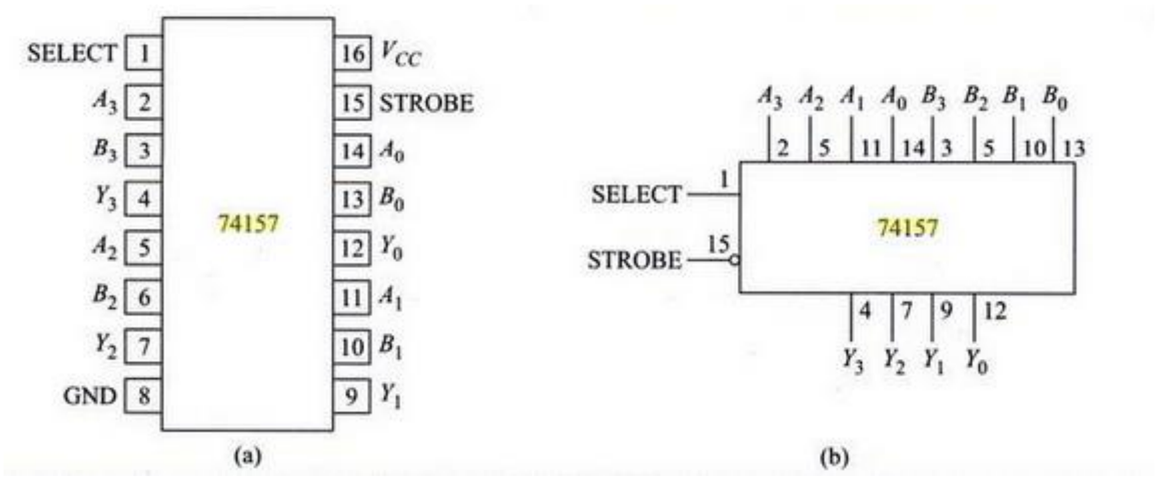
D	C	B	A	Q	D_i
0	0	0	0	1	$1D_0$
0	0	0	1	0	$1D_1$
0	0	1	0	1	$1D_2$
0	0	1	1	0	$1D_3$
0	1	0	0	0	$1D_0$
0	1	0	1	0	$1D_1$
0	1	1	0	1	$1D_2$
0	1	1	1	1	$1D_3$
1	0	0	0	0	$2D_0$
1	0	0	1	0	$2D_1$
1	0	1	0	0	$2D_2$
1	0	1	1	1	$2D_3$
1	1	0	0	0	$2D_0$
1	1	0	1	1	$2D_1$
1	1	1	0	1	$2D_2$
1	1	1	1	0	$2D_3$

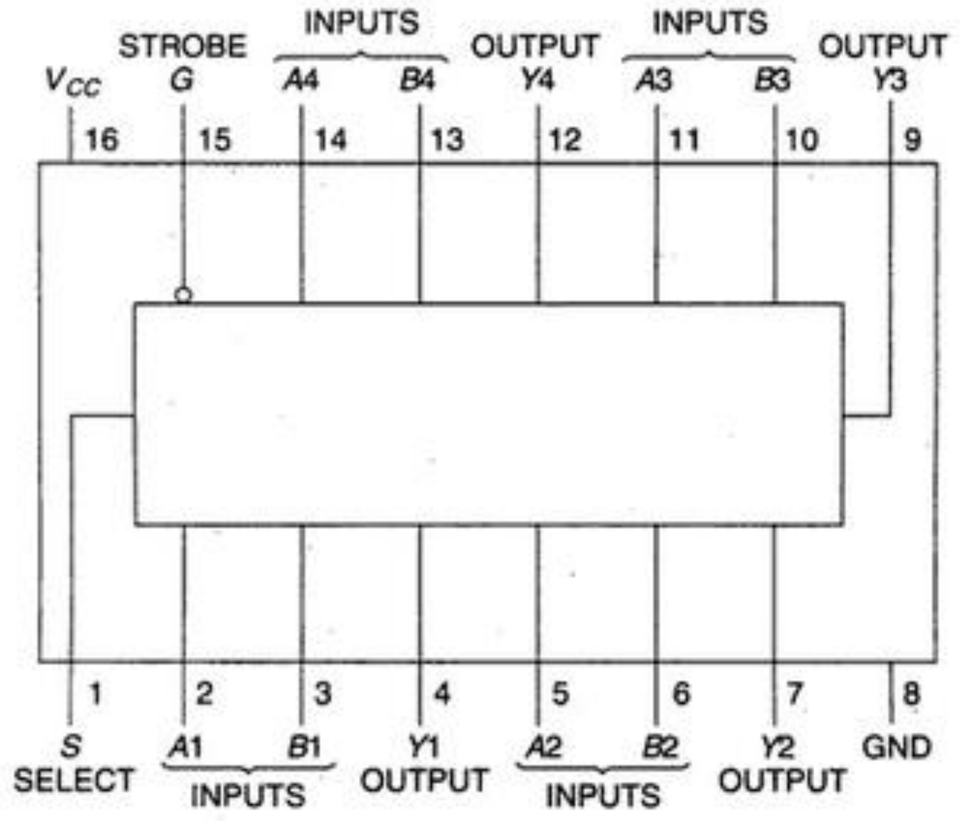
Design and implement 4:1 multiplexer.

Study of IC 74157

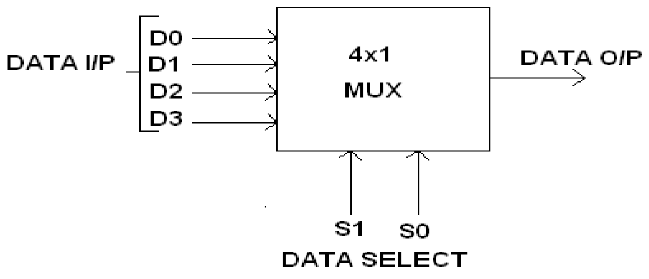


Select Data Inputs		Output
S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3


SELECT 1
A_3 2
B_3 3
Y_3 4
A_2 5
B_2 6
Y_2 7
GND 8
16 V_{CC}
15 STROBE
14 A_0
13 B_0
12 Y_0
11 A_1
10 B_1
9 Y_1
74157
(a)
A_3 A_2 A_1 A_0 B_3 B_2 B_1 B_0
2 5 11 14 3 5 10 13
SELECT 1
STROBE 15
74157
4 7 9 12
Y_3 Y_2 Y_1 Y_0
(b)


V_{CC}
STROBE G
INPUTS A4 B4
OUTPUT Y4
INPUTS A3 B3
OUTPUT Y3
16 15 14 13 12 11 10 9
1 2 3 4 5 6 7 8
S SELECT
A1 B1 INPUTS
Y1 OUTPUT
A2 B2 INPUTS
Y2 OUTPUT
GND

BLOCK DIAGRAM FOR 4:1 MULTIPLEXER:

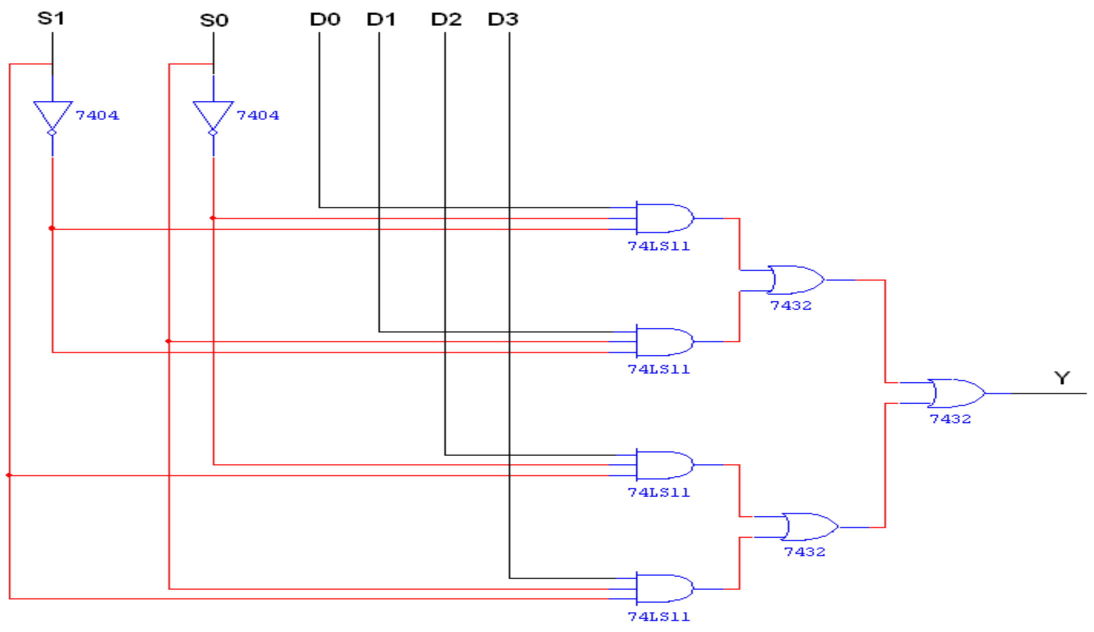


FUNCTION TABLE:

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

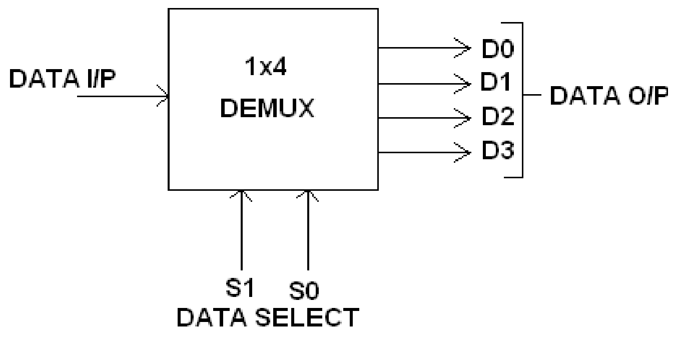
CIRCUIT DIAGRAM FOR MULTIPLEXER:



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:

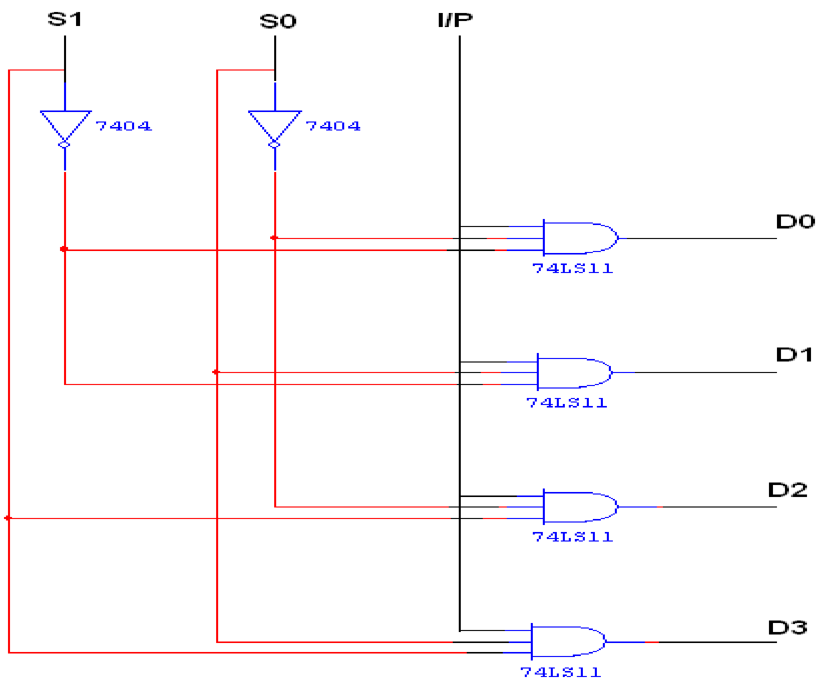
S1	S0	INPUT
0	**0**	**X → D0 = X S1’ S0’**
0	**1**	**X → D1 = X S1’ S0**
1	**0**	**X → D2 = X S1 S0’**
1	**1**	**X → D3 = X S1 S0**

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

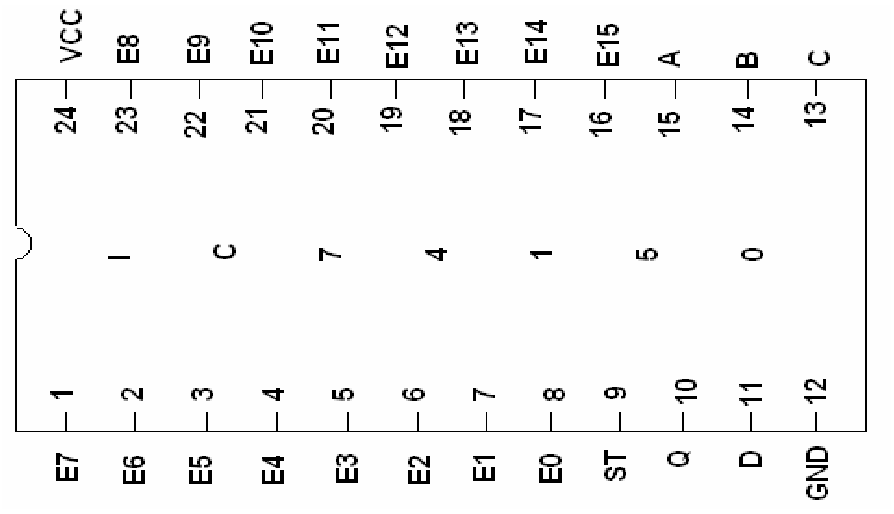
LOGIC DIAGRAM FOR DEMULTIPLEXER:



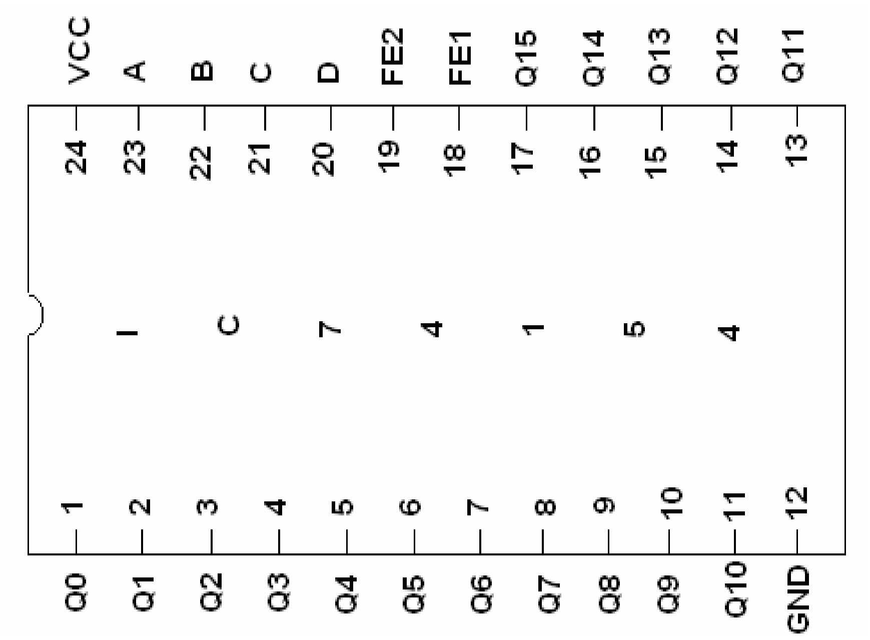
TRUTH TABLE:

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

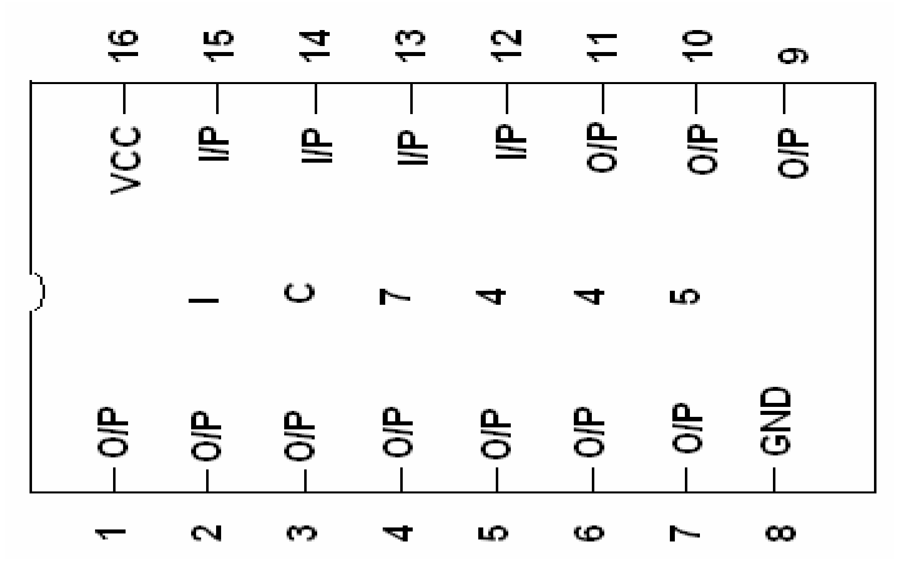
PIN DIAGRAM FOR IC 74150:



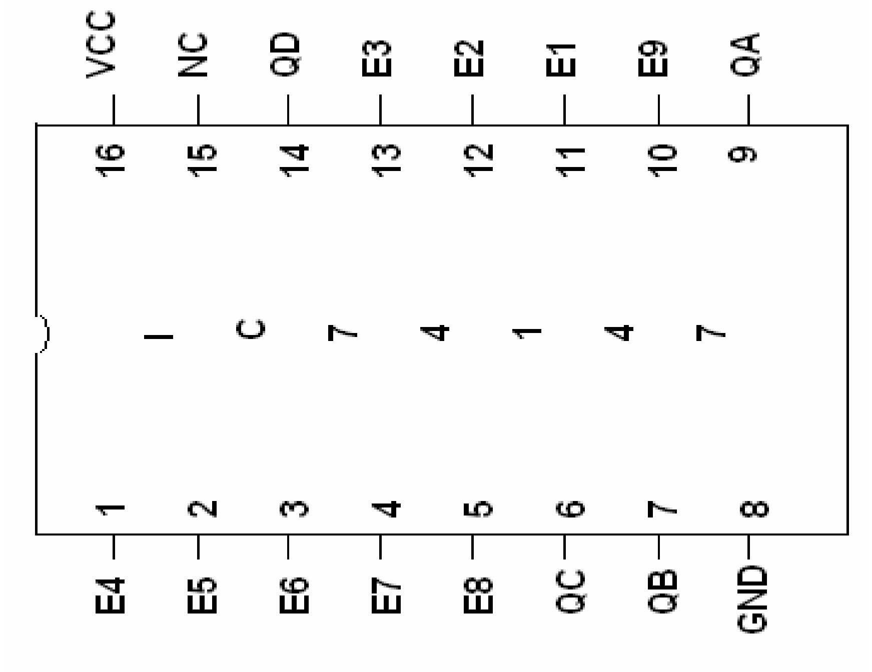
PIN DIAGRAM FOR IC 74154:



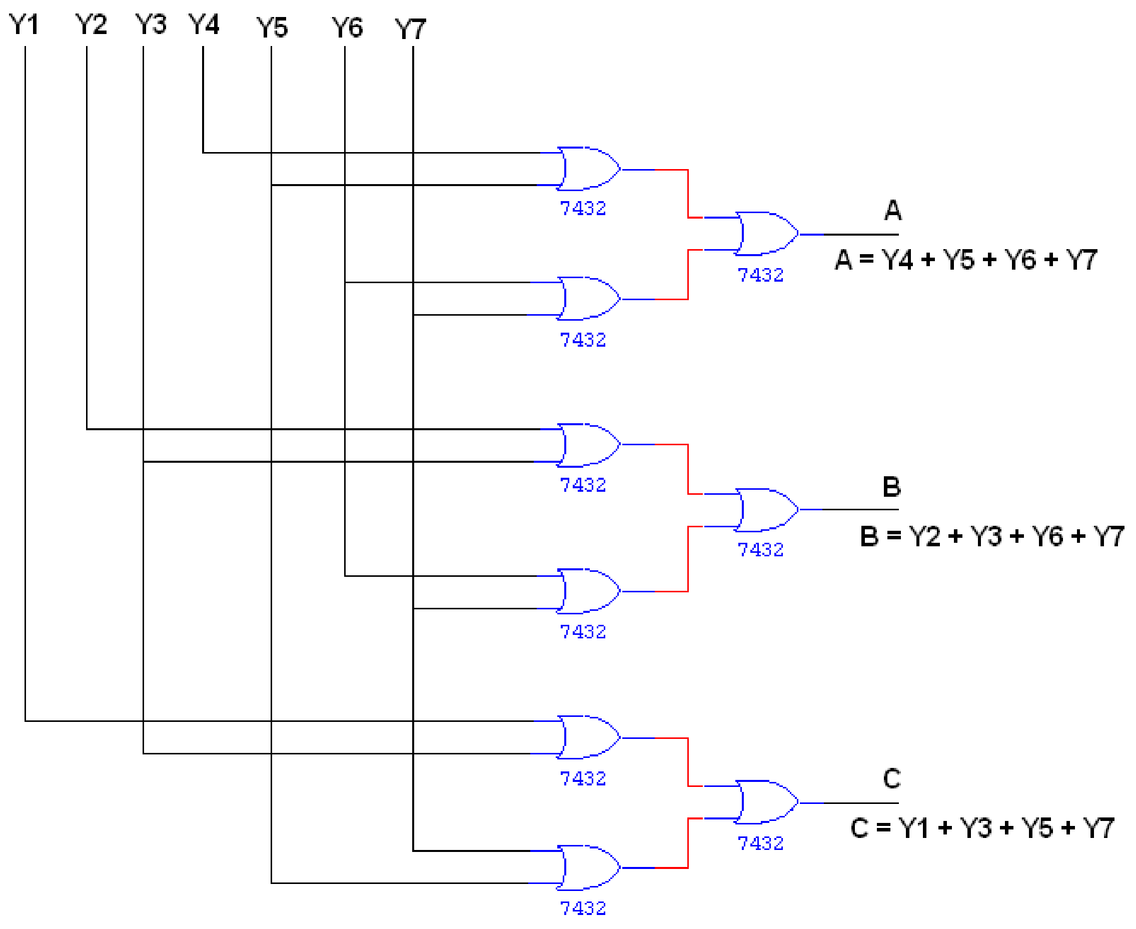
PIN DIAGRAM FOR IC 7445:
BCD TO DECIMAL DECODER:



PIN DIAGRAM FOR IC 74147:



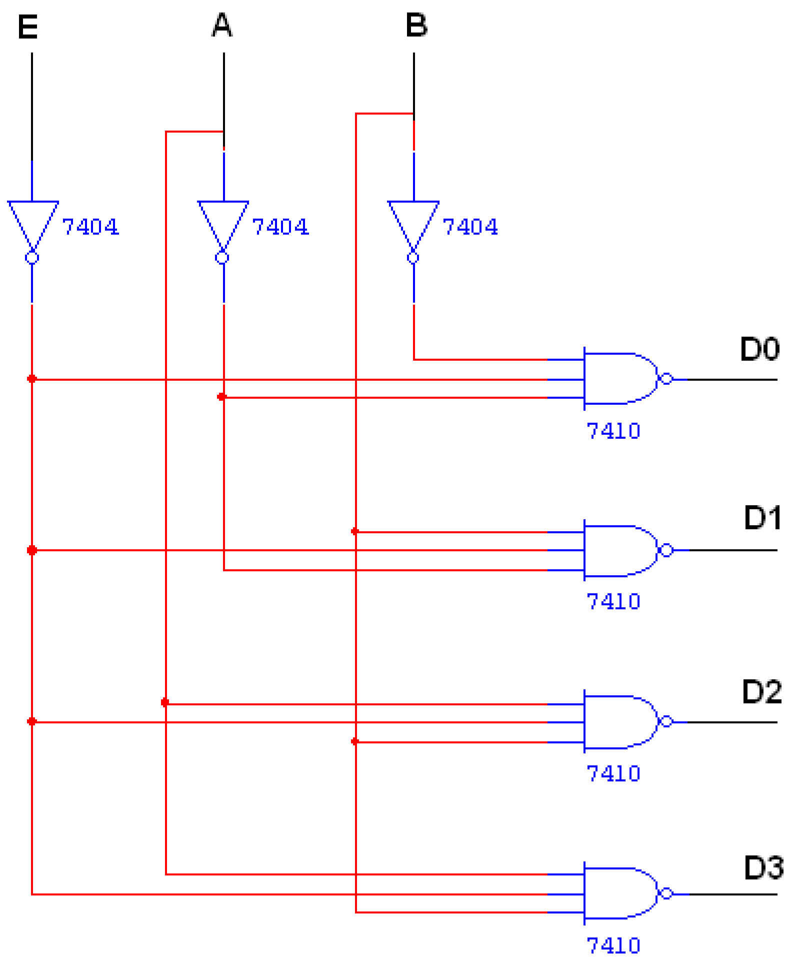
LOGIC DIAGRAM FOR ENCODER:



TRUTH TABLE:

INPUT							OUTPUT		
Y1	**Y2**	**Y3**	**Y4**	**Y5**	**Y6**	**Y7**	**A**	**B**	**C**
1	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	1
0	0	0	1	0	0	0	1	0	0
0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	1	1	1	1

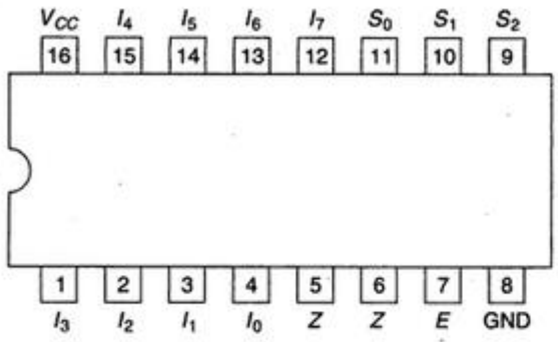
LOGIC DIAGRAM FOR DECODER:

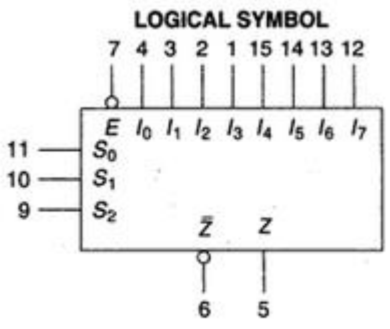


TRUTH TABLE:

INPUT			OUTPUT			
E	**A**	**B**	**D0**	**D1**	**D2**	**D3**
1	**0**	**0**	**1**	**1**	**1**	**1**
0	**0**	**0**	**0**	**1**	**1**	**1**
0	**0**	**1**	**1**	**0**	**1**	**1**
0	**1**	**0**	**1**	**1**	**0**	**1**
0	**1**	**1**	**1**	**1**	**1**	**0**

Implement the given expression using IC 74151 8:1 multiplexer

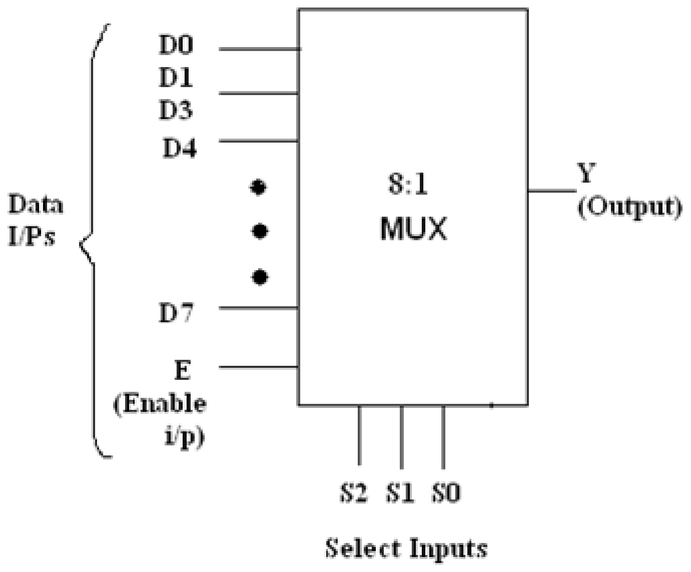


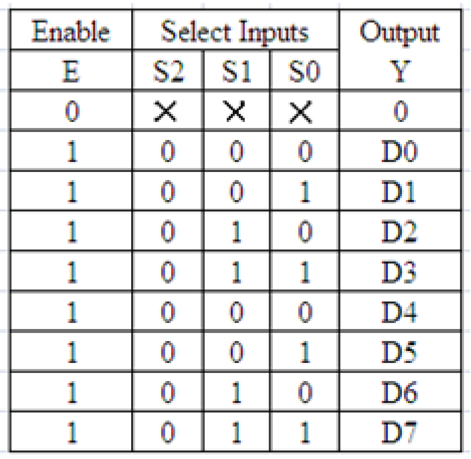


(b)

PIN NAMES

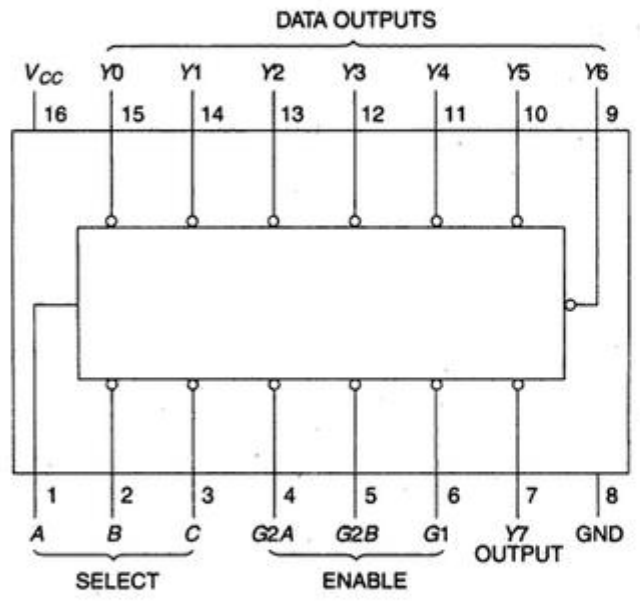
$S_0 - S_2$	Select inputs
$\bar{E}$	Enable (Active LOW) inputs
$I_0 - I_7$	Multiplexer inputs
Z	Multiplexer outputs (note *b*)
$\bar{Z}$	Complementary multiplexer output

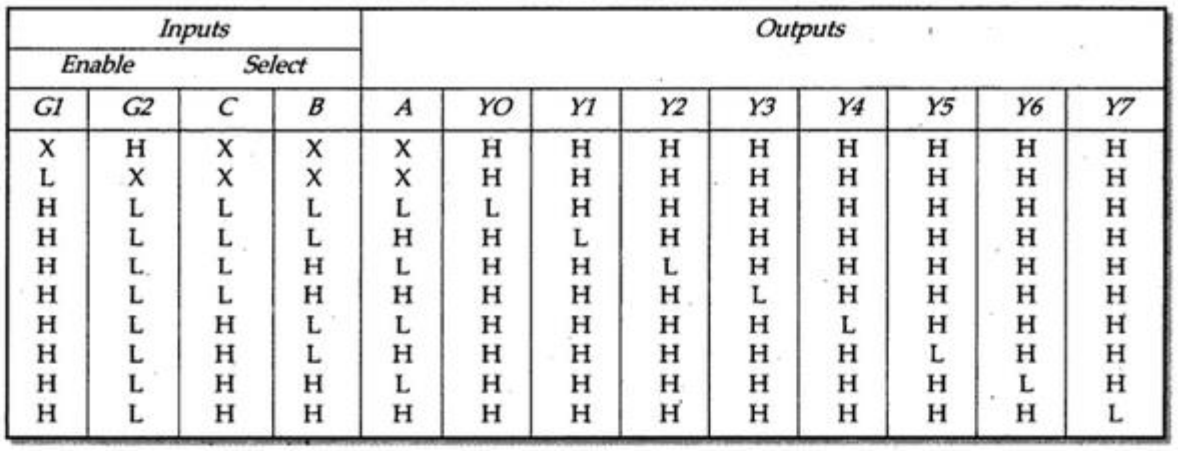




Enable	Select Inputs			Output
E	S2	S1	S0	Y
0	×	×	×	0
1	0	0	0	D0
1	0	0	1	D1
1	0	1	0	D2
1	0	1	1	D3
1	0	0	0	D4
1	0	0	1	D5
1	0	1	0	D6
1	0	1	1	D7

Implement the given expression using IC 74138 3:8 decoder





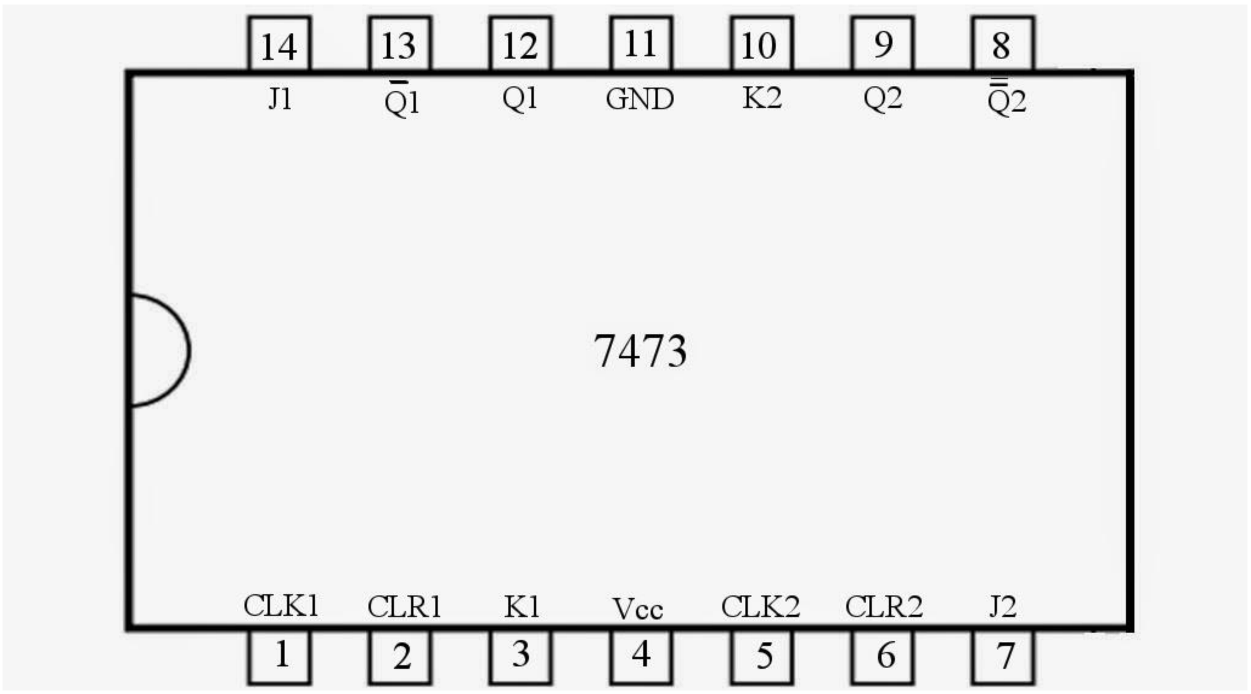
Inputs					Outputs							
Enable		Select										
G1	G2	C	B	A	YO	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H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

PRACTICAL-8
Study of flip-flops and counters

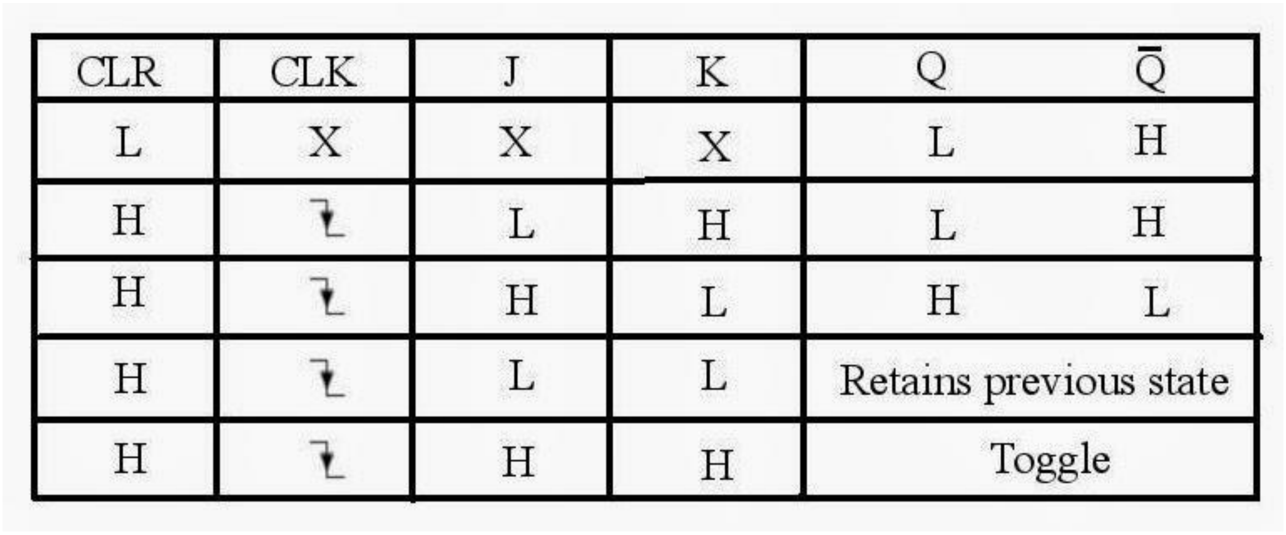
1. Study of IC 7473.
2. Study of IC 7474.
3. Study of IC 7476.
4. Conversion of Flip-flops.
5. Design of 3-bit synchronous counter using 7473 and required gates.
6. Design of 3-bit ripple counter using IC 7473.

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.



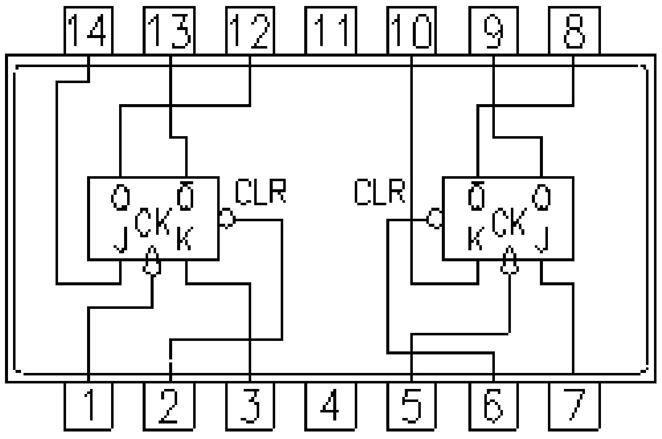
TRUTH TABLE



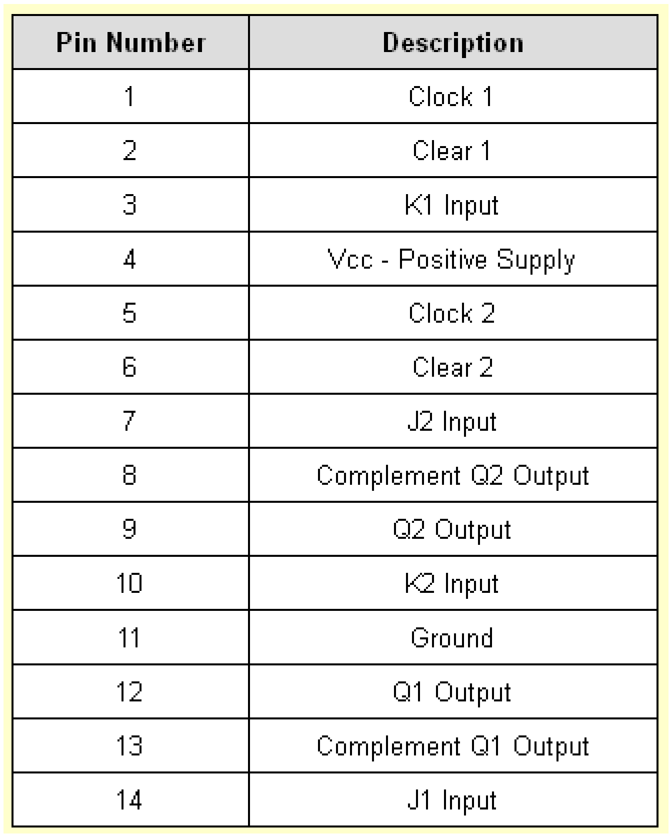
CLR	CLK	J	K	Q	$\overline{Q}$
L	X	X	X	L	H
H	↓	L	H	L	H
H	↓	H	L	H	L
H	↓	L	L	Retains previous state	
H	↓	H	H	Toggle	

Study of IC 7473

7473 Dual JK Flip-Flop with Clear
Two JK Flip-Flops with Clear



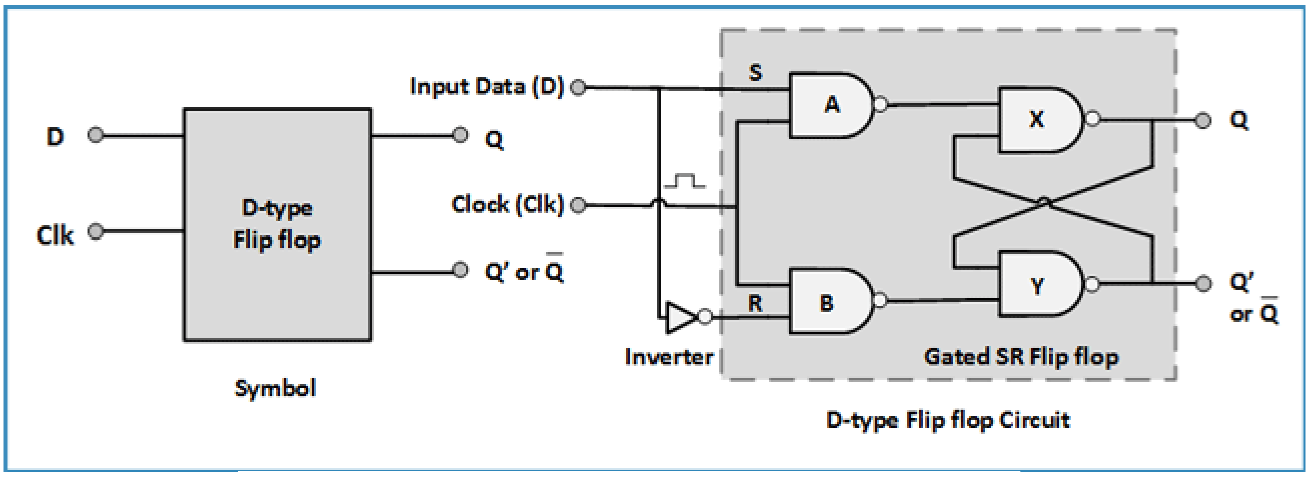
7473
Dual J-K M/S Flip-Flop
with Clear

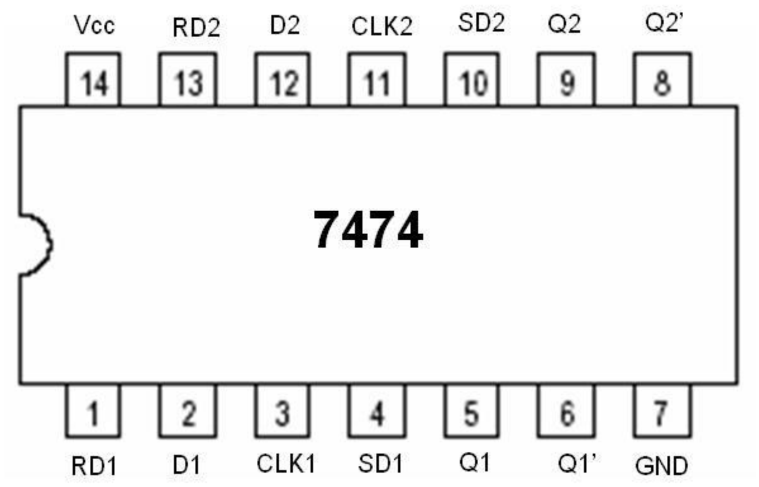


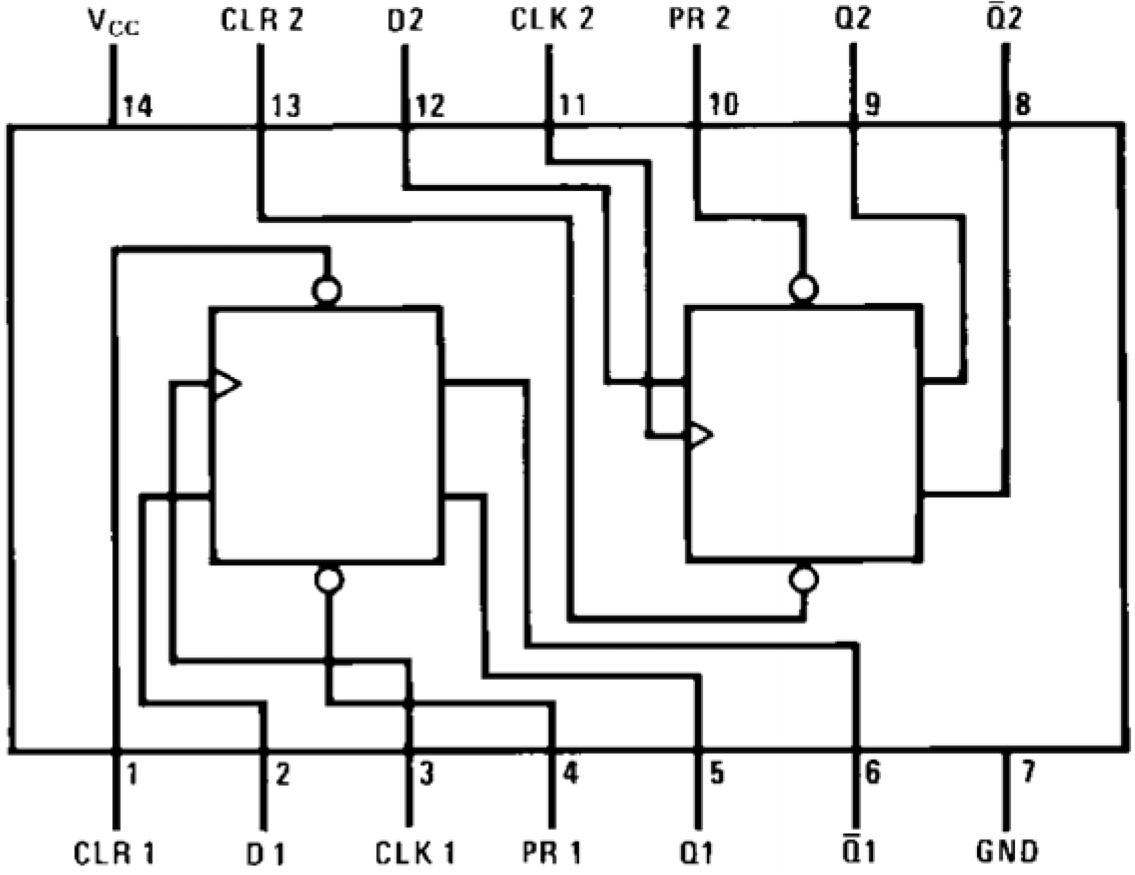
Pin Number	Description
1	Clock 1
2	Clear 1
3	K1 Input
4	Vcc - Positive Supply
5	Clock 2
6	Clear 2
7	J2 Input
8	Complement Q2 Output
9	Q2 Output
10	K2 Input
11	Ground
12	Q1 Output
13	Complement Q1 Output
14	J1 Input

Study of IC 7474.

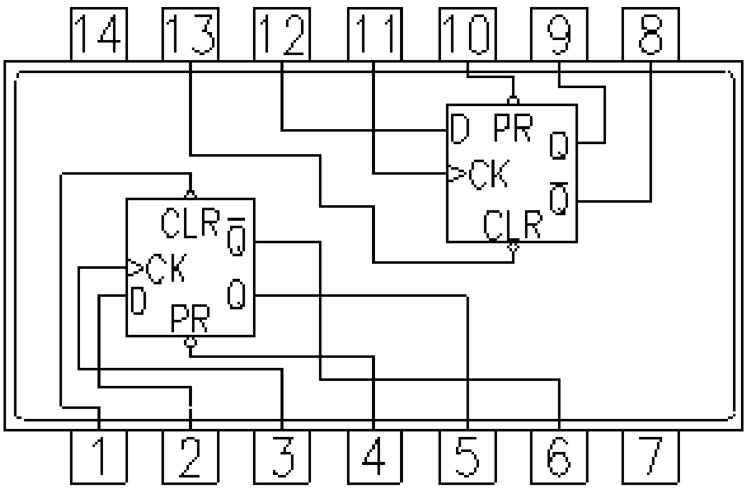
IC DM74S74N is the Dual D-type Flip-flop IC, in which there are two D-type Flip-flops, which can be either used individually or as a master-slave toggle combination
Pins for first D flip-flop are the left side and for second flip flop are at right side. Also there are PRE and CLR pins for both the D-type 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.






V_{CC}
CLR 2
D2
CLK 2
PR 2
Q2
$\bar{Q}2$
14
13
12
11
10
9
8
1
2
3
4
5
6
7
CLR 1
D 1
CLK 1
PR 1
Q1
$\bar{Q}1$
GND

7474 Dual D-Type Flip-Flop (Two D-Type Flip-Flops with Preset and Clear)



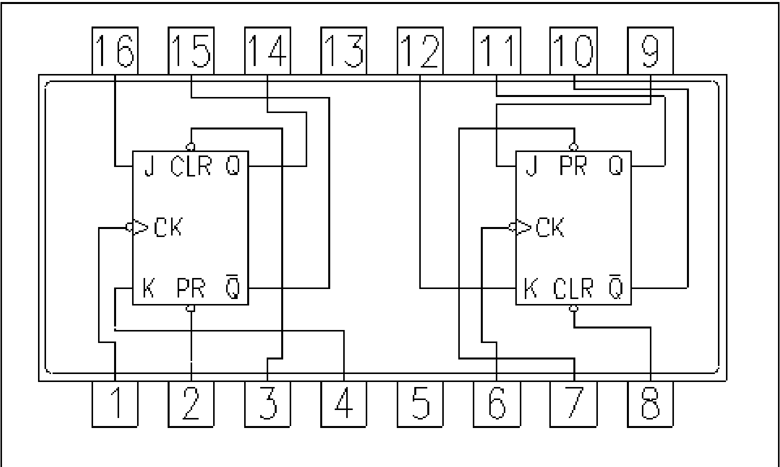
7474
Dual D Flip-Flop
with Preset and Clear

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



7476
Dual J/K M/S Flip-Flop
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.

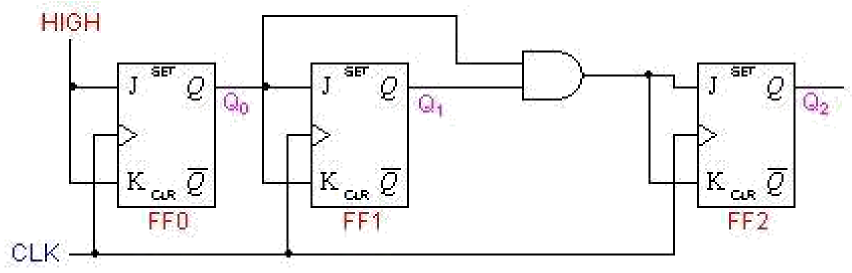


Figure 8.1 Logic diagram of 3-bit Synchronous counter

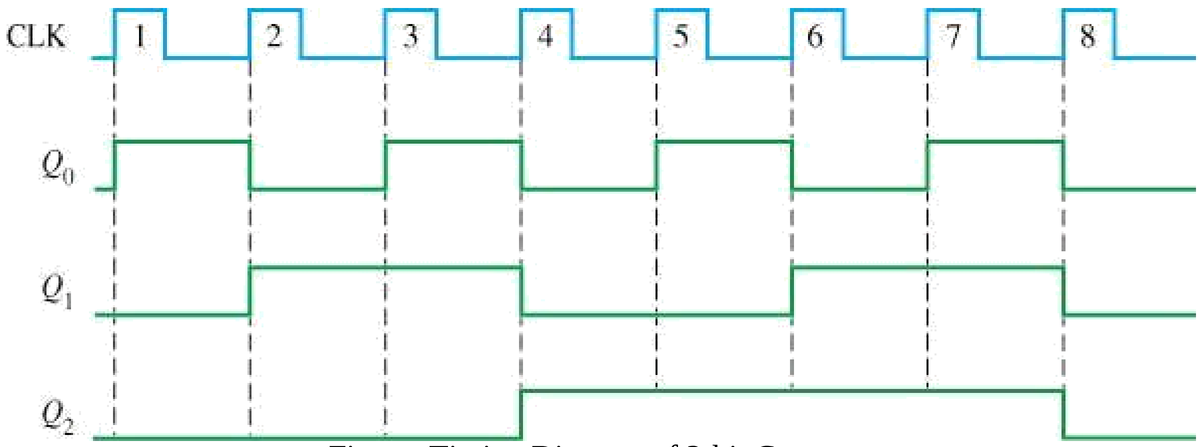
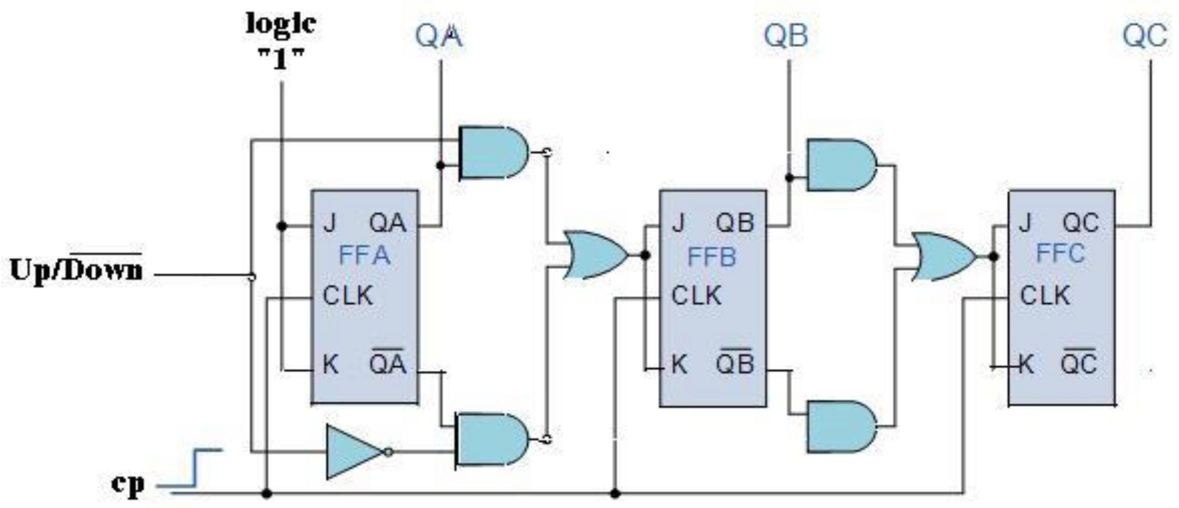


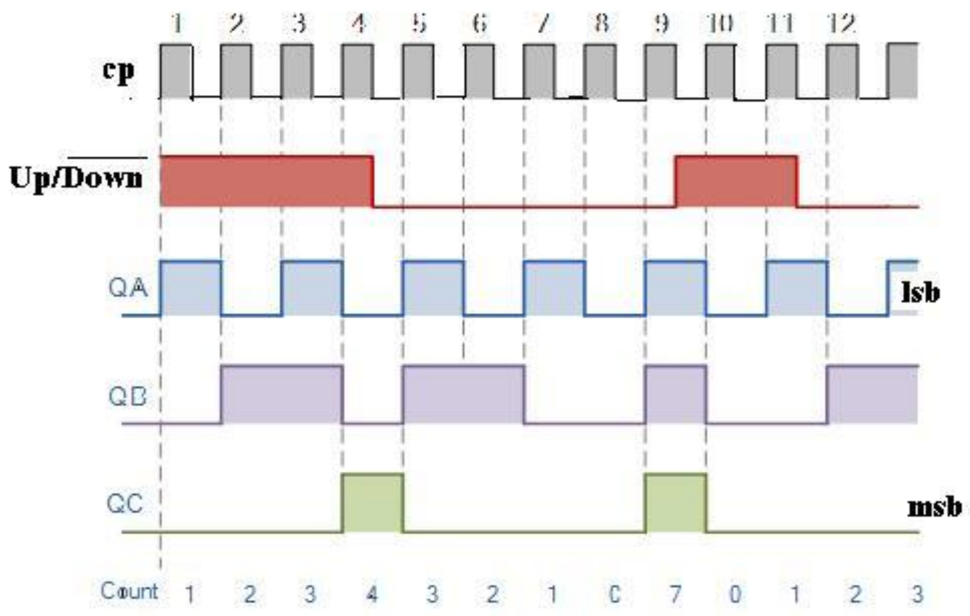
Figure: Timing Diagram of 3-bit Counter

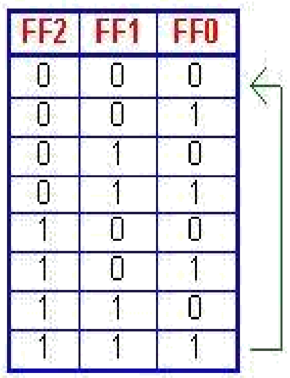
The most important advantage of synchronous counters is that there is no cumulative time delay because all flip -flops are triggered in parallel. Thus, the maximum operating frequency for this counter will be significantly higher than for the corresponding ripple counter.

Lab Procedure

1. Construct the logic circuit as shown in figure.
2. Use the up/(down)’ input to choose up counter or down counter.
3. Verify the count sequence as given in figure.


logic "1"
QA
QB
QC
Up/Down
cp
J QA FFA CLK K QA
J QB FFB CLK K QB
J QC FFC CLK K QC


1 2 3 4 5 6 7 8 9 10 11 12
cp
Up/Down
QA
lsb
QB
QC
msb
Count 1 2 3 4 3 2 1 0 7 0 1 2 3



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

Count Sequence

PreLab questions

1. How does synchronous counter differ from asynchronous counter?
2. A 4-bit up/down binary counter is in the DOWN mode and in the 1010 state. On the next clock pulse, to what state does the counter go?
3. How many flip-flops do you require to design Mod-7 counter.
4. Give the Transition table and excitation table of JK Flip flop.

Result

Thus the 3-bit synchronous up/down counter is designed and verified.

PostLab questions

1. Draw the state Diagram, state table and Timing Diagram of a 2-bit synchronous counter.
2. Deign a 3-bit Up/Down Gray Code Counter using D Flip-flop
3. Design a 11011 sequence detector using JK flip-flops. Allow overlap.
4. What is decade Counter?

Aim

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

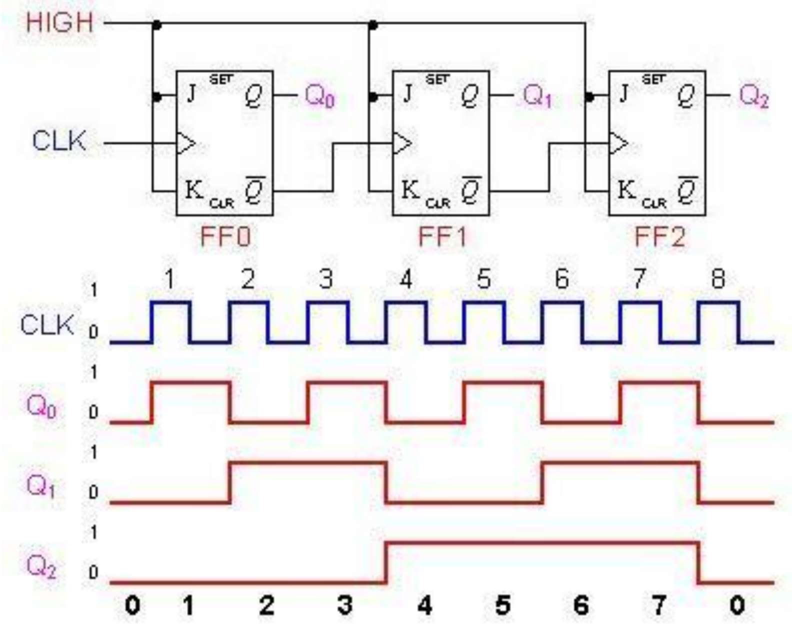
Asynchronous Counter is sequential circuit that is used to count the number of clock input signal. The output of one flip flop is given as a clock input to another flip-flop, so it is called as Serial Counter.

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.

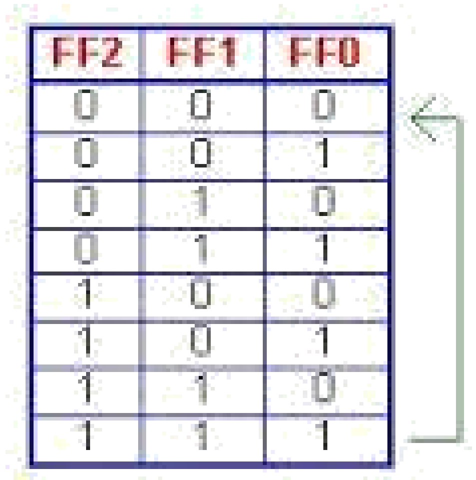
The MOD of the ripple counter or asynchronous counter is 2^n if n flip-flops are used. A three-bit asynchronous counter is shown on the below figure . The external clock is connected to the clock input of the first flip-flop (FF0) only. So, FF0 changes state at the falling edge of each clock pulse, but 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

Counting Sequence

Prelab questions

1. What do you mean by Glitch?
2. How many flip-flops are required to produce a divide-by-64 device?
3. Why Asynchronous counter is called as Ripple Counter?
4. What do you mean by synchronous reset and asynchronous reset?
5. What is the use of Preset input?
6. What is use of Ring and Johnson's Counter?

Lab Procedure

1. Construct the logic circuit as shown in Figure
2. Verify the count sequence as given in figure

Result:

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

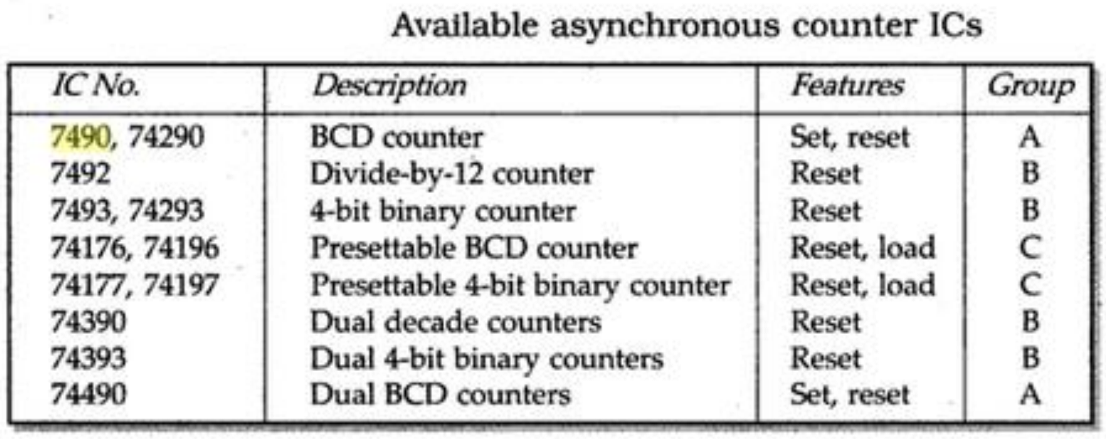
Postlab questions

1. Draw the logic diagram of Mod 12 Asynchronous Counter and its timing diagram.
2. Design a 4-bit frequency divider.
3. Design a sequential circuit that is used to generate the timing signals with a combination of Shift register and a decoder.
4. What is state table?

PRACTICAL-9

Study of counter ICs and designing Mod-N counters

1. Study of IC 7490, 7492, 7493 and designing mod-n counters using these.
2. Designing mod-n counters using IC 7473 and 7400 (NAND gates)



Available asynchronous counter ICs

IC No.	Description	Features	Group
7490, 74290	BCD counter	Set, reset	A
7492	Divide-by-12 counter	Reset	B
7493, 74293	4-bit binary counter	Reset	B
74176, 74196	Presettable BCD counter	Reset, load	C
74177, 74197	Presettable 4-bit binary counter	Reset, load	C
74390	Dual decade counters	Reset	B
74393	Dual 4-bit binary counters	Reset	B
74490	Dual BCD counters	Set, reset	A

9.1 Aim

The purpose of this experiment is to introduce the design of Mod-N Counter and to implement it using suitable Flip-flops.

9.2 Hardware Requirement

Equipment : Bread Board, Power Supply, Resistors, LEDs or Digital IC Trainer Kit
Discrete Components : IC 7473 Dual JK Flip Flop
IC 7400 NAND Gate

9.3 Theory:

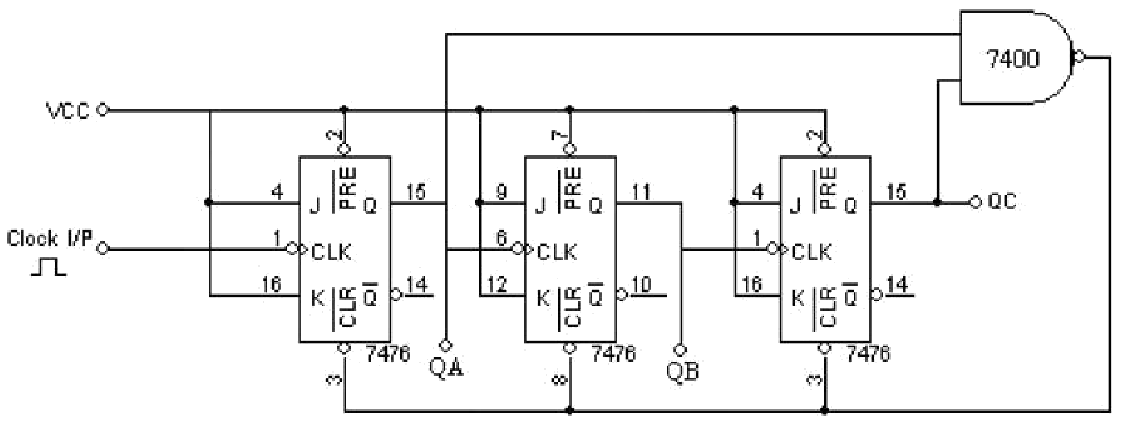
Circuits for counting events are frequently used in computers and other digital systems. Since a counter circuit must remember its past states, it has to possess memory. The number of flip flops used and how they are connected determine the number of states and the sequence of the states that the counter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

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

1. 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.
2. Synchronous Counters - all memory elements are simultaneously triggered by the same clock.

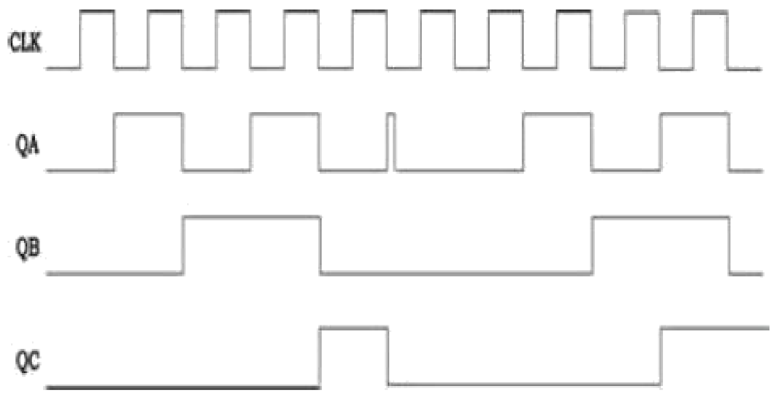
A mod N counter is a counter that has N states. Its output frequency is f/N. A counter which is reset at the fifth clock pulse is called Mod 5 counter or Divide by 5 counter. The circuit diagram of Mod 5 counter is shown in the figure. This counter contains three JKMS flip-flop.

Mod 5 Asynchronous Counter:-



A 3 bit binary counter is normally counting from 000 to 111. The actual output of a 3 bit binary counter at the fifth clock pulse is 101. A two input NAND gate is used to make a Mod 5 counter.
The outputs of the first and third flip flops (QA and QC) are connected to the input of the give NAND gate, and its output is connected to the RESET terminal of the counter, Hence the counter is reset at the fifth clock pulse, which produces the output QC,QB,QA as 000. It is called divide by 5th counter or mod 5 counter.

Mod 5 Asynchronous counter			
Clock	QC	QB	QA
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	0	0	0



9.4 Lab Procedure

1. Connections are made as per circuit diagram.
2. Clock pulses are applied one by one at the clock I/P and the O/P is observed at QA, QB & QC for IC 7476.
3. Truth table is verified.

9.5 Prelab questions

1. Which flipflop is suitable for counter? Why?
2. Draw the timing diagrams for mod 6 counter.

9.6 Result

Thus the Mod-5 counter is designed and verified.

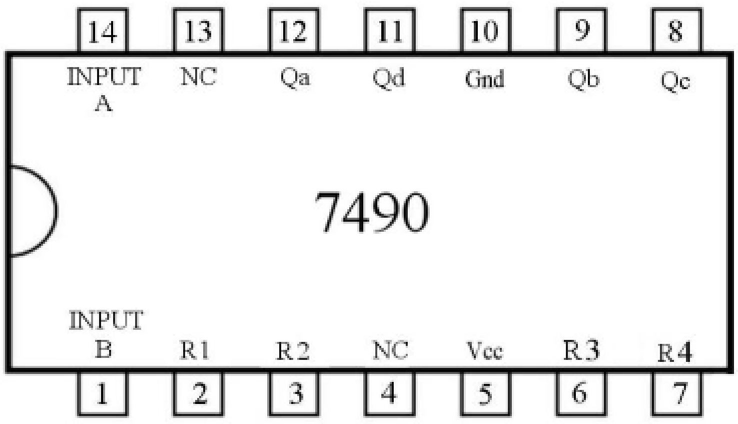
9.7 PostLab questions

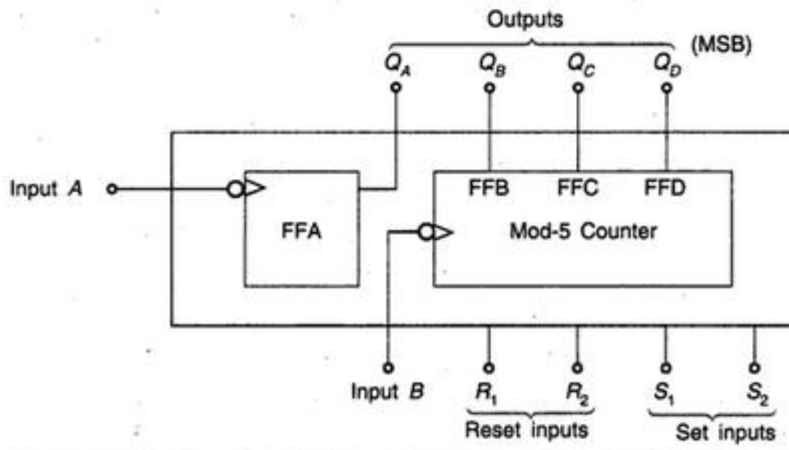
1. Draw the state Diagram, state table and Timing Diagram of a 2-bit synchronous counter.
2. Design a modulus seven synchronous counter that can count 0, 3, 5, 7, 9, 11, and 12 using D flip-flop.

Study of IC 7490

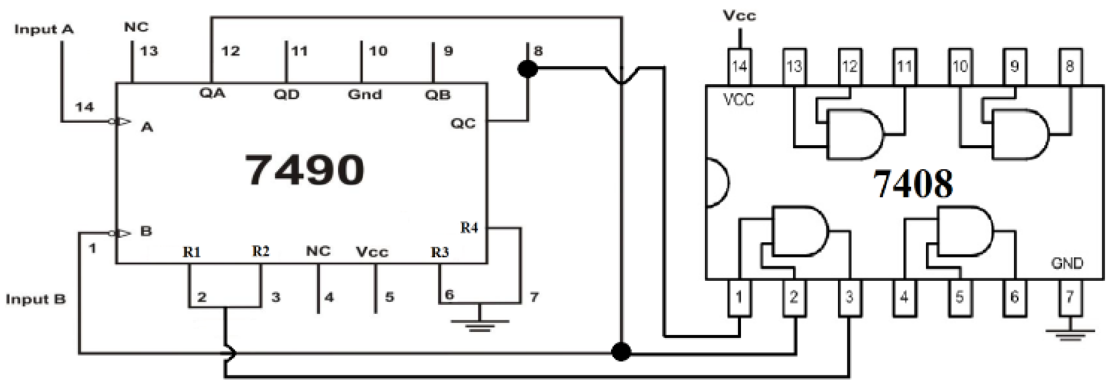
- The 74LS90 is a simple counter, it can count from 0 to 9 cyclically in its natural mode.
- It counts the input pulses and the output is received as a 4 bit binary number through pins

Q_A, Q_B, Q_C and Q_D.

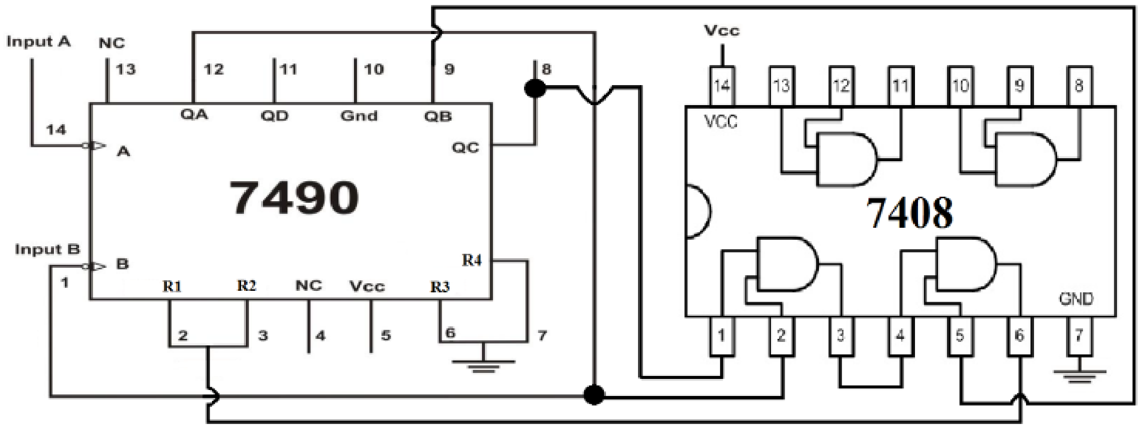




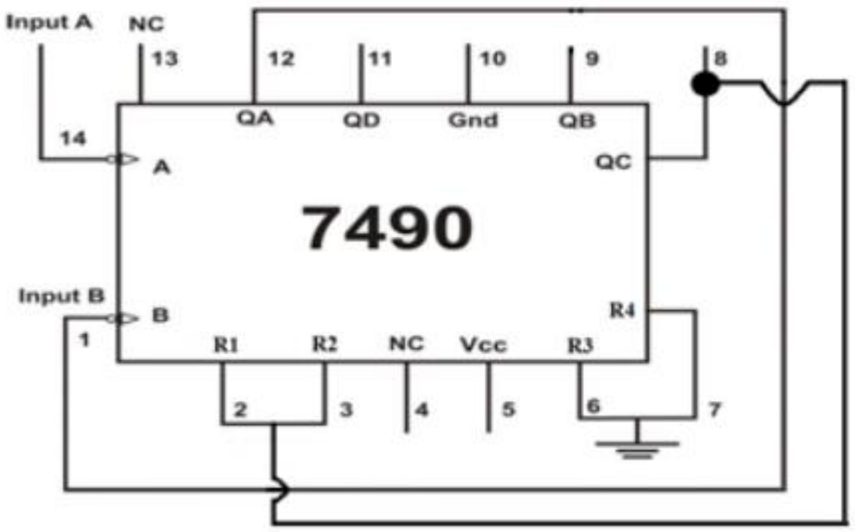
Counter state	FLIP-FLOP outputs			
	Q_D	Q_C	Q_B	Q_A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	0	0	0	0



Circuit for Mod 5 Counter

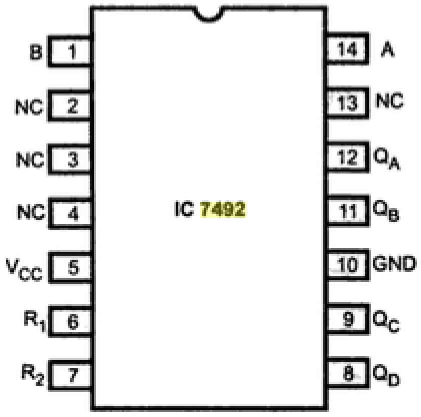


Circuit for Mod 7 Counter

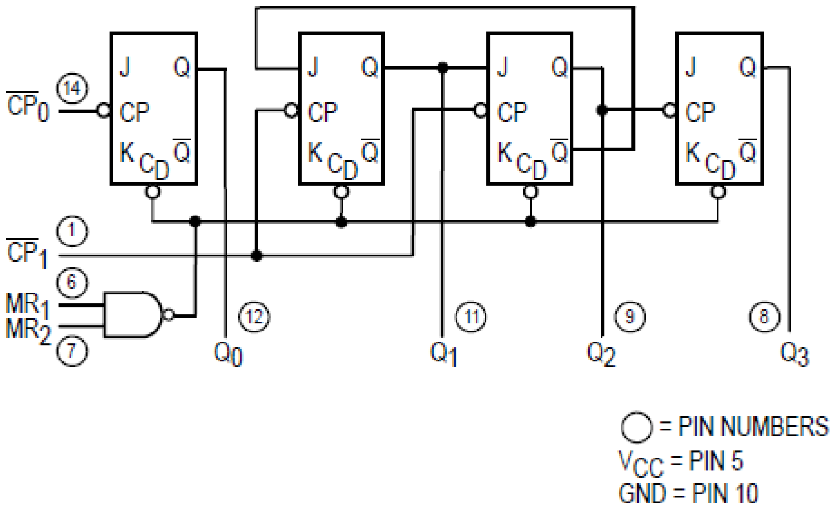


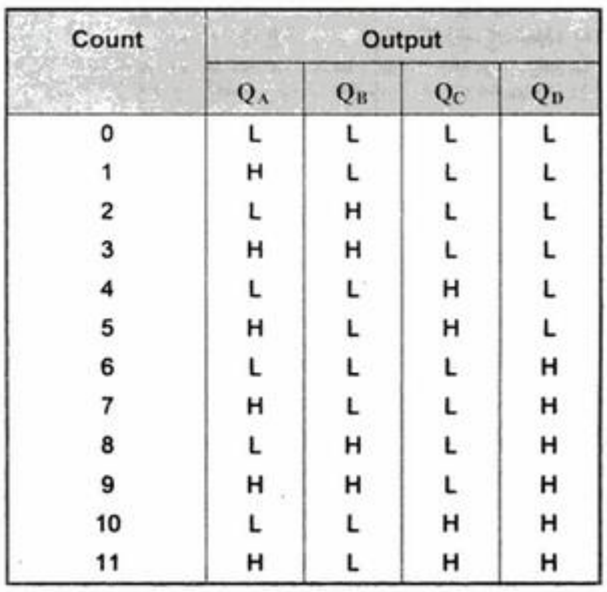
Circuit for Mod 4 Counter

Study of IC 7492



74LS92 (Divide by 12 Counter)

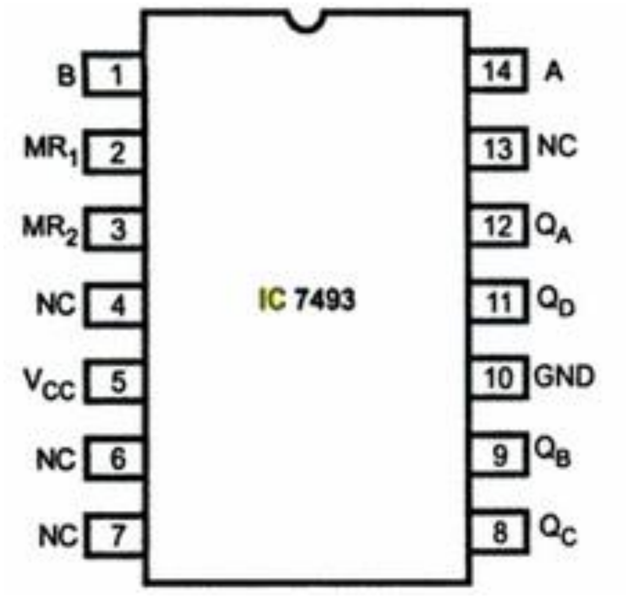


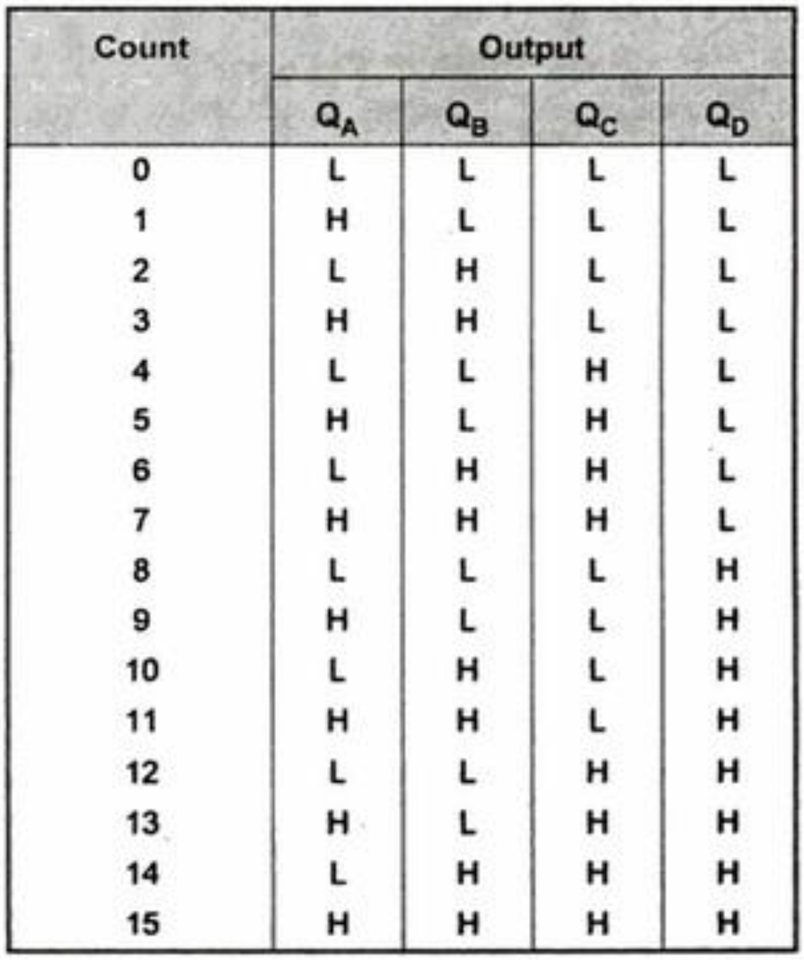


Count	Output			
	Q_A	Q_B	Q_C	Q_D
0	L	L	L	L
1	H	L	L	L
2	L	H	L	L
3	H	H	L	L
4	L	L	H	L
5	H	L	H	L
6	L	L	L	H
7	H	L	L	H
8	L	H	L	H
9	H	H	L	H
10	L	L	H	H
11	H	L	H	H

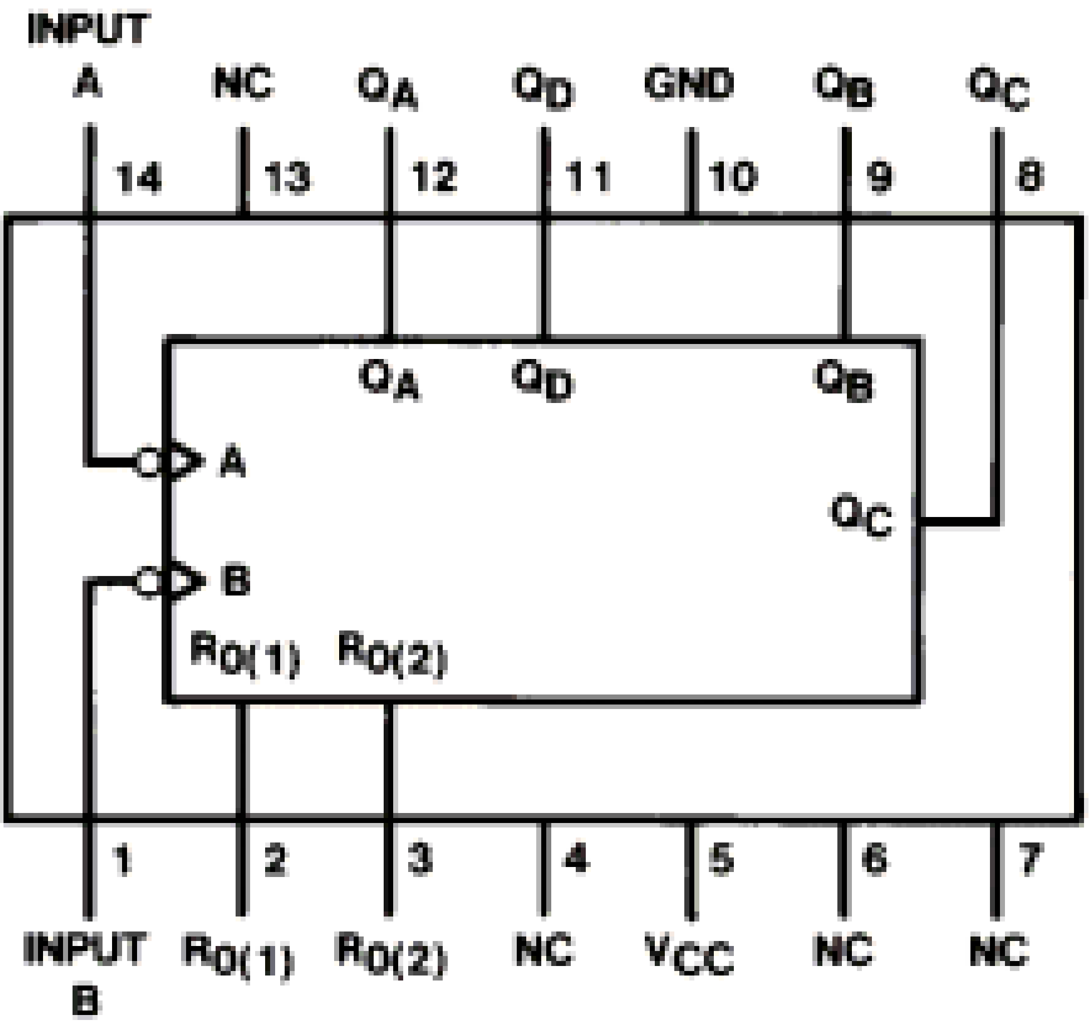
Study of IC 7493

4-bit BINARY COUNTER





Count	Output			
	Q_A	Q_B	Q_C	Q_D
0	L	L	L	L
1	H	L	L	L
2	L	H	L	L
3	H	H	L	L
4	L	L	H	L
5	H	L	H	L
6	L	H	H	L
7	H	H	H	L
8	L	L	L	H
9	H	L	L	H
10	L	H	L	H
11	H	H	L	H
12	L	L	H	H
13	H	L	H	H
14	L	H	H	H
15	H	H	H	H



Order Number DM7493AN
See NS Package Number N14A

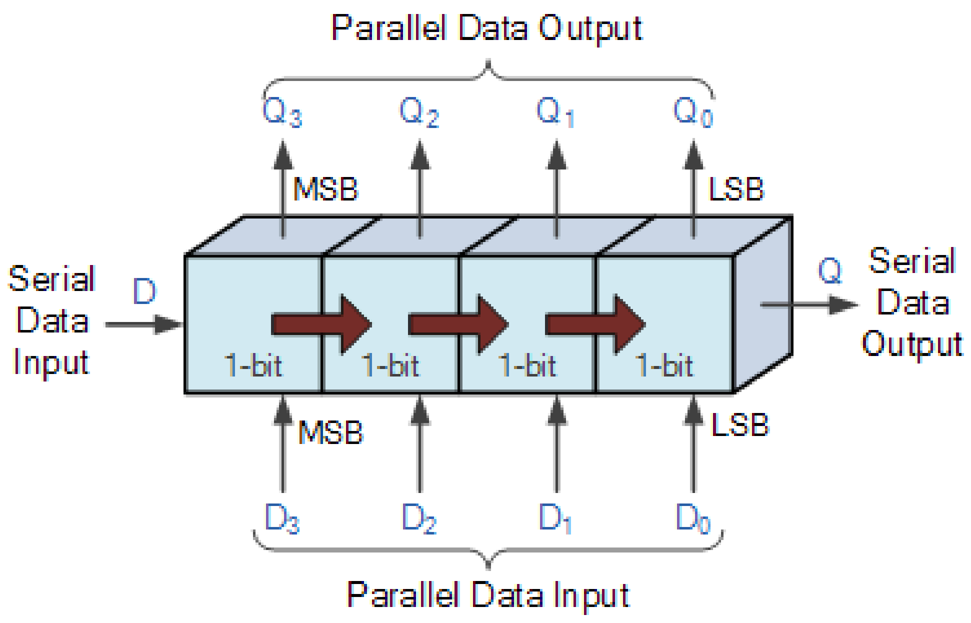
PRACTICAL-10

Design of shift registers and shift register counters

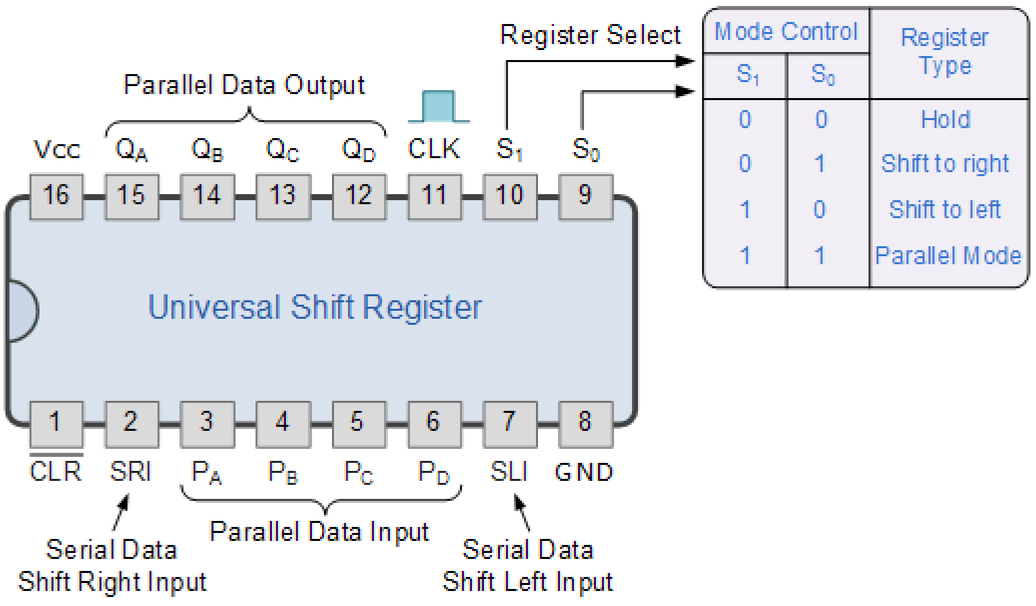
1. Design serial – in serial – out, serial – in parallel – out, parallel – in serial – out, parallel – in parallel – out and bidirectional shift registers using IC 7474
2. Study of IC 7495.
3. Implementation of digits using seven segment displays.

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”.



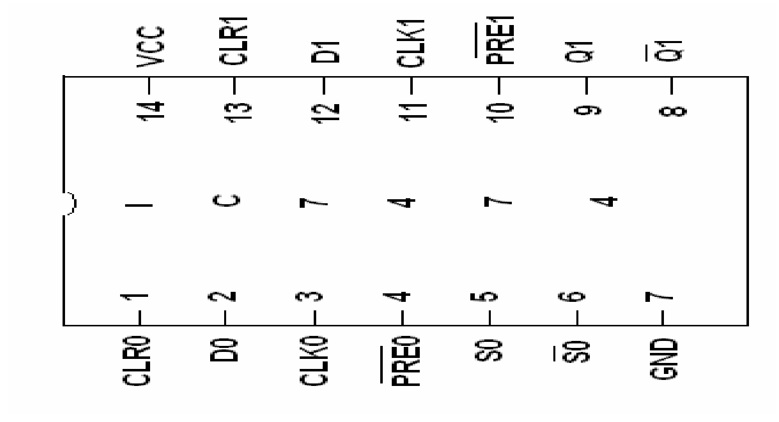
4-bit Universal Shift Register 74LS194



Mode Control		Register Type
S_1	S_0	
0	0	Hold
0	1	Shift to right
1	0	Shift to left
1	1	Parallel Mode

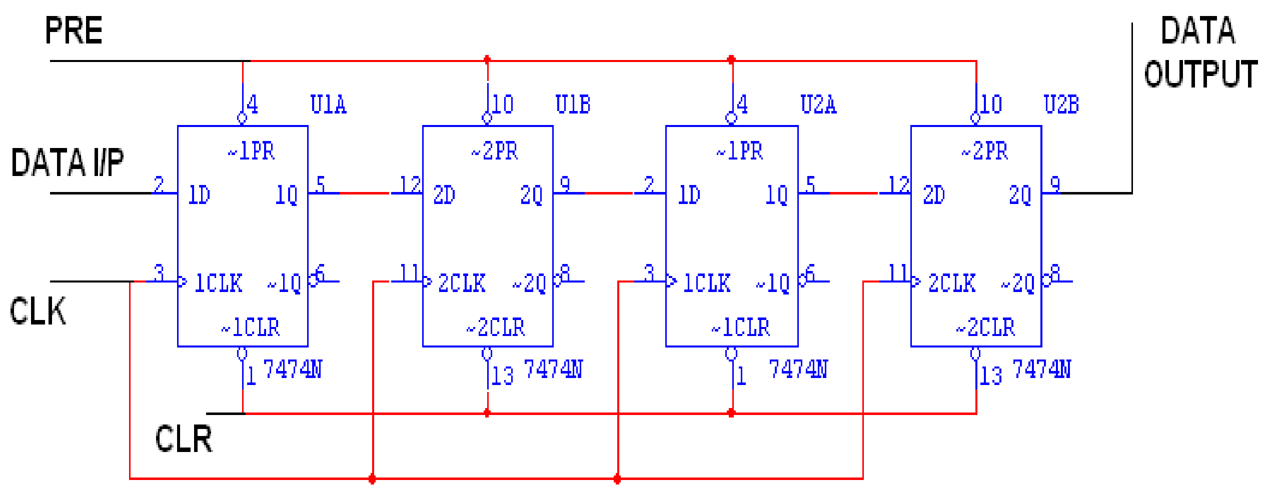
- Universal shift registers are very useful digital devices.
- They can be configured to respond to operations that require some form of temporary memory storage or for the delay of information such as the SISO or PIPO configuration modes or transfer data from one point to another in either a serial or parallel format.
- Universal shift registers are frequently used in arithmetic operations to shift data to the left or right for multiplication or division.

PIN DIAGRAM:



LOGIC DIAGRAM:

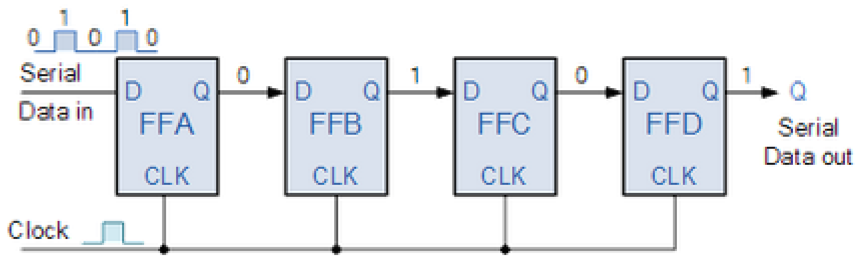
SERIAL IN SERIAL OUT:



TRUTH TABLE:

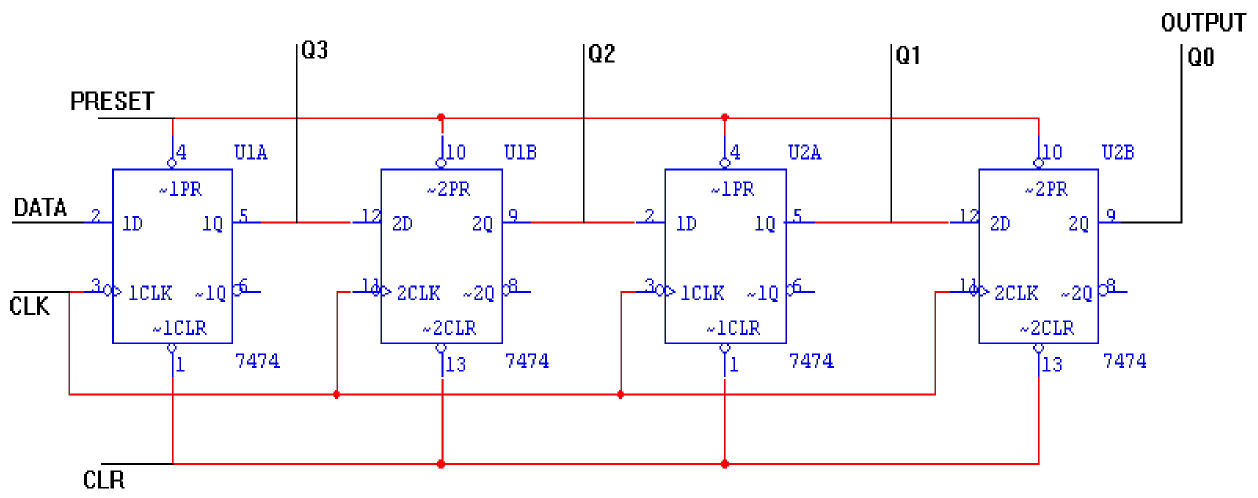
CLK	Serial in	Serial out
1	1	0
2	0	0
3	0	0
4	1	1
5	X	0
6	X	0
7	X	1

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



LOGIC DIAGRAM:

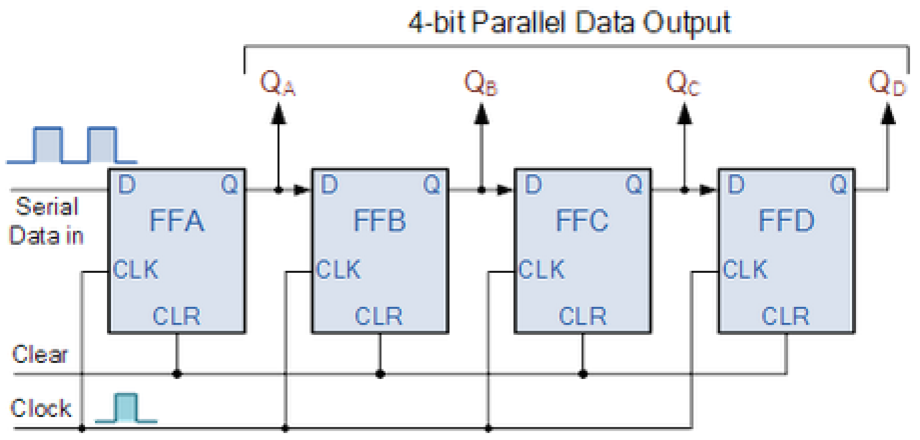
SERIAL IN PARALLEL OUT:



TRUTH TABLE:

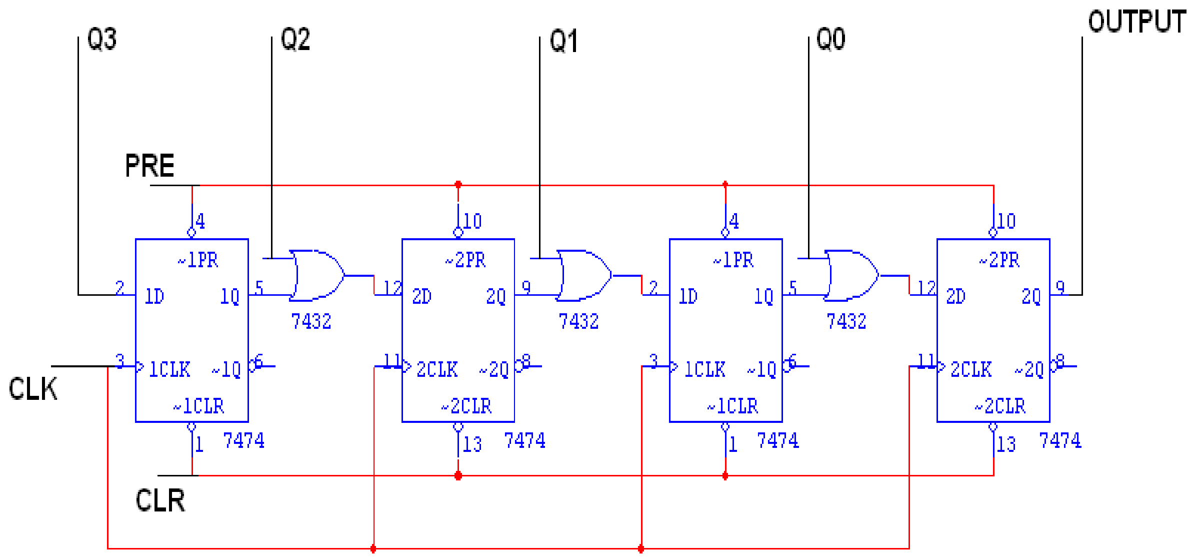
CLK	DATA	OUTPUT			
		Q_A	Q_B	Q_C	Q_D
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	0	0	1

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



LOGIC DIAGRAM:

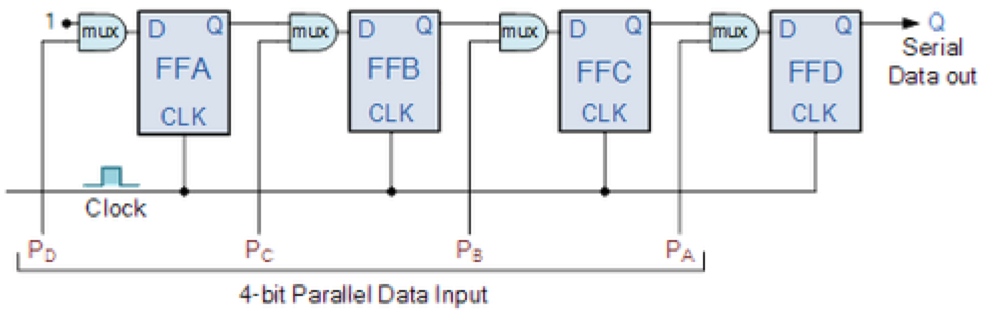
PARALLEL IN SERIAL OUT:



TRUTH TABLE:

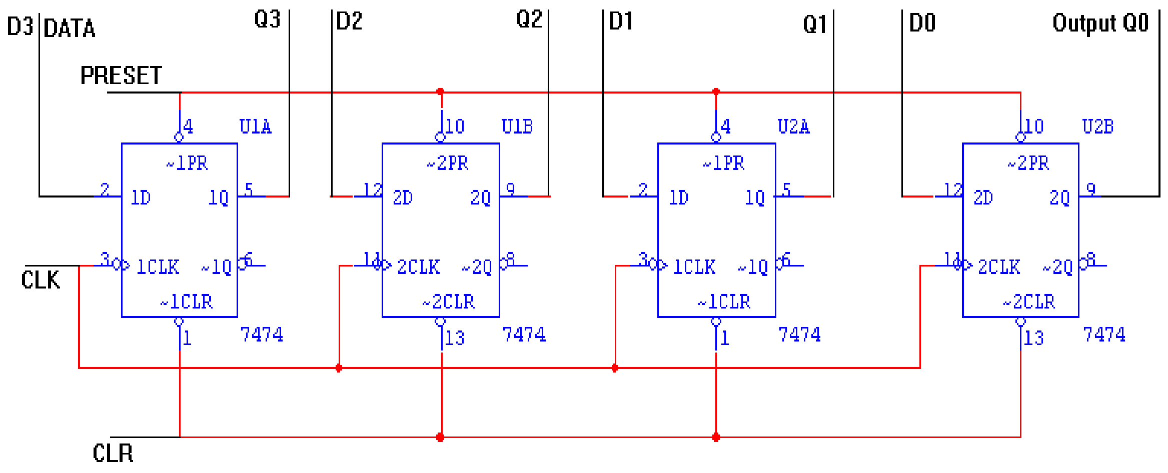
CLK	Q3	Q2	Q1	Q0	O/P
0	1	0	0	1	1
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	1

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



LOGIC DIAGRAM:

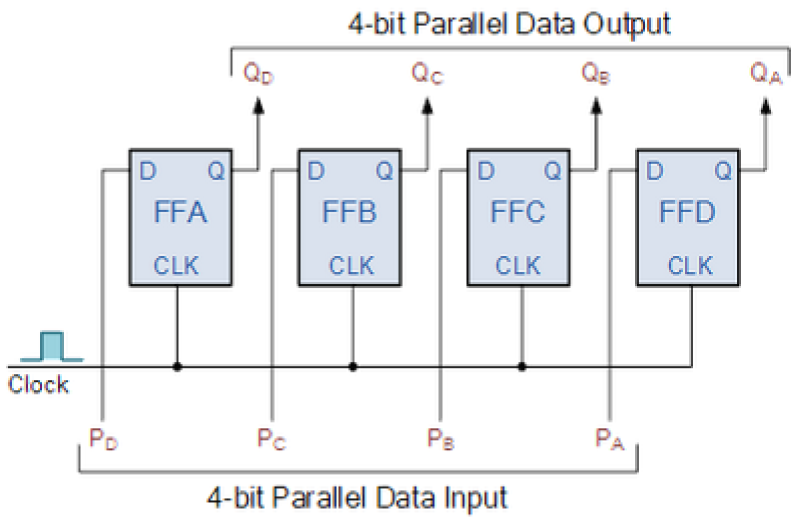
PARALLEL IN PARALLEL OUT:



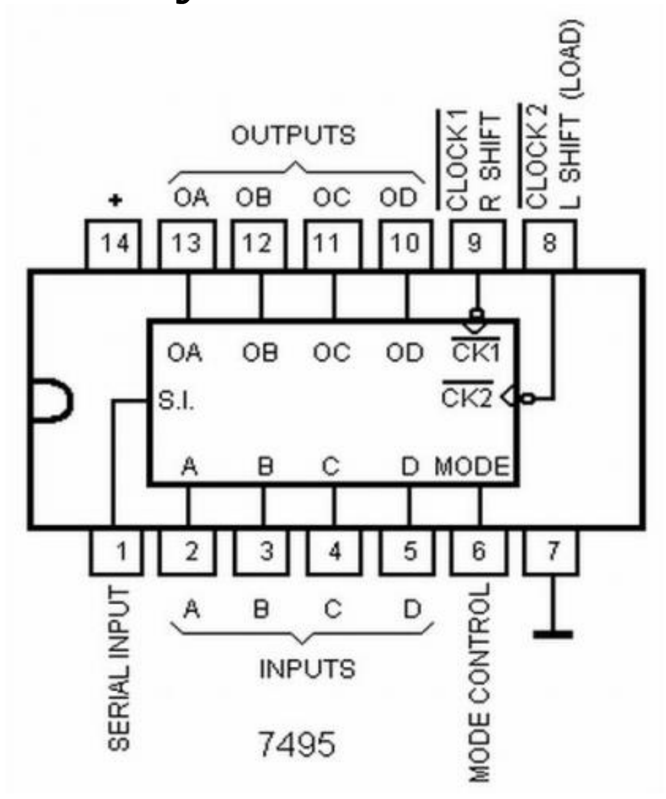
TRUTH TABLE:

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

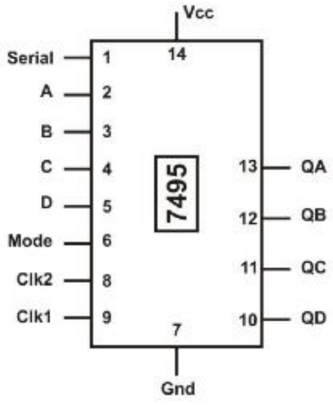
4-bit Parallel-in to Parallel-out Shift Register



Study of IC 7495



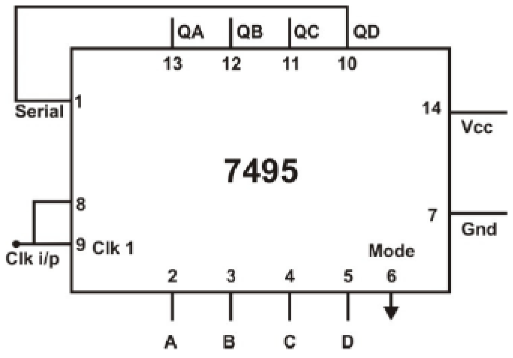
IC 7495:



OPERATION OF 7495:

Mode-0 for serial shifting of data
Mode-1 for parallel loading of data
Clk 1 is used for right shifting of data
Clk 2 is used for left shifting of data and for parallel loading of data
A, B, C, D – Parallel data inputs
QA, QB, QC, QD - Outputs

Circuit for Ring Counter :



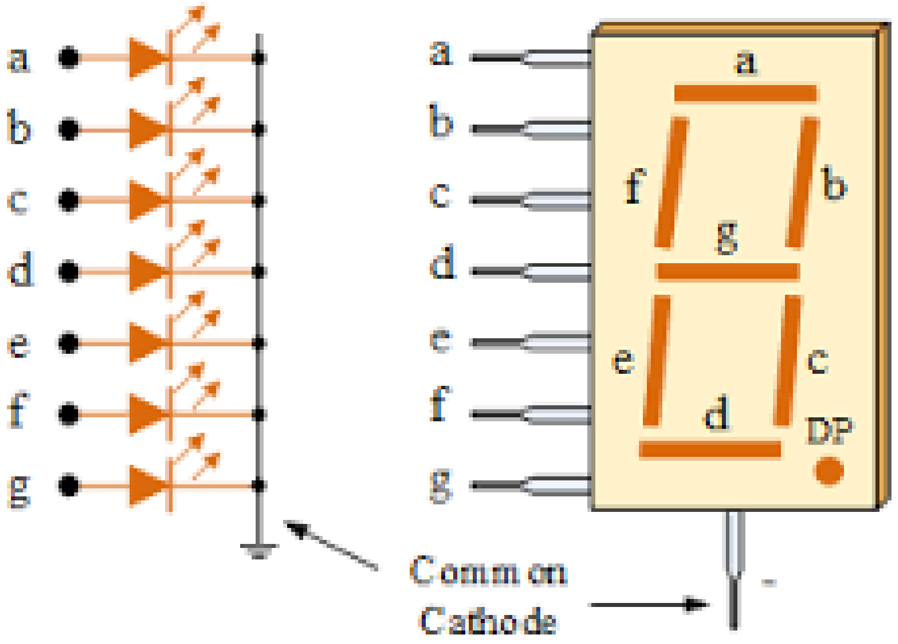
Ring counter

State Table

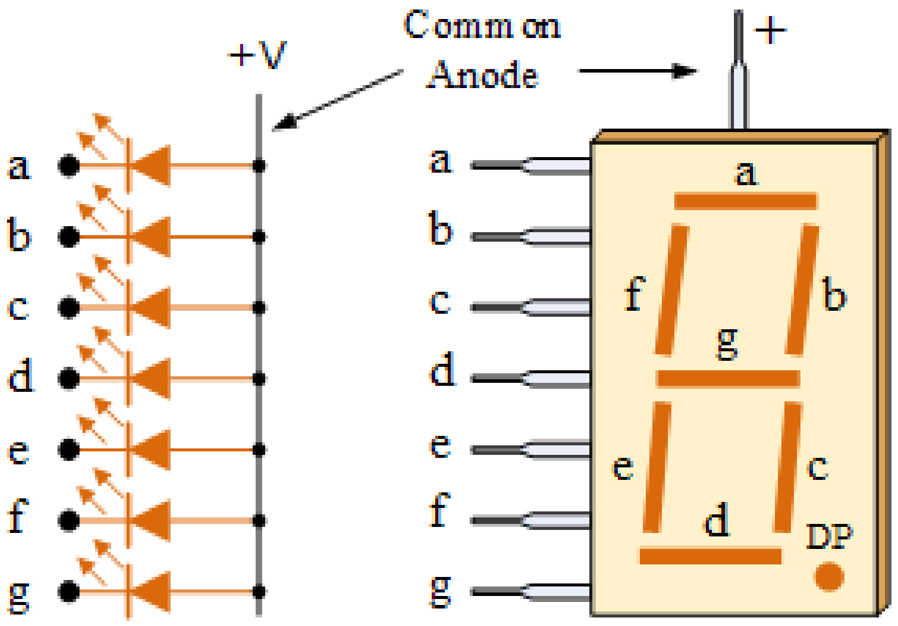
Clk	Q_A	Q_B	Q_C	Q_D
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	1	0	0	0
6	Repeats			

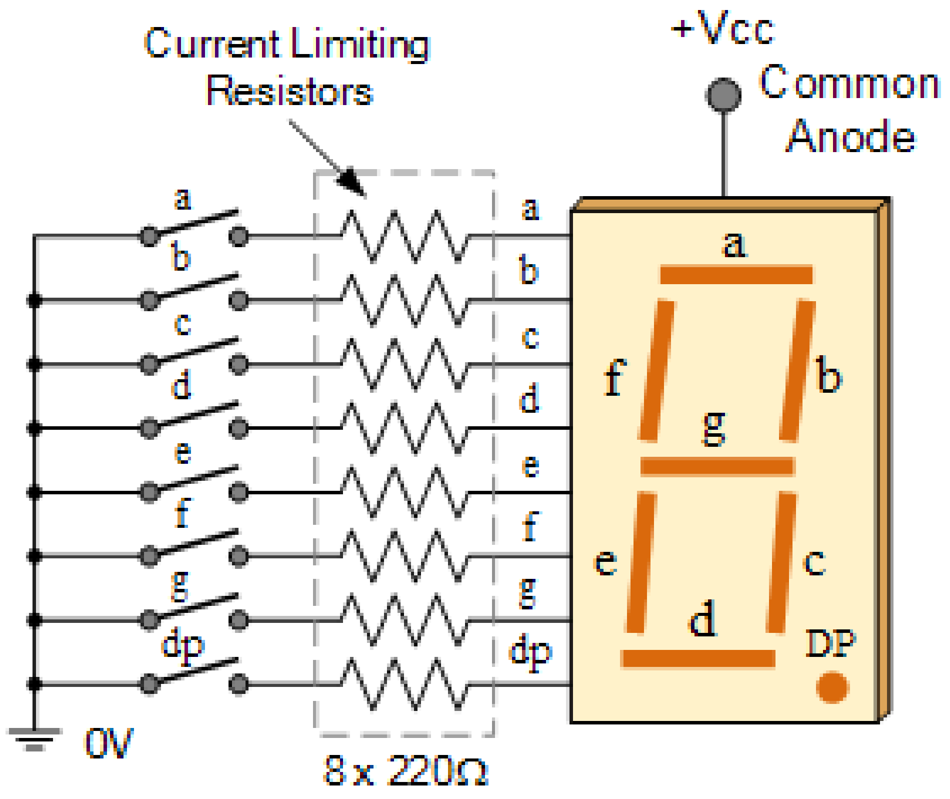
Implementation of digits using seven segment displays

Common Cathode 7-segment Display

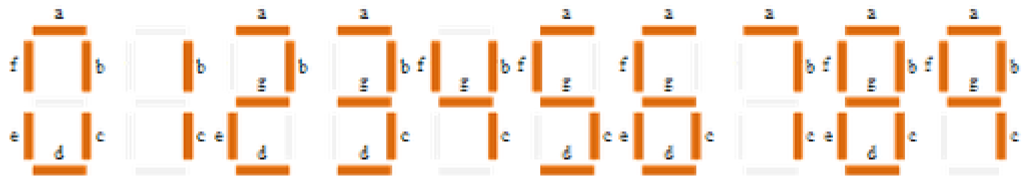


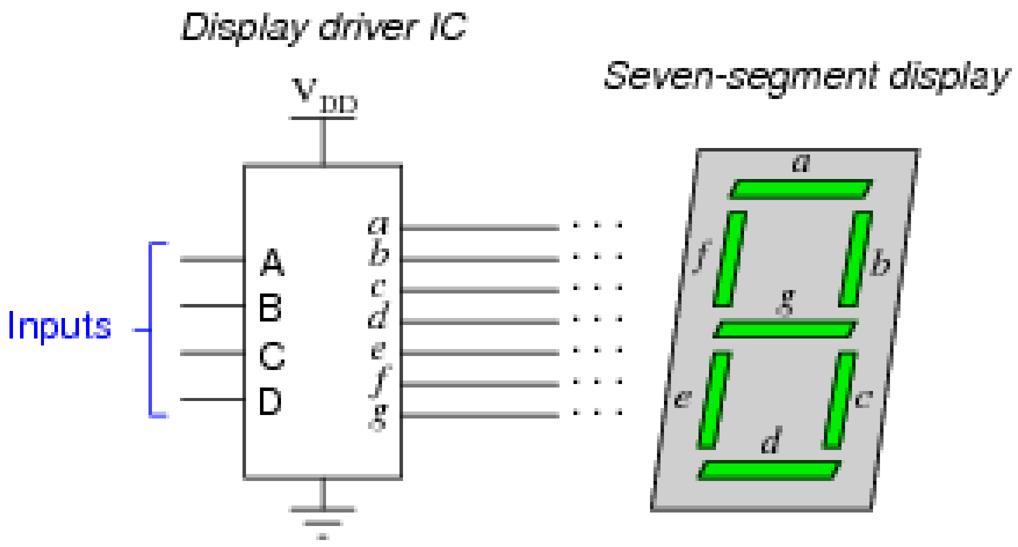
Common Anode 7-segment Display

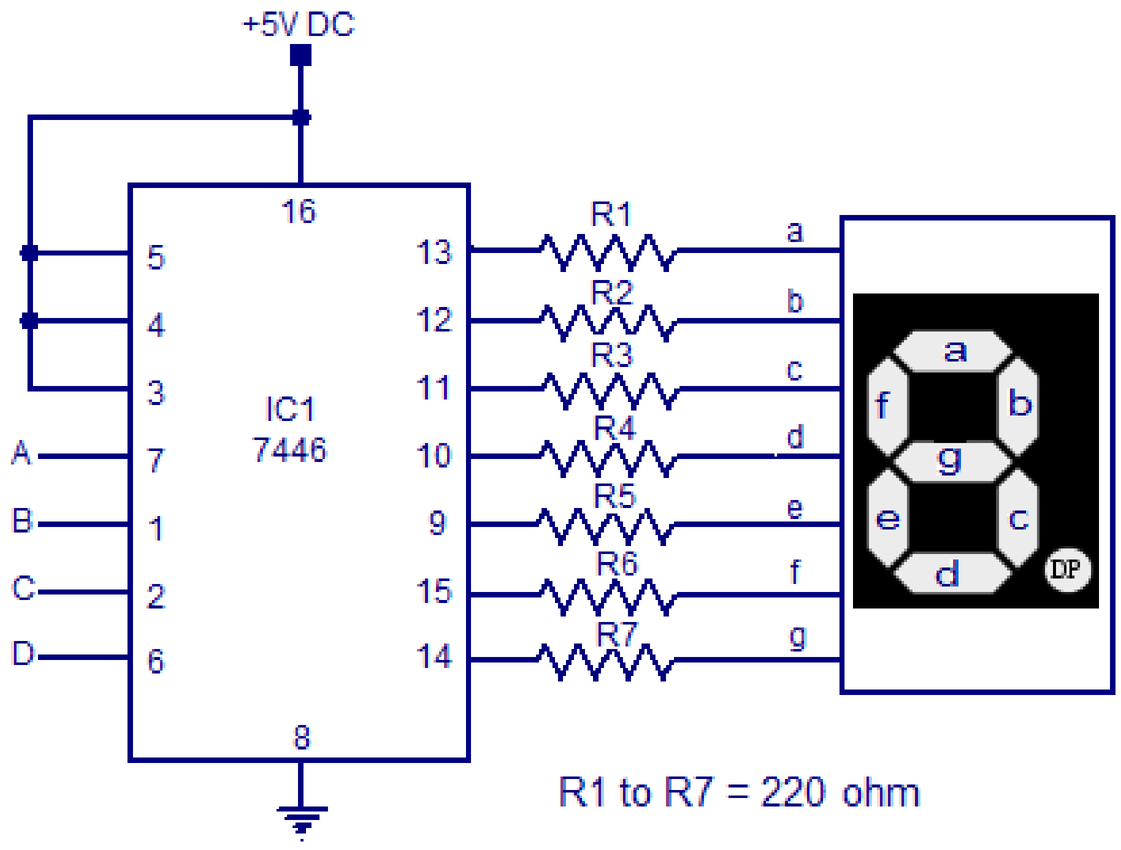


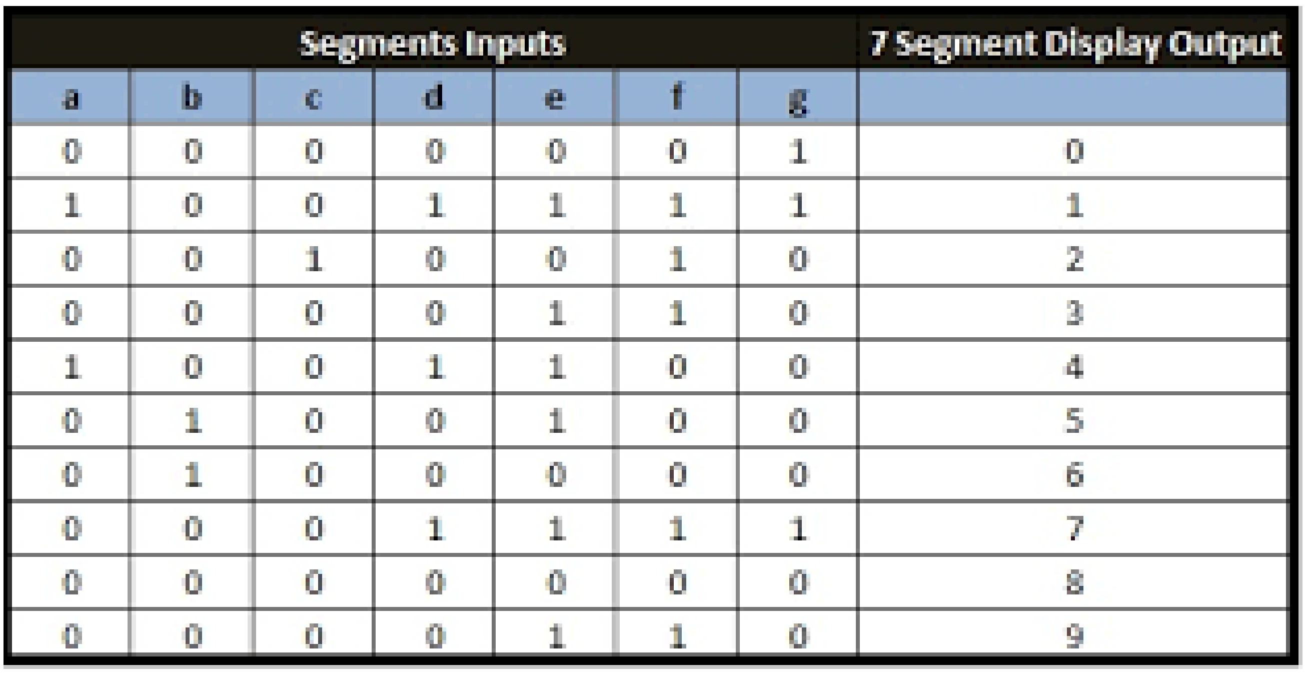


7-Segment Display Segments for all Numbers.



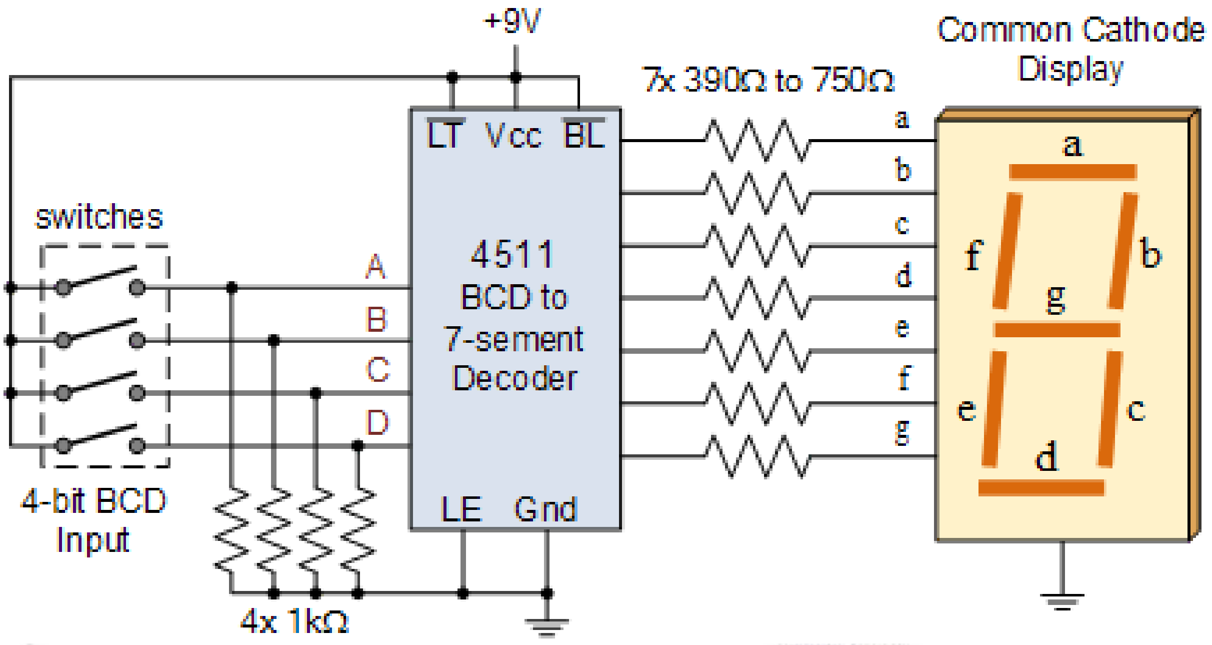


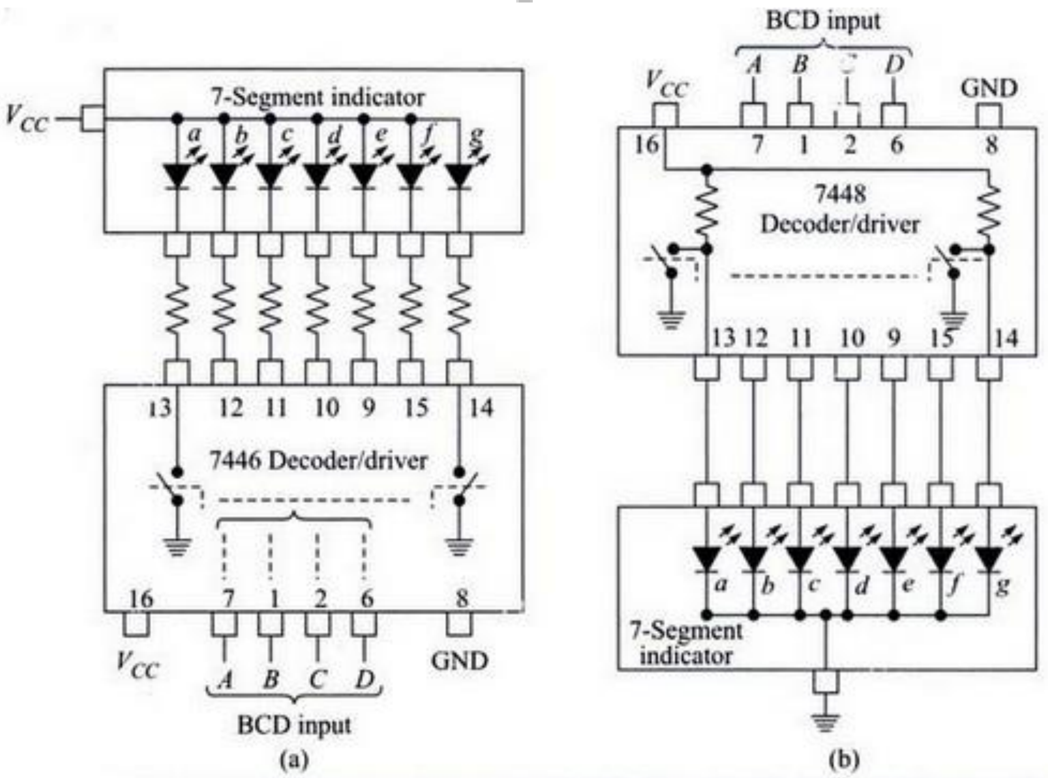




Segments Inputs							7 Segment Display Output
a	b	c	d	e	f	g	
0	0	0	0	0	0	1	0
1	0	0	1	1	1	1	1
0	0	1	0	0	1	0	2
0	0	0	0	1	1	0	3
1	0	0	1	1	0	0	4
0	1	0	0	1	0	0	5
0	1	0	0	0	0	0	6
0	0	0	1	1	1	1	7
0	0	0	0	0	0	0	8
0	0	0	0	1	1	0	9

Driving a 7-segment Display using a 4511





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