SRI SHANMUGHA COLLEGE OF ENGINEERING AND TECHNOLOGY

(Approved by AICTE, Accredited by NAAC and Affiliated to Anna University)

Pullipalayam, Morur (P.O), Sankari (T.k), Salem (D.T)

EC8382 DIGITAL SYSTEMS LABORATORY

Dept / Year: CSE/ II

Semester: III

LAB MANUAL

Department of Computer Science Engineering

LIST OF EXPERIMENTS

1. Verification of Boolean Theorems using basic gates.

2. Design and implementation of combinational circuits using basic gates for arbitrary functions, code converters.

3. Design and implement Half/Full Adder and Subtractor.

4. Design and implement combinational circuits using MSI devices:
    - 4 – bit binary adder / subtractor
    - Parity generator / checker
    - Magnitude Comparator
    - Application using multiplexers

5. Design and implement shift-registers.

6. Design and implement synchronous counters.

7. Design and implement asynchronous counters.

8. Coding combinational circuits using HDL.

9. Coding sequential circuits using HDL.

10. Design and implementation of a simple digital system (Mini Project).

INDEX

Ex.No.	Date	Title	Marks	Staff Sign.
1a		STUDY OF LOGIC GATES		
1b		VERIFICATION OF BOOLEAN THEOREMS USING DIGITAL LOGIC GATES		
2		CODE CONVERTOR		
3a		ADDER AND SUBTRACTOR		
4a		4-BIT ADDER AND SUBTRACTOR		
4b		PARITY GENERATOR & CHECKER		
4c		MAGNITUDE COMPARATOR		
4d		MULTIPLEXER AND DEMULTIPLEXER		
5		SHIFT REGISTER		
6		SYNCHRONOUS AND ASYNCHRONOUS COUNTER		
		CODING – VERILOG & VHDL		
7		BASIC LOGIC GATES		
8		COMBINATIONAL AND SEQUENTIAL CIRCUITS		

Ex.No.-1a **STUDY OF LOGIC GATES**

AIM:

To study about logic gates and verify their truth tables.

APPARATUS REQUIRED:

SL.NO.	COMPONENT	SPECIFICATION	QTY
1.	AND GATE	IC 7408	1
2.	OR GATE	IC 7432	1
3.	NOT GATE	IC 7404	1
4.	NAND GATE 2 I/P	IC 7400	1
5.	NOR GATE	IC 7402	1
6.	X-OR GATE	IC 7486	1
7.	NAND GATE 3 I/P	IC 7410	1
8.	IC TRAINER KIT	-	1
9.	PATCH CORD	-	14

THEORY:

Circuit that takes the logical decision and the process are called logic gates. Each gate has one or more input and only one output.

OR, AND and NOT are basic gates. NAND, NOR and X-OR are known as universal gates. Basic gates form these gates.

AND GATE:

The AND gate performs a logical multiplication commonly known as AND function. The output is high when both the inputs are high. The output is low level when any one of the inputs is low.

OR GATE:

The OR gate performs a logical addition commonly known as OR function. The output is high when any one of the inputs is high. The output is low level when both the inputs are low.

NOT GATE:

The NOT gate is called an inverter. The output is high when the input is low. The output is low when the input is high.

AND GATE:

The NAND gate is a contraction of AND-NOT. The output is high when both inputs are low and any one of the input is low .The output is low level when both inputs are high.

NOR GATE:

The NOR gate is a contraction of OR-NOT. The output is high when both inputs are low. The output is low when one or both inputs are high.

X- OR GATE:

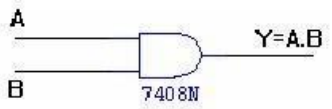
The output is high when any one of the inputs is high. The output is low when both the inputs are low and both the inputs are high.

PROCEDURE:

(i) Connections are given as per circuit diagram.
(ii) Logical inputs are given as per circuit diagram.
(iii) Observe the output and verify the truth table.

AND GATE

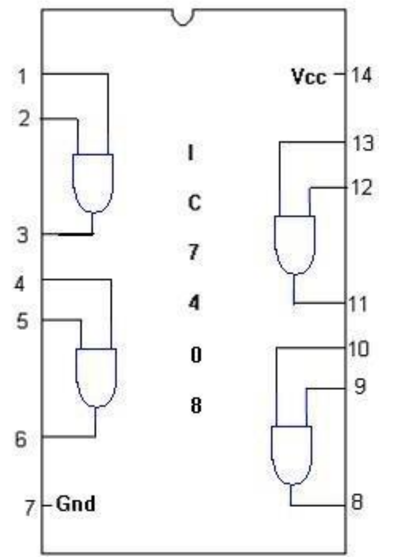
SYMBOL



TRUTH TABLE

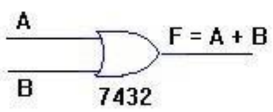
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

PIN DIAGRAM



OR GATE

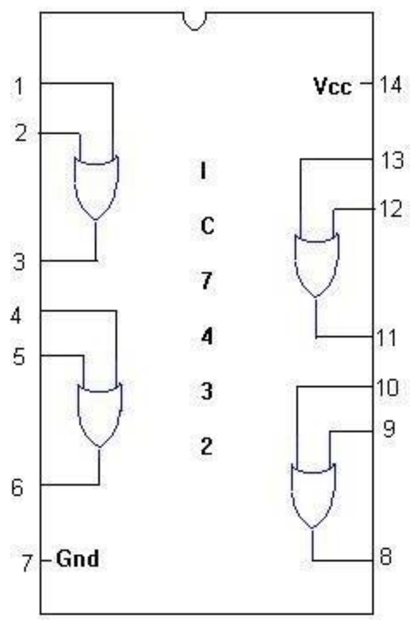
SYMBOL :



TRUTH TABLE

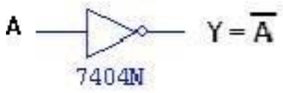
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

PIN DIAGRAM :



NOT GATE

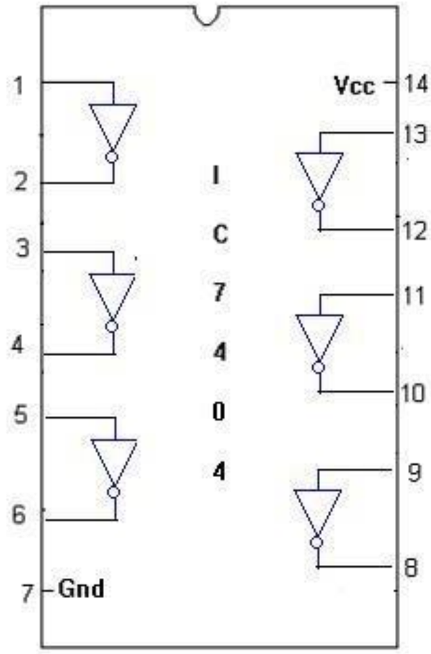
SYMBOL



TRUTH TABLE :

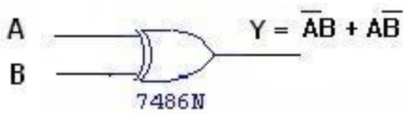
A	$\overline{A}$
0	1
1	0

PIN DIAGRAM



EX-OR GATE

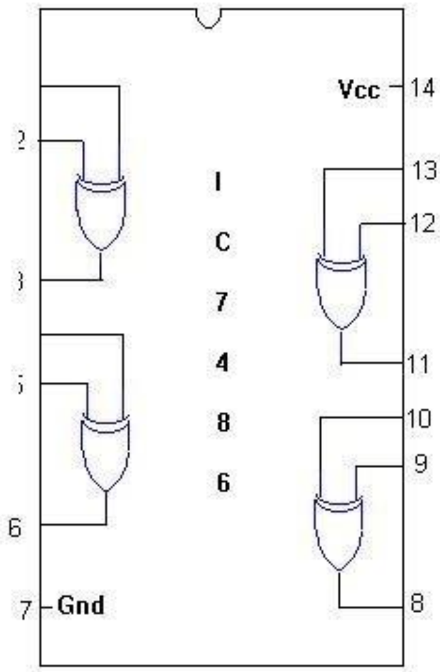
SYMBOL



TRUTH TABLE :

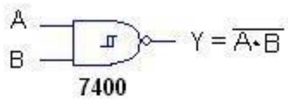
A	B	$\overline{A}B + A\overline{B}$
0	0	0
0	1	1
1	0	1
1	1	0

PIN DIAGRAM



2-INPUT NAND GATE

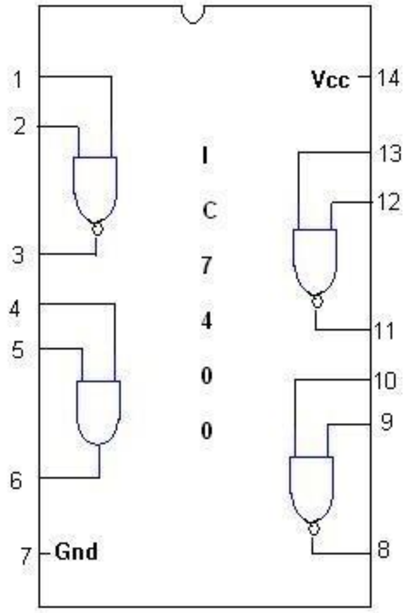
SYMBOL



TRUTH TABLE

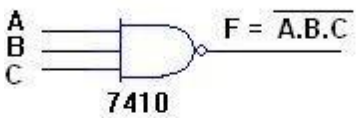
A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

PIN DIAGRAM



3-INPUT NAND GATE

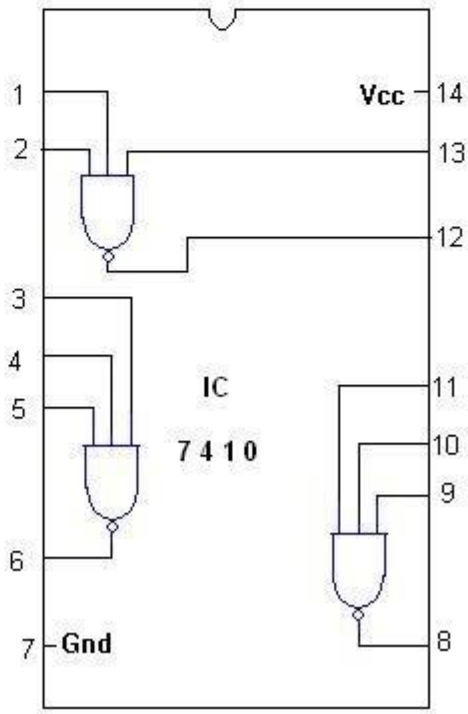
SYMBOL :



TRUTH TABLE

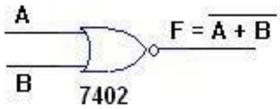
A	B	C	$\overline{A.B.C}$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

PIN DIAGRAM :



NOR GATE

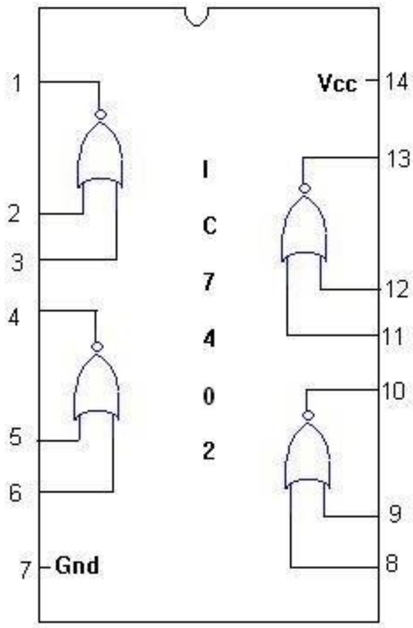
SYMBOL :



TRUTH TABLE

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

PIN DIAGRAM :



RESULT:

The logic gates are studied and its truth tables are verified.

Ex.No.-1b VERIFICATION OF BOOLEAN THEOREMS USING DIGITAL LOGIC GATES

AIM:

To verify the Boolean Theorems using logic gates.

APPARATUS REQUIRED:

SL. NO.	COMPONENT	SPECIFICATION	QTY.
1.	AND GATE	IC 7408	1
2.	OR GATE	IC 7432	1
3.	NOT GATE	IC 7404	1
4.	IC TRAINER KIT	-	1
5.	CONNECTING WIRES	-	As per required

THEORY:

BASIC BOOLEAN LAWS

1. Commutative Law
The binary operator OR, AND is said to be commutative if,

1. A+B = B+A
2. A.B=B.A

2. Associative Law
The binary operator OR, AND is said to be associative if,

1. A+(B+C) = (A+B)+C
2. A.(B.C) = (A.B).C

3. Distributive Law
The binary operator OR, AND is said to be distributive if,

1. A+(B.C) = (A+B).(A+C)
2. A.(B+C) = (A.B)+(A.C)

4. Absorption Law

1. A+AB = A
2. A+AB = A+B

5. Involution (or) Double complement Law

1. A = A

6. Idempotent Law

1. A+A = A
2. A.A = A

7. Complementary Law

1. A+A' = 1
2. A.A' = 0

8. De Morgan's Theorem

1. The complement of the sum is equal to the sum of the product of the individual complements.

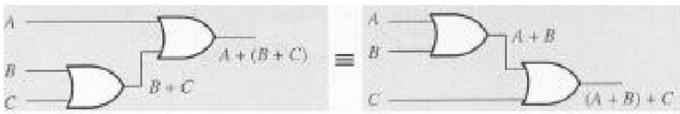
A+B = A.B

2. The complement of the product is equal to the sum of the individual complements.

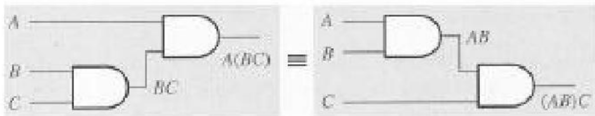
A.B = A+B

Associative Laws of Boolean Algebra

A + (B + C) = (A + B) + C



A • (B • C) = (A • B) • C



Proof of the Associative Property for the OR operation: (A+B)+C = A+(B+C)

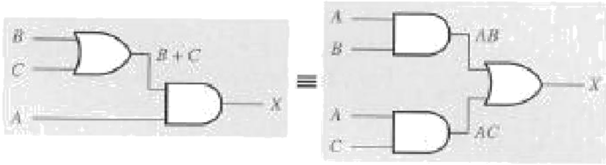
A	B	C	(A+B)	(B+C)	A+(B+C)	(A+B)+C
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	1	0	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

Proof of the Associative Property for the AND operation: (A·B)·C = A·(B·C)

A	B	C	(A·B)	(B·C)	A· (B·C)	(A·B)·C
0	0	0	0	0	0	0
0	0	1	0	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	0	0
1	1	0	1	0	0	0
1	1	1	1	1	1	1

Distributive Laws of Boolean Algebra

$$A \bullet (B + C) = A \bullet B + A \bullet C$$
$$A\,(B + C) = A\,B + A\,C$$



$X = A(B + C)$ $\quad$ $X = AB + AC$

Proof of Distributive Rule

A	B	C	A·B	A·C	(A·B)+ (A·C)	(B+C)	A·(B+C)
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	0	1	0
0	1	1	0	0	0	1	0
1	0	0	0	0	0	0	0
1	0	1	0	1	1	1	1
1	1	0	1	0	1	1	1
1	1	1	1	1	1	1	1

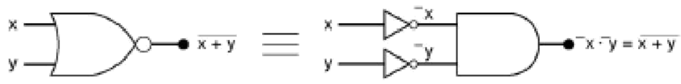
Proof of Distributive Rule

A	B	C	A+B	A+C	(A+B)· (A+C)	(B·C)	A+(B·C)
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	0	1	0	1
1	1	1	1	1	1	1	1

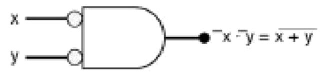
Demorgan's Theorem

a) Proof of equation (1):

Construct the two circuits corresponding to the functions A'. B'and (A+B)' respectively. Show that for all combinations of A and B, the two circuits give identical results. Connect these circuits and verify their operations.



(a)



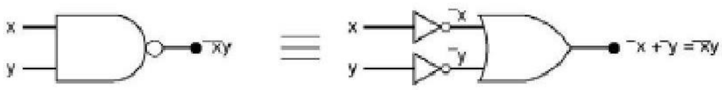
(b)

Proof (via Truth Table) of DeMorgan's Theorem $\overline{A \bullet B} = \overline{A} + \overline{B}$

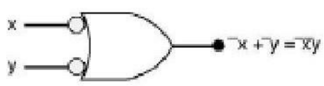
A	B	A·B	$\overline{A \bullet B}$	$\overline{A}$	$\overline{B}$	$\overline{A}+\overline{B}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

b) Proof of equation (2)

Construct two circuits corresponding to the functions A'+B'and (A.B)' A.B, respectively. Show that, for all combinations of A and B, the two circuits give identical results. Connect these circuits and verify their operations.



(a)



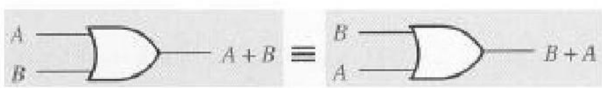
(b)

Proof (via Truth Table) of DeMorgan's Theorem $\overline{A + B} = \overline{A} \bullet \overline{B}$

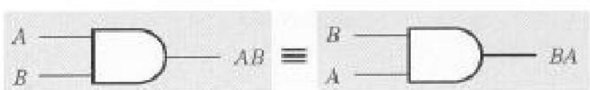
A	B	A+B	$\overline{A+B}$	$\overline{A}$	$\overline{B}$	$\overline{A} \bullet \overline{B}$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Commutative Laws of Boolean Algebra

A + B = B + A



A • B = B • A



We will also use the following set of postulates:

P1: Boolean algebra is closed under the AND, OR, and NOT operations.

P2: The identity element with respect to • is one and + is zero. There is no identity element with respect to logical NOT.

P3: The • and + operators are commutative.

P4: • and + are distributive with respect to one another. That is,
A • (B + C) = (A • B) + (A • C) and A + (B • C) = (A + B) • (A + C).

P5: For every value A there exists a value A’ such that A•A’ = 0 and A+A’ = 1.
This value is the logical complement (or NOT) of A.

P6: • and + are both associative. That is, (A•B)•C = A•(B•C) and (A+B)+C = A+(B+C).
You can prove all other theorems in boolean algebra using these postulates.

PROCEDURE:

1. Obtain the required IC along with the Digital trainer kit.
2. Connect zero volts to GND pin and +5 volts to Vcc .
3. Apply the inputs to the respective input pins.
4. Verify the output with the truth table.

RESULT:

Thus the above stated Boolean laws are verified.

Ex.No.-2 CODE CONVERTOR

AIM:

To design and implement 4-bit

(i) Binary to gray code converter
(ii) Gray to binary code converter
(iii) BCD to excess-3 code converter
(iv) Excess-3 to BCD code converter

APPARATUS REQUIRED:

SL.NO.	COMPONENT	SPECIFICATION	QTY.
1.	X-OR GATE	IC 7486	1
2.	AND GATE	IC 7408	1
3.	OR GATE	IC 7432	1
4.	NOT GATE	IC 7404	1
5.	IC TRAINER KIT	-	1
6.	PATCH CORDS	-	35

THEORY:

The availability of large variety of codes for the same discrete elements of information results in the use of different codes by different systems. A conversion circuit must be inserted between the two systems if each uses different codes for same information. Thus, code converter is a circuit that makes the two systems compatible even though each uses different binary code.

The bit combination assigned to binary code to gray code. Since each code uses four bits to represent a decimal digit. There are four inputs and four outputs. Gray code is a non-weighted code.

The input variable are designated as B3, B2, B1, B0 and the output variables are designated as C3, C2, C1, Co. from the truth table, combinational circuit is designed. The Boolean functions are obtained from K-Map for each output variable.

A code converter is a circuit that makes the two systems compatible even though each uses a different binary code. To convert from binary code to Excess-3 code, the input lines must supply the bit combination of elements as specified by code and the output lines generate the corresponding bit combination of code. Each one of the four maps represents one of the four outputs of the circuit as a function of the four input variables.

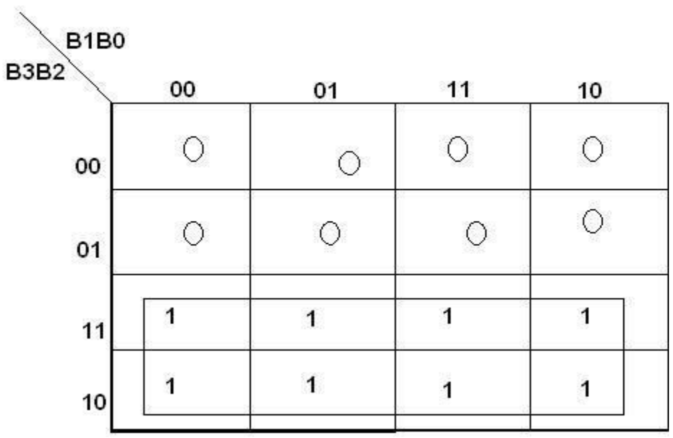
A two-level logic diagram may be obtained directly from the Boolean expressions derived by the maps. These are various other possibilities for a logic diagram that implements this circuit. Now the OR gate whose output is C+D has been used to implement partially each of three outputs.

BINARY TO GRAY CODE CONVERTOR

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

K-Map for G_3



$G_3 = B_3$

K-Map for G_2

B3B2 \ B1B0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

$$G2 = B3 \oplus B2$$

K-Map for G_1

B3B2 \ B1B0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	1	1	0	0
10	0	0	1	1

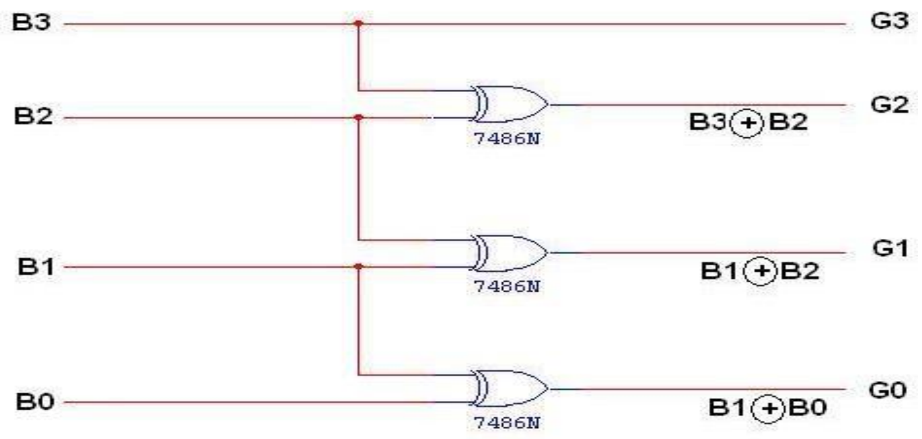
$$G1 = B1 \oplus B2$$

K-Map for G_0

B3B2 \ B1B0	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

$$G0 = B1 \oplus B0$$

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 B1:

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

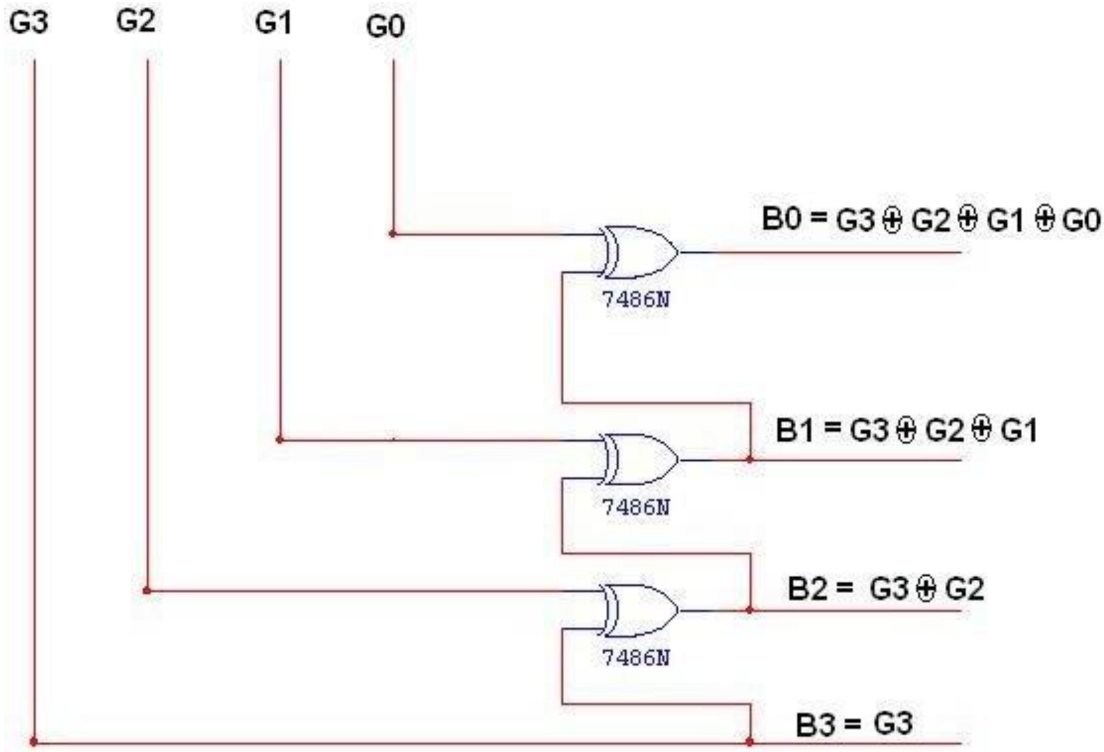
$B1 = G3 \oplus G2 \oplus G1$

K-Map for B0:

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

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

LOGIC DIAGRAM:

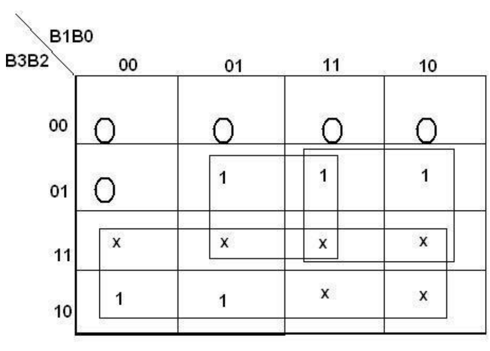


TRUTH TABLE: **BCD TO EXCESS-3 CONVERTOR**

| BCD input | Excess – 3 output |

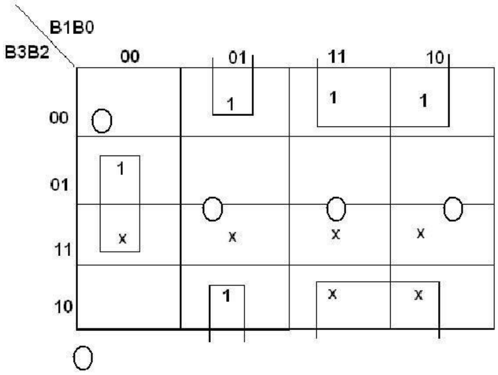
B3	B2	B1	B0	G3	G2	G1	G0
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	X

K-Map for E_3:



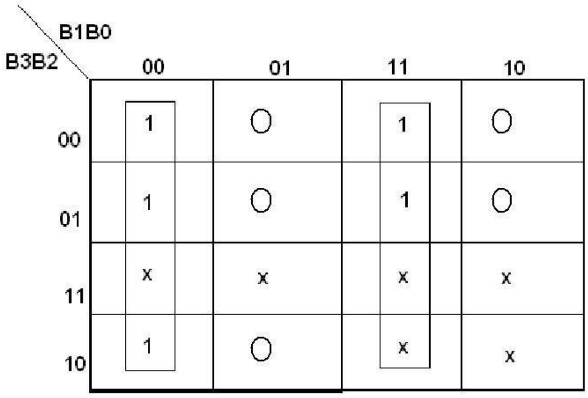
E3 = B3 + B2 (B0 + B1)

K-Map for E_2:



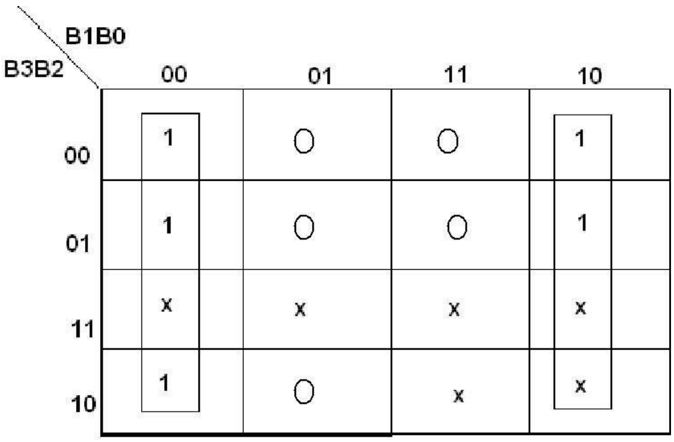
$$E2 = B2 \oplus (B1 + B0)$$

K-Map for E_1:



$$E1 = B1 \,\overline{\oplus}\, B0$$

K-Map for E_0:



$$E0 = \overline{B0}$$

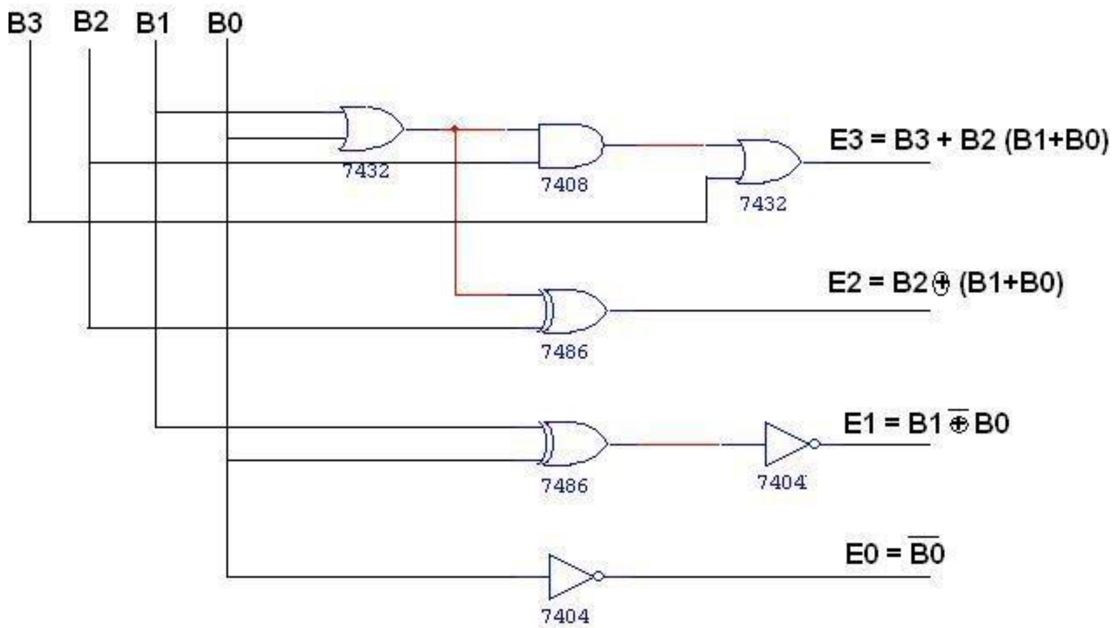
EXCESS-3 TO BCD CONVERTOR

TRUTH TABLE:

| Excess – 3 Input | BCD Output |

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

LOGIC DIAGRAM:



EXCESS-3 TO BCD CONVERTOR

K-Map for A:

X1 X2 \ X3 X4	00	01	11	10
00	X	X	0	X
01	0	0	0	0
11	1	X	X	X
10	0	0	1	0

A = X1 X2 + X3 X4 X1

K-Map for B:

X1 X2 \ X3 X4	00	01	11	10
00	X	X	0	X
01	0	0	1	0
11	0	X	X	X
10	1	1	0	1

$$B = X2 \oplus (\overline{X3} + \overline{X4})$$

K-Map for C:

X1 X2 \ X3 X4	00	01	11	10
00	X	X	0	X
01	0	1	X	1
11	0	X	X	X
10	X	1	0	1

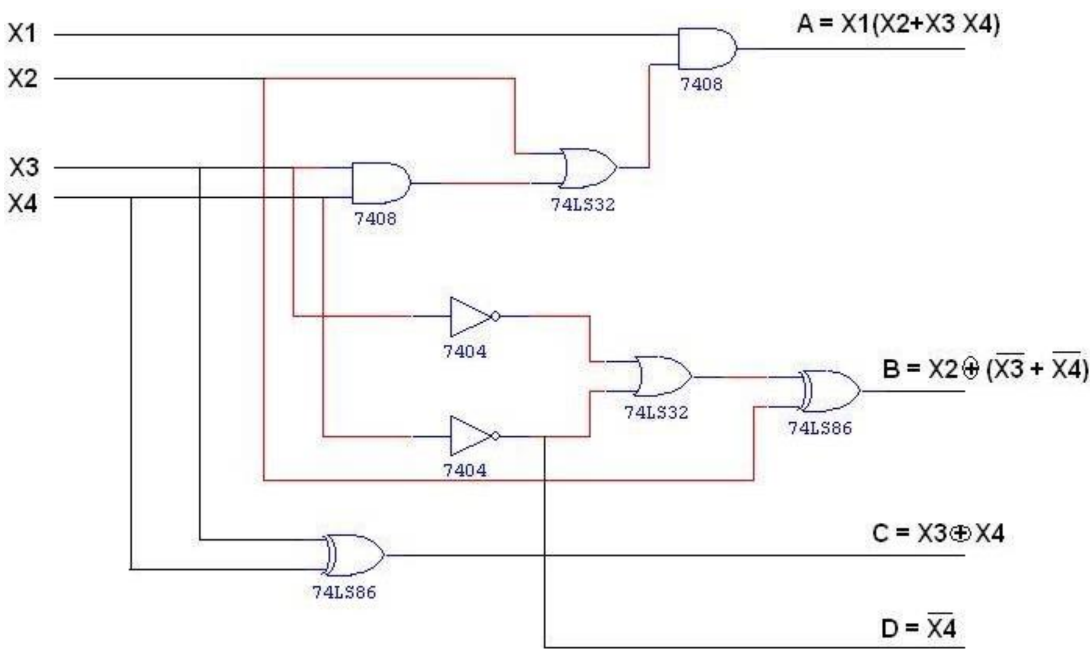
$$C = X3 \oplus X4$$

K-Map for D:

X1 X2 \ X3 X4	00	01	11	10
00	X	X	0	X
01	1	0	0	1
11	1	X	X	X
10	1	0	0	1

$$D = \overline{X4}$$

EXCESS-3 TO BCD CONVERTOR



PROCEDURE:

(i) Connections were given as per circuit diagram.
(ii) Logical inputs were given as per truth table
(iii) Observe the logical output and verify with the truth tables.

RESULT:

Thus the following 4-bit converters are designed and constructed.

(i) Binary to gray code converter
(ii) Gray to binary code converter
(iii) BCD to excess-3 code converter
(iv) Excess-3 to BCD code converter

Ex.No.-3 ADDER AND SUBTRACTOR

AIM:

To design and construct half adder, full adder, half subtractor and full subtractor circuits and verify the truth table using logic gates.

APPARATUS REQUIRED:

SL.NO.	COMPONENT	SPECIFICATION	QTY.
1.	AND GATE	IC 7408	1
2.	X-OR GATE	IC 7486	1
3.	NOT GATE	IC 7404	1
4.	OR GATE	IC 7432	1
5.	IC TRAINER KIT	-	1
6.	PATCH CORDS	-	23

THEORY:

HALF ADDER:

A half adder has two inputs for the two bits to be added and two outputs one from the sum ‘ S’ and other from the carry ‘ c’ into the higher adder position. Above circuit is called as a carry signal from the addition of the less significant bits sum from the X-OR Gate the carry out from the AND gate.

FULL ADDER:

A full adder is a combinational circuit that forms the arithmetic sum of input; it consists of three inputs and two outputs. A full adder is useful to add three bits at a time but a half adder cannot do so. In full adder sum output will be taken from X-OR Gate, carry output will be taken from OR Gate.

HALF SUBTRACTOR:

The half subtractor is constructed using X-OR and AND Gate. The half subtractor has two input and two outputs. The outputs are difference and borrow. The difference can be applied using X-OR Gate, borrow output can be implemented using an AND Gate and an inverter.

FULL SUBTRACTOR:

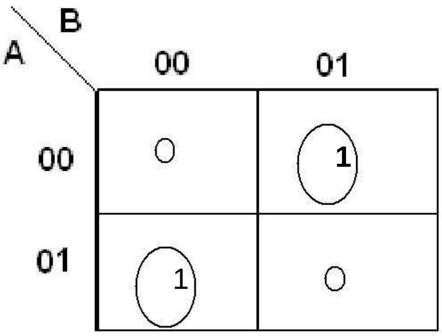
The full subtractor is a combination of X-OR, AND, OR, NOT Gates. In a full subtractor the logic circuit should have three inputs and two outputs. The two half subtractor put together gives a full subtractor .The first half subtractor will be C and A B. The output will be difference output of full subtractor. The expression AB assembles the borrow output of the half subtractor and the second term is the inverted difference output of first X-OR.

HALF ADDER

TRUTH TABLE:

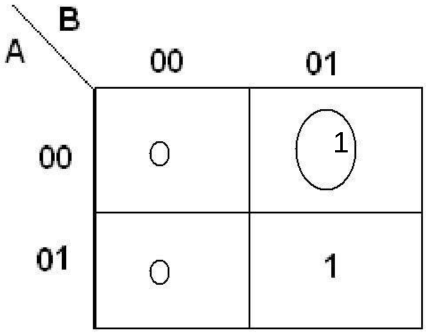
A	B	CARRY	SUM
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

K-Map for SUM:



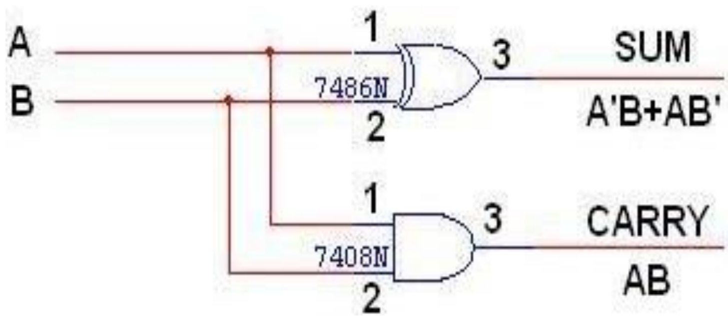
SUM = A'B + AB'

K-Map for CARRY:



CARRY = AB

LOGIC DIAGRAM:

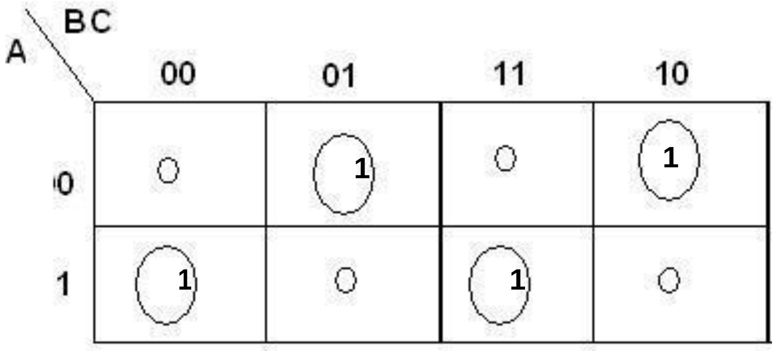


FULL ADDER

TRUTH TABLE:

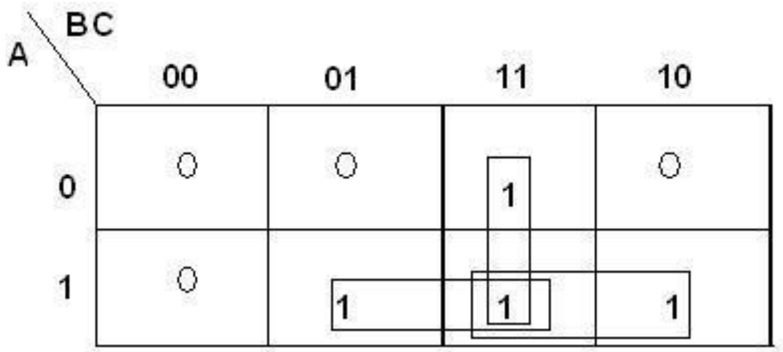
A	B	C	CARRY	SUM
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

K-Map for SUM



SUM = A’B’C + A’BC’ + ABC’ + ABC

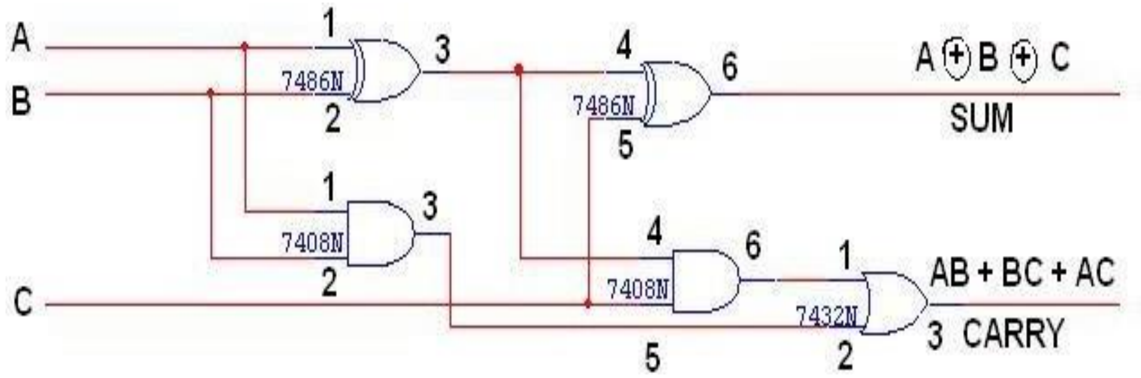
K-Map for CARRY



CARRY = AB + BC + AC

LOGIC DIAGRAM:

FULL ADDER USING TWO HALF ADDER

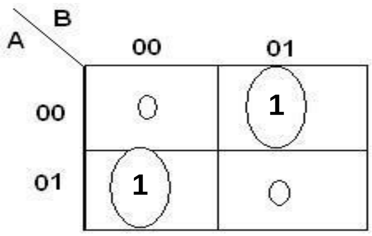


HALF SUBTRACTOR

TRUTH TABLE:

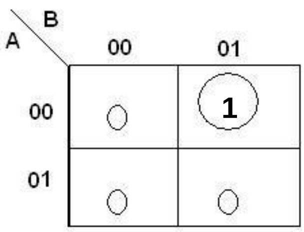
A	B	BORROW	DIFFERENCE
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

K-Map for DIFFERENCE



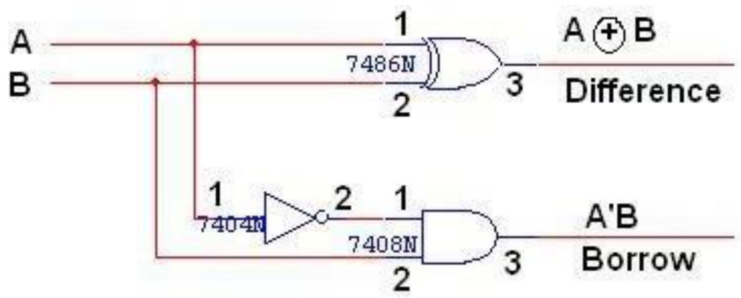
DIFFERENCE = A’B + AB’

K-Map for BORROW



BORROW = A’B

LOGIC DIAGRAM

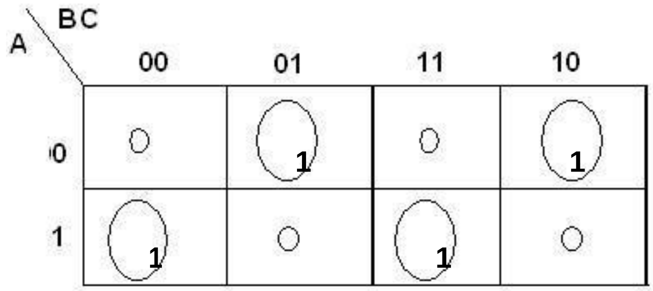


FULL SUBTRACTOR

TRUTH TABLE:

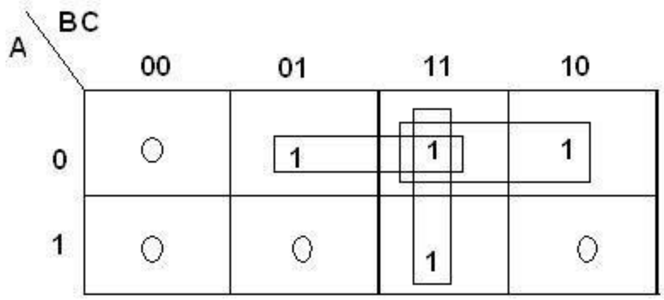
A	B	C	BORROW	DIFFERENCE
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-Map for Difference



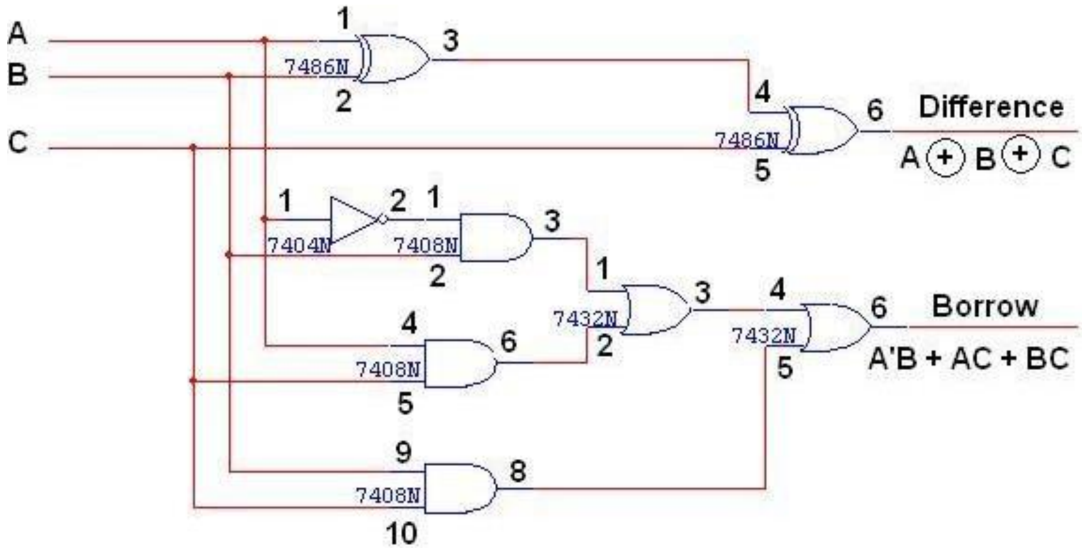
Difference = A'B'C + A'BC' + AB'C' + ABC

K-Map for Borrow

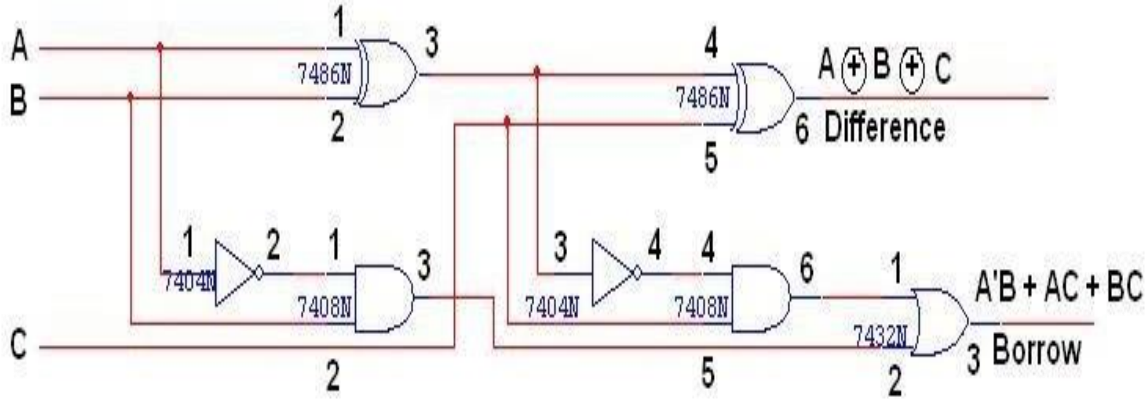


Borrow = A’B + BC + A’C

LOGIC DIAGRAM:



FULL SUBTRACTOR USING TWO HALF SUBTRACTOR



PROCEEDURE:

(i) Connections are given as per circuit diagram.

(ii) Logical inputs are given as per circuit diagram.

(iii) Observe the output and verify the truth table.

RESULT:

Thus, the half adder, full adder, half subtractor and full subtractor circuits are designed, constructed and verified the truth table using logic gates.

Ex.No.-4a 4-BIT ADDER AND SUBTRACTOR

AIM:

To design and implement 4-bit adder and subtractor using basic gates and MSI device IC 7483.

APPARATUS REQUIRED:

SL.NO.	COMPONENT	SPECIFICATION	QTY.
1.	IC	IC 7483	1
2.	EX-OR GATE	IC 7486	1
3.	NOT GATE	IC 7404	1
3.	IC TRAINER KIT	-	1
4.	PATCH CORDS	-	40

THEORY:

4 BIT BINARY ADDER:

A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of next full adder in chain. The augends bits of 'A' and the addend bits of 'B' are designated by subscript numbers from right to left, with subscript 0 denoting the least significant bits. The carries are connected in chain through the full adder. The input carry to the adder is C_0 and it ripples through the full adder to the output carry C_4.

4 BIT BINARY SUBTRACTOR:

The circuit for subtracting A-B consists of an adder with inverters, placed between each data input 'B' and the corresponding input of full adder. The input carry C_0 must be equal to 1 when performing subtraction.

4 BIT BINARY ADDER/SUBTRACTOR:

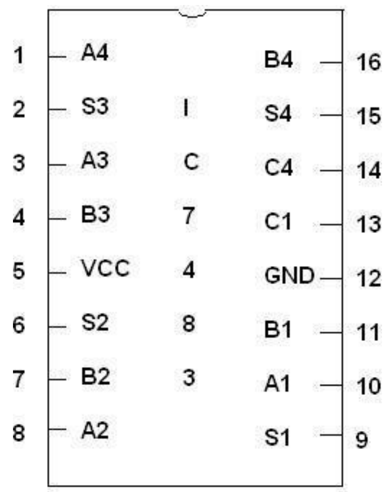
The addition and subtraction operation can be combined into one circuit with one common binary adder. The mode input M controls the operation. When M=0, the circuit is adder circuit. When M=1, it becomes subtractor.

4 BIT BCD ADDER:

Consider the arithmetic addition of two decimal digits in BCD, together with an input carry from a previous stage. Since each input digit does not exceed 9, the output sum cannot be greater than 19, the 1 in the sum being an input carry. The output of two decimal digits must be represented in BCD and should appear in the form listed in the columns.

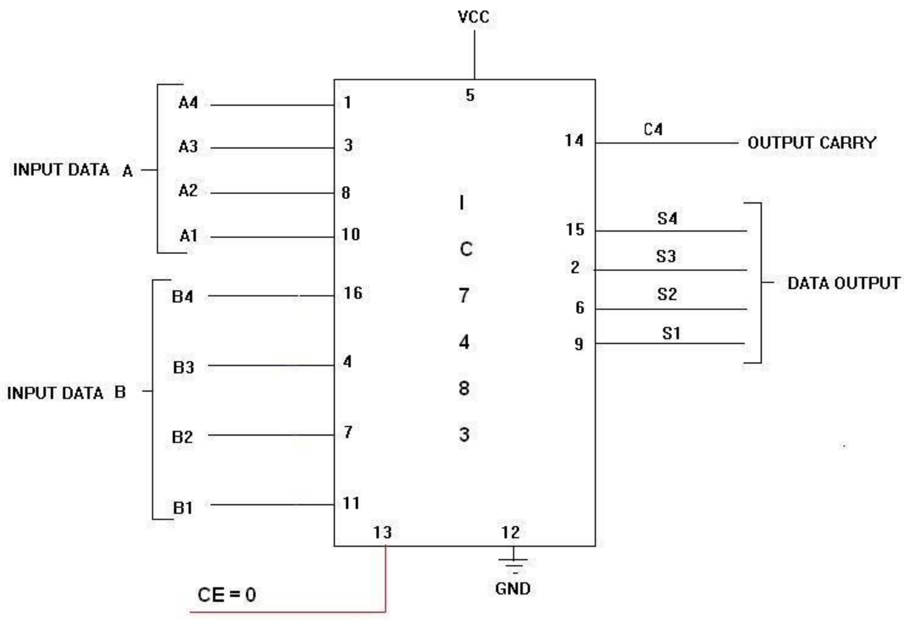
ABCD adder that adds 2 BCD digits and produce a sum digit in BCD. The 2 decimal digits, together with the input carry, are first added in the top 4 bit adder to produce the binary sum.

PIN DIAGRAM FOR IC 7483:



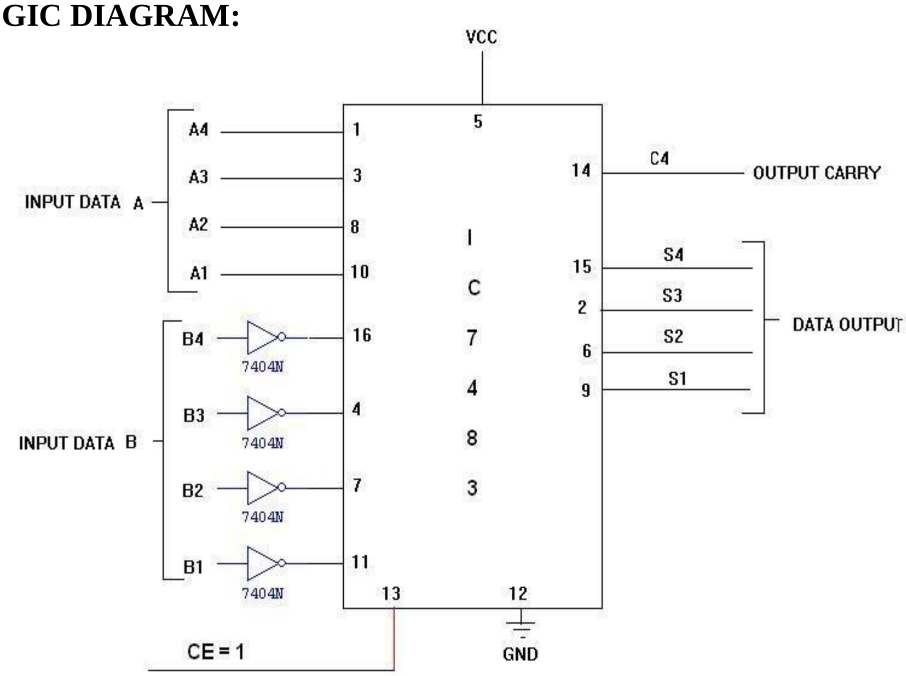
<u>4-BIT BINARY ADDER</u>

LOGIC DIAGRAM:



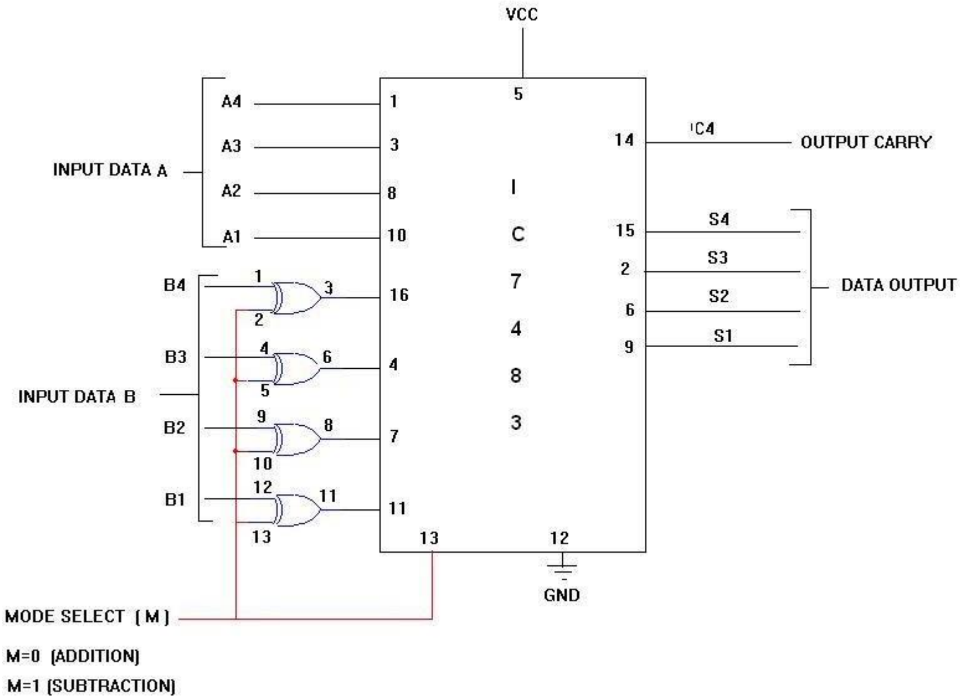
4-BIT BINARY SUBTRACTOR

LOGIC DIAGRAM:



4-BIT BINARY ADDER/SUBTRACTOR

LOGIC DIAGRAM:



TRUTH TABLE:

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

PROCEDURE:

(i) Connections were given as per circuit diagram.
(ii) Logical inputs were given as per truth table
(iii) Observe the logical output and verify with the truth tables.

RESULT:

Thus the 4-bit adder and subtractor using basic gates and MSI device IC 7483 is designed and implemented.

Ex.No.-4b PARITY GENERATOR AND CHECKER

AIM:

To design and verify the truth table of a three bit Odd Parity generator and checker.

APPARATUS REQUIRED:

SL. NO.	NAME OF THE APPARATUS	RANGE	QUANTITY
1.	Digital IC trainer kit		1
2.	EX-OR gate	IC 7486	
3.	NOT gate	IC 7404	
4.	Connecting wires		As required

THEORY:

A parity bit is used for the purpose of detecting errors during transmission of binary information. A parity bit is an extra bit included with a binary message to make the number of 1's either odd or even. The message including the parity bit is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity does not correspond with the one transmitted. The circuit that generates the parity bit in the transmitter is called a parity generator and the circuit that checks the parity in the receiver is called a parity checker.

In even parity the added parity bit will make the total number of 1's an even amount and in odd parity the added parity bit will make the total number of 1's an odd amount.

In a three bit odd parity generator the three bits in the message together with the parity bit are transmitted to their destination, where they are applied to the parity checker circuit. The parity checker circuit checks for possible errors in the transmission.

Since the information was transmitted with odd parity the four bits received must have an odd number of 1's. An error occurs during the transmission if the four bits received have an even number of 1's, indicating that one bit has changed during transmission. The output of the parity checker is denoted by PEC (parity error check) and it will be equal to 1 if an error occurs, i.e., if the four bits received has an even number of 1's.

ODD PARITY GENERATOR

TRUTH TABLE:

SL.NO.	INPUT (Three bit message)			OUTPUT (Odd Parity bit)
	A	B	C	P
1.	0	0	0	1
2.	0	0	1	0
3.	0	1	0	0
4.	0	1	1	1
5.	1	0	0	0
6.	1	0	1	1
7.	1	1	0	1
8.	1	1	1	0

From the truth table the expression for the output parity bit is,

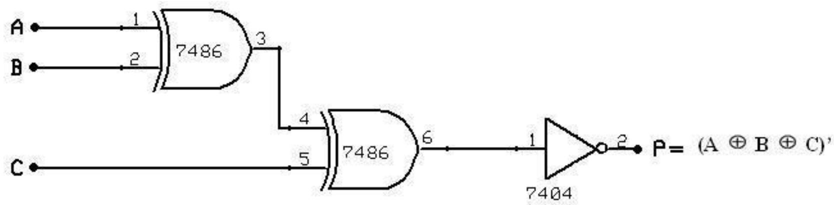
$$P(A, B, C) = \Sigma\ (0, 3, 5, 6)$$

Also written as,

$$P = A'B'C' + A'BC + AB'C + ABC' = (A \oplus B \oplus C)'$$

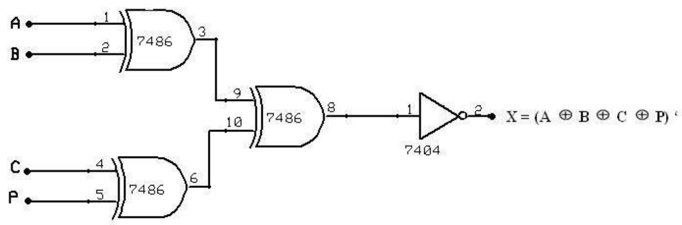
ODD PARITY GENERATOR

CIRCUIT DIAGRAM:



ODD PARITY CHECKER

CIRCUIT DIAGRAM:



ODD PARITY CHECKER

TRUTH TABLE:

SL.NO.	**INPUT** (**4 - Bit Message Received**)				**OUTPUT** (**Parity Error Check**)
	A	**B**	**C**	**P**	**X**
1.	0	0	0	0	1
2.	0	0	0	1	0
3.	0	0	1	0	0
4.	0	0	1	1	1
5.	0	1	0	0	0
6.	0	1	0	1	1
7.	0	1	1	0	1
8.	0	1	1	1	0
9.	1	0	0	0	0
10.	1	0	0	1	1
11.	1	0	1	0	1
12.	1	0	1	1	0
13.	1	1	0	0	1
14.	1	1	0	1	0
15.	1	1	1	0	0
16.	1	1	1	1	1

From the truth table the expression for the output parity checker bit is,

$$X (A, B, C, P) = \Sigma (0, 3, 5, 6, 9, 10, 12, 15)$$

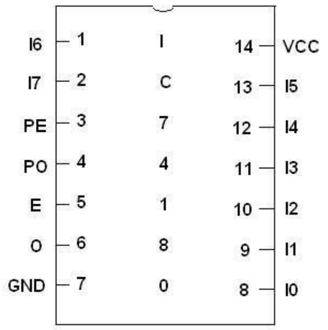
The above expression is reduced as,

$$X = (A \oplus B \oplus C \oplus P)$$

PROCEDURE:

1. Connections are given as per the circuit diagrams.
2. For all the ICs 7^{th} pin is grounded and 14^{th} pin is given +5 V supply.
3. Apply the inputs and verify the truth table for the Parity generator and checker.

PIN DIAGRAM FOR IC 74180:

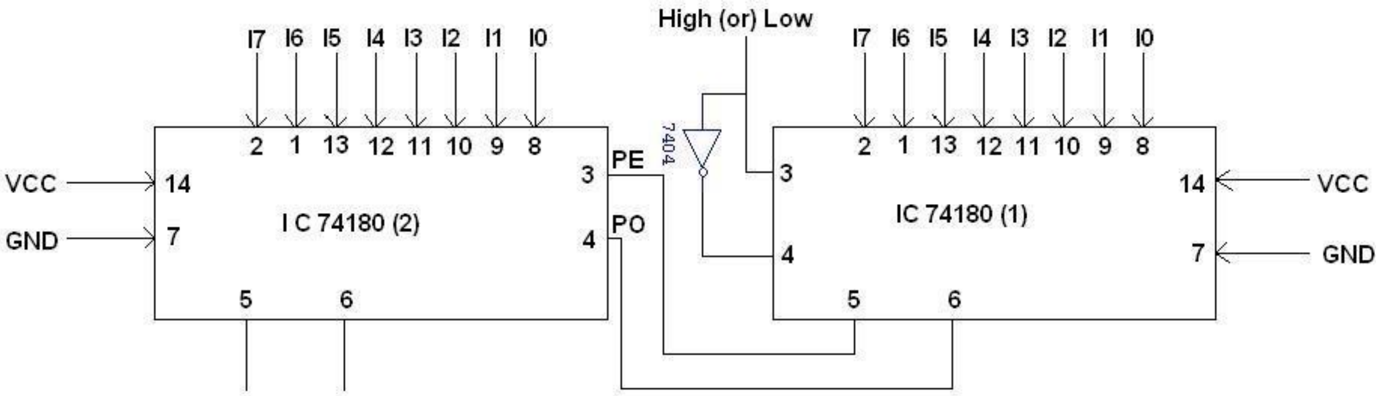


FUNCTION TABLE:

INPUTS			OUTPUTS	
Number of High Data Inputs (I0 – I7)	**PE**	**PO**	**∑E**	**∑O**
EVEN	1	0	1	0
ODD	1	0	0	1
EVEN	0	1	0	1
ODD	0	1	1	0
X	1	1	0	0
X	0	0	1	1

16 BIT ODD/EVEN PARITY GENERATOR

LOGIC DIAGRAM:

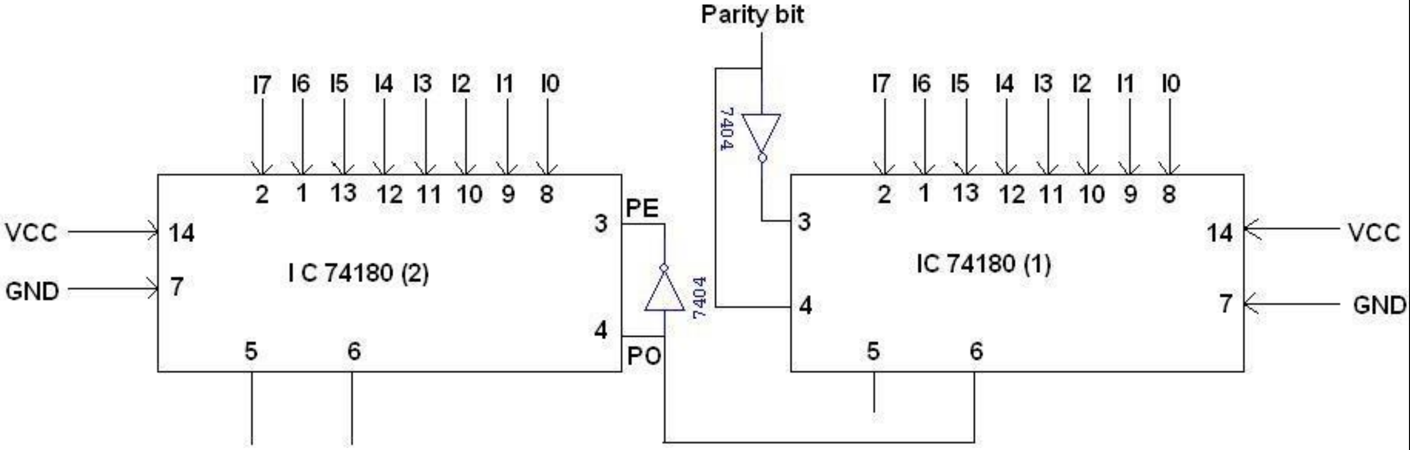


TRUTH TABLE:

I7 I6 I5 I4 I3 I2 I1 I0	I7 I6 I5 I4 I3 I2 I1 I0	Active	∑E	∑O
1 1 0 0 0 0 0 0	**1 1 0 0 0 0 0 0**	**1**	**1**	**0**
1 1 0 0 0 0 0 0	**1 1 0 0 0 0 0 0**	**0**	**0**	**1**
1 1 0 0 0 0 0 0	**0 1 0 0 0 0 0 0**	**0**	**1**	**0**

16 BIT ODD/EVEN PARITY CHECKER

LOGIC DIAGRAM



TRUTH TABLE:

I7 I6 I5 I4 I3 I2 I1 I0	I7'I6'I5'I4'I3'I2'11' I0'	Active	∑E	∑O
0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0	1	1	0
0 0 0 0 0 1 1 0	0 0 0 0 0 1 1 0	0	1	0
0 0 0 0 0 1 1 0	0 0 0 0 0 1 1 0	1	0	1

RESULT:

Thus the three bit and 16 bit odd Parity generator and checker circuits were designed, implemented and their truth tables were verified.

Ex.No.-4c **MAGNITUDE COMPARATOR**

AIM:

To design and implement the magnitude comparator using MSI device.

APPARATUS REQUIRED:

SL.NO.	COMPONENT	SPECIFICATION	QTY.
1.	AND GATE	IC 7408	2
2.	X-OR GATE	IC 7486	1
3.	OR GATE	IC 7432	1
4.	NOT GATE	IC 7404	1
5.	4-BIT MAGNITUDE COMPARATOR	IC 7485	2
6.	IC TRAINER KIT	-	1
7.	PATCH CORDS	-	30

THEORY:

The comparison of two numbers is an operator that determine one number is greater than, less than (or) equal to the other number. A magnitude comparator is a combinational circuit that compares two numbers A and B and determine their relative magnitude. The outcome of the comparator is specified by three binary variables that indicate whether A>B, A=B (or) A<B.

$A = A_3\ A_2\ A_1\ A_0$

$B = B_3\ B_2\ B_1\ B_0$

The equality of the two numbers and B is displayed in a combinational circuit designated by the symbol (A=B).

This indicates A greater than B, then inspect the relative magnitude of pairs of significant digits starting from most significant position. A is 0 and that of B is 0.

We have A<B, the sequential comparison can be expanded as

$A>B = A3B_3^1 + X_3A_2B_2^1 + X_3X_2A_1B_1^1 + X_3X_2X_1A_0B_0^1$

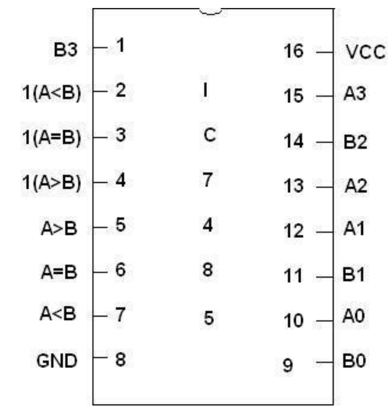
$A<B = A_3^1B_3 + X_3A_2^1B_2 + X_3X2A_1^1B_1 + X_3X_2X_1A_0^1B_0$

The same circuit can be used to compare the relative magnitude of two BCD digits. Where, A = B is expanded as,

$$A = B = (A_3 + B_3)\ (A_2 + B_2)\ (A_1 + B_1)\ (A_0 + B_0)$$

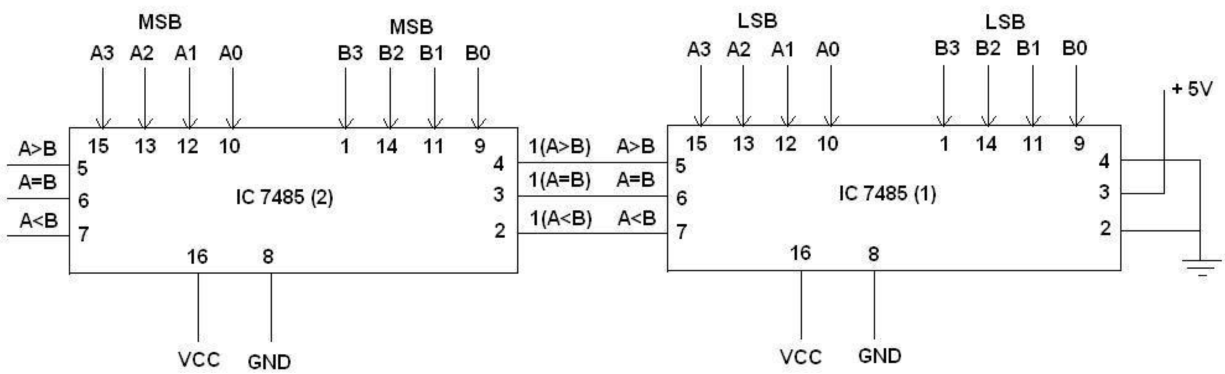
$X_3 \qquad X_2 \qquad X_1 \qquad X_0$

PIN DIAGRAM FOR IC 7485:



<u>8-BIT MAGNITUDE COMPARATOR</u>

LOGIC DIAGRAM:



TRUTH TABLE:

A		B		A>B	A=B	A<B
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0	1	0
0 0 0 1	0 0 0 1	0 0 0 0	0 0 0 0	1	0	0
0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1	0	0	1

PROCEDURE:

(i) Connections are given as per circuit diagram.
(ii) Logical inputs are given as per circuit diagram.
(iii) Observe the output and verify the truth table.

RESULT:

Thus the magnitude comparator using MSI device is designed and implemented.

Ex.No.-4d MULTIPLEXER AND DEMULTIPLEXER

AIM:

To design and implement the multiplexer and demultiplexer using logic gates and study of IC 74150 and IC 74154.

APPARATUS REQUIRED:

SL.NO.	COMPONENT	SPECIFICATION	QTY.
1.	3 I/P AND GATE	IC 7411	2
2.	OR GATE	IC 7432	1
3.	NOT GATE	IC 7404	1
2.	IC TRAINER KIT	-	1
3.	PATCH CORDS	-	32

THEORY:

MULTIPLEXER:

Multiplexer means transmitting a large number of information units over a smaller number of channels or lines. A digital multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally there are 2^n input line and n selection lines whose bit combination determine which input is selected.

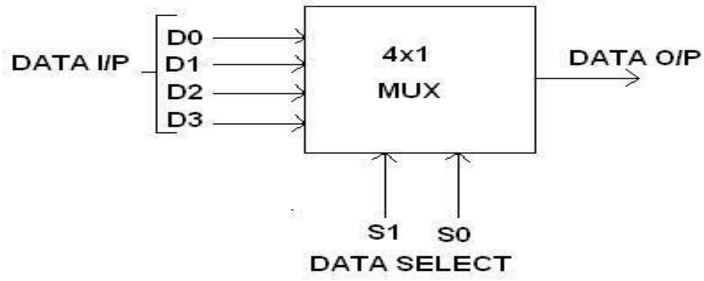
DEMULTIPLEXER:

The function of Demultiplexer is in contrast to multiplexer function. It takes information from one line and distributes it to a given number of output lines. For this reason, the demultiplexer is also known as a data distributor. Decoder can also be used as demultiplexer.

In the 1: 4 demultiplexer circuit, the data input line goes to all of the AND gates. The data select lines enable only one gate at a time and the data on the data input line will pass through the selected gate to the associated data output line.

4:1 MULTIPLEXER

BLOCK DIAGRAM FOR 4:1 MULTIPLEXER:



FUNCTION TABLE:

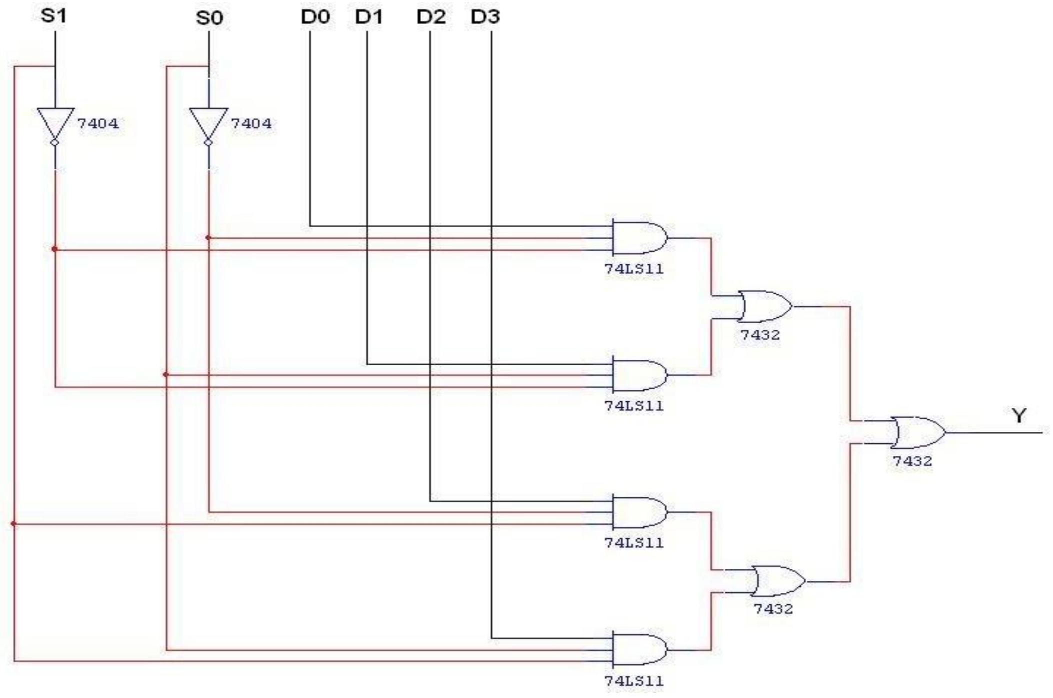
S1	S0	INPUTS Y
0	0	D0 → D0 S1’ S0’
0	1	D1 → D1 S1’ S0
1	0	D2 → D2 S1 S0’
1	1	D3 → D3 S1 S0

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

TRUTH TABLE:

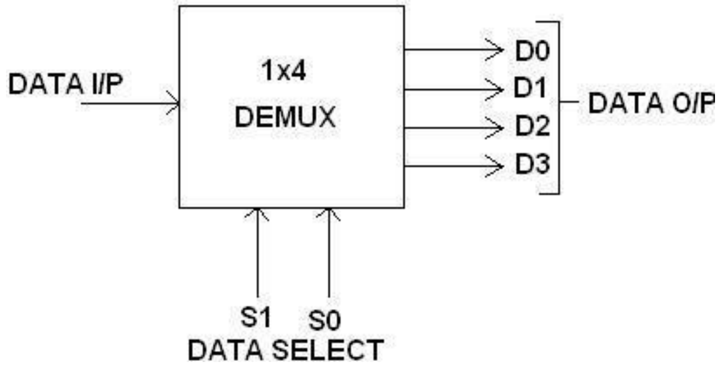
S1	S0	Y = OUTPUT
0	0	D0
0	1	D1
1	0	D2
1	1	D3

CIRCUIT DIAGRAM FOR MULTIPLEXER:



1:4 DEMULTIPLEXER

BLOCK DIAGRAM FOR 1:4 DEMULTIPLEXER:



FUNCTION TABLE:

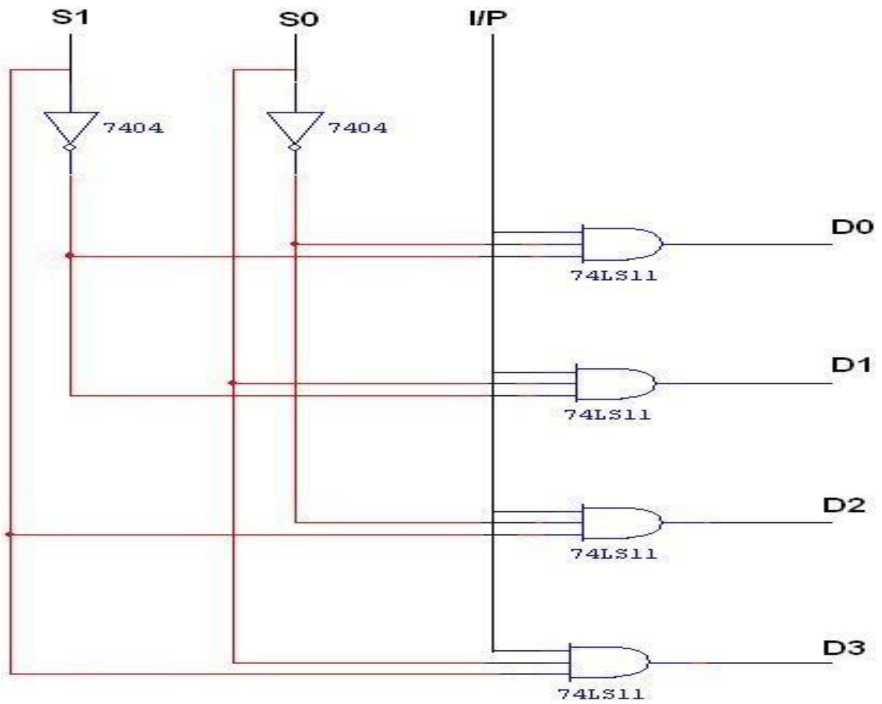
S1	S0	INPUT
0	0	X → D0 = X S1’ S0’
0	1	X → D1 = X S1’ S0
1	0	X → D2 = X S1 S0’
1	1	X → D3 = X S1 S0

Y = X S1’ S0’ + X S1’ S0 + X S1 S0’ + X S1 S0

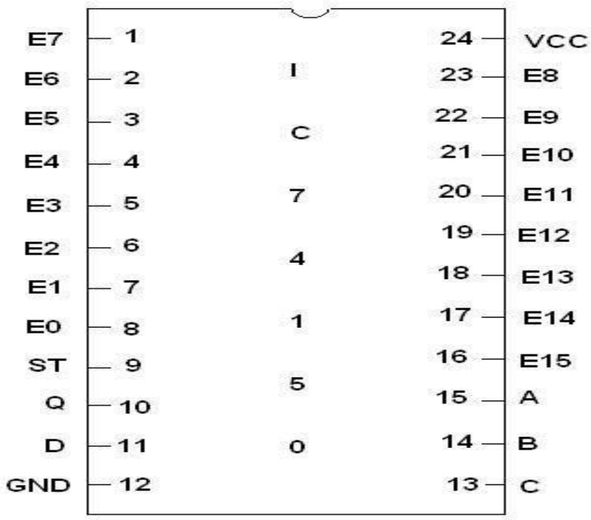
TRUTH TABLE:

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

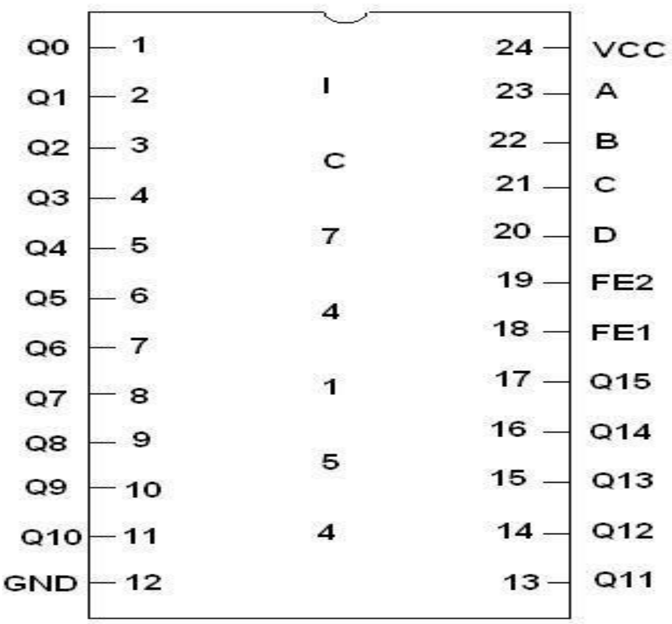
LOGIC DIAGRAM FOR DEMULTIPLEXER:



PIN DIAGRAM FOR IC 74150:



PIN DIAGRAM FOR IC 74154:



PROCEDURE:

(i) Connections are given as per circuit diagram.
(ii) Logical inputs are given as per circuit diagram.
(iii) Observe the output and verify the truth table.

RESULT:

Thus the multiplexer and demultiplexer using logic gates are designed and implemented.

Ex.No.-5 SHIFT REGISTER

AIM:

To design and implement the following shift registers

(i) Serial in serial out
(ii) Serial in parallel out
(iii) Parallel in serial out
(iv) Parallel in parallel out

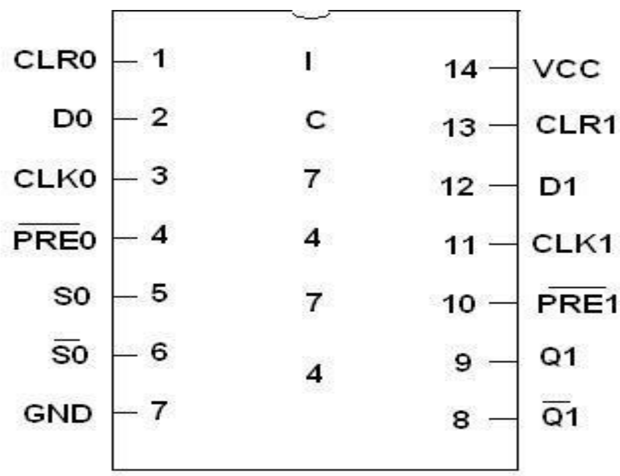
APPARATUS REQUIRED:

SL.NO.	COMPONENT	SPECIFICATION	QTY.
1.	D FLIP FLOP	IC 7474	2
2.	OR GATE	IC 7432	1
3.	IC TRAINER KIT	-	1
4.	PATCH CORDS	-	35

THEORY:

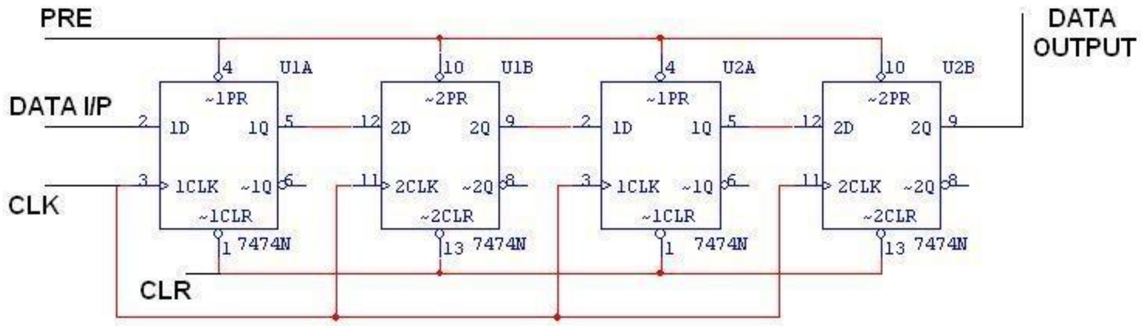
A register is capable of shifting its binary information in one or both directions is known as shift register. The logical configuration of shift register consist of a D-Flip flop cascaded with output of one flip flop connected to input of next flip flop. All flip flops receive common clock pulses which causes the shift in the output of the flip flop. The simplest possible shift register is one that uses only flip flop. The output of a given flip flop is connected to the input of next flip flop of the register. Each clock pulse shifts the content of register one bit position to right.

PIN DIAGRAM OF IC 7474:



SERIAL IN SERIAL OUT

LOGIC DIAGRAM:

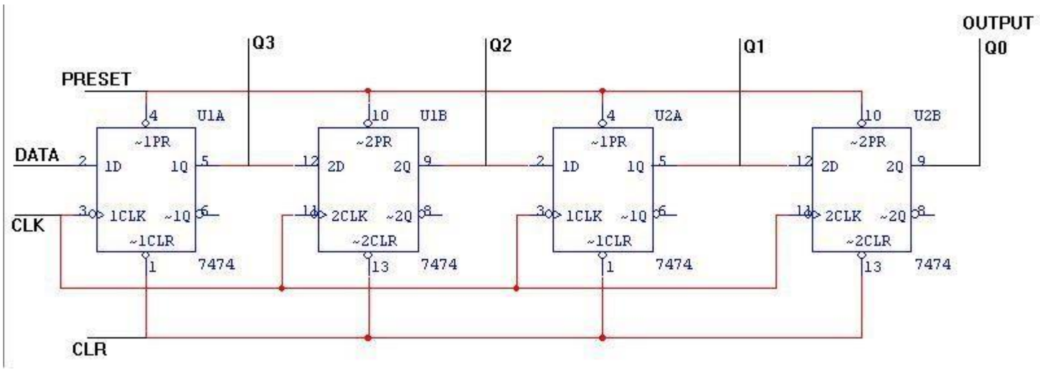


TRUTH TABLE:

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

SERIAL IN PARALLEL OUT

LOGIC DIAGRAM:

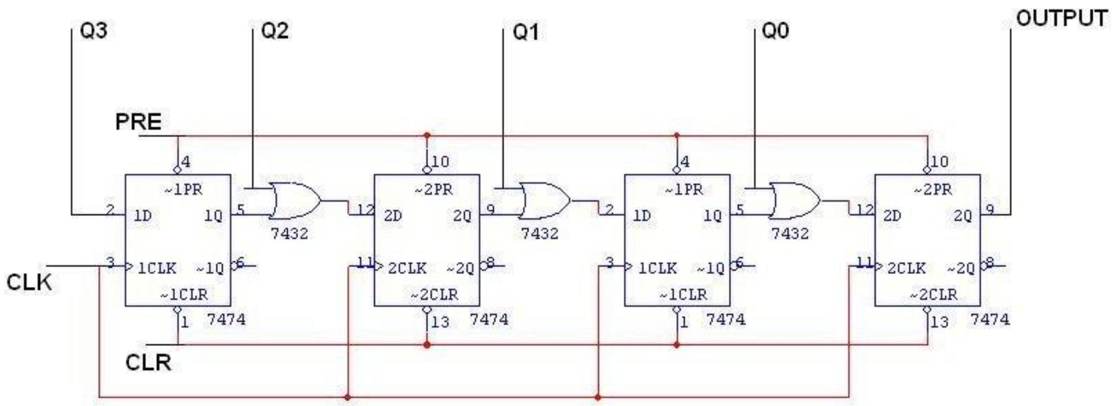


TRUTH TABLE:

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

PARALLEL IN SERIAL OUT

LOGIC DIAGRAM:

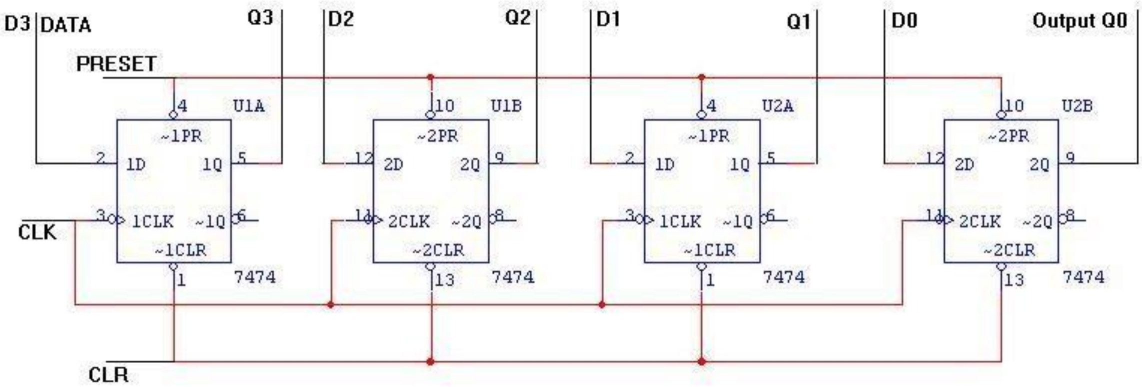


TRUTH TABLE:

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

PARALLEL IN PARALLEL OUT

LOGIC DIAGRAM:



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

PROCEDURE:

(i) Connections are given as per circuit diagram.
(ii) Logical inputs are given as per circuit diagram.
(iii) Observe the output and verify the truth table.

RESULT:

The Serial in serial out, Serial in parallel out, Parallel in serial out and Parallel in parallel out shift registers are designed and implemented.

Ex.No.-6 SYNCHRONOUS AND ASYNCHRONOUS COUNTER

AIM:

To design and implement synchronous and asynchronous counter.

APPARATUS REQUIRED:

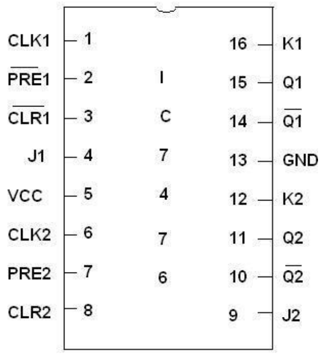
S.NO.	NAME OF THE APPARATUS	RANGE	QUANTITY
1.	Digital IC trainer kit		1
2.	JK Flip Flop	IC 7473	2
3.	D Flip Flop	IC 7473	1
4.	NAND gate	IC 7400	1
5.	Connecting wires		As required

THEORY:

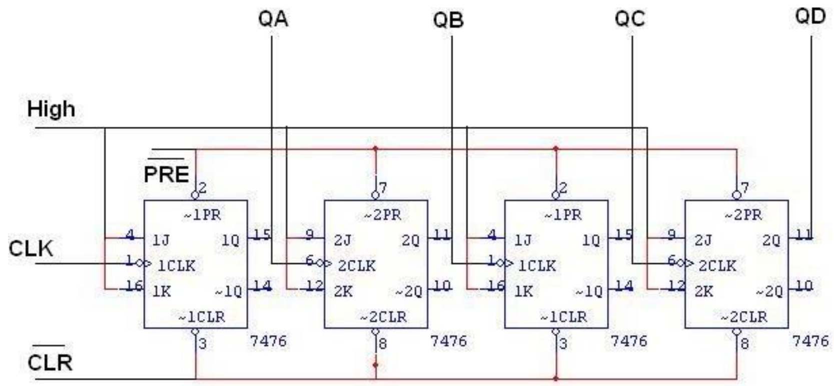
Asynchronous decade counter is also called as ripple counter. In a ripple counter the flip flop output transition serves as a source for triggering other flip flops. In other words the clock pulse inputs of all the flip flops are triggered not by the incoming pulses but rather by the transition that occurs in other flip flops. The term asynchronous refers to the events that do not occur at the same time. With respect to the counter operation, asynchronous means that the flip flop within the counter are not made to change states at exactly the same time, they do not because the clock pulses are not connected directly to the clock input of each flip flop in the counter.

A counter is a register capable of counting number of clock pulse arriving at its clock input. Counter represents the number of clock pulses arrived. A specified sequence of states appears as counter output. This is the main difference between a register and a counter. There are two types of counter, synchronous and asynchronous. In synchronous common clock is given to all flip flop and in asynchronous first flip flop is clocked by external pulse and then each successive flip flop is clocked by Q or Q output of previous stage. A soon the clock of second stage is triggered by output of first stage. Because of inherent propagation delay time all flip flops are not activated at same time which results in asynchronous operation.

PIN DIAGRAM FOR IC 7476:



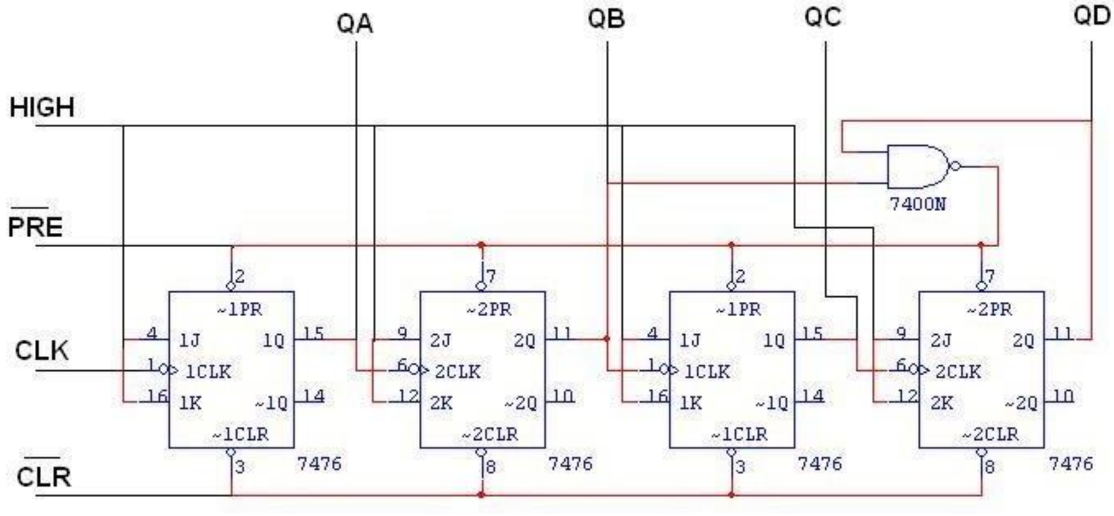
CIRCUIT DIAGRAM:



TRUTH TABLE:

CLK	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	1	1
13	1	0	1	1
14	0	1	1	1
15	1	1	1	1

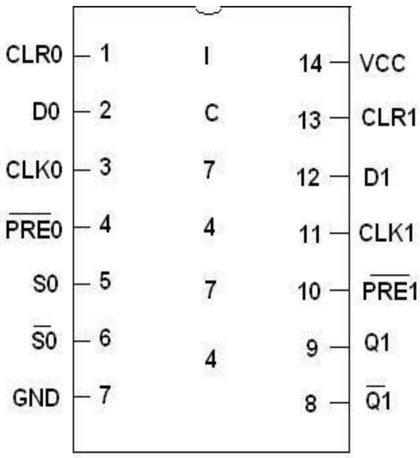
LOGIC DIAGRAM FOR MOD - 10 RIPPLE COUNTER:



TRUTH TABLE:

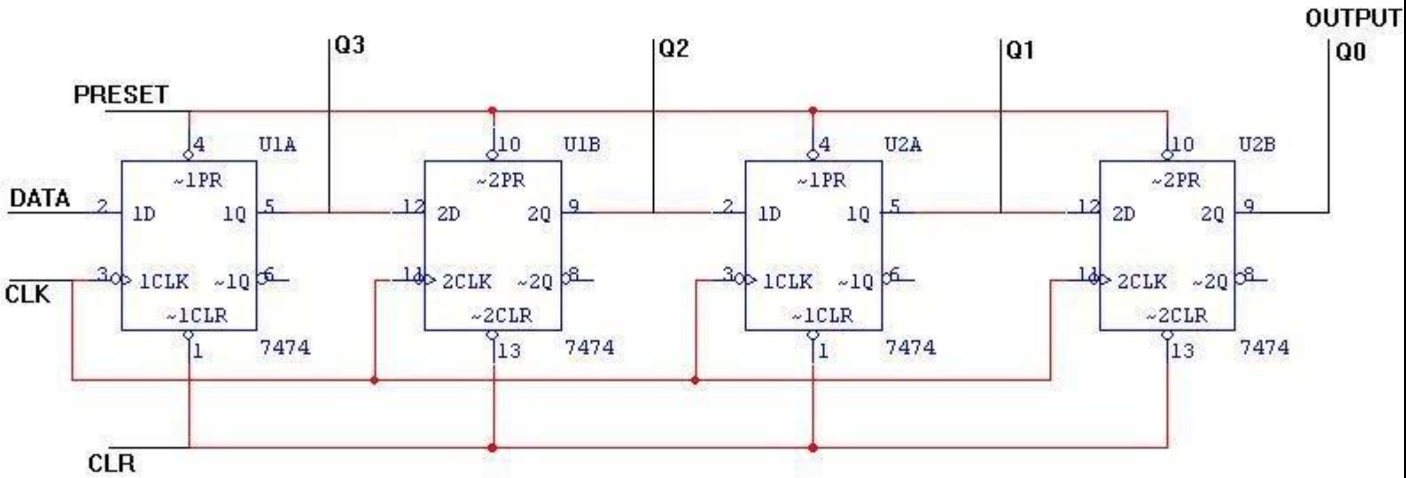
CLK	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	0	0	0

PIN DIAGRAM:



SYNCHRONOUS COUNTER

LOGIC DIAGRAM:



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

PROCEDURE:

(i) Connections are given as per circuit diagram.
(ii) Logical inputs are given as per circuit diagram.
(iii) Observe the output and verify the truth table.

RESULT:

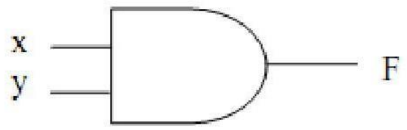
Thus the synchronous and asynchronous counter are designed and implemented.

Ex.No.- 7 IMPLEMENTATION OF BASIC LOGIC GATES

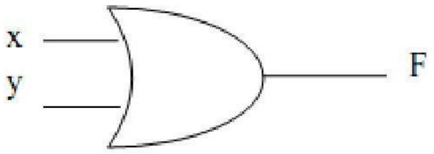
AIM:

To implement all the basic logic gates using Verilog and VHDL simulator.

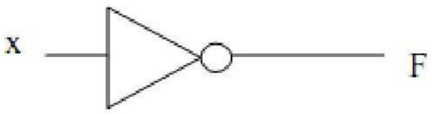
LOGIC GATE SYMBOLS **TRUTH TABLES**



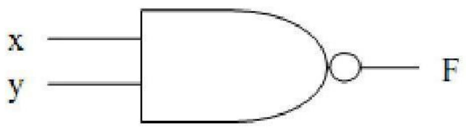
2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1



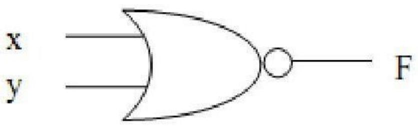
2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1



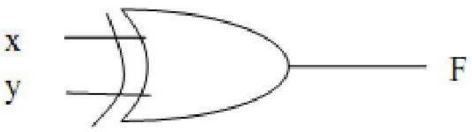
NOT gate	
A	$\overline{A}$
0	1
1	0



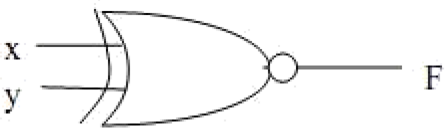
2 Input NAND gate		
A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0



2 Input NOR gate		
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0



2 Input EXOR gate		
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0



2 Input EXNOR gate		
A	B	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

VERILOG CODE

AND GATE
module and12(a,b,c);
input a;
input b;
output c;
assign c = a & b;
endmodule
	OR GATE
	module or12(a,b,d);
	input a;
	input b;
	output d;
	assign d = a | b;
	endmodule

NAND GATE *module nand12(a,b,e);* *input a;* *input b;* *output e;* *assign e = ~(a & b);* *endmodule*	**XOR GATE** *module xor12(a,b,h);* *input a;* *input b;* *output h;* *assign h = a ^ b;* *endmodule*
XNOR GATE *module xnor12(a,b,i);* *input a;* *input b;* *output i;* *assign i = ~(a ^ b);* *endmodule*	**NOR GATE** *module nor12(a,b,f);* *input a;* *input b;* *output f;* *assign f = ~(a | b);* *endmodule*
NOT GATE *module not12(a,g);* *input a;* *output g;* *assign g = ~a;* *endmodule*	

AND GATE

VERILOG CODE:

```
module and12(a,b,c);
   input a;
   input b;
   output c;
   assign c = a & b;
endmodule
```

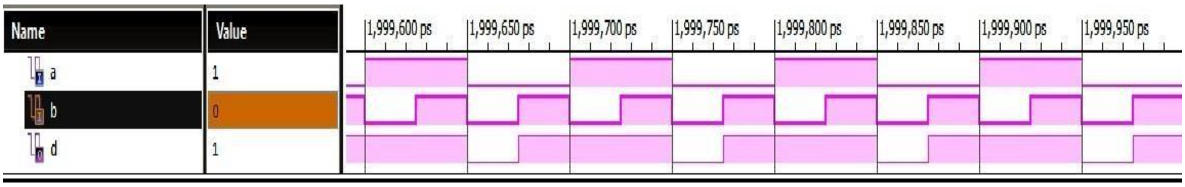
OUTPUT WAVEFORM:

OR GATE

VERILOG CODE:

```
module or12(a,b,d);
   input a;
   input b;
   output d;
   assign d = a | b;
endmodule
```

OUTPUT WAVEFORM:

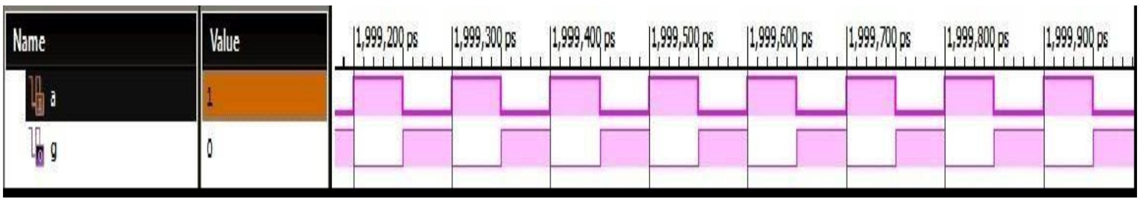


NOT GATE

VERILOG CODE:

```
module not12(a,g);
   input a;
   output g;
   assign g = ~a;
endmodule
```

OUTPUT WAVEFORM:



EX-OR GATE

VERILOG CODE:

```
module xor12(a,b,h);
        input a;
        input b;
        output h;
        assign h = a^ b;
endmodule
```

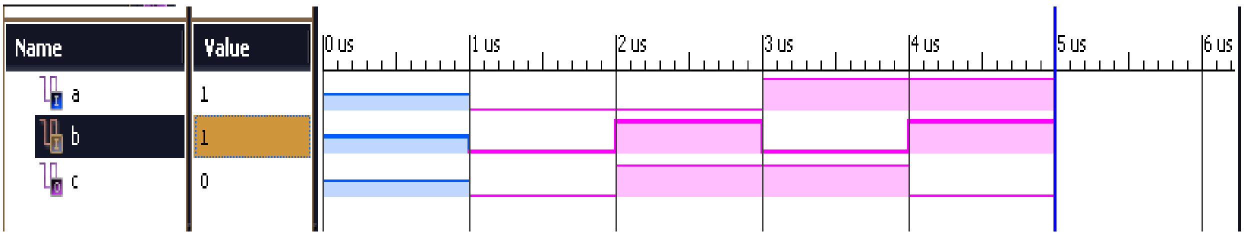
VHDL CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity xor_gate is
port (a,b : in std_logic ;
      c : out std_logic);

end xor_gate;
architecture Behavioral o f xor_gate is
begin
c <= a xor b;
end Behavioral;
```

OUTPUT WAVEFORM:



RESULT:

Thus all the basic logic gates are implemented and verified using Verilog and VHDL simulator.

Ex.No.-8 COMBINATIONAL AND SEQUENTIAL CIRCUITS

AIM:

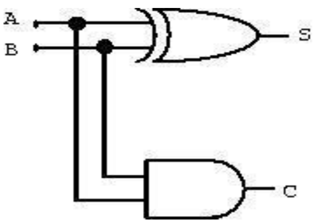
To simulate the sequential and combinational circuits using HDL simulator (Verilog and VHDL).

1. HALF ADDER

Truth Table

Input		Output	
A	B	S(Sum)	C(Carry)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Circuit Diagram



Graphical Notation

Equations

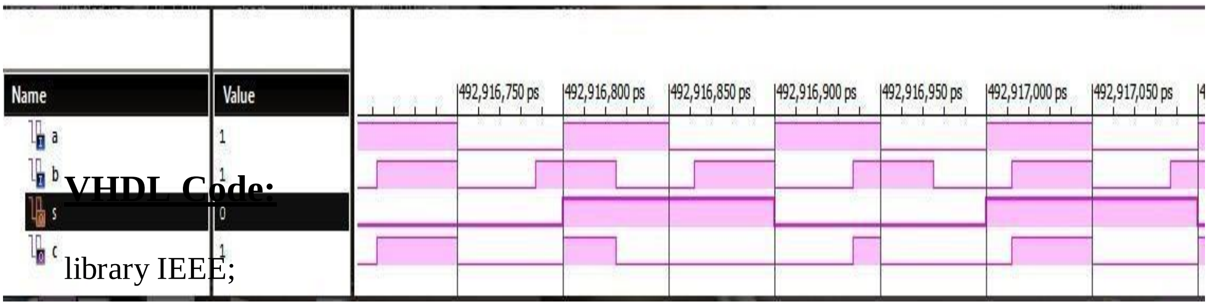
S (Sum) =A^B

C (Carry) =AB

Verilog Code:

```
module hadd(a,b,s,c); input a;
input b;
output s;
output c;
assign s = a ^ b;
assign c = a & b;
endmodule
```

Output:



VHDL Code:

```
library IEEE;
```

use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL; entity halfadder is
port(
a : in std_logic; b : in std_logic;
sum : out std_logic; carry : out std_logic);
end halfadder;
architecture Behavioral of halfadder is begin
sum <= (a xor b); carry <= (a and b); end Behavioral;

Input:

a : 1 ;

b :1;

Sum : 0

Carry : 1

Output:

2. **FULL ADDER**

Truth Table

Input			Output	
A	**B**	**C**	**SUM**	**Cout**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

K- Map for sum

A \ BC	B'C'	B'C	BC	BC'
A'	0	1	0	1
A	1	0	1	0

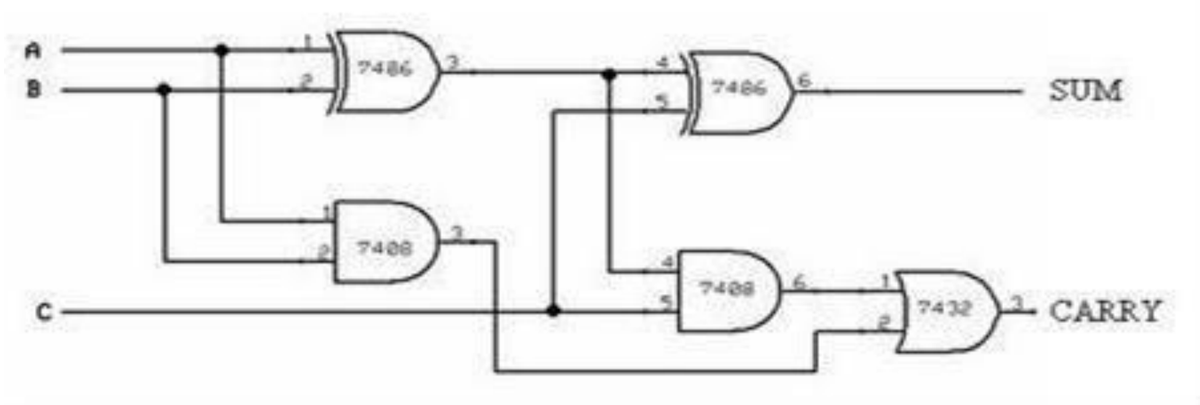
K-map for Carry

A \ BC	B'C'	B'C	BC	BC'
A'	0	0	1	0
A	0	1	1	1

H.ADDER **SUM** = **A'B'C + A'BC' + AB'C' + ABC** **Cout** = **A'BC + AB'C + ABC' +ABC**

SUM= **A^B^C** Cout= **(A^B)C+AB**

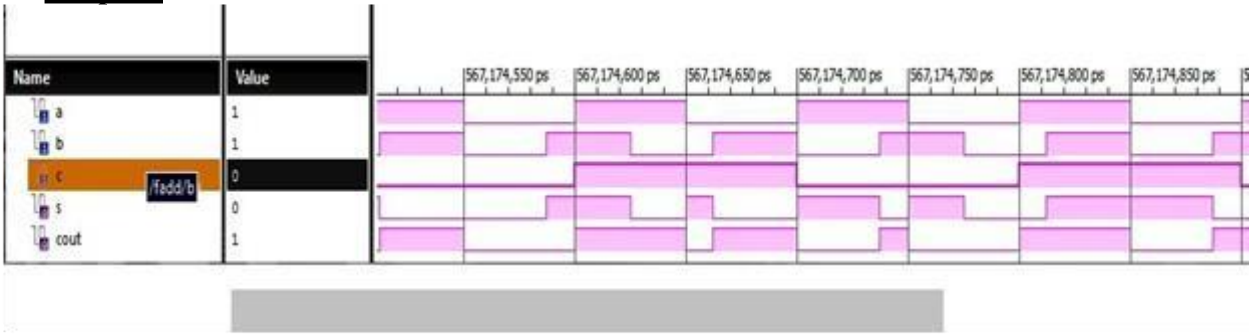
Circuit Diagram:



Verilog Code:

```
module fadd(a,b,c,s,cout);
   input a;
   input b;
   input c; output s;
   output cout;
   assign s = (a ^ b) ^ c;
   assign cout = (a & b)|( b & c)|(c & a);
endmodule
```

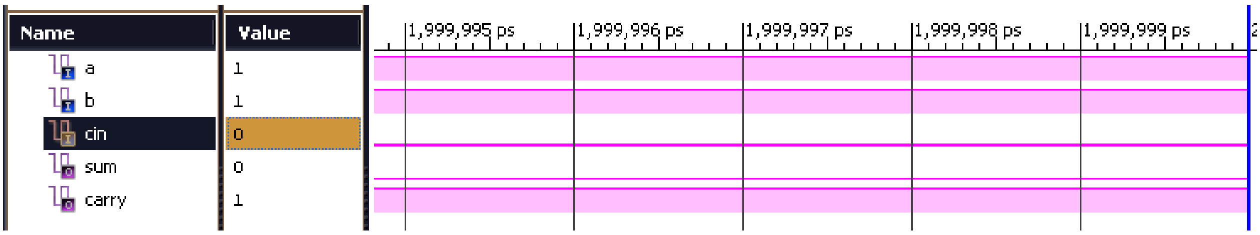
Output :



VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity fulladder is
port(
a : in std_logic;
b : in std_logic;
cin : in std_logic;
sum : out std_logic;
carry : out std_logic
);
end fulladder;
architecture Behavioral of fulladder is
begin
sum <= (a xor b xor cin);
carry <= (a and b) or (b and cin) or (a and cin);
end Behavioral;
```

Output:



3. HALF SUBTRACTOR

Verilog Code:

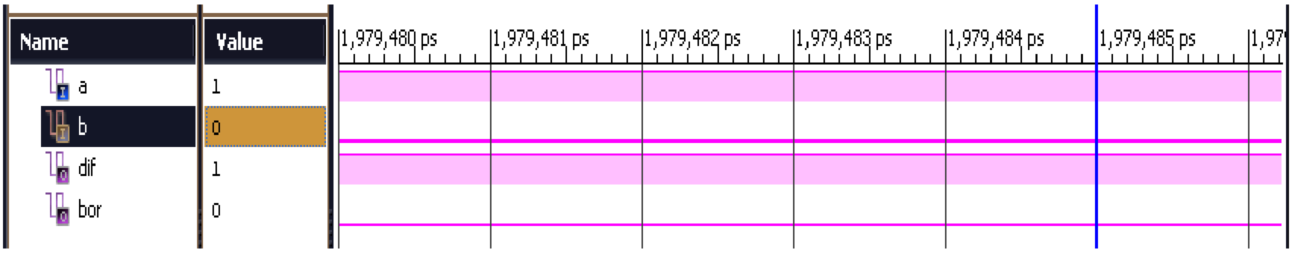
```
module hsub(a,b,d,bor);
Input a;
Input b;
output d;
output bor;
assign d=)a^b);
assign bor = (~a&~b);
end module
```

VHDL Code:

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity halfsubtractor is
port(
a : in std_logic;
b : in std_logic;
dif : out std_logic;
bor : out std_logic
);
end halfsubtractor;
architecture Behavioral of halfsubtractor is
begin
dif <= a xor b;
bor <= ((not a) and b);
end Behavioral;
```

Output:

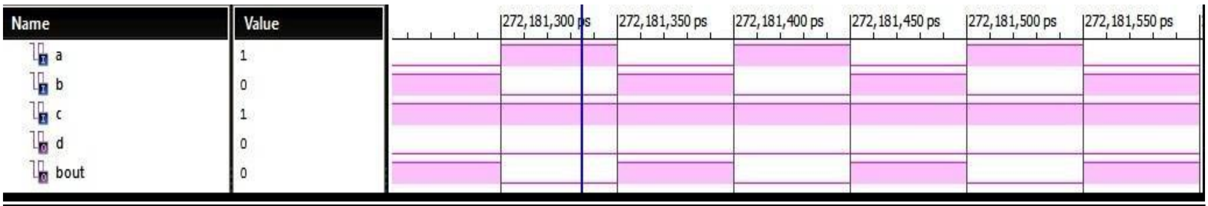


4. FULL SUBTRACTOR

Verilog Code:

```
module sub(a,b,c,d,b out);
    input a;
    input b;
    input c;
    output d;
    output bout;
    assign d = (a ^ b) ^ c;
    assign bout = (~a & b)|( b & c)|(c & ~a);
endmodule
```

Output:



VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity fullsubtractor is
port( a : in std_logic;
b : in std_logic;
cin : in std_logic;
dif : out std_logic;
bor : out std_logic );
end fullsubtractor;
architecture Behavioral of fullsubtractor is begin
dif <= a xor b xor cin;
bor <= (((not a) and b) or (( not a) and cin) or (b and cin));
end Behavioral;
```

INPUT:

a : 0 ;

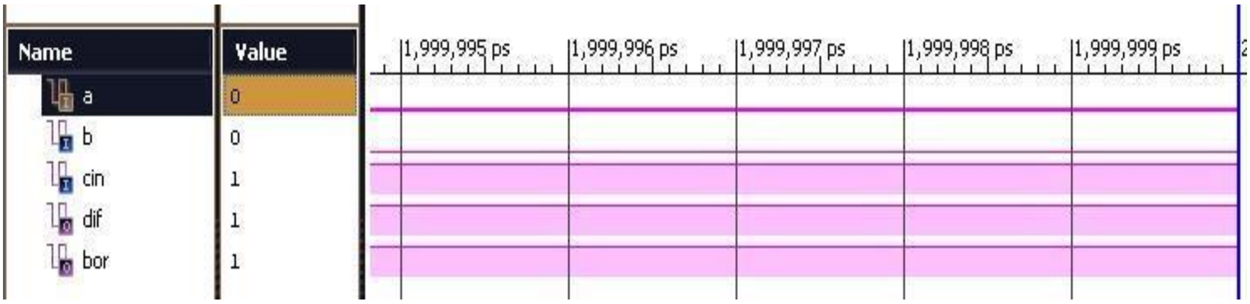
b :0;

Cin : 1

Difference : 1

Borrow : 1

Output:



5. MULTIPLEXER

Verilog Code:

```
module mux4to1(Y, I0,I1,I2,I3, sel);
   output Y;
   input I0,I1,I2,I3;
   input [1:0] sel;
   reg Y;
always @ (sel or I0 or I1 or I2 or I3)
case (sel)
```

```
        2'b00:Y=I0;
        2'b01:Y=I1;
        2'b10: Y=I2;
        2'b11: Y=I3;
endcase
endmodule
```

Output:

VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL; entity mux is
port(
inp : in std_logic_vector(3 downto 0); sel : in std_logic_vector(1 downto 0); muxout
: out std_logic --mux output line );
end mux;
architecture Behavioral of mux is begin
process(inp,sel) begin
case sel is when "00" =>
muxout <= inp(0); -- mux O/P=1 I/P-- when "01" =>
muxout <= inp(1); -- mux O/P=2 I/P-- when "10" =>
muxout <= inp(2); -- mux O/P=3 I/P-- when "11" =>
muxout <= inp(3); -- mux O/P=4 I/P-- when others =>
end case; end process;
end Behavioral;
```

Truth Table:

INPUTS						OUTPUT
sel1	sel0	inp0	inp1	inp2	inp3	muxout
0	0	I	0	0	0	I
0	1	0	I	0	0	I
1	0	0	0	I	0	I
1	1	0	0	0	I	I
NOTE : I means binary input which is either 0 or 1						

6. DEMULTIPLEXER

Verilog Code:

```
module demux(S,D,Y);
        Input [1:0] S;
        Input D;
        Output [3:0] Y; reg Y;
       always @(S OR )
       case({D,S})
           3’b100: Y=4’b0001;
           3’b101: Y=4’b0010;
           3’b110: Y=4’b0100;
           3’b111: Y=4’b1000;
           default:Y=4’b0000;
      endcase
endmodule
```

VHDL Code:

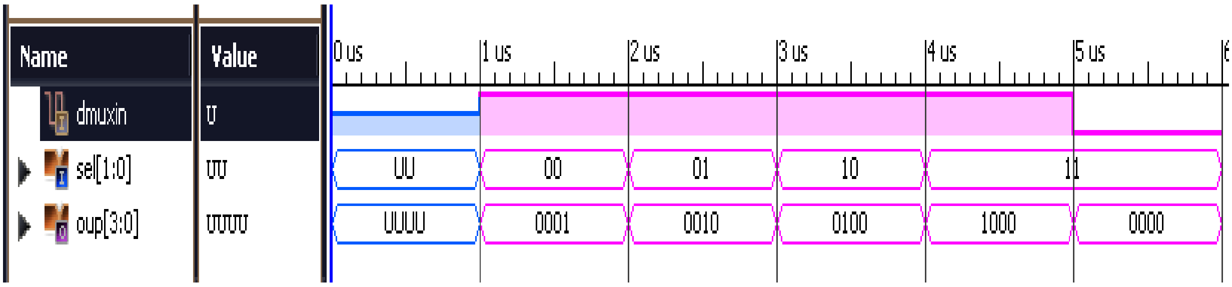
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity demux is
port(
dmuxin : in std_logic;
sel : in std_logic_vector(1 downto 0);
oup : out std_logic_vector(3 downto 0)
);
end demux;
architecture Behavioral of demux is
begin
process(dmuxin,sel)
begin
case sel is
when "00" =>
oup(0) <= dmuxin; --1 dmux o/p = dmux i/p--
oup(1) <= '0';
oup(2) <= '0';
oup(3) <= '0';
when "01" =>
oup(0) <= '0';
oup(1) <= dmuxin; --2 dmux o/p = dmux i/p--
oup(2) <= '0';
oup(3) <= '0';
when "10" =>
```

```
oup(0) <= '0';
oup(1) <= '0';
oup(2) <= dmuxin; --3 dmux o/p = dmux i/p--
oup(3) <= '0';
when "11" =>
oup(0) <= '0';
oup(1) <= '0';
oup(2) <= '0';
oup(3) <= dmuxin; --4 dmux o/p = dmux i/p--
when others =>
end case;
end process;
end Behavioral;
```

Truth Table:

INPUTS			OUTPUTS			
sel1	sel0	dmuxin	oup0	oup1	oup2	oup3
0	0	I	I	0	0	0
0	1	I	0	I	0	0
1	0	I	0	0	I	0
1	1	I	0	0	0	I
NOTE : I means binary input which is either 0 or 1						

Output:



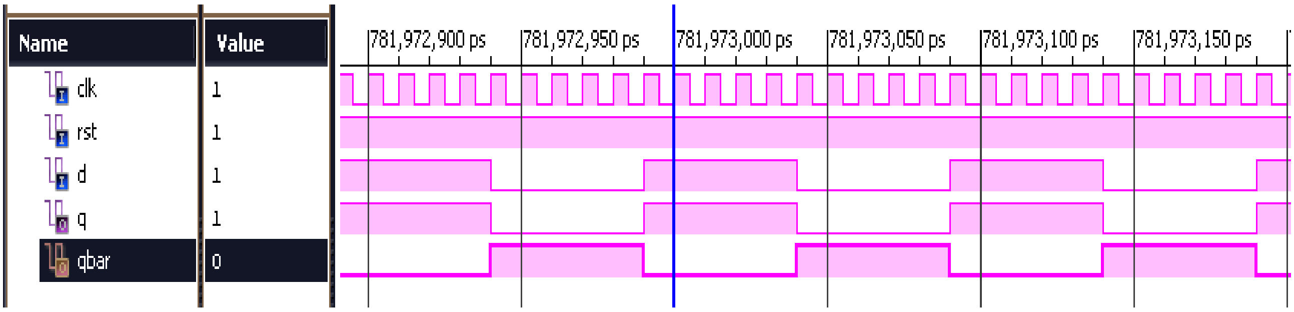
7. D FLIPFLOP

VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity dff is
```

```
port(
clk : in std_logic; --clock input
rst : in std_logic; --active low,synchronous reset
d : in std_logic; --d input
q,qbar : out std_logic --flip flop outputs ie,Qn+1 and its complement
);
end dff;
architecture Behavioral of dff is
begin
process(clk,rst)
begin
if rising_edge(clk) then
if (rst = '0') then --active low,synchronous reset
q <= '0';
qbar <= '1';
else
q <= d;
qbar <= not(d);
end if;
end if;
end process;
end Behavioral;
```

Output:



8. T FLIPFLOP

Verilog Code :

```
module tffeq(t,rst, clk,qp, qbar); input t,rst, clk;
    output qp, qbar; wire q;
    reg qp;
    always @ (posedge clk) if (rst)
             qp=0; else
              qp = q ^ t; assign qbar = ~ qp;
endmodule
```

9. JK FLIPFLOP

Verilog Code:

```
module jkff(jk,pst,clr,clk,qp,qbar);
input [1:0] jk;
input pst,clr,clk;
output qp,qbar;
   reg qp;
   wire q;
   always @ (posedge clk) if (pst)
            qp= 1;
            else
            begin
            if (clr)
            qp= 0;
            else
            begin
            case (jk)
                    2'b00: qp=q;
                    2'b01 : qp = 1'b0;
                    2'b10 : qp =1'b1;
                    2'b11 : qp = ~q;
                    default qp =0;
          endcase
          end
          end
          assign qbar = ~q;
          assign q = qp;
endmodule
```

Output:

10. RIPPLE COUNTER

Verilog Code:

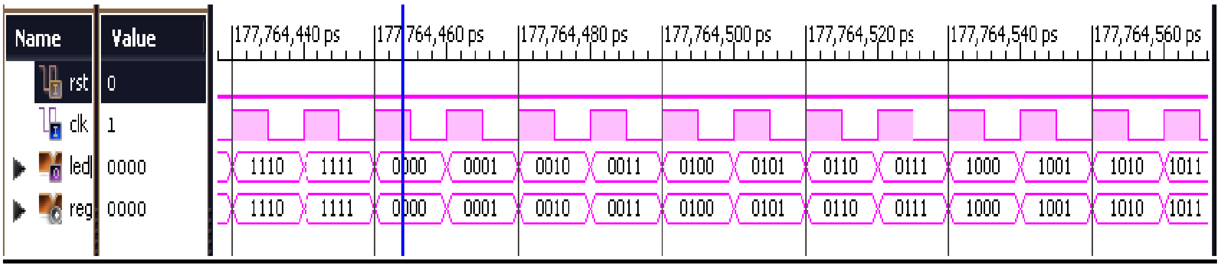
```
module ripple(clkr,st,,t,A,B,C,D);
input clk,rst,t;
output A,B,C,D;
Tff T0(D,clk,rst,t);
Tff T1(C,clk,rst,t);
Tff T2(B,clk,rst,t);
```

```
Tff T3(A,clk,rst,t);
endmodule
module Tff(q,clk,rst,t);
input clk,rst,t;
output q;
reg q;
always @(posedge clk)
begin
if(rst)
q<=1'b0; else
if(t)
q<=~q;
end
endmodule
```

VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
entity counter is
Port ( rst : in STD_LOGIC;
clk : in STD_LOGIC;
led : out std_logic_vector(3 downto 0)
);
end counter;
architecture Behavioral of counter is
signal reg :std_logic_vector(3 downto 0);
begin
process(rst,clk)
begin
if rst = '1' then
reg <= "0000";
elsif rising_edge(clk) then
reg <= reg + 1;
end if;
end process;
led(3 downto 0) <= reg(3 downto 0);
end Behavioral;
```

Output:

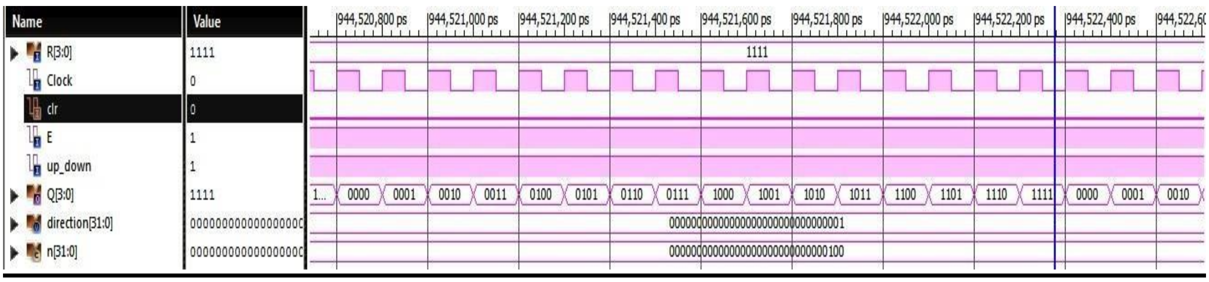


11. **UPDOWN COUNTER**

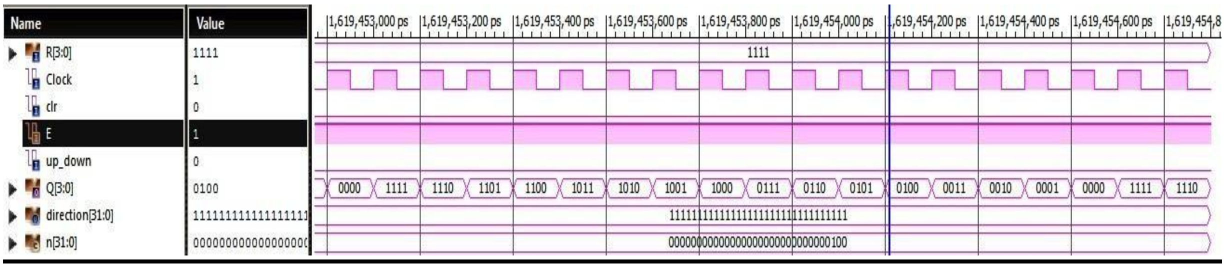
Verilog Code:

```
module updowncount (R, Clock, clr, E, up_down, Q);
parameter n = 4;
input [n-1:0] R;
input Clock, clr, E, up_down;
output [n-1:0] Q;
reg [n-1:0] Q;
integer direction;
always @(posedge Clock)
begin
if (up_down) direction = 1;
else direction = -1;
if (clr) Q <= R;
else if (E) Q <= Q + direction;
end
endmodule
```

UP Counter:



DOWN Counter:



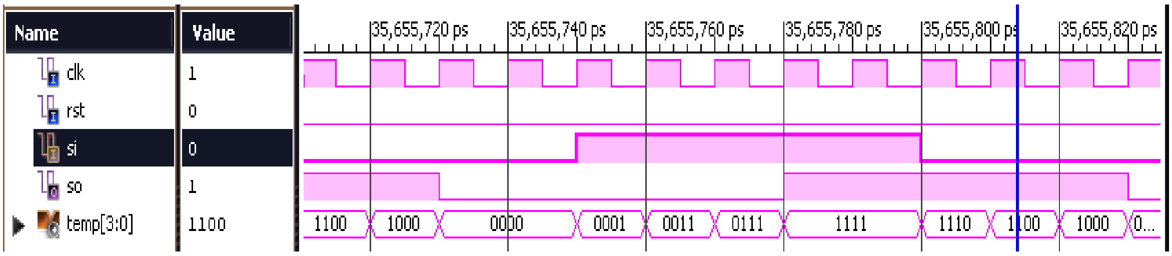
12. SHIFT REGISTER

a. Serial In Serial Out

VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--library UNISIM;
--use UNISIM.VComponents.all;
entity hj is
port(
clk : in std_logic;
rst : in std_logic;
si: in std_logic;
so: out std_logic
);
end hj;
architecture Behavioral of hj is
signal temp : std_logic_vector(3 downto 0);
begin
process(clk,rst)
begin
if rising_edge(clk) then
if rst = '1' then
temp <= (others=>'0');
else
temp <= temp(2 downto 0) & si;
end if;
end if;
end process;
so <= temp(3);
end Behavioral;
```

Output:

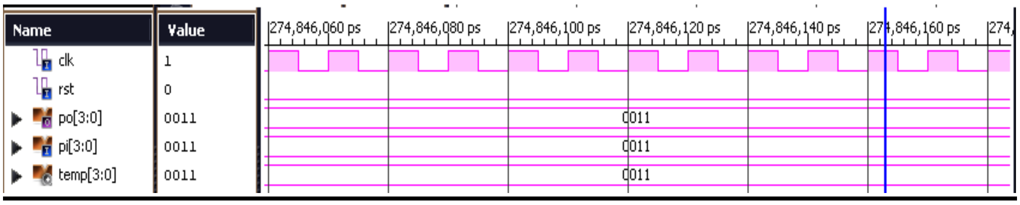


b. Parallel In Parallel Out

VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--library UNISIM;
--use UNISIM.VComponents.all;
entity hj is
port(
clk : in std_logic;
rst : in std_logic;
po: out std_logic_vector(3 downto 0);
pi: in std_logic_vector(3 downto 0)
);
end hj;
architecture Behavioral of hj is
signal temp : std_logic_vector(3 downto 0);
begin
process(clk,rst)
begin
if rising_edge(clk) then
if rst = '1' then
temp <= (others=>'0');
else
temp <= pi(3 downto 0);
end if;
end if;
end process;
po <= temp(3 downto 0);
end Behavioral;
```

Output:



RESULT:

Thus the sequential and combinational circuits are designed and implemented using HDL simulator (Verilog and VHDL).