Varuvan Vadivelan Institute of Technology

Dharmapuri – 636 703

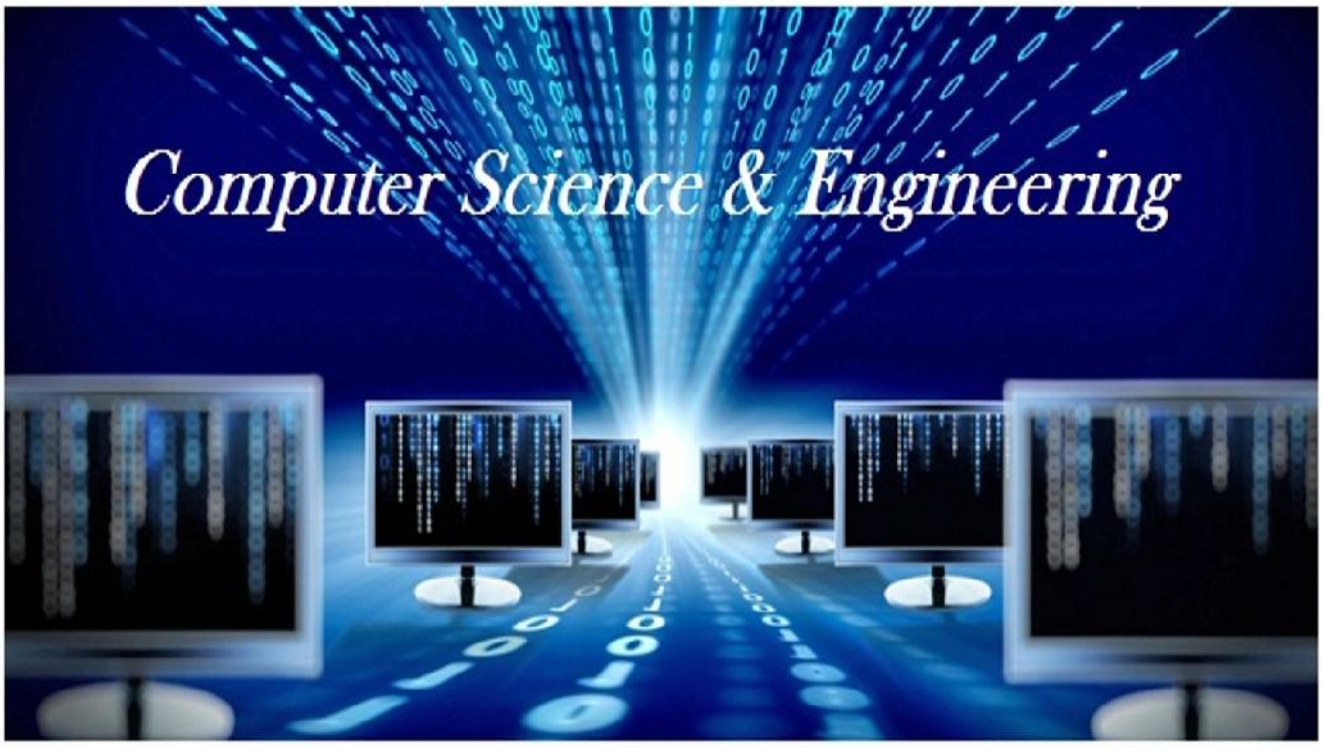
LAB MANUAL

Regulation : 2013

Branch : *B.E.* – CSE

Year & Semester : I Year / II Semester

CS6211- DIGITAL LABORATORY



ANNA UNIVERSITY: CHENNAI

REGUALTION - 2013

CS6211 - DIGITAL LABORATORY

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 implementation of combinational circuits using MSI devices:
    - 4 – bit binary adder / subtractor
    - Parity generator / checker
    - Magnitude Comparator
    - Application using multiplexers

4. Design and implementation of sequential circuits:

    - Shift –registers
    - Synchronous and asynchronous counters

5. Coding combinational / sequential circuits using HDL.

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

TOTAL: 45 PERIODS

INTRODUCTION ABOUT DIGITAL LABORATORY:

In today's modern world, the usage of digital technology is mandatory and unavoidable, applications such as internet, wireless broadcasting systems, Smart Television, computers, industry automation systems, music players etc., are really very reliable and accurate in quality and performance.

In this Lab, we learn the fundamental aspects of digital mathematical and logical operations by hardware and software (HDL simulator) methodologies.

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 and NOR are known as universal gates. 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.

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.

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 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.

A parity bit is used for detecting errors during transmission of binary information. A parity bit is an extra bit included with a binary message to make the

number is either even or odd. The message including the parity bit is transmitted and then checked at the receiver ends for errors.

An error is detected if the checked parity bit doesn't correspond to 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'. 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 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. An encoder is a digital circuit that perform inverse operation of a decoder. An encoder has 2^n input lines and n output lines. In encoder the output lines generates the binary code corresponding to the input value.

A decoder is a multiple input multiple output logic circuits which converts coded input into coded output where input and output codes are different. The input code generally has fewer bits than the output code.

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.

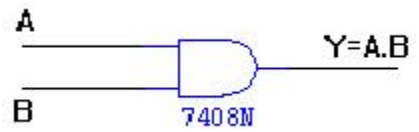
There are two types of counter, synchronous and asynchronous. A register is capable of shifting its binary information in one or both directions is known as shift register. The logical configuration of shift register consist of a D-Flip flop cascaded with output of one flip flop connected to input of next flip flop.

INDEX

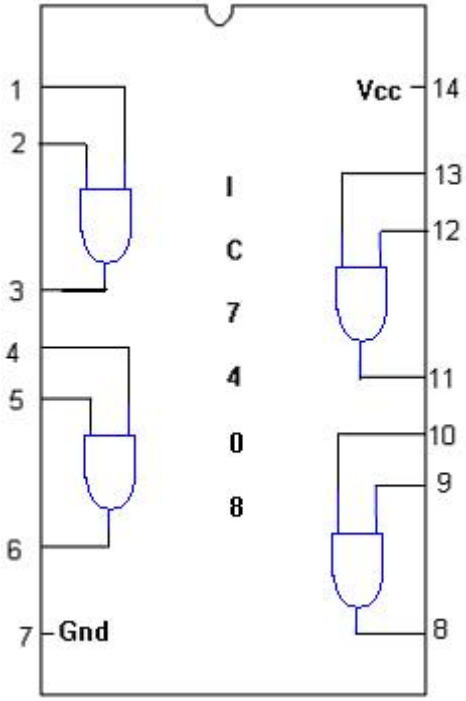
EXP No.	DATE	LIST OF EXPERIMENT	SIGNATURE OF THE STAFF	REMARKS
1		STUDY OF LOGIC GATES		
2		DESIGN OF ADDER AND SUBTRACTOR		
3		DESIGN AND IMPLEMENTATION OF CODE CONVERTORS		
4		DESIGN OF 4-BIT ADDER AND SUBTRACTOR		
5		DESIGN AND IMPLEMENTATION OF MAGNITUDE COMPARATOR		
6		16 BIT ODD/EVEN PARITY CHECKER AND GENERATOR		
7		DESIGN AND IMPLEMENTATION OF MULTIPLEXER AND DEMULTIPLEXER		
8		DESIGN AND IMPLEMENTATION OF ENCODER AND DECODER		
9		CONSTRUCTION AND VERIFICATION OF 4 BIT RIPPLE COUNTER AND MOD 10/MOD 12 RIPPLE COUNTER		
10		DESIGN AND IMPLEMENTATION OF 3 BIT SYNCHRONOUS UP/DOWN COUNTER		
11		DESIGN AND IMPLEMENTATION OF SHIFT REGISTER		
12		SIMULATION OF LOGIC GATES		
13		SIMULATION OF ADDER AND SUBTRACTOR		
14		DESIGN OF 4-BIT ADDER AND SUBTRACTOR		
15		DESIGN AND IMPLEMENTATION OF MULTIPLEXER AND DEMULTIPLEXER		
16		DESIGN AND SIMULATION OF FLIP-FLOPS		
17		DESIGN AND SIMULATION OF SHIFT REGISTER		
18		DESIGN AND IMPLEMENTATION OF DIGITAL SYSTEM (Mini Project)		

AND GATE:

SYMBOL:



PIN DIAGRAM:

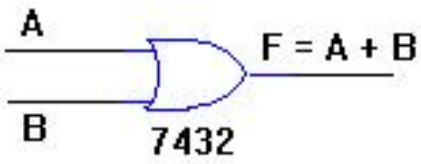


TRUTH TABLE:

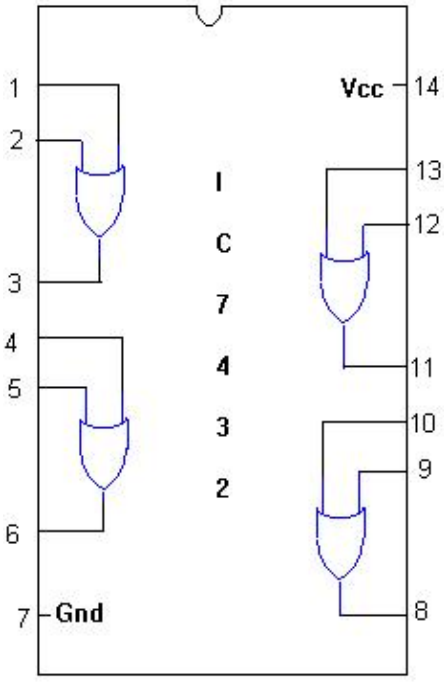
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

ORGATE:

SYMBOL:



PIN DIAGRAM:



TRUTH TABLE:

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

EX NO:1	STUDY OF LOGIC GATES
DATE:	

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	BREAD BOARD		1

THEORY:

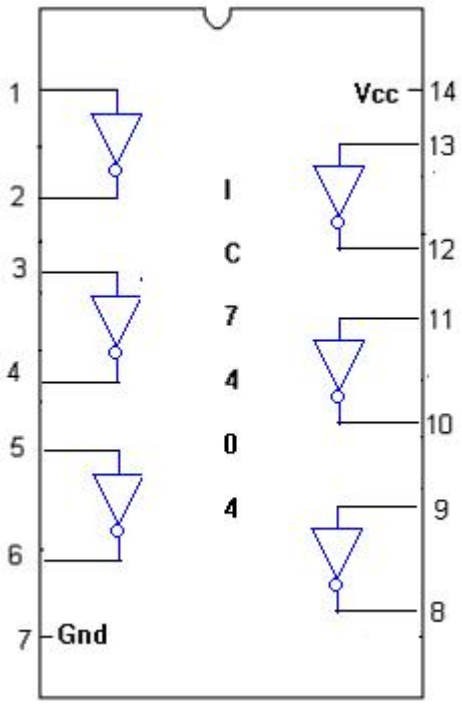
Circuit that takes the logical decision and the process are called logic gates. Each gate has one or more input and only one output.

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

NOT GATE :

SYMBOL:

PIN DIAGRAM:

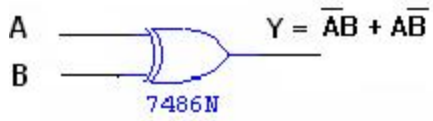


TRUTH TABLE:

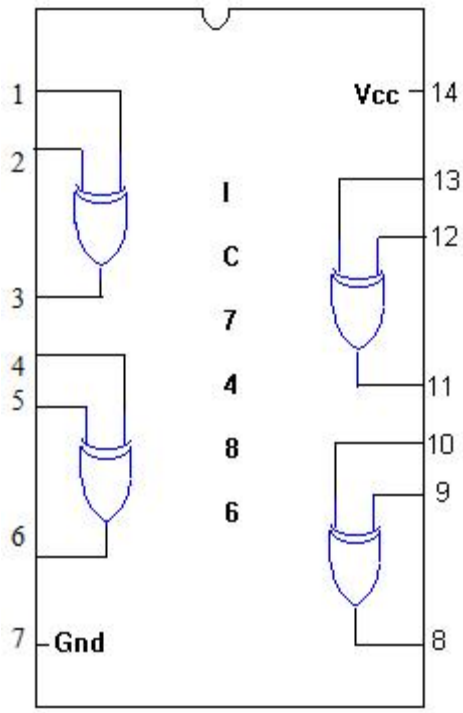
A	A’
0	1
1	0

EX-OR GATE :

SYMBOL :



PIN DIAGRAM :



TRUTH TABLE:

A	B	A’B+AB’
0	0	1
0	1	0
1	0	0
1	1	1

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

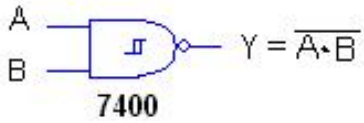
The NOT gate is called an inverter. The output is high when the input is low. The output is low when the input is high.

X-OR GATE

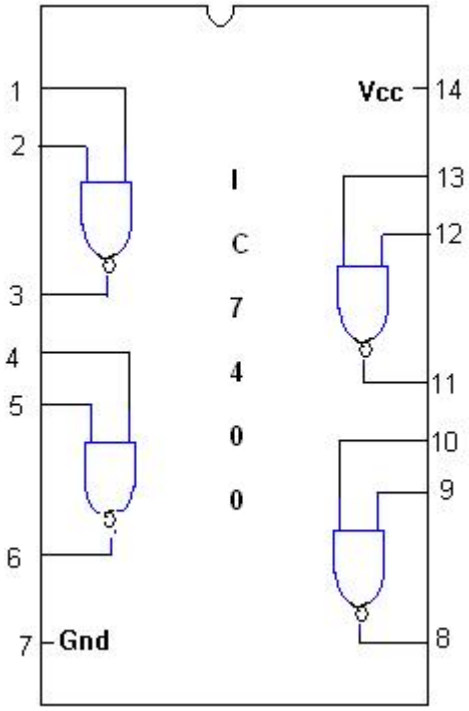
The output is high when any one of the inputs is high. The output is low when both the inputs are low and both the inputs are high.

2-INPUT NAND GATE

SYMBOL



PIN DIAGRAM

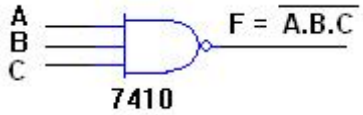


TRUTH TABLE:

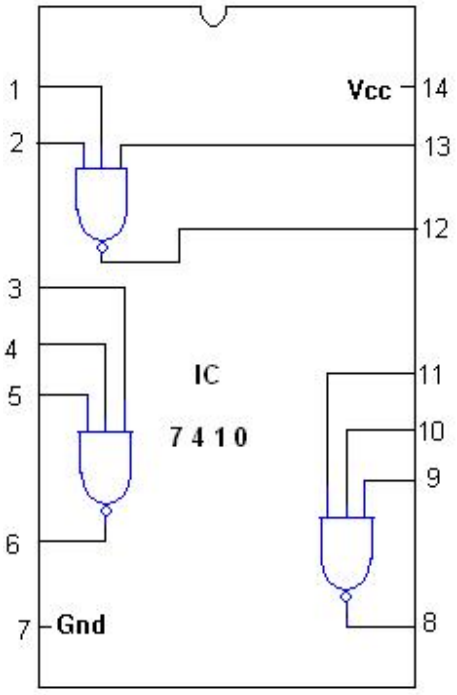
A	B	(A.B)'
0	0	1
0	1	0
1	0	0
1	1	1

3-INPUT NAND GATE

SYMBOL



PIN DIAGRAM



TRUTH TABLE:

A	B	C	(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

NAND GATE

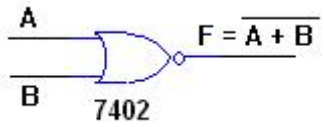
The NAND gate is a contraction of AND-NOT. The output is high when both inputs are low and any one of the input is low .The output is low level when both inputs are high.

NOR GATE

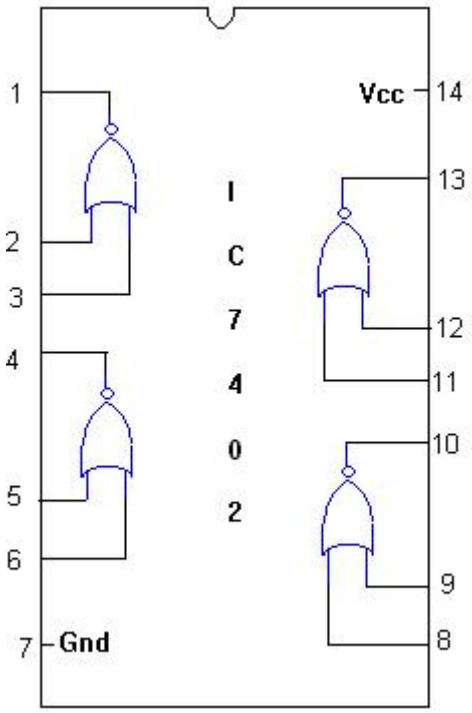
The NOR gate is a contraction of OR-NOT. The output is high when both inputs are low. The output is low when one or both inputs are high.

NOR GATE:

SYMBOL



PIN DIAGRAM



TRUTH TABLE:

A	B	(A+B)'
0	0	1
0	1	0
1	0	0
1	1	0

PROCEDURE

(i) Connections are given as per circuit diagram.

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

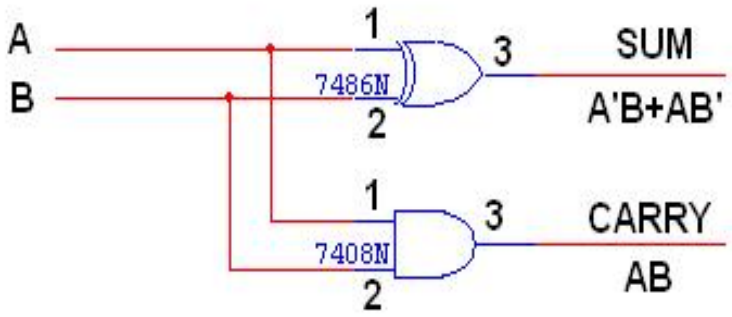
(iii) Observe the output and verify the truth table.

RESULT:

The logic gates have been studied and their truth tables have been verified.

LOGIC DIAGRAM

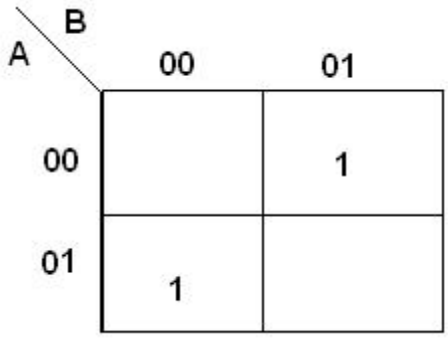
HALF ADDER



TRUTH TABLE

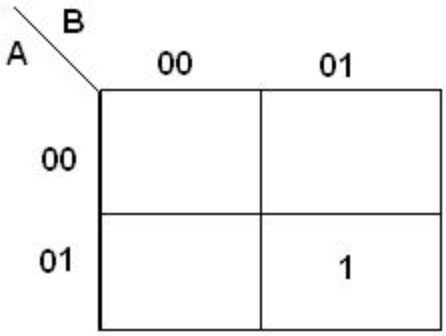
A	B	CARRY	SUM
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

K-Map for SUM



SUM = A'B + AB'

K-Map for CARRY



CARRY = AB

EX NO:2	DESIGN OF ADDER AND SUBTRACTOR
DATE:	

AIM:

To design and construct half adder, full adder, half substractor and full substractor 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.	BREADBOARD	-	1

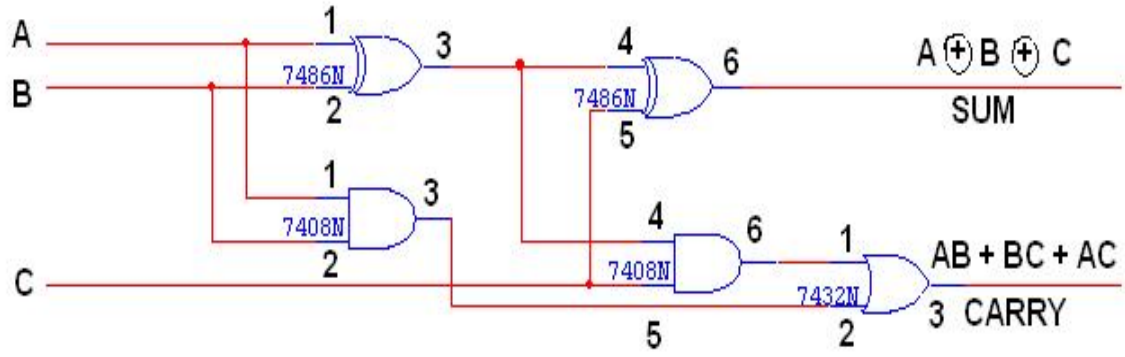
THEORY:

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.

LOGIC DIAGRAM:

FULL ADDER (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:

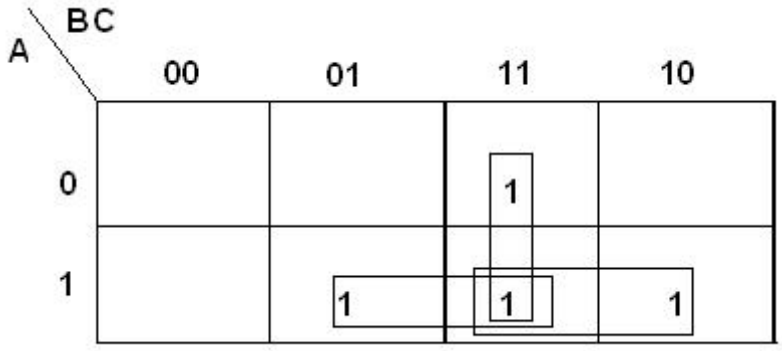
A \ BC	00	01	11	10
0		1		1
1	1		1	

SUM = A'B'C + A'BC' + ABC' + ABC

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.

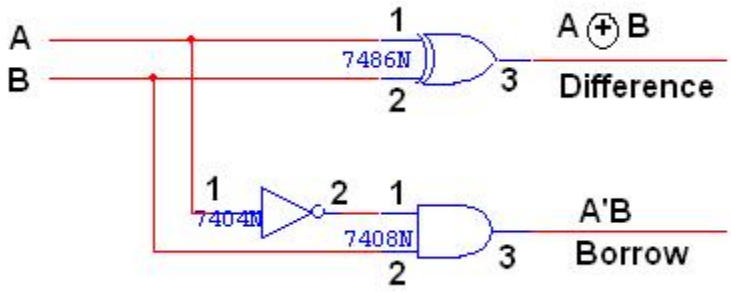
K-Map for CARRY:



CARRY = AB + BC + AC

LOGIC DIAGRAM:

HALF SUBTRACTOR



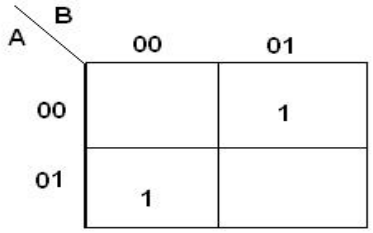
TRUTH TABLE:

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

HALF SUBTRACTOR:

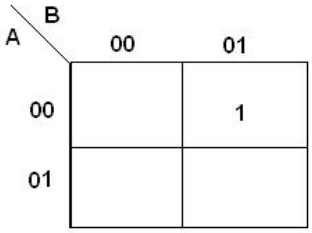
The half subtractor is constructed using X-OR and AND Gate. The half subtractor has two input and two outputs. The outputs are difference and borrow. The difference can be applied using X-OR Gate, borrow output can be implemented using an AND Gate and an inverter.

K-Map for DIFFERENCE:



DIFFERENCE = A'B + AB'

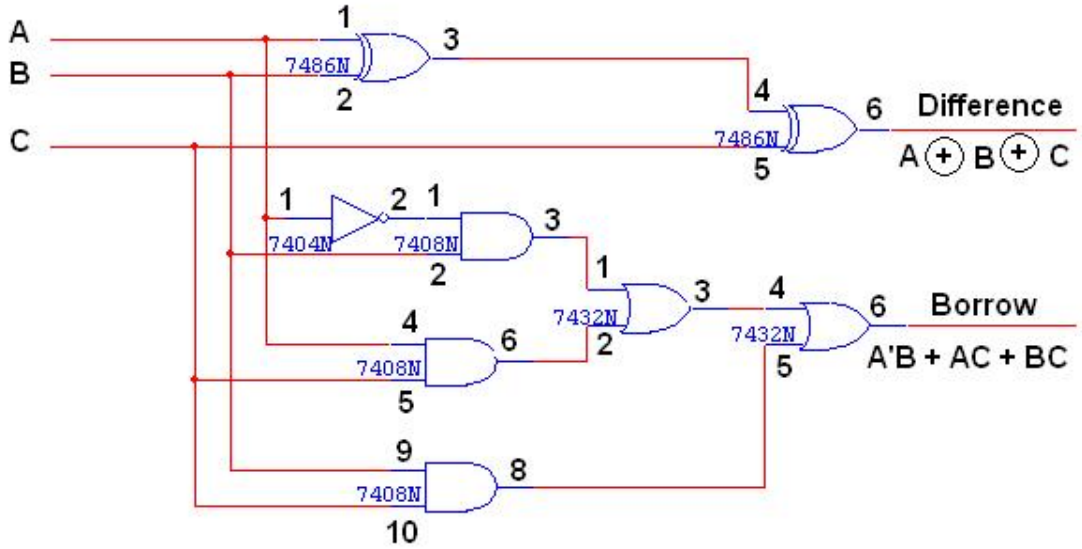
K-Map for BORROW:



BORROW = A'B

LOGIC DIAGRAM:

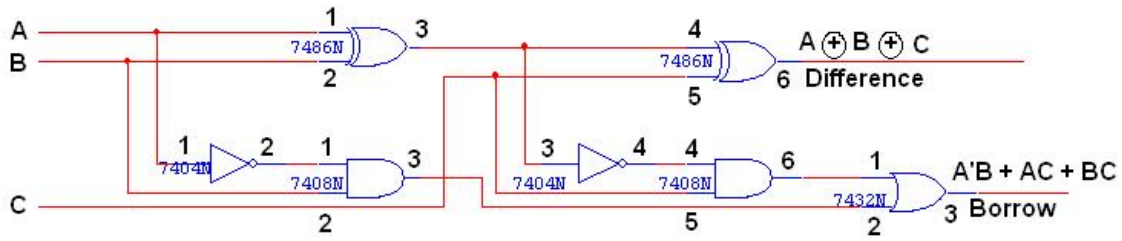
FULL SUBTRACTOR



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.

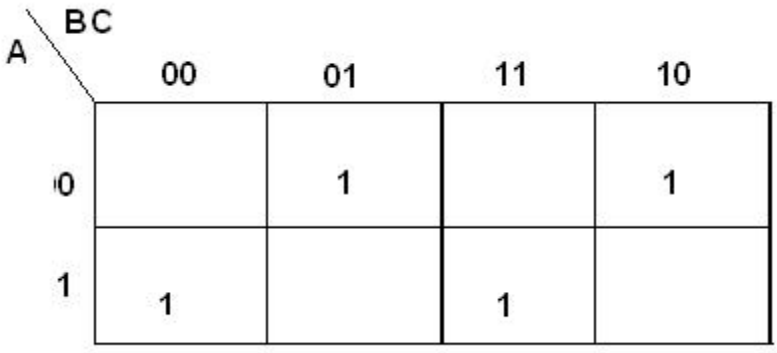
FULL SUBTRACTOR USING TWO HALF SUBTRACTOR:



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:



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

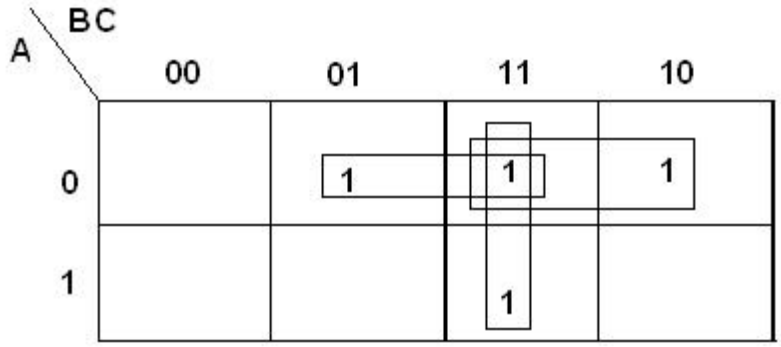
PROCEDURE:

(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.

K-Map for Borrow:



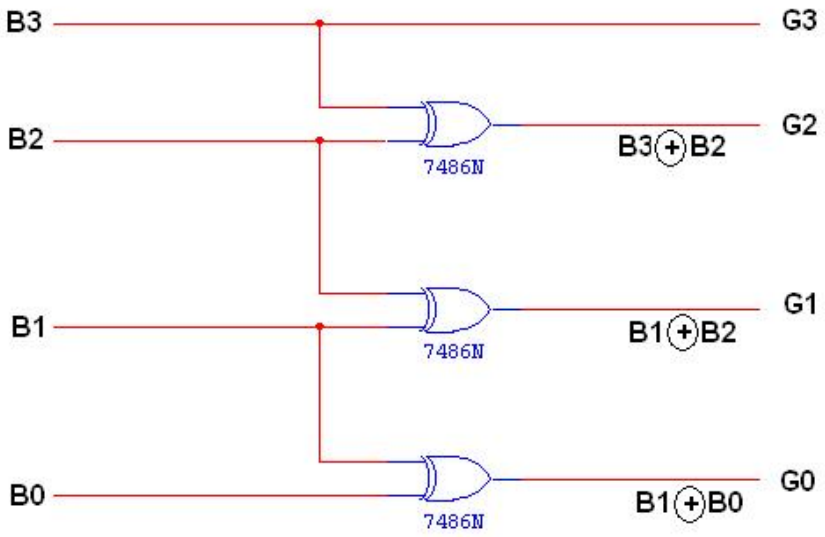
Borrow = A'B + BC + A'C

RESULT: -

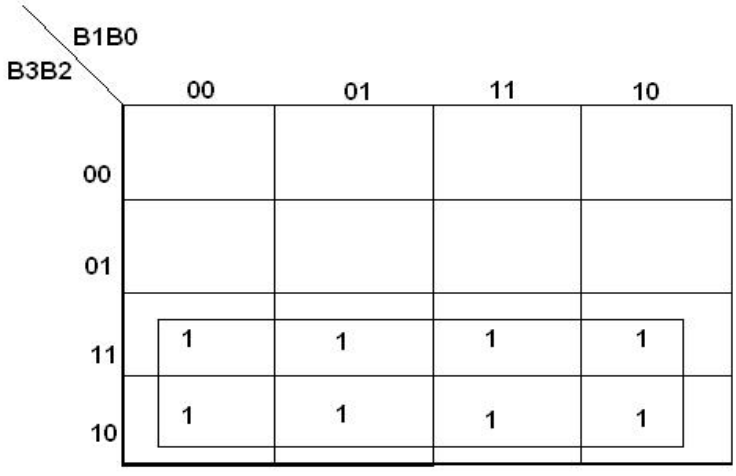
Thus the half adder, full adder, half subtractor and full subtractor circuits were designed and their logic was verified.

LOGIC DIAGRAM:.

BINARY TO GRAY CODE CONVERTOR



K-Map for G_3:



$G_3 = B_3$

EX NO:3 DATE:	DESIGN AND IMPLEMENTATION OF CODE CONVERTORS

AIM:

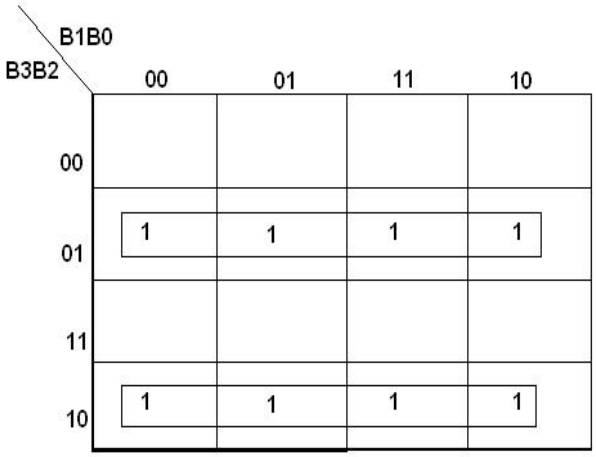
To design and implement 4-bit

(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:

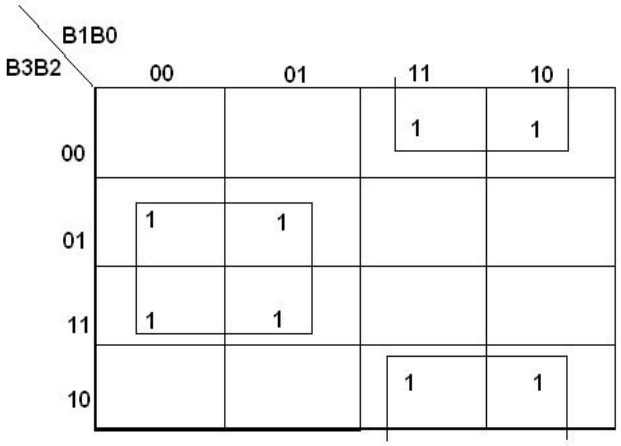
Sl.No.	COMPONENT	SPECIFICATION	QTY.
1.	X-OR GATE	IC 7486	1
2.	AND GATE	IC 7408	1
3.	OR GATE	IC 7432	1
4.	NOT GATE	IC 7404	1
5.	IC TRAINER KIT	-	1
6.	PATCH CORDS	-	35

K-Map for G_2:



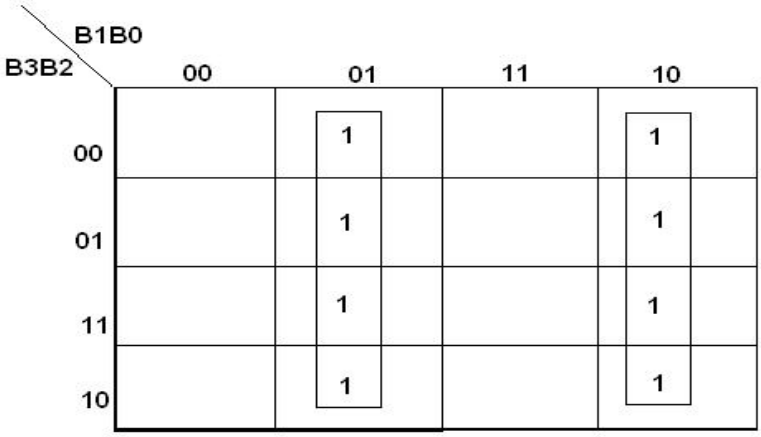
$G2 = B3 \oplus B2$

K-Map for G_1:



$G1 = B1 \oplus B2$

K-Map for G_0:



$G0 = B1 \oplus B0$

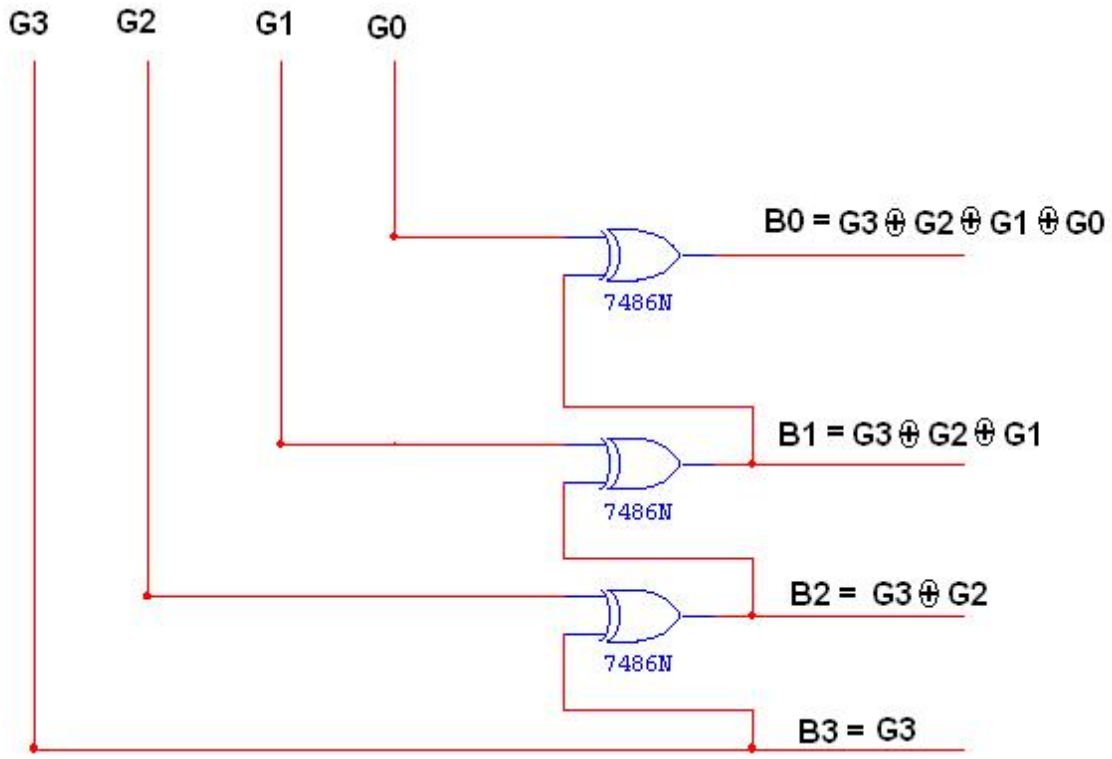
TRUTH TABLE:

| 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



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

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.

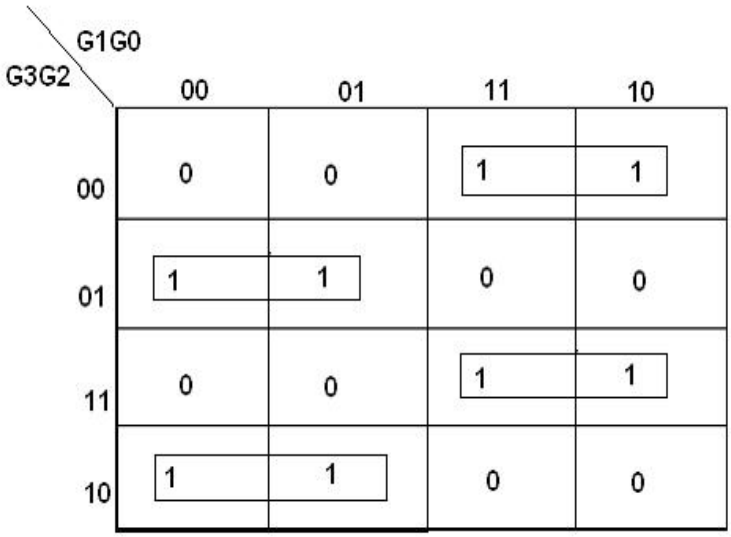
The input variable are designated as B3, B2, B1, B0 and the output variables are designated as C3, C2, C1, Co. from the truth table, combinational circuit is designed. The Boolean functions are obtained from K-Map for each output variable.

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

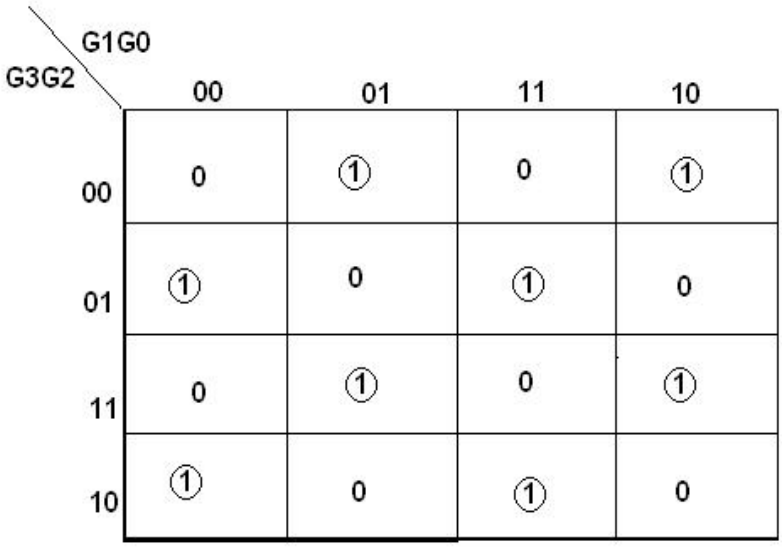
$B2 = G3 \oplus G2$

K-Map for B_1:



B1 = G3⊕G2⊕G1

K-Map for B_0:



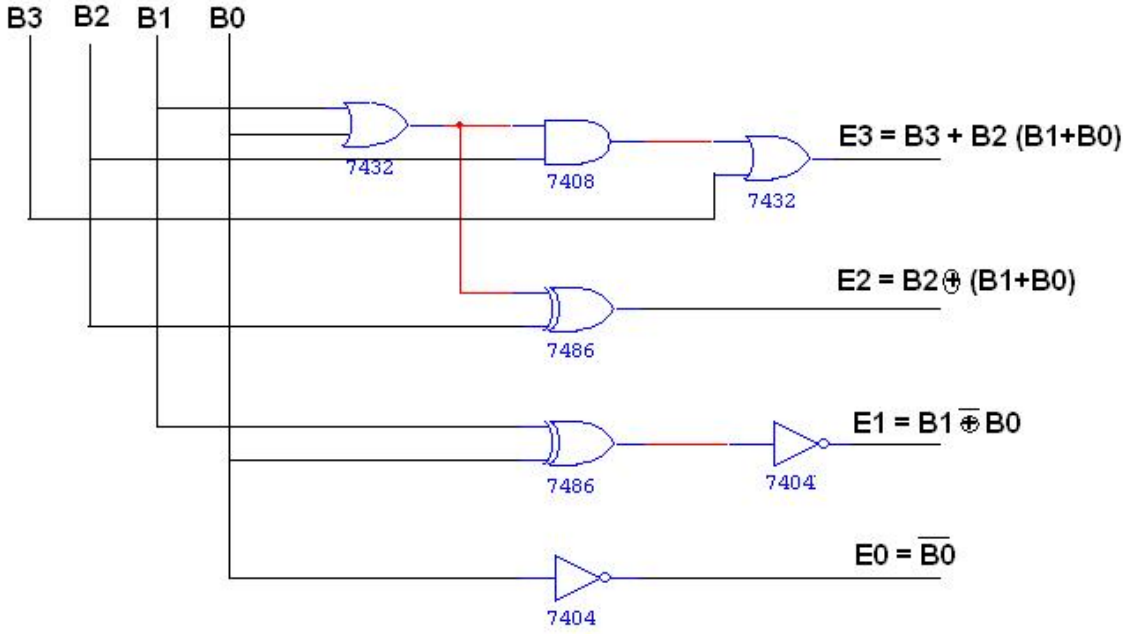
B0 = G3⊕G2⊕G1⊕G0

TRUTH TABLE:

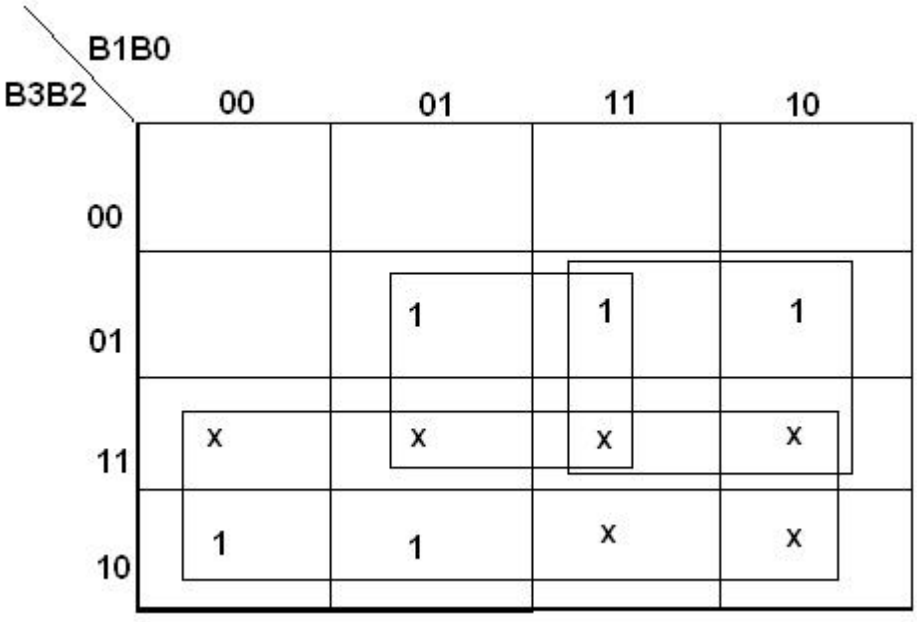
Gray Code				Binary Code			
G3	G2	G1	G0	B3	B2	B1	B0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

LOGIC DIAGRAM:

BCD TO EXCESS-3 CONVERTOR

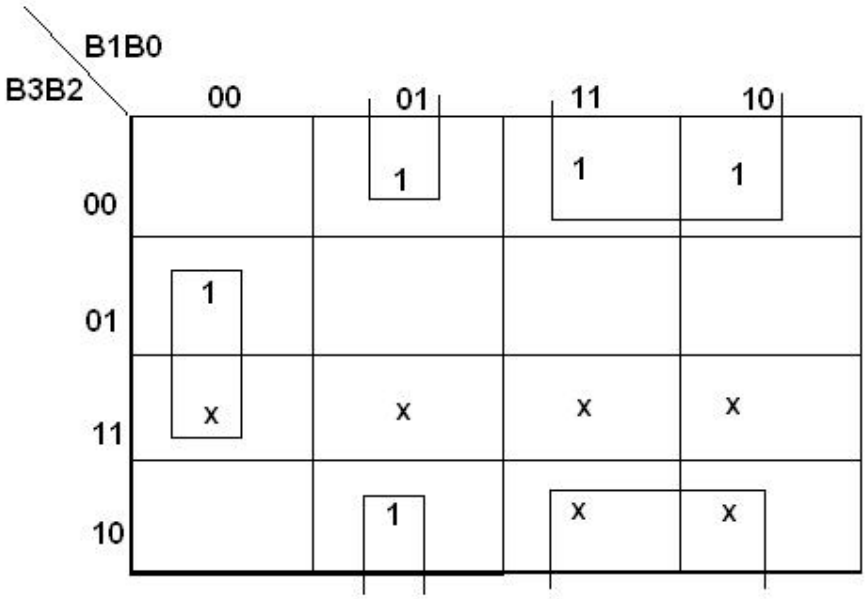


K-Map for E_3:



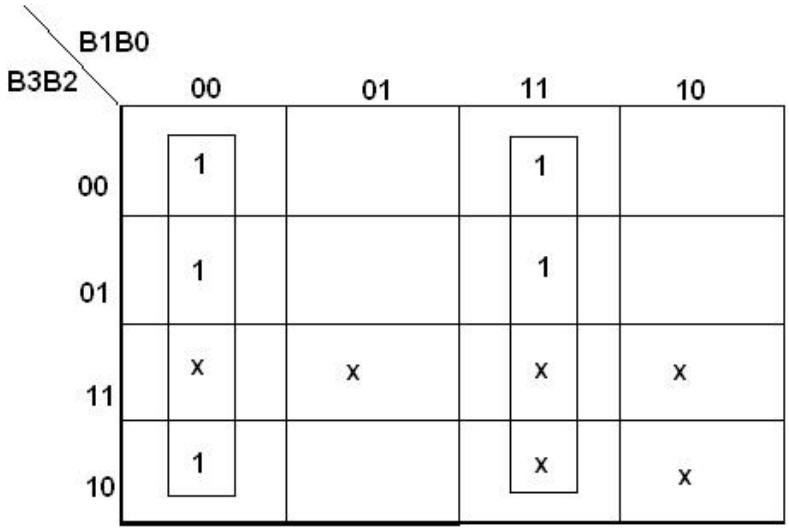
E3 = B3 + B2 (B0 + B1)

K-Map for E_2:



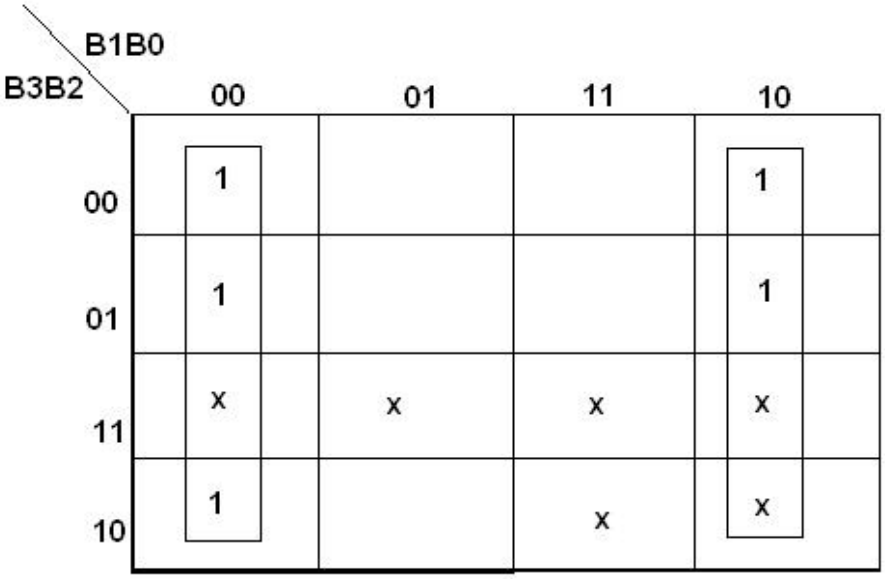
$E2 = B2 \oplus (B1 + B0)$

K-Map for E_1:



$E1 = B1 \odot B0$

K-Map for E_0:



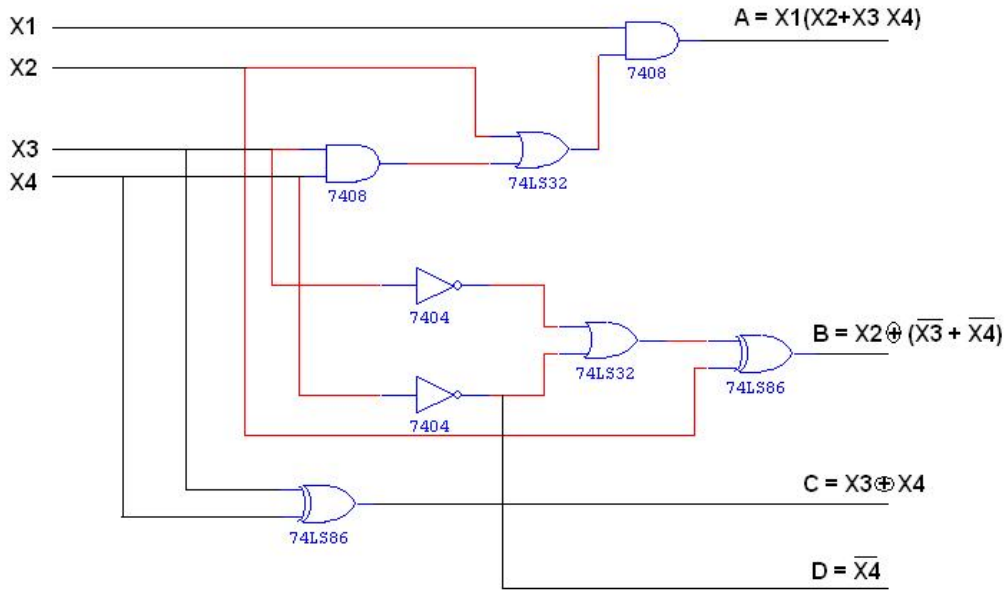
$E0 = \overline{B0}$

TRUTH TABLE:

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

LOGIC DIAGRAM:

EXCESS-3 TO BCD CONVERTOR



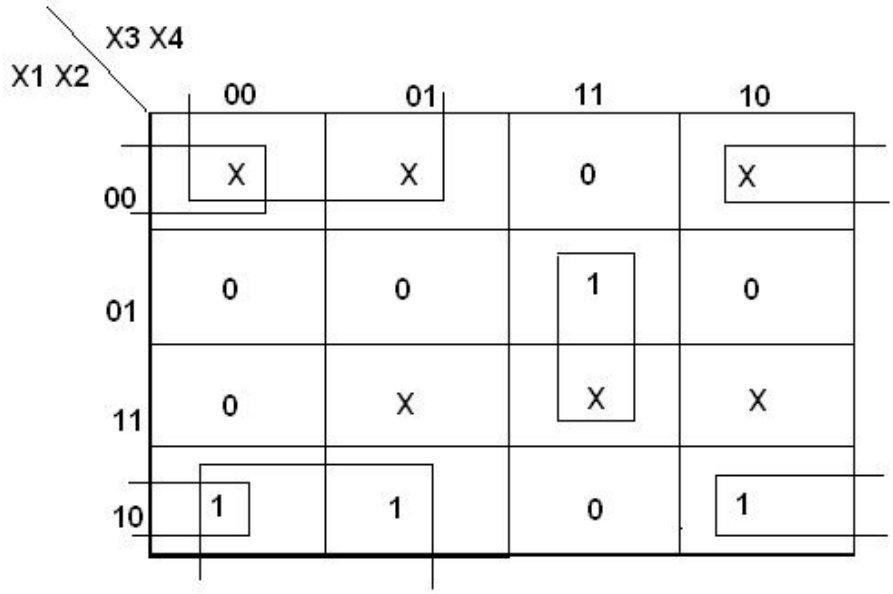
K-Map for A

:

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

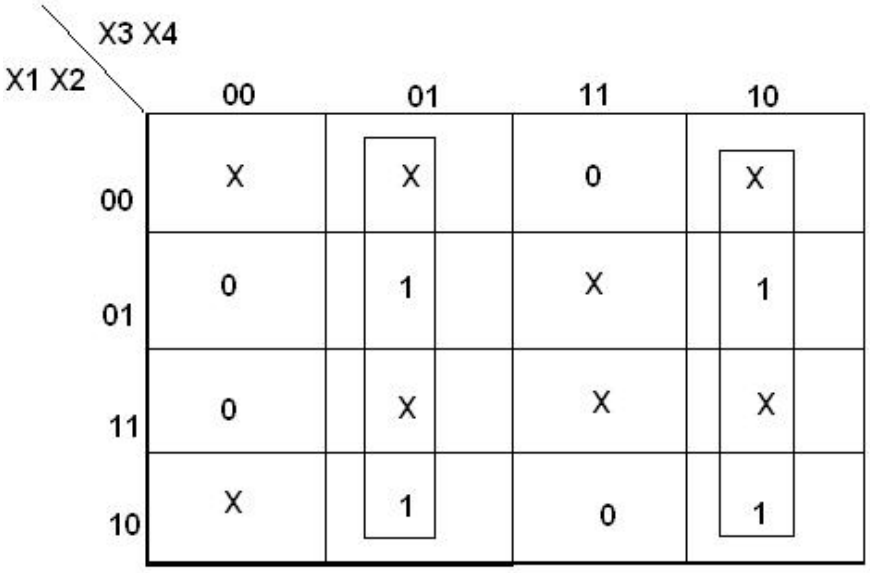
A = X1 X2 + X3 X4 X1

K-Map for B:



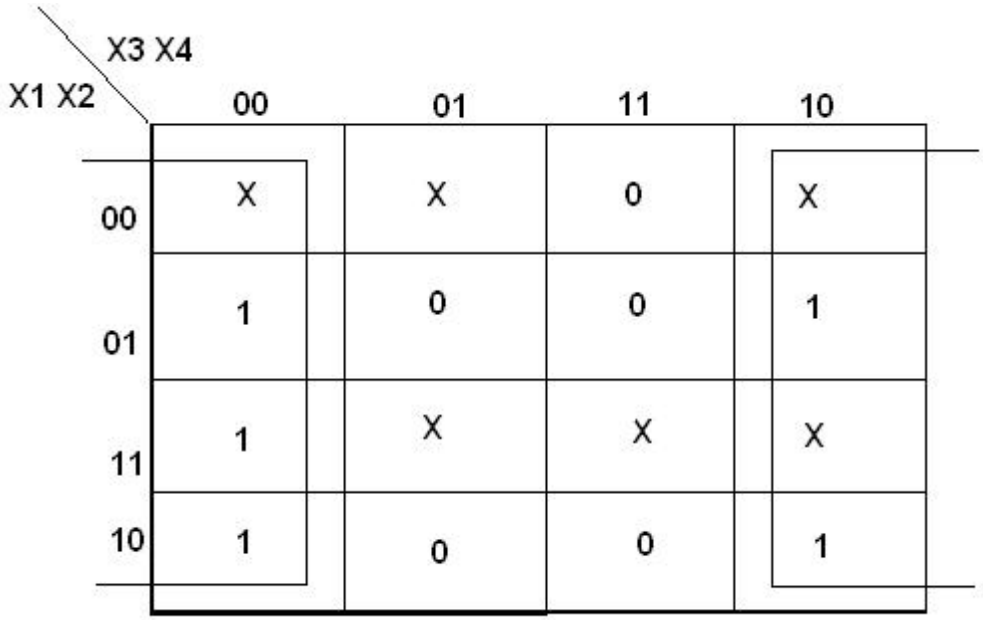
$$B = X2 \oplus (\overline{X3} + \overline{X4})$$

K-Map for C:



$$C = X3 \oplus X4$$

K-Map for D:



$D = \overline{X4}$

THEORY:

BINARY TO EXCESS-3 CODE CONVERTOR:

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.

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

PROCEDURE:

(i) Connections were given as per circuit diagram.

(ii) Logical inputs were given as per truth table

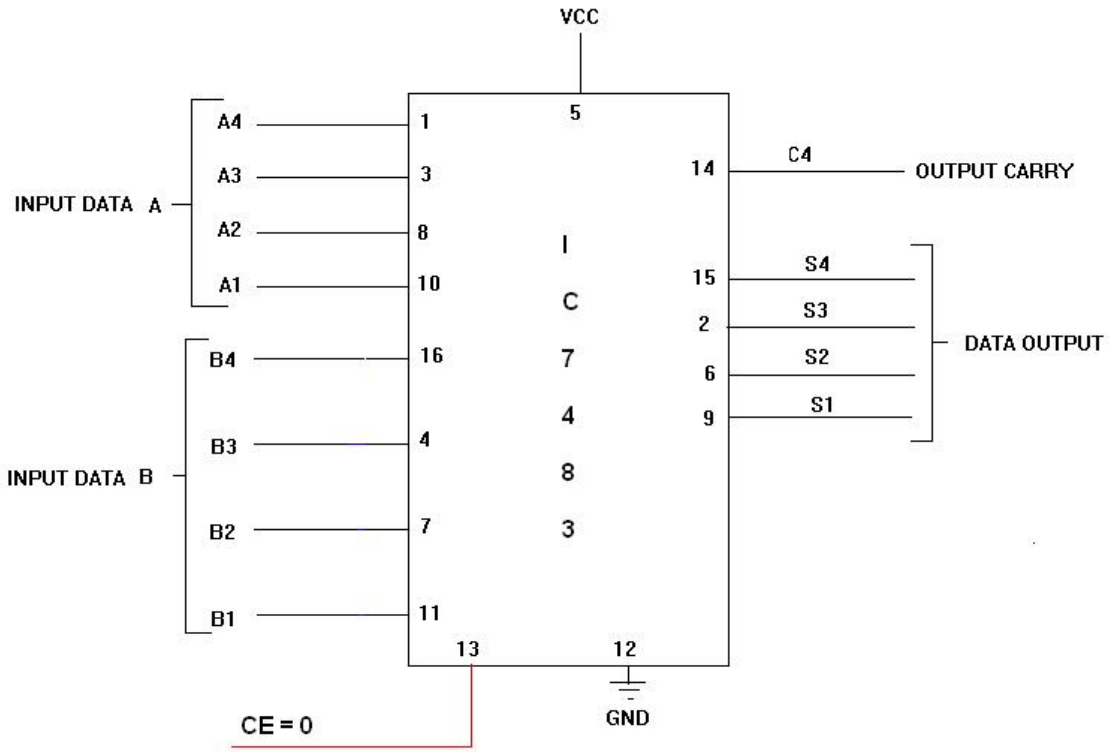
(iii) Observe the logical output and verify with the truth tables.

RESULT: -

Thus the code converter circuits were designed and their logic was verified.

LOGIC DIAGRAM:

4-BIT BINARY ADDER



EX NO:4	DESIGN OF 4-BIT ADDER AND SUBTRACTOR
DATE:	

AIM:

To design and implement 4-bit adder and subtractor using 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

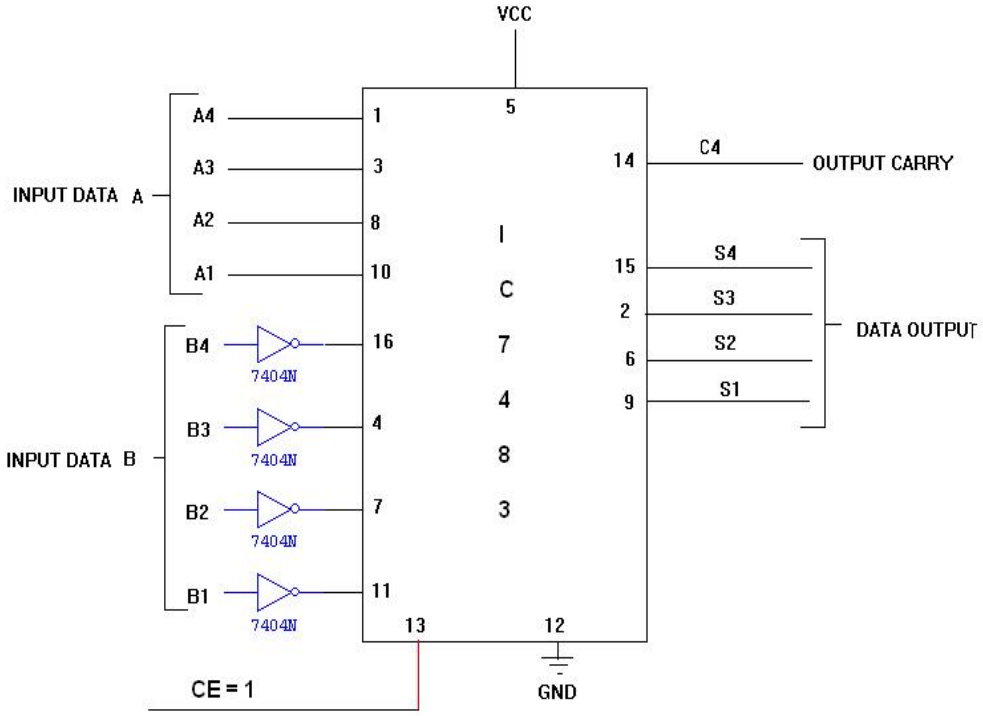
THEORY:

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.

LOGIC DIAGRAM:

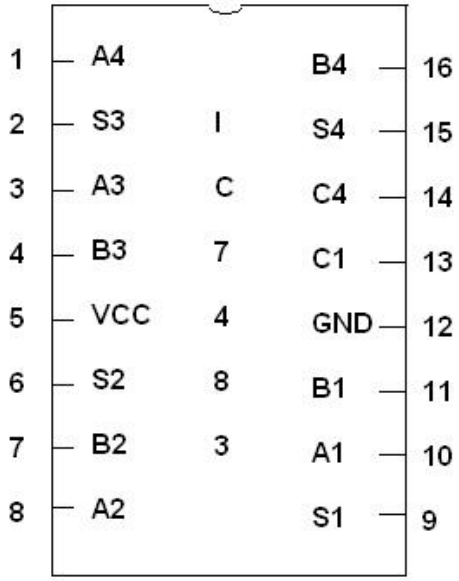
4-BIT BINARY SUBTRACTOR



4 BIT BINARY SUBTRACTOR:

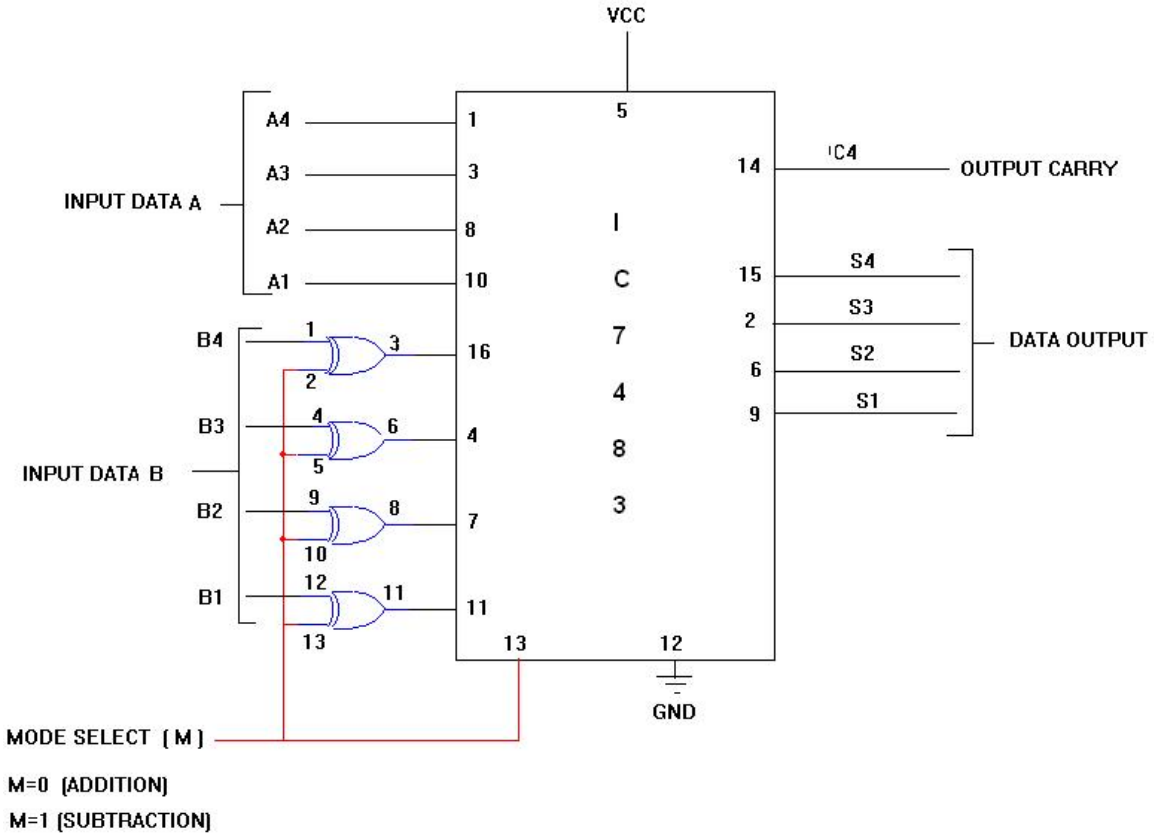
The circuit for subtracting A-B consists of an adder with inverters, placed between each data input 'B' and the corresponding input of full adder. The input carry C_0 must be equal to 1 when performing subtraction.

PIN DIAGRAM FOR IC 7483:



LOGIC DIAGRAM:

4-BIT BINARY ADDER/SUBTRACTOR



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:

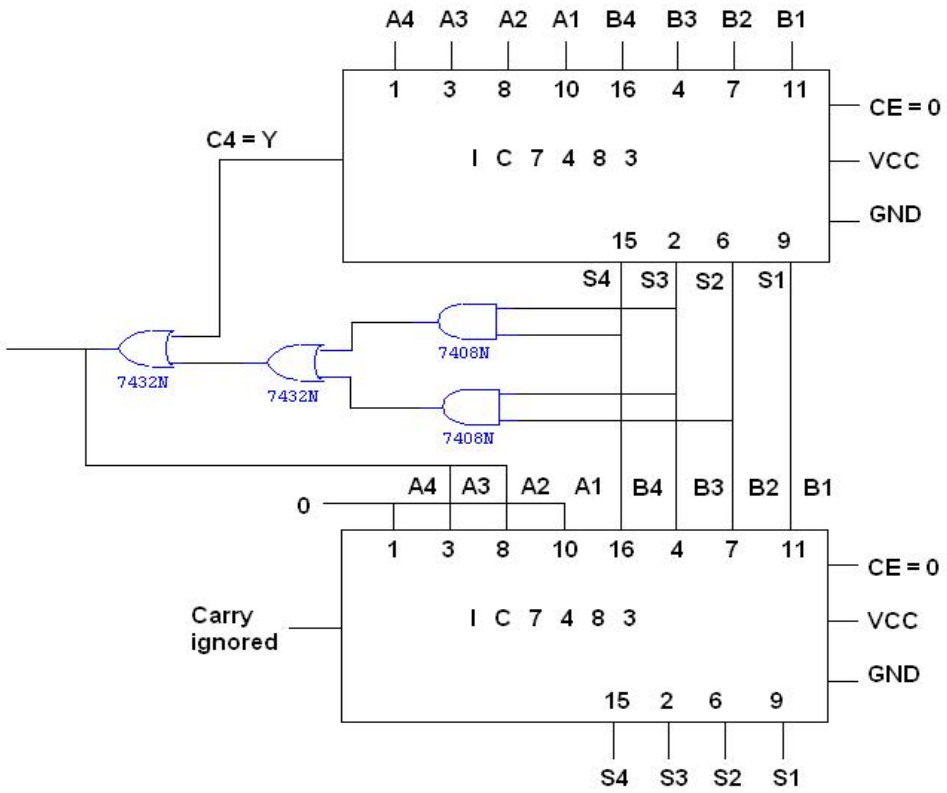
Consider the arithmetic addition of two decimal digits in BCD, together with an input carry from a previous stage. Since each input digit does not exceed 9, the output sum cannot be greater than 19, the 1 in the sum being an input carry. The output of two decimal digits must be represented in BCD and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

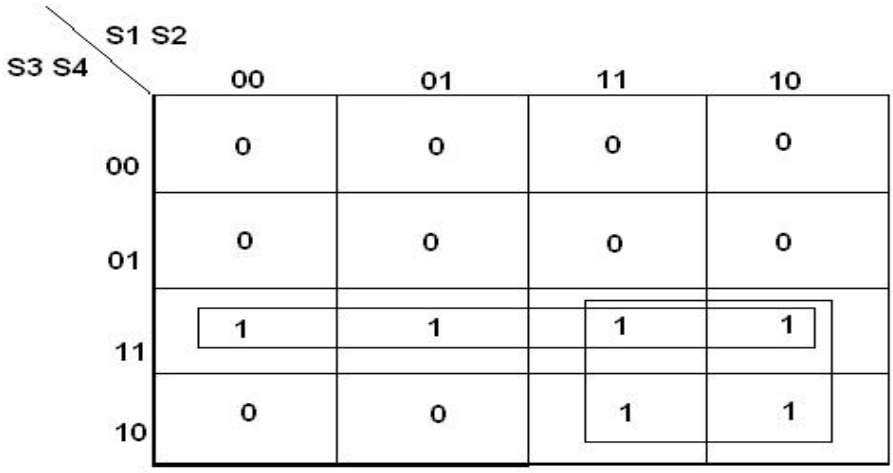
TRUTH TABLE:

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

LOGIC DIAGRAM:
BCD ADDER



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

PROCEDURE:

(i) Connections were given as per circuit diagram.

(ii) Logical inputs were given as per truth table

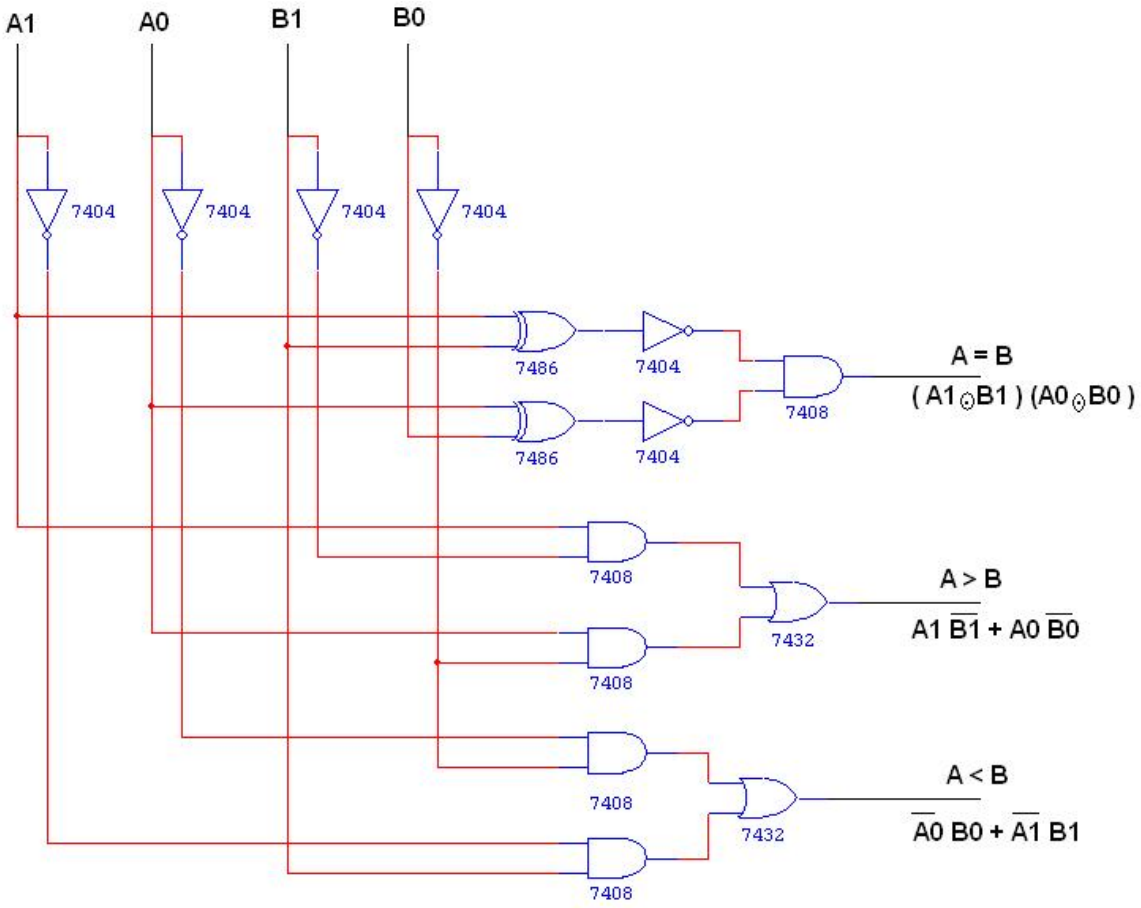
(iii) Observe the logical output and verify with the truth tables.

RESULT: -

Thus the 4 bit adder and subtractor circuits were designed and their logic was verified.

LOGIC DIAGRAM:

2 BIT MAGNITUDE COMPARATOR



EX NO: 5	DESIGN AND IMPLEMENTATION OF MAGNITUDE COMPARATOR
DATE:	

AIM:

To design and implement

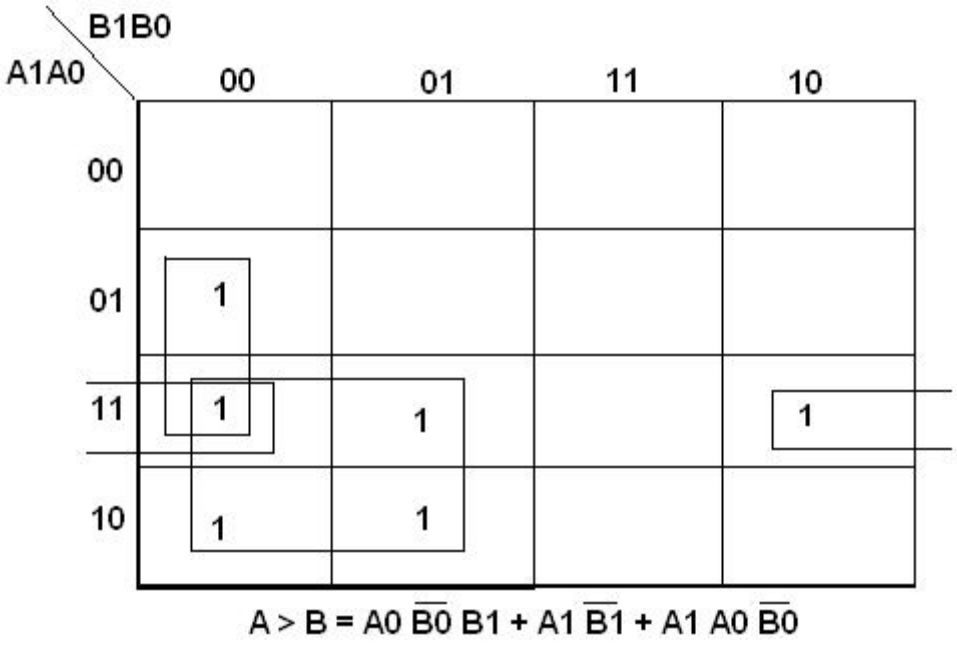
(i) 2 – bit magnitude comparator using basic gates.

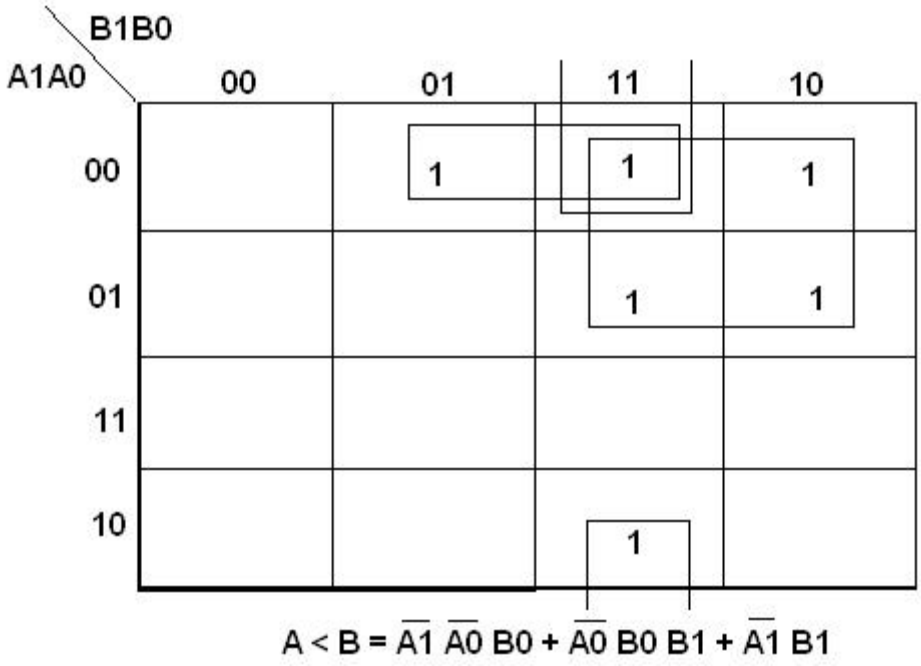
(ii) 8 – bit magnitude comparator using IC 7485.

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

K MAP


B1B0
A1A0
00
01
11
10
00
01
11
10
1
1
1
1
1
1
1
A > B = A0 $\overline{B0}$ B1 + A1 $\overline{B1}$ + A1 A0 $\overline{B0}$


B1B0
A1A0
00
01
11
10
00
01
11
10
1
1
1
1
1
1
1
A < B = $\overline{A1}$ $\overline{A0}$ B0 + $\overline{A0}$ B0 B1 + $\overline{A1}$ B1

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

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

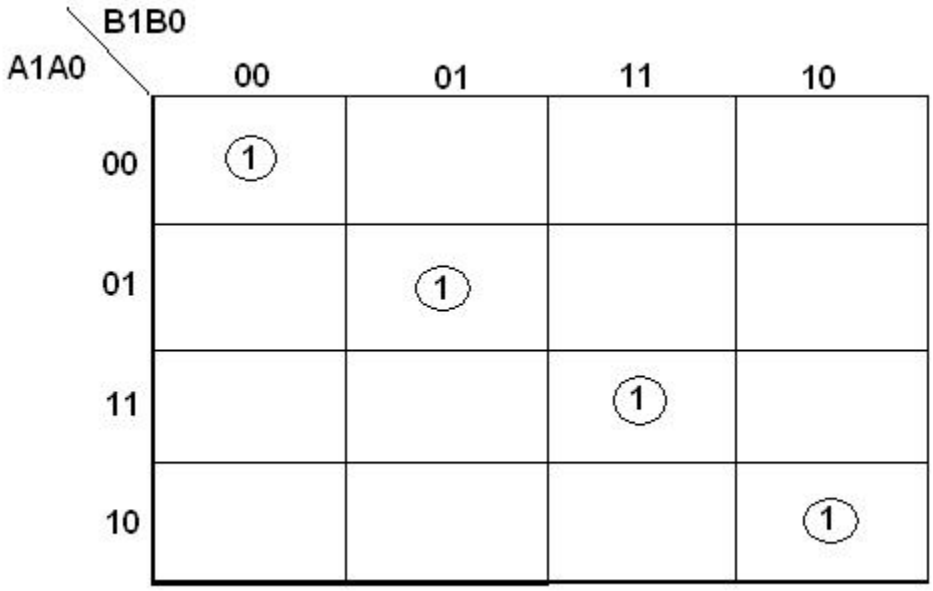
$$A<B = A_3^{\ 1}B_3 + X_3A_2^{\ 1}B_2 + X_3X2A_1^{\ 1}B_1 + X_3X_2X_1A_0^{\ 1}B_0$$

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 x_2 x_1 x_0



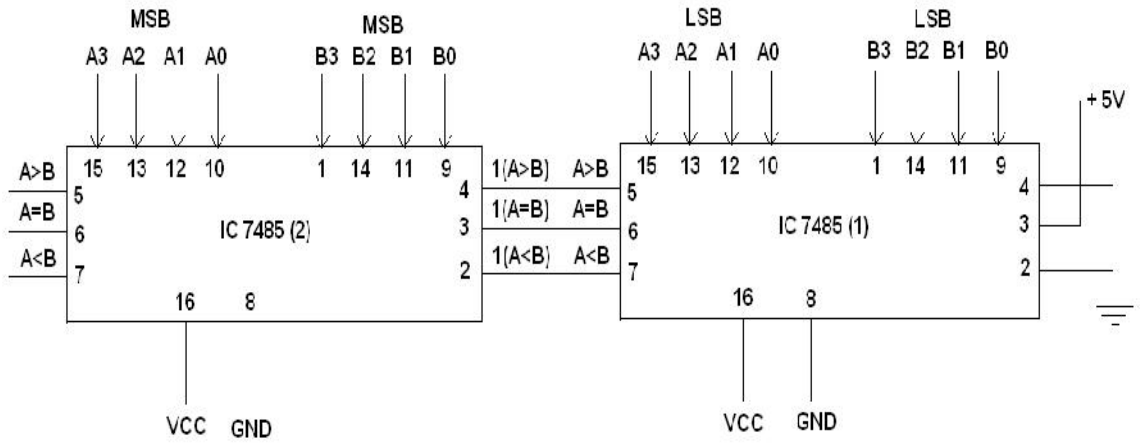
$A = B = (A0 \odot B0)(A1 \odot B1)$

TRUTH TABLE:

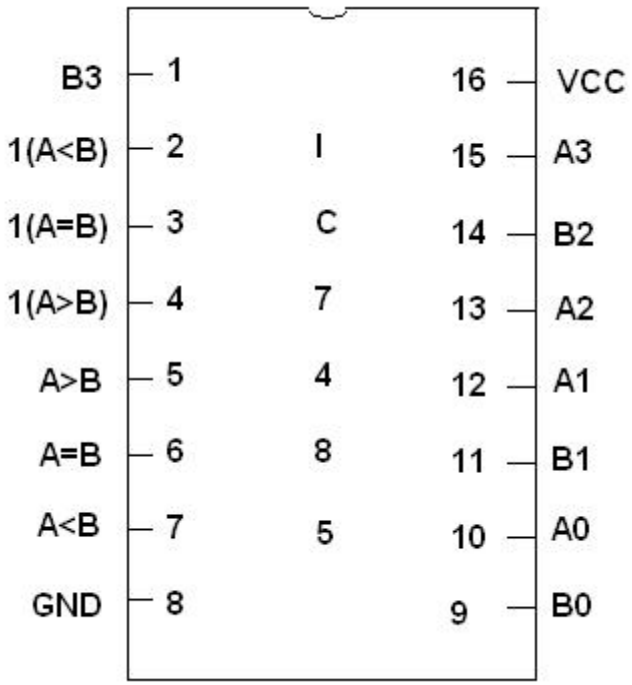
A1	A0	B1	B0	A > B	A = B	A < B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

LOGIC DIAGRAM:

8 BIT MAGNITUDE COMPARATOR



PIN DIAGRAM FOR IC 7485:



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.

TRUTH TABLE:

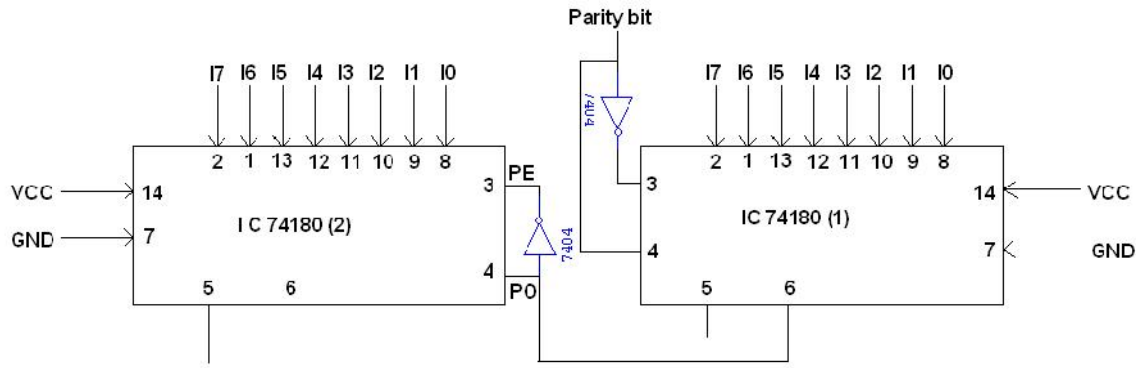
A	B	A>B	A=B	A<B
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0	1	0
0 0 0 1 0 0 0 1	0 0 0 0 0 0 0 0	1	0	0
0 0 0 0 0 0 0 0	0 0 0 1 0 0 0 1	0	0	1

RESULT: -

Thus the magnitude comparator circuits were designed and their logic was verified.

LOGIC DIAGRAM:

16 BIT ODD/EVEN PARITY CHECKER



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

EX NO: 6 DATE:	16 BIT ODD/EVEN PARITY CHECKER AND GENERATOR

AIM:

To design and implement 16 bit odd/even parity checker generator using IC 74180.

APPARATUS REQUIRED:

Sl.No.	COMPONENT	SPECIFICATION	QTY.
1.	NOT GATE	IC 7404	1
1.		IC 74180	2
2.	IC TRAINER KIT	-	1
3.	PATCH CORDS	-	30

THEORY:

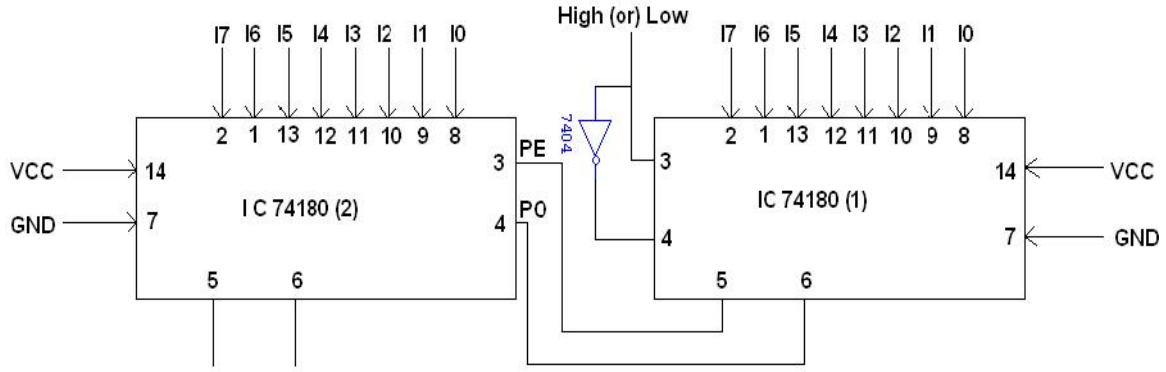
A parity bit is used for detecting errors during transmission of binary information. A parity bit is an extra bit included with a binary message to make the number is either even or odd. The message including the parity bit is transmitted and then checked at the receiver ends for errors. An error is detected if the checked parity bit doesn't correspond to 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'.

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

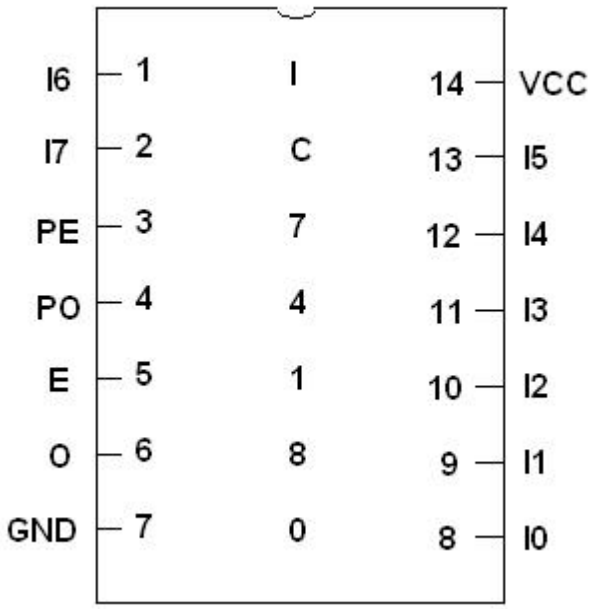
LOGIC DIAGRAM:

16 BIT ODD/EVEN PARITY GENERATOR



In even parity, the added parity bit will make the total number is even amount. In odd parity, the added parity bit will make the total number is odd amount. The parity checker circuit checks for possible errors in the transmission. If the information is passed in even parity, then the bits required must have an even number of 1's. An error occur during transmission, if the received bits have an odd number of 1's indicating that one bit has changed in value during transmission.

PIN DIAGRAM FOR IC 74180:



.

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

PROCEDURE:

(i) Connections are given as per circuit diagram.

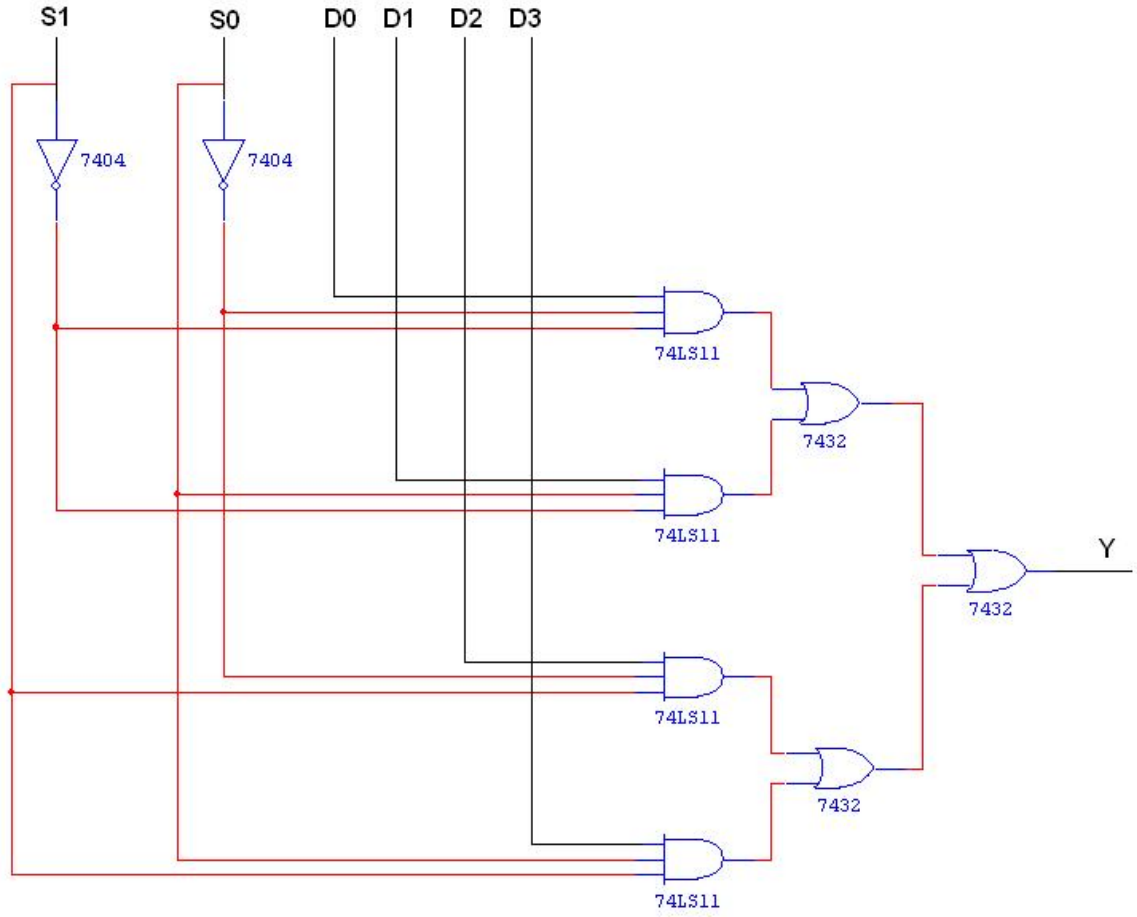
(ii) Logical inputs are given as per circuit diagram.

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

RESULT: -

Thus the i6bit odd/even parity checker/generator circuits were designed and their logic was verified.

CIRCUIT DIAGRAM FOR 4X1 MULTIPLEXER:



TRUTH TABLE:

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

EX NO: 7 DATE:	DESIGN AND IMPLEMENTATION OF MULTIPLEXER AND DEMULTIPLEXER

AIM:

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

APPARATUS REQUIRED:

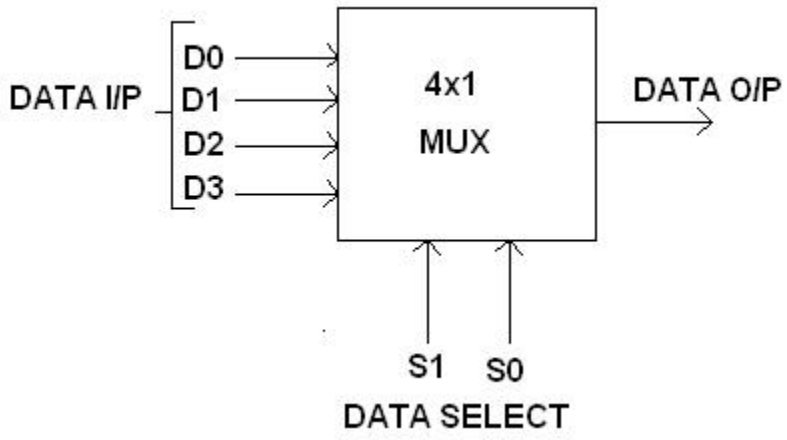
Sl.No.	COMPONENT	SPECIFICATION	QTY.
1.	3 I/P AND GATE	IC 7411	2
2.	OR GATE	IC 7432	1
3.	NOT GATE	IC 7404	1
2.	IC TRAINER KIT	-	1
3.	PATCH CORDS	-	32

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.

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

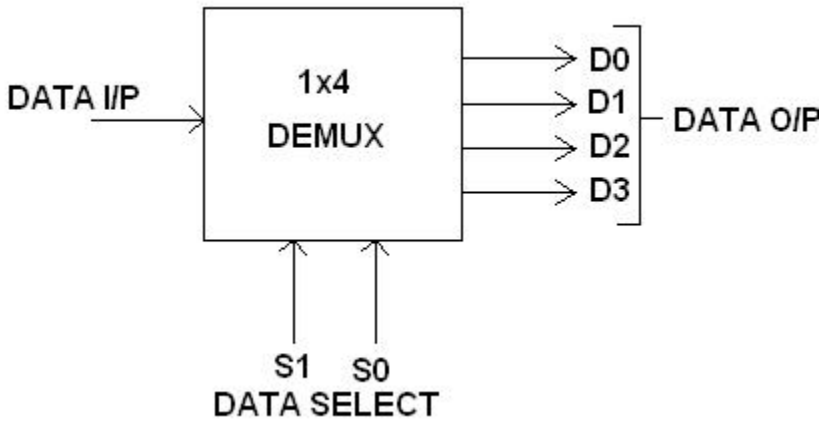
Y = D0 S1' S0' + D1 S1' S0 + D2 S1 S0' + D3 S1 S0

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.

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

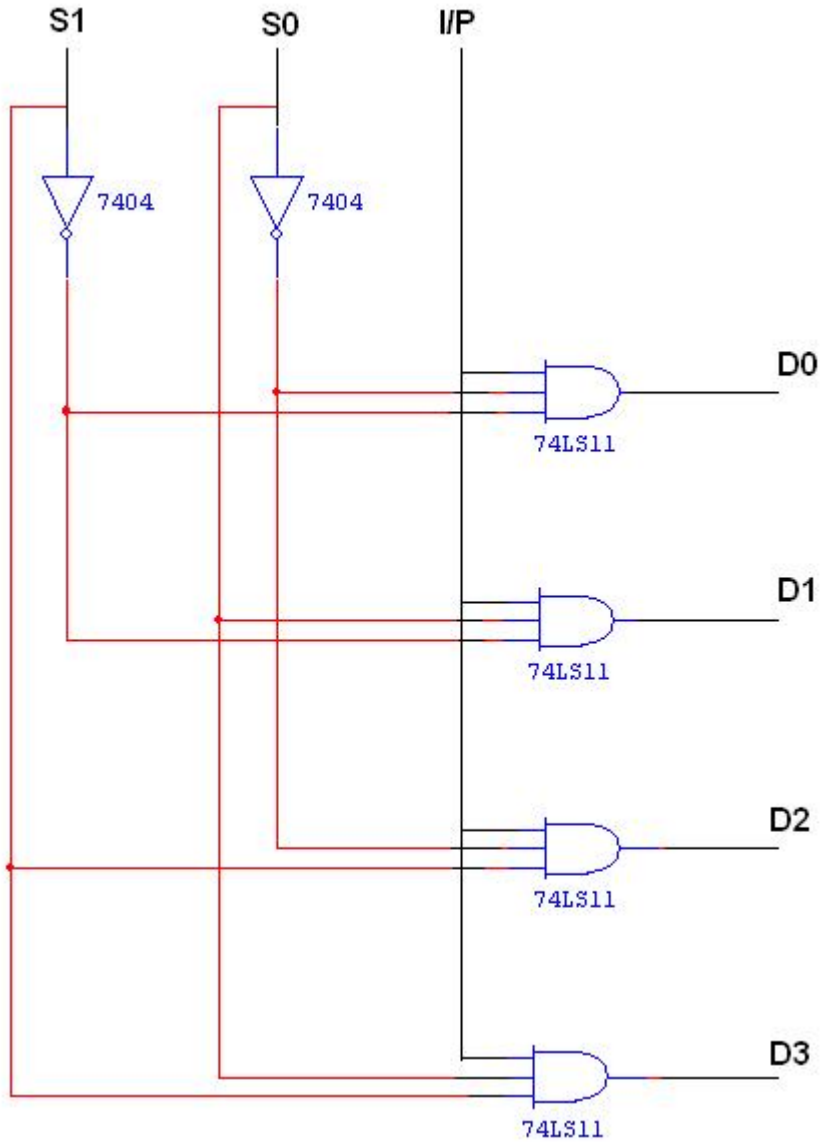
D0 = X S1' S0'

D1 = X S1' S0

D2 = X S1 S0'

D3 = X S1 S0

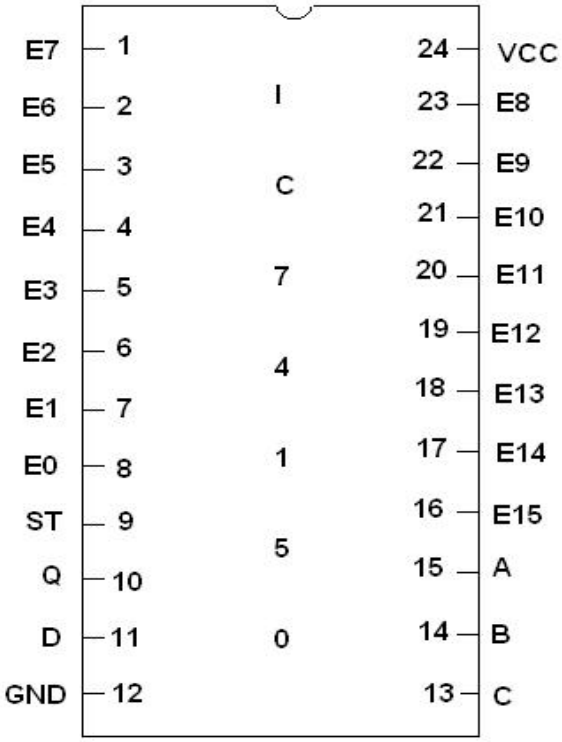
LOGIC DIAGRAM FOR DEMULTIPLEXER:



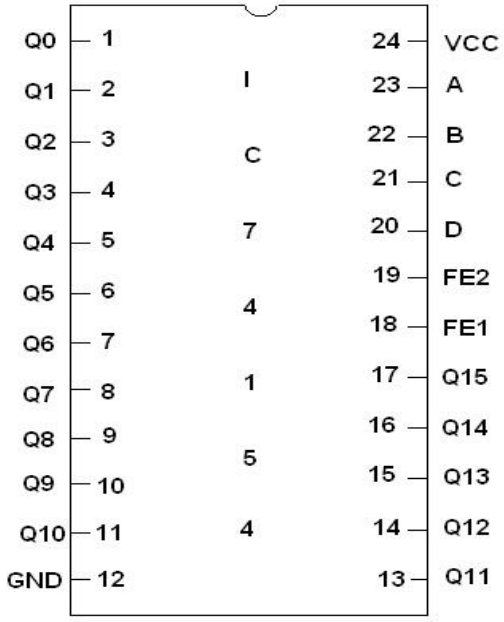
TRUTH TABLE:

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:



PROCEDURE:

(i) Connections are given as per circuit diagram.

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

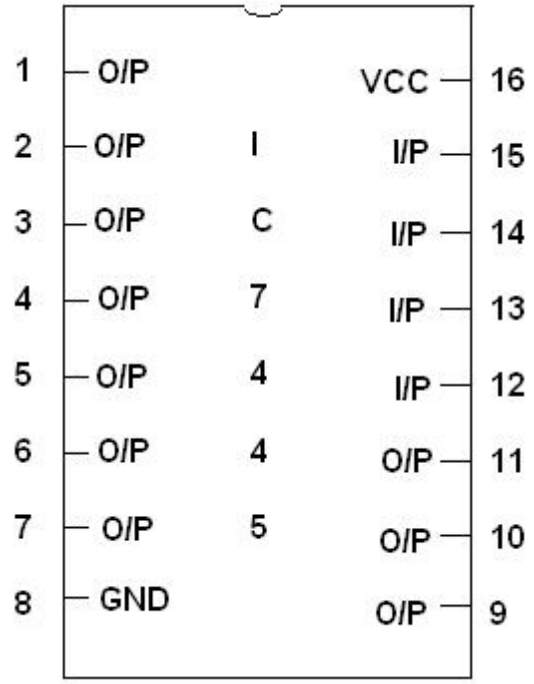
(iii) Observe the output and verify the truth table.

RESULT: -

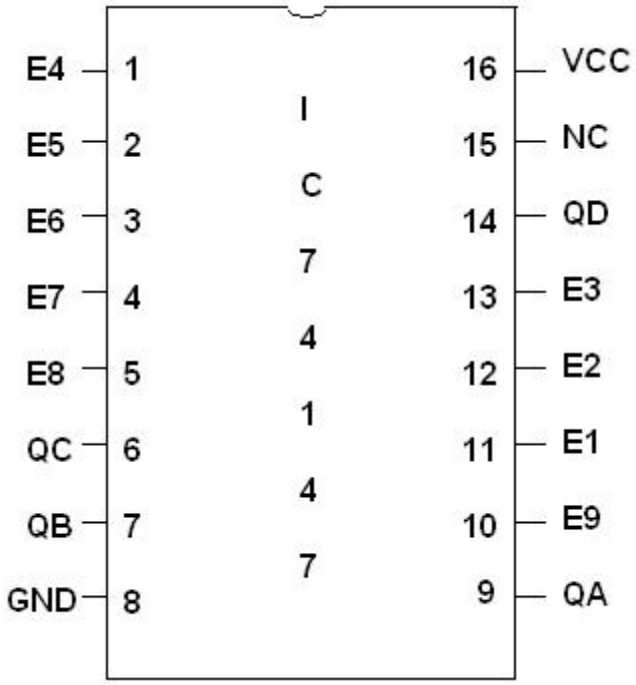
Thus the Multiplexer/Demultiplexer circuits were designed and their logic was verified.

PIN DIAGRAM FOR IC 7445:

BCD TO DECIMAL DECODER:



PIN DIAGRAM FOR IC 74147:



EX NO: 8 DATE:	DESIGN AND IMPLEMENTATION OF ENCODER AND DECODER

AIM:

To design and implement encoder and decoder using logic gates and study of IC 7445 and IC 74147.

APPARATUS REQUIRED:

Sl.No.	COMPONENT	SPECIFICATION	QTY.
1.	3 I/P NAND GATE	IC 7410	2
2.	OR GATE	IC 7432	3
3.	NOT GATE	IC 7404	1
2.	IC TRAINER KIT	-	1
3.	PATCH CORDS	-	27

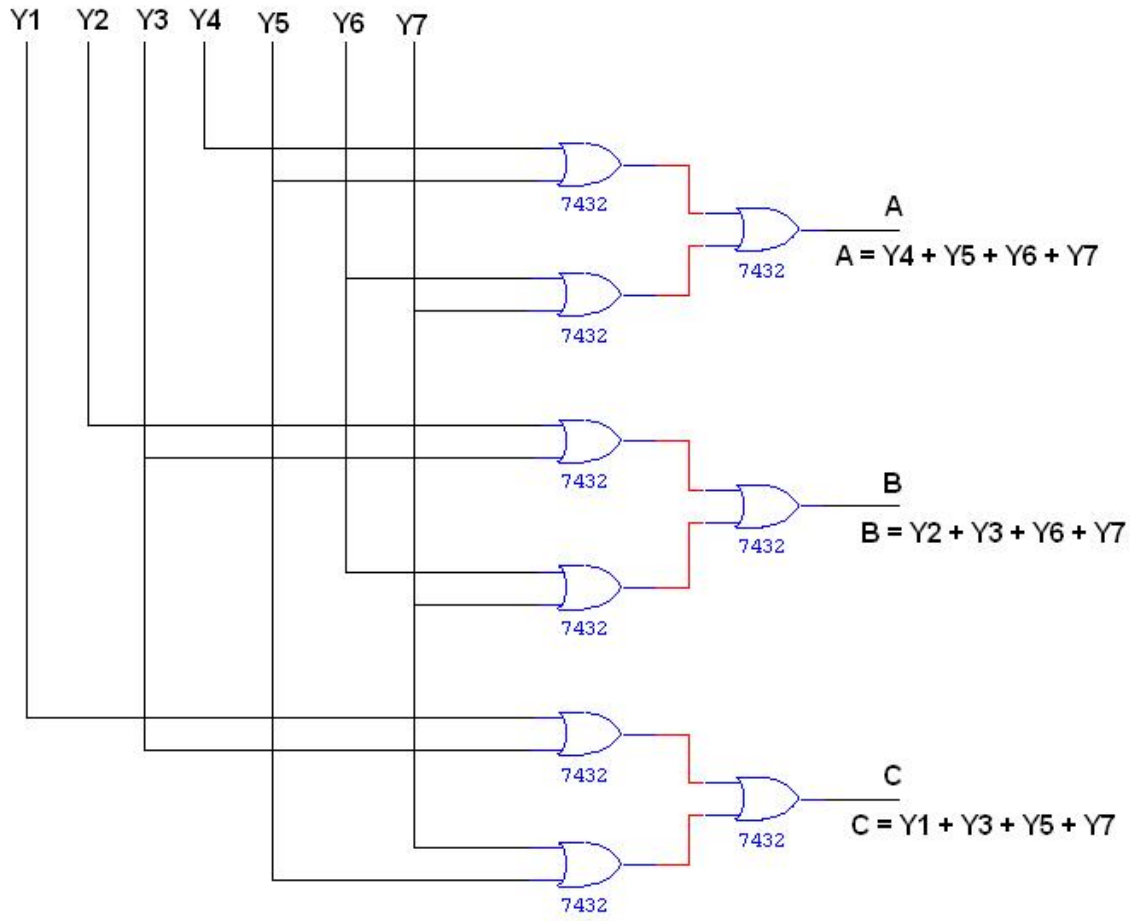
THEORY:

ENCODER:

An encoder is a digital circuit that performs inverse operation of a decoder. An encoder has 2^n input lines and n output lines. In encoder the output lines generates the binary code corresponding to the input value. In octal to binary encoder it has eight inputs, one for each octal digit and three output that generate the corresponding binary code. In encoder it is assumed that only one input has a value of one at any given time otherwise the circuit is meaningless. It has an ambiguila

that when all inputs are zero the outputs are zero. The zero outputs can also be generated when D0 = 1.

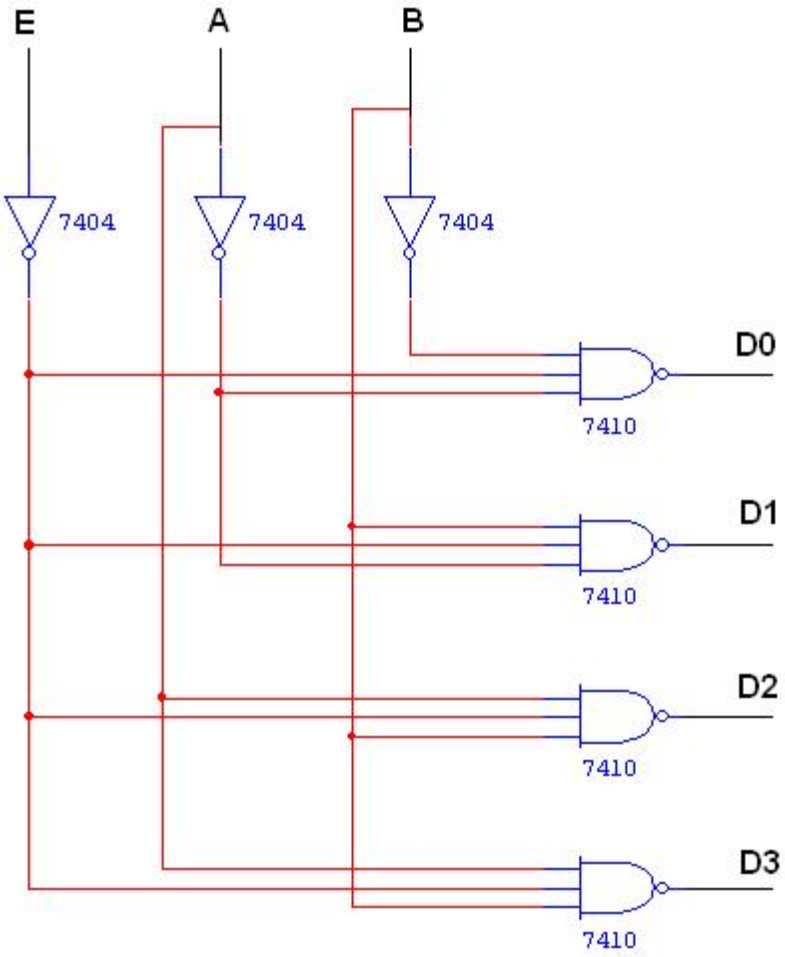
LOGIC DIAGRAM FOR ENCODER:



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:



THEORY:

DECODER:

A decoder is a multiple input multiple output logic circuit which converts coded input into coded output where input and output codes are different. The input code generally has fewer bits than the output code. Each input code word produces a different output code word i.e there is one to one mapping can be expressed in truth table. In the block diagram of decoder circuit the encoded information is present as n input producing 2^n possible outputs. 2^n output values are from 0 through out $2^n - 1$.

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**

PROCEDURE:

(i) Connections are given as per circuit diagram.

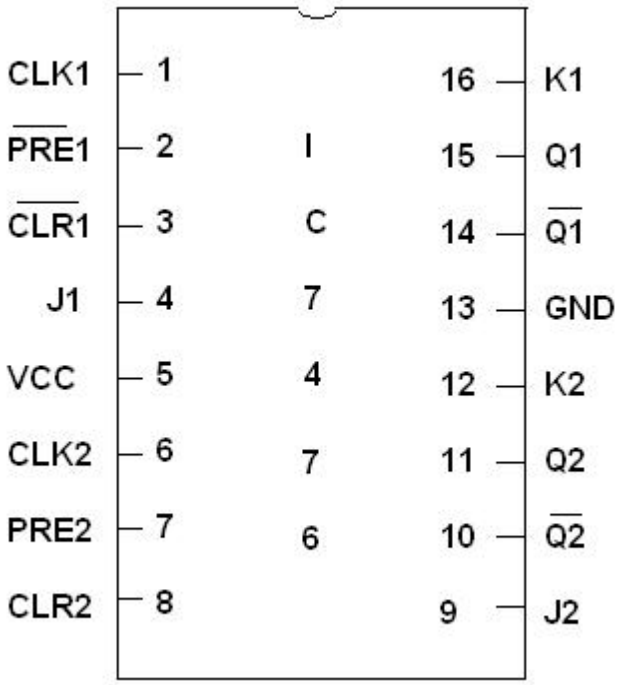
(ii) Logical inputs are given as per circuit diagram.

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

RESULT: -

Thus the encoder/decoder circuits were designed and their logic was verified.

PIN DIAGRAM FOR IC 7476:



EX NO: 9 DATE:	CONSTRUCTION AND VERIFICATION OF 4 BIT RIPPLE COUNTER AND MOD 10/MOD 12 RIPPLE COUNTER

AIM:

To design and verify 4 bit ripple counter mod 10/ mod 12 ripple counter.

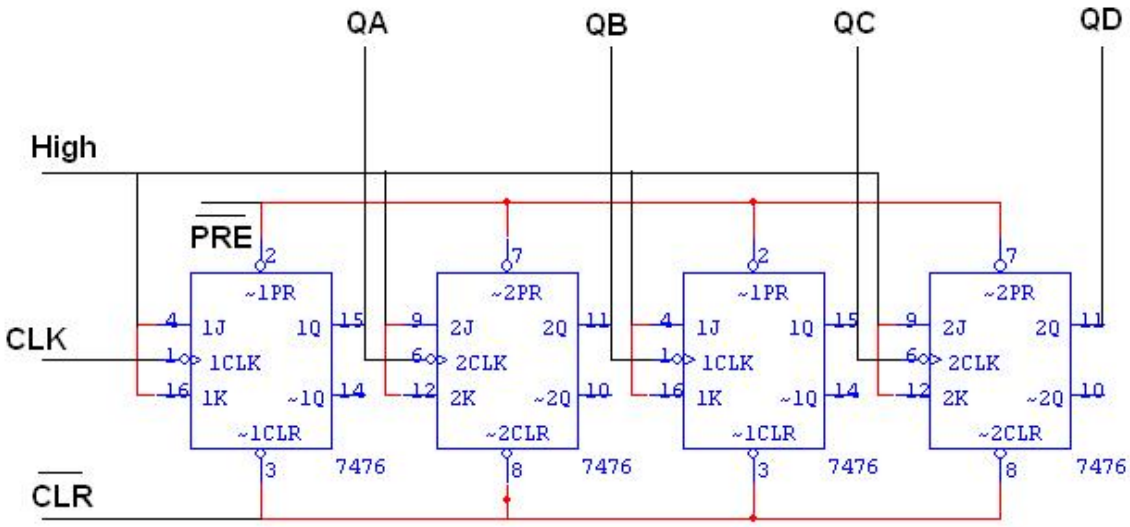
APPARATUS REQUIRED:

Sl.No.	COMPONENT	SPECIFICATION	QTY.
1.	JK FLIP FLOP	IC 7476	2
2.	NAND GATE	IC 7400	1
3.	IC TRAINER KIT	-	1
4.	PATCH CORDS	-	30

THEORY:

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.

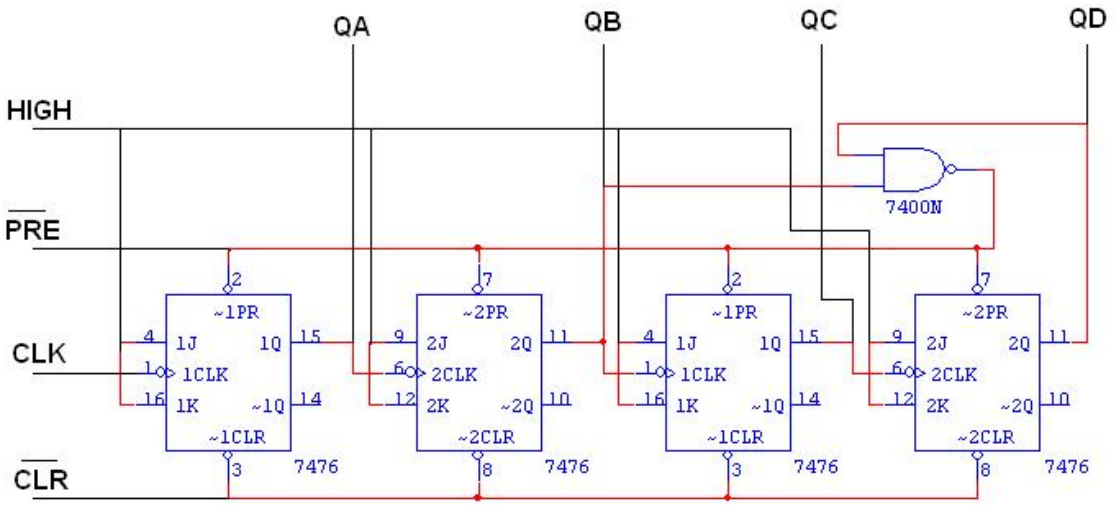
LOGIC DIAGRAM FOR 4 BIT RIPPLE COUNTER:



TRUTH TABLE:

CLK	QD	QC	QB	QA
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

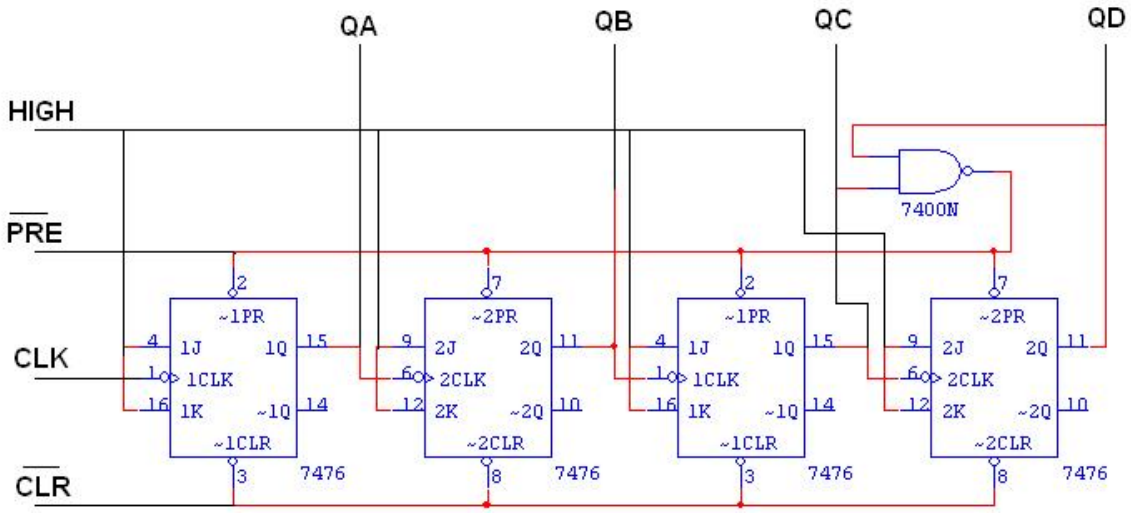
LOGIC DIAGRAM FOR MOD - 10 RIPPLE COUNTER:



TRUTH TABLE:

CLK	QD	QC	QB	QA
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

LOGIC DIAGRAM FOR MOD - 12 RIPPLE COUNTER:



TRUTH TABLE:

CLK	QD	QC	QB	QA
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	0	0

PROCEDURE:

(i) Connections are given as per circuit diagram.

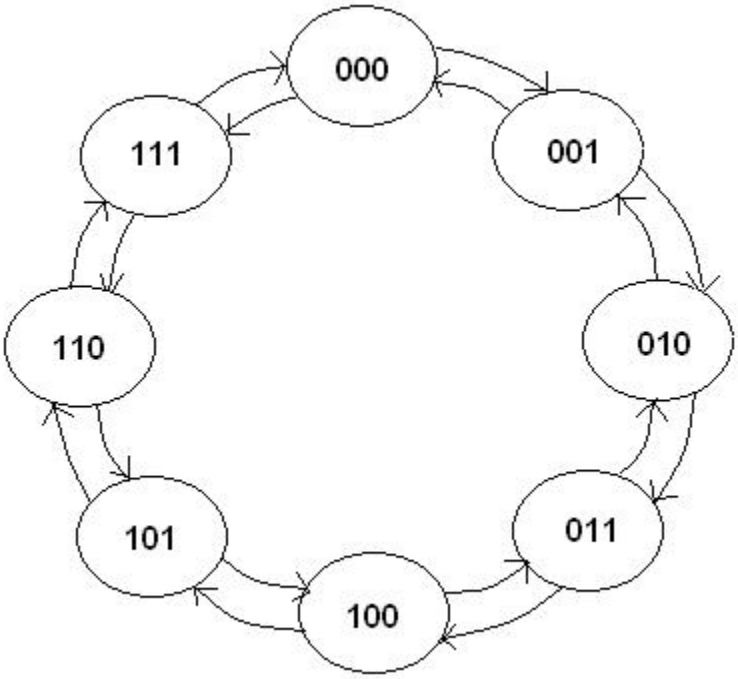
(ii) Logical inputs are given as per circuit diagram.

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

RESULT: -

Thus the 4bit ripple counter and Mod counter circuits were designed and their logic was verified.

STATE DIAGRAM:



CHARACTERISTICS TABLE:

Q	Q_{t+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

EX NO:10 DATE:	DESIGN AND IMPLEMENTATION OF 3 BIT SYNCHRONOUS UP/DOWN COUNTER

AIM:

To design and implement 3 bit synchronous up/down counter.

APPARATUS REQUIRED:

Sl.No.	COMPONENT	SPECIFICATION	QTY.
1.	JK FLIP FLOP	IC 7476	2
2.	3 I/P AND GATE	IC 7411	1
3.	OR GATE	IC 7432	1
4.	XOR GATE	IC 7486	1
5.	NOT GATE	IC 7404	1
6.	IC TRAINER KIT	-	1
7.	PATCH CORDS	-	35

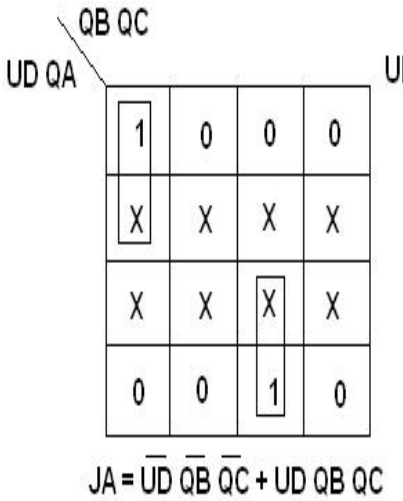
THEORY:

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. An up/down counter is one that is capable of progressing in increasing order or decreasing order through a certain sequence. An up/down counter is also called bidirectional counter. Usually up/down operation of the counter is controlled by up/down signal. When this signal is high counter goes through up sequence and when up/down signal is low counter follows reverse sequence.

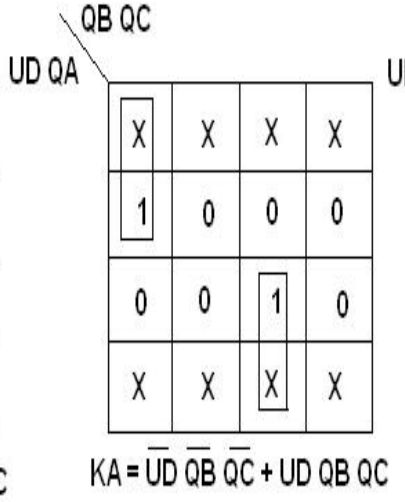
TRUTH TABLE:

Input	Present State			Next State			A		B		C	
Up/Down	Q_A	Q_B	Q_C	Q_{A+1}	Q_{B+1}	Q_{C+1}	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	1	1	1	1	X	1	X	1	X
0	1	1	1	1	1	0	X	0	X	0	X	1
0	1	1	0	1	0	1	X	0	X	1	1	X
0	1	0	1	1	0	0	X	0	0	X	X	1
0	1	0	0	0	1	1	X	1	1	X	1	X
0	0	1	1	0	1	0	0	X	X	0	X	1
0	0	1	0	0	0	1	0	X	X	1	1	X
0	0	0	1	0	0	0	0	X	0	X	X	1
1	0	0	0	0	0	1	0	X	0	X	1	X
1	0	0	1	0	1	0	0	X	1	X	X	1
1	0	1	0	0	1	1	0	X	X	0	1	X
1	0	1	1	1	0	0	1	X	X	1	X	1
1	1	0	0	1	0	1	X	0	0	X	1	X
1	1	0	1	1	1	0	X	0	1	X	X	1
1	1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	1	0	0	0	X	1	X	1	X	1

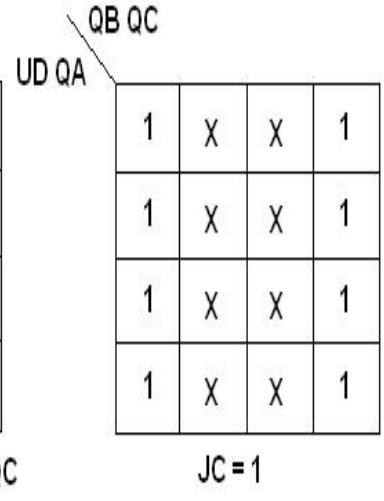
K MAP



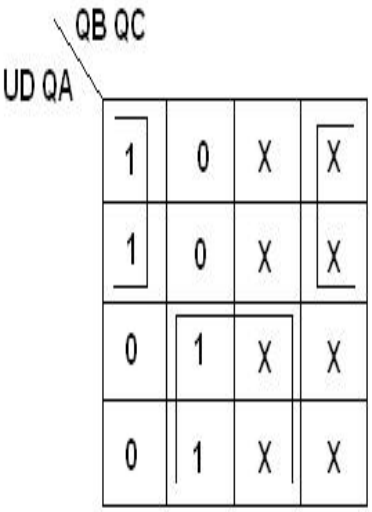
$JA = \overline{UD}\ \overline{QB}\ \overline{QC} + UD\ QB\ QC$



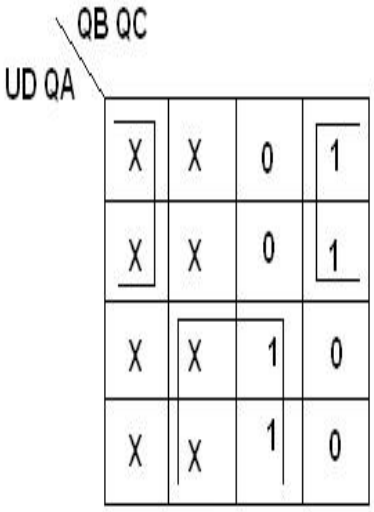
$KA = \overline{UD}\ \overline{QB}\ \overline{QC} + UD\ QB\ QC$



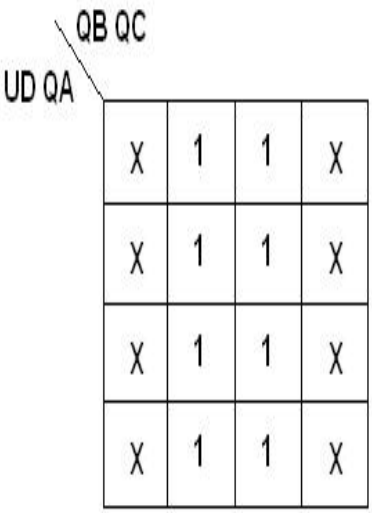
$JC = 1$



$JB = \overline{UD \oplus QC}$

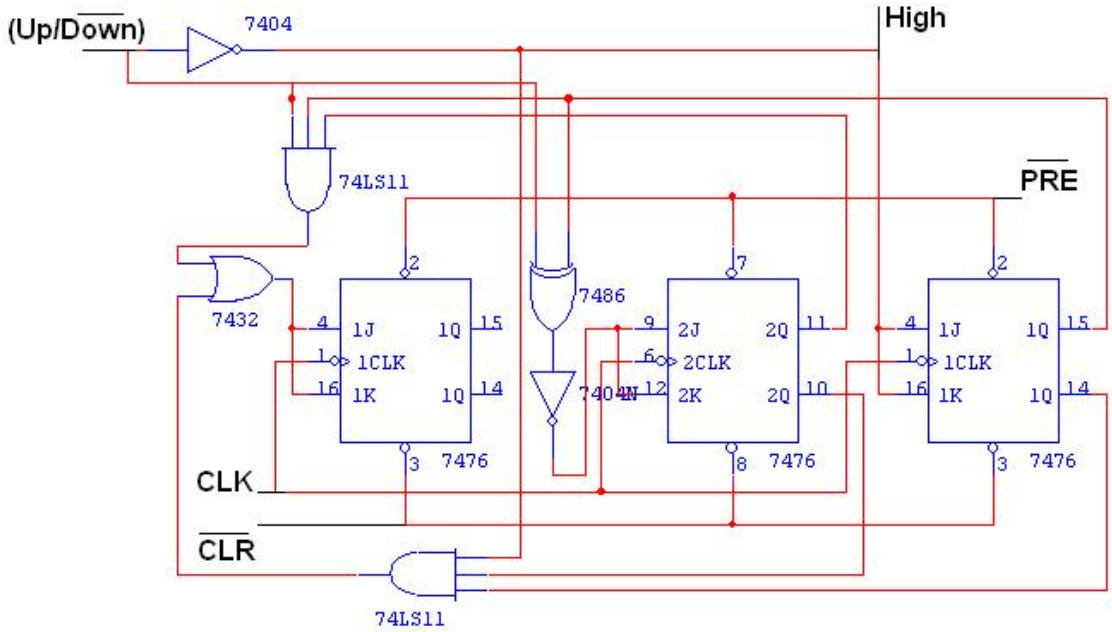


$KB = \overline{(UD \oplus QC)}$



$KC = 1$

LOGIC DIAGRAM:



PROCEDURE:

(i) Connections are given as per circuit diagram.

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

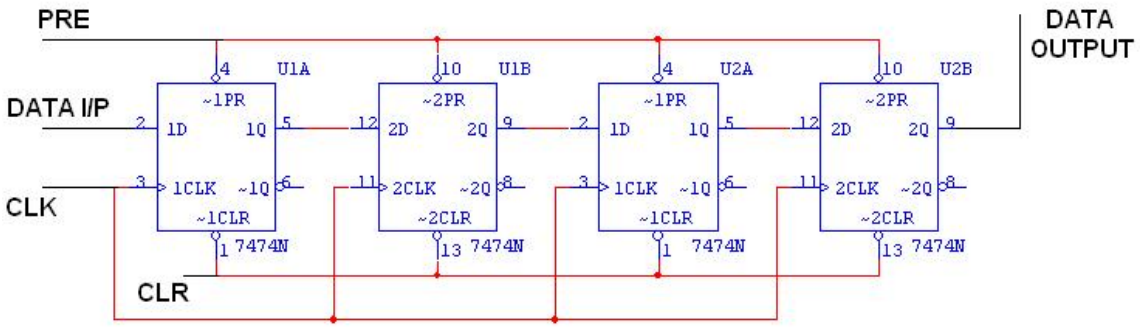
(iii) Observe the output and verify the truth table.

RESULT: -

Thus the 3 bit synchronous up/down counter circuits were designed and their logic was verified.

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

EX NO: 11 DATE:	DESIGN AND IMPLEMENTATION OF SHIFT REGISTER

AIM:

To design and implement

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

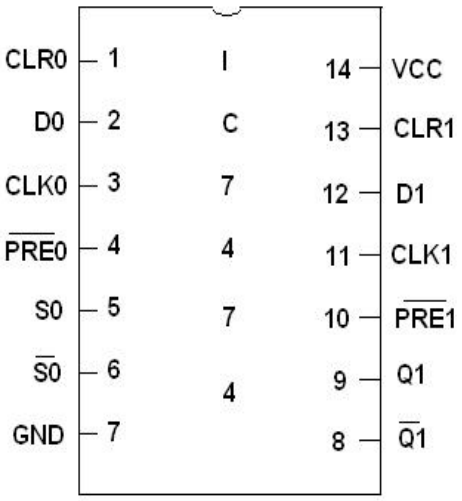
APPARATUS REQUIRED:

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:

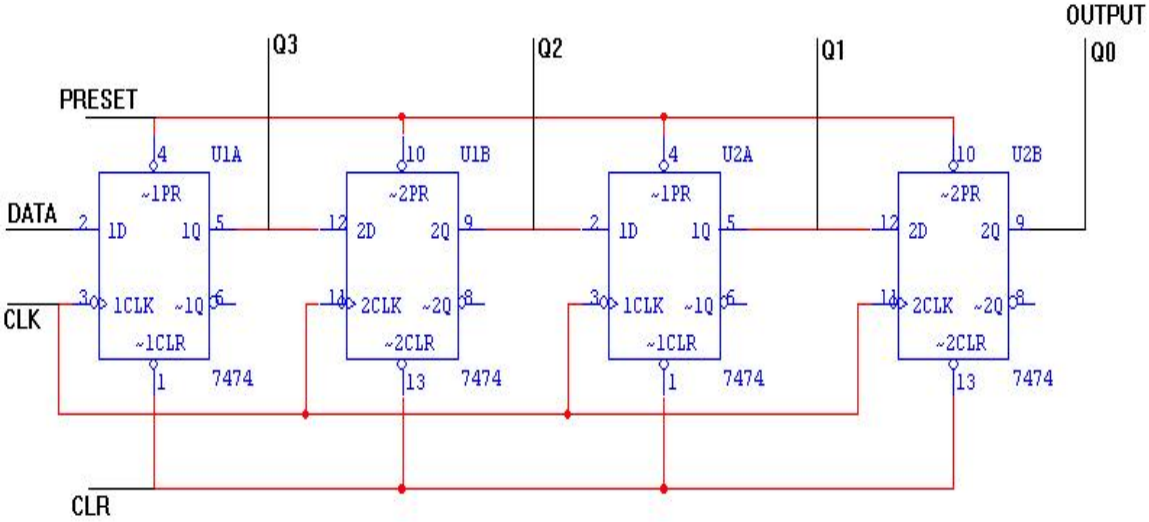
A register is capable of shifting its binary information in one or both directions is known as shift register. The logical configuration of shift register consist of a D-Flip flop cascaded with output of one flip flop connected to input of next flip flop. All flip flops receive common clock pulses which 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:



LOGIC DIAGRAM:

SERIAL IN PARALLEL OUT:



TRUTH TABLE (SERIAL IN PARALLEL OUT):

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

LOGIC DIAGRAM:

PARALLEL IN SERIAL OUT:

TRUTH TABLE (PARALLEL IN SERIAL OUT):

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

LOGIC DIAGRAM:

PARALLEL IN PARALLEL OUT:

TRUTH TABLE:

LD/ST	CLK	DATA INPUT				OUTPUT			
		D_A	D_B	D_C	D_D	Q_A	Q_B	Q_C	Q_D
1	1	1	0	0	1	1	0	0	1
0	2	1	0	0	1	0	1	0	0
0	3	1	0	0	1	0	0	1	0
0	4	1	0	0	1	0	0	0	1
0	5	1	0	0	1	0	0	0	0

PROCEDURE:

(i) Connections are given as per circuit diagram.

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

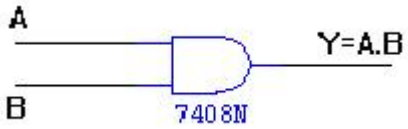
(iii) Observe the output and verify the truth table.

RESULT: -

Thus the shift register circuits were designed and their logic was verified.

AND GATE:

SYMBOL:



TRUTH TABLE:

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

PROGRAM:

```
module andg(y,a,b);
input a,b;
output y;
and g1 (y,a,b);
endmodule
```

OR GATE:

TRUTH TABLE:

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

PROGRAM:

```
module org(y,a,b);
input a,b;
output y;
or g1 (y,a,b);
endmodule
```

EX NO:12	SIMULATION OF LOGIC GATES
DATE:	

AIM:

To simulate the logic gates using Verilog HDL tool and verify their truth tables.

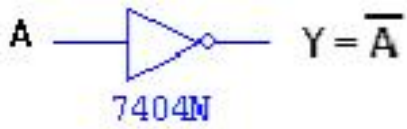
APPARATUS REQUIRED:

SL No.	COMPONENT	SPECIFICATION
1.	PC	Desktop
2.	Xilinx	14.5

LOGIC SYMBOL AND TRUTH TABLE:

NOT GATE

SYMBOL:



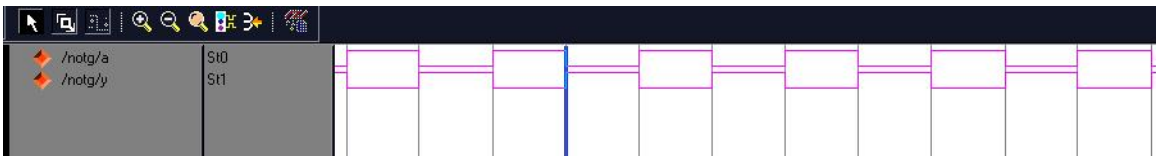
TRUTH TABLE:

A	A’
0	1
1	0

PROGRAM:

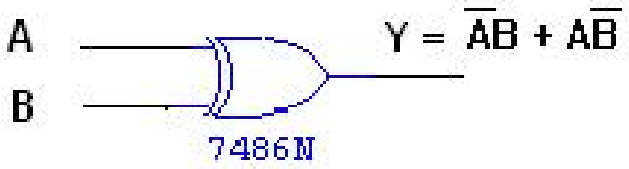
```
module notg(y,a);
input a;
output y;
assign y=~a;
endmodule
```

OUTPUT:



EX-OR GATE :

SYMBOL :



TRUTH TABLE:

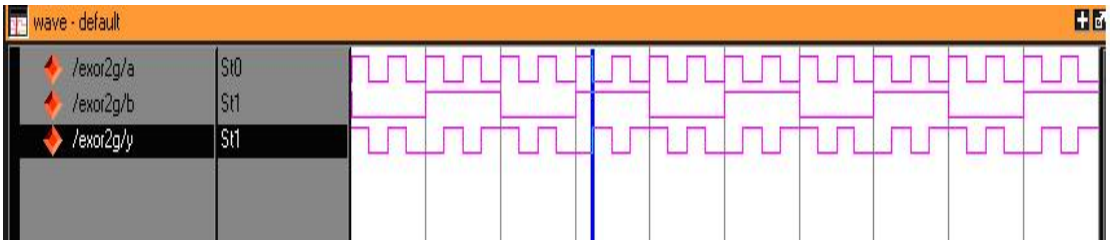
A	B	A'B+AB'
0	0	1
0	1	0
1	0	0
1	1	1

PROGRAM:

```
module exor2g(y,a,b);
input a,b;
output y;
```

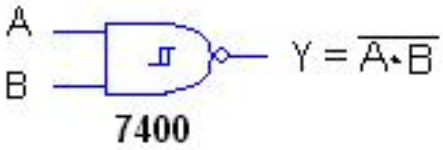
```
xor g1 (y,a,b);
endmodule
```

OUTPUT:



NAND GATE:

SYMBOL:



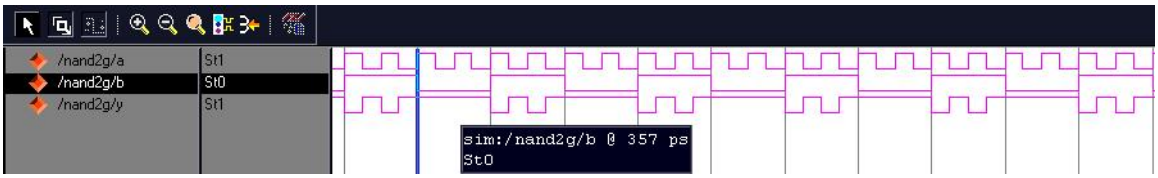
TRUTH TABLE:

A	B	(A.B)'
0	0	1
0	1	1
1	0	1
1	1	0

PROGRAM:

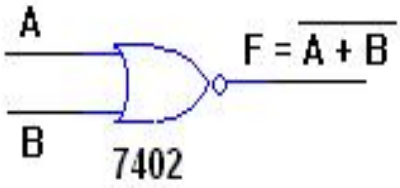
```
module nand2g(y,a,b);
input a,b;
output y;
assign y=~(a & b);
endmodule
```

OUTPUT:



NOR GATE:

SYMBOL:



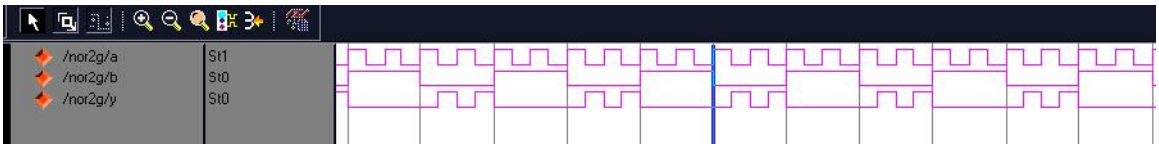
TRUTH TABLE:

A	B	(A+B)'
0	0	1
0	1	0
1	0	0
1	1	0

PROGRAM:

```
module nor2g(y,a,b);
input a,b;
output y;
nor g1 (y,a,b);
endmodule
```

OUTPUT:

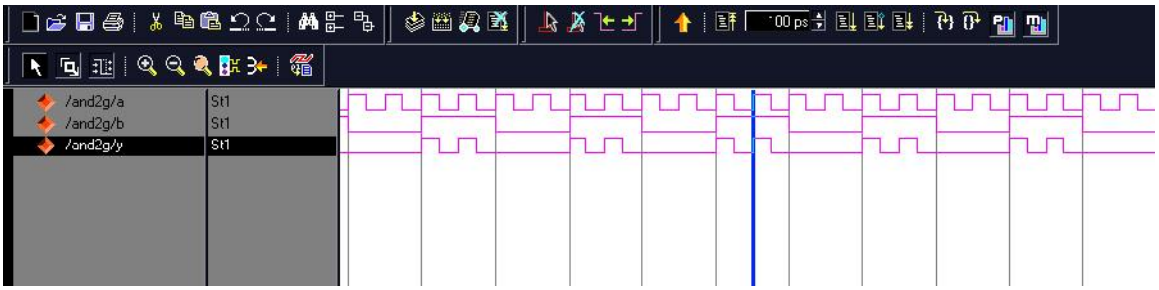


AND GATE

PROGRAM:

```
module and2g(y,a,b);
input a,b;
output y;
assign y=a & b;
endmodule
```

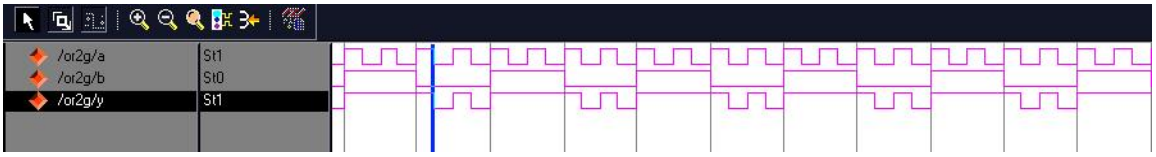
OUTPUT:



OR GATE

```
PROGRAM:
module or2g(y,a,b);
input a,b;
output y;
assign y=a | b;
endmodule
```

OUTPUT:

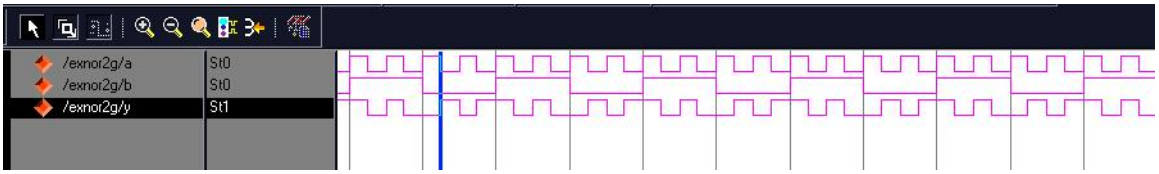


EXNOR GATE

PROGRAM:

```
module exnor2g(y,a,b);
input a,b;
output y;
xnor g1 (y,a,b);
endmodule
```

OUTPUT:



PROCEDURE:

(i) The program is written the ModelSim based on the given design.

(ii) Compile the program.

(iii) Simulate the program.

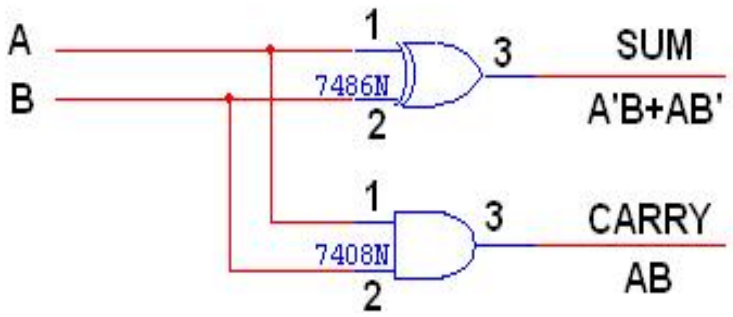
(iv) Verify the output in the waveform window.

RESULT: -

The logic gates have been simulated and their truth tables have been verified.

LOGIC DIAGRAM:

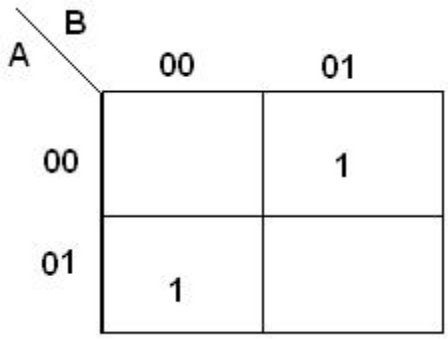
HALF ADDER



TRUTH TABLE:

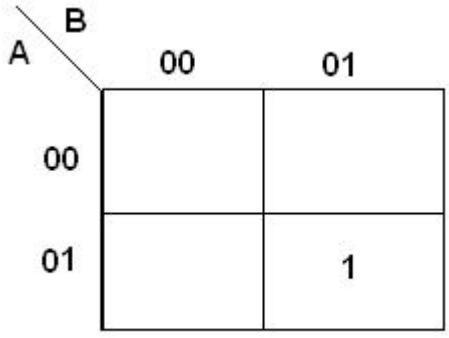
A	B	CARRY	SUM
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

K-Map for SUM:



SUM = A'B + AB'

K-Map for CARRY:



CARRY = AB

EX NO:13	SIMULATION OF ADDER AND SUBTRACTOR
DATE:	

AIM:

To design and simulate 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.	PC		1
2.	ModelSim	6.1e	1

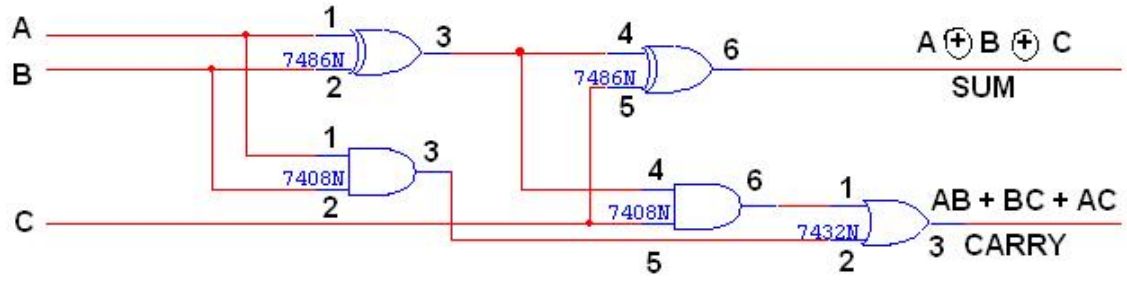
THEORY:

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.

LOGIC DIAGRAM:

FULL ADDER
(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:

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

SUM = A'B'C + A'BC' + ABC' + ABC

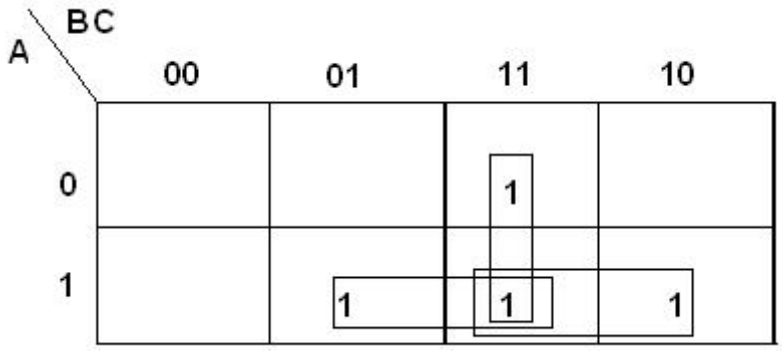
FULL ADDER:

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.

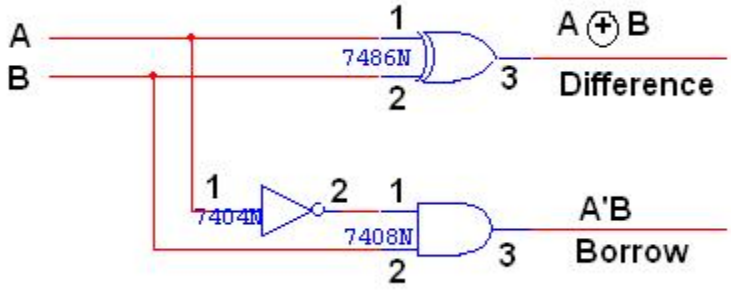
K-Map for CARRY:



CARRY = AB + BC + AC

LOGIC DIAGRAM:

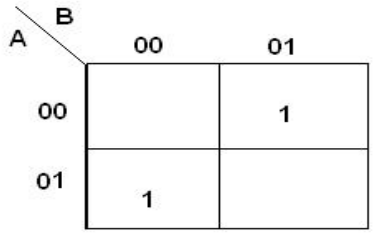
HALF SUBTRACTOR



TRUTH TABLE:

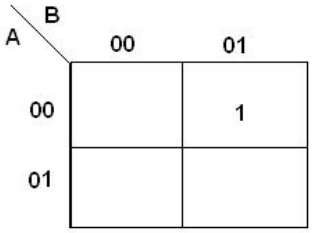
A	B	BORROW	DIFFERENCE
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

K-Map for DIFFERENCE:



DIFFERENCE = A'B + AB'

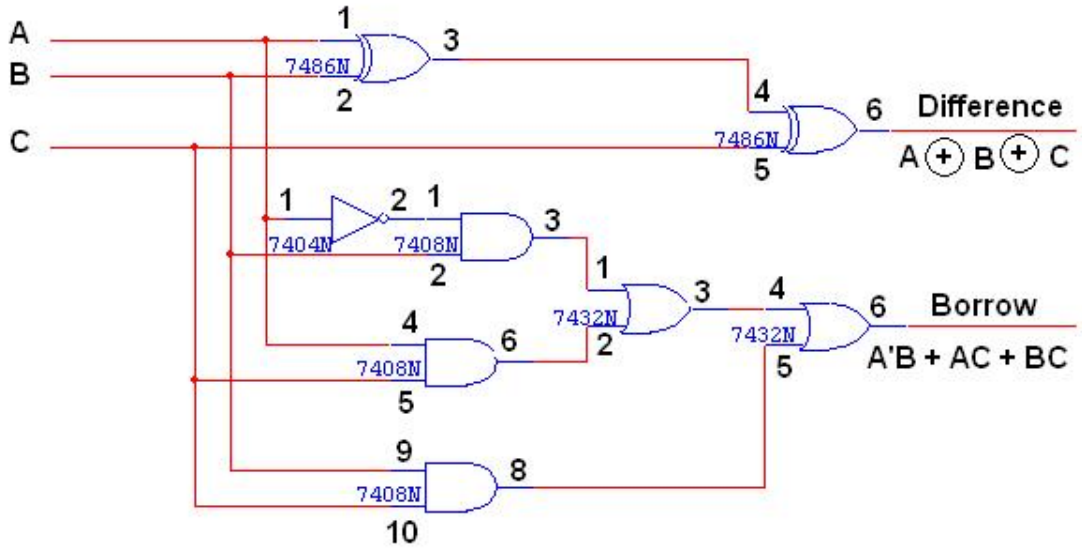
K-Map for BORROW:



BORROW = A'B

LOGIC DIAGRAM:

FULL SUBTRACTOR



FULL SUBTRACTOR:

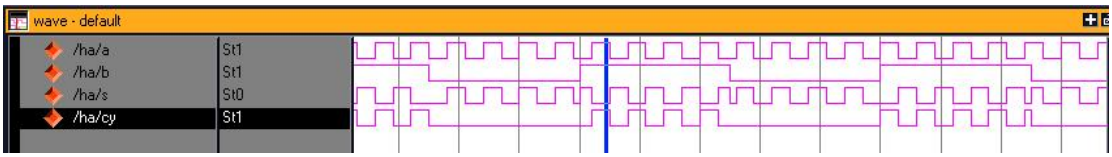
The full subtractor is a combination of X-OR, AND, OR, NOT Gates. In a full subtractor the logic circuit should have three inputs and two outputs. The two half subtractor put together gives a full subtractor .The first half subtractor will be C and A B. The output will be difference output of full subtractor.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

PROGRAM:

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

OUTPUT:

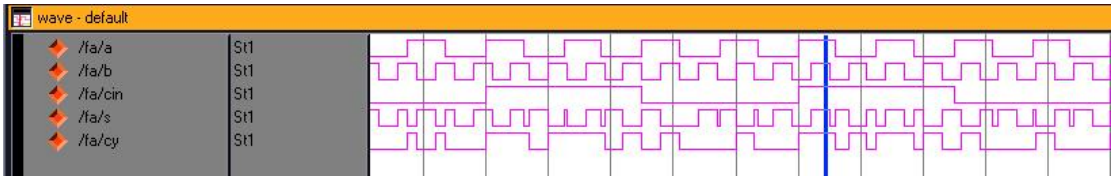


FULL ADDER

PROGRAM:

```
module fa(s,cy,a,b,cin);
    input a,b,cin;
    output s,cy;
    assign s=(a ^ b ^ cin);
    assign cy=((a & b) | (b & cin) | (cin & a));
endmodule
```

OUTPUT:

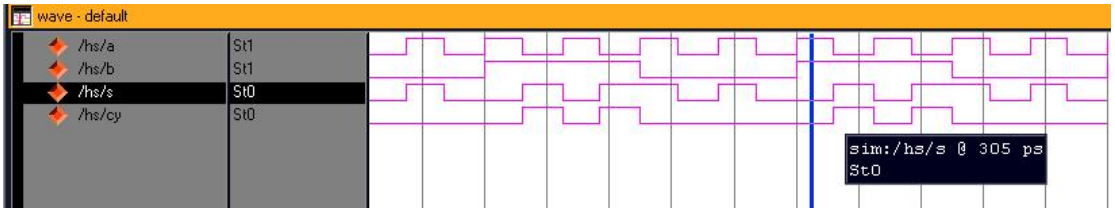


HALF SUBTRACTOR

PROGRAM:

```
module hs(s,cy,a,b);
    input a,b;
    output s,cy;
    assign s=(a ^ b);
    assign cy=((~a) & b);
endmodule
```

OUTPUT:

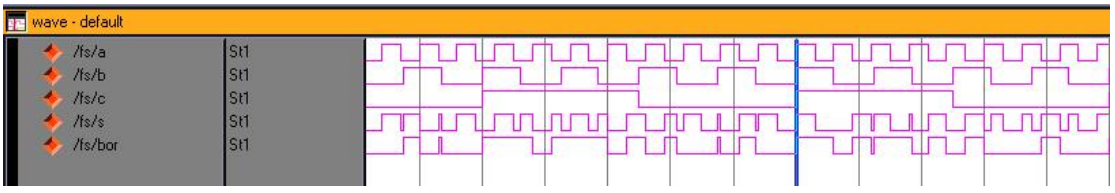


FULL SUBTRACTOR

PROGRAM:

```
module fs(s,bor,a,b,c);
    input a,b,c;
    output s,bor;
    assign s=(a ^ b ^ c);
    assign bor=(((~a) & b) | (b & c) | (c & (~a)));
endmodule
```

OUTPUT:



PROCEDURE:

(i) The program is written the ModelSim based on the given design.

(ii) Compile the program.

(iii) Simulate the program.

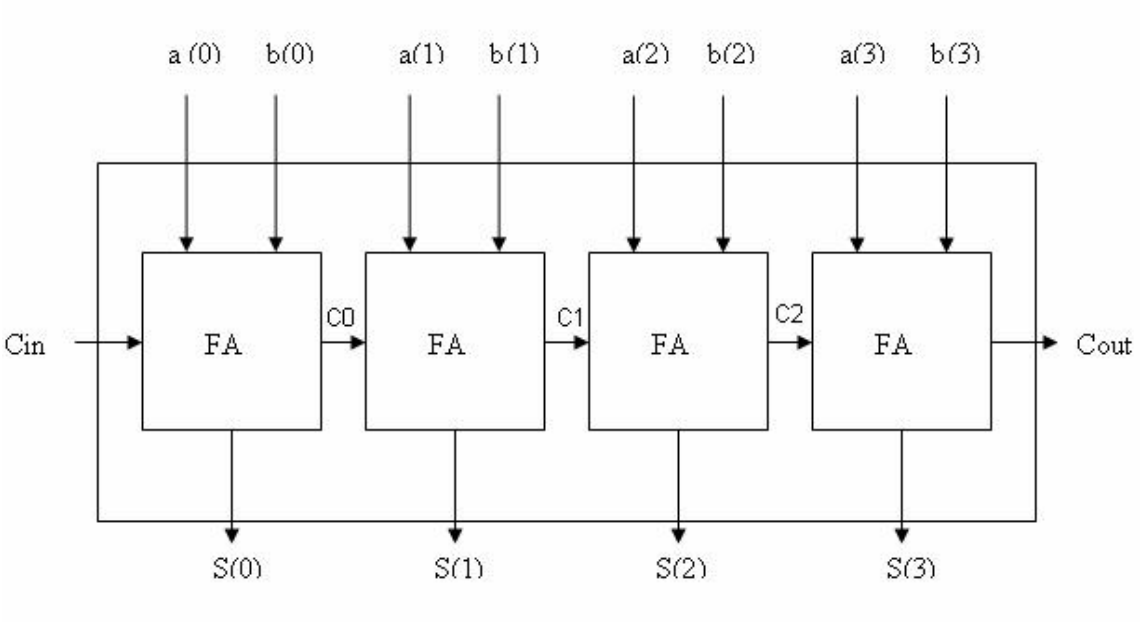
(iv) Verify the output in the waveform window.

RESULT: -

The adders and subtractors have been designed and simulated and their truth tables have been verified.

LOGIC DIAGRAM:

4-BIT BINARY ADDER



EX NO:14	DESIGN OF 4-BIT ADDER AND SUBTRACTOR
DATE:	

AIM:

To design and simulate 4-bit adder and subtractor using Verilog HDL tool.

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

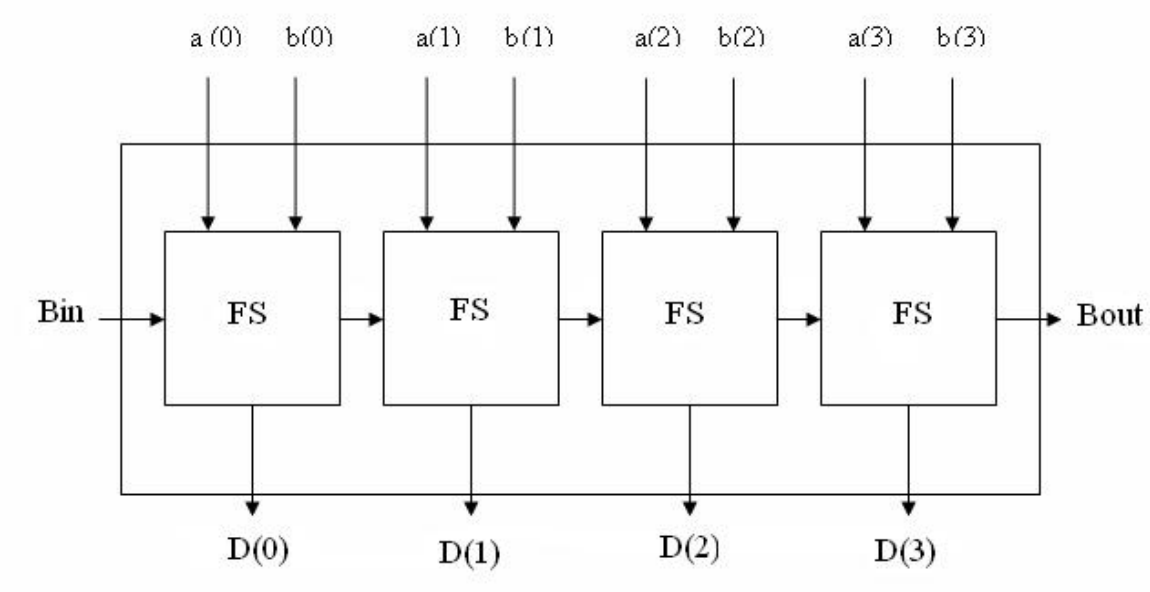
THEORY:

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.

LOGIC DIAGRAM:

4-BIT BINARY SUBTRACTOR



4 BIT BINARY SUBTRACTOR:

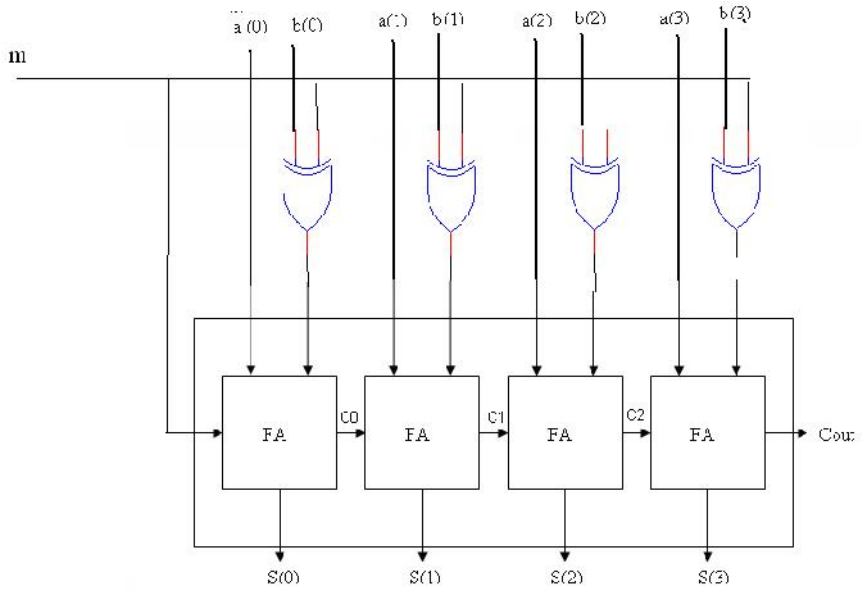
The circuit for subtracting A-B consists of an adder with inverters, placed between each data input 'B' and the corresponding input of full adder. The input carry C_0 must be equal to 1 when performing subtraction.

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.

LOGIC DIAGRAM:

4-BIT BINARY ADDER/SUBTRACTOR



4 BIT BINARY TRUTH TABLE:

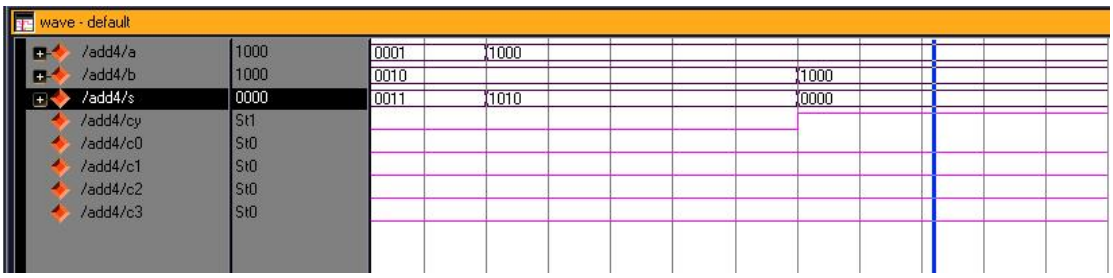
Input Data A				Input Data B				Addition					Subtraction				
A4	A3	A2	A1	B4	B3	B2	B1	C	S4	S3	S2	S1	B	D4	D3	D2	D1
1	0	0	0	0	0	1	0	0	1	0	1	0	1	0	1	1	0
1	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0
0	0	1	0	1	0	0	0	0	1	0	1	0	0	1	0	1	0
0	0	0	1	0	1	1	1	0	1	0	0	0	0	1	0	1	0
1	0	1	0	1	0	1	1	1	0	0	1	0	0	1	1	1	1
1	1	1	0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

FOUR BIT ADDER

```
module add4(s,cy,a,b);
    input [3:0] a,b ;
    output [3:0] s ;
    output cy;
    wire c0,c1,c2,c3;
assign c0=1'b0;
fa f1 (s[0],c1,a[0],b[0],c0);
fa f2 (s[1],c2,a[1],b[1],c1);
fa f3 (s[2],c3,a[2],b[2],c2);
fa f4 (s[3],cy,a[3],b[3],c3);
endmodule
```

PROGRAM:

OUTPUT:



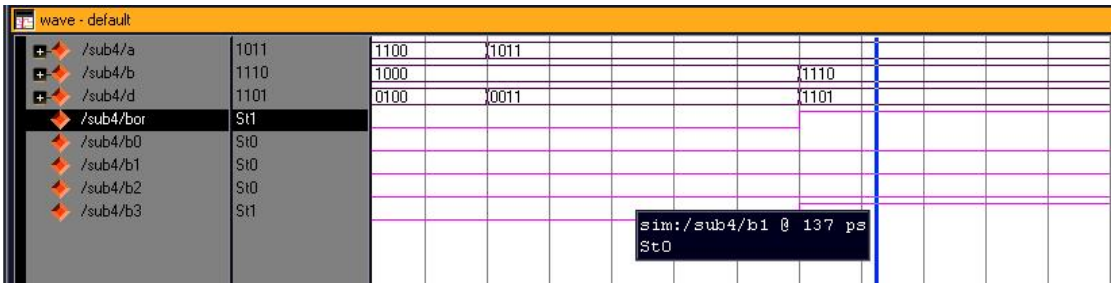
FOUR BIT SUBTRACTOR

PROGRAM:

```
module sub4(d,bor,a,b);
    input [3:0] a,b ;
    output [3:0] d ;
    output bor;
    wire b0,b1,b2,b3;
assign b0=1'b0;
fs f1 (d[0],b1,a[0],b[0],b0);
fs f2 (d[1],b2,a[1],b[1],b1);
fs f3 (d[2],b3,a[2],b[2],b2);
fs f4 (d[3],bor,a[3],b[3],b3);
```

```
endmodule
```

OUTPUT:

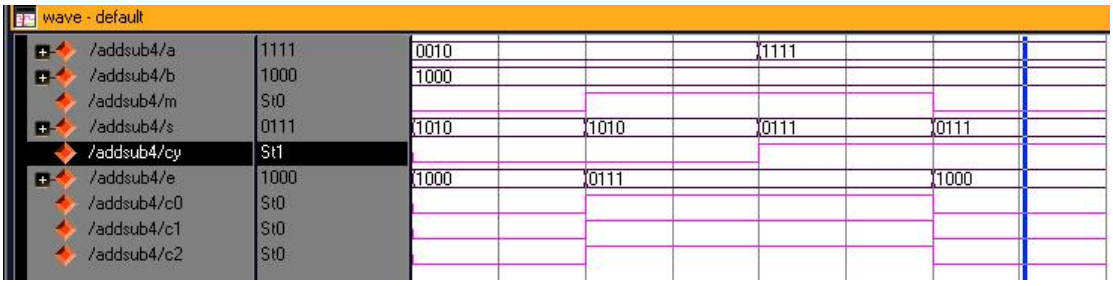


FOUR BIT ADDER / SUBTRACTOR

PROGRAM:

```
module addsub4(s,cy,a,b,m);
    input [3:0] a,b ;
    input m;
    output [3:0] s ;
    output cy;
    wire [3:0] e;
    wire c0,c1,c2;
xor e1 (e[3],m,b[3]);
xor e2 (e[2],m,b[2]);
xor e3 (e[1],m,b[1]);
xor e4 (e[0],m,b[0]);
fa f1 (s[0],c0,a[0],e[0],m);
fa f2 (s[1],c1,a[1],e[1],c0);
fa f3 (s[2],c2,a[2],e[2],c1);
fa f4 (s[3],cy,a[3],e[3],c2);
endmodule
```

OUTPUT:



PROCEDURE:

(v) The program is written the ModelSim based on the given design.

(vi) Compile the program.

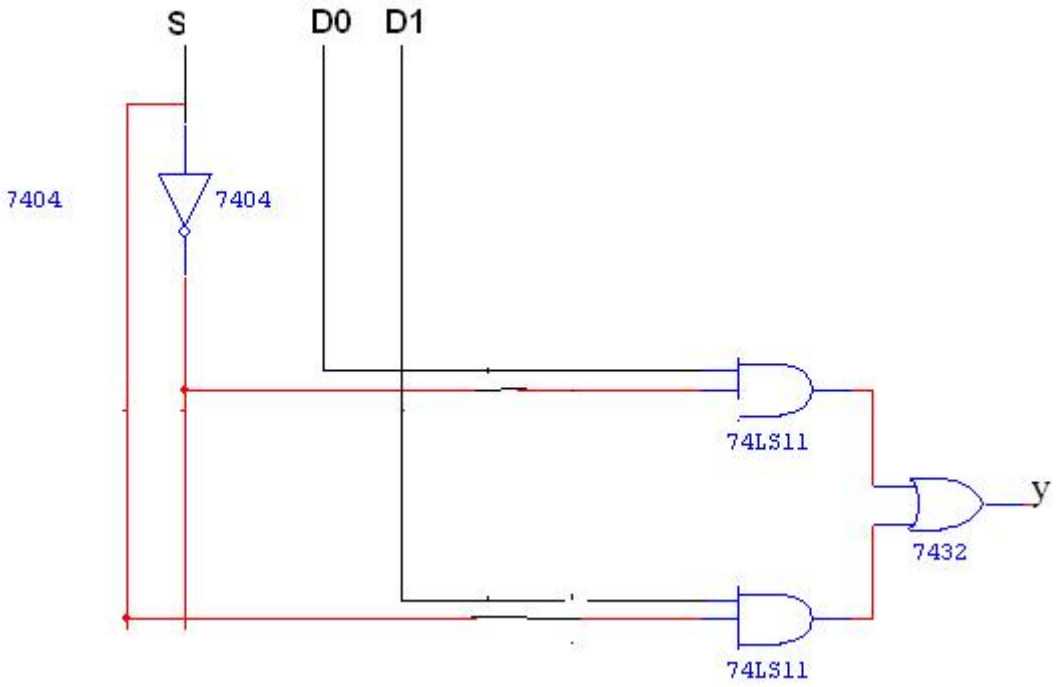
(vii) Simulate the program.

(viii) Verify the output in the waveform window.

RESULT: -

The logic gates have been simulated and their truth tables have been verified.

CIRCUIT DIAGRAM FOR 2X1 MULTIPLEXER:



TRUTH TABLE:

S	Y = OUTPUT
0	D0
1	D1

EX NO:15 DATE:	DESIGN AND IMPLEMENTATION OF MULTIPLEXER AND DEMULTIPLEXER

AIM:

To design and simulate multiplexer and demultiplexer using Verilog HDL tool.

APPARATUS REQUIRED:

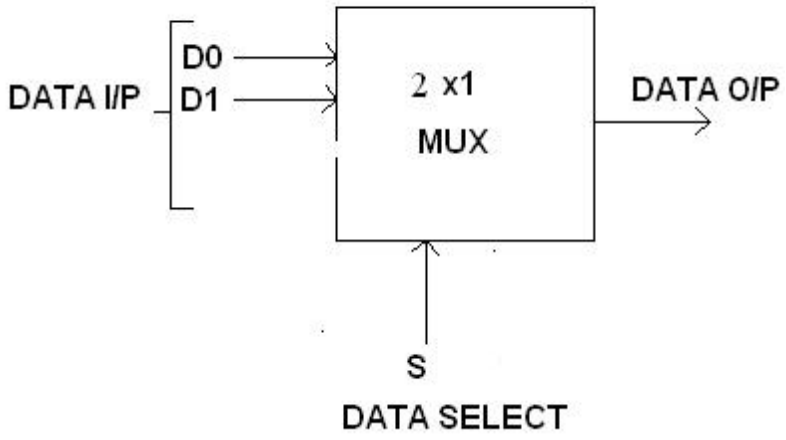
SL No.	COMPONENT	SPECIFICATION	QTY
1.	PC		1
2.	ModelSim	6.1e	1

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.

BLOCK DIAGRAM FOR 2:1 MULTIPLEXER:



FUNCTION TABLE:

S	INPUTS Y
0	D0 → D0 S’
1	D1 → D1 S

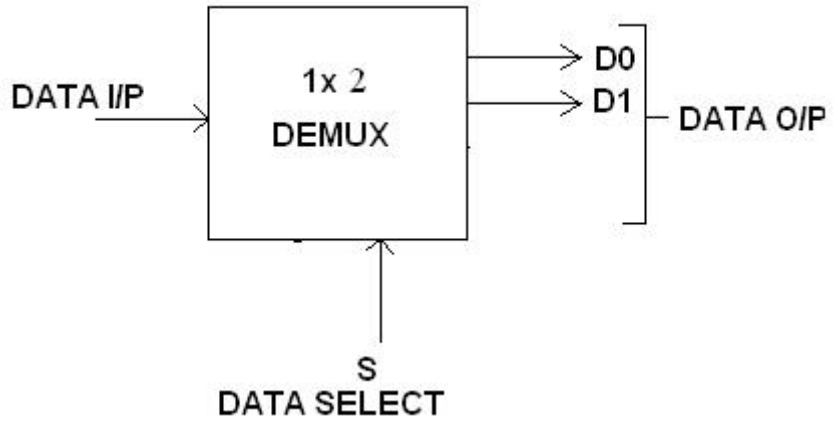
Y = D0 S’ + D1 S

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.

BLOCK DIAGRAM FOR 1:2 DEMULTIPLEXER:

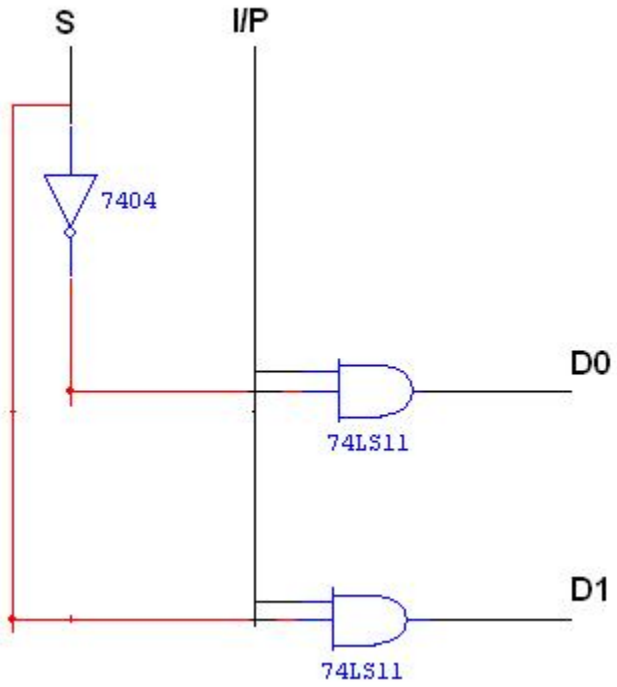


FUNCTION TABLE:

S	INPUT
0	X → D0 = X S'
1	X → D1 = X S

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

LOGIC DIAGRAM FOR DEMULTIPLEXER:



TRUTH TABLE:

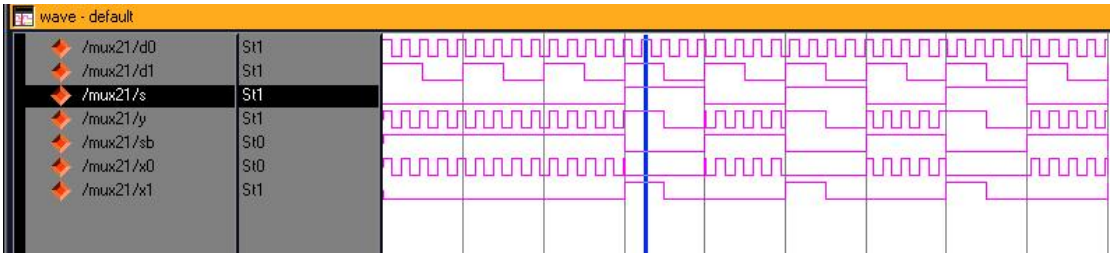
INPUT		OUTPUT	
S	I/P	D0	D1
0	0	0	0
0	1	1	0
1	0	0	0
1	1	0	1

MULTIPLEXER

PROGRAM:

```
module mux21(d0,d1,s,y);
    input d0,d1,s;
    output y;
    wire sb,x0,x1;
    not w1 (sb,s);
    and w2 (x0,d0,sb);
    and w3 (x1,d1,s);
    or w4 (y,x0,x1);
endmodule
```

OUTPUT:

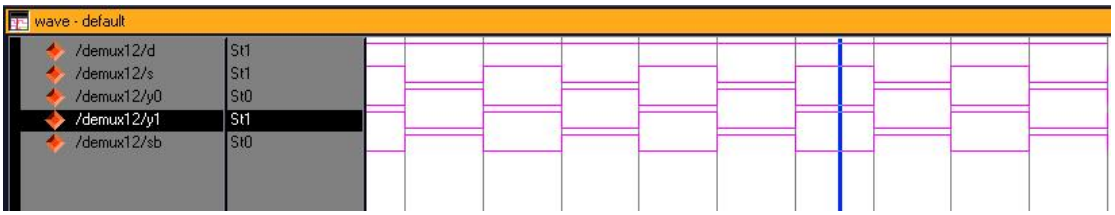


DEMULTIPLEXER

PROGRAM:

```
module demux12(y0,y1,s,d);
    input d,s;
    output y0,y1;
    wire sb;
    not n1 (sb,s);
    and f1 (y0,d,sb);
    and f2 (y1,d,s);
endmodule
```

OUTPUT:


wave - default
/demux12/d
/demux12/s
/demux12/y0
/demux12/y1
/demux12/sb
St1
St1
St0
St1
St0

PROCEDURE:

(i) The program is written the ModelSim based on the given design.

(ii) Compile the program.

(iii) Simulate the program.

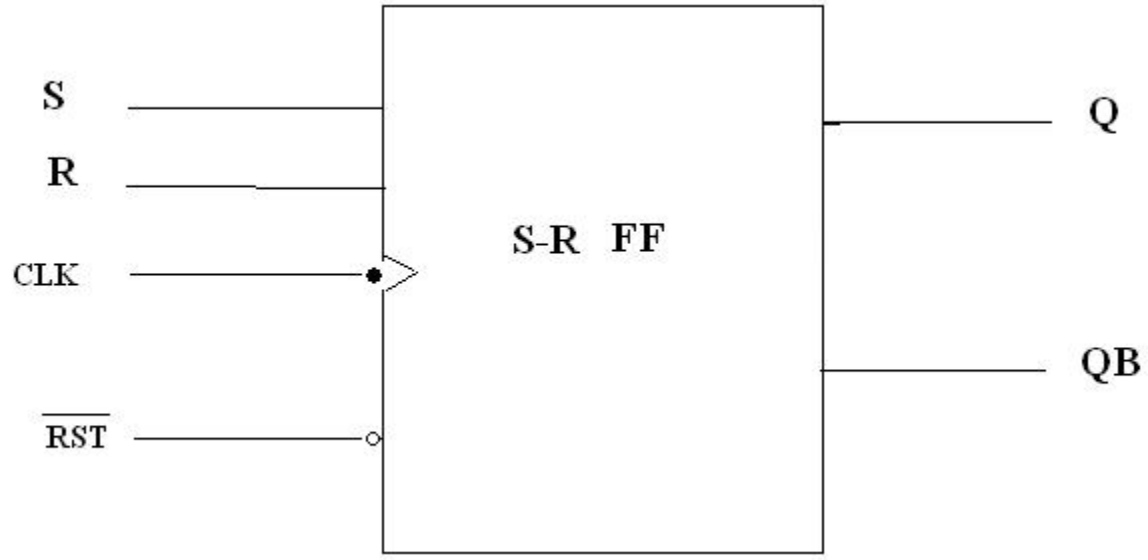
(iv) Verify the output in the waveform window.

RESULT: -

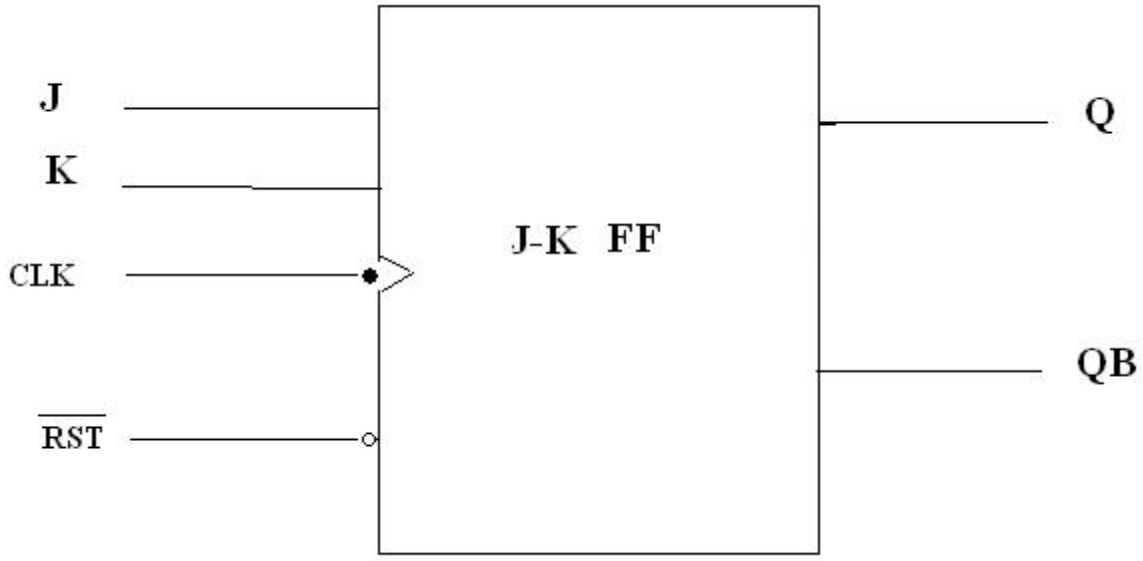
The multiplexer/demultiplexer have been simulated and their truth tables have been verified.

BLOCK DIAGRAM:

S-R FLIPFLOP



J-K FLIPFLOP



EX NO:16 DATE:	DESIGN AND SIMULATION OF FLIP-FLOPS

AIM:

To design and simulate flip-flops using Verilog HDL tool

APPARATUS REQUIRED:

SL No.	COMPONENT	SPECIFICATION
1.	PC	
2.	ModelSim	6.1e

PROCEDURE:

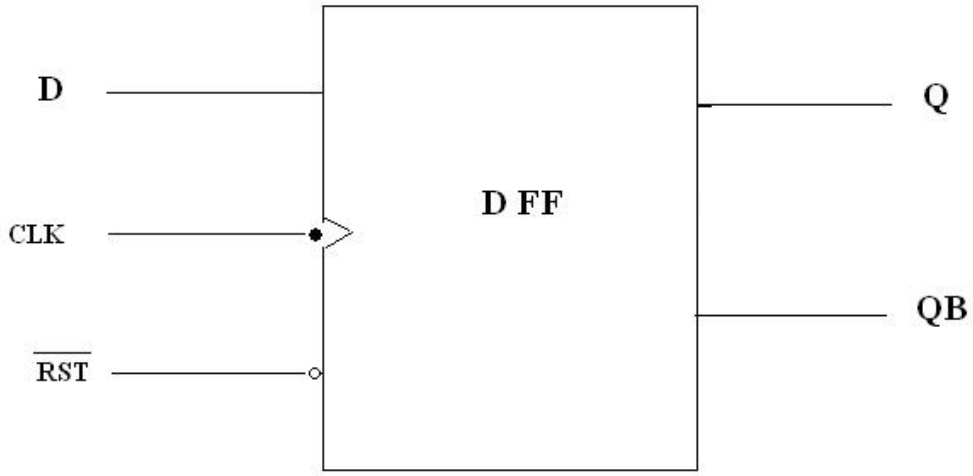
(i) The program is written the ModelSim based on the given design.

(ii) Compile the program.

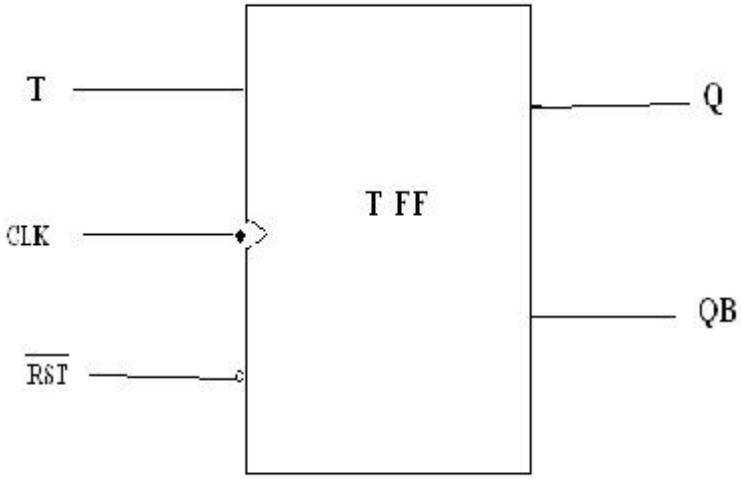
(iii) Simulate the program.

(iv) Verify the output in the waveform window.

D FLIPFLOP



T FLIPFLOP

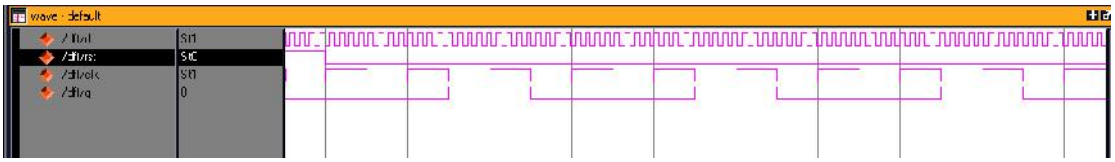


D- FLIPFLOP

PROGRAM:

```
module dff(q,d,rst,clk);
    input d,rst,clk;
    output q;
    reg q;
always@(posedge rst or negedge clk)
if (rst)
q=1'b0;
else
q=d;
endmodule
```

OUTPUT:

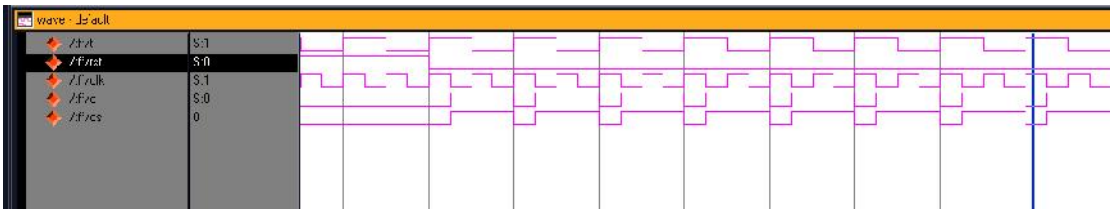


T FLIPFLOP

PROGRAM:

```
module tff(q,t,rst,clk);
    input t,rst,clk;
    output q;
    reg qs;
always@(posedge rst or negedge clk)
if (rst)
qs=1'b0;
else
assign qs=((~t) & qs) | (t & (~qs));
assign q=qs;
endmodule
```

OUTPUT:

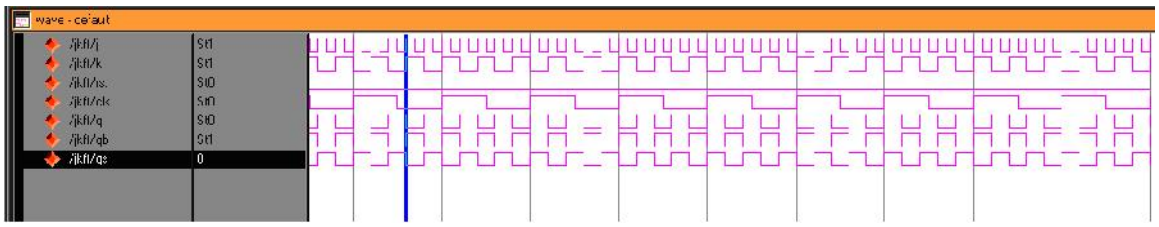


JK FLIPFLOP

PROGRAM:

```
module jkff(q,qb,j,k,rst,clk);
    input j,k,rst,clk;
    output q,qb;
    reg qs;
always@(posedge rst or negedge clk)
if (rst)
qs=1'b0;
else
assign qs=((~k) & qs) | (j & (~qs));
assign q=qs;
assign qb=~qs;
endmodule
```

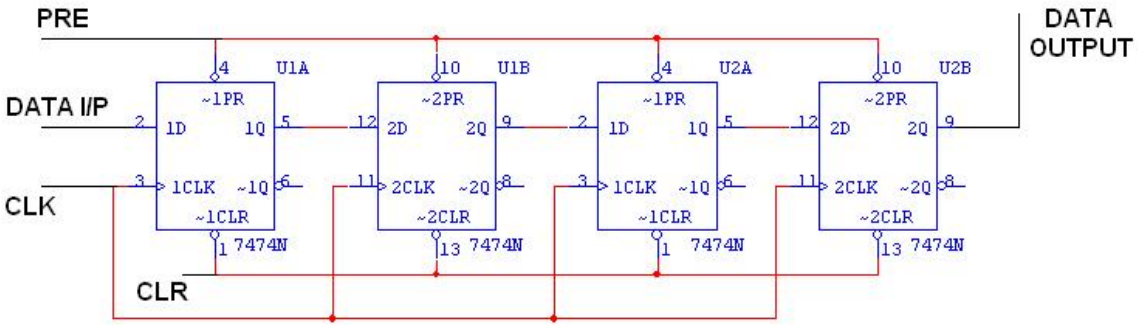
OUTPUT:



RESULT: -

The flip-flops have been simulated and their truth tables have been verified.

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

EX NO:17	DESIGN AND SIMULATION OF SHIFT REGISTER
DATE:	

AIM:

To design and simulate

(i) Serial in serial out
(ii) Serial in parallel out
(iii) Parallel in serial out
(iv) Parallel in parallel out using Verilog HDL tool.

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

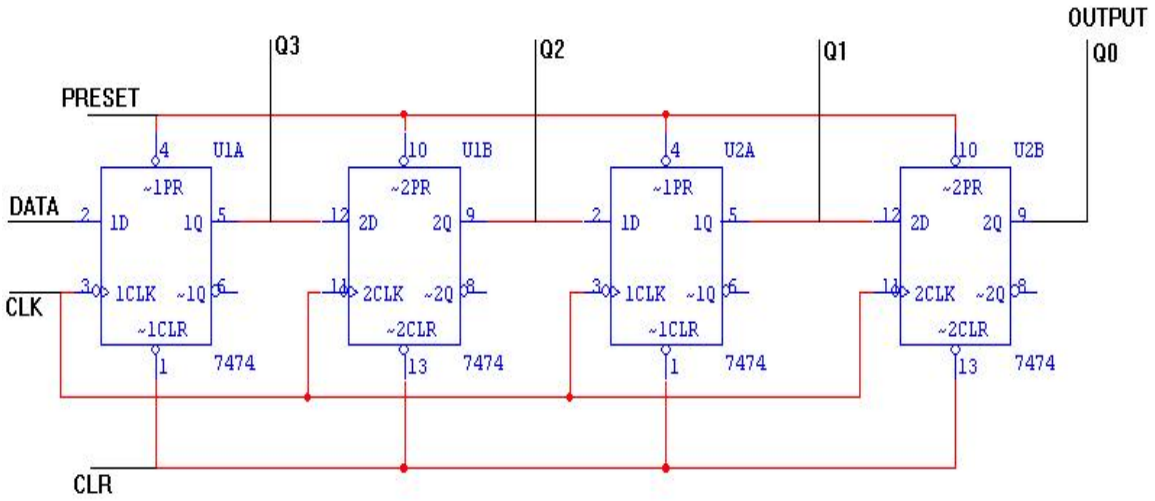
THEORY:

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.

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

SERIAL IN SERIAL OUT SHIFT REGISTER

PROGRAM:

```
module SISO(dout,din,clk,rst);
    input din,rst,clk;
    output dout;
   wire q3,q2,q1;
   dff d1(q3,din,rst,clk);
   dff d2(q2,q3,rst,clk);
   dff d3(q1,q2,rst,clk);
   dff d4(dout,q1,rst,clk);
   endmodule
```

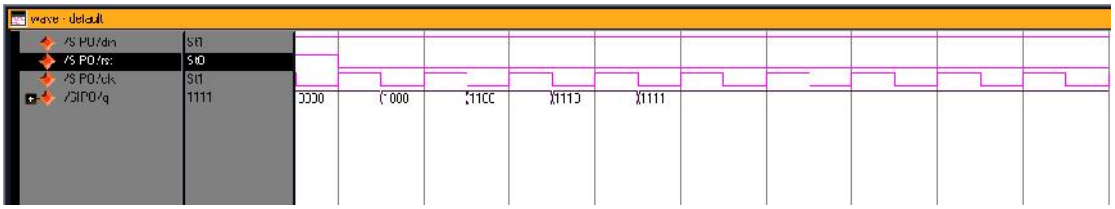
OUTPUT:

SERIAL IN PARALLEL OUT SHIFT REGISTER

PROGRAM:

```
module SIPO(q,din,clk,rst);
    input din,rst,clk;
    inout [3:0] q;
   dff d1(q[3],din,rst,clk);
   dff d2(q[2],q[3],rst,clk);
   dff d3(q[1],q[2],rst,clk);
   dff d4(q[0],q[1],rst,clk);
   endmodule
```

OUTPUT:



LOGIC DIAGRAM:

PARALLEL IN SERIAL OUT:

TRUTH TABLE:

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

PARALLEL IN SERIAL OUT SHIFT REGISTER

PROGRAM:

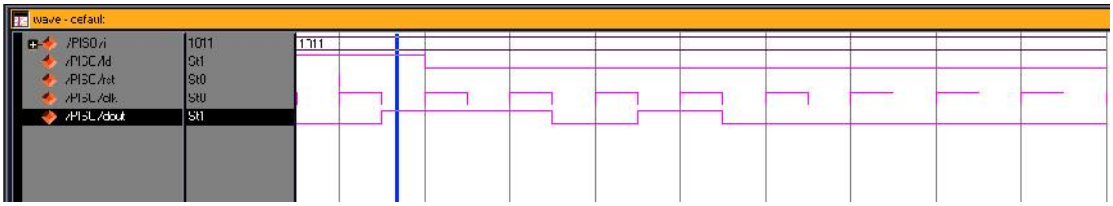
```
module PISO(dout,i,ld,rst,clk);
    input [3:0] i;
    input ld,rst,clk;
    output dout;
    wire [3:0] q;
    wire a1,a2,a3,a4,a5,a6,a7,a8,o1,o2,o3,o4;
    wire j,sh;
    assign j=1'b0;
    not n1 (sh,ld);
    and w1 (a1,j,sh);
    and w2 (a2,q[3],sh);
    and w3 (a3,q[2],sh);
    and w4 (a4,q[1],sh);

    and w5 (a5,i[3],ld);
    and w6 (a6,i[2],ld);
    and w7 (a7,i[1],ld);
    and w8 (a8,i[0],ld);

    or w9 (o1,a1,a5);
    or w10 (o2,a2,a6);
    or w11 (o3,a3,a7);
    or w12 (o4,a4,a8);

    dff d1 (q[3],o1,rst,clk);
    dff d2 (q[2],o2,rst,clk);
    dff d3 (q[1],o3,rst,clk);
    dff d4 (q[0],o4,rst,clk);
    assign dout=q[0];
endmodule
```

OUTPUT:



LOGIC DIAGRAM:

PARALLEL IN PARALLEL OUT:

TRUTH TABLE:

LD/ST	CLK	DATA INPUT				OUTPUT			
		D_A	D_B	D_C	D_D	Q_A	Q_B	Q_C	Q_D
1	1	1	0	0	1	1	0	0	1
0	2	1	0	1	0	1	0	1	0
0	3	1	0	1	0	0	1	0	1
0	4	1	0	1	0	0	0	1	0
0	5	1	0	1	0	0	0	0	1

PARALLEL IN PARALLEL OUT SHIFT REGISTER

PROGRAM:

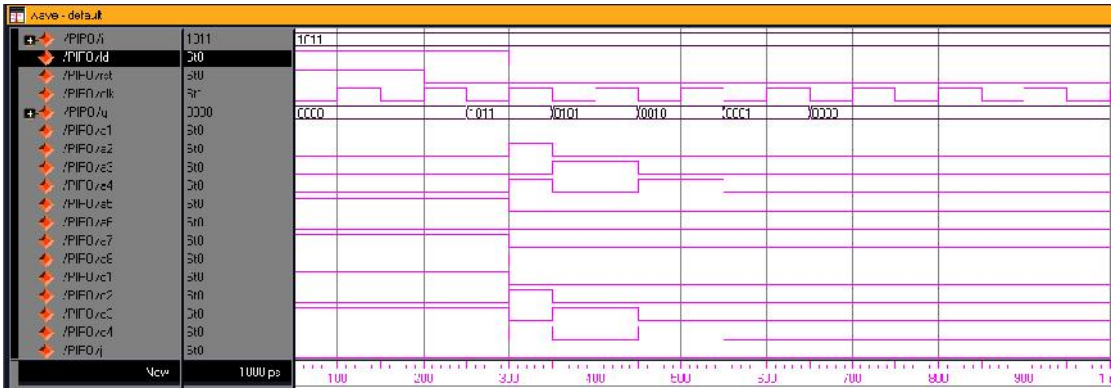
```
module PIPO(q,i,ld,rst,clk);
    input [3:0] i;
    input ld,rst,clk;
    inout [3:0] q;
    wire a1,a2,a3,a4,a5,a6,a7,a8,o1,o2,o3,o4;
    wire j,sh;
    assign j=1'b0;
    not n1 (sh,ld);
    and w1 (a1,j,sh);
    and w2 (a2,q[3],sh);
    and w3 (a3,q[2],sh);
    and w4 (a4,q[1],sh);

    and w5 (a5,i[3],ld);
    and w6 (a6,i[2],ld);
    and w7 (a7,i[1],ld);
    and w8 (a8,i[0],ld);

    or w9 (o1,a1,a5);
    or w10 (o2,a2,a6);
    or w11 (o3,a3,a7);
    or w12 (o4,a4,a8);

    dff d1 (q[3],o1,rst,clk);
    dff d2 (q[2],o2,rst,clk);
    dff d3 (q[1],o3,rst,clk);
    dff d4 (q[0],o4,rst,clk);
endmodule
```

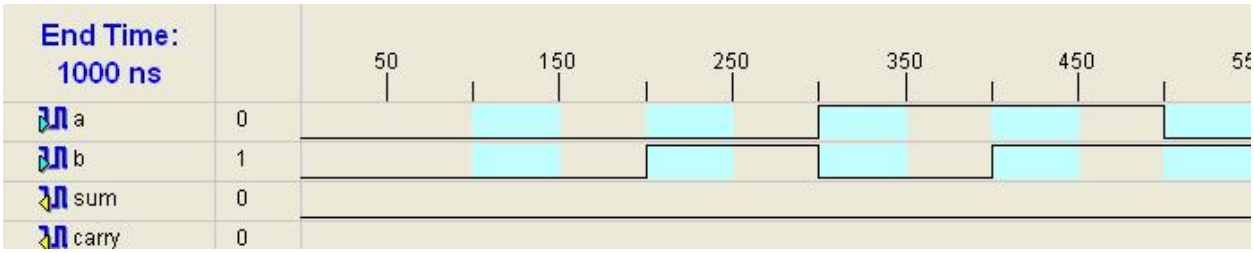
OUTPUT:



HALF ADDER: GATE LEVEL MODELLING

```
module half(sum, carry a, b);
    output sum;
    output carry;
    input a;
    input b;
     xor g1(sum,a,b);
     and g2(carry,a,b);
endmodule
```

INPUT

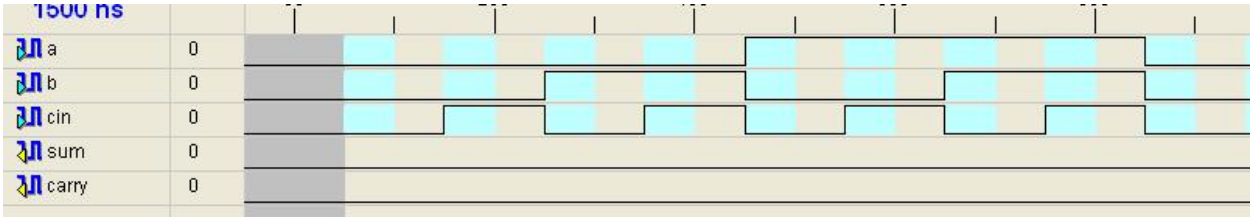


OUTPUT

FULL ADDER: GATE LEVEL MODELLING

```
module ful(a, b, cin, sum, carry);
      output sum;
      output carry;
      input a;
     input b;
      input cin;
      wire w1,w2,w3;
      xor g1(w1,a,b);
      and g2(w2,a,b);
      xor g3(sum,w1,cin);
      and g4(w3,w1,cin);
      or g5(carry,w2,w3);
endmodule
```

INPUT: FULL ADDER



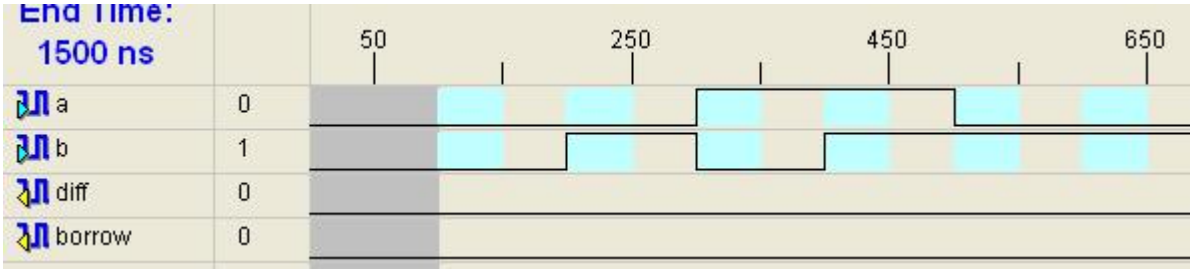
OUTPUT:

HALF SUBTRACTOR: GATE LEVEL MODELLING

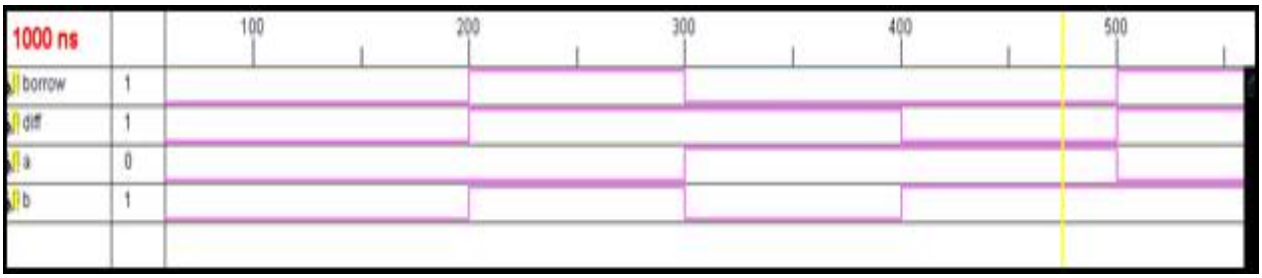
```
module sub(diff, borrow a, b);
    output diff;
    output borrow;
     input a;
     input b;
```

```
        wire w1;
    xor g1(diff,a,b);
    not g2(w1,a);
    and g3(borrow,w1,b);
endmodule
```

INPUT: HALF SUBTRACTOR



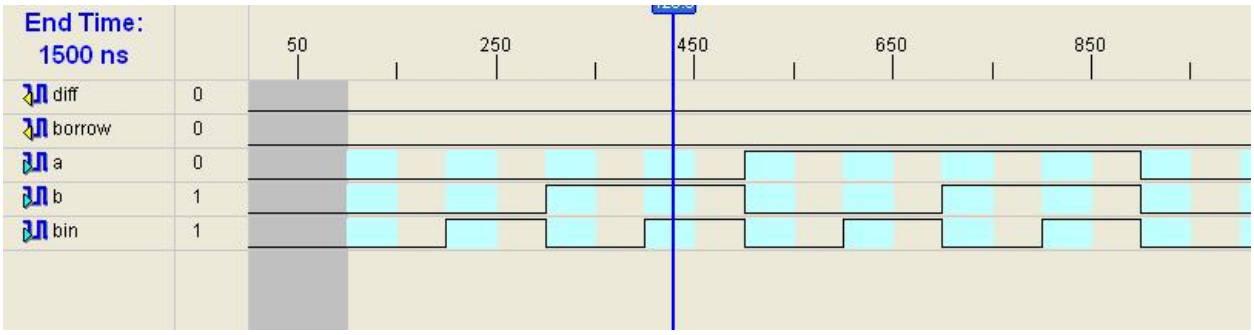
OUTPUT:



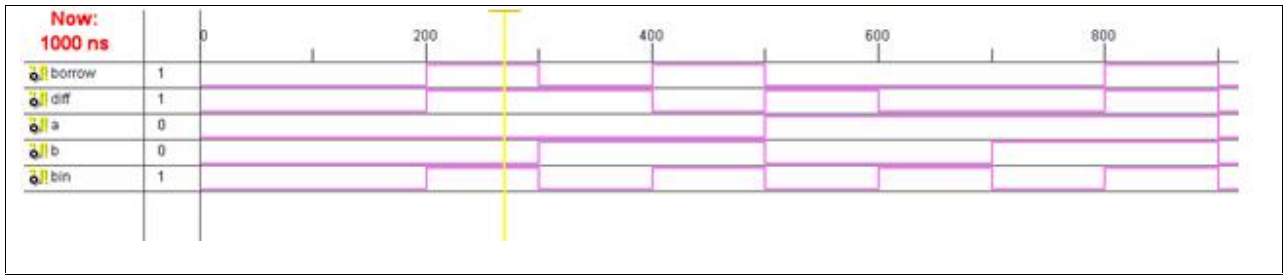
FULL SUBTRACTOR: GATE LEVEL MODELLING:

```
module sub(diff, borrow, a, b, bin);
    output diff;
    output borrow;
    input a;
    input b;
    input bin;
      wire a,f,g,h,i;
      xor g1(e,a,b);
      xor g2(diff,e,bin);
      and g3(h,f,bin);
      and g4(i,g,bin);
      or g5(borrow,h,i);
      not g6(f,e);
      not g7(g,a);
endmodule
```

INPUT:



OUTPUT:

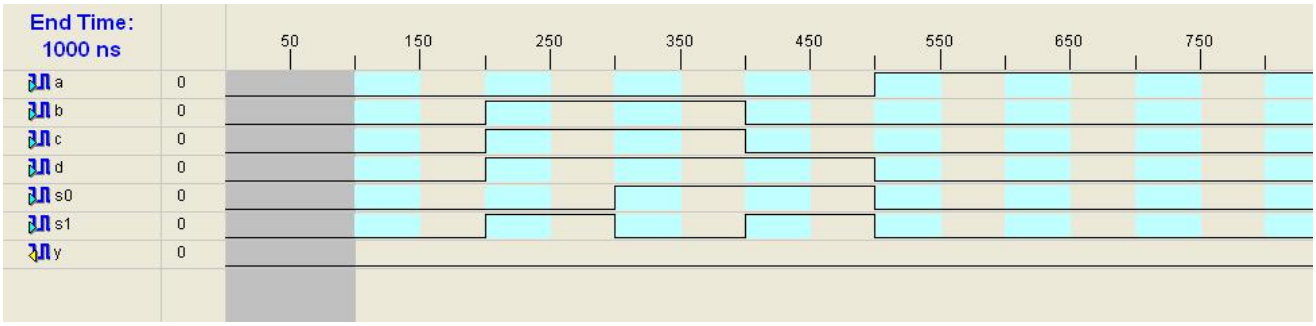


4:1 MULTIPLEXER: GATE LEVEL MODELLING

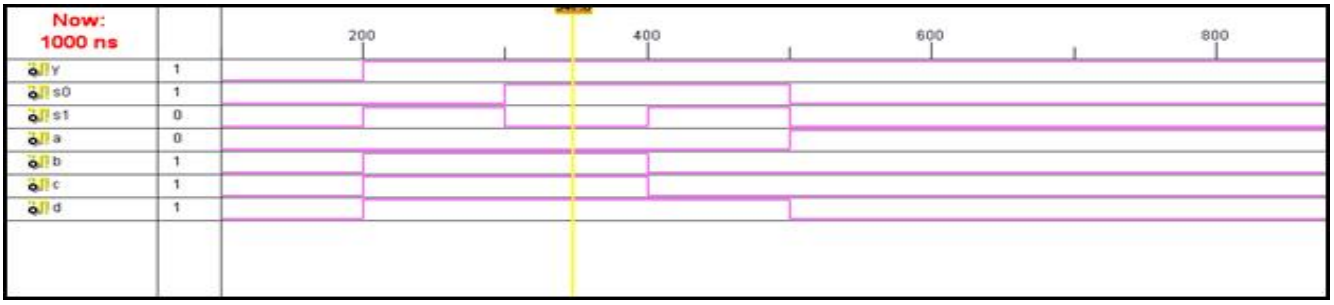
```
module multiplexer(y, a, b, c, d, s0, s1 );
    output y;
    input a;
    input b;
    input c;
    input d;
    input s0;
    input s1;
      wire w1,w2,w3,w4,w5,w6;
      not g1(w1,s0);
      not g2(w2,s1);
      and g3(w3,w1,w2,a);
      and g4(w4,w1,s1,b);
```

```
        and g5(w5,w2,s0,c);
        and g6(w6,s0,s1,d);
        or g7(y,w3,w4,w5,w6);
endmodule
```

INPUT: 4:1 MULTIPLEXER



OUTPUT

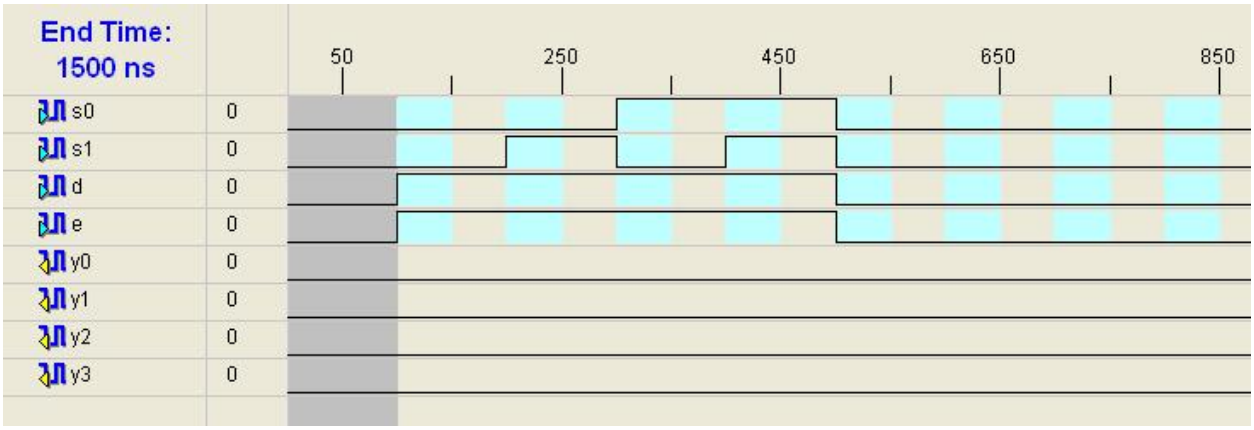


4:1 DEMULTIPLEXER: GATE LEVEL MODELLING

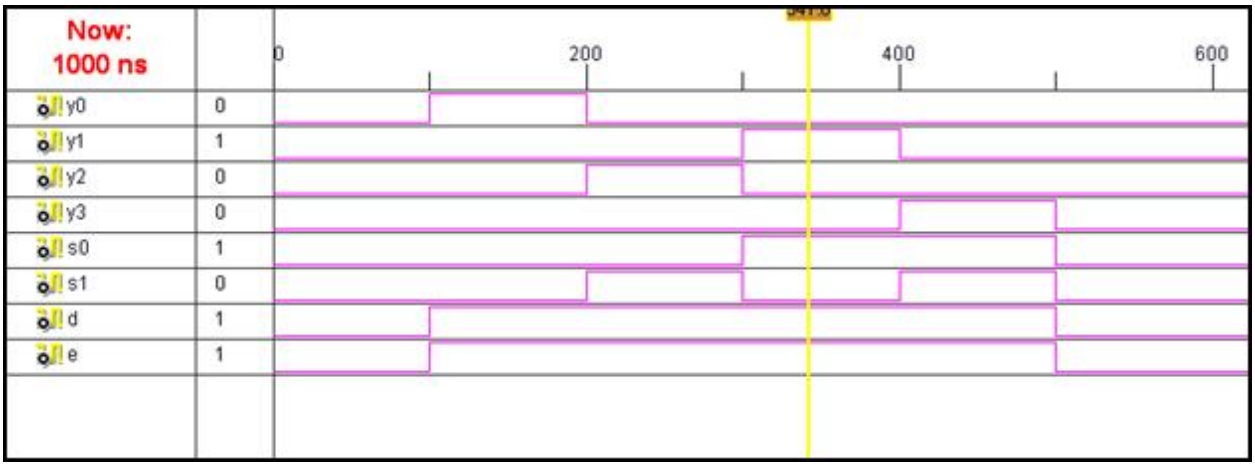
```
module demux(y0, y1, y2, y3, s0, s1, d, e);
     output y0;
    output y1;
    output y2;
    output y3;
    input s0;
    input s1;
    input d;
    input e;
   wire  w1,w2;
not  n1(w1,s1);
not n2(w2,s0);
and  a1(y0,e,d,w1,w2);
```

```
and a2(y1,e,d,s0,w1);
and a3(y2,e,d,w2,s1);
and  a4(y3,e,d,s0,s1);
endmodule
```

INPUT 4:1 DEMULTIPLEXER



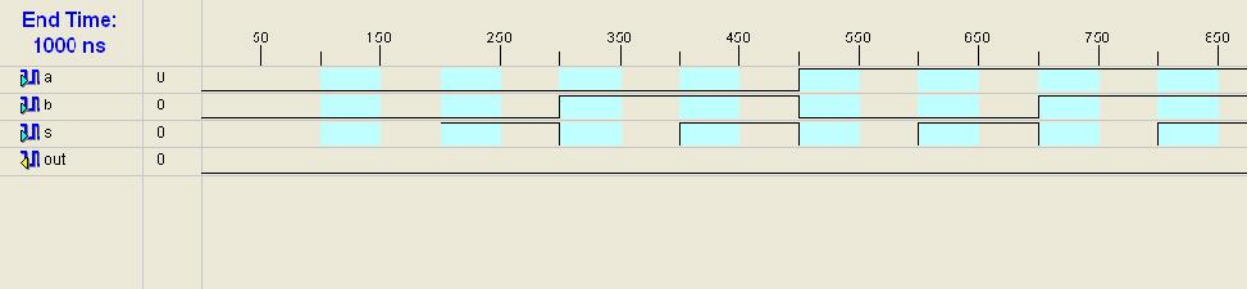
OUTPUT



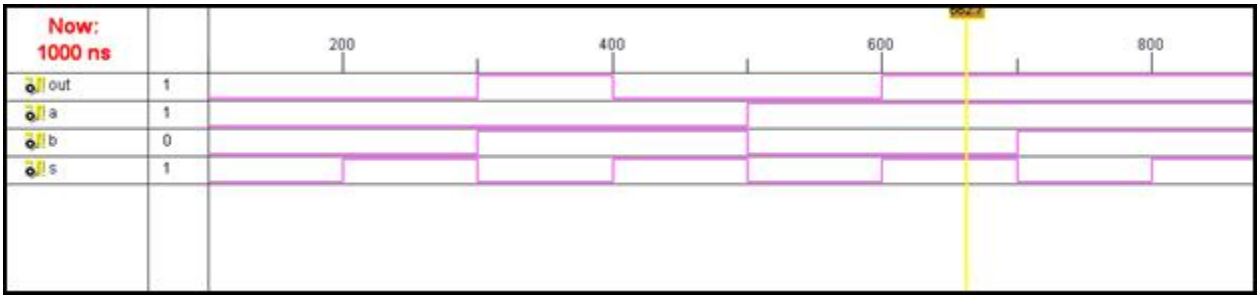
2:1 MULTIPLEXER: BEHAVIORAL MODELLING

```
module mux(out a, b, s);
    output out;
    input a;
    input b;
    input s;
              reg out;
      always @(s or a or b)
      if(s==1)out=a;
      else out=b;
endmodule
```

INPUT: 2:1 MULTIPLEXER



OUTPUT

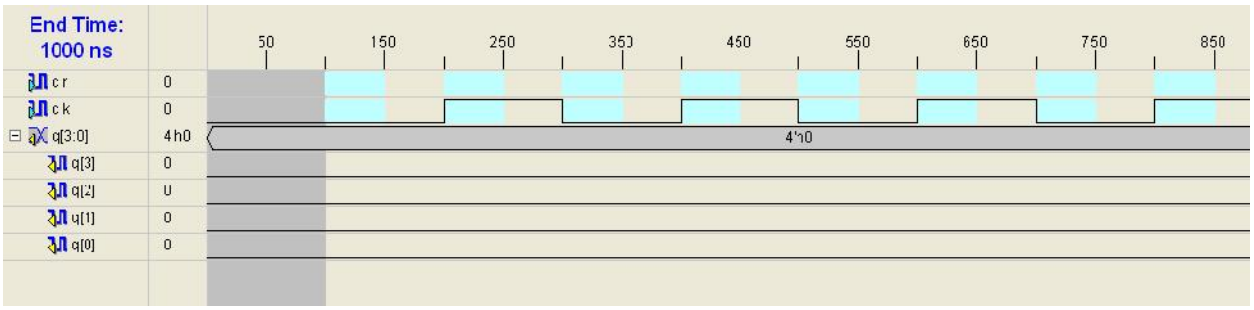


COUNTER: BEHAVIORAL MODELLING

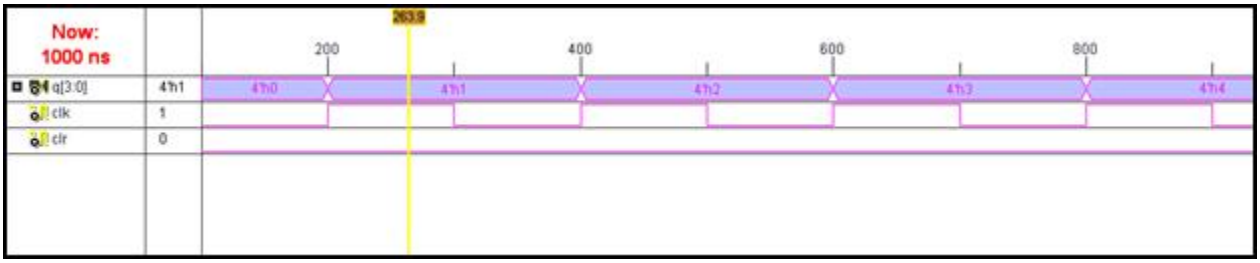
```
module count(clr, clk, q);
  output [3:0] q;
    input clr;
    input clk;
      reg [3:0]q;
      reg [3:0]z=4'b0000;
      always@(clk)
      begin
      if(clk==1'b1)
      if(clr==1'b1)
      z=4'b0001;
      else
      z=z+4'b0001;
      q=z;
      end
endmodule
```

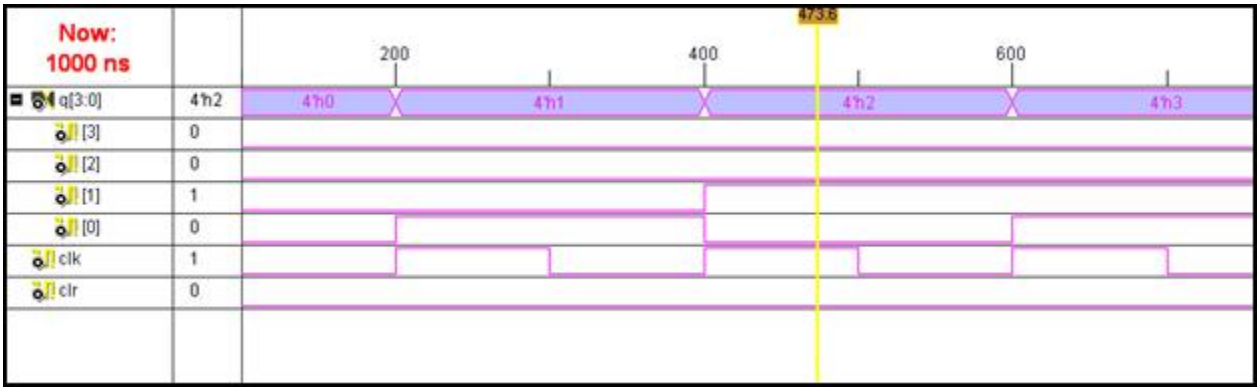
COUNTER: BEHAVIORAL MODELLING

INPUT



OUTPUT





PROCEDURE:

(i) The program is written the ModelSim based on the given design.

(ii) Compile the program.

(iii) Simulate the program.

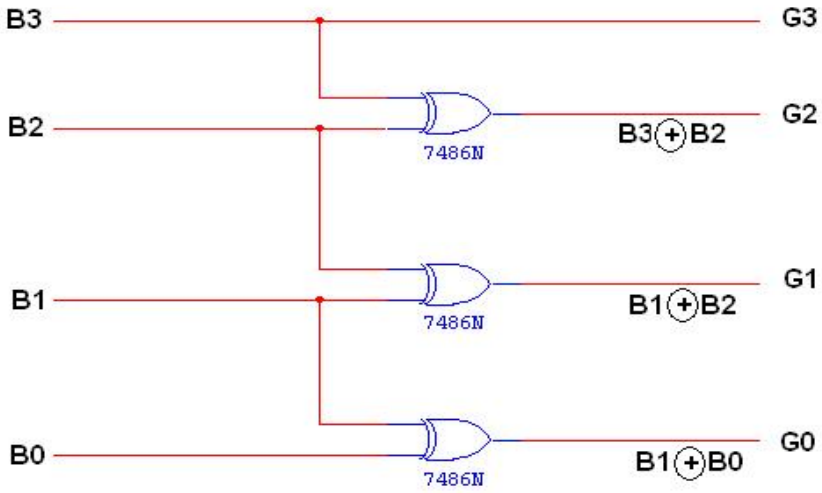
(iv) Verify the output in the waveform window.

RESULT: -

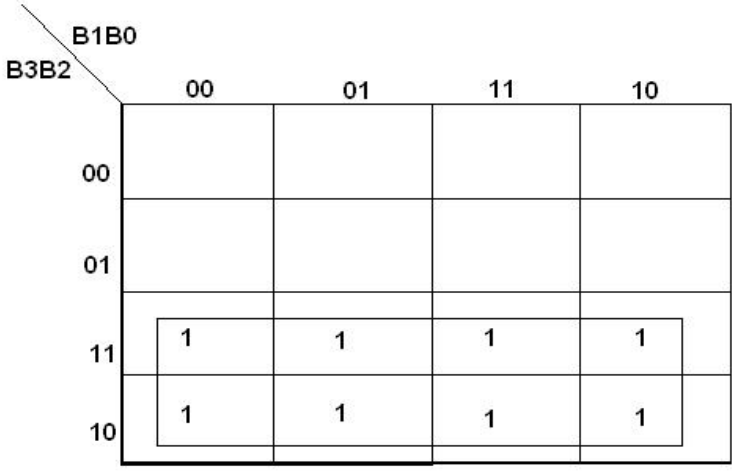
The shift registers have been simulated and their truth tables have been verified.

LOGIC DIAGRAM:

BINARY TO GRAY CODE CONVERTOR



K-Map for G_3:



$G_3 = B_3$

EX NO:18 DATE:	DESIGN AND IMPLEMENTATION OF DIGITAL SYSTEM (Mini Project)

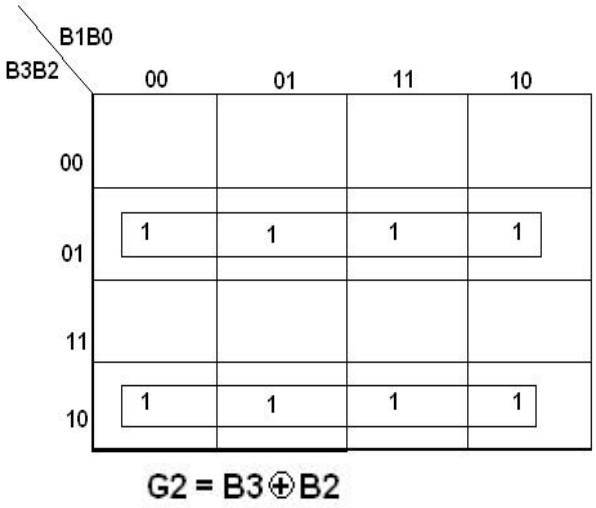
AIM:

To design and Implement a digital System of Binary Code Converter

APPARATUS REQUIRED:

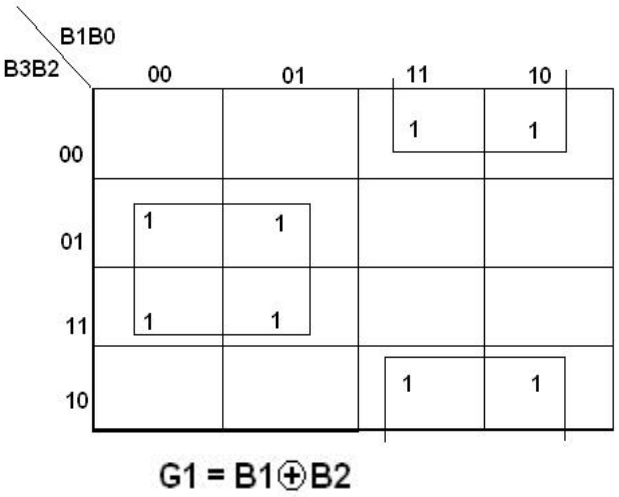
Sl.No.	COMPONENT	SPECIFICATION	QTY.
1.	X-OR GATE	IC 7486	1
2.	AND GATE	IC 7408	1
3.	OR GATE	IC 7432	1
4.	NOT GATE	IC 7404	1
5.	IC TRAINER KIT	-	1
6.	PATCH CORDS	-	35

K-Map for G_2:



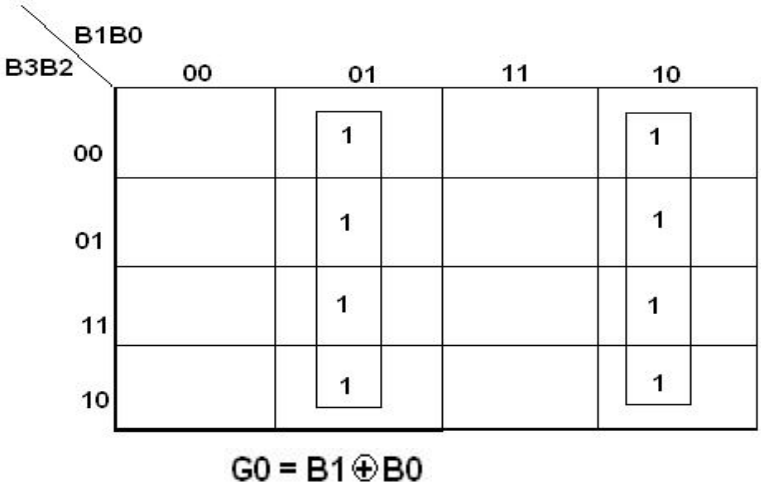
G2 = B3⊕B2

K-Map for G_1:



G1 = B1⊕B2

K-Map for G_0:



G0 = B1⊕B0

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.

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

RESULT:

Thus the digital system of Binary to Gray code converter has been designed and implemented.