SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES
SAVEETHA SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

LABORATORY MANUAL

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

SUB CODE: ECA 47

SUB NAME: PRINCIPLES OF DIGITAL SYSTEM DEIGN
LAB MANUAL

CONTENTS

ECE DEPARTMENT VISION-MISSION

Vision of the Department

The vision of ECE department is to become pioneer in higher learning and research and to produce creative solution to societal needs.

Mission of the Department

- To provide excellence in education, research and public service.
- To provide quality education and to make the students entrepreneur and employable.
- Continuous upgradation of techniques for reaching heights of excellence in a global.

PO-PSO

COs	PSOs			
	PSO1	PSO2	PSO3	PSO4
ECA 47.1	3	1	1	
ECA 47.2	3	2	1	
ECA 47.3	3	2	1	
ECA 47.4	3	2	1	1
ECA 47.5	3	2	1	1
ECA 47.6	3	2	1	1

COURSE OUTCOMES

ECA 47.1	Understand the fundamental concepts and techniques used in digital electronics.
ECA 47.2	Build Combinational Circuits that perform arithmetic operations & Code conversions.
ECA 47.3	Design and implementation of shift-registers.
ECA 47.4	Design and implementation of Sequential circuits.
ECA 47.5	Design and implementation of Asynchronous Sequential circuits
ECA 47.6	Implement Digital Logic circuits using VHDL and functions using logic gates.

Do’s

- Be punctual.
- Maintain discipline & silence.
- Keep the Laboratory clean and tidy.
- Enter Laboratory with shoes.
- Handle instruments with utmost care.
- Come prepared with circuit diagrams, writing materials and calculator.
- Follow the procedure that has been instructed.
- Return all the issued equipment properly.
- Get the signature on experiment result sheet daily.
- For any clarification contact faculty/staff in charge only.
- Shut down the power supply after the experiment.

Don’ts

- Avoid unnecessary chat or walk.
- Playing mischief in the laboratory is forbidden.
- Disfiguring of furniture is prohibited.
- Do not start the experiment without instructions.
- Avoid using cell phones unless absolutely necessary.
- Avoid late submission of laboratory reports.

S. NO	LAB EXPERIMENTS
1	Performance verification of basic logic gates using their corresponding truth tables with a digital trainer kit. • AND • OR, • NOT • XOR
2	Verification of Boolean theorems using logic gates and confirm their outputs with truth tables with a digital trainer kit.. • Associative, • Commutative • Absorption
3	Verification of De Morgan's law using basic logic gates and check its correctness by comparing the results with the truth table with a digital trainer kit..
4	Design of Code Converters and validate the operation by verifying the truth table with a digital trainer kit.. • Binary to Gray • Binary to Excess 3
5	Design adders and subtractors using logic gates, and test their functionality against the truth table with a digital trainer kit.. • Half Adder • Full Adder • Half subtractor • Full Subtractor
6	Implementation of 2-to-1 multiplexer and a 1-to-2 demultiplexer using logic gates, and verify their truth tables with a digital trainer kit..
7	Design and implement a 1-to-2 decoder and a 2-to-1 encoder and test their functionality against the corresponding truth tables with a digital trainer kit..
8	Design and verify flip-flop circuits using logic gates and verify the correct operation by testing its truth table with a digital trainer kit.. • SR flip flop • D flip flop
9	Design and Implement 2 Bit Up Counter using the following flip flops and verify with its truth table. Demonstrate with the help of Digital trainer kit. • JK FF • D FF

10	Implement the following shift registers using D flip flop and verify the same with the truth table. Demonstrate with the help of Digital trainer kit. • Serial In Serial Out • Serial In Parallel Out
11	Design adders and subtractors using logic gates, and test their functionality against the truth table with Verilog in the following 3 models (i) Structural Modelling Style, (ii) Behavioural Modelling Style,(iii) Data Flow Modelling Style. • Half Adder • Full Adder • Half subtractor • Full Subtractor
12	Design and simulate the truth table of a 4-to-1 multiplexer and 1-to-4 demultiplexer using ModelSim software and validate the results in the following 3 models (i) Structural Modelling Style, (ii) Behavioural Modelling Style,(iii) Data Flow Modelling Style
13	Design and simulate the truth table of a 2-to-4 decoder and a 4-to-2 encoder using ModelSim software and confirm the outputs in the following 3 models (i) Structural Modelling Style, (ii) Behavioural Modelling Style,(iii) Data Flow Modelling Style
14	Design and simulate the truth table of flip flop based Mod-5 counter using ModelSim software

INDEX

Ex.No.	Date	Title	Marks	Staff Sign.
1		Performance verification of basic logic gates using their corresponding truth tables with a digital trainer kit. • AND • OR, • NOT • XOR		
2		Verification of Boolean theorems using logic gates and confirm their outputs with truth tables with a digital trainer kit.. • Associative, • Commutative • Absorption		
3		Verification of De Morgan's law using basic logic gates and check its correctness by comparing the results with the truth table with a digital trainer kit..		
4		Design of Code Converters and validate the operation by verifying the truth table with a digital trainer kit.. • Binary to Gray • Binary to Excess 3		
5		Design adders and subtractors using logic gates, and test their functionality against the truth table with a digital trainer kit.. • Half Adder • Full Adder • Half subtractor • Full Subtractor		
6		Implementation of 2-to-1 multiplexer and a 1-to-2 demultiplexer using logic gates, and verify their truth tables with a digital trainer kit..		
7		Design and implement a 1-to-2 decoder and a 2-to-1 encoder and test their functionality against the corresponding truth tables with a digital trainer kit..		
8		Design and verify flip-flop circuits using logic gates and verify the correct operation by testing its truth table with a digital trainer kit..		

		• SR flip flop • D flip flop		
9		Design and Implement 2 Bit Up Counter using the following flip flops and verify with its truth table. Demonstrate with the help of Digital trainer kit. • JK FF • D FF		
10		Implement the following shift registers using D flip flop and verify the same with the truth table. Demonstrate with the help of Digital trainer kit. • Serial In Serial Out • Serial In Parallel Out		
CODING – VERILOG & VHDL				
11		Design adders and subtractors using logic gates, and test their functionality against the truth table with Verilog in the following 3 models (i) Structural Modelling Style, (ii) Behavioural Modelling Style,(iii) Data Flow Modelling Style. • Half Adder • Full Adder • Half subtractor • Full Subtractor		
12		Design and simulate the truth table of a 4-to-1 multiplexer and 1-to-4 demultiplexer using ModelSim software and validate the results in the following 3 models (i) Structural Modelling Style, (ii) Behavioural Modelling Style,(iii) Data Flow Modelling Style		
13		Design and simulate the truth table of a 2-to-4 decoder and a 4-to-2 encoder using ModelSim software and confirm the outputs in the following 3 models (i) Structural Modelling Style, (ii) Behavioural Modelling Style,(iii) Data Flow Modelling Style		
14		Design and simulate the truth table of flip flop based Mod-5 counter using ModelSim software		

Ex.No.-1 STUDY OF LOGIC GATES

AIM:

To study about logic gates and verify their truth tables.

APPARATUS REQUIRED:

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

THEORY:

Circuit that takes the logical decision and the process are called logic gates. Each gate has one or more input and only one output.
OR, AND and NOT are basic gates. NAND, NOR and X-OR are known as universal gates. Basic gates form these gates.

AND GATE:

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

OR GATE:

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

NOT GATE:

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

AND GATE:

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

NOR GATE:

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

X- OR GATE:

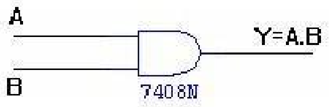
The output is high when any one of the inputs is high. The output is lowwhen both the inputs are low and both the inputs are high.

PROCEDURE:

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

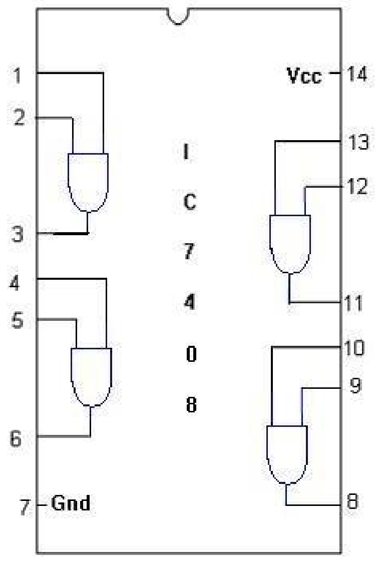
AND GATE

SYMBOL **PIN DIAGRAM**



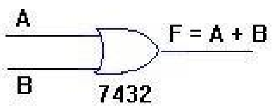
TRUTH TABLE

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

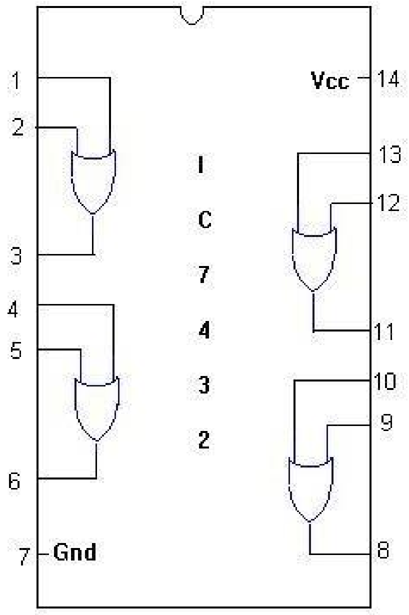


OR GATE

SYMBOL :



PIN DIAGRAM :

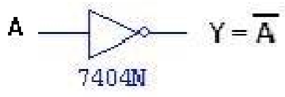


TRUTH TABLE

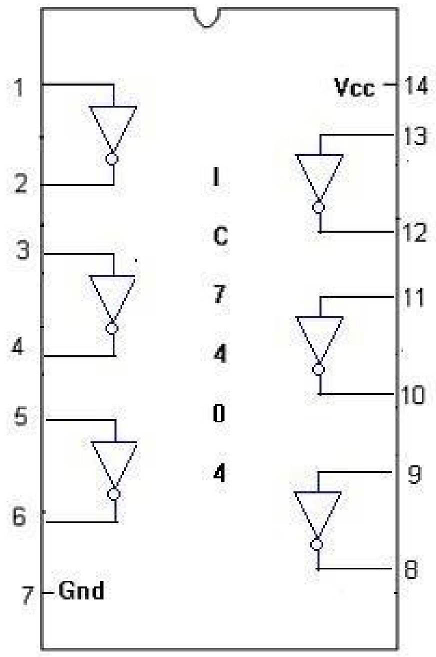
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

NOT GATE

SYMBOL



PIN DIAGRAM

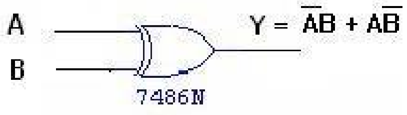


TRUTH TABLE :

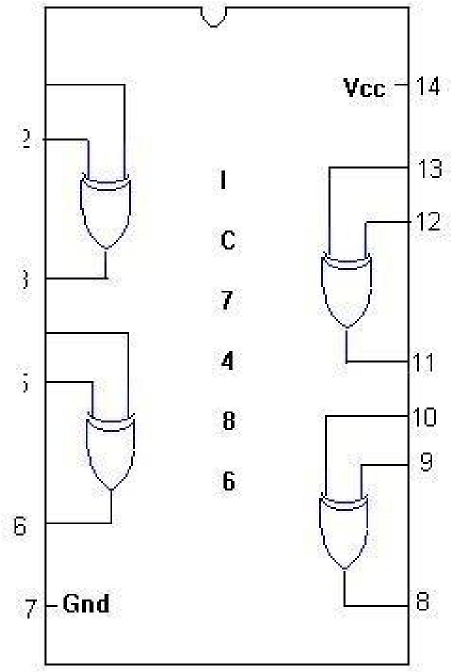
A	$\overline{A}$
0	1
1	0

EX-OR GATE

SYMBOL



PIN DIAGRAM

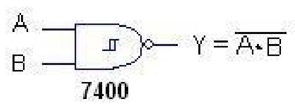


TRUTH TABLE :

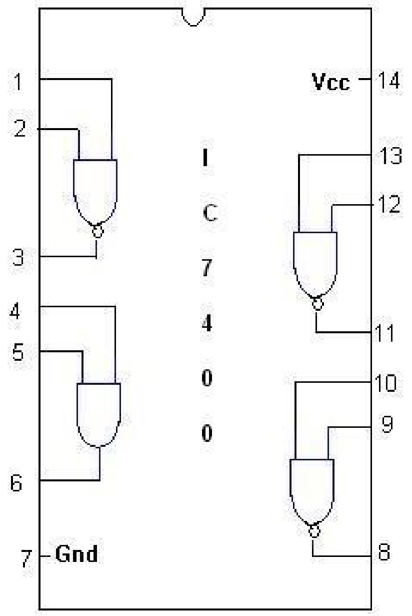
A	B	$\overline{A}B + A\overline{B}$
0	0	0
0	1	1
1	0	1
1	1	0

2-INPUT NAND GATE

SYMBOL



PIN DIAGRAM

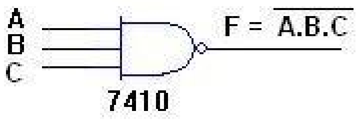


TRUTH TABLE

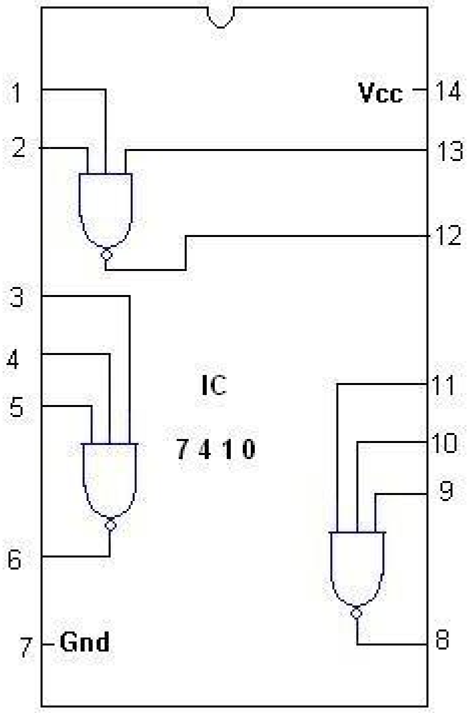
A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

3-INPUT NAND GATE

SYMBOL :



PIN DIAGRAM :



TRUTH TABLE

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

Ex.no 2 VERIFICATION OF BOOLEAN THEOREMS USING DIGITAL LOGIC GATES

AIM:

To verify the Boolean Theorems using logic gates.

APPARATUS REQUIRED:

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

THEORY:

BASIC BOOLEAN LAWS

1. Commutative Law

The binary operator OR, AND is said to be commutative if,

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

2. Associative Law

The binary operator OR, AND is said to be associative if,

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

3. Absorption Law

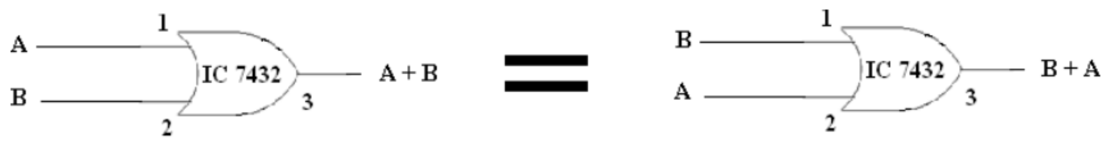
1. A+AB = A
2. A+AB = A

COMMUTATIVE

TRUTH TABLE

I/P		LHS (O/P)	RHS (O/P)
A	B	A+B	B+A
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

LOGIC DIAGRAM

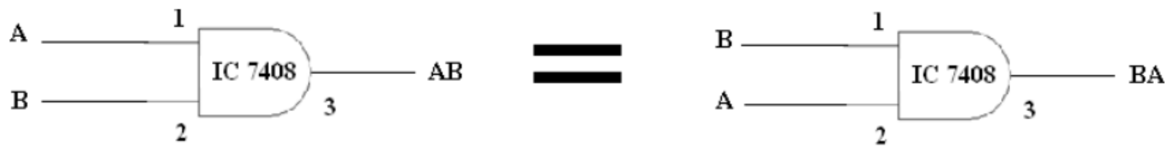


Commulative law A + B = B + A

TRUTH TABLE

I/P		LHS (O/P)	RHS (O/P)
A	B	A. B	B. A
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

LOGIC DIAGRAM



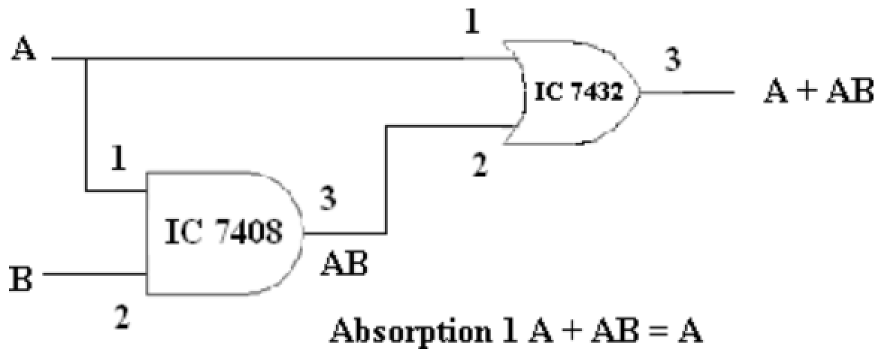
Commulative law AB = BA

ABSORPTION

TRUTH TABLE

I/P			O/P
A	B	AB	A + AB
0	0	0	**0**
0	1	0	**0**
1	0	0	**1**
1	1	1	**1**

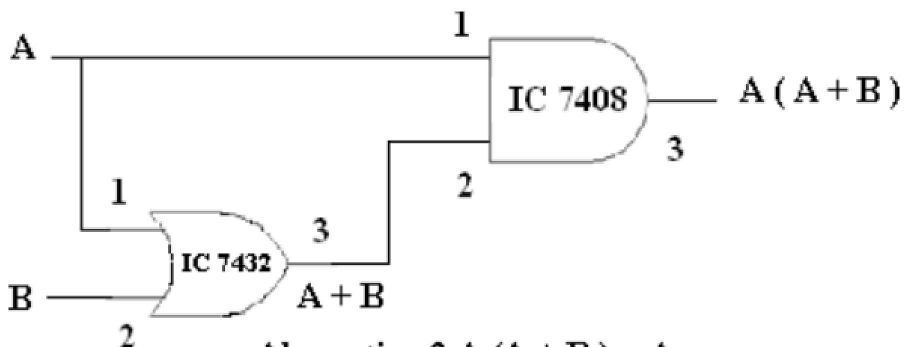
LOGIC DIAGRAM



TRUTH TABLE

I/P			O/P
A	B	A+B	**A(A + B)**
0	0	0	**0**
0	1	1	**0**
1	0	1	**1**
1	1	1	**1**

LOGIC DIAGRAM

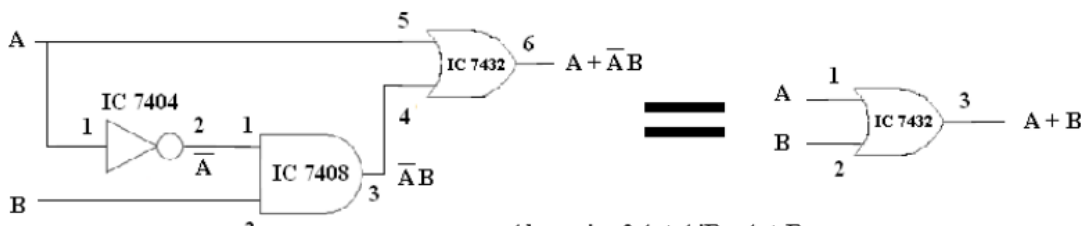


Absorption 2 A (A + B) = A

TRUTH TABLE

I/P				LHS (O/P)	RHS (O/P)
A	B	A'	A'B	A + (A'B)	A + B
0	0	1	0	0	0
0	1	1	1	1	1
1	0	0	0	1	1
1	1	0	0	1	1

LOGIC DIAGRAM

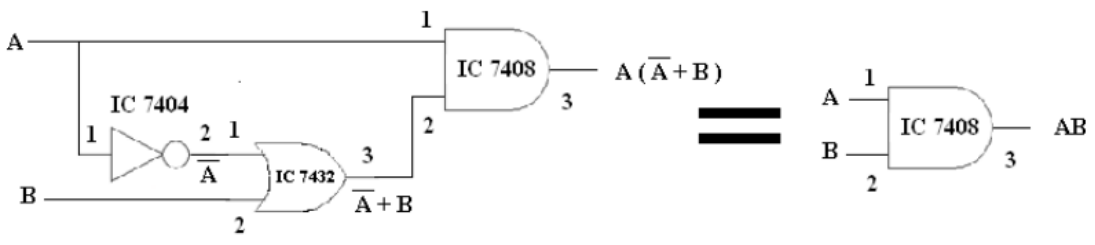


Absorption 3 A + A'B = A + B

TRUTH TABLE

I/P				LHS (O/P)	RHS (O/P)
A	B	A'	A' + B	A (A' + B)	AB
0	0	1	1	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	1	0	1	1	1

LOGIC DIAGRAM



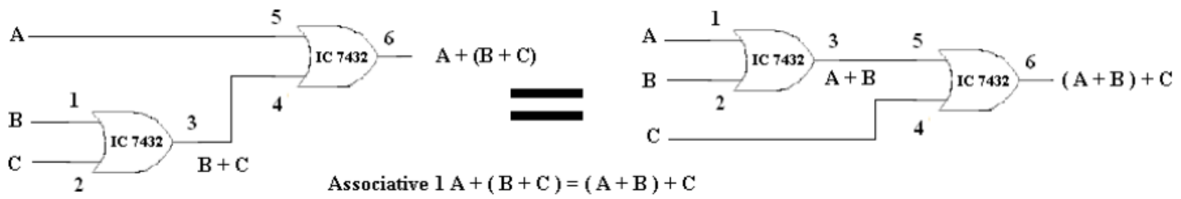
Absorption 4 A. (A'+ B) = AB

ASSOCIATIVE

TRUTH TABLE

I/P				LHS (O/P)		RHS (O/P)
A	B	C	B+C	A + (B+C)	A + B	(A + B) + C
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

LOGIC DIAGRAM

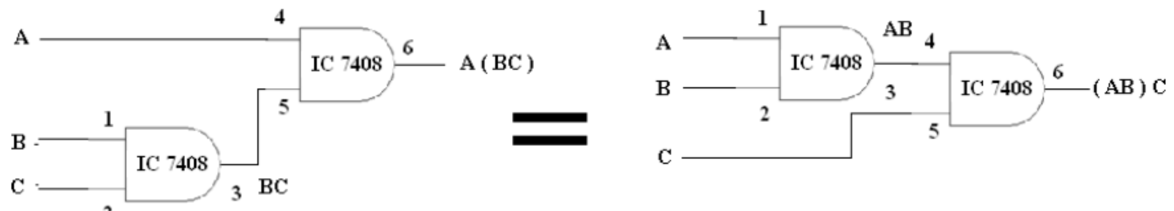


Associative 1 A + (B + C) = (A + B) + C

TRUTH TABLE

I/P				LHS (O/P)		RHS (O/P)
A	B	C	BC	A (BC)	AB	(AB) C
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	0	1	0
1	1	1	1	1	1	1

LOGIC DIAGRAM



Associative 2 A (B C) = (AB) C

Result:

Ex.no 3 DeMorgan's Law DATE :

AIM:

To verify Demorgan’s Law using 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	BREAD BOARD	-	1
5.	CONNECTING WIRES	-	10

THEORY:

Boolean algebra has postulates and identities. We can often use these laws to reduce expressions or put expressions in to a more desirable form. One of these laws is the De- Morgan's law

De-Morgan's law has two conditions, or conversely, there are two laws called De-Morgan's Laws.

First Condition or First law:

The compliment of the product of two variables is equal to the sum of the compliment of eachvariable. Thus according to De-Morgan's laws or De-Morgan's theorem if A and B are the two variablesor Boolean numbers. Then accordingly

(A.B)’ = A’ + B’

Second Condition or Second law:

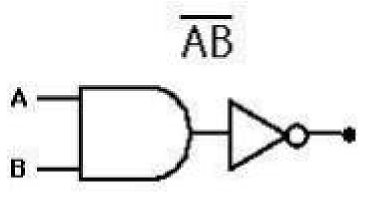
The compliment of the sum of two variables is equal to the product of the compliment of eachvariable.

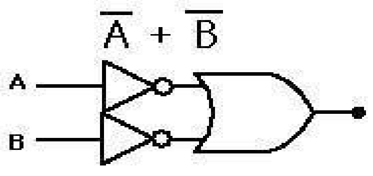
Thus according to De Morgan’s theorem if A and B are the two variables then.

(A+B)’ = A’ . B’

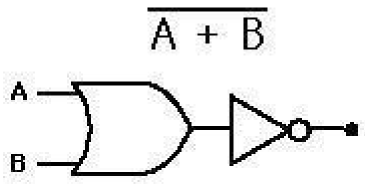
LOGIC DIAGRAM:

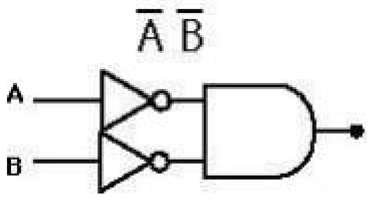
(A.B)’ = A’ + B’





(A+B)' = A' . B'





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)'**	**A' + B'**
0	0				
0	1				
1	0				
1	1				

RESULT:

The Demorgan's Theorem is verified using truth table.

Expt NO. 4 DESIGN AND IMPLEMENTATION OF CODE CONVERTER

DATE :

AIM:

To design and implement 4-bit

(i) Binary to gray code converter

(ii) Binary Excess-3 code converter

APPARATUS REQUIRED:

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

THEORY:

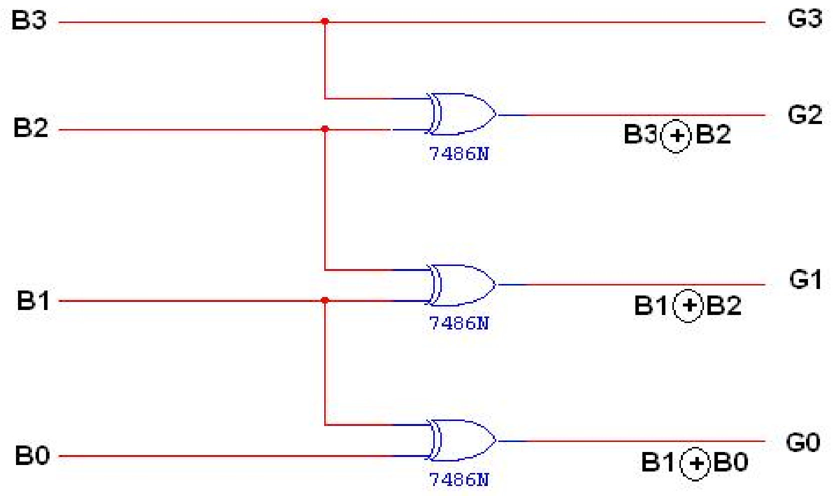
The availability of large variety of codes for the same discrete elements of information results in the use of different codes by different systems. A conversion circuit must be inserted between the two systems if each uses different codes for same information. Thus, code converter is a circuit that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

The bit combination assigned to binary code to gray code. Since each code uses four bits to represent a decimal digit. There are four inputs and four outputs. Gray code is a non-weighted code. The Boolean functions are obtained from K-Map for each output variable.

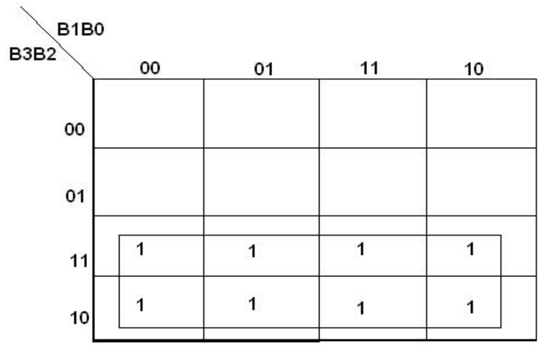
A code converter is a circuit that makes the two systems compatible even though each uses a different binary code. Each one of the four maps represents one of the four outputs of the circuit as a function of the four input variables.

LOGIC DIAGRAM:

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

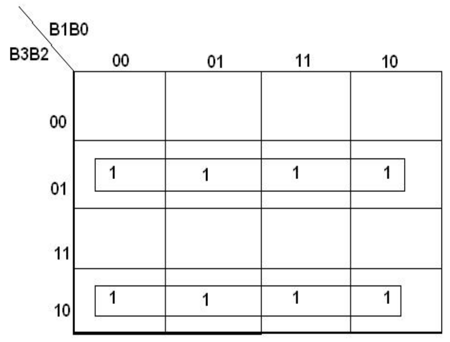


K-Map for G_3:



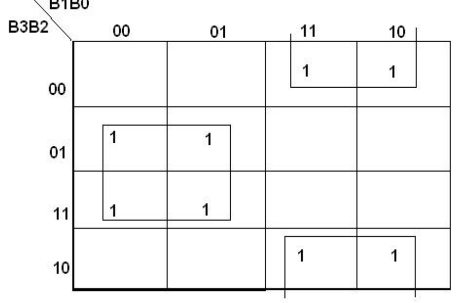
$G_3 = B_3$

K-Map for G_2:



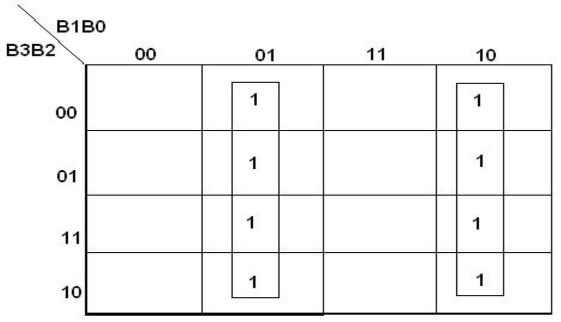
$G2=B3 \oplus B2$

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

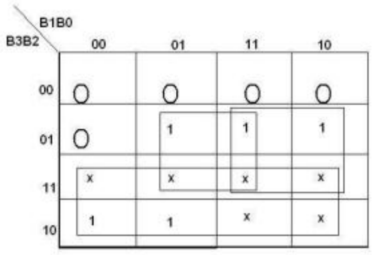
Binary to Excess3 Code

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

| **BCD input** | **Excess – 3 output** |

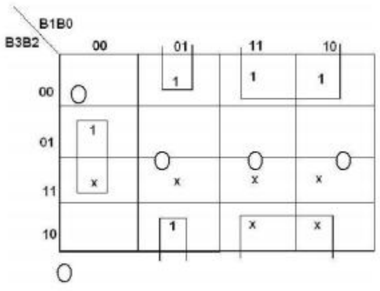
B3	B2	B1	B0	G3	G2	G1	G0
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	X

K-Map for E_3:



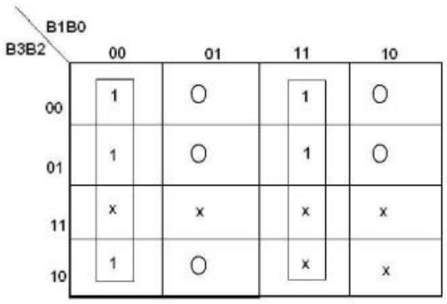
$E3 = B3 + B2\,(B0 + B1)$

K-Map for E_2:



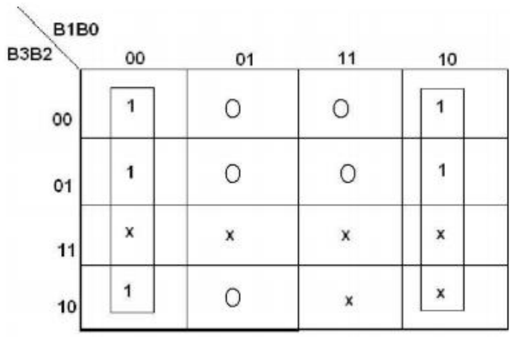
$E2 = B2 \oplus (B1 + B0)$

K-Map for E_1:

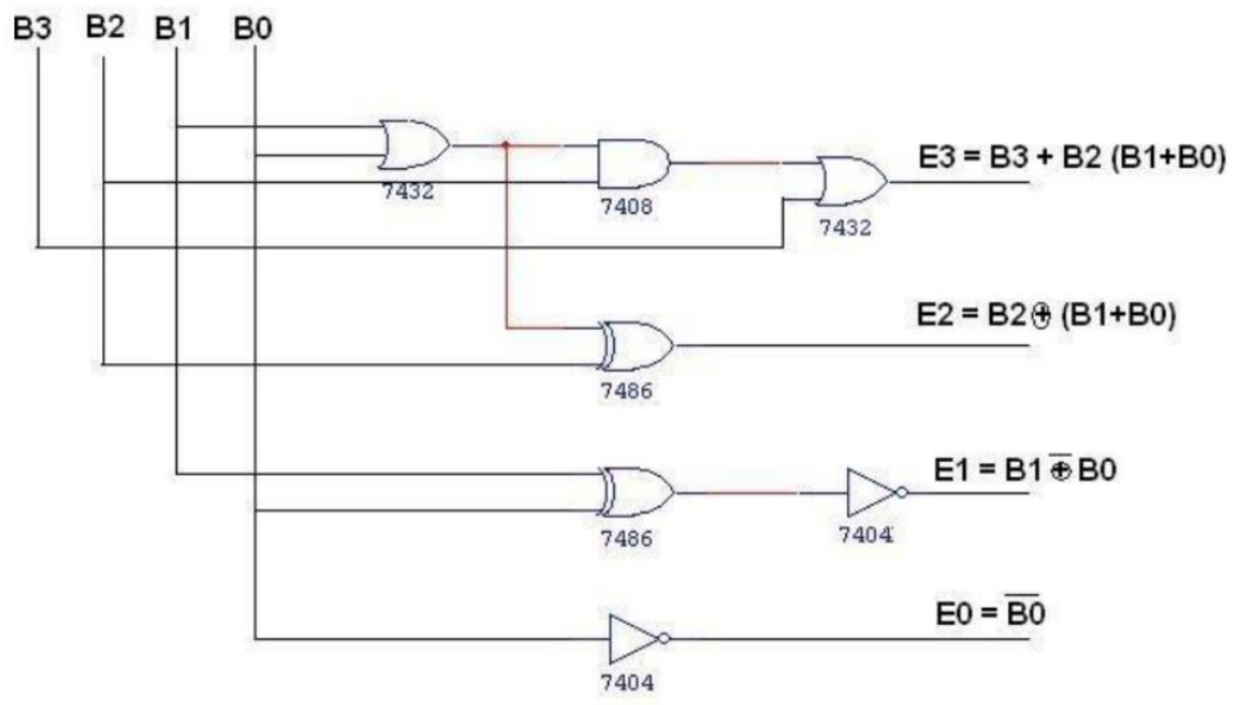


$E1 = B1 \overline{\oplus} B0$

K-Map for E_0:



$E0 = \overline{B0}$



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:

Ex.No.-5 ADDER AND SUBTRACTOR

AIM:

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

APPARATUS REQUIRED:

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

THEORY:

HALF ADDER:

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

FULL ADDER:

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

HALF SUBTRACTOR:

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

FULL SUBTRACTOR:

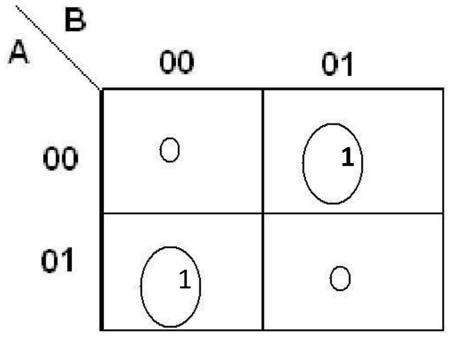
The full subtractor is a combination of X-OR, AND, OR, NOT Gates. In a full subtractor the logic circuit should have three inputs and two outputs. The two half subtractor put together gives a full subtractor .The first half subtractor will be C and A B. The output will be difference output of full subtractor.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

HALF ADDER

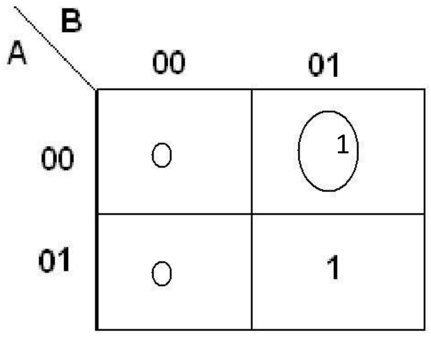
TRUTH TABLE:

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

K-Map for SUM:



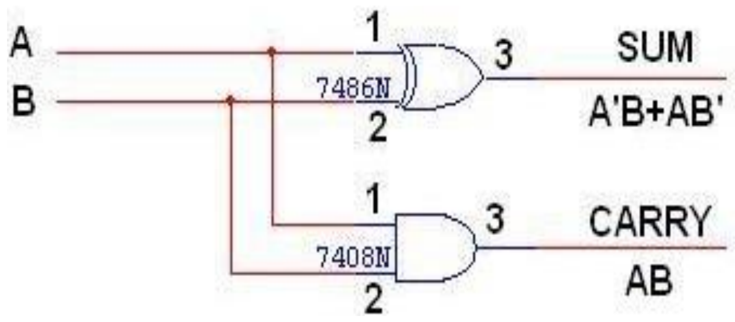
K-Map for CARRY:



SUM = A'B + AB'

CARRY = AB

LOGIC DIAGRAM:

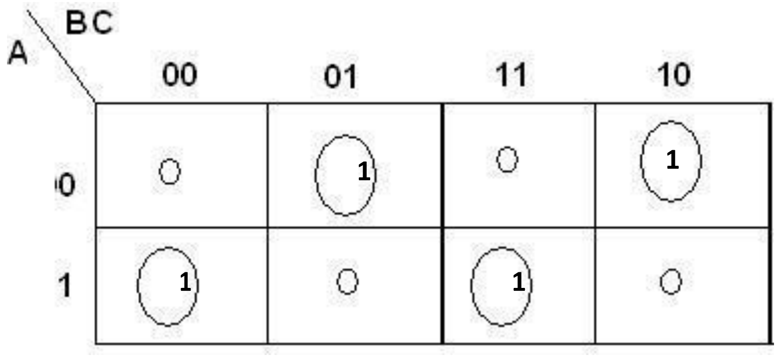


FULL ADDER

TRUTH TABLE:

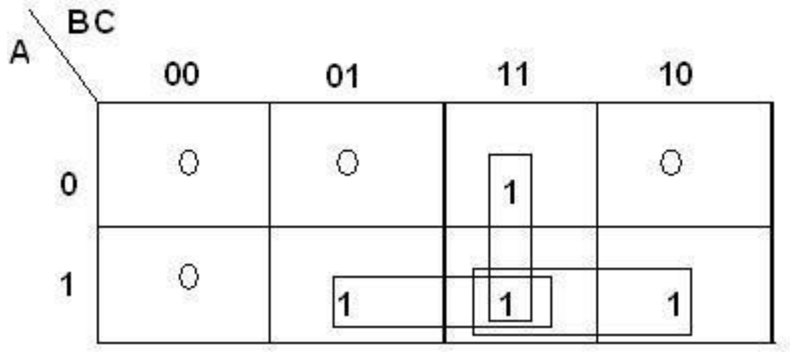
A	B	C	CARRY	SUM
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

K-Map for SUM



SUM = A’B’C + A’BC’ + ABC’ + ABC

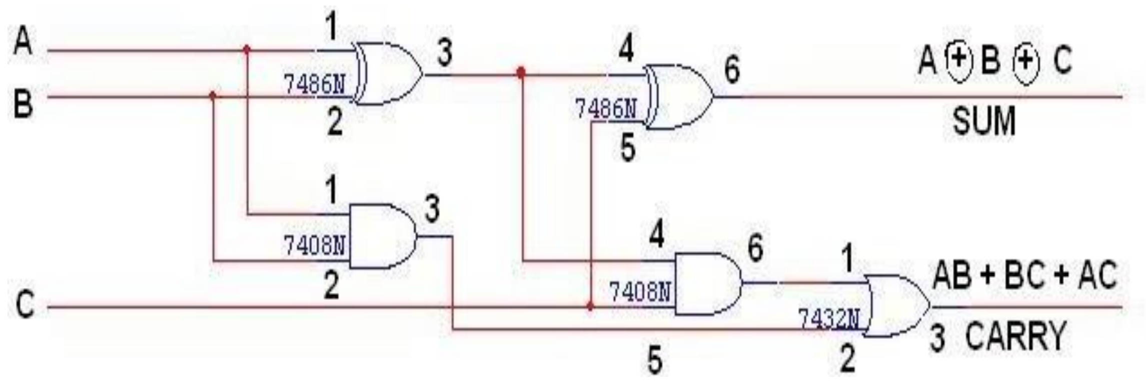
K-Map for CARRY



CARRY = AB + BC + AC

LOGIC DIAGRAM:

FULL ADDER USING TWO HALF ADDER

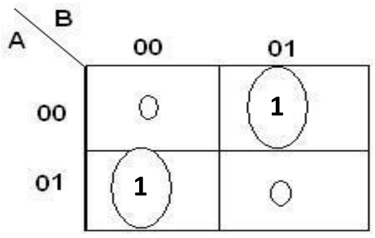


HALF SUBTRACTOR

TRUTH TABLE:

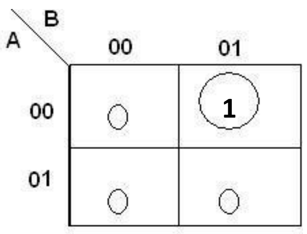
A	B	BORROW	DIFFERENCE
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

K-Map for DIFFERENCE



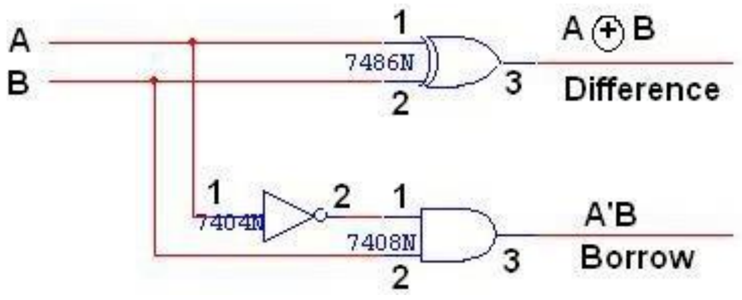
DIFFERENCE = A'B + AB'

K-Map for BORROW



BORROW = A'B

LOGIC DIAGRAM

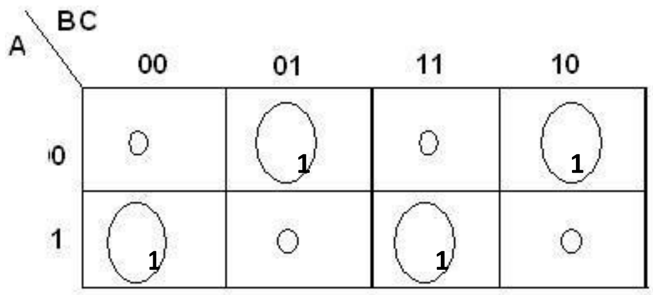


FULL SUBTRACTOR

TRUTH TABLE:

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

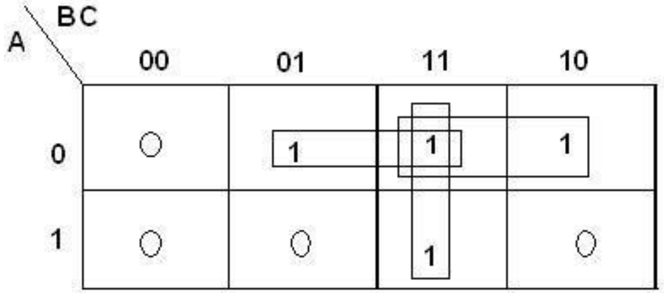
K-Map for Difference



$$\begin{array}{c|cccc} A \backslash BC & 00 & 01 & 11 & 10 \\ \hline 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{array}$$

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

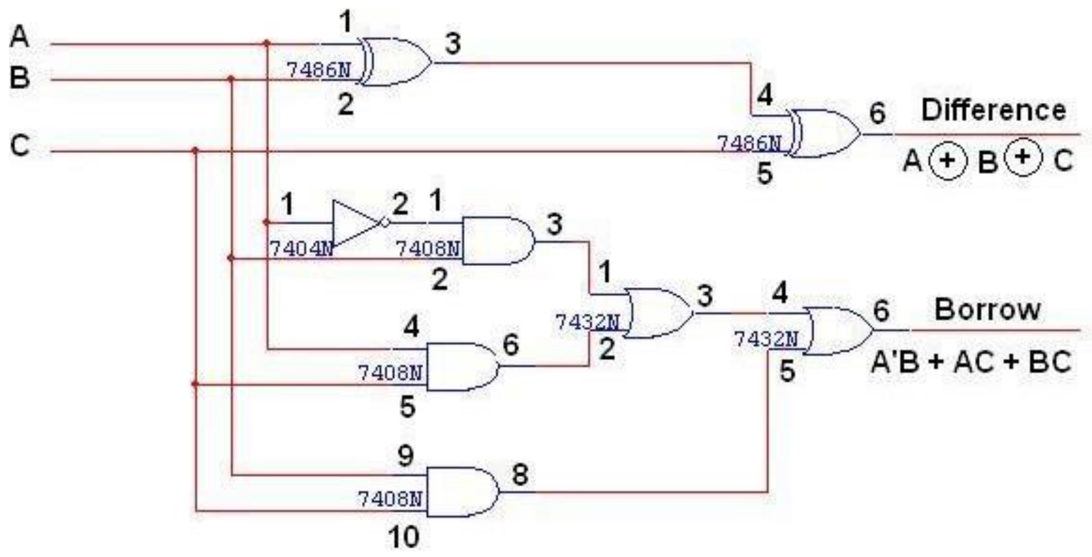
K-Map for Borrow



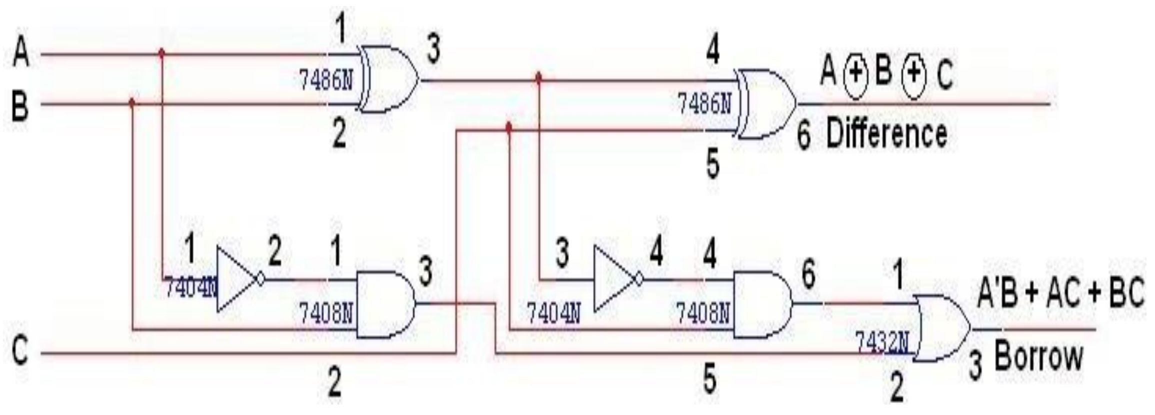
$$\begin{array}{c|cccc} A \backslash BC & 00 & 01 & 11 & 10 \\ \hline 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{array}$$

Borrow = A’B + BC + A’C

LOGIC DIAGRAM:



FULL SUBTRACTOR USING TWO HALF SUBTRACTOR



PROCEEDURE:

(i) Connections are given as per circuit diagram.

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

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

6. MULTIPLEXER AND DEMULTIPLEXER

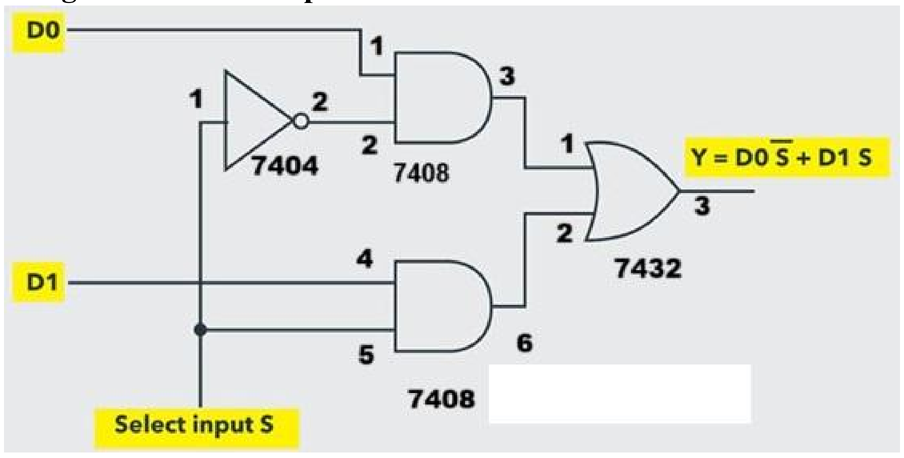
AIM:

To design and implement the multiplexer and demultiplexer using logic gates

APPARATUS REQUIRED:

SL.NO.	COMPONENT	SPECIFICATION	QTY.
1.	AND GATE	IC 7408	1
2.	OR GATE	IC 7432	1
3.	NOT GATE	IC 7404	1
2.	IC TRAINER KIT	-	1
3.	PATCH CORDS	-	32

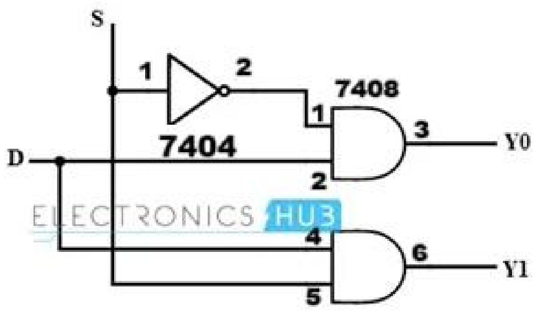
Logic Diagram of 2:1 Multiplexer



TRUTH TABLE:

Select Input S	Y = OUTPUT
0	D0
1	D1

Logic diagram of 1:4 demultiplexer



TRUTH TABLE:

Input S	**Outputs**	
	Y0	**Y1**
0	**D**	**0**
1	**0**	**d**

PROCEDURE:

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

RESULT:

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

7. IMPLEMENTATION OF DECODER AND ENCODER

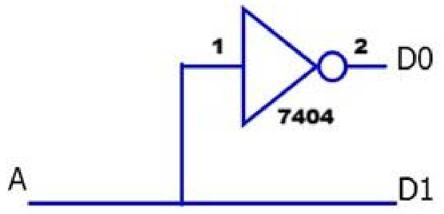
AIM:

To design and implement a 1-to-2 decoder and a 2-to-1 encoder using logic gates

APPARATUS REQUIRED:

SL.NO.	COMPONENT	SPECIFICATION	QTY.
1.	OR GATE	IC 7432	1
2.	NOT GATE	IC 7404	1
3.	IC TRAINER KIT	-	1
4.	PATCH CORDS	-	32

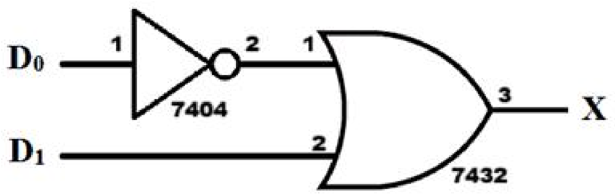
Logic diagram of 1:2 decoder



Truth table:

INPUT	OUTPUTS	
A	**D1**	**D0**
0	**0**	**1**
1	**1**	**0**

Logic diagram of 2 to 1 encoder



TRUTH TABLE

INPUTS		OUTPUT
D1	**D0**	**X**
0	**1**	**0**
1	**0**	**1**

PROCEDURE:

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

RESULT:

Thus the encoder and decoder using logic gates are designed and implemented

8. IMPLEMENTATION OF SR AND D FLIP FLOPS USING LOGIC GATES

AIM:

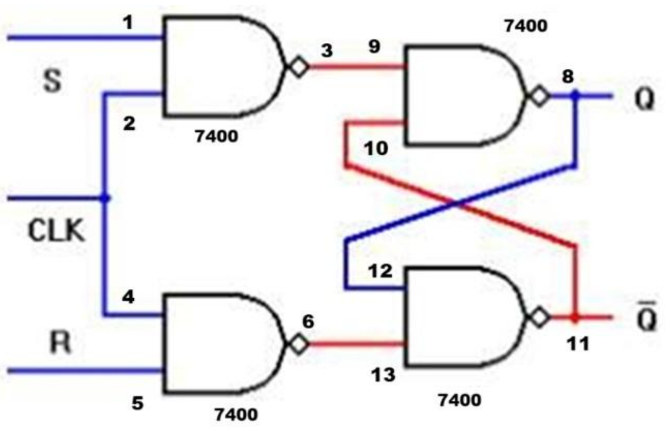
To design and implement SR and D Flip Flop using Logic gates and also verify their truth table.

APPARATUS REQUIRED:

S.No.	COMPONENT	SPECIFICATION	QTY
1.	NAND GATE	IC 7400	1
2.	IC TRAINER KIT	-	1
3.	CONNECTING WIRES	-	10

LOGIC DIAGRAM:

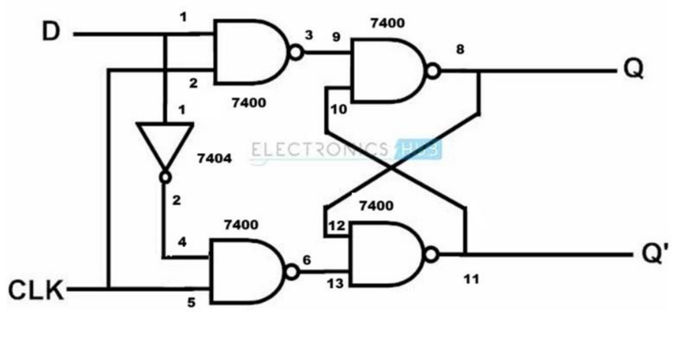
SR FLIPFLOP



Truth Table

Inputs			Output
R	S	CLK	Qt+1
0	0	↑	$Q_{t(\text{Previous State})}$
0	1	↑	1
1	0	↑	0
1	1	↑	Indeterminate state

D FLIPFLOP



Truth Table:

Inputs		Output
D	CLK	Q
0	↑	0
1	↑	1

PROCEDURE:

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

RESULT:

The SR and D flipflop was implemented and truth table is verified.

9. DESIGN OF 2 – BIT ASYNCHRONOUS UPCOUNTER

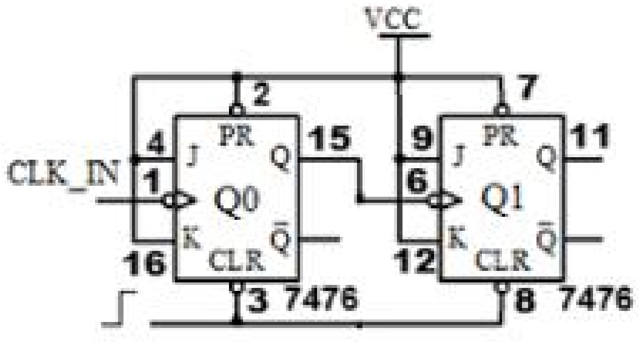
AIM:

To design and implement 2 Bit Up Counter using JK flipflops and verify its truth table.

APPARATUS REQUIRED:

COMPONENT	SPECIFICATION	QTY
JK FLIPFLOP	IC 7476	1
IC TRAINER KIT	-	1
CONNECTING WIRES	-	10

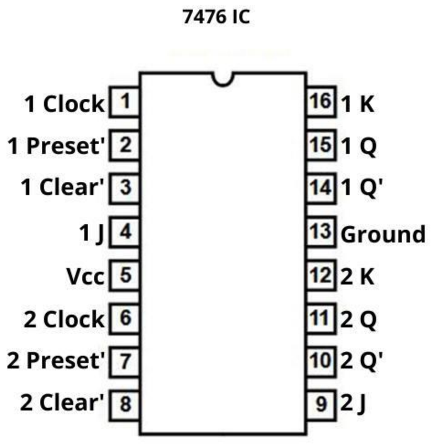
LOGIC DIAGRAM:



TRUTH TABLE:

CLK	Q_1	Q_0
1	0	0
2	0	1
3	1	0
4	1	1

Pin Diagram of 7476



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 2 Bit Up Counter using JK flipflops was implemented and truth table is verified.

10. SHIFT REGISTER

AIM:

To design and implement the following shift registers

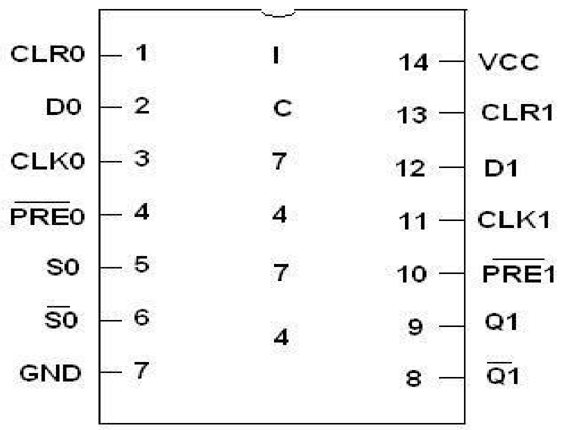
(i) Serial in serial out

(ii) Serial in parallel out

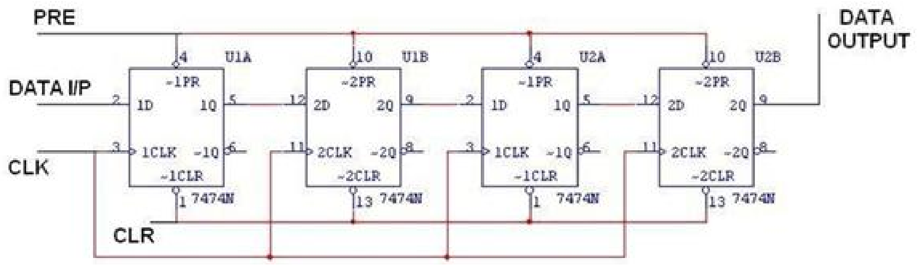
APPARATUS REQUIRED:

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

PIN DIAGRAM OF IC 7474:



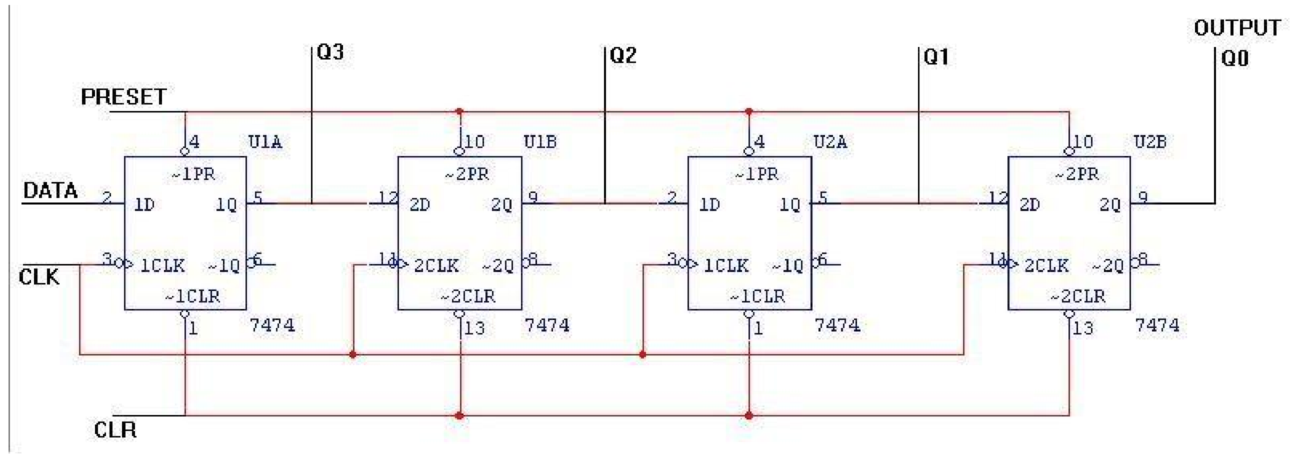
SERIAL IN SERIAL OUT LOGIC DIAGRAM



TRUTH TABLE:

CLK	Serial In	Serial Out
1	1	0
2	0	0
3	0	0
4	1	1
5	X	0
6	X	0
7	X	1

SERIAL IN PARALLEL OUT LOGIC DIAGRAM:



TRUTH TABLE:

CLK	DATA	OUTPUT			
		QA	QB	QC	QD
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	0	0	1

PROCEDURE:

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

RESULT:

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

11. SIMULATION OF ADDERS and SUBTRACTORS

AIM:

To implement adders and Subtractors using Verilog HDL

APPARATUS REQUIRED:

PC with Windows, ModelSim software.

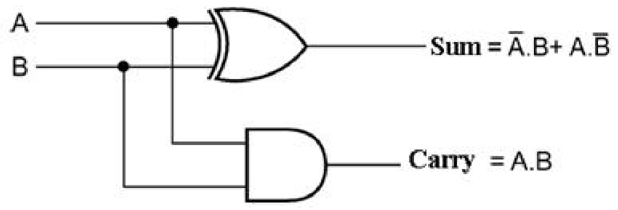
PROCEDURE:

- Write and draw the Digital logic system.
- Write the Verilog code for above system.
- Enter the Verilog code in Modelsim editor.
- Check the syntax and simulate the above verilog code and verify the output waveform as obtained.

PROGRAM

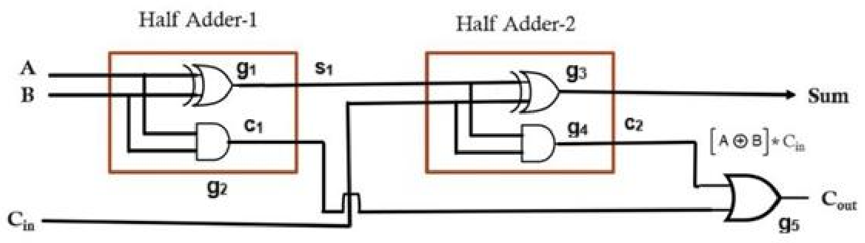
STRUCTURAL/GATE LEVEL MODELING:

Half adder



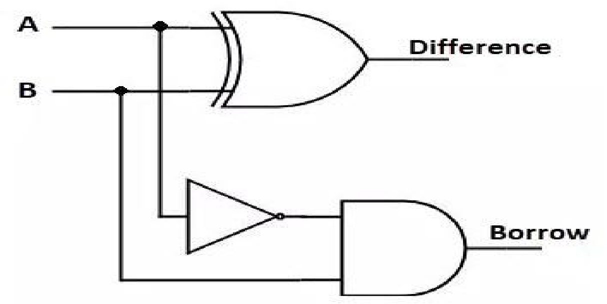
```
module half_adder(a,b,sum,carry);
input a,b;
output sum,carry;
xor (sum,a,b);
and (carry,a,b);
endmodule
```

Full adder



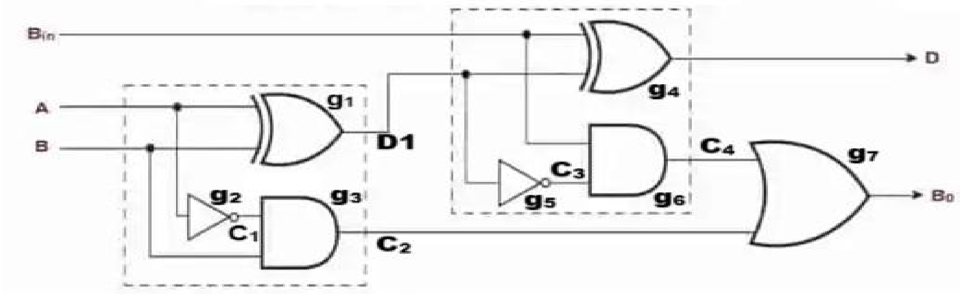
```
module fulladd(sum, cout, a, b, cin);
input a, b, cin;
output sum, cout;
wire s1, c1, c2;
 xor g1(s1, a, b);
and g2 (c1, a, b);
xor g3(sum, s1, cin);
and g4(c2, s1, cin);
or g5(cout, c2, c1);
endmodule
```

Half subtractor



```
module half_subtractor(a,b,difference,borrow);
input a,b;
output difference,borrow;
wire abar;
xor (difference,a,b);
not (abar, a)
and (borrow,abar,b);
endmodule
```

Full subtractor



```
module fullsub(D, B0, A, B, Bin);
input A, B, Bin;
output D, B0;
wire D1, C1, C2, C3, C4;
 xor g1(D1, A, B);
not g2 (C1, A);
and g3(C2,C1,B);
xor g4(D,D1,Bin);
not g5 (C3, D1);
and g6(C4, C3, Bin);
or g7(B0, C2, C4);
endmodule
```

DATAFLOW MODELING
Half Adder

```
module half_adder(a,b,sum,carry);
input a,b;
output sum,carry;
assign sum=a^b;
assign carry=a&b;
endmodule
```

Full Adder

```
module full_adder(a,b,c,sum,carry);
input a,b,c;
output sum,carry;
assign sum=a^b^c;
assign carry=a&b|b&c|c&a;
endmodule
```

Half subtractor

```
module half_subtractor(a,b, difference, borrow );
 input a,b;
output difference, borrow;
assign difference=a^b;
assign borrow=((~a)&b);
endmodule
```

Full Subtractor

```
module full_ subtractor (a,b,c, difference, borrow);
input a,b,c;
output difference, borrow;
assign difference=a^b^c;
assign borrow =(~a)&b)|(b&c)|(c&(~a);
endmodule
```

Behavioural Modeling
Half adder

```
module half_adder(a,b,sum,carry);
input a,b;
output reg sum,carry;
always @ (a or b)
begin
sum=a^b;
carry=a&b;
end
endmodule
```

Full adder

```
module full_adder(a,b,c,sum,carry);
input a,b,c;
output reg sum,carry;
always @ (a or b or c)
begin
sum=a^b^c;
carry=(a&b)|(b&c)|(c&a);
end
endmodule
```

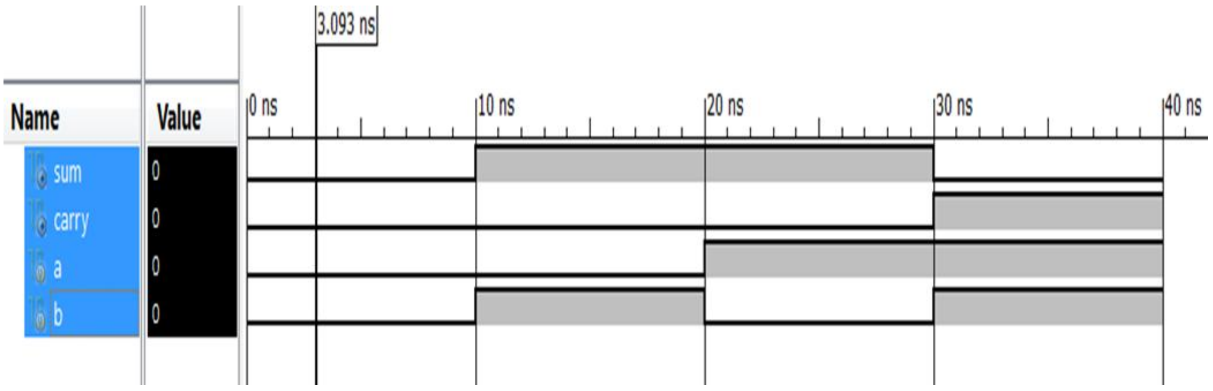
Half subtractor

```
module half_subtractor(a,b, difference, borrow );
 input a,b;
output reg difference, borrow;
always @ (a or b)
begin
difference=a^b;
borrow=((~a)&b);
end
endmodule
```

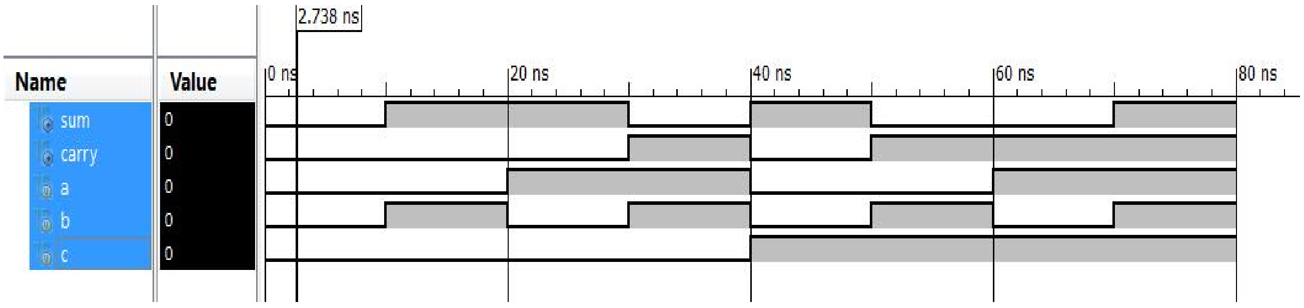
Full Subtractor

```
module full_ subtractor (a,b,c,difference, borrow);
input a,b,c;
output reg difference, borrow;
always @ (a or b or c)
begin
difference=a^b^c;
borrow=((~a)&b)|(b&c)|(c&(~a));
end
endmodule
```

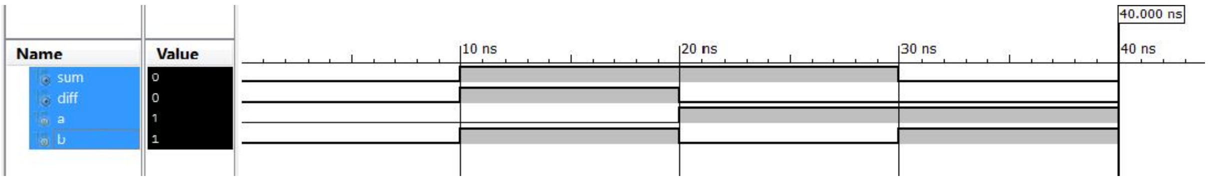
WAVEFORM:
Half Adder



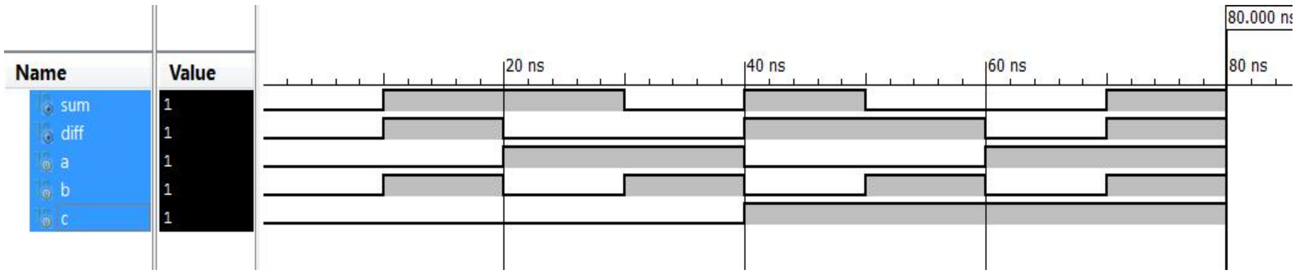
Full adder



Half Subtractor



Full Subtractor



RESULT:

Thus, the half adder, full adder, half subtractor and full subtractor are simulated and synthesized using Verilog HDL.

12. SIMULATION OF MULTIPLEXER AND DEMULTIPLEXER

AIM:

To implement 4-to-1 multiplexer and 1-to-4 demultiplexer using Verilog HDL

APPARATUS REQUIRED:

PC with Windows, ModelSim software.

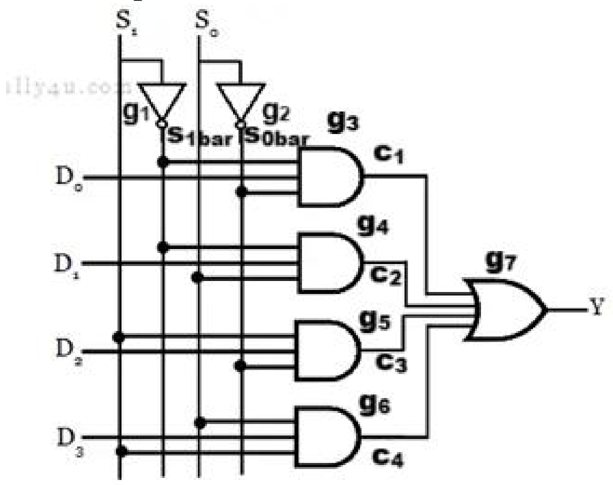
PROCEDURE:

- Write and draw the Digital logic system.
- Write the Verilog code for above system.
- Enter the Verilog code in Modelsim editor.
- Check the syntax and simulate the above verilog code and verify the output waveform as obtained.

PROGRAM

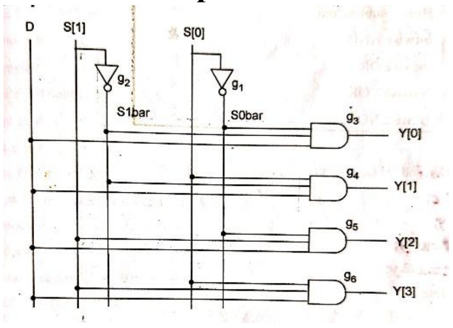
STRUCTURAL/GATE LEVEL MODELING:

4:1 Multiplexer



```
module multiplexer (Y, D, S);
output Y;
input [1:0]S;
input [3:0]D;
wire C1, C2, C3, C4, S0bar, S1bar;
not g1(S1bar, S[1] );
not g2(S0bar, S[0] );
and g3(C1 S0bar, S1bar, D[0] );
and g4 (C2, S[0] ,S1bar, D[1]);
and g5(C3 S0bar, S[1], D[2]);
and g6(C4,S[0] S[1], D[3]);
or g7 (Y, C1, C2, C3, C4);
endmodule
```

1 to 4 Demultiplexer



```
module demultiplexer (Y, D, S);
output [3:0] Y;
input [1:0] S;
input D;
wire S1bar, S0bar;
not g1( S0bar, S[0]);
not g2 (S1bar, S[1] );
and g3 (Y[0], S0bar, S1bar, D);
and g4 (Y[1], S[0], S1bar, D);
and g5 (Y[2], S0 bar, S[1], D);
and g6 (Y[3], S[0], S[1], D);
endmodule
```

DATA FLOW MODELING:

4:1 Multiplexer

```
module mux (i0,i1,i2,i3,s0,s1,y);
input s0,s1,i0,i1,i2,i3;
output y;
assign y= (~s0 & ~s1 &i0)| (~s0 & s1 &i1) | (s0 & ~s1 &i2) | (s0 & s1 &i3);
endmodule
```

1:4 demultiplexer

```
module demultiplexer (Y, D, S);
input [1:0] S;
input D;
output [3:0] Y;
assign Y[0] =D&~S[1]& ~S[0];
assign Y[1] =D& ~S[1] &S[0];
assign Y[2] =D&S[1]& ~S[0];
assign Y[3] =D&S[1]&S[0];
end module
```

BEHAVIOURAL MODELING

4 to 1 multiplexer

```
module m41 ( a, b, c, d, s0, s1, out);
input wire a, b, c, d;
input wire s0, s1;
output reg out;
always @ (a or b or c or d or s0, s1)
```

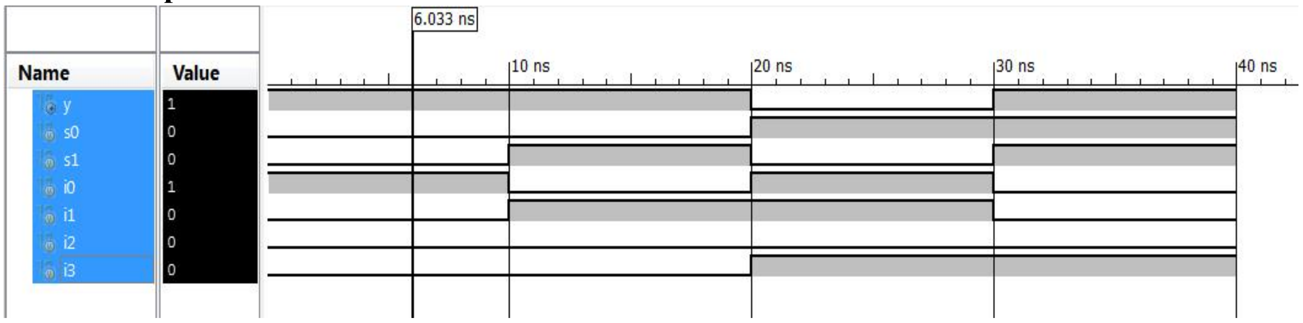
```
begin
case (s0 | s1)
2'b00 : out <= a;
2'b01 : out <= b;
2'b10 : out <= c;
2'b11 : out <= d;
endcase
end
endmodule
```

1 to 4 demultiplexer

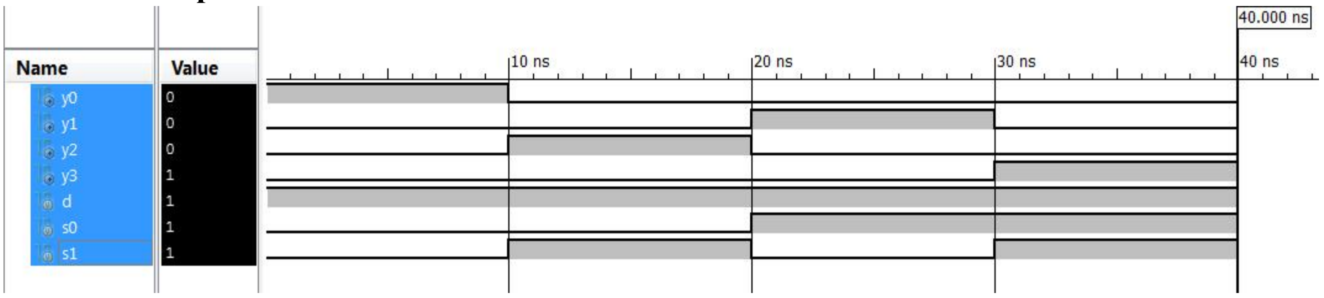
```
module demux (d,s,y);
input d;
input [2:0]s;
output reg [3:0]y;
 always @ (s or d)
 case (s)
2'b00 : begin y[0]=d; y[1]=0; y[2]=0; y[3]=0; end
2'b01 : begin y[0]=0; y[1]=d; y[2]=0; y[3]=0; end
2'b10 : begin y[0]=0; y[1]=0; y[2]=d; y[3]=0; end
2'b11 : begin y[0]=0; y[1]=0; y[2]=0; y[3]=d; end
endcase
endmodule
```

WAVEFORM:

4-to-1 multiplexer



1-to-4 demultiplexer



RESULT:

Thus, the 4-to-1 multiplexer and 1-to-4 demultiplexer are simulated and synthesized using Verilog HDL.

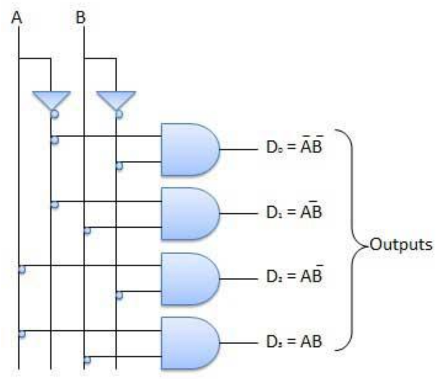
13. SIMULATION OF 2-TO-4 DECODER AND A 4-TO-2 ENCODER

AIM:
To implement 2-to-4 decoder and a 4-to-2 encoder using Verilog HDL

APPARATUS REQUIRED:
PC with Windows, ModelSim software.

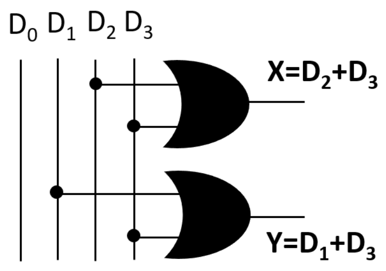
PROCEDURE:
- Write and draw the Digital logic system.
- Write the Verilog code for above system.
- Enter the Verilog code in Modelsim editor.
- Check the syntax and simulate the above verilog code and verify the output waveform as obtained.

PROGRAM
STRUCTURAL/GATE LEVEL MODELING:
2 to 4 Decoder



```
module decoder (D, A, B);
output [3:0] D;
input A, B;
wire Abar, Bbar;
not g1(Abar, A);
not g2 (Bbar, B);
and g3 ( D[0] Abar, Bbar);
and g4 ( D[1] Abar, B);
and g5 (D[2], A, Bbar);
and g6 (D[3], A, B)
end module
```

4 to 2 encoder



```
module encoder (X,Y, D);
```

```
input [7:0] D;
output X,Y;
or g1(X, D[2],D[3]);
or g2(Y, D[1],D[3]);
end module
```

DATA FLOW MODELING:

2-to-4 decoder

```
module decoder (a,b,y);
output [3:0]y;
input a,b;
assign y[0]= ~a & ~b;
assign y[1]= ~a & b;
assign y[2]=  a & ~b;
assign y[3]= a & b;
endmodule
```

4-to-2 encoder

```
module encoder (X,Y, D);
input [3:0] D;
output X,Y;
assign X= D[2]|D[3];
assign Y= D[1]|D[3];
end module
```

BEHAVIOURAL MODELING

2 to 4 decoder

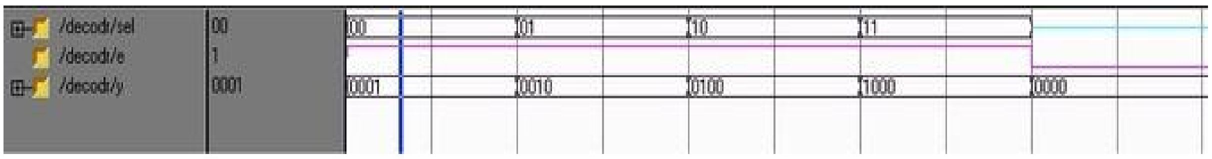
```
module decoder ( d0,d1 out0, out1, out2, out3);
input d0, d1;
output reg out0, out1, out2, out3;
always @ ( d0,d1)
begin
case ({d0,d1} )
2'b00: {out0, out1, out2, out3 \ = 4' b1000;
2'b01: {out0, outl, out2, out3} = 4' b0100;
2'b10: {out0, out1, out2, out3} = 4' b0010;
2'b11: {out0, out1, out2, out3} = 4' b0001;
default:$display ("Invalid");
end case
end
```

4 x 2 Encoder

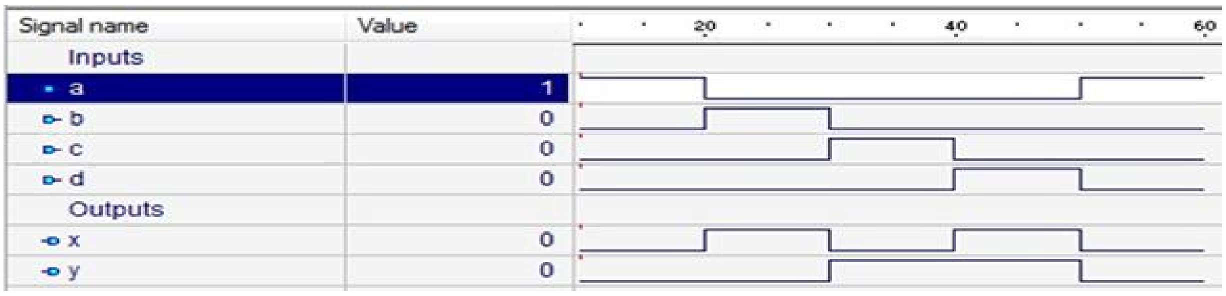
```
module encoder ( d0,d1,d2,d3 out0, out1);
input d0, d1, d2, d3;
output reg out0, out1;
always @ (d0, d1, d2, d3)
begin
case (d0, d1, d2, d3)
4'b1000: {out0, out1} = 2' b00;
4'b0100: {out0, out1} = 2' b01;
4'b0010: {out0; out1} = 2' b10;
4'b0001: {out0; out1} = 2' b11;
default:$display ("Invalid");
end case
end
end module.
```

WAVEFORM:

2 to 4 decoder



4 to 2 encoder



RESULT:

Thus, the 2-to-4 decoder and a 4-to-2 encoder are simulated and synthesized using Verilog HDL.

14. Simulation of Mod 5 Counter

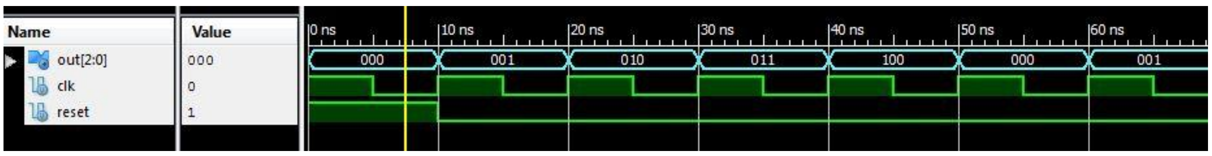
AIM:
To implement mod-5 counter using Verilog HDL

APPARATUS REQUIRED:
PC with Windows, ModelSim software.

PROCEDURE:
- Write and draw the Digital logic system.
- Write the Verilog code for above system.
- Enter the Verilog code in Modelsim editor.
- Check the syntax and simulate the above verilog code and verify the output waveform as obtained.

PROGRAM
```
module mod5( out, clk, reset);
output [2:0] out;
input clk, reset;
reg [2:0] out;
always @(posedge clk)
 if (reset)
  begin
   out <= 2'b0 ;
  end
 else if (out<4)
  begin
   out <= out + 1;
  end
 else
  begin
   out <= 2'b0 ;
  end
endmodule
```

WAVEFORM:



RESULT:
Thus, the operation of the MOD-5 counter was verified using modelsim with the help of resultant waveform.