SIMATS ENGINEERING

Engineer to Excel

SIMATS

Saveetha Institute of Medical And Technical Sciences

(Declared as Deemed to be University under Section 3 of UGC Act 1956)

Department of Electronics andCommunication Engineering

ECA02 DIGITAL CIRCUITS

LAB MANUAL

INDEX

1. STUDY OF LOGIC GATES

DATE:

AIM :

To study logic gates and verify their truth tables of all the logic functions.

APPARATUS REQUIRED:

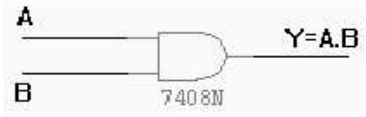
SL No.	COMPONENT	SPECIFICATION	QTY
1.	AND GATE	IC 7408	1
2.	OR GATE	IC 7432	1
3.	NOT GATE	IC 7404	1
4.	NAND GATE 2 I/P	IC 7400	1
5.	NOR GATE	IC 7402	1
6.	X-OR GATE	IC 7486	1
8.	IC TRAINER KIT	-	1
9.	CONNECTING WIRES		

PROCEDURE:

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

AND GATE:

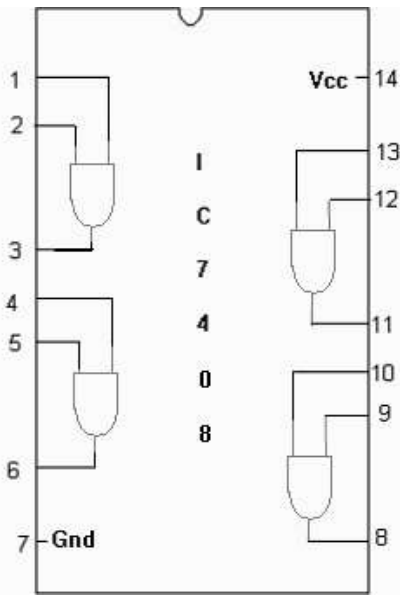
SYMBOL:



TRUTH TABLE

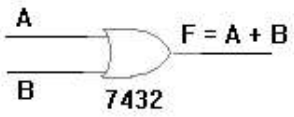
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

PIN DIAGRAM:

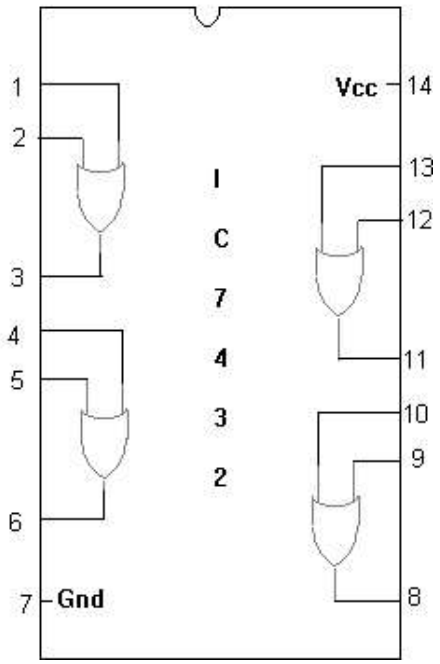


OR GATE:

SYMBOL :



PIN DIAGRAM :

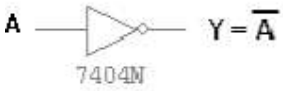


TRUTH TABLE

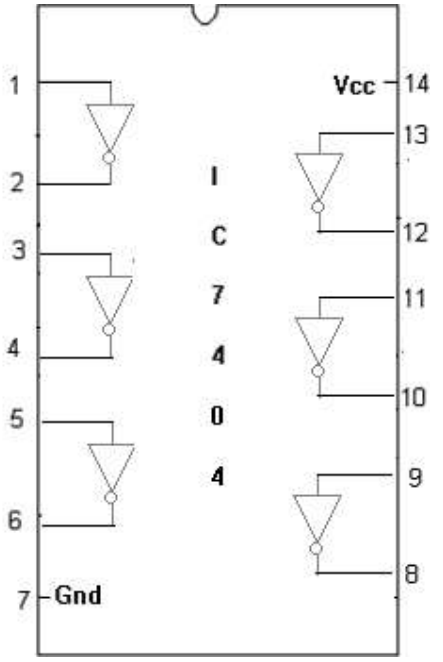
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

NOT GATE:

SYMBOL:



PIN DIAGRAM:

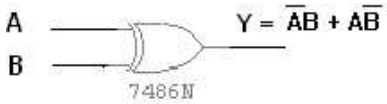


TRUTH TABLE :

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

X-OR GATE:

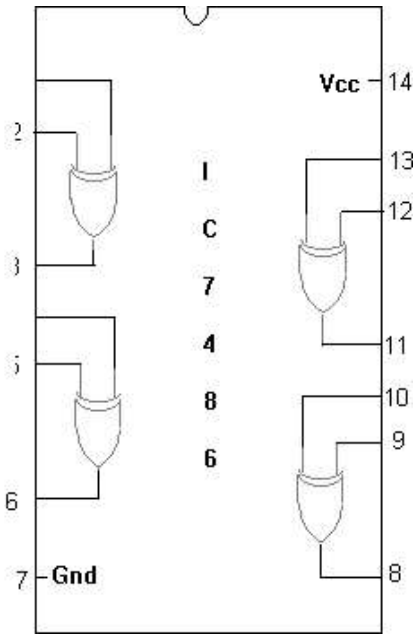
SYMBOL:



TRUTH TABLE :

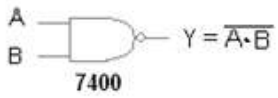
A	B	$\overline{A}B + A\overline{B}$
0	0	0
0	1	1
1	0	1
1	1	0

PIN DIAGRAM :



2-INPUT NAND GATE:

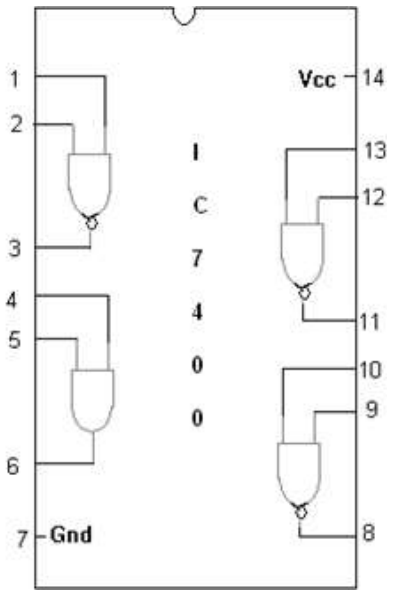
SYMBOL:



TRUTH TABLE

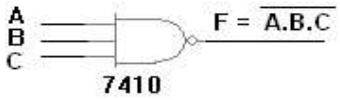
A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

PIN DIAGRAM:

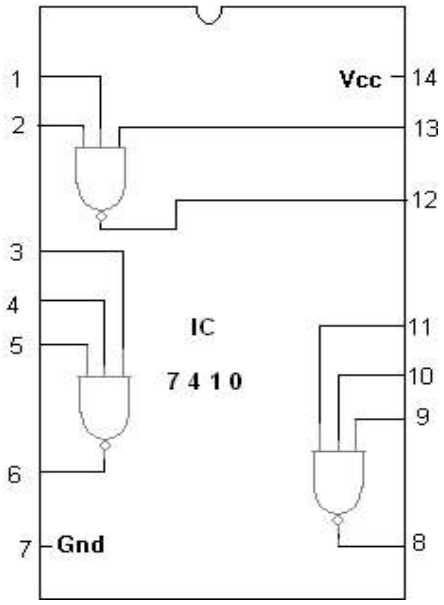


3-INPUT NAND GATE :

SYMBOL :



PIN DIAGRAM :

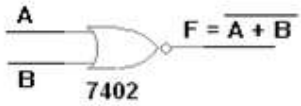


TRUTH TABLE

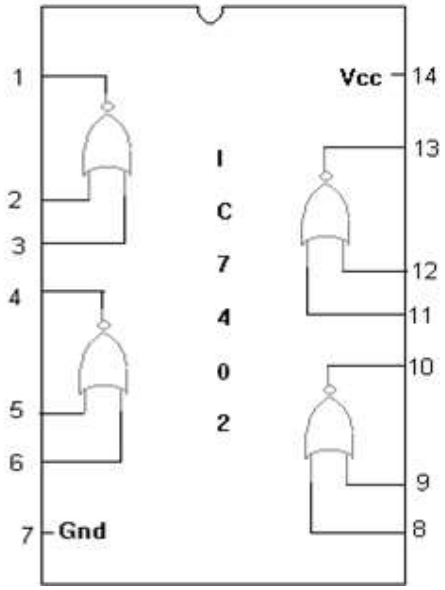
A	B	C	$\overline{A.B.C}$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

NOR GATE:

SYMBOL :



PIN DIAGRAM :



TRUTH TABLE

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

RESULT:

Thus, the gates (AND, OR, NOT, NAND, NOR, XOR, and XNOR GATES) are studied and verified using the Truth table.

Test Case1:

Design a 3-bit majority voting circuit that outputs 1 if most of the three inputs are 1

Steps:

(a)Use three 2-input AND gates to compute A·B, A·C, and B.C

(b) Use a 3-input OR gate to combine these results

Truth Table

A	B	C	A·B	A·C	B.C	F
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	0	0	0
1	0	1	0	1	0	1
1	1	0	1	0	0	1
1	1	1	1	1	1	1

Test Case 2:

Design a circuit to check the parity of a 4-bit binary number. The output should be 1 if the number has even parity (even number of 1s)

Steps: Cascade XOR gates

Compute P1=A⊕B

Compute P2=P1⊕C

Compute F=P2⊕D

Truth Table

A	B	C	D	A⊕B	P2	F
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	0	1	1	0	1	0
1	1	1	1	0	1	0

2. VERIFICATION OF BOOLEAN THEOREMS USING LOGIC GATES

DATE:

AIM :

To verify Boolean theorems using logic gates

APPARATUS REQUIRED:

Sl.No.	COMPONENT	SPECIFICATION	QTY.
1.	AND GATE	IC 7408	1
2.	NOT GATE	IC 7404	1
3.	OR GATE	IC 7432	1
4.	IC TRAINER KIT	-	1
5.	CONNECTING WIRES		

PROCEDURE:

a. Connections are given as per the logic diagram
b. Input is given to the circuit making high '1' i.e., +5 or Vcc Supply to the 14th pin, and for low '0' i.e., GND to the 7th pin of Gate IC
c. Depending upon the truth table, if the LED Glow represents 1 and else it represents '0'
d. Verify the truth table as given
e. Repeat the procedure steps for different theorems.

BOOLEAN THEOREM:

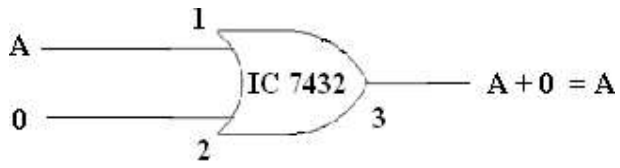
POSTULATES	**I**	**II**
Identity	A + 0 = A	A. 1 = A
Commutative	A + B = B + A	AB = BA
Distributive	A (B + C) = AB + AC	A + BC = (A + B) (A + C)
Complement	A + A' =1	A.A' = 0
Idempotency	A + A = A	A.A = A
	A + 1 = 1	A.0 = 0
Involution	(A')' = A	
Absorption	A + AB = A	A (A + B) = A
	A + A'B = A + B	A. (A'+ B) = AB
Associative	A + (B + C) = (A + B) + C	A (B C) = (AB) C
De Morgan's Law	(A + B)' =A'. B'	(AB)' = A'+ B'

1. **IDENTITY**

TRUTH TABLE

I/P		O/P
A	**0**	**A+0= A**
0	0	**0**
1	0	**1**

LOGIC DIAGRAM

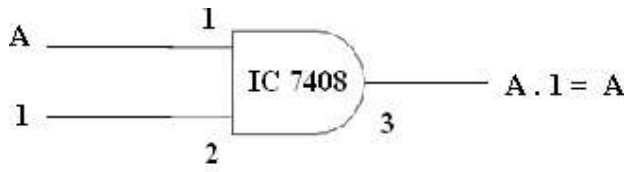


Identity law A + 0 = A

TRUTH TABLE

I/P		O/P
A	**0**	**A.1=A**
0	0	0
1	0	1

LOGIC DIAGRAM



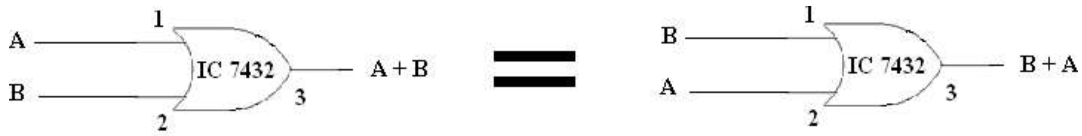
Identity law A . 1 = A

2. **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

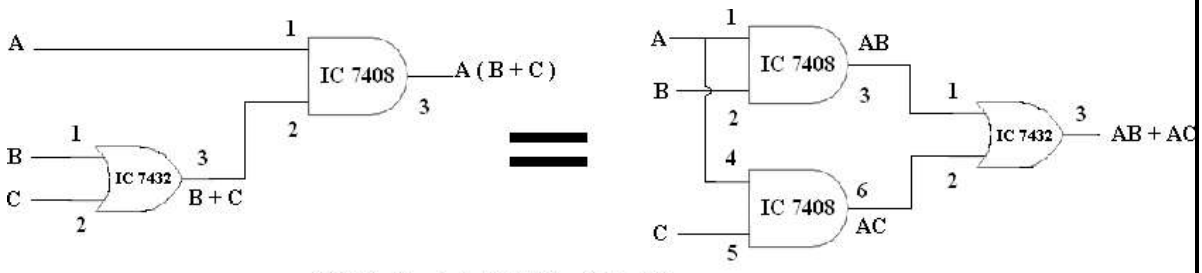
Commulative law AB = BA

3. DISTRIBUTIVE

TRUTH TABLE

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

LOGIC DIAGRAM

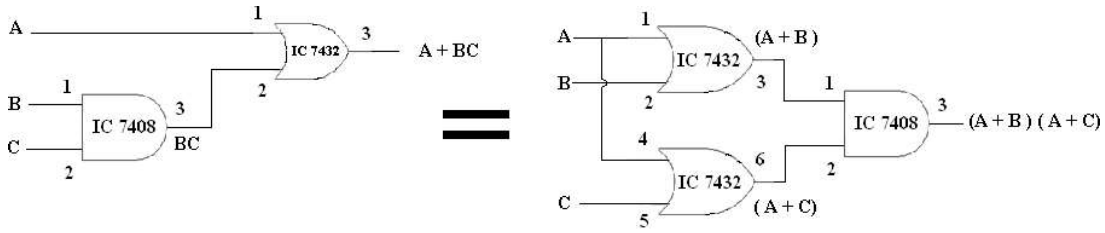


Distributive 1 A (B + C) = AB + AC

TRUTH TABLE

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

LOGIC DIAGRAM



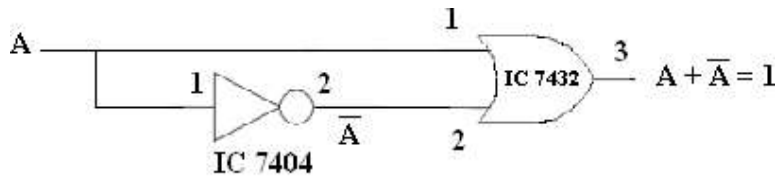
Distributive 2 A + BC = (A + B) (A + C)

4. **COMPLEMENT**

TRUTH TABLE

I/P		O/P
A	A'	A+A'
0	1	1
1	0	1

LOGIC DIAGRAM

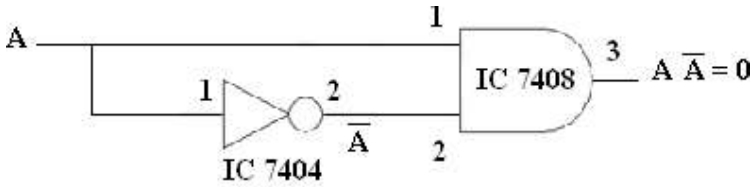


Complement 1 A + A' =1

TRUTH TABLE

I/P		O/P
A	A'	AA'
0	1	0
1	0	0

LOGIC DIAGRAM



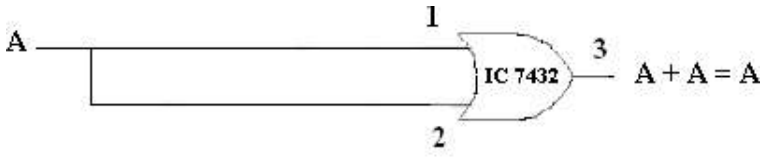
Complement 2 A.A' = 0

5. **IDEMPOTENCY**

TRUTH TABLE

I/P		O/P
A	A	A+A
0	0	0
1	1	1

LOGIC DIAGRAM

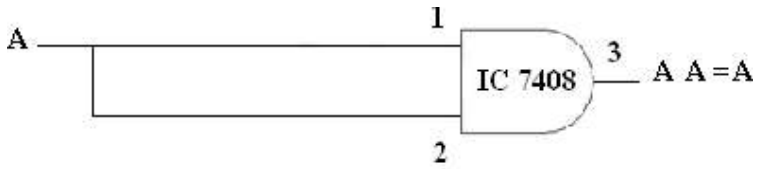


Idempotency 1 A + A = A

TRUTH TABLE

I/P		O/P
A	A	AA
0	0	0
1	1	1

LOGIC DIAGRAM

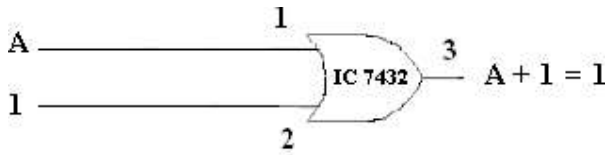


Idempotency 2 A.A = A

TRUTH TABLE

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

LOGIC DIAGRAM

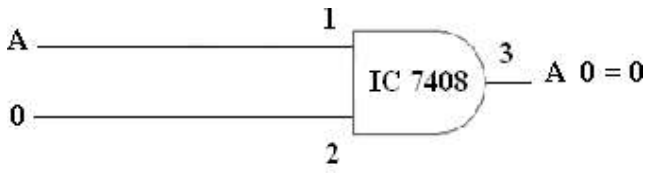


Idempotency 3 A + 1 = 1

TRUTH TABLE

I/P		O/P
A	**0**	**A.0**
0	0	0
1	0	0

LOGIC DIAGRAM



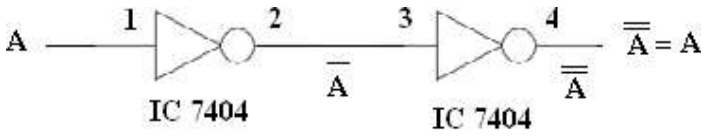
Idempotency 4 A.0 = 0

6. INVOLUTION

TRUTH TABLE

I/P		O/P
A	**A'**	**(A')'**
0	1	**0**
1	0	**1**

LOGIC DIAGRAM



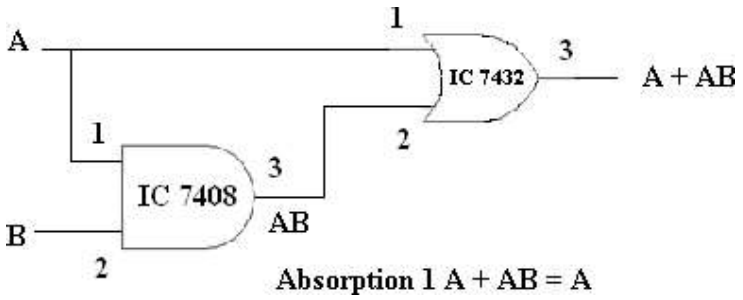
Involution (A')' = A

7. 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**

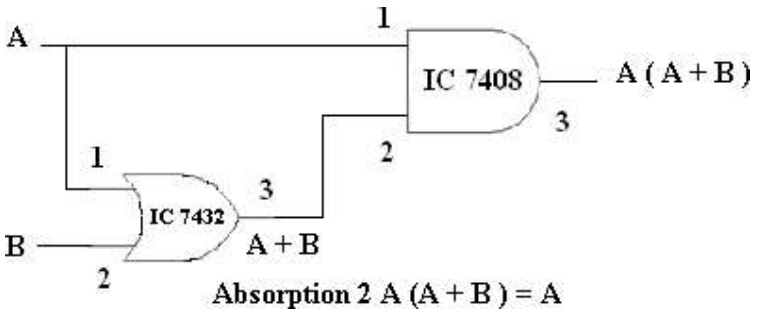
LOGIC DIAGRAM



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**

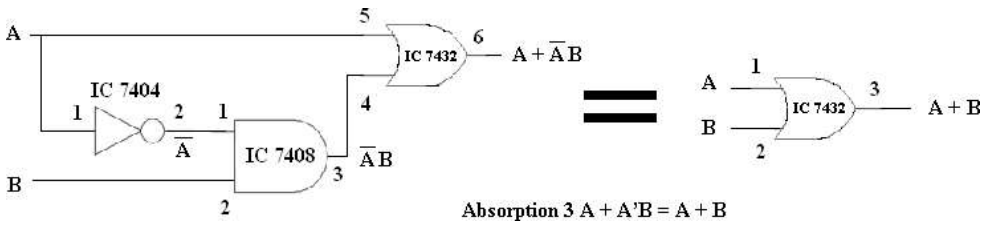
LOGIC DIAGRAM



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

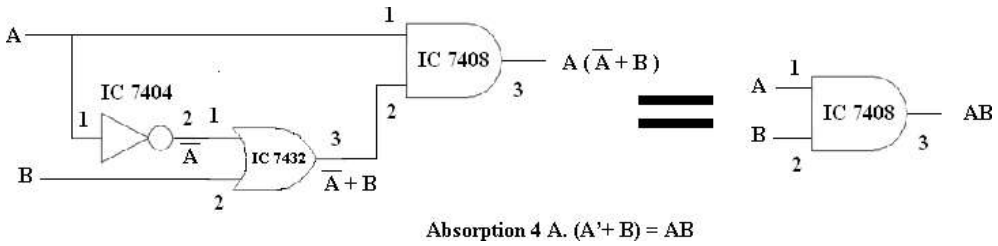


Absorption 3 A + A'B = A + B

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



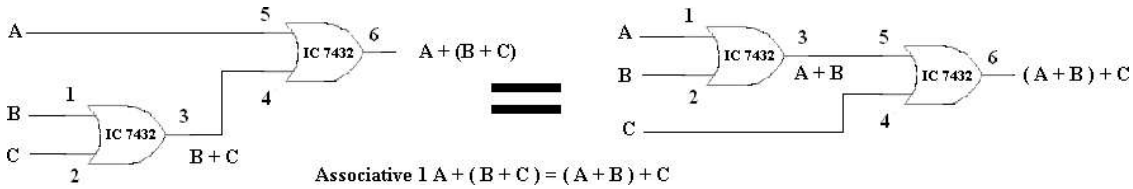
Absorption 4 A. (A'+ B) = AB

8. **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

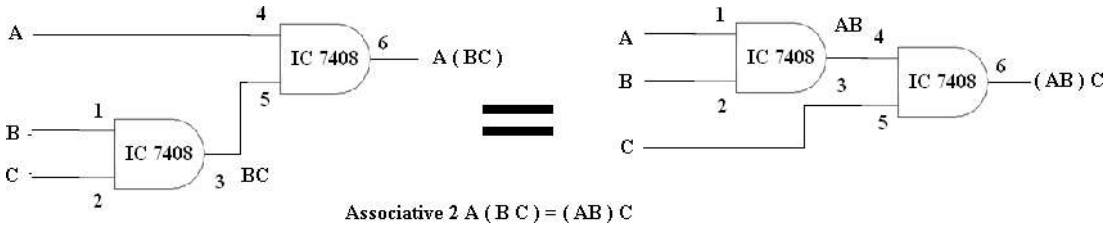


Associative 1 A + (B + C) = (A + B) + C

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



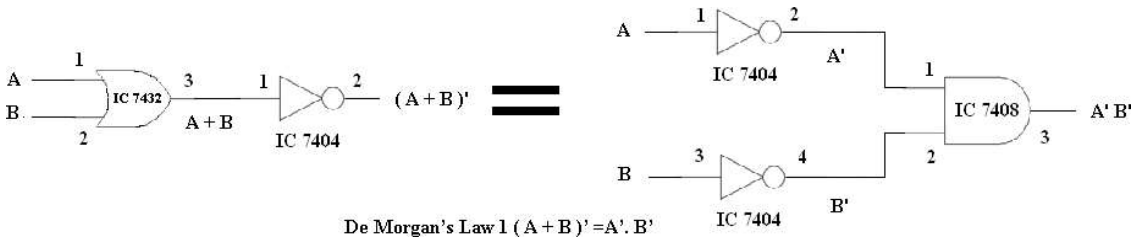
Associative 2 A (B C) = (AB) C

9. **DE MORGAN'S LAW**

TRUTH TABLE

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

LOGIC DIAGRAM

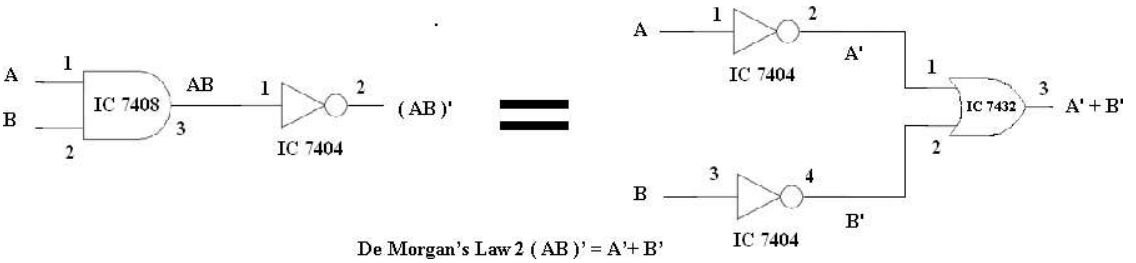


De Morgan's Law 1 (A + B)' =A'. B'

TRUTH TABLE

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

LOGIC DIAGRAM



De Morgan's Law 2 (AB)' = A'+ B'

RESULT:

The different theorems of Boolean algebra and Demorgan's theorem are designed andverified using logic gates.

Test Case 1:

Cascading De Morgan's Theorem - Verify the equivalence of the two expressions

1. ((A·B) ′+(C·D) ′) ′
2. A·B·C′+D′

Stepwise calculation - Expression 1: ((A·B) ′+(C·D) ′) ′

1. Compute A·B and C·D
2. Compute (A·B) ′ and (C·D) ′
3. Compute (A·B) ′+(C·D) ′
4. Finally, negate the result: ((A·B) ′+(C·D) ′) ′

Expression 2: A·B·(C′+D′)

1. Compute C′ and D′
2. Compute C′+D′
3. Compute A·B·(C′+D′)

Conclusion: Two expressions are not logically equivalent.

Test Case 2:

Design a digital circuit that demonstrates the practical application of the Complement Law in a multi-variable logic system. Specifically, simplify and verify the expression F=A·A·B + (A+A) ·C

Step 1: Simplify the Expression

A	B	C	A.A'.B	A+A'	(A+A').C	F=C
0	0	0	0	1	0	0
0	0	1	0	1	1	1
0	1	0	0	1	0	0
0	1	1	0	1	1	1
1	0	0	0	1	0	0
1	0	1	0	1	1	1
1	1	0	0	1	0	0
1	1	1	0	1	1	1

Simplified Expression: F=C

3. IMPLEMENTATION OF COMBINATION CIRCUIT FOR AN ARBITRARY FUNCTION

DATE:

AIM:

To Realize the Boolean Expression AB + BC + AC using Logic gates and verify its performance with its truth table and the Digital Trainer kit. **Y= AC + AB + BC**

APPARATUS REQUIRED:

Sl.No.	COMPONENT	SPECIFICATION	QTY.
1.	AND GATE	IC 7408	1
2.	OR GATE	IC 7432	1
3.	IC TRAINER KIT	-	1
4.	CONNECTING WIRES	-	few

THEORY:

Combinational logic circuits are circuits in which the output at any time depends upon the combination of input signals present at that instant only, and does not depend on any past conditions. The combinational circuit block can be considered as a network of logic gates thataccept signals from inputs and generate signals to outputs

PROCEDURE:

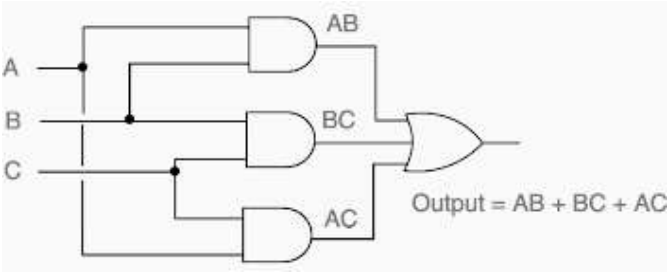
a. Connections are given as per the logic diagram

b. The input is given to the circuit, making high '1', i.e., +5 or Vcc Supply to the 14th pin, and low '0', i.e., GND to the 7th pin of the Gate IC.

c. c. Depending upon the truth table, if the LED Glows, it represents 1, and otherwise,it represents '0'.

d. Verify the truth table as given

LOGIC DIAGRAM:



TRUTH TABLE:

I/P						O/P
A	**B**	**C**	**AB**	**BC**	**AC**	**AB + BC + AC**
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	1
1	0	0	0	0	0	0
1	0	1	0	0	1	1
1	1	0	1	0	0	1
1	1	1	1	1	1	1

RESULT:

Thus, the given Arbitrary function AB+BC=AC was verified using truth table.

Test case:

- Set the values of A, B, and C using the input switches.
- Observe the output LED.
- Compare the LED output with the expected result from the truth table

Test Case 1: A=0, B=0, C=0

- Inputs: A = 0, B = 0, C = 0
- Expected Output: AB+BC+AC=0
- Reasoning: All terms AB, BC, and AC are 0, so the final output should be 0.

Test Case 2: A=0, B=0, C=1

- Inputs: A = 0, B = 0, C = 1
- Expected Output: AB+BC+AC=0
- Reasoning: All terms AB, BC, and AC are 0, so the final output should be 0.

Test Case 3: A=0, B=1, C=0

- Inputs: A = 0, B = 1, C = 0
- Expected Output: AB+BC+AC=0
- Reasoning: All terms AB, BC, and AC are 0, so the final output should be 0.

Test Case 4: A=0, B=1, C=1

- Inputs: A = 0, B = 1, C = 1
- Expected Output: AB+BC+AC=1
- Reasoning: AB=0, BC=1, AC=0. The OR gate will output 1.

Test Case 5: A=1, B=0, C=1

- Inputs: A = 1, B = 0, C = 1
- Expected Output: AB+BC+AC=1
- Reasoning: AB=0, BC=0, AC=1. The OR gate will output 1.

4. DESIGN AND IMPLEMENTATION OF CODE CONVERTERS USING LOGIC GATES

DATE:

AIM: To design and implement a 4-bit code converter for the following

a. Binary to gray code converter
b. Gray to binary code converter

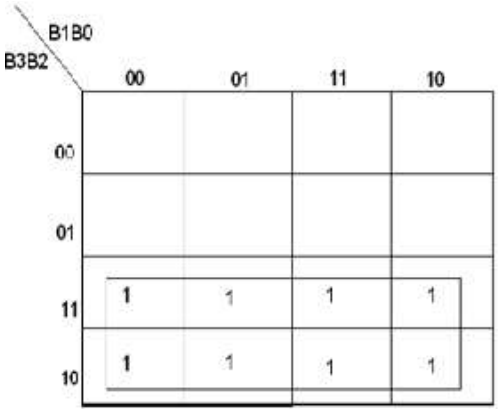
APPARATUS REQUIRED:

Sl.No.	COMPONENT	SPECIFICATION	QTY
1.	X-OR GATE	IC 7486	1
5.	IC TRAINER KIT	-	1
6.	CONNECTING WIRES	-	35

DESIGN FOR BINARY TO GRAY CODE CONVERTERTRUTH TABLE:

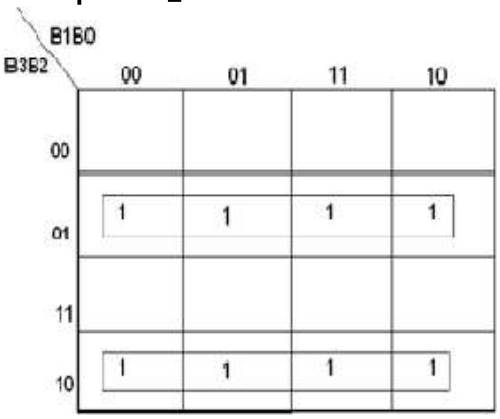
Binary input				**Gray code output**			
B3	**B2**	**B1**	**B0**	**G3**	**G2**	**G1**	**G0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**1**	**0**	**0**	**0**	**1**
0	**0**	**1**	**0**	**0**	**0**	**1**	**1**
0	**0**	**1**	**1**	**0**	**0**	**1**	**0**
0	**1**	**0**	**0**	**0**	**1**	**1**	**0**
0	**1**	**0**	**1**	**0**	**1**	**1**	**1**
0	**1**	**1**	**0**	**0**	**1**	**0**	**1**
0	**1**	**1**	**1**	**0**	**1**	**0**	**0**
1	**0**	**0**	**0**	**1**	**1**	**0**	**0**
1	**0**	**0**	**1**	**1**	**1**	**0**	**1**
1	**0**	**1**	**0**	**1**	**1**	**1**	**1**
1	**0**	**1**	**1**	**1**	**1**	**1**	**0**
1	**1**	**0**	**0**	**1**	**0**	**1**	**0**
1	**1**	**0**	**1**	**1**	**0**	**1**	**1**
1	**1**	**1**	**0**	**1**	**0**	**0**	**1**
1	**1**	**1**	**1**	**1**	**0**	**0**	**0**

K-Map for G_3:



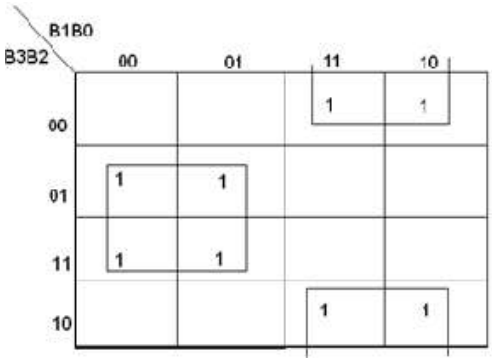
$G_3 = B_3$

K-Map for G_2:



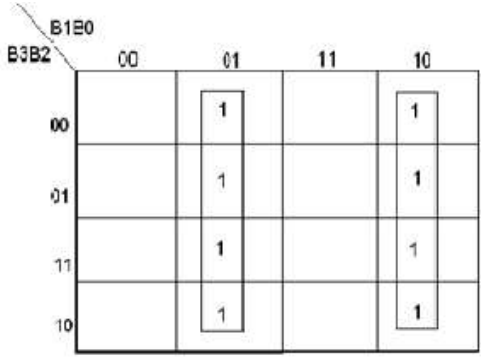
$G2 = B3 \oplus B2$

K-Map for G_1:



$G1 = B1 \oplus B2$

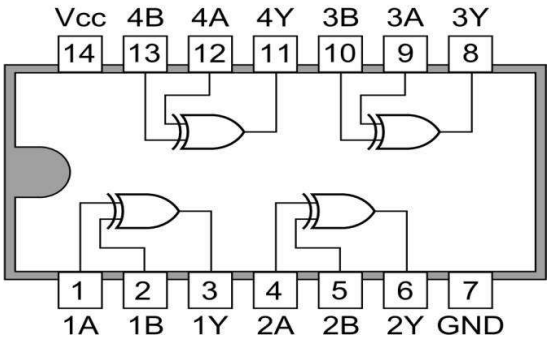
K-Map for G_0:



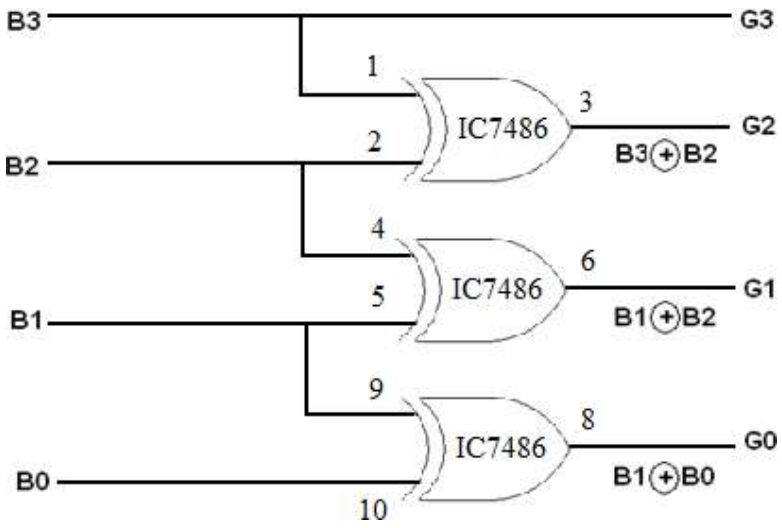
$G0 = B1 \oplus B0$

PIN DIAGRAM:

7486 Quad 2-input ExOR Gates



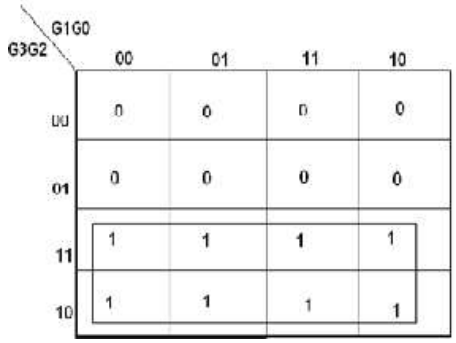
LOGIC DIAGRAM FOR BINARY TO GRAY CODE CONVERTOR:



DESIGN FOR GRAY TO BINARY CODE CONVERTER TRUTH TABLE:

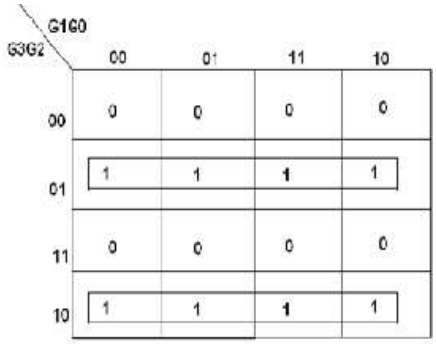
Gray Code				Binary Code			
G3	G2	G1	G0	B3	B2	B1	B0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1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

K-Map for B_3



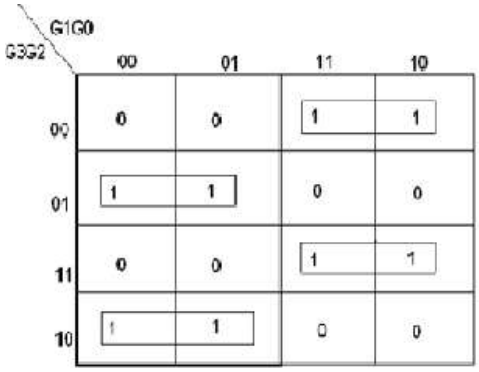
$B_3 = G_3$

K-Map for B_2



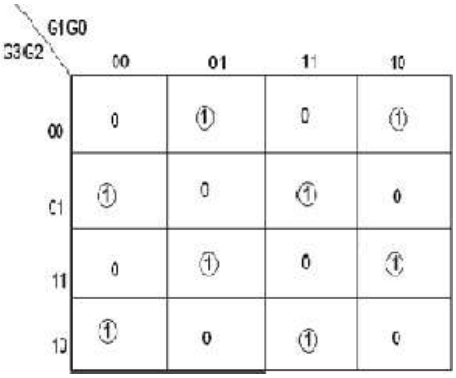
$B_2 = G_3 \oplus G_2$

K-Map for B_1



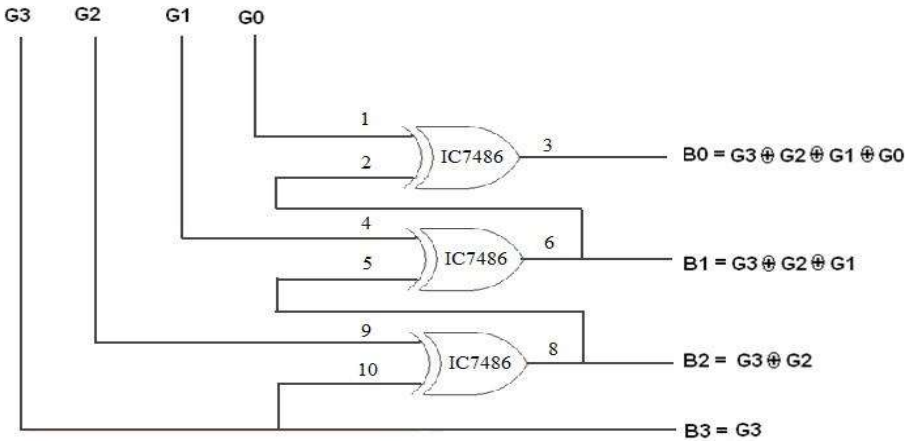
$B_1 = G_3 \oplus G_2 \oplus G_1$

K-Map for B_0



$B_0 = G_3 \oplus G_2 \oplus G_1 \oplus G_0$

LOGIC DIAGRAM FOR GRAY TO BINARY CODE CONVERTER:



RESULT:

Thus, the design for the code converter was done and verified using logic gates successfully.

Test Case 1
Input (Binary): 0000
Expected Output (Gray Code): 0000
Step-by-Step Function Execution:

- G3=B3=0
- G2=B3⊕B2=0⊕0=0
- G1=B2⊕B1=0⊕0=0
- G0=B1⊕B0=0⊕0=0

Result: Gray Code = 0000

Test Case 2
Input (Binary): 0101
Expected Output (Gray Code): 0111
Step-by-Step Function Execution:

- G3=B3=0
- G2=B3⊕B2=0⊕1=1
- G1=B2⊕B1=1⊕0=1
- G0=B1⊕B0=0⊕1=1

Result: Gray Code = 0111
Test Case 3
Input (Binary): 1000
Expected Output (Gray Code): 1100
Step-by-Step Function Execution:

- G3=B3=1
- G2=B3⊕B2=1⊕0=1
- G1=B2⊕B1=0⊕0=0
- G0=B1⊕B0=0⊕0=0

Result: Gray Code = 1100
Test Case 4
Input (Binary): 1110
Expected Output (Gray Code): 1001
Step-by-Step Function Execution:

- G3=B3=1
- G2=B3⊕B2=1⊕1=0
- G1=B2⊕B1=1⊕1=0
- G0=B1⊕B0=1⊕0=1

Result: Gray Code = 1001
Test Case 5
Input (Binary): 0011
Expected Output (Gray Code): 0010
Step-by-Step Function Execution:

- G3=B3=0
- G2=B3⊕B2=0⊕0=0
- G1=B2⊕B1=0⊕1=1
- G0=B1⊕B0=1⊕1=0

Result: Gray Code = 0010

5. DESIGN AND IMPLEMENTATION OF HALF ADDER AND FULL ADDER USING LOGIC GATES

DATE:

AIM:

To design and construct half adder, and full adder circuits and verify the truth tableusing logic gates.

APPARATUS REQUIRED:

Sl.No.	COMPONENT	SPECIFICATION	QTY.
1.	AND GATE	IC 7408	1
2.	X-OR GATE	IC 7486	1
4.	OR GATE	IC 7432	1
5.	IC TRAINER KIT	-	1
6.	CONNECTING WIRES	-	23

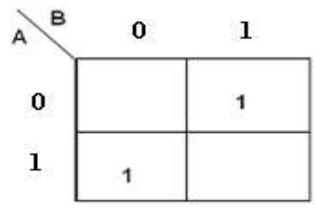
PROCEDURE:

a. Connections are given as per the logic diagram
b. Input are given to the circuit making high '1' i.e. +5 or Vcc Supply to the 14th pin and for low '0' i.e. GND to the 7th pin of the Gate IC
c. Depending upon the truth table, if the LED Glow it represents 1 and else it represents '0'
d. Verify the truth table as given

HALF ADDER: TRUTH TABLE:

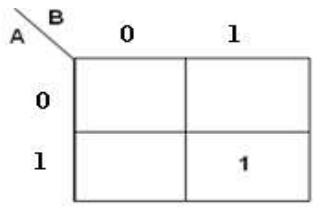
A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

K-Map for SUM:



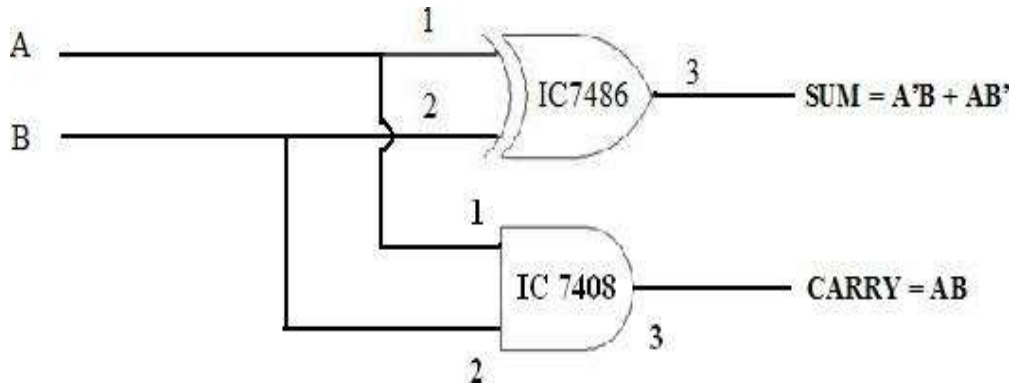
SUM = A'B + AB' = A ⊕ B

K-Map for CARRY:



CARRY = AB

LOGIC DIAGRAM FOR HALF ADDER:



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

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

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

$= A \oplus B \oplus C$

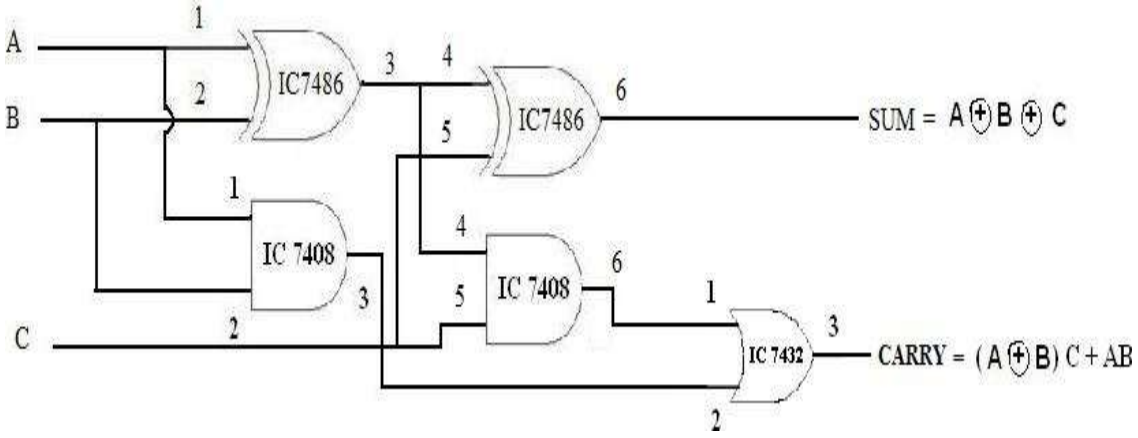
K-Map for CARRY

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

$CARRY = A'BC + AB'C + ABC' + ABC$

$= (A \oplus B)C + AB$

LOGIC DIAGRAM FOR FULL ADDER:



RESULT:

Thus, the design for half adder and full adder were designed and verified successfullyusing Truth table.

Half Adder Test Case 1: Fault Injection

Question: If the XOR gate in the Half Adder fails and behaves like an OR gate, what will be the output for A = 1 and B = 1?

Answer:
Sum = 1 (A OR B = 1 OR 1 = 1).
Carry remains correct = 1 (A AND B = 1 AND 1 = 1).
This creates an incorrect result, indicating XOR gate failure.
Test Case 2:

Question: While implementing a Half Adder in hardware, your team observes increased complexity due to XOR gates. Is it possible to design a Half Adder without using XOR gates? If so, how?

Answer:
Yes, the Sum output can be expressed using only AND, OR, and NOT gates:
Sum = (A AND NOT B) OR (NOT A AND B).
However, this increases the gate count and latency.
Full Adder Test Case 1: Multi-bit Addition
Question: Implement a 4-bit binary adder using Full Adder circuits. Show the step-by-step calculation for inputs A = 1011 and B = 1101. Include carry propagation.
Answer:
Step 1: Add LSBs: A0 = 1, B0 = 1, Cin = 0 → Sum = 0, Carry = 1.
Step 2: A1 = 1, B1 = 0, Cin = 1 → Sum = 0, Carry = 1.
Step 3: A2 = 0, B2 = 1, Cin = 1 → Sum = 0, Carry = 1.
Step 4: A3 = 1, B3 = 1, Cin = 1 → Sum = 1, Carry = 1.
Final result: Sum = 10000.
Test Case 2: Multi-Bit Binary Addition
Question: You are designing a 4-bit Ripple Carry Adder using Full Adders. Describe the circuit's operation for the binary inputs A = 1011 and B = 1101. Include details about Carry propagation.
Answer:
For each bit:
LSB: A0 = 1, B0 = 1 → Sum = 0, Carry = 1.
Second bit: A1 = 1, B1 = 0, Cin = 1 → Sum = 0, Carry = 1.
Third bit: A2 = 0, B2 = 1, Cin = 1 → Sum = 0, Carry = 1.
MSB: A3 = 1, B3 = 1, Cin = 1 → Sum = 1, Carry = 1.
Final output: Sum = 10000.

6. DESIGN AND IMPLEMENTATION OF HALF SUBTRACTOR AND FULL SUBTRACTOR USING LOGIC GATES

DATE:

AIM:

To design and construct 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
3.	IC TRAINER KIT	-	1
4.	CONNECTING WIRES	-	23

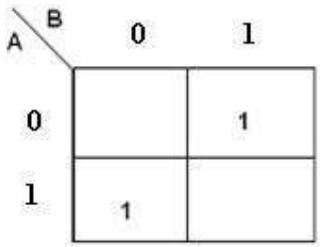
PROCEDURE:

a. Connections are given as per the logic diagram
b. Input is given to the circuit making high '1' i.e. +5 or Vcc Supply to the 14^{th} pin andfor low '0' i.e. GND to the 7^{th} pin of the Gate IC
c. Depending upon the truth table, if the LED Glow it represents 1 and else it represents '0'
d. Verify the truth table as given

HALF SUBRACTOR: TRUTH TABLE

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

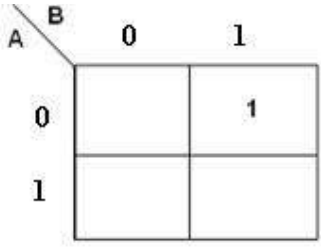
K-Map for DIFFERENCE



DIFFERENCE = A'B + AB'

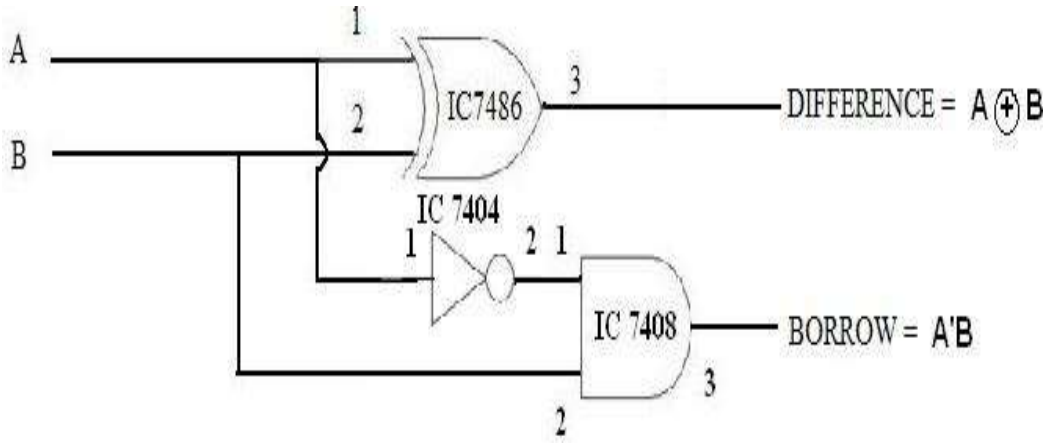
= A ⊕ B

K-Map for BORROW



BORROW = A'B

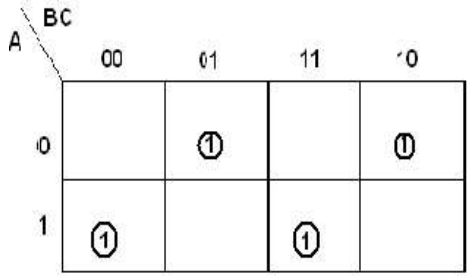
LOGIC DIAGRAM FOR HALF SUBRACTOR:



FULL SUBRACTOR: TRUTH TABLE:

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

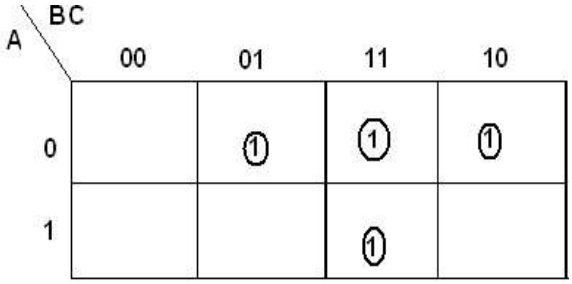
K-Map for DIFFERENCE



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

= A ⊕ B ⊕ C

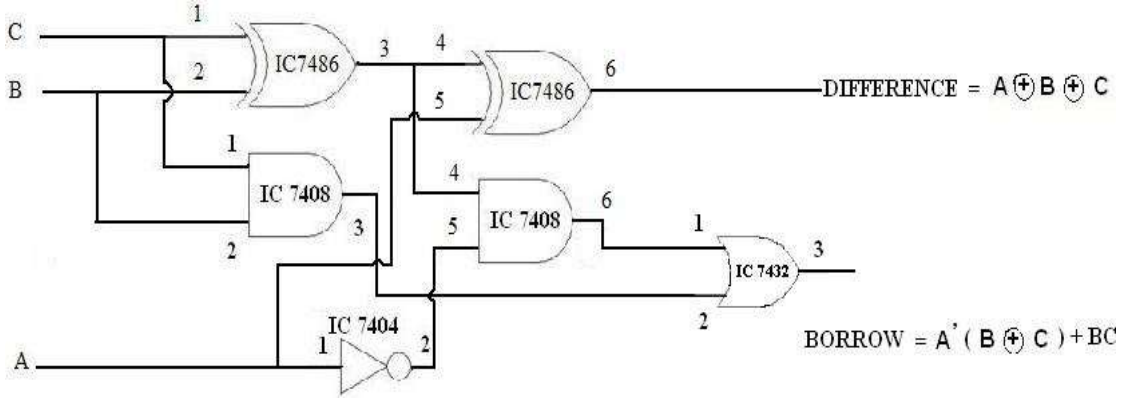
K-Map for BORROW



Borrow = A'B + BC + A'C

= A' (B ⊕ C) + BC

LOGIC DIAGRAM FOR FULL SUBTRACTOR:



RESULT:

Thus, the design for half subtractor and full subtractor were designed and verifiedsuccessfully using the Truth table.

Half Subtractor Test Case 1: Circuit Design

Question: Explain how the Half Subtractor circuit can be implemented on a digital trainer kit using basic logic gates. Why is the XOR gate used for the Difference output, and how does the AND gate with a NOT gate produce the Borrow output?
Answer:
XOR gate computes the Difference: A⊕B, as it outputs 1 when the inputs are different.
NOT gate inverts A, and the AND gate computes Borrow: $\bar{A}\cdot B$, since Borrow occurs when B is 1 and A is 0.
Test Case 2: Fault Detection
Question: During testing, you notice that the Difference output is always 0, irrespective of the inputs. Which component in the Half Subtractor circuit might have failed, and how would you troubleshoot this using the digital trainer kit?
Answer:
Likely issue: Faulty XOR gate.
Troubleshooting: Use a logic probe to verify if the XOR gate correctly processes inputs. Check inputs A and B and confirm if A⊕B is computed.

Full Subtractor Test Case 1: Circuit Design

Question: Design a Full Subtractor circuit using basic logic gates (XOR, AND, OR, NOT). Explain how the Difference and Borrow outputs are computed logically and implement the circuit on the digital trainer kit.
Expected Answer:
Logic equations:
Difference = A ⊕ B ⊕ Bin (using two XOR gates).
Borrow = $(\bar{A}\cdot B) + (\bar{A} \cdot Bin) + (B\cdot Bin)$ (using AND and OR gates).
Connect gates on the trainer kit and verify the circuit against the truth table.
Test Case 2: Validation with Multi-Bit Inputs
Question: Implement a 4-bit binary subtractor using Full Subtractors on the digital trainer kit. Test the circuit for inputs A=1011 and B=1101 with initial Borrow-In = 0. Explain the step-by-step operation and results.
Answer:
Perform subtraction bit by bit:
LSB: A0=1, B0=1, Bin=0 → Difference = 0, Borrow = 0.
Next bit: A1=1, B1=0, Bin=0 → Difference = 1, Borrow = 0.
Next bit: A2=0, B2=1, Bin=0 → Difference = 1, Borrow = 1.
MSB: A3=1, B3=1, Bin=1 → Difference = 1, Borrow = 0.
Final output: Difference = 0010, Borrow = 0.

7. DESIGN AND IMPLEMENTATION OF MULTIPLEXERS

DATE:

AIM:

To Design and implement a 2 to 1 Multiplexer using logic gates.

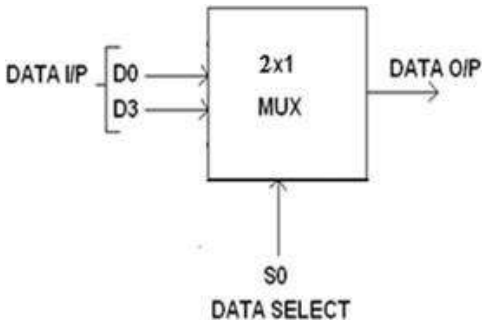
APPARATUS REQUIRED:

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.	CONNCECTING WIRES	-	32

PROCEDURE:

a. Connections are given as per the logic diagram
b. Input is given to the circuit making high '1' i.e. +5 or Vcc Supply to the 16th pin andfor low '0' i.e. GND to the 8th pin of the Gate IC
c. Depending upon the truth table, if the LED Glow represents 1 and else it represents '0'
d. Verify the truth table as given.

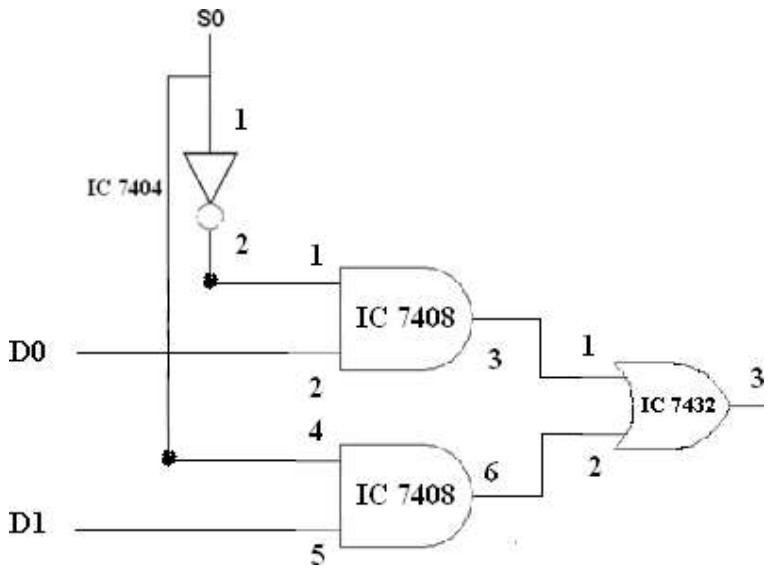
BLOCK DIAGRAM FOR 2:1 MULTIPLEXER:



MULTIPLEXER USING GATES:TRUTH TABLE:

S	Y =
0	D0
1	D1

CIRCUIT DIAGRAM FOR MULTIPLEXER:



RESULT:

Thus, the 2 to 1 Multiplexer circuit was designed using logic gates, and outputs are verifiedusing the Truth table.

Test Case 1:
S = 0, D0 = 0, D1 = 1
Output should be Y = 0 (since S = 0, output should be from D0).

Test Case 2:
S = 1, D0 = 1, D1 = 0
Output should be Y = 0 (since S = 1, output should be from D1).

Implementation using Digital Trainer Kit:

Connect the data inputs (D0, D1) to switches or inputs on the trainer kit.
Use another switch to control the select input (S).
Use the trainer's logic gates (AND, OR, NOT) to implement the above logic.
Verify the output using a LED display or an output pin to observe the result.

8. DESIGN AND IMPLEMENTATION OF DECODER

DATE:

AIM:

To design and implement a 2 to 4 Decoder using logic gates.

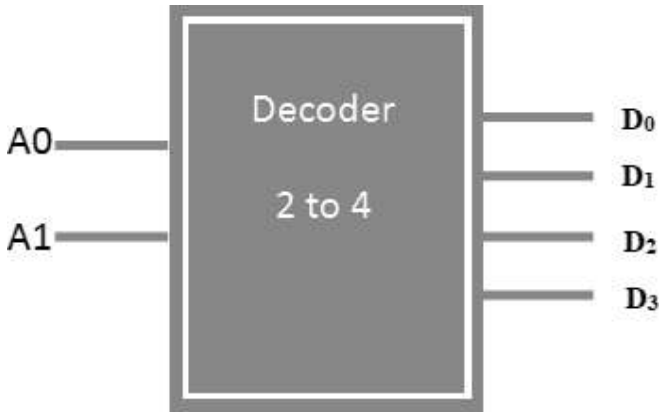
APPARATUS REQUIRED:

Sl.No.	COMPONENT	SPECIFICATION	QTY.
1.	AND GATE	IC 7408	1
2.	NOT GATE	IC7404	1
3.	IC TRAINER KIT	-	1
4.	CONNCECTING WIRES	-	few

PROCEDURE:

a. Connections are given as per the logic diagram
b. Input is given to the circuit making high '1' i.e. +5 or Vcc Supply to the 16th pin and for low '0' i.e. GND to the 8th pin of the Gate IC
c. Depending upon the truth table, if the LED Glow it represents 1 and else it represents '0'
d. Verify the truth table as given

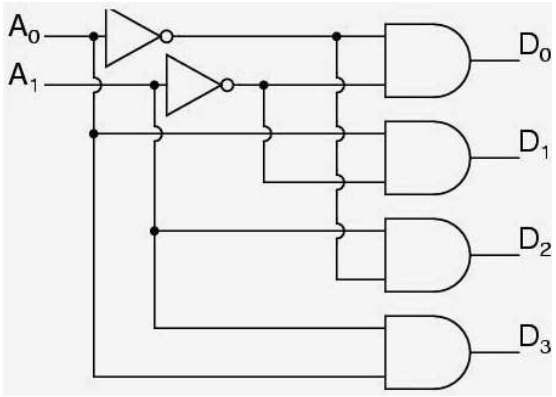
LOCK DIAGRAM FOR 2:4 DECODER:



TRUTH TABLE:

INPUT		OUTPUT			
A1	**A0**	**D0**	**D1**	**D2**	**D3**
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	0	0	0	0	1

CIRCUIT DIAGRAM FOR DECODER:



RESULT:

Thus, the 2 to 4 Decoder circuit was designed using logic gates and outputs are verifiedusing Truth table.

Test Case 1:

A = 0
Output should be Y0 = 1, Y1 = 0.

Test Case 2:

A = 1
Output should be Y0 = 0, Y1 = 1.

Implementation using Digital Trainer Kit:
Connect the input (A) to a switch or input on the trainer kit.
Use the trainer's logic gates (AND, NOT) to implement the above logic.
Observe the outputs using LEDs or output pins to check the values of Y0 and Y1 based on the input A.

9. DESIGN AND IMPLEMENTATION OF FLIP-FLOPS

DATE:

AIM:

To Design and implement Flipflops (D, SR, JK & T) using logic gates.

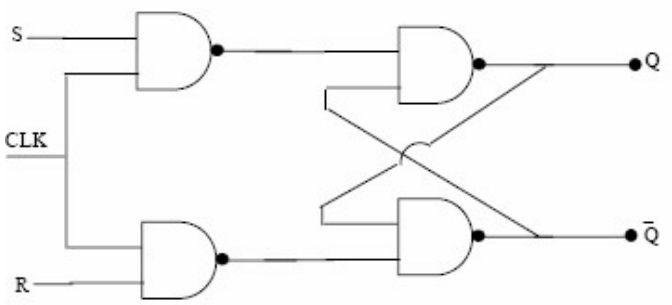
APPARATUS REQUIRED:

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.	CONNCECTING WIRES	-	32

PROCEDURE:

a. Connections are given as per the logic diagram
b. Input are given to the circuit making high '1' i.e. +5 or Vcc Supply to the 16th pin and for low '0' i.e. GND to the 8th pin of the Gate IC
c. Depending upon the truth table, if the LED Glow it represents 1 and else it represents '0'
d. Verify the truth table as given

SR FLIPFLOP
LOGIC DIAGRAM:

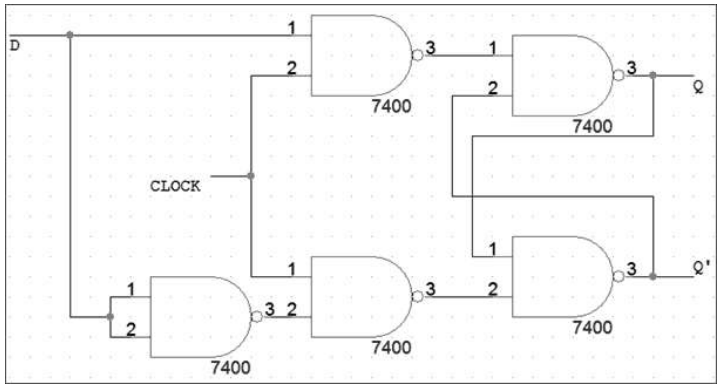


TRUTH TABLE:

Inputs			Outputs
R	S	CLK	Qt+1
0	0	↑	Qt (Previous State)
0	1	↑	1
1	0	↑	0
1	1	↑	Indeterminate state

D FLIPFLOP

LOGIC DIAGRAM:

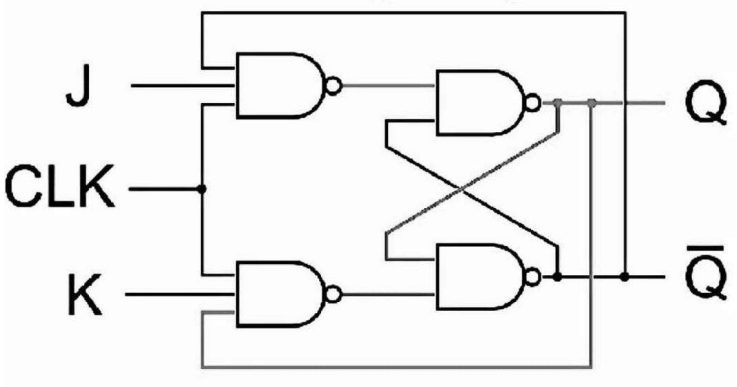


TRUTH TABLE:

Outputs		
D	CLK	Q
0	↑	0
1	↑	1

JK FLIPFLOP

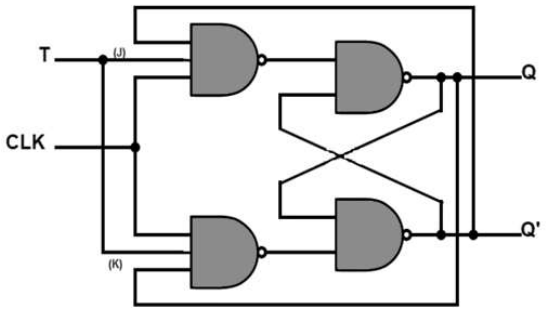
LOGIC DIAGRAM:



TRUTH TABLE:

Truth Table			
Q	**J**	**K**	**Q(t+1)**
0	x	x	Qt (PreviousState)
1	0	0	Qt (PreviousState)
1	0	1	0
1	1	0	1
1	1	1	Qt'(Toggle)

T FLIPFLOP LOGIC DIAGRAM:



TRUTH TABLE:

Q	**T**	**Q(t+1)**
0	x	Qt (Previous State)
1	0	Qt (Previous State)
1	1	Qt'(Toggle)

RESULT

Thus, the of Flipflops (D, SR, JK & T) using logic gates were Designed and verified the truth tables.

Test Case 1:

D = 1, CLK = 1 (Rising Edge) → Q should become 1, Q_bar should be 0.
D = 0, CLK = 1 (Rising Edge) → Q should become 0, Q_bar should be 1.

Test Case 2:
J = 1, K = 0 → Q should be set to 1 on the rising edge of CLK.
J = 1, K = 1 → Q should toggle on the rising edge of CLK.

Test Case 3:
S = 1, R = 0 → Q should be set to 1 on the rising edge of CLK.
S = 0, R = 1 → Q should be reset to 0 on the rising edge of CLK

Test Case 4:

T = 1 → Q should toggle on the rising edge of CLK.
T = 0 → Q should remain unchanged on the rising edge of CLK.

10. DESIGN OF 2 – BIT SYNCHRONOUS COUNTER

DATE:

AIM:

To Design and implement 2 – bit synchronous counter using Flip-Flops.

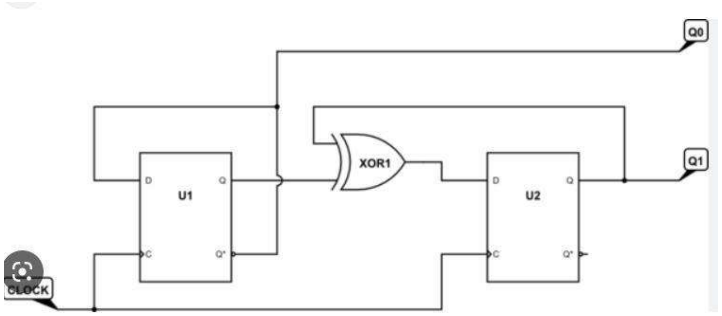
APPARATUS REQUIRED:

Sl. No	COMPONENT	SPECIFICATION	QTY.
1.	D FF	IC 7474	1
2.	OR GATE	IC 7432	1
3.	NOT GATE	IC 7404	1
4.	IC TRAINER KIT	-	1
5.	CONNECTING WIRES	-	few

PROCEDURE:

a. Connections are given as per the logic diagram
b. Input is given to the circuit making high '1' i.e. +5 or Vcc Supply to the 16th pin andfor low '0' i.e. GND to the 8th pin of the Gate IC
c. Depending upon the truth table, if the LED Glow it represents 1 and else it represents '0'
d. Verify the truth table as given

LOGIC DIAGRAM:



TRUTH TABLE:

CLK	Q_A	Q_B
1	0	0
2	0	1
3	1	0
4	1	1

RESULT:

The 2-Bit counter sequential circuit was designed, and implemented and truth tablewas verified.

Test Case 1:
Initial state Q1 = 0, Q0 = 0
Clock Cycle 1:
Q0 toggles (Q0 = 1).
Q1 stays at 0 (because Q0 was 0 before).
Result: Q1 = 0, Q0 = 1.

Clock Cycle 2:
Q0 toggles again (Q0 = 0).
Q1 toggles (Q1 = 1, because Q0 transitioned from 1 to 0).
Result: Q1 = 1, Q0 = 0.

Clock Cycle 3:
Q0 toggles (Q0 = 1).
Q1 stays at 1 (because Q0 was 0 before).
Result: Q1 = 1, Q0 = 1.

Clock Cycle 4:
Q0 toggles (Q0 = 0).
Q1 toggles (Q1 = 0, because Q0 transitioned from 1 to 0).
Result: Q1 = 0, Q0 = 0.
Expected Output Sequence:

Q1 Q0 → 00, 01, 10, 11 (and repeat).

Test Case 2:
Initial state Q1 = 1, Q0 = 1
Clock Cycle 1:
Q0 toggles (Q0 = 0).
Q1 toggles (Q1 = 0, because Q0 transitioned from 1 to 0).
Result: Q1 = 0, Q0 = 0.

Clock Cycle 2:
Q0 toggles (Q0 = 1).
Q1 stays at 0 (because Q0 was 0 before).
Result: Q1 = 0, Q0 = 1.

Clock Cycle 3:
Q0 toggles (Q0 = 0).
Q1 toggles (Q1 = 1, because Q0 transitioned from 1 to 0).
Result: Q1 = 1, Q0 = 0.

Clock Cycle 4:
Q0 toggles (Q0 = 1).
Q1 stays at 1 (because Q0 was 0 before).
Result: Q1 = 1, Q0 = 1.

Expected Output Sequence:

Q1 Q0 → 11, 10, 01, 00 (and repeat).
Implementation with Digital Trainer Kit
D Flip-Flop Counter:
Connect two D flip-flops to the clock signal.
Set D0 = Q0' to ensure Q0 toggles on every clock pulse.
Set D1 = Q0 AND Q1 to ensure Q1 toggles every time Q0 transitions from 1 to 0.
Connect LEDs or other indicators to observe the states of Q1 and Q0.

11. DESIGN AND IMPLEMENTATION OF SHIFT REGISTERS

DATE:

AIM:

To design and implement

A. Serial in serial out
B. Serial in parallel out

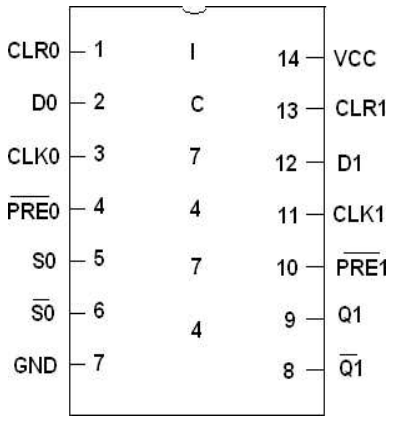
APPARATUS REQUIRED:

Sl.No.	COMPONENT	SPECIFICATION	QTY.
1.	D FLIP FLOP	IC 7474	2
2.	IC TRAINER KIT	-	1
3.	CONNECTING WIRES	-	35

PROCEDURE:

a. Connections are given as per the logic diagram
b. Input is given to the circuit making high '1' i.e., +5 or Vcc Supply to the 14^{th} pin and for low '0' i.e., GND to the 7^{th} pin of the Gate IC
c. Depending upon the truth table, if the LED Glow it represents 1 and else it represents '0'
d. Verify the truth table as given

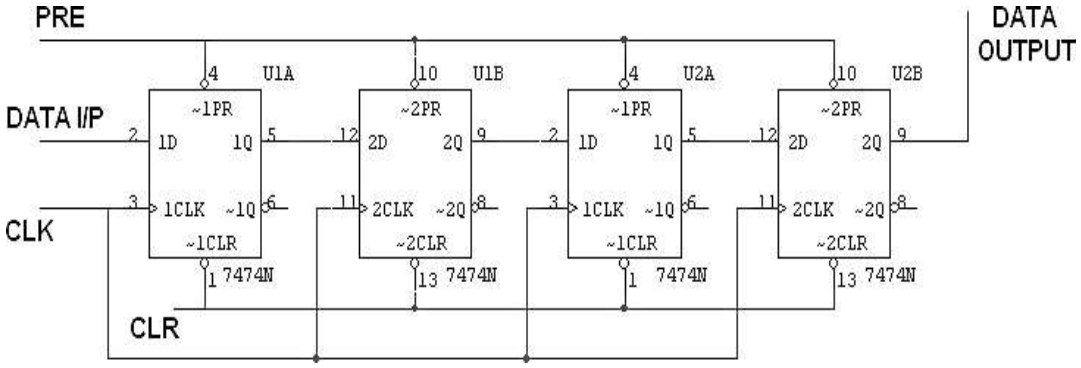
PIN DIAGRAM:



SERIAL IN SERIAL OUTTRUTH 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

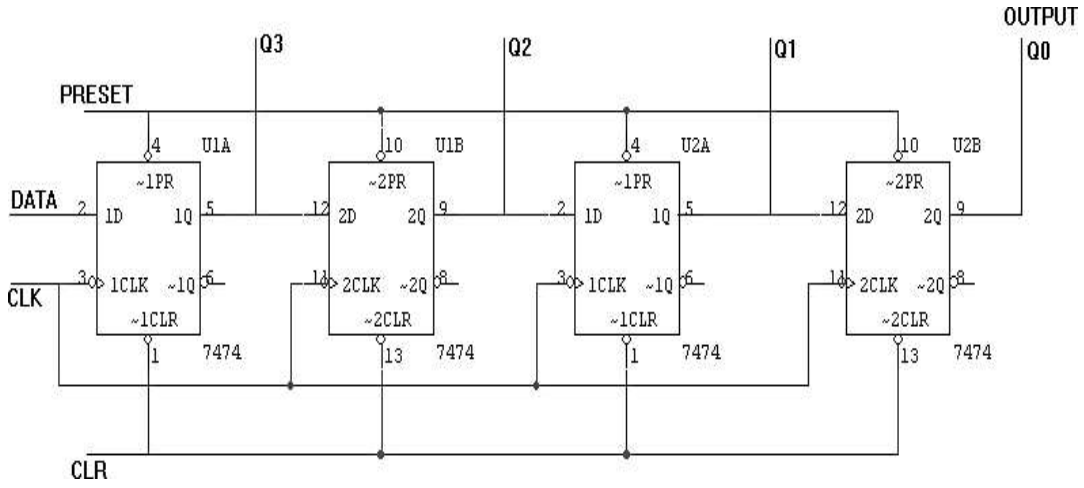
LOGIC DIAGRAM:



SERIAL IN PARALLEL OUTTRUTH 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	0
4	1	1	0	0	1

LOGIC DAIGRAM:



RESULT:

Thus, the shift register using D Flip Flop is constructed and the outputs are verifiedusing truth table.

Test Case 1:

Initial state: Q1 = 0, Q2 = 0, Q3 = 0 (all flip-flops cleared).
Data input: Serial input = 101 (to be shifted in one bit at a time).
Clock pulses: Apply 6 clock pulses.
After the first clock pulse, Q1 will become 1, and Q2 and Q3 will remain 0. After each subsequent clock pulse, the data shifts right:

After Clock 1: Q1 = 1, Q2 = 0, Q3 = 0
After Clock 2: Q1 = 1, Q2 = 0, Q3 = 0
After Clock 3: Q1 = 1, Q2 = 1, Q3 = 0
After Clock 4: Q1 = 1, Q2 = 1, Q3 = 0
After Clock 5: Q1 = 0, Q2 = 1, Q3 = 1
After Clock 6: Q1 = 1, Q2 = 0, Q3 = 1
Expected Output:

After 6 clock pulses, the output from Q3 will be 1, and the shifting sequence should be 1, 0, 1, 1, 0, 1.

Test Case 2:

Initial state: Q1 = 0, Q2 = 0, Q3 = 0.
Data input: Serial input = 110 (to be shifted in).
Clock pulses: Apply 5 clock pulses.
After each clock pulse:

After Clock 1: Q1 = 1, Q2 = 0, Q3 = 0
After Clock 2: Q1 = 1, Q2 = 1, Q3 = 0
After Clock 3: Q1 = 1, Q2 = 1, Q3 = 0
After Clock 4: Q1 = 0, Q2 = 1, Q3 = 1
After Clock 5: Q1 = 1, Q2 = 0, Q3 = 1
Expected Output:

After 5 clock pulses, the parallel output should be Q1 = 1, Q2 = 0, Q3 = 1.

12. SIMULATION OF LOGIC GATES USING VHDL

DATE:

AIM:

To write the source code and the simulation of the logic gates of AND, OR and NOT using VHDL.

SOFTWARE REQUIRED:

ModelSim Simulator

PROCEDURE:

Step1: Define the specifications and initialize the design.
Step2: Declare the name of the module by using source code.
Step3: Write the source code in VHDL.
Step4: Check the syntax and debug the errors if found, obtain the synthesis is report.
Step5: Verify the output by simulating the source code.

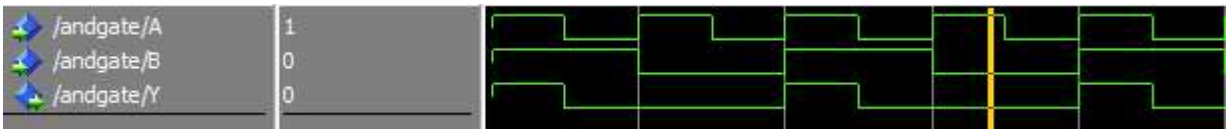
LOGIC GATES:

AND GATE:

VHDL CODE:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity andGate is
port(A : in std_logic;       -- AND gate input B : in std_logic;       -- AND gate input Y : out std_logic); --
- AND gate output
end andGate;
architecture andLogic of andGate isbegin
        Y <= A AND B;
end andLogic;
```

OUTPUT WAVEFORM:



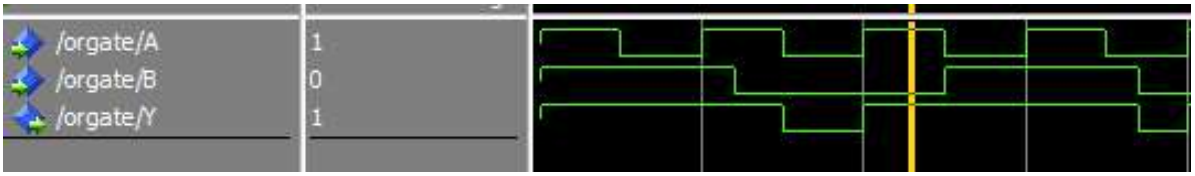
OR GATE:

VHDL CODE:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity orGate is

        port(A : in std_logic;         -- OR gate input B : in std_logic;
                -- OR gate input Y : out std_logic); -- OR gate output
end orGate;
architecture orLogic of orGate isbegin
        Y <= A OR B;
end orLogic;
```

OUTPUT:

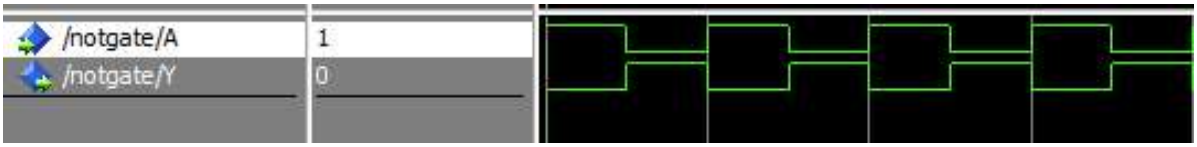


NOT GATE:

VHDL CODE:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity notGate is
port(A : in std_logic;          -- NOT gate input
Y : out std_logic);   -- NOT gate output
end notGate;
architecture notLogic of notGate isbegin
        Y <= not A;
end notLogic;
```

OUTPUT WAVEFORM:



RESULT

Thus, the VHDL code for logic gates were written, executed and verified the output.

Test Case 1: AND Gate
Test Inputs:

Test case 1: A = 0, B = 0 → Expected Output: Y = 0
Test case 2: A = 1, B = 0 → Expected Output: Y = 0
Test case 3: A = 0, B = 1 → Expected Output: Y = 0
Test case 4: A = 1, B = 1 → Expected Output: Y = 1
Expected Result: The AND gate outputs 1 only when both inputs are 1, otherwise, it outputs 0.

Test Case 2: OR Gate
Test Inputs:

Test case 1: A = 0, B = 0 → Expected Output: Y = 0
Test case 2: A = 1, B = 0 → Expected Output: Y = 1
Test case 3: A = 0, B = 1 → Expected Output: Y = 1
Test case 4: A = 1, B = 1 → Expected Output: Y = 1
Expected Result: The OR gate outputs 1 when at least one of the inputs is 1, otherwise, it outputs 0.

Test Case 3: NOT Gate
Test Inputs:

Test case 1: A = 0 → Expected Output: Y = 1
Test case 2: A = 1 → Expected Output: Y = 0
Expected Result: The NOT gate outputs the inverse of the input value (complement of A).

How to Run the Simulation:

Write the VHDL Code: Write the entity and architecture for each gate (and_gate.vhdl, or_gate.vhdl, not_gate.vhdl).
Create the Testbenches: Create the corresponding testbenches (and_gate_tb.vhdl, or_gate_tb.vhdl, not_gate_tb.vhdl).
Compile the Code: Use an HDL simulator (like ModelSim, Xilinx Vivado, or any other simulator) to compile both the logic gate code and the testbenches.
Run the Simulation: Simulate the design and observe the waveforms to verify the output of each gate.
Verify the Results: Ensure the outputs match the expected results from the truth tables provided above.

13. SIMULATION OF ADDER CIRCUITS USING VHDL

DATE:

AIM:

To write the source code and the simulate the Half adder and Full adder using VHDL.

SOFTWARE REQUIRED:

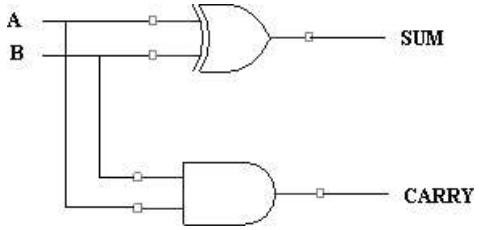
ModelSim Simulator

PROCEDURE:

Step1: Define the specifications and initialize the design.
Step2: Declare the name of the module by using source code.Step3: Write the source code in VHDL.
Step4: Check the syntax and debug the errors if found, obtain the synthesis is report.Step5: Verify the output by simulating the source code.

HALF ADDER:

LOGIC DIAGRAM:



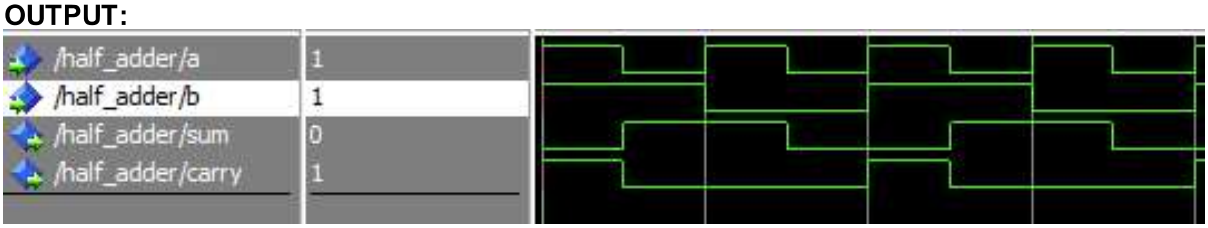
TRUTH TABLE:

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

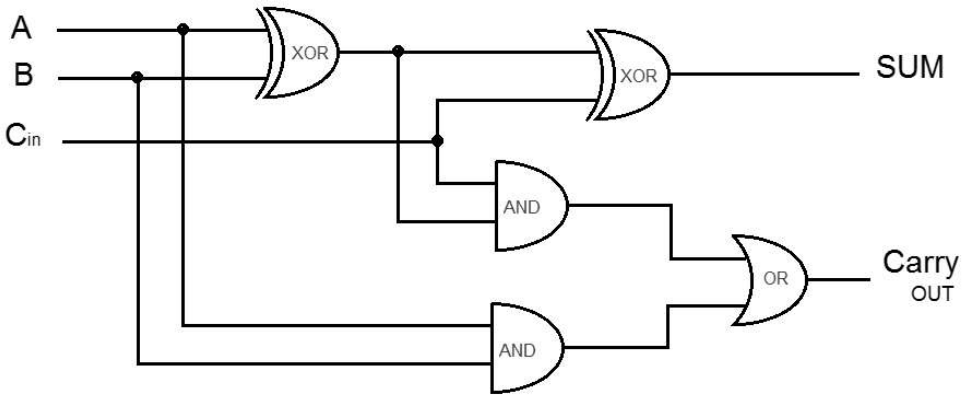
VHDL CODE:

```
Library ieee;
use ieee.std_logic_1164.all;
entity half_adder is
 port(a,b:in bit; sum,carry:out bit);end half_adder;
 architecture data of half_adder isbegin
sum<= a xor b; carry <= a
and b;
        end data;
```

OUTPUT:



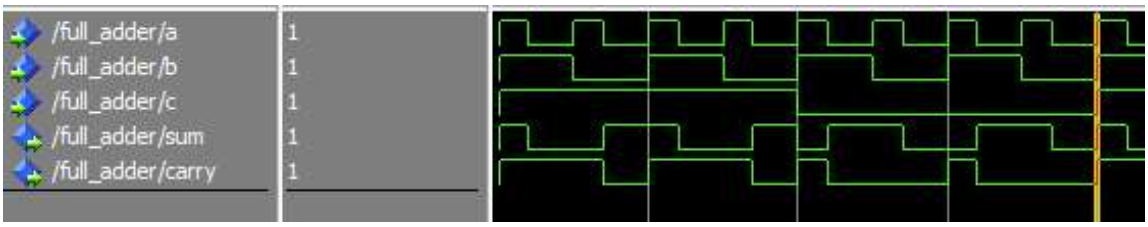
FULL ADDER Circuit Diagram:



VHDL CODE:

```
Library ieee;
use ieee.std_logic_1164.all;
entity full_adder is port (a,b,c:in bit; sum,carry:out bit);end
full_adder;
architecture data of full_adder isbegin
sum<= a xor b xor c;
carry <= ((a and b) or (b and c) or (a and c));end data;
```

Output:



RESULT

Thus, the VHDL code for half adder and full adder were written, executed and verified the output.

Test Case 1: Half Adder
Inputs:

Test case 1: A = 0, B = 0
Test case 2: A = 0, B = 1
Test case 3: A = 1, B = 0
Test case 4: A = 1, B = 1
Expected Outputs (Sum, Carry):

For A = 0, B = 0 → Sum = 0, Carry = 0
For A = 0, B = 1 → Sum = 1, Carry = 0
For A = 1, B = 0 → Sum = 1, Carry = 0
For A = 1, B = 1 → Sum = 0, Carry = 1

Test Case 2: Full Adder
Inputs:

Test case 1: A = 0, B = 0, Cin = 0
Test case 2: A = 1, B = 1, Cin = 0
Test case 3: A = 1, B = 0, Cin = 1
Test case 4: A = 1, B = 1, Cin = 1
Expected Outputs (Sum, Carry Out):

For A = 0, B = 0, Cin = 0 → Sum = 0, Carry Out = 0
For A = 1, B = 1, Cin = 0 → Sum = 0, Carry Out = 1
For A = 1, B = 0, Cin = 1 → Sum = 0, Carry Out = 1
For A = 1, B = 1, Cin = 1 → Sum = 1, Carry Out = 1

How to Simulate the Design

Write the VHDL Code: Write the VHDL code for both the Half Adder and Full Adder (entities and architectures).
Create the Testbenches: Create testbenches for each adder.
Compile the Code: Use a VHDL simulator like ModelSim, Xilinx Vivado, or Quartus to compile both the adder design and the testbenches.
Run the Simulation: Run the simulation and check the output waveform to verify the correctness of the adders.
Verify the Results: Ensure that the outputs from the testbenches match the expected values as per the truth tables above.

14. SIMULATION OF SUBTRACTOR CIRCUITS USING VHDL

DATE:

AIM:

To write the source code and simulate the Half subtractor and Full subtractor using VHDL.

SOFTWARE REQUIRED:

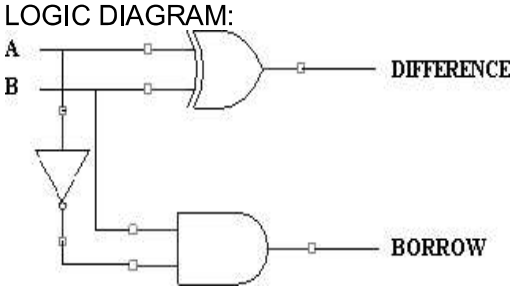
ModelSim Simulator

PROCEDURE:

Step1: Define the specifications and initialize the design.
Step2: Declare the name of the module by using source code.
Step3: Write the source code in VHDL.
Step4: Check the syntax and debug the errors if found, obtain the synthesis is report.

HALF SUBTRACTOR:

LOGIC DIAGRAM:



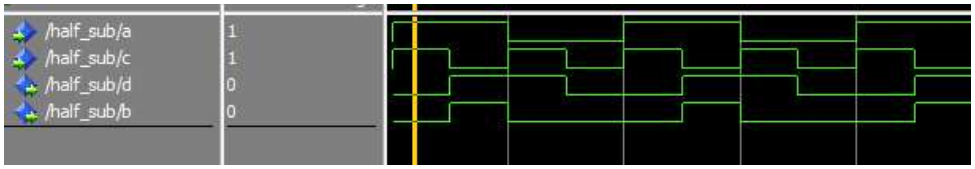
TRUTH TABLE:

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

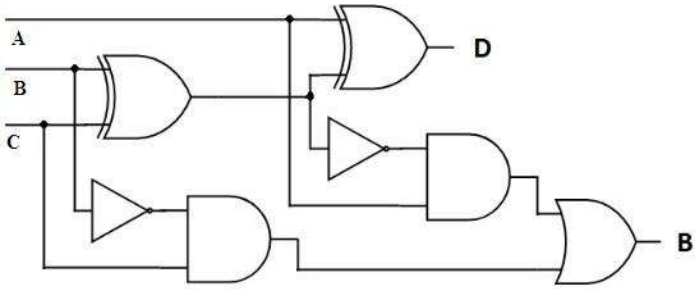
VHDL CODE:

```
  Library ieee;
use ieee.std_logic_1164.all; entity half_sub
  is port(a,c:in bit; d,b:out bit);
  end half_sub;
  architecture data of half_sub isbegin
  d<= a xor c;
     b<= (a and (not c));end data;
```

OUTPUT:



FULL SUBTRACTOR CIRCUIT DIAGRAM:



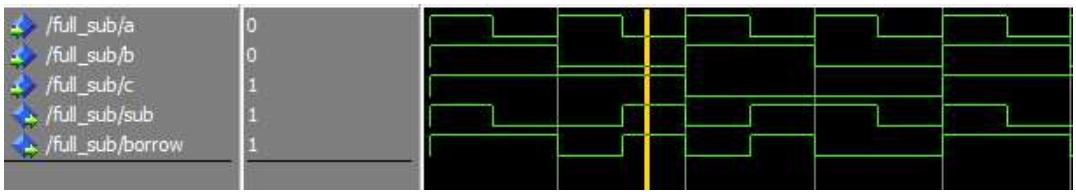
TRUTH TABLE

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

VHDL CODE:

```
 Library ieee;
use ieee.std_logic_1164.all;entity
 full_sub is
    port (a,b,c:in bit; sub,borrow:out bit);end
 full_sub;
  architecture data of full_sub isbegin
 sub<= a xor b xor c;
    borrow <= ((b xor c) and (not a)) or (b and c);end
 data;
```

Output:



RESULT

Thus, the VHDL code for half subtractor and full subtractor were written, executed andverified the output.

Test Case 1: Half Subtractor
Inputs:

Test case 1: A = 0, B = 0
Test case 2: A = 0, B = 1
Test case 3: A = 1, B = 0
Test case 4: A = 1, B = 1
Expected Outputs (Difference, Borrow):

For A = 0, B = 0 → Difference = 0, Borrow = 0
For A = 0, B = 1 → Difference = 1, Borrow = 1
For A = 1, B = 0 → Difference = 1, Borrow = 0
For A = 1, B = 1 → Difference = 0, Borrow = 0

Test Case 2: Full Subtractor
Inputs:

Test case 1: A = 0, B = 0, Bin = 0
Test case 2: A = 1, B = 1, Bin = 0
Test case 3: A = 1, B = 0, Bin = 1
Test case 4: A = 1, B = 1, Bin = 1
Expected Outputs (Difference, Borrow):

For A = 0, B = 0, Bin = 0 → Difference = 0, Borrow = 0
For A = 1, B = 1, Bin = 0 → Difference = 0, Borrow = 0
For A = 1, B = 0, Bin = 1 → Difference = 0, Borrow = 1
For A = 1, B = 1, Bin = 1 → Difference = 1, Borrow = 1

How to Simulate the Design

Write the VHDL Code: Write the VHDL code for both the Half Subtractor and Full Subtractor (entities and architectures).
Create the Testbenches: Create testbenches for each subtractor.
Compile the Code: Use a VHDL simulator like ModelSim, Xilinx Vivado, or Quartus to compile both the subtractor design and the testbenches.
Run the Simulation: Run the simulation and check the output waveform to verify the correctness of the subtractors.
Verify the Results: Ensure that the outputs from the test benches match the expected values as per the truth tables above.

15. SIMULATION OF MUTIPLEXER AND DEMULTIPLEXER USING VHDL

DATE:

AIM:

To write the source code and the simulate the 4x1 Multiplexer and 1x4 Demultiplexerusing VHDL.

SOFTWARE REQUIRED:

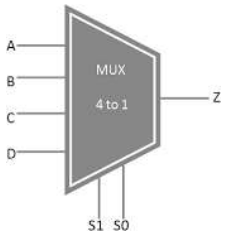
ModelSim Simulator

PROCEDURE:

Step1: Define the specifications and initialize the design.
Step2: Declare the name of the module by using source code.Step3: Write the source code in VHDL.
Step4: Check the syntax and debug the errors if found, obtain the synthesis is report.Step5: Verify the output by simulating the source code.

1x4 MUTIPLEXER:

LOGIC DIAGRAM:



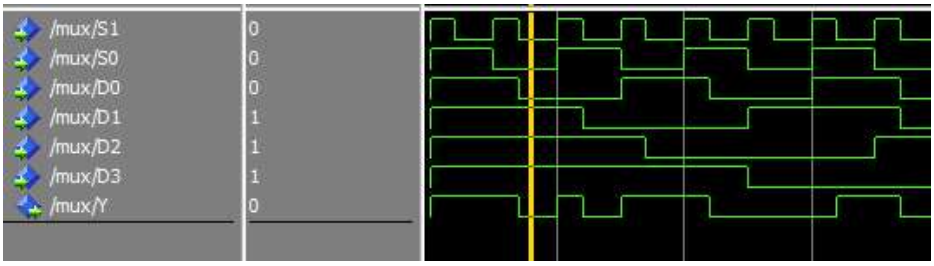
TRUTH TABLE:

S1	S0	Output
0	0	A
0	1	B

VHDL CODE:

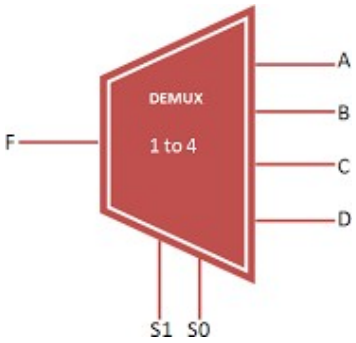
```
Library ieee;
use
  ieee.std_logic_1164.all;
  entity mux is
   port(S1,S0,D0,D1,D2,D3:in bit; Y:out bit);
end mux;
  architecture data of mux
isbegin
   Y<= (not S0 and not S1 and D0) or
      (S0 and not S1 and D1) or
      (not S0 and S1 and D2)
      or(S0 and S1 and D3);
end data;
```

OUTPUT:



4x1 DEMUTIPLEXER:

LOGIC DIAGRAM:



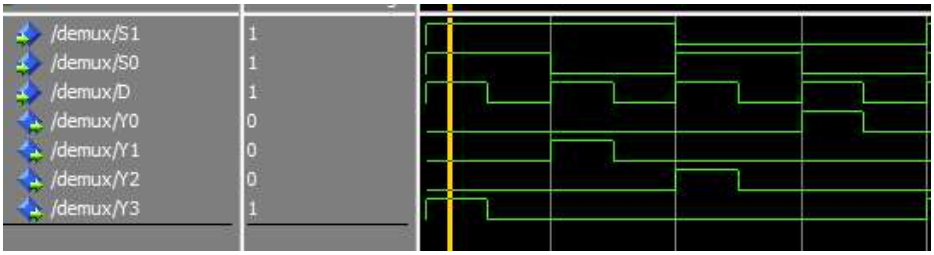
TRUTH TABLE:

INPUT			OUTPUT			
D	S0	S1	Y0	Y1	Y2	Y3
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	0	1

VHDL CODE:

```
Library ieee;
use
  ieee.std_logic_1164.all;
  entity demux is
   port(S1,S0,D:in bit; Y0,Y1,Y2,Y3:out bit);
end demux;
  architecture data of demux
isbegin
   Y0<= ((Not S0) and (Not S1) and D);
   Y1<=  ((Not S0) and S1 and D);
   Y2<=  (S0 and (Not S1) and D);
   Y3<=  (S0 and S1 and D);
end data;
```

Output:



RESULT

Thus, the VHDL code for Multiplexer and Demultiplexer were written, executed and verified the output.

Test Case 1: 4x1 Multiplexer
Test Inputs:

Test case 1: S0 = 0, S1 = 0, I0 = 1, I1 = 0, I2 = 0, I3 = 0
Test case 2: S0 = 1, S1 = 1, I0 = 0, I1 = 0, I2 = 0, I3 = 1
Expected Output:

For S0 = 0, S1 = 0 → Y = I0 = 1
For S0 = 1, S1 = 1 → Y = I3 = 1
Test Case 2: 1x4 Demultiplexer
Test Inputs:

Test case 1: D = 1, S0 = 0, S1 = 0
Test case 2: D = 1, S0 = 1, S1 = 1
Expected Output:

For S0 = 0, S1 = 0 → O0 = 1, O1 = O2 = O3 = 0
For S0 = 1, S1 = 1 → O3 = 1, O0 = O1 = O2 = 0

How to Simulate the Design

Write the VHDL Code: Write the VHDL code for both the 4x1 Multiplexer and 1x4 Demultiplexer (entities and architectures).
Create the Testbenches: Create testbenches for both the multiplexer and demultiplexer.
Compile the Code: Use a VHDL simulator such as Model Sim, Xilinx Vivado, or Quartus to compile the designs and test benches.
Run the Simulation: Execute the simulations and observe the waveform to verify that the outputs match the expected results.
Verify the Results: Compare the simulation output against the expected truth tables for each test case.

16. SIMULATION OF ENCODER AND DECODER USING VHDL

DATE:

AIM:

To write the source code and simulate the encoder and decoder using VHDL.

SOFTWARE REQUIRED:

ModelSim Simulator

PROCEDURE:

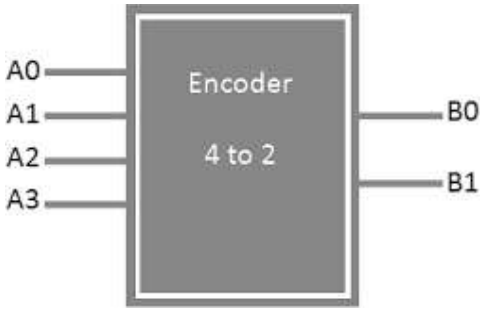
Step1: Define the specifications and initialize the design.
Step2: Declare the name of the module by using source code.Step3: Write the source code in VHDL.
Step4: Check the syntax and debug the errors if found, obtain the synthesis is report.Step5: Verify the output by simulating the source code.

4x2 ENCODER:

TRUTH TABLE:

INPUT				OUTPUT	
A3	A2	A1	A0	B1	B0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

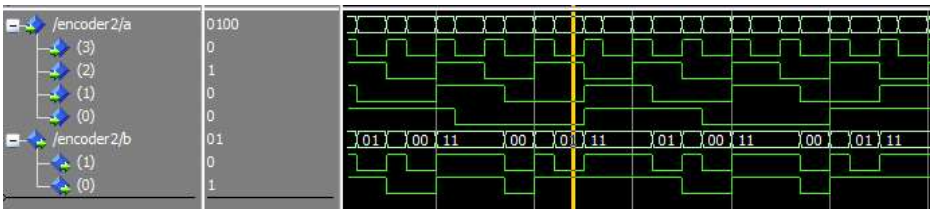
LOGIC DIAGRAM:



VHDL CODE:

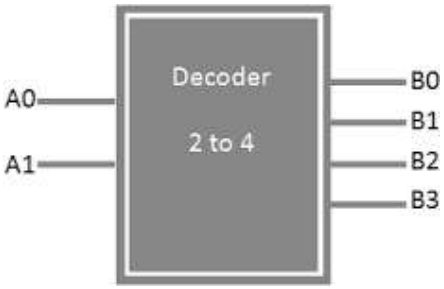
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity encoder2 is
 port(
 a : in STD_LOGIC_VECTOR(3 downto 0); b
 : out STD_LOGIC_VECTOR(1 downto 0)
 );
end encoder2;
architecture bhv of encoder2
isbegin
b(0) <= a(1) or a(2);
b(1) <= a(1) or a(3);
end bhv;
```

OUTPUT:



4x2 DECODER:

LOGIC DIAGRAM:



TRUTH TABLE:

INPUT		OUTPUT			
A1	A0	B3	B2	B1	B0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

VHDL CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity decoder2 isport( a : in STD_LOGIC_VECTOR(1 downto
 0); b : out STD_LOGIC_VECTOR(3 downto 0));
end decoder2;
architecture bhv of decoder2 is
begin
b(0) <= not a(0) and not a(1);
b(1) <= not a(0) and a(1);
b(2) <= a(0) and not a(1);
b(3) <= a(0) and a(1);
end bhv;
```

OUTPUT:

RESULT

Thus, the VHDL code for Encoder and Decoder were written, executed and verified the output.

Test Case 1: 4x2 Encoder
Test Inputs:

Test case 1: I3 = 1, I2 = 0, I1 = 0, I0 = 0
Test case 2: I3 = 0, I2 = 1, I1 = 0, I0 = 0

Expected Output:

For I3 = 1, I2 = 0, I1 = 0, I0 = 0 → A = 1, B = 1, V = 1
For I3 = 0, I2 = 1, I1 = 0, I0 = 0 → A = 1, B = 0, V = 1

Test Case 2: 2x4 Decoder
Test Inputs:

Test case 1: A = 0, B = 0
Test case 2: A = 1, B = 1

Expected Output:

For A = 0, B = 0 → O0 = 1, O1 = O2 = O3 = 0
For A = 1, B = 1 → O3 = 1, O0 = O1 = O2 = 0

How to Simulate the Design

Write the VHDL Code: Write the VHDL code for both the 4x2 Encoder and 2x4 Decoder (entities and architectures).
Create the Testbenches: Create testbenches for both the encoder and decoder.
Compile the Code: Use a VHDL simulator such as Model Sim, Xilinx Vivado, or Quartus to compile the designs and testbenches.
Run the Simulation: Execute the simulations and observe the waveform to verify that the outputs match the expected results.
Verify the Results: Compare the simulation output against the expected truth tables for each test case.