# 1. STUDY OF LOGIC GATES

**DATE :**

**AIM:**

To study about logic gates and verify their truth tables.

**APPARATUS REQUIRED:**

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

**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.

**NAND GATE:**

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

**NOR GATE:**

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

**X-OR GATE:**

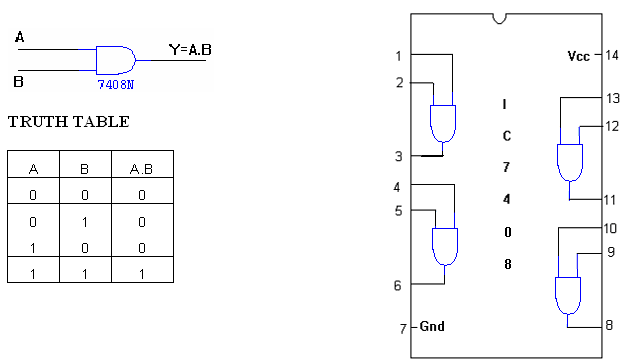
The output is high when any one of the inputs is high. The output is low when both the inputs are low and both the inputs are high.

**PROCEDURE:**

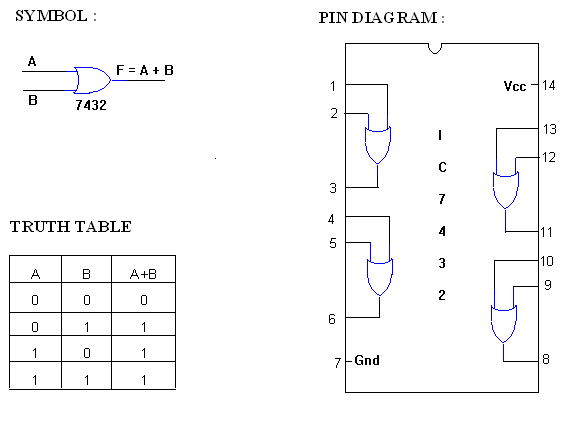
1. Connections are given as per circuit diagram.
2. Logical inputs are given as per circuit diagram.
3. Observe the output and verify the truth table.

**AND GATE:**

**SYMBOL: PIN DIAGRAM:**

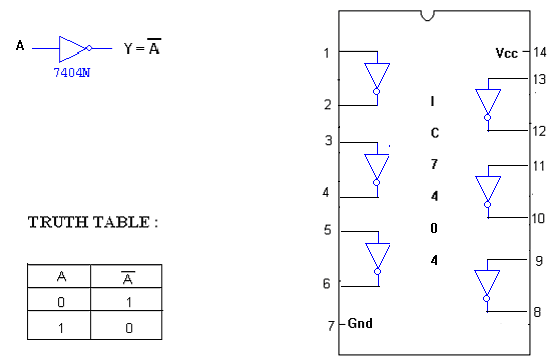


**OR GATE:**



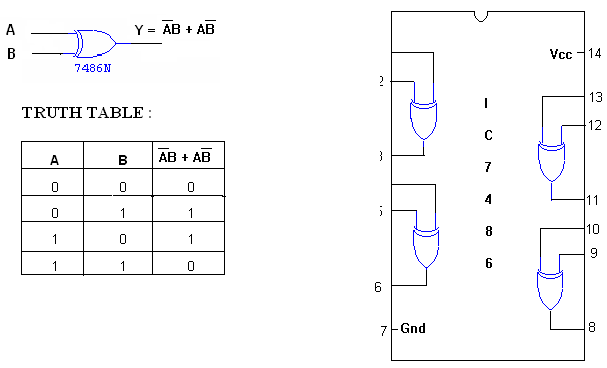
**NOT GATE:**

**SYMBOL: PIN DIAGRAM:**



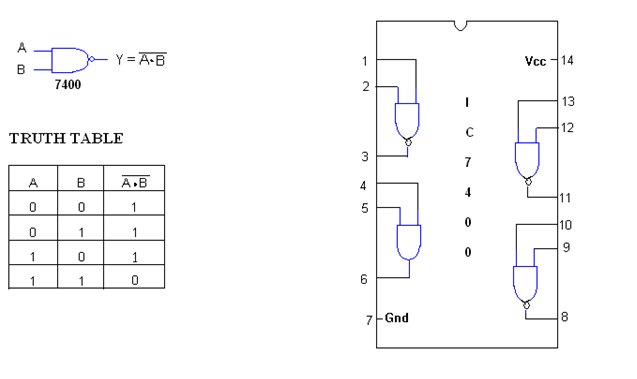
**X-OR GATE :**

**SYMBOL : PIN DIAGRAM :**

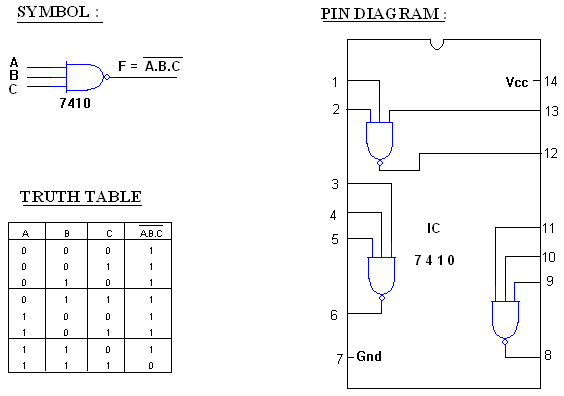
****

**2-INPUT NAND GATE:**

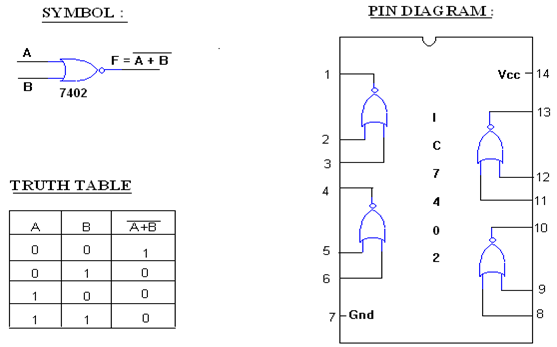
**SYMBOL: PIN DIAGRAM:**

****

**3-INPUT NAND GATE :**

****

**NOR GATE:**

****

**RESULT:**

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

**VIVA QUESTIONS**

1. What are Logic gates?
2. What are the basic digital logic gates?
3. Which gates are called as the universal gates? Why are they called so
4. What is meant by bit & byte?
5. List the different number systems
6. Define binary logic?
7. How will you use a 4 input NAND gate as a 2 input NAND gate?
8. How will you use a 4 input NOR gate as a 2 input NOR gate?
9. Show that the NAND connection is not associative
10. What happens when all the gates is a two level AND

### 2. VERIFICATION OF BOOLEAN THEOREMS USING LOGIC GATES

**DATE :**

**AIM:**

Toverify Boolean theorem 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 |  |  |

**THEORY:**

"Boolean theorem, Property or law," describes how differing variables relate to each other in a system of numbers. One of these properties is known as the *commutative property*, and it applies equally to addition and multiplication. In essence, the commutative property tells us we can reverse the order of variables that are either added together or multiplied together without changing the truth of the expression Along with the commutative properties of addition and multiplication, we have the *associative property*, again applying equally well to addition and multiplication. This property tells us we can associate groups of added or multiplied variables together with parentheses without altering the truth of the equations. Lastly, we have the *distributive property*, illustrating how to expand a Boolean expression formed by the product of a sum, and in reverse shows us how terms may be factored out of Boolean sums-of-products

**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 represent 1 and else it                  represents ‘0’

d. Verify the truth table as given

e. Repeat the procedure steps for different theorems.

**BOOLEAN THEOREM:**

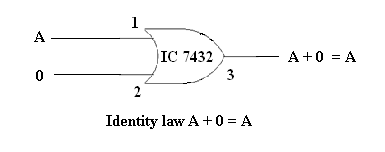
|  |  |  |
| --- | --- | --- |
| **POSTULATES** | **I** | **II** |
| **Identity** | A + 0 = A | A . 1 = A |
| **Commulative** | A + B = B + A | AB = BA |
| **Distributive** | A ( B + C ) = AB + AC | A + BC = (A + B) (A + C) |
| **Complement** | A + A’ =1 | A.A’ = 0 |
| **Idempotency** | A + A = A | A.A = 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** |

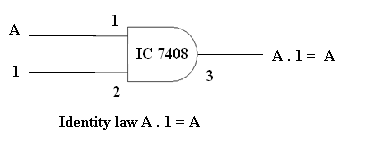
**LOGIC DIAGRAM**

****

**TRUTH TABLE**

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

**LOGIC DIAGRAM**

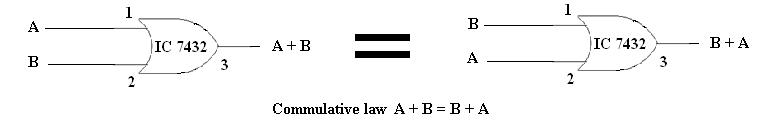
****

1. **COMMULATIVE**

**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 |

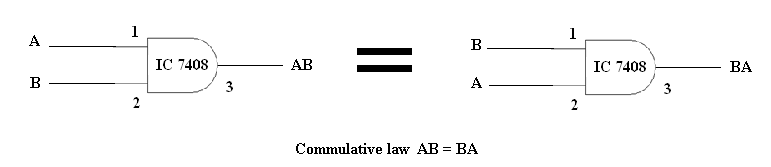
**LOGIC DIAGRAM**

****

**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**

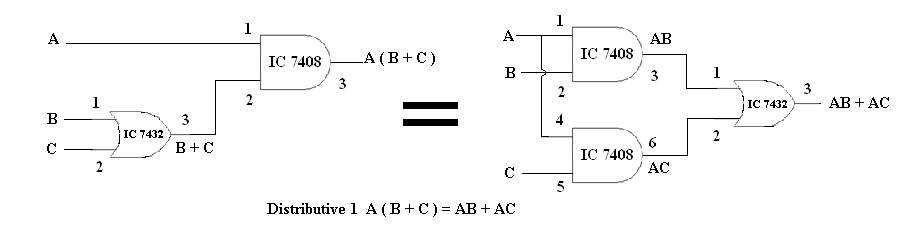
****

1. **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 |

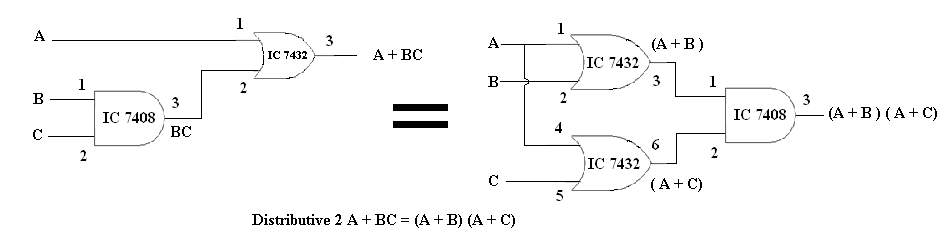
**LOGIC DIAGRAM**

****

**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**

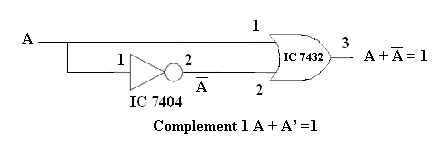
****

1. **COMPLEMENT**

**TRUTH TABLE**

|  |  |  |
| --- | --- | --- |
| **I/P** | | **O/P** |
| A | A' | A+A' |
| 0 | 1 | 1 |
| 1 | 0 | 1 |

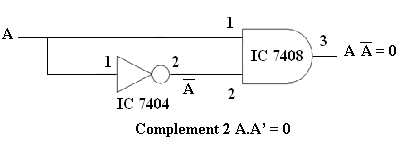
**LOGIC DIAGRAM**

****

**TRUTH TABLE**

|  |  |  |
| --- | --- | --- |
| **I/P** | | **O/P** |
| A | A' | AA' |
| 0 | 1 | 0 |
| 1 | 0 | 0 |

**LOGIC DIAGRAM**

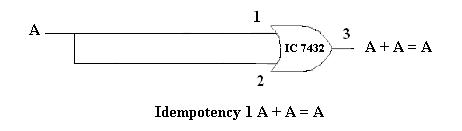
****

1. **IDEMPOTENCY**

**TRUTH TABLE**

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

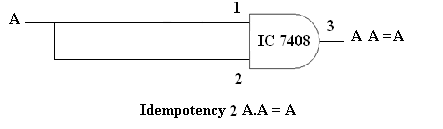
**LOGIC DIAGRAM**

****

**TRUTH TABLE**

|  |  |  |
| --- | --- | --- |
| **I/P** | | **O/P** |
| A | A | AA |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

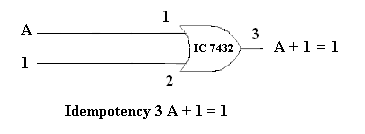
**LOGIC DIAGRAM**

****

**TRUTH TABLE**

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

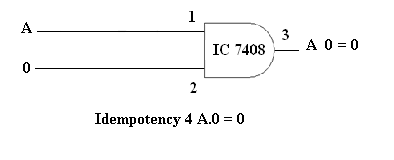
**LOGIC DIAGRAM**

****

**TRUTH TABLE**

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

**LOGIC DIAGRAM**

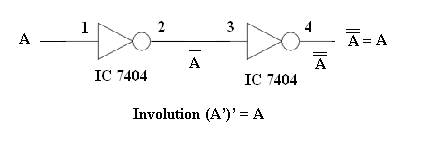
****

1. **INVOLUTION**

**TRUTH TABLE**

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

**LOGIC DIAGRAM**

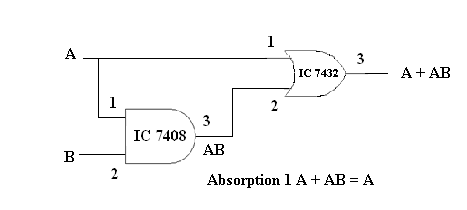
****

1. **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** |

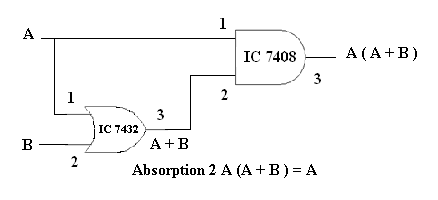
**LOGIC DIAGRAM**

****

**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** |

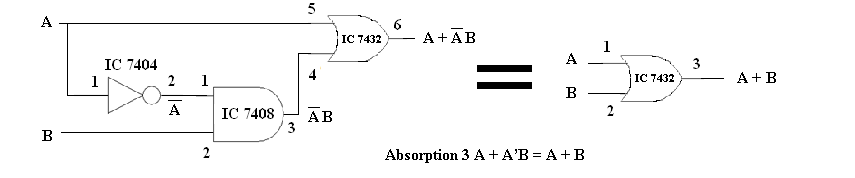
**LOGIC DIAGRAM**

****

**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 |

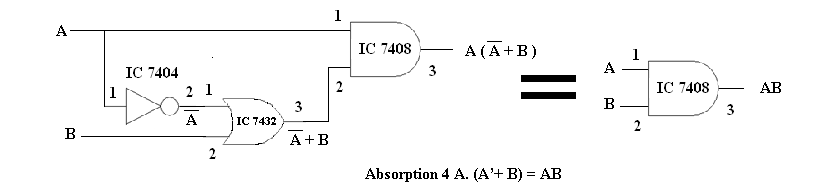
**LOGIC DIAGRAM**

****

**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**

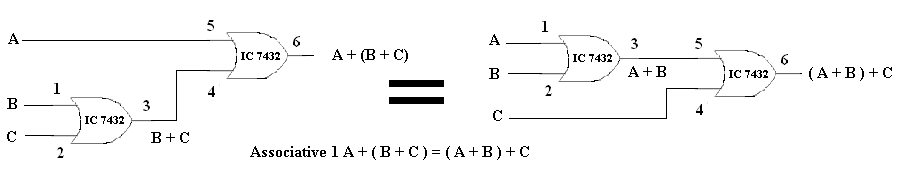
****

1. **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 |

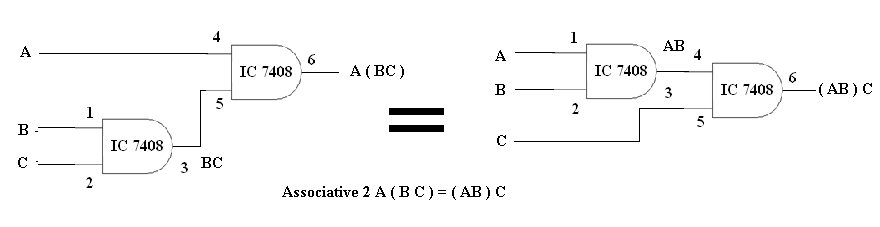
**LOGIC DIAGRAM**

****

**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**

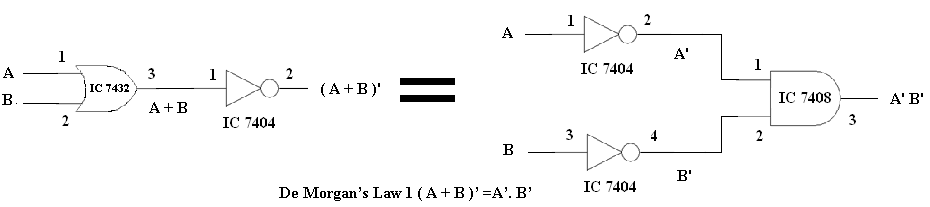
****

1. **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 |

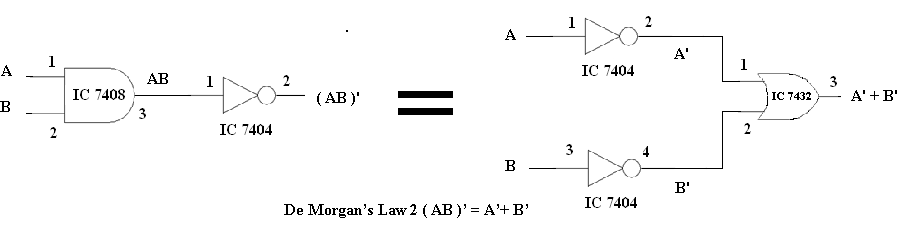
**LOGIC DIAGRAM**

****

**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**

****

**RESULT:**

The different theorems of Boolean algebra are verified.

**VIVA QUESTIONS**

1. Prove that x + x = x
2. Define Associative Law and Distributive law?
3. Define Boolean algebra?
4. Define Boolean Function?
5. Define Demorgan’s Theorem?
6. Prove that x + 1 = 1
7. Prove that x + xy = x
8. State consensus theorem
9. What is meant by Duality Theorem?
10. Find the complement of x+yz

### 3. IMPLEMENTATION OF COMBINATION CIRCUIT FOR AN ARBITARY FUNCTION

**DATE :**

**AIM:**

To simplify the given function and to implement it using logic gates.

**Y= (AC)’ + ABC + AC’**

**APPARATUS REQUIRED:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sl.No. | COMPONENT | SPECIFICATION | QTY. |
| 1. | NOT GATE | IC 7404 | 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 that accept signals from inputs and generate signals to outputs

**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 represent 1 and else it                  represents ‘0’

d. Verify the truth table as given

**SIMPLIFICATION**

**Y = (AC)’ + ABC + AC’**

**= A’ + C’ + ABC + A C’ [Using De morgans law]**

**= A’ + C’ (1 + A) + ABC**

**= A’ + C’ + ABC [ Using Idempotency 1 + A = 1 ]**

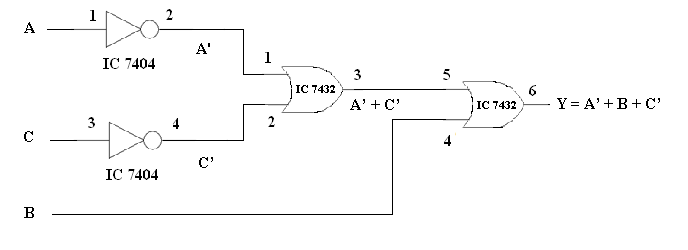
**= A’ + ABC + C’**

**= (A’ + A ) ( A’ + BC) + C’ [Distributive law A + BC = (A + B) ( A + C )]**

**= A’ + BC + C’ [Distributive law A + BC = (A + B) ( A + C )]**

**Y = A’ + B + C’**

**LOGIC DIAGRAM:**

****

**TRUTH TABLE:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **I/P** | | |  | | | **O/P** |
| **A** | **B** | **C** | **A'** | **C'** | **A' + C'** | **A' + B + C'** |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |

**RESULT:**

Thus the given Arbitrary function was reduced and verified using truth table.

**VIVA QUESTIONS**

1. List the truth table of the function F = x y + x y’ + y ’z
2. Simplify using boolean laws and draw its logic diagram.

(a). XYZ + X’Y + XYZ’

(b). XY + YZ + XY’Z

(c). AB’ + ABD + ABD’ + A’C’D’ + A’BC’

(d). BD + BCD’ + AB’C’D’

**4. DESIGN AND IMPLEMENTATION OF ADDERS AND SUBRACTORS USING LOGIC GATES**

**AIM: DATE :**

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 |
| 3. | IC TRAINER KIT | - | 1 |
| 4. | CONNECTING WIRES | - | 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.

**PROCEDURE:**

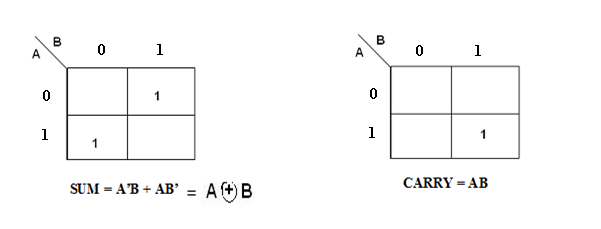
1. Connections are given as per the logic diagram
2. Input are given to the circuit making high ‘1’ i.e. +5 or Vcc Supply to the 14th pin and for low ‘0’ i.e. GND to the 7th pin of the Gate IC
3. Depending upon the truth table, if the LED Glow it represent 1 and else it                  represents ‘0’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

#### HALF ADDER:

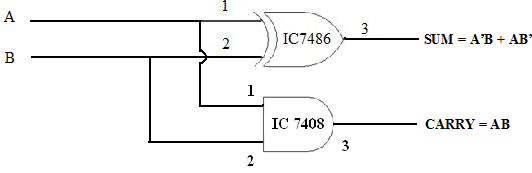
#### TRUTH TABLE:

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **SUM** | **CARRY** |
| **0** | **0** | **0** | **0** |
| **0** | **1** | **1** | **0** |
| **1** | **0** | **1** | **0** |
| **1** | **1** | **0** | **1** |

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

****

**LOGIC DIAGRAM FOR HALF ADDER:**

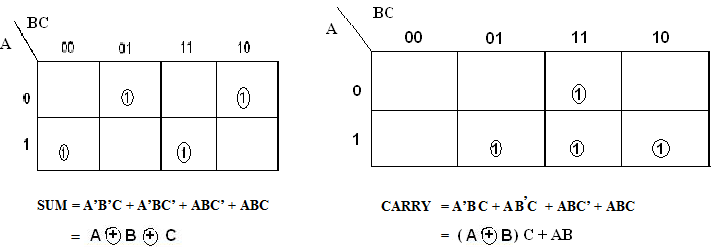
****

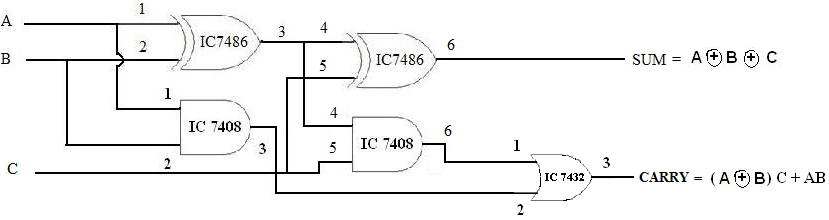
**FULLADDER:**

#### 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 K-Map for CARRY**

****

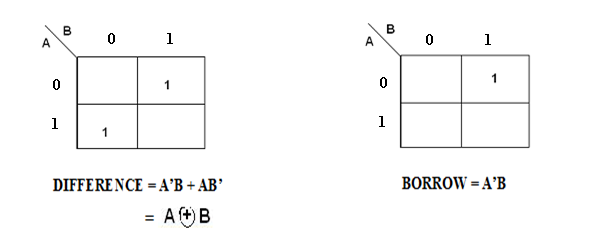
**LOGIC DIAGRAM FOR FULL ADDER: **

**HALF SUBRACTOR:**

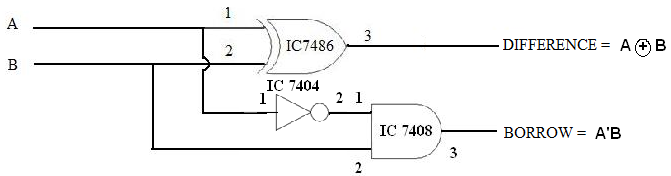
**TRUTH TABLE**

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **DIFFERENCE** | **BORROW** |
| **0** | **0** | **0** | **0** |
| **0** | **1** | **1** | **1** |
| **1** | **0** | **1** | **0** |
| **1** | **1** | **0** | **0** |

**K-Map for DIFFERENCE K-Map for BORROW**

****

**LOGIC DIAGRAM FOR HALF SUBRACTOR:**

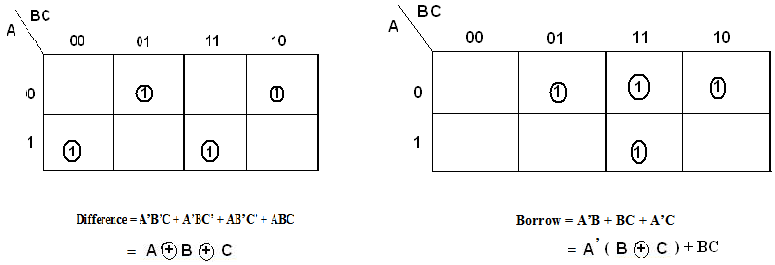
****

**FULL SUBRACTOR:**

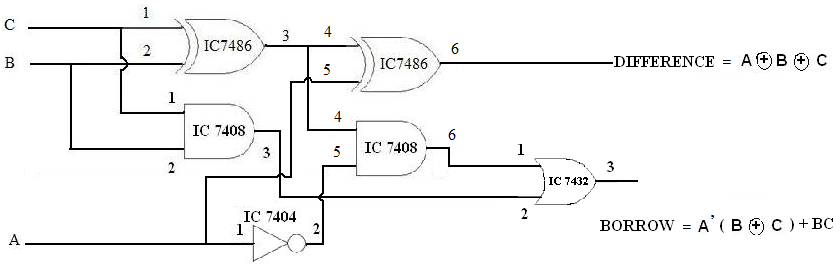
**TRUTH TABLE:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 K-Map for BORROW**

****

**LOGIC DIAGRAM FOR FULL SUBRACTOR:**

****

**RESULT:**

Thus the design for adders and subractors was done and verified successfully using Truth table.

**VIVA QUESTIONS**

1. What is meant by Half – Adder?
2. What is meant by Half – Subtractor?
3. What is meant by Full – Adder?
4. What is meant by Full – Subtractor?
5. How will you build a full adder using 2 half adders and an OR gate?
6. Draw 4 bit binary parallel adder
7. Give the four elementary operations for addition and subtraction
8. **DESIGN AND IMPLEMENTATION OF PARITY CHECKER AND PARITY GENERATOR USING LOGIC GATES.**

**AIM: DATE :**

To construct and verify the truth table of parity checker and parity generator using logic gates.

**APPARATUS REQUIRED:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sl.No. | COMPONENT | SPECIFICATION | QTY. |
| 1. | EX-OR GATE | IC 7486 | 1 |
| 2. | NOT GATE | IC 7404 | 1 |
| 3. | IC TRAINER KIT | - | 1 |
| 4. | CONNECTING WIRES | - | - |

**THEORY:**

A parity bit is used for detecting errors during transmission of binary information. A parity bit is an extra bit included with a binary message to make the number is either even or odd. The message including the parity bit is transmitted and then checked at the receiver ends for errors. An error is detected if the checked parity bit doesn’t correspond to the one transmitted. The circuit that generates the parity bit in the transmitter is called a ‘parity generator’ and the circuit that checks the parity in the receiver is called a ‘parity checker’.

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

**PROCEDURE:**

1. Connections are given as per the logic diagram
2. 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
3. Depending upon the truth table, if the LED Glow it represent 1 and else it                  represents ‘0’
4. Verify the truth table as given

**TRUTH TABLE:**

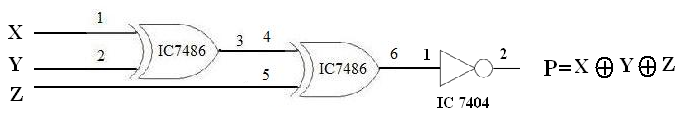
**PARITY GENERATOR PARITY CHECKER**

|  |  |  |  |
| --- | --- | --- | --- |
| 3- Bit Message | | | Odd parity Generated **(P)** |
| **X** | **Y** | **Z** |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

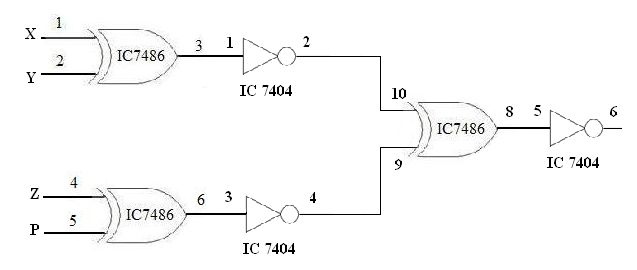
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4-Bit received | | | | Parity error check **(C)** |
| **X** | **Y** | **Z** | **P** |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**LOGIC DIAGRAM:**

**PARITY GENERATOR:**

****

**PARITY CHECKER:**

****

**RESULT:**

Thus the design for Parity checker and parity generator using Logic Gates was done and verified successfully.

**VIVA QUESTIONS**

1. What is meant by parity generator?
2. What is meant by parity checker?
3. What is parity?
4. What is even parity ?
5. What is odd parity ?

### 6. DESIGN AND IMPLEMENTATION OF CODE CONVERTERS USING LOGIC GATES

**DATE :**

**AIM:**

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

1. Binary to gray code converter
2. 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 |

**THEORY:**

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

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

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

**PROCEDURE:**

1. Connections are given as per the logic diagram
2. 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
3. Depending upon the truth table, if the LED Glow it represent 1 and else it                  represents ‘0’
4. Verify the truth table as given

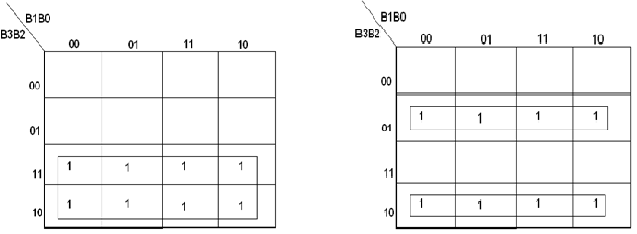
**A. DESIGN FOR BINARY TO GRAY CODE CONVERTOR**

**TRUTH TABLE:**

**| Binary input | Gray code output |**

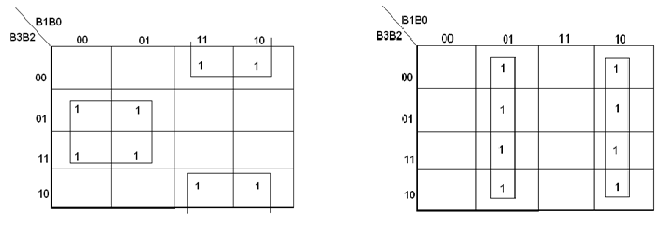
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **B3** | **B2** | **B1** | **B0** | **G3** | **G2** | **G1** | **G0** |
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **0** | **1** | **0** | **0** | **0** | **1** |
| **0** | **0** | **1** | **0** | **0** | **0** | **1** | **1** |
| **0** | **0** | **1** | **1** | **0** | **0** | **1** | **0** |
| **0** | **1** | **0** | **0** | **0** | **1** | **1** | **0** |
| **0** | **1** | **0** | **1** | **0** | **1** | **1** | **1** |
| **0** | **1** | **1** | **0** | **0** | **1** | **0** | **1** |
| **0** | **1** | **1** | **1** | **0** | **1** | **0** | **0** |
| **1** | **0** | **0** | **0** | **1** | **1** | **0** | **0** |
| **1** | **0** | **0** | **1** | **1** | **1** | **0** | **1** |
| **1** | **0** | **1** | **0** | **1** | **1** | **1** | **1** |
| **1** | **0** | **1** | **1** | **1** | **1** | **1** | **0** |
| **1** | **1** | **0** | **0** | **1** | **0** | **1** | **0** |
| **1** | **1** | **0** | **1** | **1** | **0** | **1** | **1** |
| **1** | **1** | **1** | **0** | **1** | **0** | **0** | **1** |
| **1** | **1** | **1** | **1** | **1** | **0** | **0** | **0** |

**K-Map for G3: K-Map for G2:**

****

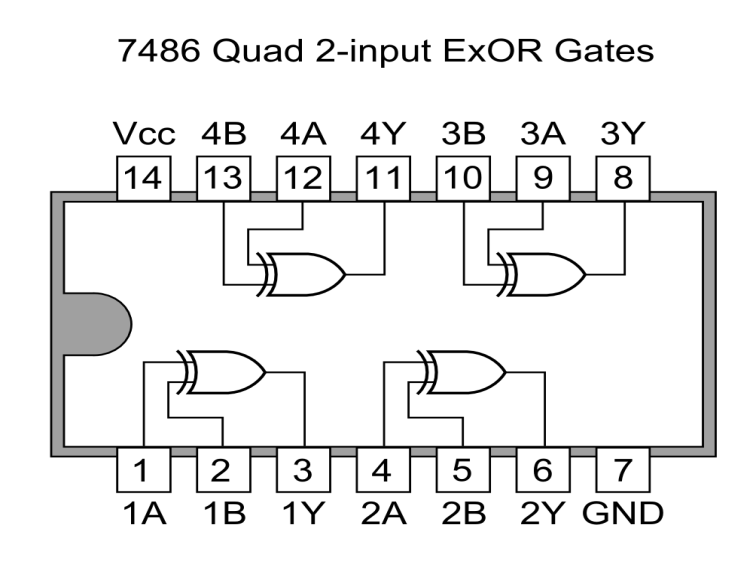
**G3 = B3 **

**K-Map for G1: K-Map for G0:**

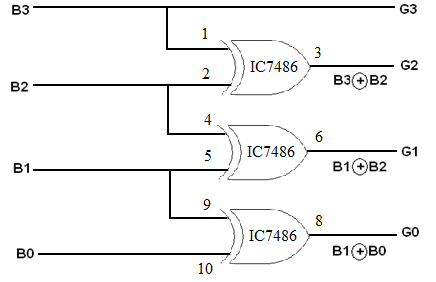
****

** **

**PIN DIAGRAM:**

****

**LOGIC DIAGRAM FOR BINARY TO GRAY CODE CONVERTOR:**

****

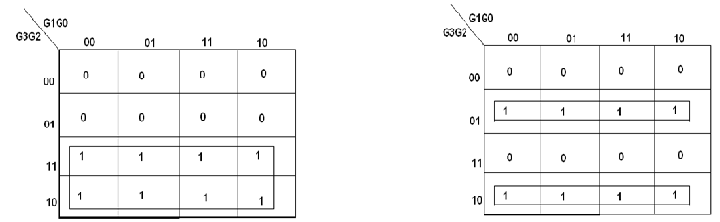
**B. DESIGN FOR GRAY TO BINARY CODE CONVERTOR:**

**TRUTH TABLE:**

**| Gray Code | Binary Code |**

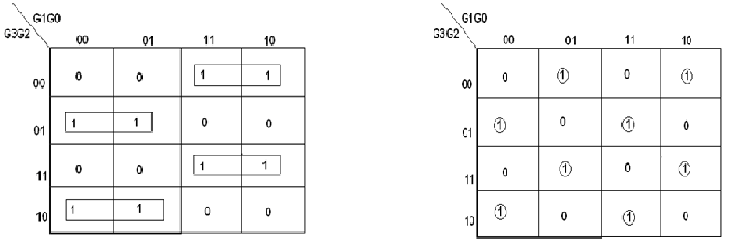
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **G3** | **G2** | **G1** | **G0** | **B3** | **B2** | **B1** | **B0** |
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **0** | **1** | **0** | **0** | **0** | **1** |
| **0** | **0** | **1** | **1** | **0** | **0** | **1** | **0** |
| **0** | **0** | **1** | **0** | **0** | **0** | **1** | **1** |
| **0** | **1** | **1** | **0** | **0** | **1** | **0** | **0** |
| **0** | **1** | **1** | **1** | **0** | **1** | **0** | **1** |
| **0** | **1** | **0** | **1** | **0** | **1** | **1** | **0** |
| **0** | **1** | **0** | **0** | **0** | **1** | **1** | **1** |
| **1** | **1** | **0** | **0** | **1** | **0** | **0** | **0** |
| **1** | **1** | **0** | **1** | **1** | **0** | **0** | **1** |
| **1** | **1** | **1** | **1** | **1** | **0** | **1** | **0** |
| **1** | **1** | **1** | **0** | **1** | **0** | **1** | **1** |
| **1** | **0** | **1** | **0** | **1** | **1** | **0** | **0** |
| **1** | **0** | **1** | **1** | **1** | **1** | **0** | **1** |
| **1** | **0** | **0** | **1** | **1** | **1** | **1** | **0** |
| **1** | **0** | **0** | **0** | **1** | **1** | **1** | **1** |

**K-Map for B3 K-Map for B2**

****

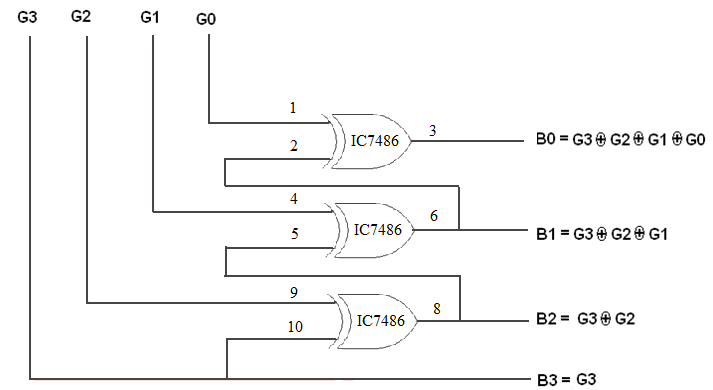
**B3 = G3 **

**K-Map for B1 K-Map for B0**

****

** **

**LOGIC DIAGRAM FOR GRAY TO BINARY CODE CONVERTOR:**

****

**RESULT:**

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

**VIVA QUESTIONS**

1. What are code converters ?
2. What is a gray code ?
3. Why gray codes are called as cyclic codes ?
4. Convert (0011011101)2 to gray code
5. Convert (001000110)g to binary code

**7. DESIGN AND IMPLEMENTATION OF MULTIPLEXERS AND DEMULTIPLEXERS**

**AIM: DATE :**

To Design and implementation Multiplexer, Demultiplexer, using logic gates and MSI device.

**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. | MULTIPLEXER IC | IC 74151 | 1 |
| 5. | DEMULTIPLEXER IC | IC 74155 | 1 |
| 2. | IC TRAINER KIT | - | 1 |
| 3. | CONNCECTING WIRES | - | 32 |

**THEORY:**

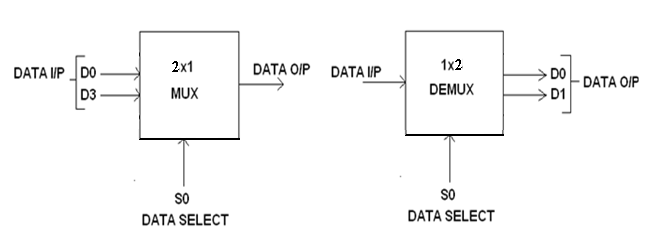
**MULTIPLEXER:** Multiplexer means transmitting a large number of information units over a smaller number of channels or lines. A digital multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular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

**DEMULTIPLEXER:** The function of Demultiplexer is in contrast to multiplexer function. It takes information from one line and distributes it to a given number of output lines. For this reason, the demultiplexer is also known as a data distributor. Decoder can also be used as demultiplexer.In the 1: 4 demultiplexer circuit, the data input line goes to all of the AND gates. The data select lines enable only one gate at a time and the data on the data input line will pass through the selected gate to the associated data output line.

**PROCEDURE:**

1. Connections are given as per the logic diagram
2. 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
3. Depending upon the truth table, if the LED Glow it represent 1 and else it                  represents ‘0’
4. Verify the truth table as given

**BLOCK DIAGRAM FOR 4:1 MULTIPLEXER AND DEMULTIPLEXER:**

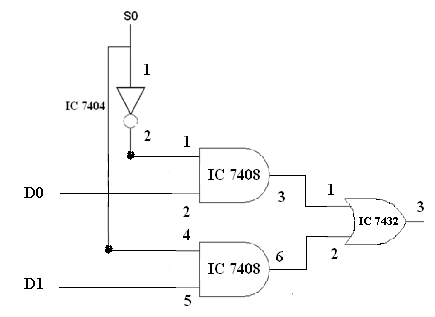


**A.MULTIPLEXER USING GATES:**

**TRUTH TABLE:**

|  |  |
| --- | --- |
| **S0** | **Y = OUTPUT** |
| 0 | D0 |
| 1 | D1 |

**CIRCUIT DIAGRAM FOR MULTIPLEXER:**

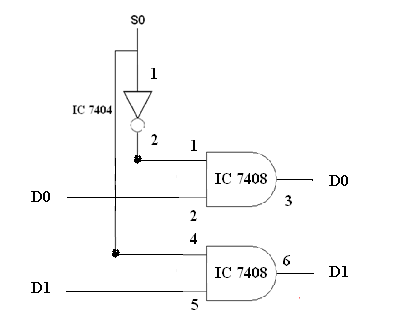
****

**DEMULTIPLEXER USING LOGIC GATRES:**

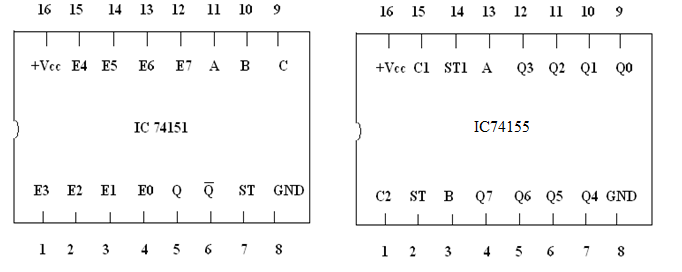
**TRUTH TABLE:**

|  |  |  |  |
| --- | --- | --- | --- |
| **INPUT** | | **OUTPUT** | |
| **S0** | **I/P(X)** | **D0** | **D1** |
| **0** | **0** | **0** | **0** |
| **0** | **1** | **1** | **0** |
| **1** | **0** | **0** | **0** |
| **1** | **1** | **0** | **1** |

**CIRCUIT DIAGRAM FOR DEMULTIPLEXER:**

****

**B.MULTIPLEXER & DEMULTIPLEXER USING MSI DEVICE:**

**PIN DIAGRAM:**

**TRUTH ABLE:**

**MULTIPLEXER DEMULTIPLEXER:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ST** | **SELECT SIGNAL** | | | **OUTPUT** |
| **C** | **B** | **A** | **Q** |
| **H** | **X** | **X** | **X** | **L** |
| **L** | **0** | **0** | **0** | **E0** |
| **L** | **0** | **0** | **1** | **E1** |
| **L** | **0** | **1** | **0** | **E2** |
| **L** | **0** | **1** | **1** | **E3** |
| **L** | **1** | **0** | **0** | **E4** |
| **L** | **1** | **0** | **1** | **E5** |
| **L** | **1** | **1** | **0** | **E6** |
| **L** | **1** | **1** | **1** | **E7** |

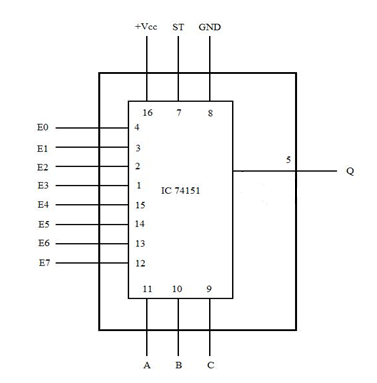
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SELECT SIGNAL** | | | | **OUTPUT** | | | | | | | |
| **ST** | **C** | **B** | **A** | **Q0** | **Q1** | **Q2** | **Q3** | **Q4** | **Q5** | **Q6** | **Q7** |
| **1** | **X** | **X** | **X** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** |
| **0** | **0** | **0** | **0** | **0** | **1** | **1** | **1** | **1** | **1** | **1** | **1** |
| **0** | **0** | **0** | **1** | **1** | **0** | **1** | **1** | **1** | **1** | **1** | **1** |
| **0** | **0** | **1** | **0** | **1** | **1** | **0** | **1** | **1** | **1** | **1** | **1** |
| **0** | **0** | **1** | **1** | **1** | **1** | **1** | **0** | **1** | **1** | **1** | **1** |
| **0** | **1** | **0** | **0** | **1** | **1** | **1** | **1** | **0** | **1** | **1** | **1** |
| **0** | **1** | **0** | **1** | **1** | **1** | **1** | **1** | **1** | **0** | **1** | **1** |
| **0** | **1** | **1** | **0** | **1** | **1** | **1** | **1** | **1** | **1** | **0** | **1** |
| **0** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **0** |

**ST**-Strobe

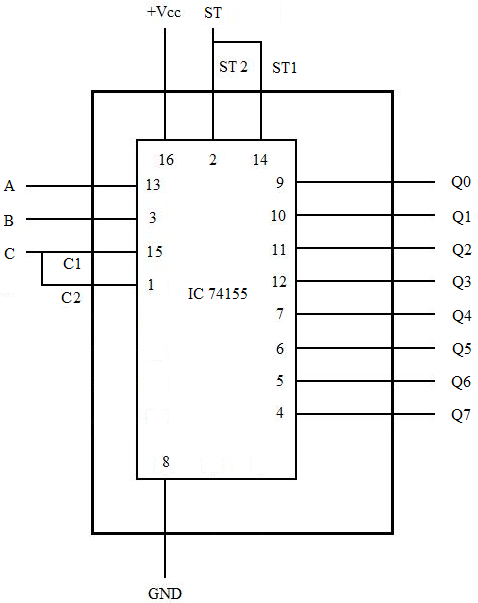
**X**- Don’t care

**L**- Low Voltage, **H**- High Voltage

**IMPLEMENTATION DIAGRAM FOR MULTIPLEXER USING IC74151:**

****

**IMPLEMENTATION DIAGRAM FOR DEMULTIPLEXER USING IC74151:**

****

**RESULT:**

Thus the Multiplexer, Demultiplexer circuit was doen using logic gates and MSI device. And the outputs are verified using Truth table.

**VIVA QUESTIONS**

1. What is a multiplexer ?
2. What is a demultiplexer
3. Give the other name for mux and demux. Why are they called so?
4. What is the significance of selector lines in mux and demux ?
5. How many selection lines are required to design a 2n: 1 mux?

**8. DESIGN AND IMPLEMENTATION OF MAGNITUDE COMPARATOR**

**DATE :**

**AIM:**

To design and implement 8 – bit magnitude comparator using MSI device.

**APPARATUS REQUIRED:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sl.No. | COMPONENT | SPECIFICATION | QTY. |
| 1. | 4-BIT MAGNITUDE COMPARATOR | IC 7485 | 1 |
| 2. | IC TRAINER KIT | - | 1 |
| 3. | CONNECTING WIRES | - |  |

**THEORY:**

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

A = A3 A2 A1 A0

B = B3 B2 B1 B0

The equality of the two numbers and B is displayed in a combinational circuit designated by the symbol (A=B).A>B indicates A greater than B, then inspect the relative magnitude of pairs of significant digits starting from most significant position. A is 0 and that of B is 0. We have A<B, the sequential comparison can be expanded as

A>B = A3B31 + X3A2B21 + X3X2A1B11 + X3X2X1A0B01

A<B = A31B3 + X3A21B2 + X3X2A11B1 + X3X2X1A01B0

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

A = B = (A3 + B3) (A2 + B2) (A1 + B1) (A0 + B0)

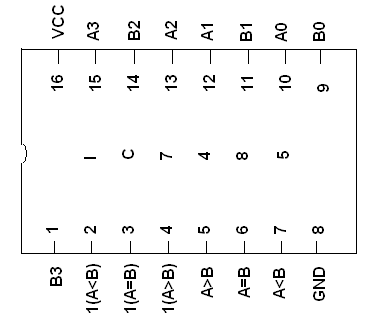
🡳 🡳 🡳 🡳

x3 x2 x1 x0

**PROCEDURE:**

1. Connections are given as per the logic diagram
2. 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
3. Depending upon the truth table, if the LED Glow it represent 1 and else it                  represents ‘0’
4. Verify the truth table as given

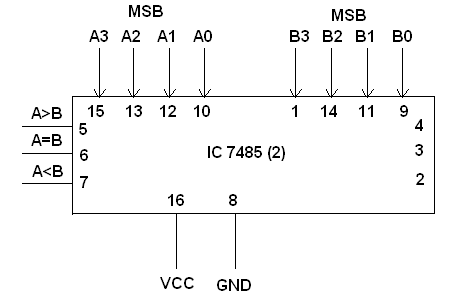
**PIN DIAGRAM FOR IC 7485:**

****

**TRUTH TABLE:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **A>B** | **A=B** | **A<B** |
| **0 0 0 0** | **0 0 0 0** | **0** | **1** | **0** |
| **0 0 0 1** | **0 0 0 0** | **1** | **0** | **0** |
| **0 0 0 0** | **0 0 0 1** | **0** | **0** | **1** |

#### LOGIC DIAGRAM 8 BIT MAGNITUDE COMPARATOR:



**RESULT:**

Thus the comparator circuir using MSI device was done and verified successfully using truth table.

**VIVA QUESTIONS**

1. What is a comparator ?
2. What are the three possible outputs in a acomparator ?
3. Which gate works like a comparator ?
4. Design a two bit magnitude comparator
5. Name the MSI device used as a magnitude comparator

**9. DESIGN AND IMPLEMENTATION OF SHIFT REGISTERS**

**AIM: DATE :**

To design and implement

1. Serial in serial out
2. Serial in parallel out
3. Parallel 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 |

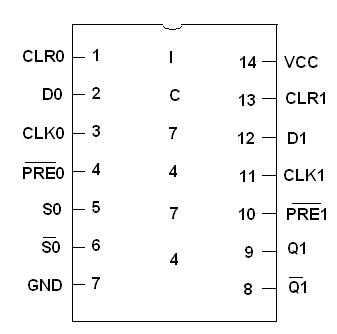
**THEORY:**

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

**PROCEDURE:**

1. Connections are given as per the logic diagram
2. 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
3. Depending upon the truth table, if the LED Glow it represent 1 and else it                  represents ‘0’
4. Verify the truth table as given

**PIN DIAGRAM:**

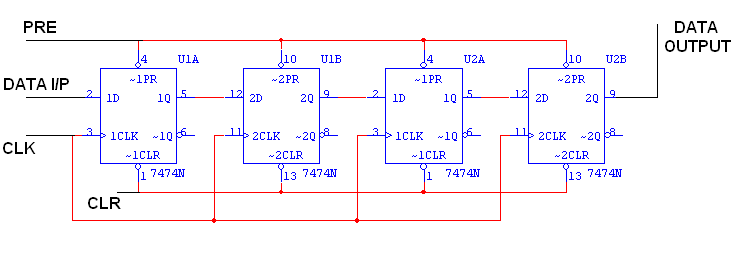
****

**A.SERIAL IN SERIAL OUT**

**TRUTH TABLE:**

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

**LOGIC DIAGRAM :**

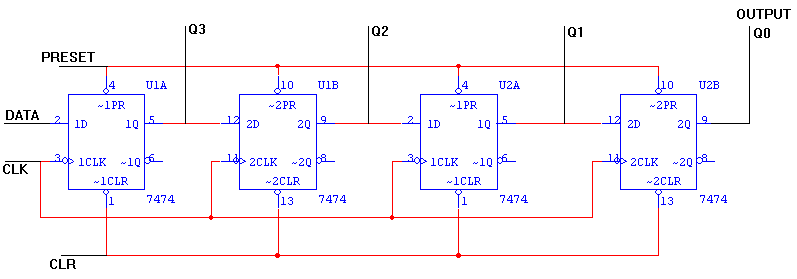
****

**B.SERIAL IN PARALLEL OUT**

**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** | **0** |
| **4** | **1** | **1** | **0** | **0** | **1** |

**LOGIC DAIGRAM:**

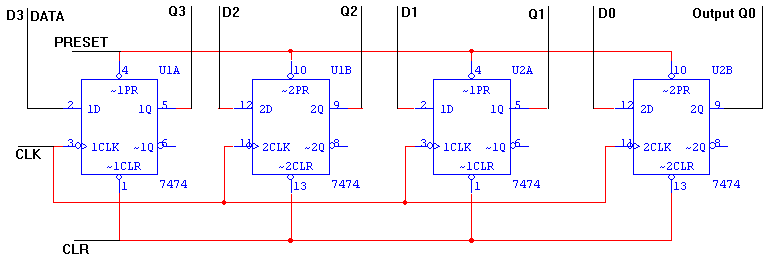
****

**C. PARALLEL IN PARALLEL OUT:**

**TRUTH TABLE:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CLK** | **DATA INPUT** | | | | **OUTPUT** | | | |
| **DA** | **DB** | **DC** | **DD** | **QA** | **QB** | **QC** | **QD** |
| **1** | **1** | **0** | **0** | **1** | **1** | **0** | **0** | **1** |
| **2** | **1** | **0** | **1** | **0** | **1** | **0** | **1** | **0** |

**LOGIC DIAGRAM:**

****

**RESULT:**

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

**VIVA QUESTIONS**

1. What are registers ?
2. What are shift registers ?
3. What is a flip flop ?
4. What are the types of shift registers ?
5. What are the two ways of shifting a data ?

**10.DESIGN AND IMPLEMENTATION OF SYNCHRONOUS AND ASYNCHRONOUS COUNTER.**

**AIM: DATE :**

To design and implement synchronous and asynchronous counter.

**APPARATUS REQUIRED:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sl.No. | COMPONENT | SPECIFICATION | QTY. |
| 1. | JK FLIP FLOP | IC 7476 | 2 |
| 2. | AND GATE | IC 7408 | 1 |
| 3. | IC TRAINER KIT | - | 1 |
| 4. | CONNECTING WIRES | - | 30 |

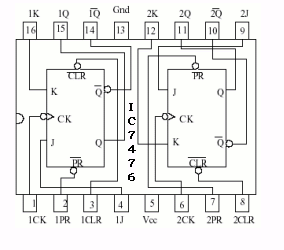
**THEORY:**

Counters belong to the class of sequential circuits. The important characteristic of these circuits is memory. When clock pulses are applied to a counter, the counter progresses from state to state and the final output of the flip-flop in the counter indicates the pulse count.There are two types of counters, (a) asynchronous counters and (b) synchronous counters. In asynchronous counters all flip-flops are not clocked at the same time, while in synchronous counters all flip-flops are clocked simultaneously.

**PROCEDURE:**

1. Connections are given as per the logic diagram
2. 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
3. Depending upon the truth table, if the LED Glow it represent 1 and else it                  represents ‘0’
4. Verify the truth table as given

**PIN DIAGRAM:**

****

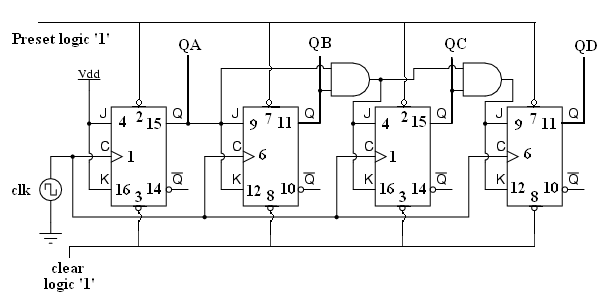
**TRUTH TABLE:**

**SYNCHRONOUS UP COUNTER SYNCHRONOUS DOWN COUNTER**

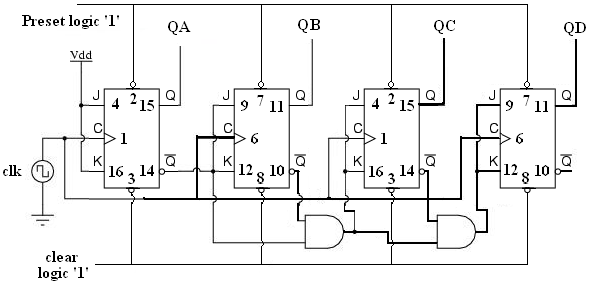
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CLK** | **QD** | **QC** | **QB** | **QA** |
| **0** | 0 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 1 |
| **2** | 0 | 0 | 1 | 0 |
| **3** | 0 | 0 | 1 | 1 |
| **4** | 0 | 1 | 0 | 0 |
| **5** | 0 | 1 | 0 | 1 |
| **6** | 0 | 1 | 1 | 0 |
| **7** | 0 | 1 | 1 |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

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

**LOGIC DIAGRAM FOR SYNCHRONOUS UP COUNTER:**

****

**LOGIC DIAGRAM FOR SYNCHRONOUS DOWN COUNTER:**



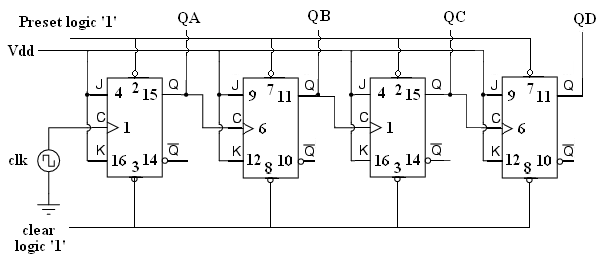
**TRUTH TABLE:**

**ASYNCHRONOUS UP COUNTER ASYNCHRONOUS DOWN COUNTER**

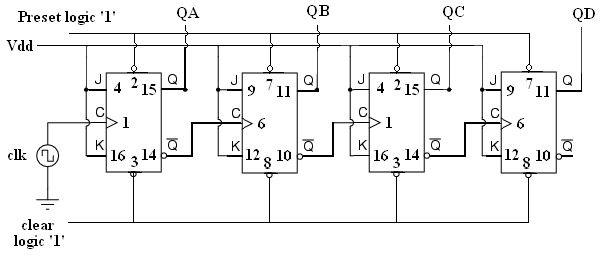
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CLK** | **QD** | **QC** | **QB** | **QA** |
| **0** | 0 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 1 |
| **2** | 0 | 0 | 1 | 0 |
| **3** | 0 | 0 | 1 | 1 |
| **4** | 0 | 1 | 0 | 0 |
| **5** | 0 | 1 | 0 | 1 |
| **6** | 0 | 1 | 1 | 0 |
| **7** | 0 | 1 | 1 |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

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

**LOGIC DIAGRAM FOR ASYNCHRONOUS UP COUNTER:**



**LOGIC DIAGRAM FOR ASYNCHRONOUS DOWN COUNTER:**

****

**RESULT:**

Thus the counter circuit is constructed and the outputs are verified using truth table.

**VIVA QUESTIONS**

1. What is a counter ?
2. What is a synchronous counter ?
3. What is a asynchronous cpounter ?
4. What are the different types of counters ?
5. Why JK flip flops are used in counter design ?

**11 CONSTRUCTION AND VERIFICATION OF 4 BIT RIPPLE COUNTER AND MOD 10/MOD 12 RIPPLE COUNTER**

**AIM:**

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

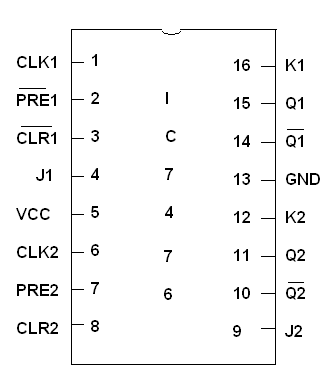
**APPARATUS REQUIRED:**

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

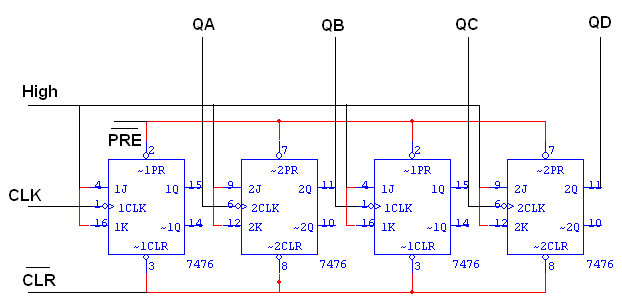
**THEORY:**

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

**PIN DIAGRAM FOR IC 7476:**

****

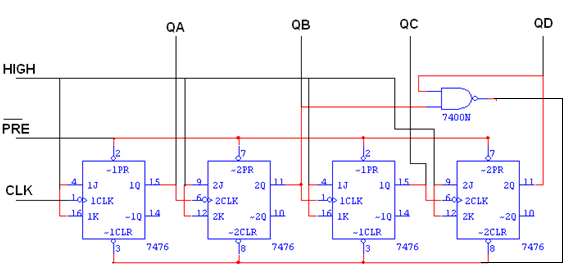
**LOGIC DIAGRAM FOR 4 BIT RIPPLE COUNTER:**



**TRUTH TABLE:**

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

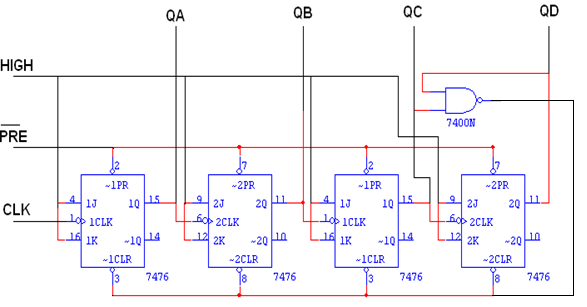
**LOGIC DIAGRAM FOR MOD - 10 COUNTER:**



**TRUTH TABLE:**

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

**LOGIC DIAGRAM FOR MOD - 12 COUNTER:**



**TRUTH TABLE:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CLK** | **QA** | **QB** | **QC** | **QD** |
| **0** | **0** | **0** | **0** | **0** |
| **1** | **1** | **0** | **0** | **0** |
| **2** | **0** | **1** | **0** | **0** |
| **3** | **1** | **1** | **0** | **0** |
| **4** | **0** | **0** | **1** | **0** |
| **5** | **1** | **0** | **1** | **0** |
| **6** | **0** | **1** | **1** | **0** |
| **7** | **1** | **1** | **1** | **0** |
| **8** | **0** | **0** | **0** | **1** |
| **9** | **1** | **0** | **0** | **1** |
| **10** | **0** | **1** | **0** | **1** |
| **11** | **1** | **1** | **0** | **1** |
| **12** | **0** | **0** | **0** | **0** |

**PROCEDURE:**

1. Connections are given as per circuit diagram.
2. Logical inputs are given as per circuit diagram.
3. Observe the output and verify the truth table.

**RESULT:**

Thus the counter circuit is constructed and the outputs are verified using truth table.

**12 DESIGN AND IMPLEMENTATION OF ADDERS AND SUBTRACTORS USING MSI DEVICE**

**DATE :**

**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. | IC | IC 7483 | 1 |
| 2. | EX-OR GATE | IC 7486 | 1 |
| 3. | NOT GATE | IC 7404 | 1 |
| 3. | IC TRAINER KIT | - | 1 |
| 4. | CONNECTING WIRES | - | 40 |

**THEORY:**

**4 BIT BINARY ADDER:**A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of next full adder in chain. The augends bits of ‘A’ and the addend bits of ‘B’ are designated by subscript numbers from right to left, with subscript 0 denoting the least significant bits. The carries are connected in chain through the full adder. The input carry to the adder is C0 and it ripples through the full adder to the output carry C4.

**4 BIT BINARY SUBTRACTOR:**The circuit for subtracting A-B consists of an adder with inverters, placed between each data input ‘B’ and the corresponding input of full adder. The input carry C0 must be equal to 1 when performing subtraction.

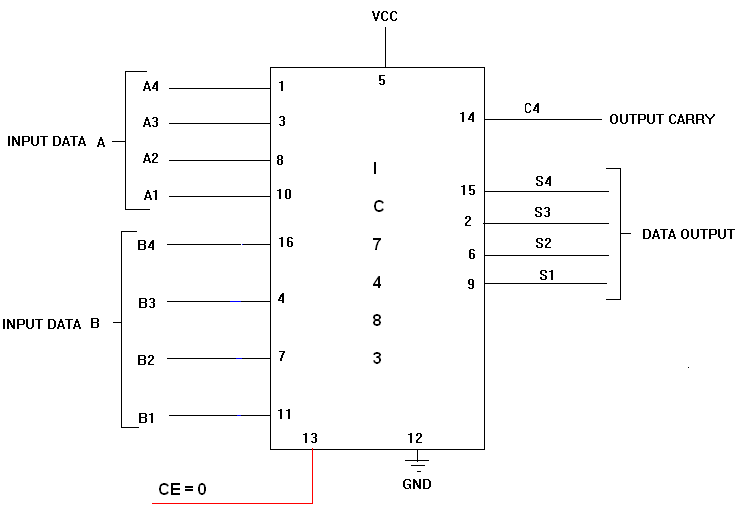
**PROCEDURE:**

1. Connections are given as per the logic diagram
2. Input are given to the circuit making high ‘1’ i.e. +5 or Vcc Supply to the 5th pin and for low ‘0’ i.e. GND to the 12th pin of the Gate IC
3. Depending upon the truth table, if the LED Glow it represent 1 and else it                  represents ‘0’
4. Verify the truth table as given

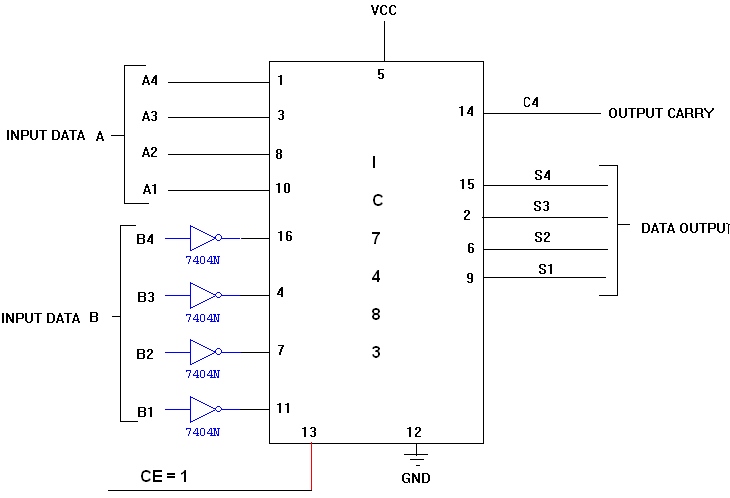
#### TRUTH TABLE:

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

#### LOGIC DIAGRAM FOR 4-BIT BINARY ADDER:



#### LOGIC DIAGRAM FOR 4-BIT BINARY SUBTRACTOR:

****

**RESULT:**

Thus the design for adders and subractors using MSI device was done and verified successfully.

**13. SIMULATION OF COMBINATIONAL CIRCUITS USING VERILOG HDL.**

**AIM:**

To develop the source code and the simulation for the following combinational logic circuit using VERILOG.

a)Logic Gates

b)Adders and subractors

c)Encoder and Decoder

d)Multiplexer and Demultiplexer.

**SOFTWARE REQUIRED:**

* Xilinx – Project Navigator
* 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 VERILOG.

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: OR GATE:**

# LOGIC DIAGRAM: TRUTH TABLE: LOGICDIAGRAM TRUTH TABLE

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Y=AB** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Y=A+B** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

 ****

**NOT GATE: NAND GATE:**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Y=(AB)’** |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# LOGIC DIAGRAM: TRUTH TABLE: LOGIC DIAGRAM TRUTH TABLE

|  |  |
| --- | --- |
| **A** | **Y=A’** |
| 0 | 1 |
| 1 | 0 |



**NOR GATE: XOR GATE:**

# LOGIC DIAGRAM: TRUTH TABLE: LOGIC DIAGRAM TRUTH TABLE

|  |  |  |
| --- | --- | --- |
| **A** | **B** |  |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Y=(A+B)’** |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |



**XNOR GATE:**

# LOGIC DIAGRAM: TRUTH TABLE:

|  |  |  |
| --- | --- | --- |
| **A** | **B** |  |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

****

**VERILOG SOURCE CODE:**

**AND GATE:** module andgate(a,b, c); **OR GATE:** module orgate(a,b, c);

input a,b; input a,b;

output c; output c;

assign c= a&b; assign c=a | b;

endmodule endmodule

**NOT GATE:** module notgate(a, abar); **NAND GATE:** module nandgate(a,b, c);

input a; input a,b;

output abar; output c;

assign abar = ~a; assign c = ~(a & b);

endmodule endmodule

**NOR GATE:** module norgate(a,b, c); **XOR GATE:** module exorgate(a,b, c);

input a,b; input a,b;

output c; output c;

assign c = ~(a | b); assign c = a ^ b;

endmodule endmodule

**XNOR GATE:** module exnorgate(a,b, c);

input a,b;

output c;

assign c = ~(a ^ b);

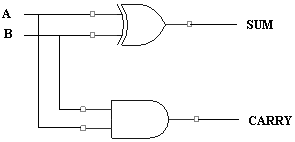
endmodule

**ADDERS AND SUBRACTORS:**

**HALF ADDER:**

LOGIC DIAGRAM: TRUTH TABLE:

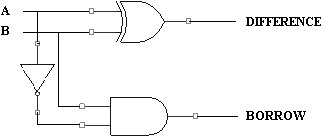
|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **SUM** | **CARRY** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

****

**HALF SUBTRACTOR:**

LOGIC DIAGRAM: TRUTH TABLE

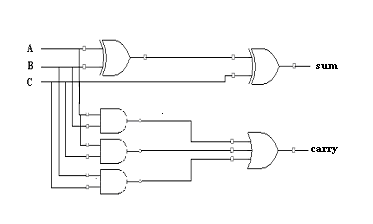
|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **DIFFERENCE** | **BORROW** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

****

**FULL ADDER:**

LOGIC DIAGRAM: TRUTH TABLE:

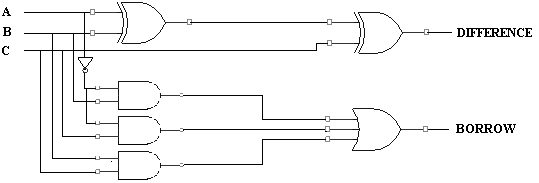
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **SUM** | **CARRY** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

****

**FULL SUBSTRACTOR:**

LOGIC 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 |

****

**VERILOG SOURCE CODE:**

**HALF ADDER:**

module halfadder(a,b, sum,carry);

input a,b;

output sum,carry;

assign sum=a^b;

assign carry=a&b;

endmodule

**HALF SUBTRACTOR:**

module halfsub(a,b, diff,br);

input a,b;

output diff,br;

wire abar;

assign abar=~a;

assign diff=a^b;

assign br=abar & b;

endmodule

**FULL ADDER:**

module fulladder(a,b,cin, sum,carry);

input a,b,cin;

output sum,carry;

wire p,q,r,s;

assign p=a^b;

assign sum=p^cin;

assign q=a&b;

assign r=b&cin;

assign s=cin&a;

assign carry=q|r|s;

endmodule

**FULL SUBTRACTOR:**

module fullsub(a,b,c, diff,br);

input a,b,c;

output diff,br;

wire abar;

assign abar=~a;

assign diff=a^b^c;

assign br=(abar&b)|(b&c)|(c&abar);

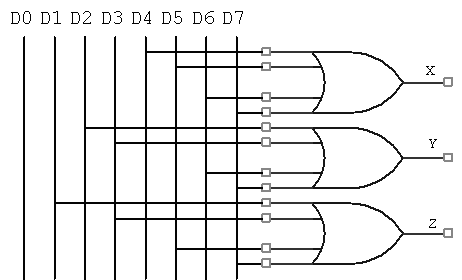
endmodule

**ENCODER AND DECODERS:**

**ENCODER:**

LOGIC DIAGRAM: TRUTH TABLE:

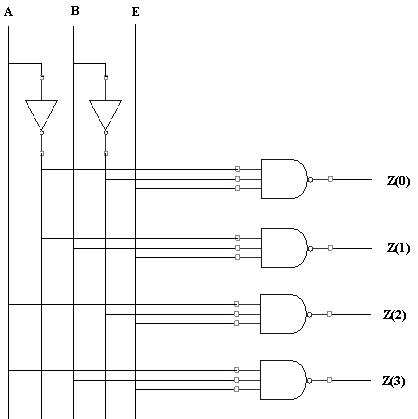
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **D0** | **D1** | **D2** | **D3** | **D4** | **D5** | **D6** | **D7** | **X** | **Y** | **Z** |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |



**DECODERS:**

LOGIC DIAGRAM: TRUTH TABLE:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **Z(0)** | **Z(1)** | **Z(2)** | **Z(3)** |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 |



**VERILOG SOURCE CODE:**

**ENCODER:**

module encoder(D, x,y,z);

input [7:0] D;

output x,y,z;

assign x=D[4]|D[5]|D[6]|D[7];

assign y=D[2]|D[3]|D[6]|D[7];

assign z=D[1]|D[3]|D[5]|D[7];

endmodule

**DECODER**

module decoder(a,b,e,d);

input a,b,e;

output [3:0] d;

wire abar,bbar;

assign abar=~a;

assign bbar=~b;

assign d[0]=abar&bbar&e;

assign d[1]=abar&b&e;

assign d[2]=a&bbar&e;

assign d[3]=a&b&e;

endmodule

**MULTIPLEXER AND DE-MULTIPLEXER:**

**MULTIPLEXER:**

**LOGIC DIAGRAM: TRUTH TABLE:**

|  |  |  |
| --- | --- | --- |
| **SELECT INPUT** | | **OUTPUT** |
| S1 | S0 | Y |
| 0 | 0 | D0 |
| 0 | 1 | D1 |
| 1 | 0 | D2 |
| 1 | 1 | D3 |



**DEMULTIPLEXER:**

**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 |



**VERILOG SOURCE CODE:**

**MULTIPLEXER:**

module mux(d, a,b, y);

input [3:0] d;

input a,b;

output y;

wire abar,bbar,p,q,r,s;

assign abar=~a;

assign bbar=~b;

assign p=d[0]&abar&bbar;

assign q=d[1]&abar&b;

assign r=d[2]&a&bbar;

assign s=d[3]&a&b;

assign y=p|q|r|s;

endmodule

**DEMULTIPLEXER:**

module demux(din,e,a,b, y);

input din,e,a,b;

output [3:0] y;

wire abar,bbar;

assign abar=~a;

assign bbar=~b;

assign y[0]=din&abar&bbar&e;

assign y[1]=din&abar&b&e;

assign y[2]=din&a&bbar&e;

assign y[3]=din&a&b&e;

endmodule

**RESULT:**

Thus the source code for combinational circuit was developed and simulated with ModelSim simulator.

**14. SIMULATION OF SEQUENTIAL CIRCUITS USING VERILOG HDL.**

**AIM:**

To develop the source code for flip-flops by using VERILOG and obtain the simulation.

**SOFTWARE REQUIRED:**

* Xilinx – Project Navigator
* ModelSim Simulator

**PROCEDURE:**

Step1: Define the specifications and initialize the design.

Step2: Declare the name of the module by using VERILOG source code.

Step3: Write the source code in VERILOG.

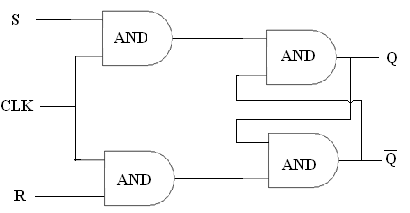
Step4: Check the syntax and debug the errors if found, obtain the synthesis is report.

Step5: Verify the output by simulating the source code.

**SR FLIP FLOP:**

**LOGIC DIAGRAM: TRUTH TABLE:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Q(t)** | **S** | **R** | **Q(t+1)** |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | X |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | x |

****

**VERILOG SOURCE CODE:**

module srff\_clk(s, r, clk, q, qn);

input s, r, clk, q;

output qn;

reg qn;

always @(posedge clk,s,r,q)begin

qn=(s|((~r)&q));

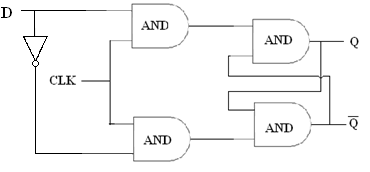
end

endmodule

**D FLIP FLOP:**

**LOGIC DIAGRAM: TRUTH TABLE:**

|  |  |  |
| --- | --- | --- |
| **Q(t)** | **D** | **Q(t+1)** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

****

**VERILOG SOURCE CODE:**

module DFF(d, clk, q);

input d;

input clk;

output q;

reg q;

always @(posedge clk)begin

q=d;

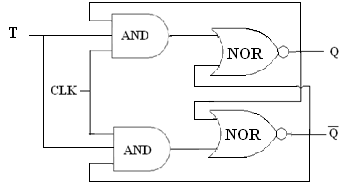
end

endmodule

**T- FLIP FLOP:**

**LOGIC DIAGRAM: TRUTH TABLE:**

|  |  |  |
| --- | --- | --- |
| **Q(t)** | **T** | **Q(t+1)** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

****

**VERILOG SOURCE CODE:**

module TFF(t, clk, q,qout);

input t,q;

input clk;

output qout;

reg qout;

always @(posedge clk)begin

qout=t^q;

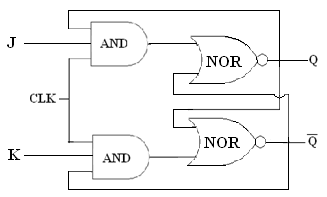
end

endmodule

**JK FLIP FLOP:**

**LOGIC DIAGRAM: TRUTH TABLE:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Q(t)** | **J** | **K** | **Q(t+1)** |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

****

**VERILOG SOURCE CODE:**

module JKFF(J,K, clk, q,qout);

input J,K,q;

input clk;

output qout;

reg qout;

always @(posedge J,K,q, clk)begin

qout=(J&(~q))|(q&(~K));

end

endmodule

**RESULT:**

Thus the source code for flip flops(SRFF,DFF,TFF.JKFF) was developed and simulated with ModelSim simulator.