**DEPARTMENT OF**
**ELECTRONICS & COMMUNICATION ENGINEERING**

# DIGITAL LOGIC DESIGN LAB MANUAL

**II B.TECH I SEM**

**(PVP20 REGULATIONS)**

**PRASAD V POTLURI SIDDHARTHA INSTITUTE OF TECHNOLOGY**
**(Autonomous, Accredited by NBA & NAAC, an ISO 9001:2015 certified institution)**
**(Sponsored by Siddhartha Academy of General & Technical Education)**
**VIJAYAWADA – 520 007**

| **Experiment No.** | **Contents** | **Mapped CO** |
|---|---|---|
| I | Verification of Truth Tables of Logic gates. | CO1 |
| II | Implementation of Logic gates using Universal Gates. | CO1 |
| III | Implementation of the given Boolean functions using logic gates. | CO2 |
| IV | Simplification of the given Boolean functions using K-map and implementation using logic gates. | CO2 |
| V | Realization and verification of Full adder and Full Subtractor using logic gates. | CO1, CO2 |
| VI | Implementation of 2x4 Decoder and 4x1 Multiplexer using Logic Gates. | CO1, CO2 |
| VII | Implementation of the given function using decoder and logic gates. | CO2, CO3 |
| VIII | Implementation of the given function using Multiplexer. | CO2, CO3 |
| IX | Verification of State Tables of SR, JK, D and T-Flip-Flops. | CO1, CO3 |
| X | Design and Verify the operation of 3-bit Ripple Counter using JK flip-flops. | CO1, CO2, CO4 |
| XI | Design and Verify the operation of 4-bit Synchronous Counter using T flip-flops. | CO1, CO2, CO4 |
| XII | Design and verify the operation of 4-bit Shift Register. | CO3, CO4 |
| XIII | Mini Project. | CO1, CO2, CO3, CO4 |

|  |  |
|---|---|
| **Expt. No: 1** | **VERIFICATION OF TRUTH TABLES OF LOGIC GATES** |
| **Dt:** |  |

**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. | XOR GATE | IC 7486 | 1 |
| 8. | DIGITAL IC TRAINER KIT |  | 1 |
| 9. | CONNECTING WIRES |  |  |

**THEORY:**

**Logic gates** are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on certain logic.

OR, AND & NOT gates are called as basic gates. NAND & NOR are considered as universal gates because any other gate or any Boolean expression can be implemented by using these gates.

**AND GATE:**

The AND gate performs logical multiplication operation on binary variables. 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 on binary variables. 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 performs complement operation. It is also called as 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 performs the complement of AND operation. It is a combination of AND-NOT gates. The output is high when both inputs are low or any one of the input is low. The output is low level when both inputs are high.

**NOR GATE:**

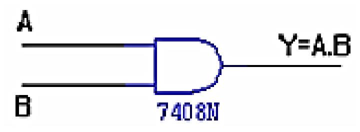
The NOR gate performs the complement of OR operation. It is a combination of OR-NOT gates. The output is high when both inputs are low. The output is low when one or both inputs are high.

**XOR GATE:**

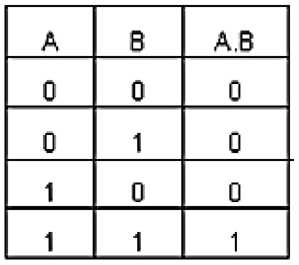
The output is high when any one of the inputs is high. The output is low either both the inputs are low or high.

**AND GATE:**

**SYMBOL:**

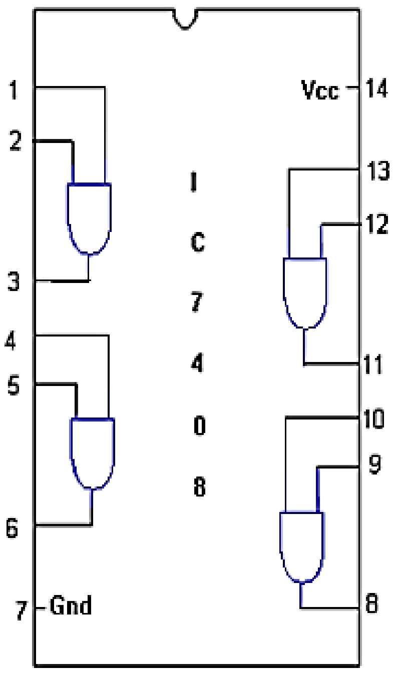


TRUTH TABLE



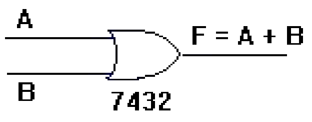
| A | B | A.B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**PIN DIAGRAM:**

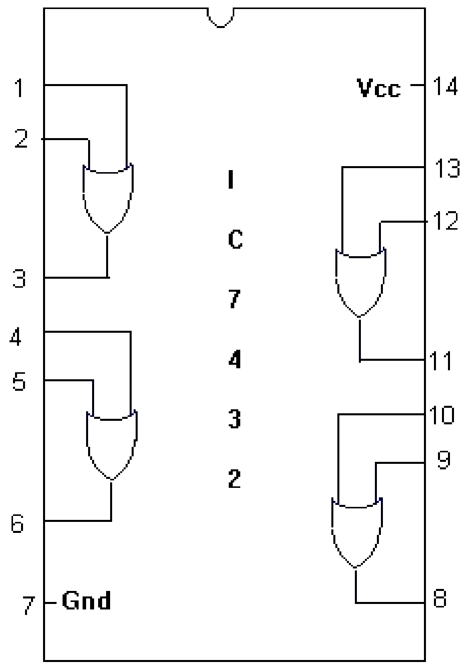


**OR GATE:**

SYMBOL :



PIN DIAGRAM :

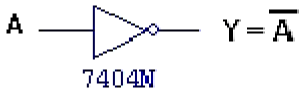


TRUTH TABLE

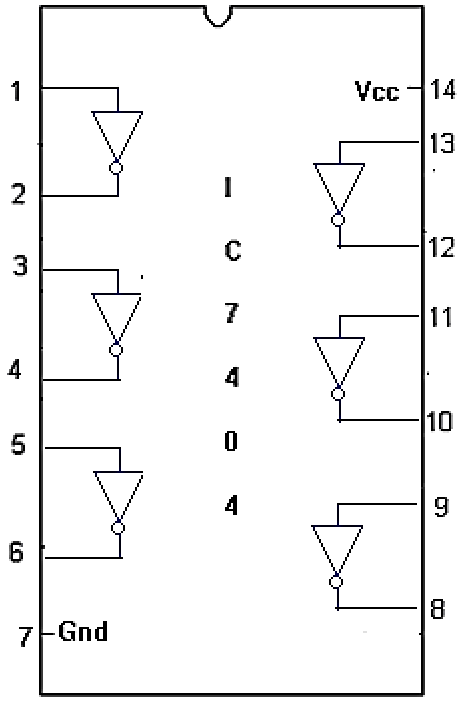
| A | B | A+B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**NOT GATE:**

**SYMBOL:**



**PIN DIAGRAM:**

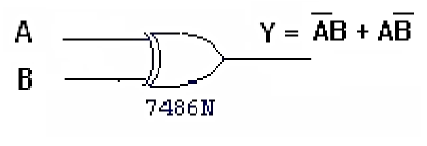


TRUTH TABLE :

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

**XOR GATE:**

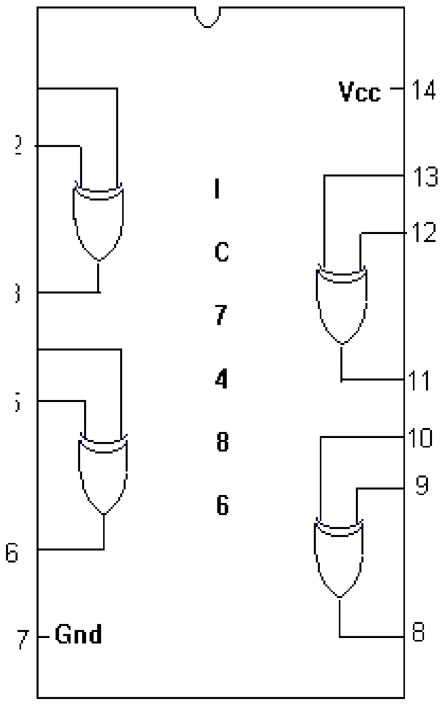
**SYMBOL:**



TRUTH TABLE :

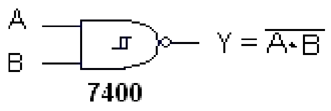
| A | B | $\overline{A}B + A\overline{B}$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**PIN DIAGRAM :**



**2-INPUT NAND GATE:**

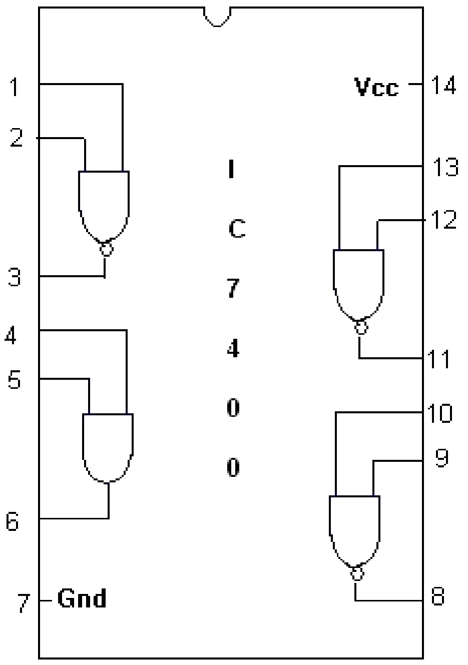
**SYMBOL:**



TRUTH TABLE

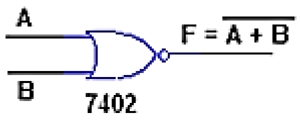
| A | B | $\overline{A \cdot B}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**PIN DIAGRAM:**



## NOR GATE:

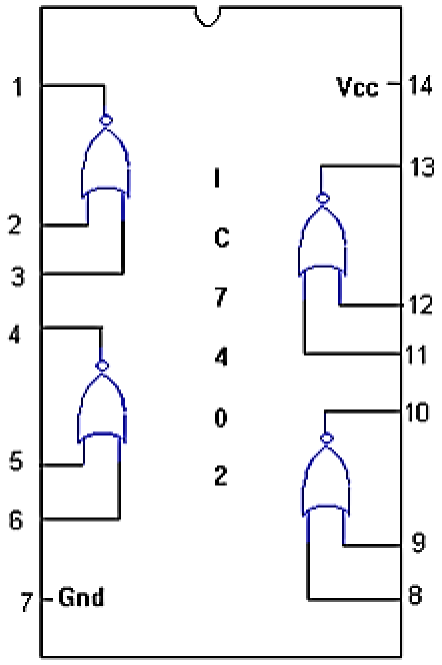
SYMBOL :



TRUTH TABLE

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

PIN DIAGRAM :



**PROCEDURE:**

1. Insert the appropriate IC into the Breadboard.
2. Make connections as shown in the circuit diagram.
3. Connect Pin-14 of all ICs to +5V and Pin-7 to ground.
4. Provide the input data via input switches.
5. Feed the logic 0 (0V) or logic 1 (5V) in different combinations at the inputs according to truth table.
6. Observe the output on output LEDs.
7. Give all possible combinations of inputs and note down the output.
8. Verify the truth tables of logic gates.

**RESULT:**

Thus the logic gates are studied and their truth tables are verified.

| Expt. No: 2 | IMPLEMENTATION OF LOGIC GATES USING UNIVERSAL GATES |
|---|---|
| Dt: | |

**AIM:**

To implement logic gates by using universal (NAND & NOR) gates and verify the truth tables.

**APPARATUS REQUIRED:**

| Sl. No. | COMPONENT | SPECIFICATION | QTY. |
|---|---|---|---|
| 1. | NAND GATE | IC 7400 | 1 |
| 2. | NOR GATE | IC 7402 | 1 |
| 3. | DIGITAL IC TRAINER KIT | | 1 |
| 4. | CONNECTING WIRES | | |

**THEORY:**

A universal gate is a gate which can implement any Boolean function without need to use any other gate type. The NAND and NOR gates are universal gates from which any other logic gates can be built.. In practice, this is advantageous since NAND and NOR **gates** are economical and easier to fabricate and are the basic gates used in all IC digital logic families.

## ***<u>NAND Logic:</u>***

**NOT Gate**

A NOT gate is made by joining the inputs of a NAND gate. Since a NAND gate is equivalent to an AND gate followed by a NOT gate, joining the inputs of a NAND gate leaves only the NOT part.

**AND Gate**

An **AND** gate is made by following a NAND gate with a NOT gate.

**OR Gate**

If the truth table for a NAND gate is examined or by applying De Morgan's Laws, it can be seen that if any of the inputs are 0, then the output will be 1. However to be an OR gate, if any input is 1, the output must also be 1. Therefore, if the inputs are inverted, any high input will trigger a high output.

**NOR Gate**

A NOR gate is simply an OR gate with an inverted output:

**XOR Gate**

An XOR gate is constructed similarly to an OR gate, except with an additional NAND gate inserted such that if both inputs are high, the inputs to the final NAND gate will also be high, and the output will be low.

## ***NOR Logic:***

**NOT Gate**

This is made by joining the inputs of a NOR gate. As a NOR gate is equivalent to an OR gate leading to NOT gate, this automatically sees to the "OR" part of the NOR gate, eliminating it from consideration and leaving only the NOT part.

**OR Gate**

The OR gate is simply a NOR gate followed by a NOT gate.

**AND Gate**

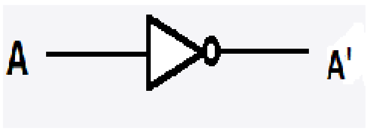
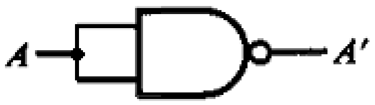
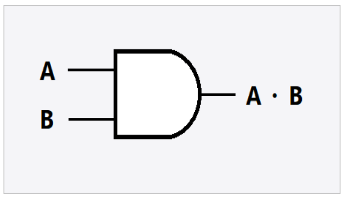
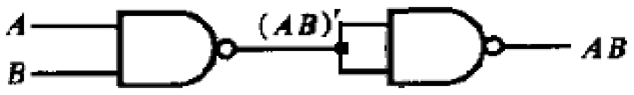
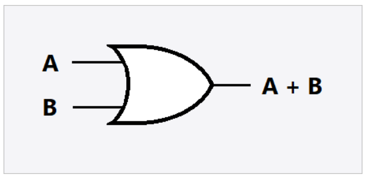
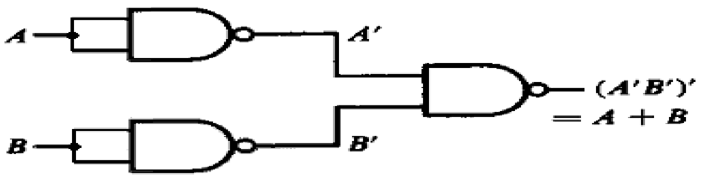
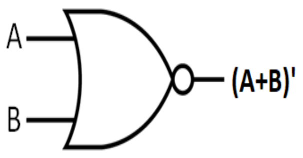
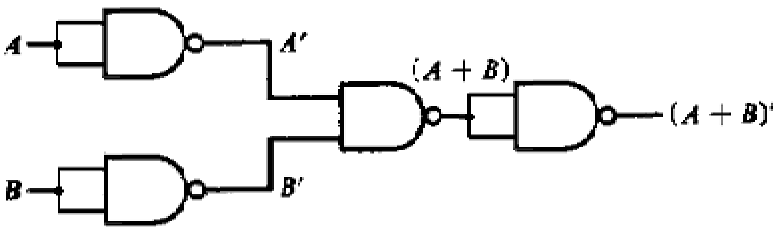
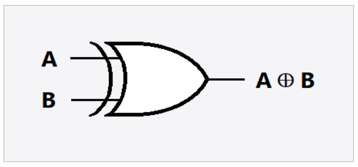
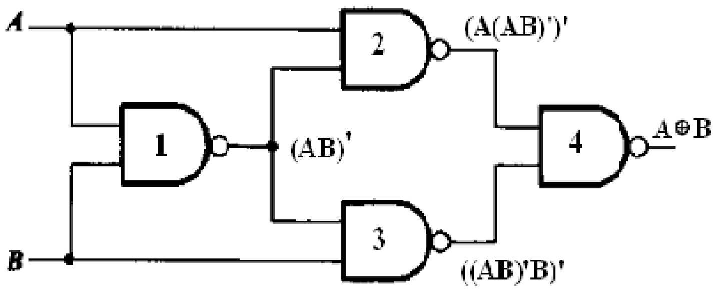
An AND gate gives a 1 output when both inputs are 1; a NOR gate gives a 1 output only when both inputs are 0. Therefore, an AND gate is made by inverting the inputs to a NOR gate.

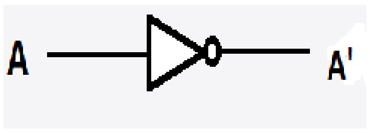
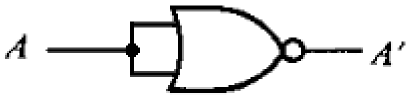
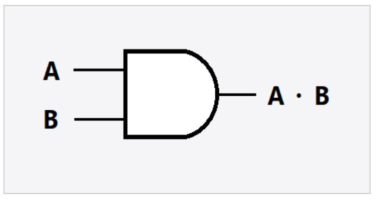
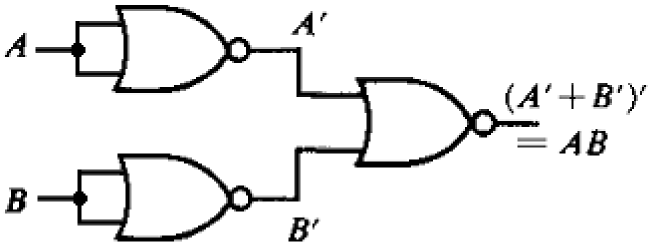
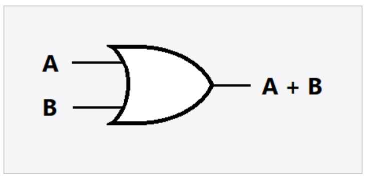
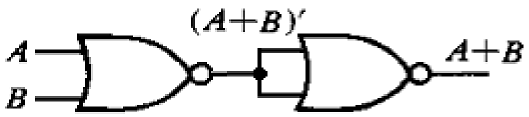
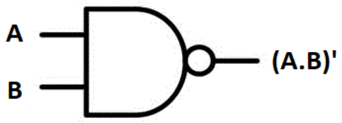
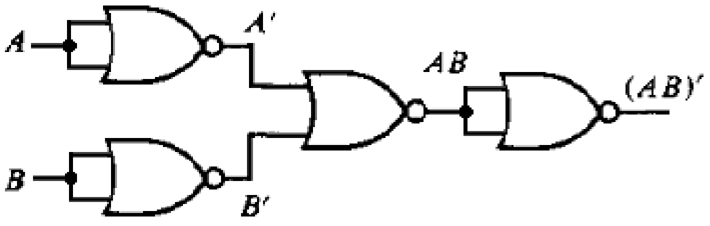
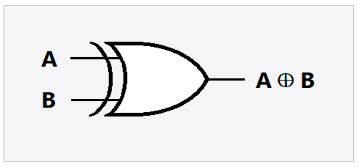
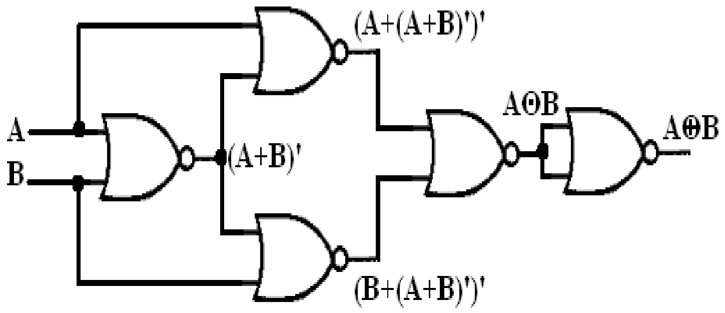
**NAND Gate**

A NAND gate is made using an AND gate in series with a NOT gate:

**XOR Gate**

An XOR gate is made by connecting the output of 3 NOR gates (connected as an AND gate) and the output of a NOR gate to the respective inputs of a NOR gate.

| DESIRED GATE | NAND CONSTRUCTION |
|---|---|
|  <br>NOT GATE |  |
|  <br>AND GATE |  |
|  <br>OR GATE |  |
|  <br>NOR GATE |  |
|  <br>XOR GATE |  A<br>B<br>1<br>2<br>3<br>4<br>(AB)'<br>(A(AB)')'<br>((AB)'B)'<br>A⊕B |

| DESIRED GATE | NOR CONSTRUCTION |
|---|---|
|  <br>NOT GATE |  |
|  <br>AND GATE |  |
|  <br>OR GATE |  |
|  <br>NAND GATE |  |
|  <br>XOR GATE |  (A+(A+B)')'<br>A<br>B<br>(A+B)'<br>AΘB<br>AΘB<br>(B+(A+B)')' |

**PROCEDURE:**

1. Insert the appropriate IC into the Breadboard.
2. Make connections as shown in the circuit diagram.
3. Connect Pin-14 of all ICs to +5V and Pin-7 to ground.
4. Provide the input data via input switches.
5. Feed the logic 0 (0V) or logic 1 (5V) in different combinations at the inputs according to truth table.
6. Observe the output on output LEDs.
7. Give all possible combinations of inputs and note down the output.
8. Verify the truth tables of logic gates.

**RESULT:**

Thus the logic gates are constructed using universal gates and their truth tables are verified.

|  |  |
|---|---|
| **Expt. No: 3** | **IMPLEMENTATION OF BOOLEAN FUNCTIONS USING LOGIC GATES** |
| **Dt:** |  |

**AIM:** To implement the following Boolean functions using Logic Gates.

(i) $Y = A + B C'$

(ii) $F = (A + B).(B + C')$

**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. | DIGITAL IC TRAINER KIT |  | 1 |
| 5. | CONNECTING WIRES |  | 1 |

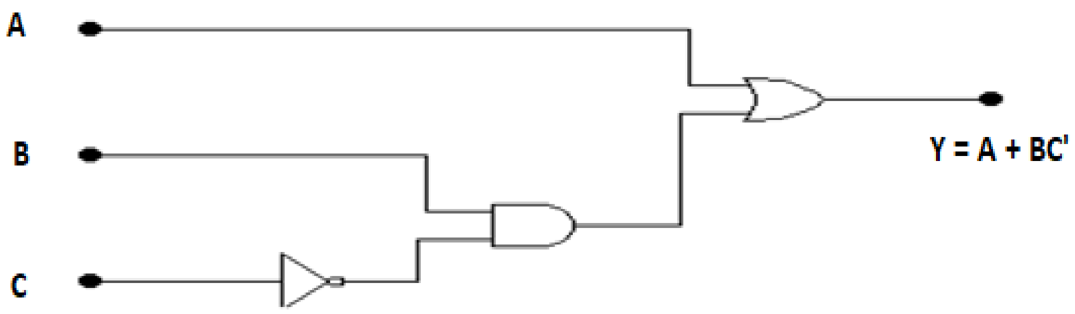
**THEORY:**

Any Boolean function can be represented by using a number of logic gates by interconnecting them. Logic gates implementation or logic representation of Boolean functions is very simple.

Boolean function can be represented easily in SOP (sum of products) form and POS (product of sums) form. To represent these standardized equations logically, we use the logic gates.

The implementation of Boolean functions by using logic gates involves in connecting one logic gate’s output to another gate’s input and involves in using AND, OR, NOT gates.

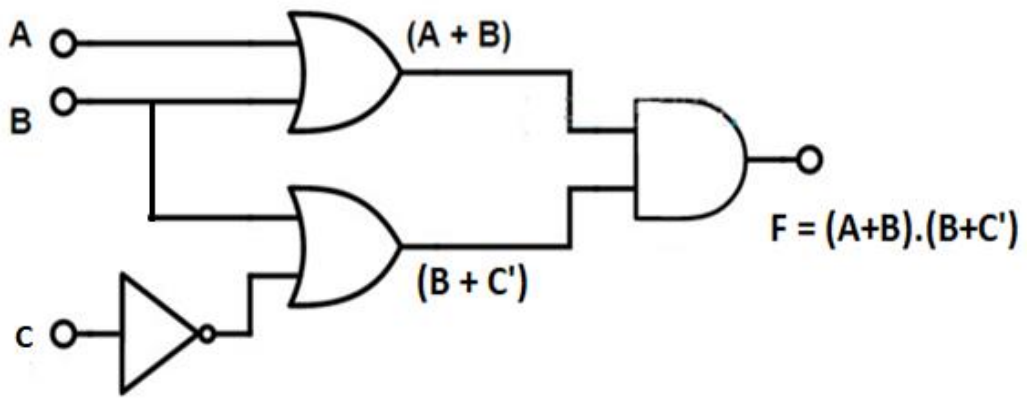
**LOGIC DIAGRAM:**



**TRUTH TABLE:**

| A | B | C | C’ | BC’ | Y = A + BC’ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 |

**LOGIC DIAGRAM:**



**TRUTH TABLE:**

| A | B | C | A+B | B+C’ | F = (A + B).(B+C’) |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

**PROCEDURE:**

1. Insert the appropriate IC’s into the Breadboard.
2. Make connections as shown in the circuit diagram.
3. Connect Pin-14 of all ICs to +5V and Pin-7 to ground.
4. Provide the input data via input switches.
5. Feed the logic 0 (0V) or logic 1 (5V) in different combinations at the inputs according to truth table.
6. Observe the output on output LEDs.
7. Give all possible combinations of inputs and note down the output.
8. Verify the truth table of the functions.

**RESULT:**

Thus the given Boolean functions are implemented using logic gates and the operation is verified using truth table.

| Expt. No: 4 | SIMPLIFICATION OF BOOLEAN FUNCTIONS USING K-MAP AND IMPLEMENTATION USING LOGIC GATES |
|---|---|
| Dt: | |

**AIM:**

To simplify the following Boolean Functions using Karnaugh map and implement the simplified functions using logic gates.

(i) $F_1 (X, Y, Z) = \Sigma m (3, 4, 6, 7)$

(ii) $F_2 (A, B, C, D) = \Sigma m (2, 6, 8, 9, 10, 11, 14)$

**APPARATUS:**

| SL No. | COMPONENT | SPECIFICATION | QTY |
|---|---|---|---|
| 1. | NAND GATE | IC 7400 | 2 |
| 2. | DIGITAL IC TRAINER KIT | | 1 |
| 3. | CONNECTING WIRES | | 1 |

**THEORY:**

The Karnaugh map method provides a simple straight forward procedure for minimizing Boolean functions. This method may be regarded either as a pictorial form of a truth table or as an extension of the Venn diagram. The map method, first proposed by Veitch and modified by Karnaugh, is also known as the "Veitch diagram" or the "Karnaugh map".

The map is a diagram made up of squares. Each square represents one minterm. Since any Boolean function can be expressed as a sum of minterms, it follows that a Boolean function is recognized graphically in the map from the area enclosed by those squares whose min terms are included in the function.

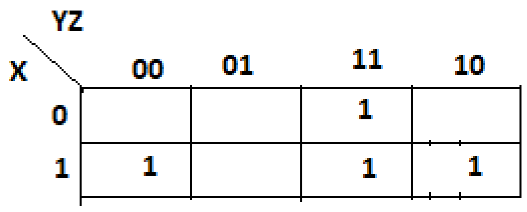
In fact, the map presents a visual diagram of all possible ways a function may be expressed in a standard form. By recognizing various patterns, the user can derive simplified algebraic expressions for the function.

**TRUTH TABLE:**

$F_1 (X, Y, Z) = \Sigma m (3, 4, 6, 7)$

| X | Y | Z | $F_1$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

K-Map Simplification:

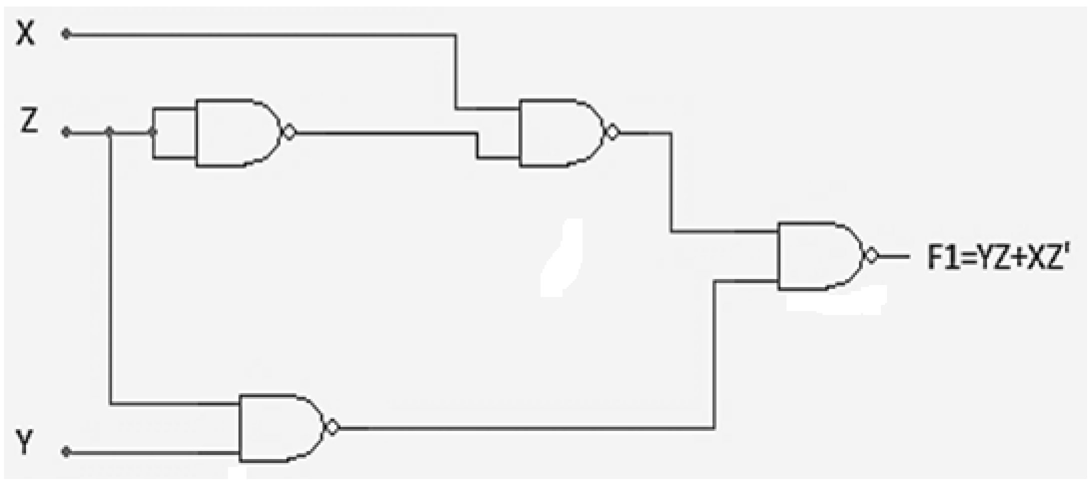


| X \ YZ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | | | 1 | |
| 1 | 1 | | 1 | 1 |

Simplified expression for $F_1$ using K-map is

$F_1 = YZ + XZ'$

**LOGIC DIAGRAM:**

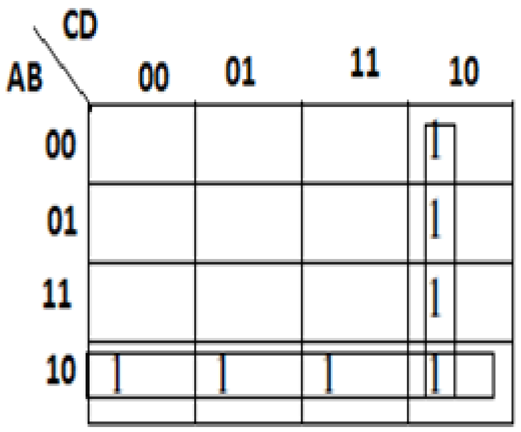


**TRUTH TABLE:**

$F_2$ (A, B, C, D) = Σm (2, 6, 8, 9, 10, 11, 14)

| INPUTS | | | | OUTPUT |
|---|---|---|---|---|
| A | B | C | D | Y |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

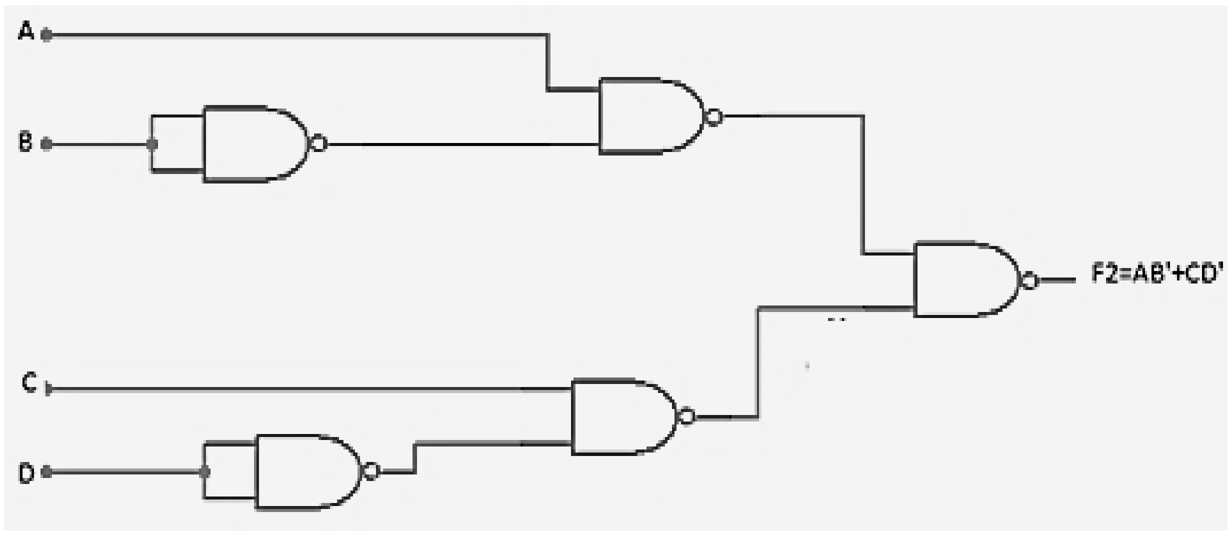
K-Map Simplification:



Simplified expression for $F_2$ using K-map is

$F_2$ = AB' + CD'

**LOGIC DIAGRAM:**



**PROCEDURE:**

1. Insert the appropriate IC's into the Breadboard.
2. Make connections as shown in the circuit diagram.
3. Connect Pin-14 of all ICs to +5V and Pin-7 to ground.
4. Provide the input data via input switches.
5. Feed the logic 0 (0V) or logic 1 (5V) in different combinations at the inputs according to truth table.
6. Observe the output on output LEDs.
7. Give all possible combinations of inputs and note down the output.
8. Verify the truth table of the functions.

**RESULT:**

Thus Boolean functions are simplified using Karnaugh map, implemented using logic gates and the operation is verified using truth table.

| Expt. No: 5 | REALIZATION OF FULL ADDER AND FULL SUBTRACTOR |
|---|---|
| Dt: | |

**AIM:**

To design and realize Full Adder and Full Subtractor circuits using logic gates and verify the truth tables.

**APPARATUS REQUIRED:**

| Sl.No. | COMPONENT | SPECIFICATION | QTY. |
|---|---|---|---|
| 1. | AND GATE | IC 7408 | 1 |
| 2. | XOR GATE | IC 7486 | 1 |
| 3. | NOT GATE | IC 7404 | 1 |
| 4. | OR GATE | IC 7432 | 1 |
| 3. | DIGITAL IC TRAINER KIT | | 1 |
| 4. | CONNECTING WIRES | - | |

**THEORY:**

**FULL ADDER:**

A full adder is a combinational circuit that performs the arithmetic sum of three input bits. It consists of three inputs and two outputs. Two of the input variables, denoted by A and B, represent the two significant bits to be added. The third input Cin represents the carry from the previous lower significant position. Two outputs are necessary because the arithmetic sum of three binary digits ranges in value from 0 to 3, and binary 2 or 3 needs two digits. The two outputs are designated by the symbols S for sum and Cout for output carry.

$S = A \oplus B \oplus C$

$Cout = A.B + A.C + B.C$

**FULL SUBTRACTOR:**

Full subtractor is combinational circuit that performs subtraction of 3-bits. It has 3 inputs & 2 outputs. Here Outputs are known as borrow (Bout) and difference (D).

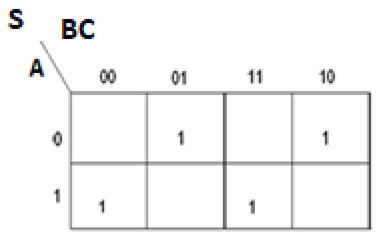
$D = A \oplus B \oplus C$

$Bout = A'.B + BC + A'.C$

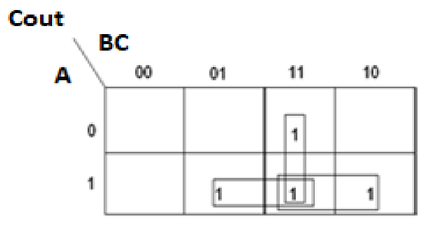
**TRUTH TABLE OF FULL ADDER:**

| A | B | C | Cout | S |
|---|---|---|---|---|
| 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 output:**



**K-Map for CARRY output:**



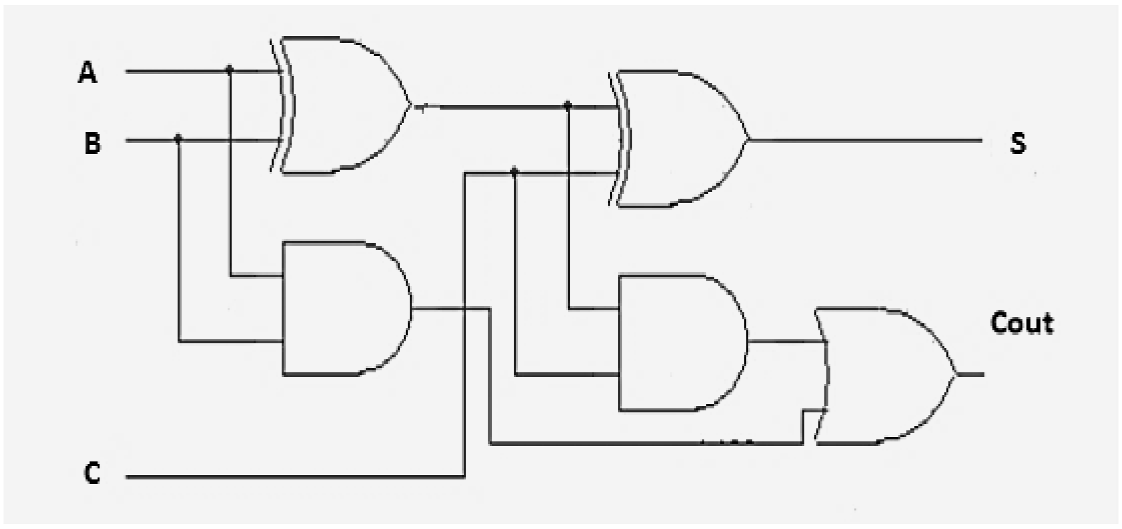
S = A’B’C + A’BC’ + AB’C’ + ABC

S = A ⊕ B ⊕ C

Cout = AB + BC + AC

Cout = (A ⊕ B)C + AB

**LOGIC DIAGRAM OF FULL ADDER:**

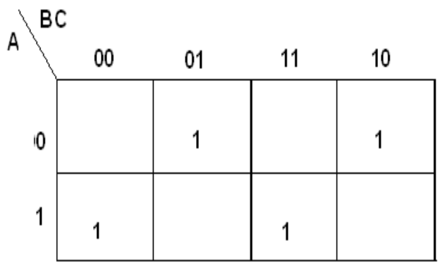


**FULL SUBTRACTOR:**

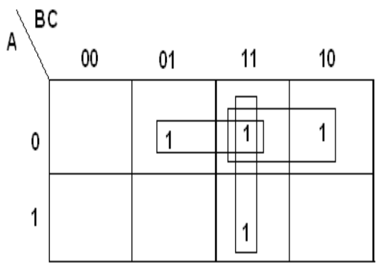
**TRUTH TABLE:**

| A | B | C | Bout | D |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**K-Map for Difference output:**



**K-Map for Borrow output:**



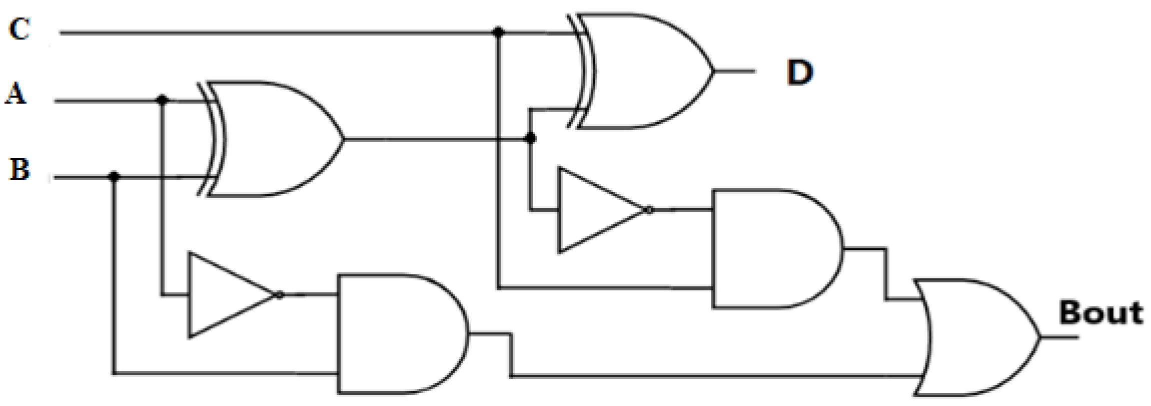
$D = A'B'C + A'BC' + AB'C' + ABC$

$D = A \oplus B \oplus C$

$B_{out} = A'B + BC + A'C$

$B_{out} = (A \oplus B)'C + A'B$

**LOGIC DIAGRAM OF FULL SUBTRACTOR:**



**PROCEDURE:**

1. Insert the appropriate IC’s into the Breadboard.
2. Make connections as shown in the circuit diagram.
3. Connect Pin-14 of all ICs to +5V and Pin-7 to ground.
4. Provide the input data via input switches.
5. Feed the logic 0 (0V) or logic 1 (5V) in different combinations at the inputs according to truth table.
6. Observe the output on output LEDs.
7. Give all possible combinations of inputs and note down the output.
8. Verify the operation of the circuits using truth table.

**RESULT:**

Thus the Full Adder and Full Subtractor circuits are designed and the truth tables are verified.

|  |  |
|---|---|
| **Expt. No:6** | **IMPLEMENTATION OF 2x4 DECODER & 4x1 MULTIPLEXER** |
| **Dt:** |  |

**AIM:**

To implement 2x4 Decoder and 4x1 Multiplexer using logic gates and verify their truth tables.

**APPARATUS REQUIRED:**

| SL No. | COMPONENT | SPECIFICATION | QTY |
|---|---|---|---|
| 1 | AND GATE | IC 7411 | 2 |
| 2 | NOT GATE | IC 7404 | 1 |
| 3 | OR GATE | IC 7432 | 1 |
| 4 | DIGITAL IC TRAINER KIT |  | 1 |
| 5 | CONNECTING WIRES |  |  |

**THEORY:**

**DECODER:**

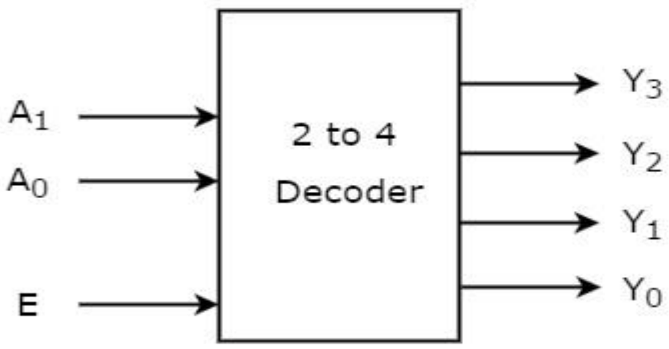
Decoder is a combinational circuit that has ‘n’ input lines and maximum of $2^n$ output lines. One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code. The outputs of the decoder are nothing but the min terms of ‘n’ input variables, when it is enabled.

**MULTIPLEXER:**

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

**2 x 4 DECODER:**

2x4 Decoder has two inputs $A_1$, $A_0$ one enable input E and four outputs $Y_3$, $Y_2$, $Y_1$ & $Y_0$. The block diagram of 2 to 4 decoder is shown in the following figure.



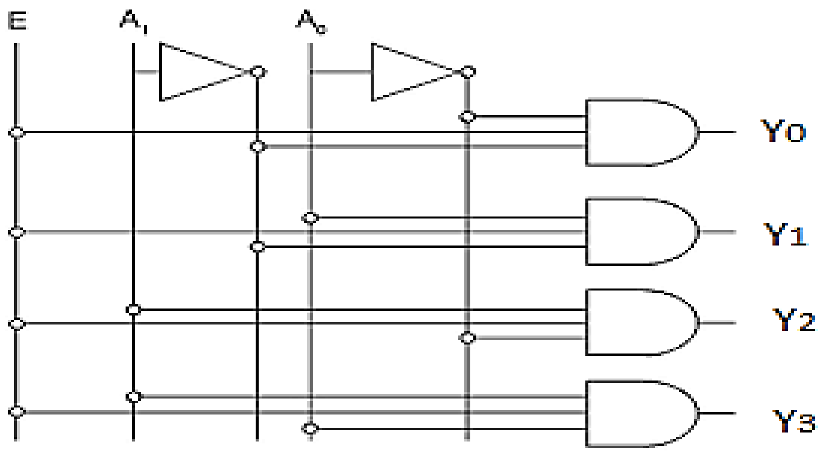
**TRUTH TABLE:**

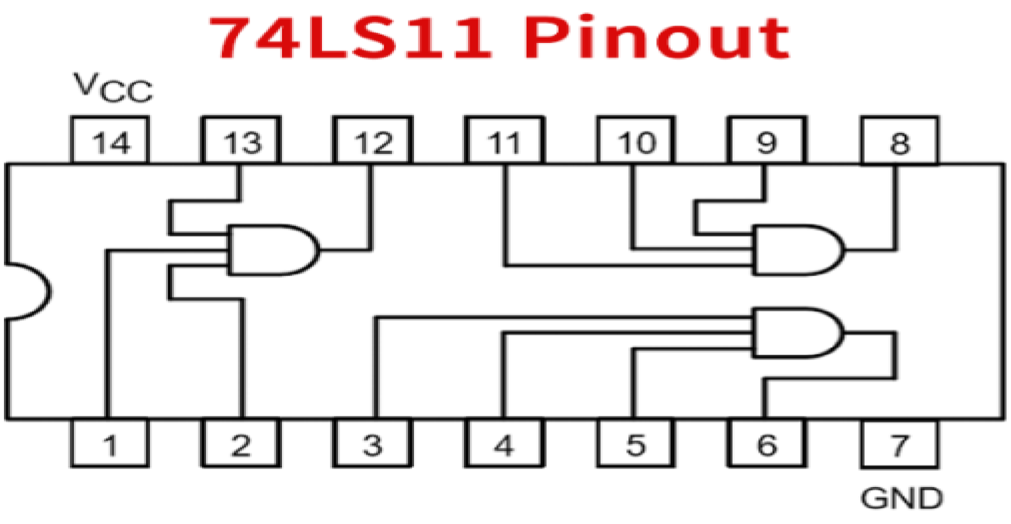
| Enable | Inputs | | Outputs | | | |
|---|---|---|---|---|---|---|
| **E** | **$A_1$** | **$A_0$** | **$Y_3$** | **$Y_2$** | **$Y_1$** | **$Y_0$** |
| 0 | X | X | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |

$Y_3 = E.A_1.A_0$ $\quad$ $Y_2 = E.A_1.A_0'$

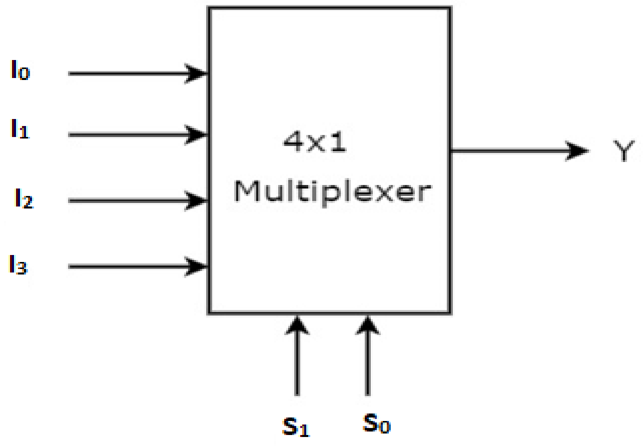
$Y_1 = E.A_1'.A_0$ $\quad$ $Y_0 = E.A_1'.A_0'$

**LOGIC DIAGRAM:**





**4:1 MULTIPLEXER:**



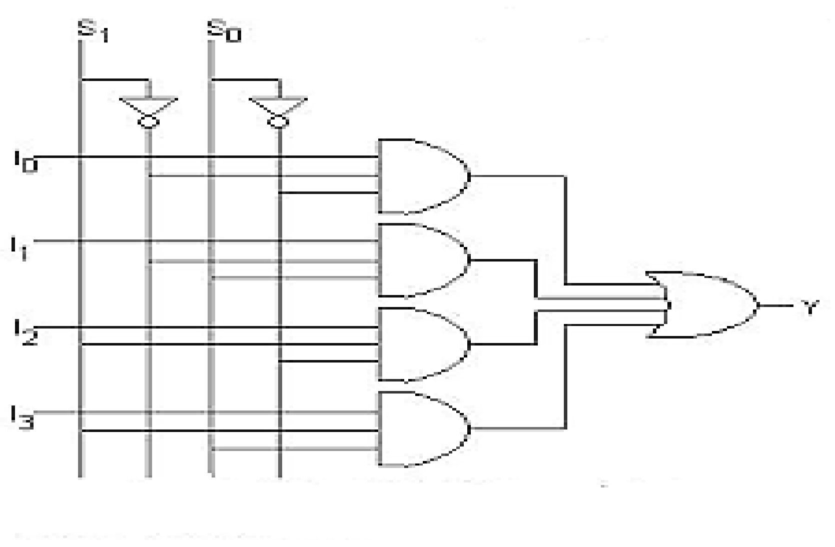
**FUNCTION TABLE:**

| Selection Lines | | Output |
|---|---|---|
| **$S_1$** | **$S_0$** | **Y** |
| 0 | 0 | $I_0$ |
| 0 | 1 | $I_1$ |
| 1 | 0 | $I_2$ |
| 1 | 1 | $I_3$ |

From Truth table, we can directly write the Boolean function for output Y as

$$Y = S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_3$$

**LOGIC DIAGRAM OF 4:1 MULTIPLEXER:**



**PROCEDURE:**

1. Insert the appropriate IC's into the Breadboard.
2. Make connections as shown in the circuit diagram.
3. Connect Pin-14 of all ICs to +5V and Pin-7 to ground.
4. Provide the input data via input switches.
5. Feed the logic 0 (0V) or logic 1 (5V) in different combinations at the inputs according to truth table.
6. Observe the output on output LEDs.
7. Give all possible combinations of inputs and note down the output.
8. Verify the operation of the circuits using truth table.

**RESULT:**

Thus 2x4 Decoder and 4x1 Multiplexer are implemented using logic gates and the operation is verified using truth table.

|  |  |
|---|---|
| **Expt. No:7** | **IMPLEMENTATION OF BOOLEAN FUNCTIONS USING DECODER AND LOGIC GATES** |
| **Dt:** |  |

**AIM:**

To implement the given Boolean functions using Decoder and logic gates and verify their operation using truth table.

**APPARATUS REQUIRED:**

| SL No. | COMPONENT | SPECIFICATION | QTY |
|---|---|---|---|
| 1 | 3x8 DECODER | IC 74LS138 | 1 |
| 2 | NAND GATE | IC 7420 | 1 |
| 3 | DIGITAL IC TRAINER KIT |  |  |
| 4 | CONNECTING WIRES |  |  |

**THEORY:**

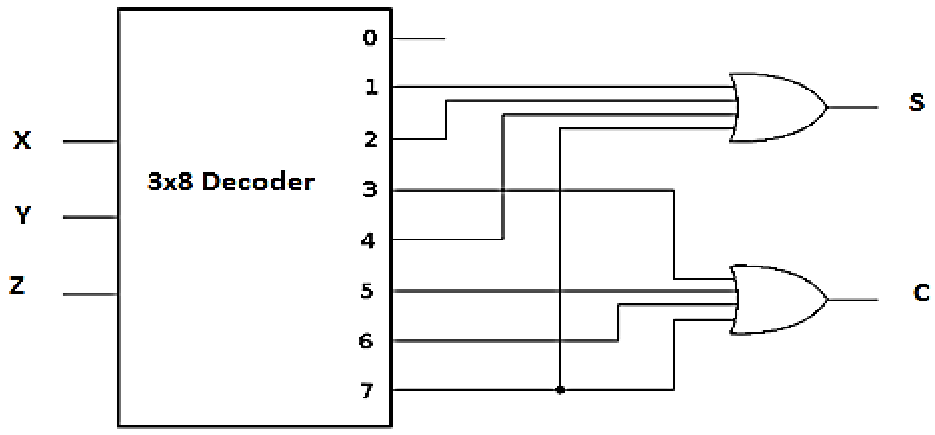
Since any Boolean function can be expressed as a sum of minterms, a decoder that can generate these minterms along with external OR gates that form their logical sums, can be used to form a circuit of any Boolean function.

In order to implement the logic of a full adder, we need a 3:8 decoder and OR gates. The inputs to the full adder are used as inputs to the decoder. Let X, Y and Z represent three inputs and S, C represents the two outputs of a full adder.

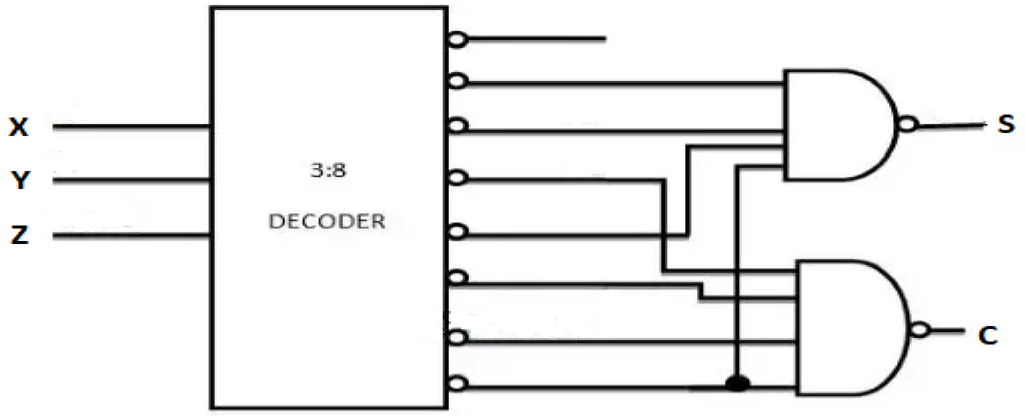
| X | Y | Z | C | S |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

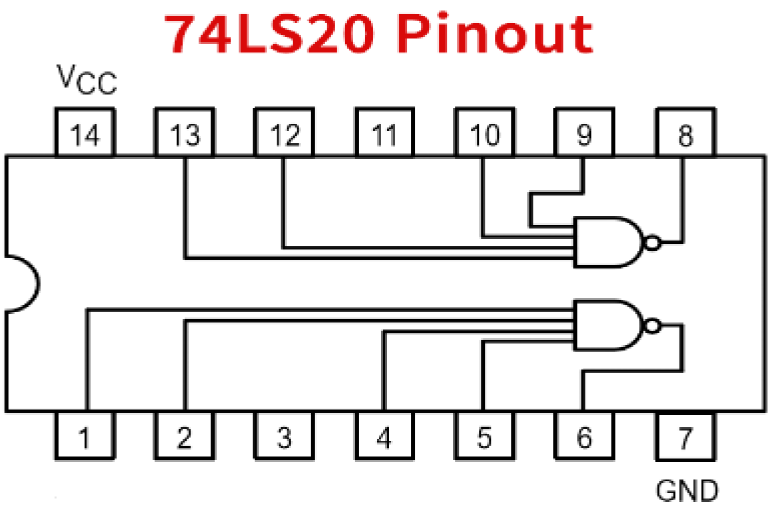
$S = \Sigma m\,(1, 2, 4, 7)$; $C = \Sigma m\,(3, 5, 6, 7)$

**LOGIC DIAGRAM:**

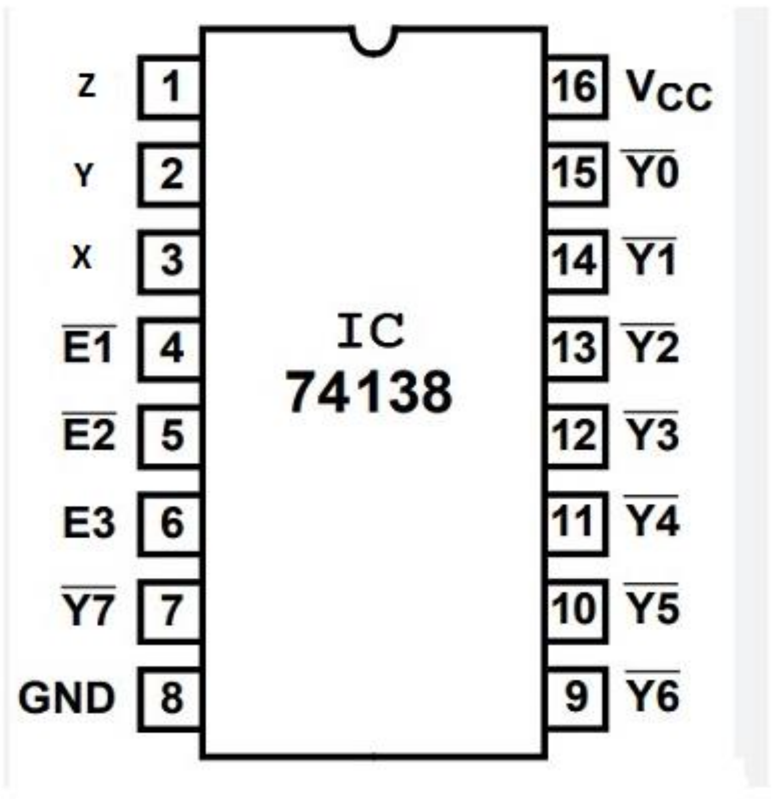


**Logic Diagram using IC 74x138 and NAND Gates:**





**Pin Diagram of IC 74x138:**



**PROCEDURE:**

1. Insert the appropriate IC's into the Breadboard.
2. Make connections as shown in the circuit diagram.
3. Provide the input data via input switches.
4. Feed the logic 0 (0V) or logic 1 (5V) in different combinations at the inputs according to truth table.
5. Observe the output on output LEDs.
6. Give all possible combinations of inputs and note down the output.
7. Verify the operation of the circuit using truth table.

**RESULT:**

Thus the given Boolean functions are implemented using decoder and logic gates and verified successfully.

| Expt. No: 8 | IMPLEMENTATION OF BOOLEAN FUNCTION USING MULTIPLEXER |
|---|---|
| Dt: | |

**AIM:**

To implement the given four variable Boolean function using 8x1 Multiplexer and verify its operation using truth table.

$F(A, B, C, D) = \Sigma m\,(1, 3, 4, 11, 12, 13, 14, 15)$

**APPARATUS REQUIRED:**

| Sl.No. | COMPONENT | SPECIFICATION | QTY. |
|---|---|---|---|
| 1. | 8X1 MULTIPLEXER | IC 74151 | 1 |
| 2. | NOT GATE | IC 7404 | 1 |
| 3. | DIGITAL IC TRAINER KIT | | |
| 4. | CONNECTING WIRES | | |

**THEORY:**

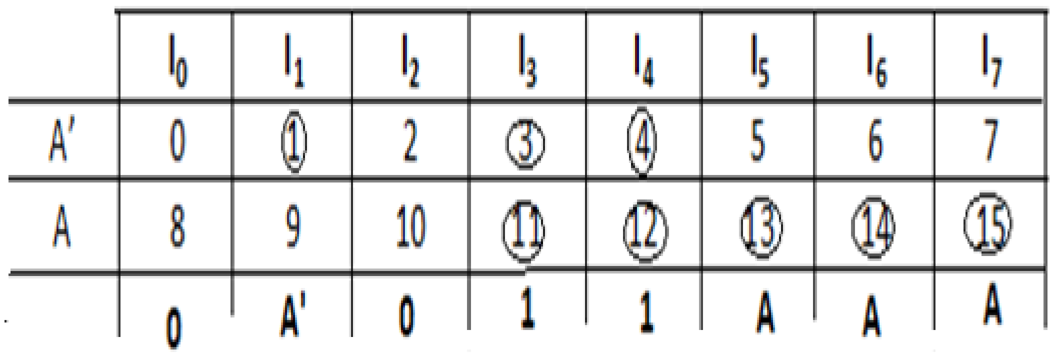
In order to implement an ‘n’ variable Boolean function using Multiplexer, we take (N-1) variables of the function on the selection lines and ‘1’ variable is used for inputs of a multiplexer. As we have N-1 variables on selection lines we need to have $2^{N-1}$ to 1 Multiplexer. The inputs of the Multiplexer can be ‘0’, ‘1’, the variable, or the complement of the variable. These values are then applied to the data inputs of the Multiplexer in the proper order.

**TRUTH TABLE:**

$F(A, B, C, D) = \Sigma m\,(1, 3, 4, 11, 12, 13, 14, 15)$

| Inputs | | | | Output |
|---|---|---|---|---|
| **A** | **B** | **C** | **D** | **F** |
| **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **0** | **1** | **1** |
| **0** | **0** | **1** | **0** | **0** |
| **0** | **0** | **1** | **1** | **1** |
| **0** | **1** | **0** | **0** | **1** |
| **0** | **1** | **0** | **1** | **0** |
| **0** | **1** | **1** | **0** | **0** |
| **0** | **1** | **1** | **1** | **0** |
| **1** | **0** | **0** | **0** | **0** |
| **1** | **0** | **0** | **1** | **0** |
| **1** | **0** | **1** | **0** | **0** |
| **1** | **0** | **1** | **1** | **1** |
| **1** | **1** | **0** | **0** | **1** |
| **1** | **1** | **0** | **1** | **1** |
| **1** | **1** | **1** | **0** | **1** |
| **1** | **1** | **1** | **1** | **1** |

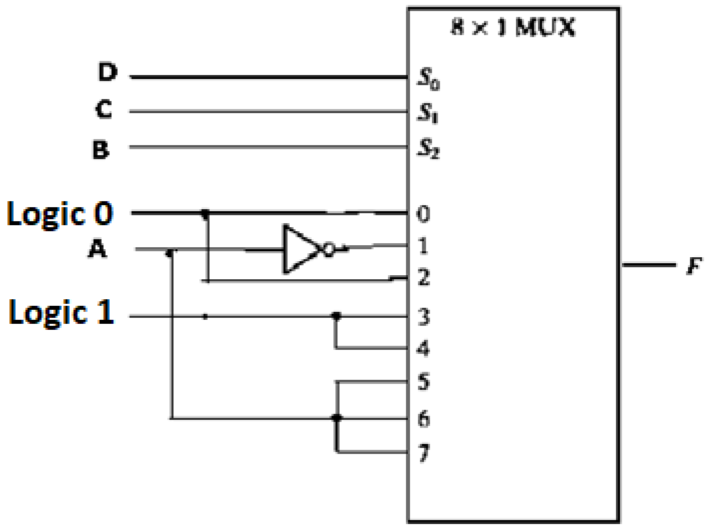
**Method 1:** If the inputs B, C & D are considered as select lines and A is considered as data input of a Multiplexer, then the implementation table is



| | $I_0$ | $I_1$ | $I_2$ | $I_3$ | $I_4$ | $I_5$ | $I_6$ | $I_7$ |
|---|---|---|---|---|---|---|---|---|
| A′ | 0 | ① | 2 | ③ | ④ | 5 | 6 | 7 |
| A | 8 | 9 | 10 | ⑪ | ⑫ | ⑬ | ⑭ | ⑮ |
| | 0 | A′ | 0 | 1 | 1 | A | A | A |

Implementation table of a MUX

**LOGIC DIAGRAM:**



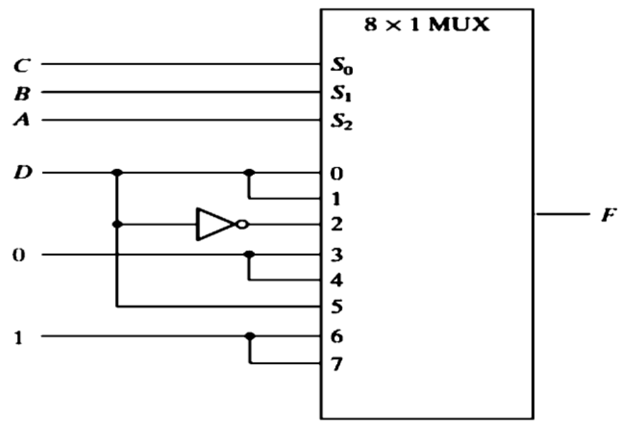
**Method 2:** If the inputs A, B & C are considered as select lines and D is considered as data input of Multiplexer, then

| $A$ | $B$ | $C$ | $D$ | $F$ | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | $F = D$ |
| 0 | 0 | 0 | 1 | 1 | |
| 0 | 0 | 1 | 0 | 0 | $F = D$ |
| 0 | 0 | 1 | 1 | 1 | |
| 0 | 1 | 0 | 0 | 1 | $F = D'$ |
| 0 | 1 | 0 | 1 | 0 | |
| 0 | 1 | 1 | 0 | 0 | $F = 0$ |
| 0 | 1 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 0 | 0 | $F = 0$ |
| 1 | 0 | 0 | 1 | 0 | |
| 1 | 0 | 1 | 0 | 0 | $F = D$ |
| 1 | 0 | 1 | 1 | 1 | |
| 1 | 1 | 0 | 0 | 1 | $F = 1$ |
| 1 | 1 | 0 | 1 | 1 | |
| 1 | 1 | 1 | 0 | 1 | $F = 1$ |
| 1 | 1 | 1 | 1 | 1 | |

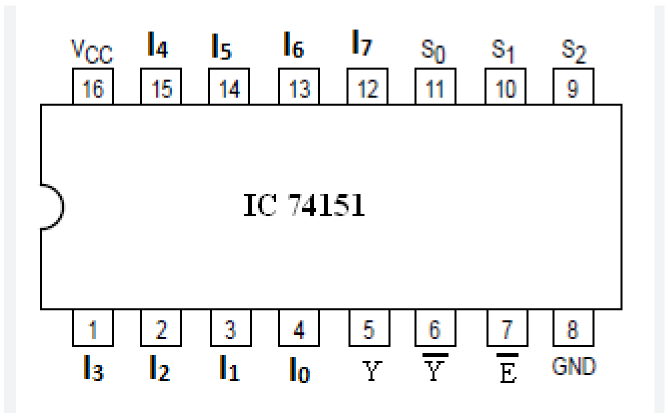
Data inputs of a multiplexer in terms of D, D’, logic ‘1’ & logic ‘0’ are

$I_0 = I_1 = I_5 = D$; $I_2 = D'$; $I_3 = I_4 = 0$; $I_6 = I_7 = 1$

## LOGIC DIAGRAM



## Pin Diagram of IC 74x151:



## PROCEDURE:

1. Insert the appropriate IC’s into the Breadboard.
2. Make connections as shown in the circuit diagram.
3. Provide the input data via input switches.
4. Feed the logic 0 (0V) or logic 1 (5V) in different combinations at the inputs according to truth table.
5. Observe the output on output LEDs.
6. Give all possible combinations of inputs and note down the output.
7. Verify the operation of the circuit using truth table.

## RESULT:

Thus the given four variable Boolean function is implemented using 8:1 multiplexer and verified its operation using truth table.

| Expt. No:9 | VERIFICATION OF STATE TABLES OF SR, JK, D & T-FLIP-FLOPS |
|---|---|
| Dt: | |

**AIM:** To study the operation of SR, JK, D and T flip-flops and verify their state tables.

**APPARATUS REQUIRED:**

| SL No. | COMPONENT | SPECIFICATION | QTY |
|---|---|---|---|
| 1 | NAND | 7400 | 1 |
| 2 | 7410 (3-input NAND gate) | 7410 | 1 |
| 3 | DIGITAL IC TRAINER KIT | | 1 |
| 4 | CONNECTING WIRES | | |

**THEORY:**

Flip-flops are the basic building blocks of sequential circuits. The clocked flip-flops change their o/p state depending upon inputs at certain interval of time synchronized with the clock pulse applied to it.

Different types of Flip-flops are SR, JK, D & T. Their operations are described by the respective truth tables.

**TRUTH TABLES:**

**SR FLIP-FLOP:**

| S | R | $Q_n$ | $Q_{n+1}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | ? |
| 1 | 1 | 1 | ? |

Characteristic equation of SR flip-flop is $Q_{n+1} = S + R'Q$

Alternatively, truth table of SR flip-flop can be specified as

| S | R | $Q_{n+1}$ |
|---|---|---|
| 0 | 0 | $Q_n$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | ? |

**JK FLIP-FLOP:**

| J | K | $Q_n$ | $Q_{n+1}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | Race around |
| 1 | 1 | 1 | Race around |

Characteristic equation of JK flip-flop is $Q_{n+1} = JQ' + KQ$

Alternatively, truth table of JK flip-flop can be specified as

| J | K | $Q_{n+1}$ |
|---|---|---|
| 0 | 0 | $Q_n$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | Race around |

**D FLIP-FLOP:**

| D | $Q_n$ | $Q_{n+1}$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Characteristic equation of D flip-flop is $Q_{n+1} = D$

Alternatively, truth table of D flip-flop can be specified as

| D | $Q_{n+1}$ |
|---|---|
| 0 | 0 |
| 1 | 1 |

**T FLIP-FLOP:**

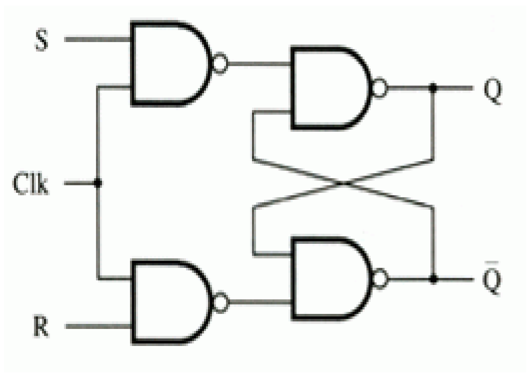
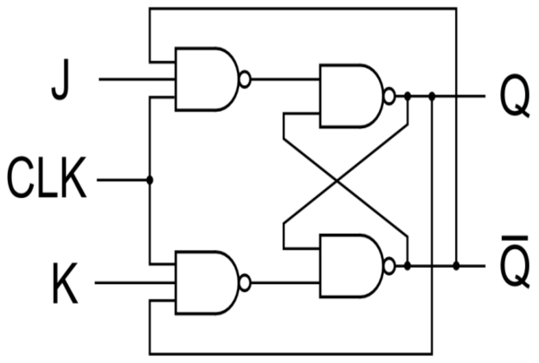
| T | $Q_n$ | $Q_{n+1}$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

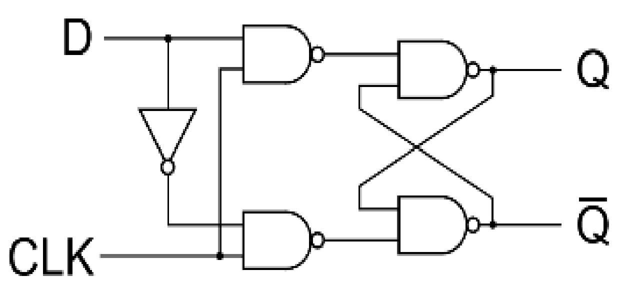
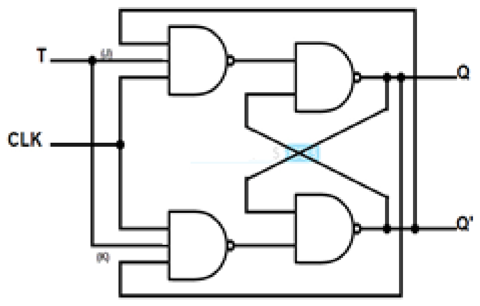
Characteristic equation of T flip-flop is $Q_{n+1} = T \oplus Q$

Alternatively, truth table of T flip-flop can be specified as

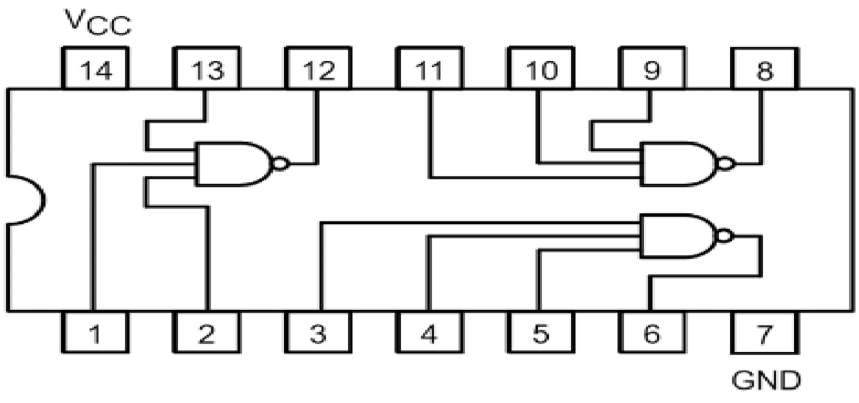
| T | $Q_{n+1}$ |
|---|---|
| 0 | $Q_n$ |
| 1 | $Q_n'$ |

**LOGIC DIAGRAM:**

| **SR FLIP FLOP USNIG NAND gates** | **JK FLIP FLOP USNING NAND gates** |
|---|---|
|  |  |

| D FLIP FLOP USNIG NAND gates | T FLIP FLOP USNING NAND gates |
|---|---|
|  |  |

74LS10 Pinout



**PROCEDURE:**

1. Insert the appropriate IC's into the Breadboard.
2. Make connections as shown in the circuit diagram.
3. Connect Pin-14 of all ICs to +5V and Pin-7 to ground.
4. Provide the input data via input switches.
5. Feed the logic 0 (0V) or logic 1 (5V) in different combinations at the inputs according to truth table.
6. Observe the output on output LEDs.
7. Give all possible combinations of inputs and note down the output.
8. Verify the operation of the circuits using truth table.

**RESULT:**

Thus RS, JK, D & T Flip-Flops are constructed using NAND gates and verified their operation using truth tables.

| Expt. No: 10 | 3-BIT RIPPLE COUNTER USING JK FLIP-FLOPS |
|---|---|
| Dt: | |

**AIM:**

To design and verify the operation of a 3-bit Ripple counter using JK Flip-flops.

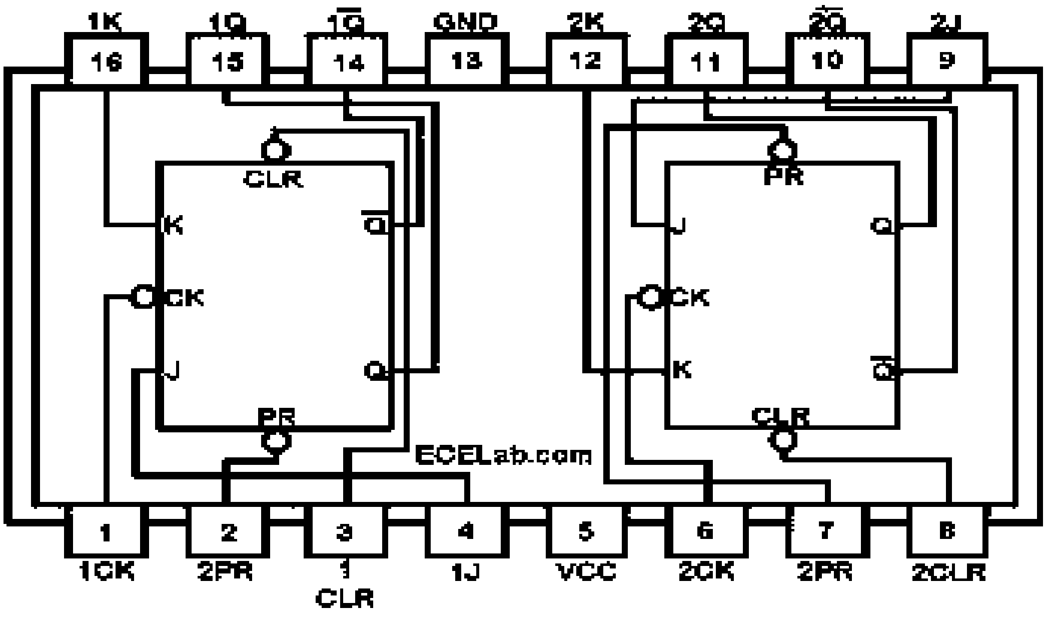
**APPARATUS REQUIRED:**

| SL No. | COMPONENT | SPECIFICATION | QTY |
|---|---|---|---|
| 1 | JK FLIP-FLOPS | IC7476 | 2 |
| 2 | DIGITAL IC TRAINER KIT | | 1 |
| 3 | CONNECTING WIRES | | |

**THEORY:**

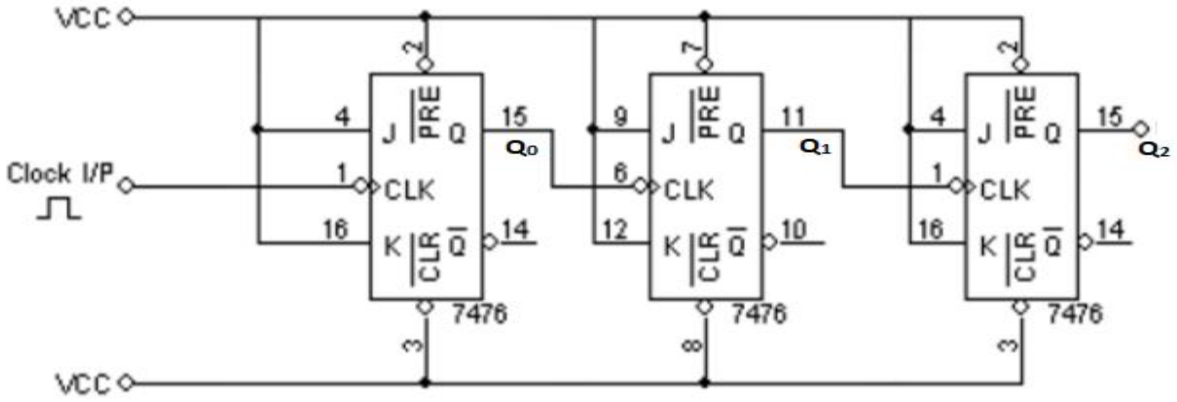
A counter is a register capable of counting number of clock pulses arriving at its clock input. A specified sequence of states appears as counter output. There are two types of counter, synchronous and asynchronous counters. In synchronous counters, common clock is given to all flip-flops. In asynchronous counters, first flip flop is clocked by external pulse and then each successive flip flop is clocked by the output of previous stage. Asynchronous counters are also called as Ripple counters.

**PIN DIAGRAM OF IC7476:**

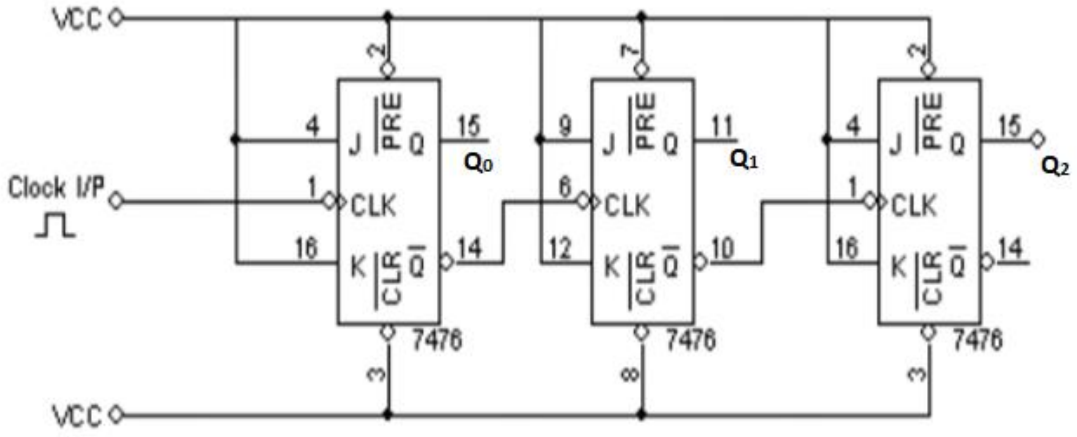


**LOGIC DIAGRAM:**

**3-bit Asynchronous (Ripple) up counter:**



**3-bit Asynchronous (Ripple) down counter:**



**TRUTH TABLE:**

| **3-bit up counter** | | | | **3-bit down counter** | | | |
|---|---|---|---|---|---|---|---|
| **CLK** | **Q2** | **Q1** | **Q0** | **CLK** | **Q2** | **Q1** | **Q0** |
| **0** | **0** | **0** | **0** | **0** | **1** | **1** | **1** |
| **1** | **0** | **0** | **1** | **1** | **1** | **1** | **0** |
| **2** | **0** | **1** | **0** | **2** | **1** | **0** | **1** |
| **3** | **0** | **1** | **1** | **3** | **1** | **0** | **0** |
| **4** | **1** | **0** | **0** | **4** | **0** | **1** | **1** |
| **5** | **1** | **0** | **1** | **5** | **0** | **1** | **0** |
| **6** | **1** | **1** | **0** | **6** | **0** | **0** | **1** |
| **7** | **1** | **1** | **1** | **7** | **0** | **0** | **0** |

**PROCEDURE:**

1. Insert the appropriate IC’s into the Breadboard.
2. Make connections as shown in the circuit diagram.
3. Provide the input data via input switches.
4. Feed the logic 0 (0V) or logic 1 (5V) in different combinations at the inputs according to truth table.
5. Observe the output on output LEDs.
6. Apply the clock signal and observe the states of a counter.
7. Verify the operation of the circuit using truth table.

**RESULT:**

Thus 3-bit Asynchronous up and down counters are constructed and counter states are verified using the truth table.

| Expt. No: 11 | 3-BIT SYNCHRONOUS COUNTER USING T FLIP-FLOPS |
|---|---|
| Dt: | |

**AIM:**

To design and verify the operation of a 3-bit Synchronous counter using T Flip-flops.

**APPARATUS REQUIRED:**

| SL No. | COMPONENT | SPECIFICATION | QTY |
|---|---|---|---|
| 1 | JK FLIP-FLOPS | IC7476 | 2 |
| 2 | AND GATE | IC 7408 | 1 |
| 3 | DIGITAL IC TRAINER KIT | | 1 |
| 4 | CONNECTING WIRES | | |

**THEORY:**

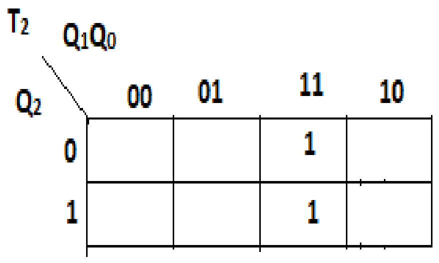
As all the flip-flops do not change states simultaneously in asynchronous counter, spike occur at the output. To avoid this, strobe pulse is required. Because of the propagation delay the operating speed of asynchronous counter is low. This problem can be solved by triggering all the flip-flops in synchronous with the clock signal and such counters are called as synchronous counters. An ‘N’ bit Synchronous binary counter consists of ‘N’ flip-flops. It is capable of counting $2^N$ clock pulses.

Design Procedure of synchronous counters is

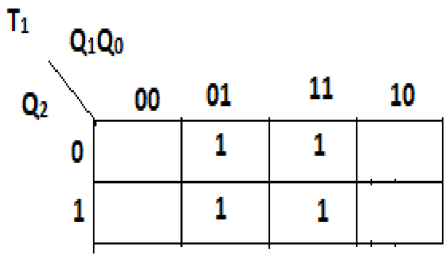
1. Determine the number of Flip-flops needed to support the counting sequence.
2. Build a State Transition Diagram. Be sure to include all states.
3. Build a State/Excitation Truth Table.
4. Obtain the simplified expressions for flip-flop inputs using K-Map.
5. Implement the Synchronous Counter using flip-flops and logic gates.

**TRUTH TABLE:**

| Present State | | | Next State | | | Flip-flop Inputs | | |
|---|---|---|---|---|---|---|---|---|
| Q2 | Q1 | Q0 | Q2 | Q1 | Q0 | $T_2$ | $T_1$ | $T_0$ |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |



**$T_2 = Q_1Q_0$**

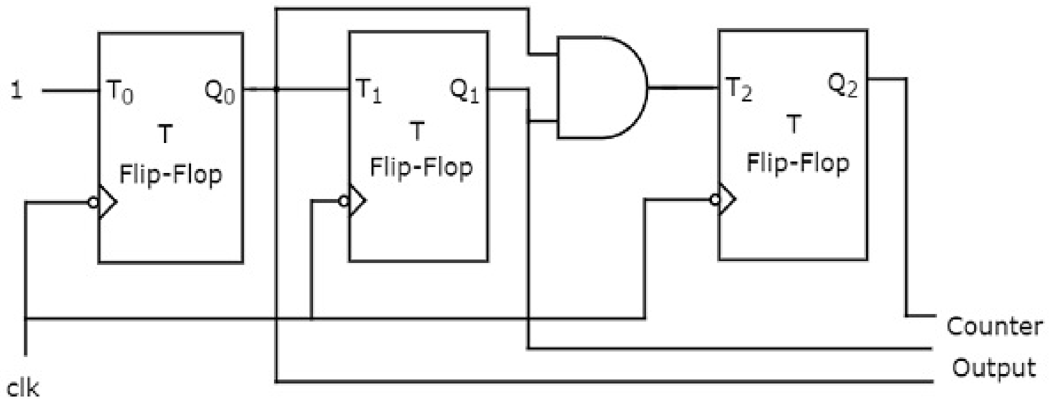


**$T_1 = Q_0$**

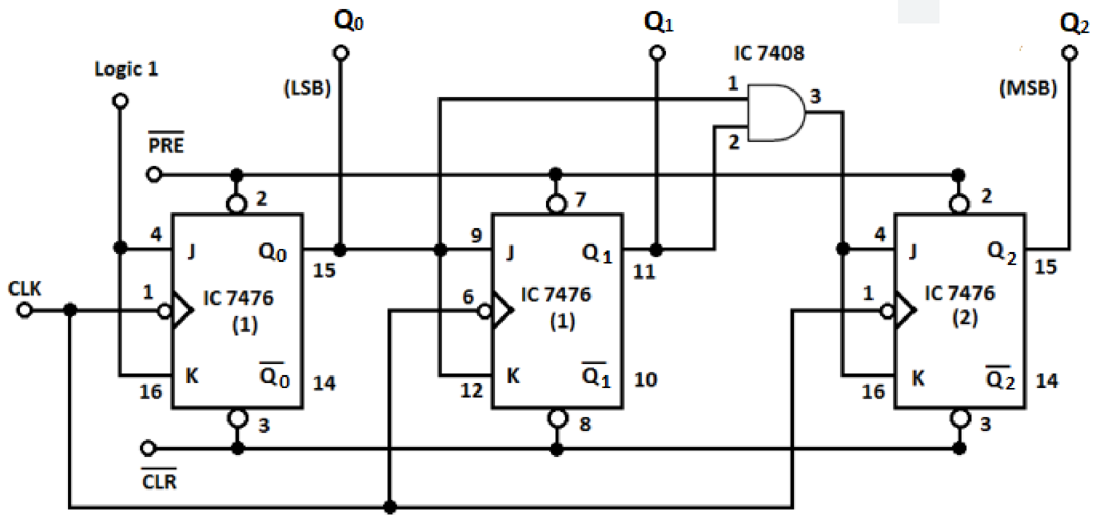
| $T_0$ $Q_2$ \ $Q_1Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

$T_0 = 1$

**LOGIC DIAGRAM:**



**Logic Diagram using IC74x76:**



**PROCEDURE:**

1. Insert the appropriate IC’s into the Breadboard.
2. Make connections as shown in the circuit diagram.
3. Provide the input data via input switches.
4. Feed the logic 0 (0V) or logic 1 (5V) in different combinations at the inputs according to truth table.
5. Observe the output on output LEDs.
6. Apply the clock signal and observe the states of a counter.
7. Verify the operation of the circuit using truth table.

**RESULT:**

Thus 3-bit synchronous counter is designed and constructed using T Flip-flops and counter states are verified using the truth table.

|  |  |
|---|---|
| **Expt. No: 12** | **4-Bit SHIFT REGISTER** |
| **Dt:** |  |

**AIM:**

To design and verify the operation of a 4-Bit Serial in Serial out Shift Register

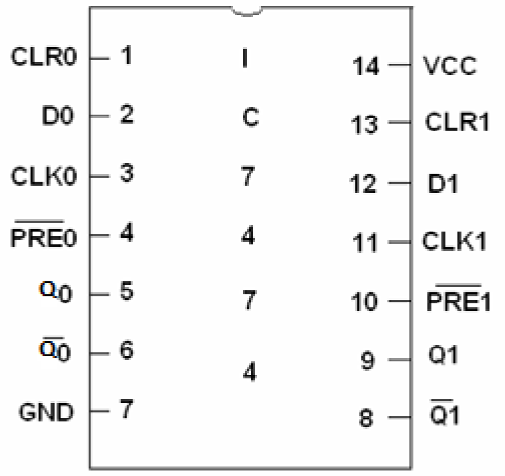
**APPARATUS REQUIRED:**

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

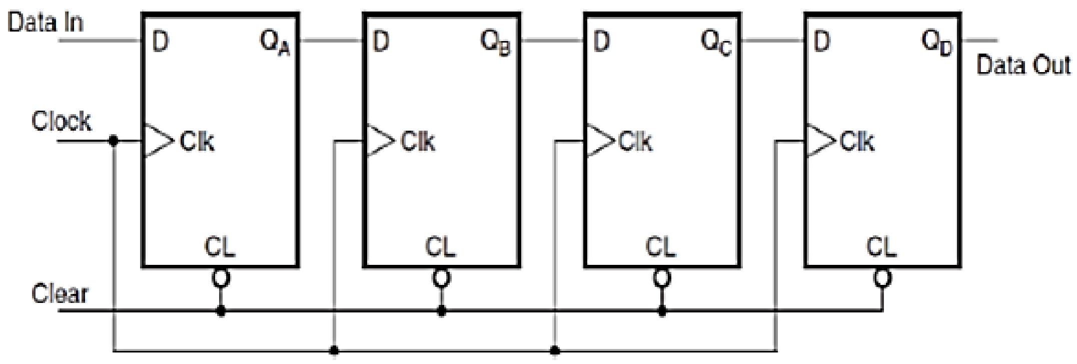
**THEORY:**

A register is simply a group of flip-flops that can be used to store a binary number. There must be one flip-flop for each bit in the binary number. 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 consists of a D Flip-flops 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.

**PIN DIAGRAM OF IC7474:**



**LOGIC DIAGRAM:**

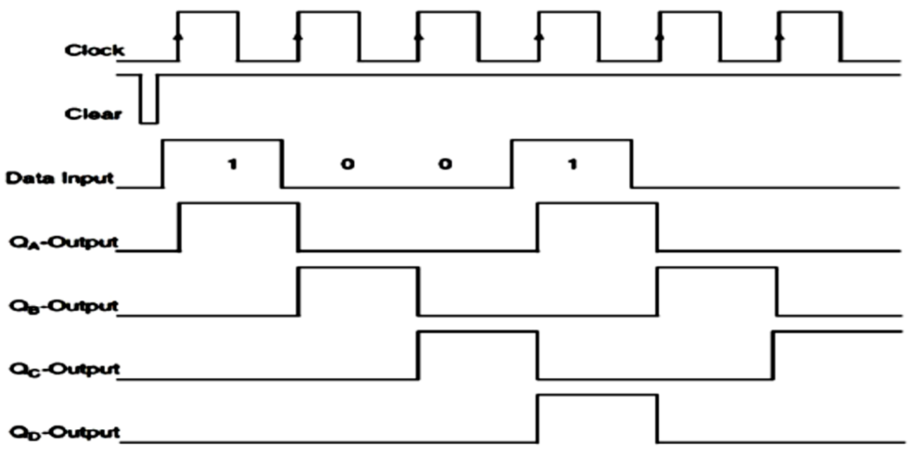


**TRUTH TABLE:**

Let Data In = 1001

Contents of four bit serial–in serial–out shift register for the first eight clock cycles.

| **Clock** | **$Q_A$** | **$Q_B$** | **$Q_C$** | **$Q_D$** |
|---|---|---|---|---|
| Initial contents | 0 | 0 | 0 | 0 |
| After first clock transition | 1 | 0 | 0 | 0 |
| After second clock transition | 0 | 1 | 0 | 0 |
| After third clock transition | 0 | 0 | 1 | 0 |
| After fourth clock transition | 1 | 0 | 0 | 1 |
| After fifth clock transition | 0 | 1 | 0 | 0 |
| After sixth clock transition | 0 | 0 | 1 | 0 |
| After seventh clock transition | 0 | 0 | 0 | 1 |
| After eighth clock transition | 0 | 0 | 0 | 0 |



**PROCEDURE:**

1. Insert the appropriate IC’s into the Breadboard.
2. Make connections as shown in the circuit diagram.
3. Provide the input data via input switches.
4. Feed the logic 0 (0V) or logic 1 (5V) in different combinations at the inputs according to truth table.
5. Observe the output on output LEDs.
6. Apply the clock signal and observe the states of a Register.
7. Verify the operation of the circuit using truth table.

**RESULT:**

Thus the serial in serial out shift register is implemented using IC7474 and verified successfully.