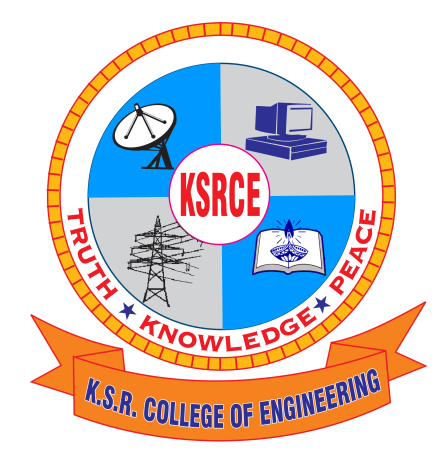
**K.S.R College of Engineering, Tiruchengode-637215**

(An Autonomous Institution Affiliated to Anna University, Chennai)

**Department of Computer Science and Engineering**

**DIGITAL SYSTEMS LABORATORY MANUAL**

****

|  |  |  |
| --- | --- | --- |
| **Register Number** | : |  |
| **Name of the Student** | : |  |
| **Section** | : |  |
| **Year** | : | II |
| **Semester** | : | III |
| **Academic Year** | : | 2020 – 2021 |
| **Subject Code** | : | 18EC325 |
| **Laboratory Name** | : | Digital Systems Laboratory |

|  |  |
| --- | --- |
| STAFF INCHARGE | HOD |

**LIST OF EXPERIMENTS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EX. NO** | **DATE** | **NAME OF EXPRIMENT** | **SIGNATURE** | **MARKS** |
| 1. |  | Study of Logic Gates |  |  |
| 2. |  | Verification of Boolean theorem using digital Logic Gates |  |  |
| 3. |  | Design and Implementation of Half Adder and Full Adder |  |  |
| 4. |  | Design and Implementation of Half Subtractor and Full Subtractor |  |  |
| 5. |  | Design and Implementation of 4 bit binary adder/subtractor |  |  |
| 6. |  | Design and Implementation of Magnitude Comparator |  |  |
| 7. |  | Design and Implementation of Code Converters |  |  |
| 8. |  | Design and Implementation of Parity Checker and Generators |  |  |
| 9 |  | Design and Implementation of Multiplexer and De multiplexer |  |  |
| 10 |  | Design and Implementation of Shift Registers |  |  |
| 11 |  | Design and Implementation of 3 bit synchronous up and down counter |  |  |
| 12 |  | Simulation of Combinational circuits using VHDL |  |  |
| 13 |  | Simulation of Sequential circuits using VHDL |  |  |



**Expt.No.1:** **STUDY OF BASIC LOGIC GATES**

Aim:

To verify the truth table of basic digital IC’s of AND, OR, NOT, NAND, NOR, EX-OR gates

Apparatus required:

|  |  |  |  |
| --- | --- | --- | --- |
| Sl.No | Name of the Apparatus | Range | Quantity |
| 1. | Digital IC trainer kit |  | 1 |
| 2. | AND gate | IC 7408 | 1 |
| 3. | OR gate | IC 7432 | 1 |
| 4. | NOT gate | IC 7404 | 1 |
| 5. | NAND gate | IC 7400 | 1 |
| 6. | NOR gate | IC 7402 | 1 |
| 7. | EX-OR gate | IC 7486 | 1 |
| 8. | Connecting wires |  | As required |



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

A NOT gate is the physical realization of the complementation operation. 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

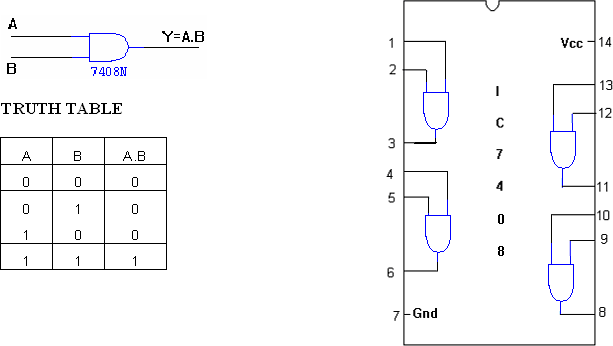
The NOR gate is a contraction of OR-NOT. The output is high when both inputs are low. The output is low when one or both inputs are high.

EX-OR gate

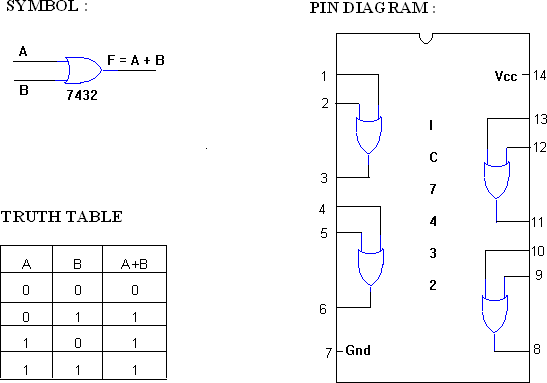
An Ex-OR gate performs the following Boolean function, A  B = ( A . B’ ) + ( A’ . B ). It is similar to OR gate but excludes the combination of both A and B being equal to one. The exclusive OR is a function that give an output signal ‘0’ when the two input signals are equal either ‘0’ or ‘1’.

**AND GATE:**

**SYMBOL: PINDIAGRAM:**

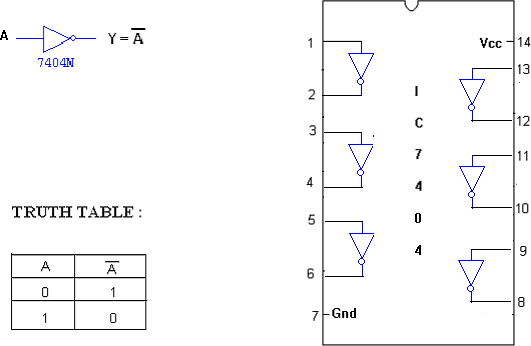


**OR GATE:**



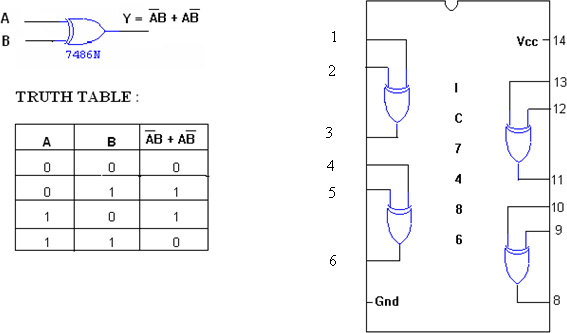
**NOT GATE:**

**SYMBOL: PINDIAGRAM:**



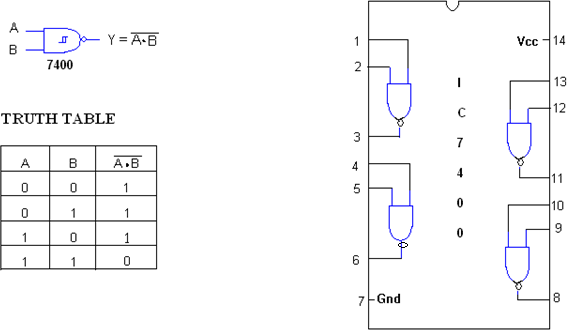
**X-OR GATE :**

**SYMBOL: PIN DIAGRAM:**

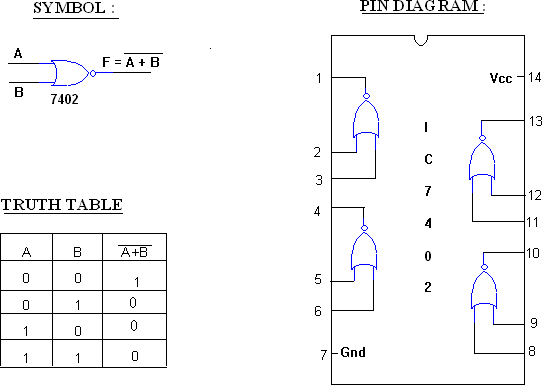


**2 INPUT NANDGATE:**

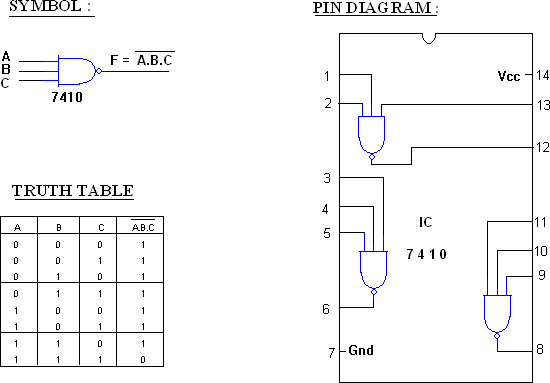
**SYMBOL: PINDIAGRAM:**



**NOR GATE**



**3 INPUT NAND GATE:**



Procedure:

1. Connections are given as per the circuit diagram.
2. For all the IC’s 7th pin is grounded and 14th pin is given +5 supply.
3. Apply the inputs and verify the truth table for all gates.

Result:

The truth tables of all the basic logic gates were verified.

**Expt.No.2: Verification of Boolean theorem using digital Logic Gates**

Aim: To verification of Boolean theorems using logic gates

Apparatus required:

|  |  |  |  |
| --- | --- | --- | --- |
| Sl.No | Name of the Apparatus | Range | Quantity |
| 1. | Digital IC trainer kit |  | 1 |
| 2. | AND gate | IC 7408 | 1 |
| 3. | OR gate | IC 7432 | 1 |
| 4. | NOT gate | IC 7404 | 1 |
| 5. | NAND gate | IC 7400 | 1 |
| 6. | NOR gate | IC 7402 | 3 |
| 7. | EX-OR gate | IC 7486 | 1 |
| 8. | Connecting wires |  | As required |

**Theory:**

BASIC Boolean Laws

1. Commutative Law

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

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

1. Associative Law

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

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

1. Distributive Law

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

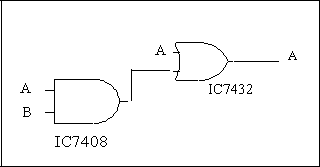
* + 1. A+(B.C) = (A+B).(A+C)
    2. A.(B+C) = (A.B)+(A.C)

1. Absorption Law
   1. A+AB = A
   2. A+AB =A+B
2. Idempotent Law
   1. A+A=A
      1. A.A=A
3. Complementary Law
   1. A+A' = 1
   2. A.A' = 0
4. De Morgan’s Theorem
   * + 1. The complement of the sum is equal to the sum of the product of the individual complements.

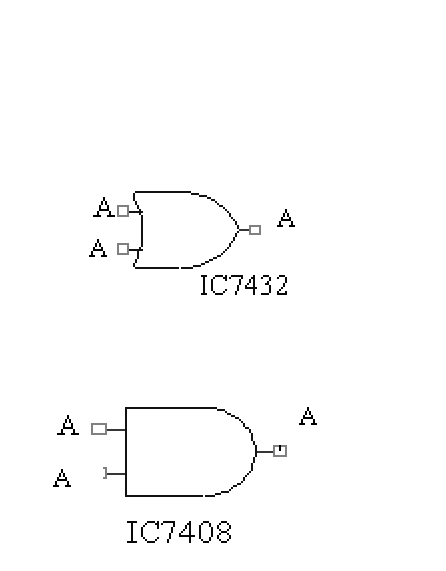
A+B = A.~~B~~

1. The complement of the product is equal to the sum of the individual complements. A.B = A+

Design

1. Absorption Law

A+AB = A

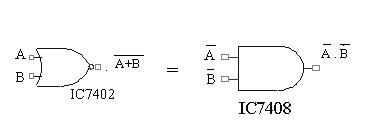
1. Involution (or) Double complement Law

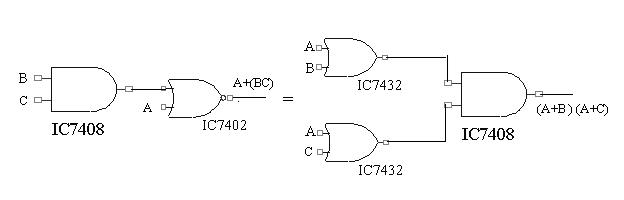
A = A

1. Idempotent Law
   1. A+A=A
   2. A.A=A

4. Demorgan’s Law

A+B = A.B



1. Distributive Law

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

**Procedure:**

1. Obtain the required IC along with the Digital trai er kit.
2. Connect zero volts to GND pin and +5 volts to Vcc .
3. Apply the inputs to the respective input pins.
4. Verify the output with the truth table.

**Result:**

Thus the above stated Boolean laws are verified.

**Expt.No.3:** **Design and Implementation of Half Adder and Full Adder**

Aim:

To design and verify the truth table of the Half Adder & Full Adder circuits

Apparatus required:

|  |  |  |  |
| --- | --- | --- | --- |
| S. No. | Name of the Apparatus | Range | Quantity |
| 1. | Digital IC trainer kit |  | 1 |
| 2. | AND gate | IC 7408 | 1 |
| 3. | OR gate | IC 7432 | 1 |
| 4. | NOT gate | IC 7404 | 1 |
| 5. | EX-OR gate | IC 7486 | 1 |
| 6. | Connecting wires | As required |  |

Theory:

The most basic arithmetic operation is the addition of two binary digits. There are four possible elementary operations, namely,

0+0=0

0+1=1

1+0=1

1 + 1 = 0 (w th 1 as carry)

The first three operations produce a sum of whose le gth is one digit, but when the last operation is performed the sum is two digits. The higher significant bit of this result is called a carry and lower significant bit is called the sum.

Half adder:

A combinational circuit which performs the ddition of two bits is called half adder. The input variables designate the augend and the addend bit, whereas the output variables produce the sum and carry bits.

Full adder:

A combinational circuit hich performs the arithmetic sum of three input bits is called full adder. The three input bits include t o significant bits and a previous carry bit. A full adder circuit can be implemented with two half adders and one OR gate. From the truth table, the expression for sum and carry bits of the output can be obtained as,

SUM = A’B’C + A’BC’ + AB’C’ + ABC

CARRY = A’BC + AB’C + ABC’ +ABC

Half Adder

Truth table:

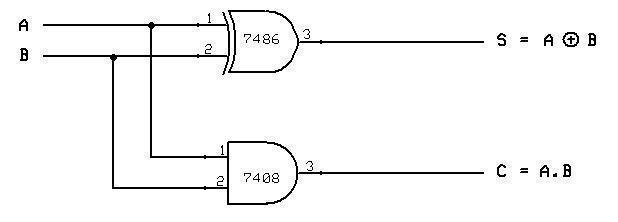
|  |  |  |  |
| --- | --- | --- | --- |
| Input | | Output | |
| A | B | S | C |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 |

From the truth table the expression for sum and carry bits of the output can be obtained as, Sum,

S=A  B

Carry, C = A .

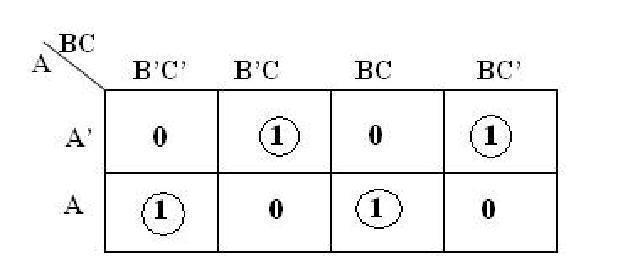
Circuit diagram:

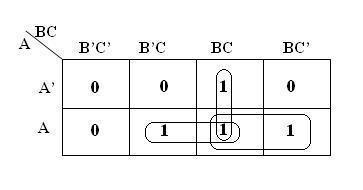


|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input | | Output | | |
| **A** | **B** | **C** | **S** | **C** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

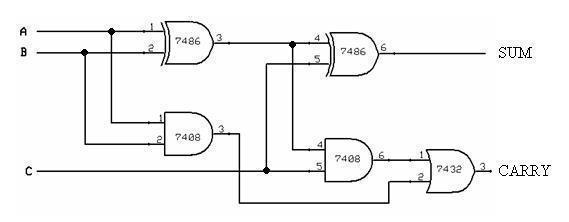
Using Karnaugh maps the reduced expression for the output bits can be obtained as,

Sum: Carry:





SUM = A’B’C + A’BC’ + AB’C’ + ABC = A  B  C CARRY = AB + AC + BC

**Logic Diagram:**

Procedure:

1. Connections are given as per the circuit diagrams.
2. For all the IC’s 7th pin is grounded and 14th pin is given +5 V supply.
3. Apply the inputs and verify the truth table for the half adder and full adder circuits.

**Result:**

The design of the half adder and full adder circuits was done and their truth tables were verified.

**Expt.No.4: Design and Implementation of Half Subtractor and Full Subtractor**

Aim:

To design and verify the truth table of the half subtractor & full subtractor circuits

Apparatus required:

|  |  |  |  |
| --- | --- | --- | --- |
| Sl.No | Name of the Apparatus | Range | Quantity |
| 1. | Digital IC trainer kit |  | 1 |
| 2. | AND gate | IC 7408 | 1 |
| 3. | OR gate | IC 7432 | 1 |
| 4. | NOT gate | IC 7404 | 1 |
| 5. | EX-OR gate | IC 7486 | 1 |
| 6. | Connecting wires | As required |  |

Theory:

The subtraction of two binary digits has four possible operations. In all operations, each subtrahend bit is subtracted from the minuend bit. In case of the second operation the minuend bit is smaller than the subtrahend bit, hence 1 is borrowed.

Half subtractor:

A combinational circuit which performs the subtraction of two bits is called half subtractor. The input variables designate the minuend and the subtrahend bit, whereas the output variables produce the difference and borrow bits.

Full subtractor:

A combinational circuit which performs the subtraction of three input bits is called full subtractor. The three input bits include two significant bits and a previous borrow bit. A full subtractor circuit can be implemented with two half subtractors and one OR gate.

From the truth table the expression for difference and borrow bits of the output can be obtained as,

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

Borrow, BORR = A’BC + AB’C + ABC’ +ABC

Half subtractor

Truth table:

|  |  |  |  |
| --- | --- | --- | --- |
| Input | | Output | |
| A | B | Difference | Borrow |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

From the truth table the expression for difference and borrow bits of the output can be obtained as,

Difference, DIFF = A  B

Borrow, BORR = A’. B

Logic diagram:



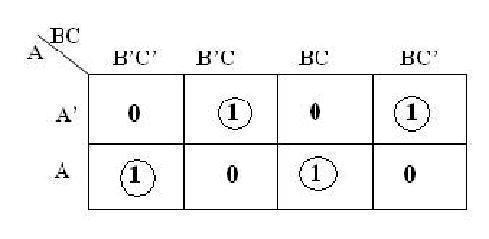
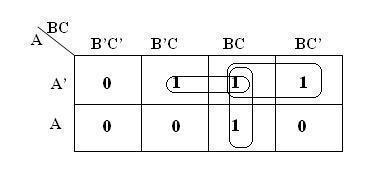
1. Full subtractor Truth table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input | | | Output | |
| 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 |

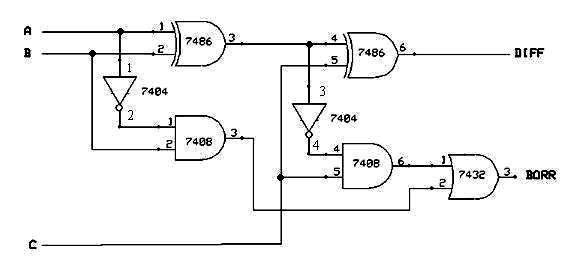


Using Karnaugh maps the reduced expression for the output bits can be obtained as,

Difference Borrow



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

Circuit diagram:

**Result:**

The design of the half subtractor and full subtractor circuits was done and their truth tables were verified

**Expt.No.5:** **Design and Implementation of 4 bit binary adder/subtractor**

Aim:

To design and implement 4-bit adder and subtractor using IC 7483

Apparatus required:

|  |  |  |  |
| --- | --- | --- | --- |
| Sl.No | Name of the Apparatus | Range | Quantity |
| 1. | Digital IC trainer kit |  | 1 |
| 2. | IC | IC 7483 | 1 |
| 3. | NOT gate | IC 7404 | 1 |
| 4. | EX-OR gate | IC 7486 | 1 |
| 5. | Connecting wires |  | As required |

Theory:

4 BIT BINARY ADDER:

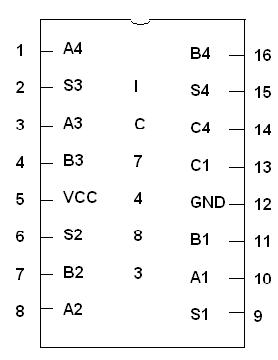
A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of next full adder in chain. The augends bits of ‘A’ and the addend bits of ‘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 C0and it ripples through the full adder to the output carryC4.

4 BIT BINARY SUBTRACTOR:

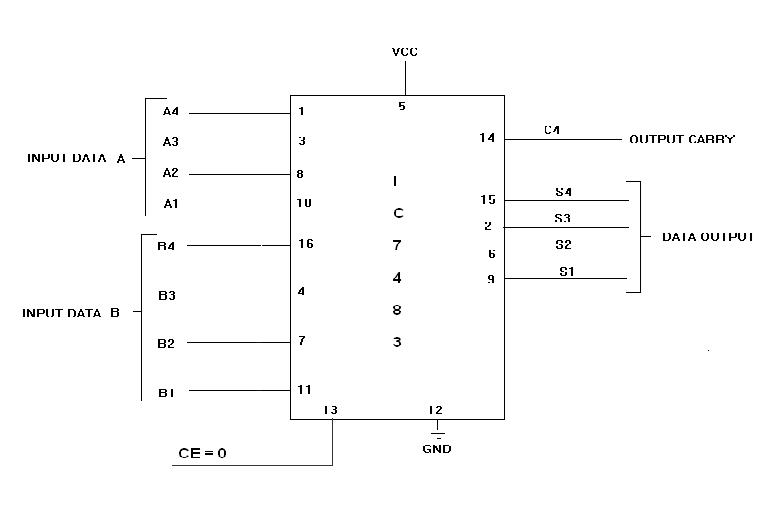
The circuit for subtracting A-B consists of an adder with inverters, placed between each data input ‘B’ and the corresponding input of full adder. The input carry C0 must be equal to 1 when performing subtraction.

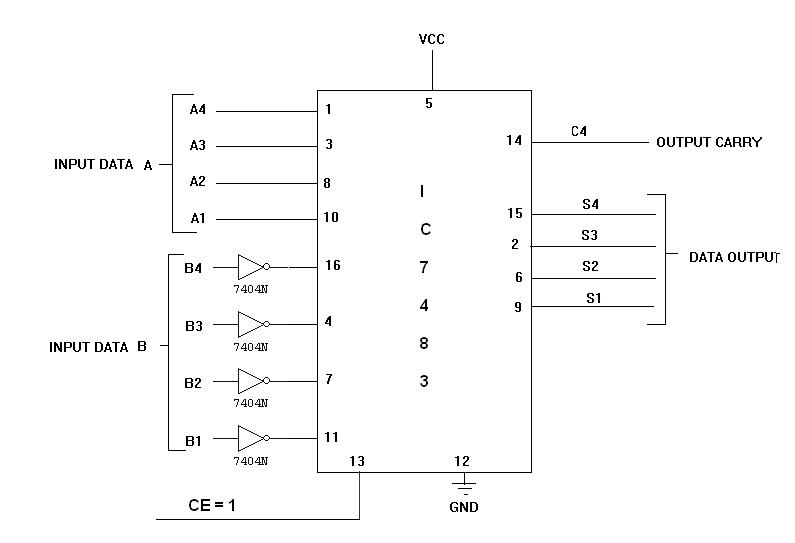
4 BIT BINARY ADDER/SUBTRACTOR:

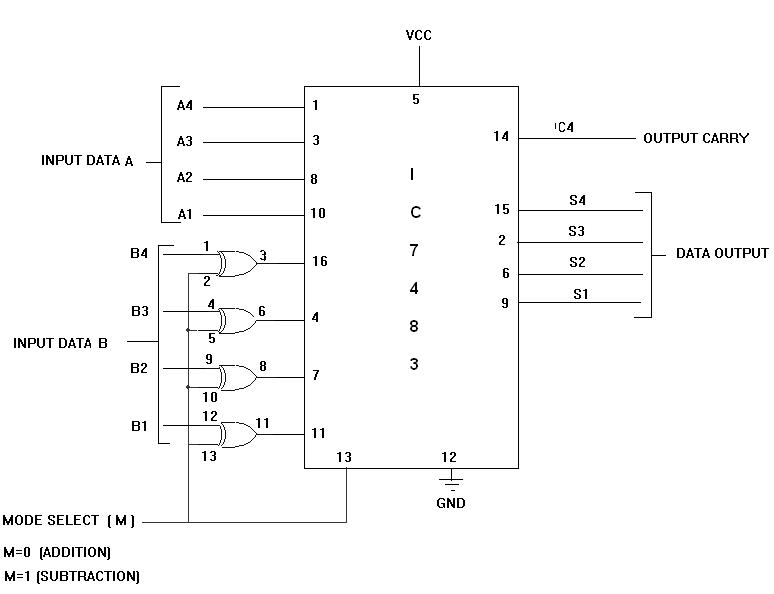
The addition and subtraction operation can be combined into one circuit with one common binary adder. The mode input M controls the operation. When M=0, the circuit is adder circuit. When M=1, it becomes subtractor.PIN Diagram for IC 7483:



Logic Diagram: 4-Bit Binary Diagram:



Logic diagram: 4-Bit Binary Subtractor:



4-Bit Binary Adder /Subtractor

Truth table:

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

Procedure:

1. Connections are given as per the circuit diagrams.

2. Logical inputs ere given as per circuit diagram.

3. Apply the inputs and verify the truth table for the 4-bit adder and subtractor.

**Result:**

The design of the 4-bit Binary adder and l subtractor circuit was done and its truth table was verified.

**Expt.No.6:** **Design and Implementation of Magnitude Comparator**

Aim:

To design, construct and study the performance of 2 bit magnitude comparator

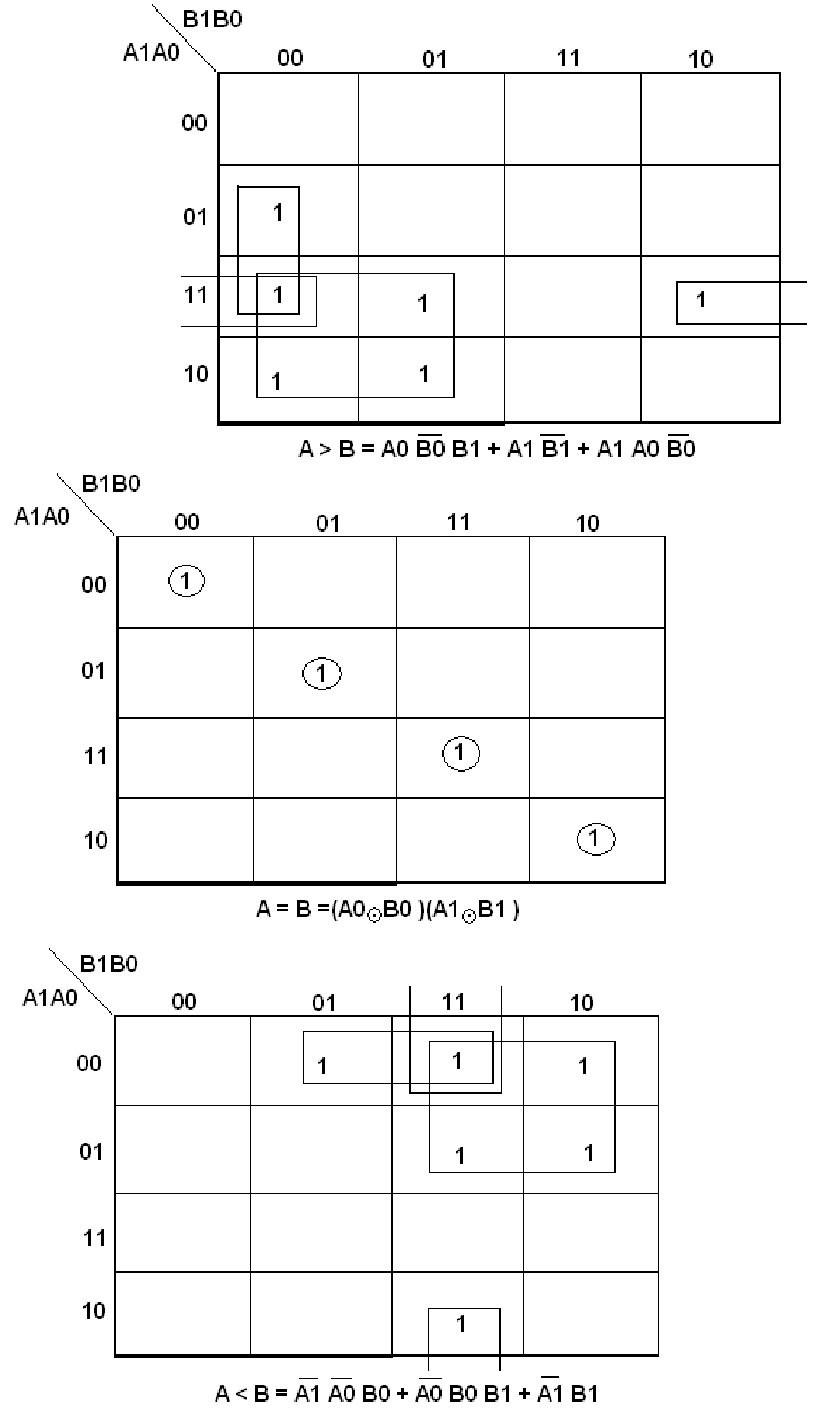
Apparatus required:

|  |  |  |  |
| --- | --- | --- | --- |
| Sl.No | Name of the Apparatus | Range | Quantity |
| 1. | Digital IC trainer kit |  | 1 |
| 2. | AND gate | IC 7408 | 1 |
| 3. | OR gate | IC 7432 | 1 |
| 4. | NOT gate | IC 7404 | 1 |
| 5. | Magnitude comparator | IC 7485 | 2 |
| 6. | EX-OR gate | IC 7486 | 1 |
| 7. | Connecting wires |  | As required |

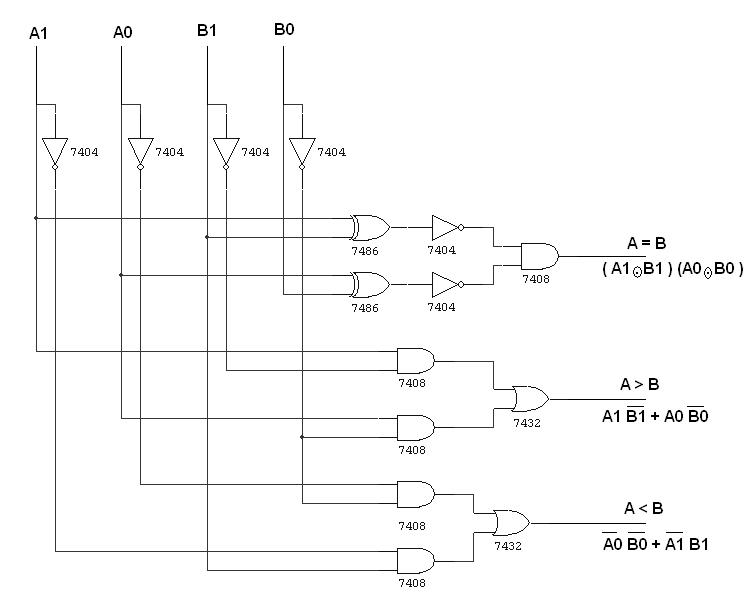
Theory:

The comparison of two numbers is an operator that determines o e umber 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 determines their relative magnitude. The outcome of the comparator is specified by three binary variables that indicate whether A>B, A=B (or) A<B.

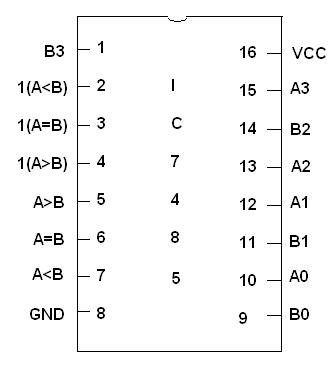
Truth table:

K-map

**Logic Diagram:**

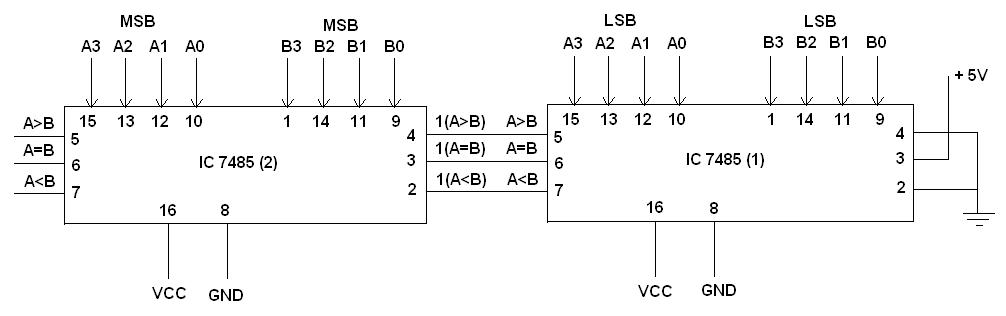


**Pin Diagram for IC 7485:**



**Logic Diagram:**

**8 Bit Magnitude Comparator:**



**Truth table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **A>B** | **A=B** | **A<B** |
| 0000 | 0000 | 0 | 1 | 0 |
| 0001 | 0000 | 1 | 0 | 0 |
| 0000 | 0001 | 0 | 0 | 1 |

**Procedure:**

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

**Result:**

Thus the 2-bit and 8-bit magnitude comparator was designed and verified using the logic gates.

**Expt.No.7: Design and Implementation of Code Converters**

**Aim:**

To design, construct and study the performance of 4-bit different code converters

1. Binary to gray code converter
2. Gray to binary code converter
3. BCD to excess-3 code converter
4. Excess-3 to BCD code converter

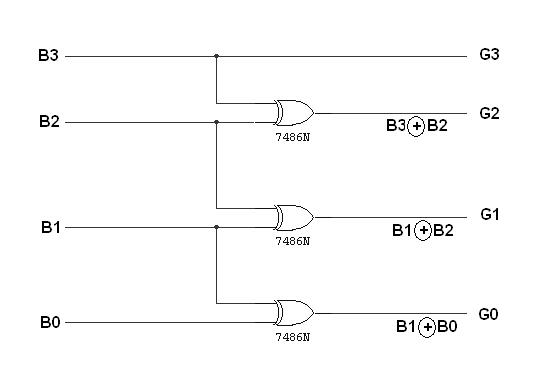
**Apparatus required:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sl.No | Name of the Apparatus | Range | Quantity |
| 1. | Digital IC trainer kit |  | 1 |
| 2. | AND gate | IC 7408 | 1 |
| 3. | OR gate | IC 7432 | 1 |
| 4. | NOT gate | IC 7404 | 1 |
| 5. | Magnitude comparator | IC 7485 | 2 |
| 6. | EX-OR gate | IC 7486 | 1 |
| 7. | Connecting wires |  | As required |

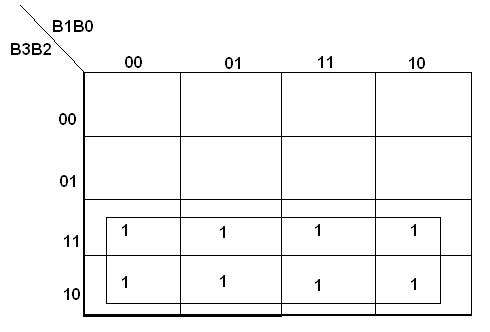
**Theory:**

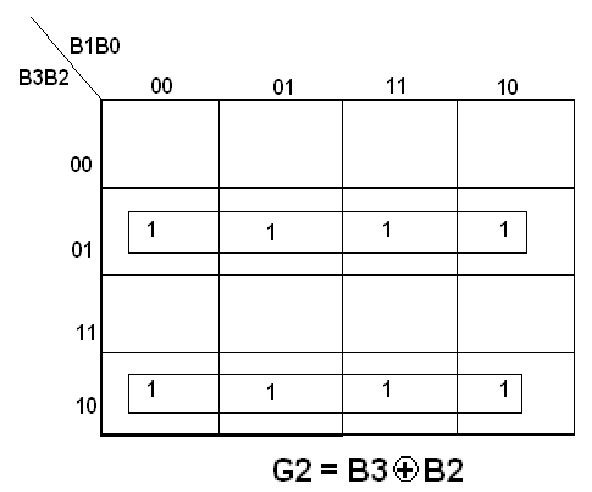
The availability of large variety of codes for the same discrete elements of information results in the use of different codes by different systems. A co version circuit must be inserted between the two systems if each uses different codes for same information. Thus, code converter is a circuit that 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. A code converter is a circuit that makes the two systems compatible even though each uses a different binary code. To convert from binary code to Excess-3 code, the input lines must supply the bit combination of elements as specified by code and the output lines generate the corresponding bit combination of code. Each one of the four maps represents one of the four outputs of the circuit as a function of the four input variables. A two-level logic diagram may be obtained directly from the Boolean expressions derived by the maps. These are various other possibilities for a logic diagram that implements this circuit. Now the OR gate whose output is C+D has been used to implement partially each of three outputs.

**Logic diagram:**

**Binary to gray code converter Logic Diagram**:

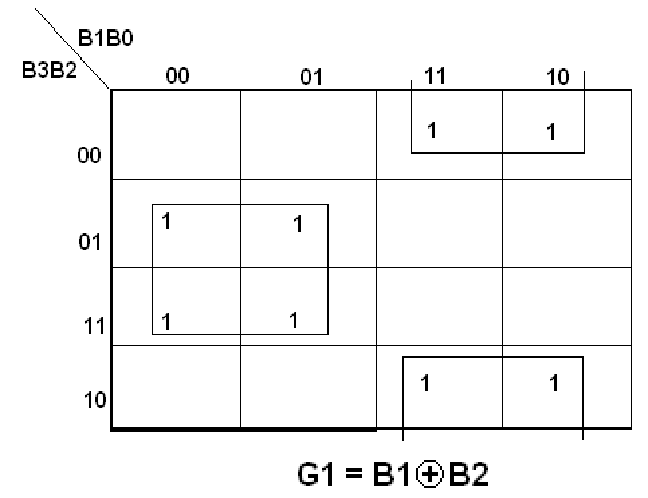
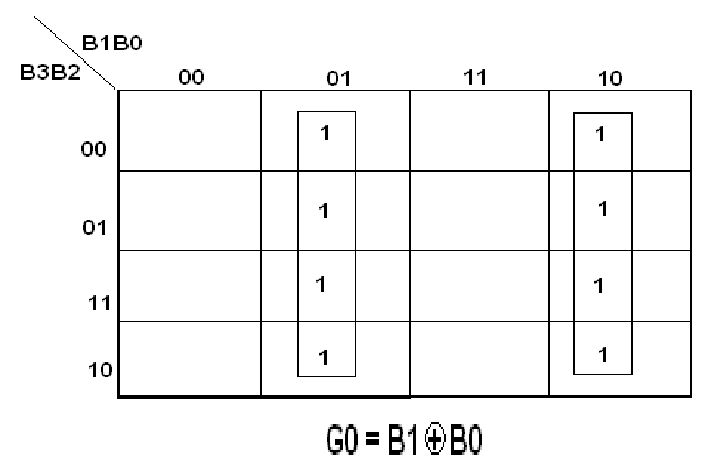
**K map for G3:**  **K map for G2:**





G3=B3

**K map for G1**  **K map for G0:**

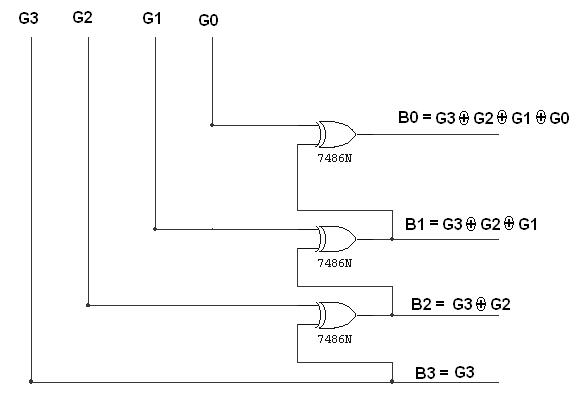


**Truth table:**

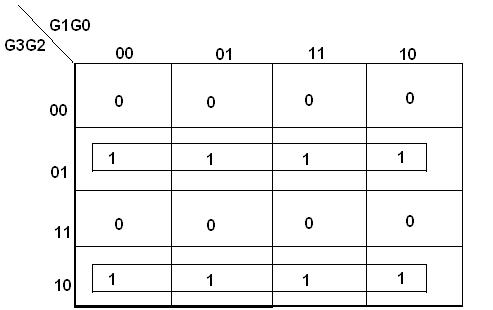
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **0** | **1** | **0** | **0** | **0** | **1** |
| **0** | **0** | **1** | **0** | **0** | **0** | **1** | **1** |
| **0** | **0** | **1** | **1** | **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** |

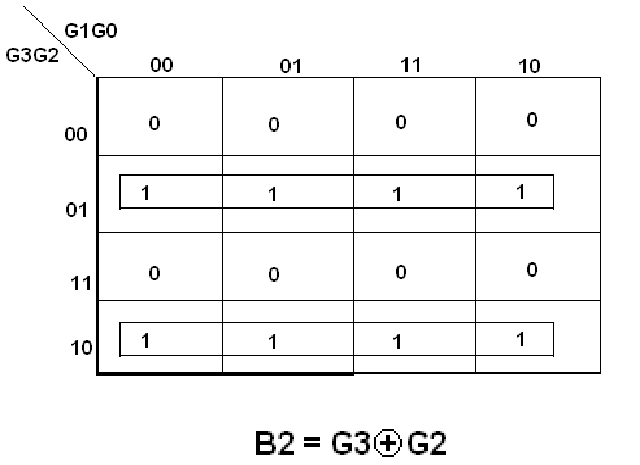
**(ii) Gray to binary code converter**

**Logic Diagram:**



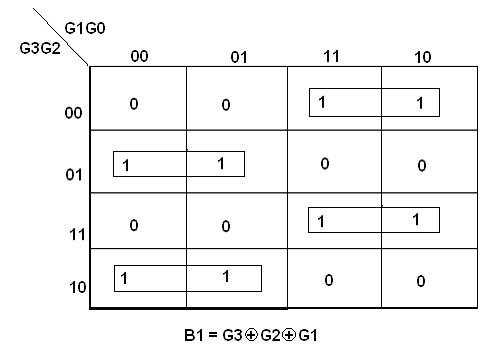
**K map for B3:**  **K map for B2:**

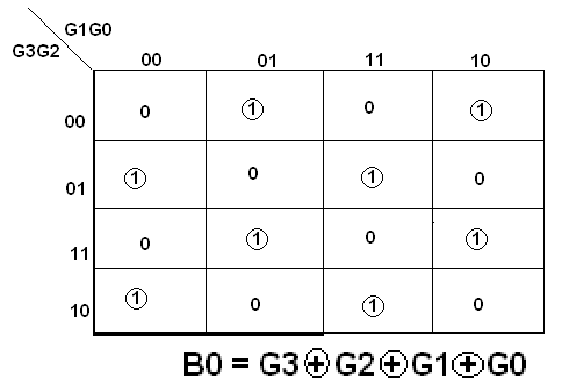




**B3=G3**

**K map for B1:**   **K map for B0:**

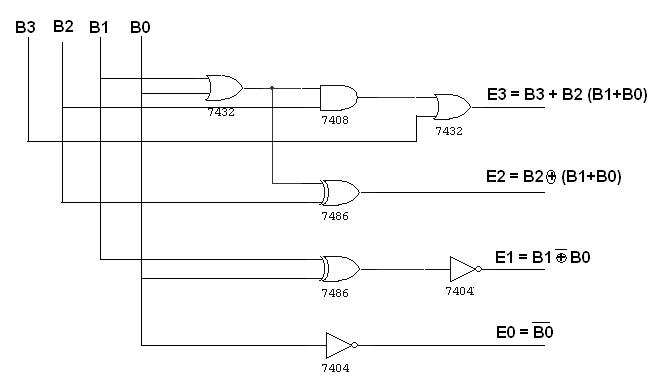




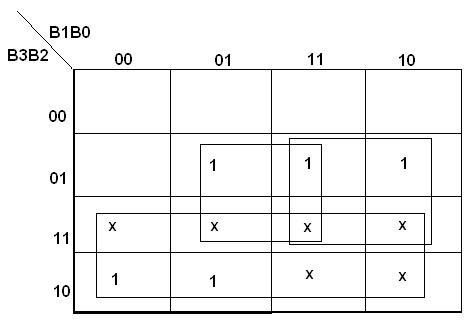
**Truth table:**

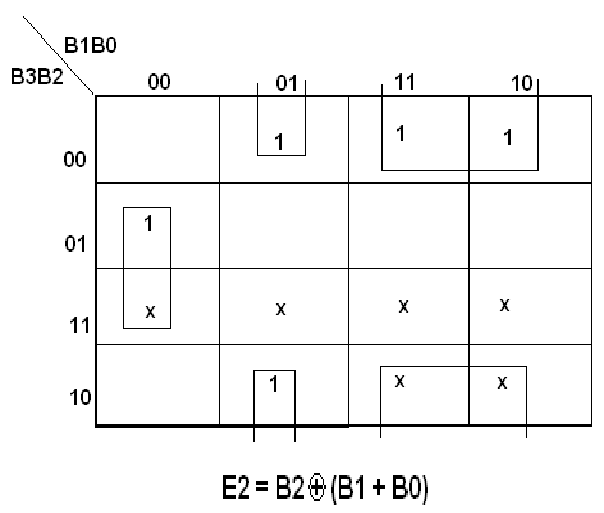
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **G3** | **G2** | **G1** | **G0** | **B3** | **B2** | **B1** | **B0** |
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **0** | **1** | **0** | **0** | **0** | **1** |
| **0** | **0** | **1**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

**BCD to excess-3 code converter Logic Diagram:**

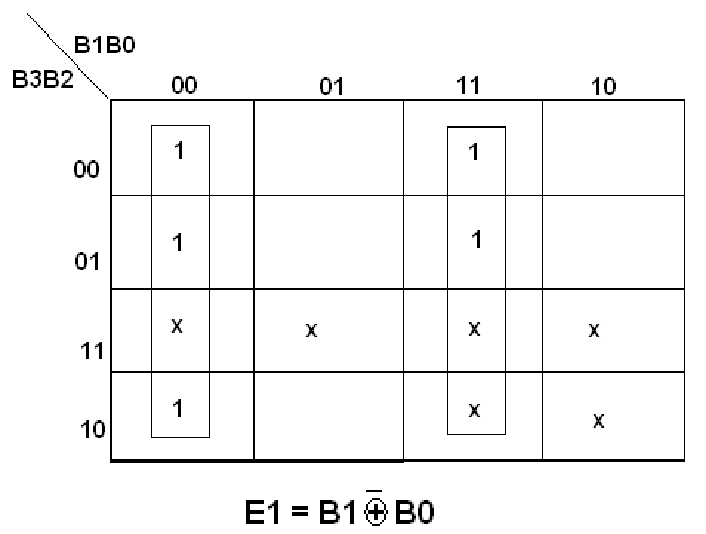


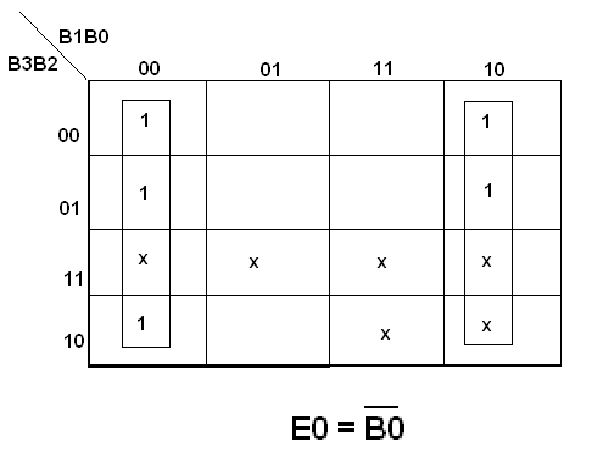
**K map for E3:**   **K map for E2:**





**E3=B3+B2(B0+B1)**

**K map for E1:**   **K map for E0:**

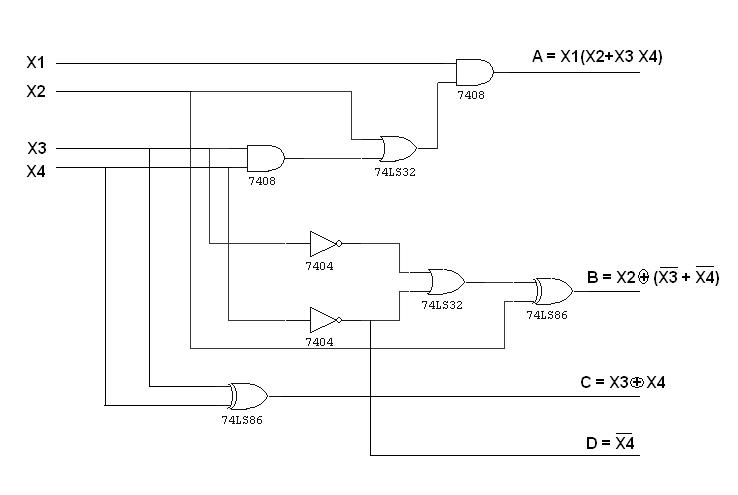


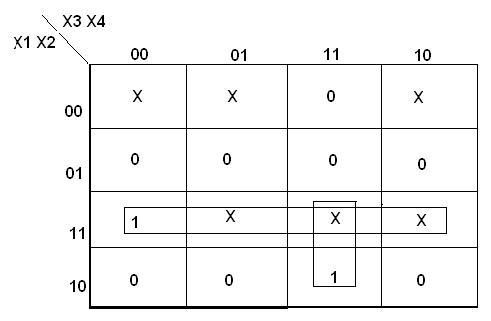
**Truth table:**

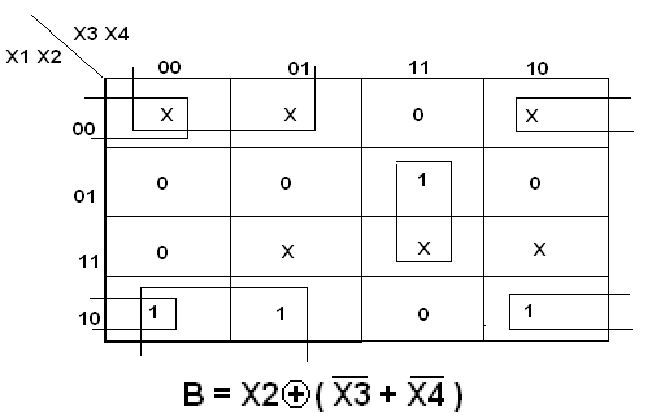
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **B3** | **B2** | **B1** | **B0** | **G3** | **G2** | **G1** | **G0** |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0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

**iv. Excess-3 to BCD code converter**

**Logic Diagram:**

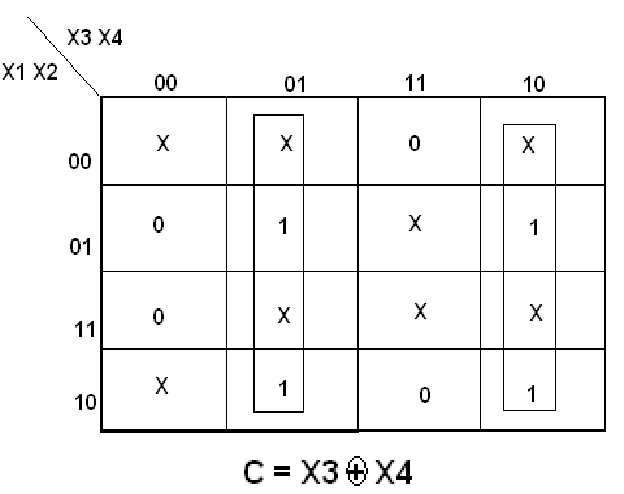


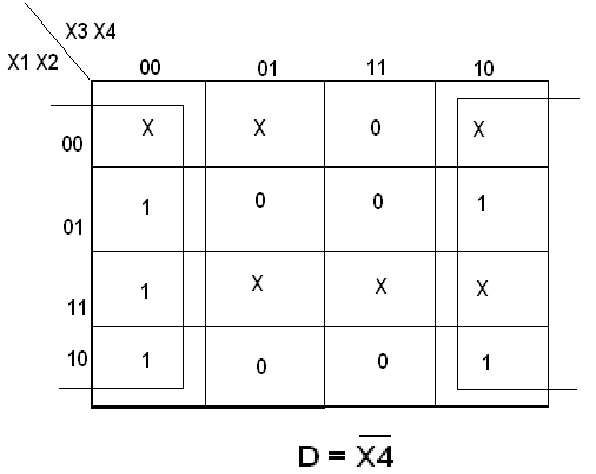
**K map for A: K map for B:**



A=X1X2+X3X4X1

**K map for C: K map for D:**





**Truth table:**

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

Procedure:

1. Connections were given as per circuit diagram.
2. Logical inputs were given as per truth table
3. Observe the logical output and verify with the truth tables.

Result:

Thus the code converters were designed and verified using the corresponding truth table.

**Expt.No.8:** **Design and Implementation of Parity Checker and Generators**

**AIM:**

To construct 16 bit odd/even parity checker/generator using IC 74180.

**APPARATUS REQUIRED:**

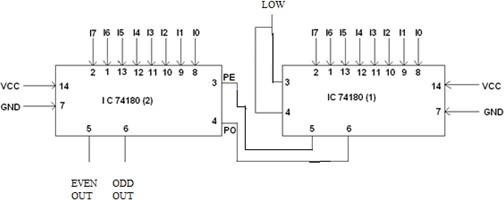
|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **NAME OF THE APPPARATUS** | **RANGE** | **QUANTITY** |
| 1. | NOT GATE | IC 7404 | 1 |
| 2. | PARITY GENERATOR/CHECKER IC | IC 74180 | 2 |
| 3. | IC TRAINER KIT | - | 1 |
| 4. | PATCH CORDS | - | AS REQUIRED |

**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.

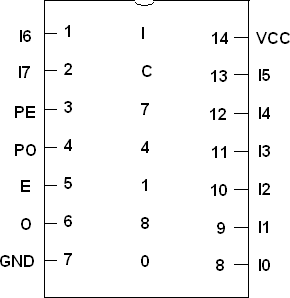
## 16 BIT ODD/EVEN PARITY CHECKER LOGIC DIAGRAM:



**TRUTH TABLE**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **INPUTS (1st IC’s)** | | | | | | | | **INPUTS (2nd IC’s)** | | | | | | | | **OUTPUT** | |
| **I0** | **I1** | **I2** | **I3** | **I4** | **I5** | **I6** | **I7** | **I0** | **I1** | **I2** | **I3** | **I4** | **I5** | **I6** | **I7** | **EVEN** | **ODD** |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |

**PINCONFIGURATION: IC74180**



**FUNCTION TABLE:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **INPUTS** |  | | **OUTPUTS** | |
| **Number of High Data**  **Inputs (I0 – I7)** | **PE** | **PO** | **∑E** | ∑**O** |
| **EVEN** | **1** | **0** | **1** | **0** |
| **ODD** | **1** | **0** | **0** | **1** |
| **EVEN** | **0** | **1** | **0** | **1** |
| **ODD** | **0** | **1** | **1** | **0** |
| **X** | **1** | **1** | **0** | **0** |
| **X** | **0** | **0** | **1** | **1** |

**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 parity generator and checker were constructed and their functions were verified.

**Expt.No.9:** **Design and Implementation of Multiplexer and De multiplexer**

Aim:

To design and verify the truth table of a 4X1 multiplexer & 1X4 demultiplexer

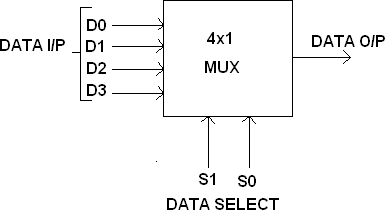
Apparatus required:

|  |  |  |  |
| --- | --- | --- | --- |
| Sl. No | Name of the Apparatus | Range | Quantity |
| 1. | Digital IC trainer kit |  | 1 |
| 2. | OR gate | IC 7432 | 1 |
| 3. | NOT gate | IC 7404 | 1 |
| 4. | AND gate ( three input ) | IC 7411 | 1 |
| 5. | Connecting wires |  | As required |

Theory:

Multiplexing 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 i formation from one of many input lines and directs it to a single output line. The selection of particular nput line is controlled by a set of selection lines. Normally, there are 2n input lines and n selection lin s whose bit combinations determines which input is selected. A multiplexer is called a data selector, since it selects one of many inputs and steers the binary information to the output line. A Strobe s also provided to allow the designer to disable all output data until a specified time. Then, by allowing the STROBE to go low, the proper lead can be selected. This feature is very useful where data might be changing the same time DATA SELECT leads change. It is a very useful Medium Scale Integration (MSI) function and has a multitude of applications. It is used for connecting two or more sources to a single destination among the computer units and it is useful for constructing a common bus system. A decoder with an enable input can function as a demultiplexer. A Demultiplexer is a circuit that receives information on a single line and transmits this information on one of 2n possible output lines. The selection of specific output line is controlled by the bit values of n selection lines. The decoder and demultiplexer operations are obtained from the same circuit; a decoder with an enable input is referred to as a decoder / de-multiplexer. The Strobe lead can be used to active or de-active the entire IC, allowing time for the address lines to change the information is fed to the output. Demultiplexers are useful anytime information from one source must be fed several places.

**4 X 1 MULTIPLEXER**

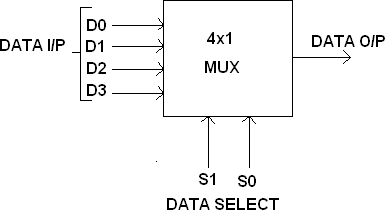


**FUNCTION TABLE:**

|  |  |  |
| --- | --- | --- |
| **S1** | **S0** | **INPUTS Y** |
| **0** | **0** | **D0 → D0 S1’ S0’** |
| **0** | **1** | **D1 → D1 S1’ S0** |
| **1** | **0** | **D2 → D2 S1 S0’** |
| **1** | **1** | **D3 → D3 S1 S0** |

**Y = D0 S1’ S0’ + D1 S1’ S0 + D2 S1 S0’ +D3 S1 S0**

**BLOCK DIAGRAM FOR 1:4 DEMULTIPLEXERS:**

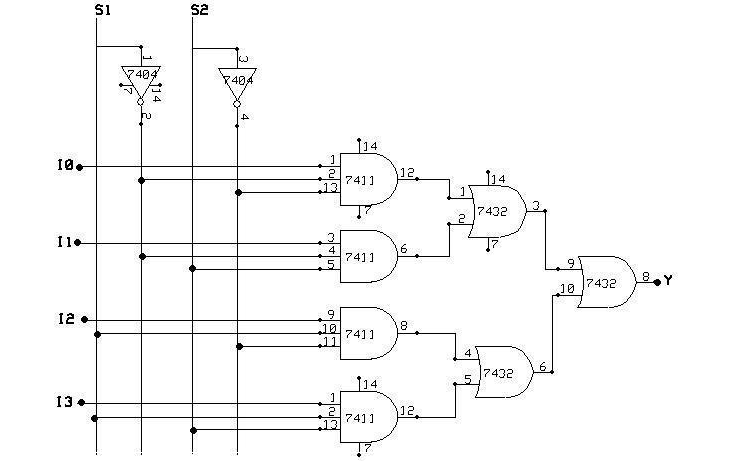


**FUNCTION TABLE:**

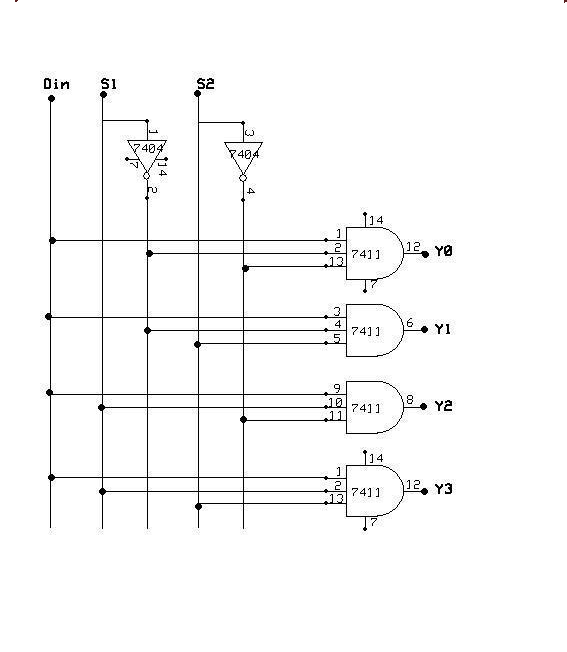
|  |  |  |
| --- | --- | --- |
| **S1** | **S0** | **INPUT** |
| **0** | **0** | **X → D0 = X S1’ S0’** |
| **0** | **1** | **X → D1 = X S1’ S0** |
| **1** | **0** | **X → D2 = X S1 S0’** |
| **1** | **1** | **X → D3 = X S1 S0** |

**Y = X S1’ S0’ + X S1’ S0 + X S1 S0’ + X S1 S0**

**CIRCUIT DIAGRAM:**



**CIRCUIT DIAGRAM:**



**Result:**

The design of the 4x1 Multiplexer and 1x4 Demultiplexer circuits was done and their truth tables were verified.

**EXP NO: 10** **Design and Implementation of Shift Registers**

**Aim:**

To design and implement the various shift register

**Apparatus required:**

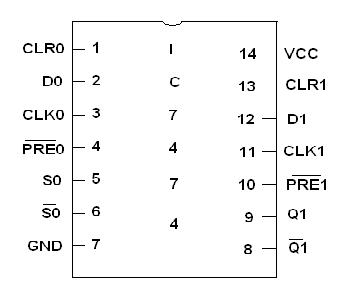


|  |  |  |  |
| --- | --- | --- | --- |
| Sl. No | Name of the Apparatus | Range | Quantity |
| 1. | D flip flop | IC 7474 | 2 |
| 2. | OR gate | IC 7432 | 1 |
| 3. | IC Trainer kit |  | 1 |
| 5. | Connecting wires |  | As required |



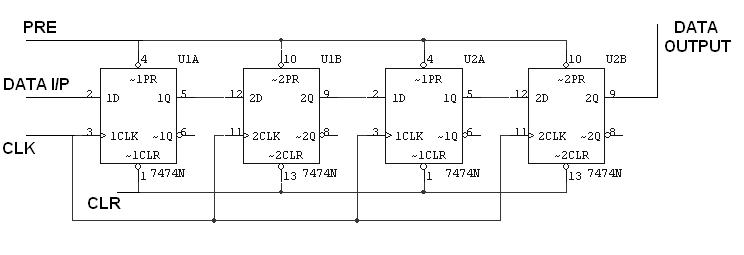
**Theory:**

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 co s st of a D-Flip flop cascaded with output of one flip flop connected to input of next flip flop. All flip flops receive common clock pulses which causes the shift in the output of the flip flop.The simplest possible shift register is one that uses only flip flop. The output of a given flip flop is connected to the input of next flip flop of the register. Each clock pulse shifts the content of register one bit position to right.

**PIN Diagram:**

**Logic Diagram:**

**SERIAL IN SERIAL OUT**



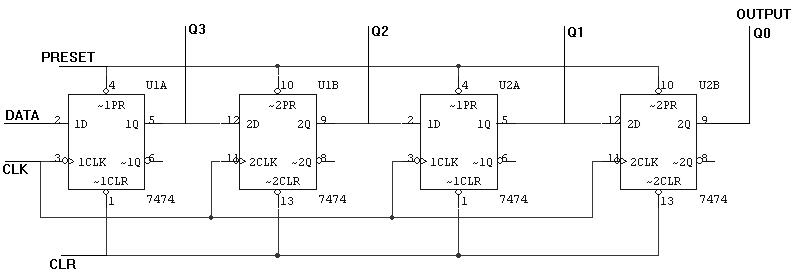


**Truth Table:**

|  |  |  |
| --- | --- | --- |
| **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:**

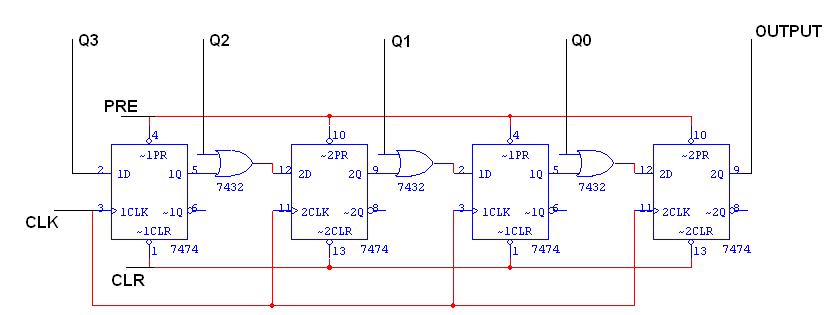
**Serial in parallel out:**



**Truth Table:**

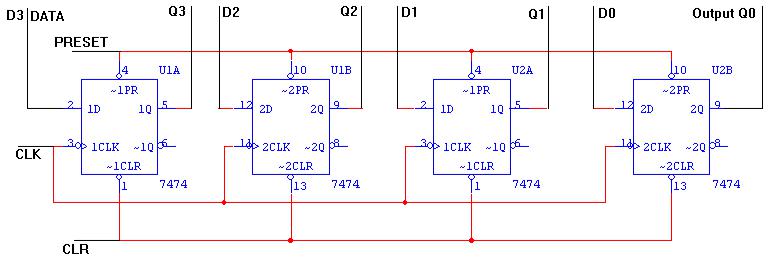
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CLK | DATA |  | OUTPUT |  |  |
| QA | QB | QC | QD |
|
| 1 | 1 | 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 0 | 1 | 1 |
| 4 | 1 | 1 | 0 | 0 | 1 |

**Parallel in Serial Out:**



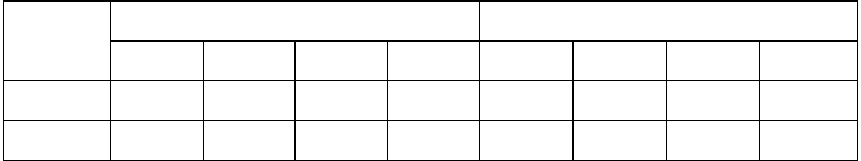
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **TRUTH TABLE FOR PISO SHIFT REGISTER:** | | | | | |  |  |
|  | **CLK** | **Q3** | **Q2** | **Q1** |  | **Q0** | **O/P** |
|  |  |  |  |  |  |  |  |
|  | **0** | **1** | **0** | **0** |  | **1** | **1** |
|  |  |  |  |  |  |  |  |
|  | **1** | **0** | **0** | **0** |  | **0** | **0** |
|  |  |  |  |  |  |  |  |
|  | **2** | **0** | **0** | **0** |  | **0** | **0** |
|  |  |  |  |  |  |  |  |
|  | **3** | **0** | **0** | **0** |  | **0** | **1** |
|  |  |  |  |  |  |  |  |

**Parallel in Parallel Out:**



**PARALLEL IN PARALLEL OUT**

**Truth Table:**



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **DATA INPUT** | |  |  | **OUTPUT** | |  |
| **CLK** | **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** |

**Procedure:**

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

**Result:**

Thus the implementation of shift registers using flip flops was completed successfully.

**EXP NO: 11 Design And Implementation Of 3 Bit Synchronous Up / Down Counter**

**AIM:**

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

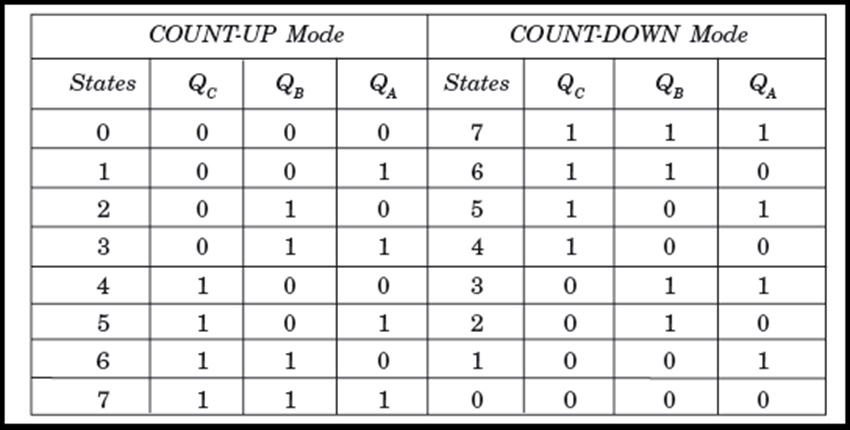
**APPARATUS REQUIRED:**

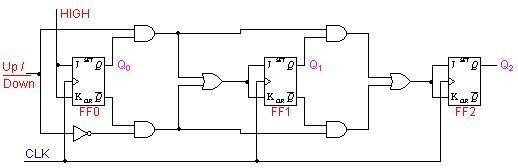
|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **NAME OF THE APPARATUS** | **RANGE** | **QUANTITY** |
| 1. | JK FLIP FLOP | IC 7476 | 2 |
| 2. | 3 I/P AND GATE | IC 7411 | 1 |
| 3. | OR GATE | IC 7432 | 1 |
| 4. | XOR GATE | IC 7486 | 1 |
| 5. | NOT GATE | IC 7404 | 1 |
| 6. | IC TRAINER KIT | - | 1 |
| 7. | PATCH CORDS | - | AS REQUIRED |

**THEORY:**

A counter is a register capable of counting number of clock pulse arriving at its clock input. Counter represents the number of clock pulses arrived. An up/down counter is one that is capable of progressing in increasing order or decreasing order through a certain sequence. An up/down counter is also called bidirectional counter. Usually up/down operation of the counter is controlled by up/down signal. When this signal is high counter goes through up sequence and when up/down signal is low counter follows reverse sequence.

**TRUTH TABLE:**





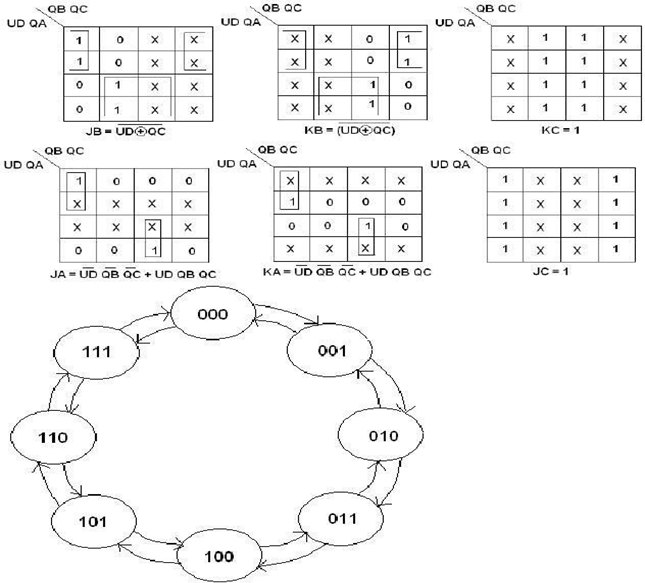
:

**Circuit Diagram**

**EXCITATION TABLE:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input**  **Up/Down** | **Present State**  **QA QBQC** | **Next State**  **QA+1 Q B+1 QC+1** | **A**  **JA KA** | **B**  **JB KB** | **C**  **JC KC** |
| 0 | 0 0 0 | 1 1 1 | 1 X | 1 X | 1 X |
| 0 | 1 1 1 | 1 1 0 | X 0 | X 0 | X 1 |
| 0 | 1 1 0 | 1 0 1 | X 0 | X 1 | 1 X |
| 0 | 1 0 1 | 1 0 0 | X 0 | 0 X | X 1 |
| 0 | 1 0 0 | 0 1 1 | X 1 | 1 X | 1 X |
| 0 | 0 1 1 | 0 1 0 | 0 X | X 0 | X 1 |
| 0 | 0 1 0 | 0 0 1 | 0 X | X 1 | 1 X |
| 0 | 0 0 1 | 0 0 0 | 0 X | 0 X | X 1 |
| 1 | 0 0 0 | 0 0 1 | 0 X | 0 X | 1 X |
| 1 | 0 0 1 | 0 1 0 | 0 X | 1 X | X 1 |
| 1 | 0 1 0 | 0 1 1 | 0 X | X 0 | 1 X |
| 1 | 0 1 1 | 1 0 0 | 1 X | X 1 | X 1 |
| 1 | 1 0 0 | 1 0 1 | X 0 | 0 X | 1 X |
| 1 | 1 0 1 | 1 1 0 | X 0 | 1 X | X 1 |
| 1 | 1 1 0 | 1 1 1 | X 0 | X 0 | 1 X |
| 1 | 1 1 1 | 0 0 0 | X 1 | X 1 | X 1 |

**K MAP AND STATE DIAGRAM:**



**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 3 bit synchronous up/ down counter is designed and its truth table is verified.

**EXP NO: 12 VHDL Program For Combinational Circuits**

**AIM**

To design combinational circuit using VHDL & simulate using modelsim.

**TOOLS REQUIRED**

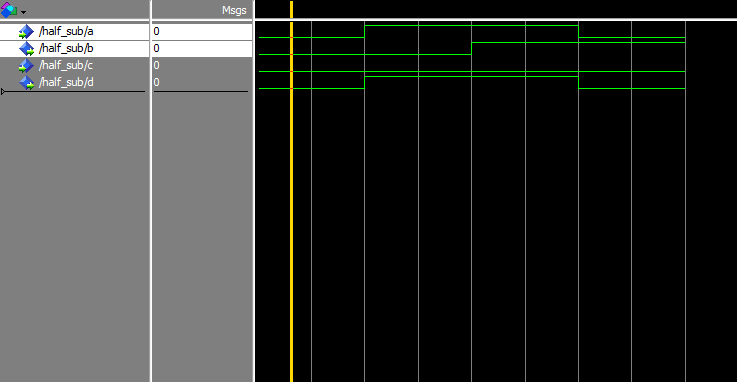
Simulation : modelsim

**PROCEDURE FOR SIMULATION**

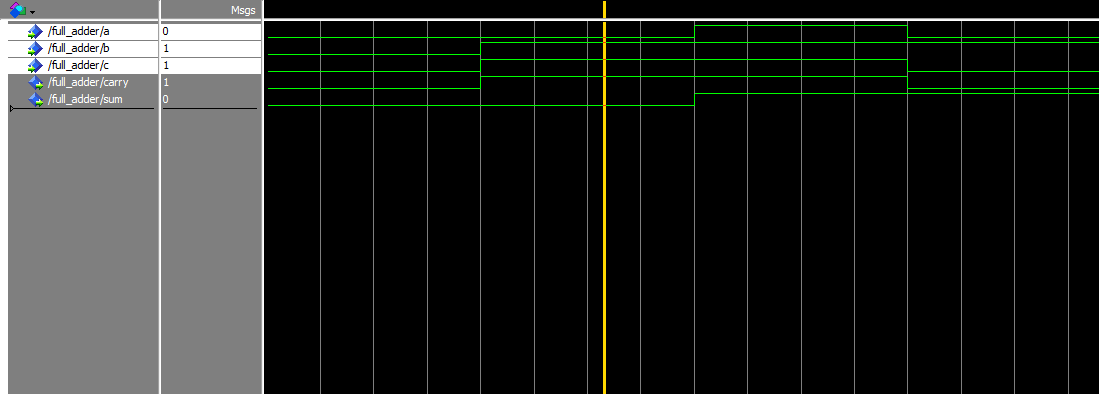
1. To start the program clicks the modelsim software.
2. The main page is opened; click the file option to create a new source in VHDL.
3. After the program is typed, it is saved in a name with exrension„.vhd‟.
4. Then the program is complied and errors are checked.
5. After that it is simulated.
6. Then the program is viewed and the signal option is clicked from the view menu and input signals are given.
7. Then in the edit option, force is selected and the values are given.
8. Finally Add-wave is clicked view the result waveform.

|  |  |
| --- | --- |
| **VHDL Code for a Half-Adder**  Library ieee;  use ieee.std\_logic\_1164.all;    entity half\_adder is  port(a,b:in bit; sum,carry:out bit);  end half\_adder;    architecture data of half\_adder is  begin  sum<= a xor b;  carry <= a and b;  end data; | https://sub.allaboutcircuits.com/images/04454.png |
| **VHDL Code for a Full Adder**  Library ieee;  use ieee.std\_logic\_1164.all;    entity full\_adder is port(a,b,c:in bit; sum,carry:out bit);  end full\_adder;    architecture data of full\_adder is  begin  sum<= a xor b xor c;  carry <= ((a and b) or (b and c) or (a and c));  end data; | Image result for full adder circuit |
| **VHDL Code for a Half-Subtractor**  Library ieee;  use ieee.std\_logic\_1164.all;    entity half\_sub is  port(a,c:in bit; d,b:out bit);  end half\_sub;  architecture data of half\_sub is  begin  d<= a xor c;  b<= (a and (not c));  end data; | Image result for half subtractor |
| **VHDL Code for a Full Subtractor**  Library ieee;  use ieee.std\_logic\_1164.all;    entity full\_sub is  port(a,b,c:in bit; sub,borrow:out bit);  end full\_sub;    architecture data of full\_sub is  begin  sub<= a xor b xor c;  borrow <= ((b xor c) and (not a)) or (b and c);  end data; | https://www.csetutor.com/wp-content/uploads/2018/02/full-subtractor-circuit-diagram.png |
| **VHDL Code for a Multiplexer**  Library ieee;  use ieee.std\_logic\_1164.all;    entity mux is  port(S1,S0,D0,D1,D2,D3:in bit; Y:out bit);  end mux;    architecture data of mux is  begin  Y<= (not S0 and not S1 and D0) or  (S0 and not S1 and D1) or  (not S0 and S1 and D2) or  (S0 and S1 and D3);  end data; | Image result for 4 to 1multiplexer |
| **VHDL Code for a Demultiplexer**  Library ieee;  use ieee.std\_logic\_1164.all;    entity demux is  port(S1,S0,D:in bit; Y0,Y1,Y2,Y3:out bit);  end demux;    architecture data of demux is  begin  Y0<= ((Not S0) and (Not S1) and D);  Y1<= ((Not S0) and S1 and D);  Y2<= (S0 and (Not S1) and D);  Y3<= (S0 and S1 and D);  end data; | Image result for demultiplexer |
| **VHDL Code for a 8 x 3 Encoder**  library ieee;  use ieee.std\_logic\_1164.all;    entity enc is  port(i1,i2,i3,i4,i5,i6,i7:in bit; o0,o1,o2: out bit);  end enc;    architecture vcgandhi of enc is  begin  o0<=i4 or i5 or i6 or i7;  o1<=i2 or i3 or i6 or i7;  o2<=i1 or i3 or i5 or i7;  end vcgandhi; | Image result for 8 x 3 Encoder |
| **VHDL Code for a 3 x 8 Decoder**  library ieee;  use ieee.std\_logic\_1164.all;  entity dec is  port(i0,i1,i2:in bit; o0,o1,o2,o3,o4,o5,o6,o7: out bit);  end dec;    architecture vcgandhi of dec is  begin  o0<=(not i0) and (not i1) and (not i2);  o1<=(not i0) and (not i1) and i2;  o2<=(not i0) and i1 and (not i2);  o3<=(not i0) and i1 and i2;  o4<=i0 and (not i1) and (not i2);  o5<=i0 and (not i1) and i2;  o6<=i0 and i1 and (not i2);  o7<=i0 and i1 and i2;  end vcgandhi; | Image result for 3 x 8 Decoder |

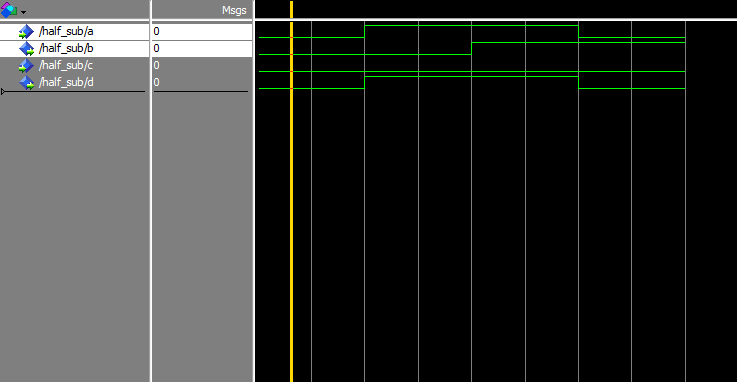
**Output:**



**HALF ADDER**



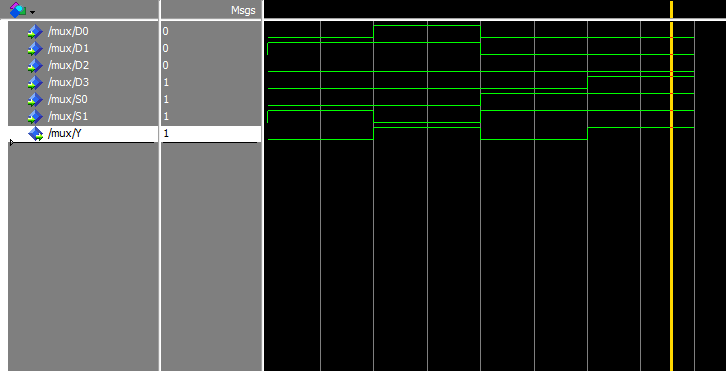
**FULL ADDER**

****

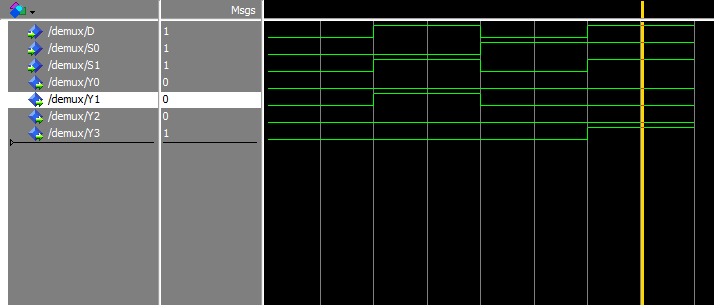
**HALF SUBTRACTOR**

****

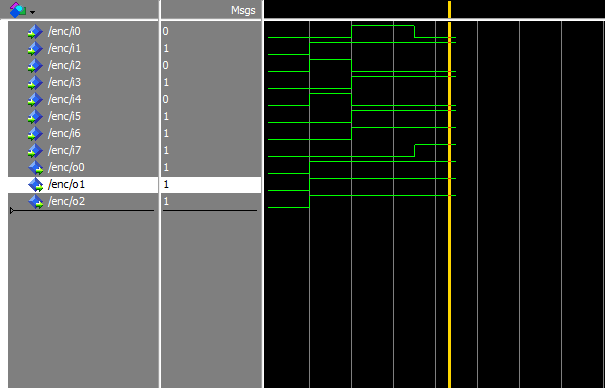
**FULL SUBTRACTOR**

****

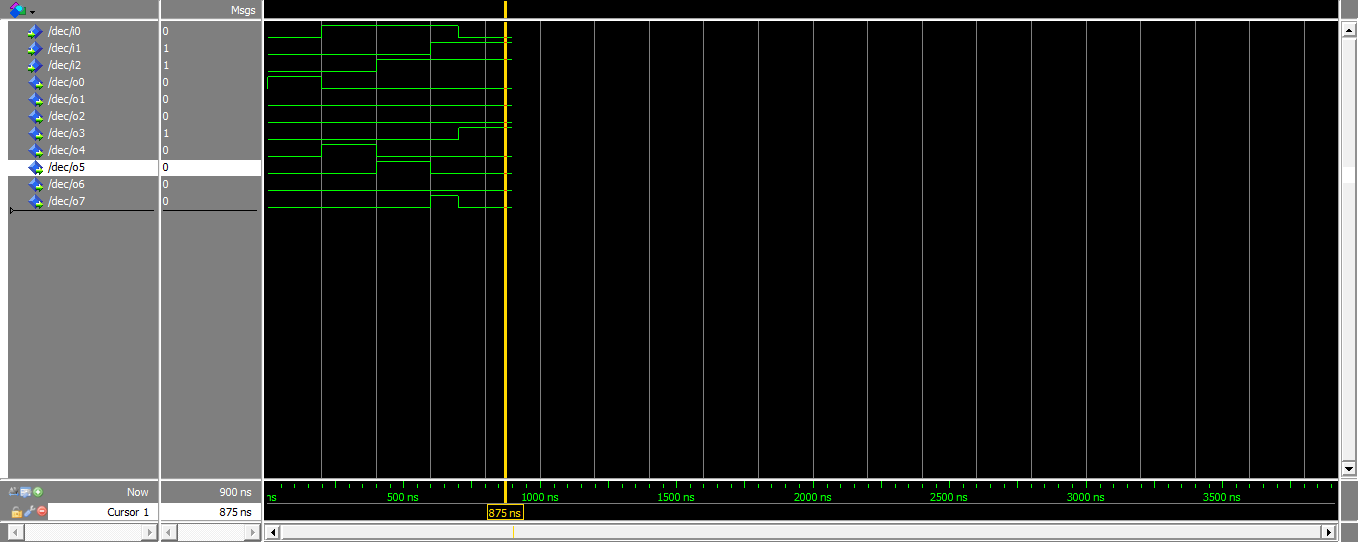
**4:1 MULTIPLEXER**

****

**1:4 DEMULTIPLEXER**

****

**ENCODER**

****

**DECODER**

**RESULT:**

Thus the VHDL program for Combinational circuits were verified.

**EXP NO: 13 VHDL Program for Sequential Circuits**

## AIM:

To write a HDL program for sequential circuits and verify their simulation result.

## SOFTWARE REQUIRED :

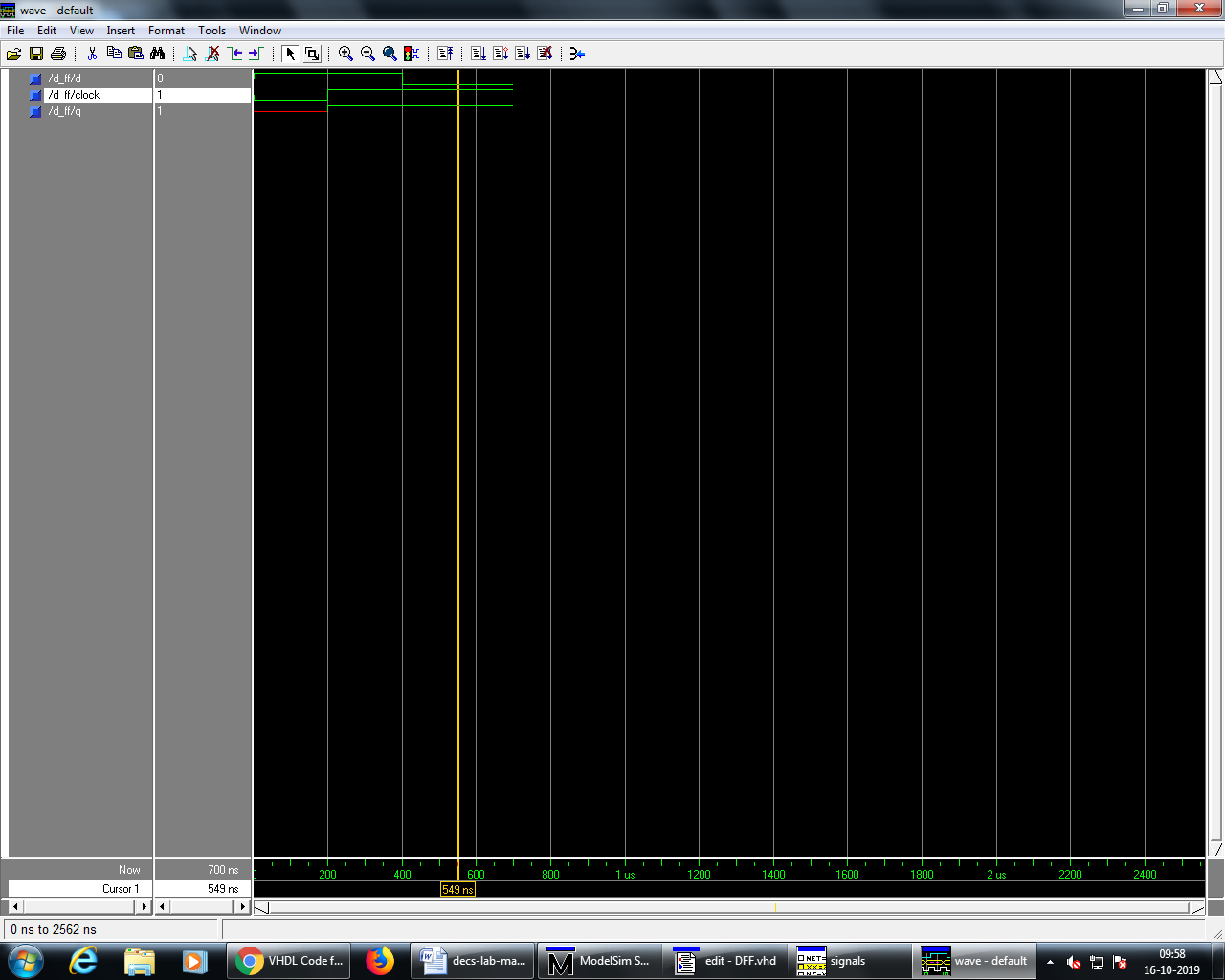
MODEL SIM software.

## PROCEDURE:

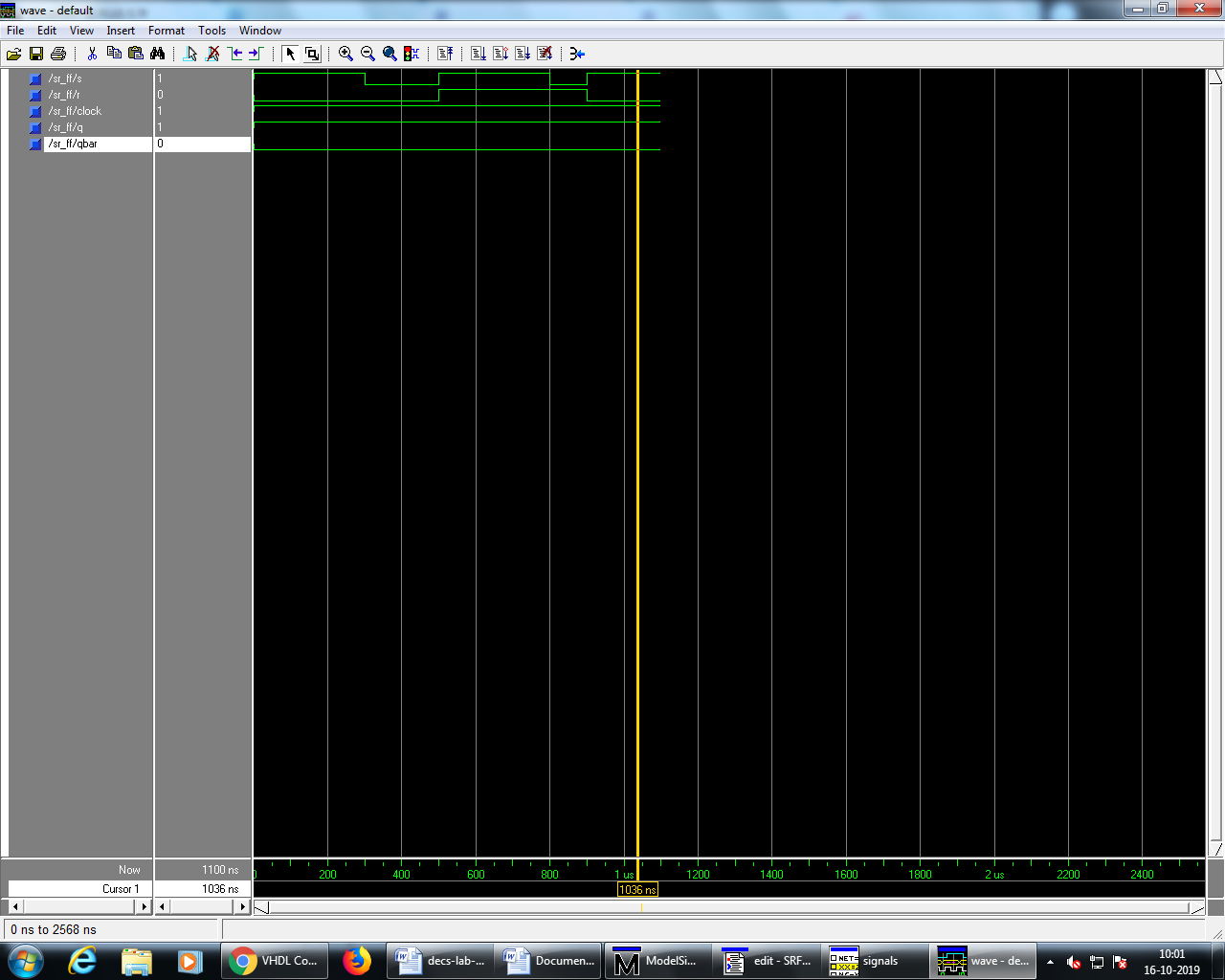
* + Click – start – MODEL SIM SE5.79
  + Go to file menu and click new source –HDL
  + Type the program by using file menu HDL (E: or F:)
  + Compile the program
    - Tools – compile (Alt +Tab)
  + For simulation
    - Click – work –ok
    - Click – view –signals
    - Click – Edit – force – give the value
    - Click – Add –wave
    - Click – signals in region
    - Click – run

|  |  |
| --- | --- |
| **VHDL Code for D FlipFlop**  library ieee;  use ieee. std\_logic\_1164.all;  use ieee. std\_logic\_arith.all;  use ieee. std\_logic\_unsigned.all;    entity D\_FF is  PORT( D,CLOCK: in std\_logic;  Q: out std\_logic);  end D\_FF;    architecture behavioral of D\_FF is  begin  process(CLOCK)  begin  if(CLOCK='1' and CLOCK'EVENT) then  Q <= D;  end if;  end process;  end behavioral; | http://allaboutfpga.com/wp-content/uploads/2014/07/D-FLIPFLOP.png |
| **VHDL Code for SR FlipFlop**  library ieee;  use ieee. std\_logic\_1164.all;  use ieee. std\_logic\_arith.all;  use ieee. std\_logic\_unsigned.all;    entity SR\_FF is  PORT( S,R,CLOCK: in std\_logic;  Q, QBAR: out std\_logic);  end SR\_FF;    Architecture behavioral of SR\_FF is  begin  PROCESS(CLOCK)  variable tmp: std\_logic;  begin  if(CLOCK='1' and CLOCK'EVENT) then  if(S='0' and R='0')then  tmp:=tmp;  elsif(S='1' and R='1')then  tmp:='Z';  elsif(S='0' and R='1')then  tmp:='0';  else  tmp:='1';  end if;  end if;  Q <= tmp;  QBAR <= not tmp;  end PROCESS;  end behavioral; | http://allaboutfpga.com/wp-content/uploads/2014/07/SR-FLIPFLOP.png |
| **VHDL code for Parallel In Parallel Out Shift Register**  library ieee;  use ieee.std\_logic\_1164.all;    entity pipo is  port(  clk : in std\_logic;  D: in std\_logic\_vector(3 downto 0);  Q: out std\_logic\_vector(3 downto 0) );  end pipo;    architecture arch of pipo is  begin  process (clk)  begin  if (CLK'event and CLK='1') then  Q <= D;  end if;  end process;  end arch; | Image result for Parallel In Parallel Out Shift Register |
| **VHDL For Serial in Serial out Shift Register**  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.STD\_LOGIC\_ARITH.ALL;  use IEEE.STD\_LOGIC\_UNSIGNED.ALL;  entity shiftregister is  Port ( si : in STD\_LOGIC;  clk : in STD\_LOGIC;  so : out STD\_LOGIC);  end shiftregister;  architecture Behavioral of shiftregister is  signal temp : std\_logic\_vector(3 downto 0);  begin  process(clk)  begin  if(clk'event and clk ='1')then  temp <= si & temp(3 downto 1);  end if;  end process;  so <= temp(0);  end Behavioral; | Image result for SISO Shift Register |

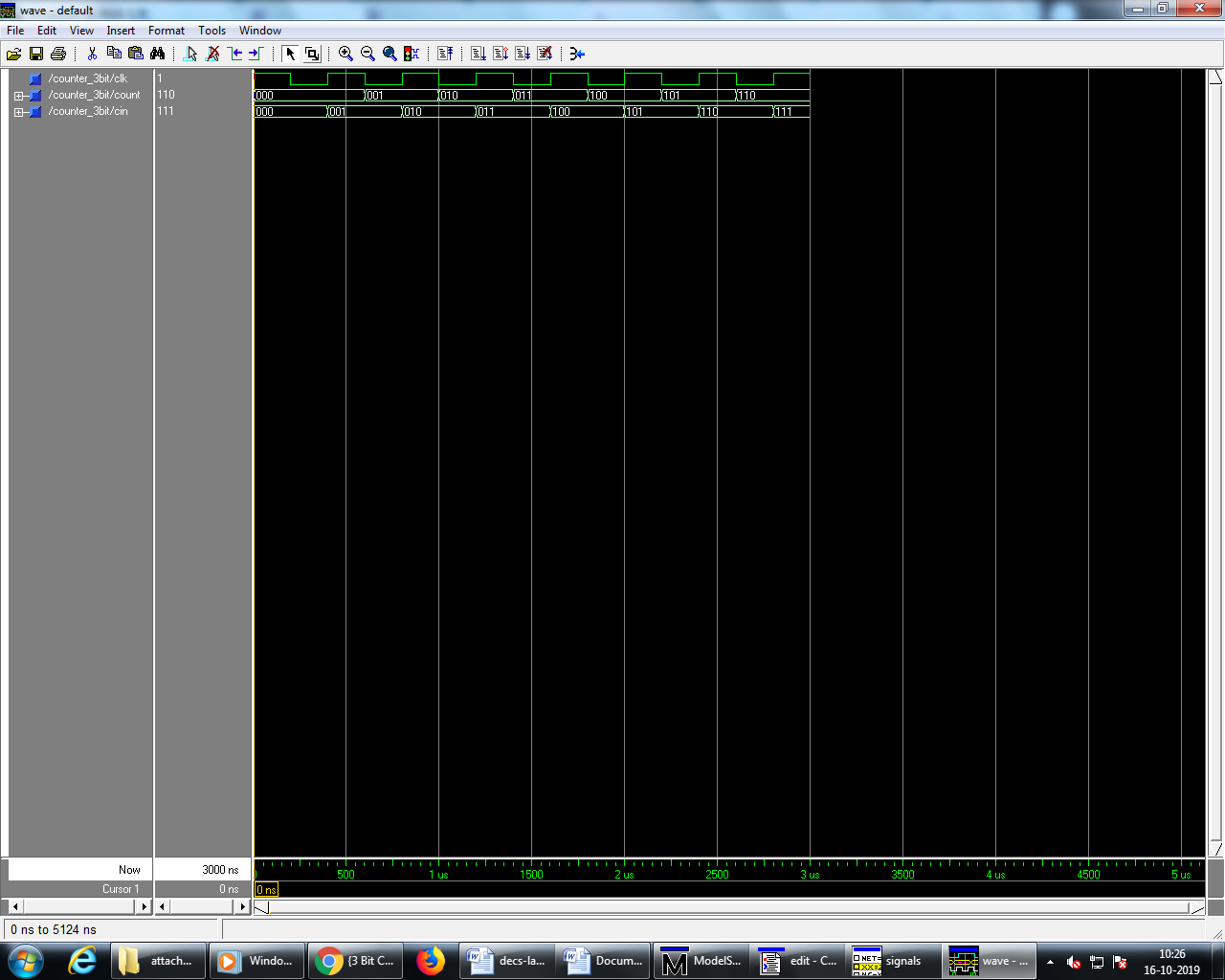
**Output:**



**D FLIP FLOP**



**SR FLIP FLOP**



3 Bit up Counter



Parallel in Parallel out Shift Register



Serial in Serial out Shift Register

## RESULT:

Thus the VHDL programs for Sequential circuits were verified.