Chettinad
College of Engineering & Technology
Approved by AICTE-New Delhi and Affiliated to Anna University-Chennai

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CS3351 DIGITAL PRINCIPLE AND COMPUTER ORGANIZATION

ACADEMIC YEAR: 2024-2025

II YEAR/III TH Semester Lab Manual

Expt. No:	Verification Of Boolean Theorems Using Logic Gates	Date:

AIM:

To study and verify the truth table of basic logic gates NOT, AND, OR, NAND, NOR and EX-OR and to verify the Boolean theorems.

COMPONENTS / EQUIPMENTS REQUIRED:

S.No.	Name of the apparatus	Range	Quantity
1	Digital Trainer kit		
2	NOT gate	IC7404	1
3	AND gate	IC7408	1
4	OR gate	IC7432	1
5	EX-OR gate	IC7486	1
6	NAND gate	IC7400	1
7	NOR gate	IC7402	1
8	Connecting wires		

THEORY:

In Boolean algebra three basic logic operations are available. They are OR, AND, NOT. These logic gates are digital circuits constructed from diodes, transistors, and resistors connected in such a way that the circuit output is the result of a basic logic operation (OR, AND, NOT) performed on the inputs.

Truth table: A truth table is a means for describing how a logic circuit's output depends on the logic levels present at the circuit's inputs

DeMorgan's Theorem:

DeMorgan's theorems are extremely useful in simplifying expressions in which a product or sum of variables is inverted. The two theorems are:

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

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

Commutative Law:

$A.B = B.A$

$A+B = B+A$

Associative Law:
A. (B.C) = (A.B). C

Commutative Law: A.B = B.A

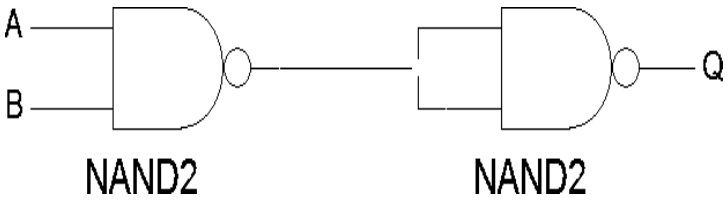


Fig: 1.1

Associative Law: A. (B.C) = (A.B). C

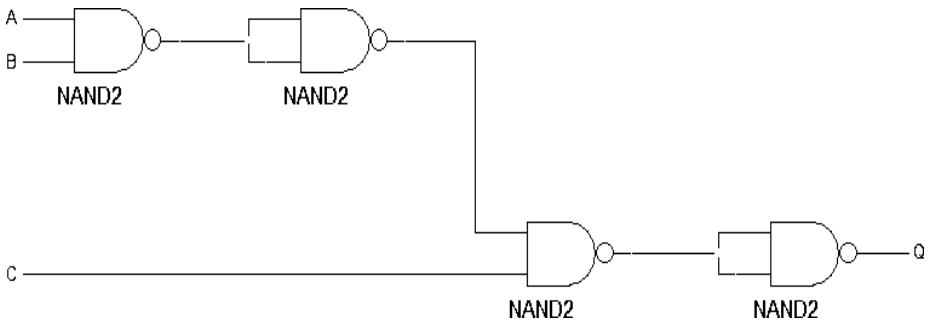


Fig: 1.2

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

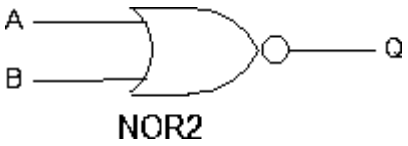


Fig: 1.3

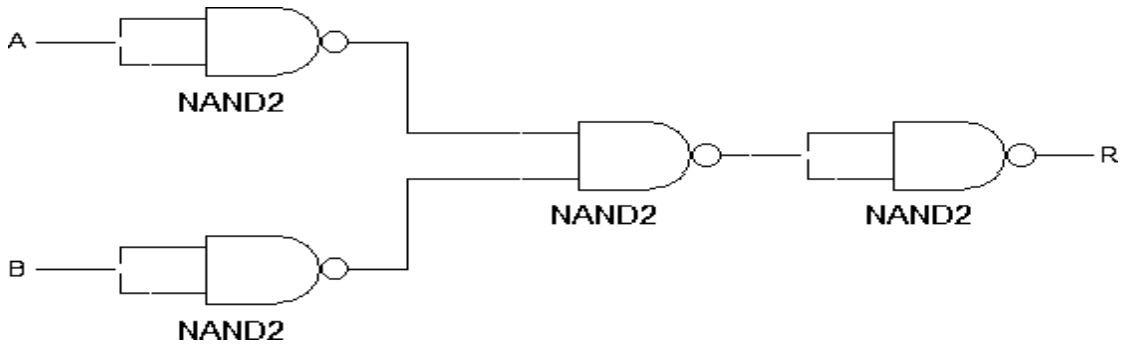


Fig: 1.4

Observations:

Commutative Law:

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

Tab: 1.1

Associative Law:

A	B	C	A(BC)	(AB)C
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Tab: 1.2

DeMorgans Law:

A	B	(A+B)'	A'.B'
0	0		
0	1		
1	0		
1	1		

Tab: 1.3

INFERENCE AND CONCLUSION:
Thus the digital logic gates were studied and the Boolean theorems were verified using the truth table.

VIVA QUESTIONS:

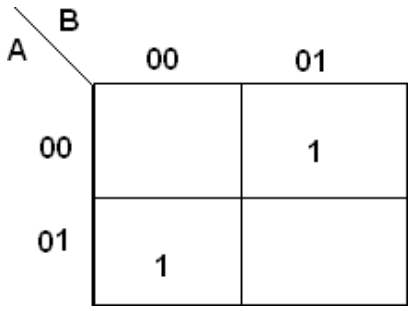
1. Obtain AND gate using only NAND gates.
2. What are universal gates?
3. State Demorgans theorem:
4. Implement OR gate using only NAND gate:
5. Write the truth table for EX-OR gate:
6. What is a logic gate?
7. State the consensus theorem in Boolean algebra:
8. What are don't care conditions?
9. What is the the need for Quine Mccluskey method ?
10. What are minterm and maxterm?

Truth Table for Half Adder

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

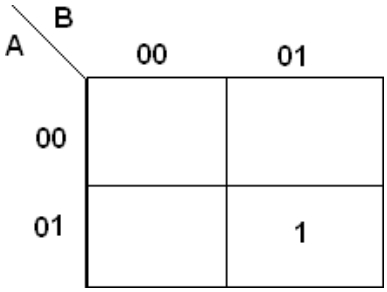
K-map simplification for Half Adder

K-Map for SUM:



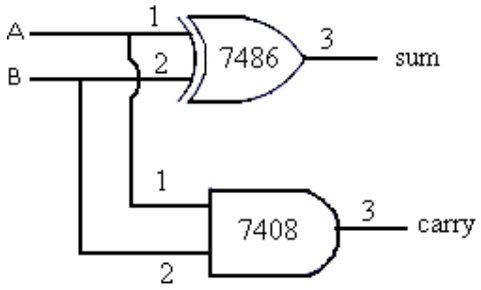
SUM = A'B + AB'

K-Map for CARRY:

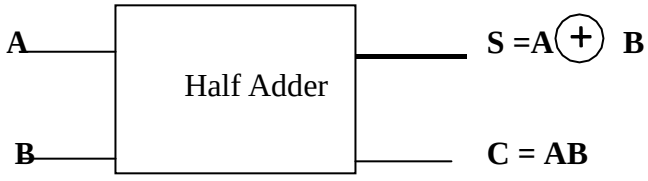


CARRY = AB

LOGIC DIAGRAM OF HALF ADDER:



BLOCK SCHEMATIC OF HALF ADDER:



Expt. No:	Implementation of Boolean Functions - Adders And Subtractors using Logic Gates	Date:

AIM:

To design and verify the adders and subtractors using logic gates.

COMPONENTS / EQUIPMENTS REQUIRED:

S. No	Components / Equipments	Specifications	Quantity
1.	Digital IC trainer	---	1
2.	NOT, AND, OR, Ex-OR Gate	IC 7404,7408,7432,7486	Each one
3.	Connecting wires	---	Sufficient numbers

THEORY:

Half Adder:

The half adder is an example of a simple, functional digital circuit built from two logic gates. The half adder adds to one-bit binary numbers (AB). The output is the sum of the two bits (S) and the carry (C). Note how the same two inputs are directed to two different gates. The inputs to the XOR gate are also the inputs to the AND gate. The input "wires" to the XOR gate are tied to the input wires of the AND gate; thus, when voltage is applied to the A input of the XOR gate, the A input to the AND gate receives the same voltage. Select an input combination from the pull-down selector and view the resulting output.

The logic circuit for the addition of two one bit numbers is referred to as a half adder. Here A and B are the two inputs and S (sum) and C (carry) are two outputs. The two outputs, the sum and carry equations are carried from the truth table of addition of two numbers.

They are $\mathbf{S = \bar{A}B + AB,}$

$\mathbf{C = AB.}$

Full Adder:

A half adder has only two inputs and there s no provision to add a carry coming from the lower order bits when multi bit additions performed. For this purpose, a third input terminal is added and this circuit is used to add A_n, B_n and C_{n-1} where A_n and B_n are the nth order bits of the numbers A and B respectively and C_{n-1} is the carry generated from the addition of (n-1) th order bits. This circuit is referred to as full adder. The sum and carry equations are carried from the truth table of addition of three numbers.

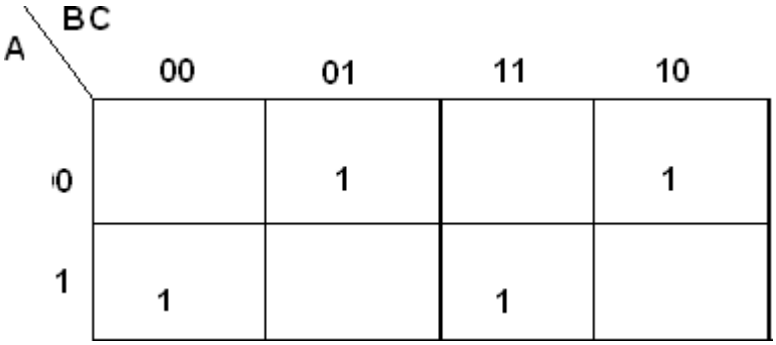
They are $\mathbf{S = A \oplus B \oplus C,}$

$\mathbf{C = AB + BC + CA.}$

Truth Table for Full Adder

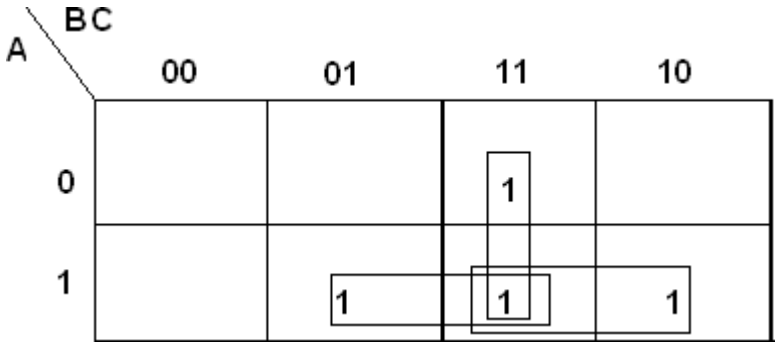
A	B	C	SUM	CARRY
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

K-Map for SUM:



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

K-Map for CARRY:



CARRY = AB + BC + AC

Half Subtractor:

A logic circuit for the subtractor of B(subtrahend) from A (minuend) where A and B are 1bit numbers is referred to as a half subtractor. The two outputs, the difference (D) and borrow (B) equations are carried from the truth table of subtraction of two numbers.

They are

Difference = $A\bar{B} + \bar{A}B$,

Borrow = $\bar{A}B$.

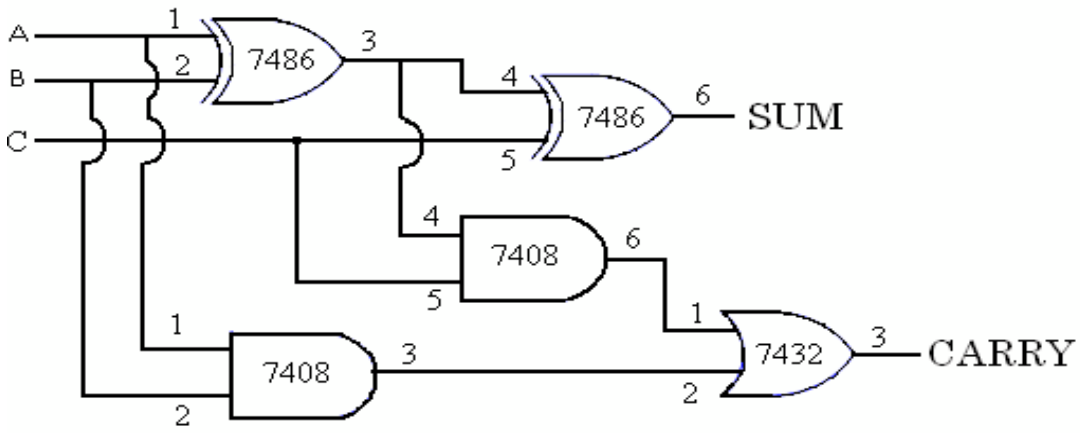
Full subtractor:

Just like a full adder circuit, full subtractor circuit performs multiple subtraction wherein borrow from the previous bit position may also be there. A full subtractor will have three inputs, A_n (minuend), B_n (subtrahend) and C_{n-1} (borrow from previous stage) and two outputs D_n (difference) and C_n (borrow). The difference and carry equations carried out from the truth table of subtraction.

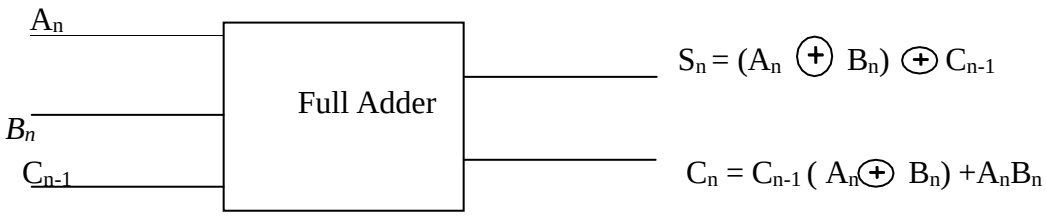
Diff.(D) = $A \oplus B \oplus C$,

Borrow = AB+AC+BC

LOGIC DIAGRAM OF FULL ADDER



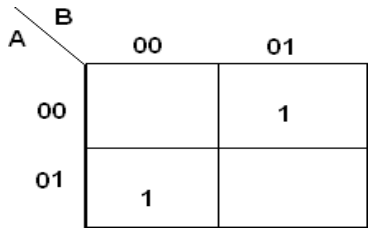
BLOCK DIAGRAM OF FULL ADDER



Truth Table for Half Subtractor

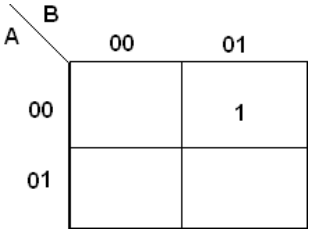
A	B	DIFFERENCE	BORROW
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

K-Map for DIFFERENCE:



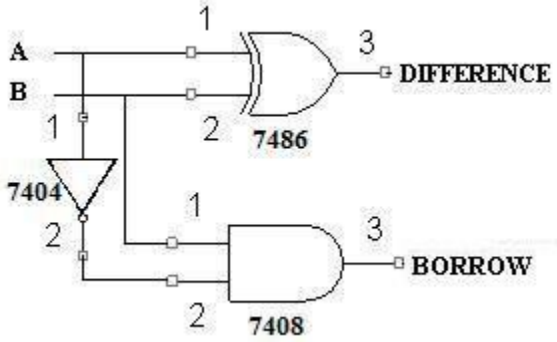
DIFFERENCE = A'B + AB'

K-Map for BORROW:

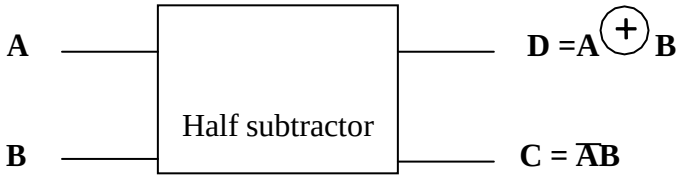


BORROW = A'B

LOGIC DIAGRAM FOR HALF SUBTRACTOR



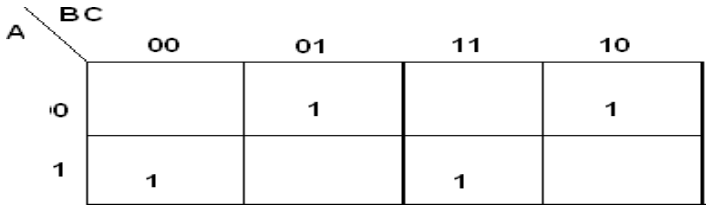
BLOCK SCHEMATIC OF HALF SUBTRACTOR:



Truth Table for Full Subtractor

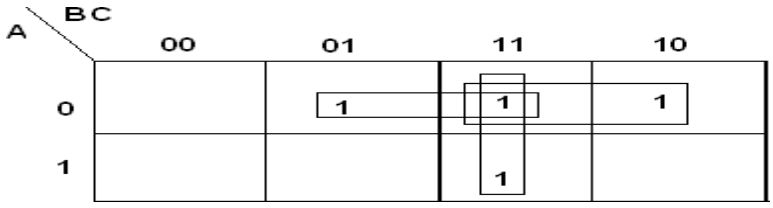
A	B	C	BORROW	DIFFERENCE
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-Map for Difference:



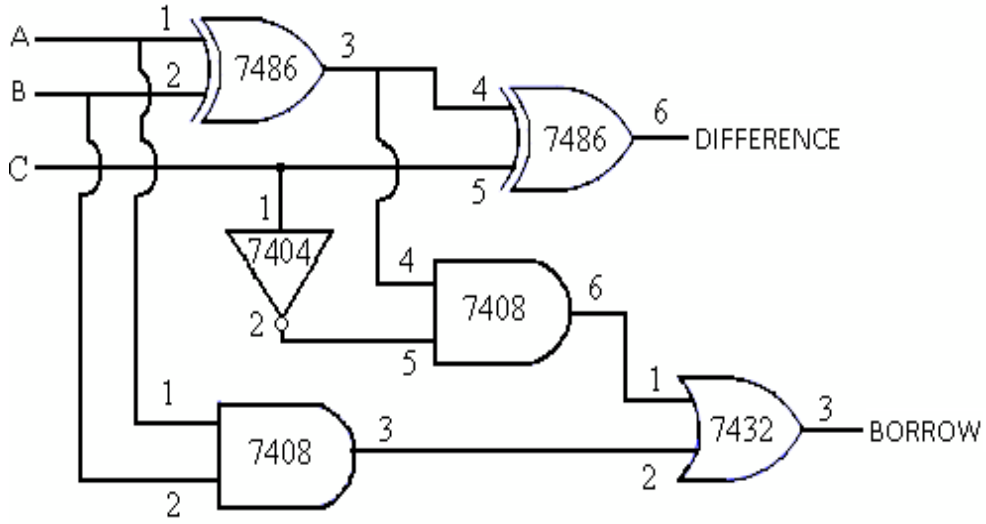
Difference = A'B'C + A'BC' + AB'C' + ABC

K-Map for Borrow:

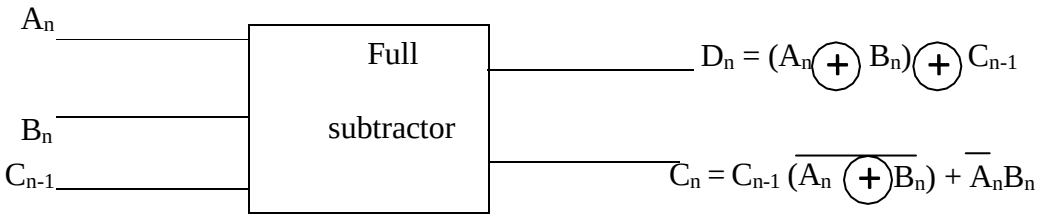


Borrow = A'B + BC + A'C

LOGIC DIAGRAM FOR FULL SUBTRACTOR



BLOCK SCHEMATIC OF FULL SUBTRACTOR



PROCEDURE:

1. Give the connections as per the Logic diagram.
2. Switch on the trainer kit.
3. Apply the binary inputs at the appropriate terminal and observe the corresponding output.
4. Verify the truth table and repeat the above procedure for other circuits.

INFERENCE AND CONCLUSION:

Thus the adders and sub tractors were constructed using the logic gates and the truth tables were verified.

REVIEW QUESTIONS:

1. What is the sum of binary number 1010+0100?
2. Parallel adders are____logic circuits.
3. What is the IC number of 4 bit binary adder?
4. Find the 1's complement of 10101010.
5. Subtract using 2's complement 110102 – 011002.
6. Why NAND & NOR gates are called Universal gates?
7. Realize the EX-OR gates using minimum number of NAND gates?
8. Realize the AND gate using NOR gate.

Functional symbol for IC 7483:

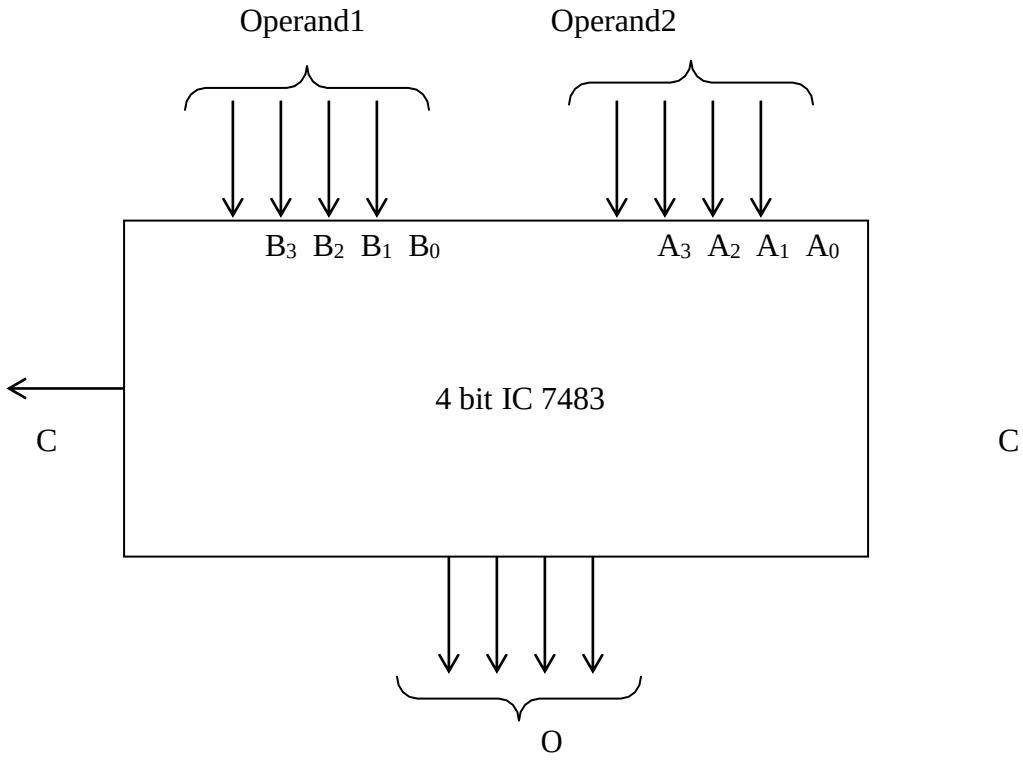


Fig: 4.1

Pin Diagram of IC7483:

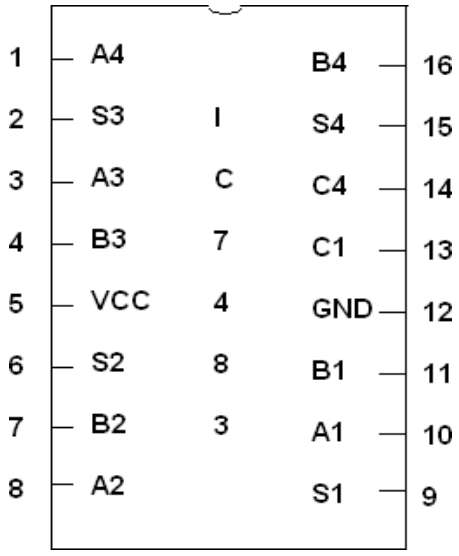


Fig: 4.2

Expt. No:	DESIGN AND IMPLEMENTATION OF 4 BIT BINARY ADDER / SUBTRACTOR USING IC 7483	Date:

AIM:

To study the 4 bit binary Adder/Subtractor using IC7483.

COMPONENTS / EQUIPMENTS REQUIRED:

S.No.	Name of the apparatus	Specifications	Quantity
1	Digital Trainer kit	-	1
2	OR gate	IC 7432	1
3	AND gate	IC 7408	1
4	Binary Adder / Subtractor	IC 7483	2
5	Connecting wires	-	some

THEORY:

The full adder/sub tractors are capable of adding/subtracting only two single digit binary numbers along with a carry input. But in practice we need to add/subtract binary numbers, which are much longer than just one bit. To add/subtract two n-bit binary numbers we need to use the n-bit parallel subtractor/adder.

Binary adder:

IC type 7483 is a 4-bit binary parallel adder/subtractor .The two 4-bit input binary numbers are A1 through A4 and B1 through B4. The sum is obtained from S1 through S4. C0 is the input carry and C4 the output carry. Test the 4-bit binary adder 7483 by connecting the power supply and ground terminals. Then connect the four A inputs to a fixed binary numbers such as 1001 and the B inputs and the input carry to five toggle switches. The five outputs are applied to indicator lamps. Perform the addition of a few binary numbers and check that the output sum and output carry give the proper values. Show that when the input carry is equal to 1, it adds 1 to the output sum.

Binary subtractor:

The subtraction of two binary numbers can be done by taking the 2's complement of the subtrahend and adding it to the minuend. The 2's complement can be obtained by taking the 1's complement and adding. To perform A-B, we complement the four bits of B, add them to the four bits of A, and add 1 through the input carry. The four XOR gates complement the bits of B when the mode select M=1(because $x \oplus 0 = \bar{x}$) and leave the bits of B unchanged when M=0(because $x \oplus 0 = x$) .Thus , when the mode select M is

Circuit Diagram for 4-bit Binary adder/subtractor:

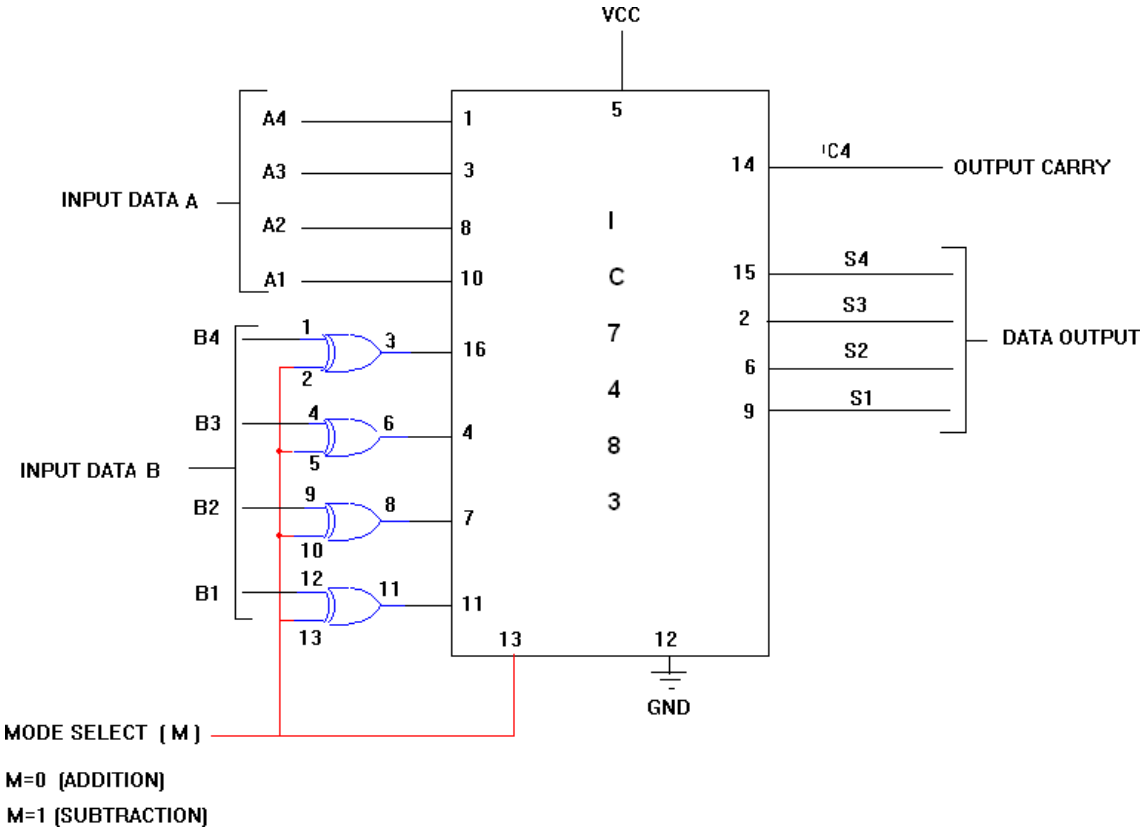
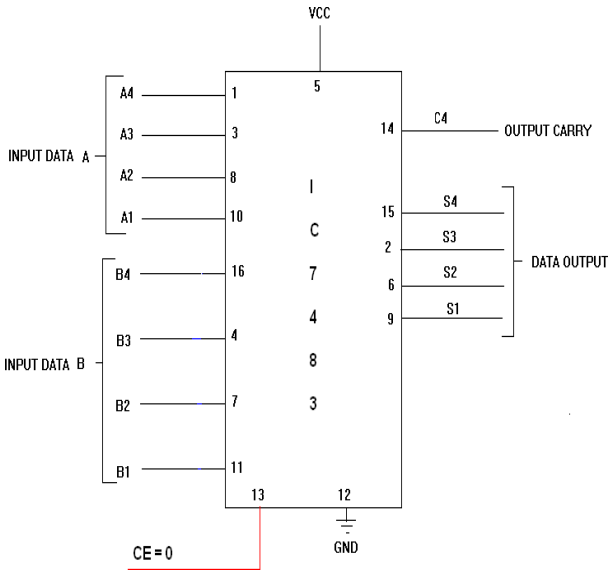
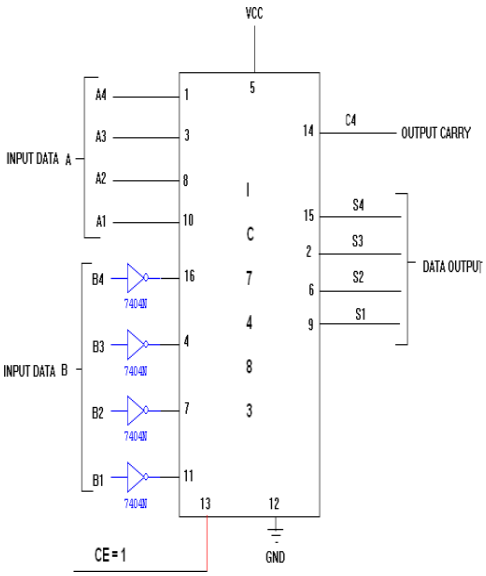


Fig: 4.3

4-BIT BINARY ADDER:



4-BIT BINARY SUBTRACTOR:

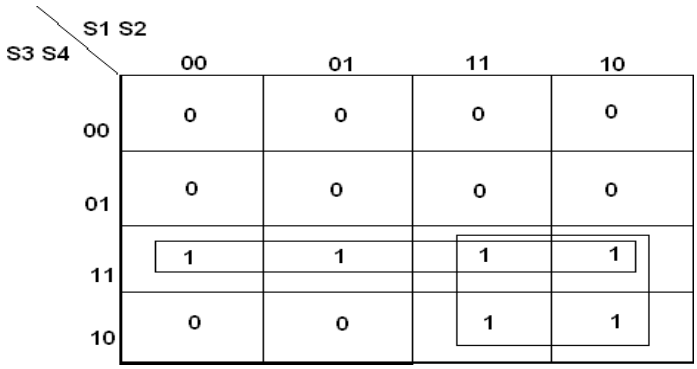


equal to 1, the input carry C0 is equal 1 and the sum output is A plus the 2's complement of B. when M is equal to 0, the input carry is equal to 0 and the sum generates A+B

TRUTH TABLE FOR BCD ADDER:

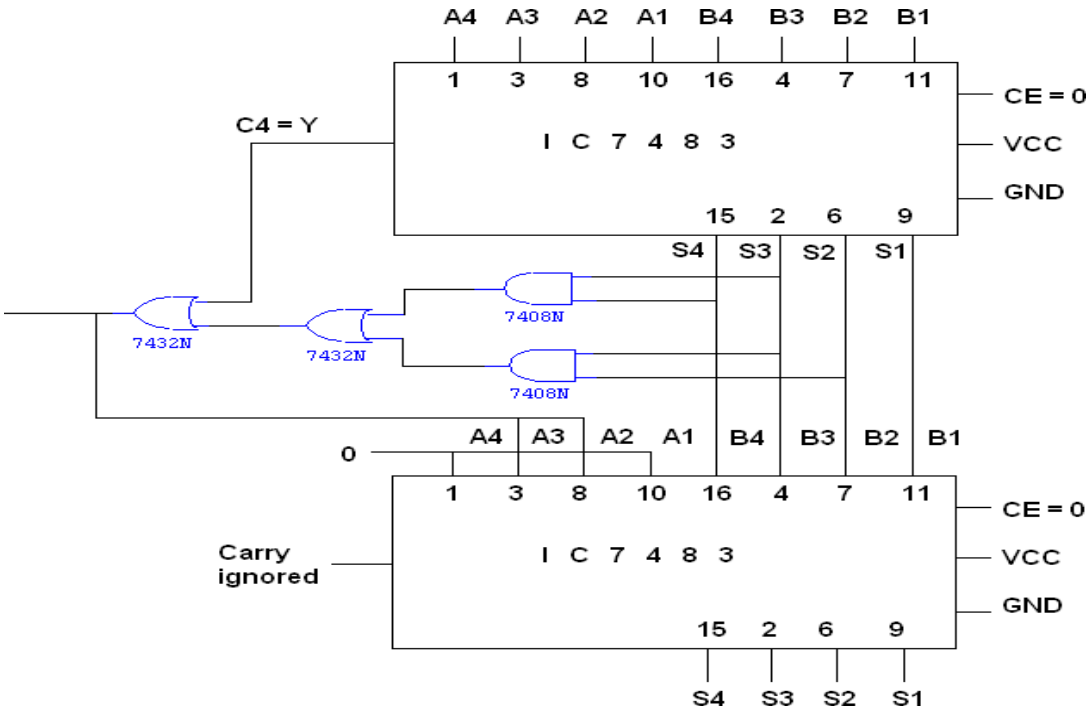
BCD SUM				CARRY
S4	S3	S2	S1	C
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

K MAP:



Y = S4 (S3 + S2)

LOGIC DIAGRAM OF BCD ADDER:



PROCEDURE:

1. Connections are given as per the Logic diagram.
2. Set mode M =0 such that the circuit will operate in addition mode.
3. Set the Value of inputs A as 1001 and B as 1001 note the sum and output carry.
4. Repeat the same step in step 3 by keeping M=1 such that circuit will operate in subtraction mode.

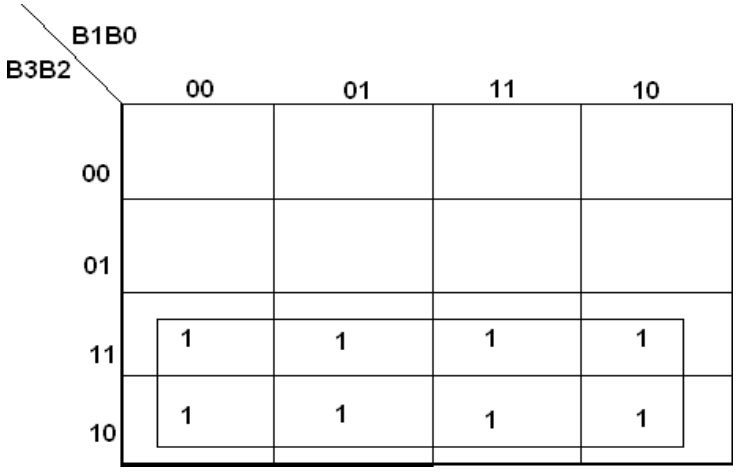
INFERENCE AND CONCLUSION:

Thus the 4 bit Binary Adder / Subtractor using IC7483 is been implemented for both addition and subtraction and the corresponding truth tables are verified.

TRUTH TABLE FOR BINARY TO GRAY CODE CONVERTER:

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

K-Map for G_3:



$G_3 = B_3$

Expt. No.	DESIGN AND IMPLEMENTATION OF CODE CONVERTERS	Date:

AIM:

To design and verify the truth table of the following code converters

1. Binary to Gray converter
2. Gray to Binary converter &
3. BCD to Excess3 &
4. Excess3 to BCD.

COMPONENTS / EQUIPMENTS REQUIRED:

S. No.	Components / Equipments	Specifications	Quantity
1.	Digital IC trainer	---	1
2.	NOT, AND, OR, Ex-OR Gate	IC7404,7408,7432,7486	1
3.	Connecting wires		Sufficient numbers

THEORY:

Binary to GRAY Converter:

By representing the ten decimal digits with a four bit Gray code, we have another form of BCD code. The Gray code however can be extended to any number of bits and conversion between binary code and Gray code is sometimes useful. The following rules apply for conversion:

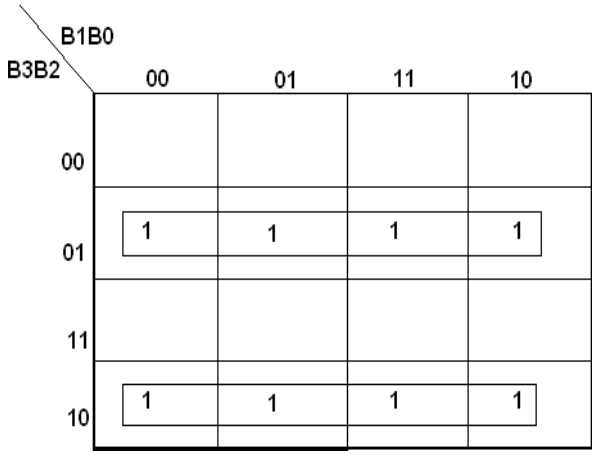
1. The MSB in the Gray code is the same as the corresponding bit in the binary number.
2. Going from left to right, add each adjacent pair of binary bits to get the next Gray code bit. Disregard carries.

GRAY to Binary Converter:

To convert from Gray code to binary code, A similar method is used, at there are some differences. The following rules apply:

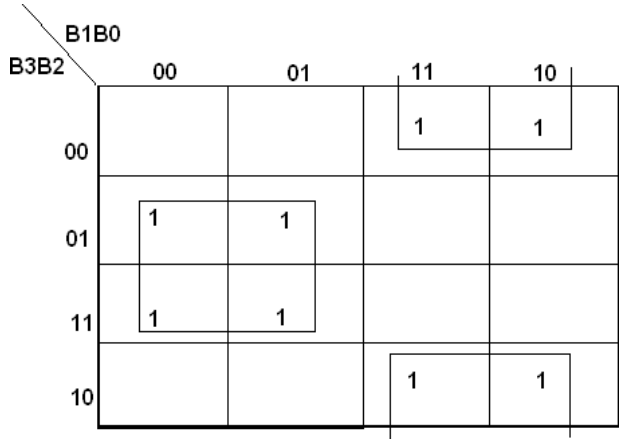
1. The MSB in the binary code is the same as the corresponding digit in the Gray code
2. Add each binary digit generated to the gray digit in the next adjacent position Disregard carries.

K-Map for G_2:



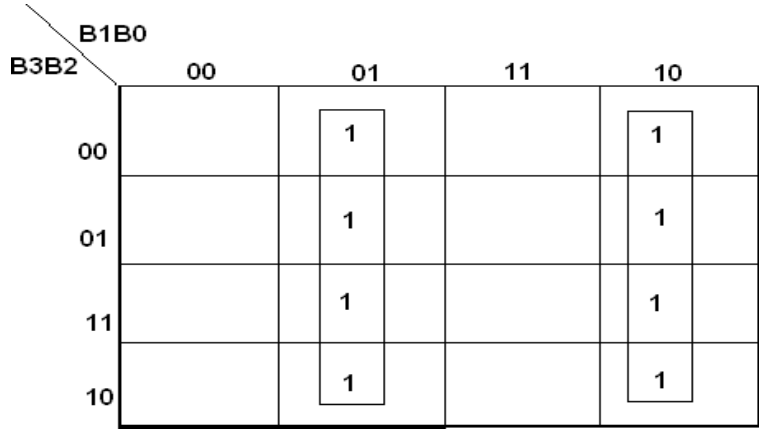
$G2 = B3 \oplus B2$

K-Map for G_1:



$G1 = B1 \oplus B2$

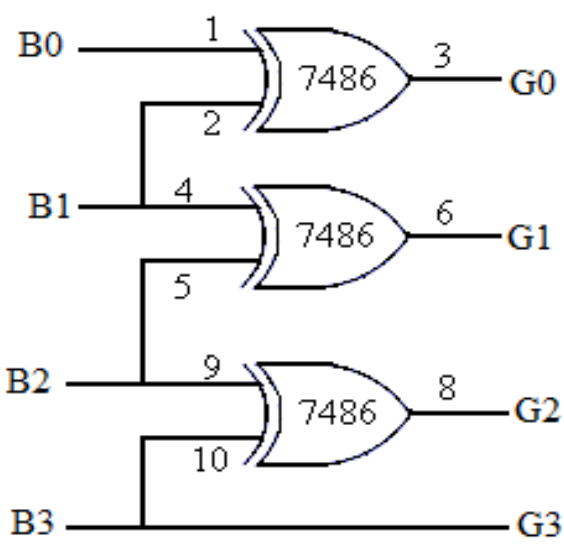
K-Map for G_0:



$G0 = B1 \oplus B0$

LOGIC DIAGRAM:

BINARY TO GRAY

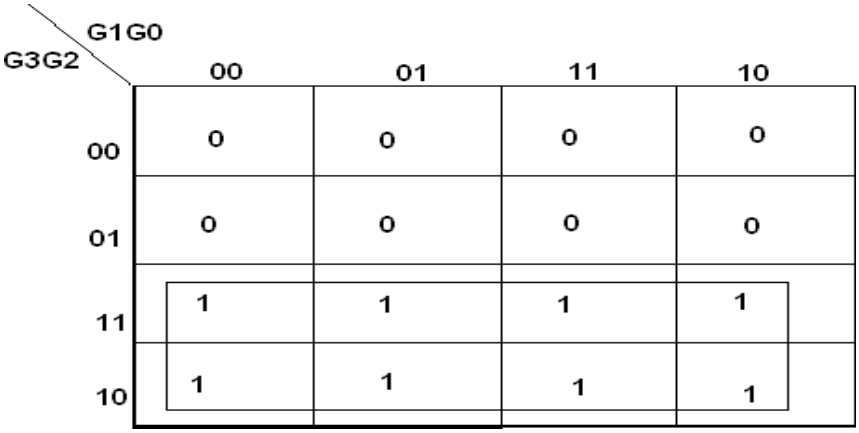


TRUTH TABLE FOR GRAY CODE TO BINARY CONVERTOR:

| Gray Code | Binary Code

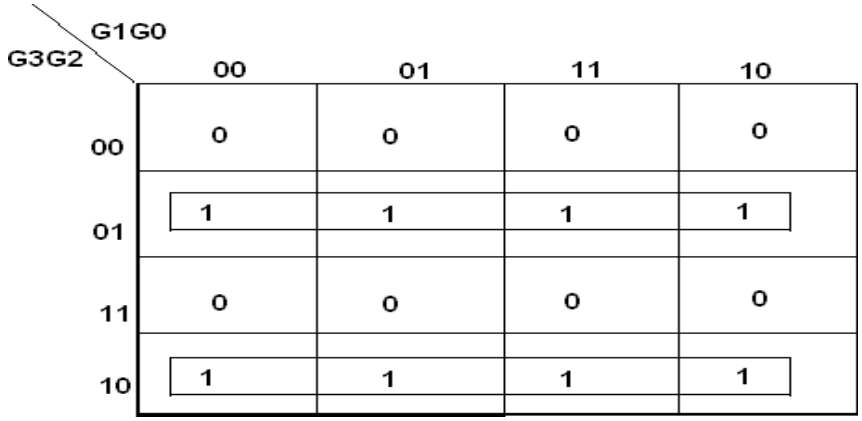
G3	G2	G1	G0	B3	B2	B1	B0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

K-Map for B_3:



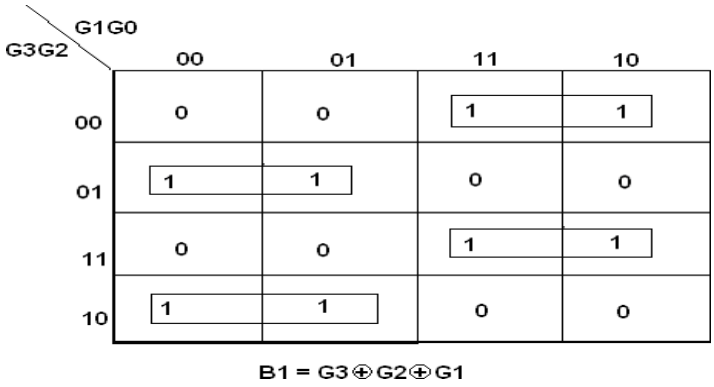
B3 = G3

K-Map for B_2:



B2 = G3⊕G2

K-Map for B_1:



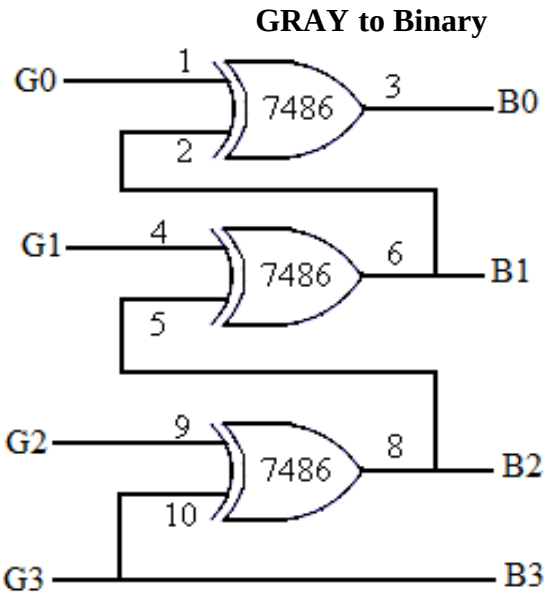
B1 = G3⊕G2⊕G1

K-Map for B_0:

G3G2 \ G1G0	00	01	11	10
00	0	①	0	①
01	①	0	①	0
11	0	①	0	①
10	①	0	①	0

$$B0 = G3 \oplus G2 \oplus G1 \oplus G0$$

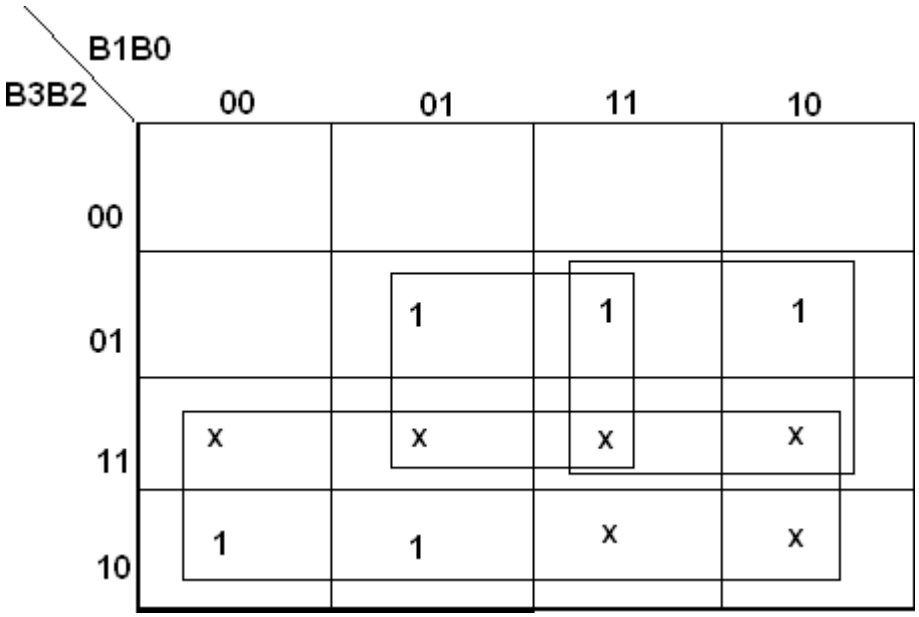
LOGIC DIAGRAM:



TRUTH TABLE FOR BCD TO EXCESS-3 CONVERTOR:

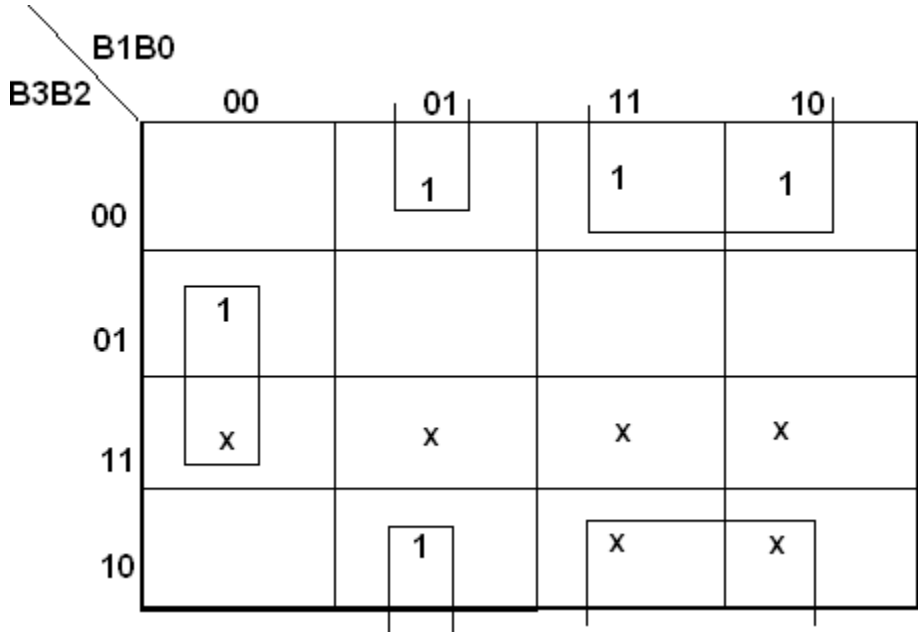
BCD input				Excess – 3 output			
B3	**B2**	**B1**	**B0**	**G3**	**G2**	**G1**	**G0**
0	**0**	**0**	**0**	**0**	**0**	**1**	**1**
0	**0**	**0**	**1**	**0**	**1**	**0**	**0**
0	**0**	**1**	**0**	**0**	**1**	**0**	**1**
0	**0**	**1**	**1**	**0**	**1**	**1**	**0**
0	**1**	**0**	**0**	**0**	**1**	**1**	**1**
0	**1**	**0**	**1**	**1**	**0**	**0**	**0**
0	**1**	**1**	**0**	**1**	**0**	**0**	**1**
0	**1**	**1**	**1**	**1**	**0**	**1**	**0**
1	**0**	**0**	**0**	**1**	**0**	**1**	**1**
1	**0**	**0**	**1**	**1**	**1**	**0**	**0**
1	**0**	**1**	**0**	**x**	**x**	**x**	**x**
1	**0**	**1**	**1**	**x**	**x**	**x**	**x**
1	**1**	**0**	**0**	**x**	**x**	**x**	**x**
1	**1**	**0**	**1**	**x**	**x**	**x**	**x**
1	**1**	**1**	**0**	**x**	**x**	**x**	**x**
1	**1**	**1**	**1**	**x**	**x**	**x**	**x**

K-Map for E_3:



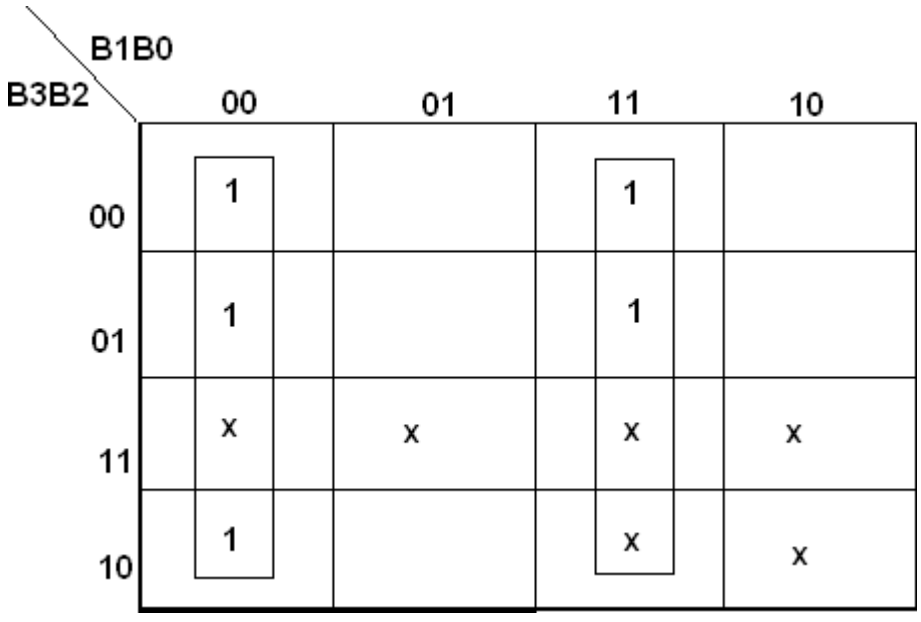
E3 = B3 + B2 (B0 + B1)

K-Map for E_2:



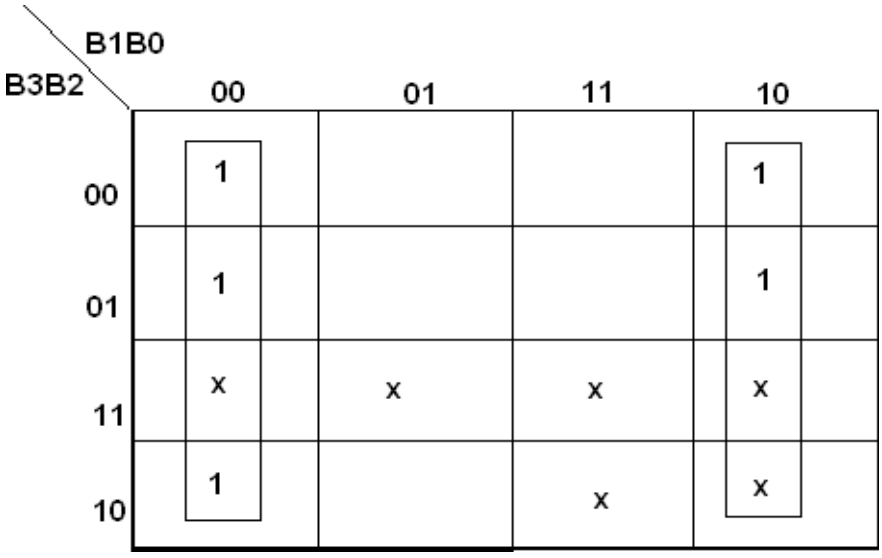
$E2 = B2 \oplus (B1 + B0)$

K-Map for E_1:



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

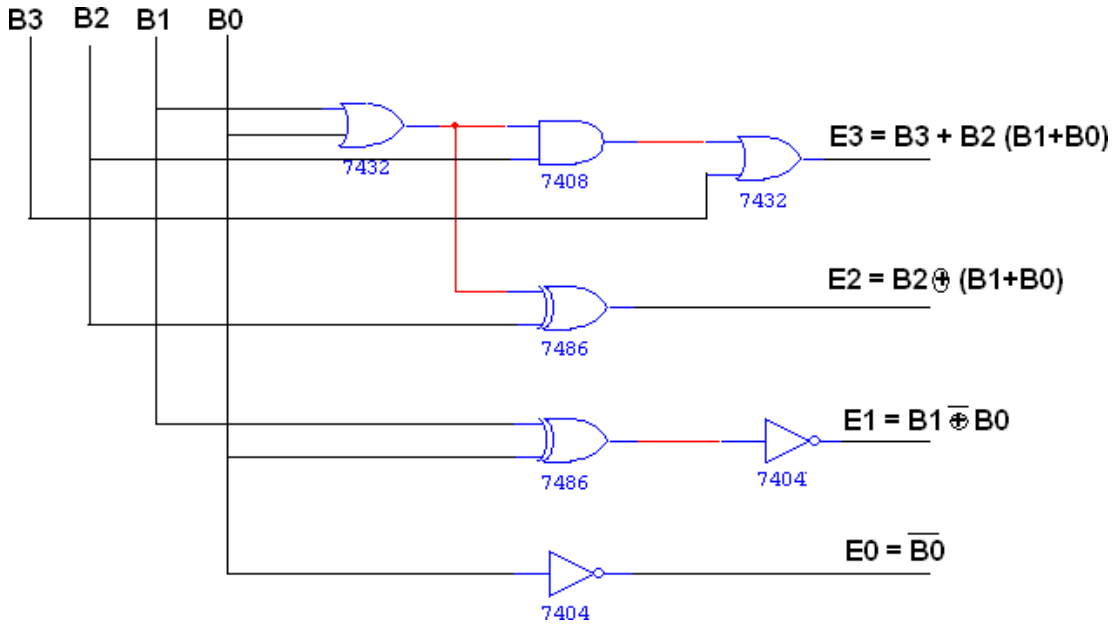
K-Map for E_0:



$E0 = \overline{B0}$

LOGIC DIAGRAM:

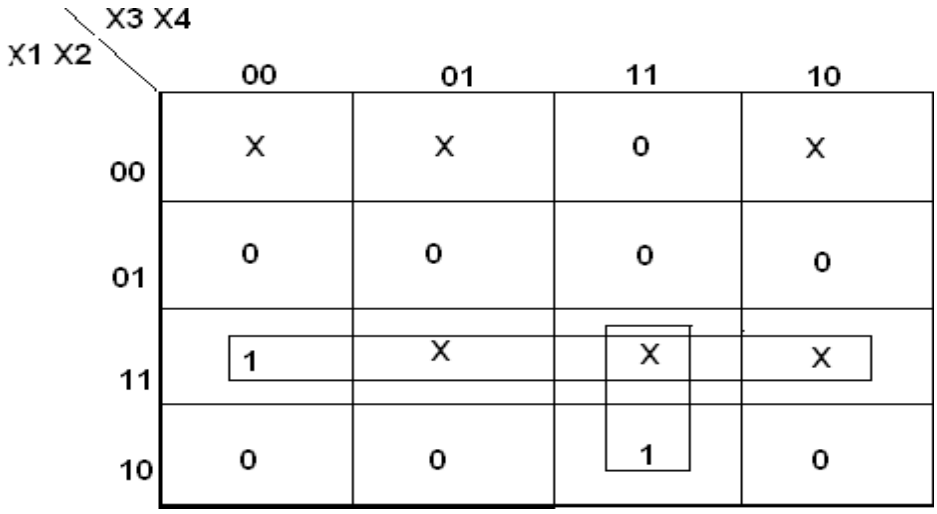
BCD TO EXCESS-3 CONVERTOR



TRUTH TABLE FOR EXCESS-3 TO BCD CONVERTOR:

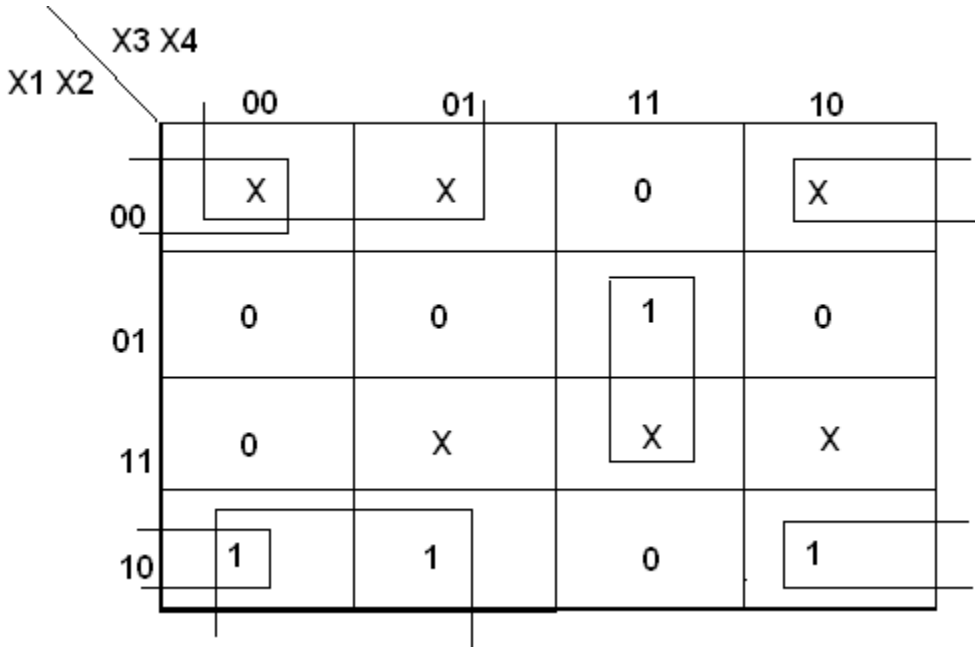
Excess – 3 Input				BCD Output			
B3	**B2**	**B1**	**B0**	**G3**	**G2**	**G1**	**G0**
0	**0**	**1**	**1**	**0**	**0**	**0**	**0**
0	**1**	**0**	**0**	**0**	**0**	**0**	**1**
0	**1**	**0**	**1**	**0**	**0**	**1**	**0**
0	**1**	**1**	**0**	**0**	**0**	**1**	**1**
0	**1**	**1**	**1**	**0**	**1**	**0**	**0**
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**

K-Map for A:



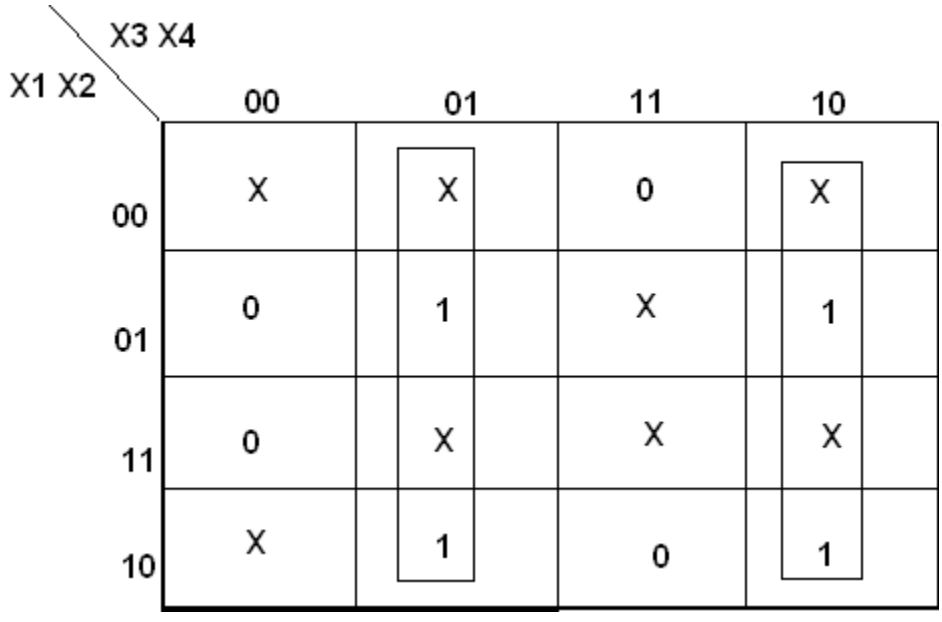
A = X1 X2 + X3 X4 X1

K-Map for B:



$$B = X2 \oplus (\overline{X3} + \overline{X4})$$

K-Map for C:



$$C = X3 \oplus X4$$

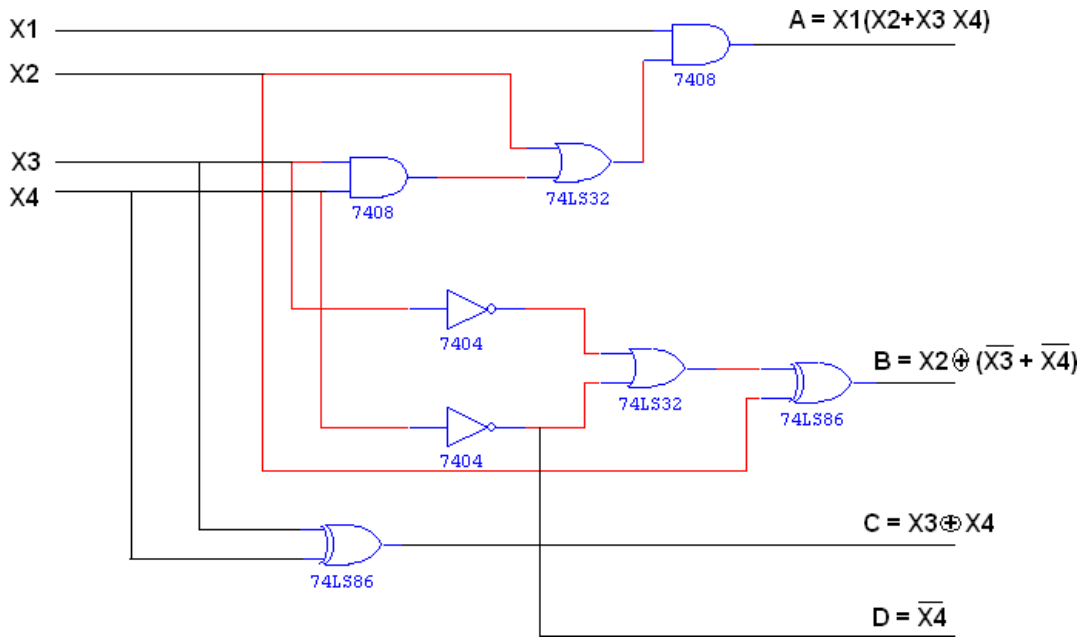
K-Map for D:

X1 X2 \ X3 X4	00	01	11	10
00	X	X	0	X
01	1	0	0	1
11	1	X	X	X
10	1	0	0	1

$D = \overline{X4}$

LOGIC DIAGRAM:

EXCESS-3 TO BCD CONVERTOR



PROCEDURE:

1. Connections are given as per the Logic diagram .
2. Switch on the power supply.
3. Verify the truth table given for different inputs.
4. Repeat the above procedures for other converters.

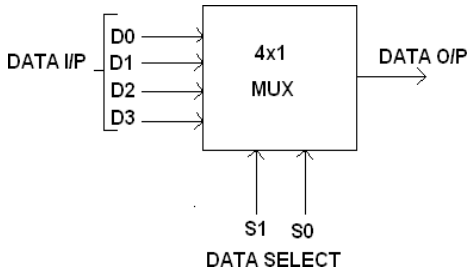
INFERENCE AND CONCLUSION:

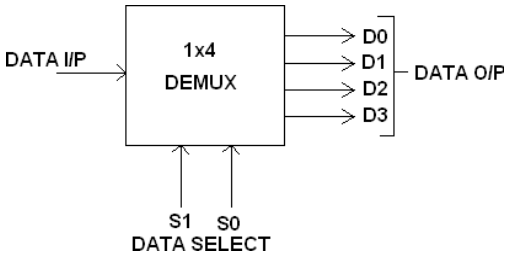
Thus the truth tables for Binary to Gray, Gray to Binary and BCD to Excess3 converters were verified.

REVIEW QUESTIONS:

1. State some examples of weighted codes.
2. Convert Gray code to binary: 11010101.
3. What is the significance of gray code?
4. Express the following decimals number into excess 3.(i)124 (ii) 7621
5. What is the use of parity bit?

BLOCK DIAGRAM FOR 4:1 MULTIPLEXER & 1:4 DEMULTIPLEXER:





MULTIPLEXER

Logic Diagram:

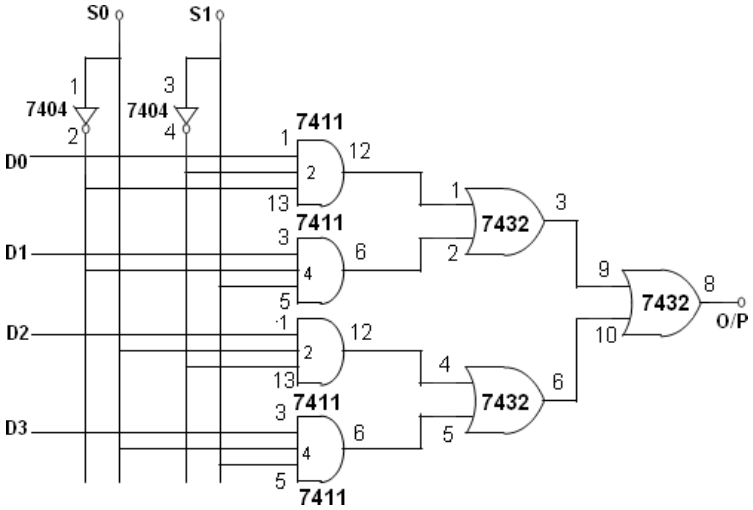


Fig: 8.1

Truth Table:

S1	S0	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

Tab: 8.1

DEMULTIPLEXER

Logic Diagram:

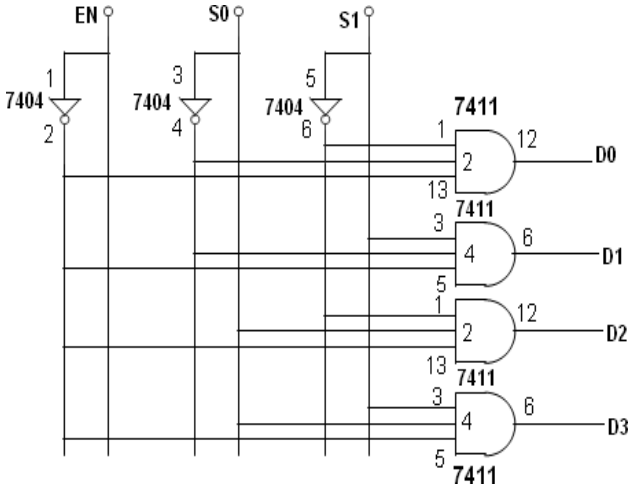


Fig: 8.2

Truth Table:

Selection Lines		**OUTPUT V0**
S1	S0	
0	0	D0=Di
0	1	D1= Di
1	0	D2= Di
1	1	D3= Di

Tab: 8.2

Expt. No:	DESIGN AND IMPLEMENTATION OF MULTIPLEXER AND DEMULTIPLEXER USING LOGIC GATES	Date:

AIM:

To design and implement multiplexer and demultiplexer using logic gates

COMPONENTS / EQUIPMENTS REQUIRED:

S.No.	Name of the apparatus	Specification	Quantity
1	Digital Trainer kit	-	
2	OR gate	IC7432	1
3	AND gate	IC7411	1
4	NOT gate	IC7404	1
5	Connecting wires	-	

THEORY:

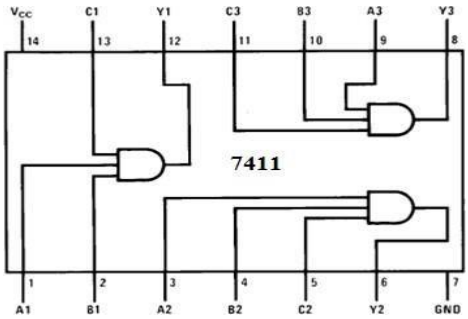
Multiplexer:

It has a group of data inputs and a group of control inputs. The control inputs are used to select one of the data inputs and connected to the output terminal. It selects one information out of many information lines and directed to a single output line.

Demultiplexer:

Demultiplexers perform the opposite function of multiplexers. They transfer a small number of information units (usually one unit) over a larger number of channels under the control of selection signals. Fig shows a 1-line to 2-line Demultiplexer circuit. Construct this circuit; connect an LED to each of the outputs D0 and D1. Set the select signal S to logic 1 or logic 0, and toggle the input I between logic 1 and logic 0. Which output followed the input when S = 1 and S = 0.

Pin Diagram for 3 Pin AND GATE IC 7411:



PROCEDURE:

1. Connections are given as per in the Logic diagram.
2. Inputs are given through the logic switches.
3. Outputs are noted and verified with truth table

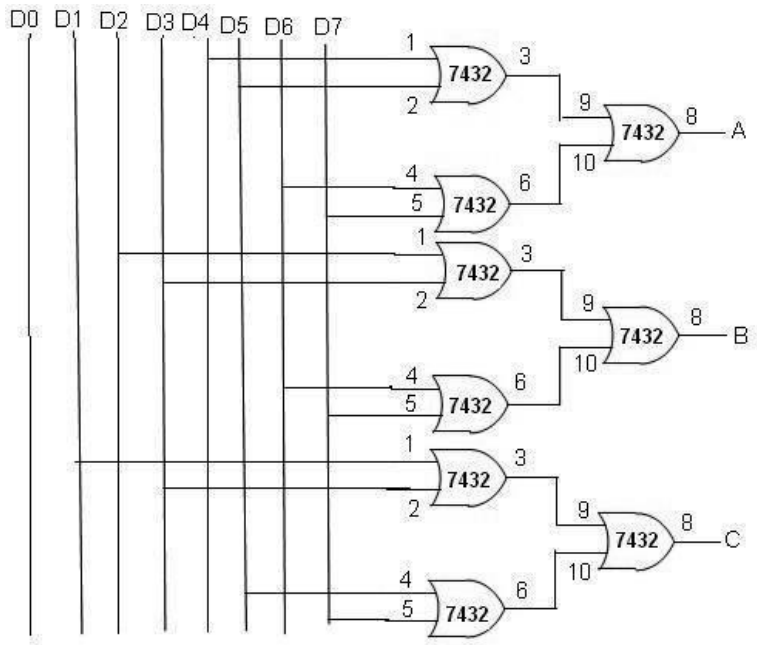
INFERENCE AND CONCLUSION:

Thus the truth table of multiplexer and demultiplexer was studied and verified using logic gates.

VIVA QUESTIONS:

1. What is a multiplexer?
2. What are the applications of multiplexer?
3. What is the difference between multiplexer & demultiplexer?
4. In $2^{n:}$ 1 multiplexer how many selection lines are used?
5. Draw a 2 to 1 multiplexer circuit
6. Draw a 1 to 2 demultiplexer circuit.

CIRCUIT DIAGRAM: **Encoder**



Truth Table:

INPUT								OUTPUT		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	A	B	C
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Outputs:

$A = D_4 + D_5 + D_6 + D_7$

$B = D_2 + D_3 + D_6 + D_7$

$C = D_1 + D_3 + D_5 + D_7$

Expt. No.	DESIGN AND IMPLEMENTATION OF ENCODER	Date:

AIM:

To construct and verify the 8 X 3 Encoder.

COMPONENTS / EQUIPMENTS REQUIRED:

S. No	Components / Equipments	Specification	Quantity
1.	Digital IC trainer kit	-	1
2.	OR Gate	IC7432	3
3.	Connecting Wires	-	Sufficient Numbers

THEORY:

Digital Computers, Microprocessors and other digital systems are binary operated whereas our language of communication is in decimal numbers and alphabetical characters only. Therefore, the need arises for interfacing between digital system and human operators. To accomplish this task, Encoder is used.

PROCEDURE:

1. Construct the circuit as per the diagram
2. Switch on the power supply.
3. Apply the necessary input and observe the outputs to verify the truth table.

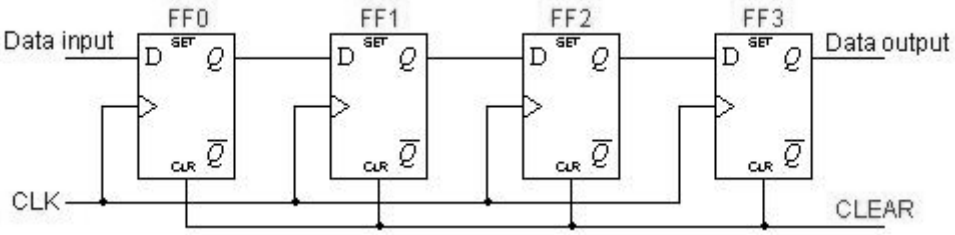
INFERENCE AND CONCLUSION:

Thus an 8 x 3 encoder is constructed and verified.

REVIEW QUESTIONS:

1. Draw the basic block diagram of a practical decoder.
2. What is the need for decoder?
3. Name the procedure involved in decoding.
4. Give some practical applications where decoding is necessary.
5. List the advantages of decoding.

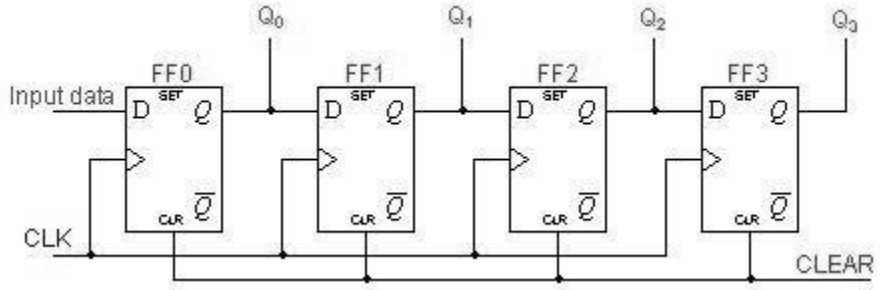
Serial in Serial out:



CLEAR	FF0	FF1	FF2	FF3
	0	0	0	0

Fig: 9.1

Serial in Parallel Out:



CLEAR 1001	Q0	Q1	Q2	Q3
	0	0	0	0

Fig: 9.2

Parallel In Parallel Out

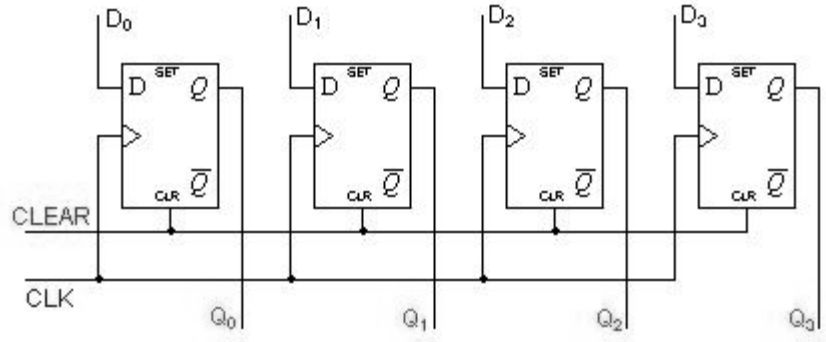


Fig: 9.3

Expt. No:	IMPLEMENTATION OF SISO, SIPO, PISO AND PIPO SHIFT REGISTERS	Date:

AIM:

To implement the 4 bit shift register using flip flops and to study the operations in the following modes.

(i) Serial in serial out
(ii) Serial in parallel out
(iii) Parallel in parallel out
(iv) Parallel in serial out

COMPONENTS / EQUIPMENTS REQUIRED:

S.No.	Name of the apparatus	Range	Quantity
1	Digital Trainer kit		1
2	D Flip Flop	IC 7474	2
3	Connecting wires		some

THEORY:

SHIFT REGISTER:

A register is a device capable of storing a bit. The data can be serial or parallel. The register can convert a data from serial to parallel and vice versa shifting then digits to left and right is the important aspect for arithmetic operations,

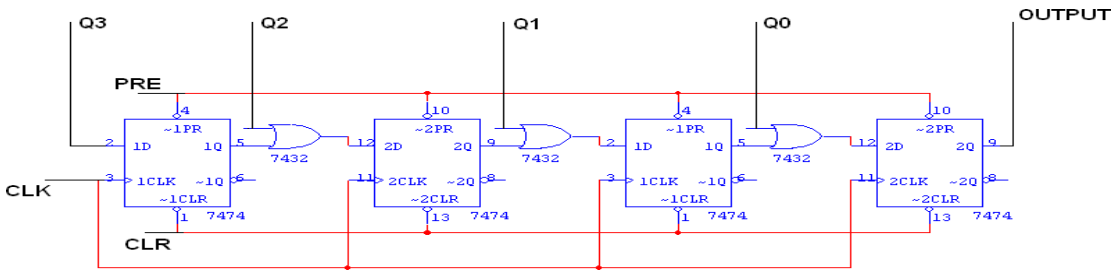
A register capable of shifting its binary information either to the right or to the left is called a shift register. An N bit shift register consists of N flip-flops and the gates that control the shift operation. A shift register can be used in four different configurations depending upon the way in which the data are entered into and taken out of it. These four configurations are:

- Serial-input, Serial-output
- Parallel-input, Serial-output
- Serial-input, parallel-output
- Parallel-output, parallel-output

The serial input is a single line going to the input of the leftmost flip-flop of the register. The serial output is a single line from the output of the rightmost flip-flop of the register, so that the bits stored in the register can come out through this line one at a time.

The parallel output consists of N lines, one for each of the flip-flops in the register, so the information stored in the register can be inspected through these lines all at once.

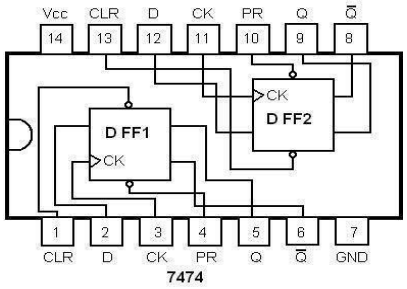
PARALLEL IN SERIAL OUT

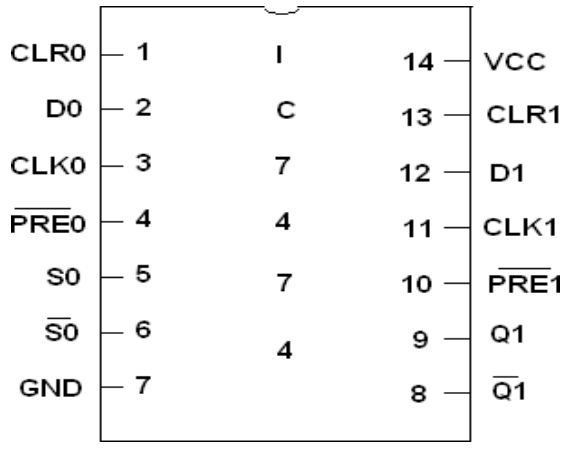


TRUTH TABLE FOR PISO SHIFT REGISTER:

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

Pin Diagram IC7474:





PROCEDURE:

1. The flip-flop is connected using connecting wires as shown in the circuit.
2. The flip flop are then reset to zero internally with the help of reset to set inputs.
3. The bits are shifted in by giving suitable clock input.
4. Thus the truth table is then verified.

INFERENCE AND CONCLUSION:

Thus the operation of 4 bit shift register for SISO, SIPO, and PIPO was studied and verified.

Expt. No:	DESIGN AND IMPLEMENTATION OF SYNCHRONOUS COUNTER	Date:

AIM:

- To construct and verify the synchronous up/down Counters.
- To design and implementation of 4-bit synchronous Johnson counter using D flip flop.

COMPONENTS / EQUIPMENTS REQUIRED:

S.No.	Name of the apparatus	Range	Quantity
1	Digital Trainer kit		1
2	D Flip Flop JK Flip-Flop, AND Gate	IC 7474 IC 7473,7408	2 2,1
3	Connecting wires		some

THEORY:

Johnson counter

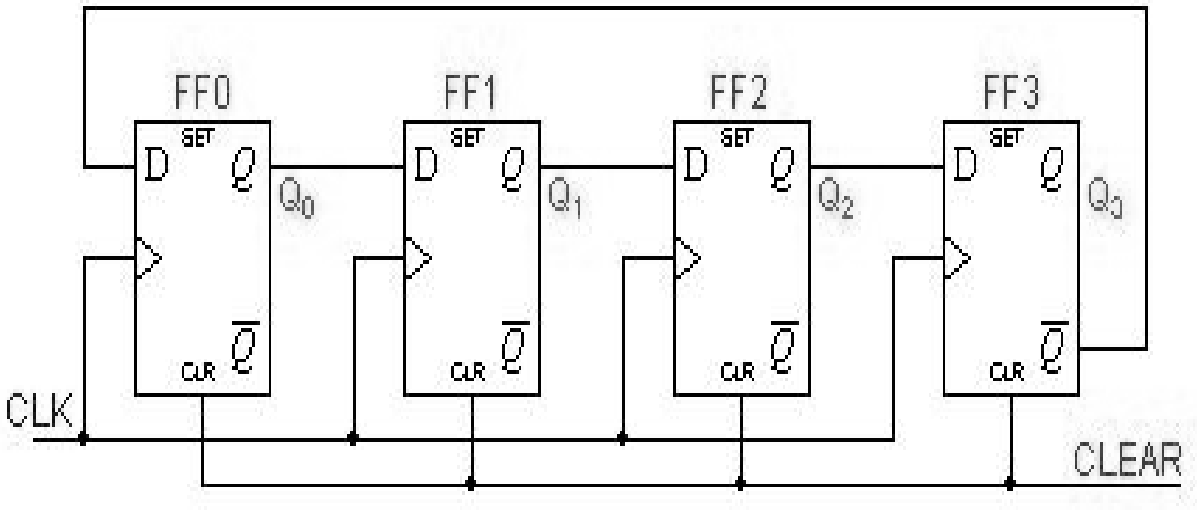
The Johnson counter, also known as the twisted-ring counter, is exactly the same as the ring counter except that the inverted output of the last flip-flop is connected to the input of the first flip-flop.

The Johnson counter works in the following way: Take the initial state of the counter to be 000. On the first clock pulse, the inverse of the last flip-flop will be fed into the first flip-flop, producing the state 100. On the second clock pulse, since the last flip-flop is still at level 0, another 1 will be fed into the first flip-flop, giving the state 110. On the third clock pulse, the state 111 is produced. On the fourth clock pulse, the inverse of the last flip-flop, now a 0, will be shifted to the first flip-flop, giving the state 011. On the fifth and sixth clock pulse, using the same reasoning, we will get the states 001 and 000, which is the initial state again. Hence, this Johnson counter has six distinct states: 000, 100, 110, 111, 011 and 001, and the sequence is repeated so long as there is input pulse. Thus this is a MOD-6 Johnson counter.

A Johnson counters represent a middle ground between ring counters and binary counters. A Johnson counter requires fewer flip-flops than a ring counter but generally more than a binary counter; it has more decoding circuitry than a ring counter but less than a binary counter.

Johnson counter

Circuit Diagram:



4-bit Synchronous Johnson Counter

Fig: 10.1

Truth Table:-

Clock Pulse	Q3	Q2	Q1	Q0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	1	1	1
4	1	1	1	1
5	1	1	1	0
6	1	1	0	0
7	1	0	0	0

CLEAR	FF0	FF1	FF2	FF3
	0	0	0	0

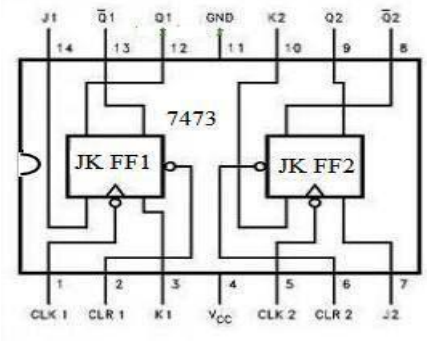
Tab: 10.1

THEORY:

Synchronous Counter

Clock input is applied simultaneously to all flip-flops. The output of the first FLIP-FLOP is connected to the input of second FLIP-FLOP and so on.
Design of synchronous counter
Step 1: Find the number of flip-flops required. For an n-bit counter, n- flip-flops is required. Step 2: Write the count sequence in tabular form.
Step 3: Determine the flip-flop inputs, which must be present for the desired next State from the present state using excitation table of flip-flops.
Step 4: Prepare K-map for each flip-flop input in terms of flip-flop output as input
Variables. Simplify the K-map and obtain the minimized expressions.
Step 5: Connect the circuit using the flip-flops.

Pin Diagram for IC7473:



PROCEDURE:

1. The connections are made as per the circuit diagram.
2. Switch on the power supply.
3. The input is given at the appropriate terminal and corresponding output is observed and truth table is verified.

INFERENCE AND CONCLUSION:

Thus, the operations of 4 bit synchronous Johnson counter using D flip flop and 3 bit Synchronous Up/Down Counter was studied and verified.

VIVA QUESTIONS:

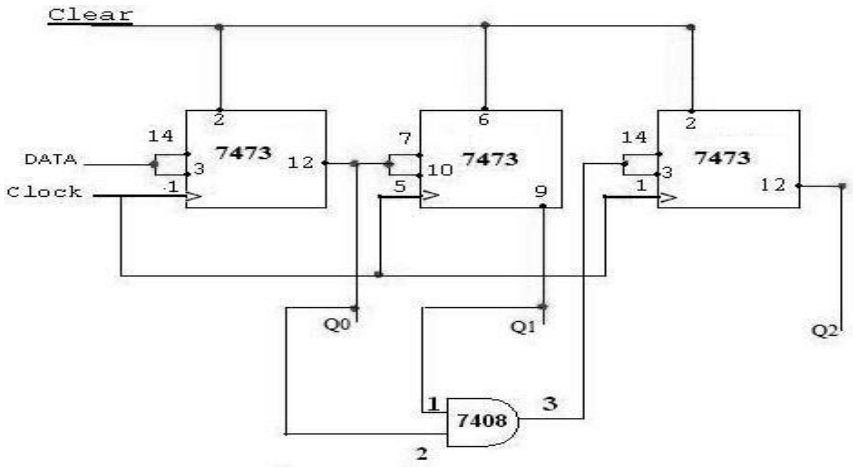
1. What is the difference between Register & counter?

2. What is Johnson Counter?

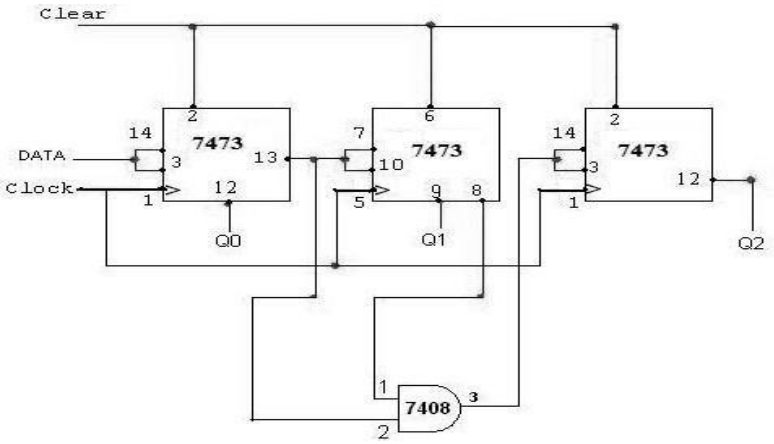
3. What is ring counter?

4. Which flip flop is used in counters?

CIRCUIT DIAGRAM: ***Design of 3-bit synchronous up:***



Design of 3-bit synchronous down counter:



Pin Diagram

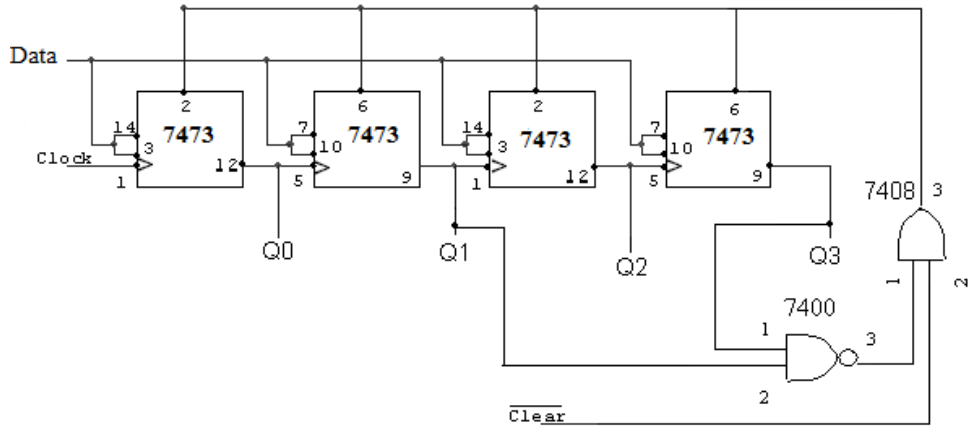
Truth Table:

3 Bit Synchronous UP Counter				***3 Bit Synchronous DOWN Counter***			
Clock	**Q2**	**Q1**	**Q0**	**Clock**	**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

Truth Table:

MOD 12 Counter				
Clock	**Q0**	**Q1**	**Q2**	**Q3**
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	0	0	0	0

MOD 10 COUNTERS



Truth Table:

MOD 10 Counter				
Clock	**Q0**	**Q1**	**Q2**	**Q3**
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	0	0	0	0

Simulator based study of Computer Architecture.

A computer architecture simulator is a program that simulates the execution of computer architecture.

Computer architecture simulators are used for the following purposes:

- Lowering cost by evaluating hardware designs without building physical hardware systems.
- Enabling access to unobtainable hardware.
- Increasing the precision and volume of computer performance data.
- Introducing abilities that are not normally possible on real hardware such as running code backwards when an error is detected or running in faster-than-real time.

  Computer architecture simulators can be classified into many different categories depending on the context.
- Scope:

  Microarchitecture simulators model the microprocessor and its components. Full-system simulators also model the processor, memory systems, and I/O devices.
- Detail: Functional simulators, such as instruction set simulators, achieve the same function as modeled components. They can be simulated faster if timing is not considered. Timing simulators are functional simulators that also reproduce timing. Timing simulators can be further categorized into digital cycle-accurate and analog sub-cycle simulators.
- Workload: Trace-driven simulators (also called event-driven simulators) react to pre-recorded streams of instructions with some fixed input. Execution-driven simulators allow dynamic change of instructions to be executed depending on different input data.

Full-system simulators

A full-system simulator is execution-driven architecture simulation at such a level of detail that complete software stacks from real systems can run on the simulator without any modification. A full system simulator provides virtual hardware that is independent of the nature of the host computer. The full-system model typically includes processor cores, peripheral devices, memories, interconnection buses, and network connections. Emulators are full system simulators that imitate obsolete hardware instead of under development hardware.

The defining property of full-system simulation compared to an instruction set simulator is that the model allows real device drivers and operating systems to be run, notjust single programs. Thus, full-system simulation makes it possible to simulate individual computers and networked computer nodes with all their software, from network device drivers to operating systems, network stacks, middleware, servers, and application programs.

Full system simulation can speed the system development process by making it easier to detect, recreate and repair flaws. The use of multi-core processors is driving the need for full system simulation, because it can be extremely difficult and time-consuming to recreate and debug errors without the controlled environment provided by virtual hardware.[1] This also allows the software development to take place before the hardware is ready,[2] thus helping to validate design decisions.

Cycle-accurate simulator

A cycle-accurate simulator is a computer program that simulates a microarchitecture on a cycle-by-cycle basis. In contrast an instruction set simulator simulates an instruction set architecture usually faster but not cycle-accurate to a specific implementation of this architecture; they are often used when emulating older hardware, where time precision is important for legacy reasons. Often, a cycle-accurate simulator is used when designing new microprocessors – they can be tested, and benchmarked accurately (including running full operating system, or compilers) without actually building a physical chip, and easily change design many times to meet expected plan.

Cycle-accurate simulators must ensure that all operations are executed in the proper virtual (or real if it is possible) time – branch prediction, cache misses, fetches, pipeline stalls, thread context switching, and many other subtle aspects of microprocessors.