AVIT
AARUPADAI VEEDU INSTITUTE OF TECHNOLOGY
VINAYAKA MISSION'S RESEARCH FOUNDATION
(Deemed to be University under section 3 of the UGC Act 1956)
Accredited by NAAC
Approved by AICTE

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DIGITAL LOGIC CIRCUITS & DESIGN LAB

Program/ Branch	**: B. E., /ECE**
Year / Semester	**: II/ III**
Academic Year	**: 2021 – 2022 (Odd Semester)**
Regulation	**: R 2017**

[illegible]

HOD / ECE

17ECCC82	DIGITAL LOGIC CIRCUITS & DESIGN LAB	Category	L	T	P	Credit
		CC	0	0	4	2

PREAMBLE

To provide experience & explore designs in analyzing and testing of digital logic circuits like combinational and sequential circuits using lab instruments as well as simulation software.

Prerequisite : Basic Electrical and Electronics Engineering

PRERQUISITE

17EEES03 - Basics of Electrical and Electronics Engineering

COURSE OBJECTIVES

1	To impart the knowledge in analysis and design of various combinational logic circuits.
2	To learn about design and analysis of sequential circuits using flip flops.
3	To Expose students about design and simulation of logic circuits using HDL.

COURSE OUTCOMES

On the successful completion of the course, students will be able to

CO1.Construct various logic circuits.	Apply
CO2. Demonstrate the various combinational logic circuits by using discrete components	Apply
CO3. Analyze different sequential logic circuits by using discrete components.	Analyze
CO4. Test the various digital logic circuits by using simulation software.	Evaluate
CO5. Measure and record the experimental data for various digital circuits.	Evaluate

MAPPING WITH PROGRAMME OUTCOMES AND PROGRAMME SPECIFIC OUTCOMES

COs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	S	-	-	-	M	-	-	-	M	-	-	L	S	-	-
CO2	S	-	-	-	S	-	-	-	M	-	-	L	S	M	-
CO3	S	M	M	M	M	-	-	-	M	-	-	L	S	M	-
CO4	S	M	-	-	M	-	-	-	M	-	-	L	S	S	M
CO5	S	M	-	-	M	-	-	-	M	-	-	L	S	M	-

S- Strong; M-Medium; L-Low

List of Experiments

Hardware Experiments

1. Design and implementation of Adders using logic gates.
2. Design and implementation of Sub tractors using logic gates.
3. Design and implementation of BCD to Excess -3 code converter using logic gates
4. Design and implementation of Binary to Gray code converter using logic gates
5. Design and implementation of 4 bit BCD adder using IC 7483
6. Design and implementation of 2 Bit Magnitude comparator using logic gates
7. Design and implementation of Multiplexer and De-Multiplexer using logic gates
8. Design and implementation of encoder and decoder using logic gates
9. Design and implementation of 3 bit synchronous up/down counter.
10. Implementation of SISO, SIPO, and PISO shift registers using flip flops.

Software Experiments using HDL

1. Design and Simulation of Full adder circuit using Gate level modelling
2. Design and Simulation of 2X2 multiplier circuit using structural level modeling.
3. Design and Simulation of 8 to 1 Multiplexer circuit using behavioural level modeling.

COURSE DESIGNERS

S.No.	Name of the Faculty	Designation	Department	Mail ID
1	Mr.B.Rajasekaran	Associate Professor	ECE	rajasekaran@vmkvec.edu.in
2	Mrs.S.Valarmathy	Associate Professor	ECE	valarmathy@vmkvec.edu.in
3	Ms.R.Mohana Priya	Assistant Professor (Gr-II)	ECE	mohanapriya@avit.ac.in

[illegible]

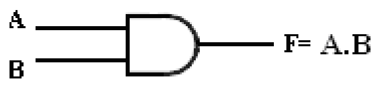
HOD / ECE

[illegible]

HOD / ECE

AND gate:

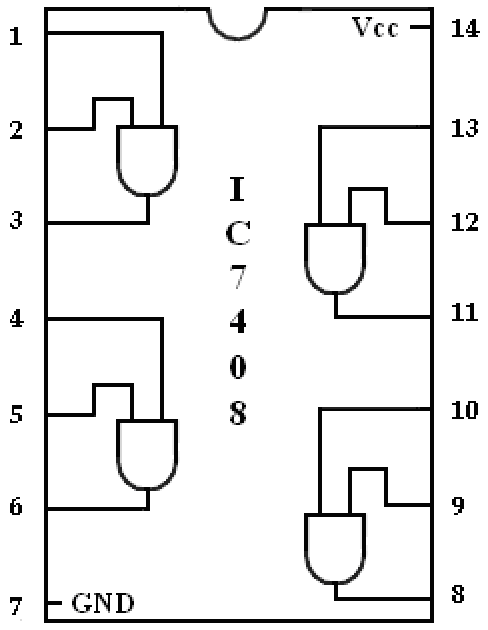
SYMBOL



TRUTH TABLE

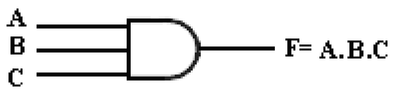
A	B	F= A.B
0	0	0
0	1	0
1	0	0
1	1	1

PIN DIAGRAM



3-Input AND gate:

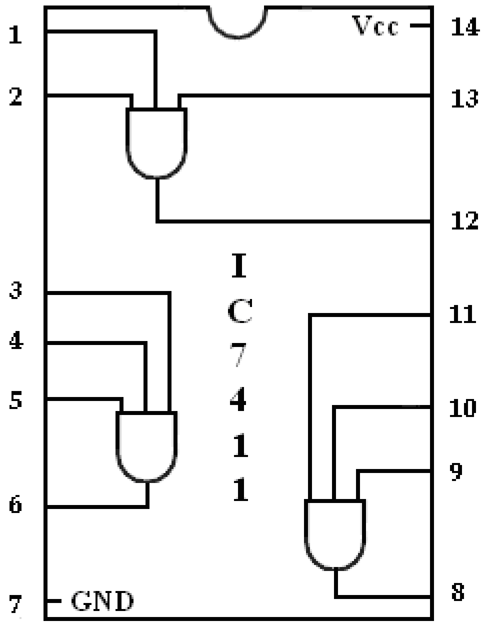
SYMBOL



TRUTH TABLE

A	B	C	F= A.B.C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

PIN DIAGRAM



[illegible]

HOD / ECE

STUDY OF LOGIC GATES

AIM:

To study about logic gates and verify their truth tables**.**

APPARATUS REQUIRED:

SL.NO	COMPONENTS	SPECIFICATION	QUANTITY
1.	2- I/P AND gate	IC 7408	1
2.	3-I/P AND gate	IC7411	1
3.	OR gate	IC 7432	1
4.	NOT gate	IC 7404	1
5.	2- I/P NAND gate	IC7400	1
6.	3-I/P NAND gate	IC7410	1
7.	NOR gate	IC7402	1
8.	EX-OR gate	IC7486	1
9.	IC Trainer Kit	-	1
10.	Patch cords	-	Few

THOERY:

Logic gates are the basic elements that make up a digital system. The gate is a digital circuit with one or more inputs, but only one output. By connecting the different gates in different ways, we can build circuits that perform arithmetic and other functions.

The operation of a logic gate can be easily understood with the help of "truth table". A truth table is a table that shows all the input-output possibilities of a logic circuit ie., the truth table indicates the outputs for different possibilities of the inputs.

The types of gates available are the AND, OR, NOT, NAND, NOR, exclusive-OR and the exclusive-NOR. Except for the exclusive-NOR gate they are available in monolithic integrated form.

AND gate:

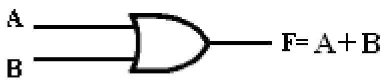
The AND gates has two or more inputs. It performs a logical multiplication. The output is HIGH (1), when both the inputs are 1; otherwise the output from the gate is LOW (0). The output from the AND gate is written as **A.B**.

[illegible]

HOD / ECE

OR gate:

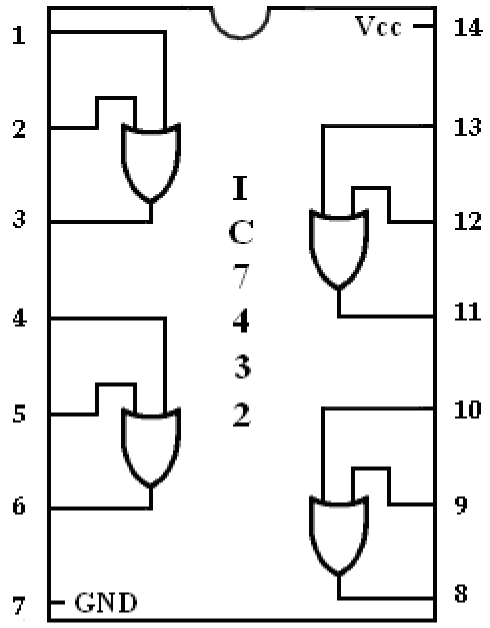
SYMBOL



TRUTH TABLE

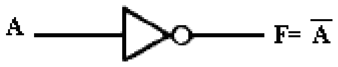
A	B	F= A+B
0	0	0
0	1	1
1	0	1
1	1	1

PIN DIAGRAM



NOT gate:

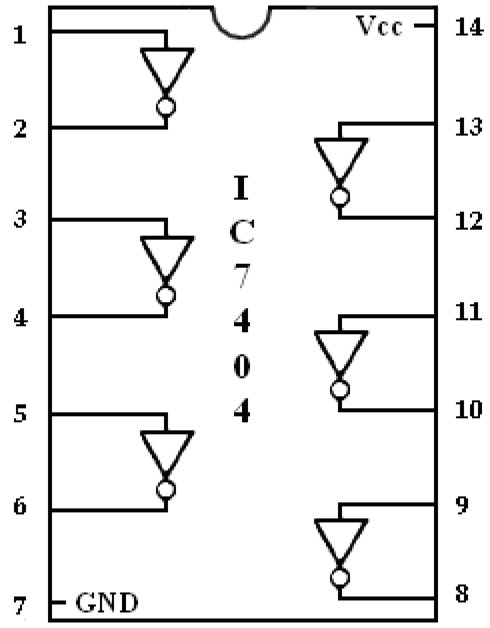
SYMBOL



TRUTH TABLE

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

PIN DIAGRAM



[illegible]

HOD / ECE

OR gate:

The OR gates has two or more inputs. It performs a logical addition. The output is HIGH (1), if any of the inputs are 1; the output is LOW (0) if and only if all the inputs are 0. The output from the OR gate is written as **A+B**.

NOT gate:

The NOT gate has only one input. It performs a basic logic function called inversion. The output is HIGH (1), when the input is 0; the output is LOW (0) when the input is 1. The output from the NOT gate is written as **A'**.

NAND gate:

The NAND gate is a contraction of AND-NOT. It has two or more inputs. The output is HIGH (1), when any of the inputs are 0; the output is LOW (0), if and only if all the inputs are 1. The output from the AND gate is written as (**A.B)'**. It is a universal gate.

NOR gate:

The NOR gate is a contraction of OR-NOT. It has two or more inputs. The output is HIGH (1), when all inputs are 0; the output is LOW (0), when any of the inputs are 1. The output from the AND gate is written as (**A+B)'**. It is a universal gate.

EX-OR gate:

The EX-OR gate has two or more inputs. The output is HIGH (1), when odd number of inputs is 1. The output from the AND gate is written as (**A⊕B).**

<u>PROCEDURE:</u>

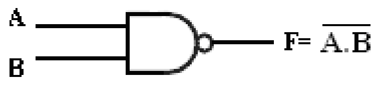
1. Connections are given as per the logic diagram.
2. Logic inputs are given as per the truth table.
3. Observe the logic output and verify with the truth table.

[illegible]

HOD / ECE

2-Input NAND gate:

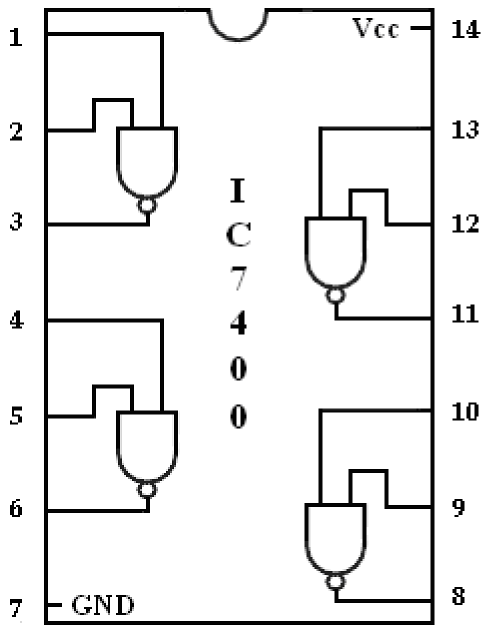
SYMBOL



TRUTH TABLE

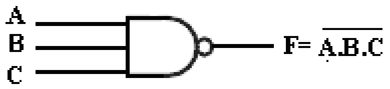
A	B	F= $\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

PIN DIAGRAM



3-Input NAND gate:

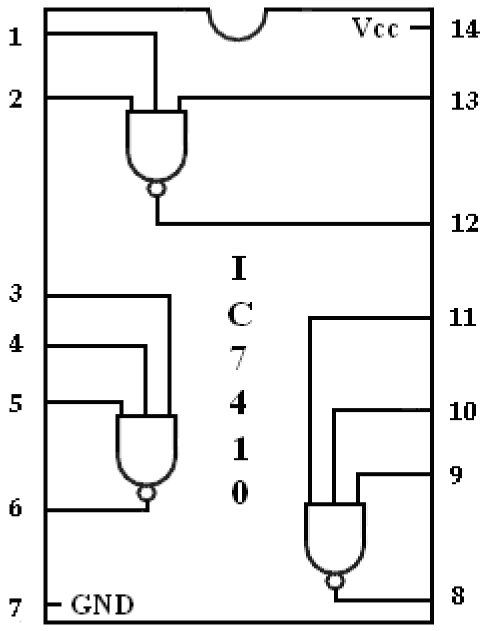
SYMBOL



TRUTH TABLE

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

PIN DIAGRAM

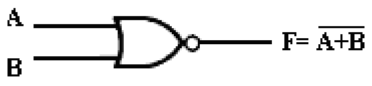


[illegible]

HOD / ECE

NOR gate:

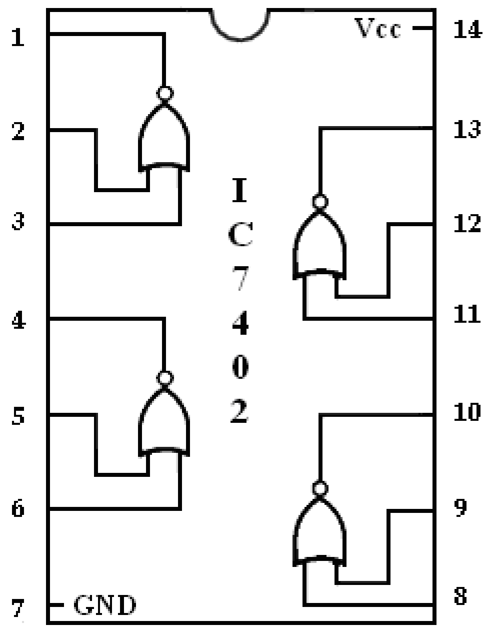
SYMBOL



TRUTH TABLE

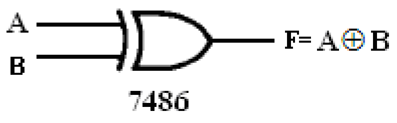
A	B	F= $\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

PIN DIAGRAM



EX-OR gate:

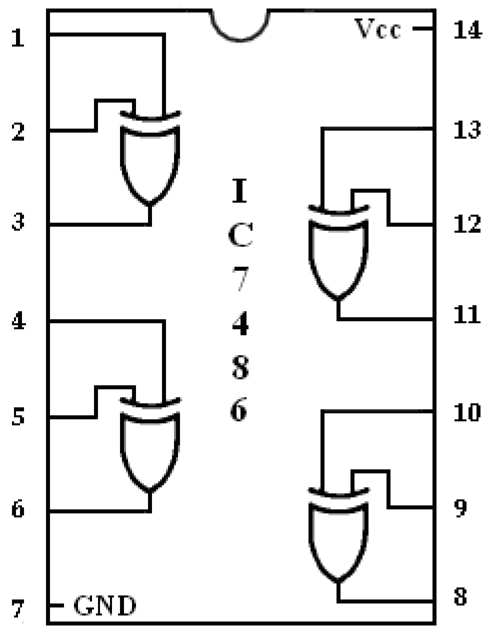
SYMBOL



TRUTH TABLE

A	B	F= A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

PIN DIAGRAM



RESULT:

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

[illegible]

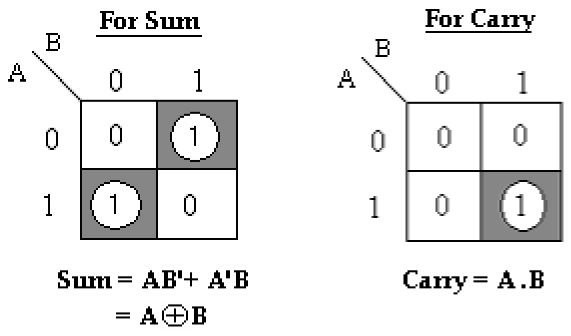
HOD / ECE

HALF ADDER:

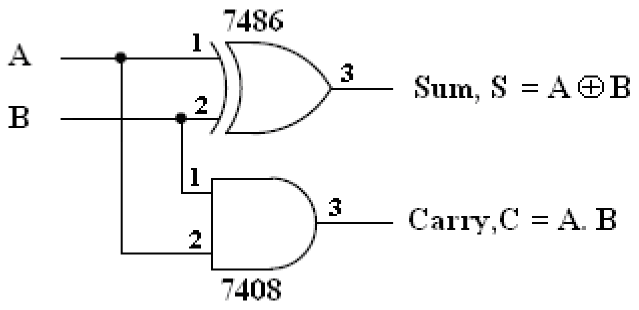
TRUTH TABLE:

Inputs		Outputs	
A	**B**	**Carry (C)**	**Sum (S)**
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

K- MAP SIMPLIFICATION:



LOGIC DIAGRAM:



FULL ADDER:

TRUTH TABLE:

Inputs			Outputs	
A	**B**	**C_{in}**	**Sum (S)**	**Carry (C_{out})**
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

[illegible]

HOD / ECE

EX.NO: 1

DATE :

DESIGN OF ADDER & SUBTRACTOR

AIM:

To design and construct a half adder, full adder, half subtractor and full subtractor circuits and verify their truth table using logic gates.

APPARATUS REQUIRED:

SL.NO	COMPONENTS	SPECIFICATION	QUANTITY
1.	IC Trainer kit	-	1
2.	EX-OR gate	IC 7486	1
3.	NOT gate	IC 7404	1
4.	OR gate	IC 7432	1
5.	AND gate	IC7408	1
6.	Patch cords	-	Few

THEORY:

Half Adder:

A half-adder is a combinational circuit that can be used to add two binary bits. It has two inputs that represent the two bits to be added and two outputs, with one producing the SUM output and the other producing the CARRY. The Sum can be applied using EX-OR gate, carry output can be applied using an AND gate.

Full Adder:

A full adder is a combinational circuit that forms the arithmetic sum of three input bits. It consists of 3 inputs and 2 outputs. Two of the input variables, represent the significant bits to be added. The third input represents the carry from previous lower significant position.

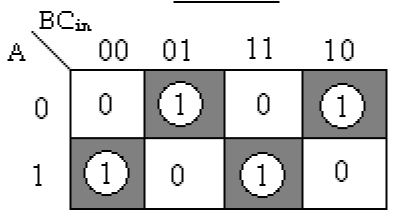
The logic diagram of the full adder can also be implemented with two half-adders and one OR gate. The S output from the second half adder is the exclusive-OR of C_{in} and the output of the first half-adder

[illegible]

HOD / ECE

K-MAP SIMPLIFICATION:

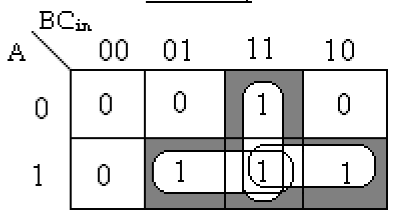
For Sum



Sum, $S = A'B'C_{in} + A'BC'_{in} + AB'C'_{in} + ABC_{in}$

$= C_{in} \oplus (A \oplus B)$

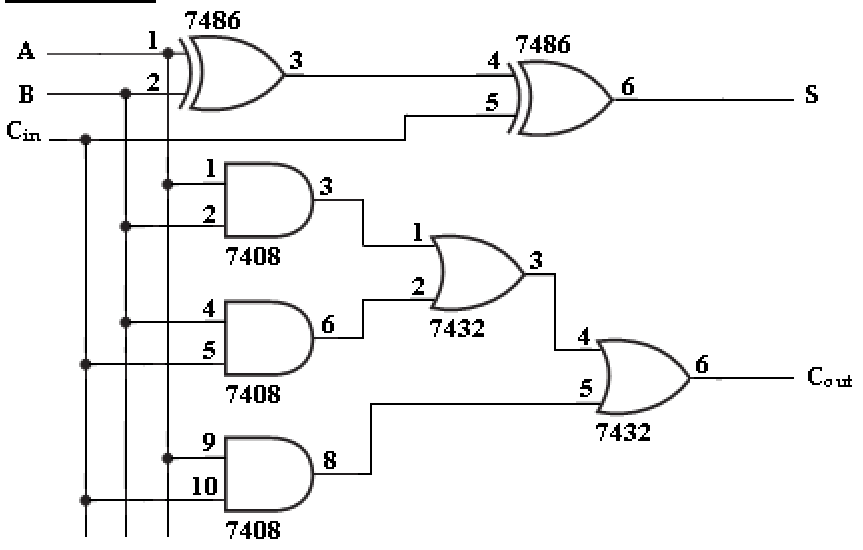
For Carry



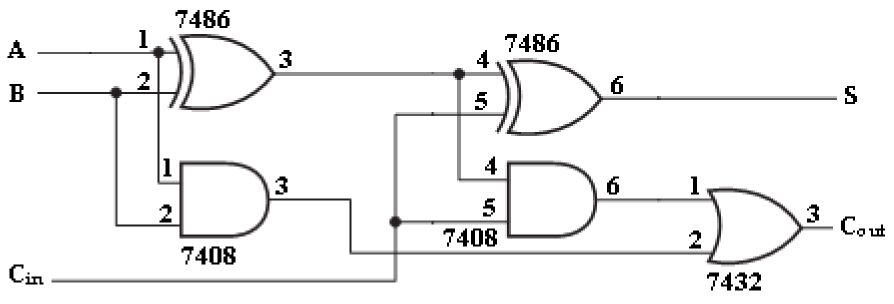
Carry, $C_{out} = AB + AC_{in} + BC_{in}$

LOGIC DIAGRAM:

Full Adder:



Full Adder using Two Half Adders:



[illegible]

HOD / ECE

Half Subtractor:

A half-subtractor is a combinational circuit that can be used to subtract one binary digit from another to produce a DIFFERENCE output and a BORROW output. The BORROW output here specifies whether a ‘1’ has been borrowed to perform the subtraction. The difference can be applied using EX-OR gate, borrow output can be applied using an AND gate and an inverter.

Full Subtractor:

A full subtractor performs subtraction operation on two bits, a minuend and a subtrahend, and also takes into consideration whether a ‘1’ has already been borrowed by the previous adjacent lower minuend bit or not.

As a result, there are three bits to be handled at the input of a full subtractor, namely the two bits to be subtracted and a borrow bit designated as B_{in}. There are two outputs, namely the DIFFERENCE output D and the BORROW output B_o. The BORROW output bit tells whether the minuend bit needs to borrow a ‘1’ from the next possible higher minuend bit.

<u>PROCEDURE</u>:

1. Connections are given as per the logic diagram.
2. Logic inputs are given as per the truth table.
3. Observe the logic output and verify with their truth tables.

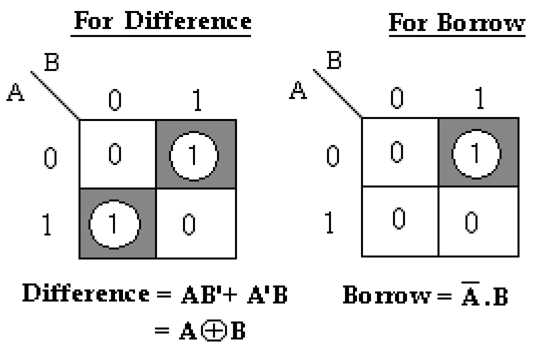
<u>HALF SUBTRACTOR</u>:

[illegible]

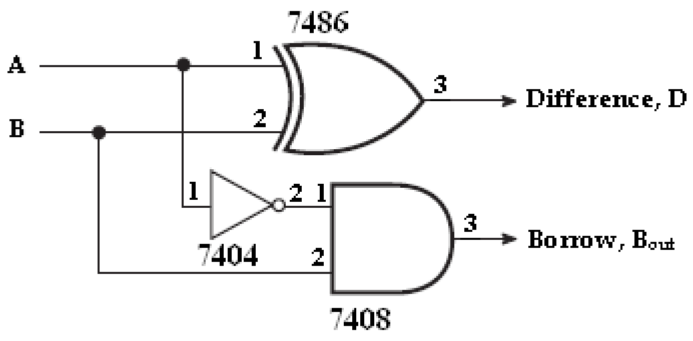
TRUTH TABLE:

Input		Output	
A	**B**	**Difference (D)**	**Borrow (B_{out})**
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

K- MAP SIMPLIFICATION:



LOGIC DIAGRAM:



FULL SUBTRACTOR:

TRUTH TABLE:

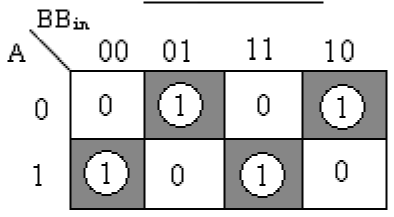
Inputs			Outputs	
A	**B**	**B_{in}**	**Difference(D)**	**Borrow(B_{out})**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K- MAP SIMPLIFICATION:

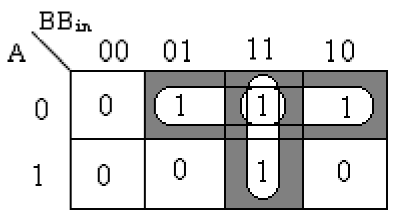
[illegible]

HOD / ECE

For Difference



For Borrow



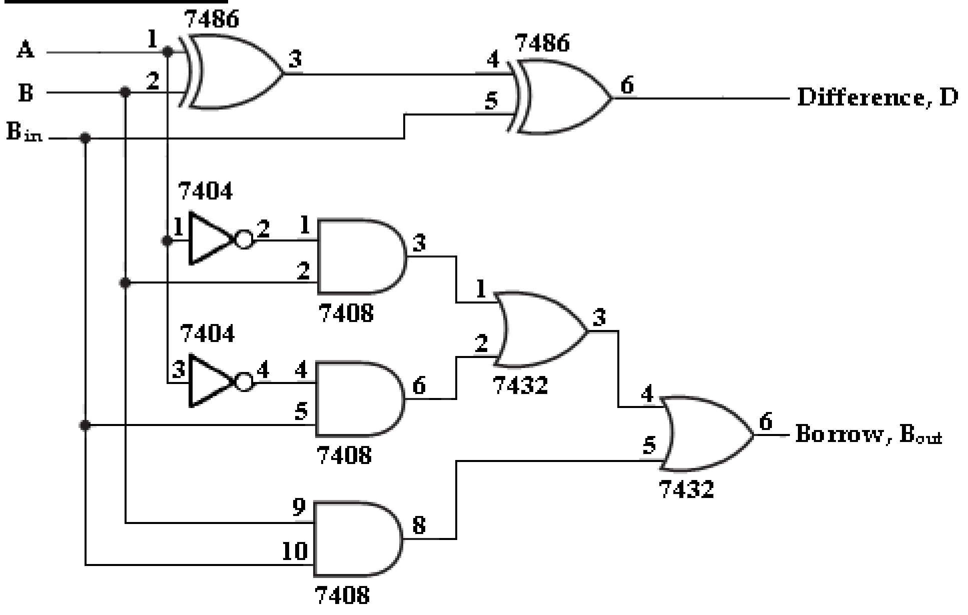
Difference, D $= A'B'B_{in} + A'BB'_{in} + AB'B'_{in} + ABB_{in}$

$= B_{in} \oplus (A \oplus B)$

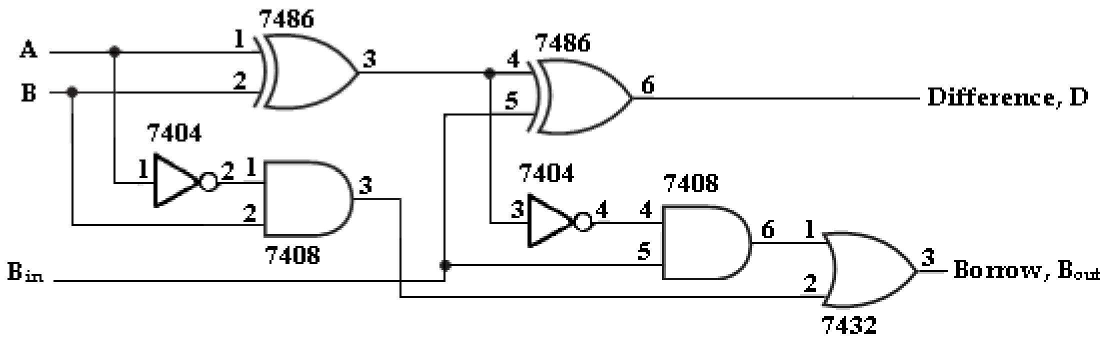
Borrow, $B_{out} = A'B + A'B_{in} + BB_{in}$

LOGIC DIAGRAM:

Full Subtractor:



Full Subtractor with Two Half Subtractors:



RESULT:

Thus half adder, full adder, half subtractor and full subtractor circuits was designed using logic gates and their truth tables were verified.

BINARY TO GRAY CODE CONVERTER:

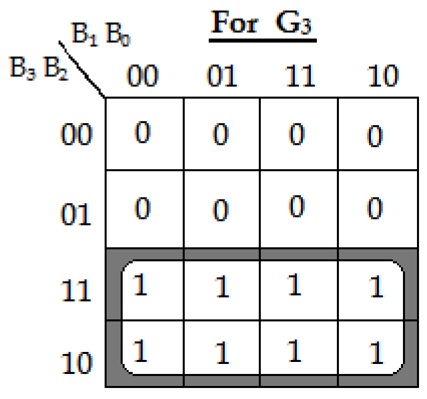
[illegible]

HOD / ECE

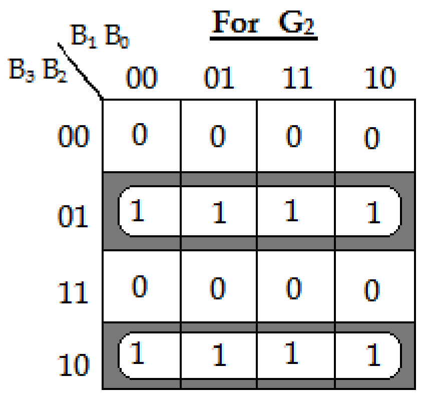
TRUTH TABLE:

Binary code				Gray code			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
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

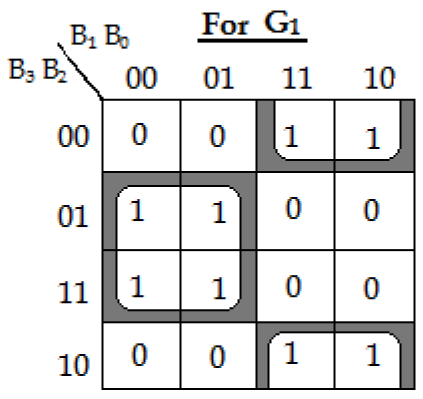
K- Map Simplification:



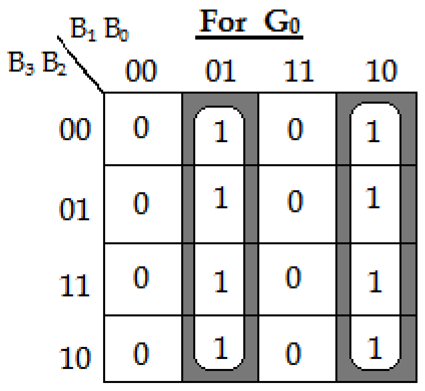
$G_3 = B_3$



$G_2 = B_3'B_2 + B_3B_2'$
$= B_3 \oplus B_2$



$G_1 = B_2'B_1 + B_2B_1'$
$= B_2 \oplus B_1$



$G_0 = B_1'B_0 + B_1B_0'$
$= B_1 \oplus B_0$

[illegible]

HOD / ECE

EX.NO: 2

DATE :

DESIGN AND IMPLEMENTATION OF CODE CONVERTERS

AIM:

To design and implement 4-bit

1. Binary to Gray code Converter
2. Gray to Binary code Converter
3. BCD to Excess-3 code Converter
4. Excess-3 code to BCD Converter

APPARATUS REQUIRED:

SL.NO	COMPONENTS	SPECIFICATION	QUANTITY
1.	IC Trainer kit	-	1
2.	EX-OR gate	IC7486	1
3.	NOT gate	IC7404	1
4.	OR gate	IC7432	1
5.	2-Input AND gate	IC7408	1
6.	3-Input AND gate	IC7411	1
7.	Patch cords	-	As Required

THEORY:

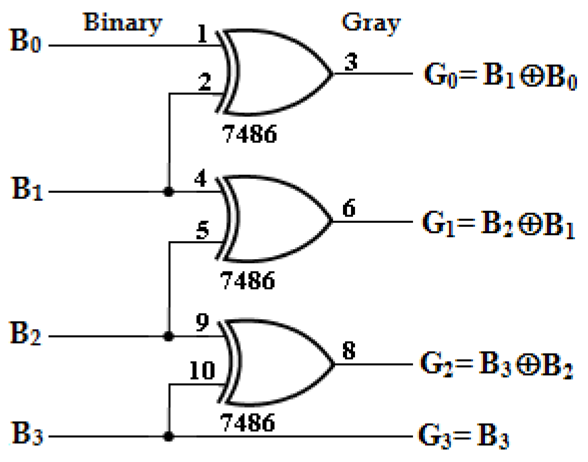
An availability of large variety of codes for the same discrete elements of information results in the use of different codes by different systems. A conversion circuit must be inserted between the two systems if each uses different codes for the same information. Thus, code converter is a circuit that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

The input variable are designed as B_3,B_2,B_1,B_0 and the output variables are designed as G_3,G_2,G_1,G_0. From the truth table, combinational circuit is designed. The Boolean functions are obtained from K-Map for each output variable.

[illegible]

HOD / ECE

Logic Diagram:



GRAY TO BINARY CODE CONVERTER:

TRUTH TABLE:

Gray code				Binary code			
G_3	G_2	G_1	G_0	B_3	B_2	B_1	B_0
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	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	0	1	0	1	1
1	1	1	1	1	0	1	0

[illegible]

HOD / ECE

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.

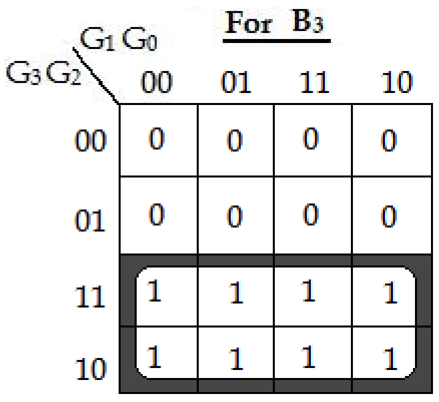
PROCEDURE:

1. Connections are given as per the logic diagram.
2. Logic inputs are given as per the truth table.
3. Observe the logic output and verify with the truth tables.

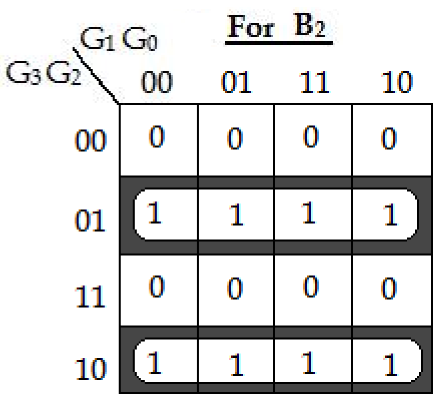
[illegible]

HOD / ECE

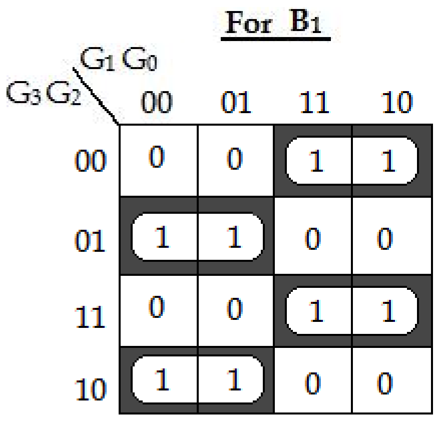
K-Map Simplification:



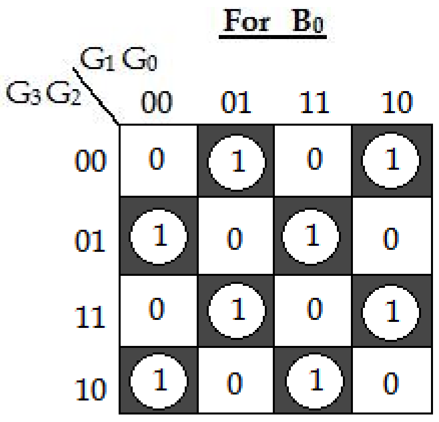
$B_3 = G_3$



$B_2 = G_3'G_2 + G_3G_2'$
$= G_3 \oplus G_2$

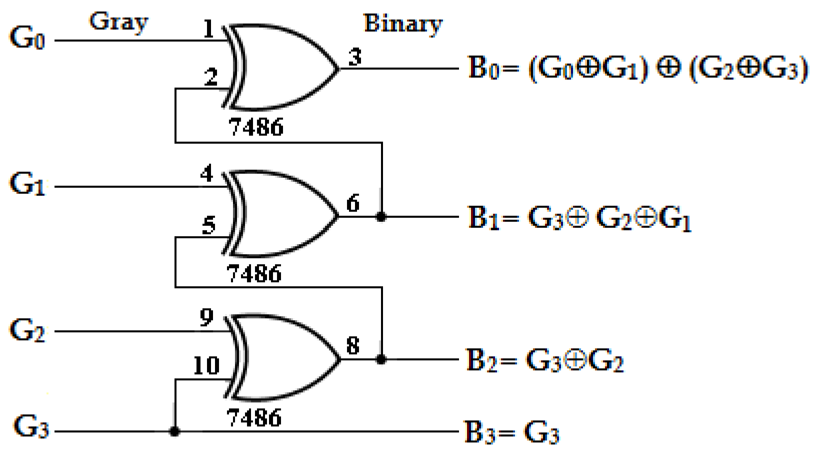


$B_1 = G_3 \oplus G_2 \oplus G_1$



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

Logic Diagram:



[illegible]

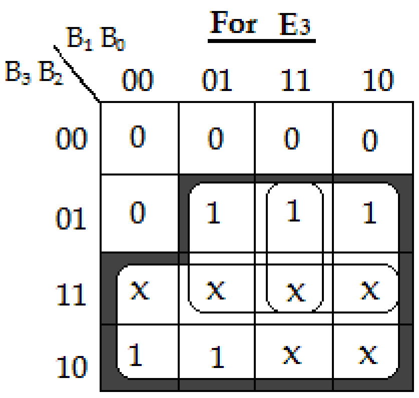
HOD / ECE

BCD TO EXCESS-3 CODE:

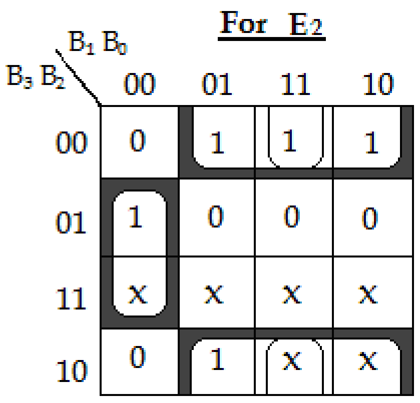
Truth table:

BCD code				Excess-3 code			
B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
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

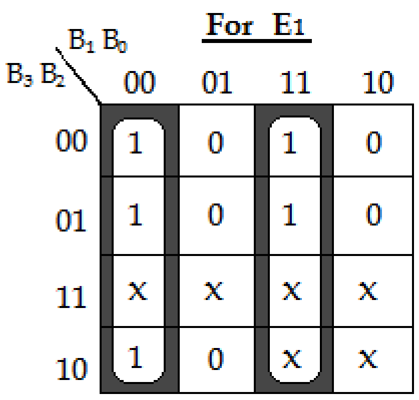
K-Map Simplification:



$E_3 = B_3 + B_2\,(B_0 + B_1)$

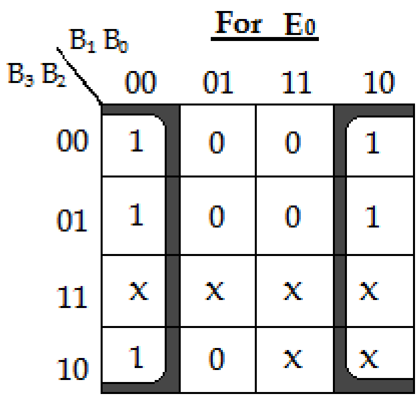


$E_2 = B_2B_1'B_0' + B_2'\,(B_0 + B_1)$



$E_1 = B_1'B_0' + B_1B_0$

$= B_1 \odot B_0$

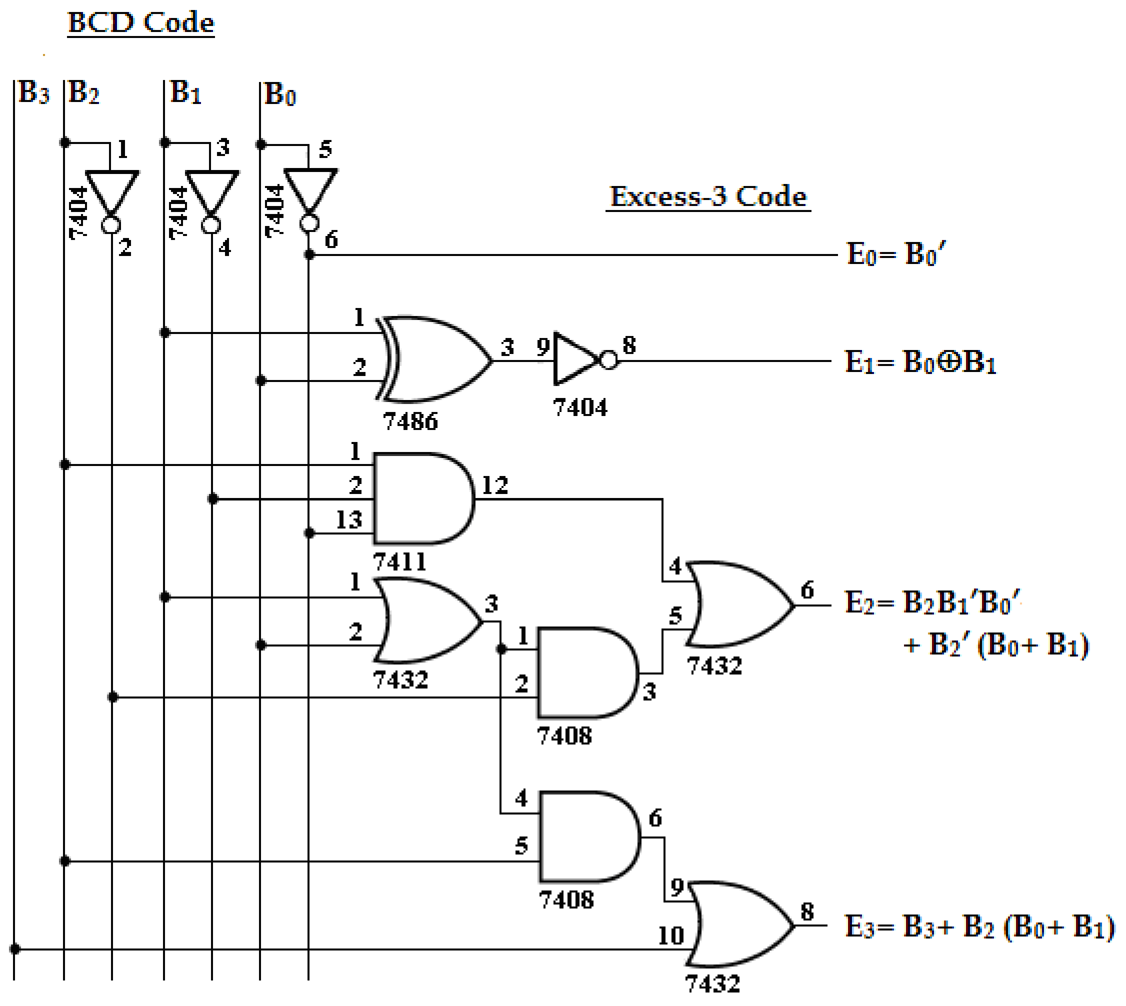


$E_0 = B_0'$

[illegible]

HOD / ECE

Logic Diagram:



[illegible]

HOD / ECE

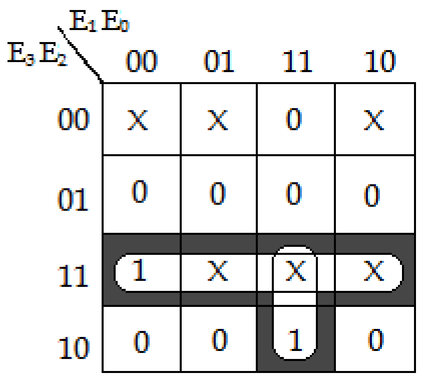
EXCESS-3 TO BCD CONVERTER:

Truth Table:

Excess-3 code				BCD code			
E_3	E_2	E_1	E_0	B_3	B_2	B_1	B_0
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

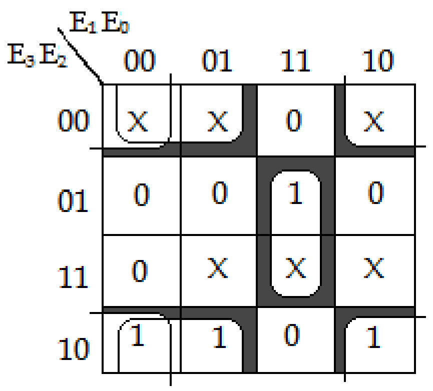
K-Map Simplification:

For B_3



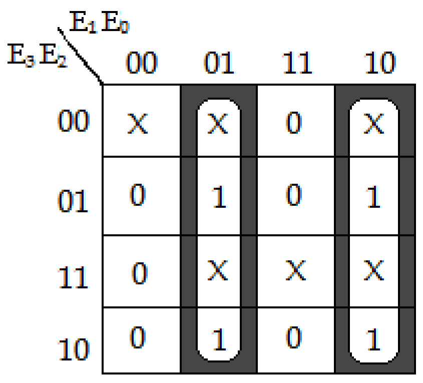
$B_3 = E_3E_2 + E_3E_1E_0$

For B_2



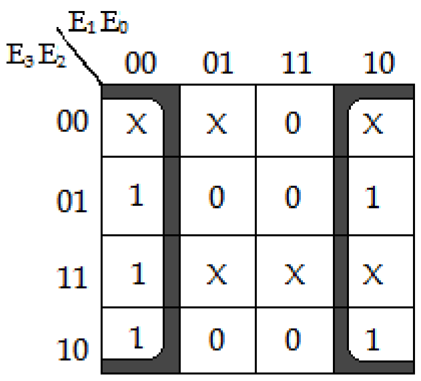
$B_2 = E_2'E_1' + E_2E_1E_0 + E_2'E_0'$

For B_1



$B_1 = E_1'E_0 + E_1E_0'$
$= E_1 \oplus E_0$

For B_0



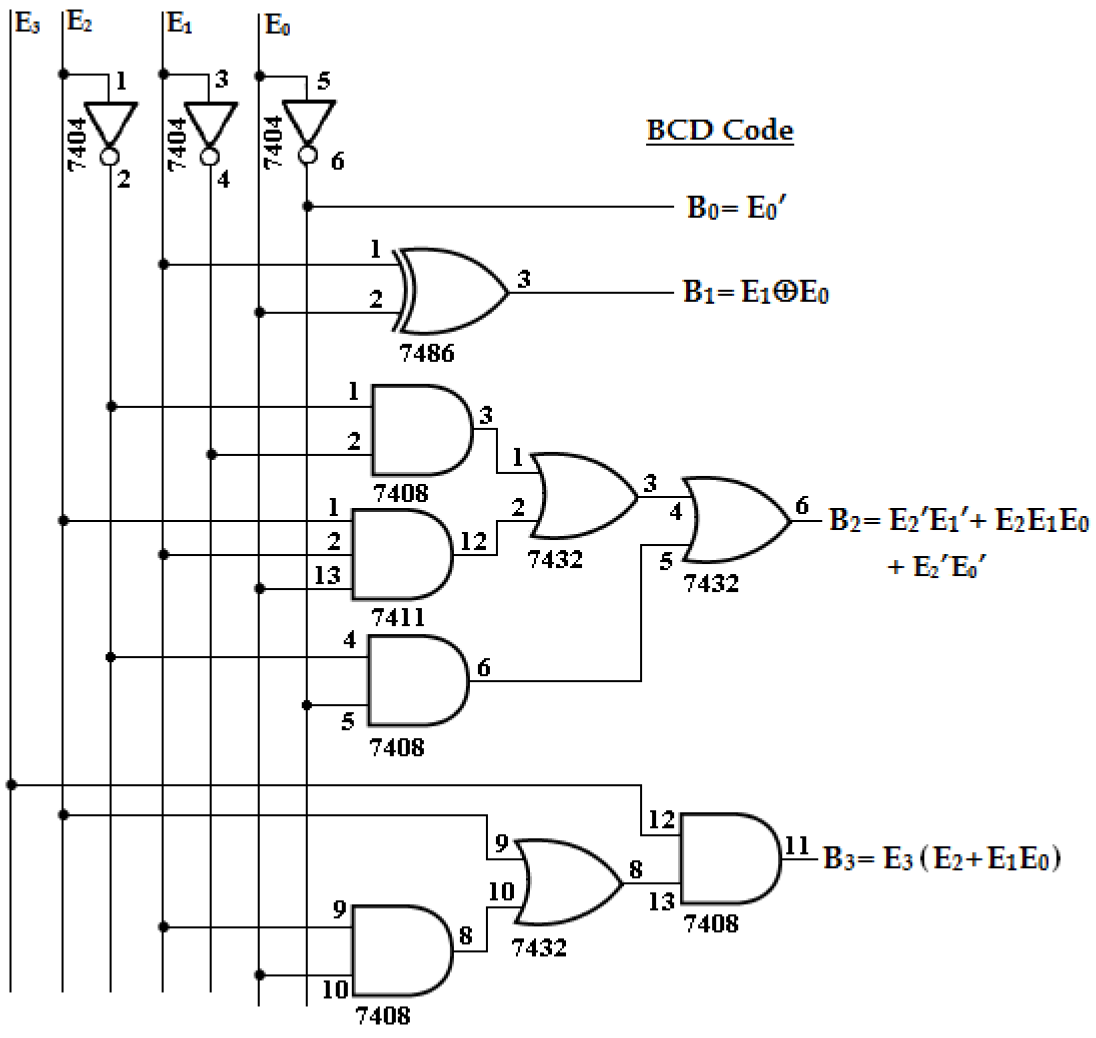
$B_0 = E_0'$

[illegible]

HOD / ECE

Logic Diagram:

Excess-3 Code



RESULT:

Thus the 4-bit

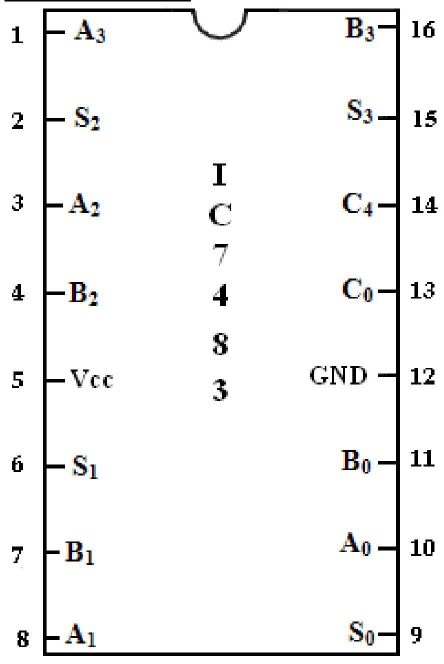
1. Binary to Gray code Converter
2. Gray to Binary code Converter
3. BCD to Excess-3 code Converter
4. Excess-3 code to BCD Converter was designed and implemented.

[illegible]

HOD / ECE

4- BIT BINARY ADDER/ SUBTRACTOR:

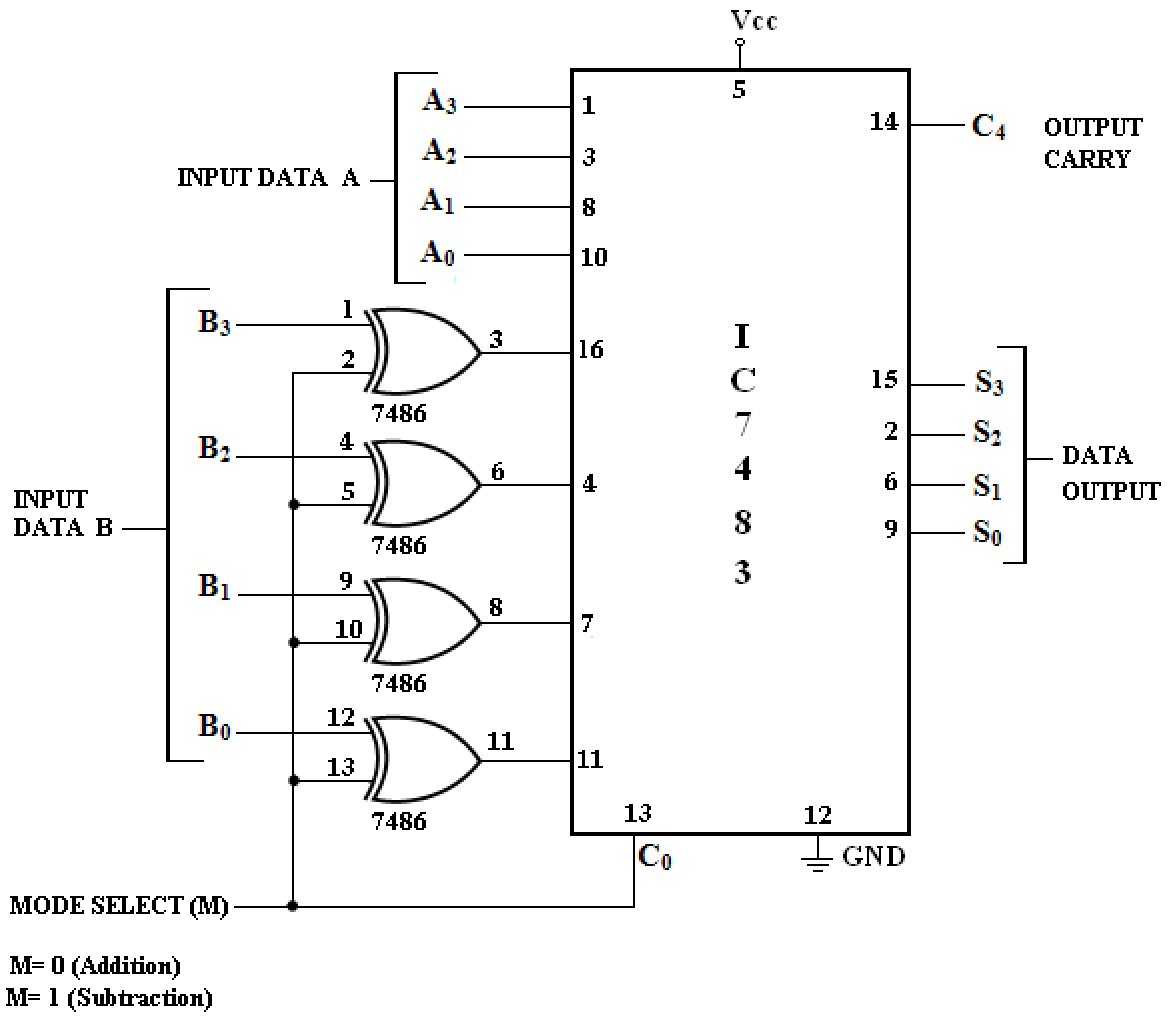
PIN DIAGRAM:



LOGIC DIAGRAM:

[illegible]

HOD / ECE



EX.NO: 3

DATE :

DESIGN OF 4-BIT ADDER/ SUBTRACTOR & BCD ADDER

AIM:

To Design and implement the 4-bit adder/ subtractor and BCD adder using IC 7483.

APPARATUS REQUIRED:

SL.NO	COMPONENT	SPECIFICATION	QUANTITY

[illegible]

HOD / ECE

1.	IC Trainer kit	-	1
2.	4-bit binary full adder	IC 7483	2
3.	EX-OR gate	IC 7486	1
4.	AND gate	IC 7408	1
5.	OR gate	IC 7432	1
6.	Patch cords	-	Few

THEORY:

4-Bit binary adder/ subtractor:

The 4-bit binary adder/ subtractor circuit performs the operation of both addition and subtraction. It has two 4-bit inputs A_0, A_1, A_2, A_3 and B_0, B_1, B_2, B_3. The mode input M controls the operation of the circuit. When M= 0, the circuit is an adder and when M=1, the circuit becomes a Subtractor. Each exclusive-OR gate receives input M and one of the inputs of B.

When M=0, the operation is $B\oplus 0= B$. The full adders receive the value of B and the input carry is 0, and the circuit performs the addition operation, **A+ B**.

When M=1, the operation is $B\oplus 1= B'$ and C_0=1. The B inputs are all complemented and a 1 is added through the input carry. Thus the circuit performs the subtraction operation, i.e., A+ (2's complement of B) = **A- B**.

TRUTH TABLE:

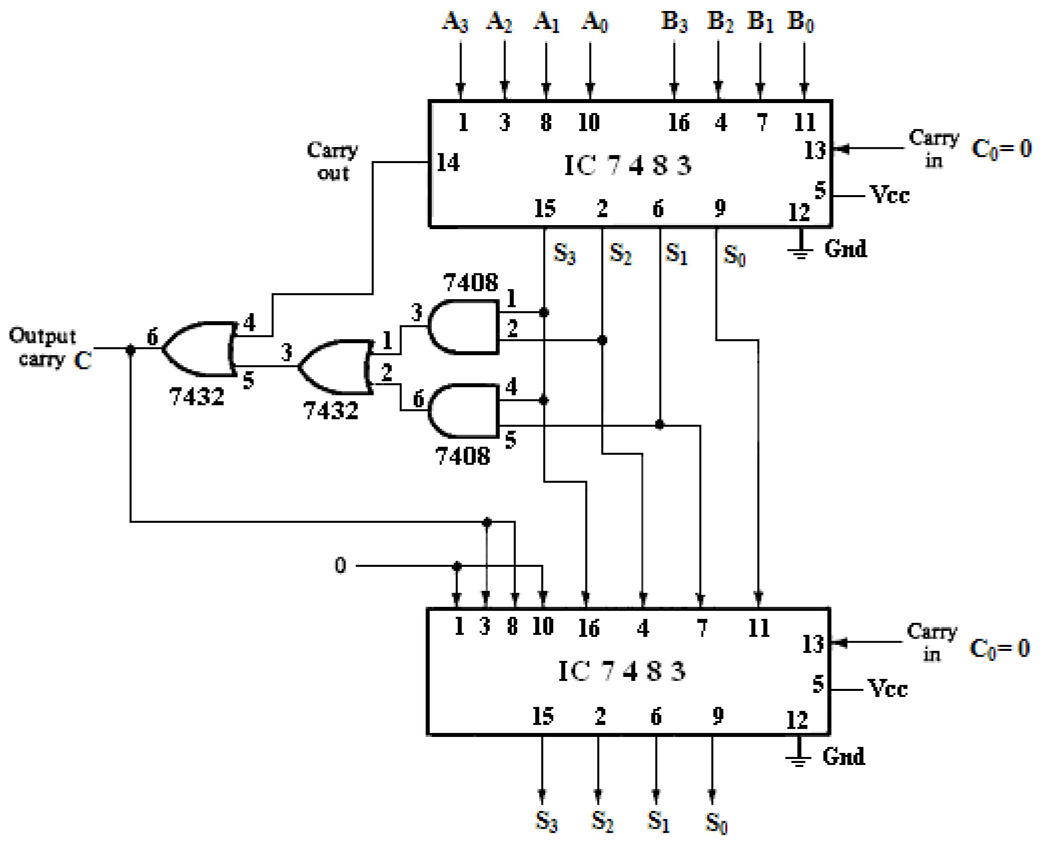
Input data A				Input data B				Addition					Subtraction				
A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0	C	S_3	S_2	S_1	S_0	B	D_3	D_2	D_1	D_0
1	0	0	0	0	0	1	0	0	1	0	1	0	1	0	1	1	0
1	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0
0	0	1	0	1	0	0	0	0	1	0	1	0	0	1	0	1	0
0	0	0	1	0	1	1	1	0	1	0	0	0	0	1	0	1	0
1	0	1	0	1	0	1	1	1	0	1	0	1	0	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	0	1	0	1	1	1	1
1	0	1	0	1	1	0	1	1	0	1	1	1	0	1	1	0	1

BCD ADDER:

[illegible]

HOD / ECE

LOGIC DIAGRAM:



4- Bit BCD Adder:

The digital system handles the decimal number in the form of binary coded decimal numbers (BCD). A BCD adder is a circuit that adds two BCD bits and produces a sum digit also in BCD.

In examining the contents of the table, it is apparent that when the binary sum is equal to or less than $(1001)_2$, the corresponding BCD number is identical, and therefore no conversion is needed. When the binary sum is greater than 9 $(1001)_2$, we obtain a non-valid BCD representation. The addition of binary 6 $(0110)_2$ to the binary sum converts it to the correct BCD representation and also produces an output carry as required.

The logic circuit to detect sum greater than 9 can be determined by simplifying the Boolean expression of the given truth table.

[illegible]

HOD / ECE

The two decimal digits, together with the input carry, are first added in the top 4-bit binary adder to provide the binary sum. When the output carry is equal to zero, nothing is added to the binary sum. When it is equal to one, binary $(0110)_2$ is added to the binary sum through the bottom 4-bit adder. The output carry generated from the bottom adder can be ignored, since it supplies information already available at the output carry terminal. The output carry from one stage must be connected to the input carry of the next higher-order stage.

PROCEDURE:

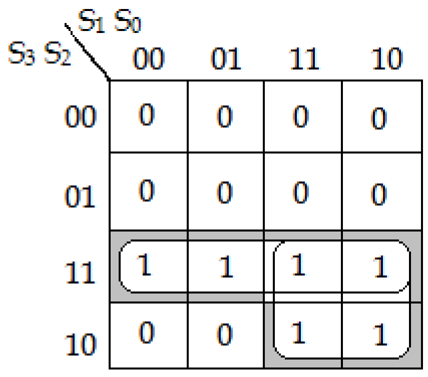
1. Connections are given as per the logic diagram.
2. Logic inputs are given as per the truth table.
3. Observe the logic output and verify with the truth tables.

Truth table:

[illegible]

HOD / ECE

Inputs				Output
S_3	S_2	S_1	S_0	Y
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



$Y = S_3S_2 + S_3S_1$

<u>RESULT:</u>

Thus the 4-bit adder/ subtractor and BCD adder using IC 7483 was designed and implemented.

<u>2-BIT MAGNITUDE COMPARATOR</u>:

[illegible]

HOD / ECE

TRUTH TABLE:

Inputs				Outputs		
A_1	A_0	B_1	B_0	A>B	A=B	A<B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

DESIGN:

$A = A_1A_0$
$B = B_1\,B_0$
$X_i = A_iB_i + A_i'B_i'$
$X_i = (A_i \oplus B_i)'$ *for* $i = 0,1$
$\mathbf{(A = B) = X_1\,X_0}$
$\mathbf{(A>B) = A_1B_1' + X_1A_0B_0'}$
$\mathbf{(A<B) = A_1'B_1 + X_1A_0'B_0}$

[illegible]

HOD / ECE

EX.NO: 4

DATE :

DESIGN AND IMPLEMENTATION OF MAGNITUDE COMPARATOR

AIM:

To design and implement

(i) 2-bit magnitude comparator using logic gates.

(ii) 8-bit magnitude comparator using IC 7485.

APPARATUS REQUIRED:

SL.NO	COMPONENT	SPECIFICATION	QUANTITY
1.	IC Trainer kit	-	
2.	EX-OR gate	IC7486	1
3.	NOT gate	IC7404	1
4.	OR gate	IC7432	1
5.	AND gate	IC7408	1
6.	4-bit Magnitude Comparator	IC 7485	2
7.	Connecting Wires	-	Few

THEORY:

A ***magnitude comparator*** is a combinational circuit that compares two given numbers (A and B) and determines whether one is equal to, less than or greater than the other. The output is in the form of three binary variables representing the conditions A = B, A>B and A<B, if A and B are the two numbers being compared.

The two binary numbers A and B with two digits each, written in descending order as,

$\mathbf{A = A_1A_0}$

$\mathbf{B = B_1\,B_0}$

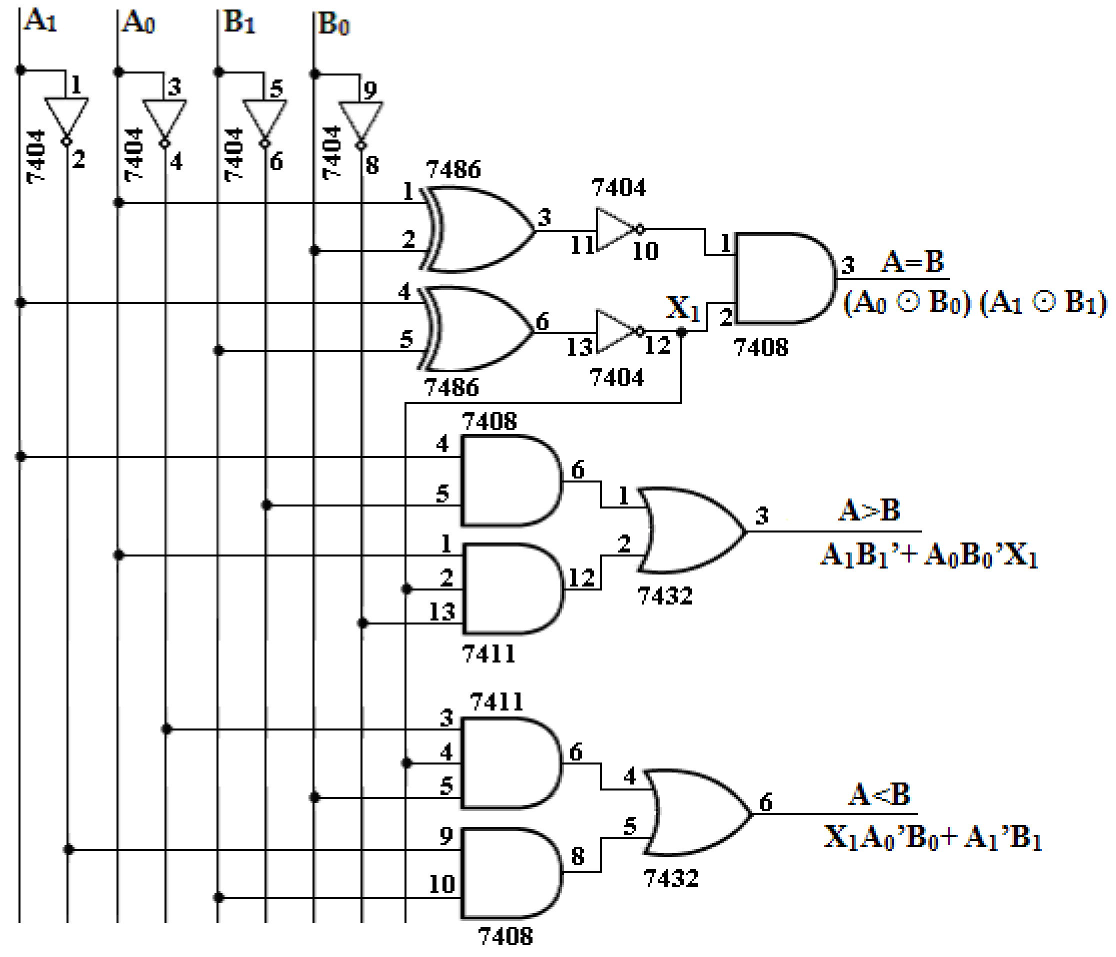
Each subscripted letter represents one of the digits in the number. It is observed from the bit contents of two numbers that A = B, when $A_1 = B_1$ and $A_0 = B_0$. When the numbers are binary they possess the value of either 1 or 0, the equality relation of each pair can be expressed logically by the equivalence function as,

[illegible]

HOD / ECE

LOGIC DIAGRAM:

2-bit Magnitude Comparator:



[illegible]

HOD / ECE

$\mathbf{X_i = A_iB_i + A_i'B_i'}$ for $i = 1, 2, 3, 4$.

Or, $\mathbf{X_i = (A \oplus B)'}$ or, $\mathbf{X_i' = A \oplus B}$

Or, $\mathbf{X_i = (A_iB_i' + A_i'B_i)'}$

where,

$X_i = 1$ only if the pair of bits in position i are equal

To satisfy the equality condition of two numbers A and B, it is necessary that all X_i must be equal to logic 1. This indicates the AND operation of all X_i variables. In other words, we can write the Boolean expression for two equal 2-bit numbers.

$$\mathbf{(A = B) = X_1 X_0.}$$

The binary variable (A=B) is equal to 1 only if all pairs of digits of the two numbers are equal.

To determine if A is greater than or less than B, we inspect the relative magnitudes of pairs of significant bits starting from the most significant bit. If the two digits of the most significant position are equal, the next significant pair of digits is compared. The comparison process is continued until a pair of unequal digits is found. It may be concluded that A>B, if the corresponding digit of A is 1 and B is 0. If the corresponding digit of A is 0 and B is 1, we conclude that A<B. Therefore, we can derive the logical expression of such sequential comparison by the following two Boolean functions,

$$\mathbf{(A>B) = A_1B_1' + X_1A_0B_0'}$$

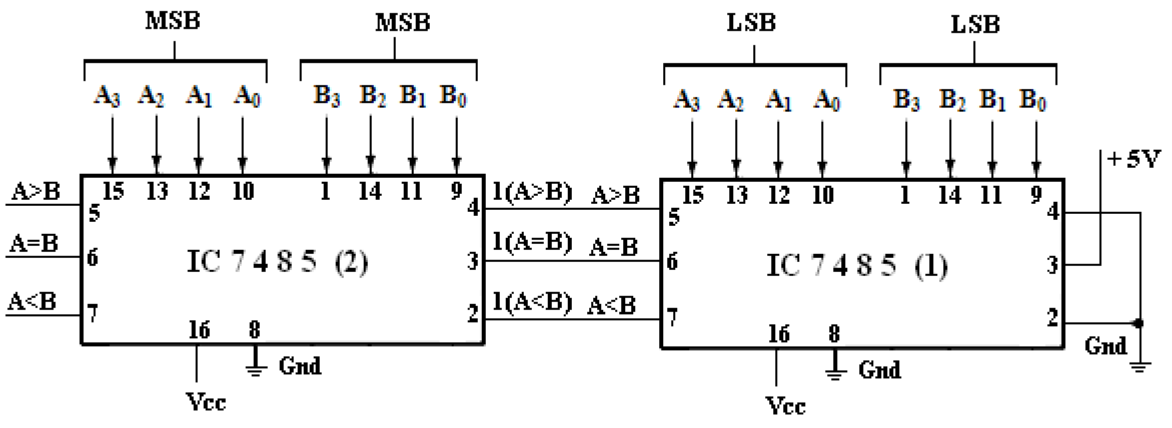
$$\mathbf{(A<B) = A_1'B_1 + X_1A_0'B_0}$$

The symbols (A>B) and (A<B) are binary output variables that are equal to 1 when A>B or A<B, respectively.

<u>8- BIT MAGNITUDE COMPARATOR:</u>

[illegible]

HOD / ECE



Truth Table:

A	B	A>B	A=B	A<B
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0	1	0
0 0 0 1 0 0 0 1	0 0 0 0 0 0 0 0	1	0	0
0 0 0 0 0 0 0 0	0 0 0 1 0 0 0 1	0	0	1

PROCEDURE:

1. Connections are given as per the logic diagram.
2. Logic inputs are given as per the truth table.
3. Observe the logic output and verify with the truth tables.

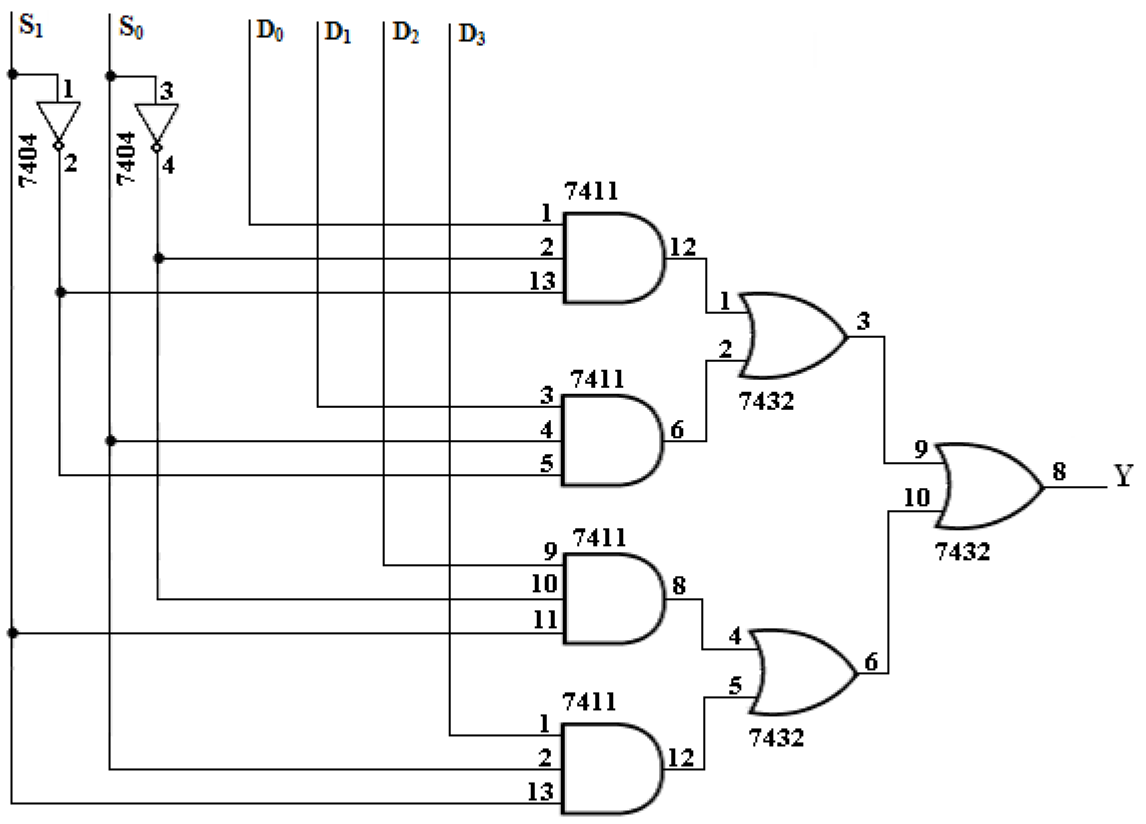
RESULT:

Thus the 2-bit magnitude comparator was designed and implemented using logic gates and 8-bit magnitude comparator using IC 7485.

4:1 MULTIPLEXER:

[illegible]

HOD / ECE



Truth table:

S_1	S_0	OUTPUTS, (Y)
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

[illegible]

HOD / ECE

EX.NO: 5

DATE :

DESIGN AND IMPLEMENTATION OF MULTIPLEXER AND DEMULTIPLEXER

AIM:

To design and implement multiplexer and demultiplexer using logic gates.

APPARATUS REQUIRED:

SL.NO	COMPONENT	SPECIFICATION	QUANTITY
1.	IC Trainer kit	-	1
2.	3-I/P AND GATE	IC7411	2
3.	NOT GATE	IC7404	1
4.	OR GATE	IC7432	1
5.	Patch cords	-	Few

THEORY:

Multiplexer:

Multiplexer means transmitting a large number of information units over a small number of channels or lines. A digital multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally there are 2^n input line and 'n' selection lines whose bit combination determine which input is selected. It is called as **data selector**, because the output depends on the input data bit that is selected.

Demultiplexer:

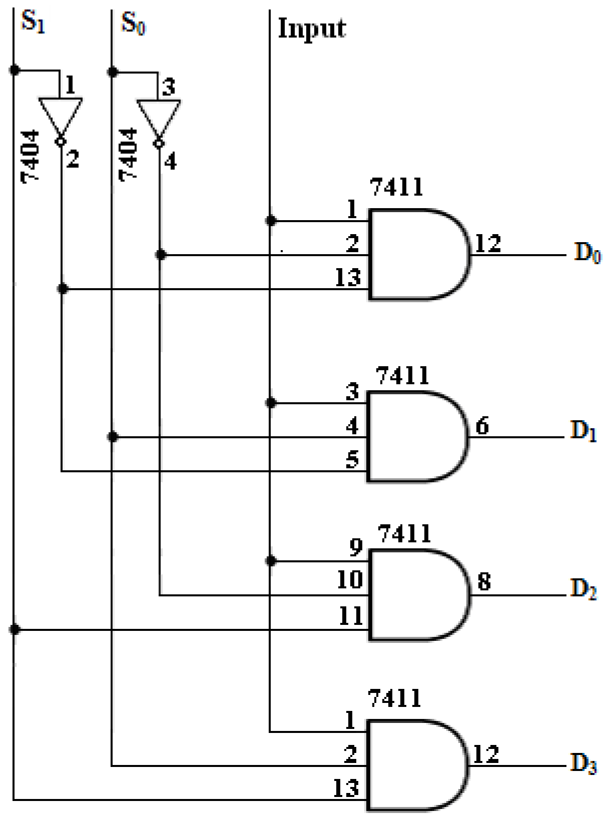
The function of Demultiplexer is in contrast to multiplexer function. It takes information from one line and distributes it to a given number of output lines. For this reason, the demultiplexer is also known as a **data distributor**. Decoder can also be used as Demultiplexer.

In the 1:4 demultiplexer circuit, the data input line goes to all of the AND gates. The data select lines enable only one gate at a time and the data on the data input line will pass through the selected gate to the associated data output line.

[illegible]

HOD / ECE

1: 4 DEMULTIPLEXER:



Truth Table:

INPUT			OUTPUT			
S_1	S_0	I/P	D_0	D_1	D_2	D_3
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	1	0
1	1	0	0	0	0	0
1	1	1	0	0	0	1

[illegible]

HOD / ECE

PROCEDURE:

1. Connections are given as per the logic diagram.
2. Logic inputs are given as per the truth table.
3. Observe the logic output and verify with the truth tables.

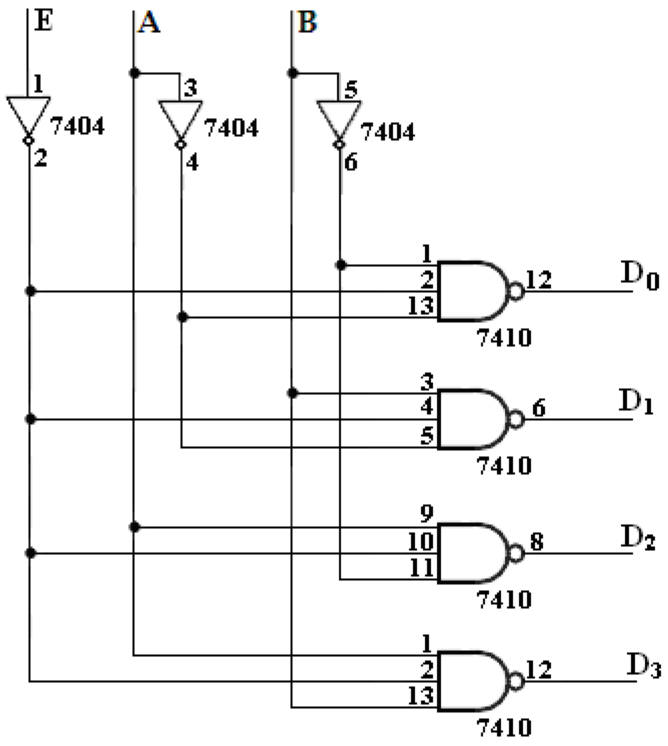
RESULT:

Thus the multiplexer and demultiplexer was designed and implemented using logic gates.

[illegible]

HOD / ECE

Logic Diagram (2-to-4- Line Decoder with Enable Input):



Truth Table:

INPUTS			OUTPUTS			
E	**A**	**B**	**D_0**	**D_1**	**D_2**	**D_3**
1	x	x	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

[illegible]

HOD / ECE

EX.NO: 6

DATE :

DESIGN AND IMPLEMENTATION OF ENCODER AND DECODER

AIM:

To design and implementation encoder and decoder using logic gates.

APPARATUS REQUIRED:

SL.NO	COMPONENT	SPECIFICATION	QUANTITY
1.	IC Trainer kit	-	1
2.	3-I/P NAND gate	IC7410	2
3.	NOT gate	IC7404	1
4.	OR gate	IC7432	3
5.	Patch cords	-	Few

THEORY:

Encoder:

An encoder is a digital circuit that performs inverse operation of a decoder. An encoder has 2^n input lines and 'n' output lines. In encoder the output lines generates the binary code corresponding to the input value. In octal to binary encoder it has eight inputs, one for each octal digit and three output that generates the corresponding binary code. In encoder it is assumed that only one input has a value of one at any given time otherwise the circuit is meaningless. It has an ambiguila that when all inputs are zero the outputs are zero. The zero outputs can also be generated when D0=1.

Decoder:

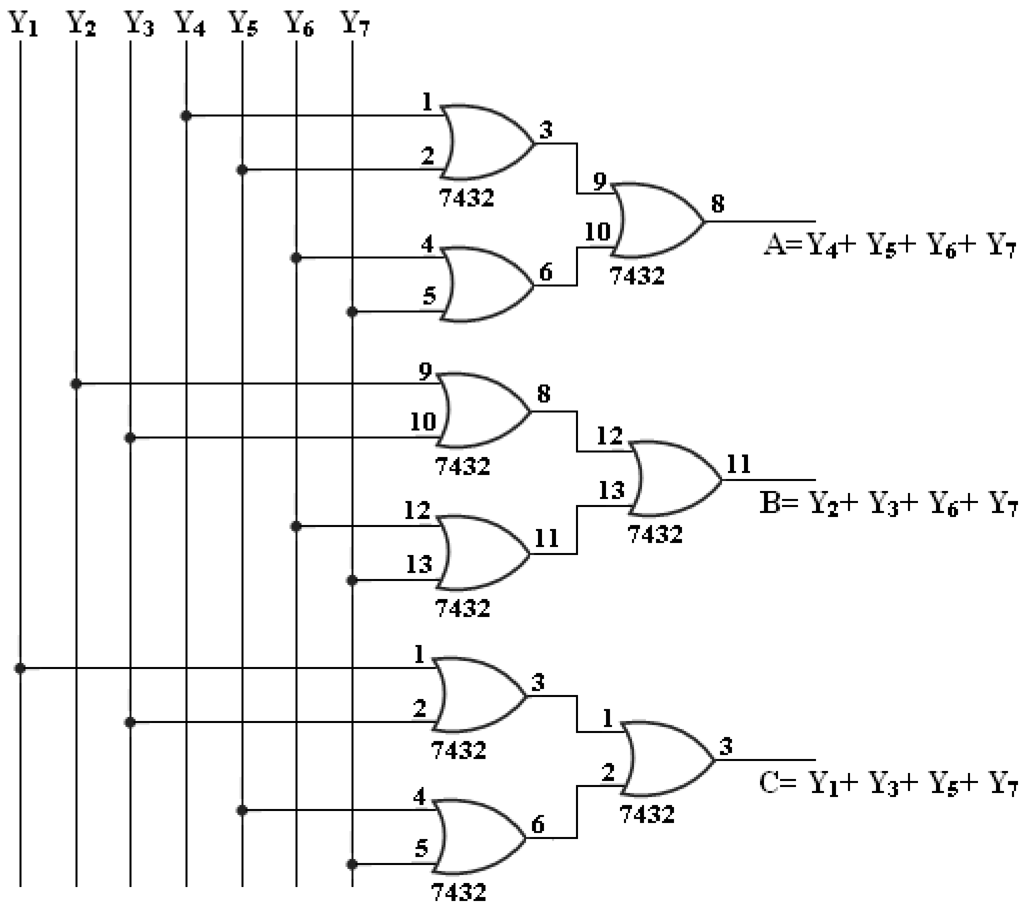
A decoder is a multiple output logic circuit which converts input into coded output where input and output codes are different. The input code generally has few bits than the output code. Each input code word produces a different output code word i.e., there is one to one mapping can be expressed in truth table. In block diagram of decoder circuit the encoded information is present as n input producing 2^n possible outputs. The 2^n output values are from 0 through out 2^n-1.

[illegible]

HOD / ECE

ENCODER:

Logic Diagram:



Truth Table:

INPUTS							OUTPUTS		
Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	A	B	C
1	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	1
0	0	0	1	0	0	0	1	0	0
0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	1	1	1	1

[illegible]

HOD / ECE

PROCEDURE:

1. Connections are given as per the logic diagram.
2. Logic inputs are given as per the truth table.
3. Observe the logic output and verify with the truth tables**.**

RESULT:

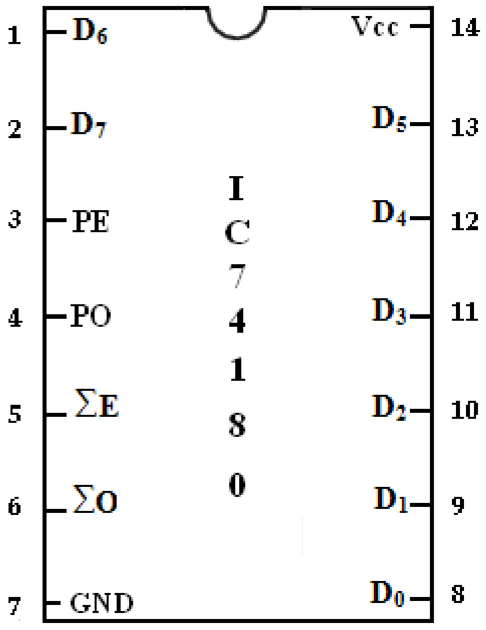
Thus the design and implementation of encoder and decoder using logic gates.

[illegible]

HOD / ECE

8- Bit ODD/EVEN PARITY GENERATOR/ CHECKER:

PIN DIAGRAM:



FUNCTION TABLE:

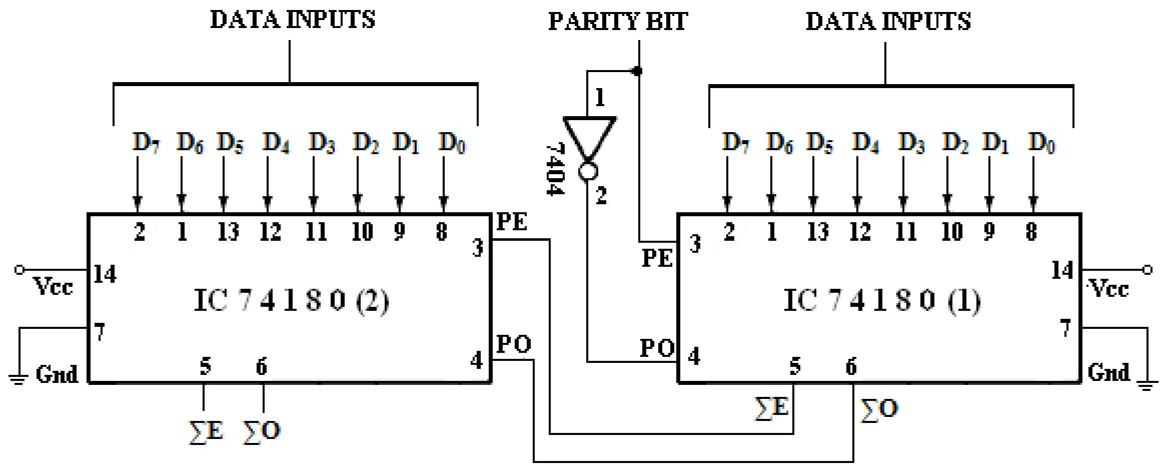
INPUTS			OUTPUTS	
Number of Data Inputs (D_0 - D_7)	**PE**	**PO**	**∑E**	**∑O**
EVEN	1	0	1	0
ODD	1	0	0	1
EVEN	0	1	0	1
ODD	0	1	1	0
X	1	1	0	0
X	0	0	1	1

16- Bit ODD/EVEN PARITY GENERATOR:

LOGIC DIAGRAM:

[illegible]

HOD / ECE



EX.NO: 7

DATE :

DESIGN AND IMPLEMENTATION OF 16-BIT ODD/EVEN PARITY GENERATOR/CHECKER

AIM:

To design and implement 16 bit odd /even parity checker generator using IC 74180.

APPARATUS REQUIRED:

SL.NO	COMPONENTS	SPECIFICATION	QUANTITY
1.	IC Trainer kit	-	1
2.	NOT gate	IC7404	1
3.	8-bit parity generator/ checker	IC74180	2
4.	Patch cords	-	Few

THEORY:

A *Parity* is a very useful tool in information processing in digital computers to indicate any presence of error in binary information. External noise and loss of signal strength causes loss of data bit information while transporting data from one device to other device, located inside the computer or externally. To indicate any occurrence of error, an extra bit is included with the message according to the total number of 1s in a set of data, which is called *parity*.

[illegible]

HOD / ECE

If the extra bit is considered 0 if the total number of 1s is even and 1 for odd quantities of 1s in a set of data, then it is called *even parity*. On the other hand, if the extra bit is 1 for even quantities of 1s and 0 for an odd number of 1s, then it is called *odd parity*.

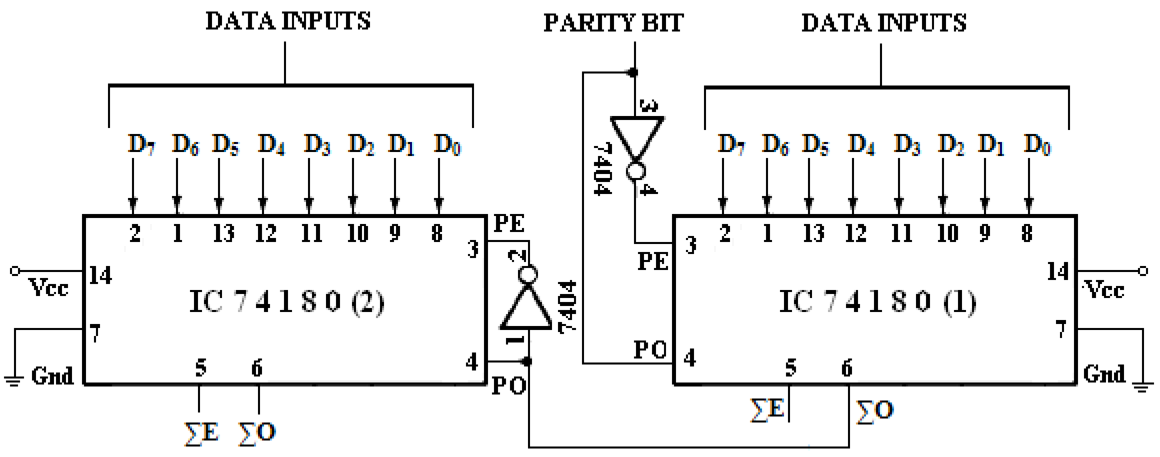
The message including the parity is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity does not correspond with the one transmitted. The circuit that generates the parity bit in the transmitter is called a parity generator and the circuit that checks the parity in the receiver is called a parity checker.

TRUTH TABLE:

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	PE	PO	∑E	∑O
1 1 0 0 0 0 0 0	1 1 0 0 0 0 0 0	1	0	1	0
1 1 0 0 0 0 0 0	1 1 0 0 0 0 0 0	0	1	0	1
1 1 0 0 0 0 0 0	0 1 0 0 0 0 0 0	0	1	1	0

16- Bit ODD/EVEN PARITY CHECKER:

LOGIC DIAGRAM:



TRUTH TABLE:

[illegible]

HOD / ECE

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	**D_7' D_6' D_5' D_4' D_3' D_2' D_1'D_0'**	**PE**	**PO**	**∑E**	**∑O**
0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0	0	1	1	0
0 0 0 0 0 1 1 0	0 0 0 0 0 1 1 0	1	0	1	0
0 0 0 0 0 1 1 0	0 0 0 0 0 1 1 0	0	1	0	1

The parity checker circuit produces a check bit and is very similar to the parity generator circuit. If the check bit is 1, then it is assumed that the received data is incorrect. The check bit will be 0 if the received data is correct. The table shows the truth table for the even parity checker.

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

PROCEDURE:

1. Connections are given as per the logic diagram.
2. Logic inputs are given as per the truth table.
3. Observe the logic output and verify with the truth tables

[illegible]

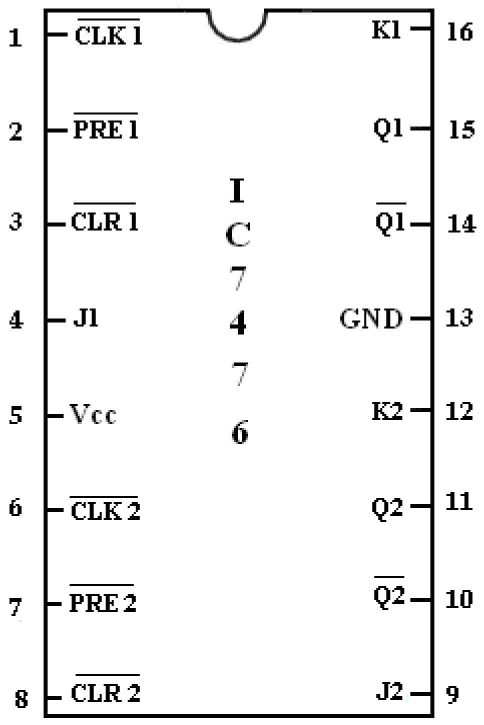
HOD / ECE

RESULT:

Thus the 16 bit odd /even parity checker generator was designed and implemented using IC 74180.

4- BIT RIPPLE COUNTER:

PIN DIAGRAM: (JK Flip-Flop)



Function Table for 7476:

[illegible]

HOD / ECE

Inputs					Outputs	
Preset	**Clear**	**Clock**	**J**	**K**	**Q**	**Q'**
0	1	X	X	X	1	0
1	0	X	X	X	0	1
0	0	X	X	X	1	1
1	1	↓	0	0	No Change	
1	1	↓	0	1	0	1
1	1	↓	1	0	1	0
1	1	↓	1	1	Toggle	

EX.NO: 8

DATE :

CONSTRUCTION AND VERIFICATION OF 4-BIT RIPPLE COUNTER & MOD-10/ MOD-12 RIPPLE COUNTERS

AIM:

To construct and verify 4 bit ripple counter, MOD-10 and MOD-12 ripple counter.

APPARATUS REQUIRED:

SL.NO	COMPONENTS	SPECIFICATION	QUANTITY
1.	IC Trainer kit	-	1
2.	JK Flip-flop	IC7476	2
3.	NAND gate	IC7400	1
4.	Patch cords	-	Few

THEORY:

A counter is a register capable of counting number of clock pulse arriving at its clock input. Counter represents the number of clock pulse arrived. A specified

[illegible]

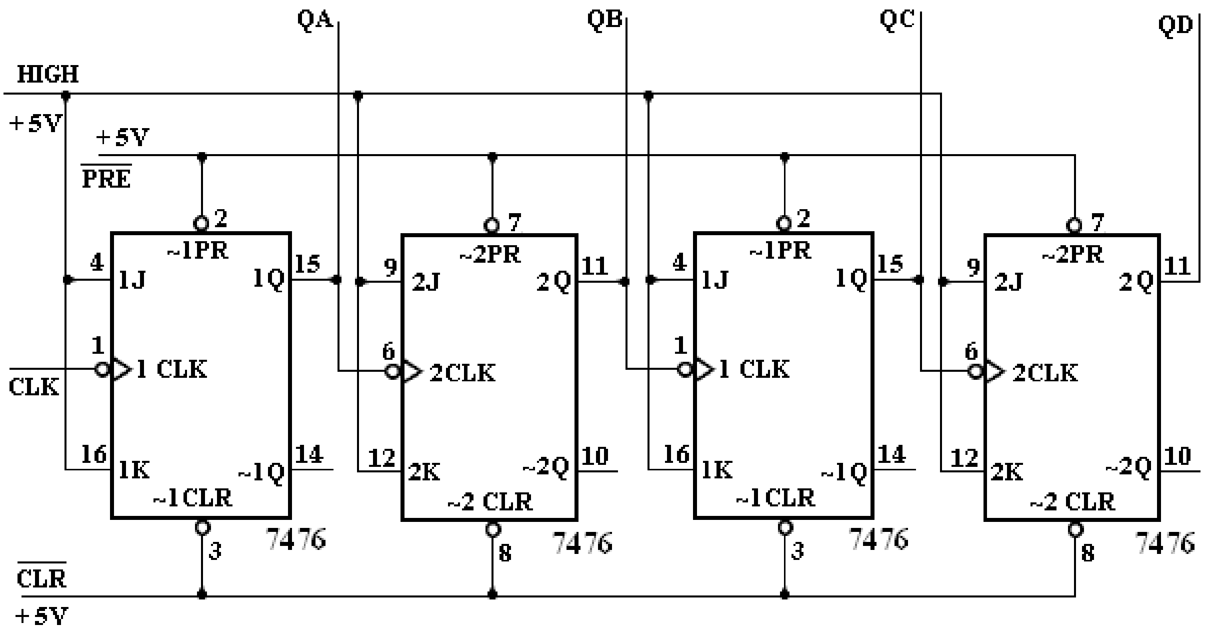
HOD / ECE

sequence of states appears as counter output. This is the main difference between a register and a counter. There are two types of counter, synchronous and asynchronous. In synchronous common clock is given to all flip flop and in asynchronous, first flip flop is clocked by external pulse and then each successive flip flop is clocked by Q or Q' output of pervious stage.

A ***ripple counter*** is a cascaded arrangement of flip-flops where the output of one flip-flop drives the clock input of the following flip-flop. The number of flip-flops in the cascaded arrangement depends upon the number of different logic states that it goes through before it repeats the sequence, a parameter known as the modulus of the counter.

In a ripple counter, also called an *asynchronous counter* or a *serial counter*, the clock input is applied only to the first flip-flop, also called the input flip-flop, in the cascaded arrangement. The clock input to any subsequent flip-flop comes from the output of its immediately preceding flip-flop. For instance, the output of the first flip-flop acts as the clock input to the second flip-flop, the output of the second flip-flop feeds the clock input of the third flip-flop and so on.

LOGIC DIAGRAM: (4-Bit Ripple Counter)



TRUTH TABLE:

CLK	Q_A	Q_B	Q_C	Q_D
0	0	0	0	0

[illegible]

HOD / ECE

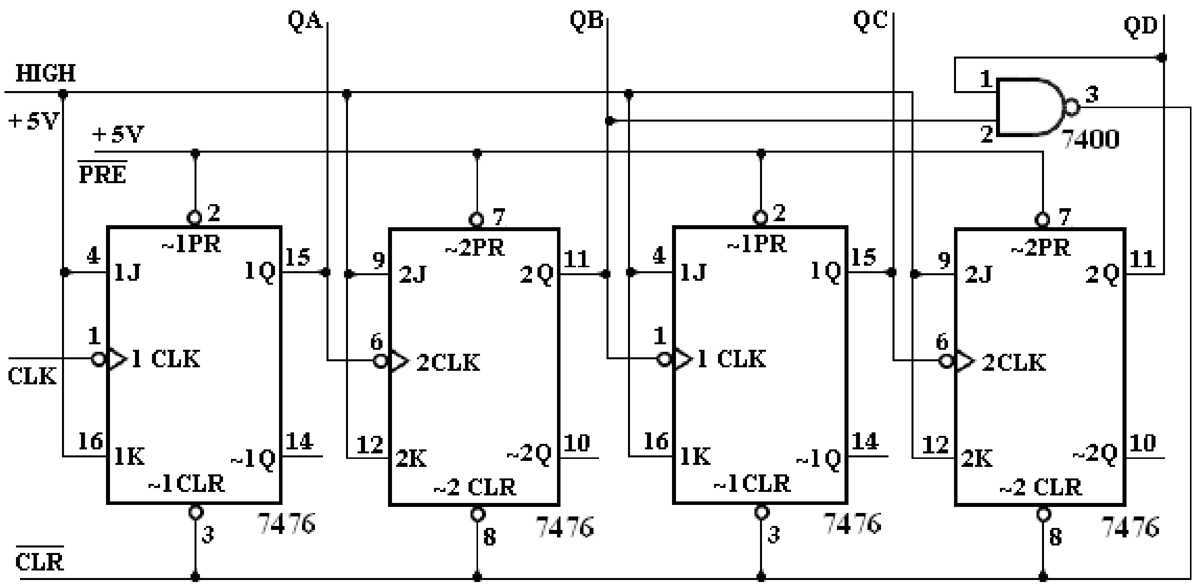
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	1	1
13	1	0	1	1
14	0	1	1	1
15	1	1	1	1

A four-bit ripple counter is implemented with negative edge-triggered *J-K* flip-flops wired as toggle flip-flops. The output of the first flip-flop feeds the clock input of the second, and the output of the second flip-flop feeds the clock input of the third, the output of which in turn feeds the clock input of the fourth flip-flop. The outputs of the four flip-flops are designated as Q0 (LSB flip-flop), Q1, Q2 and Q3 (MSB flip-flop).

<u>LOGIC DIAGRAM</u>: (MOD-10 Ripple Counter)

[illegible]

HOD / ECE



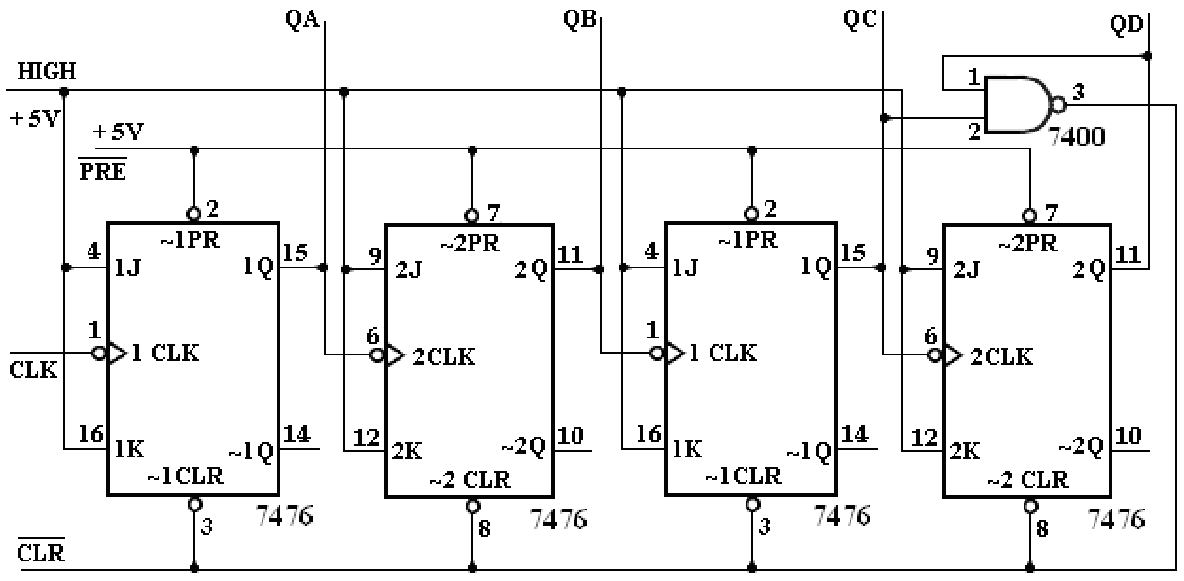
TRUTH TABLE:

CLK	Q_A	Q_B	Q_C	Q_D
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	0	0	0

LOGIC DIAGRAM: (MOD-12 Ripple Counter)

[illegible]

HOD / ECE



TRUTH TABLE:

CLK	Q_A	Q_B	Q_C	Q_D
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	0	0

PROCEDURE:

1. Connections are given as per the logic diagram.
2. Logic inputs are given as per the logic diagram.
3. Observe the logic output and verify with the truth tables.

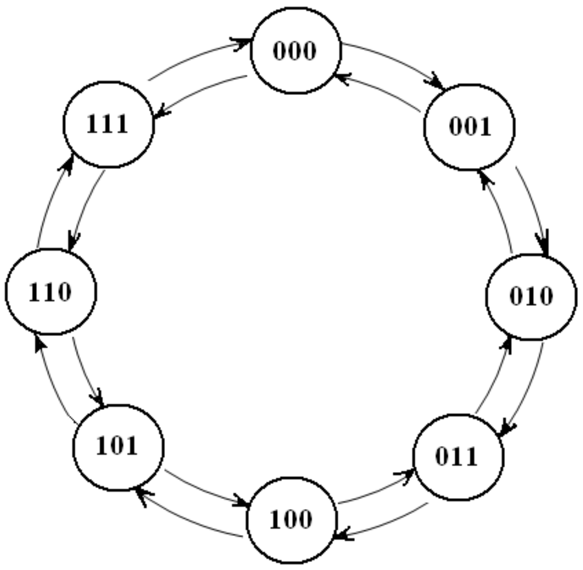
RESULT: Thus 4-bit ripple counter, MOD-10 and MOD-12 ripple counter was constructed and verified successfully.

3- BIT SYNCHRONOUS UP/DOWN COUNTER:

[illegible]

HOD / ECE

STATE DIAGRAM:



TRUTH TABLE:

Input	Present State			Next State			A		B		C	
Up/Down'	Q_A	Q_B	Q_C	Q_{A+1}	Q_{B+1}	Q_{C+1}	J_A	K_A	J_B	K_B	J_C	K_C
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

[illegible]

HOD / ECE

EX.NO: 9

DATE :

DESIGN AND IMPLEMENTATION OF 3-BIT SYNCHRONOUS UP/DOWN COUNTER

AIM:

To design and implement 3 bit synchronous up/down counter using JK flip-flop.

APPARATUS REQUIRED:

SL.NO	COMPONENTS	SPECIFICATION	QUANTITY
1.	IC Trainer kit	-	1
2.	JK Flip-flop	IC7476	2
3.	3-I/P NAND gate	IC7411	1
4.	NOT gate	IC7404	1
5.	OR gate	IC7432	1
6.	EX-OR gate	IC7486	1
7.	Patch Cords		Few

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 bi-directional counter. Usually up/down operation of the counter is controlled by up/down signal. When this signal high counter goes through up sequence and when up/down signal is low counter follows reverse sequence.

The counter counts upwards when UP control are logic '1' and DOWN control is logic '0'. In this case the clock input of each flip-flop other than the LSB flip-flop is fed from the normal output of the immediately preceding flip-flop. The counter counts downwards when the UP controls input are logic '0' and DOWN control is logic '1'. In this case, the clock input of each flip-flop other than the LSB flip-flop is fed from the complemented output of the immediately preceding flip-flop.

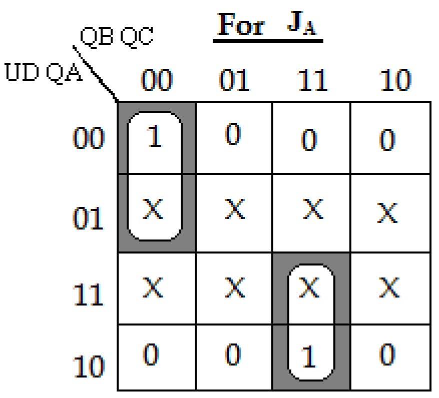
[illegible]

HOD / ECE

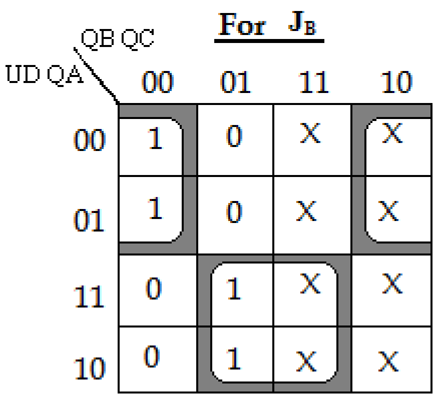
EXCITATION TABLE: (JK Flip-Flop)

Q	Q_{t+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

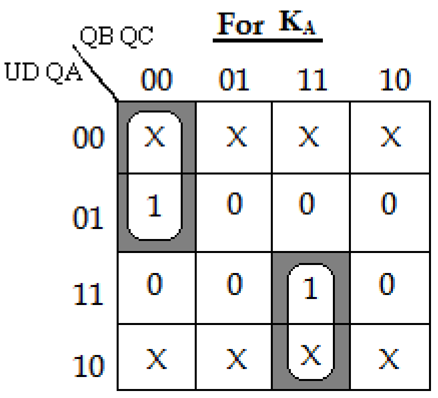
K-MAP SIMPLIFICATION:



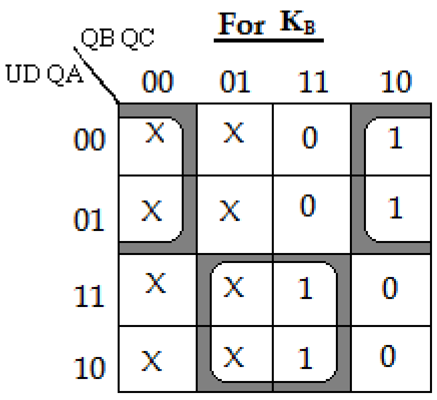
$J_A = \overline{UD}\;\overline{QB}\;\overline{QC} + UD\;QB\;QC$



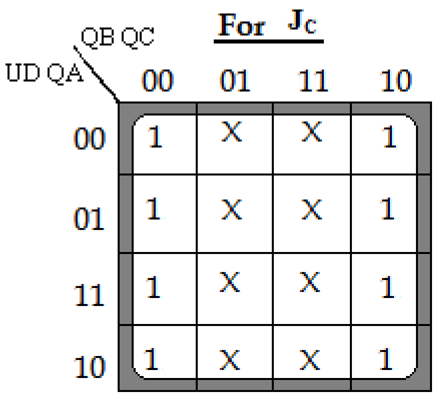
$J_B = \overline{(UD \oplus QC)}$



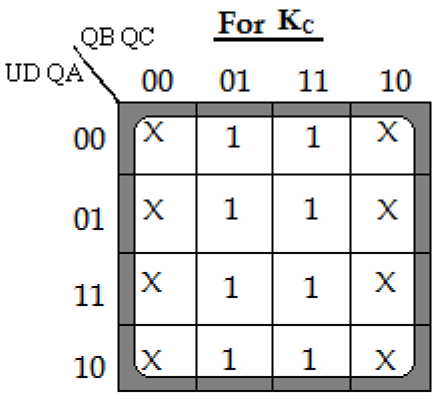
$K_A = \overline{UD}\;\overline{QB}\;\overline{QC} + UD\;QB\;QC$



$K_B = \overline{(UD \oplus QC)}$



$J_C = 1$

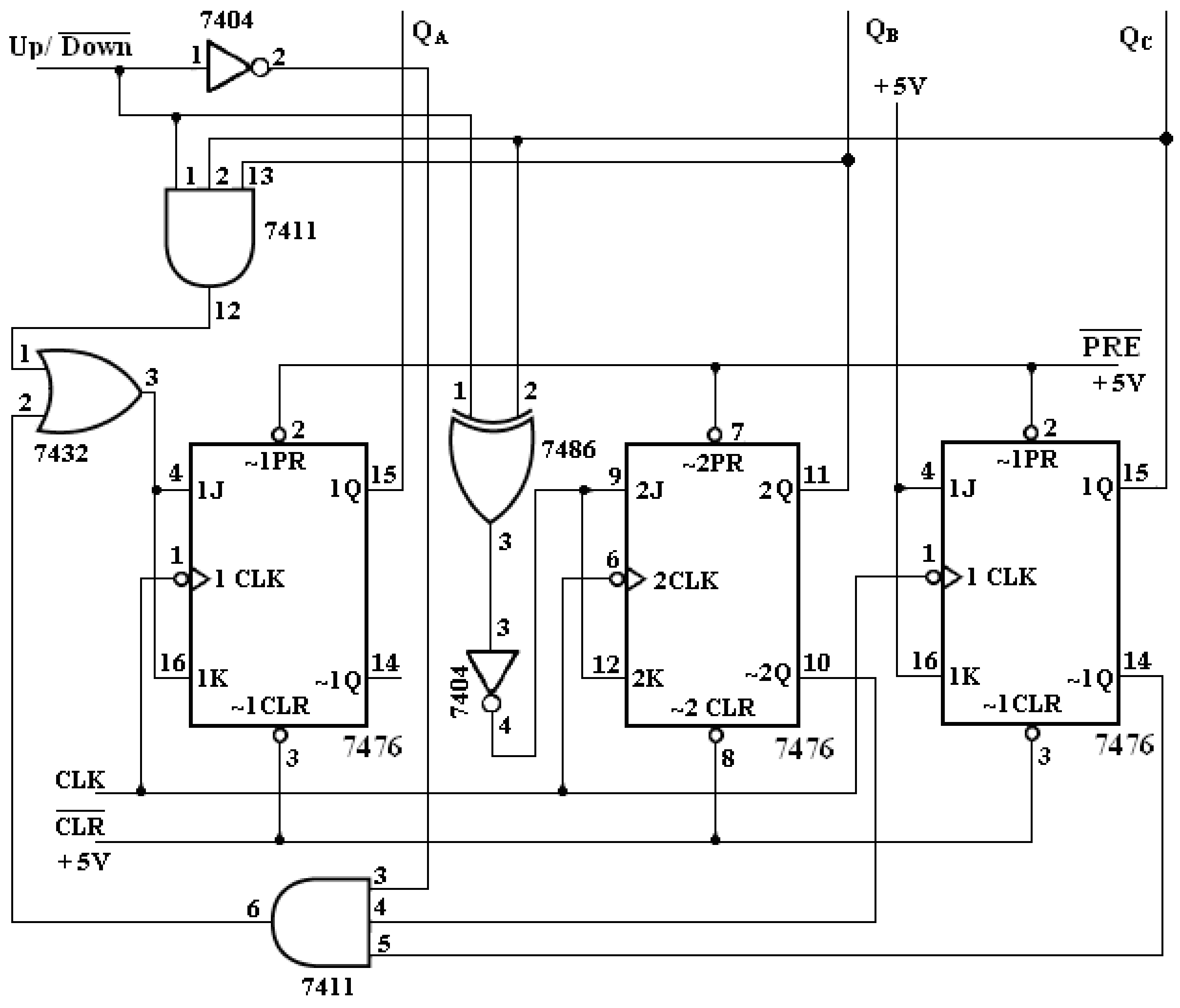


$K_C = 1$

[illegible]

HOD / ECE

LOGIC DIAGRAM:



PROCEDURE:

1. Connections are given as per the logic diagram.
2. Logic inputs are given as per the logic diagram.
3. Observe the logic output and verify with the truth tables.

RESULT:

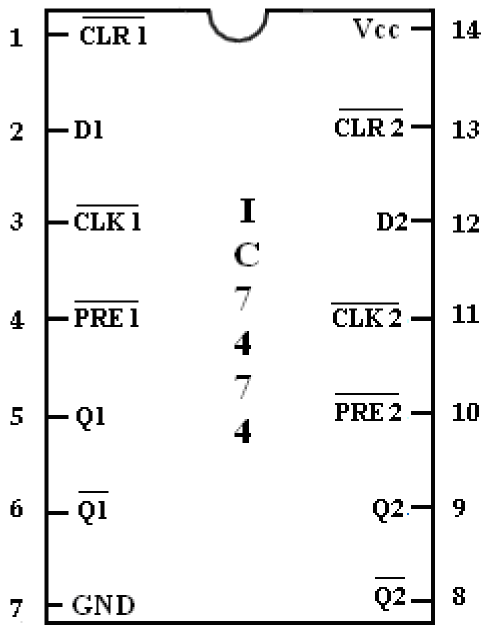
Thus 3- bit synchronous up/down counter was designed and implemented successfully.

[illegible]

HOD / ECE

SHIFT REGISTER:

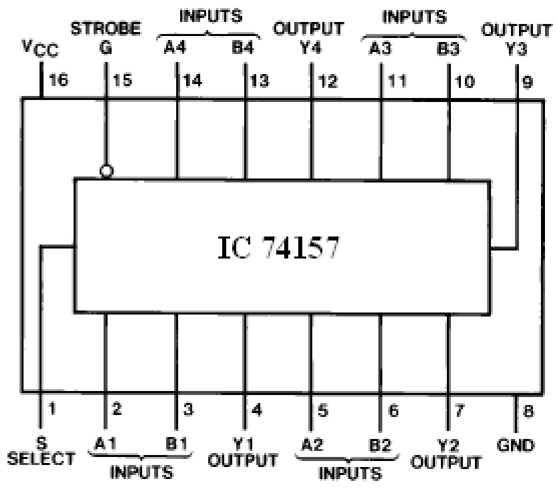
PIN DIAGRAM: (D-Flip-Flop)



Function Table:

Inputs				Outputs	
Preset	**Clear**	**Clock**	**D**	**Q**	**Q'**
0	1	X	X	1	0
1	0	X	X	0	1
0	0	X	X	1	1
1	1	↑	0	0	1
1	1	↑	1	1	0
1	1	0	X	No Change	

PIN DIAGRAM: (QUAD 2-LINE TO 1-LINE MULTIPLEXERS)



[illegible]

HOD / ECE

EX.NO: 10

DATE :

IMPLEMENTATION OF SHIFT REGISTERS USING FLIP-FLOPS

AIM:

To design and implement

1. Serial in serial Out(SISO)
2. Serial in parallel Out(SIPO)
3. Parallel in serial Out(PISO)
4. Parallel in parallel Out(PIPO)

APPARATUS REQUIRED:

SL.NO	COMPONENTS	SPECIFICATION	QUANTITY
1.	IC Trainer kit	-	1
2.	D-Flip flop	IC7474	2
3.	Quad 2-Line to 1-Line Multiplexer	IC74157	1
4.	Patch cords	-	Few

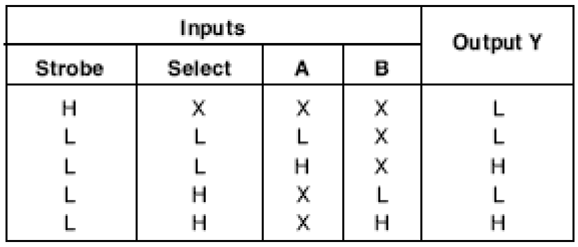
THEORY:

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

[illegible]

HOD / ECE

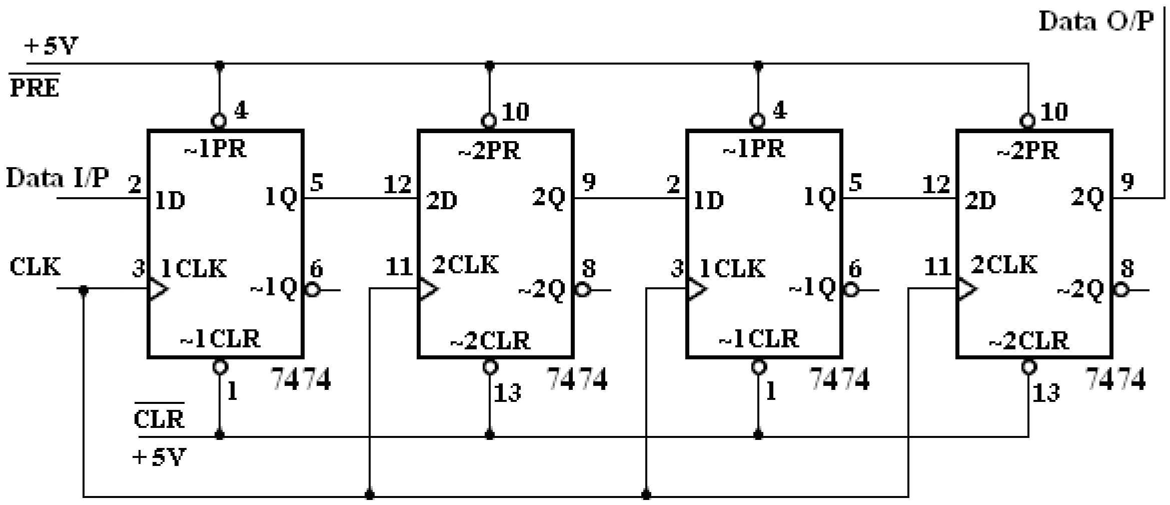
Function Table: **(74157)**



Inputs				Output Y
Strobe	Select	A	B	
H	X	X	X	L
L	L	L	X	L
L	L	H	X	H
L	H	X	L	L
L	H	X	H	H

SERIAL IN SERIAL OUT:

LOGIC DIAGRAM:



TRUTH TABLE:

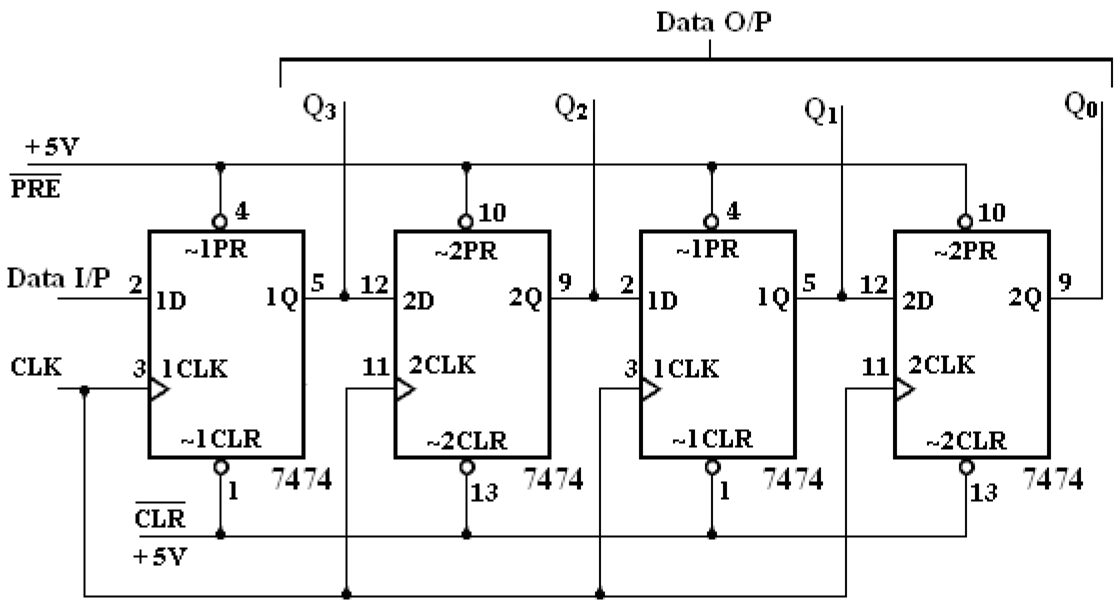
CLK	Serial IN	Serial OUT
1	1	0
2	1	0
3	1	0
4	1	1
5	0	1
6	0	1
7	0	1
8	0	0

[illegible]

HOD / ECE

Serial IN Parallel OUT:

LOGIC DIAGRAM:



TRUTH TABLE:

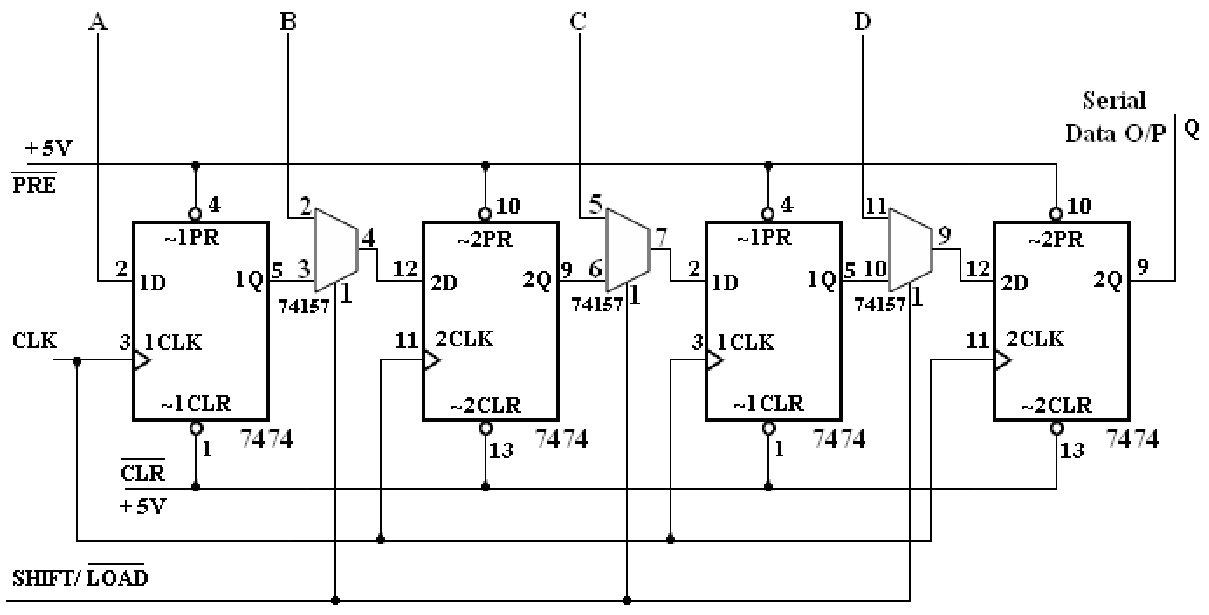
CLK	DATA	OUTPUT			
		Q_3	Q_2	Q_1	Q_0
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	0
4	1	1	0	0	1

[illegible]

HOD / ECE

Parallel IN Serial OUT:

LOGIC DIAGRAM:



Note: Pin 15 (Strobe) of 74157 is connected to the ground

TRUTH TABLE:

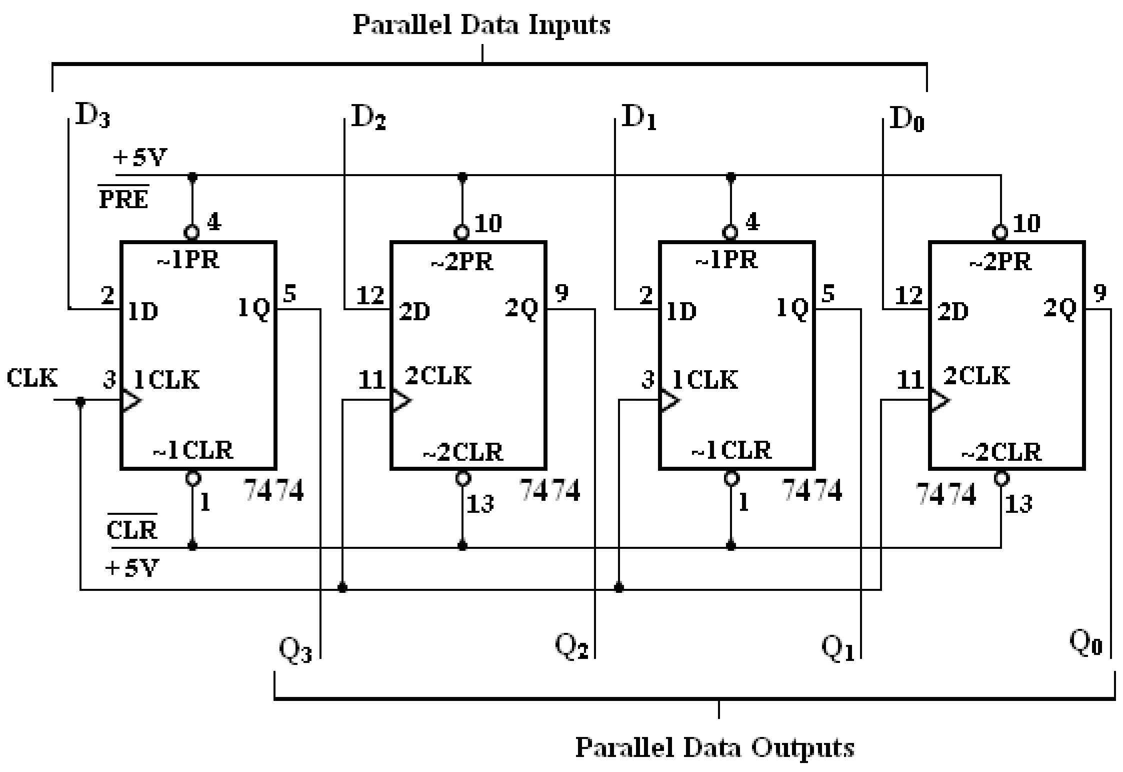
SHIFT/ LOAD'	CLK	INPUTS				OUTPUT
		A	B	C	D	Q
0	0	1	0	1	0	0
1	1	1	0	1	0	1
1	2	1	0	1	0	0
1	3	1	0	1	0	1

[illegible]

HOD / ECE

Parallel IN Parallel OUT:

Logic Diagram:



TRUTH TABLE:

CLK	DATA INPUTS				OUTPUT			
	D_3	D_2	D_1	D_0	Q_3	Q_2	Q_1	Q_0
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 the logic diagram.
2. Logic inputs are given as per the logic diagram.
3. Observe the logic output and verify with the truth tables.

RESULT:

Thus the design and implementation of

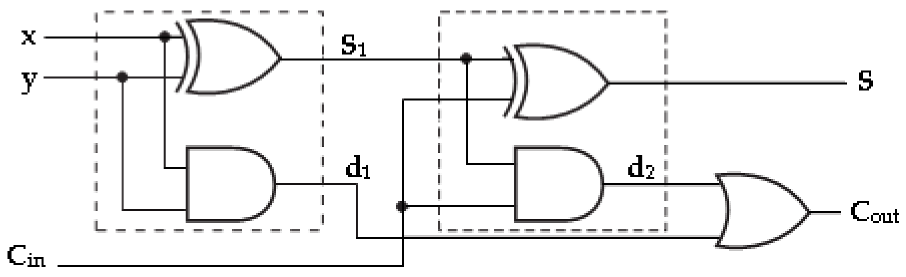
1. Serial in serial Out (SISO)
2. Serial in parallel Out (SIPO)
3. Parallel in serial Out (PISO)
4. Parallel in parallel Out (PIPO) were done successfully.

[illegible]

HOD / ECE

Logic Diagram:

Fulladder using two half adders:



Truth Table:

Inputs			Outputs	
x	**y**	**C_{in}**	**Sum (S)**	**Carry (C_{out})**
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

[illegible]

HOD / ECE

EX.NO: 11

DATE :

VERILOG HARDWARE DESCRIPTION LANGUAGE

Aim:

To design the following experiment using Verilog HDL

- Full adder and Full subtractor
- Multiplexer and Demultiplexer
- 4 – Bit Ripple Counter, MOD10 and MOD12 counter
- SISO, SIPO, PISO, PIPO Shift register

Software Used:

1. Xilinx 9.1i

Program:

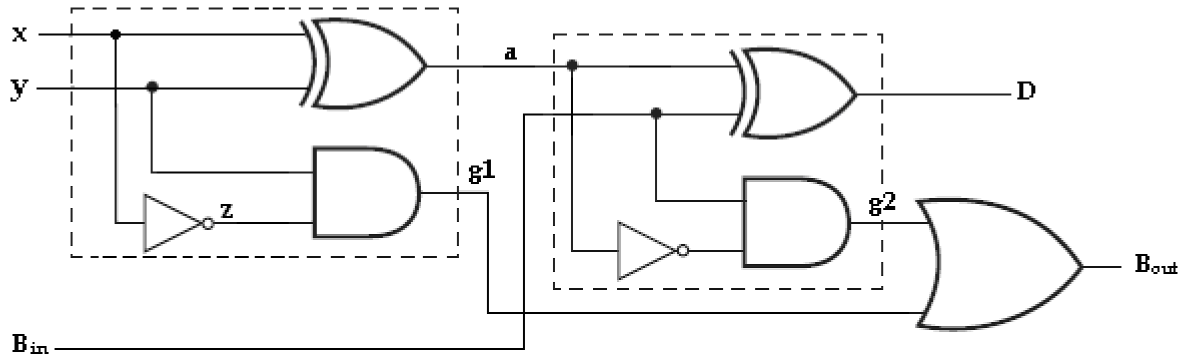
```
//Gate-level description of Full Adder using two Half Adder
//Description of Half Adder
module halfadder(s,co,x,y);
input x,y;
output s,co;
//Instatiate primitive gates
xor (s,x,y);
and (co,x,y);
endmodule
//Description of Full Adder
module fulladder(s,co,x,y,ci);
input x,y,ci;
output s,co;
wire s1,d1,d2;    //Outputs of first XOR and AND gates
//Instantiate Half Adder
halfadder ha_1(s1,d1,x,y);
halfadder ha_2(s,d2,s1,ci);
or or_gate(co,d2,d1);
endmodule
```

[illegible]

HOD / ECE

Logic Diagram:

Full Subtractor using two half subtractors:



Truth table:

Inputs			Outputs	
x	y	B_{in}	Difference(D)	Borrow(B_{out})
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

[illegible]

HOD / ECE

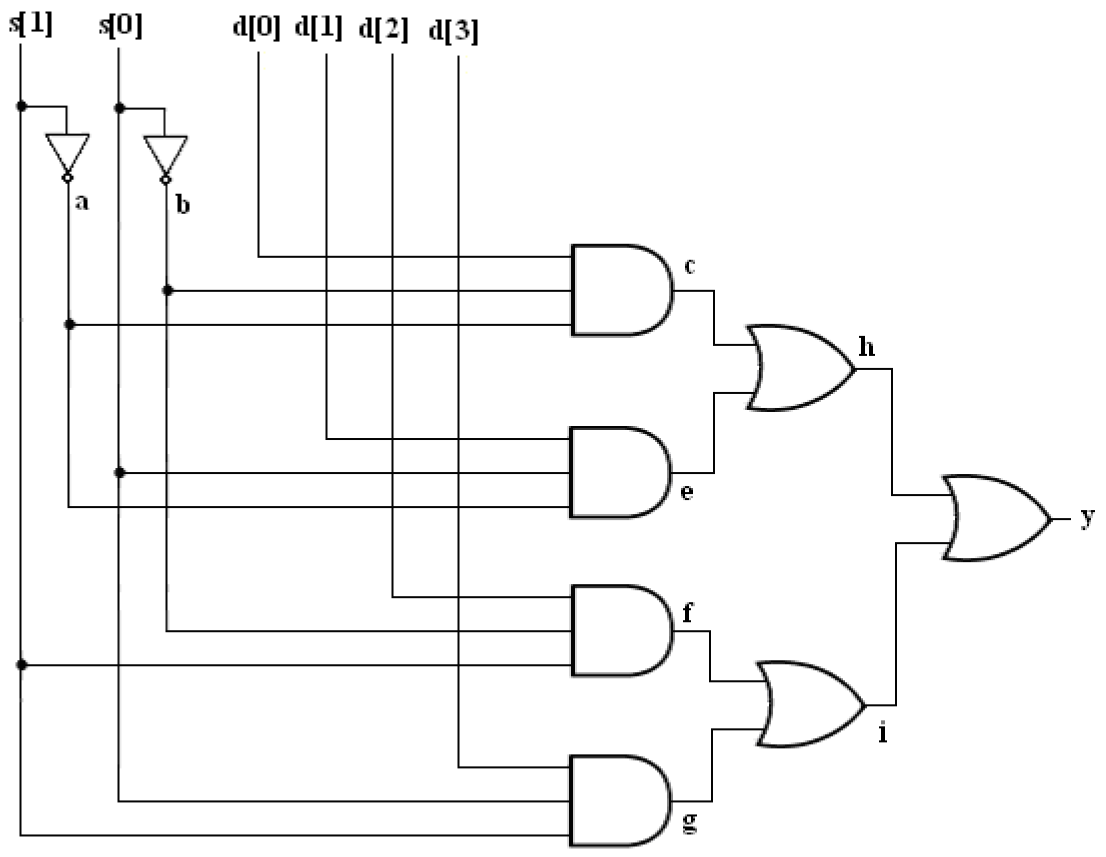
Program:

```
//Gate-level description of Full Subtractor using two Half Subtractor
//Description of Half Subtractor
module halfsubtractor(d,bo,x,y);
input x,y;
output d,bo;
wire z;     //Output of NOT gate
//Instatiate primitive gates
xor (d,x,y);
not (z,x);
and (bo,z,y);
endmodule
//Description of Full Subtractor
module fullsubtractor(d,bo,x,y,bi);
input x,y,bi;
output d,bo;
wire a,g1,g2;    //Outputs of first XOR and AND gates
//Instantiate Half Subtractor
halfsubtractor hs_1(a,g1,x,y);
halfsubtractor hs_2(d,g2,a,bi);
or or_gate(bo,g2,g1);
endmodule
```

[illegible]

HOD / ECE

Logic Diagram:
4 to 1 Multiplexer:



Truth table:

INPUT		OUTPUT
s[1]	**s[0]**	**y**
0	0	D[0]
0	1	D[1]
1	0	D[2]
1	1	D[3]

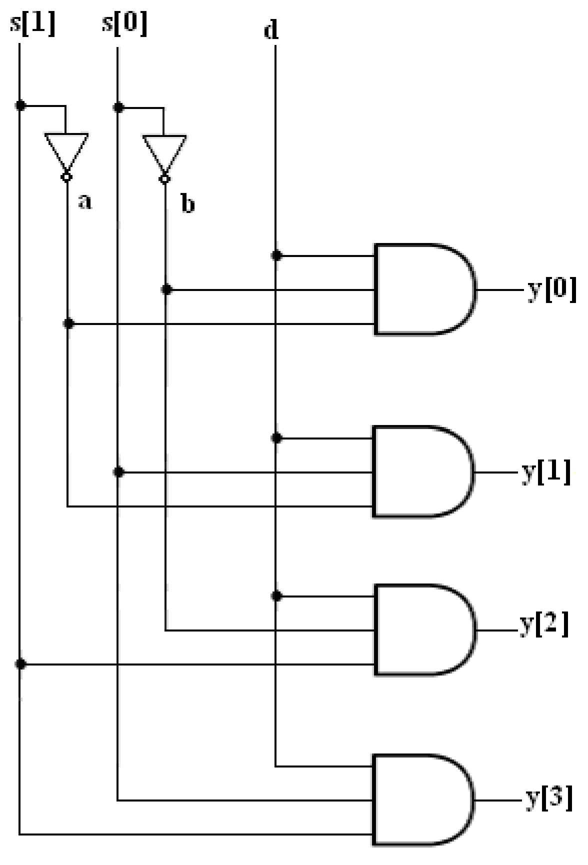
[illegible]

HOD / ECE

```
//Gate-level description of 4 to 1 Multiplexer
module multiplexer(y,d,s);
output y;
input [3:0] d;
input [1:0] s;
wire a,b,c,e,f,g,h,i;
//Instantiate Primitive gates
not (a,s[1]);
not (b,s[0]);
and (c,d[0],b,a);
and (e,d[1],s[0],a);
and (f,d[2],b,s[1]);
and (g,d[3],s[0],s[1]);
or (h,c,e);
or (i,f,g);
or (y,h,i);
endmodule
```

Logic Diagram:

1 to 4 Demultiplexer:



[illegible]

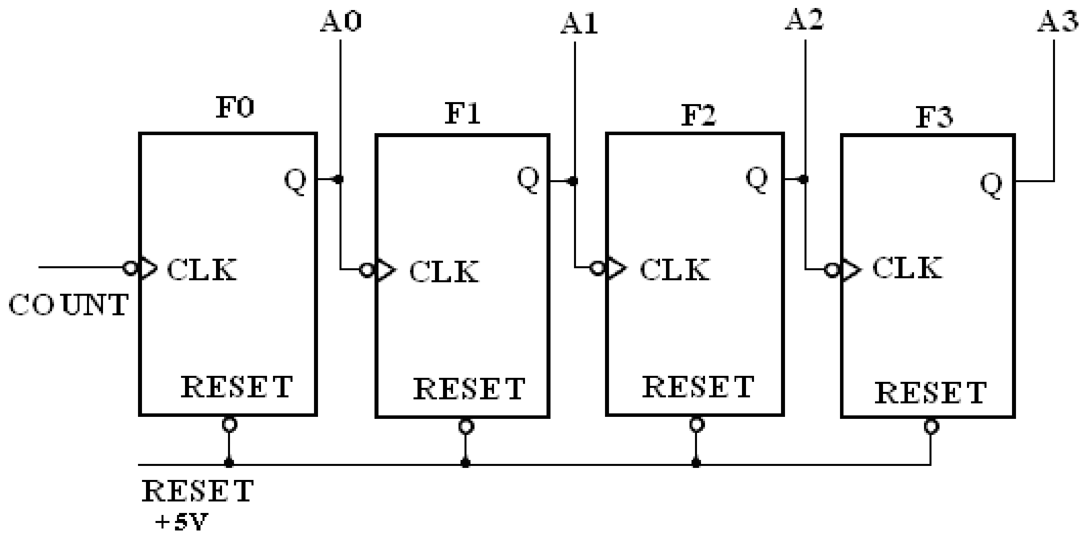
HOD / ECE

Truth Table:

INPUT			OUTPUT			
s[1]	**s[0]**	**D**	**y[0]**	**y[1]**	**y[2]**	**y[3]**
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	1	0
1	1	0	0	0	0	0
1	1	1	0	0	0	1

```
//Gate-level description of 1 to 4 Demultiplexer
module demultiplexer(y,d,s);
output [3:0]y;
input d;
input [1:0] s;
wire a,b;
//Instantiate Primitive gates
not (a,s[1]);
not (b,s[0]);
and (y[0],d,b,a);
and (y[1],d,s[0],a);
and (y[2],d,b,s[1]);
and (y[3],d,s[0],s[1]);
endmodule
```

LOGIC DIAGRAM: 4-Bit Ripple Counter:



TRUTH TABLE:

[illegible]

HOD / ECE

COUNT	A0	A1	A2	A3
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	1	1
13	1	0	1	1
14	0	1	1	1
15	1	1	1	1

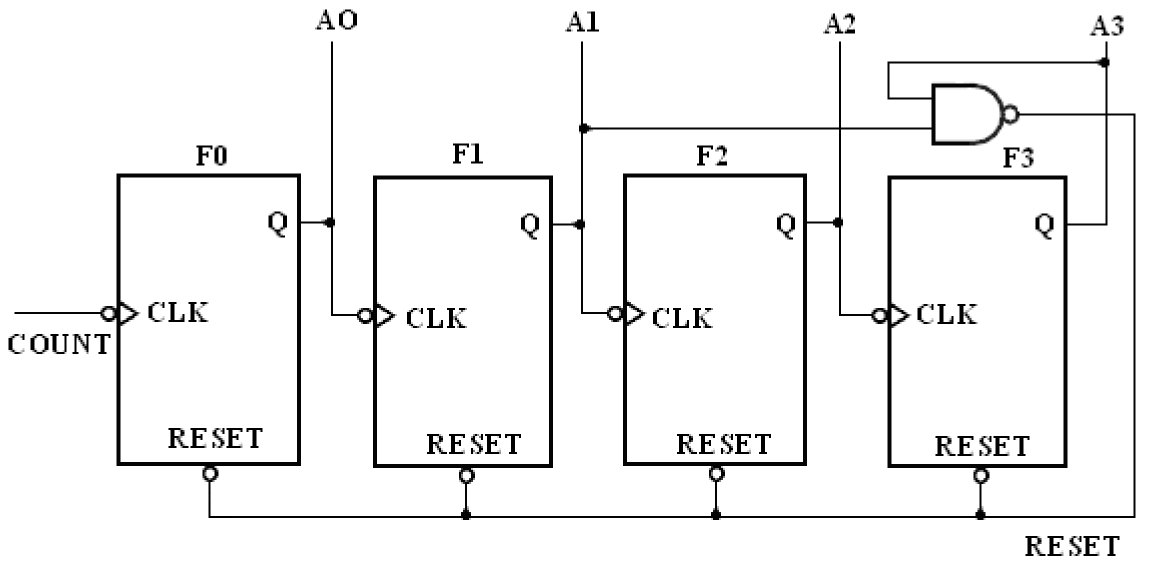
```
//Structural description of Ripple Counter
module ripplecounter(A0,A1,A2,A3,COUNT,RESET);
output A0,A1,A2,A3;
input COUNT,RESET;
//Instantiate  Flip-Flop
FF F0(A0,COUNT,RESET);
FF F1(A1,A0,RESET);
FF F2(A2,A1,RESET);
FF F3(A3,A2,RESET);
endmodule
//Description of  Flip-Flop
module FF(Q,CLK,RESET);
output Q;
input CLK,RESET;
reg Q;
always @(negedge CLK or negedge RESET)
if(~RESET)
Q=1'b0;
else
Q=(~Q);
endmodule
```

LOGIC DIAGRAM:

[illegible]

HOD / ECE

MOD-10 Ripple Counter:



TRUTH TABLE:

COUNT	A0	A1	A2	A3
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	0	0	0

//Structural description of MOD10 Counter

[illegible]

HOD / ECE

```
module MOD10(A0,A1,A2,A3,COUNT);
output A0,A1,A2,A3;
input COUNT;
wire RESET;
//Instantiate Flip-Flop
FF F0(A0,COUNT,RESET);
FF F1(A1,A0,RESET);
FF F2(A2,A1,RESET);
FF F3(A3,A2,RESET);
//Instantiate Primitive gate
nand (RESET,A1,A3);
endmodule
```

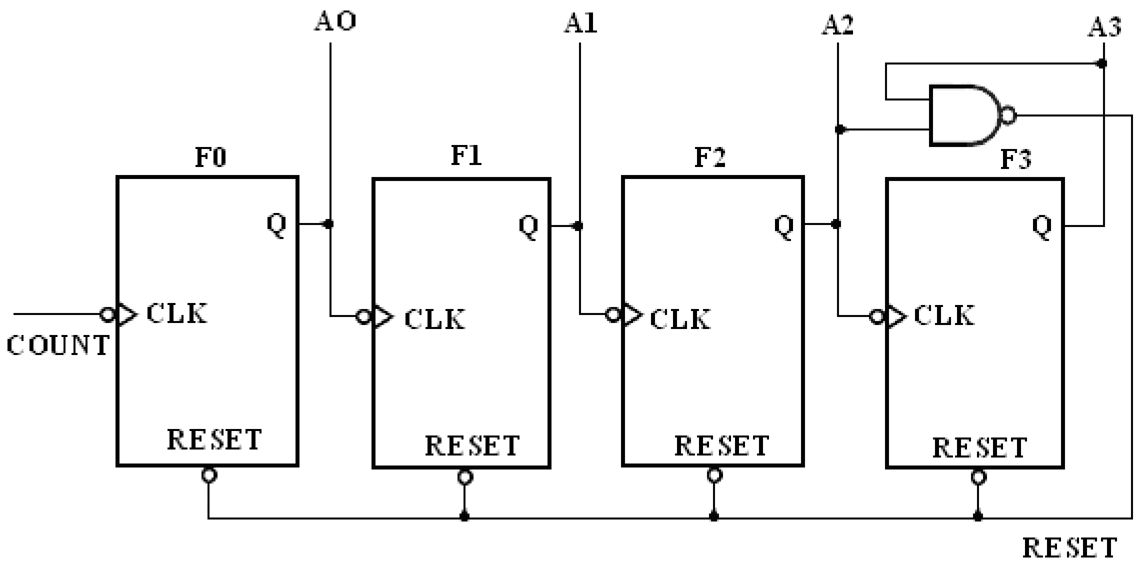
```
//Description of Flip-Flop
module FF(Q,CLK,RESET);
output Q;
input CLK,RESET;
reg Q=1'b0;
always @(negedge CLK or negedge RESET)
if(~RESET)
Q=1'b0;
else
Q=(~Q);
endmodule
```

[illegible]

HOD / ECE

LOGIC DIAGRAM:

MOD-12 Ripple Counter:



TRUTH TABLE:

COUNT	A0	A1	A2	A3
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	0	0

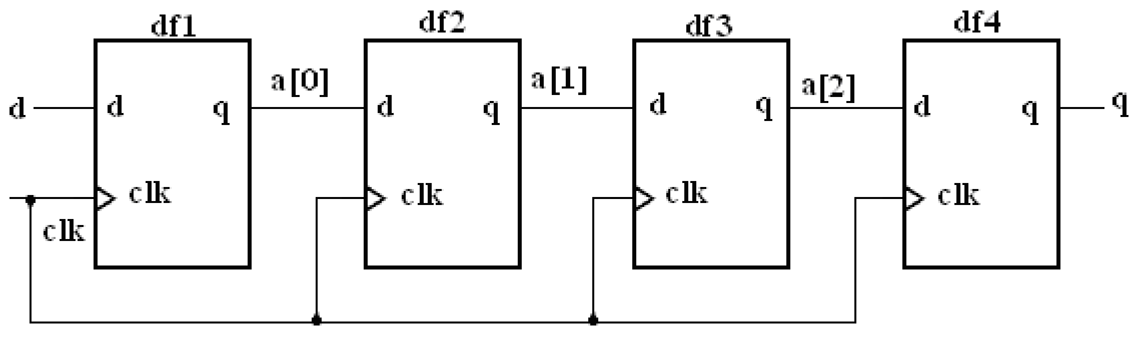
[illegible]

HOD / ECE

```
//Structural description of MOD12 Counter
module MOD12(A0,A1,A2,A3,COUNT);
output A0,A1,A2,A3;
input COUNT;
wire RESET;
//Instantiate Flip-Flop
FF F0(A0,COUNT,RESET);
FF F1(A1,A0,RESET);
FF F2(A2,A1,RESET);
FF F3(A3,A2,RESET);
//Instantiate Primitive gates
nand (RESET,A2,A3);
endmodule

//Description of Flip-Flop
module FF(Q,CLK,RESET);
output Q;
input CLK,RESET;
reg Q=1'b0;
always @(negedge CLK or negedge RESET)
if(~RESET)
Q=1'b0;
else
Q=(~Q);
endmodule
```

LOGIC DIAGRAM:

SERIAL IN SERIAL OUT:



TRUTH TABLE:

[illegible]

HOD / ECE

Clk	d	q
1	1	0
2	1	0
3	1	0
4	1	1
5	0	1
6	0	1
7	0	1
8	0	0

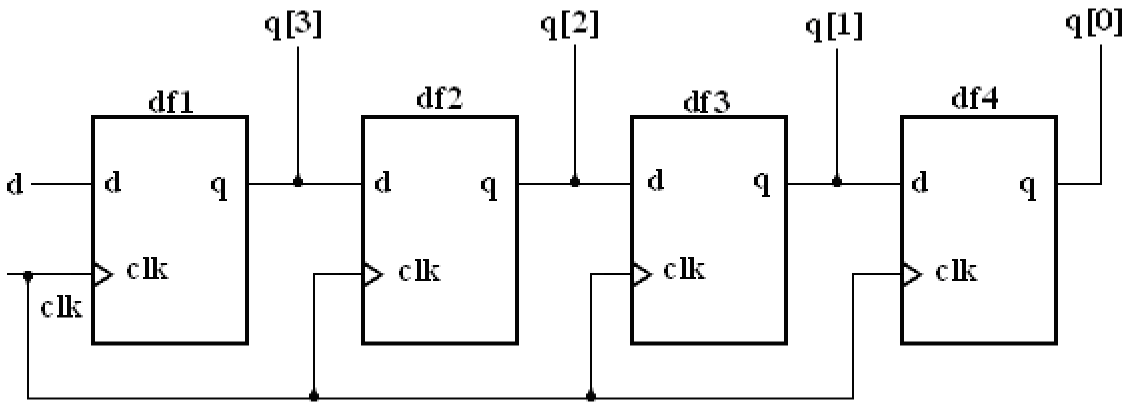
```
//Structural description of Serial in Serial Out Shift Register
module siso(q,d,clk);
output q;
input d,clk;
wire [2:0] a;
dff df1(a[0],d,clk);
dff df2(a[1],a[0],clk);
dff df3(a[2],a[1],clk);
dff df4(q,a[2],clk);
endmodule
```

```
//Description of D - Flipflop
module dff(q,d,clk);
output q;
input d,clk;
reg q=1'b0;
always @(posedge clk)
q=#5 d;
endmodule
```

LOGIC DIAGRAM:
Serial IN Parallel OUT:

[illegible]

HOD / ECE



TRUTH TABLE:

Clk	d	OUTPUT			
		Q[3]	Q[2]	Q[1]	Q[0]
1	1	1	0	0	0
2	1	1	1	0	0
3	1	1	1	1	0
4	1	1	1	1	1

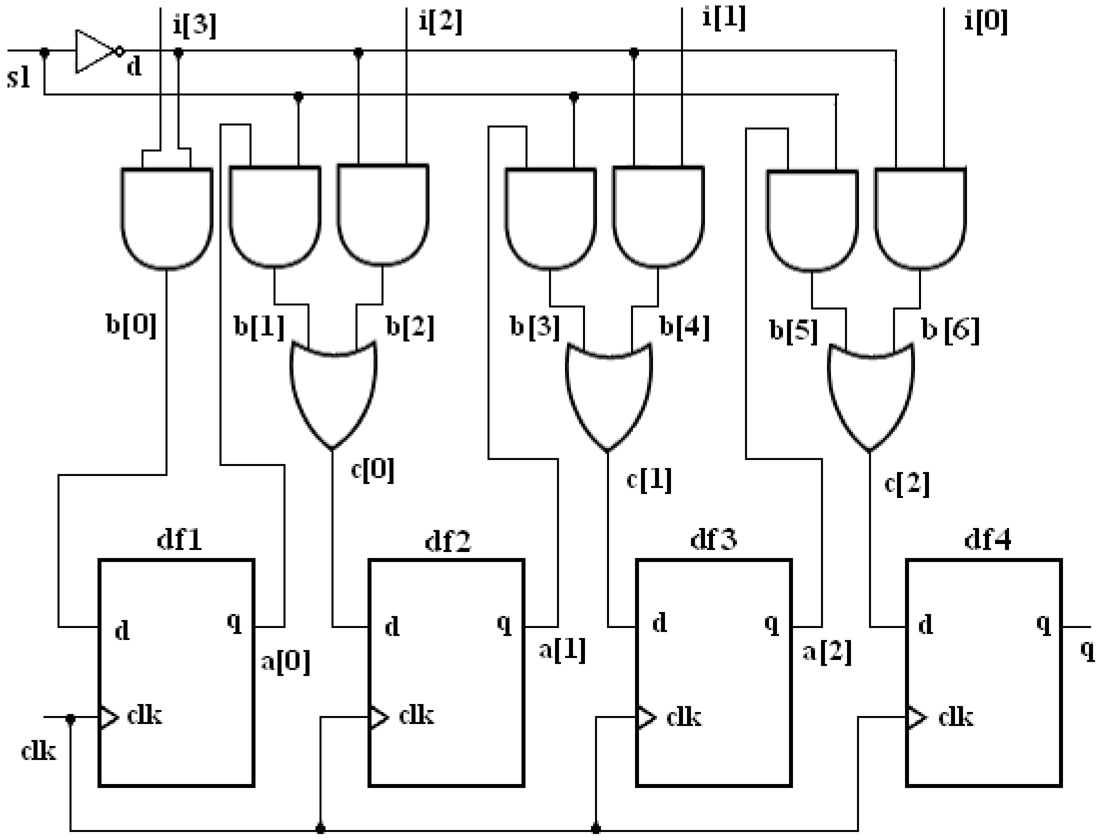
```
//Structural description of Serial in Parallel Out Shift Register
module sipo(q,d,clk);
output [3:0] q;
input d,clk;
dff df1(q[3],d,clk);
dff df2(q[2],q[3],clk);
dff df3(q[1],q[2],clk);
dff df4(q[0],q[1],clk);
endmodule

//Description of D - Flipflop
module dff(q,d,clk);
output q;
input d,clk;
reg q=1'b0;
always @(posedge clk)
q=#5 d;
endmodule
```

LOGIC DIAGRAM:

Parallel IN Serial OUT:

[illegible]

HOD / ECE



TRUTH TABLE:

Sl	Clk	INPUT				OUTPUT
		i[3]	i[2]	i[1]	i[0]	q
0	0	1	0	0	1	1
1	1	1	0	0	1	0
1	2	1	0	0	1	0
1	3	1	0	0	1	1

//Structural description of Parallel in Serial Out Shift Register
module piso(q,i,clk,sl);

[illegible]

HOD / ECE

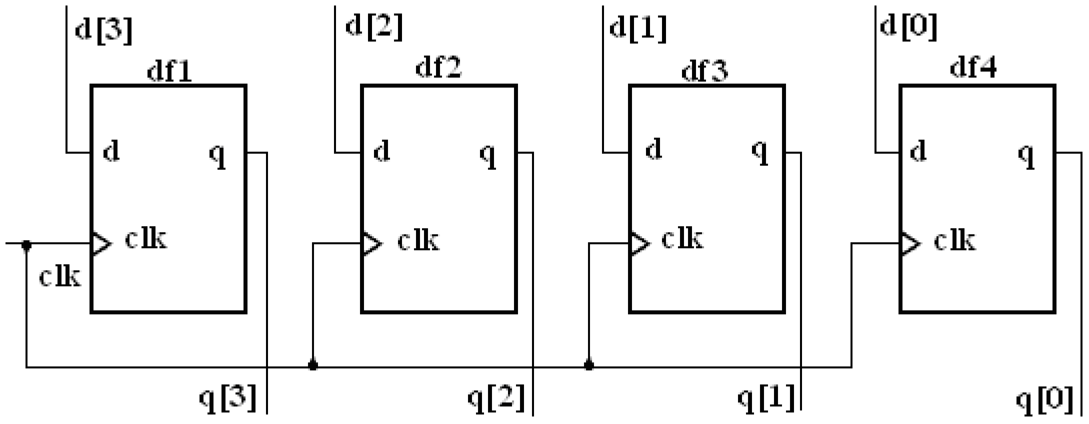
```
output q;
wire [2:0] a;
wire [6:0] b;
wire [2:0] c;
wire d;
input clk,sl;
input [3:0]i;
//Instantiate D – Flipflop
dff df1(a[0],b[0],clk);
dff df2(a[1],c[0],clk);
dff df3(a[2],c[1],clk);
dff df4(q,c[2],clk);
//Instantiate Primitive gates
not (d,sl);
and (b[0],d,i[3]);
and (b[1],a[0],sl);
and (b[2],d,i[2]);
and (b[3],a[1],sl);
and (b[4],d,i[1]);
and (b[5],a[2],sl);
and (b[6],d,i[0]);
or (c[0],b[1],b[2]);
or (c[1],b[3],b[4]);
or (c[2],b[5],b[6]);
endmodule
//Description of D - Flipflop
module dff(q,d,clk);
output q;
input d,clk;
reg q=1'b0;
always @(posedge clk)
q=d;
endmodule
```

Logic Diagram:

Parallel IN Parallel OUT:

[illegible]

HOD / ECE



<u>**TRUTH TABLE:**</u>

Clk	**INPUT**				**OUTPUT**			
	d[3]	**d[2]**	**d[1]**	**d[0]**	**q[3]**	**q[2]**	**q[1]**	**q[0]**
1	1	0	1	0	1	0	1	0
2	1	0	0	1	1	0	0	1

```
//Structural description of Parallel in Parallel Out Shift Register
module pipo(q,d,clk);
output [3:0] q;
input [3:0] d;
input clk;
//Instantiate D - Flipflop
dff df1(q[3],d[3],clk);
dff df2(q[2],d[2],clk);
dff df3(q[1],d[1],clk);
dff df4(q[0],d[0],clk);
endmodule

//Description of D - Flipflop
module dff(q,d,clk);
output q;
input d,clk;
reg q=1'b0;
always @(posedge clk)
q=#5 d;
endmodule
```

[illegible]

HOD / ECE