DIGITAL LOGIC DESIGN (EE-211)

Update History: Version-4 Fall 2020 (M. Junaid Khalid)

Name of Student: ……
Roll No: ……..
Teacher Name: ……

Marks Obtained: ……..
Remarks: ……..

Instructor's Signature: ……

Electrical Engineering Department
College of Engineering and Technology,
University of Sargodha.

LIST OF EXPERIMENTS

Lab No.	Title
1	TO STUDY BASIC LOGIC GATE INTEGRATED CIRCUITS AND VERIFICATION OF THEIR TRUTH TABLES
2	IMPLEMENTATION OF THE UNIVERSALITY OF NAND AND NOR GATES
3	IMPLEMENTATION OF THE HALF ADDER AND FULL ADDER
4	IMPLEMENTATION OF THE 4-BIT PARALLEL ADDER USING IC 74283
5	IMPLEMENT OF THE HALF AND FULL SUBTRACTOR
6	IMPLEMENTATION OF THE CODE CONVERTERS USING GATES
7	TO IMPLEMENT THE ENCODER AND DECODER USING IC 74138 & 74148
8	IMPLEMENTATION OF MULTIPLEXER AND DEMULTIPLEXER USING IC74151& IC74138
9	VERIFICATION OF LATCH AND FLIP FLOP OPERATION USING GATES AND FLIP FLOP'S IC
10	COUNTERS
11	IMPLEMENTATION OF SERIES AND PARALLEL REGISTERS
12	STUDY OF THE COMMANDS OF SHIFT AND ROTATE INSTRUCTIONS
13	ALU DESIGN IN VERILOG.
14	SEMESTER PROJECT EVALUATION

LIST OF CLOs

CLO	Domain Level	CLOs	PLO
CLO:1	P1	Describe and illustrate fundamentals of Digital Logic Design	1
CLO:2	P3	Demonstrate the acquired knowledge to apply techniques related to the design and analysis of digital logic circuits	2
CLO:3	P4	Design and Implement small-scale logic circuit (basic, combinational & sequential digital circuit) for desired output.	3
CLO:4	A2	Function individually as well as a team.	9

MAPPING OF CLOS WITH PLOs

PLOs	CLO 1	CLO 2	CLO 3	CLO4	CLO5	CLO6	CLO7	CLO8	CLO9
PLO:1 (Engineering Knowledge)	√								
PLO:2 (Problem Analysis)		√							
PLO:3 (Design Development of Solutions)			√						
PLO:4 (Investigation)									
PLO:5 (Modern Tool Usage)									
PLO:6 (Engineer & Society)									
PLO:7 (Environment and Sustainability)									
PLO:8 (Ethics)									
PLO:9 (Individual & Team Work)				√					
PLO:10 (Communication)									
PLO:11 (Project Management)									
PLO:12 (Life Long Learning)									

EXPERIMENT NO 1

TO STUDY BASIC LOGIC GATE INTEGRATED CIRCUITS AND VERIFICATION OF THEIR TRUTH TABLES

OBJECTIVES

To understand the different options, facilities and provisions provided on the Digital Logic Trainer

To recognize the different logic gates ICs

To verify the truth tables of basic logic gates

INTRODUCTION

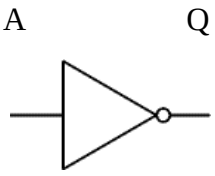
A logic gate is an elementary building block of a digital circuit. Most logic gates have two inputs and one output. At any given moment, every terminal is in one of the two logic states, i.e. LOW/FALSE/ (0) *or* HIGH/TRUE/ (1), represented by different voltage levels. The logic state of a terminal changes as the circuit processes data. In most logic gates, the LOW state corresponds to zero volts (0 V), while the HIHG state corresponds to positive five volts **(+5 V).** There are three basic logic gates, i.e. NOT-gate, AND-gate and OR-gate. A combination on these basic gates has given birth to some advanced gates which are widely used, e.g. NAND-gate, NOR-gate, EX-OR (Exclusive OR) gate and EX-NOR (Exclusive NOR) gate.

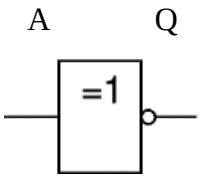
LOGIC GATE SYMBOLS AND TRUTH-TABLES

NOT GATE

NOT gate has only one input and one output. The output Q is true when the input A is NOT true, i.e. the output is the inverse of the input, mathematically we write it as Q = NOT (A). A NOT gate is also called an inverter.

*** IEC =**





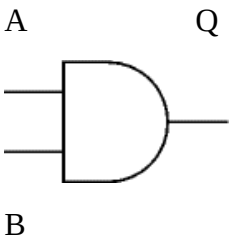
Input A	Output Q
0	1
1	0

Traditional symbol IEC symbol***** Truth Table

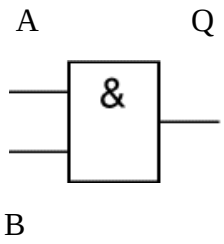
INTERNATIONAL ELECTRO TECHNICAL COMMISSION

AND GATE

A basic AND gate has two inputs and one output. The output Q is true if **both** the inputs A **AND** B are simultaneously true, mathematically this is stated as Q = A **AND** B. Some AND gates can have more than two inputs, in that case the output is true when **ALL** the inputs are true.



Traditional symbol



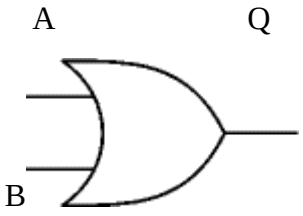
IEC symbol

Input A	Input B	Output Q
0	0	0
0	1	0
1	0	0
1	1	1

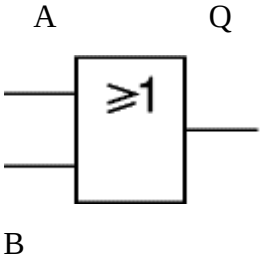
Truth Table

OR GATE

A basic OR gate also has two inputs and one output. The output Q is true if either of the two inputs A OR B is true (or when both of them are true), i.e. Q = A OR B. Some OR gates can have more than two inputs, in that case, the output is true if at least one input is true.



Traditional symbol



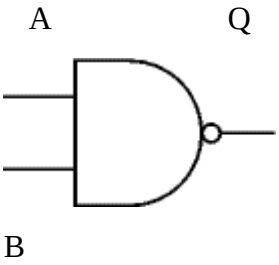
IEC symbol

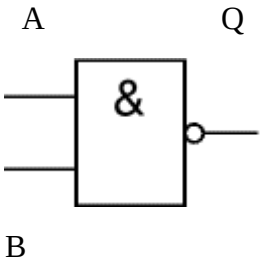
Input A	Input B	Output Q
0	0	0
0	1	1
1	0	1
1	1	1

Truth Table

NAND GATE

NAND gate is basically an AND gate but with the output inverted, as shown by the 'o' on the output of the AND gate symbol. Thus the output is true if both the inputs A AND B are NOT true simultaneously, in equation form we can write it as Q = NOT (A AND B). Like AND gates, some NAND gates can have more than two inputs, in that case, the output is true if NOT ALL the inputs are true.



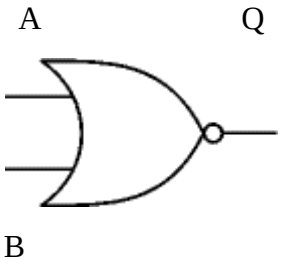


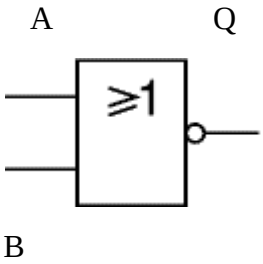
Input A	Input B	Output Q
0	0	1
0	1	1
1	0	1
1	1	0

Traditional symbol IEC symbol Truth Table

NOR GATE

NOR gate is basically an OR gate but with the output inverted, as shown by the 'o' on the output of the OR gate symbol. Thus the output is true if neither of the inputs A OR B is true, in equation form we can write it as Q = NOT (A OR B). Like OR gates, some NOR gates can have more than two inputs, in that case, the output is true if NONE of the inputs is true.



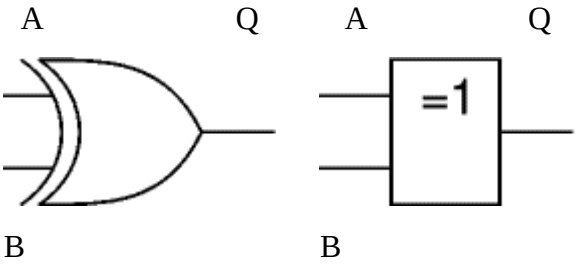


Input A	Input B	Output Q
0	0	1
0	1	0
1	0	0
1	1	0

Traditional symbol IEC symbol Truth Table

X-OR (EXCLUSIVE-OR) GATE

A basic X-OR gate also has two inputs and one output. The output Q is true if either of the two inputs A OR B (but not both) is true, mathematically it is written as Q = (A AND NOT(B)) OR (B AND NOT(A)). Hence it is like an OR gate but excluding the case when both the inputs are true simultaneously, in other words the output is true if the inputs are DIFFERENT. Advanced X-OR gates can have more than two inputs, in that case, the output is true only when an ODD number of inputs are true.



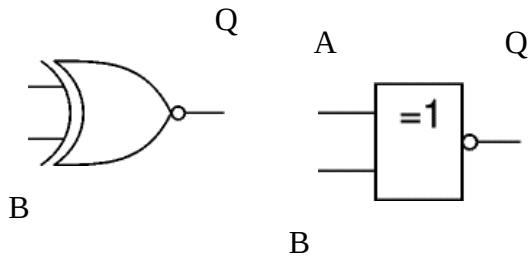
Traditional symbol IEC symbol

Input A	Input B	Output Q
0	0	0
0	1	1
1	0	1
1	1	0

Truth Table

X-NOR (EXCLUSIVE-NOR) GATE

This is equivalent to an X-OR gate with the output inverted, as shown by the 'o' on the output of the X-OR gate symbol. Thus the output Q is true if both the inputs A and B are the same, i.e. either both are true or both are false, in equation form this is described as Q = (A AND B) OR (NOT(A) AND NOT(B)). Like X-OR, X-NOR gates have more than two inputs, in that case, the output is true when an EVEN number of inputs are true.



Input A	Input B	Output Q
0	0	1
0	1	0
1	0	0
1	1	1

Traditional symbol IEC symbol Truth Table

UNIVERSAL GATES

The NAND and NOR gates can be said to be universal gates, since combinations of them can be used to accomplish any of the basic operations and can, thus produce an inverter, an OR gate or an AND gate. The non-inverting gates do not have this versatility since they can't produce an inversion.

COMPONENTS

1. Power supply
2. Components (ICs):

74LS00 (NAND Gate) --------- 1

74LS02 (AND Gate) --------- 1

74LS04 (NOT Gate) --------- 1

74LS08 (AND Gate) --------- 1

74LS32 (OR Gate) --------- 1

74LS86 (EX-OR Gate) --------- 1

3. Connecting wires
4. Bread board
5. LED

PROCEDURE

1. Connect the DC power supply to 220V Ac power supply.
2. Turn on the DC power supply and verify the DC voltage by using voltmeter, it should be almost 5.0 volts (specifically between 4.75V – 5.25V). If not consult the Lab Supervisor.

3. Install the IC chip under experiment, on breadboard.
4. Connect the +Vcc (pin # 14) and Ground (pin # 7) pins of the IC to +5V and Ground supply of the trainer board. (Consult Fig 1.2 for the pin diagrams of the IC under test)
5. Make the appropriate circuit connections as shown in Fig 1.1(a, b, c,e,f) for the particular IC under test. Use logic switches to provide "0" and "1" at the inputs and use the trainer's LEDs to display the output. Note that there is more than one gate in each IC chip, so you can use any one of these gates to make your connections (Consult Fig 1.2 for the pin numbers corresponding to each gate in that particular chip).
6. Record your observations according to Table 2.1 and verify whether the output conform to the truth tables of each gate.
7. Repeat steps 3-6 for each of the IC chips.
8. Write down your observations & comments at the end, as per your concept developed during this experimental work.

CIRCUIT DIAGRAMS

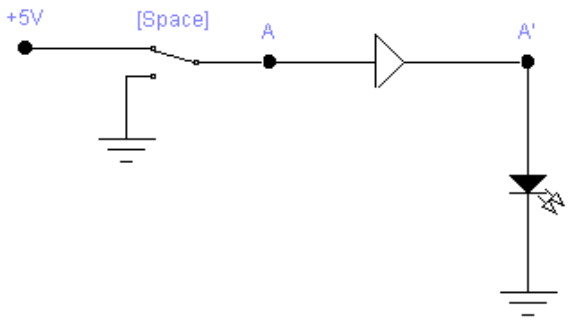


Fig 1.1 (a) NOT Gate (7404)

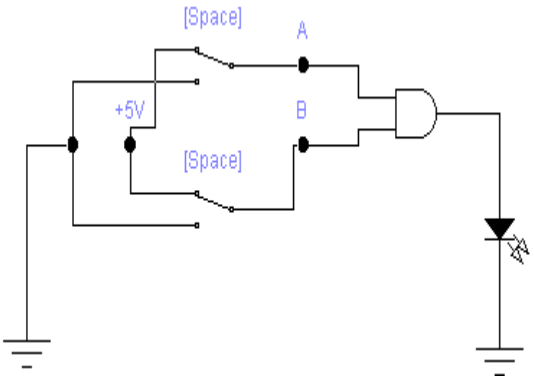


Fig 1.1 (b) AND Gate (74LS08)

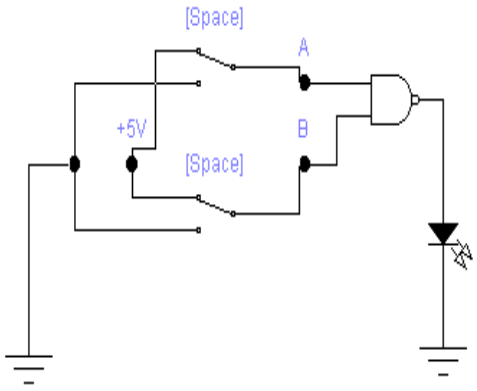


Fig 1.1 (c) NAND Gate (74LS00)

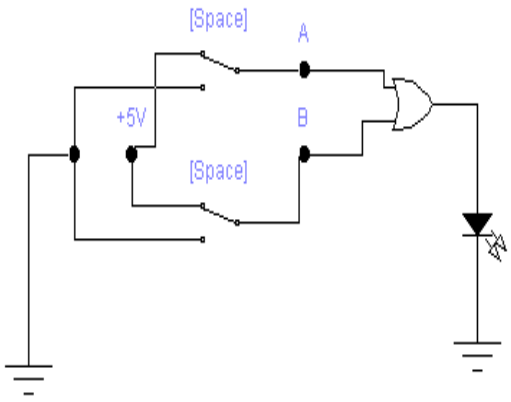


Fig 1.1 (d) OR neither Gate (74LS32)

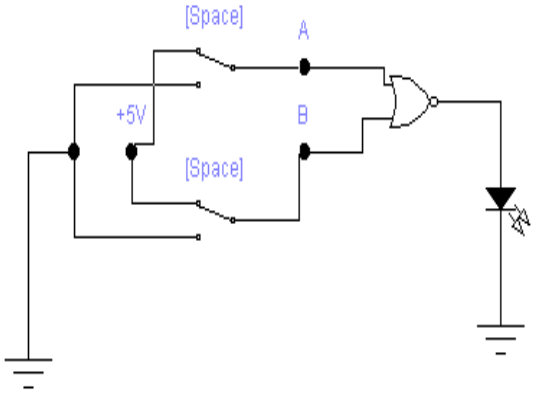
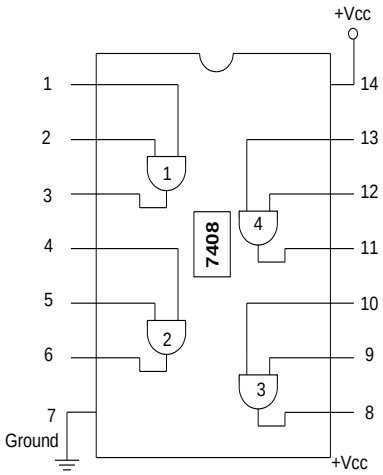
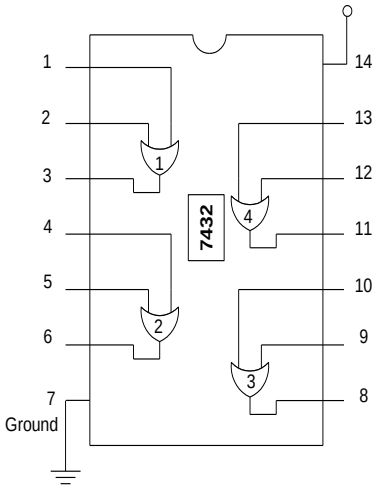


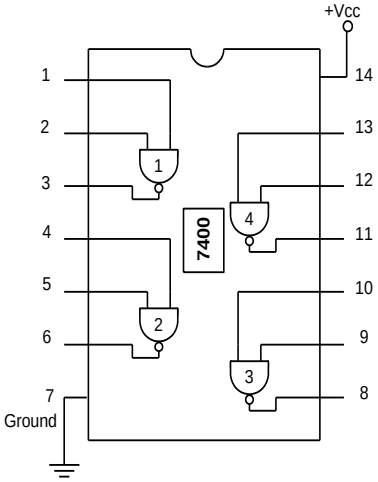
Fig 1.1 (e) NOR Gate (74LS02)



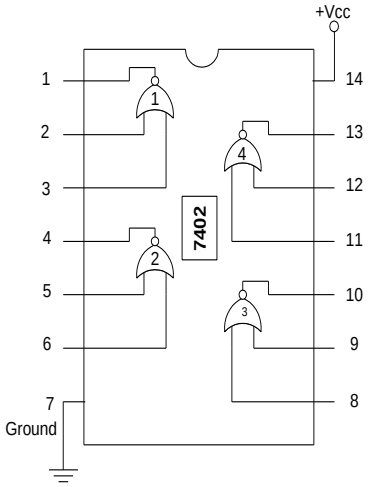
AND GATE



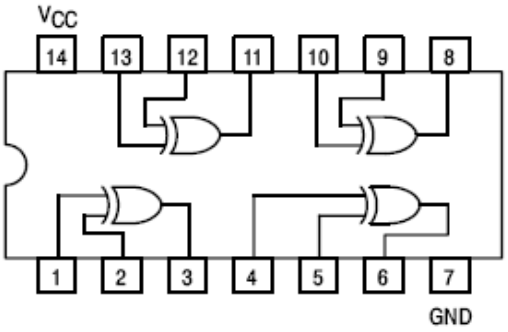
OR GATE



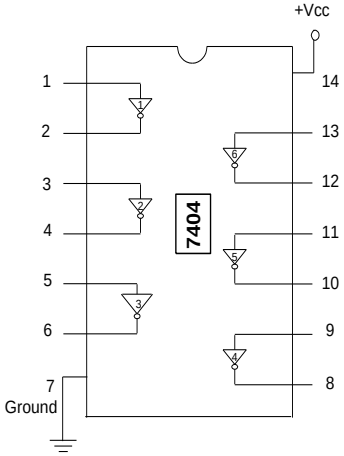
NAND GATE



NOR GATE



EX-OR GATE



NOT GATE

Fig 1.2: Pin Diagrams of IC's

EXPERIMENTAL RESULTS

Table 1.3 Truth Tables of Different Logic Gates

Input A	Output Q
0	
1	

NOT GATE

Input A	Input B	Output Q
0	0	
0	1	
1	0	
1	1	

OR GATE

Input A	Input B	Output Q
	0	
0	1	
1	0	
1	1	

AND GATE

Input A	Input B	Output Q
0	0	
0	1	
1	0	
1	1	

NOR GATE

Input A	Input B	Output Q
0	0	
0	1	
1	0	
1	1	

NAND OR GATE

Input A	Input B	Output Q
0	0	
0	1	
1	0	
1	1	

EX-GATE

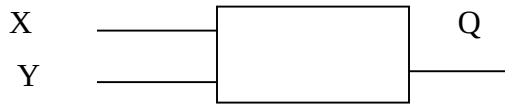
PRECAUTIONS

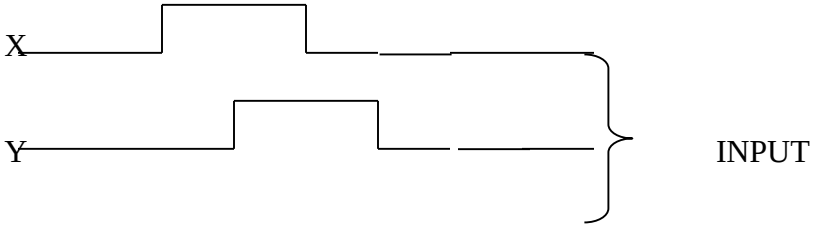
- Check the power supply for correct voltage.

- Check the Vcc (pin # 14) and Ground (pin # 7) connections of the IC under test.
- Check all the wire connections and remove any possible breaks.
- Check the IC under test using truth table.

QUESTIONS

The input/output signals are shown by means of the following diagram. Corresponding to the Input X and Y given below; Draw waveform for given gates.





AND

OR

NOT

Conclusion & Comments

EXPERIMENT NO 2

IMPLEMENTATION OF UNIVERSALITY OF NAND AND NOR GATES

OBJECTIVE

To study and implement any logic expression by using only NAND or NOR gates.

THEORY

Digital circuits are more frequently constructed with NAND or NOR gates than with AND and OR gates. NAND and NOR gates are easier to fabricate with electronic components and are the basic gates used in all IC digital logic families. Because of the prominence of NAND and NOR gates in the design of digital circuits, rules and procedures have been developed for conversion from Boolean function given in terms of AND, OR, and NOT into equivalent NAND and NOR logic diagram.

TASK 1: NAND GATE IMPLEMENTATION OF BOOLEAN FUNCTIONS

If we can show that the logical operations AND, OR, and NOT can be implemented with NAND gates, then it can be safely assumed that any Boolean function can be implemented with NAND gates. Figure-2 below shows such implementation:

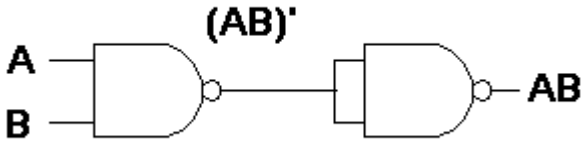


Figure 2a: AND gate operation

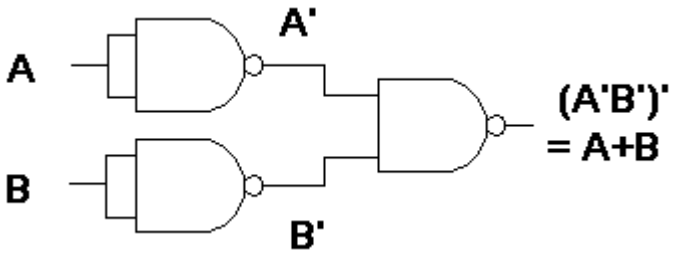


Figure 2b: OR gate operation

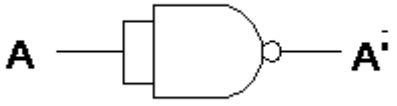


Figure 2c: NOT gate operation

PROCEDURE

1. Insert the IC on the trainer's breadboard.

2. Use any one or more of the NAND gates of the IC for this experiment.

3. Any one or more Logic Switches of the trainer (S2 to S9) can be used for input to the NAND gate.

4. For output indication, connect the output pin of the circuit to any one of the LEDs of the trainer (L0 to L15).

5. Connect the circuit as per Fig. 1(a) above.

6. Connect +5V to pin 14 (Vcc) and Ground to pin 7 (GND) of the IC.

7. By setting the switches to 1 and0, verify that the output of the circuit conforms to that of an AND gate. Record your observation in the table below:

Inputs		Output Desired	
A	B	x=A.B	Observed
0	0	0	
0	1	0	
1	0	0	
1	1	1	

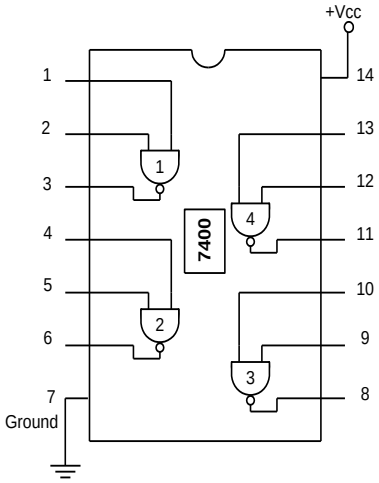
Verify OR gate operation using NAND gates. Show your results to the lab instructor.

Inputs		Desired Output	
A	B	x=A+B	Observed
0	0	0	
0	1	1	
1	0	1	
1	1	1	

Verify NOT gate operation using NAND gates

Show your results to the lab instructor.

Inputs	Output	
A	Desired X=A'	Observed
0	1	
1	0	



LAB ASSIGNMENT

P1. Implement AND, OR and NOT operation using NOR gate. Show truth table and logic diagram.

Conclusion & Comments

EXPERIMENT NO 3

IMPLEMENTATION OF HALF ADDER AND FULL ADDER

OBJECTIVE

To study Half and Full adder operations.

THEORY

Digital computers perform a variety of information-processing tasks. Among the basic functions encountered are various arithmetic operations. The most basic arithmetic operations are, no doubt, the addition and subtraction of binary digits (bit).

HALF ADDER

The possible operations, when we want to add only two bits, would be the followings:

0 + 0 = 0 0 + 1 = 1

1 + 0 = 1

1 + 1 = 0 & Carry 1

Above mentioned operation could be performed by a **Half Adder** circuit.

FULL ADDER

We know that in practice, all addition operations must take into account the Carry bit (or digit) from the previous operation. Adders in digital computers also take into account the Carry bit from last operation and add it with the Augend and Addend bits of the present operation to complete the addition operation. The possible operations are:

0 + 0 + 0 (carry) = 0

0 + 0 + 1 (carry) = 1 0 + 1 + 0 (carry) = 1

0 + 1 + 1 (carry) = 0 & carry 1 (to be added to next higher digit)

1 + 1 + 0 (carry) = 0 & carry 1 (to be added to next higher digit)

1 + 1 + 1 (carry) = 1 & carry 1 (to be added to next higher digit)

The adder that performs the addition of three bits (two significant bits and a previous carry) is called a **Full Adder**.

TASK 1: HALF ADDER

We arbitrarily assign symbols **A** and **B** to the two inputs and **S** (for sum) and **Cout** (for Carry) to the two outputs. Truth table for Half Adder as shown below:-

Table 1: Truth Table for Half Adder operation

Input		Output Desired		Observed	
A	**B**	**S**	**Cout**	**S**	**Cout**
0	0	0	0		
0	1	1	0		
1	0	1	0		
1	1	0	1		

The simplified Boolean function for the two outputs can be written from this truth table as:-

$$S = A'.B + A.B'$$

$$Cout = A.B$$

The circuit diagram for the Half Adder to implement above mentioned Boolean function could be quite a few. We will however verify only one.

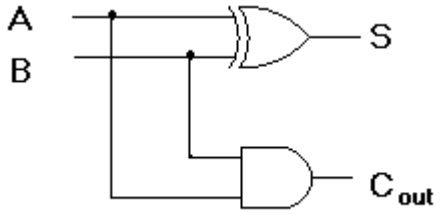


Figure 1: Half Adder using AND and XOR gates.

PROCEDURE

1. Wire the circuit as per figure 2 above.

2. Use any two Logic Switches of the trainer (S2 to S9) for the input and any two of the LEDs of the trainer (L0 to L15) as output indication.

3. Connect +5V to pin 14 (Vcc) and Ground to pin 7 (GND) of the ICs.

4. By setting various combinations of the two switches verify that the output of the circuit is in accordance with the Truth Table shown above. Record your observation.

TASK 2: FULL ADDER

As mentioned in the beginning, a full-adder is a combinational circuit that forms the arithmetic sum of three input bits (two significant bits and a previous carry bit) and two output bits. We arbitrarily assign symbols **A** and **B** to the two significant bit inputs and **Cin** for the Carry from the previous lower significant position, and **S** (for sum) and **Cout** (for Carry) to the two outputs.

Truth table for the Full Adder is shown below:

Input			Output Desired		Output Observed	
A	**B**	**Cin**	**S**	**Cout**	**S**	**Cout**
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		

Simplified Boolean function for the two outputs can be written from this truth table as:-

S = A ⊕ B ⊕ Bin CC

Cout = (A ⊕ B) C +A.B

The circuit diagram for the Full Adder is as under:

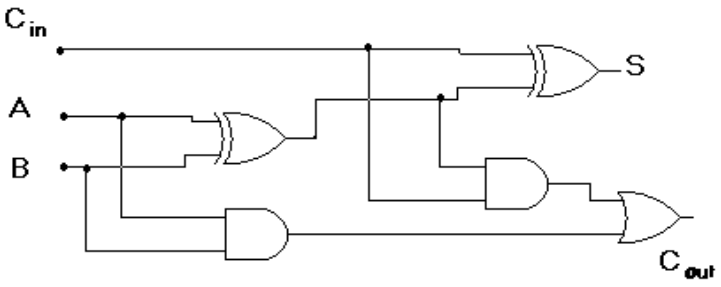


Figure 2: Full Adder comprising of two Half Adders and an OR gate.

PROCEDURE

1. Wire the circuit as per figure 2 above.

2. Use any three Logic Switches of the trainer (S2 to S9) for the input and any two of the LEDs of the trainer (L0 to L15) as output indication.

3. Connect +5V to pin 14 (Vcc) and Ground to pin 7 (GND) of the ICs.

4. By setting various combinations of the two switches verify that the output of the circuit is in accordance with the Truth Table shown above. Record your observation.

Conclusion & Comments

EXPERIMENT NO 4

IMPLEMENTATION OF 4-BIT PARALLEL ADDER USING IC 74283

OBJECTIVE

To study 4-bit parallel operation using IC 74283.

THEORY

Adders that are available in integrated circuit form are parallel binary adders. A 4-Bit parallel adder actually consists of four full adders connected in parallel. The carry output of each adder is internally connected to the carry input of the next higher order adder. Fig 5 shows the internal functional structure of 7483 IC in which 4 full adders are shown as separate entity. Figure is connection diagram for full adder function.

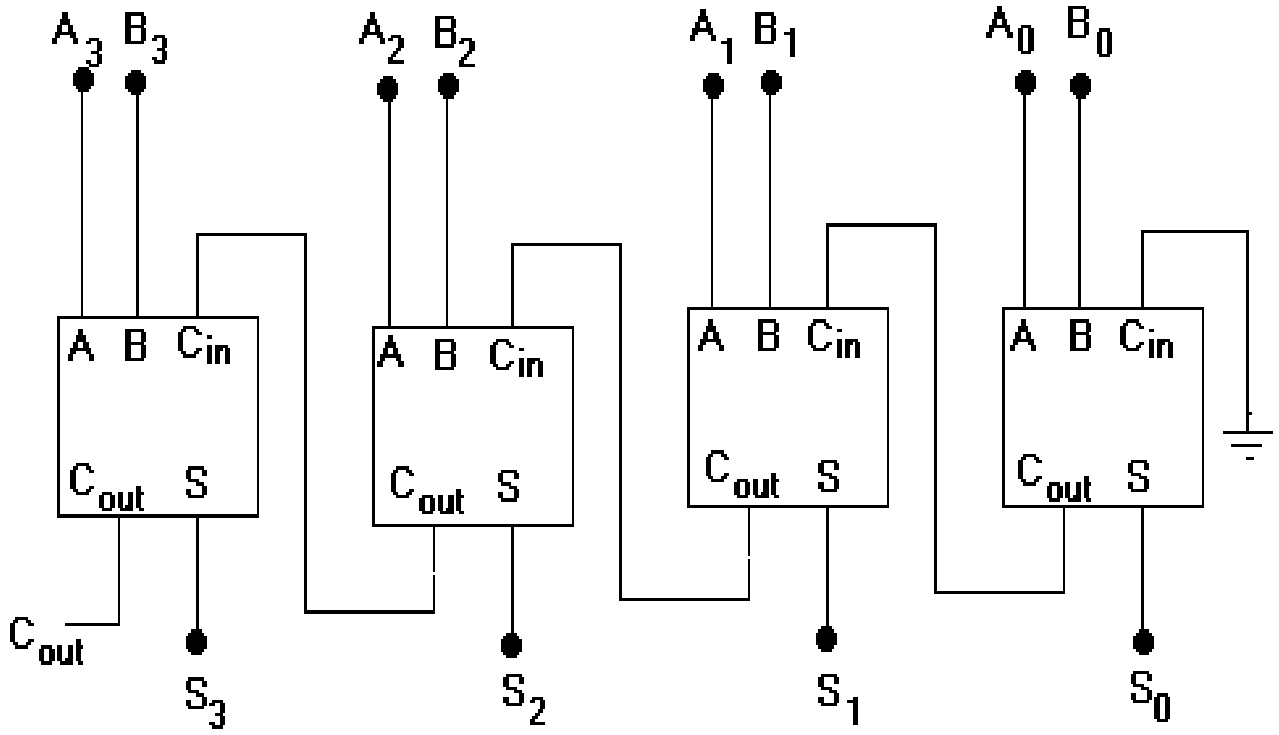


Figure 3: IC 7483 Internal Functional Structure

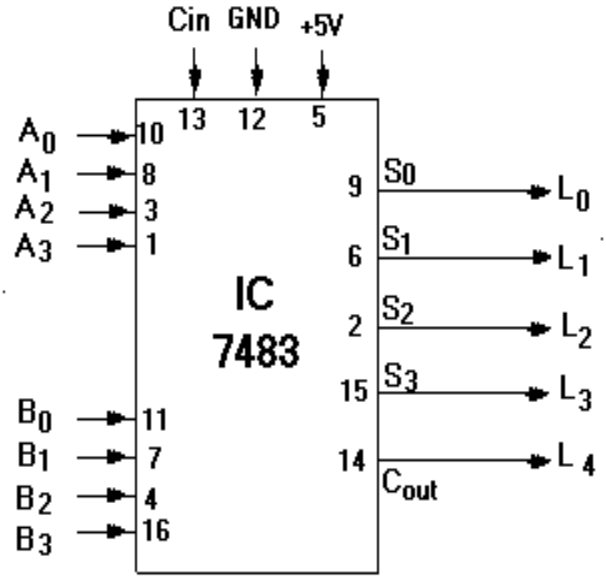


Figure 4: Connection Diagram for 4-Bit Parallel Adder.

PROCEDURE

1. Wire the circuit as per figure 6 above.

2. Use first four Logic Switches of the trainer for the inputs A0 to A3, and next four switches for inputs B0 to B3.

3. Connect Cin (pin 13) to GND (we are assuming initial carry to be zero).

4. Use first four LEDs of the trainer to indicate Sum outputs S0 to S3, and another LED to indicate status of the most significant carry bit cout.

5. Connect +5V to pin 5 (Vcc) and Ground to pin 12 (GND) of the ICs.

6. By setting various combinations of the two sets of input switches verify that the output of the circuit is in accordance with the Table shown below (only few of the possible additions have been shown here). Record your observation.

Table

Inputs										Desired Output						Observed Output				
Binary Augends				Binary Addend				Decimal Value for Ref		Binary Output					Decimal value for Ref	Binary Output				
A3	**A2**	**A1**	**A0**	**B3**	**B2**	**B1**	**B0**	A	B	*Co*	**S3**	**S2**	**S1**	**S0**	Sum	**Co**	**S3**	**S2**	**S1**	**S0**
0	0	0	1	0	0	0	1	1	1	0	0	0	1	0	2					
0	0	1	1	0	0	0	1	3	1	0	0	1	0	0	4					
0	1	0	0	0	0	1	0	4	2	0	0	1	1	0	6					
0	1	0	0	0	1	0	0	4	4	0	1	0	0	0	8					
0	1	0	1	0	1	0	0	5	4	0	1	0	0	1	9					
0	1	0	1	0	1	0	1	5	5	0	1	0	1	0	10					
1	0	0	0	0	1	0	0	8	4	0	1	1	0	0	12					
1	0	0	1	0	1	1	0	9	6	0	1	1	1	1	15					
1	0	0	1	1	0	0	0	9	8	1	0	0	0	1	17					
1	0	0	1	1	0	0	1	9	9	1	0	0	1	0	18					

Observations/Comments/Explanation of Results

(Please write in your own words the objectives and yours learning during the experiment. Also explain the results and comment on it.)

Conclusion & Comments

EXPERIMENT NO 05
IMPLEMENTATION OF HALF AND FULL SUBTRACTOR

OBJECTIVE

To learn Half and Full Subtraction Operations.

THEORY

Digital computers perform a variety of information-processing tasks. Among the basic functions encountered are various arithmetic operations. The most basic arithmetic operations are, no doubt, the addition and subtraction of binary digits (bit).

APPARATUS REQUIRED

Sl.No.	COMPONENT	SPECIFICATION	QTY.
1.	AND GATE	IC 7408	1
2.	X-OR GATE	IC 7486	1
3.	NOT GATE	IC 7404	1
4.	OR GATE	IC 7432	1
3.	IC TRAINER KIT	-	1
4.	PATCH CORDS	-	23

HALF SUBTRACTOR

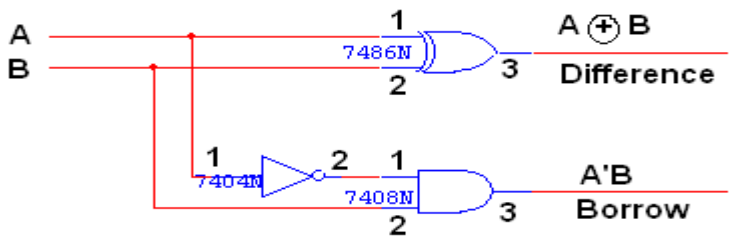
The half subtractor is constructed using X-OR and AND Gate. The half subtractor has two input and two outputs. The outputs are difference and borrow. The difference can be applied using X-OR Gate, borrow output can be implemented using an AND Gate and an inverter.

FULL SUBTRACTOR

The full subtractor is a combination of X-OR, AND, OR, NOT Gates. In a full subtractor the logic circuit should have three inputs and two outputs. The two half subtractor put together gives a full subtractor .The first half subtractor will be C and A B. The output will be difference output of full subtractor. The expression AB assembles the borrow output of the half subtractor and the second term is the inverted difference output of first X-OR.

LOGIC DIAGRAM:

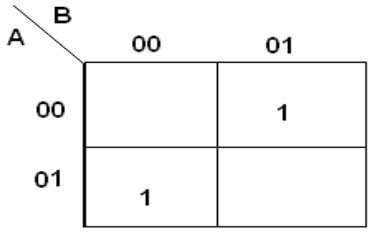
HALF SUBTRACTOR



TRUTH TABLE

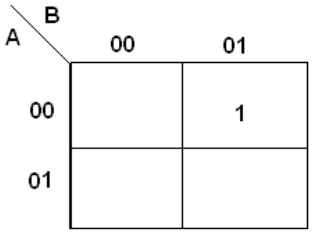
A	B	BORROW	DIFFERENCE
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

K-Map for DIFFERENCE:



DIFFERENCE = A'B + AB'

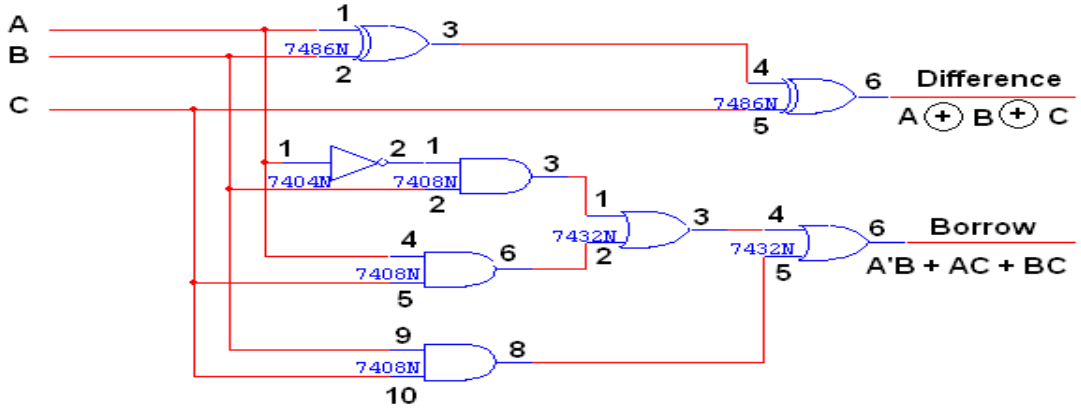
K-Map for BORROW:



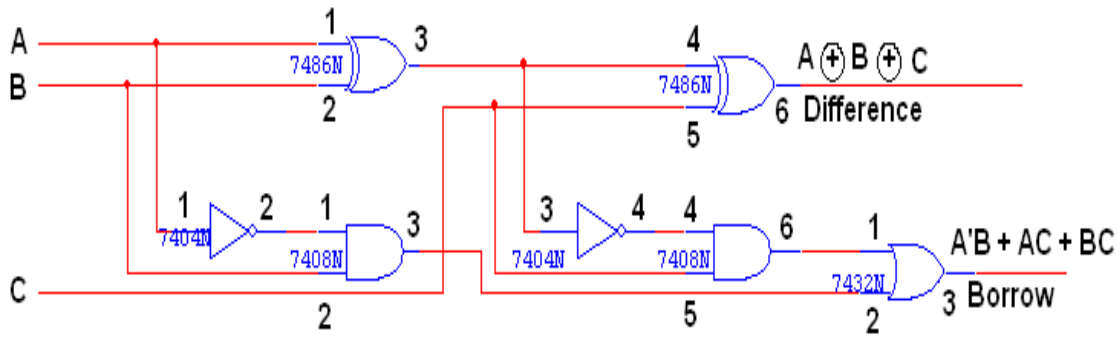
BORROW = A'B

LOGIC DIAGRAM

FULL SUBTRACTOR



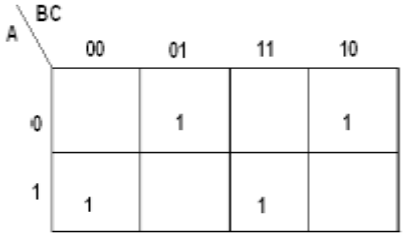
FULL SUBTRACTOR USING TWO HALF SUBTRACTOR



TRUTH TABLE

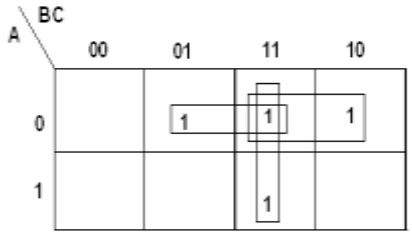
A	B	C	BORROW	DIFFERENCE
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-Map for Difference



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

K-Map for Borrow



Borrow = A'B + BC + A'C

PROCEDURE

1. Wire the circuit as per figure 2 above.
2. Use any three Logic Switches of the trainer (S2 to S9) for the input and any two of the LEDs of the trainer (L0 to L15) as output indication.
3. Connect +5V to pin 14 (Vcc) and Ground to pin 7 (GND) of the ICs.
4. By setting various combinations of the two switches verify that the output of the circuit is in accordance with the Truth Table shown above. Record your observation.

CONCLUSION & RESULTS

Conclusion & Comments

EXPERIMENT NO 06
IMPLEMENTATION OF ENCODER AND DECODER USING IC 74138 & 74148

COMPONENTS

1. Digital Logic Trainer
2. IC 74138 IC 74148

RELEVANT THEORY TOPICS

Combinational logic Decoders, Encoders (Refer Article 4.10, 4.11of Digital design, 4^{th} Edition by Morris Mano)

OBJECTIVE

To study Encoder and Decoder.

THEORY

An encoder circuit has more input lines and fewer output lines.

A decimal to BCD encoder (10 line to 4 line) will convert (at any one time) one active input out of ten to a BCD code output.

An octal-to-binary encoder (8 line to 3 line) will convert (at any one time) one-of-eight inputs to a binary code output

A decoder circuit few input lines and more output lines.

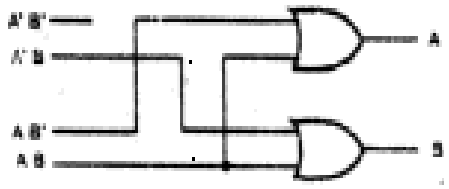
A binary-to-octal decoder converts 3 binary bits into 8 outputs (only one which will be active at one time)

A BCD decoder converts a 4-bit BCD code on into 10 output outputs (only one which will be active at one time).

A hexadecimal decoder converts a 4-bit binary code on the input to a 1-of-16 output.

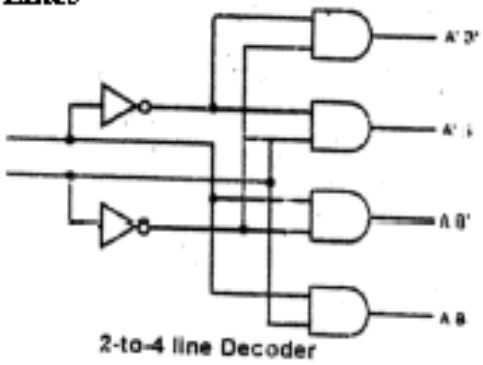
Decoders are often used in microprocessor systems to decode the address information from the microprocessor in order to select the correct memory chip.

More Input Lines **Less Output Lines**



4-to-2 line Encoder

Less Input Lines **More Output Lines**



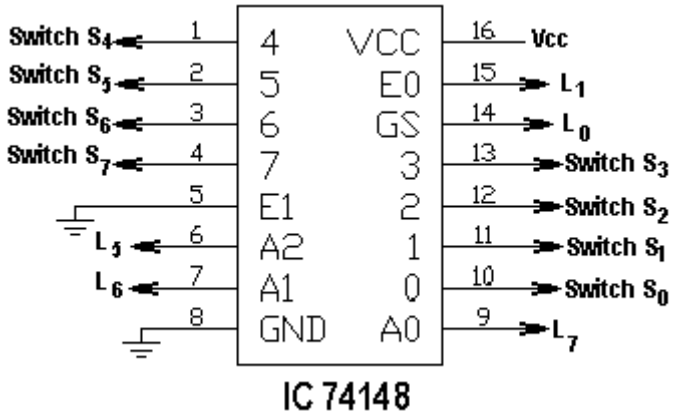
2-to-4 line Decoder

TASK 1: ENCODER

IC 74148 is a 8- Line-to- 3-line (octal to binary) Priority Encoder. It has 8 Inputs (0-7), an Enable Input **EI**, an Enable Output **EO**, 3 Output (A0-A2), and a **Gs** Output. Details as under:

- **A0-A2** outputs reflect a code that is equal to the highest valued active input.
- **Gs output** goes **low** any time any of the input goes low (this low signal is Used for interrupt request to CPU, when connected for the purpose).
- **EI and EO** are used for cascading more than one 74148 together.

Functional block diagram of IC 74148 is attached. Connection diagram and Truth table is shown below:



IC 74148

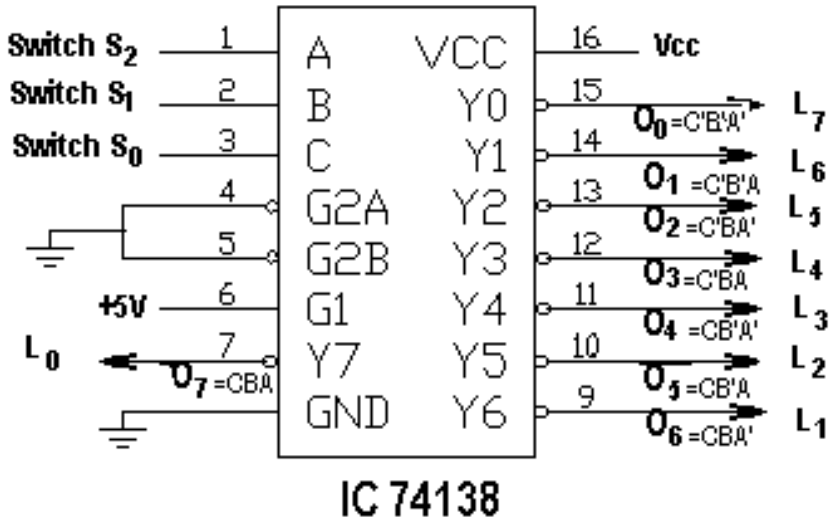
INPUT									OUTPUT				
EI	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**A2**	**A1**	**A0**	**Gs**	**Eo**
0	X	X	x	x	x	X	X	0				0	1
0	X	X	x	x	x	X	0	1				0	1
0	X	X	x	x	x	0	1	1				0	1
0	X	X	x	x	0	1	1	1				0	1
0	X	X	x	0	1	1	1	1				0	1
0	X	X	0	1	1	1	1	1				0	1
0	X	0	1	1	1	1	1	1				0	1
0	0	1	1	1	1	1	1	1				0	1

Wire the circuit as per figure above and fill in the blanks in the truth table.

TASK 2: DECODERS:

IC 74138 has been used as a decoder.

It has **3**-Select Inputs and 8- Data Outputs. Functional block diagram of IC 74138 is attached. **Note** that the IC 74138 has **Enable Inputs** which we will not use during decoder operation, therefore we will keep G1 as high and G2A and G2B as low so that the output of Enable gate remains always high and does not interfere with our desired result. Also **note** that the **output of the IC is active low**. Connection diagram and Truth table of the IC 74138 when used as decoder is shown below:



IC 74138

Note: Output of the IC 74138 is active low, so the output line having a Zero In the Truth Table will be selected

SELECT INPUT			DATA OUTPUT							
C	**B**	**A**	**O7**	**O6**	**O5**	**O4**	**O3**	**O2**	**O1**	**O0**
0	0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0	1
0	1	0	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	1	1	0	1	1	1	1
1	0	1	1	1	0	1	1	1	1	1
1	1	0	1	0	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1

Wire the circuit as per figure 1 above and verify the results.

Observations/Comments/Explanation of Results:

_____

_____

_____

_____

ASSIGNMENT

P1. Design priority encoder.

P2. Implement full adder and full subtractor using decoder.

Conclusion & Comments

EXPERIMENT NO 07

IMPLEMENTATION OF CODE CONVERTERS USING GATES

APPARATUS / COMPONENTS

- Digital Logic Trainer
- IC 7447, IC 7486, AND, OR and NOT gates
- 7-Segment Display
- Resistors 180 Ohms

Relevant Theory Topic: **Binary Codes, K-Map implementation (**Refer to Article 1.7, 3.2, 3.3, of Digital design, 4^{th} Edition by Morris Mano)

OBJECTIVE

To study BCD to Seven Segment Display Code, Gray Code and Excess-3 Code Conversion.

THEORY

The availability of large variety of codes for the same discrete elements of information results in the use of different codes by different digital systems. Sometimes it becomes necessary to use output one system as input to another system. A conversion circuit must be inserted between the two systems, if each uses different code for the same information. When a decimal number is decoded such that each digit of the number is represented by a 4-bit binary number, it is called a 8421 Binary Coded Decimal Code or more simply a BCD code. Here, ten out of sixteen possible combinations of the code are selected to represent decimal 0 through 9. Most commonly used BCD codes are given below:

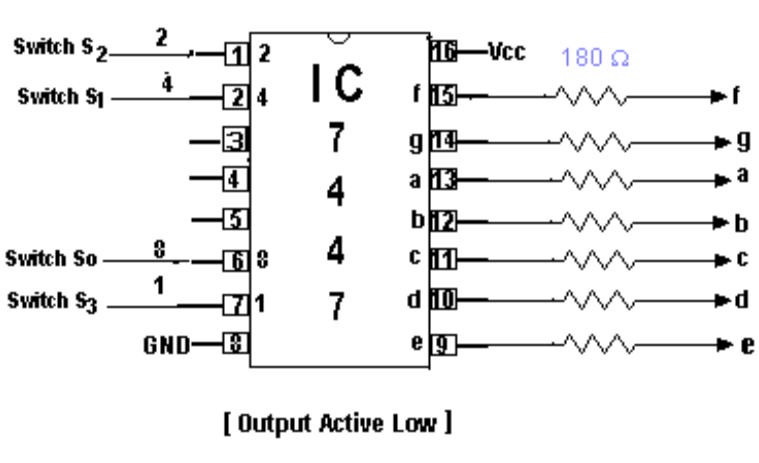
DECIMAL	BCD	GRAY CODE	EXCESS-3 CODE
0	0 0 0 0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 0 1 1	0 1 0 1
3	0 0 1 1	0 0 1 0	0 1 1 0
4	0 1 0 0	0 1 1 0	0 1 1 1
5	0 1 0 1	0 1 1 1	1 0 0 0
6	0 1 1 0	0 1 0 1	1 0 0 1
7	0 1 1 1	0 1 0 0	1 0 1 0
8	1 0 0 0	1 1 0 0	1 0 1 1
9	1 0 0 1	1 1 01	1 1 0 0

The important characteristics of the Gray code is that only one digit changes as we count from top to bottom; that is why it is termed as minimum change code. The Gray code is used for input and output devices. Primary use is in numeric input encoding applications, where we expect nonrandom input value change (i.e. value n changes either to n-1 or to n+1). Another decimal code that has been used in some old computers is Excess-3 code. Its code assignment is obtained from the corresponding value of BCD after the addition of 3. The code is used in many arithmetic circuits because it is self-complementing (i.e. the 9's complement value of the decimal number can be obtained by complementing each bit of the code).

TASK 1: BCD TO SEVEN SEGMENT DISPLAY CODE CONVERSION

Most Digital equipment has some means for displaying information in a form that can be understood readily by the user or operator. One of the simplest and most popular methods for displaying numerical digits uses a 7-segment configuration. To form decimal characters 0 through 9 and sometimes hex characters A through F. A BCD to 7-Segment Driver (IC 7447) is used to take four bit BCD input and provides the outputs that will pass current through the appropriate segment of the display to generate desired output/ number. Truth Table for Active High and Active Low cases are shown below:

INPUT –**BCD**				Decimal	Output- **Seven Segment Decoder (Active Low -IC 7447)**							**Display Output**
S0	**S1**	**S2**	**S3**		a	b	c	d	e	f	g	
0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	1	1	0	0	1	1	1	1	1
0	0	1	0	2	0	0	1	0	0	1	0	2
0	0	1	1	3	0	0	0	0	1	1	0	3
0	1	0	0	4	1	0	0	1	1	0	0	4
0	1	0	1	5	0	1	0	0	1	0	0	5
0	1	1	0	6	1	1	0	0	0	0	0	6
0	1	1	1	7	0	0	0	1	1	1	1	7
1	0	0	0	8	0	0	0	0	0	0	0	8
1	0	0	1	9	0	0	0	1	1	0	0	9



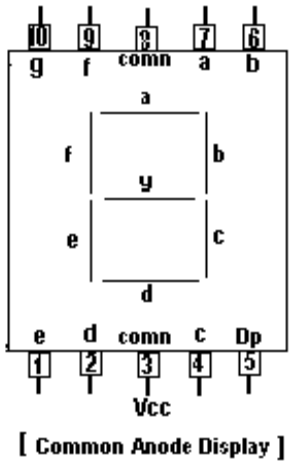


Figure 1: BCD to Seven Segment Converter Circuit.

The segments of Seven Segment display are made of LEDs. Depending on the arrangements of the LEDs, the display could be Common Anode or Common Cathode type. We are using common anode type of display, which would require that either pin 3 or pin 8 is connected to Vcc and the input is active low.

PROCEDURE

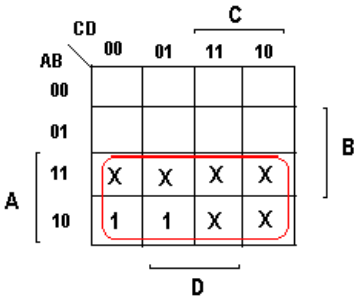
1. Wire the circuit as per figure 1 above. Connect pin 3 or pin 8 to Vcc.
2. By setting various combinations of the switches verify the result.

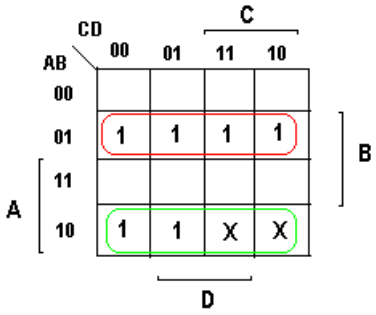
TASK 2: BCD TO GRAY CODE CONVERSION

The bit combination for the BCD and Gray code are listed in the table below. Since each code uses four bits to represent a decimal digit, there must be four input variables and four output variables.

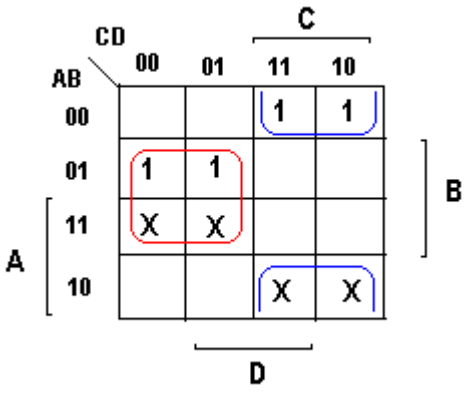
Input BCD				**Output Gray Code**				**Observed Output**			
A	**B**	**C**	**D**	**W**	**x**	**y**	**Z**	**W**	**x**	**y**	**z**
0	0	0	0	0	0	0	0				
0	0	0	1	0	0	0	1				
0	0	1	0	0	0	1	1				
0	0	1	1	0	0	1	0				
0	1	0	0	0	1	1	0				
0	1	0	1	0	1	1	1				
0	1	1	0	0	1	0	1				
0	1	1	1	0	1	0	0				
1	0	0	0	1	1	0	0				
1	0	0	1	1	1	0	1				

With the help of K- Map, the simplified output can be obtained as:

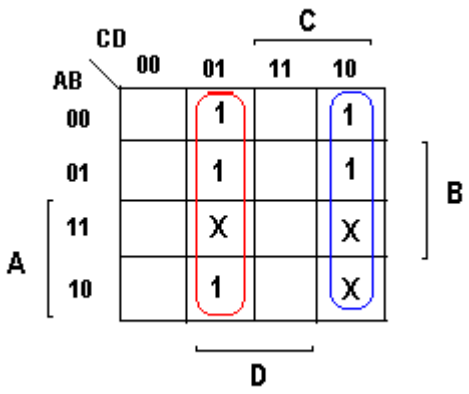




w = A



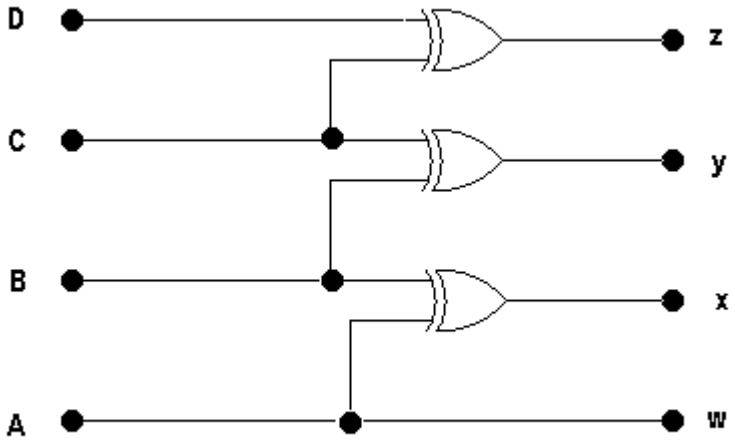
x=A'B+AB'=A ⊕ B



y = BC'+B'C=B ⊕ C

z = CD'+C'D=C ⊕ D

Figure 2: Logic Diagram for BCD to Gray code converter.



PROCEDURE

1. Wire the circuit as per Fig. 2 above.
2. Use any four Logic Switches of the trainer (S2 to S9) for the input and any four of the LEDs of the trainer (L0 to L15) as output indication.
3. Connect +5V to pin 14 (Vcc) and Ground to pin 7 (GND) of the ICs.
4. By setting various combinations of the two switches verify that the output of

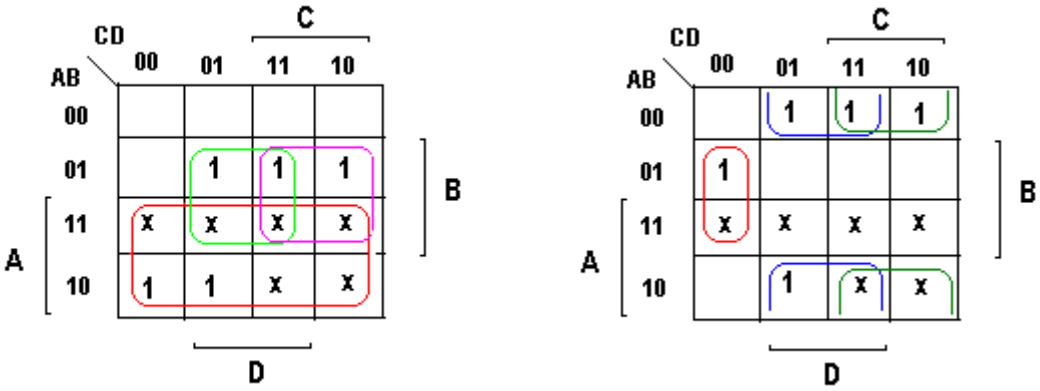
the circuit is in accordance with the Truth Table shown above. Record your observation.

TASK 3: BCD TO EXCESS-3 CODE CONVERSION.

The bit combination for the BCD and excess-3 codes is listed in the table below. Since each code uses four bits to represent a decimal digit, there must be four input and four output variables.

Input BCD				**Output Excess-3 Code**				**Observed Output**			
A	**B**	**C**	**D**	**w**	**x**	**y**	**Z**				
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				

With the help of K- Map, the simplified output can be obtained as:



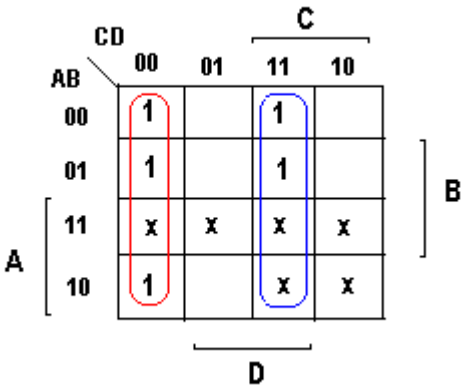
(1)

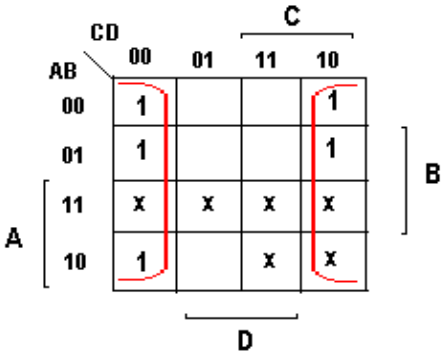
w = A + BC + BD = A + B(C+D)

(2)

x = B'C + B'D + B

=B'(C+D)+BC'D'





(3) **y = CD + C'D' = CD + (C+D)'**

(4) **z = D'**

PROCEDURE

1. Wire the circuit as per above Fig. 3 and verify the results.

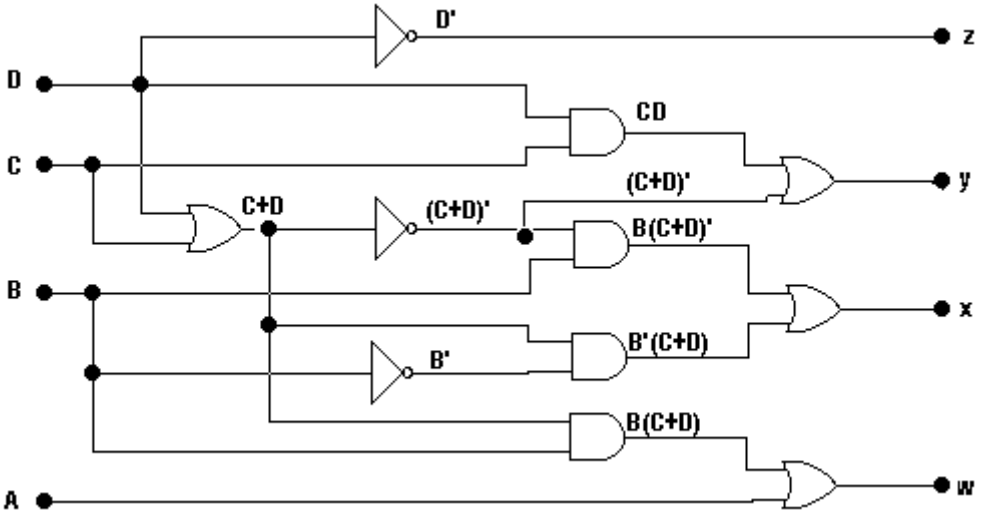


Figure 3: Logic Diagram for BCD to excess-3 code converter.

LAB ASSIGNMENT

1. Design BCD to 2421 convertor.
2. Design a circuit which calculates 9’s complement of a four bit number.

Conclusion & Comments

EXPERIMENT NO 08

IMPLEMENTATION OF MULTIPLEXER AND DEMULTIPLEXER USING IC74151& IC74138

COMPONENTS

1. Digital Logic Trainer 2. IC 74151 & 74138

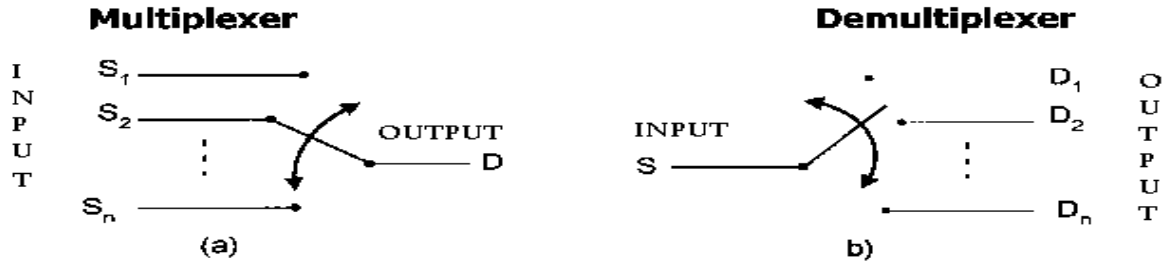
RELEVANT THEORY TOPICS

Combinational logic Decoders, Encoders Multiplexer, De-Multiplexers (Refer Chapter 4.9-4.11 Digital design, 4^{th} Edition by Morris Mano)

OBJECTIVE

To study Multiplexer and Demultiplexer.

THEORY



(a) A multiplexer and (b) a demultiplexer modeled as mechanical switches

MULTIPLEXER

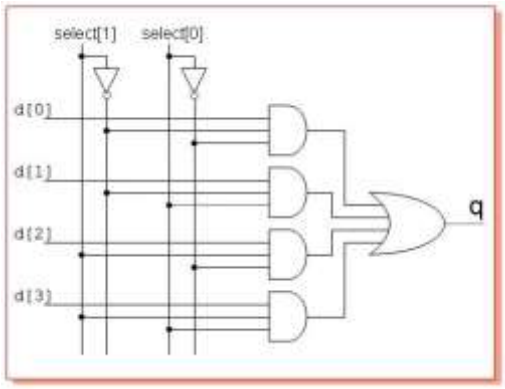
1. The multiplexer circuit is used to place two or more digital signals (from two or more sources) onto a single line, by placing them there at different time interval technically it is known as time-division- multiplexing).

2. The multiplexer (also known as data selector) will select data from several transmission lines to be gated to the single output transmission line.

3. The multiplexer will have a number of control inputs that are used to select the appropriate data channel for input.

4. The number of data inputs is equal to 2^n where n is the number of control selecting leads.

5. A multiplexer can be used to convert parallel data to serial data.



DEMULTIPLEXER

1. A demultiplexer (data distributor) will receive information from a single line and selectively transmits it to several output lines/channels (one at a time).

2. Demultiplexer has several control select lines which are used to determine (or select) the output transmission path.

3. The number of data output lines is 2^n, where N is the number of control select leads.

4. Demultiplexers are used to convert serial data to parallel data.

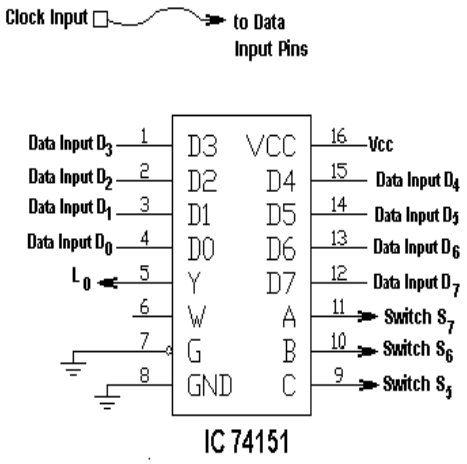
TASK 1: MULTIPLEXER

IC 74151 is a 8-to-1-Line Multiplexer. It has following features:-

1. 8 Data Inputs (DO-D7).

2. Three Select Inputs (A,B,C). 3. An Enable (or Strobe) G

4. A one bit output Y (and its complement W)



PROCEDURE

1. Wire the circuit as per figure above.
2. Connect "Clock Input" (very low frequency) to Input pins of the IC (D0 – D1) and see if the Output LED is pulsating. Confirm your finding on the truth table.

Functional block diagram of IC 74151 is attached.

Connection diagram and Truth table is shown below:

Select			**Strobe**	**Output**	**Output**
C	**B**	**A**	**G** (or S)	**Y**	**Observed**
0	0	0	0	Output **Y** is linked with input present at **D0**	
0	0	1	0	Output **Y** is linked with input present at **D1**	
0	1	0	0	Output **Y** is linked with input present at **D2**	
0	1	1	0	Output **Y** is linked with input present at **D3**	
1	0	0	0	Output **Y** is linked with input present at **D4**	
1	0	1	0	Output **Y** is linked with input present at **D5**	
1	1	0	0	Output **Y** is linked with input present at **D6**	
1	1	1	0	Output **Y** is linked with input present at **D7**	

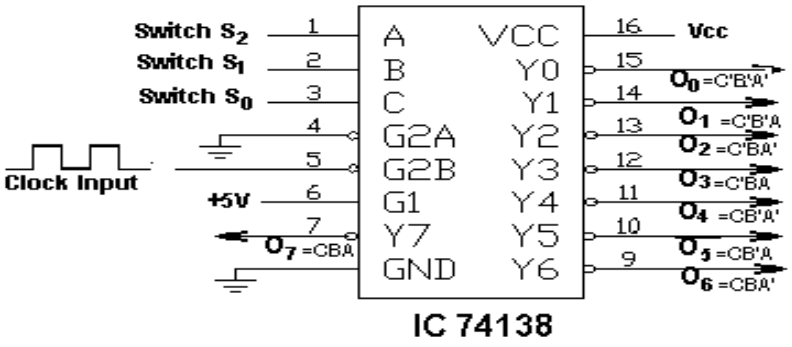
TASK 2: DEMULTIPLEX

A 1-Line-to-8-Line demultiplexer distributes one input to 8 output lines. IC 74138 which was used as a decoder in the last experiment will be used here as Demultiplexer. The only difference between the previous circuit and present circuit will be addition of an INPUT (through Enable AND gate) to the 4^{th} pin of all the 8 NAND gates. The A, B and C inputs will serve as SELECT input (to select a particular output line).

Note that the Enable Inputs of IC 74138 was not used during decoder operation. We will now

use **G2B** pin of the IC for **Data/Signal Input**. We therefore need to keep pins G1 as high and G2A as low, so that the Input Data/Signal remains present at output of Enable gate and consequently on the 4th input pin of all the 8 NAND gates.

Connection diagram and Truth table of the IC 74138 when used as demultiplexer is shown below. Clock signal (very low frequency) has been used as Input (so that blinking of the LEDs can be observed):



Note: Output of the IC 74138 is active low, so the output line having a Zero in the Truth Table will be selected

Input			**Output**							
C	**B**	**A**	**O7**	**O6**	**O5**	**O4**	**O3**	**O2**	**O1**	**O0**
0	0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0	1
0	1	0	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	1	1	0	1	1	1	1
1	0	1	1	1	0	1	1	1	1	1
1	1	0	1	0	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1

TASK 3: DATA COMMUNICATION USING MULTIPLEXER & DEMULTIPLEXER.

Multiplexer IC 74151 and Demultiplexer IC 74138 have been utilized to demonstrate single-line data communication. The 3-bit select code will determine which data input will be steered to the **Y** output of the Demultiplexer.

Select			Applied Signal at Data Input Pin	Observed Signal Out at Output Pin	When Clock Signal is Applied At All of D Pins of MUX
C	**B**	**A**	**D**	**Y**	**Observed Output at Pin of DEMUX**
0	0	0	**D0**	**Y0**	
0	0	1	**D1**	**Y1**	
0	1	0	**D2**	**Y2**	
0	1	1	**D3**	**Y3**	
1	0	0	**D4**	**Y4**	
1	0	1	**D5**	**Y5**	
1	1	0	**D6**	**Y6**	
1	1	1	**D7**	**Y7**	

PROCEDURE

1. Wire the circuit as per figure above and verify result first by giving clock signal to One Input pin at a time of the IC (D0 – D1) and then to all the pins simultaneously.

Conclusion & Comments

EXPERIMENT NO 09
VERIFICATION OF LATCH AND FLIP FLOP OPERATION USING GATES AND FLIP FLOP'S IC

APPARATUS / COMPONENTS

Digital Logic Trainer.

IC 7402, 7400, 7404 , 7410 , 7474 & 7475

Relevant Theory Topics: Synchronous Sequential Logic, Latches/Flip-flops

Refer Chapter 05 Digital design, 4^{th} Edition by Morris Mano)

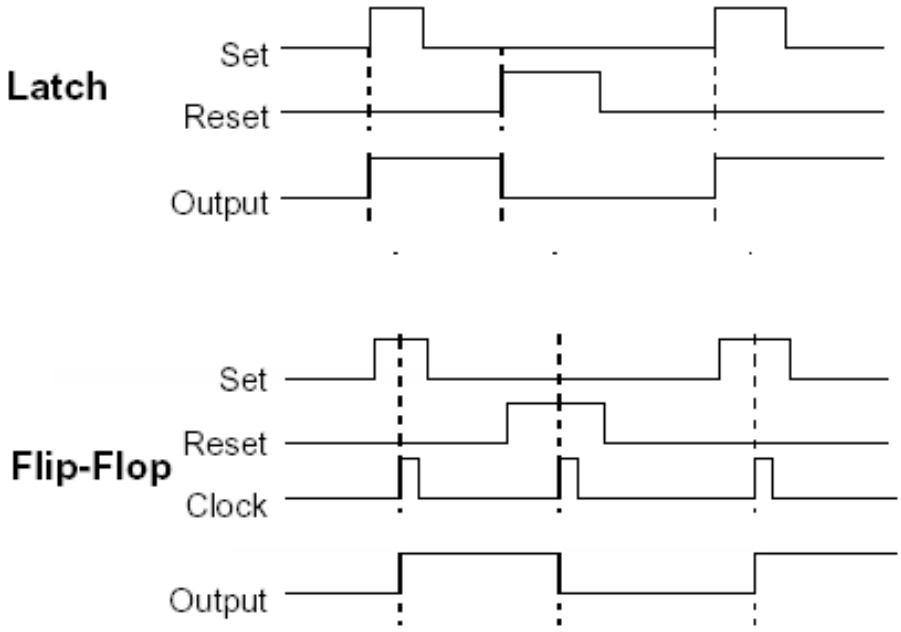
OBJECTIVE

To learn about various types of Latches and Flip-Flops

THEORY

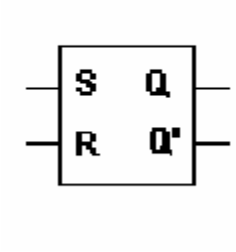
a. A Flip Flop is a logic circuit that has two stable states Low or High. Enable input signal may be used to enable or disable a flip flop. Clock signal is used to synchronize operations of flip -flops. Most (if not all) of the system output can change state only when the clock makes a transition.

b. Latches are a form of Flip Flop, which do not require clock pulse to latch or hold data present at its input.



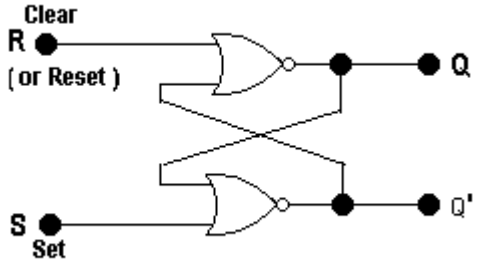
TASK 1: SR (or RS or SC) Latch

The SR is the simplest form of Flip Flop or Latch. It can be constructed from NOR gates or NAND gates. Standard logic symbol of SR flip flop and its truth table is given below:-



Input		Output
S Set	R Clear(Reset)	Q
0	0	Last State
1	0	1
0	1	0
1	1	? (forbidden)

NOR Gate SR Latch:



Input		Output	Action	Observation
S Set	R Clear(Reset)	Q		
1	0	1	Set	
0	0	Last State	No Change	
0	1	0	Clear	
0	0	Last State	No Change	
1	1	?	Forbidden	

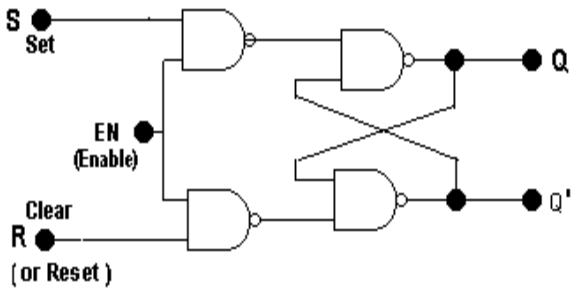
PROCEDURE

Wire the circuit as per figure above and verify the result.

NOTE: NAND Gate SR Latch has active low input, hence its truth table is different from the standard i.e a **low** at the set terminal will set the latch.

TASK 2: GATED FLIP FLOPS Gated SR Flip Flop :

Works only when Enable is High.

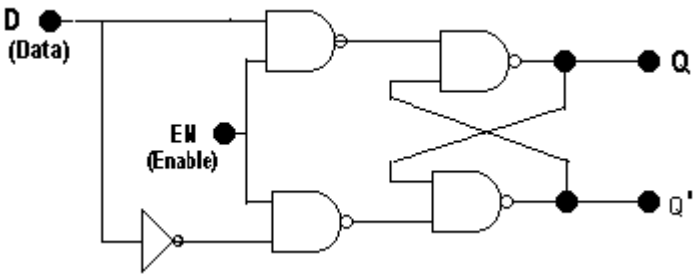


Input			Output	Action	Observation
EN (Enable)	S Set	R Clear(Reset)	Q		
1	1	0	1	Set	
1	0	0	Last State	No Change	
1	0	1	0	Clear	
1	0	0	Last State	No Change	
1	1	1	?	Forbidden	
0	X	X	Last State	No Change	

PROCEDURE

Wire the circuit as per figure above and verify the result.

GATED D – FLIP FLOP



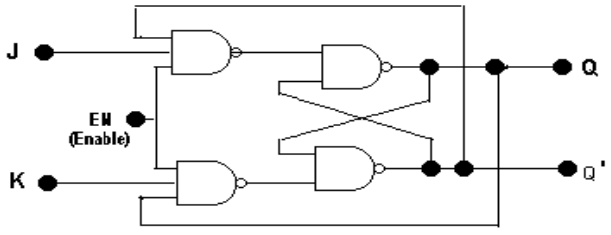
Input		Output	Action	Observatio
EN (Enable)	D	Q		
1	1	1	Set	
1	0	0	Clear	
0	X	Last State	No Change	

PROCEDURE

Wire the circuit as per figure above and verify the result.

GATED J-K FLIP FLOP

Input			Output	Action	Observation
EN (Enable)	J Set	K Clear(Reset)	Q		
1	1	0	1	Set	
1	0	1	0	Clear	
1	0	0	Last State	No Change	
1	1	1	$\overline{Q_n}$	Toggle	
0	X	X	Last State	No Change	

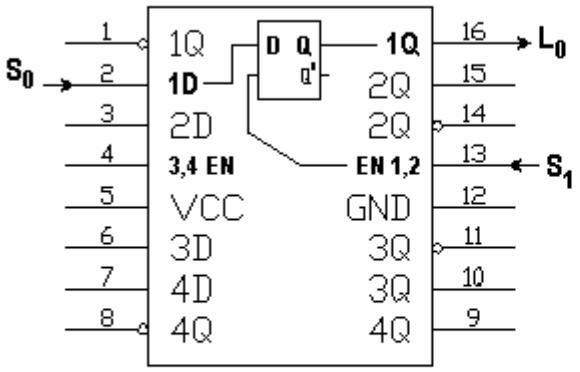


PROCEDURE

Wire the circuit as per figure above and verify the result

TASK 3: D-LATCH AND D-FLIP FLOP OPERATIONS

Figure: 1

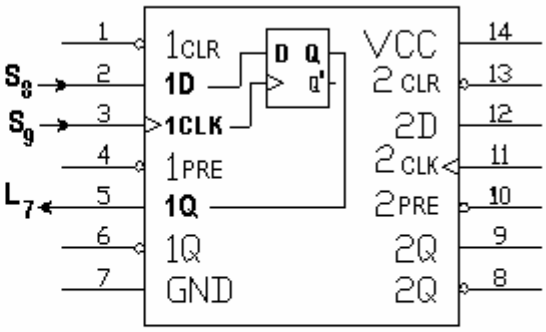


IC 7475

D — Latch

(Quad)

Figure: 2



IC 7474

D-Flip Flop (Edge Triggered)

(Dual)

PROCEDURE

1. Wire the circuit as per figure 1 & 2 above
2. Connect +5V to Vcc and Ground to pin GND of the ICs.

VERIFICATION – D LATCH:

1. Verify that the data at the input terminal is reflected at the output only when Enable (EN) input is high.

2. Verify that for the duration that the Enable input remains high, all changes in the data input are reflected at the output.

Input	Enable	Output
D	EN	Q
1	0	
0	0	
1	1	
0	1	

VERIFICATION – D FLIP FLOP:

1. Verify that the data at the input terminal is reflected at the output only during Positive going edge of the Clock pulse.

2. Verify that the duration of the Clock pulse has nothing to do with data transfer in case of Flip Flop.

Input	Clock	Output
D	Clk	Q
1	↑ Positive edge triggered	
0		
0	↑ Positive edge triggered	
1		

LAB ASSIGNMENT

P1. Construct SR latch using NAND gates

P2.What changes would you make in the NAND gate latch shown above, so it behaves exactly like a NOR gate S-R latch.

ANS.

_____
_____
_____

Conclusion & Comments

EXPERIMENT NO 10

IMPLEMETATION OF 4-BIT SYNCHRONOUS BINARY COUNTER

APPARATUS / COMPONENTS

Digital Logic Trainer.

IC 74HC163

Relevant Theory Topics: Asynchronous Counter Operation (8-1), Synchronous Counter Operation (8-2), Up/Down Synchronous Counters (8-3).

OBJECTIVE

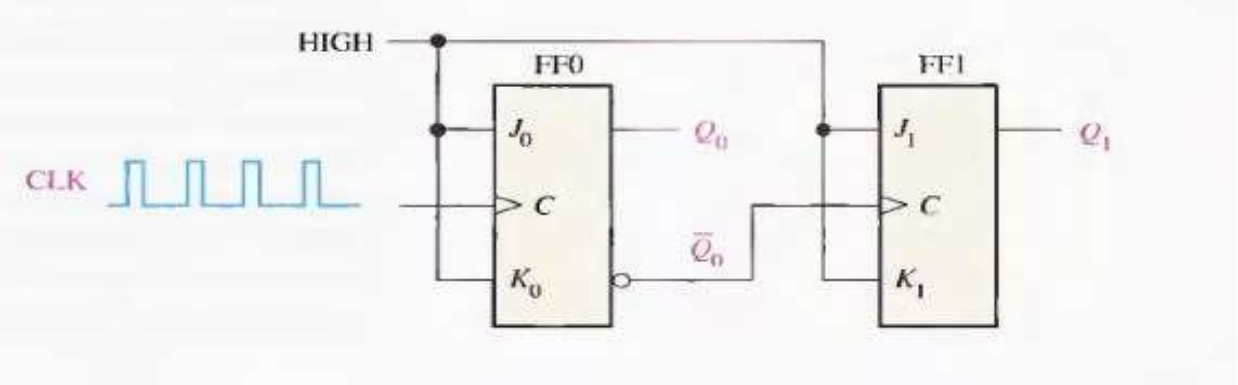
To understand and analyze various types of Counter, difference b/w synchrounous and asynchronous counters. To learn and understand the operation of counters timing diagram, desining and implementaion of counters.

THEORY

A counter is a sequential logic circuit that goes through a prescribed sequence of states upon the application of input pulses. The prescribed sequence can be a binary sequence or any other sequence. A counter that goes through 2N (N is the number of flip-flops in the series) states is called a binary counter. The modulus of a counter is the number of different states it is allowed to have. Counter modulus is normally 2N unless controlled by a feedback circuit which limits the number of possible states (an example being the decimal counter). Counters are very widely used in almost all computers and other digital electronic systems. There are two major categories of counters: asynchronous counters and synchronous counters.

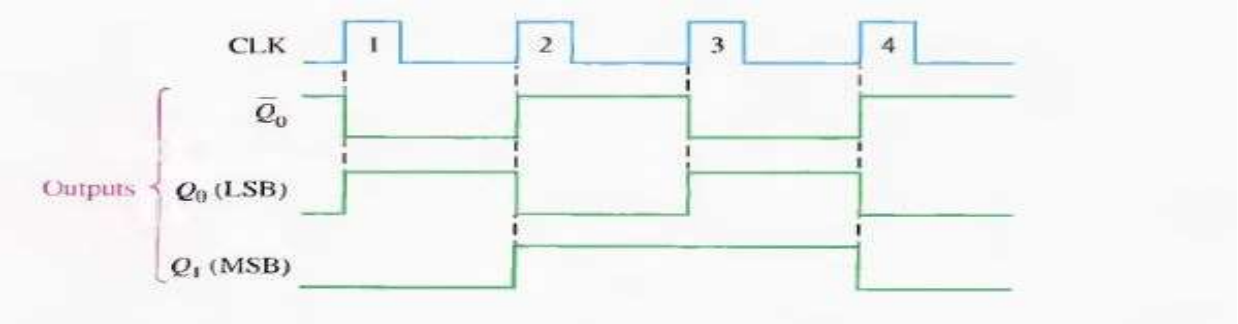
A 2-Bit Asynchronous Binary Counter

Following figure 1 shows a 2-bit counter connected for asynchronous operation. Notice that the clock (CLK) is applied to the clock input of only the first flop-flop, FFO, whic is always the least significant bit (LSB). The second flip-flop, FFI, is triggered by the Qo out-put of FFO. FFO changes state at the positive-going edge of each clock pulse. But FFI changes only when triggered by a positive-going transition of the Qo output ofFFO. Because of the inherent propagation delay tie through a flip-flop, a transition of the input clock pulse (CLK) and a transition of the Qo output of FFO can never occur at exactly the same time. Therefore, the two flip-flops are never simultaneously triggered, so the counter oper- ation is asynchronous.



Timing Diagram

The Timing Diagram Let's examine the basic operation of the asynchronous counter of above figure by applying four clock pulses to FFO and observing the Q output of each tlip- flop. Figure below illustrates the changes in the state ofthe flip-flop outputs in response to the clock pulses. Both flip-flops are connected for toggle operation (j = I. K = I) and are assumed to be initially RESET (Q LOW).



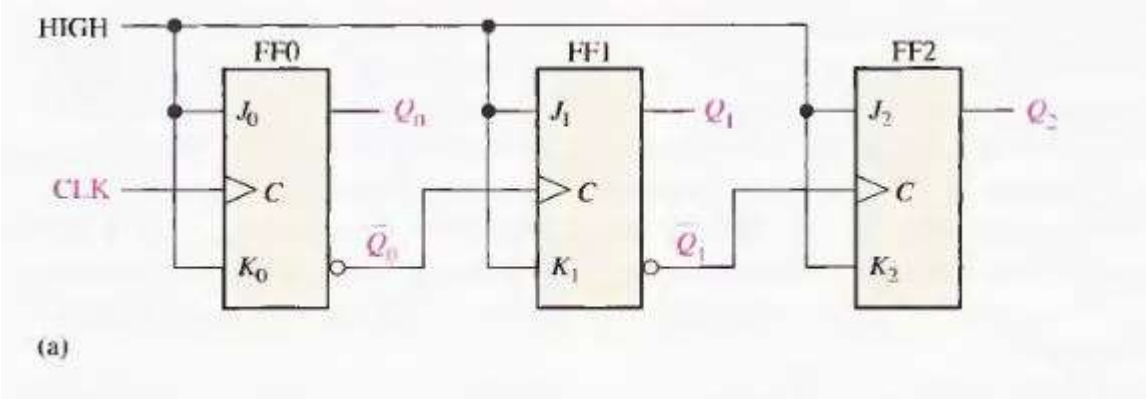
Binary State Sequence for the Counter

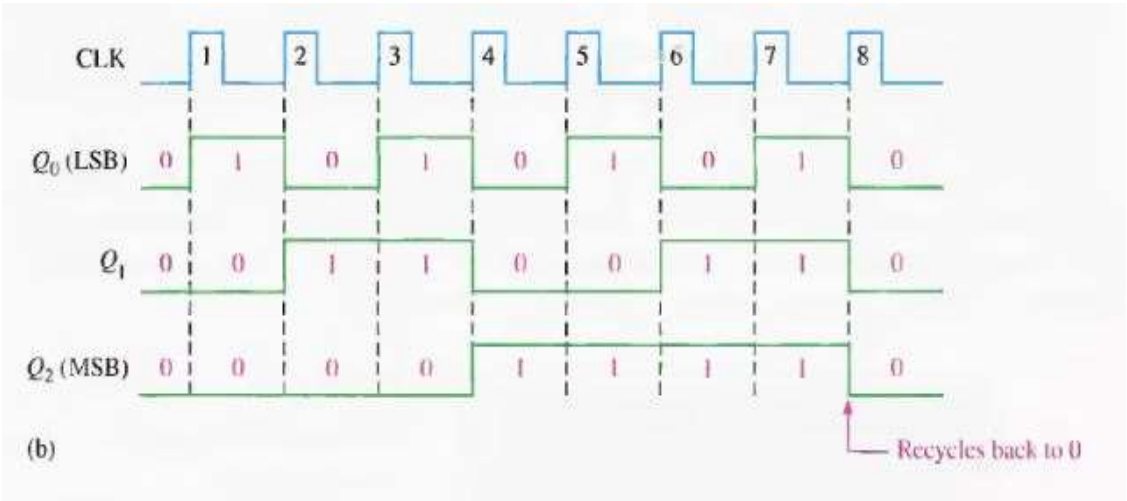
CLOCK PULSE	Q_1	Q_0
Initially	0	0
1	0	1
2	1	0
3	1	1
4 (recycles)	0	0

Since it goes through a binary sequence, the counter in Figure 1 is a binary counter. It actualIy counts the number of clock pulses up to three, and on the fourth pulse it recycles to its original state (Qo = 0, Q, = 0). The term recycle is commonly applied to counter operation; it refers to the transition of the counter from its final state back to its original state.

A 3-Bit Asynchronous Binary Counter

The state sequence for a 3-bit binary counter is listed in Table given below, and a 3-bit asynchronous binary counter is shown in Figure (a). The basic operation is the same as that of the 2-bit counter except that the 3-bit counter has eight states, due to its three flip-flops. A timing diagram is shown in Figure (b) for eight clock pulses. Notice that the counter progresses through a binary count of zero through seven and then recycles to the zero state. This counter can be easily expanded for higher count, by connecting additional toggle flip-flops.

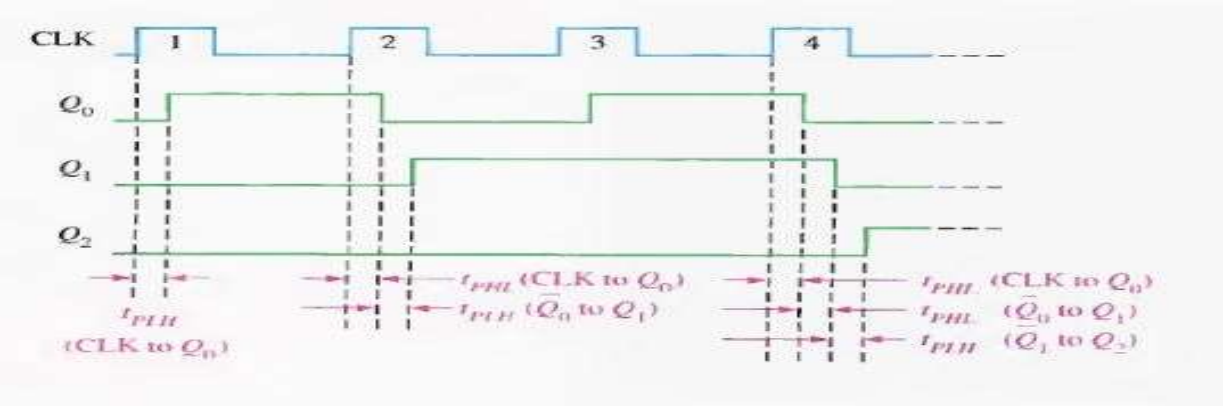




Propagation Delay

Asynchronous counters are commonly referred to as ripple counters for the following reason: The effect of the input clock pulse is first '"felt" by FFO. This effect cannot get to FFI immediately because of the propagation delay through FFO. Then there is the propagation delay through FFI before FF2 can be triggered. Thus, the effect of an input clock pulse "ripples" through the counter taking some time, due to propagation delays, to reach the last flip-flop.

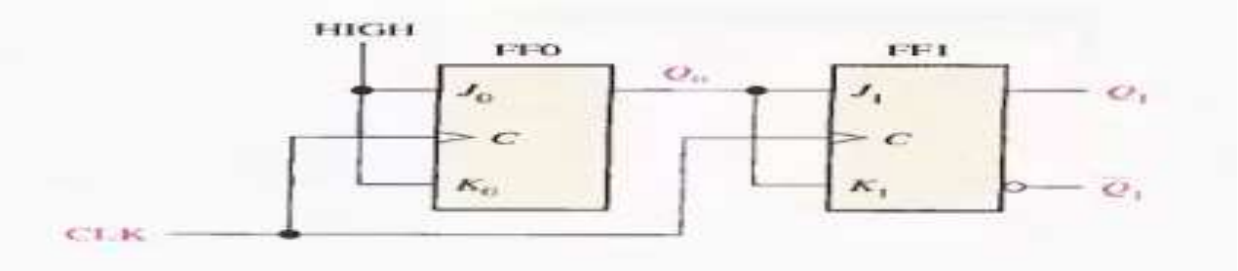
Propagation delays in a 3-bit asynchronous (ripple-clocked) binary counter is given below:



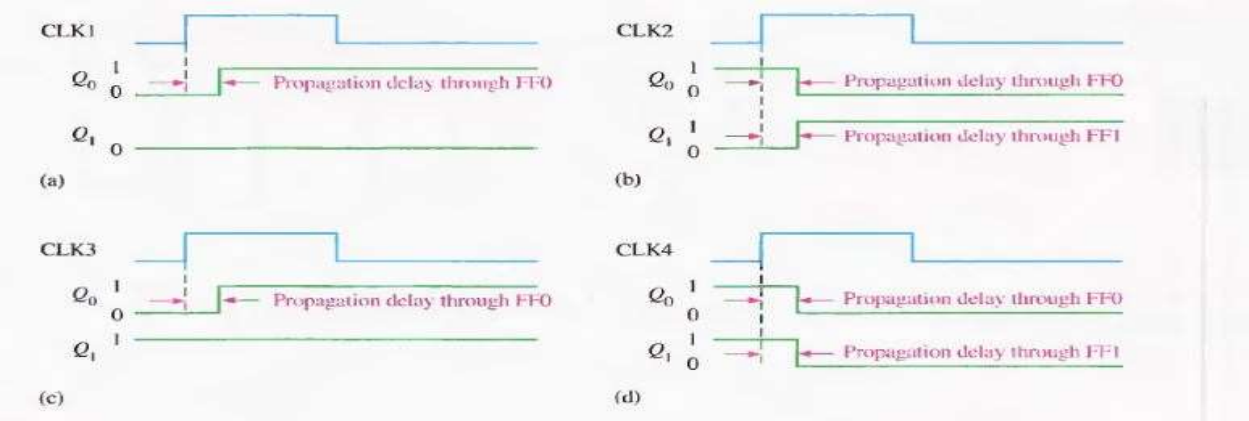
To illustrate, notice that all three flip-flops in the counter of above Figure change state on the leading edge of CLK4. This ripple clocking effect is shown in Figure 8—4 for the first four clock pulses, with the propagation delays indicated. The LOW-to-HIGH transition of Qo occurs one delay time (tPLH) after the positive-going transition of the clock pulse. The LOW to HIGH transition of Q, occurs one delay time (tPLH) after the positive-going transition of Qo. The LOW-to-HGH transition of Q2 occurs one delay time (tPLH) after the positive-going transition of QJ. As you can see. FF2 is not triggered until two delay times after the positive-going edge of the clock pulse, CLK4. Thus, it takes three propagation delay times for the effect of the clock pulse, CLK4, to ripple through the counter and change Q2 from LOW to HIGH.

A 2-Bit Synchronous Binary Counter

Figure given below shows a 2-bit synchronous binary counter. Notice that an arrangement different from that for the asynchronous counter must be used for the J) and K. inputs of FFI in order to achieve a binary sequence.



The operation of this synchronous counter is as follows: First, assume that the counter is initially in the binary 0 state: that is both flip-flops are RESET. When the positive edge of the first clock pulse is applied, FFO will toggle and Qo will therefore go HIGH. What happens to FFI at the positive-going edge of CLK1? To find out, let's look at the input conditions of FFI. Inputs 1 1 and KJ are both LOW because Qo, to which they are connected, has not yet gone HIGH. Remember, there is a propagation delay from the triggering edge of the clock pulse until the Q output actually makes a transition. So, J = 0 and K = 0 when the leading edge of the first clock pulse is applied. This is a no-change condition, and therefore FFI does not change state. A timing detail of this portion of the counter operation is shown in following Figure (a).
(The propagation delays of both flip-flops are assumed to be equal).

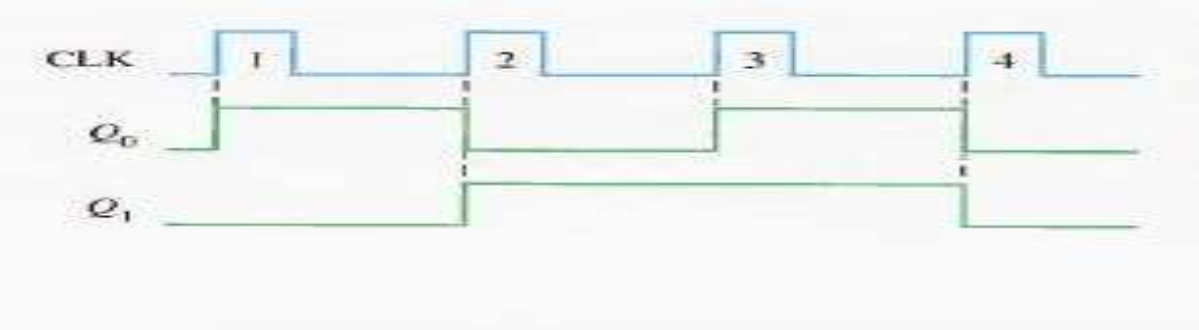


After CLK1, Qo = I and Q, = 0 (which is the binary I state). When the leading edge of CLK2 occurs, FFO will toggle and Qo will go LOW. Since FFI has a HIGH (Qo = 1) on its 1, and KI inputs at the triggering edge of this clock pulse, the flip-flop toggles and QJ goes HIGH. Thus, after CLK2, Qo = 0 and Q] = I (which is a binary 2 state). The timing detail for this condition is shown in Figure (b).

When the leading edge of CLK3 occurs. FFO again toggles to the SET state (Qo = I), and FFI remains SET (QI = I) because its J 1 and KJ inputs are both LOW (Qo = 0). After this triggering edge, Qo = I and QI = I (which is a binary 3 state). The timing detail is shown in Figure ©.

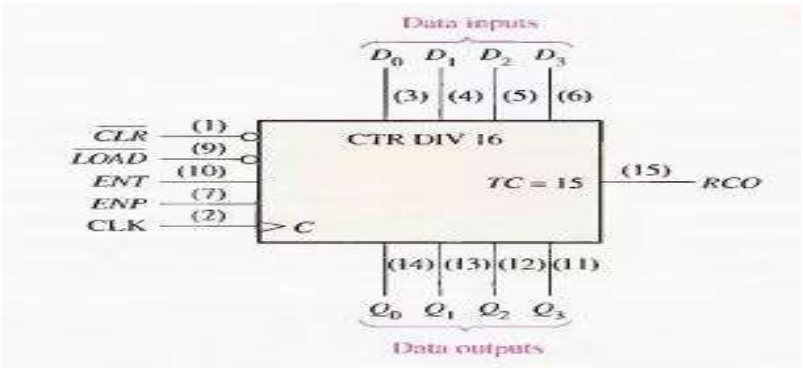
Finally, at the leading edge of CLK4, Qo and QI go LOW because they both have a toggle condition on their 1 and K inputs. The timing detail is shown in Figure (d). The counter has now recycled to its original state, binary 0.

Timing diagram for the 2-Bit Synchronous Counter



THE 74HC163 4-BIT SYNCHRONOUS BINARY COUNTER

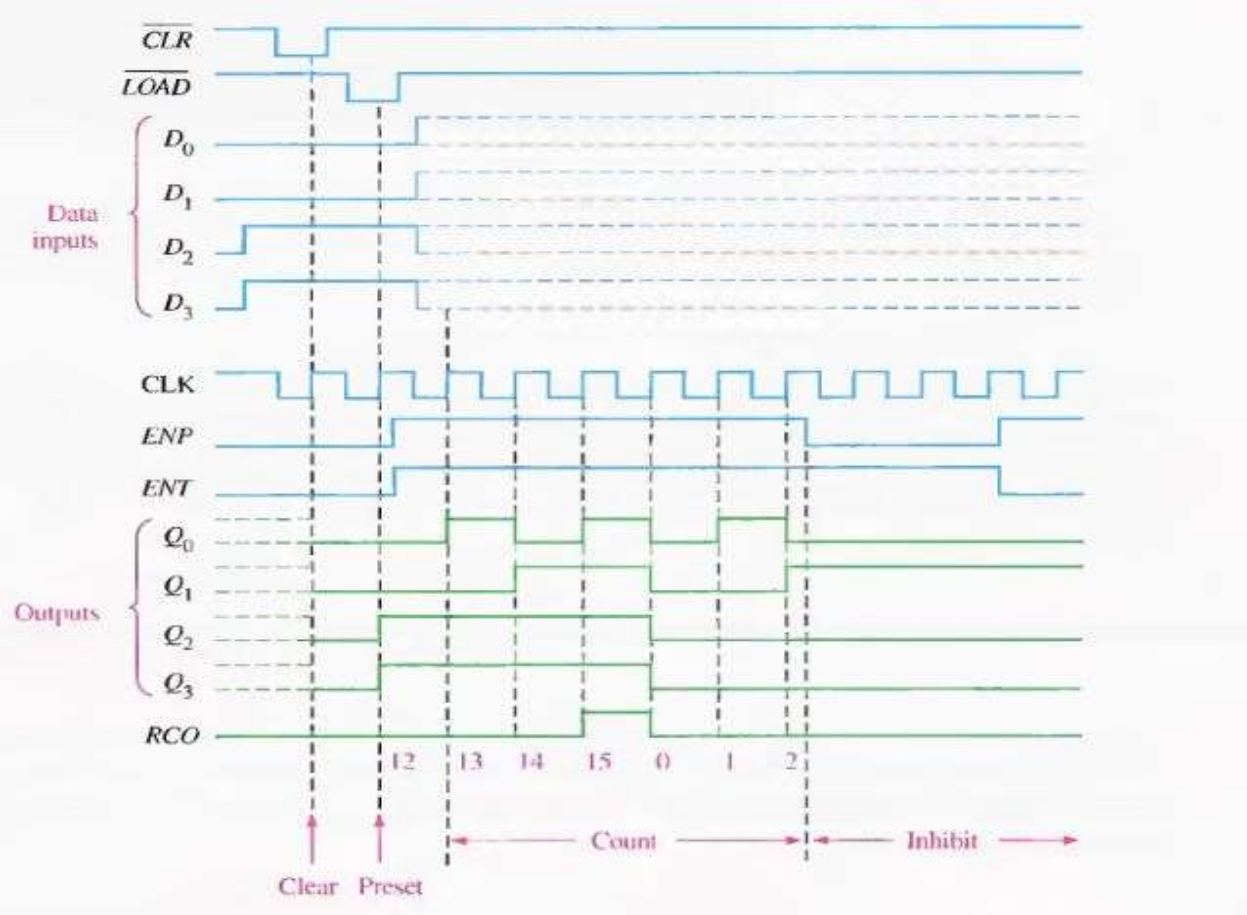
The 74HC163 is an example of an integrated circuit 4-bit synchronous binary counter. A logic symbol is shown in following figure with pin numbers in parentheses. This counter has several features in addition to the basic functions previously discussed for the general synchronous binary counter.
The 74HC163 4-bitsynchronous binary counter. (The qualifying label UR DIV 16 indicates a counter with sixteen states.)



First, the counter can be synchronously preset to any 4-bit binary number by applying the proper levels to the parallel data inputs. When a LOW is applied to the LOAD input, the counter will assume the state of the data inputs on the next dock pulse. Thus, the counter sequence can be started with any 4-bit binary numer.

Also, there is an active-LOW clear input (CLR), which synchronously resets all four flip-flops in the counter. There are two enable inputs. ENP and ENT. These inputs must both be HIGH for the counter to sequence through its binary states. When at least one input is LOW, the counter is disabled. The ripple clock output (RCO) goes HIGH when the counter reaches the last state in its sequence of fifteen, called the terminal count (TC = 15). This output, in conjunction with the enable inputs, allows these counters to be cascaded for higher count sequences.

Figure below shows a timing diagram of 74HC163, this counter being preset to twelve (1100) and then counting up to its terminal count, fifteen (1111). Input Do is the least significant input bit and Qo is the least significant output bit.

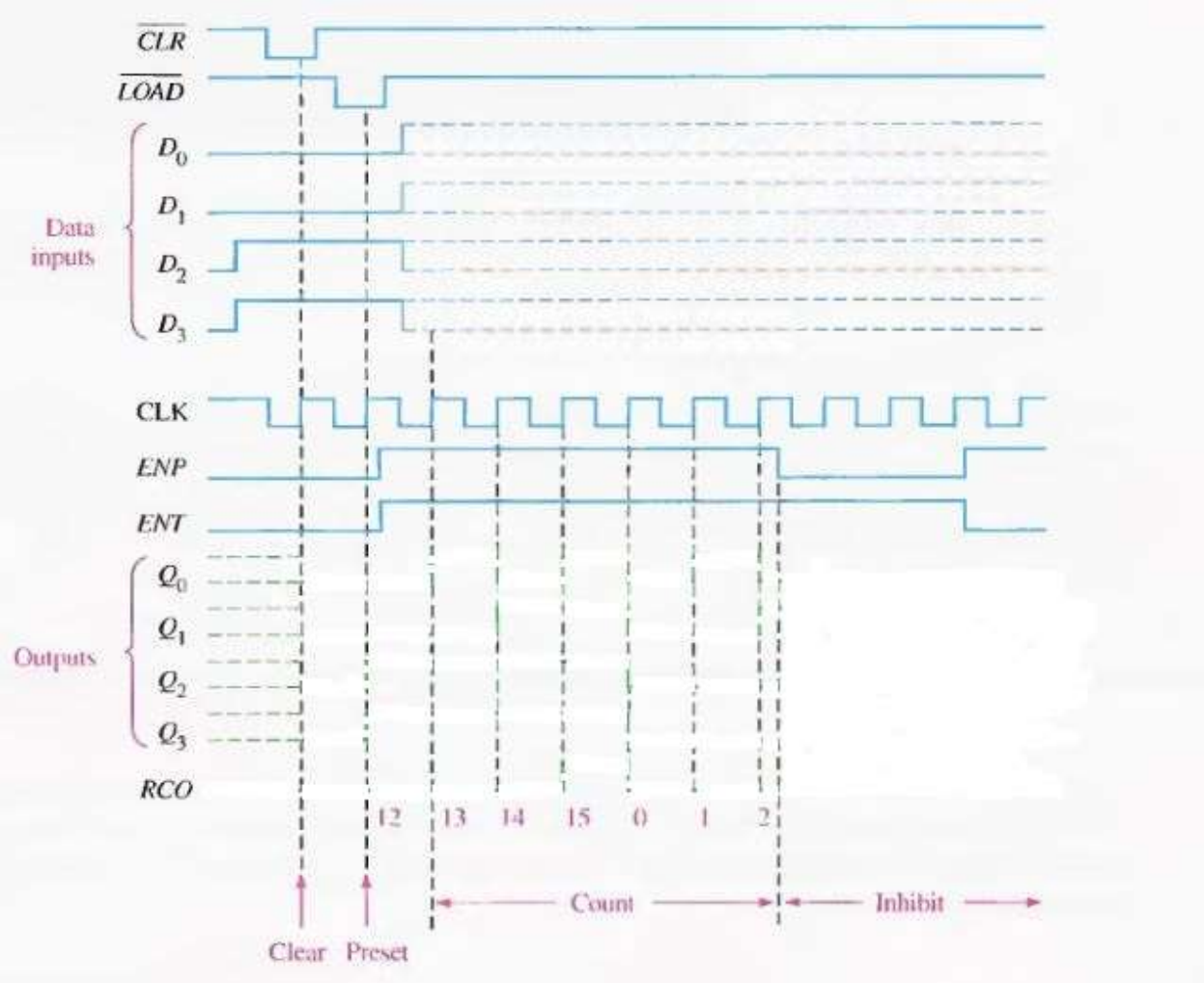


Let's examine this timing diagram in detail. This will aid you in interpreting timing diagr ams i n this chapter or on manufacturers' data sheets. To begin, the LOW level pulse on the CLR input causes all the outputs (Qo, Q b Q2, and Q3) to go LOW.

Next, the LOW level pulse on the LOAD input synchronously enters the data on the data inputs (Do, D1, D2, and D3) into the counte r. These data appear on the Q outputs at the time of the first positive-going clock edge after LOAD goes LOW. This is the preset operation. In this particular example, Qo is LOW, QJ is LOW, Qz is HIGH, and Q3 is HIGH. This, of course, is a binary 12 (Qo is the LSB).

The counter now advances through states 13, 14, and 15 on the next three positive-going clock edges. It then recycles to 0, 1,2 on the following clock pulses. Notice that both ENP and ENTinputs are HIGH during the state sequence. When ENP goes LOW, the counter is inhibited and remains in the binary 2 state.

Observations:



Assignment/Task:

1. Diiferentiate between Synchronous and Asynchronous Counter.

2. State synchoronous counter desinging stages and design 3-bit Gray Code sequence Synchronous counter.

Conclusion/Comments:

\

EXPERIMENT NO 11

IMPLEMENTATION OF SERIES AND PARALLEL REGISTERS

COMPONENTS

1. Digital Logic Trainer.
2. IC 7474 , 74166

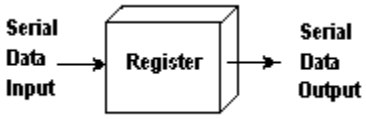
OBJECTIVE

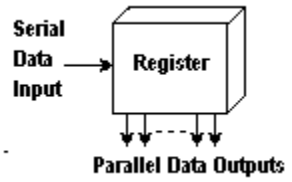
To study various types of registers.

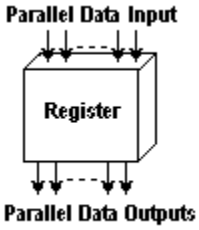
THEORY

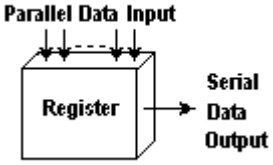
A **Register** is simply a group of flip-flops that can be used to store a binary number. Each flip flop of a register (called cell) stores one bit of word. The bits in a binary number (let's call it data) can be moved from one place to another in either of two ways. The first method involves shifting the data 1 bit at a time, in a serial fashion, beginning with either MSB (most significant bit) or LSB (least significant bit). This technique is known as serial shifting. The second method involves shifting all the data bits simultaneously and is referred to as parallel shifting.

There are two ways of shifting data into a register (serial or parallel) and similarly two ways to shift data out of the register. It is therefore possible to construct four basic types of registers as shown below:



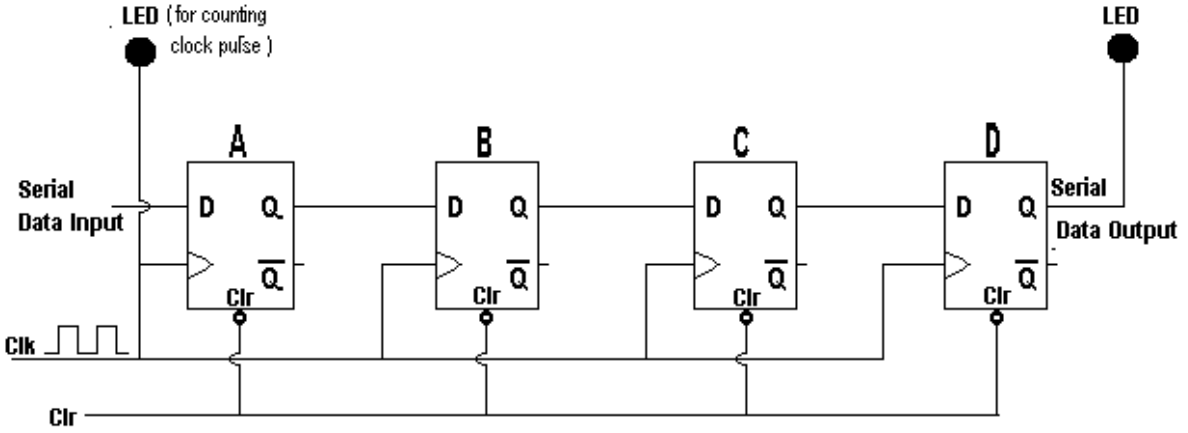


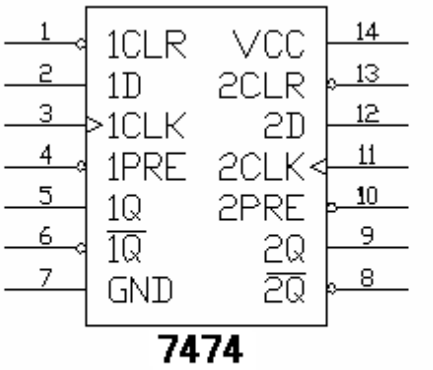




TASK 1: SERIAL-IN SERIAL-OUT SHIFT REGISTER

The Serial-in Serial-out register accepts data serially that is one bit at a time on a single line. It produces the stored information on its output in serial form. The figure below shows a 4-bit Serial-in Serial-out shift register. It is called 4-bit shift register because it has 4-places to store data A,B,C& D.





PROCEDURE

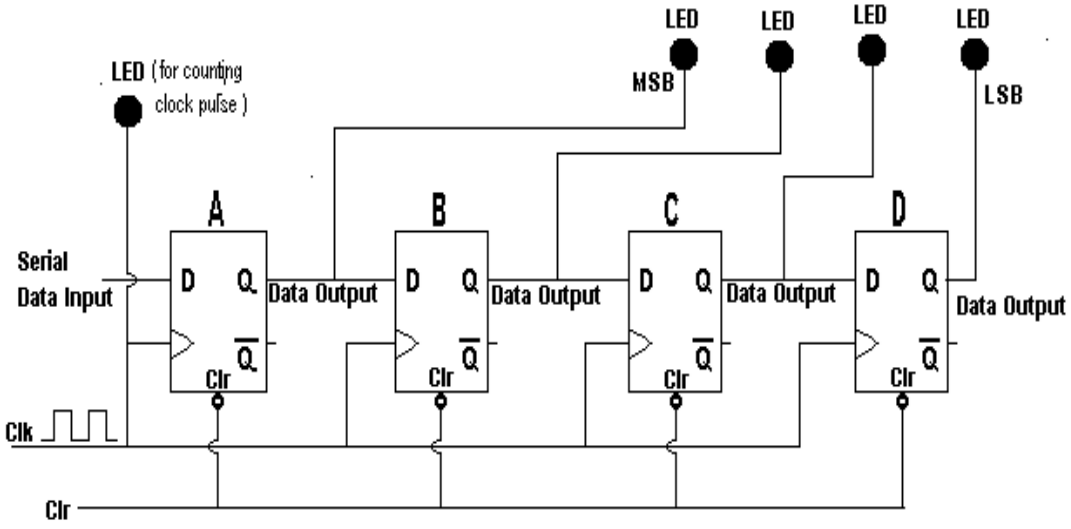
Wire the circuit as per figure above and complete the table below.

Data In	Clock Pulse	Data Out
0	0	
1(LSB)	1	
1	2	
1	3	
1	4	
0	5	
0	6	
1	7	
1(MSB)	8	
-	9	
-	10	
-	11	
-	12	

TASK 2:

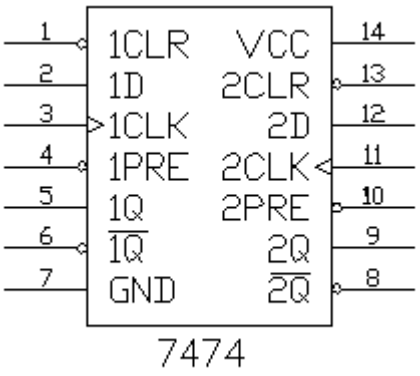
SERIAL-IN PARALLEL-OUT SHIFT REGISTER

The Serial-in Parallel-out register accepts data serially that is one bit at a time on a single line. It produces the stored information on its output in parallel form. The figure below shows a 4-bit Serial-in Parallel-out shift register.



Clock Pulse	Data In	Data Out
1	1 (LSB)	
2	0	
3	1	
4	1 (MSB)	
	-	

				Data Out	
Preset	Clear	Clock	**D**	**Q**	Q’
L	H	X	X	H	
H	L	X	X	L	
L	L	X	X	H	
H	H	↑	**H**	**H**	
H	H	↑	**L**	**L**	
H	H	L	X	QO	

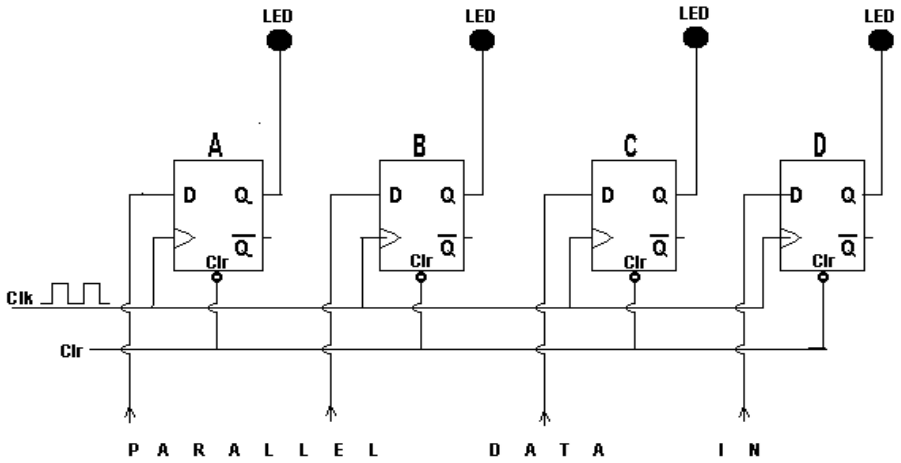


PROCEDURE

Wire the circuit as per figure above and write your observation.

TASK 3: PARALLEL-IN PARALLEL-OUT SHIFT REGISTER

The Parallel-in Parallel-out register accepts data in parallel manner that is all bits at a time on parallel lines. It also produces the information on its output in parallel form. The figure below shows a 4-bit Parallel-in Parallel-out shift register.



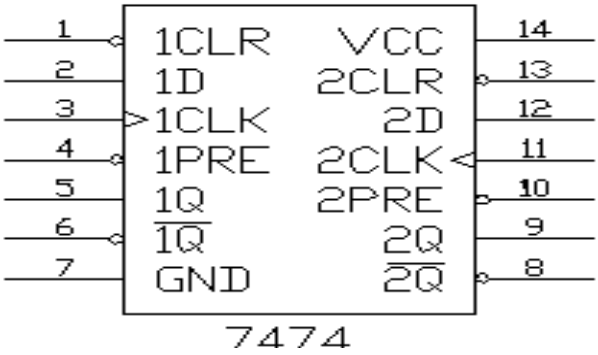
Data In				Clock	Data Out			

PROCEDURE

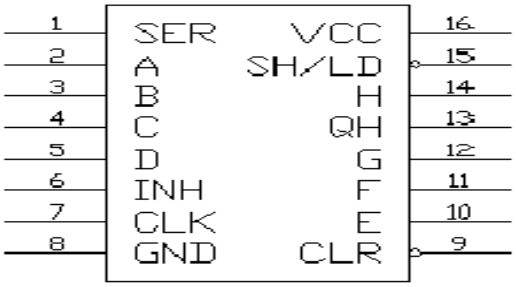
Wire the circuit as per figure above and write your observations.

TASK 4: PARALLEL-IN SERIAL-OUT SHIFT REGISTER

The Parallel-in Serial-out register accepts data in parallel manner that is all bits at a time on parallel lines. It however produces the information on its output in serial form. IC 74166, shown below is shift register. It can shift either Serial or Parallel data entry or it outputs Serial data. In this experiment we will use Parallel input and Serial Output capabilities of the IC Pin 15 is Shift and Load pin. When It is LOW, the register loads data. It is HIGH, the register outputs data.



7474



74166
Parallel Load 8 Bit Shift Register

CONNECTION

1. Do not connect Pin 1 (it is for serial input)

2. Connect inputs A,B,C,D,E,F,G,H to switches S0 to S7.
3. Connect Clock Inhibit Pin 6 to Ground.
4. Connect Clock Pulse to pin 7 and also to LED L8 (for counting clock pulse)
5. Connect Output QH to LED L0
6. Connect a long wire to the IC pin 15 and keep other terminal of the wire hanging (due to limited number of switches in the trainer, we will manually connect this end of the wire to ground for data loading and keep it free (or High)for data shifting
7. Connect Vcc and GND.

PROCEDURE

Wire the circuit as per figure above. Select various combinations of inputs. Load the input and then shift it to output. Observe output (the state of the LED when data is loaded shows state of pin

LAB ASSIGNMENT

P1. Can you make necessary changes so that above register becomes a **Circular Shift Register**?

Conclusion & Comments

EXPERIMENT NO. 12

Tutorials – Introduction to ModelSim and Verilog

TOOL/SIMULATOR

1. ModelSim

OBJECTIVE

Introduction to digital design and simulation.

THEORY

A Register is simply a group of flip-flops that can be used to store a binary number. Each flip flop of a register (called cell) stores one bit of word. The bits in a binary number (let's call it data) can be moved from one place to another in either of two ways. The first method involves shifting the data 1 bit at a time, in a serial fashion, beginning with either MSB (most significant bit) or LSB (least significant bit). This technique is known as serial shifting. The second method involves shifting all the data bits simultaneously and is referred to as parallel shifting.

There are two ways of shifting data into a register (serial or parallel) and similarly two ways to shift data out of the register. It is therefore possible to construct four basic types of registers as shown below:

VERILOG

Can I model combinational logic using always statements? How?

Ideally, concurrent statements are used to model combinational logic and always statements are used to model sequential logic (flip flops and latches). However, always statements are not restricted to that. You can model combinational logic using them. But it is important to note that when using an always statement to make combinational logic, the sensitivity list of the always statement should contain all the signals which are being 'read' in that always block. In other words, to synthesize combinational logic using an always block, all inputs must appear in the sensitivity list.

Can I model combinational logic using always statements? How?

```
always @(a, b, sel)
begin
        if (sel == 1) z <= a;
        else z <= b;
end
```

Using a always statement to model combinational logic is handy because statements like if, case, etc (which are very useful and intuitive) can only be written inside always statements.

What care should I take when using the always statement to write sequential logic?

When using an always statement to model sequential logic, the only thing in the sensitivity list of the always statement should be the clock (or a reset signal, if it is an asynchronous reset). And there should be a' 'posedge' or 'negedge' in the sensitivity list before the clk. This is because flip-flops are edge triggered elements.

Flip-flop without a reset

```
always @(posedge clk) //positive edge triggered
begin
        q <= d;
end
```

Flip-flop with an async reset

```
always @(posedge clk, negedge rst) //positive edge triggered with reset
begin
        if(rst == 0) //async active low reset
         begin
         q <= 0;
         end
else
         begin
        q <= d;
         end
end
```

Flip-flop with a sync reset

```
always @(posedge clk) //positive edge triggered
begin
if(rst == 0) //sync active low reset
        begin
        q <= 0;
        end
else
        begin
        q <= d;
        end
end
```

On the other hand, a latch is a level triggered element. A resettable latch can be modeled as:

```
always @(en, rst, d)
begin
        if(rst == 0)
```

```
            begin
            q <= 0;
        end
        else if(en == 1)
        begin
        q <= d;
        end
end
```

EXAMPLE

This lab is a tutorial lab. You don't have to design anything in this lab, just go through the tutorials and perform them on the lab computers individually. In this course, in almost all the labs we will be doing the following steps:
Step 1: Writing Verilog code of the circuit we want to implement
Step 2: Simulating the Verilog code using a simulator (ModelSim) to check if the intended functionality has been achieved.

Activity 1: ModelSim tutorial
Mentor Graphic's Modelsim tool will be used to perform the functional simulation of our Verilog code for the course. This software is available in all of the ENS labs. Modelsim is also available as a free download with Xilinx's Webpack software so you can install it on your own computer.

Problem 1: Subtractor Design

Write Verilog code for a 1-bit full subtractor using logic equations (Difference = A-B-Bin). If you use delays, make sure to simulate for long enough to see the final result

Write Verilog code for a 4-bit subtractor using the module defined in part (a) as a component. If you use delays, make sure to simulate for long enough to see the final result. Test it for the following input combinations:

1. A = 1001, B = 0011, Bin = 1
2. A = 0011, B = 0110, Bin = 1

1-bit full subtractor truth table:

A	B	Bin	Diff	Bout
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

Problem 2: ALU Design
Design an Arithmetic and Logic Unit (ALU) that implements 8 functions as described in Table 1. Table 1 also illustrates the encoding of the control input.

The 4-bit ALU has the following inputs:

- A: 4-bit input
- B: 4-bit input
- Cin: 1-bit input
- Output: 4-bit output
- Cout: 1-bit output
- Control: 3-bit control input

Control	Instruction	Operation
000	Add	Output <= A + B + Cin; Cout contains the carry
001	Sub	Output <= A – B - Cin; Cout contains the borrow
010	Or	Output <= A or B
011	And	Output <= A and B
100	Shl	Output <= A[2:0] & '0'
101	Shr	Output <= '0' & A[3:1]
110	Rol	Output <= A[2:0] & A[3]
111	Ror	Output <= A[0] & A [3:1]

The following points should be taken care of:

- Use a case statement (or a similar 'combinational' statement) that checks the input combination of "Code" and acts on A, B, and Cin as described in Table 1.
- The above circuit is completely combinational. The output should change as soon as the code combination or any of the input changes.
- You can use arithmetic and logical operators to realize your design.

Simulate this circuit by using the "force" and "run" statements in the transcript window to provide inputs and observe outputs on the waveform window.

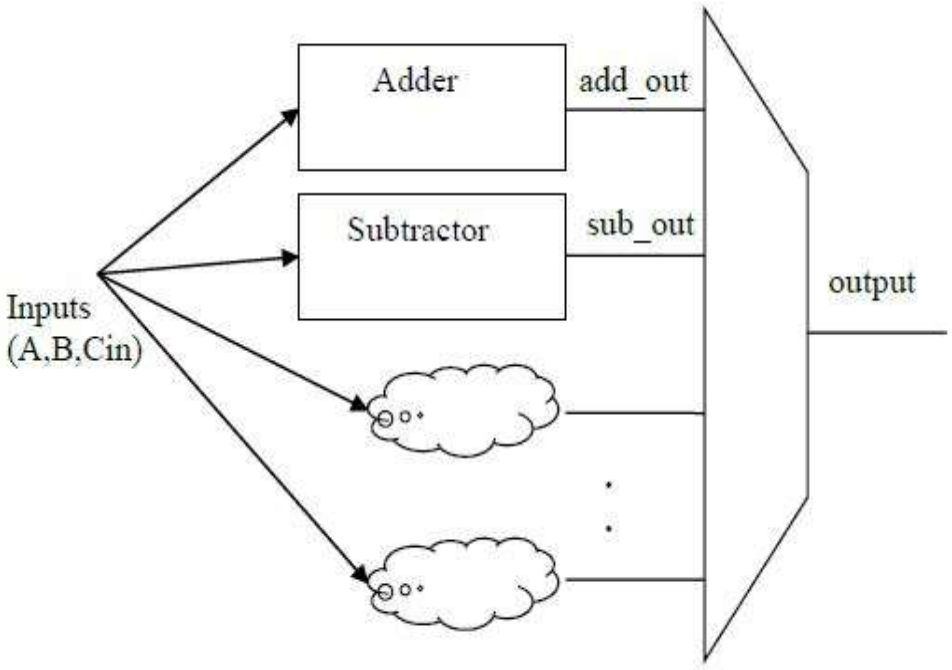
Useful Information

For problem 2, you can use the subtractor block from problem 1 for doing the subtraction (although just using the arithmetic operators will make your design easier). If you use the subtractor from problem 1, remember that we are designing hardware. So, doing something like the following is incorrect:

```
module xyz(…)
always @(…)
case (control)
0 : four_bit_sub sub_inst(in1,in2,bin,output);
……
```

1. First of all, it is important to realize that instantiating a module is not like 'calling' a function in C. Once instantiated, the module is always evaluating its inputs. Therefore, it cannot be conditional. It is always present. So, you should do something like this:

```
module xyz(…)
four_bit_sub sub_inst(in1,in2,bin,sub_out)
always @(…)
case (control)
0 : output <= sub_out;
……
```



2. Make sure that your designs work by testing them sufficiently thoroughly. You should not just use the test inputs in the lab description. Also, it is always better to submit a do-file which has sufficient number of input combinations (not just the ones given in the lab description).

3. Do not use # statements in your design for providing delays.

   #15 X <= A or B;

Report Requirements

You will be expected to describe briefly the codes for problems 1 and 2, simulate and show waveforms in Modelsim, and answer verbal questions. Also, for the last problem you will have to demonstrate that your circuit works on the board

Conclusion & Comments