DAYANANDA SAGAR COLLEGE OF ENGINEERING
(An Autonomous Institution affiliated by VTU, Belagavi)
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING (DATA SCIENCE)

Digital Principles And System Design Lab Manual

Prepared By

Prof. Ankita Mandore

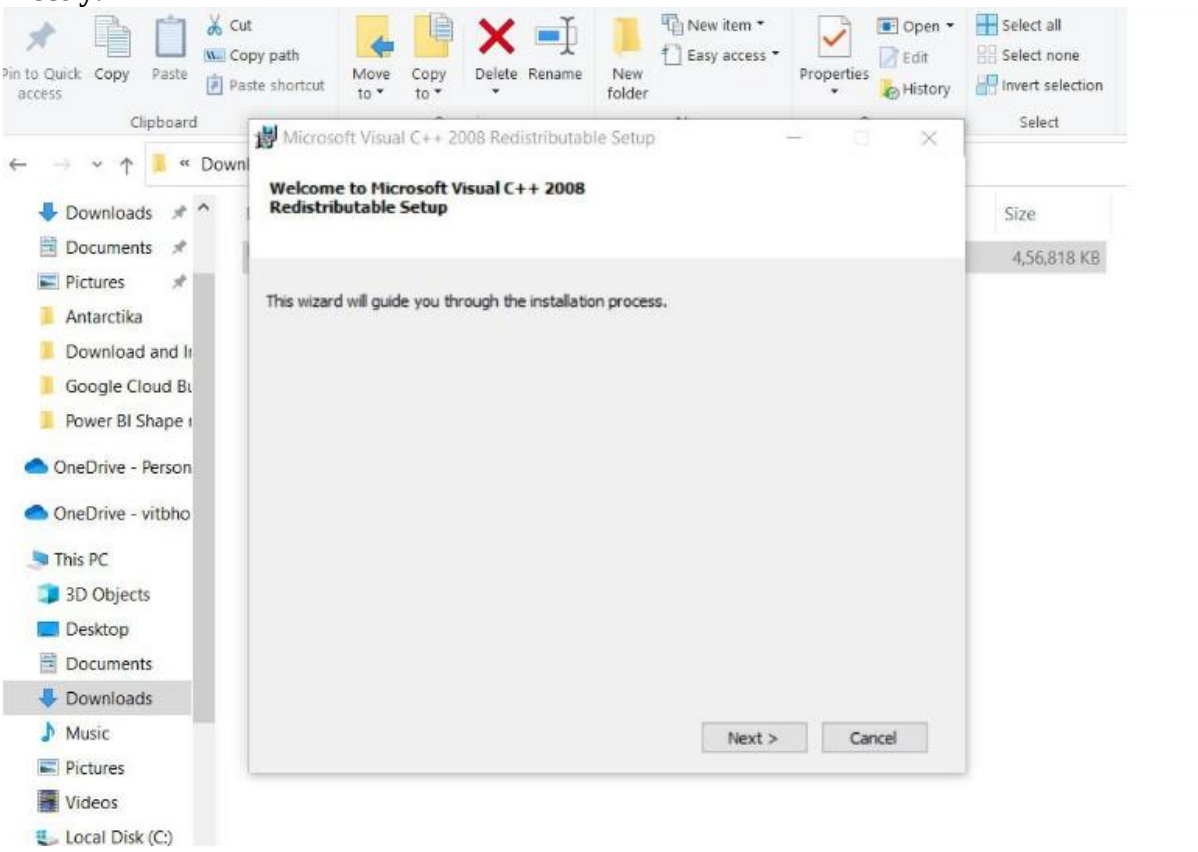
Assistant Professor, CSE (Data Science)

Laboratory Experiments	
Expt No	**Content of the Experiment**
1	Analyze and Implement AND, OR, NOT, NAND, and NOR Gates.
2	Realization of Basic gates using universal Gates.
3	Design and implement Adder and Subtractor
4	Design and implement 8:1 Multiplexer Circuit
5	Realization of Excess-3 Code converter with Parallel Adder and Subtractor using 4-bit adder.
6	Analyze and implement the S-R flip flop
7	Analyze and implement the D-Flip flop
8	Analyze and implement the J K -Flip flop
9	Realization of single digit Seven segment display using the BCD to seven segment decoders
10	Realization of Ring counter.

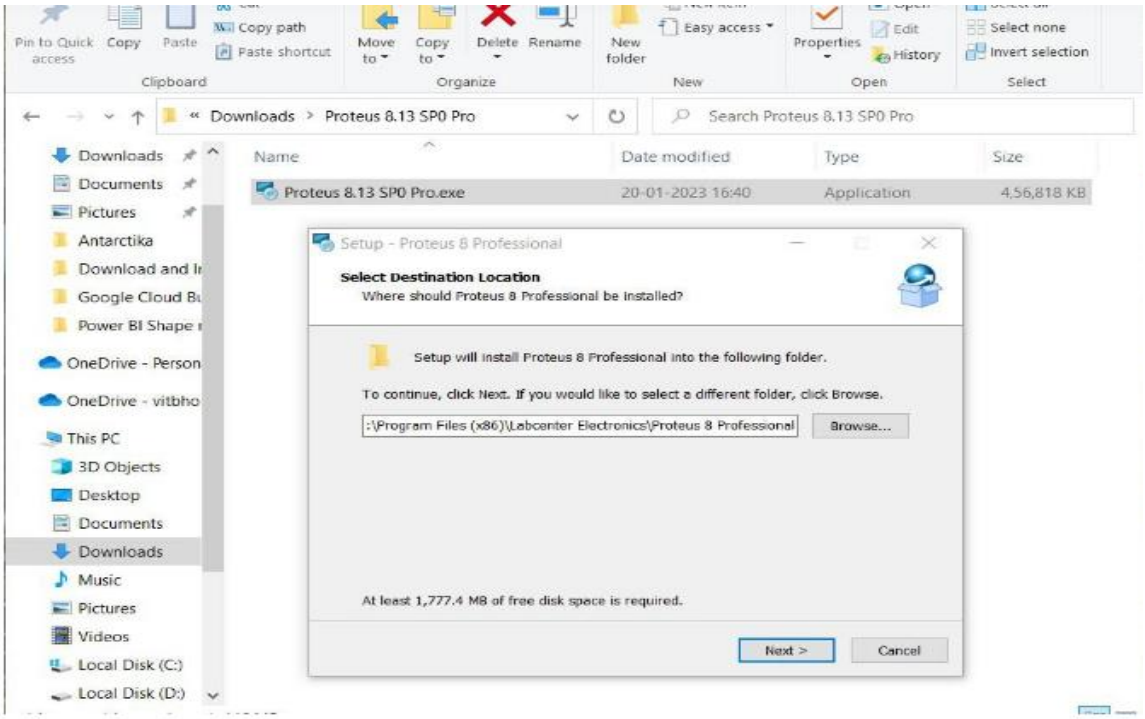
Steps to Install Proteus Software

Step 1: First we need to **Download Proteus.** Here we are going to **install Proteus 8.** Let's For Downloading Click on Download. It will show the following interface now click on **Download**.

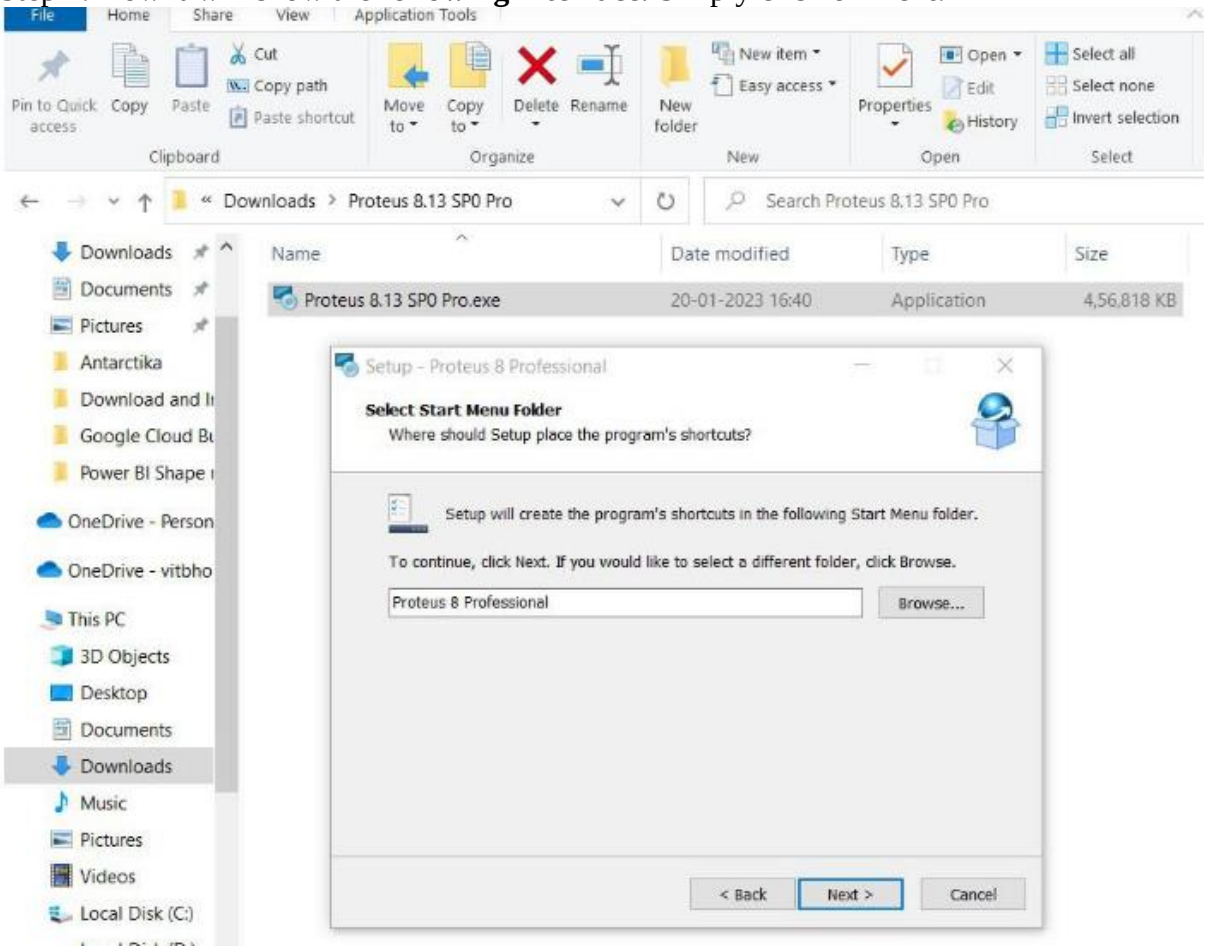
Step 2: In case select you don't have an installation wizard then **first it will ask the Click an install wizard.** Click next and mark check on **Agree**. Then it will install the wizard, it is an **optional process**, if you **already have this wizard then it will go to the next step directly.**



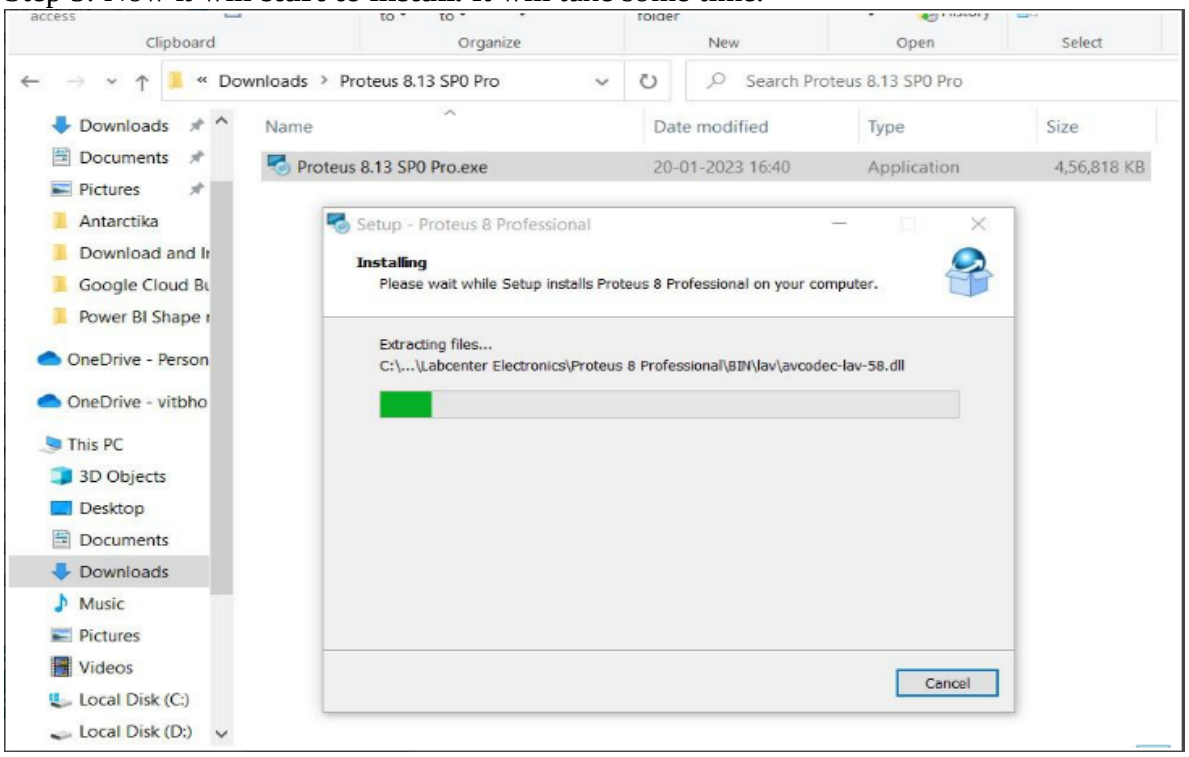
Step 3: Next it will show the following interface and **we need to select the location where we want to install this software.** Click on **Next**.



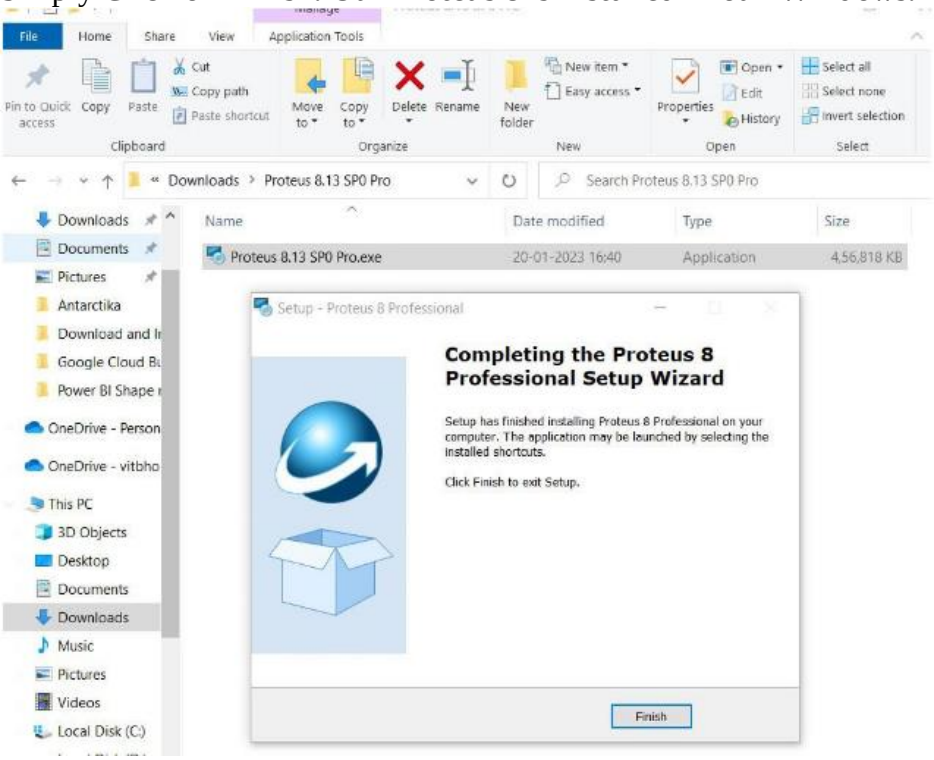
Step 4: Now it will show the **following interface.** Simply click on **Next**.



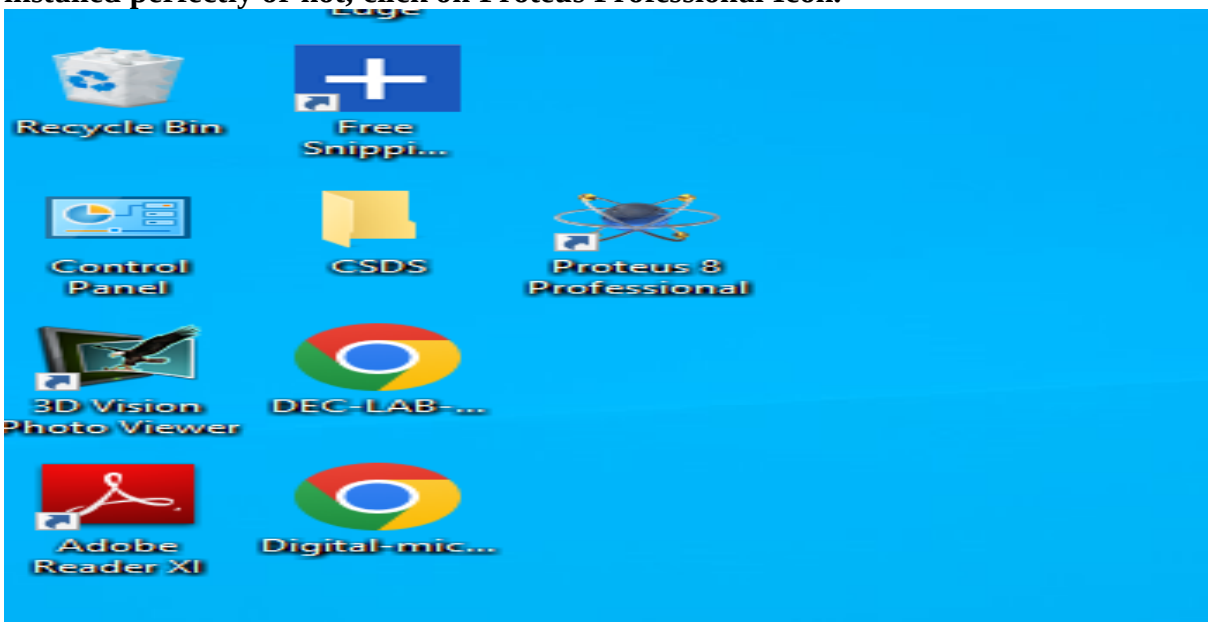
Step 5: Now it will **start to install.** It will take some time.



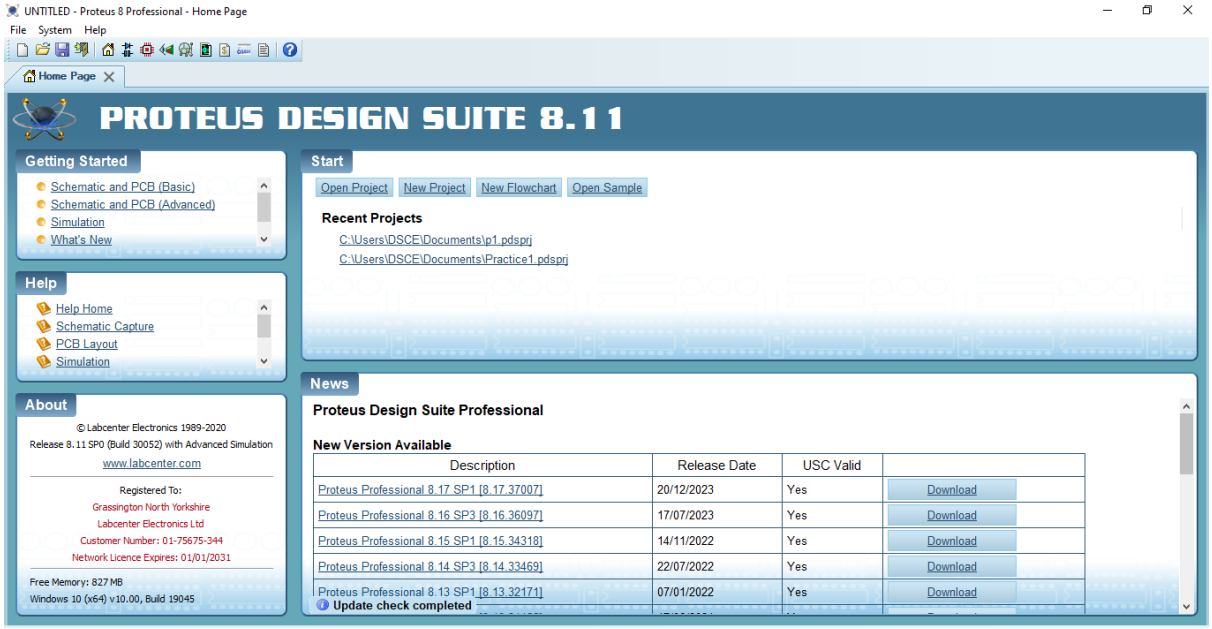
Step 06: After **Completion of Installation** it will show the following interface. Simply **Click on Finish.** Our **Proteus 8** is **installed in our Windows.**



Step 07: In the desktop icon of this software will show like this. For checking whether **it is installed perfectly or not, click on Proteus Professional Icon.**



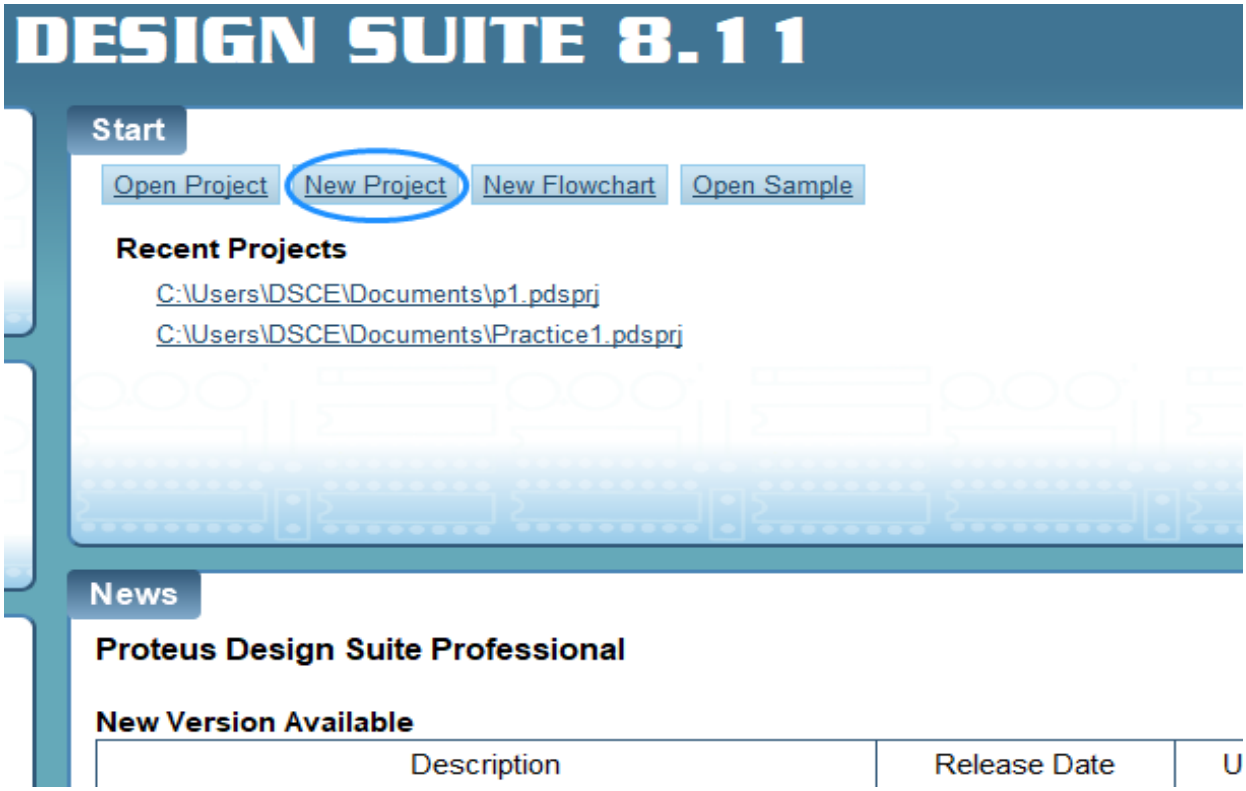
Step 08: It is opening correctly and now **we can start to work with this Proteus.**



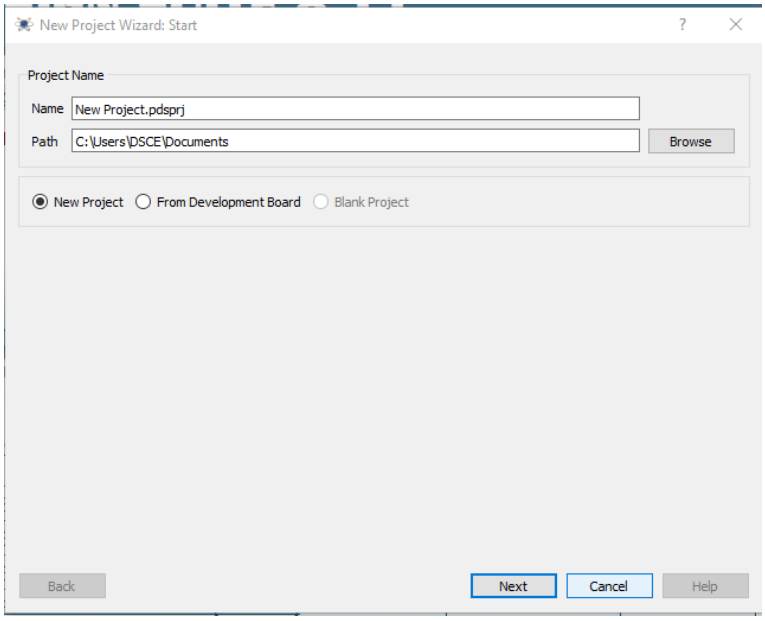
New Project creation in Proteus:

Every time when you start creating project right click on Proteus icon and select option Run as Administrator and perform following steps.

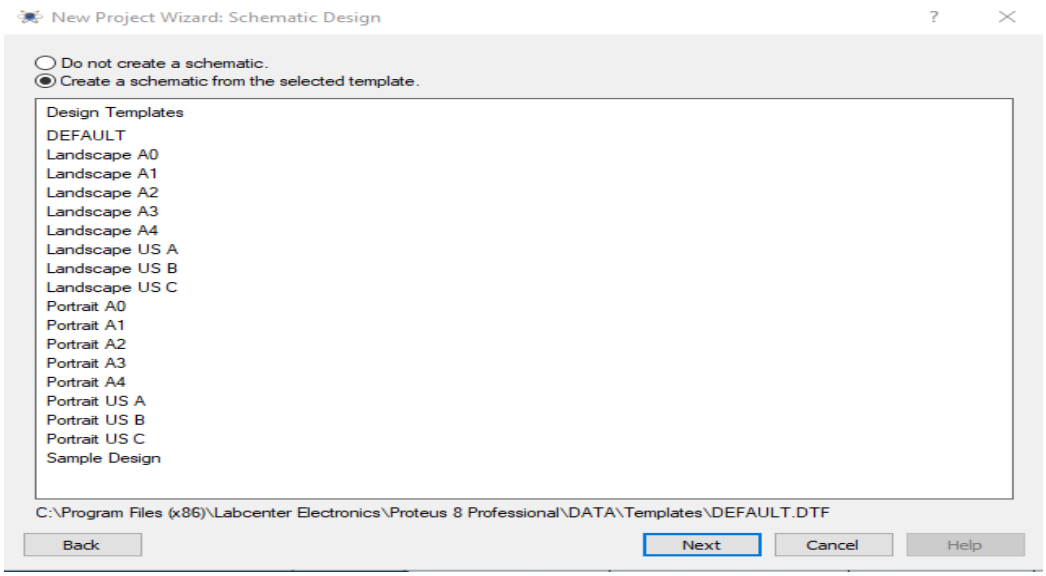
Step 1: Click on "New Project" under the "Start" header in the home page:



Step 2: Choose a name and file-path for your project:

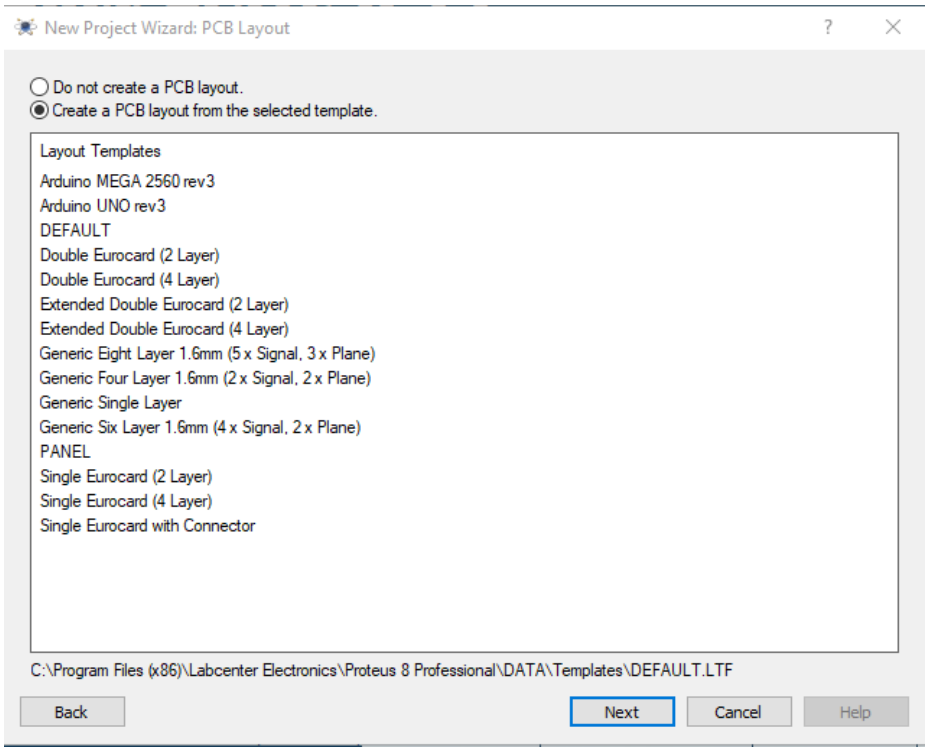


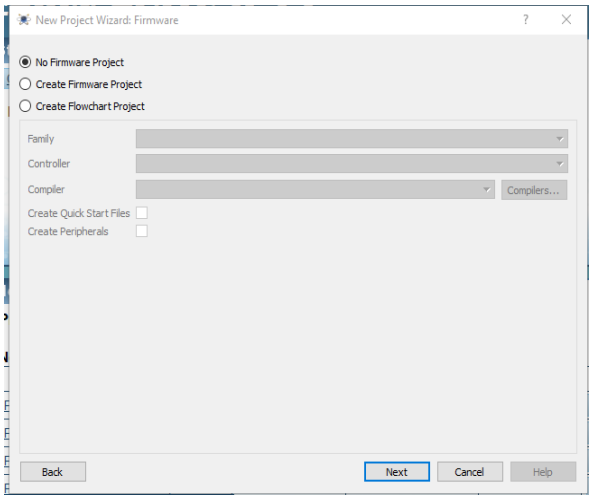
Step 3: Select the Schematic Design:

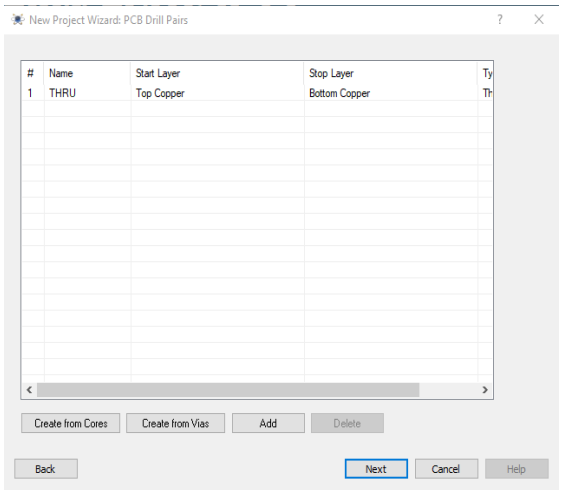


Unless certain dimensions are required (for printing or otherwise), “DEFAULT” will suffice. Click “Next”

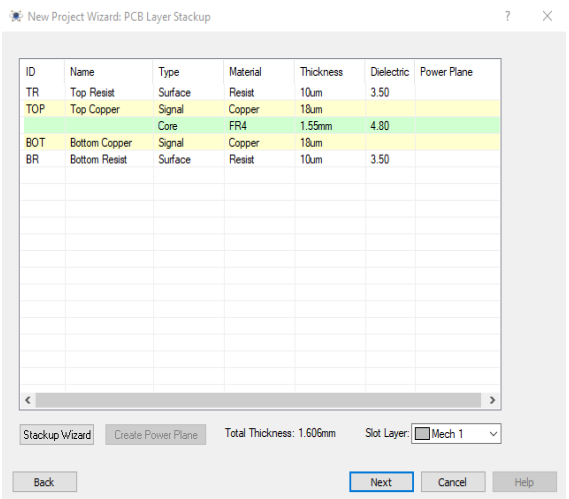
Step 4: Select create PCB layout and set it as default:

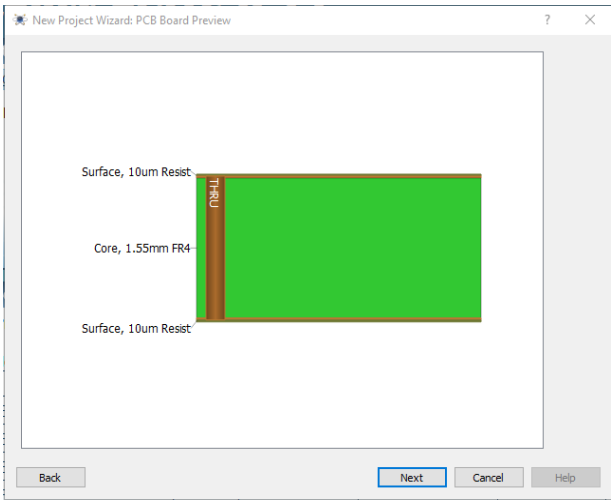


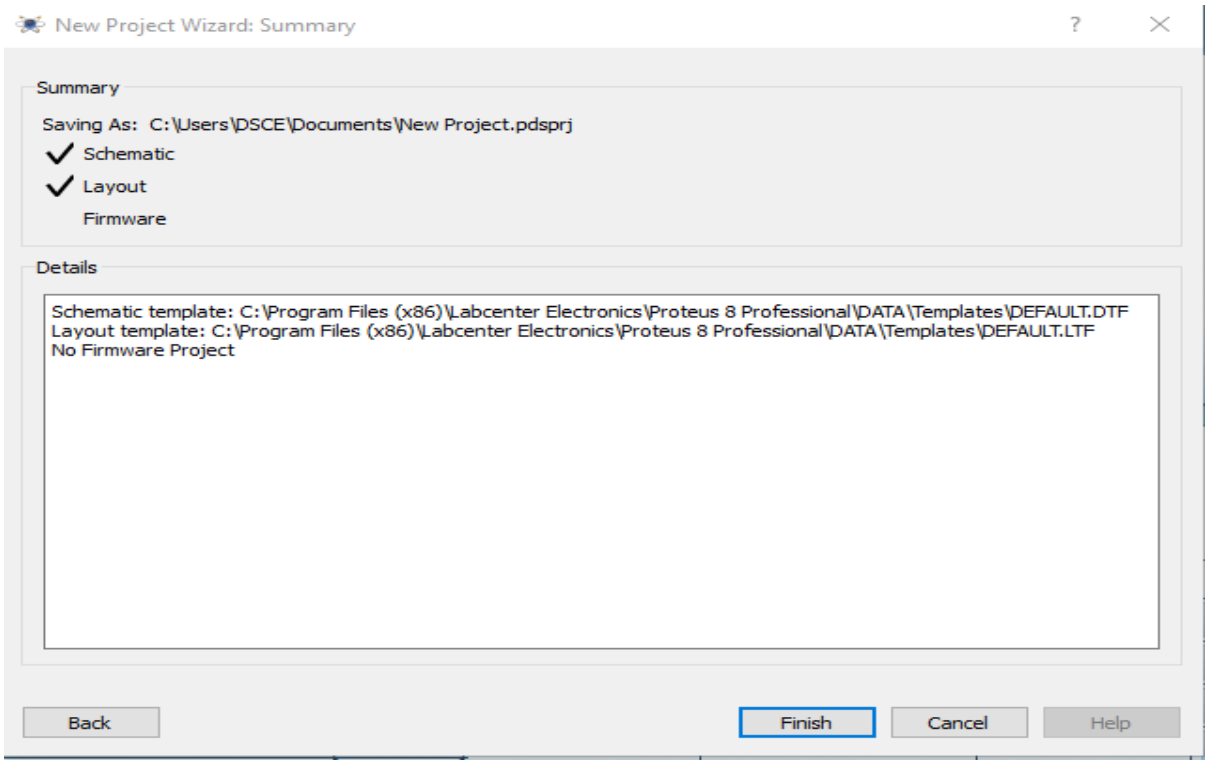




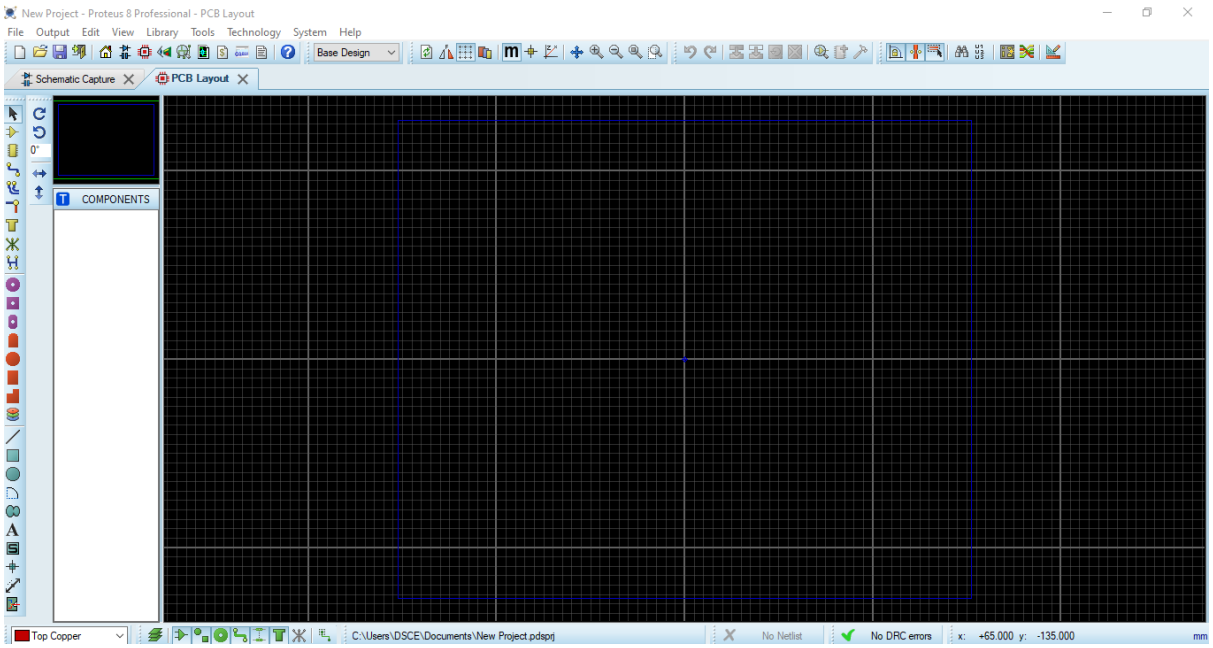
Click on the Next till you get Finish







Step 5: Click on Finish, Following window will be displayed with two options PCB layout and Schematic capture.

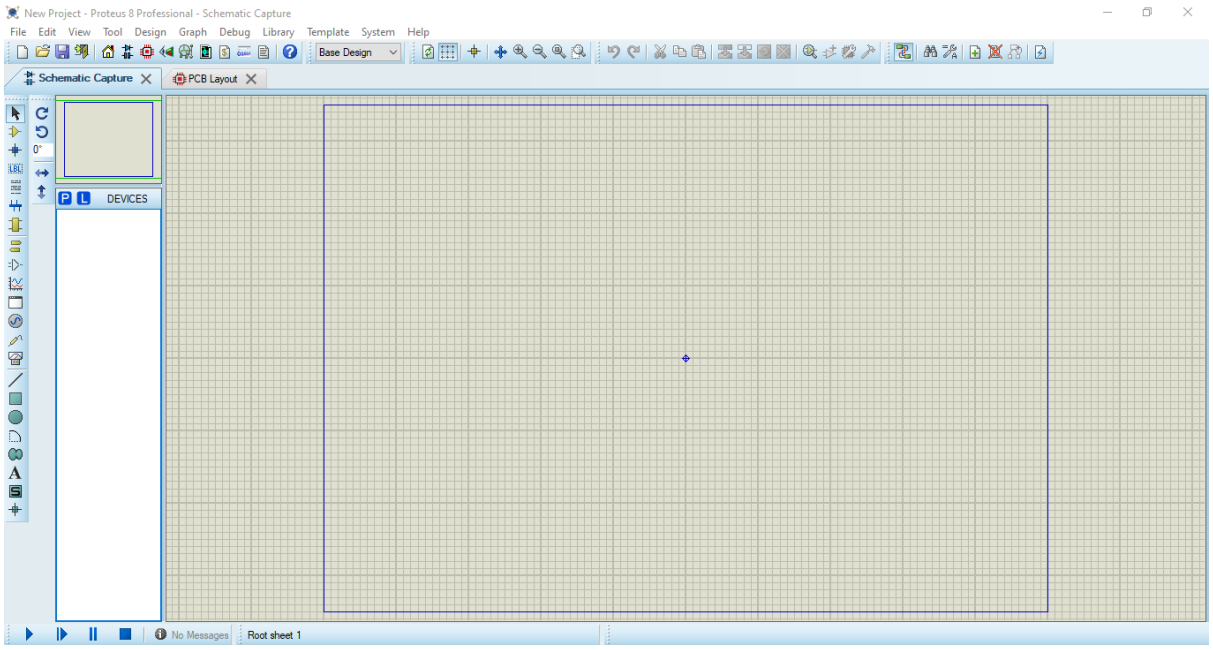


Step 6: Select Schematic capture window to perform practical.

DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institution affiliated by VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING (DATA SCIENCE)



Experiment 1: Analyse and Implement AND, OR, NOT, NAND, and NOR Gates.

AIM: To verify the truth table for all the basic gates and universal gates.

COMPONENTS REQUIRED: AND, OR, NOT, NAND and NOR gates.

THOERY:

Logic gates are the basic building blocks of any digital system. Logic gates are electronic circuits having one or more than one input and only one output. The relationship between the input and the output is based on a certain logic. Based on this, logic gates are named as

1. AND gate
2. OR gate
3. NOT gate
4. NAND gate
5. NOR gate

1) AND gate

The AND gate is an electronic circuit that gives a high output (1) only if all its inputs are high. A dot (.) is used to show the AND operation i.e. A.B or can be written as AB

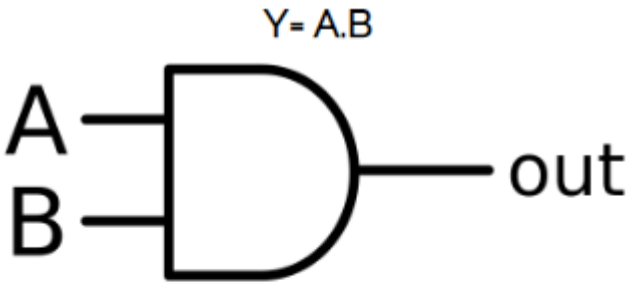


Figure-1:Logic Symbol of AND Gate

Input		Output
A	B	Y=A.B
0	0	0
0	1	0
1	0	0
1	1	1

Figure-2:Truth Table of AND Gate

2) OR gate

The OR gate is an electronic circuit that gives a high output (1) if one or more of its inputs are high. A plus (+) is used to show the OR operation.

Y= A+B

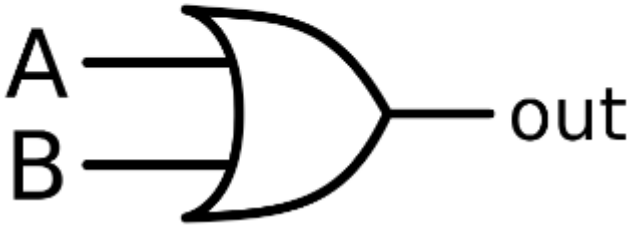


Figure-4:Logic Symbol of OR Gate

Input		Output
A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

Figure-5:Truth Table of OR Gate

3) NOT gate

The NOT gate is an electronic circuit that produces an inverted version of the input at its output. It is also known as an inverter. If the input variable is A, the inverted output is known as NOT A. This is also shown as A' or A with a bar over the top, as shown at the outputs.

Y= A'

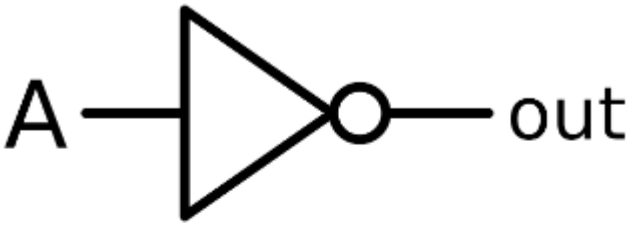


Figure-7:Logic Symbol of NOT Gate

Input	Output
A	Y
0	1
1	0

Figure-8:Truth Table of NOT Gate

4) NAND gate

This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate. The outputs of all NAND gates are high if any of the inputs are low. The symbol is an AND gate with a small circle on the output. The small circle represents inversion.

$Y = \overline{AB}$

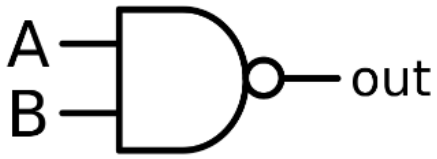


Figure-10:Logic Symbol of NAND Gate

Input	Input	Output
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Figure-11:Truth Table of NAND Gate

5) NOR gate

This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate. The outputs of all NOR gates are low if any of the inputs are high. The symbol is an OR gate with a small circle on the output. The small circle represents inversion.

$$Y = \overline{A+B}$$

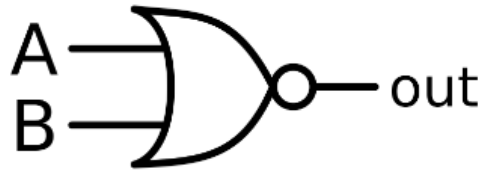


Figure-13:Logic Symbol of NOR gate

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

Figure-14:Truth Table of NOR gate

Procedure:

Output:

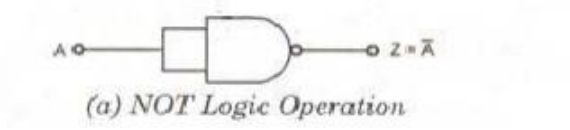
Experiment 2: Realization of Basic gates using universal Gates.

AIM: To realize the implementation of basic gates using universal gates.

To verify the truth table for the implementation of basic gates using universal gates.

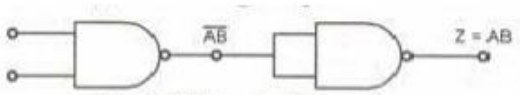
COMPONENTS REQUIRED: IC 7400 (NAND gate) , IC 7402(NOR gate)

NAND OR NOR gates are sufficient for the realization of any logic expression. Because of this reason, NAND and NOR gates are known as UNIVERSAL gates. 1. For NAND gate as universal gate.



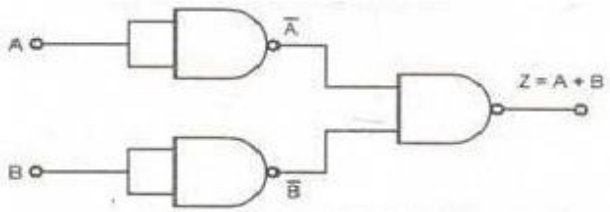
(a) NOT Logic Operation

A	Ā
0	1
1	0



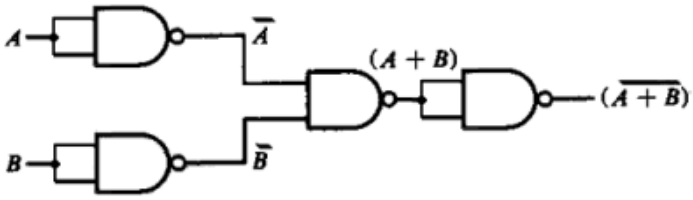
(b) AND Logic Operation

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1



(c) OR Logic Operation

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1



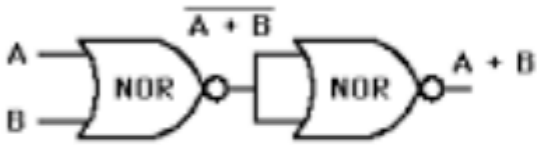
NOR Logic operation

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

2. For NOR gate as universal gate

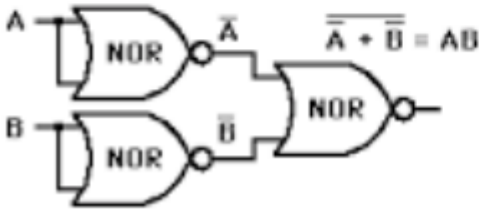
NOT Logic operation

A	$\bar{A}$
0	1
1	0



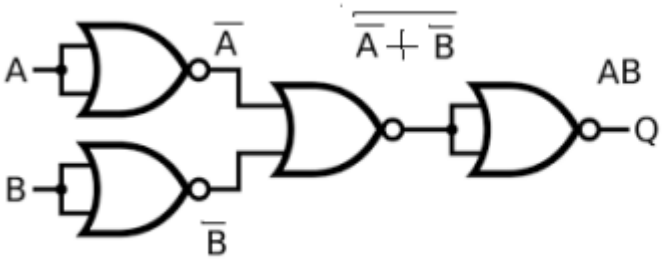
OR Logic operation

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1



AND Logic operation

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1



NAND Logic operation

A	B	AB
0	0	1
0	1	1
1	0	1
1	1	0

Experiment 3: Design and implement Adder and Subtractor

AIM: To design and verify i. Half adder and Full adder ii. Half subtractor and Full subtractor using basic and NAND gates.

COMPONENTS REQUIRED: IC 7400, IC 7408, IC 7486, and IC 7432

THEORY:

Half-Adder: A combinational logic circuit that performs the addition of two data bits, A and B, is called a half-adder. Addition will result in two output bits; one of which is the sum bit, S, and the other is the carry bit, C. The Boolean functions describing the half-adder are:

$S = A \oplus B$

$C = A\,B$

Full-Adder: The half-adder does not take the carry bit from its previous stage into account. This carry bit from its previous stage is called carry-in bit. A combinational logic circuit that adds two data bits, A and B, and a carry-in bit, Cin, is called a full-adder. The Boolean functions describing the full-adder are:

$S = (x \oplus y) \oplus Cin$

$C = xy + Cin\,(x \oplus y)$

I. TO REALIZE HALF ADDER

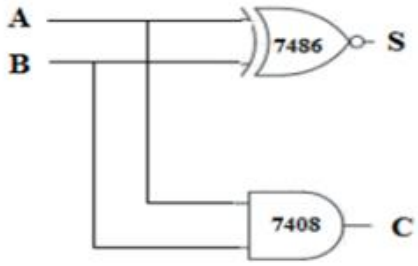
TRUTH TABLE

INPUTS		OUTPUTS	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

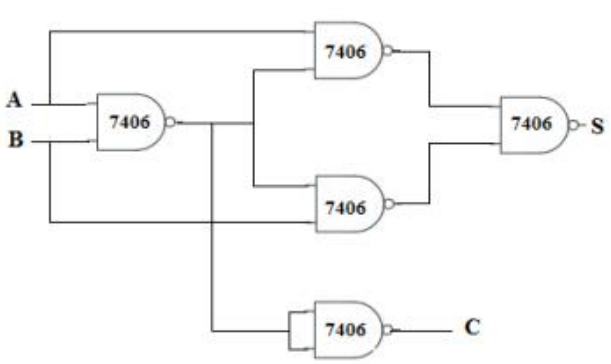
BOOLEAN EXPRESSIONS:

S=A ⊕ B
C=A B

Basic gates



NAND gates



II. FULL ADDER

TRUTH TABLE

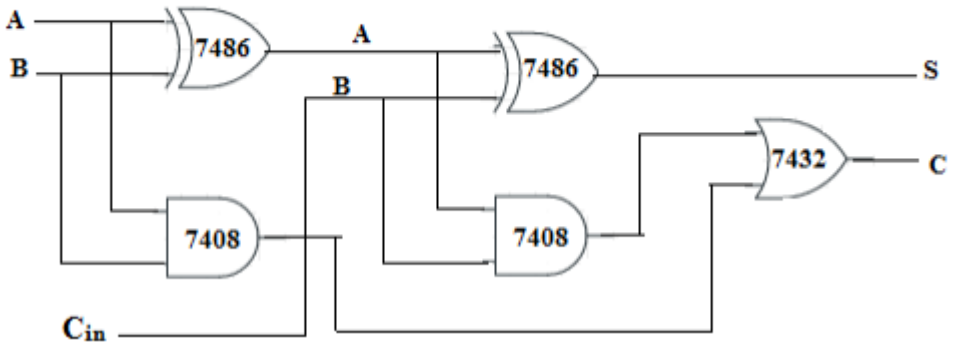
INPUTS			OUTPUTS	
A	B	Cin	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

BOOLEAN EXPRESSIONS:

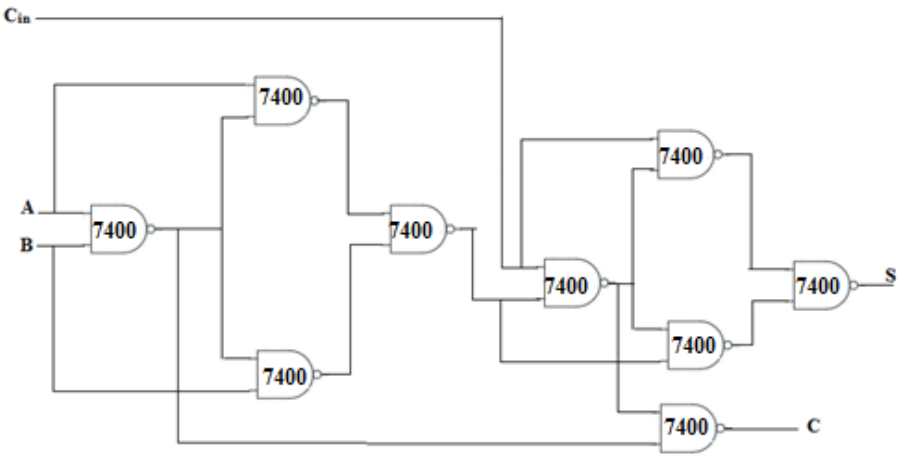
$S = A \oplus B \oplus C$

$C = A\,B + B\,Cin + A\,Cin$

BASIC GATES



i) NAND GATES



III. HALF SUBTRACTOR

TRUTH TABLE

INPUTS		OUTPUT	
A	B	D	Br
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

A —
B —

BOOLEAN EXPRESSIONS:

$$D = A \oplus B$$

$$Br = \bar{A}B$$

IV. FULL SUBTRACTOR

TRUTH TABLE

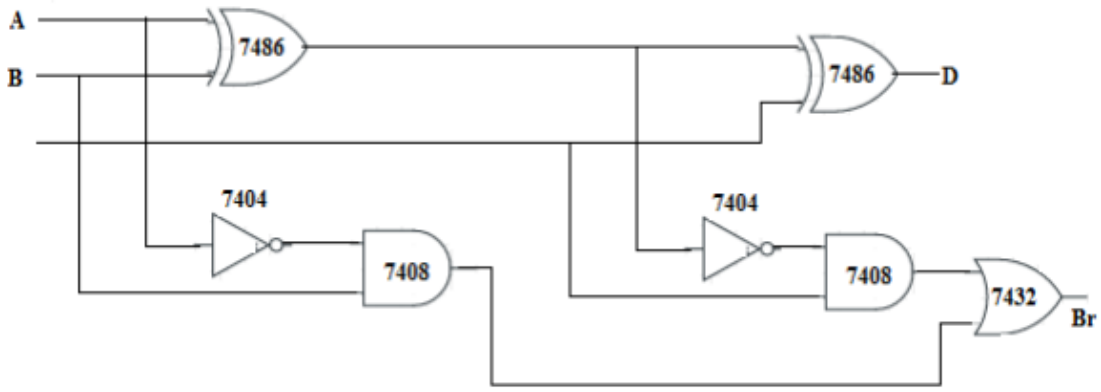
INPUTS			OUTPUTS	
A	B	Cin	D	Br
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

BOOLEAN EXPRESSIONS:

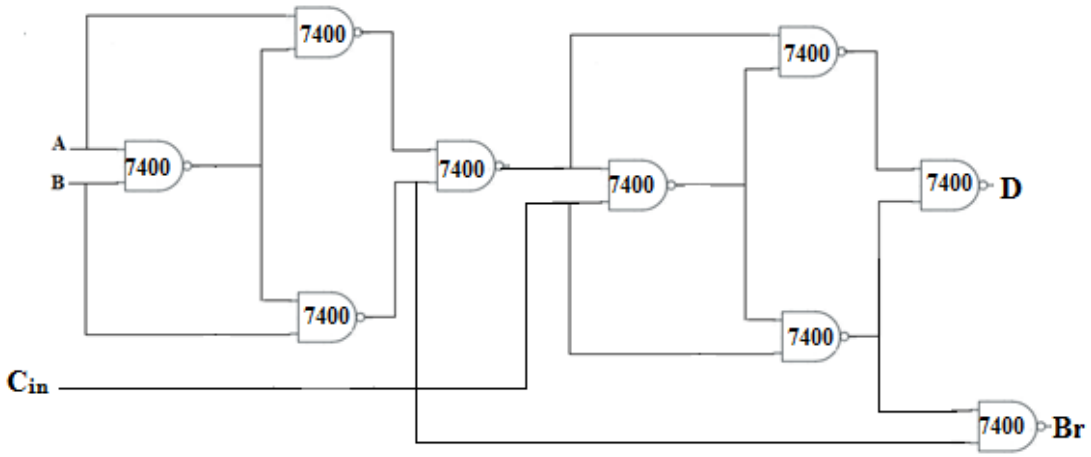
$$D = A \oplus B \oplus C$$

$$Br = \bar{A}\,B + B\,Cin + \bar{A}\,Cin$$

BASIC GATES



To realize the FULL subtractor using NAND Gates only



Procedure:

Output:

Experiment 4: Design and implement 8:1 Multiplexer Circuit

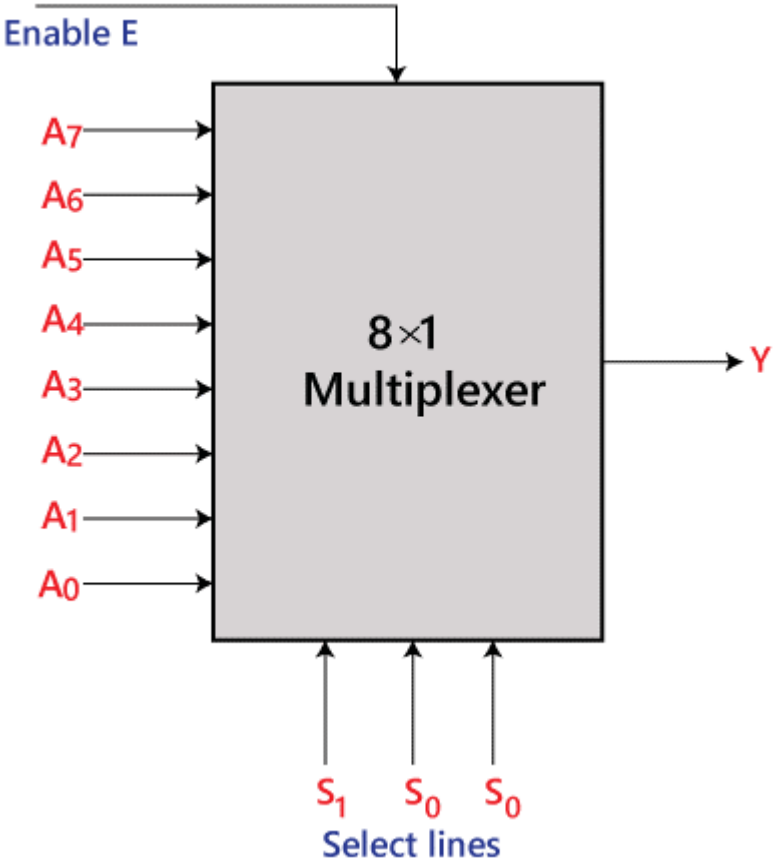
AIM: To verify the implementation of multiplexer.

COMPONANT REQUIRED: AND, NOT and OR gates

THEORY:

In the 8 to 1 multiplexer, there are total eight inputs, i.e., A_0, A_1, A_2, A_3, A_4, A_5, A_6, and A_7, 3 selection lines, i.e., S_0, $S_1$and S_2 and single output, i.e., Y. On the basis of the combination of inputs that are present at the selection lines S^0, $S^{1,}$ and S_2, one of these 8 inputs are connected to the output. The block diagram and the truth table of the 8×1 multiplexer are given below.

Block Diagram:



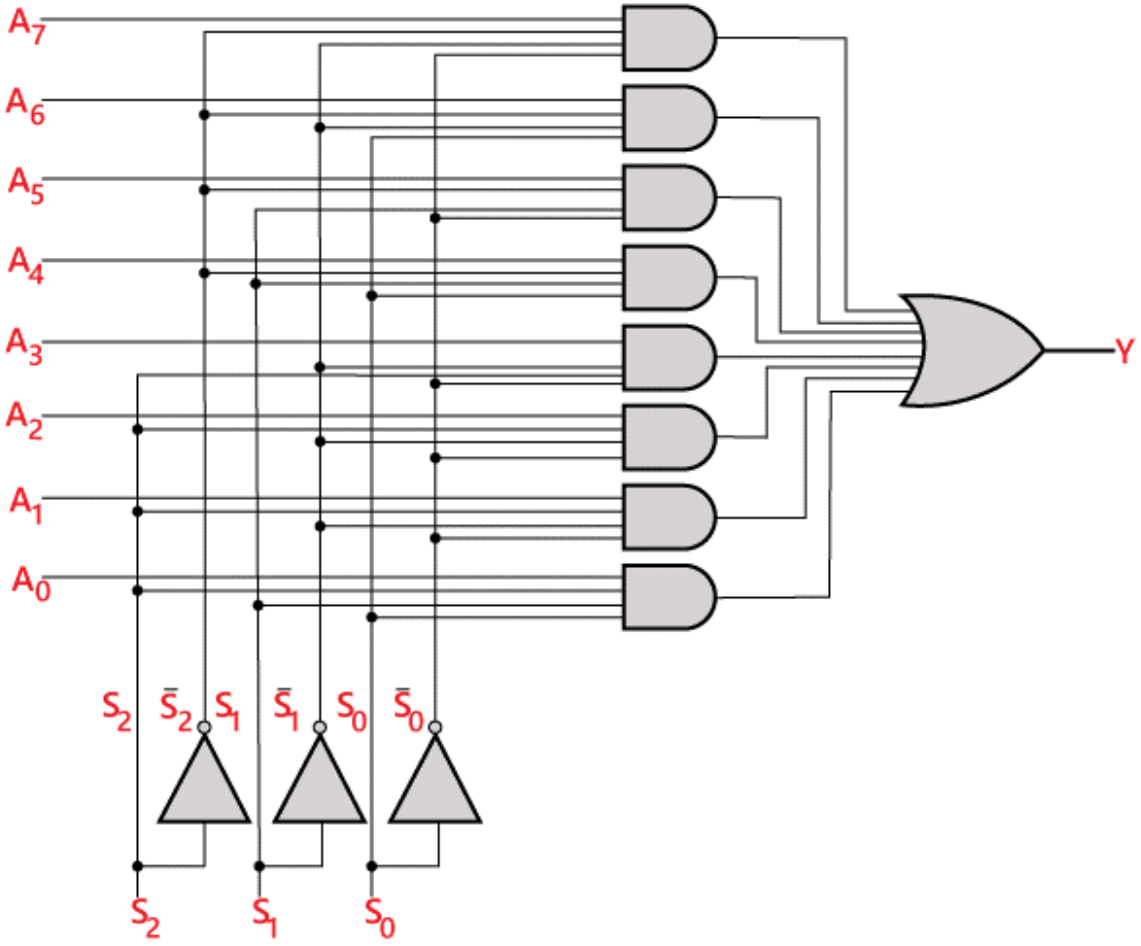
Truth Table:

INPUTS			Output
S_2	S_1	S_0	Y
0	0	0	A_0
0	0	1	A_1
0	1	0	A_2
0	1	1	A_3
1	0	0	A_4
1	0	1	A_5
1	1	0	A_6
1	1	1	A_7

The logical expression of the term Y is as follows:

$Y=S_0'.S_1'.S_2'.A_0+S_0.S_1'.S_2'.A_1+S_0'.S_1.S_2'.A_2+S_0.S_1.S_2'.A_3+S_0'.S_1'.S_2\ A_4+S_0.S_1'.S_2\ A_5+S_0'.S_1.S_2\ .\ A_6+S_0.S_1.S_3.A_7$

Logical circuit of the above expression is given below:



Procedure:

Output:

Experiment 5: Realization of Excess-3 Code converter with Parallel Adder and Subtractor using 4-bit adder.

AIM: To design and set up the following:

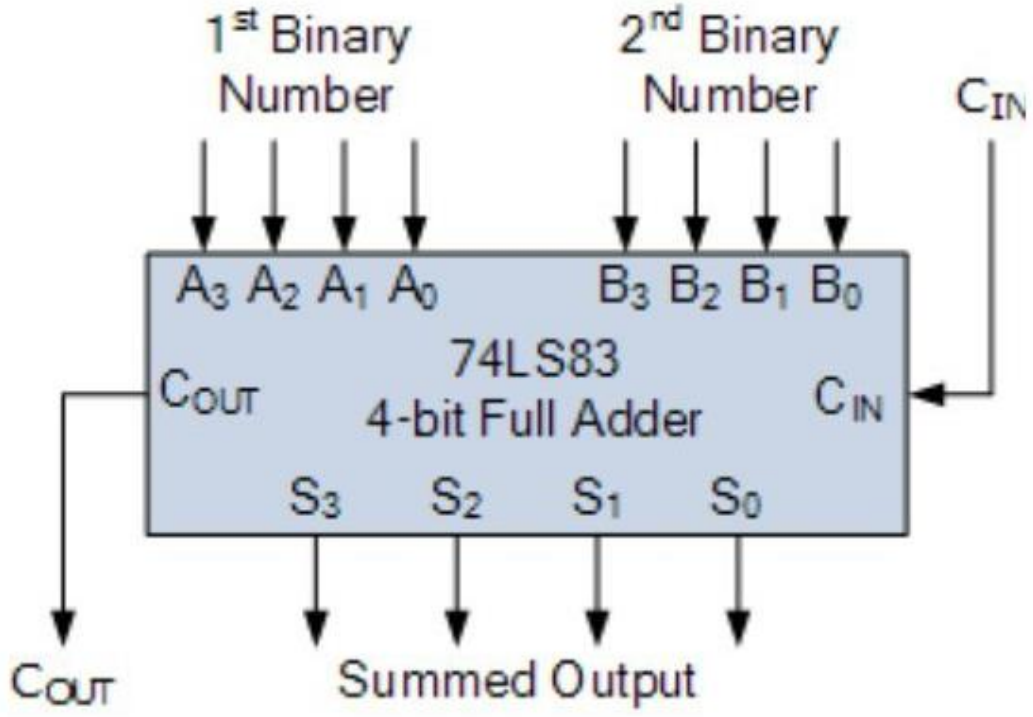
1. 4 bit binary adder and Subtractor.
2. Code conversion BCD to Excess-3.

COMPONANT REQUIRED: IC 7483, IC 7486

THEORY:

Excess-3 code we get by adding decimal 3 (or BCD 0011) to BCD code. So we here we will reverse the procedure to get BCD code out of excess 3 code input.

Let excess 3 input be $A_3A_2A_1A_0$



So to get BCD code $S_3S_2S_1S_0$ from this excess 3 code we will subtract 3 from it.

We will carry out this subtraction using twos complement method.

$A_3A_2A_1A_0 - (3)_{10} = A_3A_2A_1A_0 +$ 1s complement of $3_{10} + 1$

$= A_3A_2A_1A_0 + (1100)_2 + (1)_2$

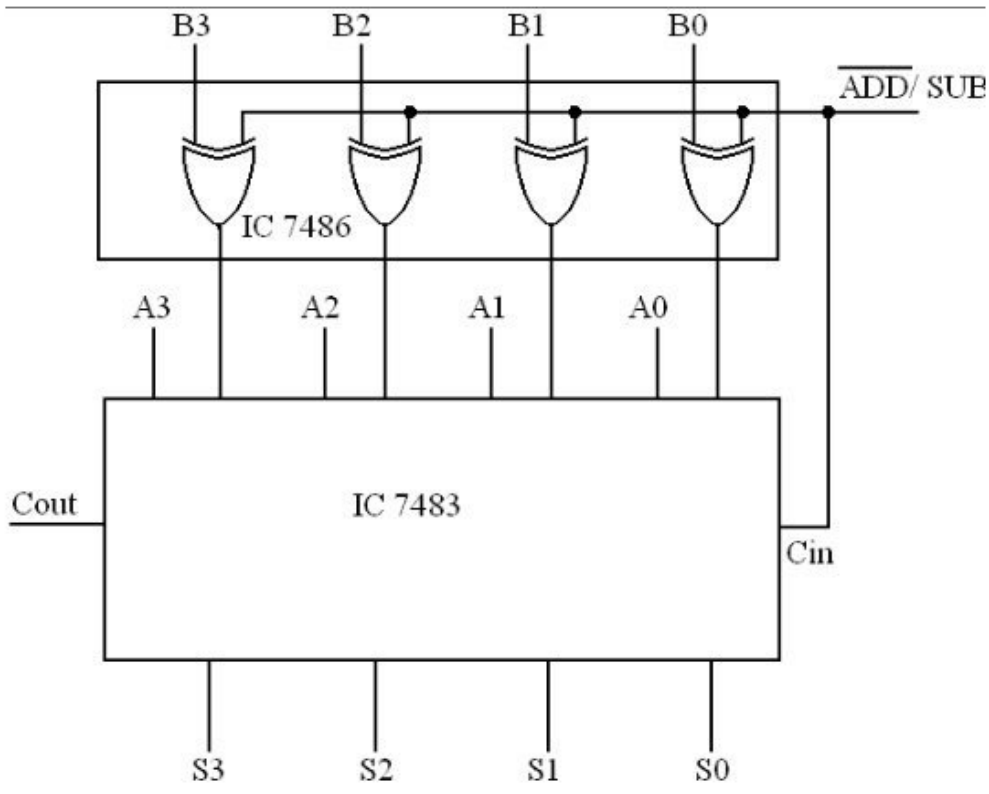
So in adder we will assign

$B_3B_2B_1B_0 = (1100)_2$

And $C_{in} = (1)_2$

And the output $S_3S_2S_1S_0$ will give BCD code.

The other way to do this is by using following 4 bit Adder/subtractor circuit.



Here XOR gates are used as controlled inverter and will be used for 1's complement of B.

So here,

$A_3A_2A_1A_0 - (3)_{10} = A_3A_2A_1A_0 +$ 1s complement of $3_{10} + 1$

So, $B_3B_2B_1B_0 = (3)_{10} = (0011)_2$

And $C_{in} = 1$.

Decimal Numerals	BCD Input				Excess-3 Output			
Decimal	B3	B2	B1	B0	E3	E2	E1	E0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Procedure:

Output:

Experiment 6: Analyze and implement the S-R flip flop.

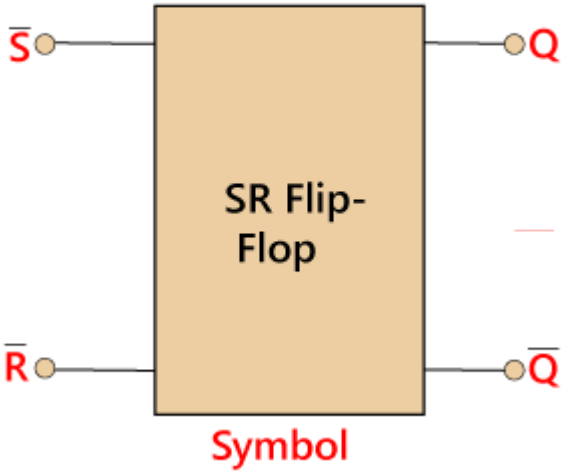
AIM: To verify the working of S-R flip flop.

COMPONANT REQUIRED: AND, OR gates, Clock

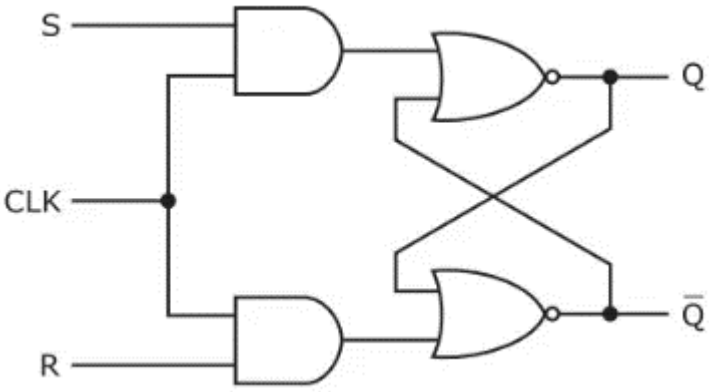
THEORY:

The **SR Flip-Flop** is also known as the gated or clocked SR latch. The clocked SR latch or SR flip-flop temporarily stores or holds the information until it is needed in digital circuits. 'S' and 'R' are the two inputs to the SR flip-flop. It has two outputs, the main output 'Q' and the complements of the main output ' Q' '. The SR Flip-Flop is a storage element with only one bit. The SR flip-flop is a gated SR flip-flop with a clock input circuitry that does not prevent the illegal or invalid output state that can arise when both inputs S and R are equal to logic level "1". The SR latch is constructed using two cross-coupled NAND gates.

Block Diagram:



Symbol



SR Flip-Flop Truth Table

Clock	S	R	Q_{n+1}	State
0	X	X	Qn	X
1	0	0	Qn	Hold
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	X	Invalid

SR Flip-Flop Characteristic Table

S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

Procedure:

Output:

Experiment 7: Analyze and implement the D-Flip flop.

AIM: To verify the working of D-Flip Flop.

COMPONANT REQUIRED: NOT, NAND gate

THEORY:

D flip flop is an electronic devices that is known as "delay flip flop" or "data flip flop" which is used to store single bit of data.D flip flops are synchronous or asynchronous. The clock single required for the synchronous version of D flip flops but not for the asynchronous one.The D flip flop has two inputs, data and clock input which controls the flip flop. when clock input is high, the data is transferred to the output of the flip flop and when the clock input is low, the output of the flip flop is held in its previous state.

Working of D Flip Flop

D flip flop consist of a single input D and two outputs (Q and Q'). The basic working of D Flip Flop is as follows:

- When the clock signal is low, the flip flop holds its current state and ignores the D input.
- When the clock signal is high, the flip flop samples and stores D input.
- The value that was previously fed into the D input is reflected at the flip flop's Q output.
    - If D = 0 then Q will be 0.
    - If D = 1 then Q will be 1.
- The Q' output of the flip flop is complemented by the Q output.
    - If Q = 0 then Q' will be 1.
    - If Q = 1 then Q' will be 0.

D	CLK	$\bar{Q}$
0	1 (Raising Edge)	0
1	1 (Raising Edge)	1

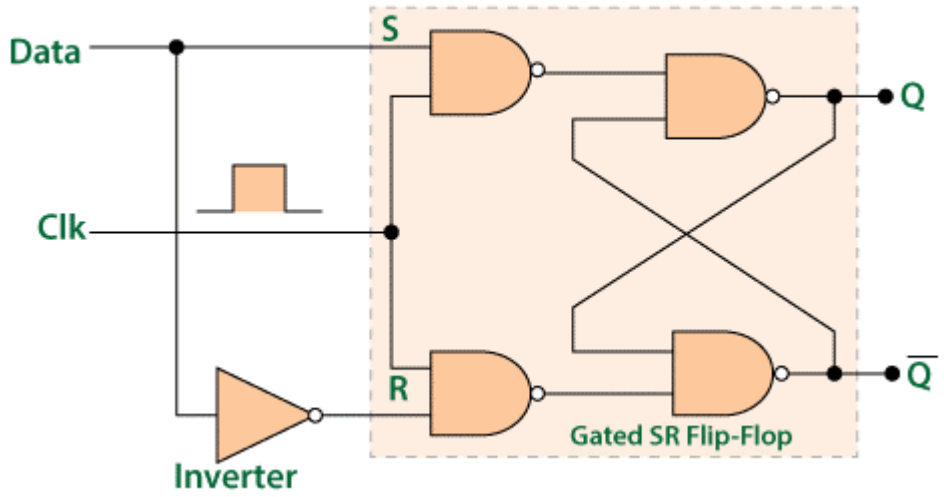
Truth Table of D Flip Flop

Characteristic Table of D Flip Flop

The characteristic table of the D flip flop displays the behavior of the flip flop for each combination of input and current state. The characteristic table for a D flip flop is as follows.

D	Q(Current)	Q(n+1) (Next)
0	0	0
0	1	0
1	0	1
1	1	1

Characteristics table of D Flip Flop



Procedure:

Output:

Experiment 8: Analyze and implement the J K -Flip flop.

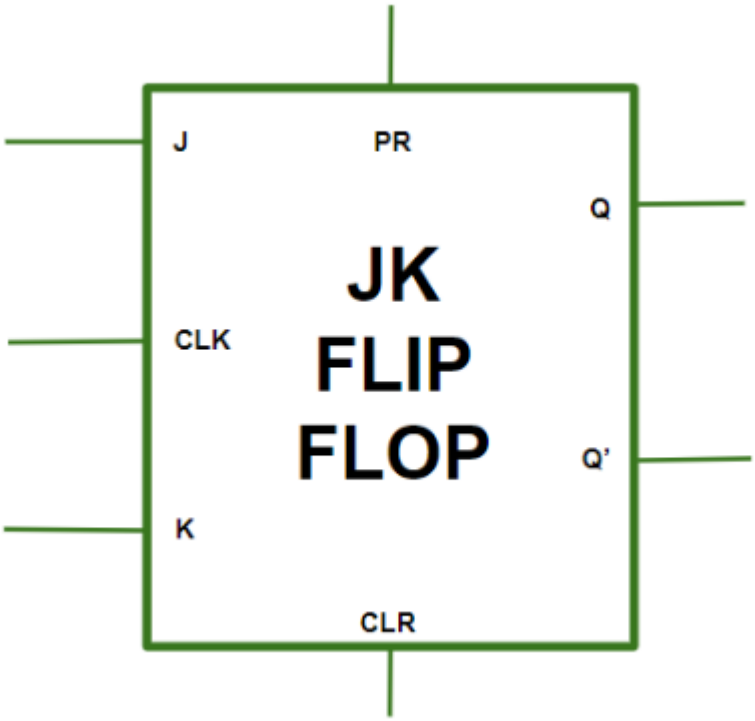
AIM: To verify working of J-K Flip Flop.

COMPONANT REQUIRED:

THEORY:

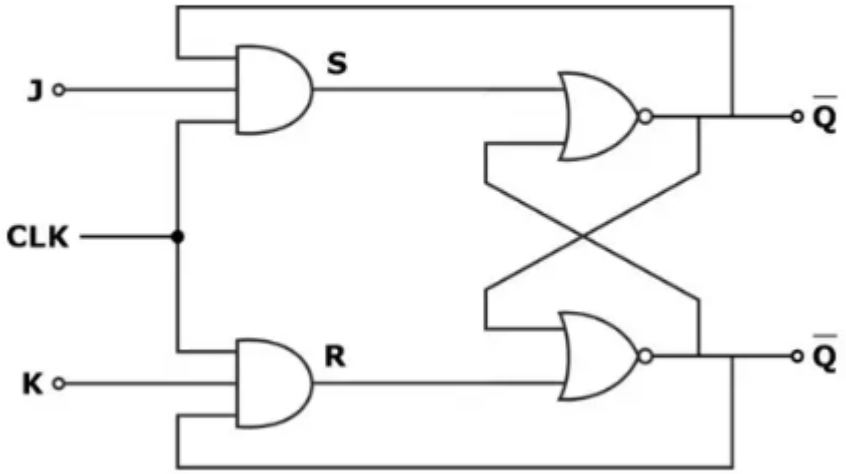
It is one kind of sequential logic circuit which stores binary information in bitwise manner. It consists of two inputs and two outputs. Inputs are Set(J) & Reset(K) and their corresponding outputs are Q and Q'. JK flipflop has two modes of operation which are synchronous mode and asynchronous mode. In synchronous mode, the state will be changed with the clock(clk) signal, and in asynchronous mode, the change of state is independent from its clock signal.

Block Diagram:



Clock	J	K	Q_{n+1}	State
0	x	x	Q_n	
1	0	0	Q_n	Hold
1	0	1	0	Reset
1	1	1	1	Set
1	1	1	$\overline{Q}_n$	Toggle

$S = J\overline{Q}$ and $R = KQ$



Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	0
1	0	X	1
1	1	X	0

- Case-A: When, $Q_n = 0$ and $Q_{n+1} = 0$
  This condition can happen with either J = 0 and K = 0 or J = 0 and K = 1

(Characteristic table)
Therefore, the desired output $Q_{n+1} = 0$ is obtained when J= 0 and K= X (don't care).

- Case-B: When, $Q_n = 0$ and $Q_{n+1} = 1$
  This can happen with either J = 1 and K = 0 or J= 1 and K= 1 (toggle condition), which means in the toggle mode a jk flip-flop has J= 1 and K= 1.
  Therefore the desired output $Q_{n+1} = 1$ is obtained when J= 1 and K=X (don't care).
- Case-C: When, $Q_n = 1$ and $Q_{n+1} = 0$
  This can happen with either J=0 and K= 1 or J= 1 and K=1.
- Therefore, the desired output $Q_{n+1} = 0$ is obtained when J= X (don't care) and K=1.
- Case-D: When, $Q_n = 1$ and $Q_{n+1} = 1$
  This condition can happen with either J= 0 and K= 0 or J= 1 and K=0.
  Thus, the desired output $Q_{n+1} = 1$ is obtained with J = X and K=0.

Procedure:

Output:

Experiment 9: Realization of single digit Seven segment display using the BCD to seven segment decoders.

AIM: Realizing the implementation of Seven segment display using the BCD to seven segment decoders.

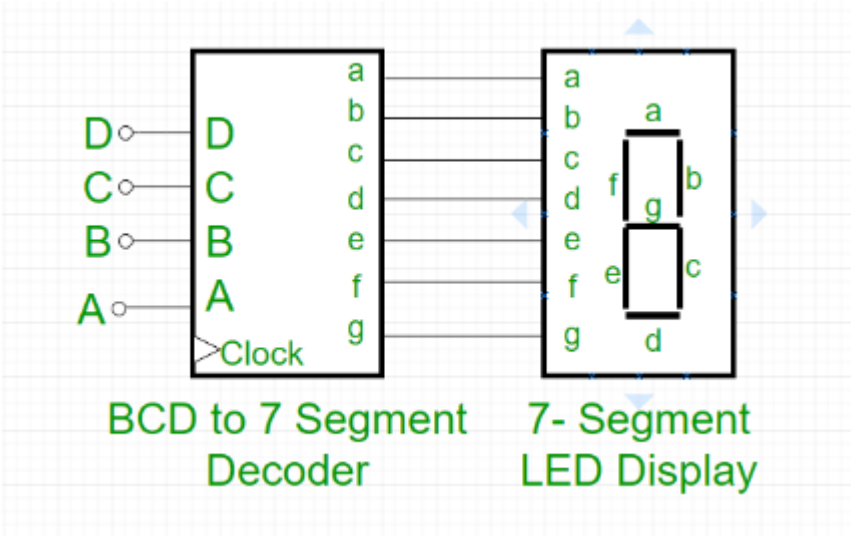
COMPONANT REQUIRED:

THEORY:

In **Binary Coded Decimal (BCD)** encoding scheme each of the decimal numbers(0-9) is represented by its equivalent binary pattern(which is generally of 4-bits).
Whereas, **Seven segment** display is an electronic device which consists of seven Light Emitting Diodes (LEDs) arranged in a some definite pattern (common cathode or common anode type), which is used to display Hexadecimal numerals(in this case decimal numbers,as input is BCD i.e., 0-9).

But, seven segment display does not work by directly supplying voltage to different segments of LEDs. First, our decimal number is changed to its BCD equivalent signal then BCD to seven segment decoder converts that signals to the form which is fed to seven segment display.

This BCD to seven segment decoder has four input lines (A, B, C and D) and 7 output lines (a, b, c, d, e, f and g), this output is given to seven segment LED display which displays the decimal number depending upon inputs.

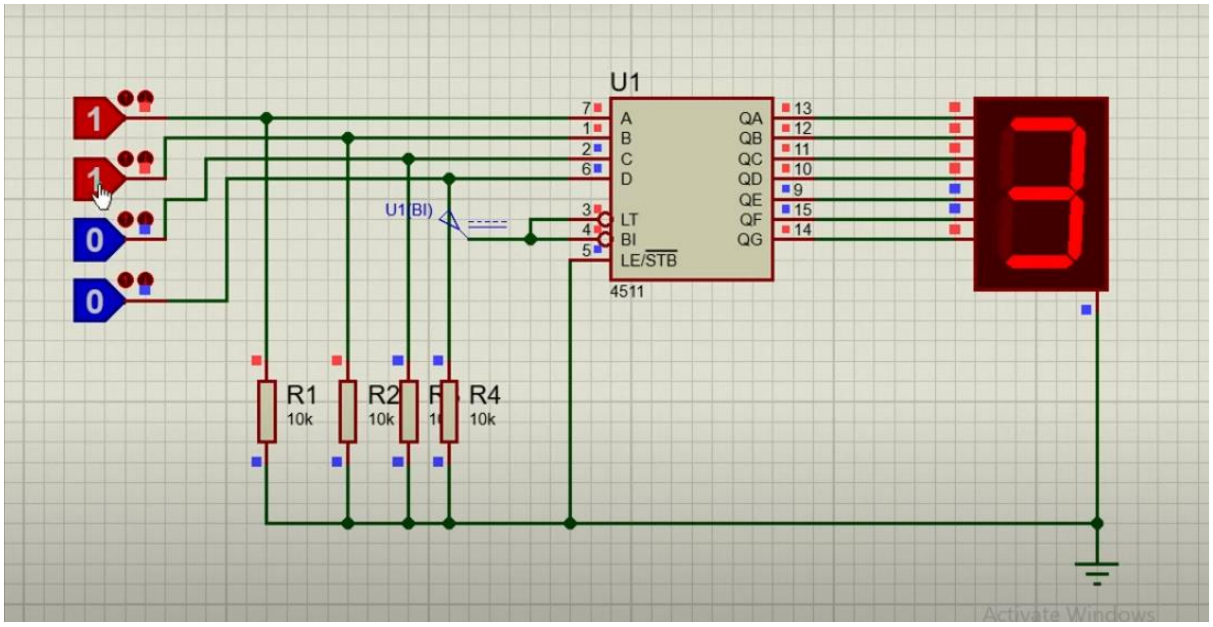


Truth Table – For common cathode type BCD to seven segment decoder:

A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

Procedure:

Output:



Experiment 10: Realization of Ring counter.

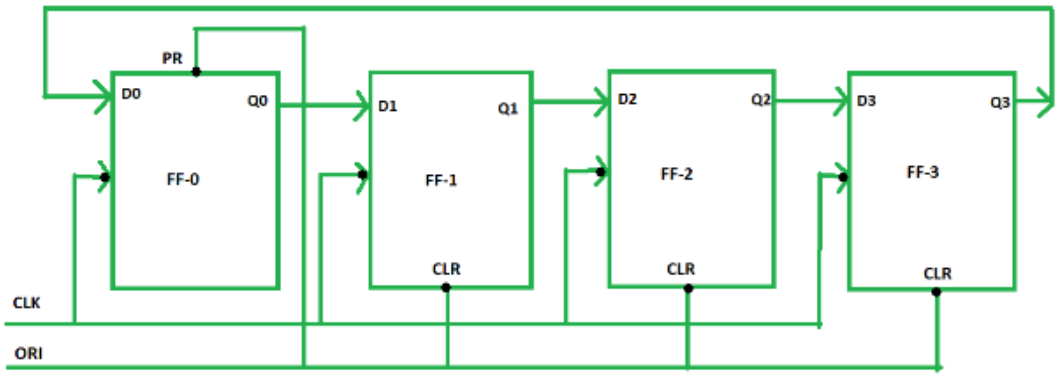
AIM: To implement Ring Counter using D Flip Flop.

COMPONANT REQUIRED: 7474 (D flip flop), LOGICSTATE, LOGICPROBE, CLOCK, GROUND.

THEORY:

A ring counter is a typical application of the Shift register. The ring counter is almost the same as the shift counter. The only change is that the output of the last flip-flop is connected to the input of the first flip-flop in the case of the ring counter but in the case of the shift register it is taken as output. Except for this, all the other things are the same.

No. of states in Ring counter = No. of flip-flop used



Ring Counter

In this diagram, we can see that the clock pulse (CLK) is applied to all the flip-flops simultaneously. Therefore, it is a Synchronous Counter. Also, here we use Overriding input (ORI) for each flip-flop. Preset (PR) and Clear (CLR) are used as ORI. When PR is 0, then the output is 1. And when CLR is 0, then the output is 0. Both PR and CLR are active low signal that always works in value 0.

PR = 0, Q = 1

CLR = 0, Q = 0

These two values are always fixed. They are independent of the value of input D and the Clock pulse (CLK). **Working –** Here, ORI is connected to Preset (PR) in FF-0 and it is connected to Clear (CLR) in FF-1, FF-2, and FF-3. Thus, output Q = 1 is generated at FF-0, and the rest of the flip-flop generates output Q = 0. This output Q = 1 at FF-0 is known as Pre-set 1 which is used to form the ring in the Ring Counter.

PRESETED 1

ORI	CLK	Q0	Q1	Q2	Q3
low	X	1	0	0	0
high	low	0	1	0	0
high	low	0	0	1	0
high	low	0	0	0	1
high	low	1	0	0	0

This Preseted 1 is generated by making ORI low and that time Clock (CLK) becomes don't care. After that ORI is made to high and apply low clock pulse signal as the Clock (CLK) is negative edge triggered. After that, at each clock pulse, the preseted 1 is shifted to the next flip-flop and thus forms a Ring. From the above table, we can say that there are 4 states in a 4-bit Ring Counter.
4 states are:

1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1

Procedure:

Output: