DEPARTMENT OF PURE AND APPLIED PHYSICS

GURU GHASIDAS VISHWAVIDYALAYA, BILASPUR (C.G.)

(Student copy)

LAB MANUAL

Simulation and Design of Digital Circuits Components Lab

B.Sc. II Sem. (Electronics)

SESSION- ……………………

Course Code SECPL02

Credit= 1 (0+0+1)

STUDENT DETAILS

Name – ……………………………………

Class – ……………………………………..

Batch – ……………………………………..

Roll No. - …………………………………….

DEPARTMENT OF PURE AND APPLIED PHYSICS

GURU GHASIDAS VISHWAVIDYALAYA, BILASPUR (C.G.)

B.Sc. (HON'S) ELECTRONICS II SEM

(Simulation and Design of Digital Circuit Components Lab)

List Of Experiment

S. NO.	EXPERIMENT NAME	PAGE NO.
1.	Design the OR, AND & NOT Gate circuits using software and verify with experiments.	
2.	Design the NAND Gate circuit using software and verify with experiments.	
3.	Design the NOR Gate circuit using software and verify with experiments.	
4.	Design the Half-Adder using NAND Gate using software and verify with experiments.	
5.	Design the Full-Adder using NAND Gate using software and verify with experiments.	
6.	Design the Comparator using NAND Gate using software and verify with experiments.	

Experiment No. 1

Aim: Design the OR, AND & NOT Gate circuits using software and verify with experiments.
Components: IC 7400, 7402, 7404,7408,7432,7486.
Software: MATLAB Software.
Theory :

OR, AND, and NOT Gates are **basic gates**. Basically, logic gates are electronic circuits because they are made up of number of electronic devices and components. Inputs and outputs of logic gates can occur only in two levels. These two levels are term as HIGH & LOW, or TRUE & FALSE, or ON & OFF, or simply 1 & 0. A table which lists all possible combinations of input variables and the corresponding outputs is called a "truth table". It shows how the logic circuit's output responds to various combinations of logic levels at the inputs.

AND GATE:

An AND gate has two or more inputs but only one output. The output assumes the logic 1 state only when each one of its inputs is at logic 1 state. The output assumes logic 0 state even if one of its inputs is at logic 0 state. AND gate is also called an "all or nothing" gate.

The logic symbol & truth table of two input AND gate are shown in figure 1.a & 1.b respectively. The symbol for AND operation is ". (Dot)".

With input variables A & B the Boolean expression for output can be written as;

X = A.B

Logic symbol:

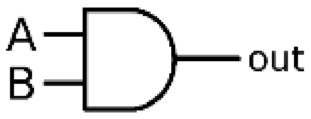


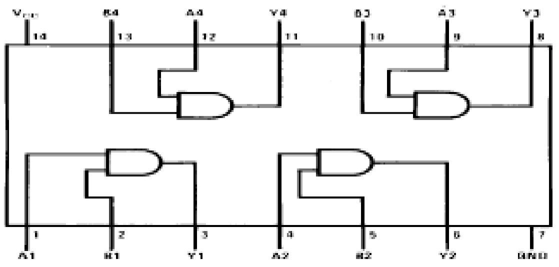
Fig.1.a

Truth table:

Input		Output
A	B	X
0	0	0
0	1	0
1	0	0

Fig.1.b

Pin diagram of IC**74LS08**:



OR GATE:

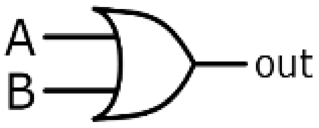
Similar to an AND gate, an OR gate may have two or more inputs but only one output. The output assumes the logic 1 state, even if one of its inputs is in logic 1 state. Its output assumes logic 0 state, only when each one of its inputs is in logic 0 state. OR gate is also called an “any or all gate”. It can also be called an inclusive OR gate because it includes the condition if both the input can be present.

The logic symbol & truth table of two input OR gate are shown in figure 1.c & 1.d respectively. The symbol for OR operation is “+”.

With input variables A & B the Boolean expression for output can be written as;

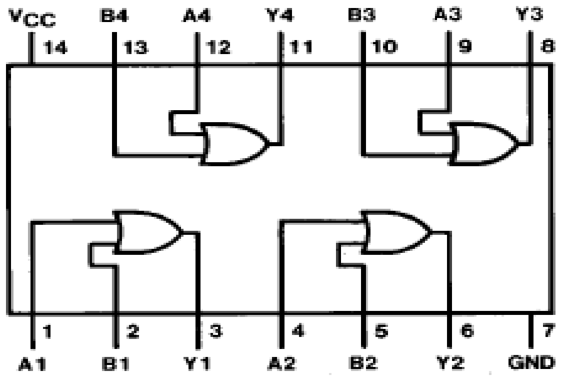
$$\mathbf{X = A + B}$$

Logic symbol: Truth table:



Input		Output
A	**B**	**X**
0	0	0
0	1	1
1	0	1
1	1	1

Fig. 1.c Fig. 1.d



Pin diagram of IC**74LS32**:

NOT GATE:

A NOT gate is also known as an inverter, has only one input and only one output. It is a device whose output is always the complement of its input. That is the output of a not gate assumes the logic 1 state when its input is in logic 0 state and assumes the logic 0 state when its input is in logic 1 state.

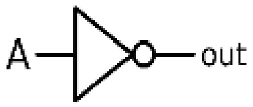
The logic symbol & truth table of NOT gate are shown in figure 1.e & 1.f respectively. The symbol for NOT operation is "‾" (bar).

With input variable A the Boolean expression for output can be written as;

$$X = \bar{A}$$

This is read as "X is equal to bar".

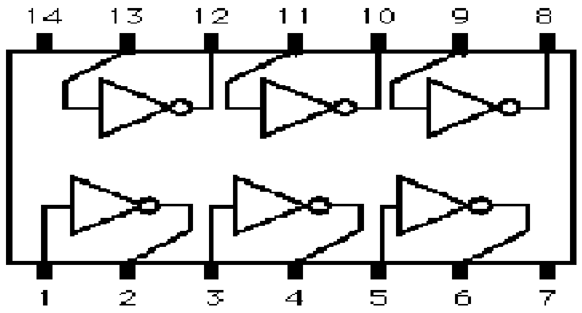
Logic symbol: Truth table:



Input	Output
A	**X**
0	1
1	0

Fig. 1.e Fig. 1.f

Pin diagram for IC**74LS04**:



To design and simulate basic logic gates (OR, AND, and NOT) in MATLAB, we use the built-in Simulink library for logic gates.

Procedure:

1 Open MATLAB software on your computer.

2 In the MATLAB command window, type `simulink` and press Enter. This will open the Simulink library browser.

3 In the Simulink library browser, search for "Logic Gates" in the search bar. You will see different logic gates available in the library.

4 Drag and drop the OR, AND, and NOT gates into the Simulink canvas.

5 Connect the gates as per the logic circuit design you want to implement.

6 Add input signals to the gates. These can be constant signals (0 or 1) or variable signals if you want to simulate dynamic behavior.

7 Set simulation parameters such as simulation time, sample time, etc., according to your requirements.

8 Run the simulation by clicking the "Run" button in the Simulink toolbar.

9 After the simulation completes, you can analyze the output signals of the gates to verify if they behave as expected according to the logic gate's truth tables.

MATLAB code to create and simulate an OR gate:

```
% Create a new Simulink model

model = 'logic_gate_simulation';

open_system(new_system(model));


% Add OR gate block

add_block('simulink/Commonly Used Blocks/Logic Gates/OR', [model '/OR Gate']);
```

```
% Add Constant blocks for inputs
add_block('simulink/Sources/Constant', [model '/Constant1']);
add_block('simulink/Sources/Constant', [model '/Constant2']);

% Connect inputs to OR gate
add_line(model, 'Constant1/1', 'OR Gate/1');
add_line(model, 'Constant2/1', 'OR Gate/2');

% Set Constant block values (Input values)
set_param([model '/Constant1'], 'Value', '0');
set_param([model '/Constant2'], 'Value', '1');

% Simulate the model
sim(model);

% Plot results
t = simout.time;
output = simout.signals.values;
plot(t, output);
xlabel('Time');
ylabel('Output');
title('OR Gate Simulation');
legend('OR Output');
```

Truth Table:

Sr. No.	Input (A)	Input(B)	Output (OR) Y=A+B	Output (AND) Y=A.B	Output (NOT) Y=Ā
1	0	0	0	0	0
2	0	1	1	0	1
3	1	0	1	0	0
4	1	1	1	1	0

We follow similar steps for AND and NOT gates by using the respective blocks from the Simulink library and connecting them accordingly.

Observation Table:

Sr. No.	Input (A)	Input (B)	Output (OR) Y=A+B	Output (AND) Y=A.B	Output (NOT) Y=Ā
1					
2					
3					
4					

Result: All the logic gates (OR, AND & NOT) simulated in MATLAB using Simulink library and their functionality is verified.

Experiment No. 2

Aim: Design the NAND Gate circuit using software and verify with experiments.
Components: IC 7400, 7402, 7404,7408,7432,7486.
Software: MATLAB Software.
NAND GATE

NAND gate is **universal gate**. It can perform all the basic logic function. NAND means NOT AND that is, AND output is NOTed. So NAND gate is combination of an AND gate and a NOT gate. The output is logic 0 level, only when each of its inputs assumes a logic 1 level. For any other combination of inputs, the output is logic 1 level. NAND gate is equivalent to a bubbled OR gate.

The logic symbol & truth table of two input NAND gate are shown in figure 2.a & 2.b respectively. With input variables A & B the Boolean expression for output can be written as;

$$X=\overline{A.B}$$

Logic symbol:

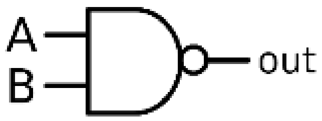


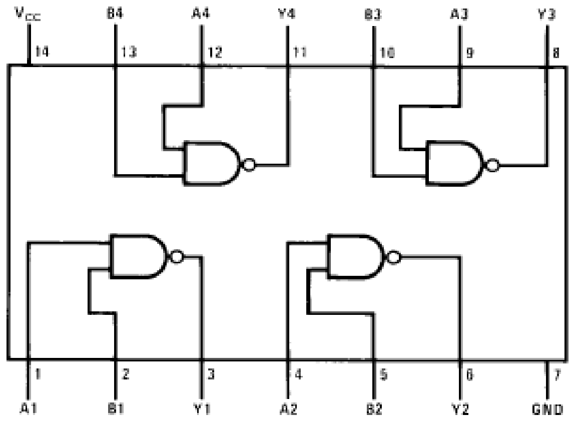
Fig.2.a

Truth table:

Input		Output
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

Fig.2.b

Pin diagram for IC**74LS00**:



To design and simulate basic NAND gate in MATLAB, we use the built-in Simulink library for logic gates.

Procedure:

1 Open MATLAB software on your computer.

2 In the MATLAB command window, type `simulink` and press Enter. This will open the Simulink library browser.

3 In the Simulink library browser, search for "Logic Gates" in the search bar. You will see different logic gates available in the library.

4 Drag and drop NAND gate into the Simulink canvas.

5 Connect the gates as per the logic circuit design you want to implement.

6 Add input signals to the gates. These can be constant signals (0 or 1) or variable signals if you want to simulate dynamic behavior.

7 Set simulation parameters such as simulation time, sample time, etc., according to your requirements.

8 Run the simulation by clicking the "Run" button in the Simulink toolbar.

9 After the simulation completes, you can analyze the output signals of the gates to verify if they behave as expected according to the logic gate's truth tables.

MATLAB code to create and simulate NAND gate:

```
% Create a new Simulink model

model = 'nand_gate_simulation';

open_system(new_system(model));


% Add NAND gate block

add_block('simulink/Commonly Used Blocks/Logic Gates/NAND', [model '/NAND Gate']);


% Add Constant blocks for inputs

add_block('simulink/Sources/Constant', [model '/Constant1']);

add_block('simulink/Sources/Constant', [model '/Constant2']);
```

```
% Connect inputs to NAND gate
add_line(model, 'Constant1/1', 'NAND Gate/1');
add_line(model, 'Constant2/1', 'NAND Gate/2');

% Set Constant block values (Input values)
set_param([model '/Constant1'], 'Value', '0');
set_param([model '/Constant2'], 'Value', '1');

% Simulate the model
sim(model);

% Plot results
t = simout.time;
output = simout.signals.values;
plot(t, output);
xlabel('Time');
ylabel('Output');
title('NAND Gate Simulation');
legend('NAND Output');
```

Truth Table:

Sr. No.	Input (A)	Input (B)	Output NAND $Y=\overline{A.B}$
1	0	0	1
2	0	1	1
3	1	0	1

4	1	1	0

Observation Table:

Sr. No.	Input (A)	Input (B)	Output NAND $Y=\overline{A.B}$
1			
2			
3			
4			

Result: NAND gate simulated in MATLAB using Simulink library and their functionality is verified.

Experiment No. 3

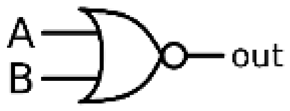
Aim: Design the NOR Gate circuit using software and verify with experiments.

Components: IC 7400, 7402, 7404,7408,7432,7486.
Software: MATLAB Software.
NOR GATE:

NOR gate is also a universal gate. It can perform all the basic logic function. NOR means NOT OR that is, OR output is NOTed. So NOR gate is combination of an OR gate and a NOT gate. The output is logic 1 level, only when each of its inputs assumes a logic 0 level. For any other combination of inputs, the output is logic 0 level. NOR gate is equivalent to a bubbled AND gate. The logic symbol & truth table of two inputs NOR gate are shown in figure 3.a & 3.b respectively. With input variables A & B the Boolean expression for output can be written as;

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

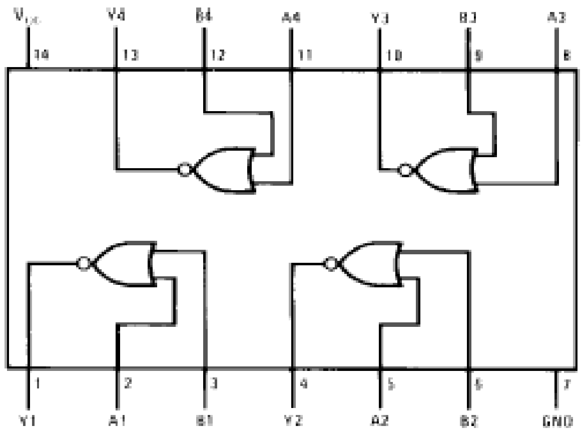
Symbol Truth table



Input		Output
A	**B**	**X**
0	0	1
0	1	0
1	0	0
1	1	0

Fig.3.a Fig.3.b

Pin diagram for IC**74LS02**:



To design and simulate basic NOR gate in MATLAB, we use the built-in Simulink library for logic gates.

Procedure:

1 Open MATLAB software on your computer.

2 In the MATLAB command window, type `simulink` and press Enter. This will open the Simulink library browser.

3 In the Simulink library browser, search for "Logic Gates" in the search bar. You will see different logic gates available in the library.

4 Drag and drop NOR gate into the Simulink canvas.

5 Connect the gates as per the logic circuit design you want to implement.

6 Add input signals to the gates. These can be constant signals (0 or 1) or variable signals if you want to simulate dynamic behavior.

7 Set simulation parameters such as simulation time, sample time, etc., according to your requirements.

8 Run the simulation by clicking the "Run" button in the Simulink toolbar.

9 After the simulation completes, you can analyze the output signals of the gates to verify if they behave as expected according to the logic gate's truth tables.

MATLAB code to create and simulate NOR gate:

% Create a new Simulink model

```
model = 'nor_gate_simulation';
open_system(new_system(model));

% Add NOR gate block
add_block('simulink/Commonly Used Blocks/Logic Gates/NOR', [model '/NOR Gate']);

% Add Constant blocks for inputs
add_block('simulink/Sources/Constant', [model '/Constant1']);
add_block('simulink/Sources/Constant', [model '/Constant2']);

% Connect inputs to NOR gate
add_line(model, 'Constant1/1', 'NOR Gate/1');
add_line(model, 'Constant2/1', 'NOR Gate/2');

% Set Constant block values (Input values)
set_param([model '/Constant1'], 'Value', '0');
set_param([model '/Constant2'], 'Value', '1');

% Simulate the model
sim(model);

% Plot results
t = simout.time;
output = simout.signals.values;
plot(t, output);
xlabel('Time');
ylabel('Output');
title('NOR Gate Simulation');
```

```
legend('NOR Output');
```

Truth Table:

Sr. No.	Input (A)	Input (B)	Output NOR $Y=\overline{A+B}$
1	0	0	1
2	0	1	0
3	1	0	0
4	1	1	0

Observation Table:

Sr. No.	Input (A)	Input (B)	Output NOR $Y=\overline{A+B}$
1			
2			
3			
4			

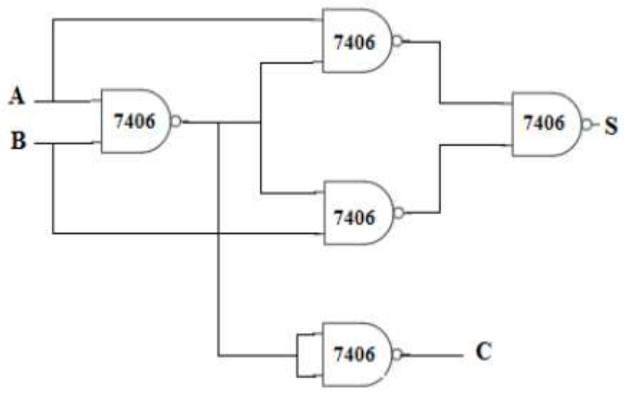
Result: NOR gate simulated in MATLAB using Simulink library and their functionality is verified.

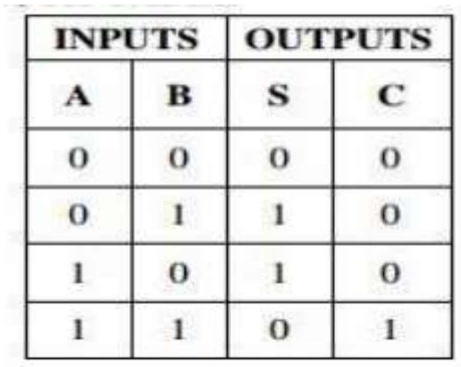
Experiment No. 4

Aim: Design the Half adder using NAND Gate using software and verify with experiments.

Components: IC 7400, 7402, 7404,7406,7432,7486.
Software: MATLAB Software.
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: $S = A \oplus B$ $C = A.B$



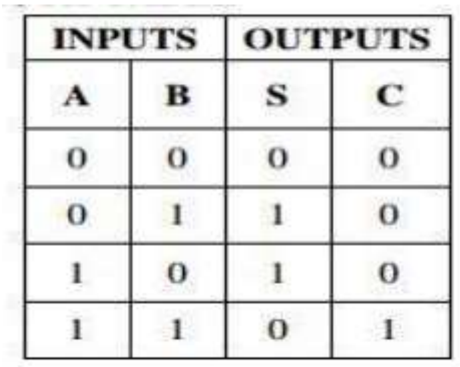


INPUTS		OUTPUTS	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Fig. 4.a circuit diagram of half-adder using NAND gates. Fig. 4.b Truth table

To design a half adder using NAND gates in MATLAB and verify it with experiments, we'll first create a truth table for a half adder. Then, we'll implement the logic using NAND gates in Simulink, a tool in MATLAB, and simulate the circuit.

Here's the truth table for a half adder:



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

Based on the truth table, we'll implement the half adder logic using NAND gates. The sum (S) output can be obtained by NANDing the NANDed inputs A and B with the NANDed inputs A and B inverted. The carry (C) output can be obtained by NANDing the inputs A and B directly.

Below is the MATLAB code to design and simulate a half adder using NAND gates:

```
% Create a new Simulink model
model = 'half_adder_simulation';
open_system(new_system(model));

% Add NAND gate blocks
add_block('simulink/Commonly Used Blocks/Logic Gates/NAND', [model '/NAND1']);
add_block('simulink/Commonly Used Blocks/Logic Gates/NAND', [model '/NAND2']);
add_block('simulink/Commonly Used Blocks/Logic Gates/NAND', [model '/NAND3']);

% Add Constant blocks for inputs A and B
add_block('simulink/Sources/Constant', [model '/ConstantA']);
add_block('simulink/Sources/Constant', [model '/ConstantB']);

% Connect Constant blocks to NAND gates
add_line(model, 'ConstantA/1', 'NAND1/1');
add_line(model, 'ConstantB/1', 'NAND2/2');

% Add invert blocks
add_block('simulink/Commonly Used Blocks/Logic Gates/Inverter', [model '/InverterA']);
add_block('simulink/Commonly Used Blocks/Logic Gates/Inverter', [model '/InverterB']);
```

```
% Connect invert blocks to NAND gates
add_line(model, 'ConstantA/1', 'InverterA/1');
add_line(model, 'ConstantB/1', 'InverterB/1');

% Connect invert blocks to NAND gates
add_line(model, 'InverterA/1', 'NAND1/2');
add_line(model, 'InverterB/1', 'NAND2/1');

% Connect NAND outputs to NAND gate for SUM
add_line(model, 'NAND1/3', 'NAND3/1');
add_line(model, 'NAND2/3', 'NAND3/2');

% Add scope blocks
add_block('simulink/Commonly Used Blocks/Scope', [model '/ScopeSum']);
add_block('simulink/Commonly Used Blocks/Scope', [model '/ScopeCarry']);

% Connect NAND output to scope for SUM
add_line(model, 'NAND3/3', 'ScopeSum/1');

% Connect NAND1 and NAND2 outputs to NAND gate for CARRY
add_line(model, 'NAND1/3', 'ScopeCarry/1');

% Set Constant block values (Input values)
set_param([model '/ConstantA'], 'Value', '0');
set_param([model '/ConstantB'], 'Value', '1');

% Simulate the model
sim(model);
```

```
% Plot results
t = simout.time;
sum_output = simout.signals(1).values;
carry_output = simout.signals(2).values;

figure;
subplot(2,1,1);
plot(t, sum_output);
xlabel('Time');
ylabel('Sum (S)');
title('Half Adder Sum (S) Output');

subplot(2,1,2);
plot(t, carry_output);
xlabel('Time');
ylabel('Carry (C)');
title('Half Adder Carry (C) Output');
```

This code creates a half adder using NAND gates in Simulink. It connects the inputs A and B to NAND gates and their respective inverters. The NAND gate outputs are then connected to obtain the sum and carry outputs. Finally, the outputs are plotted using scope blocks.

Run this code in MATLAB to simulate and verify the behavior of the half adder circuit using NAND gates. Adjust the input values and experiment with different configurations to validate the half adder's functionality.

Observation Table:

Sr. No.	INPUTS		OUTPUTS	
	A	B	S	C
1				
2				
3				
4				

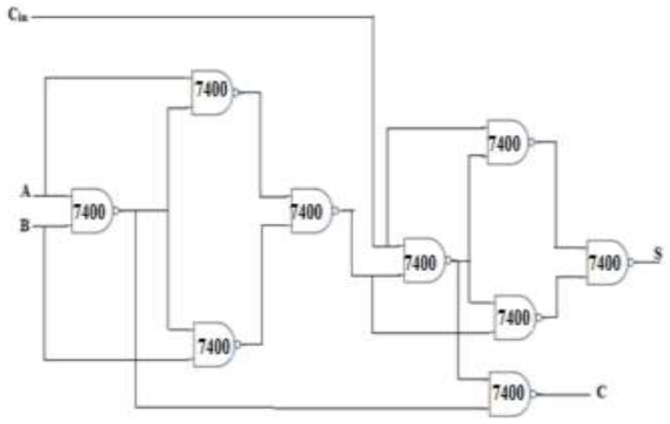
Result: Half adder circuit simulated in MATLAB and their functionality is verified.

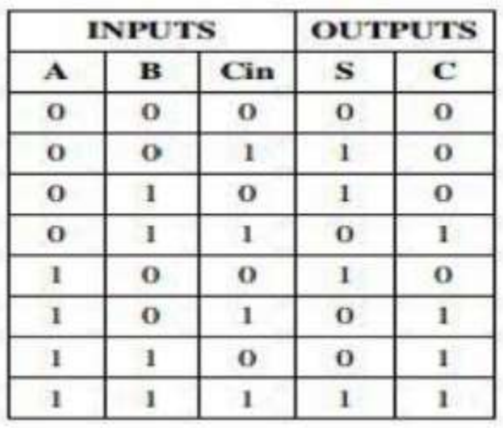
Experiment No. 5

Aim: Design the Full adder using NAND Gate using software and verify with experiments.

Components: IC 7400, 7402, 7404,7408,7432,7486.
Software: MATLAB Software.
Theory:
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:

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





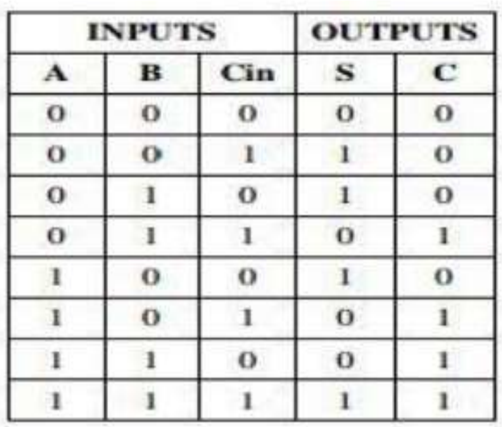
INPUTS			OUTPUTS	
A	B	Cin	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Fig. Circuit diagram of Full adder using NAND gate. Fig. Truth table.

Procedure:-

To design a Full Adder using NAND gates in MATLAB and verify it with experiments, we'll first create a truth table for a Full Adder. Then, we'll implement the logic using NAND gates in Simulink, a tool in MATLAB, and simulate the circuit.

<u>Here's the truth table for a Full Adder:</u>



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

Based on the truth table, we'll implement the Full Adder logic using NAND gates. The sum (S) output can be obtained by NANDing the NANDed inputs A, B, and Carry in with the NANDed inputs A, B, and Carry in inverted. The Carry Out (C) output can be obtained by NANDing the Carry in with the NANDed inputs A and B, and NANDing the NANDed inputs A and B with the NANDed inputs A, B, and Carry In.

Below is the MATLAB code to design and simulate a Full Adder using NAND gates:

```
% Create a new Simulink model
model = 'full_adder_simulation';
open_system(new_system(model));

% Add NAND gate blocks
add_block('simulink/Commonly Used Blocks/Logic Gates/NAND', [model '/NAND1']);
add_block('simulink/Commonly Used Blocks/Logic Gates/NAND', [model '/NAND2']);
add_block('simulink/Commonly Used Blocks/Logic Gates/NAND', [model '/NAND3']);
add_block('simulink/Commonly Used Blocks/Logic Gates/NAND', [model '/NAND4']);
add_block('simulink/Commonly Used Blocks/Logic Gates/NAND', [model '/NAND5']);
add_block('simulink/Commonly Used Blocks/Logic Gates/NAND', [model '/NAND6']);

% Add Constant blocks for inputs A, B, and Carry In
add_block('simulink/Sources/Constant', [model '/ConstantA']);
add_block('simulink/Sources/Constant', [model '/ConstantB']);
add_block('simulink/Sources/Constant', [model '/ConstantC']);

% Connect Constant blocks to NAND gates
add_line(model, 'ConstantA/1', 'NAND1/1');
add_line(model, 'ConstantB/1', 'NAND1/2');
add_line(model, 'ConstantC/1', 'NAND4/2');
```

```
% Add invert blocks
add_block('simulink/Commonly Used Blocks/Logic Gates/Inverter', [model '/InverterA']);
add_block('simulink/Commonly Used Blocks/Logic Gates/Inverter', [model '/InverterB']);
add_block('simulink/Commonly Used Blocks/Logic Gates/Inverter', [model '/InverterC']);

% Connect invert blocks to NAND gates
add_line(model, 'ConstantA/1', 'InverterA/1');
add_line(model, 'ConstantB/1', 'InverterB/1');
add_line(model, 'ConstantC/1', 'InverterC/1');

% Connect invert blocks to NAND gates
add_line(model, 'InverterA/1', 'NAND2/2');
add_line(model, 'InverterB/1', 'NAND2/1');
add_line(model, 'InverterC/1', 'NAND5/1');

% Connect NAND outputs to NAND gates for SUM
add_line(model, 'NAND1/3', 'NAND3/1');
add_line(model, 'NAND2/3', 'NAND3/2');

% Connect NAND outputs to NAND gates for CARRY OUT
add_line(model, 'NAND1/3', 'NAND4/1');
add_line(model, 'NAND5/3', 'NAND4/2');

% Connect NAND outputs to NAND gates for SUM
add_line(model, 'NAND4/3', 'NAND6/1');
add_line(model, 'NAND3/3', 'NAND6/2');
```

```
% Add scope blocks
add_block('simulink/Commonly Used Blocks/Scope', [model '/ScopeSum']);
add_block('simulink/Commonly Used Blocks/Scope', [model '/ScopeCarryOut']);

% Connect NAND output to scope for SUM
add_line(model, 'NAND6/3', 'ScopeSum/1');

% Connect NAND1 and NAND2 outputs to NAND gate for CARRY OUT
add_line(model, 'NAND4/3', 'ScopeCarryOut/1');

% Set Constant block values (Input values)
set_param([model '/ConstantA'], 'Value', '0');
set_param([model '/ConstantB'], 'Value', '1');
set_param([model '/ConstantC'], 'Value', '1');

% Simulate the model
sim(model);

% Plot results
t = simout.time;
sum_output = simout.signals(1).values;
carry_out_output = simout.signals(2).values;

figure;
subplot(2,1,1);
plot(t, sum_output);
xlabel('Time');
ylabel('Sum (S)');
```

```
title('Full Adder Sum (S) Output');


subplot(2,1,2);

plot(t, carry_out_output);

xlabel('Time');

ylabel('Carry Out (C)');

title('Full Adder Carry Out (C) Output');
```

This code creates a full adder using NAND gates in Simulink. It connects the inputs A, B, and Carry in to NAND gates and their respective inverters. The NAND gate outputs are then connected to obtain the sum and carry out outputs. Finally, the outputs are plotted using scope blocks.

Run this code in MATLAB to simulate and verify the behavior of the full adder circuit using NAND gates. Adjust the input values and experiment with different configurations to validate the full adder's functionality.

Observation Table:

Sr. No.	INPUTS			OUTPUTS	
	A	B	C_{in}	S	C
1					
2					
3					
4					
5					
6					
7					
8					

Result: Full adder circuit simulated in MATLAB and their functionality is verified.

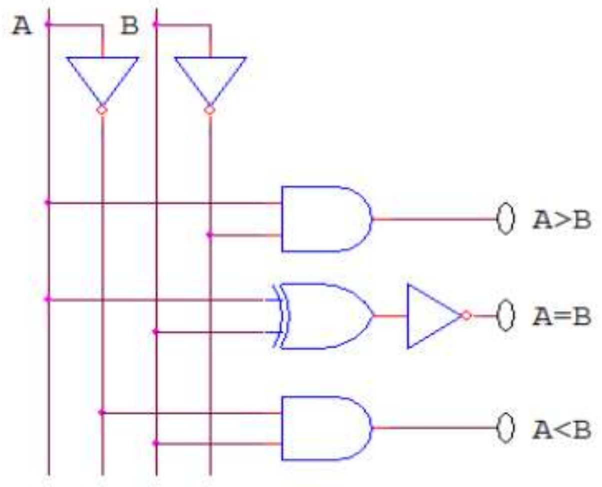
Experiment No. 6

Aim: Design the Comparator using NAND Gate using software and verify with experiments.

Components: IC 7400, 7402, 7404,7408,7432,7486.
Software: MATLAB Software.
Theory:
Magnitude Comparator is a logical circuit, which compares two signals A and B and generates three logical outputs, whether A > B, A = B, or A < B. The A = B Input must be held high for proper compare operation.

1. **1 Bit comparator**

$A>B = A\overline{B}$
$A<B = \overline{A}B$
$A=B = \overline{A}\,\overline{B}+AB$



INPUTS		OUTPUTS		
A	B	A > B	A = B	A < B
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Fig. Circuit diagram of 1-Bit comparator. Fig. Truth Table.

Procedure:
To design a 1-bit comparator using NAND gates in MATLAB and verify it with experiments, we need to define the truth table for a 1-bit comparator first. Then, we'll implement the logic using NAND gates in Simulink, a tool in MATLAB, and simulate the circuit.

Here's the truth table for a 1-bit comparator:

INPUTS		OUTPUTS		
A	B	A > B	A = B	A < B
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Based on the truth table, we'll implement the 1-bit comparator logic using NAND gates. The outputs A > B, A = B, and A < B can be obtained by NANDing the appropriate inputs and their complements.

Below is the MATLAB code to design and simulate a 1-bit comparator using NAND gates:

```
% Create a new Simulink model
model = '1bit_comparator_simulation';
open_system(new_system(model));

% Add NAND gate blocks
add_block('simulink/Commonly Used Blocks/Logic Gates/NAND', [model '/NAND1']);
add_block('simulink/Commonly Used Blocks/Logic Gates/NAND', [model '/NAND2']);
add_block('simulink/Commonly Used Blocks/Logic Gates/NAND', [model '/NAND3']);
add_block('simulink/Commonly Used Blocks/Logic Gates/NAND', [model '/NAND4']);
add_block('simulink/Commonly Used Blocks/Logic Gates/NAND', [model '/NAND5']);
add_block('simulink/Commonly Used Blocks/Logic Gates/NAND', [model '/NAND6']);
add_block('simulink/Commonly Used Blocks/Logic Gates/NAND', [model '/NAND7']);
add_block('simulink/Commonly Used Blocks/Logic Gates/NAND', [model '/NAND8']);

% Add Constant blocks for inputs A and B
add_block('simulink/Sources/Constant', [model '/ConstantA']);
add_block('simulink/Sources/Constant', [model '/ConstantB']);

% Add invert blocks
add_block('simulink/Commonly Used Blocks/Logic Gates/Inverter', [model '/InverterA']);
add_block('simulink/Commonly Used Blocks/Logic Gates/Inverter', [model '/InverterB']);

% Connect Constant blocks to NAND gates
```

```
add_line(model, 'ConstantA/1', 'NAND1/1');
add_line(model, 'ConstantB/1', 'NAND1/2');

% Connect invert blocks to NAND gates
add_line(model, 'ConstantA/1', 'InverterA/1');
add_line(model, 'ConstantB/1', 'InverterB/1');

% Connect invert blocks to NAND gates
add_line(model, 'InverterA/1', 'NAND2/2');
add_line(model, 'InverterB/1', 'NAND3/1');
add_line(model, 'InverterA/1', 'NAND4/1');
add_line(model, 'InverterB/1', 'NAND5/2');

% Connect NAND outputs to NAND gates for A > B
add_line(model, 'NAND1/3', 'NAND6/1');

% Connect NAND outputs to NAND gates for A = B
add_line(model, 'NAND2/3', 'NAND7/1');
add_line(model, 'NAND3/3', 'NAND7/2');

% Connect NAND outputs to NAND gates for A < B
add_line(model, 'NAND4/3', 'NAND8/1');

% Add scope blocks
add_block('simulink/Commonly Used Blocks/Scope', [model '/ScopeAGTB']);
add_block('simulink/Commonly Used Blocks/Scope', [model '/ScopeAEQB']);
add_block('simulink/Commonly Used Blocks/Scope', [model '/ScopeALTB']);
```

```
% Connect NAND output to scope for A > B
add_line(model, 'NAND6/3', 'ScopeAGTB/1');

% Connect NAND output to scope for A = B
add_line(model, 'NAND7/3', 'ScopeAEQB/1');

% Connect NAND output to scope for A < B
add_line(model, 'NAND8/3', 'ScopeALTB/1');

% Set Constant block values (Input values)
set_param([model '/ConstantA'], 'Value', '0');
set_param([model '/ConstantB'], 'Value', '1');

% Simulate the model
sim(model);

% Plot results
t = simout.time;
A_GT_B_output = simout.signals(1).values;
A_EQ_B_output = simout.signals(2).values;
A_LT_B_output = simout.signals(3).values;

figure;
subplot(3,1,1);
plot(t, A_GT_B_output);
xlabel('Time');
ylabel('A > B');
title('1-bit Comparator A > B Output');
```

```
subplot(3,1,2);
plot(t, A_EQ_B_output);
xlabel('Time');
ylabel('A = B');
title('1-bit Comparator A = B Output');

subplot(3,1,3);
plot(t, A_LT_B_output);
xlabel('Time');
ylabel('A < B');
title('1-bit Comparator A < B Output');
```

This code creates a 1-bit comparator using NAND gates in Simulink. It connects the inputs A and B to NAND gates and their respective inverters. The NAND gate outputs are then connected to obtain the A > B, A = B, and A < B outputs. Finally, the outputs are plotted using scope blocks.

Run this code in MATLAB to simulate and verify the behavior of the 1-bit comparator circuit using NAND gates. Adjust the input values and experiment with different configurations to validate the comparator's functionality.

Observation Table:

Sr. No.	INPUTS		OUTPUTS		
	A	B	A=B	A<B	A>B
1					
2					
3					
4					

Result: 1-Bit comparator circuit simulated in MATLAB and their functionality is verified.