RAISONI GROUP
a vision beyond

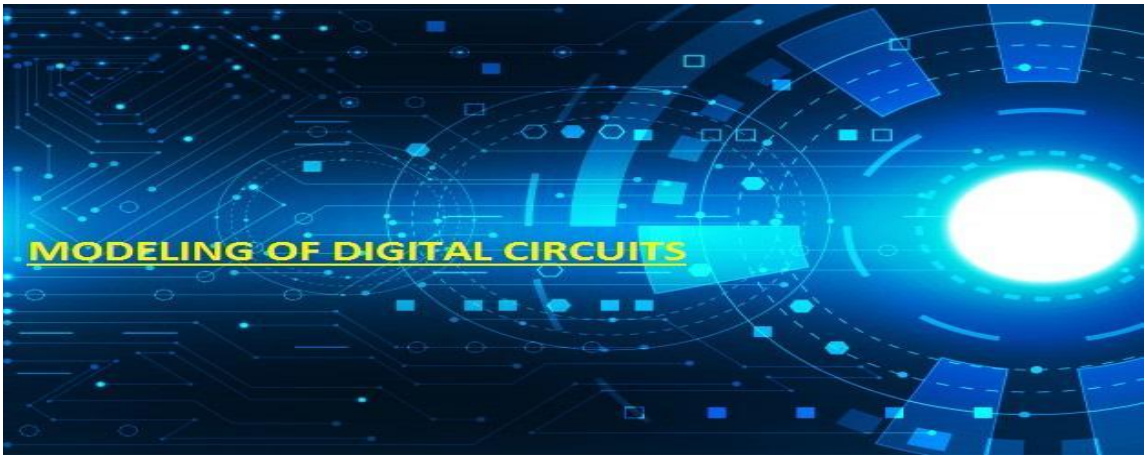
# G H RAISONI COLLEGE OF ENGINEERING & MANAGEMENT

**(An Autonomous Institute Affiliated to SPPU)**

**WAGHOLI, PUNE – 412 207**

## DEPARTMENT OF FIRST YEAR B.TECH.

## LAB MANUAL




## F.Y.B tech

**Subject Code : UECP 103**

## Academic Year 2020-21

**Name of Student : Pratik Rajesh Jade**

**Division : A Roll No. 72 Reg. No. 2020AAIT1111019**

# Department of First Year Engineering

## List of Experiments

| | |
|---|---|
| 1 | To study Basic Gates and verify its Truth Table |
| 2 | To realize Basic Gates from Universal Gates. |
| 3 | To construction Half and Full Adder using XOR, NAND Gates and verification of its truth tables. |
| 4 | To study Half and Full Subtractor |
| 5 | To study implementation & verification of Decoder , Encoder using logic gates |
| 6 | To study implementation & verification of Multiplexer & Demultiplexer using logic gates |
| 7 | To study implementation & verification of RS , JK Flip Flop using NAND , NOR Gates |
| 8 | To study implementation & verification of Synchronous or Asynchronous counter using JK Flip Flop. |
| 9 | To study Implementation and verification of One bit & Two bit comparator using Logic gates. |

**Dr. Praveen Jangade** **Mr. Bharat Sonnis**

**Dean F.Y.Btech** **Lab Incharge**

# Experiment- 1

## AIM :

To verify and interpret the logic and truth table for AND, OR, NOT, NAND, NOR, Ex-OR, Ex-NOR gates using RTL (Resistor Transistor Logic), DTL (Diode Transistor Logic) and TTL (Transistor Transistor Logic) logics in simulator 1 and verify the truth table for AND, OR, NOT, NAND, NOR, Ex-OR, Ex-NOR gates in simulator 2.

## THEORY :

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
6. Ex-OR gate
7. Ex-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

**Y= A.B**

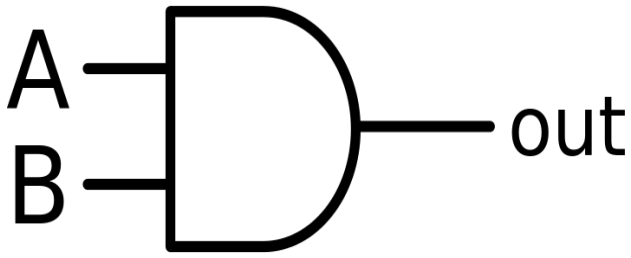
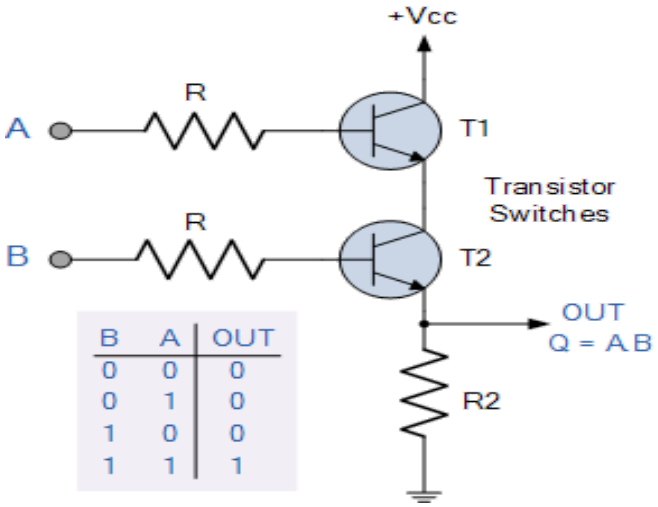



**Figure-: Logic Symbol of AND Gate**

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

**Logic Symbol of AND Gate**

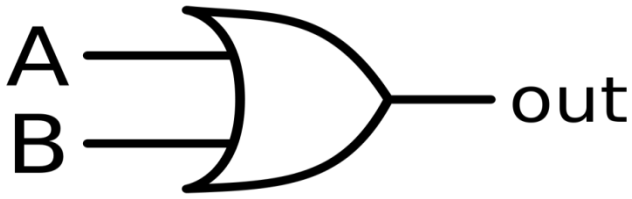
A simple 2-input logic AND gate can be constructed using RTL (Resistor-Transistor-Logic) switches connected together as shown below with the inputs connected directly to the transistor bases. Both transistors must be saturated "ON" for an output at Q.




**AND Gate through RTL logic**

## 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 OR operation. **Y= A+B**



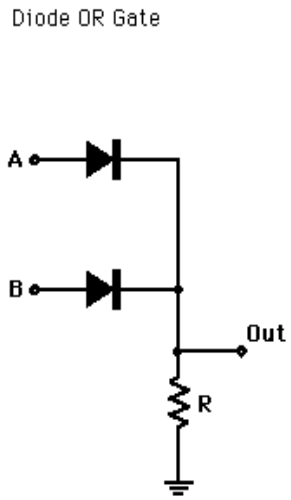

**Logic Symbol of OR Gate**

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

**Truth Table of OR Gate**

OR gate can be realized by DRL (Diode-Resistance-Logic) or by TTL (Transistor-Transistor-Logic). Presently, we will learn how to implement the OR gate using DRL (Diode-Resistance-Logic). To realise OR gate, we will use a diode at every input of the OR gate. The anode part of diode is connected with input while the cathode part is joined together and a resistor, connected with the cathode is grounded. In this case, we have taken two inputs which can be seen in the circuit below.

When both the inputs are at logic 0 or low state then the diodes D1 and D2 become reverse biased. Since the anode terminal of diode is at lower voltage level than the cathode terminal, so diode will act as open circuit so there is no voltage across resistor and hence output voltage is same as ground. When either of the diodes is at logic 1 or high state then the diode corresponding to that input is forward bias. Since this time anode is at high voltage than cathode therefore current will flow through forward biased diode and this current then appears on resistor causing high voltage at output terminal also. Hence at output we get high or logic 1 or +5V. So, if any or both inputs are high, the output will be high or “1”.

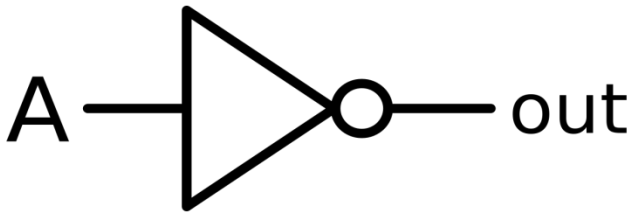



**OR Gate through DRL logic**

## 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'**




**Logic Symbol of NOT Gate**

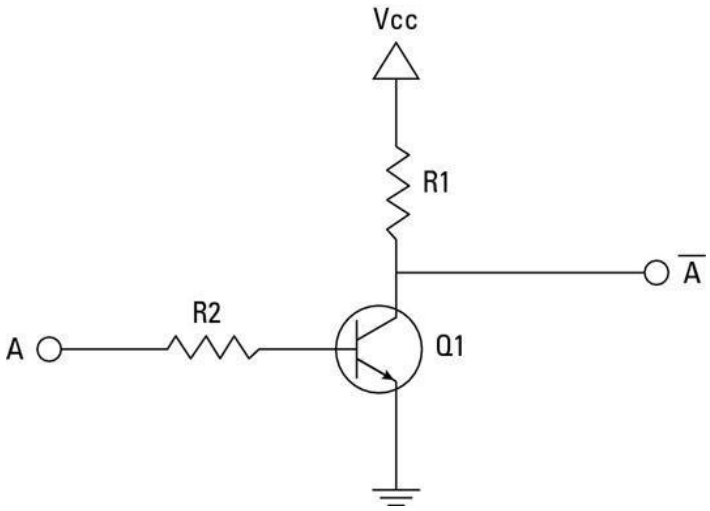
| Input | Output |
|---|---|
| A | Y |
| 0 | 1 |
| 1 | 0 |

**Truth Table of NOT Gate**

NOT gate can be realized through transistor.The input is connected through resistor R2 to the transistor's base. When no voltage is present on the input, the transistor turns off. When the transistor is off, no current flows through the collector-emitter path. Thus, current from the supply voltage (Vcc) flows through resistor R1 to the output. In this way, the circuit's output is high when its input is low.

When voltage is present at the input, the transistor turns on, allowing current to flow through the collector-emitter circuit directly to ground. This ground path creates a shortcut that bypasses the output, which causes the output to go low.

In this way, the output is high when the input is low and low when the input is high.

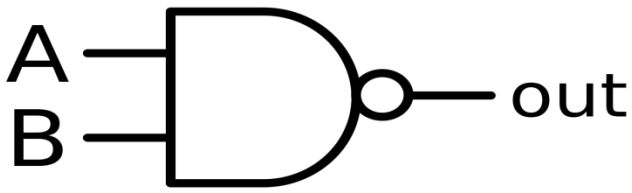



**NOT Gate through Transistor**

**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 circuit represent inversion

**Y= AB**

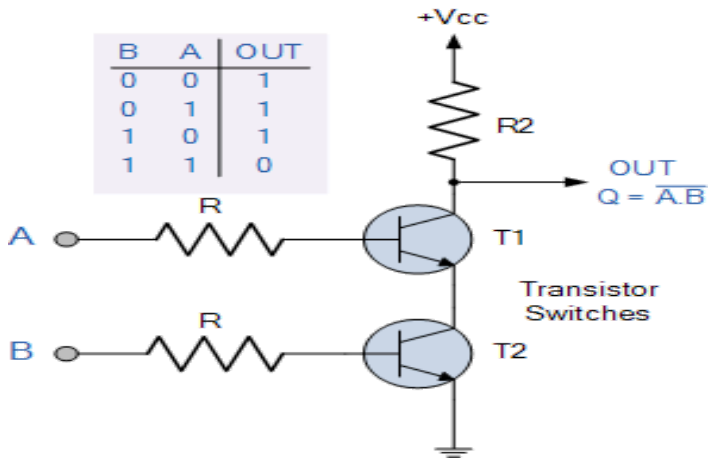



**Logic Symbol of NAND Gate**

| Input | Input | Output |
|---|---|---|
| A | B | Y |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Truth Table of NAND Gate**

A simple 2-input logic NAND gate can be constructed using RTL (Resistor-transistor-logic) switches connected together as shown below with the inputs connected directly to the transistor bases. Either transistor must be cut-off or "OFF" for an output at Q.

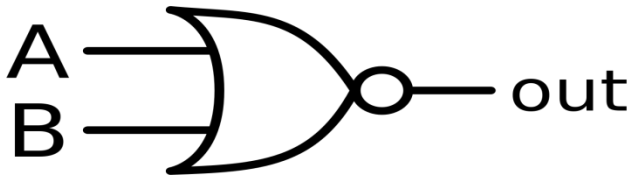



**NAND gate through RTL Logic.**

**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= A+B**

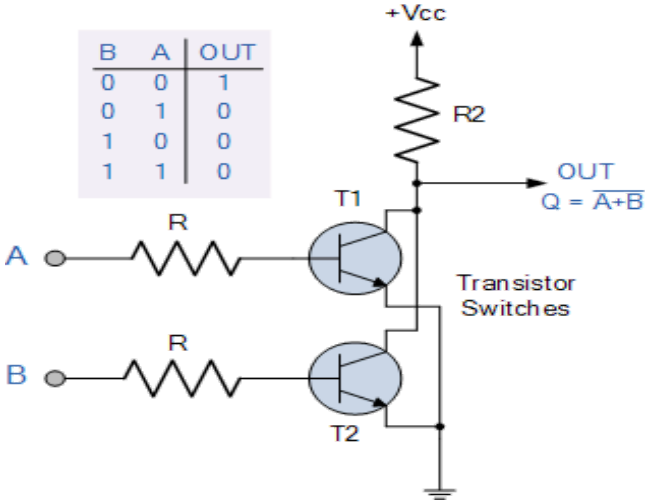



**Logic Symbol of NOR gate**

| A | B | F |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

**Truth Table of NOR gate**

A simple 2-input logic NOR gate can be constructed using RTL (Resistor-transistor-logic) switches connected together as shown below with the inputs connected directly to the transistor bases. Both transistors must be cut-off or “OFF” for an output at Q.

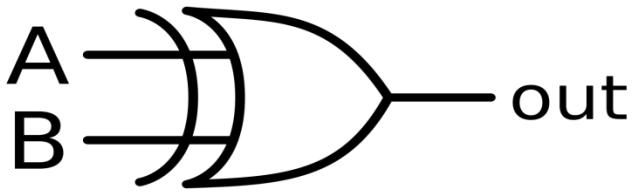



**NOR gate through RTL Logic.**

## 6) Ex-OR gate

The 'Exclusive-OR' gate is a circuit which will give a high output if either, but not both of its two inputs are high. An encircled plus sign (⊕) is used to show the Ex-OR operation.

**Y= A⊕B**

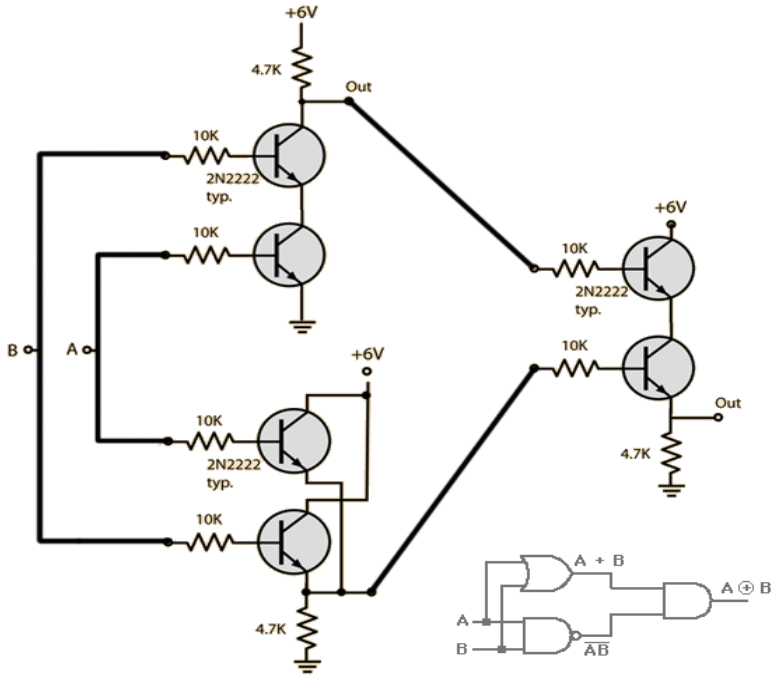



**Logic Symbol of Ex-OR gate**

| A | B | A **XOR** B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Truth Table of Ex-OR gate**

Ex-OR gate is created from AND, NAND and OR gates.The output is high only when both the inputs are different.

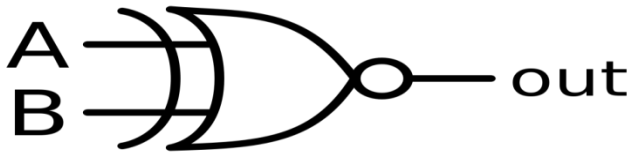



**Ex-OR gate through RTL Logic.**

## 7) Ex-NOR gate

The 'Exclusive-NOR' gate circuit does the opposite to the EX-OR gate. It will give a low output if either, but not both of its two inputs are high. The symbol is an EX-OR gate with a small circle on the output. The small circle represents inversion.

$$\mathbf{Y = A \oplus B}$$




**Logic Symbol of Ex-NOR gate**

| XNOR Truth Table | | |
|---|---|---|
| A | B | Q |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Truth Table of Ex-NOR gate**

Ex-NOR gate is created from AND, NOT and OR gates. The output is high only when both the inputs are same.

## PROCEDURE :

**Simulator 1:** AND Gate

Step-1) Connect the supply(+5V) to the circuit.
Step-2) Press the switches for inputs "A" and "B".

The switch in ON state is  and the switch in OFF state is 

Step-3) The bulb does not glow if any one or both the switches (2 and 3) are OFF and glows only if both the switches (2 and 3) are ON.The bulb in OFF state is  and the bulb in ON state is  .

Step-4) Repeat step-2 and step-3 for all state of inputs.

**Simulator 2:**

Step-1) Enter the Boolean input "A" and "B".
Step-2) Enter the Boolean output for your corresponding inputs.
Step-3) Click on "Check" Button to verify your output.
Step-4) Click "Print" if you want to get print out of Truth Table.

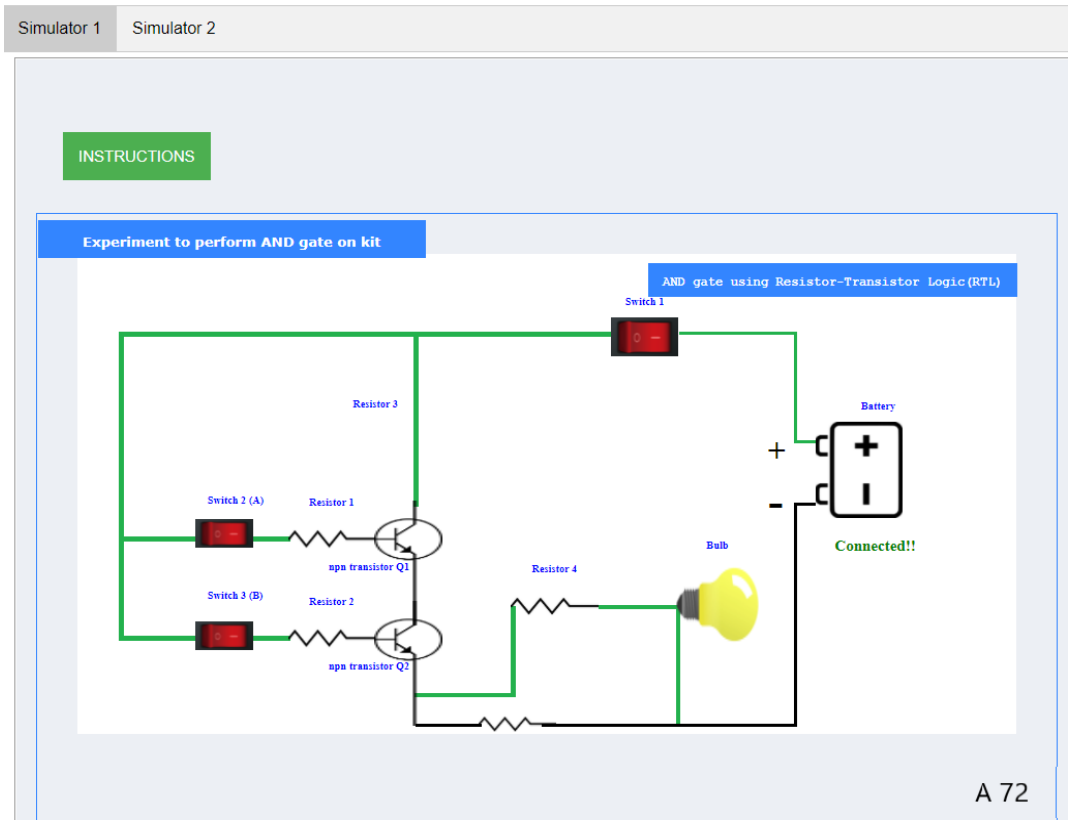
Follow the respective Gates method likewise with the help of truth table.

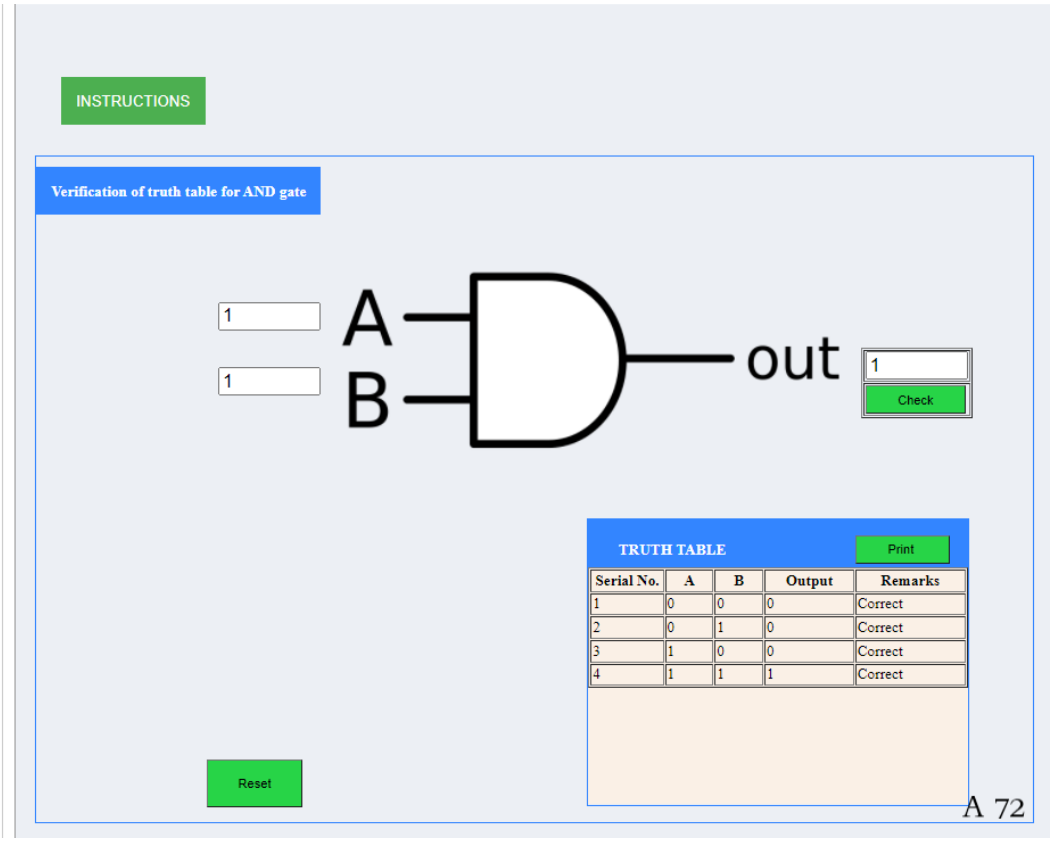
## TEST:

1) In Boolan algebra, the bar sign (-) indicates- **NOT operation**
2) The NAND gate is AND gate followed by- **NOT gate**
3) The NOR gate is OR gate followed by- **NOT gate**
4) The expression of an Ex-OR gate is- **A'B+AB'**

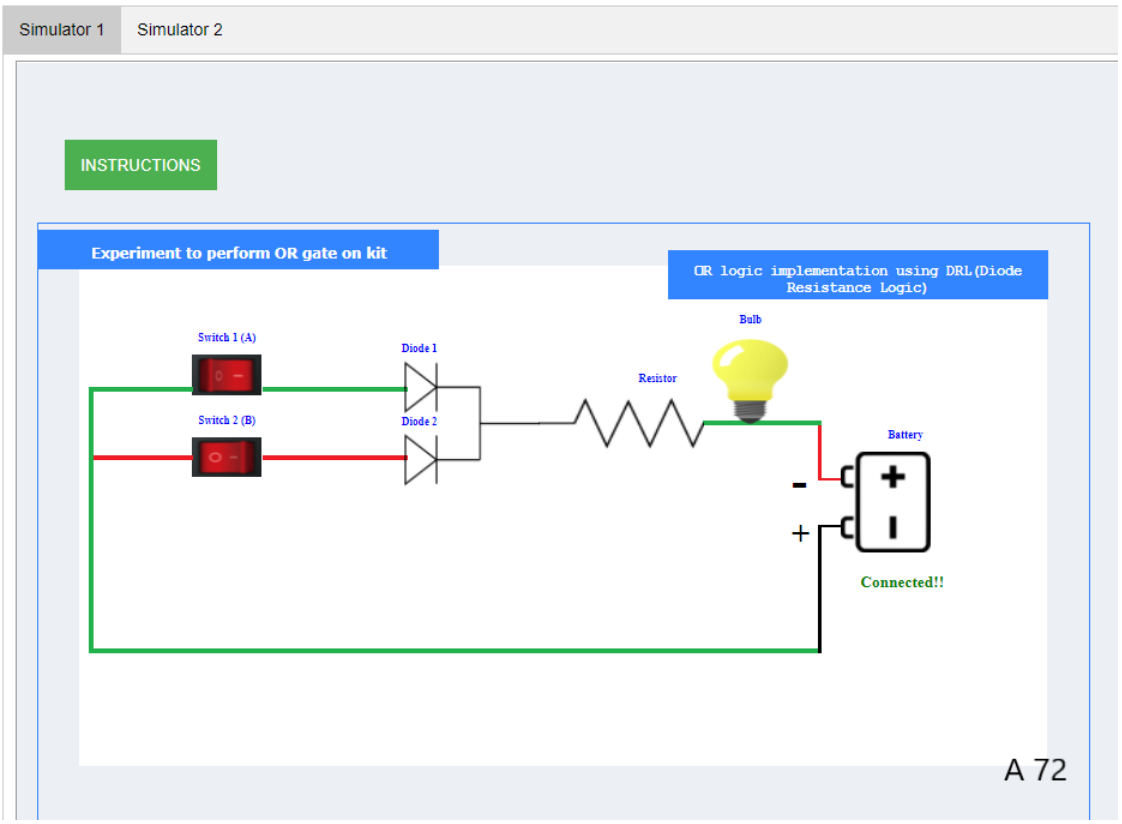
## SIMULATION RESULTS :

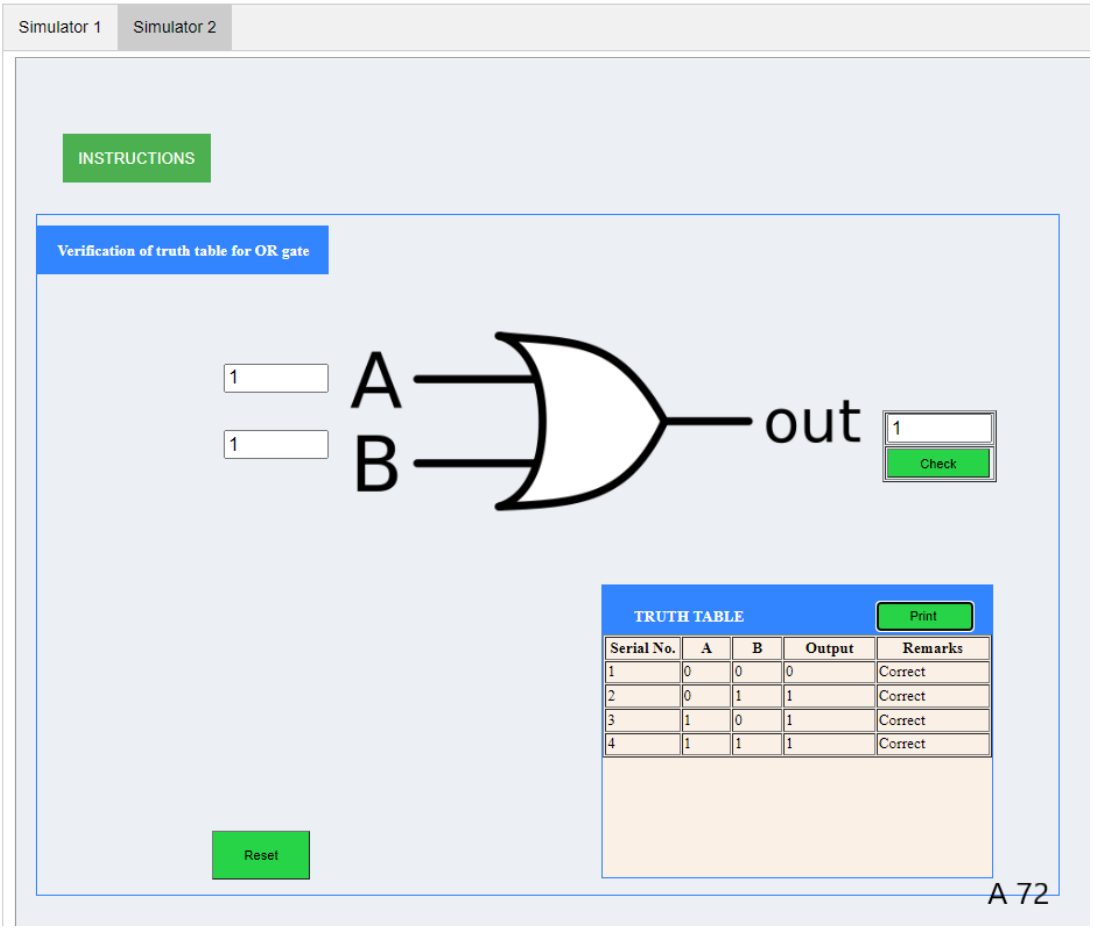
### 1) AND Gate



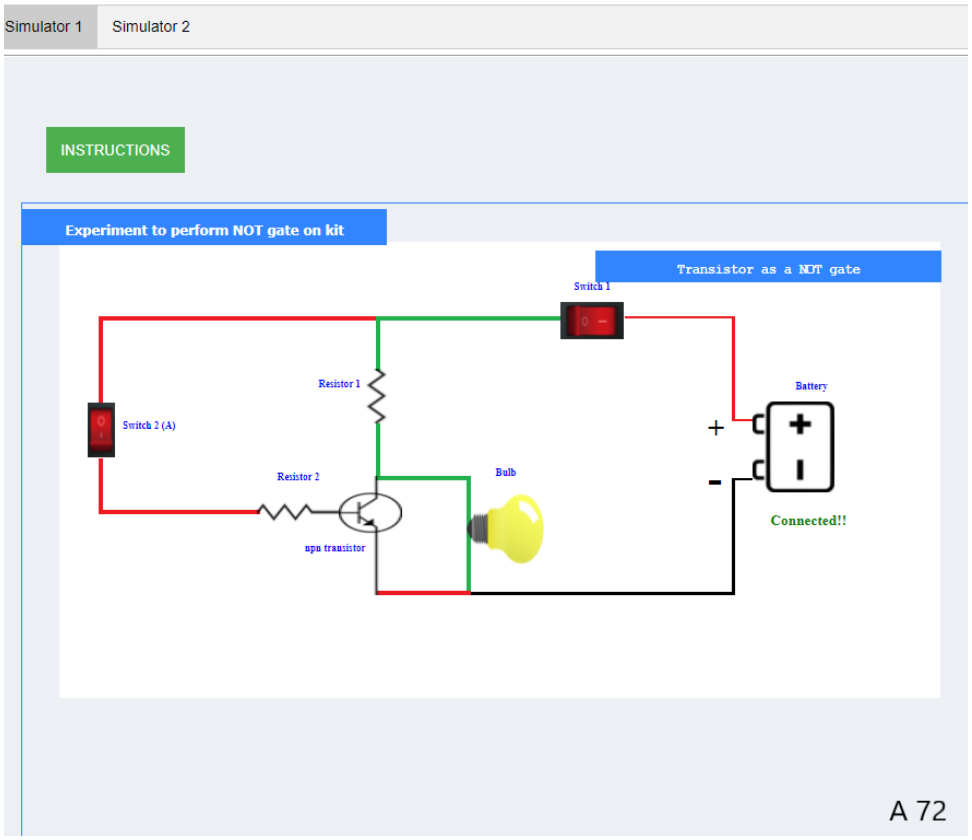

INSTRUCTIONS
Verification of truth table for AND gate
1
1
A
B
out
1
Check
TRUTH TABLE
Print
Serial No. | A | B | Output | Remarks
1 | 0 | 0 | 0 | Correct
2 | 0 | 1 | 0 | Correct
3 | 1 | 0 | 0 | Correct
4 | 1 | 1 | 1 | Correct
Reset
A 72

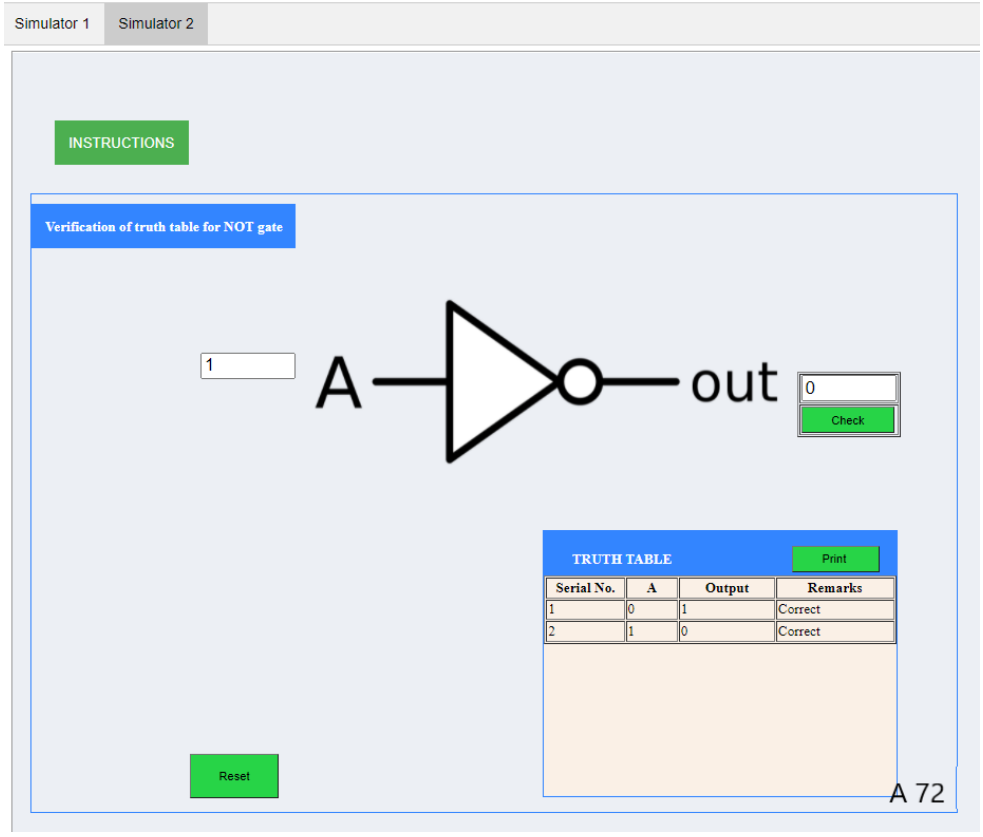
## 2) OR Gate


Simulator 1
Simulator 2
INSTRUCTIONS
Experiment to perform OR gate on kit
OR logic implementation using DRL(Diode Resistance Logic)
Switch 1 (A)
Switch 2 (B)
Diode 1
Diode 2
Resistor
Bulb
Battery
-
+
Connected!!
A 72

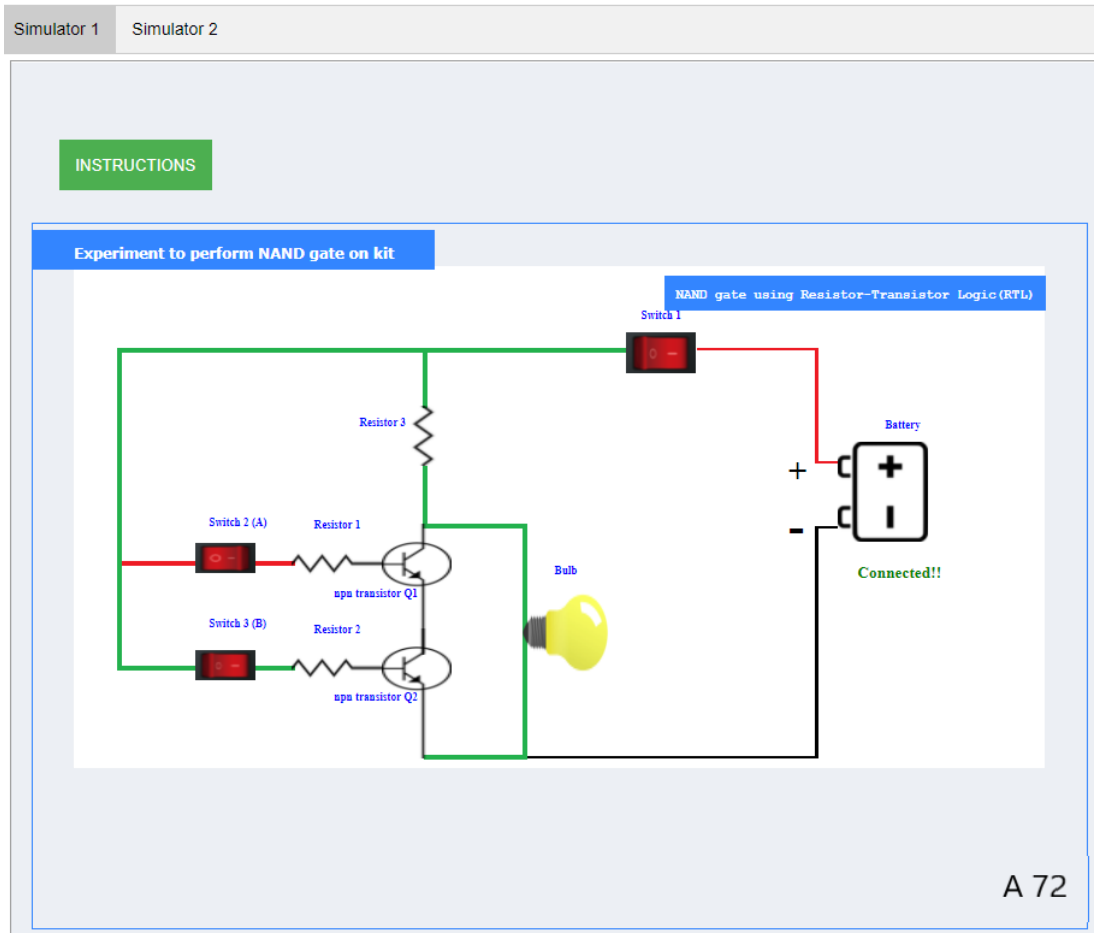

Simulator 1
Simulator 2
INSTRUCTIONS
Verification of truth table for OR gate
1
1
A
B
out
1
Check
TRUTH TABLE
Print
Serial No. | A | B | Output | Remarks
1 | 0 | 0 | 0 | Correct
2 | 0 | 1 | 1 | Correct
3 | 1 | 0 | 1 | Correct
4 | 1 | 1 | 1 | Correct
Reset
A 72

# 3)NOTGate


Simulator 1
Simulator 2
INSTRUCTIONS
Experiment to perform NOT gate on kit
Transistor as a NOT gate
Switch 1
Resistor 1
Switch 2 (A)
Resistor 2
npn transistor
Bulb
Battery
+
-
Connected!!
A 72


Simulator 1
Simulator 2
INSTRUCTIONS
Verification of truth table for NOT gate
1
A
out
0
Check
TRUTH TABLE
Print
Serial No.
A
Output
Remarks
1
0
1
Correct
2
1
0
Correct
Reset
A 72

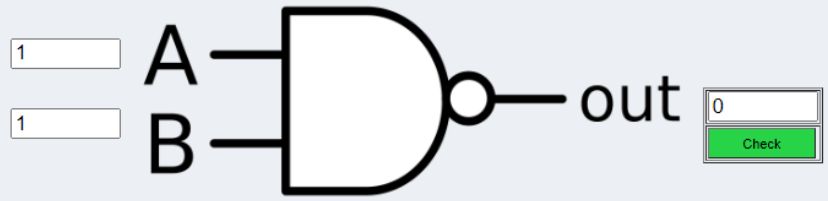
# 4) NAND Gate


Simulator 1
Simulator 2
INSTRUCTIONS
Experiment to perform NAND gate on kit
NAND gate using Resistor-Transistor Logic(RTL)
Switch 1
Resistor 3
Battery
+
-
Switch 2 (A)
Resistor 1
Bulb
npn transistor Q1
Connected!!
Switch 3 (B)
Resistor 2
npn transistor Q2
A 72

Simulator 1 | Simulator 2

Instructions

Verification of truth table for NAND gate




TRUTH TABLE

Print

| Serial No. | A | B | Output | Remarks |
|---|---|---|---|---|
| 1 | 0 | 0 | 1 | Correct |
| 2 | 0 | 1 | 1 | Correct |
| 3 | 1 | 0 | 1 | Correct |
| 4 | 1 | 1 | 0 | Correct |

Reset

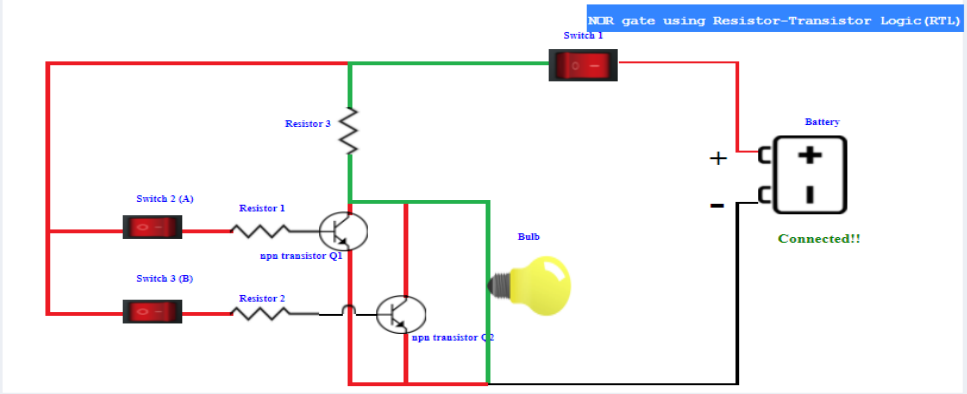
A 72

# 5) NOR Gate

Simulator 1 | Simulator 2

INSTRUCTIONS

Experiment to perform NOR gate on kit




A 72

Simulator 1 | Simulator 2

INSTRUCTIONS

**Verification of truth table for NOR gate**


TRUTH TABLE — Print

| Serial No. | A | B | Output | Remarks |
|---|---|---|---|---|
| 1 | 0 | 0 | 1 | Correct |
| 2 | 0 | 1 | 0 | Correct |
| 3 | 1 | 0 | 0 | Correct |
| 4 | 1 | 1 | 0 | Correct |

Reset

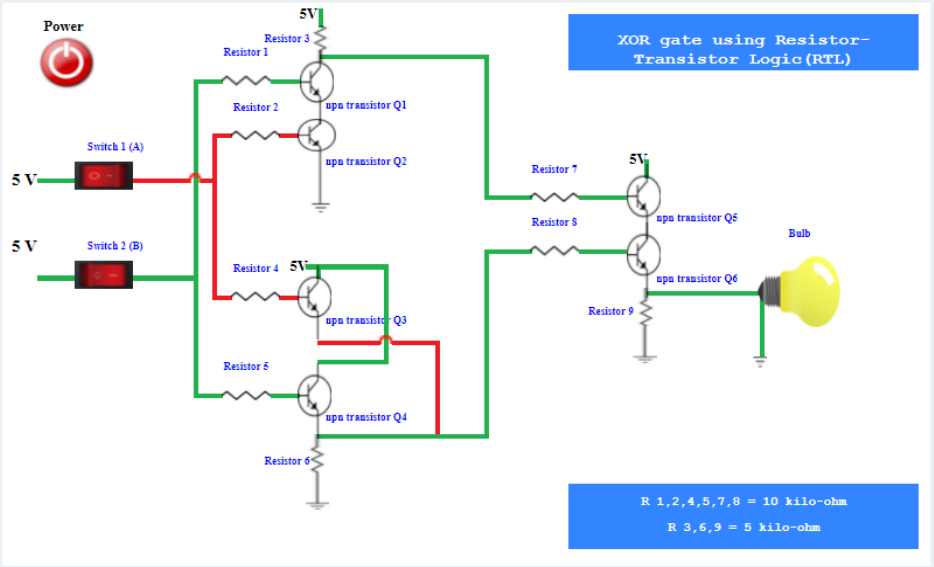
A 72

# 6)EX- OR Gate

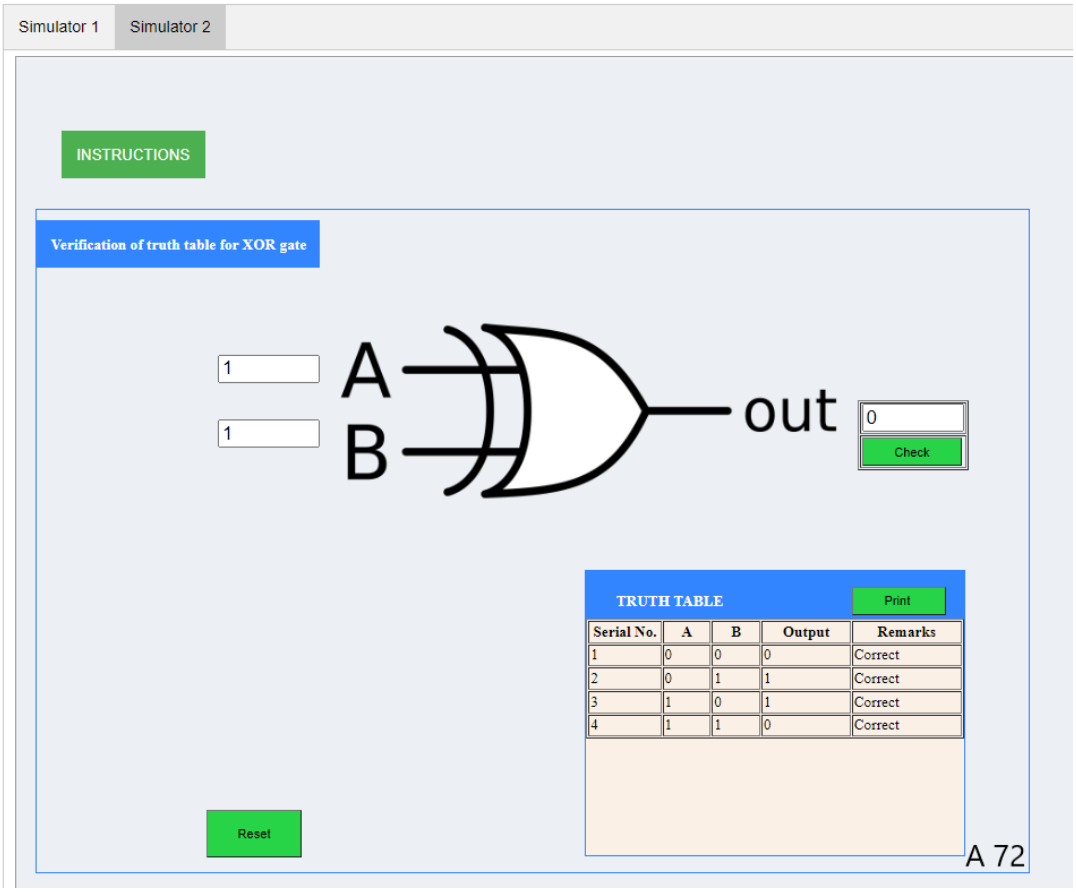
Simulator 1 | Simulator 2

INSTRUCTIONS

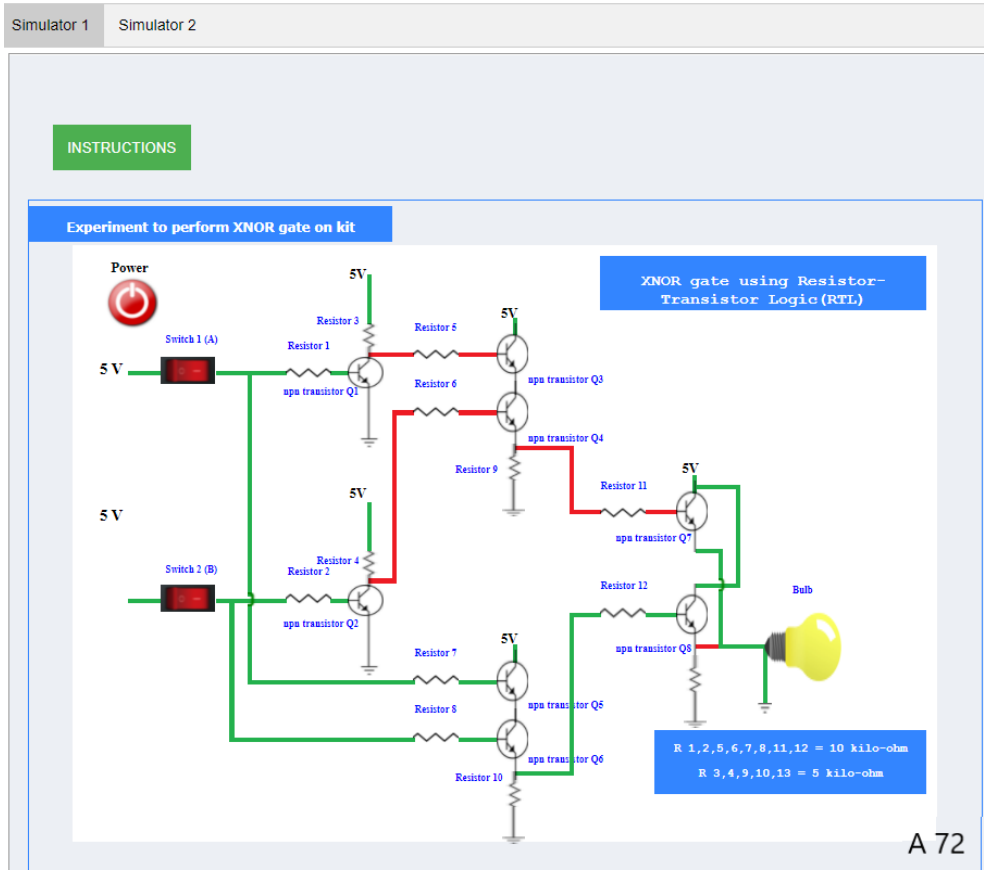
**Experiment to perform XOR gate on kit**




A 72


Simulator 1
Simulator 2
INSTRUCTIONS
Verification of truth table for XOR gate
1
1
A
B
out
0
Check
TRUTH TABLE
Print
Serial No. | A | B | Output | Remarks
1 | 0 | 0 | 0 | Correct
2 | 0 | 1 | 1 | Correct
3 | 1 | 0 | 1 | Correct
4 | 1 | 1 | 0 | Correct
Reset
A 72

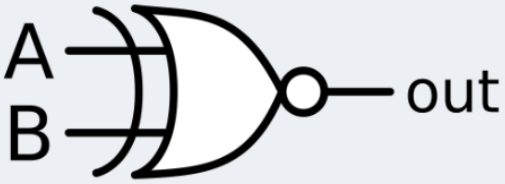
# 7) EX- NOR Gate


Simulator 1
Simulator 2
INSTRUCTIONS
Experiment to perform XNOR gate on kit
XNOR gate using Resistor-Transistor Logic(RTL)
Power
5V
Switch 1 (A)
Switch 2 (B)
Resistor 1
Resistor 2
Resistor 3
Resistor 4
Resistor 5
Resistor 6
Resistor 7
Resistor 8
Resistor 9
Resistor 10
Resistor 11
Resistor 12
npn transistor Q1
npn transistor Q2
npn transistor Q3
npn transistor Q4
npn transistor Q5
npn transistor Q6
npn transistor Q7
npn transistor Q8
Bulb
R 1,2,5,6,7,8,11,12 = 10 kilo-ohm
R 3,4,9,10,13 = 5 kilo-ohm
A 72

Simulator 1 | Simulator 2

INSTRUCTIONS

**Verification of truth table for XNOR gate**




**TRUTH TABLE** Print

| Serial No. | A | B | Output | Remarks |
|---|---|---|---|---|
| 1 | 0 | 0 | 1 | Correct |
| 2 | 0 | 1 | 0 | Correct |
| 3 | 1 | 0 | 0 | Correct |
| 4 | 1 | 1 | 1 | Correct |

Reset

A 72

# Experiment- 2

## AIM :

To implement the logic functions i.e. AND, OR, NOT, Ex-OR, Ex- NOR and a logical expression with the help of NAND and NOR universal gates respectively.

## THEORY :

Logic gates are electronic circuits which perform logical functions on one or more inputs to produce one output. There are seven logic gates. When all the input combinations of a logic gate are written in a series and their corrresponding outputs written along them, then this input/ output combination is called Truth Table.

### 1)Nand gate as Universal gate

NAND gate is actually a combination of two logic gates i.e. AND gate followed by NOT gate. So its output is complement of the output of an AND gate.This gate can have minimum two inputs. By using only NAND gates, we can realize all logic functions: AND, OR, NOT, Ex-OR, Ex-NOR, NOR. So this gate is also called as universal gate.

## 1.1)NAND gates as OR gate

From DeMorgan's theorems:
(A.B)' = A' + B'
(A'.B')' = A'' + B'' = A + B
So, give the inverted inputs to a NAND gate, obtain OR operation at output.

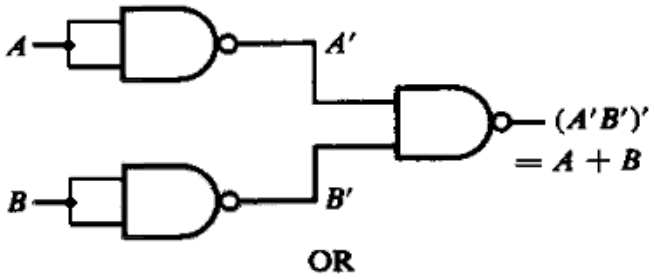



**Figure-1:NAND gates as OR gate**

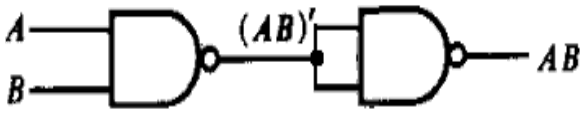
| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**Figure-2:Truth table of OR**

## 1.2)NAND gates as AND gate

A NAND produces complement of AND gate. So, if the output of a NAND gate is inverted, overall output will be that of an AND gate.

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




AND

**Figure-3:NAND gates as AND gate**

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

**Figure-4:Truth table of AND**

## 1.3)NAND gates as Ex-OR gate

The output of a two input Ex-OR gate is shown by: Y = A'B + AB'. This can be achieved with the logic diagram shown in the left side.

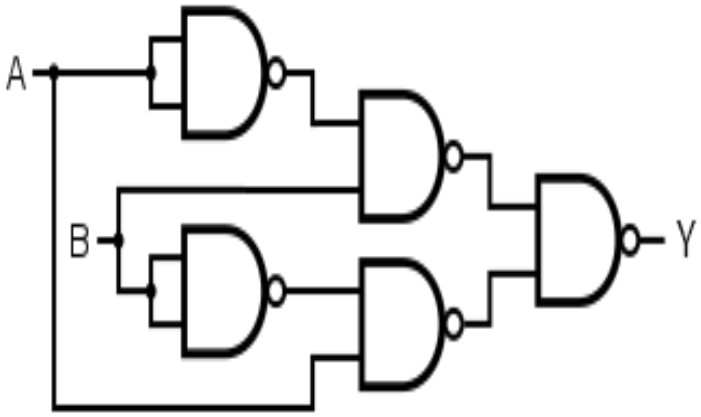



**Figure-5:NAND gate as Ex-OR gate**

| A | B | A **XOR** B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Figure-6:Truth table of Ex-OR**

## 1.4)NAND gates as Ex-NOR gate

Ex-NOR gate is actually Ex-OR gate followed by NOT gate. So give the output of Ex-OR gate to a NOT gate, overall output is that of an Ex-NOR gate.

$$\mathbf{Y = AB + A'B'}$$

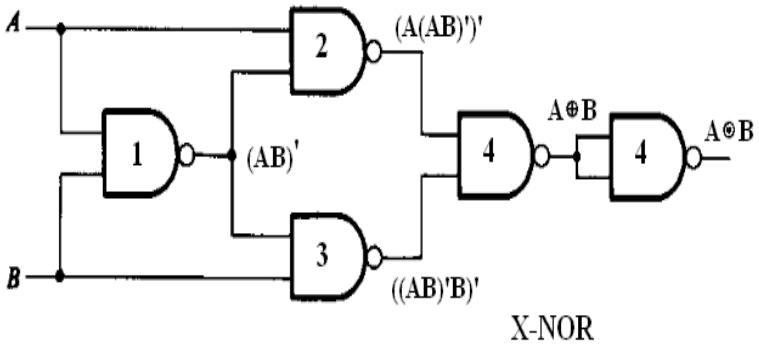



**Figure-7:NAND gates as Ex-NOR gate**

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Figure-8:Truth table of Ex-NOR**

## 1.5) Implementing the simplified function with NAND gates only

We can now start constructing the circuit. First note that the entire expression is inverted and we have three terms ANDed. This means that we must use a 3-input NAND gate. Each of the three terms is, itself, a NAND expression. Finally, negated single terms can be generates with a 2-input NAND gate acting as an inverted. Figure 8 illustrates a circuit using NAND gates only.

**F=((C'.B.A)'(D'.C.A)'(C.B'.A)')'**

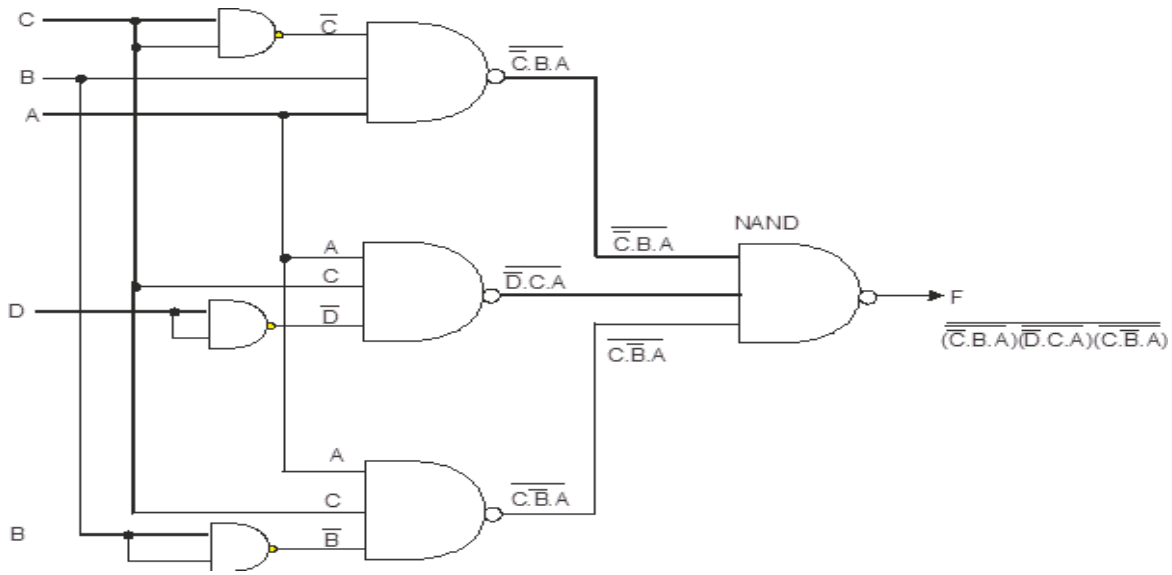



**Figure-9:Implementing the simplified function with NAND gates only**

## 2 )Nor gate as Universal Gate

NOR gate is actually a combination of two logic gates: OR gate followed by NOT gate. So its output is complement of the output of an OR gate.This gate can have minimum two inputs, output is always one. By using only NOR gates, we can realize all logic functions: AND, OR, NOT, Ex-OR, Ex-NOR, NAND. So this gate is also called universal gate.

### 2.1)NOR gates as OR gate

A NOR produces complement of OR gate. So, if the output of a NOR gate is inverted, overall output will be that of an OR gate.

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

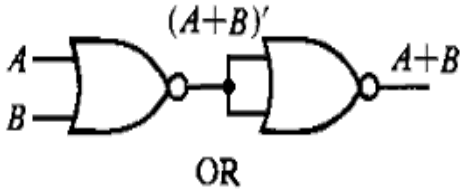



**Figure-10:NOR gates as OR gate**

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**Figure-11:Truth table of OR**

## 2.2)NOR gates as AND gate

From DeMorgan's theorems:
(A+B)' = A'B'
(A'+B')' = A''B'' = AB
So, give the inverted inputs to a NOR gate, obtain AND operation at output.

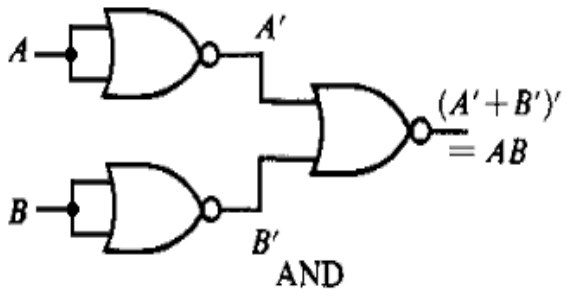



**Figure-12:NOR gates as AND gate**

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

**Figure-13:Truth table of AND**

## 2.4)NOR gates as Ex-OR gate

Ex-OR gate is actually Ex-NOR gate followed by NOT gate. So give the output of Ex-NOR gate to a NOT gate, overall output is that of an Ex-OR gate.
Y = A'B+ AB'

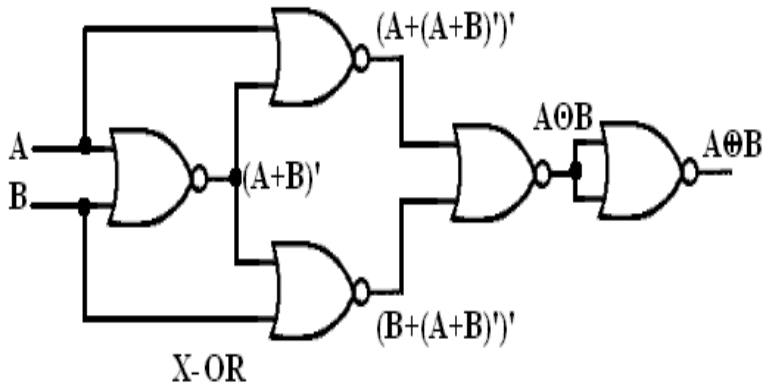



**Figure-14:NOR gates as Ex-OR gate**

| A | B | A **XOR** B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Figure-15:Truth table of Ex-OR**

## 2.3)NOR gates as Ex-NOR gate

The output of a two input Ex-NOR gate is shown by: Y = AB + A'B'. This can be achieved with the logic diagram shown in the left side.

| Gate No. | Inputs | Output |
|---|---|---|
| 1 | A, B | (A + B)' |
| 2 | A, (A + B)' | (A + (A+B)')' |
| 3 | (A + B)', B | (B + (A+B)')' |
| 4 | (A + (A + B)')', (B + (A+B)')' | AB + A'B' |

Now the ouput from gate no. 4is the overall output of the configuration.

$$
\begin{aligned}
Y &= ((A + (A+B)')' (B +( A+B)')')' \\
&= (A+(A+B)')''.(B+(A+B)')'' \\
&= (A+(A+B)').(B+(A+B)') \\
&= (A+A'B').(B+A'B') \\
&= (A + A').(A + B').(B+A')(B+B') \\
&= 1.(A+B').(B+A').1 \\
&= (A+B').(B+A') \\
&= A.(B + A') +B'.(B+A') \\
&= AB + AA' +B'B+B'A' \\
&= AB + 0 + 0 + B'A' \\
&= AB + B'A' \\
\Rightarrow \quad Y &= AB + A'B'
\end{aligned}
$$

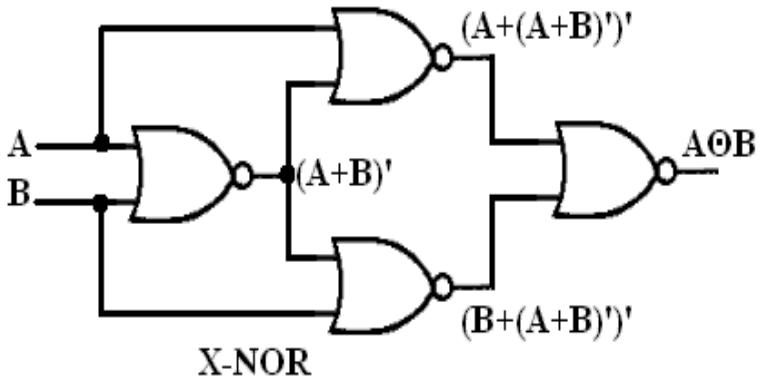



**Figure-16:NOR gates as Ex-NOR gate**

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Figure-17:Truth table of Ex-NOR**

## 2.5)Constructing a circuit with NOR gates only

Designing a circuit with NOR gates only uses the same basic techniques as designing a circuit with NAND gates; that is, the application of deMorgan's theorem. The only difference between NOR gate design and NAND gate design is that the former must eliminate product terms and the later must eliminate sum terms.

**F=(((C.B'.A)+(D.C'.A)+(C.B'.A))')'**

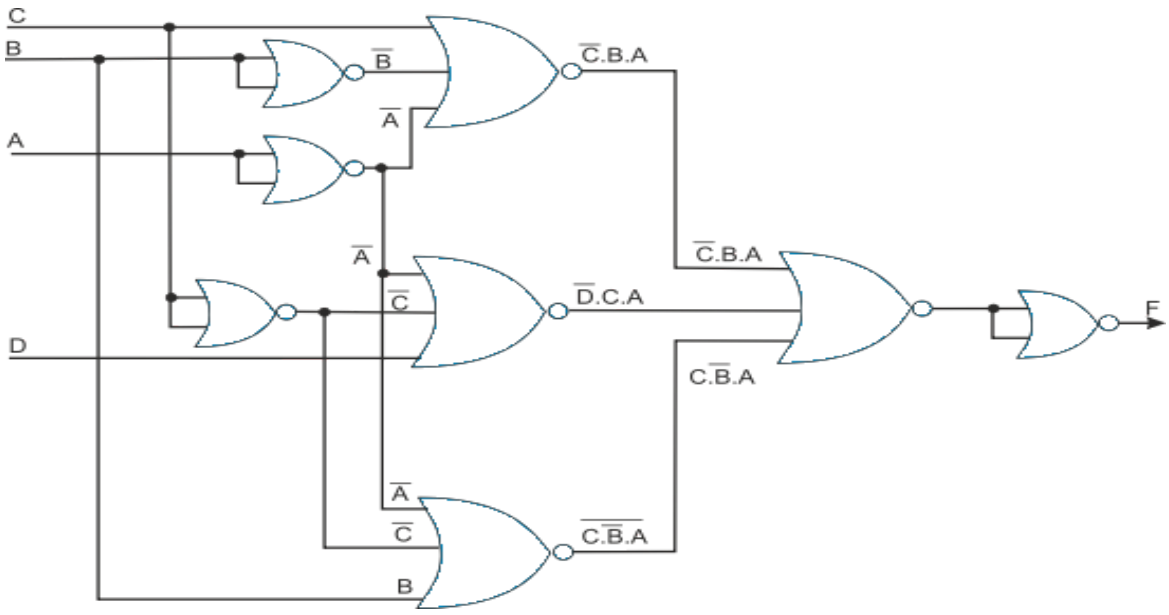



**Figure-18:Implementing the simplified function with NOR gates only**

## PROCEDURE :

Step-1) Select and drag "  " for generate wire of the circuit.

Step-2) Join the wire to perform the require logic.

Follow these steps for other part of experiments

## TEST :

1) The output of an AND gate with three inputs, A, B, and C, is HIGH when **A=1,B=1,C=1**

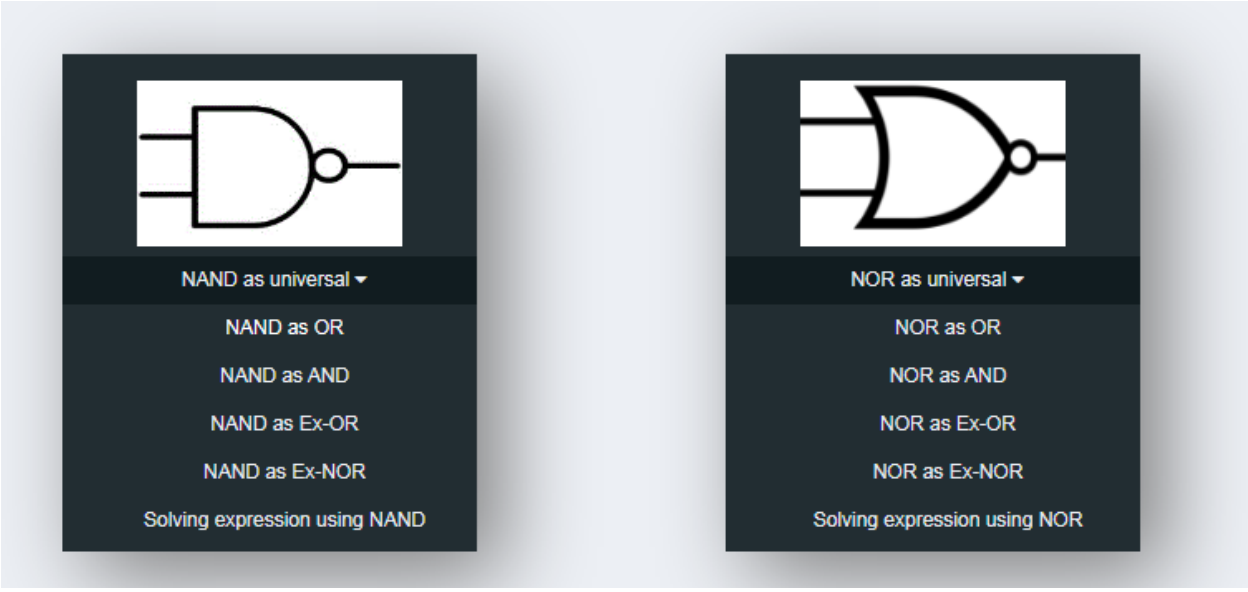
2) If a 3-input NOR gate has eight input possibilities, how many of those possibilities will result in a HIGH output **1 .**

3) A = 1, B = 1, C = 0, and D = 0 accurately describe a four-input OR gate **1+1+0+0 = 1**

4) A single transistor can be used to build **NOT gates**.

5) The output of an AND gate with three inputs, A, B, and C, is HIGH when **A=1,B=1,C=1.**

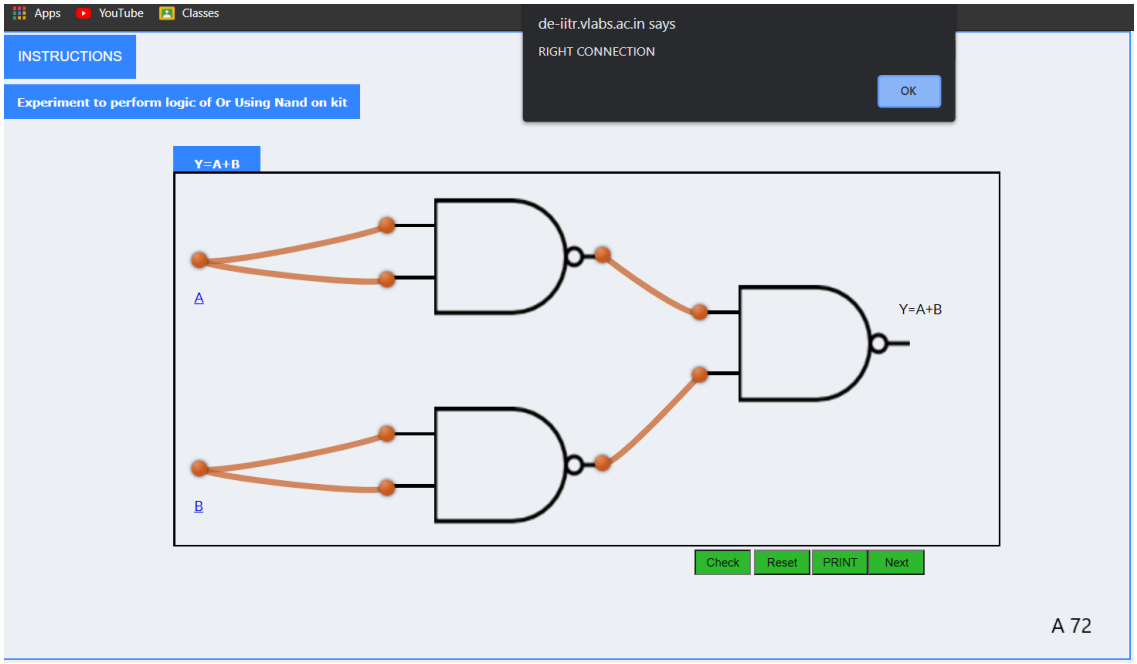
# SIMULATION RESULTS :



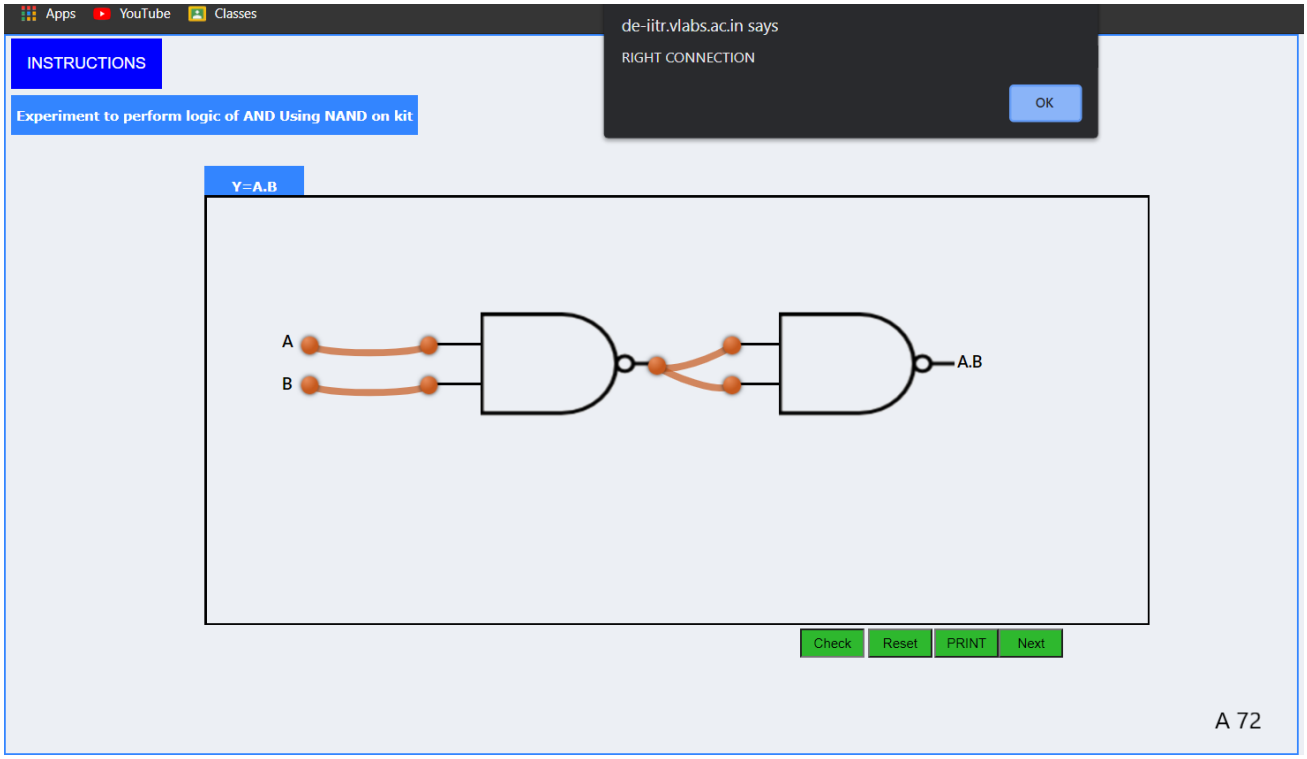

( Total 4 Simulation for NAND & 4 Simulation for NOR Gate )

**NAND as universal**

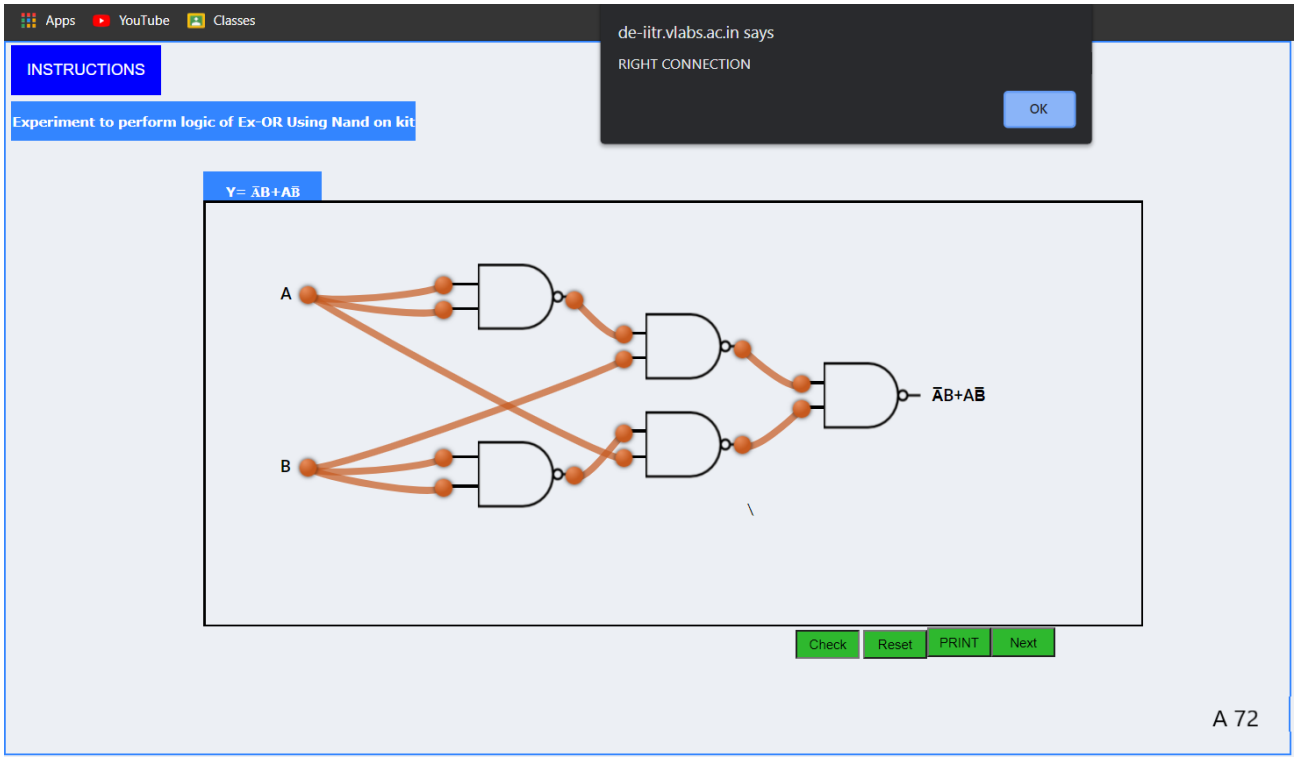
1) NAND as OR



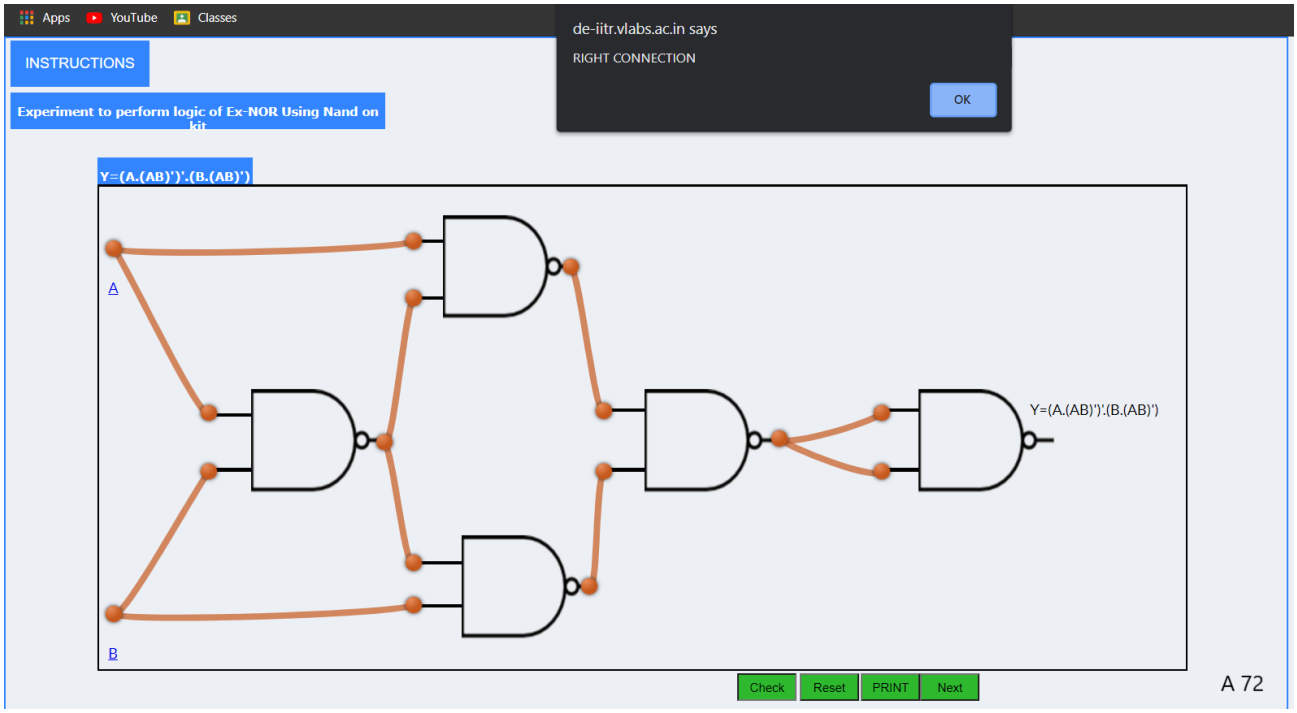
## 2) NAND as AND



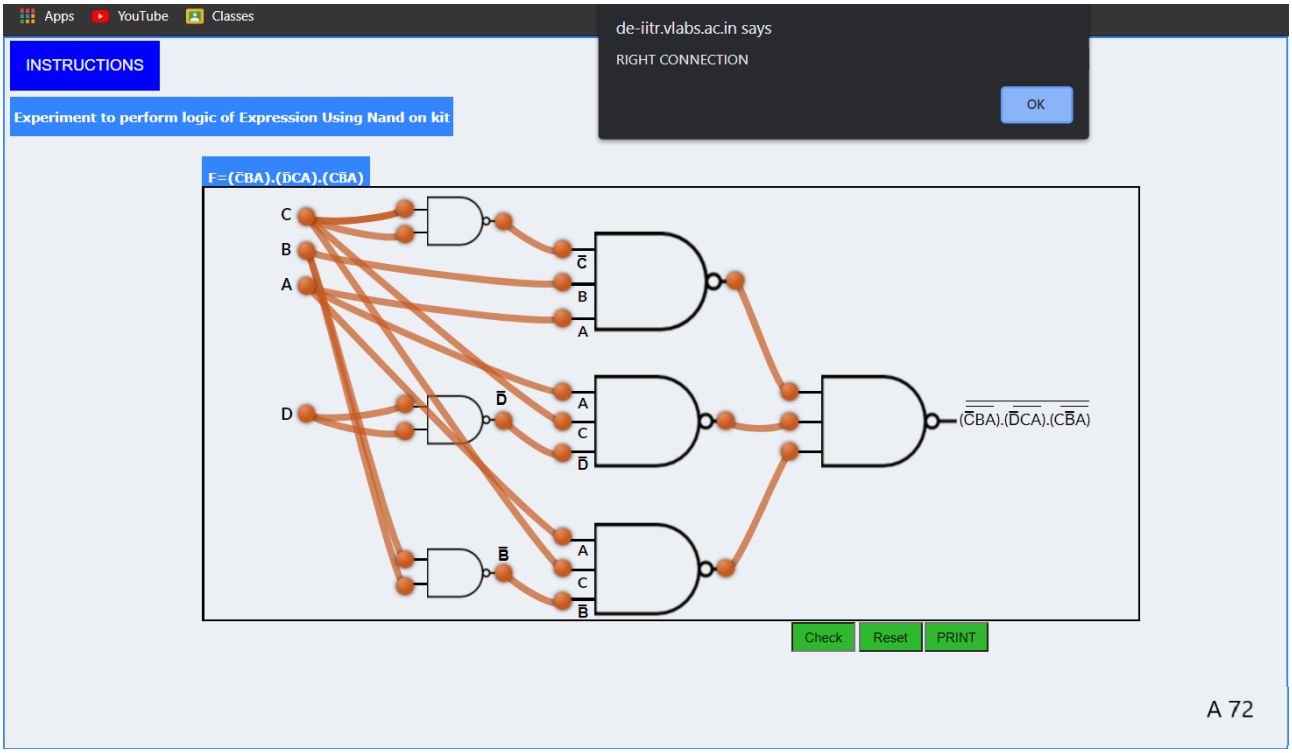

## 3) NAND AS EX- OR



## 4) NAND as Ex- NOR

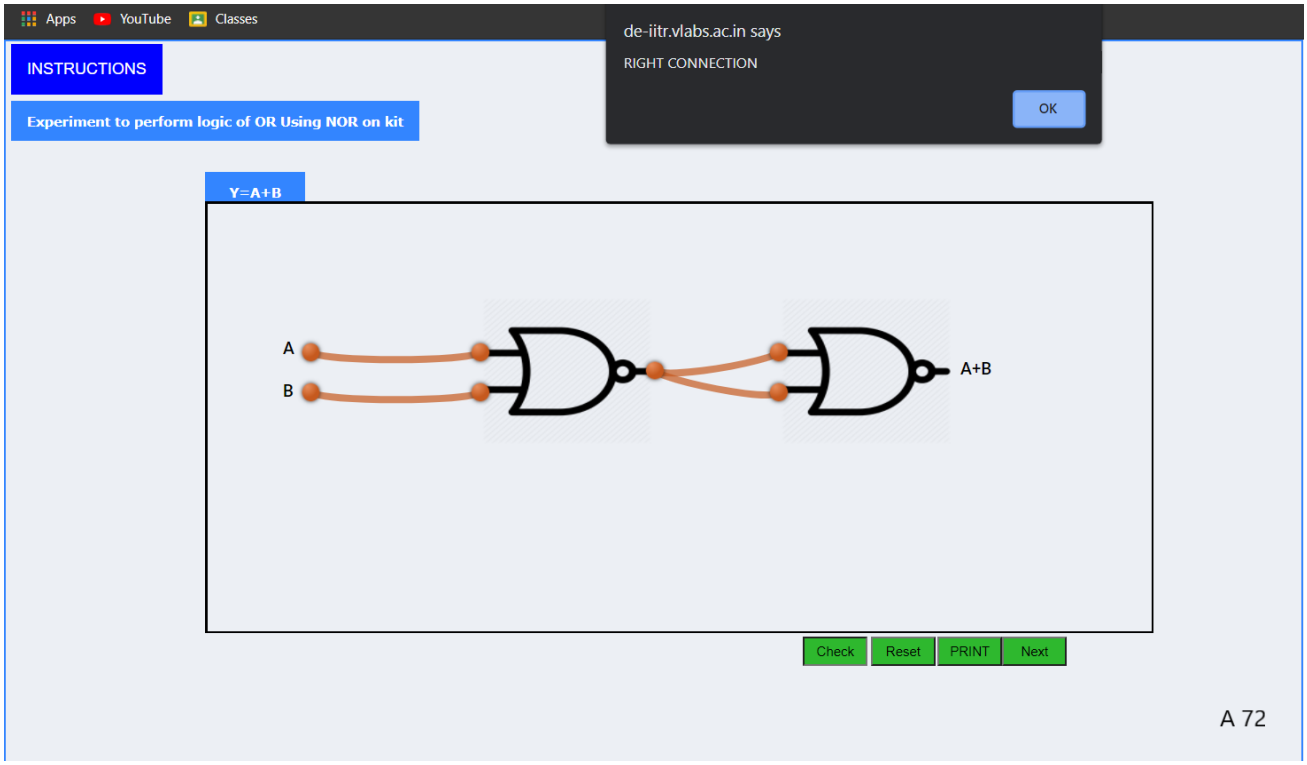



## 5) Logic Of Expression using NAND on kit

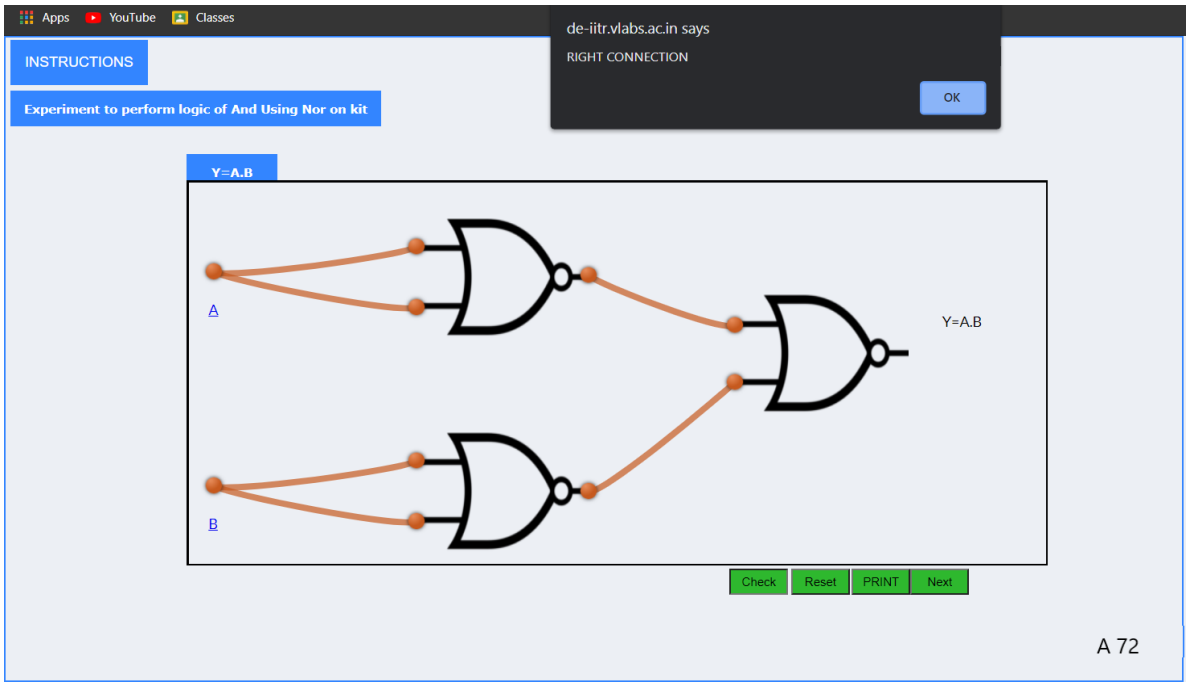


# NOR as universal

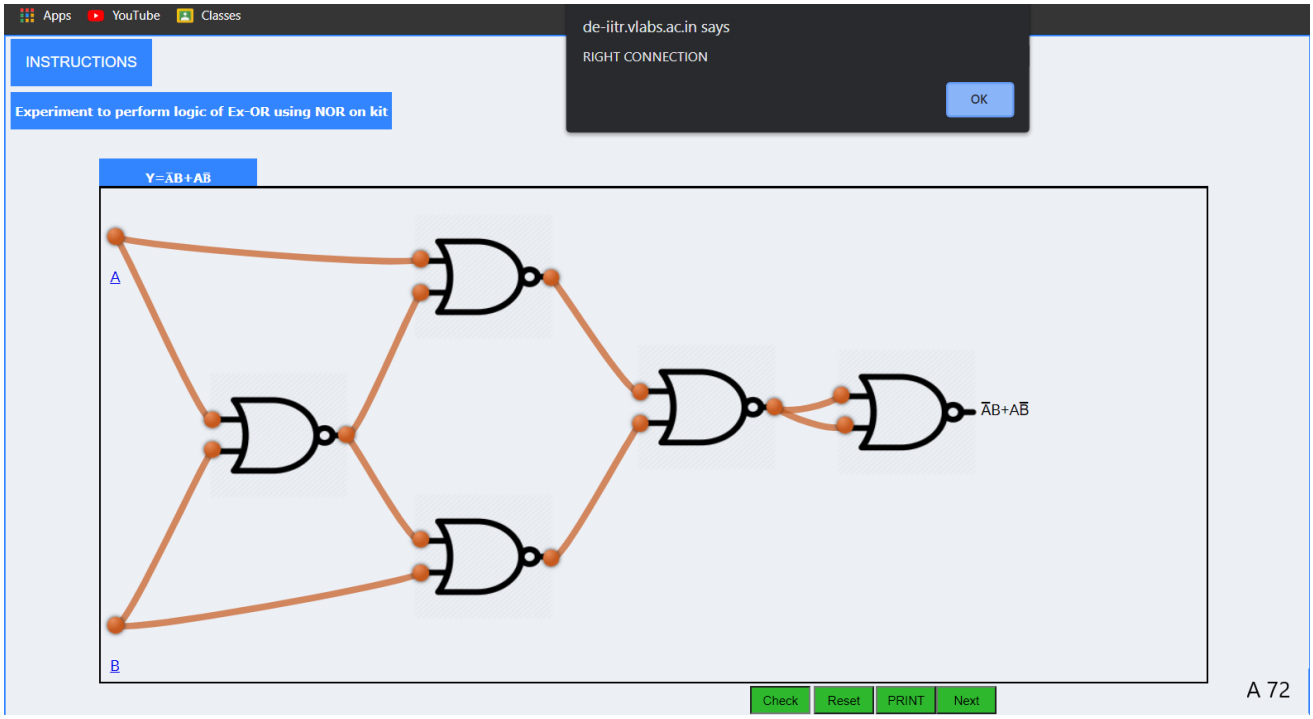
## 1) NOR as OR Gate



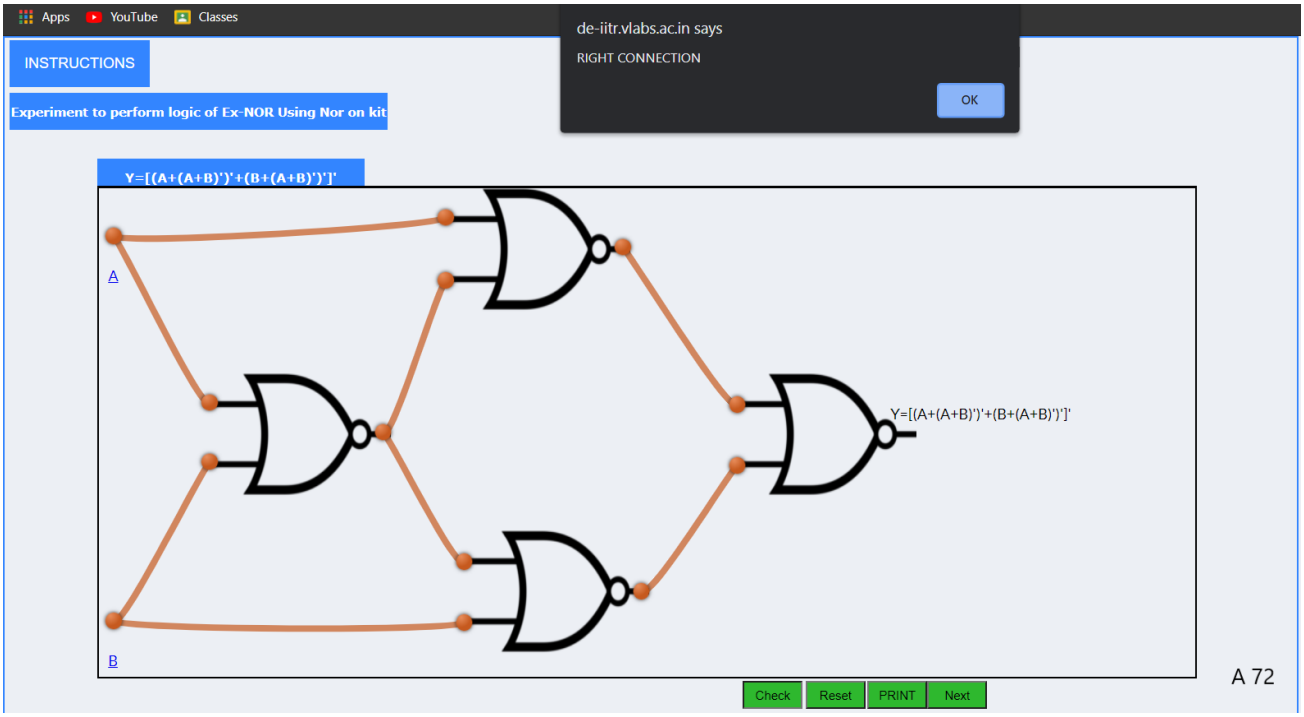

## 2) NOR as AND Gate



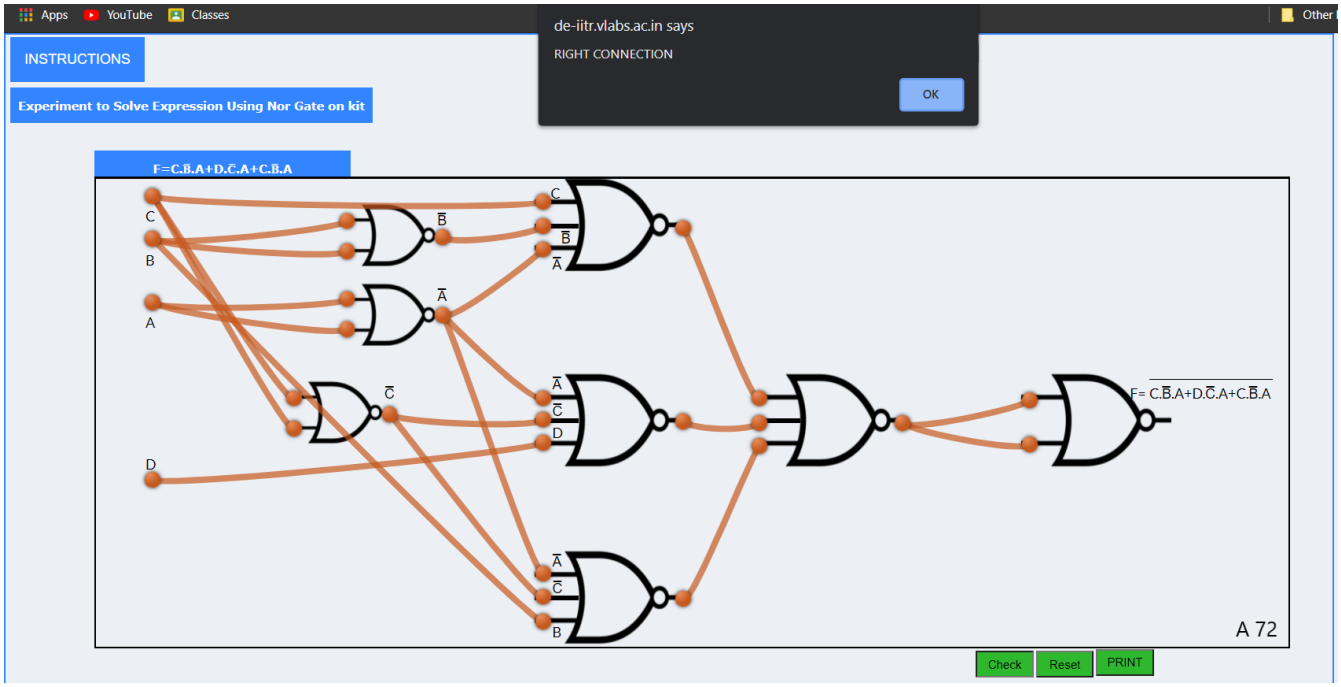
## 3) NOR as EX- OR Gate




## 4) NOR as EX- NOR Gate



5) Expression using NOR Gate on Kit



# Experiment- 3

## AIM :

To verify the truth table of half adder and full adder by using XOR and NAND gates respectively and analyse the working of half adder and full adder circuit with the help of LEDs in simulator 1 and verify the truth table only of half adder and full adder in simulator 2**.**

## THEORY :

Adders are digital circuits that carry out addition of numbers. Adders are a key component of arithmetic logic unit. Adders can be constructed for most of the numerical representations like Binary Coded Decimal (BCD), Excess – 3, Gray code, Binary etc. out of these, binary addition is the most frequently performed task by most common adders. Apart from addition, adders are also used in certain digital applications like table index calculation, address decoding etc.

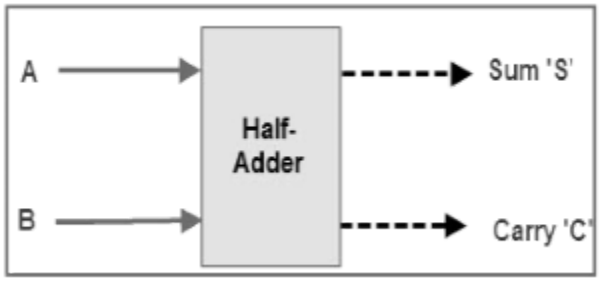
Binary addition is similar to that of decimal addition. Some basic binary additions are shown below.

| | | | |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| +0 | +1 | +0 | +1 |
| 0 | 1 | 1 | (carry) 1 0 |

**Figure 1. Schematic representation of half adder**

1) Half Adder

Half adder is a combinational circuit that performs simple addition of two binary numbers. If we assume A and B as the two bits whose addition is to be performed,the block diagram and a truth table for half adder with A, B as inputs and Sum, Carry as outputs can be tabulated as follows.

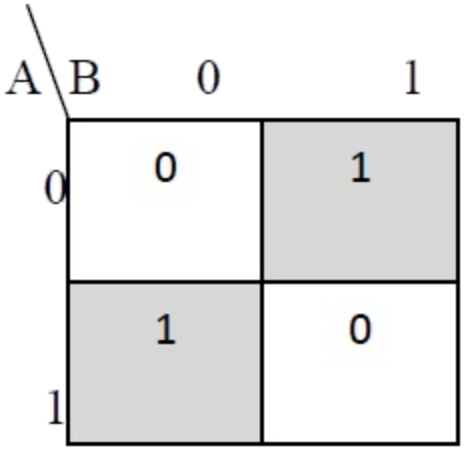


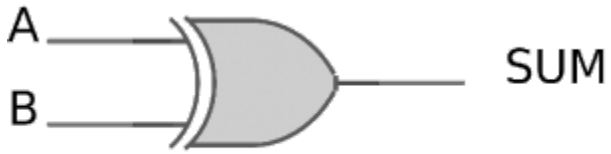

| Truth Table | | | |
|---|---|---|---|
| Input | | Output | |
| A | B | Sum | Carry |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

**Figure 2. Block diagram and truth table of half adder**

The sum output of the binary addition carried out above is similar to that of an Ex-OR operation while the carry output is similar to that of an AND operation. The same can be verified with help of Karnaugh Map.
The truth table and K Map simplification and logic diagram for sum output is shown below.

| Truth Table | | |
|---|---|---|
| A | B | SUM |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



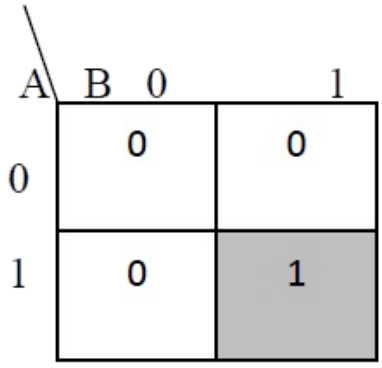

2 Input EX-OR

**Figure 3. Truth table, K Map simplification and Logic diagram for sum output of half adder**

**Sum = A B' + A' B**

The truth table and K Map simplification and logic diagram for carry is shown below.

| Truth Table | | |
|---|---|---|
| A | B | Carry |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



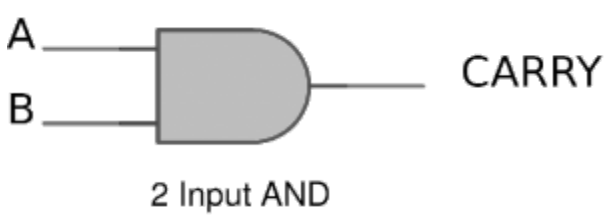



**Figure 4. Truth table, K Map simplification and Logic diagram for sum output of half adder**

**Carry = AB**

If A and B are binary inputs to the half adder, then the logic function to calculate sum S is Ex – OR of A and B and logic function to calculate carry C is AND of A and B. Combining these two, the logical circuit to implement the combinational circuit of half adder is shown below.

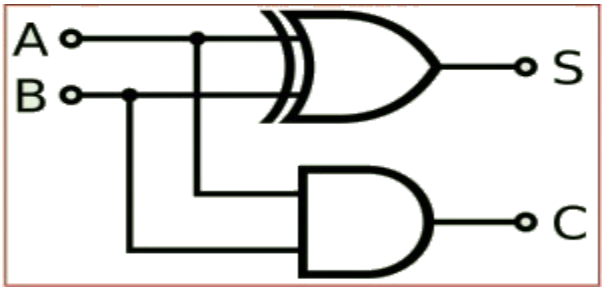



**Figure 5. Half Adder Logic Diagram**

As we know that NAND and NOR are called universal gates as any logic system can be implemented using these two, the half adder circuit can also be implemented using them. We know that a half adder circuit has one Ex – OR gate and one AND gate.

1)Half Adder using NAND gates

Five NAND gates are required in order to design a half adder. The circuit to realize half adder using NAND gates is shown below.

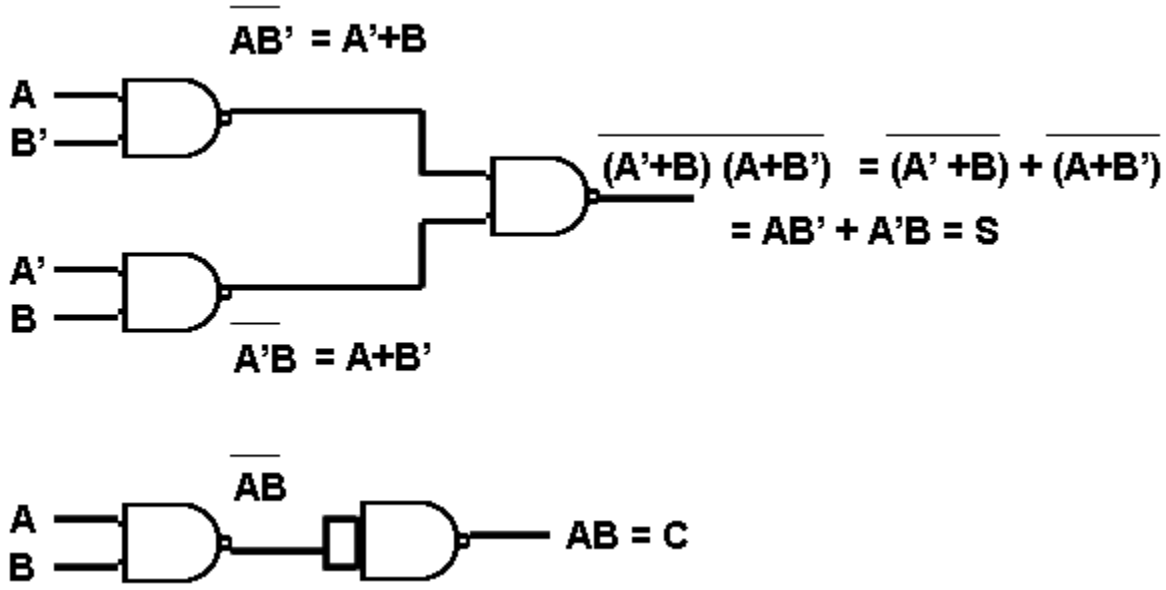



**Figure 6. Realization of half adder using NAND gates**

1.2)Half Adder using NOR gates

Five NOR gates are required in order to design a half adder. The circuit to realize half adder using NOR gates is shown below.

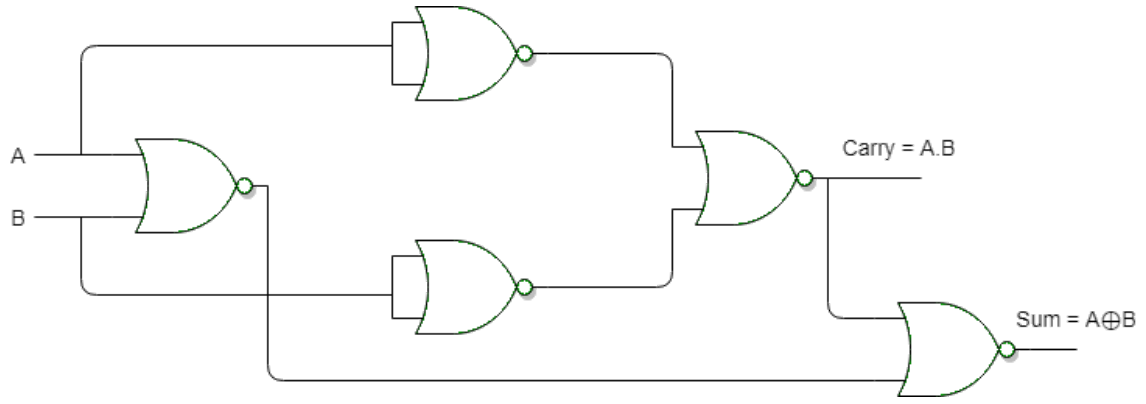
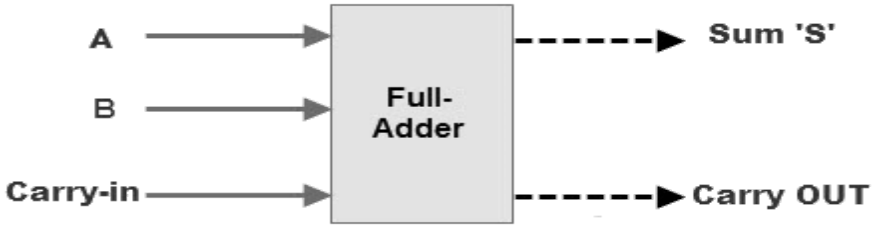



**Figure 7. Realization of half adder using NOR Gates**

## 2) Full Adder

Full adder is a digital circuit used to calculate the sum of three binary bits. Full adders are complex and difficult to implement when compared to half adders. Two of the three bits are same as before which are A, the augend bit and B, the addend bit. The additional third bit is carry bit from the previous stage and is called 'Carry' – in generally represented by CIN. It calculates the sum of three bits along with the carry. The output carry is called Carry – out and is represented by Carry OUT.
The block diagram of a full adder with A, B and CIN as inputs and S, Carry OUT as outputs is shown below.




| Input | | | Output | |
|---|---|---|---|---|
| A | B | Cin | Sum | Carry |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**Figure 8. Full Adder Block Diagram and Truth Table**

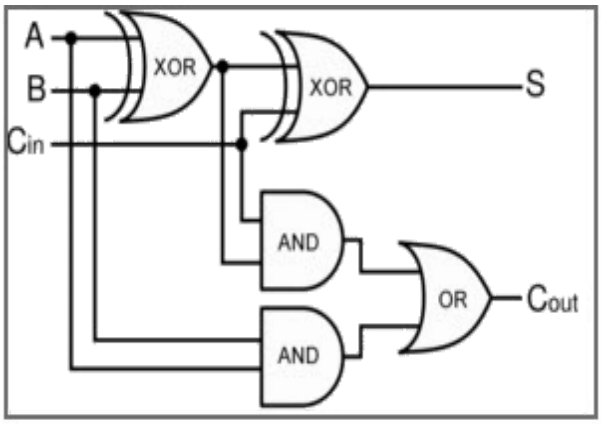



**Figure 9. Full Adder Logic Diagram**

Based on the truth table, the Boolean functions for Sum (S) and Carry – out (COUT) can be derived using K – Map.

| A \ $BC_{IN}$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

**Figure 10. The K-Map simplified equation for sum is S = A'B'Cin + A'BCin' + ABCin**

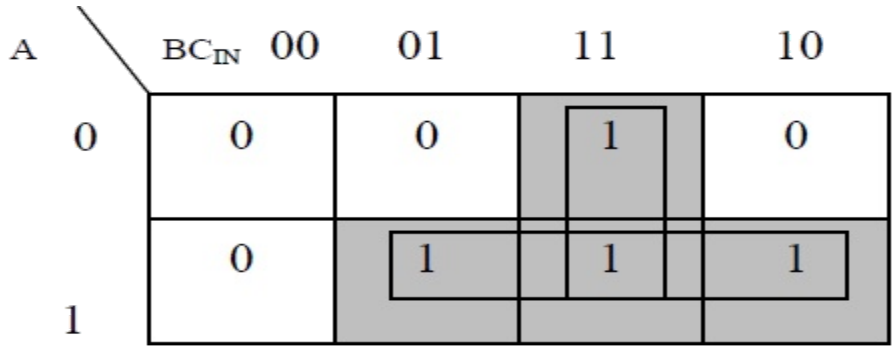



**Figure 11. The K-Map simplified equation for COUT is COUT = AB + ACIN + BCIN**

In order to implement a combinational circuit for full adder, it is clear from the equations derived above, that we need four 3-input AND gates and one 4-input OR gates for Sum and three 2-input AND gates and one 3-input OR gate for Carry – out.

2.1)Full Adder using NAND gates

As mentioned earlier, a NAND gate is one of the universal gates and can be used to implement any logic design. The circuit of full adder using only NAND gates is shown below.

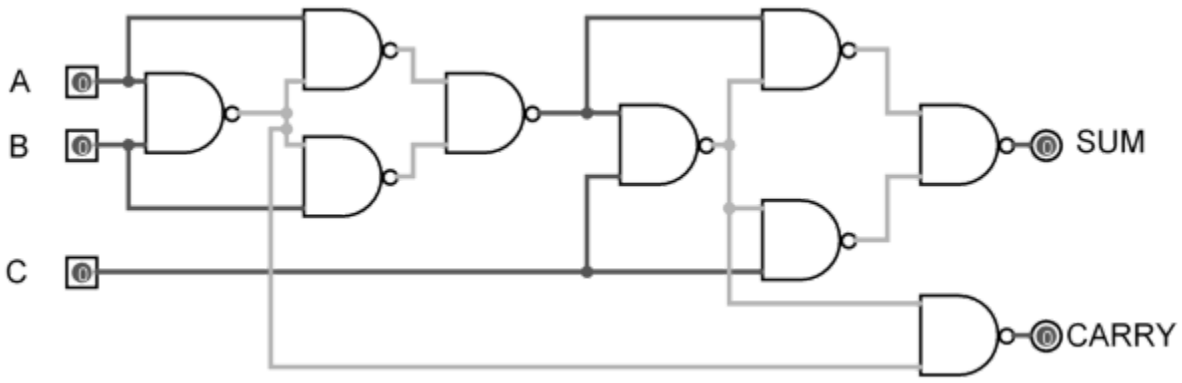



**Figure 12. Full Adder using NAND gates**

## 2.2)Full Adder using NOR gates

As mentioned earlier, a NOR gate is one of the universal gates and can be used to implement any logic design. The circuit of full adder using only NOR gates is shown below.

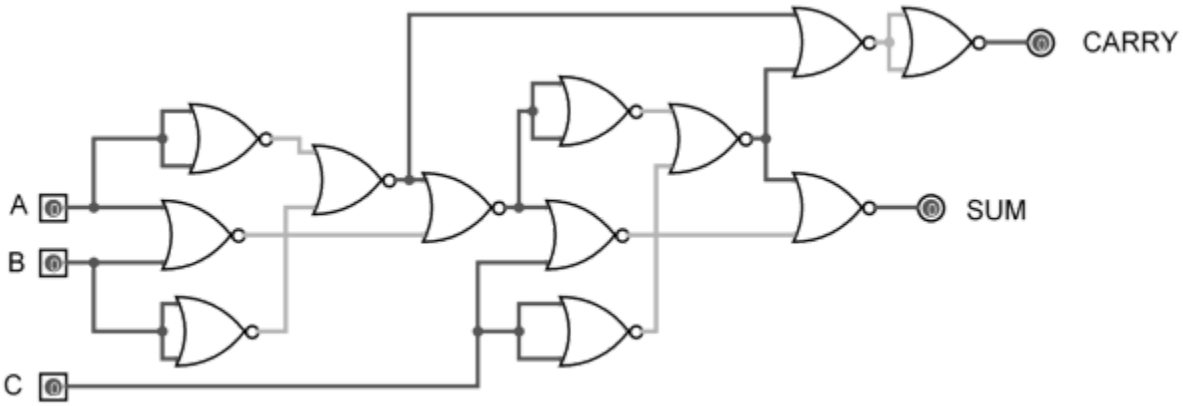



**Figure 13. Full Adder using NOR gates**

## PROCEDURE :

### 1) HALF ADDER

Simulator 1:

Step-1) Connect the supply(+5V)  to the circuit.
Step-2) First press "ADD" button to add basic state of your output in the given table.
Step-3) Press the switches to select the required inputs "A" and "B".
Step-4) Press "ADD" button to add your inputs and outputs in the given table.
Step-5) Repeat steps 3 & 4 for next state of inputs and their corresponding outputs.
Step-6) Press the "PRINT" button after completing your simulation to get your results.
Step-7) Press the "RESET" button whenever you want to refresh your simulator.

Simulator 2:

Step-1) Enter the Boolean input "A" and "B".
Step-2) Enter the Boolean output for your corresponding inputs.
Step-3) Click on "Circuit" button to check the circuit diagram for half adder.
Step-4) Click on "Check" Button to verify your output.
Step-5) Click "Print" if you want to get print out of Truth Table.
Step-6) Click on "Reset" button if you want to reset input and outputs.

## 2)FULL ADDER

Simulator 1:

Step-1) Connect the supply(+5V)  to the circuit.
Step-2) First press "ADD" button to add basic state of your output in the given table.
Step-3) Press the switches to select the required inputs "A" and "B" and "$C_{in}$".
Step-4) Press "ADD" button to add your inputs and outputs in the given table.
Step-5) Repeat steps 3 & 4 for next state of inputs and their corresponding outputs.
Step-6) Press the "PRINT" button after completing your simulation to get your results.
Step-7) Press the "RESET" button whenever you want to refresh your simulator.

Simulator 2:

Step-1) Enter the Boolean input "A" and "B" and "Cin".
Step-2) Enter the Boolean output for your corresponding inputs.
Step-3) Click on "Circuit" button to check the circuit diagram for full adder.
Step-4) Click on "Check" Button to verify your output.
Step-5) Click "Print" if you want to get print out of Truth Table.
Step-6) Click on "Reset" button if you want to reset input and outputs.

## TEST :

1) Half-adders have a major limitation in that they cannot

➔ **Accept a carry bit from a previous stage**

2) If A, B and C are the inputs of a full adder then the sum is given by

➔ **A Ex-OR B XOR C**

3) If A, B and C are the inputs of a full adder then the carry is given by

➔ **A AND B OR (A OR B) AND C**

4) Total number of inputs in a half adder is

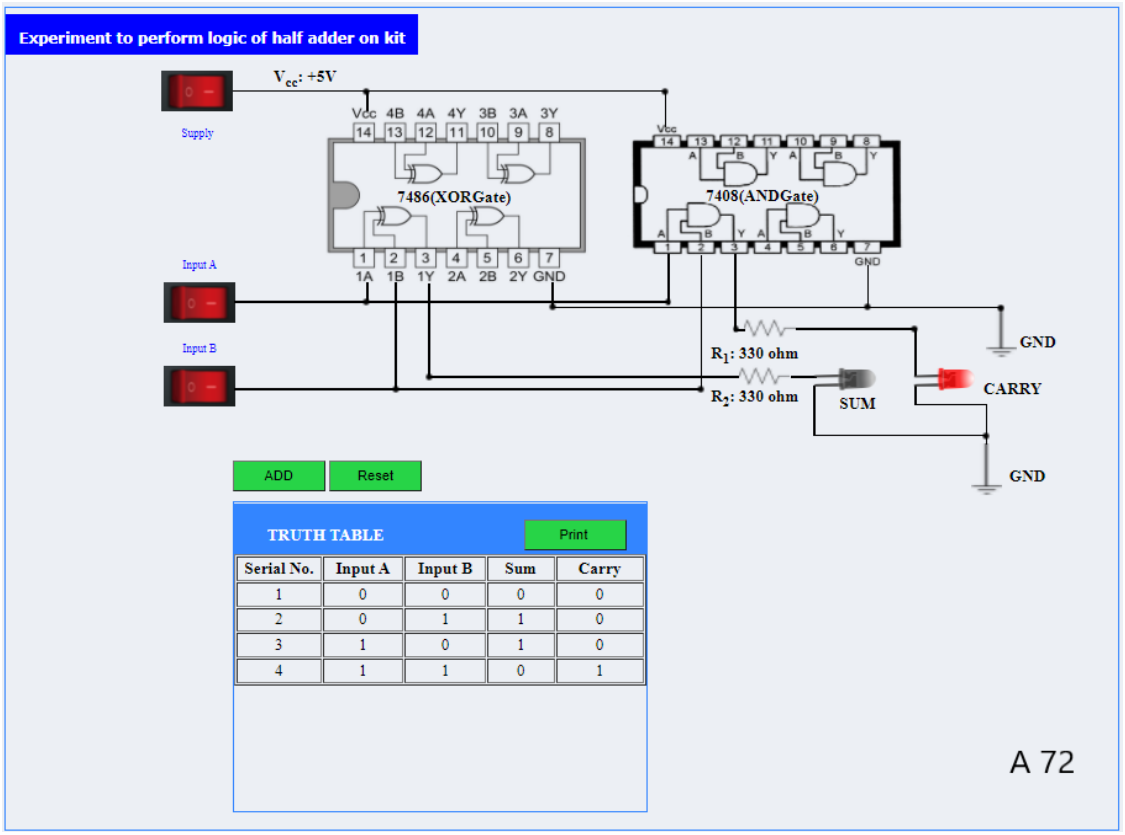
➔ **Two inputs**

5) If A and B are the inputs of a half adder, the sum is given by

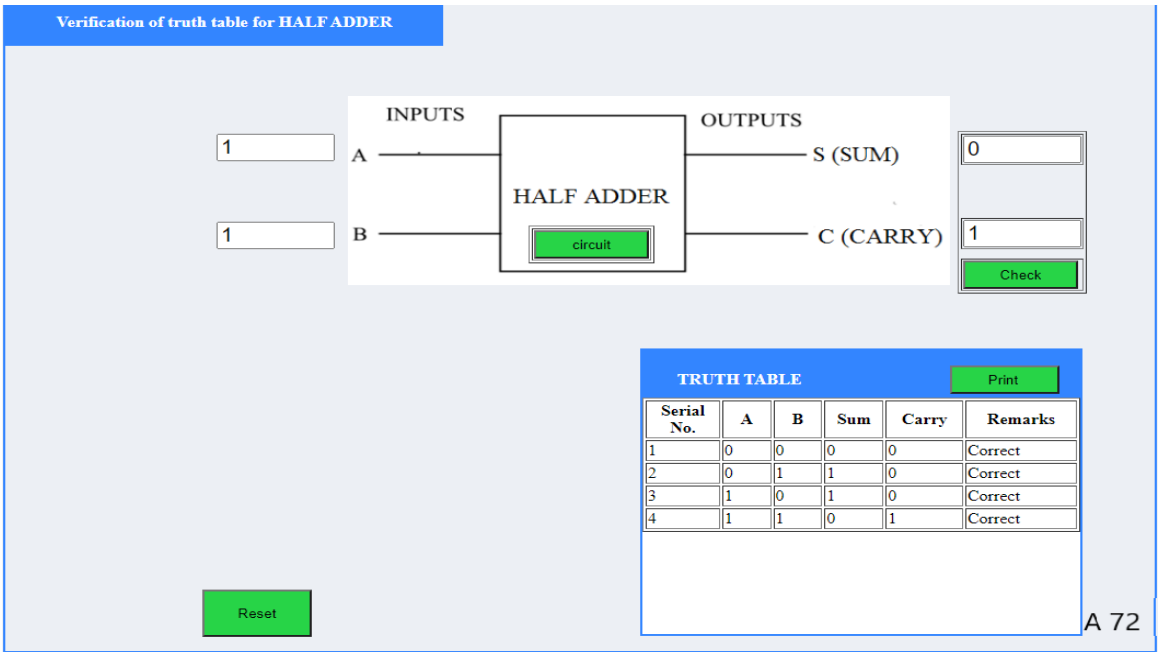
➔ **A XOR B**

# SIMULATION RESULTS :

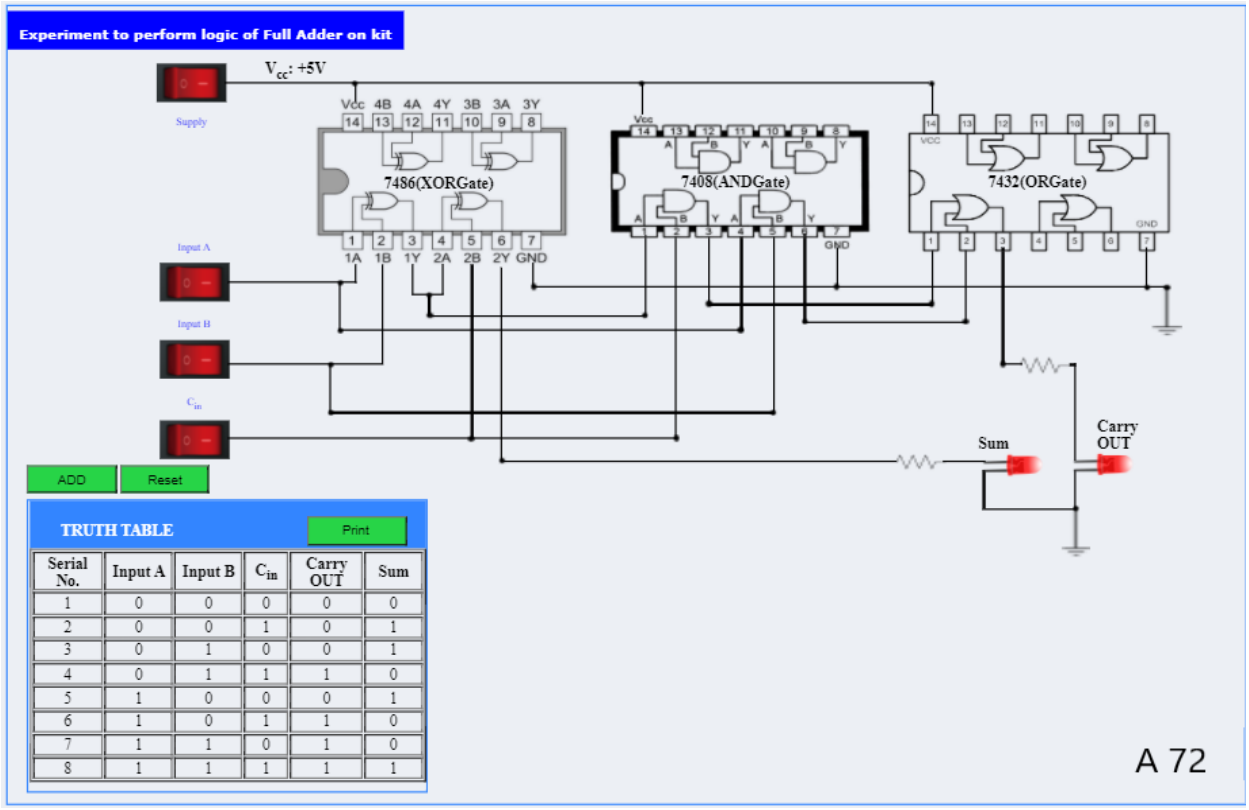
## Half Adder :




## Simulation : 2

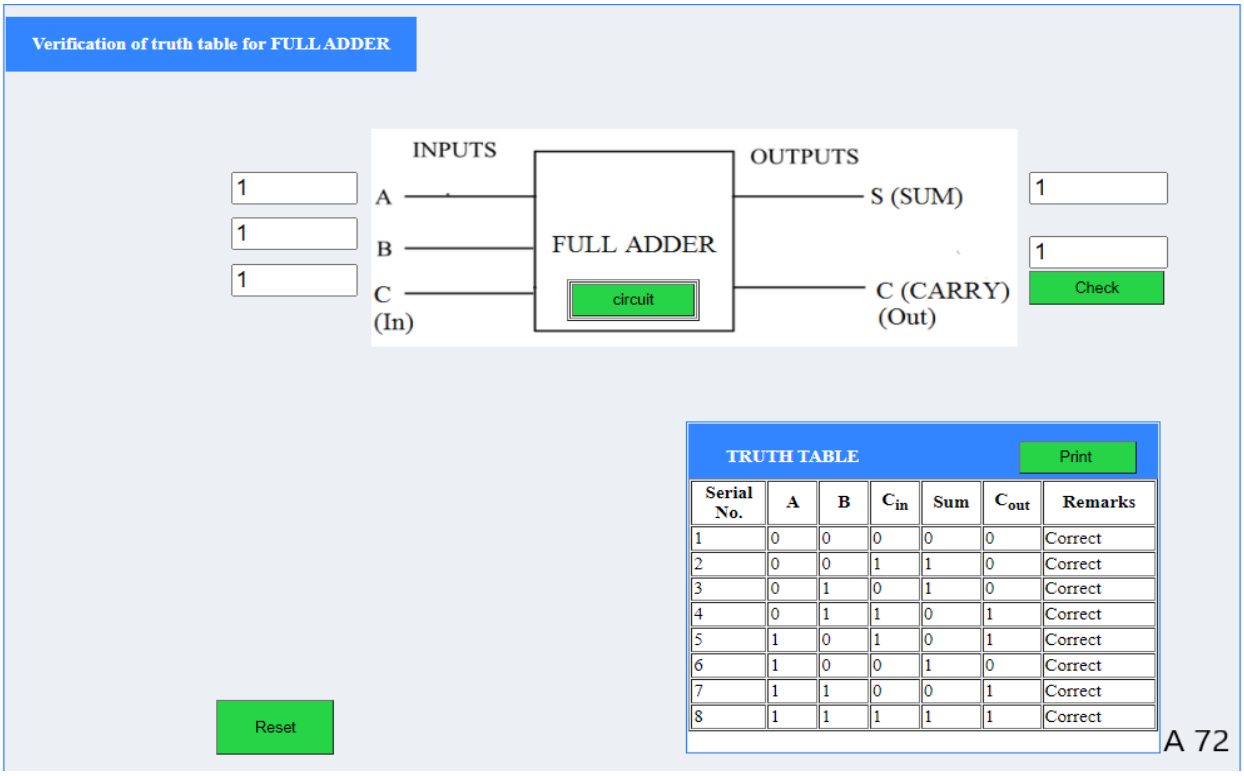


# Full Adder :




| Serial No. | Input A | Input B | $C_{in}$ | Carry OUT | Sum |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | 1 |
| 3 | 0 | 1 | 0 | 0 | 1 |
| 4 | 0 | 1 | 1 | 1 | 0 |
| 5 | 1 | 0 | 0 | 0 | 1 |
| 6 | 1 | 0 | 1 | 1 | 0 |
| 7 | 1 | 1 | 0 | 1 | 0 |
| 8 | 1 | 1 | 1 | 1 | 1 |

# Simulator - 2




| Serial No. | A | B | $C_{in}$ | Sum | $C_{out}$ | Remarks |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | Correct |
| 2 | 0 | 0 | 1 | 1 | 0 | Correct |
| 3 | 0 | 1 | 0 | 1 | 0 | Correct |
| 4 | 0 | 1 | 1 | 0 | 1 | Correct |
| 5 | 1 | 0 | 1 | 0 | 1 | Correct |
| 6 | 1 | 0 | 0 | 1 | 0 | Correct |
| 7 | 1 | 1 | 0 | 0 | 1 | Correct |
| 8 | 1 | 1 | 1 | 1 | 1 | Correct |

# Experiment- 4

## AIM :

To verify the truth table of half subtractor by using the ICs of XOR, NOT and AND gates and of full subtractor by using the ICs of XOR, AND, NOT and OR gates respectively and analyse the working of half subtractor and full subtractor circuit with the help of LEDs in simulator 1 and verify the truth table only of half subtractor and full subtractor in simulator 2

## THEORY :

Subtractor circuits take two binary numbers as input and subtract one binary number input from the other binary number input. Similar to adders, it gives out two outputs, difference and borrow (carry-in the case of Adder). There are two types of subtractors.

1. Half Subtractor
2. Full Subtractor

## 1) Half Subtractor

The half-subtractor is a combinational circuit which is used to perform subtraction of two bits. It has two inputs, A (minuend) and B (subtrahend) and two outputs Difference and Borrow. The logic symbol and truth table are shown below.

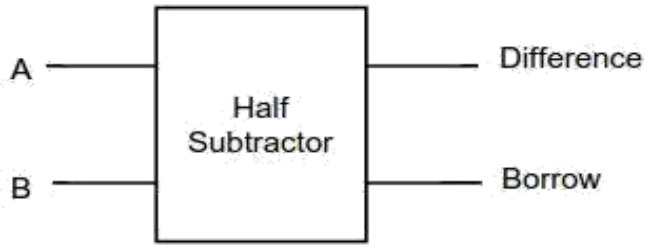



**Figure-1:Logic Symbol of Half subtractor**

| Inputs | | Outputs | |
|---|---|---|---|
| A | B | Difference | Borrow |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

**Figure-2:Truth Table of Half subtractor**

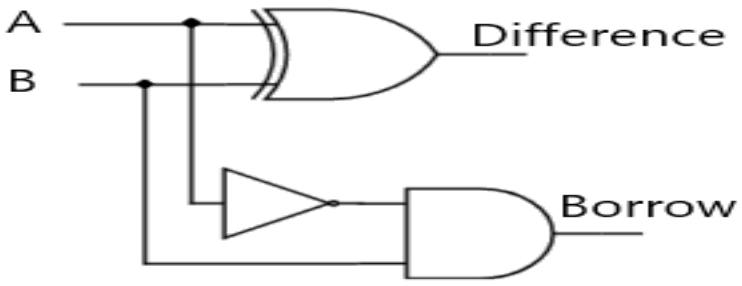



**Figure-3:Circuit Diagram of Half subtractor**

From the above truth table we can find the boolean expression.

$$\textbf{Difference} = A \oplus B$$
$$\textbf{Borrow} = A' B$$

From the equation we can draw the half-subtractor circuit as shown in the figure 3.

**2) Full Subtractor**

A full subtractor is a combinational circuit that performs subtraction involving three bits, namely A (minuend), B (subtrahend), and Bin (borrow-in) . It accepts three inputs: A (minuend), B (subtrahend) and a Bin (borrow bit) and it produces two outputs: D (difference) and Bout (borrow out). The logic symbol and truth table are shown below.

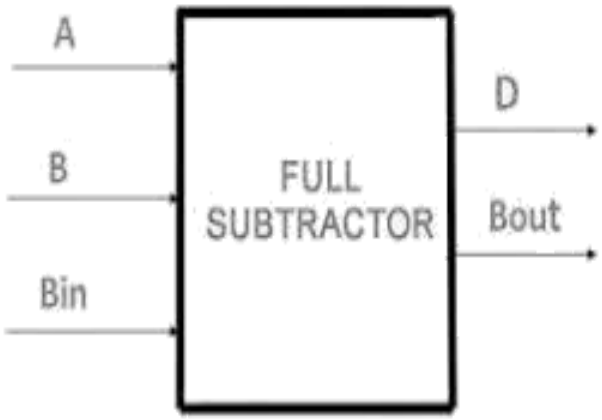



**Figure-4:Logic Symbol of Full subtractor**

| A | B | $B_{in}$ | D | $B_{out}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**Figure-5:Truth Table of Full subtractor**

From the above truth table we can find the boolean expression.

$$D = A \oplus B \oplus Bin$$

$$Bout = A' Bin + A' B + B Bin$$

From the equation we can draw the Full-subtractor circuit as shown in the figure 6.

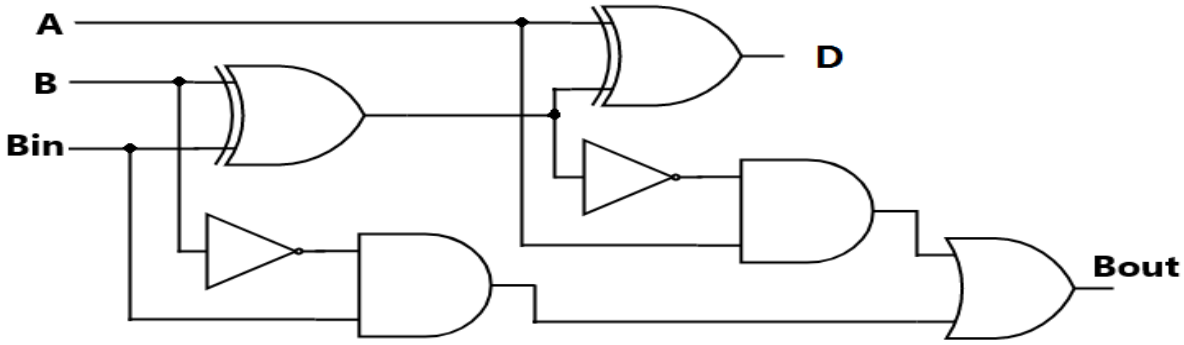



**Figure-6:Circuit Diagram of Full subtractor**

## PROCEDURE :

### 1)HALF SUBTRACTOR

## Simulator 1:

Step-1) Connect the Supply(+5V)  to the circuit.
Step-2) First press "ADD" button to add basic state of your output in the given table.
Step-3) Press the switches to select the required inputs "A" and "B".
Step-4) Press "ADD" button to add your inputs and outputs in the given table.
Step-5) Repeat steps 3&4 for next state of inputs and their corresponding outputs.
Step-6) Press the "PRINT" button after completing your simulation to get your results.

## Simulator 2:

Step-1) Enter the Boolean input "A" and "B".
Step-2) Enter the Boolean output for your corresponding inputs.
Step-3) Click on "Check" Button to verify your output.
Step-4) Click "Print" if you want to get print out of Truth Table.
Step-5) Click "Reset" if you want to reset inputs and outputs.

## 2)FULL SUBTRACTOR

## Simulator 1:

Step-1) Connect the Supply(+5V)  to the circuit.
Step-2) First press "ADD" button to add basic state of your output in the given table.
Step-3) Press the switches to select the required inputs "A" and "B" and "Bin".
Step-4) Press "ADD" button to add your inputs and outputs in the given table.
Step-5) Repeat steps 3&4 for next state of inputs and their corresponding outputs.
Step-6) Press the "PRINT" button after completing your simulation to get your results.

## Simulator 2:

Step-1) Enter the Boolean inputs "A" and "B" and "Bin".
Step-2) Enter the Boolean output for your corresponding inputs.
Step-3) Click on "Check" Button to verify your output.
Step-4) Click "Print" if you want to get print out of Truth Table.
Step-5) Click "Reset" if you want to reset inputs and outputs.

**TEST:**

1) Half subtractor is used to perform subtraction of

➔ **2 bits**

2) For subtracting 1 from 0, we use to take a **Borrow** from neighbouring bits

3) A and B is the input of a subtractor then the difference output will be

➔ **0 .**

4) Full subtractor is used to perform subtraction of

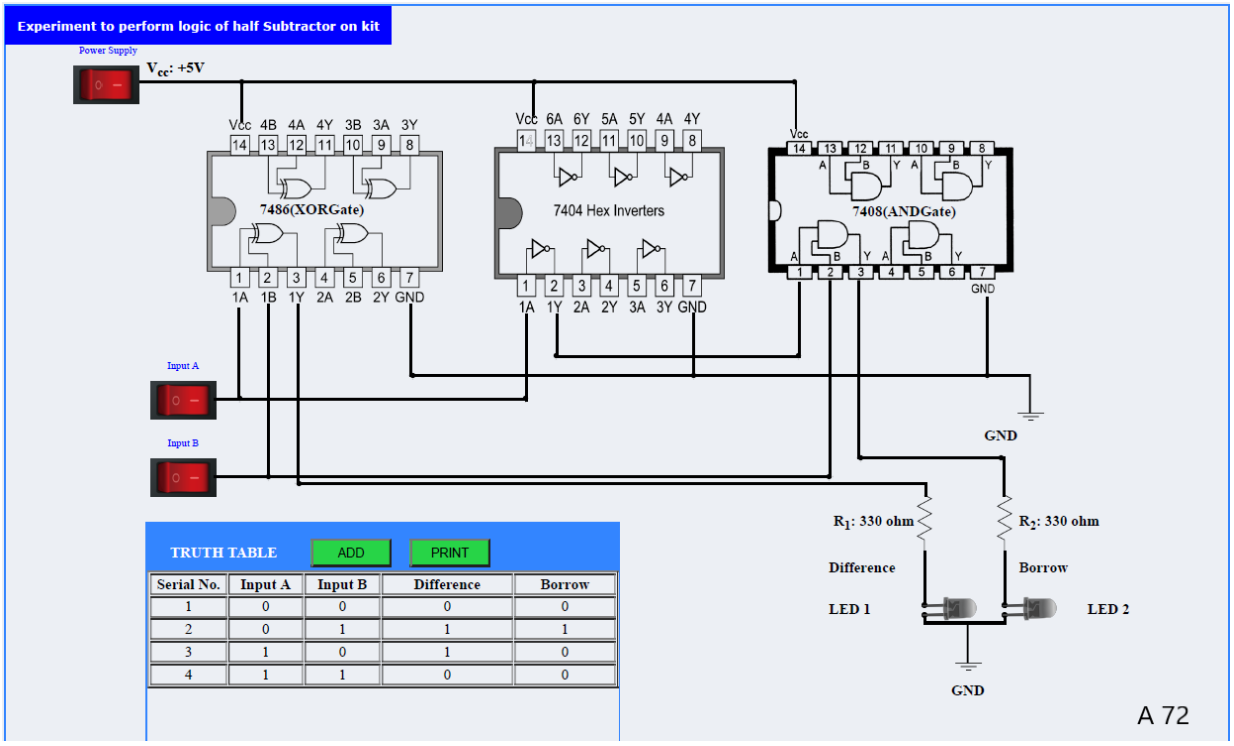
➔ **3 bits**

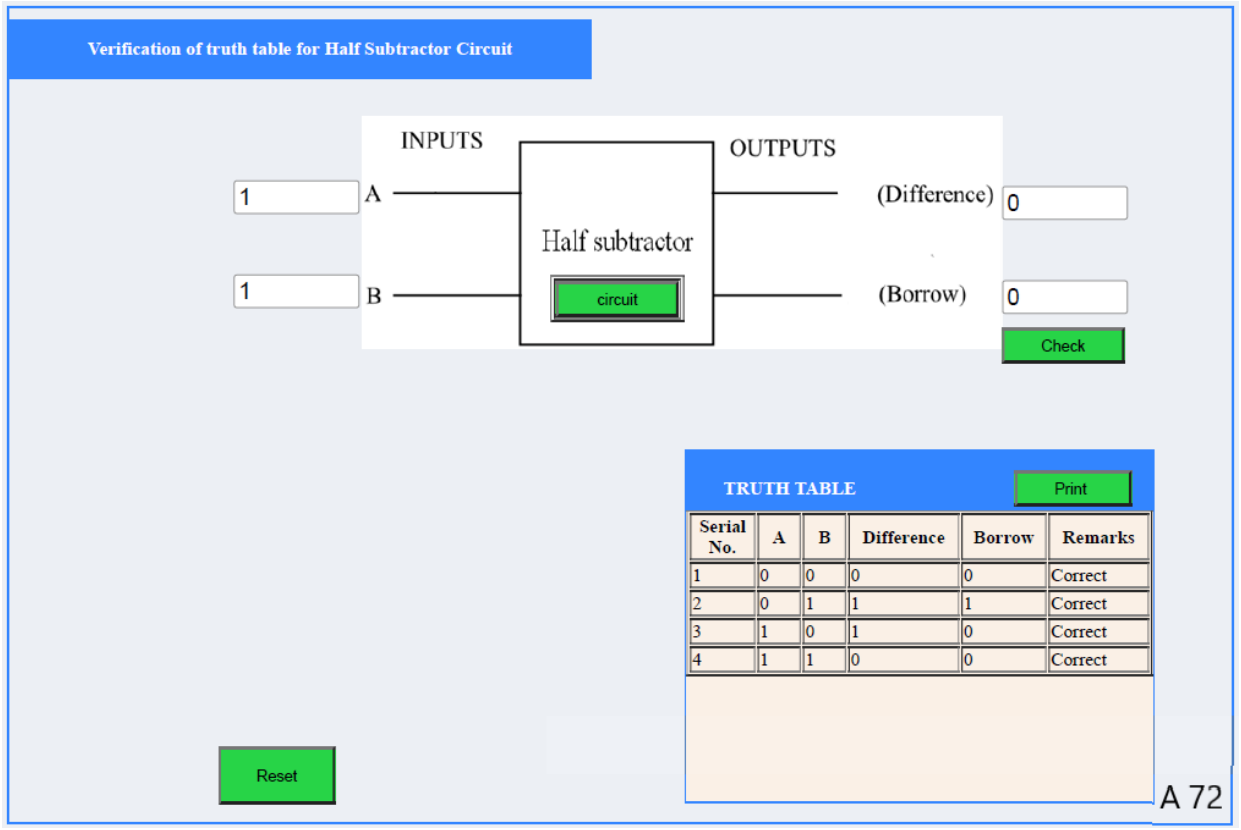
5) The output of a full subtractor is same as

➔ **Full adder**

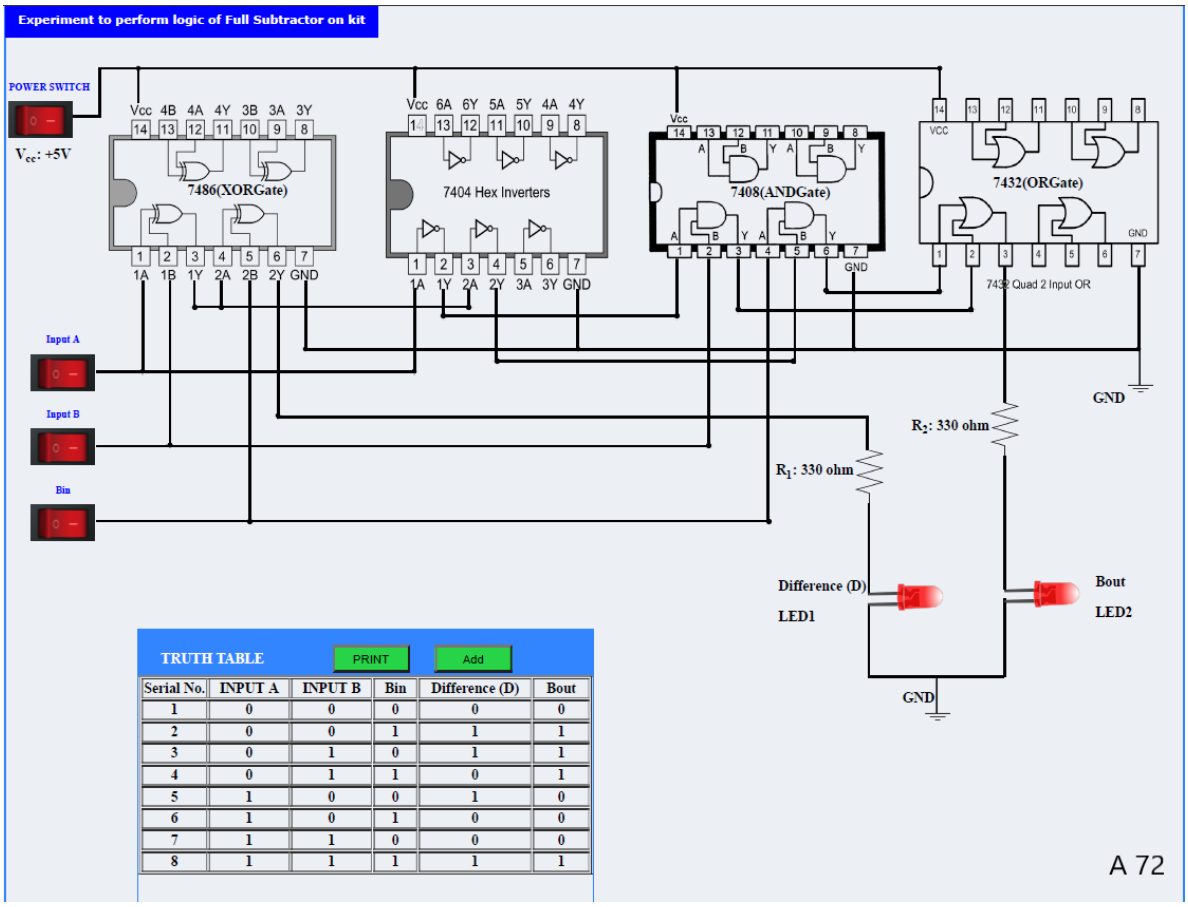
# SIMULATION RESULTS:

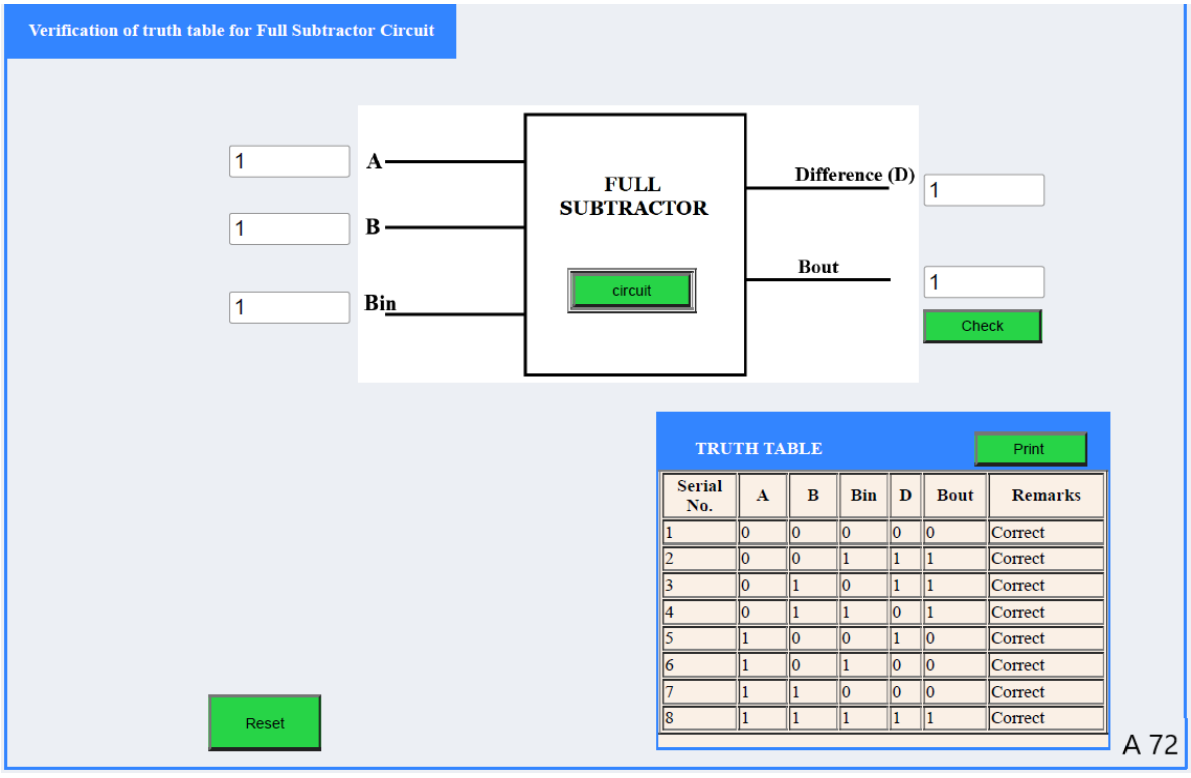
## 1) Half Subtractor :



# 2) Full Subtractor :



# Experiment- 5

## AIM :

To analyse the truth table of 4 * 2 decoder/de-multiplexer using NOT (7404) and AND (7408) logic gate ICs and 2 * 4 encoder using OR (7403) logic gate IC and to understand the working of 4 * 2 decoder and 2 * 4 encoder circuit with the help of LEDs display.

## THOERY :

Binary code of N digits can be used to store $2^N$ distinct elements of coded information. This is what encoders and decoders are used for. Encoders convert $2^N$ lines of input into a code of N bits and Decoders decode the N bits into $2^N$ lines.

### 1) 2x4 Decoder

The name “Decoder” means to translate or decode coded information from one format into another, so a digital decoder transforms a set of digital input signals into an equivalent decimal code at its output

A decoder is a combinational circuit that converts binary information from n input lines to a maximum of m=2^n unique output lines.

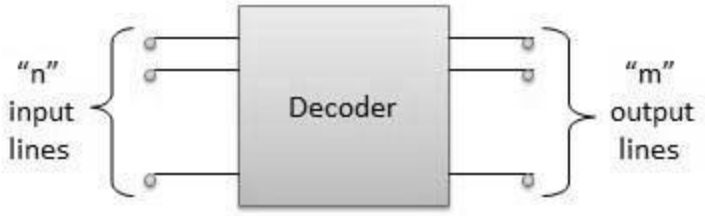



**Figure 1. Logic Diagram of Decoder**

## 1.1) 2-to-4 Binary Decoder

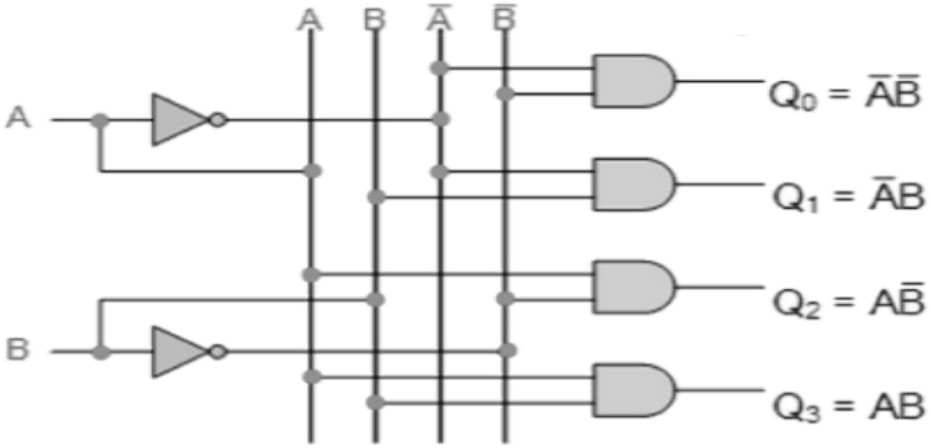
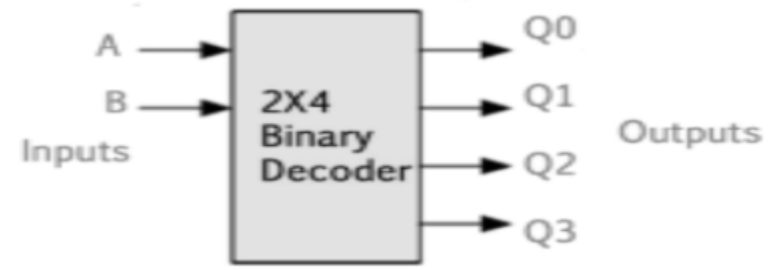



**Figure 2. Circuit Diagram of 2-to-4 Decoder**

The 2-to-4 line binary decoder depicted above consists of an array of four AND gates. The 2 binary inputs labelled A and B are decoded into one of 4 outputs, hence the description of 2-to-4 binary decoder. Each output represents one of the minterms of the 2 input variables, (each output = a minterm).




| A | B | Q0 | Q1 | Q2 | Q3 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

**Figure 3. Logic Diagram and Truth table of 2-to-4 Decoder**

The binary inputs A and B determine which output line from Q0 to Q3 is "HIGH" at logic level "1" while the remaining outputs are held "LOW" at logic "0" so only one output can be active (HIGH) at any one time.

Therefore, whichever output line is "HIGH" identifies the binary code present at the input, in other words it "decodes" the binary input.Some binary decoders have an additional input pin labelled "Enable" that controls the outputs from the device.
This extra input allows the decoders outputs to be turned "ON" or "OFF" as required. Output is only generated when the Enable input has value 1; otherwise, all outputs are 0. Only a small change in the implementation is required: the Enable input is fed into the AND gates which produce the outputs.

If Enable is 0, all AND gates are supplied with one of the inputs as 0 and hence no output is produced. When Enable is 1, the AND gates get one of the inputs as 1, and now the output depends upon the remaining inputs. Hence the output of the decoder is dependent on whether the Enable is high or low.

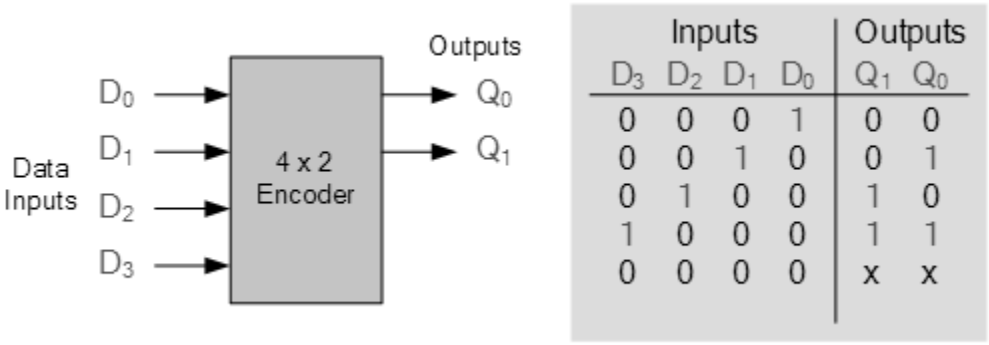
## 2) Encoder

An Encoder is a combinational circuit that performs the reverse operation of Decoder.It has maximum of $2^n$ input lines and 'n' output lines, hence it encodes the information from $2^n$ inputs into an n-bit code. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes $2^n$ input lines with 'n' bits.


**Figure 4. Logic Diagram of ENCODER**

### 2.1 ) 4 : 2 Encoder

The 4 to 2 Encoder consists of four inputs Y3, Y2, Y1 & Y0 and two outputs A1 & A0. At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output.




| Inputs | | | | Outputs | |
|---|---|---|---|---|---|
| $D_3$ | $D_2$ | $D_1$ | $D_0$ | $Q_1$ | $Q_0$ |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | x | x |

**Figure 5. Logic symbol and truth table of 4 to 2 encoder**

# PROCEDURE :

## 1)2X4 DECODER / De-multiplexer

Step-1) Connect the supply(+5V)  to the circuit.
Step-2) First press "ADD" button to add basic state of your output in the given table.
Step-3) Press the switches to select the required inputs named "A" and "B". Also check their corresponding outputs named "$Q_0$" and "$Q_1$" and "$Q_2$" and "$Q_3$".
Step-4) Press "ADD" button to add your inputs and outputs in the given table.
Step-5) Repeat step 3 & step 4 for next state of inputs and their corresponding outputs.
Step-6) Press the "PRINT" button after completing your simulation to get your results.

## 2)4X2 ENCODER

Step-1) Connect the supply(+5V)  to the circuit.
Step-2) First press "ADD" button to add basic state of your output in the given table.
Step-3) Press the switches to select the required inputs "$D_0$" and "$D_1$" and "$D_2$" and "$D_3$". Also check their corresponding ouyputs named "$Q_1$" and "$Q_0$".
Step-4) Press "ADD" button to add your inputs and outputs in the given table.
Step-5) Repeat step 3 & step 4 for next state of inputs and their corresponding outputs.
Step-6) Press the "PRINT" button after completing your simulation to get your results.

## TEST :

1) Invalid BCD can be made to valid BCD by adding with **0110**

2) Circuit that changes a code into a set of signals is called **Decoder**

3) A decoder converts n inputs to $2^n$ outputs.

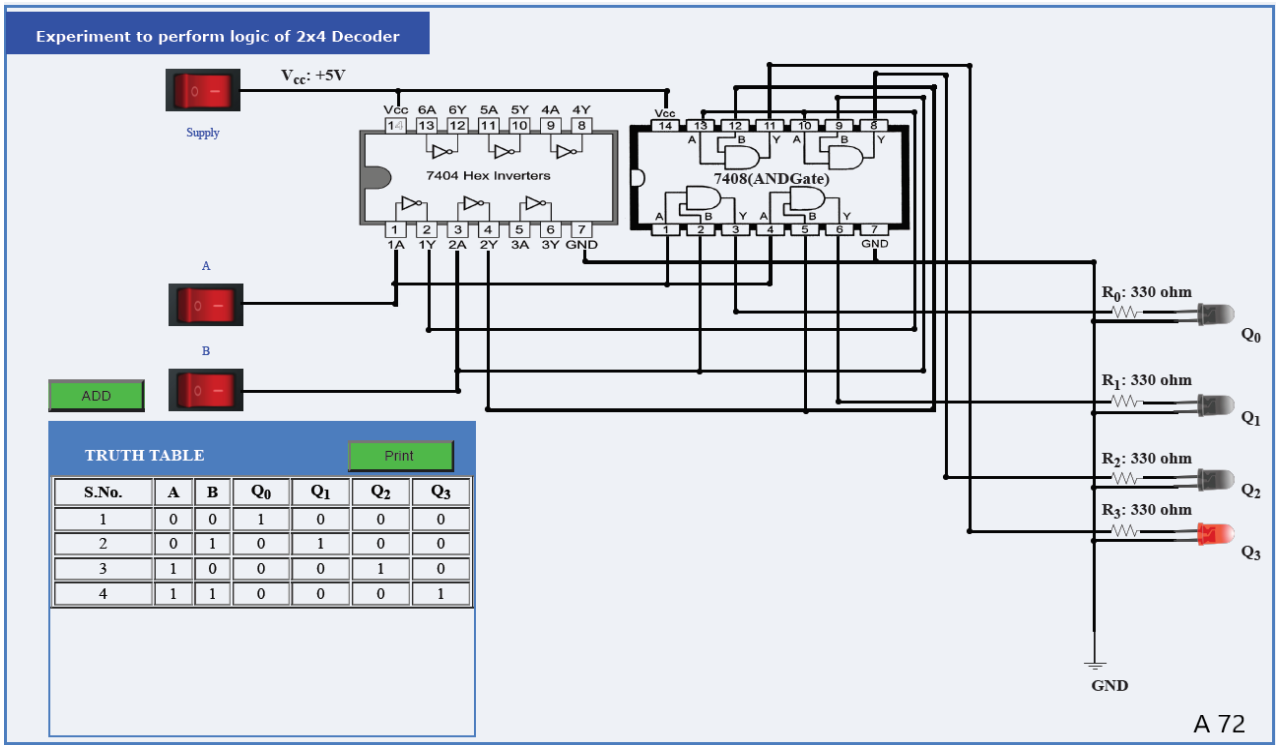
4) Decoders and Encoders are doing reverse operation.

**True**

5) BCD to 7 segment conversion in a **Decoder**

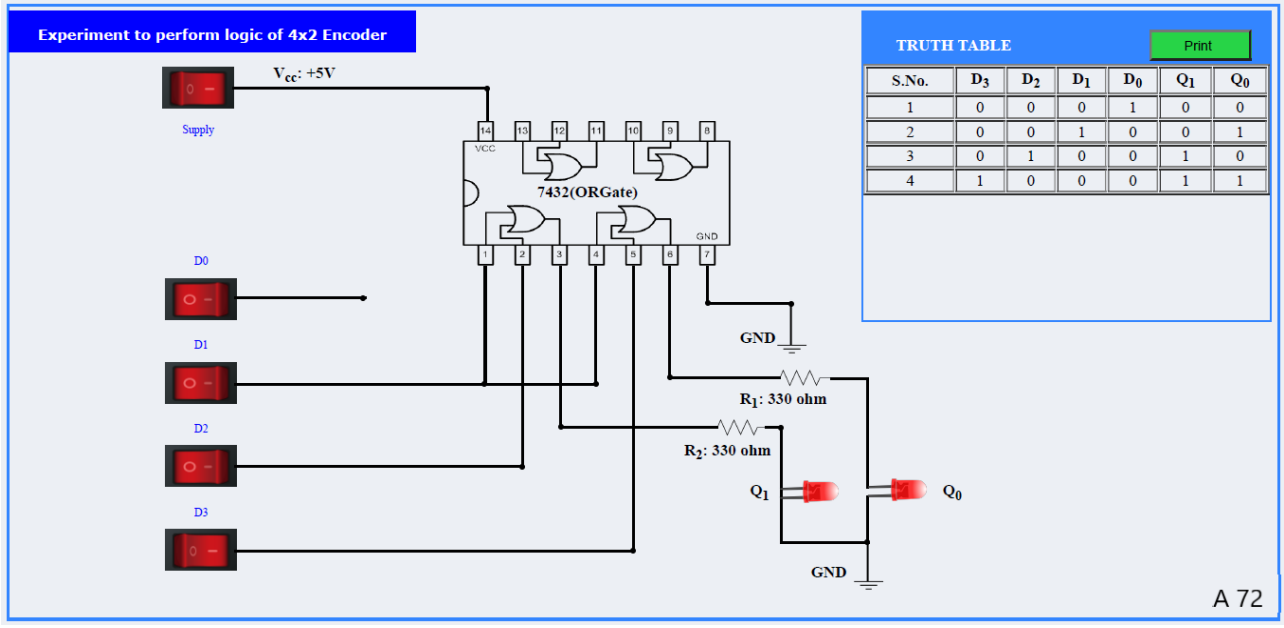
# SIMULATION RESULTS:

## 1) Decoder :




| S.No. | A | B | $Q_0$ | $Q_1$ | $Q_2$ | $Q_3$ |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 1 | 0 | 0 |
| 3 | 1 | 0 | 0 | 0 | 1 | 0 |
| 4 | 1 | 1 | 0 | 0 | 0 | 1 |

## 2) Encoder :




| S.No. | $D_3$ | $D_2$ | $D_1$ | $D_0$ | $Q_1$ | $Q_0$ |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 | 1 |
| 3 | 0 | 1 | 0 | 0 | 1 | 0 |
| 4 | 1 | 0 | 0 | 0 | 1 | 1 |

# Experiment- 6

## AIM :

To analyze the truth table and working of 1x4 De-Multiplexer by using 3-input NAND and 1-input NOT logic gate ICs and 4x1 Multiplexer by using 3-input AND, 3-input OR, and 1-input NOT logic gate ICs.

## THEORY :

The function of a multiplexer is to select the input of any 'n' input lines and feed that to one output line. The function of a de-multiplexer is to inverse the function of the multiplexer and the shortcut forms of the multiplexer. The de-multiplexers are mux and demux. Some multiplexers perform both multiplexing and de-multiplexing operations.

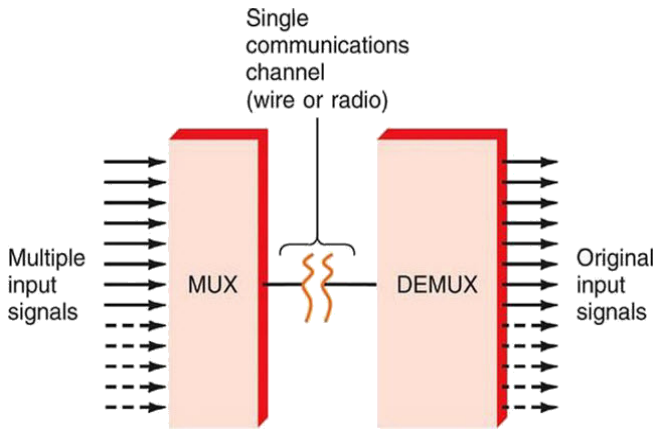



**Figure-1:Block diagram of Multiplexer and De-multiplexer**

1) Multiplexer is a device that has multiple inputs and a single line output. The select lines determine which input is connected to the output, and also to increase the amount of data that can be sent over a network within certain time. It is also called a data selector.

Multiplexers are classified into four types:

a) 2-1 multiplexer (1 select line)
b) 4-1 multiplexer (2 select lines)
c) 8-1 multiplexer(3 select lines)
d) 16-1 multiplexer (4 select lines)

**1.1) 4x1 Multiplexer**

4x1 Multiplexer has four data inputs D0, D1, D2 & D3, two selection lines S0 & S1 and one output Y. The block diagram of 4x1 Multiplexer is shown in the following figure.One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. Truth table of 4x1 Multiplexer is shown below.

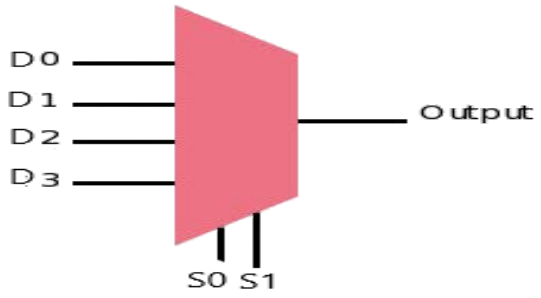



**Figure-2:Block diagram of 4x1 Multiplexer**

| Selection Lines | | Output |
|---|---|---|
| S0 | S1 | Y |
| 0 | 0 | $D_0$ |
| 0 | 1 | $D_1$ |
| 1 | 0 | $D_2$ |
| 1 | 1 | $D_3$ |

**Figure-3:Truth table of 4x1 Multiplexer**

2) De-multiplexer De-multiplexer is also a device with one input and multiple output lines. It is used to send a signal to one of the many devices. The main difference between a multiplexer and a de-multiplexer is that a multiplexer takes two or more signals and encodes them on a wire, whereas a de-multiplexer does reverse to what the multiplexer does.

De-multiplexer are classified into four types:

a)1-2 demultiplexer (1 select line)
b)1-4 demultiplexer (2 select lines)
c)1-8 demultiplexer (3 select lines)
d)1-16 demultiplexer (4 select lines)

## 2.2) 1x4 De-multiplexer

1x4 De-Multiplexer has one input Data(D), two selection lines, S0 & S1 and four outputs Y0, Y1, Y2 & Y3. The block diagram

of 1x4 De-Multiplexer is shown in the following figure.

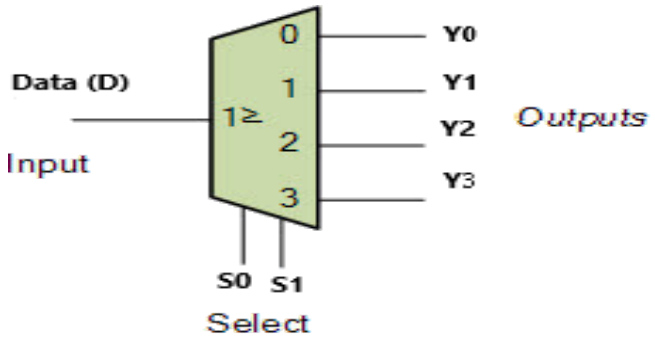



**Figure-4:Block diagram of 1x4 De-Multiplexer**

| Selection Inputs | | Outputs | | | |
|---|---|---|---|---|---|
| S0 | S1 | $Y_3$ | $Y_2$ | $Y_1$ | $Y_0$ |
| 0 | 0 | 0 | 0 | 0 | D |
| 0 | 1 | 0 | 0 | D | 0 |
| 1 | 0 | 0 | D | 0 | 0 |
| 1 | 1 | D | 0 | 0 | 0 |

**Figure-5:Truth table of 1x4 De-Multiplexer**

# PROCEDURE:

## 1) 4x1 Multiplexer

Step-1) Connect the supply(+5V)  to the circuit.
2) First press "ADD" button to add basic state of your output in the given table.
3) Press the switches "S0" and "S1"to select the desired input line.
4) Press "D0"/"D1"/"D2"/"D3" any one button to add your inputs.
5) Press "ADD" button to add your inputs and outputs in the given table.
6) Repeat step 3, 4 and step 5 for next state of inputs and their corresponding outputs.
7) Press the "PRINT" button after completing your simulation to get your results.

## 2) 1x4 De-Multiplexer

Step-1) Connect the supply(+5V)  to the circuit.
2) First press "ADD" button to add basic state of your output in the given table.
3) Press the switch Data(D) for Input.
4) Press switches "S0" and "S1" to select the desired input line.
5) Press "ADD" button to add your inputs and outputs in the given table.
6) Repeat step 4 and step 5 for next state of inputs and their corresponding outputs.
7) Press the "PRINT" button after completing your simulation to get your results.

**TEST :**

1) In a multiplexer, the selection of a particular input line is controlled by **Selected lines**

2) If the number of n selected input lines is equal to 2^m then it requires **m** select lines

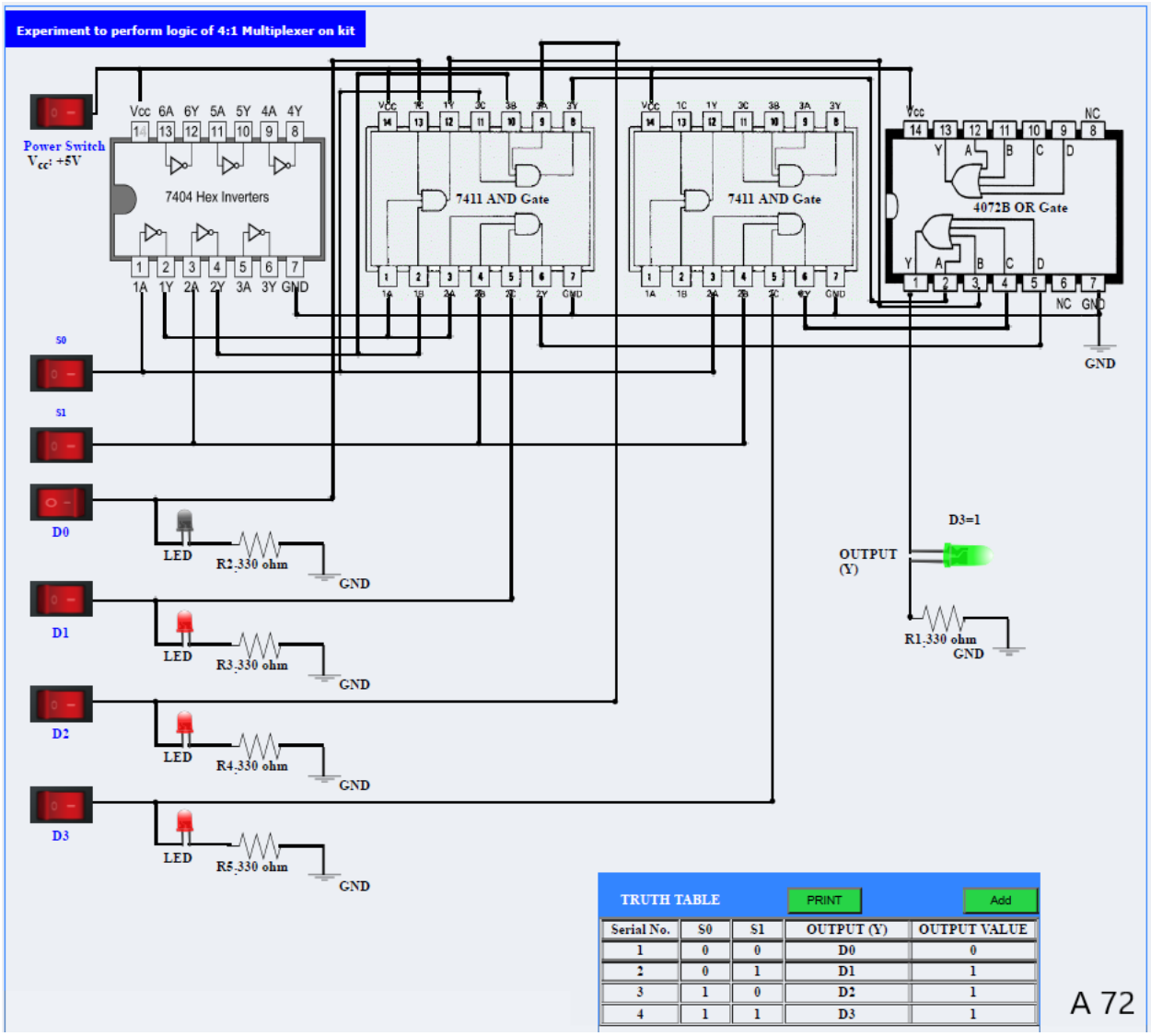
3) How many select lines would be required for an 8-line-to-1-line multiplexer **3 .**

4) How many NOT gates are required for the construction of a 4-to-1 multiplexer **2 .**

5) How many select lines are required for a 1-to-8 demultiplexer **3 .**

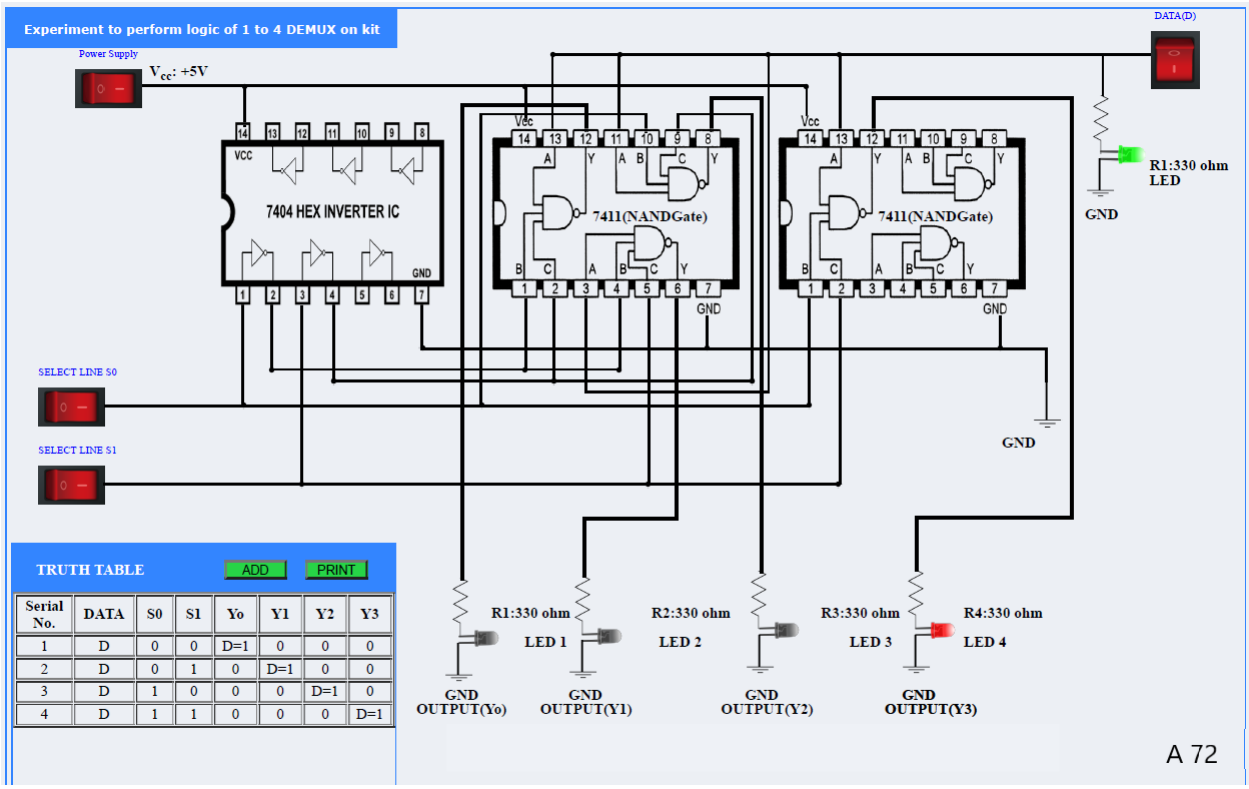
# SIMULATION RESULTS :

## 1) Multiplexer




| Serial No. | S0 | S1 | OUTPUT (Y) | OUTPUT VALUE |
|---|---|---|---|---|
| 1 | 0 | 0 | D0 | 0 |
| 2 | 0 | 1 | D1 | 1 |
| 3 | 1 | 0 | D2 | 1 |
| 4 | 1 | 1 | D3 | 1 |

# 2) De-Multiplexer




| Serial No. | DATA | S0 | S1 | Yo | Y1 | Y2 | Y3 |
|---|---|---|---|---|---|---|---|
| 1 | D | 0 | 0 | D=1 | 0 | 0 | 0 |
| 2 | D | 0 | 1 | 0 | D=1 | 0 | 0 |
| 3 | D | 1 | 0 | 0 | 0 | D=1 | 0 |
| 4 | D | 1 | 1 | 0 | 0 | 0 | D=1 |

# Experiment- 7

## AIM :

To verify the truth table and timing diagram of RS, JK, T and D flip-flops by using NAND & NOR gates ICs and analyses the circuit of RS, JK, T and D flip-flops with the help of LEDs display.

## THOERY :

A flip flop is an electronic circuit with two stable states that can be used to store binary data. The stored data can be changed by applying varying inputs. Flip-flops and latches are fundamental building blocks of digital electronics systems used in computers, communications, and many other types of systems.

1. R-S flip flop
2. D flip flop
3. J-K flip flop
4. T flip flop

### 1) RS flip flop

The basic NAND gate RS flip flop circuit is used to store the data and thus provides feedback from both of its outputs again back to its inputs. The RS flip flop actually has three inputs, SET, RESET and clock pulse.

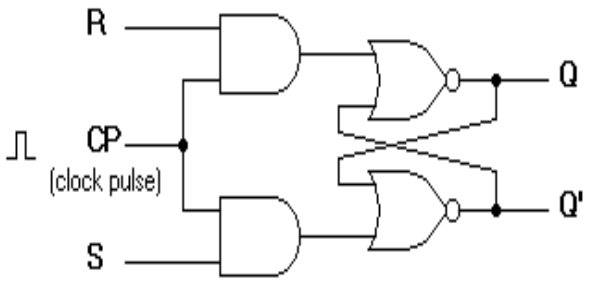



**Figure-1:R-S flip flop circuit diagram**

| INPUTS | | | OUTPUT | STATE |
|---|---|---|---|---|
| CLK | S | R | Q | |
| X | 0 | 0 | No Change | Previous |
| ↑ | 0 | 1 | 0 | Reset |
| ↑ | 1 | 0 | 1 | Set |
| ↑ | 1 | 1 | - | Forbidden |

**Figure-2:Characteristics table of R-S flip flop**

## 2) D flip flop :

A D flip flop has a single data input. This type of flip flop is obtained from the SR flip flop by connecting the R input through an inverter, and the S input is connected directly to data input. The modified clocked SR flip-flop is known as D-flip-flop and is shown below. From the truth table of SR flip-flop we see that the output of the SR flip-flop is in unpredictable state when the inputs are same and high. In many practical applications, these input conditions are not required. These input conditions can be avoided by making them complement of each other.

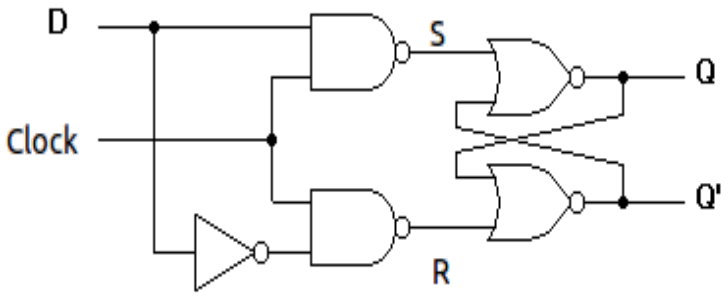



**Figure-3:Circuit diagram of D flip flop**

| Input | | | Output | |
|---|---|---|---|---|
| D | reset | clock | Q | Q' |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 |

**Figure-4:Characteristics table of D flip flop**

## 3) J-K flip flop

In a RS flip-flop the input R=S=1 leads to an indeterminate output. The RS flip-flop circuit may be re-joined if both inputs are 1 than also the outputs are complement of each other as shown in characteristics table below.

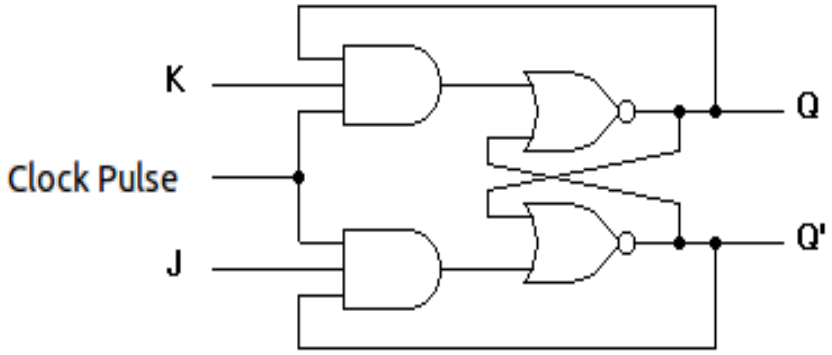



**Figure-5:Circuit diagram of J-K flip flop**

| Trigger | Inputs | | Output | | | | Inference |
|---|---|---|---|---|---|---|---|
| | | | Present State | | Next State | | |
| CLK | J | K | Q | Q' | Q | Q' | |
| ⊠ | x | x | - | | - | | Latched |
| ⊓ | 0 | 0 | 0 | 1 | 0 | 1 | No Change |
| ⊓ | | | 1 | 0 | 1 | 0 | |
| ⊓ | 0 | 1 | 0 | 1 | 0 | 1 | Reset |
| ⊓ | | | 1 | 0 | 0 | 1 | |
| ⊓ | 1 | 0 | 0 | 1 | 1 | 0 | Set |
| ⊓ | | | 1 | 0 | 1 | 0 | |
| ⊓ | 1 | 1 | 0 | 1 | 1 | 0 | Toggles |
| ⊓ | | | 1 | 0 | 0 | 1 | |

**Figure-6:Characteristics table of J-K flip flop**

## 4) T flip flop

T flip-flop is known as toggle flip-flop. The T flip-flop is modification of the J-K flip-flop. Both the JK inputs of the JK flip – flop are held at logic 1 and the clock signal continuous to change as shown in table below.

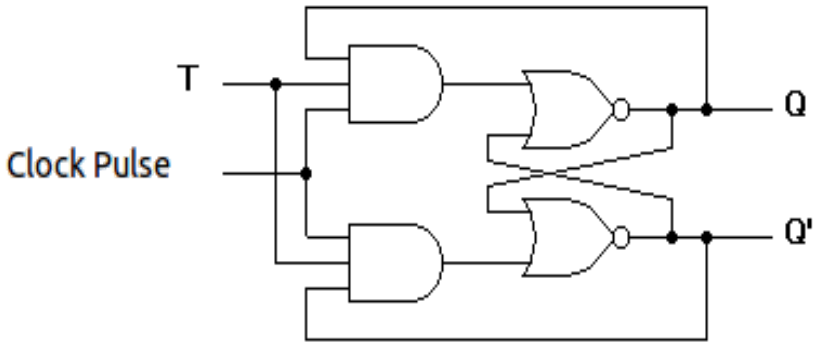



**Figure-7:Circuit diagram of T flip flop**

T flip-flop

| T | Clock | Q | Q' |
|---|---|---|---|
| 0 | ↑ | Q | Q' |
| 1 | ↑ | Q' | Q |
| X | ↓ | Q | Q' |

**Figure-8:Characteristics table of T flip flop**

# PROCEDURE :

## 1) SR flip flop

Step-1) Connect the supply(+5V)  to the circuit.
Step-2) First press "ADD" button to add basic state of your output in the given table.
Step-3) Press the switches to select the required inputs "S" and "R" and apply the clock pulse.
Step-4) Press "ADD" button to add your inputs and outputs in the given table and their corresponding graph.
Step-5) Repeat steps 3&4 for next state of inputs and their corresponding outputs.
Step-6) Press the "Print" button after completing your simulation to get your results.

## 2) J-K flip flop

Step-1) Connect the supply(+5V)  to the circuit.
Step-2) First press "ADD" button to add basic state of your output in the given table.
Step-3) Press the switches to select the required inputs "J" and "K" and apply the clock pulse.
Step-4) Press "ADD" button to add your inputs and outputs in the given table and their corresponding graph.
Step-5) Repeat step 3 & step 4 for next state of inputs and their corresponding outputs.
Step-6) Press the "Print" button after completing your simulation to get your results.

**TEST:**

1) The output of latches will remain in set/reset until

➔ **The trigger pulse is given to change the state**

2) The sequential circuit is also called

➔ **Latch**

3) The basic latch consists of

➔ **Two inverters**

4) What is a trigger pulse?

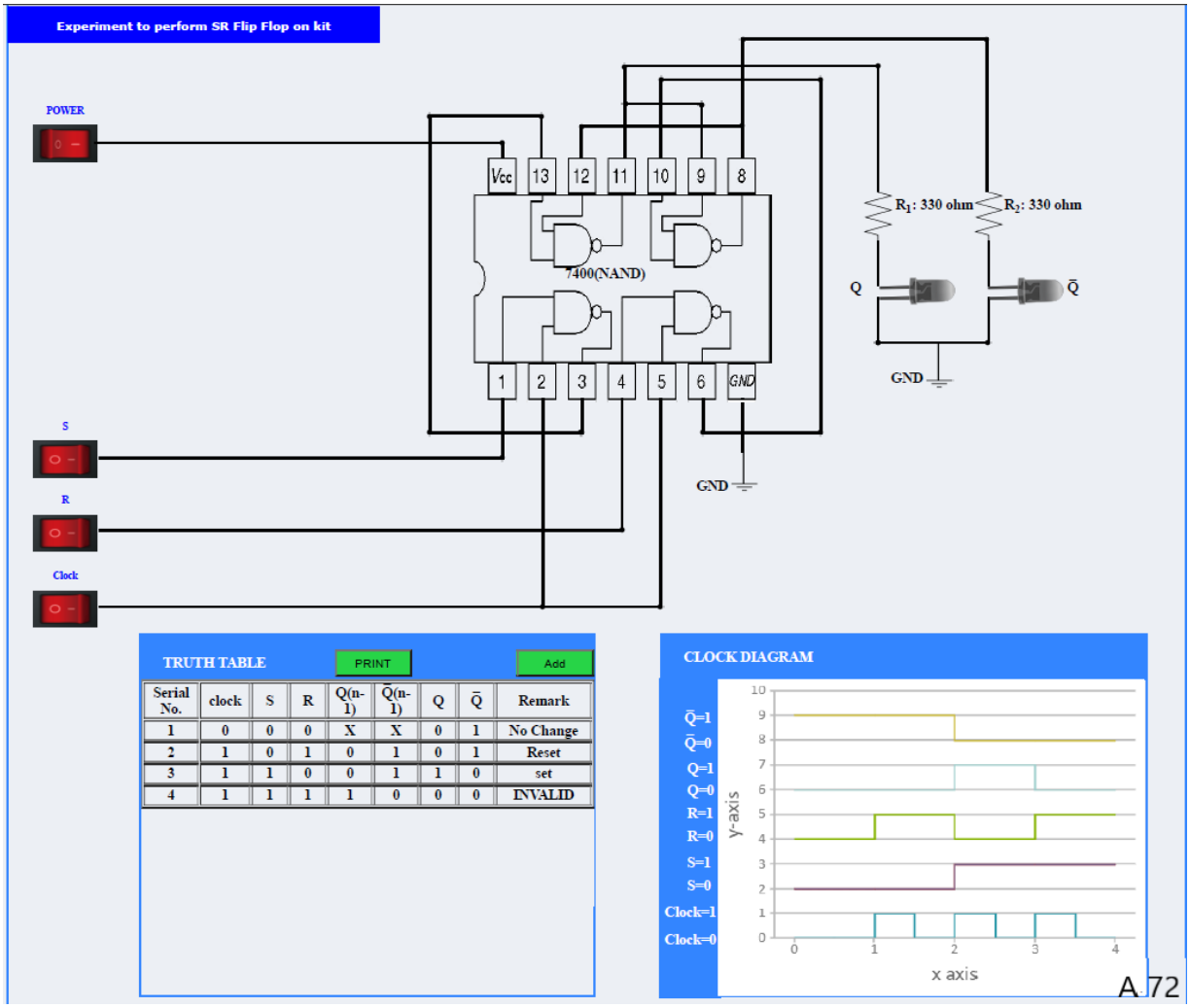
➔ **A pulse that starts a cycle of operation**

5) The truth table for an S-R flip-flop has how many valid entries

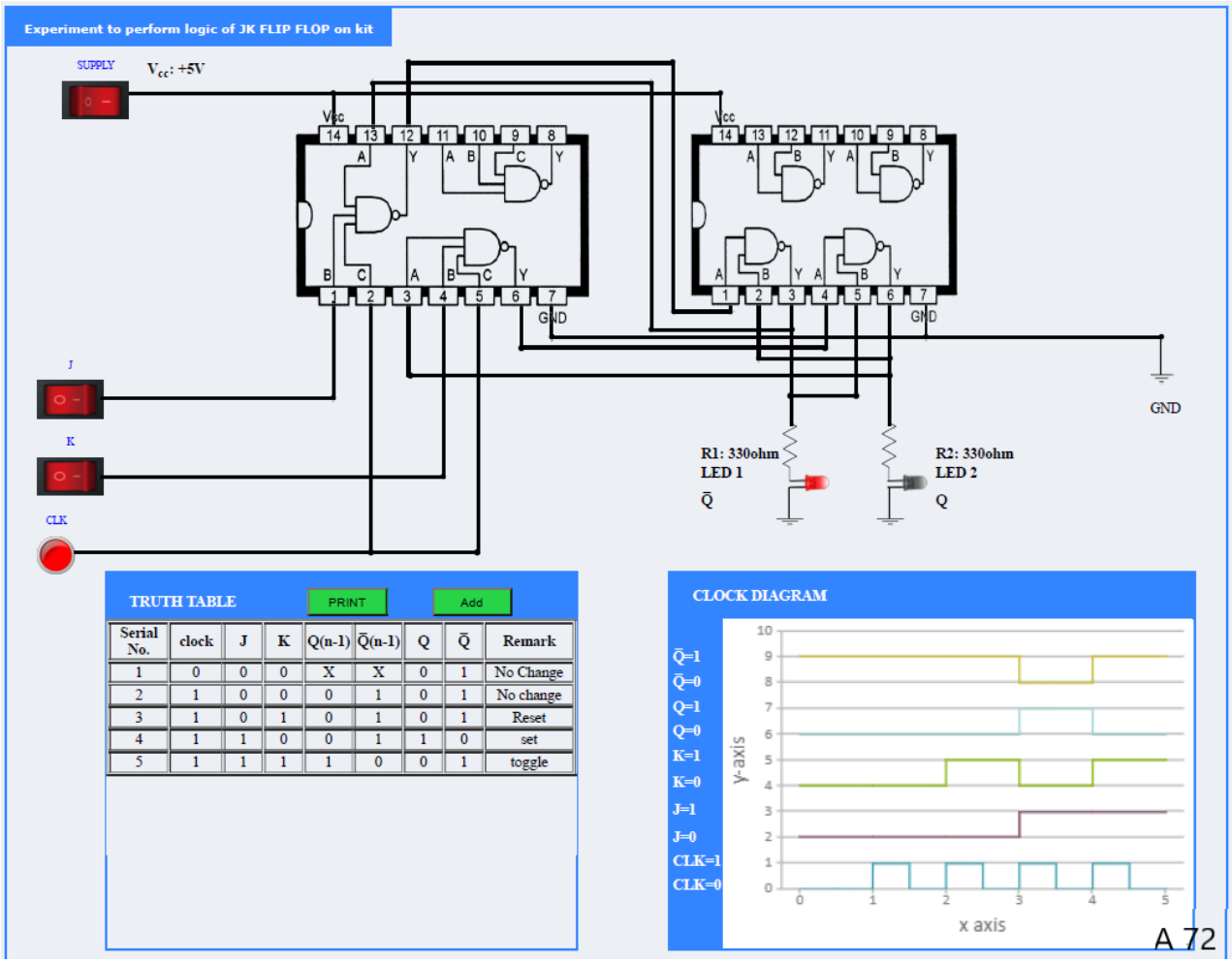
➔ **3 .**

# SIMULATION RESULTS:

## 1) R-S Flip Flop-



# 2) J- K Flip Flop-




| Serial No. | clock | J | K | Q(n-1) | Q̄(n-1) | Q | Q̄ | Remark |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | X | X | 0 | 1 | No Change |
| 2 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | No change |
| 3 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | Reset |
| 4 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | set |
| 5 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | toggle |

# Experiment - 8

## AIM :

To verify the truth table and timing diagram of 4-bit synchronous parallel counter and 4-bit asynchronous parallel counter by using JK flip flop ICs and analyse the circuit of 4-bit synchronous parallel counter and 4-bit asynchronous parallel counter with the help of LEDs display.

## THEORY :

A counter is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal. Counters are used in digital electronics for counting purpose, they can count specific event happening in the circuit. For example, in UP counter a counter increases count for every rising edge of clock.

### Classification of Counters

Counters are broadly divided into two categories

1. Asynchronous counter
2. Synchronous counter

## 1) Asynchronous Counter

In asynchronous counter we don't use universal clock, only first flip flop is driven by main clock and the clock input of rest of the following counters is driven by output of previous flip flops. We can understand it by following diagram-

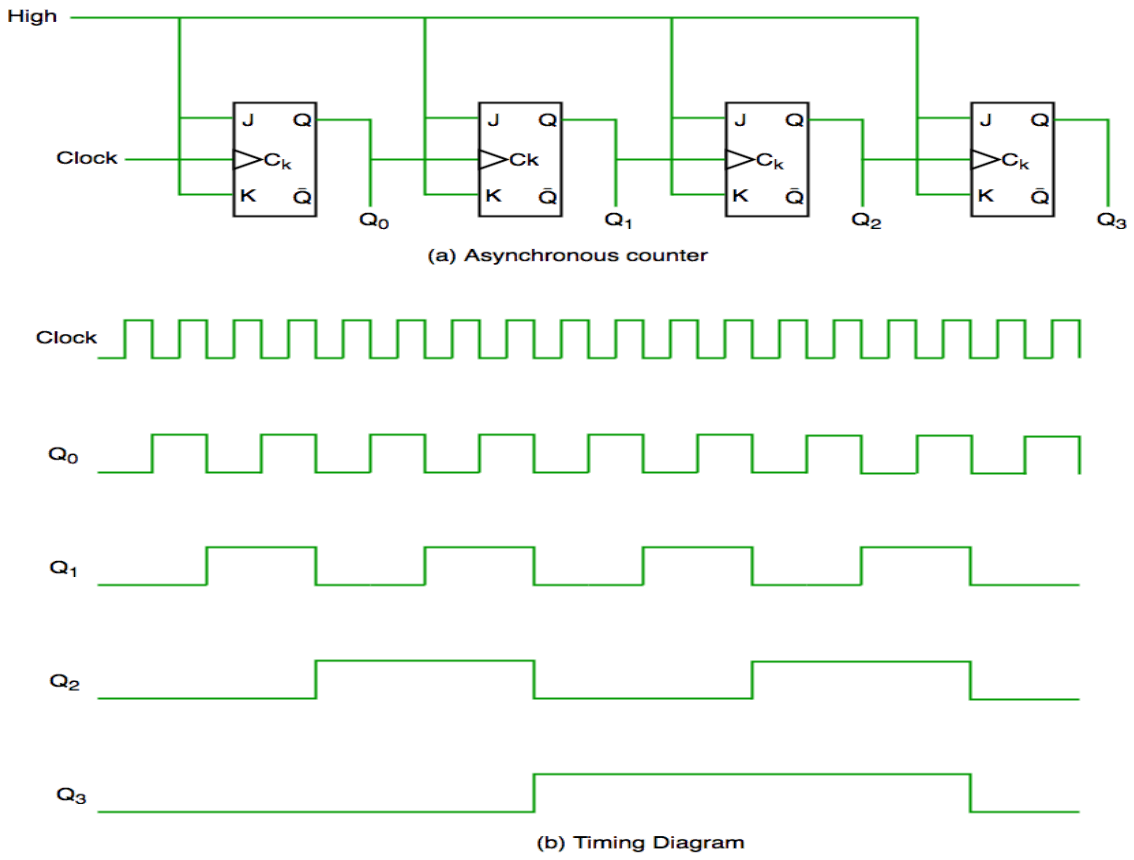


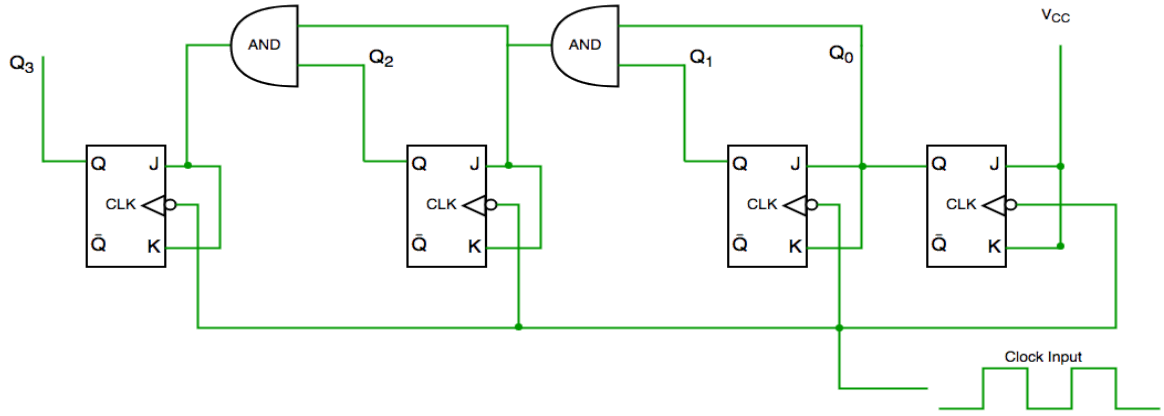

**Figure-1: Asynchronous Counter Circuit and Timing Diagram**

It is evident from timing diagram that Q0 is changing as soon as the rising edge of clock pulse is encountered, Q1 is changing when rising edge of Q0 is encountered(because Q0 is like clock

pulse for second flip flop) and so on. In this way ripples are generated through Q0,Q1,Q2,Q3 hence it is also called RIPPLE counter.

## 2) Synchronous Counter

Unlike the asynchronous counter, synchronous counter has one global clock which drives each flip flop so output changes in parallel. The one advantage of synchronous counter over asynchronous counter is, it can operate on higher frequency than asynchronous counter as it does not have cumulative delay because of same clock is given to each flip flop.



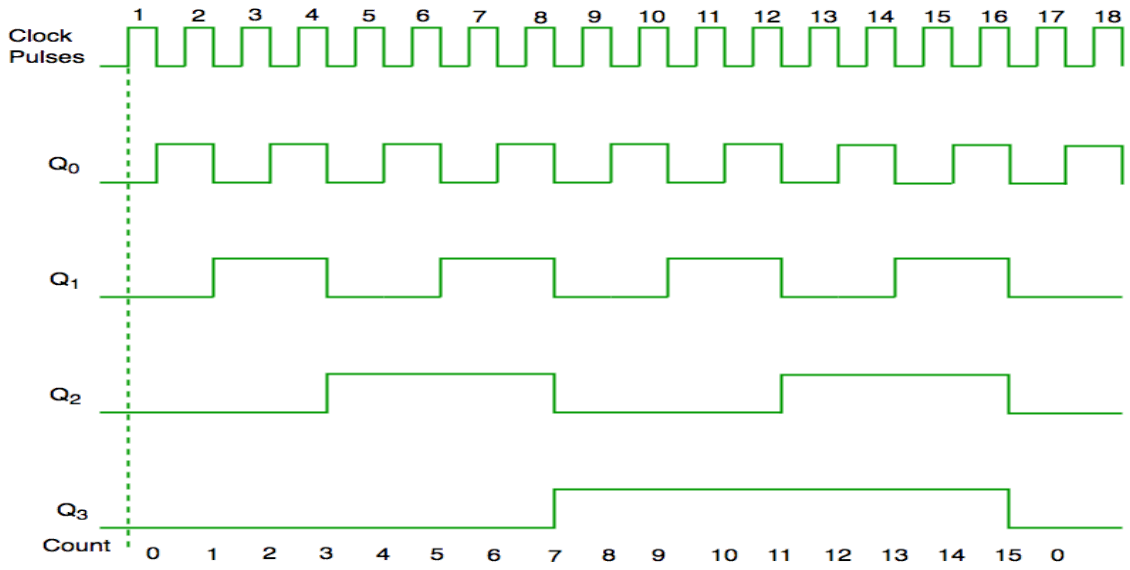



**Figure-2: Synchronous Counter Circuit and Timing Diagram**

From circuit diagram we see that Q0 bit gives response to each falling edge of clock while Q1 is dependent on Q0, Q2 is dependent on Q1 and Q0 , Q3 is dependent on Q2,Q1 and Q0.

## PROCEDURE:

### Synchronous Counter

Step-1) Press Switch to supply 5V to the circuit.

The switch in ON state is  and the switch in OFF state is 

Step-2) Press Counter button to start the counter.

Step-3) Different combination of LEDs lit up for different combination of inputs.

The LED in OFF state is  and the LED in ON state is 
.
The data is simultaneously added to the Truth Table.
Step-4) Repeat Steps 2 to 3 for another set of data.
Step-5) Click on "Generate Waveform" Button to generate the Timing Diagram .
Step-6) Click "Print" to get the print out of the Truth Table and the Timing Diagram.

**Asynchronous Counter**

Step-1) Press Switch to supply 5V to the circuit.

The switch in ON state is  and the switch in OFF state is 
Step-2) Press Counter button to start the counter.
Step-3) Different combination of LEDs lit up for different combination of inputs.

The LED in OFF state is  and the LED in ON state is 
.
The data is simultaneously added to the Truth Table.

Step-4) Repeat Steps 2 to 3 for another set of data.
Step-5) Click on "Generate Waveform" Button to generate the Timing Diagram .
Step-6) Click "Print" to get the print out of the Truth Table and the Timing Diagram.

**TEST :**

1) In digital logic, a counter is a device which

➔ **Stores the number of times a particular event or process has occurred**

2) A counter circuit is usually constructed of

➔ **A number of flip-flops connected in cascade**

3) Parallel outputs of a counter circuit represent the

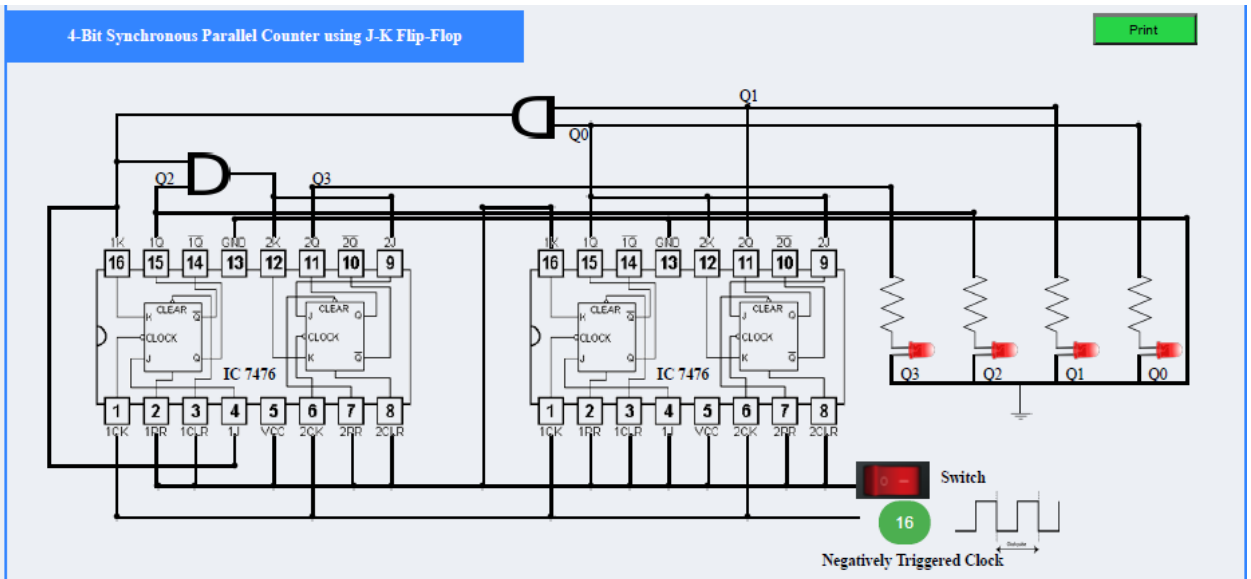
➔ **Clock count**

4) A decimal counter has **10** states.

5) A counter is implemented using three (3) flip-flops, possibly it will have **8** maximum output status

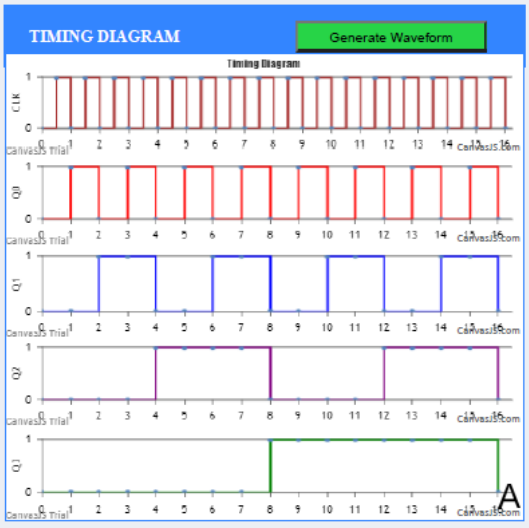
# SIMULATION RESULTS:

## 1) Synchronous Counter :

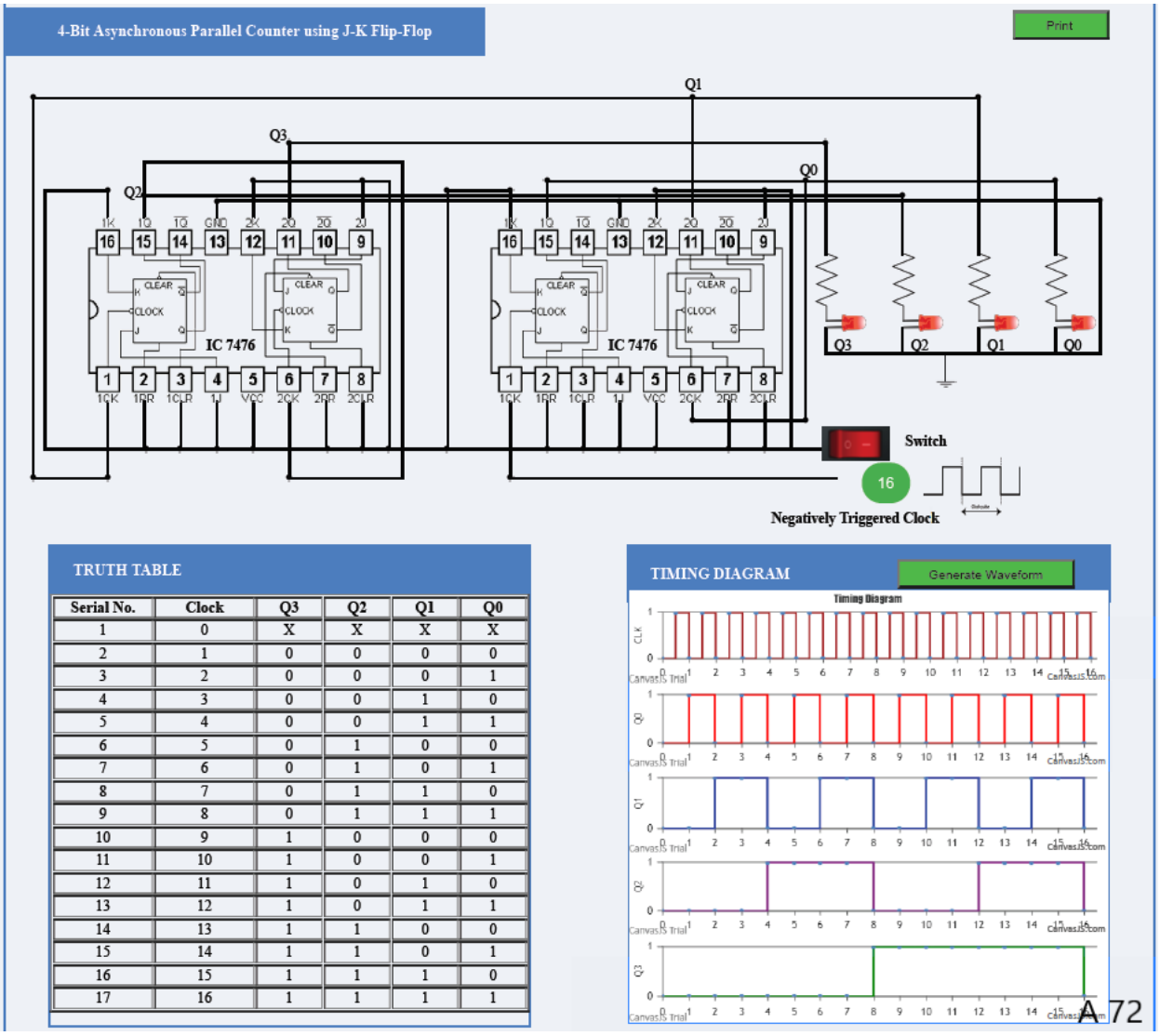



**TRUTH TABLE**

| Serial No. | Clock | Q3 | Q2 | Q1 | Q0 |
|---|---|---|---|---|---|
| 1 | 0 | X | X | X | X |
| 2 | 1 | 0 | 0 | 0 | 0 |
| 3 | 2 | 0 | 0 | 0 | 1 |
| 4 | 3 | 0 | 0 | 1 | 0 |
| 5 | 4 | 0 | 0 | 1 | 1 |
| 6 | 5 | 0 | 1 | 0 | 0 |
| 7 | 6 | 0 | 1 | 0 | 1 |
| 8 | 7 | 0 | 1 | 1 | 0 |
| 9 | 8 | 0 | 1 | 1 | 1 |
| 10 | 9 | 1 | 0 | 0 | 0 |
| 11 | 10 | 1 | 0 | 0 | 1 |
| 12 | 11 | 1 | 0 | 1 | 0 |
| 13 | 12 | 1 | 0 | 1 | 1 |
| 14 | 13 | 1 | 1 | 0 | 0 |
| 15 | 14 | 1 | 1 | 0 | 1 |
| 16 | 15 | 1 | 1 | 1 | 0 |
| 17 | 16 | 1 | 1 | 1 | 1 |



# 2) Asynchronous Counter :




TRUTH TABLE

| Serial No. | Clock | Q3 | Q2 | Q1 | Q0 |
|---|---|---|---|---|---|
| 1 | 0 | X | X | X | X |
| 2 | 1 | 0 | 0 | 0 | 0 |
| 3 | 2 | 0 | 0 | 0 | 1 |
| 4 | 3 | 0 | 0 | 1 | 0 |
| 5 | 4 | 0 | 0 | 1 | 1 |
| 6 | 5 | 0 | 1 | 0 | 0 |
| 7 | 6 | 0 | 1 | 0 | 1 |
| 8 | 7 | 0 | 1 | 1 | 0 |
| 9 | 8 | 0 | 1 | 1 | 1 |
| 10 | 9 | 1 | 0 | 0 | 0 |
| 11 | 10 | 1 | 0 | 0 | 1 |
| 12 | 11 | 1 | 0 | 1 | 0 |
| 13 | 12 | 1 | 0 | 1 | 1 |
| 14 | 13 | 1 | 1 | 0 | 0 |
| 15 | 14 | 1 | 1 | 0 | 1 |
| 16 | 15 | 1 | 1 | 1 | 0 |
| 17 | 16 | 1 | 1 | 1 | 1 |

# Experiment - 9

## AIM :

To analyse the truth table of 1-bit comparator by using NOT, AND and NOR logic gate ICs and 2-bit comparator by using 1-input NOT, 3-input AND, 2-input AND, 3-input OR and 2-input Ex-NOR logic gate ICs and to understand the working of 1-bit comparator and 2- bit comparator with the help of LEDs display.

## THEORY :

A magnitude digital comparator is a combinational circuit that compares two digital or binary numbers in order to find out whether one binary number is equal, less than or greater than the other binary number. We logically design a circuit for which we will have two inputs one for A and other for B and have three output terminals, one for A > B condition, one for A = B condition and one for A < B condition.

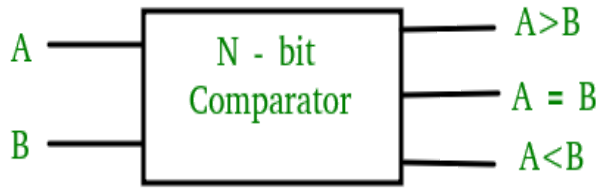



**Figure-1: Block Diagram of Comparator**

1) 1-Bit Magnitude Comparator :

A comparator used to compare two bits is called a single bit comparator. It consists of two inputs each for two single bit numbers and three outputs to generate less than, equal to and greater than between two binary numbers. The truth table for a 1-bit comparator is given below :

| A | B | A<B | A=B | A>B |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

**Figure-2: Truth Table of 1-Bit Comparator**

From the above truth table logical expressions for each output can be expressed as follows:
A > B : AB'
A < B : A'B
A = B : A'B' + AB
By using these Boolean expressions, we can implement a logic circuit for this comparator as given below :

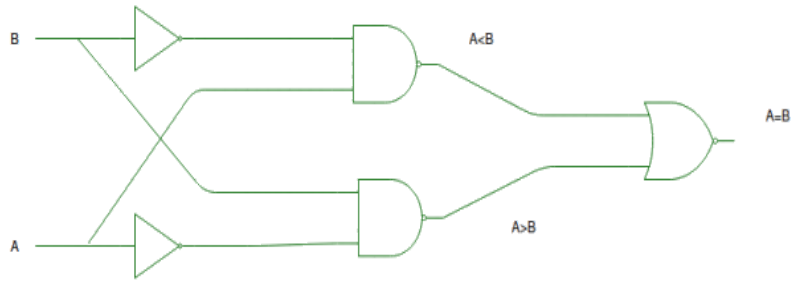



**Figure-3: Logic Circuit of 1-Bit Comparator**

2) 2-Bit Magnitude Comparator :

A comparator used to compare two binary numbers each of two bits is called a 2-bit magnitude comparator. It consists of four inputs and three outputs to generate less than, equal to and greater than between two binary numbers.

The truth table for a 2-bit comparator is given below:

| INPUT | | | | OUTPUT | | |
|---|---|---|---|---|---|---|
| A1 | A0 | B1 | B0 | A<B | A=B | A>B |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |

**Figure-4: Truth Table of 2-Bit Comparator**

From the above truth table logical expressions for each output can be expressed as follows:

From the above truth table logical expressions for each output can be expressed as follows:

**A > B : $A_1B_1' + A_0B_1'B_0' + A_1A_0B_0'$**

**A = B : $A_1'A_0'B_1'B_0' + A_1'A_0B_1'B_0 + A_1A_0B_1B_0 + A_1A_0'B_1B_0'$**
**: $A_1'B_1' (A_0'B_0' + A_0B_0) + A_1B_1 (A_0B_0 + A_0'B_0')$**
**: $(A_0B_0 + A_0'B_0') (A_1B_1 + A_1'B_1')$**
**: ($A_0$ Ex-Nor $B_0$) ($A_1$ Ex-Nor $B_1$)**

**A < B : $A_1'B_1 + A_0'B_1B_0 + A_1'A0'B0$**

By using these Boolean expressions, we can implement a logic circuit for this comparator as given below :

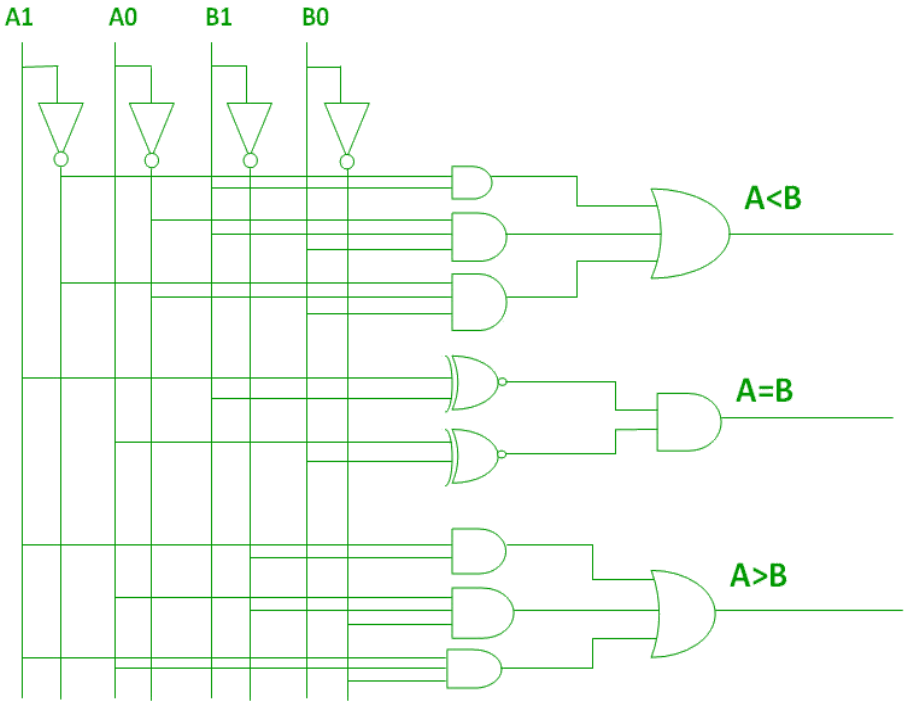



**Figure-5: Logic Circuit of 2-Bit Comparator**

Applications of Comparators : 1. Comparators are used in central processing units (CPUs) and microcontrollers (MCUs).
2. These are used in control applications in which the binary numbers representing physical variables such as temperature, position, etc. are compared with a reference value.
3. Comparators are also used as process controllers and for Servo motor control.
4. Used in password verification and biometric applications.

## PROCEDURE:

### 1- Bit Comparator

Simulator 1:

Step-1) Switch ON the power supply button to supply 5V to the circuit.
Step-2) Press Switch 1 for input A and Switch 2 for input B.

The switch in ON state is  and the switch in OFF state is 
Step-3) i)When the input A is greater than the input B,LED 1 lits up.
ii)When the input A is lesser than the input B,LED 2 lits up.
iii)When the input A is equal to the input B,LED 3 lits up.

The LED in OFF state is  and the LED in ON state is 
Step-4) Click on "Add" Button to add data to the Truth Table.
Step-5) Repeat Steps 2 to 4 for another set of data.
Step-6) Click "Print" to get the print out of the Truth Table.

Simulator 2:

Step-1) Enter the Boolean input "A" and "B".
Step-2) Enter the Boolean output for your corresponding inputs.
Step-3) Click on "Check" Button to verify your output.
Step-4) Click "Print" if you want to get print out of Truth Table.

**2- Bit Comparator**

Simulator 1:

Step-1) Switch ON the power supply button to supply 5V to the circuit.
Step-2) Press Switch 1 for input $A_1$,Switch 2 for input $A_0$,Switch 3 for input $B_1$and Switch 4 for input $B_0$.

The switch in ON state is  and the switch in OFF state is 
Step-3) i)When the input $A_1A_0$ is greater than the input $B_1B_0$,LED 1 lits up.
ii)When the input $A_1A_0$ is lesser than the input $B_1B_0$,LED 2 lits up.
iii)When the input $XA_1A_0$ is equal to the input $B_1B_0$,LED 3 lits up.

The LED in OFF state is  and the LED in ON state is 
Step-4) Click on "Add" Button to add data to the Truth Table.
Step-5) Repeat Steps 2 to 4 for another set of data.
Step-6) Click "Print" to get the print out of the Truth Table.

Simulator 2:

Step-1) Enter the two bit Boolean input "A" and "B".
Step-2) Inputs should be written such that for 'A'="A1A0" and for 'B'="B1B0"

Step-3) Enter the Boolean output for your corresponding inputs.
Step-4) Click on "Check" Button to verify your output.
Step-5) Click "Print" if you want to get print out of Truth Table.

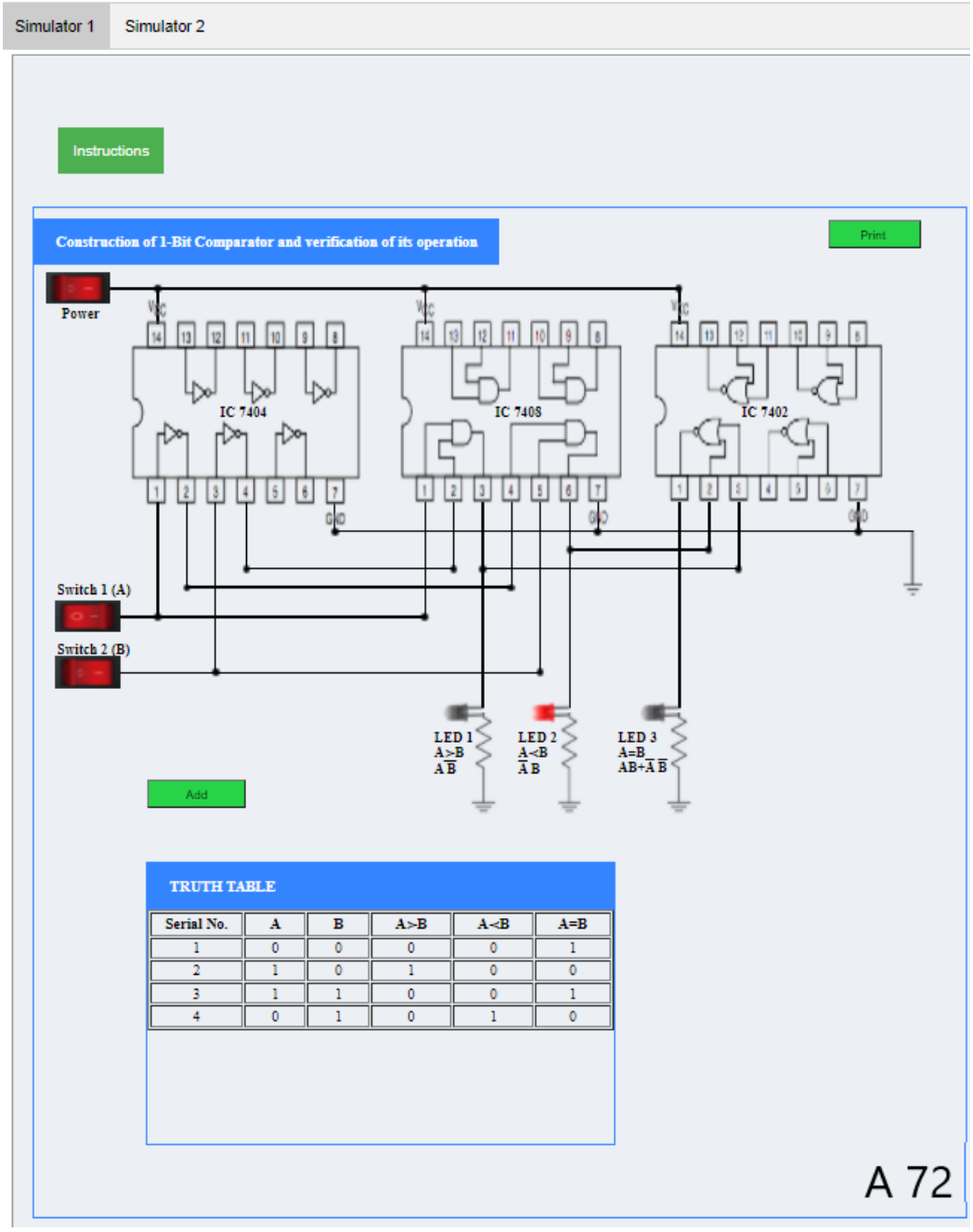
**TEST:**

1) The purpose of a Digital Comparator is **To compare a set of variables or unknown numbers**
2) In a comparator, if we get input as A>B then the output will be **1 .**
3) If two numbers are not equal then binary variable will be **0**
4) One that is not the outcome of magnitude comparator is **A - B .**
5) Comparators are used in **CPU**

# SIMULATION RESULTS:

1) 1 Bit comparator

Simulation 1



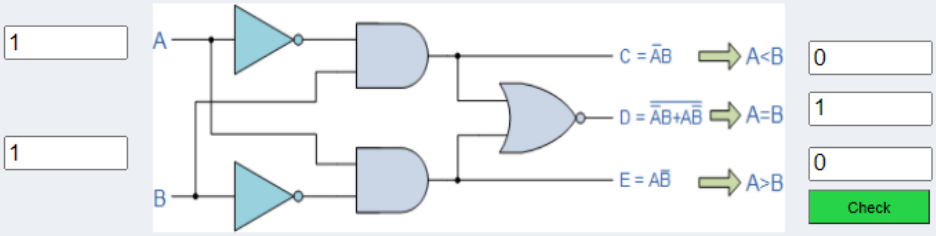

| Serial No. | A | B | A>B | A<B | A=B |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 2 | 1 | 0 | 1 | 0 | 0 |
| 3 | 1 | 1 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 | 1 | 0 |

# Simulation 2

Simulator 1 | Simulator 2

Instructions

**Verification of truth table of one bit Comparator**




**TRUTH TABLE** Print

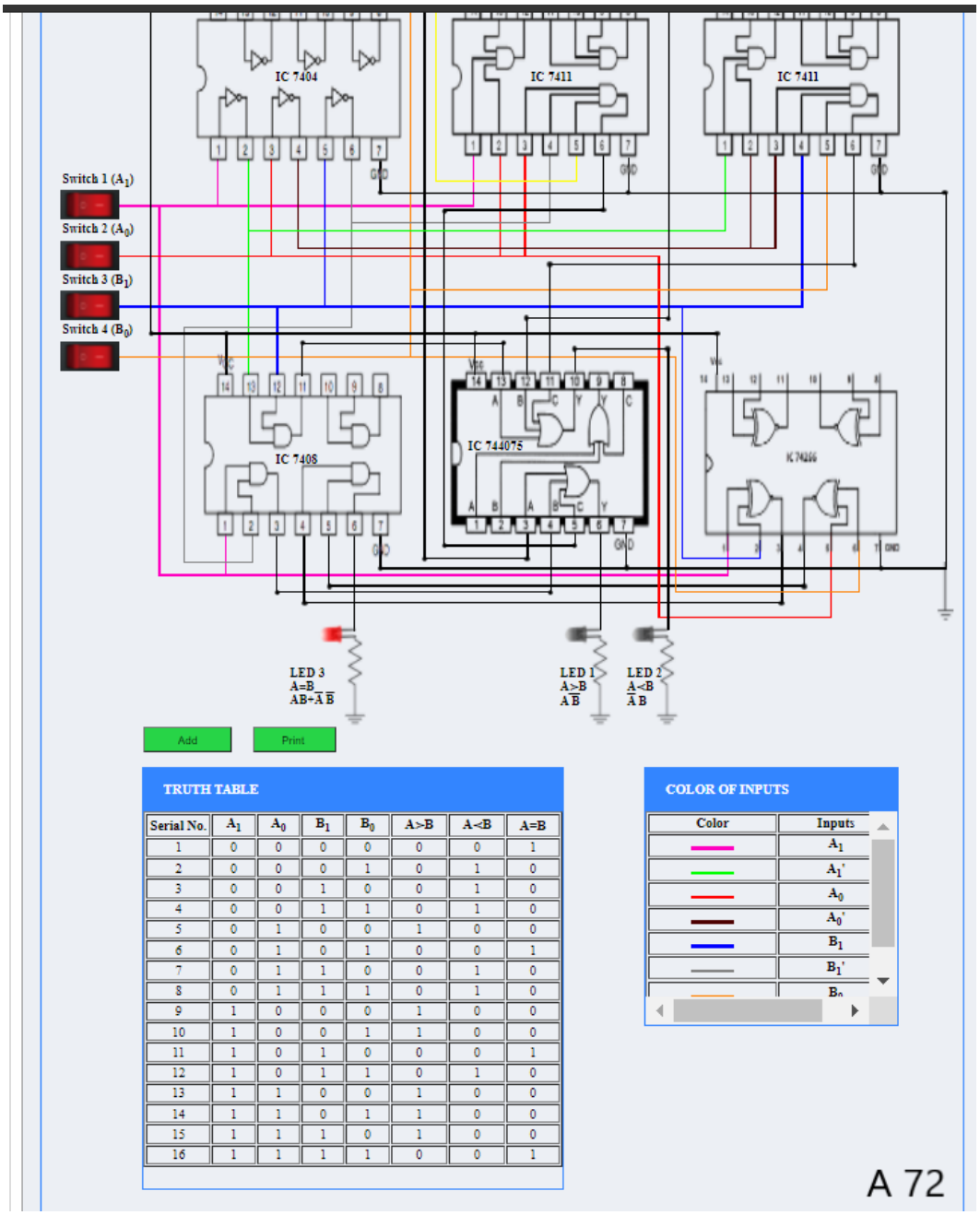
| Serial No. | A | B | A < B | A = B | A > B | Remarks |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 0 | Correct |
| 2 | 0 | 1 | 1 | 0 | 0 | Correct |
| 3 | 1 | 0 | 0 | 0 | 1 | Correct |
| 4 | 1 | 1 | 0 | 1 | 0 | Correct |

Reset

A 72

## 2) 2 Bit comparator

### Simulation 1




| Serial No. | $A_1$ | $A_0$ | $B_1$ | $B_0$ | A>B | A<B | A=B |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 5 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 6 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 7 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 8 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 9 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 10 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 11 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 12 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 13 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 14 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 15 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 16 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |

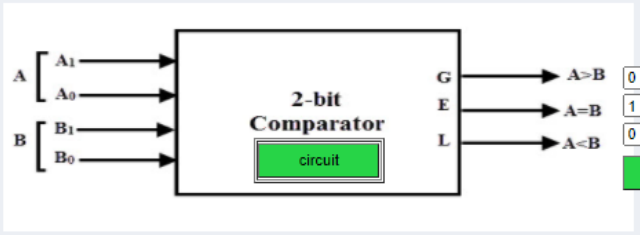
| Color | Inputs |
|---|---|
| | $A_1$ |
| | $A_1'$ |
| | $A_0$ |
| | $A_0'$ |
| | $B_1$ |
| | $B_1'$ |

# Simulation 2

Simulator 1 | Simulator 2

Instructions

**Verification of truth table of Two bit Comparator**




**TRUTH TABLE** Print

| Serial No. | A | B | A < B | A = B | A > B | Remarks |
|---|---|---|---|---|---|---|
| 1 | 00 | 00 | 0 | 1 | 0 | Correct |
| 2 | 00 | 01 | 0 | 0 | 1 | Correct |
| 3 | 00 | 10 | 0 | 0 | 1 | Correct |
| 4 | 00 | 11 | 0 | 0 | 1 | Correct |
| 5 | 01 | 00 | 1 | 0 | 0 | Correct |
| 6 | 01 | 01 | 0 | 1 | 0 | Correct |
| 7 | 01 | 10 | 0 | 0 | 1 | Correct |
| 8 | 01 | 11 | 0 | 0 | 1 | Correct |

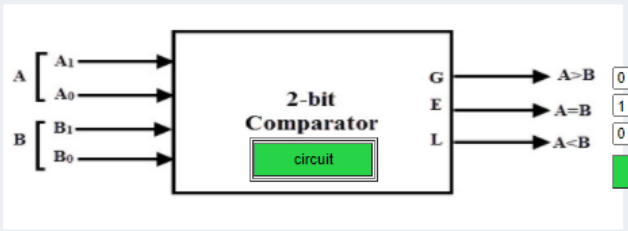
Reset

A 72

Simulator 1 | Simulator 2

Instructions

## Verification of truth table of Two bit Comparator




TRUTH TABLE

Print

| | | | | | | |
|---|---|---|---|---|---|---|
| 8 | 01 | 11 | 0 | 0 | 1 | Correct |
| 9 | 10 | 00 | 1 | 0 | 0 | Correct |
| 10 | 10 | 01 | 1 | 0 | 0 | Correct |
| 11 | 10 | 10 | 0 | 1 | 0 | Correct |
| 12 | 10 | 11 | 0 | 0 | 1 | Correct |
| 13 | 11 | 00 | 1 | 0 | 0 | Correct |
| 14 | 11 | 01 | 1 | 0 | 0 | Correct |
| 15 | 11 | 10 | 1 | 0 | 0 | Correct |
| 16 | 11 | 11 | 0 | 1 | 0 | Correct |

Reset