Experiment No.1

Aim: To study and verify the Truth Tables of AND, OR, NOT, NAND, NOR, EXOR logic gates

Components: IC 7400, 7402, 7404,7408,7432,7486
Apparatus: Prototyping board (breadboard), DC Power Supply, Connecting Wires

Theory:

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

AND GATE:-

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

The logic symbol & truth table of two input AND gate are shown in figure 1.a & 1.b respectively. The symbol for AND operation is ‘.’.

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

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

Logic symbol:

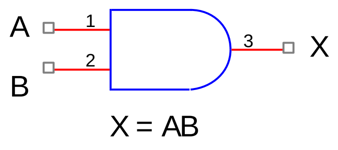


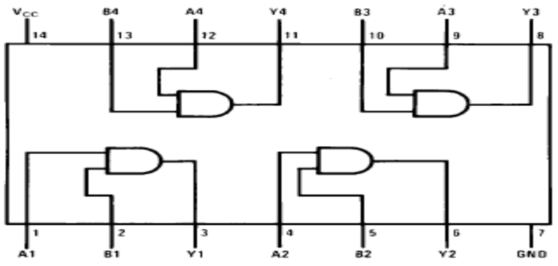
Fig.1.a

Truth table:

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

Fig.1.b

Pin diagram of IC**74LS08**:



OR GATE

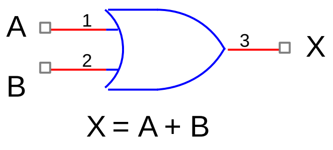
Like an AND gate, an OR gate may have two or more inputs but only one output. The output assumes the logic 1 state, even if one of its inputs is in logic 1 state. Its output assumes logic 0 state, only when each one of its inputs is in logic 0 state. OR gate is also called an 'any or all' gate. It can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

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

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

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

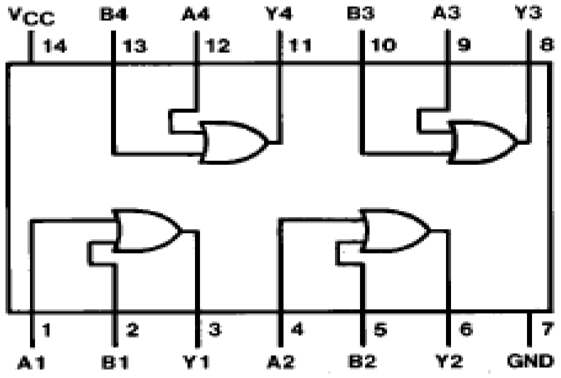
Logic symbol: Truth table:



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

Fig.1.c Fig.1.d

Pin diagram of IC**74LS32**:



NOT GATE

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

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

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

$$\mathbf{X = \bar{A}}$$

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

Logic symbol:

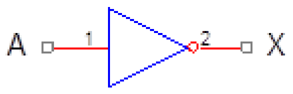


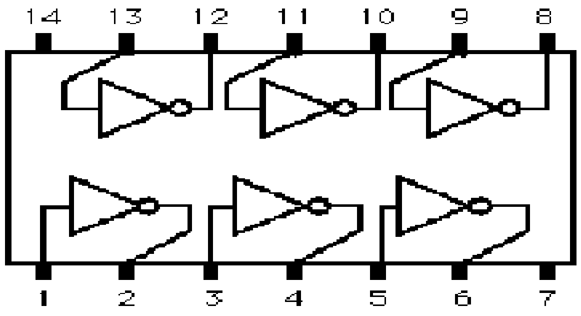
Fig.1.e

Truth table:

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

Fig.1.f

Pin diagram for IC**74LS04**:



NAND GATE

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

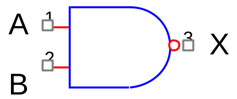
The logic symbol & truth table of two input NAND gate are shown in figure 1.g & 1.h respectively.

With input variables A & B the Boolean expression for

output can be written as;

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

Logic symbol:



$X = \overline{AB}$

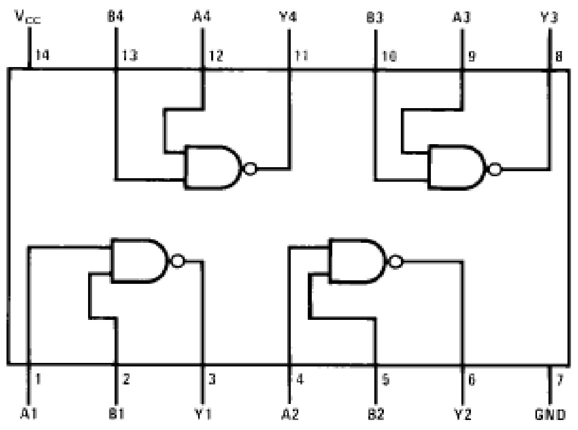
Fig.1.g

Truth table:

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

Fig.1.h

Pin diagram for IC**74LS00**:

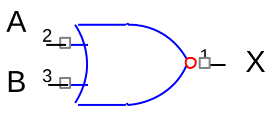


NOR GATE

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

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

Logic symbol:



$X = \overline{A + B}$

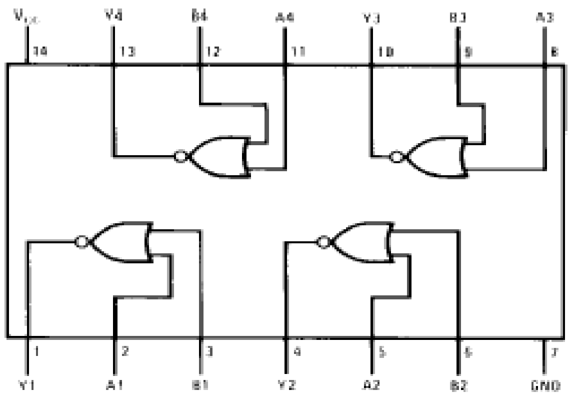
Fig.1.i

Truth table:

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

Fig.1.j

Pin diagram for IC**74LS02**:

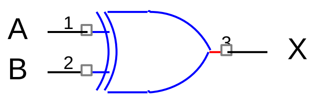


EXCLUSIVE-OR (X-OR) GATE

An X-OR gate is a two input, one output logic circuit, whose output assumes a logic 1 state when one and only one of its two inputs assumes a logic 1 state. Under the condition when both the inputs are same either 0 or 1, the output assumes a logic 0 state. Since an X-OR gate produces an output 1 only when the inputs are not equal, it is called as an anti-coincidence gate or inequality detector. The output of an X-OR gate is the modulo sum of its two inputs. The logic symbol & truth table of two input X-OR gate are shown in figure 1.k & 1.l respectively. The symbol for X-OR operation is '$\oplus$'. With input variables A & B the Boolean expression for output can be written as;

$$X = A \oplus B$$

Logic symbol:



$X = A \oplus B$

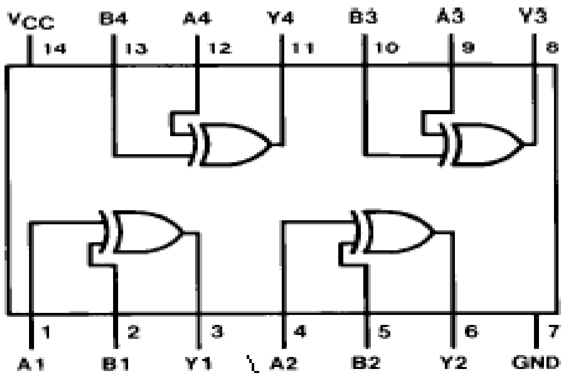
Fig.1.k

Truth table:

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

Fig.1.l

Pin diagram for IC**74LS86**:



Procedure:-

- Set up the breadboard.
- Test all required ICs.
- Give various combinations of inputs and check the output.
- Repeat the procedure for each IC.

Observation Table: LED ON (RED light): Logic 1
LED OFF (Green Light): Logic 0
Input variables: A, B
Output variable: Y

Sr. No	Input(A) LED	Input(B) LED	Output (OR) $Y = \overline{A}$	Output (AND) Y=AB	Output (OR) Y=A+B	Output (NAND) $Y = \overline{AB}$	Output (NOR) $Y= \overline{A+B}$	Output (XOR) $Y=A\oplus B$
1								
2								
3								
4								

Results and Analysis:
NOT Gate: When logic 1 is applied to one of NOT gate of 7404 IC, then output becomes zero. When input LED is ON (RED), the output LED become OFF (Green) vice versa.

OR Gate: The output of an OR gate is a 1 if one or the other or both of the inputs are 1, but a 0 if both inputs are 0. When One or the other or Both of the input LEDS are ON (RED Light), then output LED is ON(RED) otherwise Output LED is OFF(Green Light)

AND Gate: The output of an AND gate is only 1 if both its inputs are 1. For all other possible inputs the output is 0.When both the LEDS are On, then output LED is ON (RED Light) otherwise Output LED is OFF.

NOR Gate: The output of the NOR gate is a 1 if both inputs are 0 but a 0 if one or the other or both the inputs are 1.
NAND Gate: The output of the NAND gate is a 0 if both inputs are 1 but a 1 if one or the other or both the inputs are 0.

EXOR gate: The output of the XOR gate is a 1 if either but not both inputs are 1 and a 0 if the inputs are both 0 and both 1.

Conclusion: Any Boolean expression can be realized using NOT, AND, OR, NAND,NOR, EXOR gates.

Experiment No.2

Aim: - realization of logic gates using NAND and NOR gates.

Components: - IC 7400, 7402

Apparatus: - Digital trainer kit, wires, probes, etc.

Theory:-

Universal gates: -

The Nand and Nor gates are called as universal gates, because it is possible to implement any Boolean expression with the help of only Nand or only Nor gates.

Hence a user can build any combinational circuit with the help of only Nand gates or only Nor gates.

The NAND & NOR gates are called as ‘Universal gates’. Because it is possible to implement any Boolean expression with the help of only NAND or only NOR gate. We can construct AND, OR, NOT, X-OR & X-NOR gates.

The Boolean expression for NAND gate is,

$$X = \overline{AB}$$

The Boolean expression for NOR gate is,

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

This is a great advantage because a user will have to make a stock of only Nand or Nor gates.

All gates using Nand Gate:-

1) Not using Nand:-

The Boolean expression for NOT gate is $A = \overline{A}$ Fig. shows the realization of a NOT gate using a two i/p NAND gate. As both i/p’s are connected together we can write i/p A=B=A

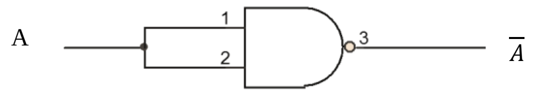
So o/p is given as

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

$$Y = \overline{A.A} \qquad \because A = B$$

But A.A=A $\quad \because$ by AND law

$$Y = \overline{A}$$



2) AND using NAND:-

The Boolean expression for an AND gate is Y=A.B

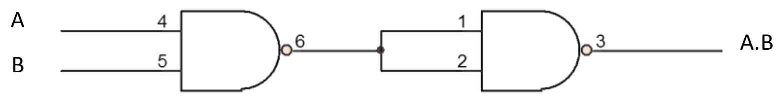
Taking double inversion,

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

$$\text{But } \overline{\overline{A}} = A$$

Y=A.B

This equation can be realized using only NAND gate as shown in fig.



3) OR using NAND:-

The Boolean expression for an OR gate is Y=A + B

Taking double inversion,

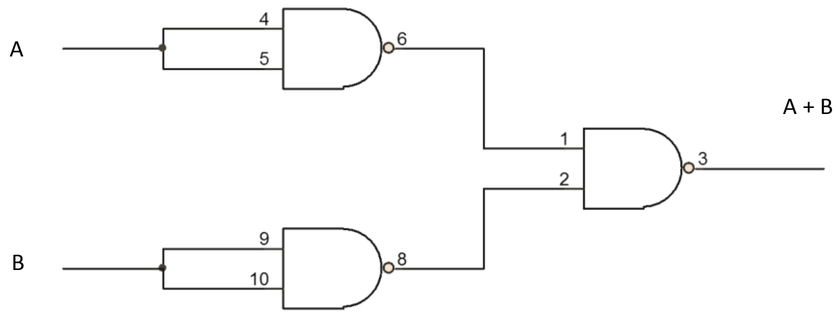
$$Y = \overline{\overline{A + B}}$$

But by DE-Morgan's theorem

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

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

This is required expression for OR gate



4) NOR using NAND:-

The Boolean expression for an NOR gate is $Y = \overline{A + B}$

But by DE-Morgan's theorem

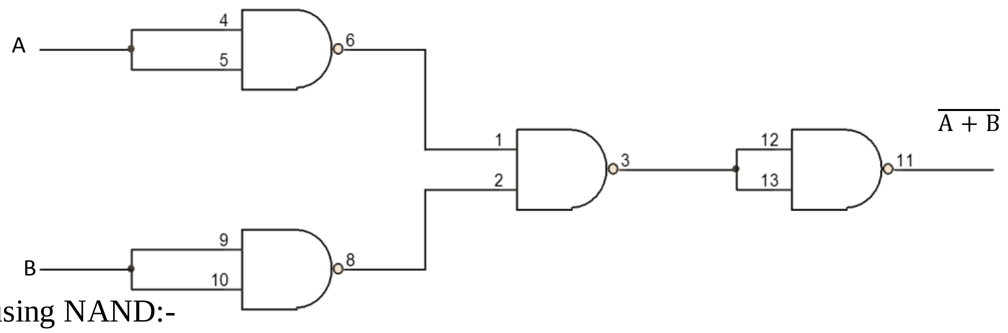
$$\overline{A + B} = \overline{A}.\overline{B}$$

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

Taking double inversion,

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

This is required expression for NOR gate



5) Ex-OR using NAND:-

The expression for Ex- OR gate is

$$Y = A \oplus B$$

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

Taking double inversion

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

Let $\overline{A}\,.\,B$ =X and $A.\overline{B}$ =Z

$$Y=\overline{\overline{X + Z}}$$

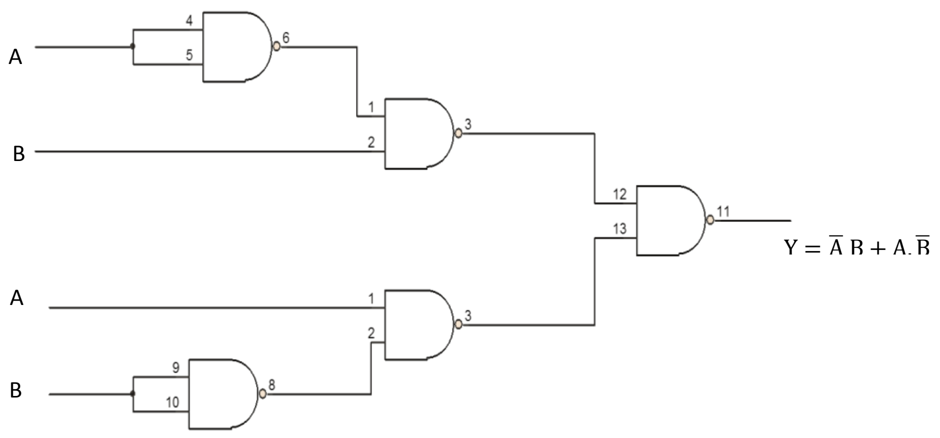
$$\text{using De} - \text{Morgan's theorem}$$

$$\overline{X + Z} = \overline{X}.\overline{Z}$$

$$Y = \overline{\overline{X}\,.\,\overline{Z}}$$

$$Y = \overline{\overline{(\overline{A}B)}\,.\,\overline{(A.\overline{B})}}$$

This is required expression for Ex-OR gate using NAND gate.



All gates using NOR Gate:-

1) Not using NOR:-

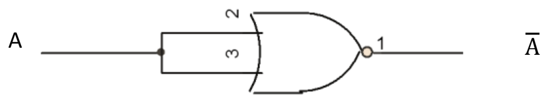
The Boolean expression for NOT gate is $A = \bar{A}$ Fig. shows the realization of a NOT gate using a two i/p NOR gate. As both i/p's are connected together we can write i/p A=B=A So o/p is given as

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

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

But A+A=A $\qquad \because$ by OR law

$$Y = \bar{A}$$



2) OR using NOR:-

The Boolean expression for an OR gate is Y=A+B

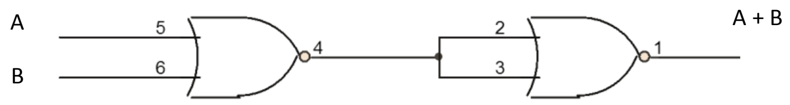
Taking double inversion,

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

But $\bar{\bar{A}} = A$

Y=A+B

This equation can be realized using only NAND gate as shown in fig.



3) AND using NOR:-

The Boolean expression for an AND gate is Y=A.B

Taking double inversion,

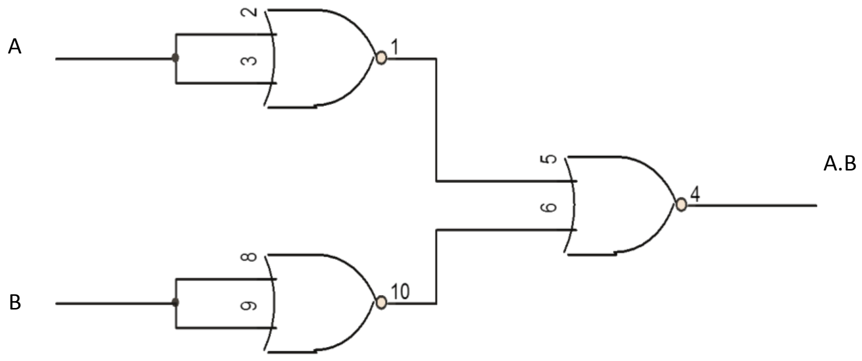
$$Y = \overline{\overline{A.B}}$$

But by DE-Morgan's theorem

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

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

This is required expression for AND gate



4) NAND using NOR:-

The Boolean expression for an NOR gate is $Y = \overline{A\,.\,B}$

But by DE-Morgan's theorem

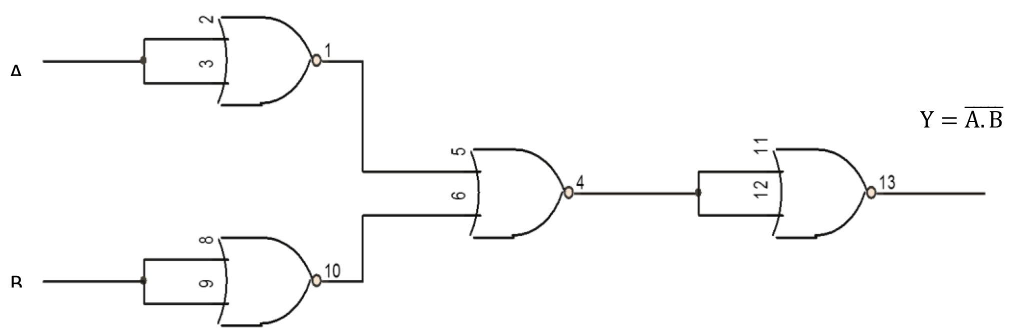
$$\overline{A.B} = \overline{A} + \overline{B}$$

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

Taking double inversion,

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

This is required expression for NAND gate



5)EX-OR using NOR:-

The expression for EX- OR gate is

$$Y = A \oplus B$$

$$Y = \bar{A} . B + A.\bar{B}$$

Taking double inversion

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

Let $\bar{A} . B$ =X and $A.\bar{B}$ =Z

$$Y=\overline{\overline{X + Z}}$$

$$\text{using De} - \text{Morgan's theorem}$$

$$\overline{X + Z} = \bar{X}.\bar{Z}$$

$$Y = \overline{\bar{X} . \bar{Z}}$$

$$Y = \overline{\overline{(\bar{A}.B)}.\overline{(A.\bar{B})}}$$

But $\overline{\bar{A}.B} = A + \bar{B}$ and $A.\overline{\bar{B}} = \bar{A} + B$

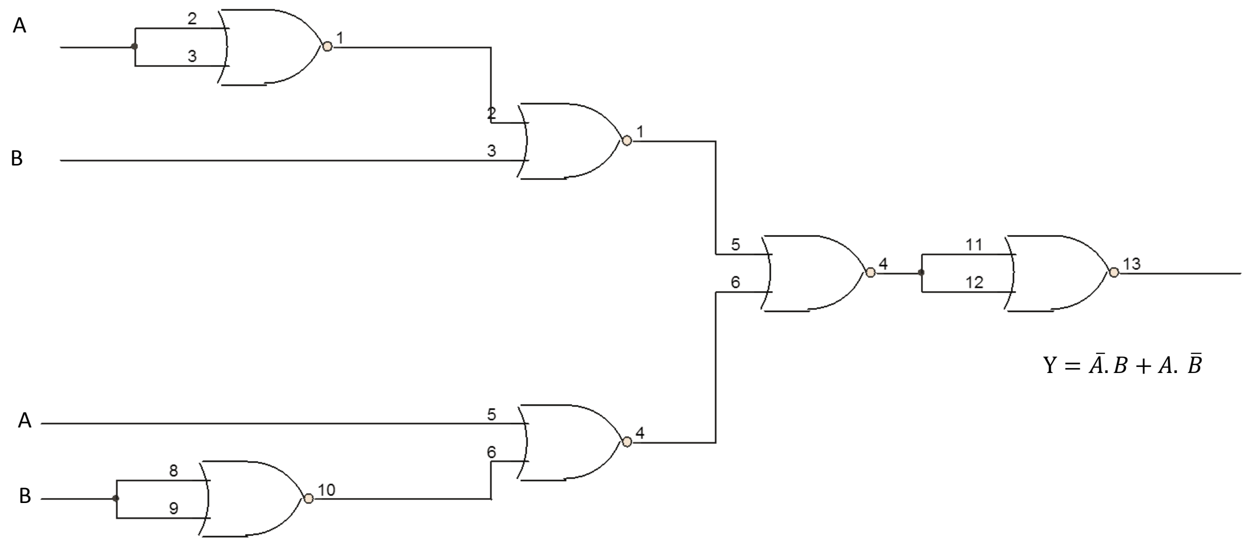
$$Y = \overline{(A + \bar{B}).(\bar{A} + B)}$$

$$Y = \overline{(A + \bar{B})} + \overline{(\bar{A} + B)}$$

Taking double inversion, we get

$$Y = \overline{\overline{\overline{(A + \bar{B})} + \overline{(\bar{A} + B)}}}$$

This is required expression for EX-OR gate using NOR gate.



$Y = \bar{A}.B + A.\bar{B}$

Conclusion:-

All the gates are realized using NAND and NOR gates and truth tables are verified.

Experiment No.3

Aim: Design and Implementation of 3 Bit Binary To Gray Code Convertor

Components: - IC 7486

Apparatus: Digital trainer kit, wires, probes, etc.

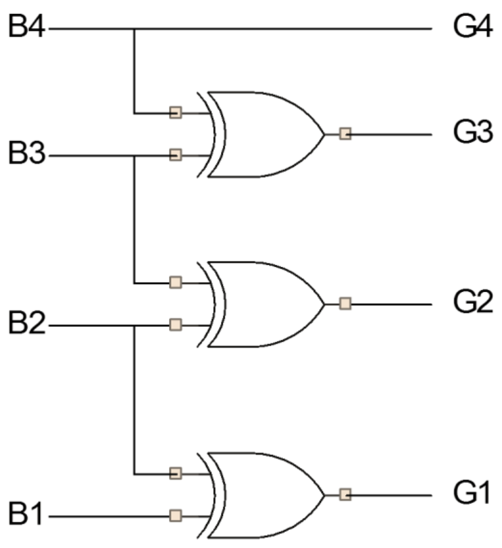
Theory:

Gray code is a non-weighted code. It is not an arithmetic code. It has a very special feature that only one bit in the gray code will change, each time the decimal number is incremented. As only one bit changes at a time, the gray code is called 'unit distance code'. The gray code is cyclic code is also exhibits the reflective property.

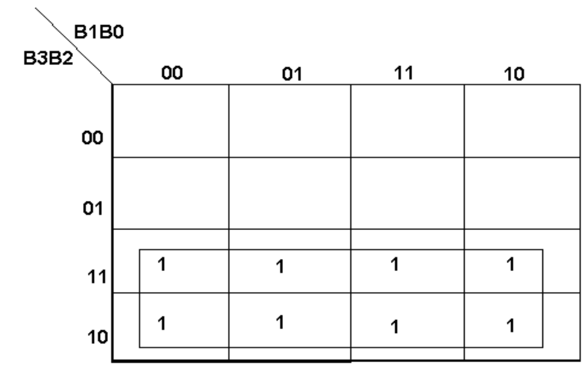
Steps for converting Binary to Gray code

1) Record the MSB bit as it is.
2) Add this bit to the next bit, recording the sum and neglecting the carry bit(EX-OR operation).
3) Record Successive sum until completion of all bits.

Logic Diagram: Binary to Gray Code Convertor

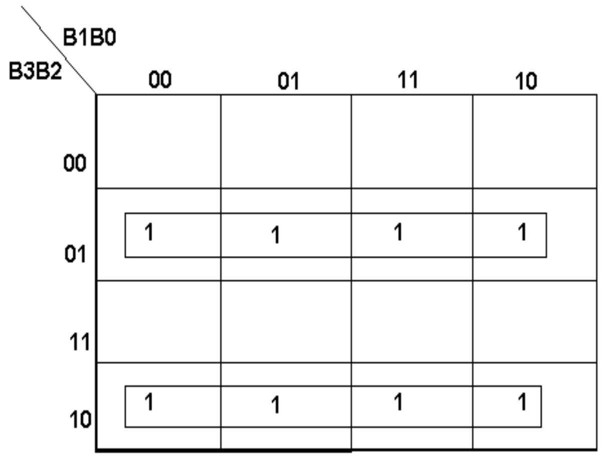


K-Map for G3:



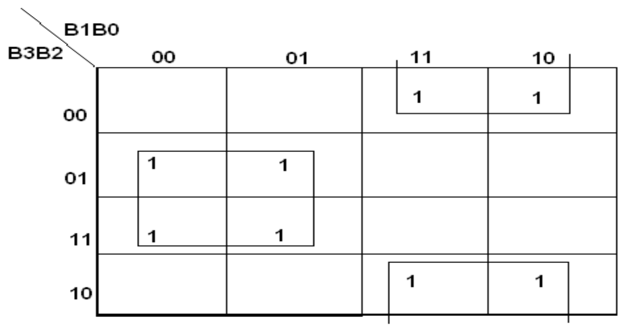
G3 = B3

K-Map for G_2:



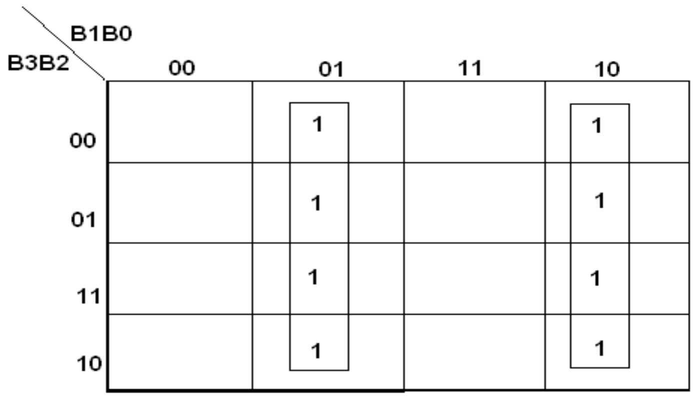
$$G2 = B3 \oplus B2$$

K-Map for G_1:



$$G2 = B2 \oplus B1$$

K-Map for G_0:



$$G2 = B1 \oplus B0$$

TRUTH TABLE:

| Binary input | Gray code output |

B3	B2	B1	B0	G3	G2	G1	G0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0

1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Conclusion:-

Thus result for Binary-to-Gray conversion is verified.

EXPERIMENT NO. 4

Aim: - to study adder/Subtractor using IC 7483.

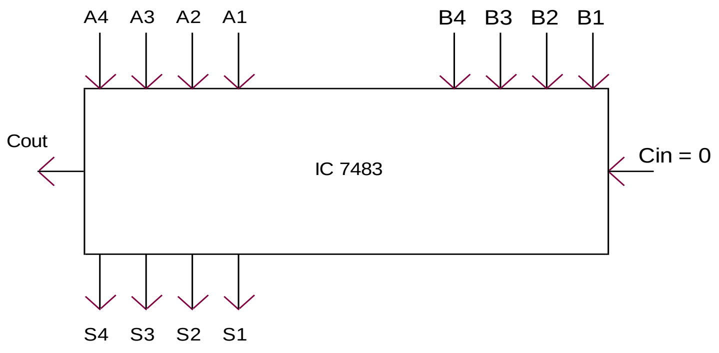
Components: - IC 7483

Apparatus: Digital trainer kit, wires, probes, etc.

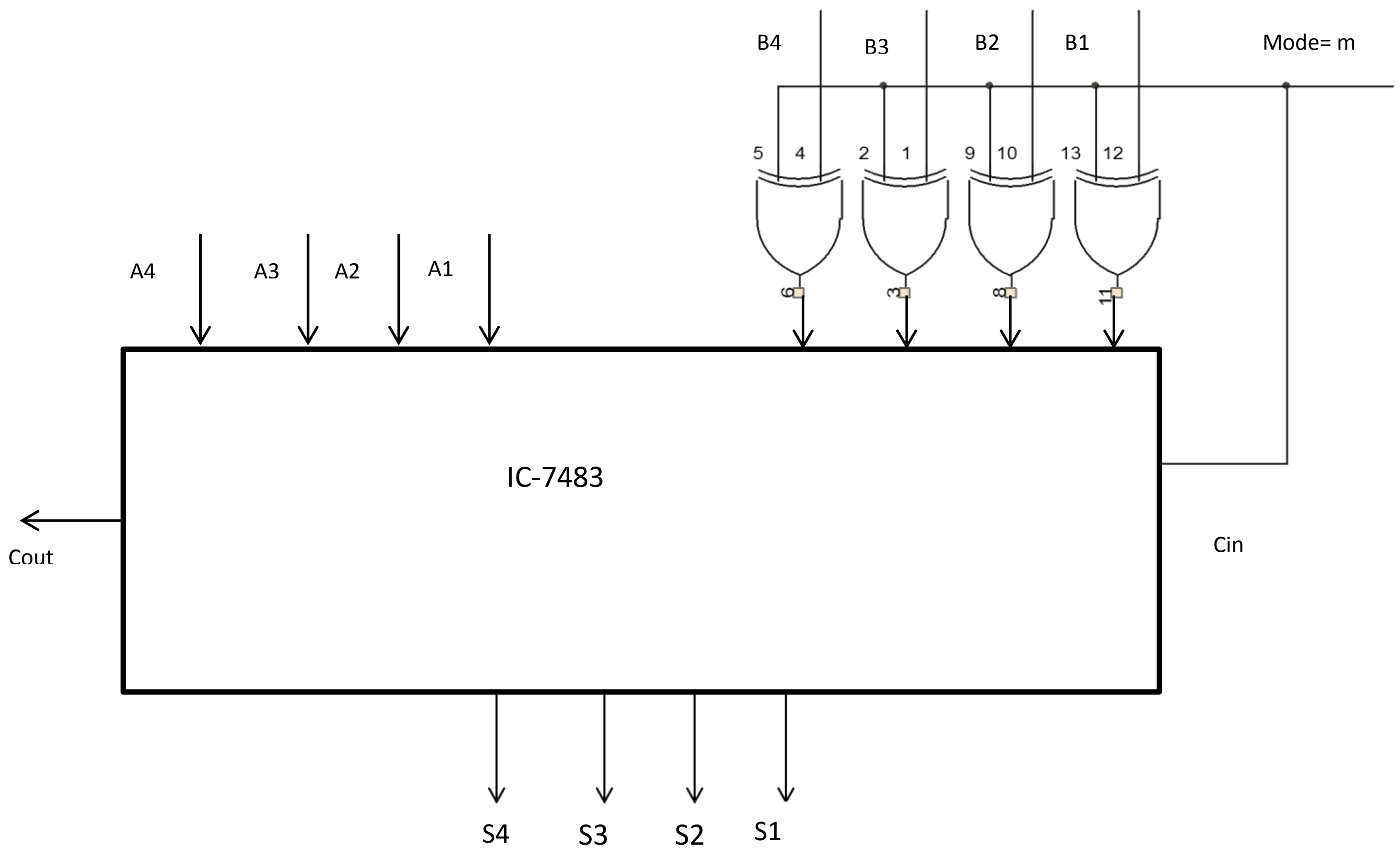
Theory:-

Binary adder/ Subtractor

The addition/ subtraction operations are combined into one circuit with one common mode signal. This is done by including EX-OR gate with each full adder. The mode i/p m controls the operation of either addition or subtraction. When m=0 the circuit can be used as adder and when m=1 then circuit can be used as Subtractor. Each EX-OR gate receives i/p m and one of the i/p B. When m=0 we have $B \oplus 0 = B$. The full adder receives the value of B, the i/p carry is 0 and the circuit performs $A + B$. When m=1, we have $B \oplus 1 = \overline{B}$ and $C_1 = 1$. The B i/p's are complemented and 1 is added through the i/p carry. The circuit performs the operation A plus 2's complement of B.



Block diagram of 4-bit binary adder using IC 7483


B4
B3
B2
B1
Mode= m
5
4
2
1
9
10
13
12
6
3
8
11
A4
A3
A2
A1
IC-7483
Cout
Cin
S4
S3
S2
S1

EXPERIMENT NO. 5

Aim:-to design 2-bit synchronous up-down counter using J-K Fli-flop.

Components:- IC 7473,7486

Theory:-

Counter:-

The digital circuit used for counting pulses is known as counter. It is a sequential circuit. Counter is the widest application of flip-flops.it is a group of flip-flops with a clock signal applied.

Counter count the number of clock pulses. Hence with some modifications it can be used for measuring frequency or time period.

Types of counters:-

1) Asynchronous or ripple counters:-
   For these counters the external clock is applied to one of the flip-flop and then the output of preceding flip-flop is connected to the clock of next flip-flop.
2) Synchronous counter:-
   In synchronous counters all flip-flops receive the external clock pulses simultaneously. Ring counter and Johnson counter are the example of synchronous counter.

J-K Flip-flop:-

The J-K Flip-flop is very versatile and also the most widely used. The J and K are the synchronous control i/p.

The functioning of J-K Flip-flop is identical to that of S-R Flip-flop, except that it has no invalid state like that of S-R Flip-flop.

Case 1) when J=0 and K=0

When both the i/p's are 0, and the clock pulse is present the o/p of the flip-flop is same as that of previous value.

Case 2) when J=0 and K=1

When J=0 and K=1 then the o/p of flip-flop resettled i.e. Q=0. This is called as reset state of flip-flop.

Case 3) when J=1 and K=0

When J=1 and K=0 then the o/p of flip-flop is set i.e. Q=1. This is called as set state of flip-flop.

Case 4) when J=1 and K=1

When J=1 and K=1 then the o/p of flip-flop is toggle i.e. complement of previous state o/p. This is called as toggle state of flip-flop.

Modulus of counter:-

Number of states through which counter passes before returning to the starting state is called modulus of counter. It is equal to total number of distinct states i.e. count including 0.

Procedure:-

- Test IC 7473 & IC 7486.
- Mount the circuit as per logic diagram.
- Make mode signal M high. Give manual clock pulses and observe the output.
- Make mode signal M low. Give manual clock pulses and observe the output. .

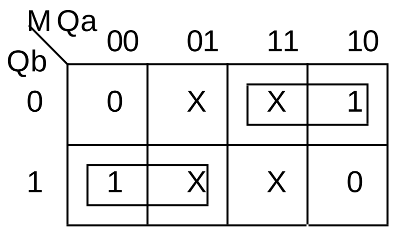
Conclusion:-

The 2-bit UP-DOWN counter is implemented using IC 7483, 7486 .When mode signal is low, it acts as up counter and acts as down counter when mode signal is high..

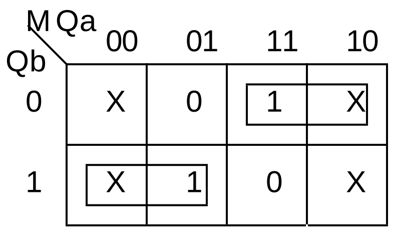
Truth table:-

M	**Qa**	**Qb**	**Qa+**	**Qb+**	**Ja**	**Ka**	**Jb**	**Kb**
0	0	0	0	1	0	X	1	X
0	0	1	1	0	1	X	X	1
0	1	0	1	1	X	0	1	X
0	1	1	0	0	X	1	X	1
1	0	0	1	1	1	X	1	X
1	0	1	0	0	0	X	X	1
1	1	0	0	1	X	1	1	X
1	1	1	1	0	X	0	X	1

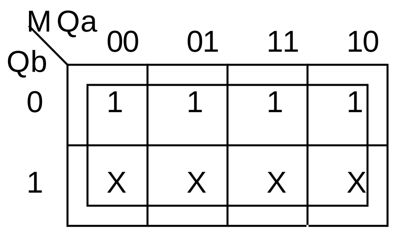
K-Map:-



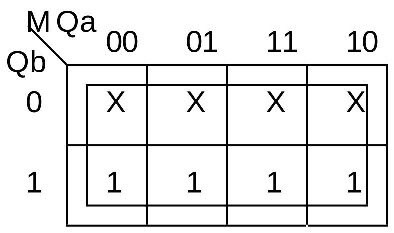
Ja = M ⊕ Qb



Ka = M ⊕ Qb

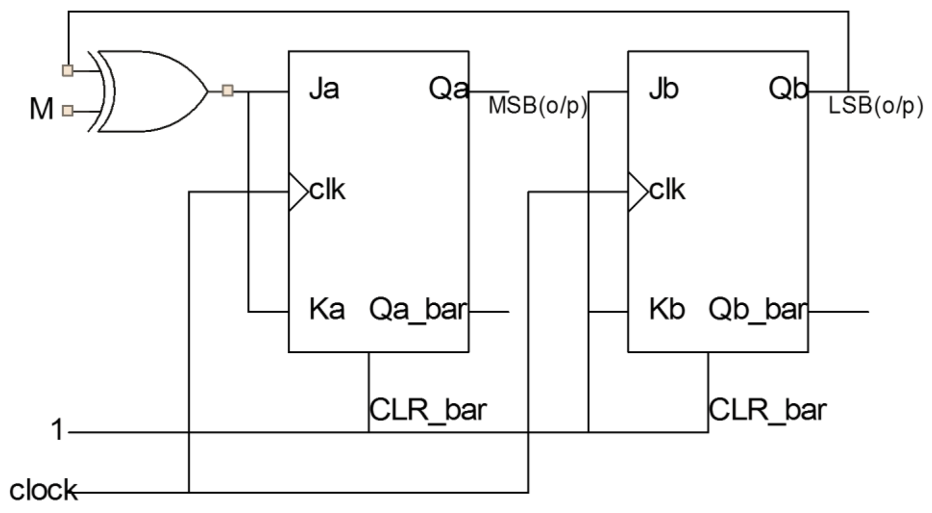


Jb = 1



Kb = 1

Logic Diagram:



EXPERIMENT NO. 6

Aim: To design 1 bit ALU using VHDL.

Software:- Xilinx

Theory:

Modeling Styles:-

VHDL provides a broad set of constructs (statements). VHDL language constructs are divided into different modeling styles by their level of abstraction: - Behavioral, Dataflow, Structural

1) Behavioral style:-

The highest level of abstraction supported in VHDL is called the behavioral level of abstraction.

Describing a circuit at the behavioral level is very similar to writing a computer program. It has all standard high level programming language constructs (like C, BASIC), such as the FOR LOOP, WHILE LOOP, IF-ELSE and other variable assignments.

When creating a behavioral description of a circuit, you will describe your circuit in terms of its operation over time.

A behavioral design method defines a circuit in terms of a textual language rather then a schematic of interconnected symbols.

In behavioral modeling we must requires the behavior of design or simply truth table of design. No need of logical circuit diagram.

2) Data-flow modeling:-

The view of data as flowing through a design, from input to output.
An operation is defined in terms of a collection of data transformations, expressed as concurrent statements.
Each of the statements can be activated when any of its input signals changes its value.
While these statements describe the behavior of the circuit, a lot of information about its structure can be extracted from the description as well.
Data flow modeling which has a set of concurrent assignment statements.
In the data flow level of abstraction, you describe how information is passed in circuit.
The built in operations in VHDL are used in expression such as AND, OR, XOR, NOT, etc.

3) Structural style:-
   It is the view closest to hardware, a model where the components of a design are interconnected. This view is expressed by instantiations.
   It is set of interconnected components.
   It is structure explicit.
   It may hierarchical.
   There is need of logical diagram as design specifications.
4) Mixed style:-
   The mixed modeling style is any combination of behavior, data flow, and structural modeling in a single architecture body.
   In it we could use component instantiation statements, concurrent single assignment statements, and sequential single assignment statements.
   The most popular method to modeling large system is mixed style containing structural plus behavioral.
5) VHDL program format:-
   A stand-alone piece of VHDL is composed of at least 3 fundamental sections:
   a) LIBRARY declarations:-
      It contains a list of all libraries to be used in the design.
      A library is collection of commonly used pieces of code.
      The code is usually written in the form of functions, procedures or components which are placed inside packages and then (combined) compiled into the destination library.
      Syntax for declaration:-
      LIBRARY Library_name;
      Use library_name_package_name.package_parts;
   b) Entity:-
      A VHDL entity specifies the name of the entity, the ports of the entity and entity related information. All designs are created using one or more entities.
      Entity is the declaration of interface between a design and external environment.
      Syntax:-
      ```
      Entity entity_name;
      Port (port_name: mode port_type;
            port_name: mode port_type);
      end entity_name;
      ```
   c) Architecture:-
      The architecture describes underlying functionality or internal organization of the entity and contains the statements that model the behavior of entity.
      Architecture body is used to describe the behavior, dataflow or structure of a design entity.
      The syntax is
      Architecture behavior of entity_name is

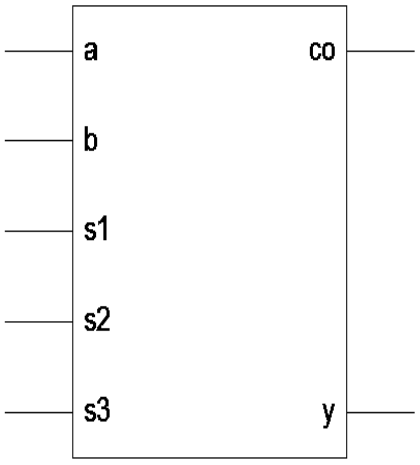
```
          { block _declarative_item}
      Begin
      {concurrent_statement}
      End architecture_name:
```

Program:-

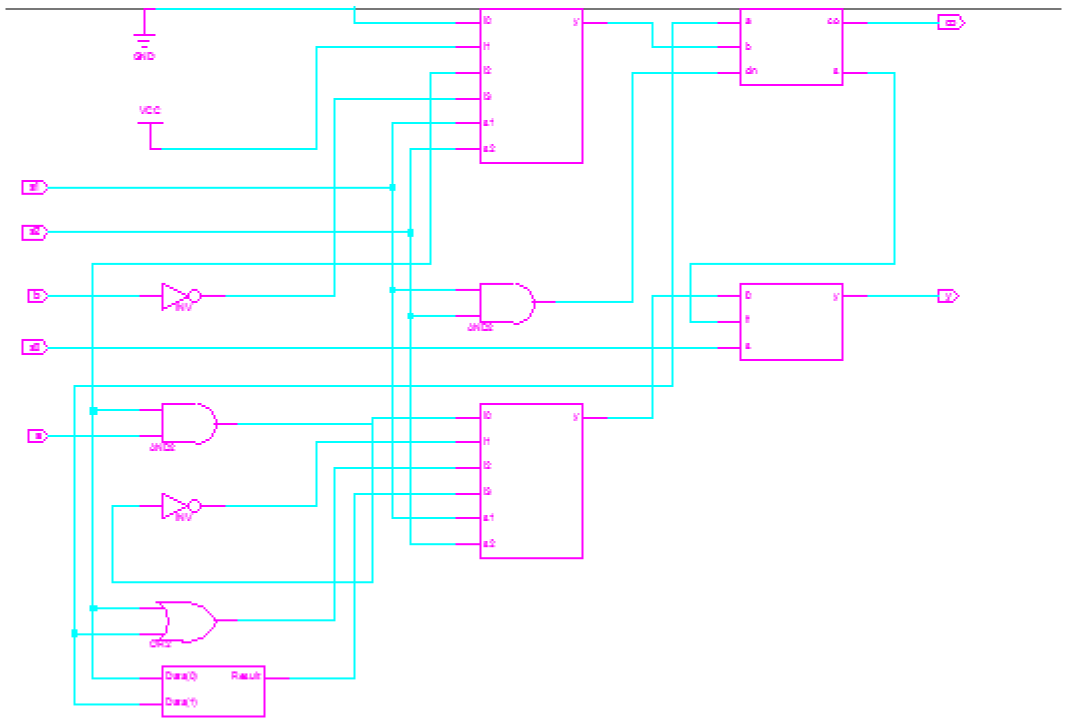
```
Library IEEE;

Use IEEE.STD_LOGIC_1164.ALL;

Use IEEE.STD_LOGIC_ARITH.ALL;

Use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity alu1bit is

   Port (a, b, s3, s2, s1: in STD_LOGIC;

         y, co : out  STD_LOGIC);

end alu1bit;

architecture Behavioral of alu1bit is

signal w1,w2,w3,w4,w5,w6,w7,w8,w9: std_logic;

Component fa is

   Port ( a, b, cin : in  STD_LOGIC;

         s, co : out  STD_LOGIC);

End component;

Component mux41 is

   Port (i0, i1, i2, i3, s2, s1: in STD_LOGIC;

         y: out  STD_LOGIC);

end component;

Component mux21 is

   Port (i0, i1, s: in STD_LOGIC;

         y: out  STD_LOGIC);
```

```
end component;

begin

w1<=not b;

w2<= a and b;

w3<= a nand b;

w4<= a or b;

w5<= a xor b;

w7<= s2 and s1;

u1: mux41 port map ('0','1',b,w1,s2,s1,w8);

u2: mux41 port map (w2, w3, w4, w5, s2, s1, w6);

u3: fa port map (a, w8, w7, w9, co);

u4: mux21 port map (w6, w9, s3,y);

end Behavioral;
```

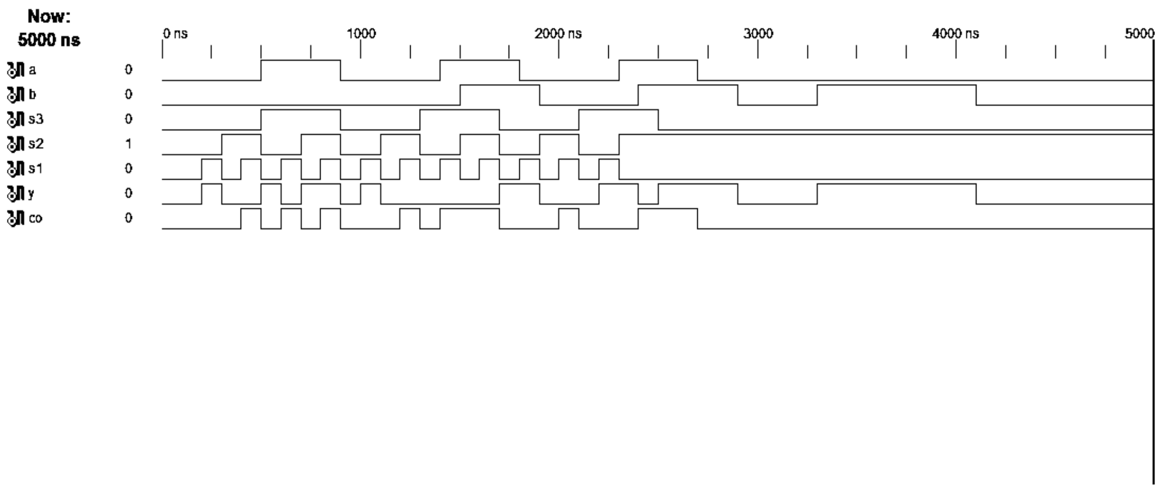
Diagram:-



Top entity



Test bench waveforms:-



Test bench waveform

Conclusion:-

Hence we have studied 1bit ALU using VHDL by Structural modeling style.

Experiment No. 7

Aim: - to design 2 bit up-down counter using VHDL.

Software: - Xilinx

Theory:-

The digital circuit used for counting pulses is known as counter. It is a sequential circuit.

Counter is the widest application of flip-flops.it is a group of flip-flops with a clock signal applied.

Counter count the number of clock pulses. Hence with some modifications it can be used for measuring frequency or time period.

Types of counters:-

3) Asynchronous or ripple counters:-
   For these counters the external clock is applied to one of the flip-flop and then the output of preceding flip-flop is connected to the clock of next flip-flop.
4) Synchronous counter:-
   In synchronous counters all the flip-flops receive the external clock pulse simultaneously.
   Depending on the way in which the counting progresses, counters are classified as:
   i) Up - counters: In this type, o/p goes on increasing as they receive clock pulse.
   ii) Down - counters: In this type, o/p goes on decreasing as they receive clock pulse.
   iii) Up – down counters:- it is the combination of up and down counter.

Modulus of counter:-

Number of states through which counter passes before returning to the starting state is called modulus of counter. It is equal to total number of distinct states i.e. count including 0.

Process statements in VHDL:-

A process statement contains a set of sequential statements. Although all processes in a design execute consequently, the sequential statements within each process one at a time.

A process communicates with the rest of the design by reading values from or writing them to signals or ports outside the process.

The process statement in VHDL is the primary means by which sequential operations can be described.

The process statement represents the behavior of some portion of the design. It consist of the sequential statements whose execution is made in order defined by user. Each process can be assigned an optional label.

A process statement consists of following items:-

- An optional process name
- The process keyword
- An optional sensitivity list, indicating which signals results in the process "executing": when there is some event detected.
- An optional declarations section, allowing local objects and subprograms to be executed.
- A begin keyword
- A sequence of statements to be executed when the program runs.
- An end statement.

The syntax is:-

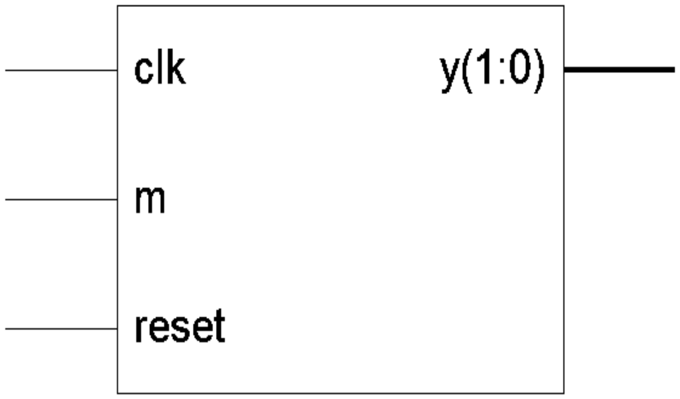
```
[Label:] process [(sensitivity_list)]

Begin

        {sequential_statement}

End process [label];
```

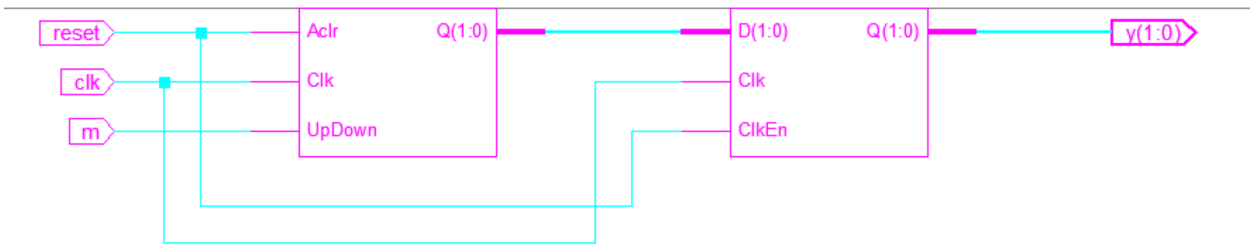
Program:-

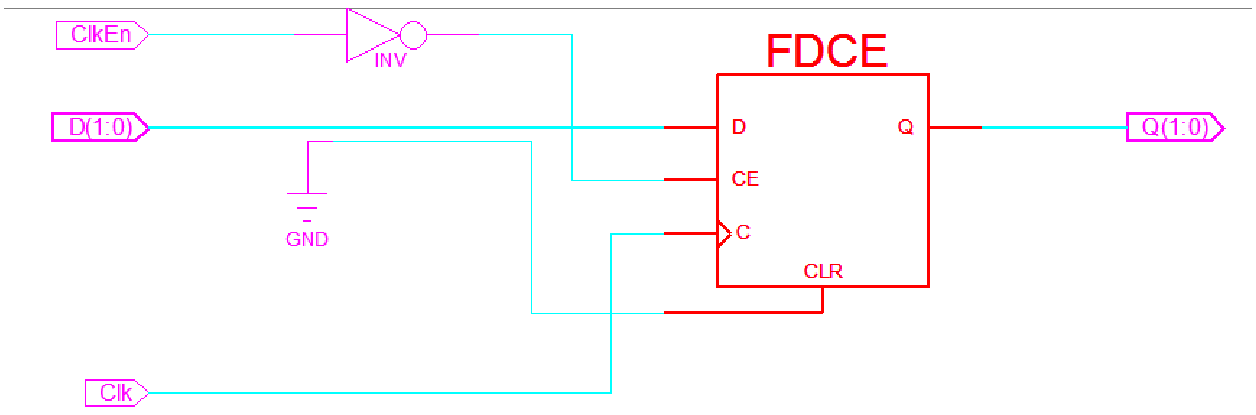
```
Library IEEE;

Use IEEE.STD_LOGIC_1164.ALL;

Use IEEE.STD_LOGIC_ARITH.ALL;

Use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity c1 is

   Port (clk, reset: in STD_LOGIC;

        y: out  STD_LOGIC_VECTOR (01 downto 0));

end c1;

Architecture Behavioral of c1 is

Signal temp:std_logic_vector(01 downto 0);
```

```
Begin

Process (reset, m, clk)

Begin

if (reset='1') then

Temp<="00";

else

if (clk'event and clk='1') then

temp<=temp;

if (m='1') then

temp<=temp+1;

else

temp<=temp-1;

end if;

y<=temp;

end if;

end if;

end process;

end Behavioral;
```

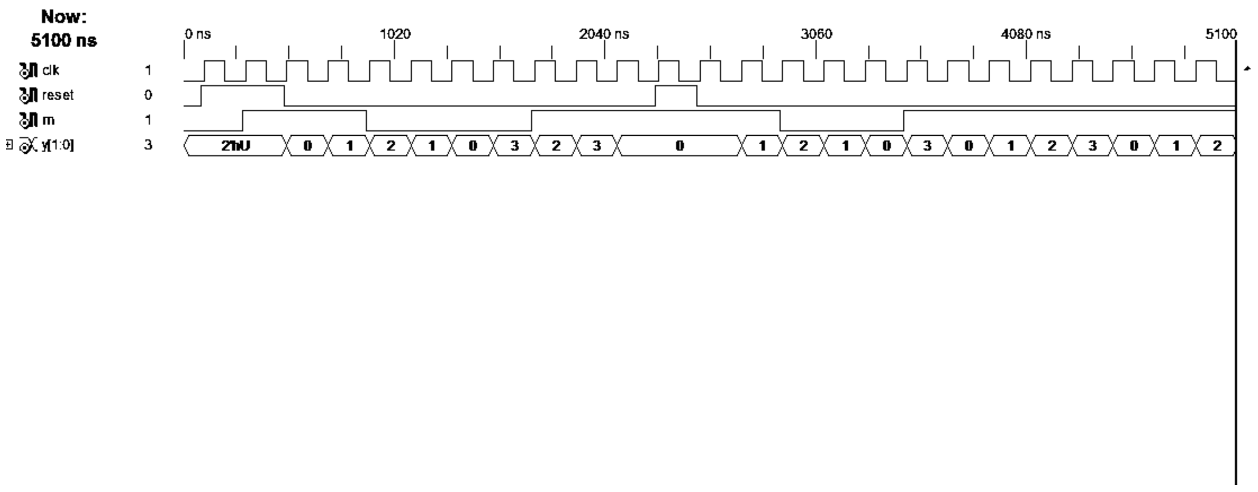
Diagram:







Test bench waveforms:-



Conclusion:-

Hence we have studied 2 bit up-down counter using VHDL.