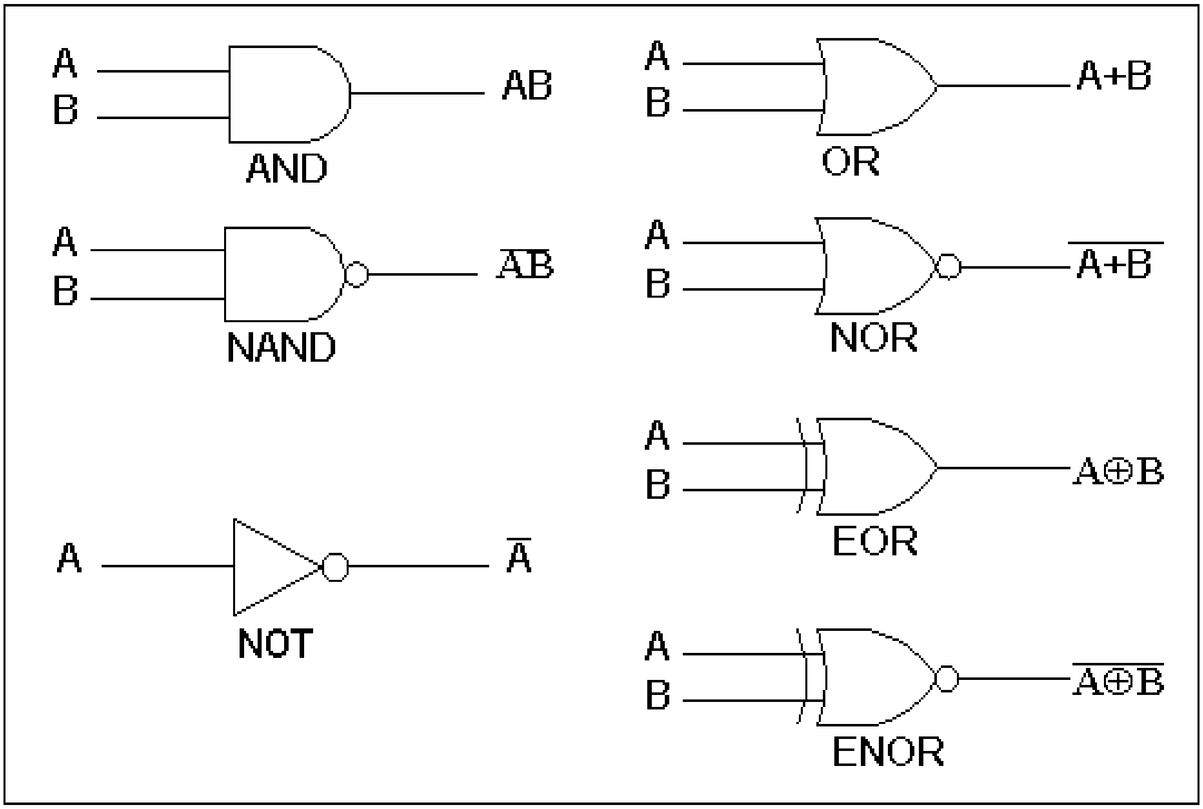
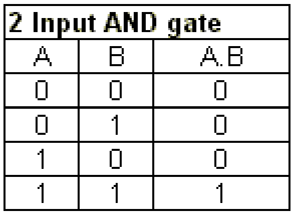
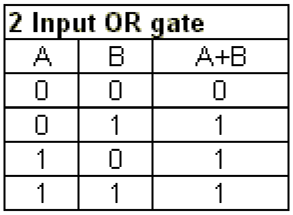
Logic Gate Symbols:



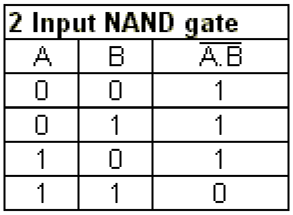
Truth Tables:



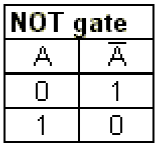
2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1



2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1



2 Input NAND gate		
A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0



NOT gate	
A	$\overline{A}$
0	1
1	0

EXPERIMENT NO.1

LOGIC GATES

AIM:
PROGRAMS:
TWO-INPUT LOGIC GATES

1) AND Gate:

```
library ieee;
use ieee.std_logic_1164.all;
entity and2 is
        port(a,b:in bit;
        c:out bit);
end and2;
architecture anddf of and2 is
begin
        c<=a and b;
end anddf;
```

2) OR Gate:

```
library ieee;
use ieee.std_logic_1164.all;
entity or2 is
        port(a,b:in bit;
        c:out bit);
end or2;
architecture ordf of or2 is
begin
        c<=a or b;
end ordf;
```

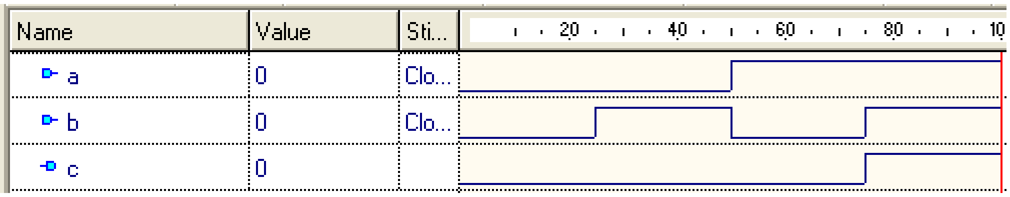
2 Input NOR gate		
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

2 Input EXOR gate		
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

2 Input EXNOR gate		
A	B	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

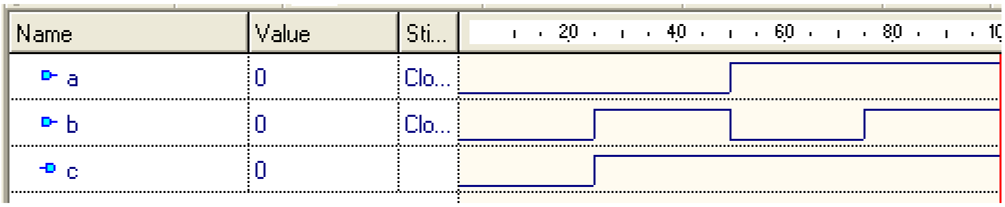
OUTPUT WAVEFORMS:

AND GATE:



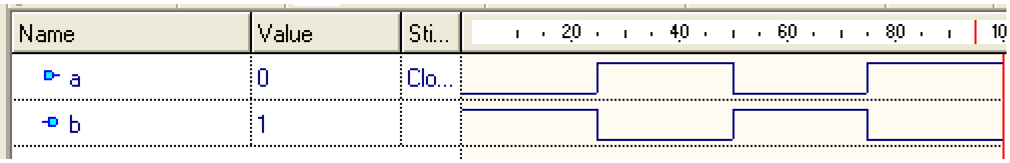
Name	Value	Sti...
a	0	Clo...
b	0	Clo...
c	0	

OR GATE:



Name	Value	Sti...
a	0	Clo...
b	0	Clo...
c	0	

NOT GATE:



Name	Value	Sti...
a	0	Clo...
b	1	

3) EX-0R Gate:

```
library ieee;
use ieee.std_logic_1164.all;
entity xor2 is
        port(a,b:in bit;
        c:out bit);
end xor2;
architecture xordf of xor2 is
begin
        c<=a xor b;
end xordf;
```

4) XNOR Gate:

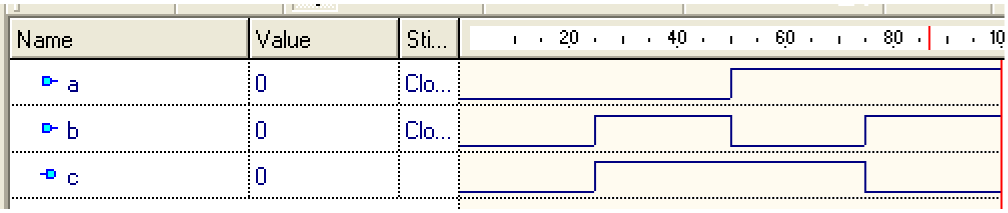
```
library ieee;
use ieee.std_logic_1164.all;
entity xnor2 is
        port(a,b:in bit;
        c:out bit);
end xnor2;
architecture xnordf of xnor2 is
begin
        c<=a xnor b;
end xnordf;
```

(B) ONE INPUT LOGIC GATE

NOT Gate:

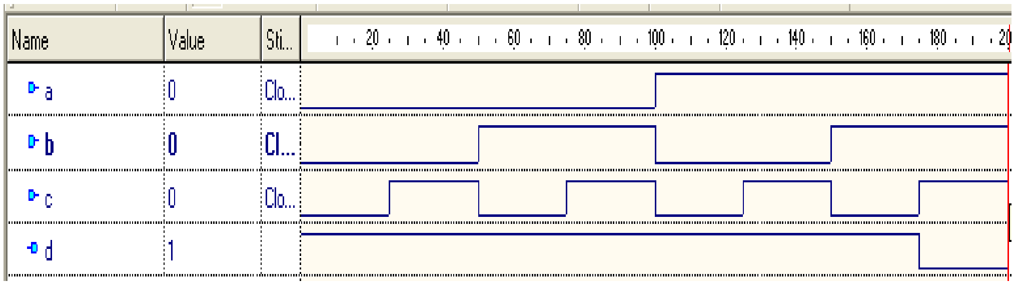
```
library ieee;
use ieee.std_logic_1164.all;
entity not1 is
        port(a:in bit;
        b:out bit);
end not1;
architecture notdf of not1 is
begin
        b<=not a;
end notdf;
```

EX-OR GATE:



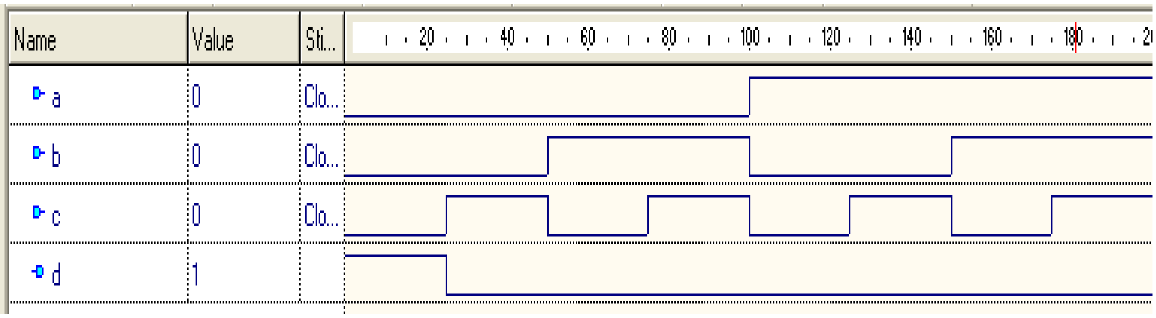
Name	Value	Sti...
a	0	Clo...
b	0	Clo...
c	0	

NAND GATE:



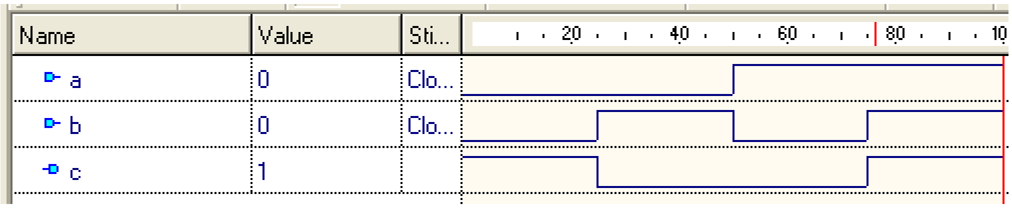
Name	Value	Sti...
a	0	Clo...
b	0	Cl...
c	0	Clo...
d	1	

NOR GATE:



Name	Value	Sti...
a	0	Clo...
b	0	Clo...
c	0	Clo...
d	1	

XNOR GATE:



Name	Value	Sti...
a	0	Clo...
b	0	Clo...
c	1	

(C) THREE INPUT LOGIC GATES

1) NAND Gate:

```
library ieee;
use ieee.std_logic_1164.all;
entity nand3 is
        port(a,b,c:in bit;
        d:out bit);
end nand3;
architecture nanddf of nand3 is
begin
        d<=(not a)or(not b)or(not c);
end nanddf;
```

2)NOR Gate:

```
library ieee;
use ieee.std_logic_1164.all;
    entity nor3 is
            port(a,b,c:in bit;
            d:out bit);
end nor3;
architecture nordf of nor3 is
    begin
            d<=(not a)or(not b)or(not c);
end nordf;
```

THEORY:

AND: 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. Bear in mind that this dot is sometimes omitted i.e. AB

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

NOT: 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. The diagrams below show two ways that the NAND logic gate can be configured to produce a NOT gate. It can also be done using NOR logic gates in the same way.

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

NOR: 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.

EX-OR: 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 EOR operation.

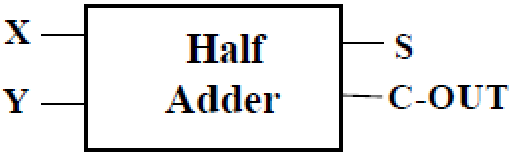
EX-NOR: The **'Exclusive-NOR'** gate circuit does the opposite to the EOR gate. It will give a low output if **either, but not both**, of its two inputs are high. The symbol is an EXOR gate with a small circle on the output. The small circle represents inversion.

PROCEDURE :

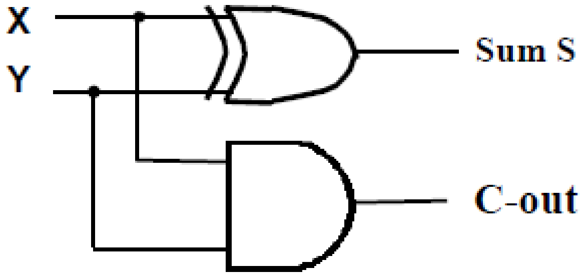
RESULT:

HALF ADDER:

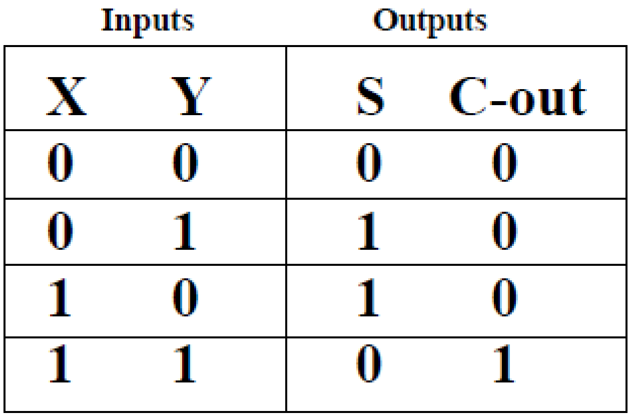
Symbol:



Logic diagram:



Truth Table:



Inputs		Outputs	
X	Y	S	C-out
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = X \oplus Y$$
$$C\text{-out} = XY$$

EXPERIMENT NO.2

ADDERS

AIM:

PROGRAMS:

HALF ADDER:

```
library ieee;
use ieee.std_logic_1164.all;
entity ha is
        port(a,b:in bit;
        s,co:out bit);
end ha;
architecture hadf of ha is
begin
        s<= a xor b;
        co<= a and b;
end hadf;
```

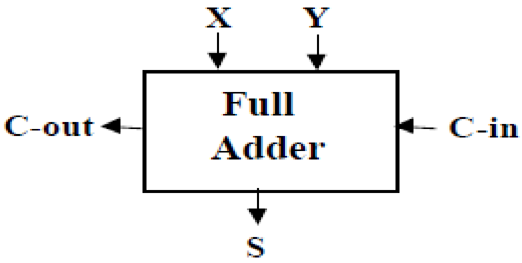
FULL ADDER:

1) Data Flow Model:

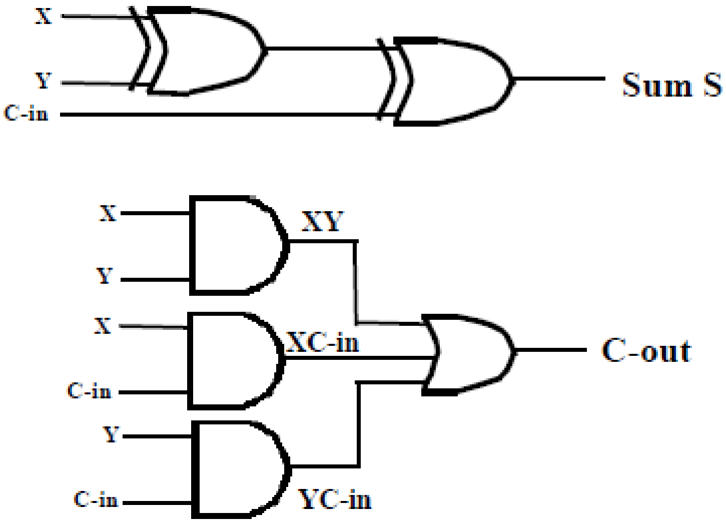
```
library ieee;
use ieee.std_logic_1164.all;
entity fa is
        port(a,b,ci:in bit;
        s,co:out bit);
end fa;
architecture fadf of fa is
begin
        s<=a xor b xor ci;
        co<= (a and b)or(b and ci)or(ci and a);
end fadf;
```

FULL ADDER:

Symbol:



Logic Diagram:



Truth Table:

Inputs			Outputs	
X	Y	C-in	S	C-out
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

2) Behavioural Model:

```
library ieee;
use ieee.std_logic_1164.all;
entity fa is
        port(a,b,ci:in bit;
        s,co:out bit);
end fa;
architecture fabh of fa is
begin
        process(a,b,ci)
        begin
        s<=a xor b xor ci;
        co<= (a and b)or(b and ci)or(ci and a);
        end process;
end fabh;
```

THEORY:

A **half adder** has two inputs, generally labelled *A* and *B*, and two outputs, the sum *S* and carry *C*. *S* is the two-bit XOR of *A* and *B*, and *C* is the AND of *A* and *B*. Essentially the output of a half adder is the sum of two one-bit numbers, with *C* being the most significant of these two outputs. A **half adder** is a logical circuit that performs an addition operation on two binary digits. The half adder produces a sum and a carry value which are both binary digits.The drawback of this circuit is that in case of a multibit addition, it cannot include a carry.

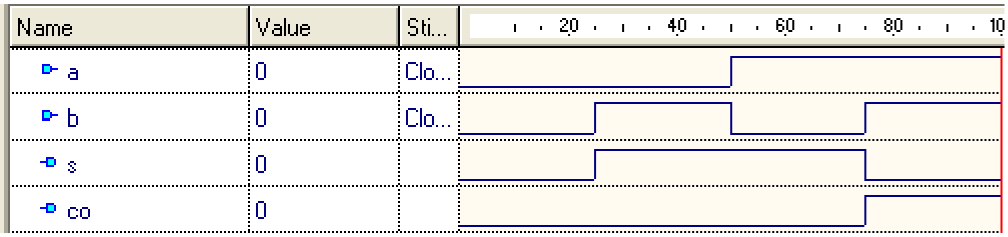
$$S = A \oplus B$$
$$C = A \cdot B$$

A **full adder** has three inputs - *A*, *B*, and a carry in *C*, such that multiple adders can be used to add larger numbers. To remove ambiguity between the input and output carry lines, the carry in is labelled C_i or C_{in} while the carry out is labelled C_o or C_{out}. A **full adder** is a logical circuit that performs an addition operation on three binary digits. The full adders produces a sum and carry value, which are both binary digits

$$S = (A \oplus B) \oplus C_{in}$$
$$C_{out} = (A \cdot B) + (C_{in} \cdot (A \oplus B)) = (A \cdot B) + (C_{in} \cdot B) + (C_{in} \cdot A)$$

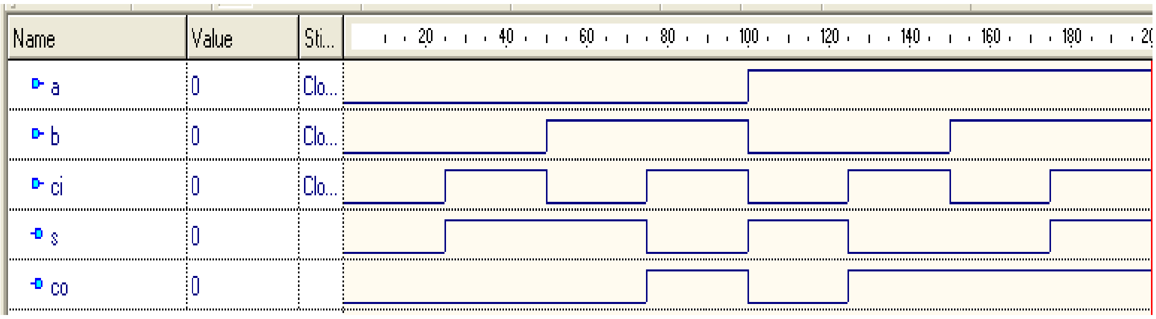
OUTPUT WAVEFORMS:

HALF ADDER:



Name	Value	Sti...
a	0	Clo...
b	0	Clo...
s	0	
co	0	

FULL ADDER:



Name	Value	Sti...
a	0	Clo...
b	0	Clo...
ci	0	Clo...
s	0	
co	0	

PROCEDURE:

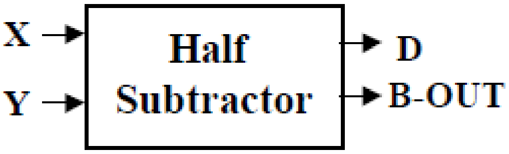
1). To start working with the program go to the **Start-Programs-ActiveHDL7.2 SE** program group and click the **Active-HDL7.2 SE**. The Active-HDL7.3 should start loading.
2). Ensure that the **Create new workspace** option is checked and click the **OK** button. It will create a **new workspace**. Enter the workspace name. Select the location of your workspace folder. Then Click **OK.**
3).New Design Wizard has three options. Here ensure that the **Create an empty design** is checked and click Next button. It will go to next step to **specify additional information about the new design.**
4). Keep the default options and click **Next** to **specify basic information about the new design.**
5). In the **Type the design name**, enter a name. Also type or select the desired location of the design folder in the appropriate field.
6) Click the **Next** button to advance to the next page. Then click **Finish** and this will bring out the **Design Flow Manager**. The next paragraph introduces the **Design Browser.**
7). The **Design Browser** is a window showing the design contents. Double-click the **Add New File** in Design Browser to **Add New File** *(file name)* in the Name item. Then click **OK** to get an empty VHDL file. (Make sure that the VHDL Source Code icon is highlighted in the Empty Files field.)
8). Now we can edit the VHDL code. The HDL Editor is a text editor with VHDL keywords coloring and standard editing features.
9). Select *(filename)*.vhd in the **Design Browser**, then go to the **Design menu,** click the **Compile** option from the menu or press **F11** to compile *(filename)*.vhd file. If there is any error, the file icon changes to red error, erroneous lines are underlined and the VHDL console window contains the error description. You need to correct the error based on the error information in the VHDL Console.
10). To begin a simulation, you have to first initialize the simulator using the **Initialize Simulation** option from the **Simulation** menu.
After the simulator has been initialized, you have to open a new Waveform window. Click the New Waveform toolbar button . The new Waveform window appears.

11). Signals for simulation can be added using the drag and drop feature. On the Structure tab of the Design Browser you have to select the component whose ports you want to observe and simply holding the left button, drag it to the left-hand section (pane) of the waveform window and release the button (typical drag-and-drop operation). Follow the procedure described above to drag all ports of the components to(*filename*) the waveform window.
12) Go to the left pane of the Waveform window and select a signal. Press the right button to invoke a pop-up menu. Choose the **Stimulators** item and assign signal values for each signal and close the pop-up menu.
13). You can choose the command of **Run** or **Run Until** or **Run For** from main menu of **Stimulator**. **Run** keeps the simulator run forward .**Run For** runs the simulator for defined time every step and **Run Until** runs the simulators in the defined time.

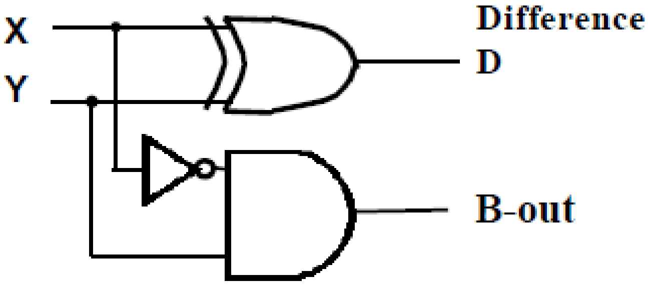
RESULT:

HALF SUBTRACTOR:

Symbol:



Logic Diagram:



Truth Table:

Inputs		Outputs	
X	Y	D	B-out
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$$D = X \oplus Y$$

$$B\text{-out} = X'Y$$

EXPERIMENT NO.3

SUBTRACTORS

AIM:

PROGRAMS:

HALF SUBTRACTOR:

```
library ieee;
use ieee.std_logic_1164.all;
entity hs is
        port(a,b:in bit;
        d,bo:out bit);
end hs;
architecture hsdf of hs is
begin
        d<= a xor b;
        bo<= (not a) and b;
end hsdf;
```

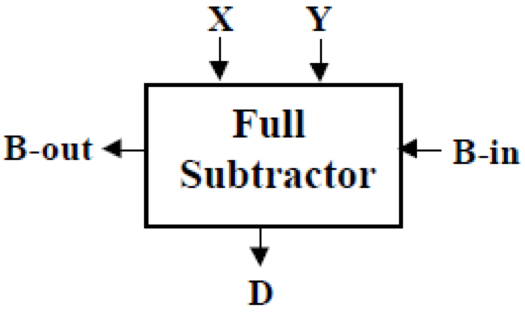
FULL SUBTRACTOR:

1) Data Flow Model:

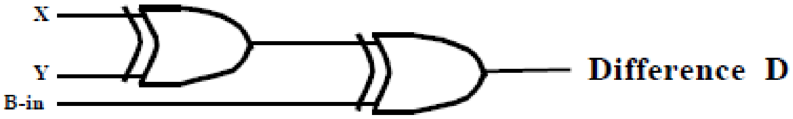
```
library ieee;
use ieee.std_logic_1164.all;
entity fs is
        port(a,b,bi:in bit;
        d,bo:out bit);
end fs;
architecture fsdf of fs is
begin
        d<=a xor b xor bi;
        bo<= ((not a) and b)or(b and bi)or(bi and (not a));
        end fsdf;
```

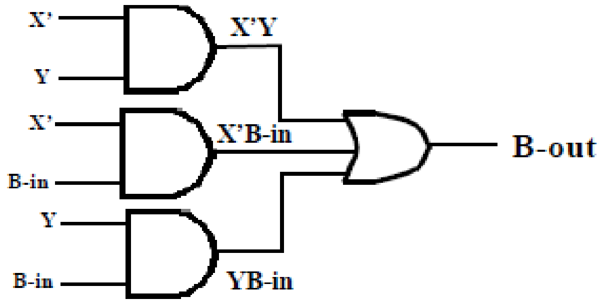
FULL SUBTRACTOR:

Symbol:



Logic Diagram:





Truth Table:

Inputs			Outputs	
X	Y	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

2) Behavioral Model:

```
library ieee;
use ieee.std_logic_1164.all;
entity fs is
        port(a,b,bi:in bit;
        d,bo:out bit);
end fs;
architecture fsbh of fs is
begin
        process(a,b,bi)
        begin
        d<=a xor b xor bi;
        bo<= ((not a) and b)or(b and bi)or(bi and (not a));
        end process;
end fsbh;
```

THEORY:

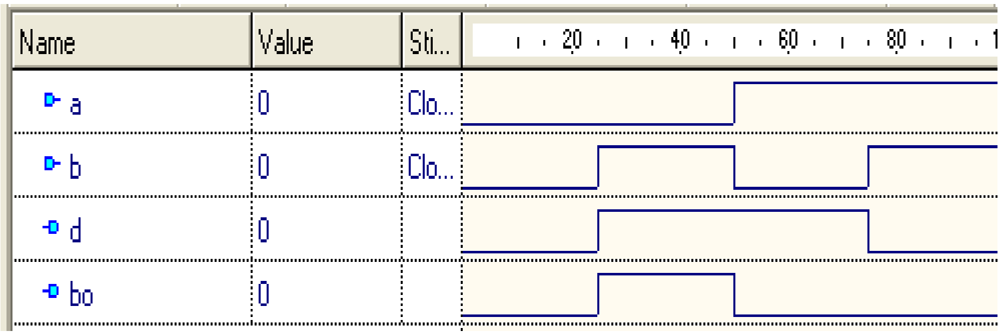
The half-subtractor is a combinational circuit which is used to perform subtraction of two bits. It has two inputs, X(minuend) and Y(subtrahend) and two outputs D (difference) and B (borrow).
The Unit that performs 1-bit subtraction with borrow-in is defined as a fullsubtractor.It has input bits A,B and Bin(borrow in) and the output bits D(difference)and Bout (borrow out) .

PROCEDURE:

RESULT:

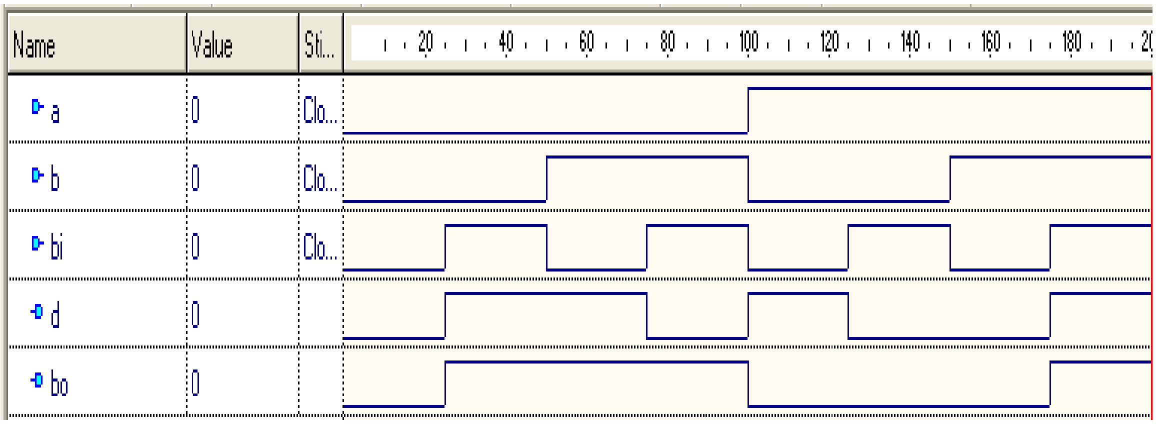
OUTPUT WAVEFORMS:

HALF SUBTRACTOR



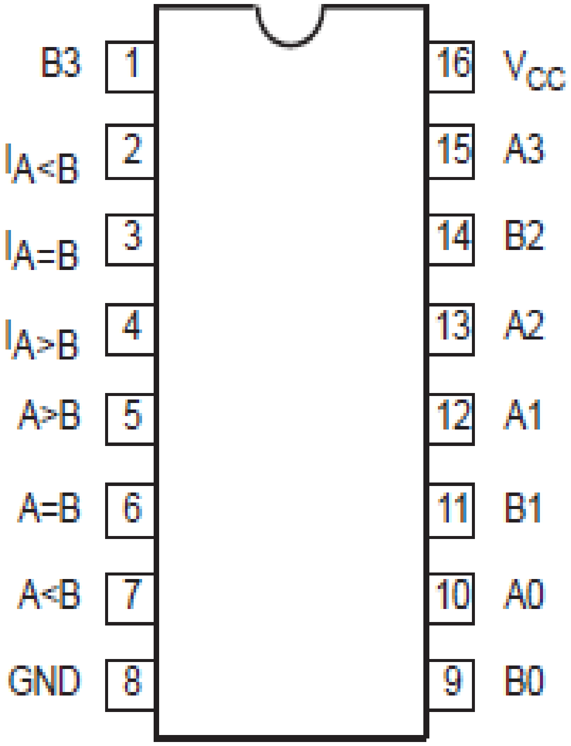
Name	Value	Sti...
a	0	Clo...
b	0	Clo...
d	0	
bo	0	

FULL SUBTRACTOR:

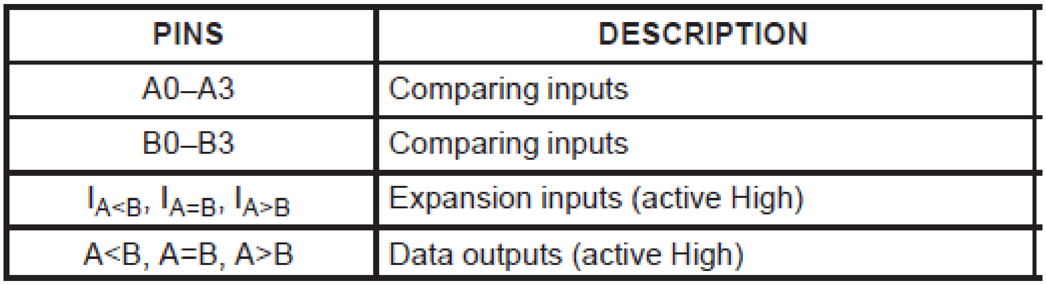


Name	Value	Sti...
a	0	Clo...
b	0	Clo...
bi	0	Clo...
d	0	
bo	0	

PIN DIAGRAM – 7485 (4 BIT COMPARATOR):



PIN DESCRIPTION:



PINS	DESCRIPTION
A0–A3	Comparing inputs
B0–B3	Comparing inputs
$I_{A<B}$, $I_{A=B}$, $I_{A>B}$	Expansion inputs (active High)
A<B, A=B, A>B	Data outputs (active High)

EXPERIMENT NO.4

4 BIT COMPARATOR-7485

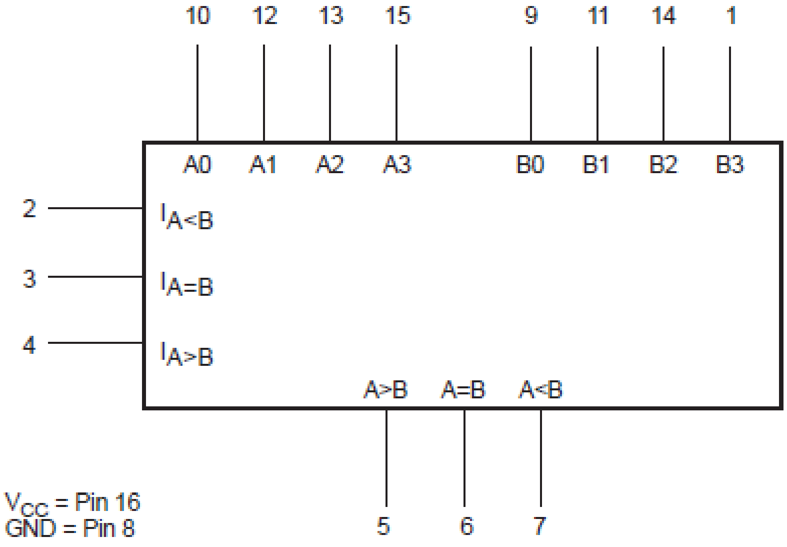
AIM:

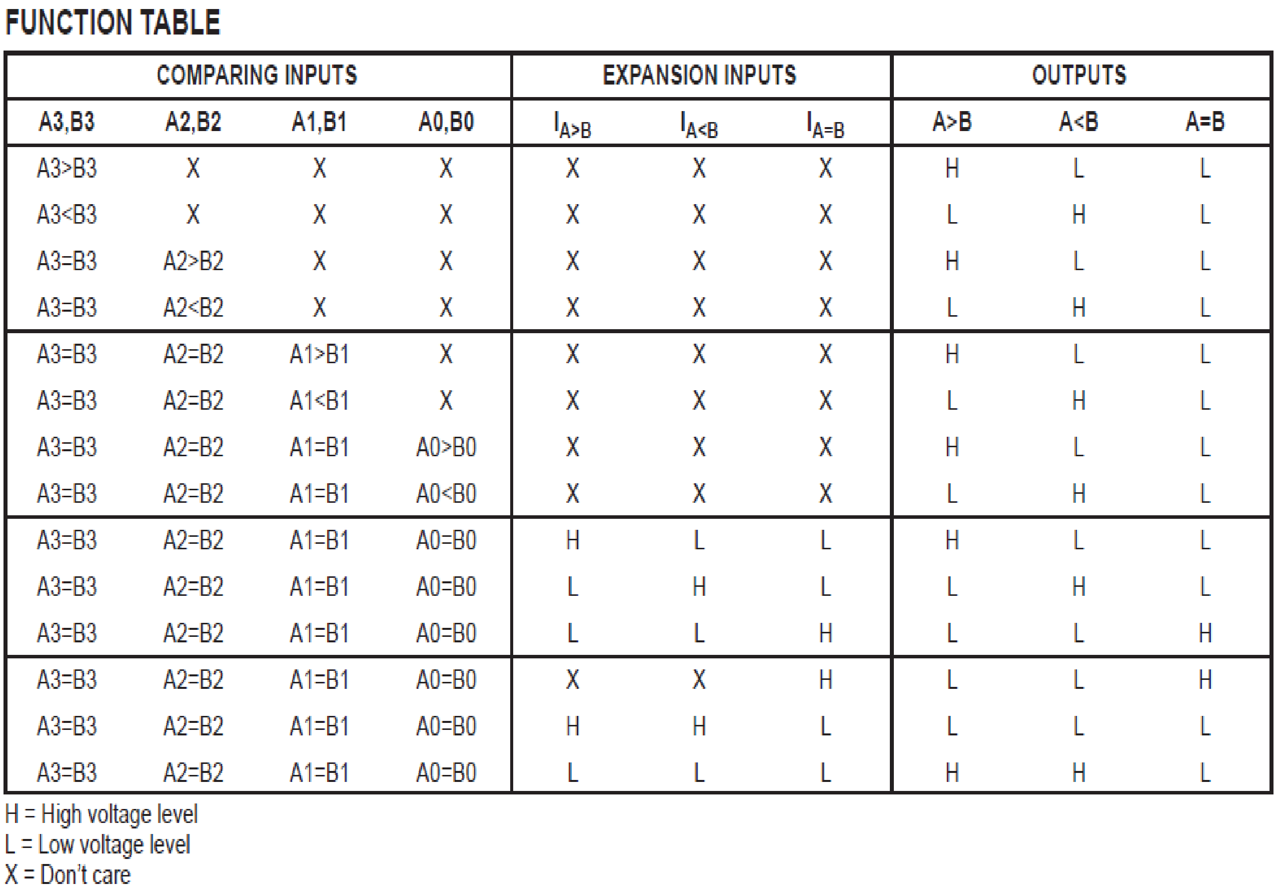
PROGRAM:

1) Data Flow Model:

```
 library ieee;
use ieee.std_logic_1164.all;
entity comp is
        port(a,b:in bit_vector(3 downto 0);
        AeqB,AltB,AgtB:out bit);
end comp;
architecture compdf of comp is
begin
        AeqB<='1' when a=b else'0';
        AltB<='1' when a<b else'0';
        AgtB<='1' when a>b else'0';
end compdf;
```

LOGIC SYMBOL:





FUNCTION TABLE

COMPARING INPUTS				EXPANSION INPUTS			OUTPUTS		
A3,B3	A2,B2	A1,B1	A0,B0	$I_{A>B}$	$I_{A<B}$	$I_{A=B}$	A>B	A<B	A=B
A3>B3	X	X	X	X	X	X	H	L	L
A3<B3	X	X	X	X	X	X	L	H	L
A3=B3	A2>B2	X	X	X	X	X	H	L	L
A3=B3	A2<B2	X	X	X	X	X	L	H	L
A3=B3	A2=B2	A1>B1	X	X	X	X	H	L	L
A3=B3	A2=B2	A1<B1	X	X	X	X	L	H	L
A3=B3	A2=B2	A1=B1	A0>B0	X	X	X	H	L	L
A3=B3	A2=B2	A1=B1	A0<B0	X	X	X	L	H	L
A3=B3	A2=B2	A1=B1	A0=B0	H	L	L	H	L	L
A3=B3	A2=B2	A1=B1	A0=B0	L	H	L	L	H	L
A3=B3	A2=B2	A1=B1	A0=B0	L	L	H	L	L	H
A3=B3	A2=B2	A1=B1	A0=B0	X	X	H	L	L	H
A3=B3	A2=B2	A1=B1	A0=B0	H	H	L	L	L	L
A3=B3	A2=B2	A1=B1	A0=B0	L	L	L	H	H	L

H = High voltage level
L = Low voltage level
X = Don't care

2) Behavioral Model:

,

```
library ieee;
use ieee.std_logic_1164.all;
entity comp is
        port(a,b:in bit_vector(3 downto 0);
        AeqB,AltB,AgtB:out bit);
end comp;
architecture compbh of comp is
begin
        process(a,b)
        begin
        AeqB<='0';AltB<='0';AgtB<='0';
        if a=b then   AeqB<='1';
        elsif a<b then AltB<='1';
        elsif a>b then AgtB<='1';
        end if;
        end process;
end compbh;
```

THEORY:

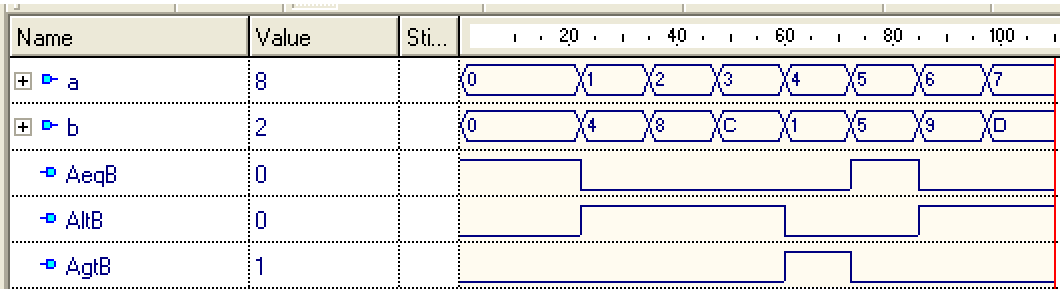
The 7485 is a 4-bit magnitude comparator that can be expanded to almost any length. It compares two 4-bit binary, BCD codes and presents the three possible magnitude results at the outputs. The 4-bit inputs are weighted (A0-A3) and (B0-B3) where A3 and B3 are the most significant bits. The operation of the 7485 is described in the function table, showing all possible logic conditions.

PROCEDURE:

RESULT:

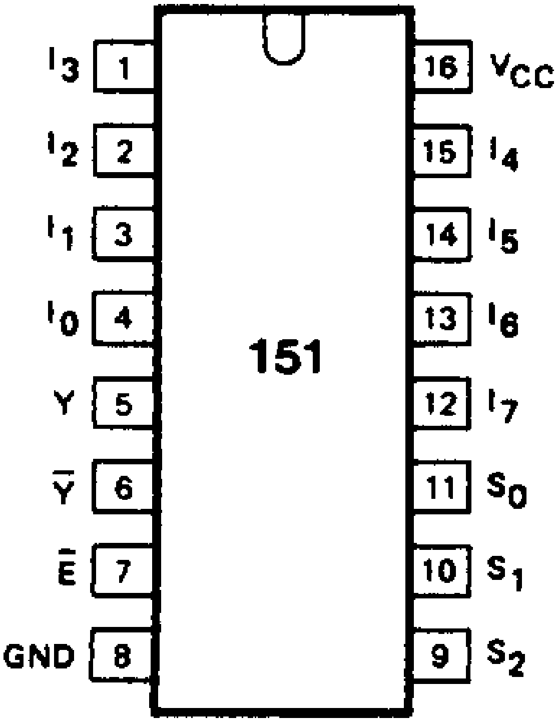
OUTPUT WAVEFORM:

COMPARATOR:



Name	Value	Sti...
a	8	
b	2	
AeqB	0	
AltB	0	
AgtB	1	

PIN DIAGRAM – 74151(8X1 MULTIPLEXER):



PIN DESCRIPTION:

PIN NO.	SYMBOL	NAME AND FUNCTION
4, 3, 2, 1, 15, 14, 13, 12	I_0 to I_7	multiplexer inputs
5	Y	multiplexer output
6	$\overline{Y}$	complementary multiplexer output
7	$\overline{E}$	enable input (active LOW)
8	GND	ground (0 V)
11, 10, 9	S_0, S_1, S_2	select inputs
16	V_{CC}	positive supply voltage

EXPERIMENT NO.5

MULTIPLEXERS

AIM:

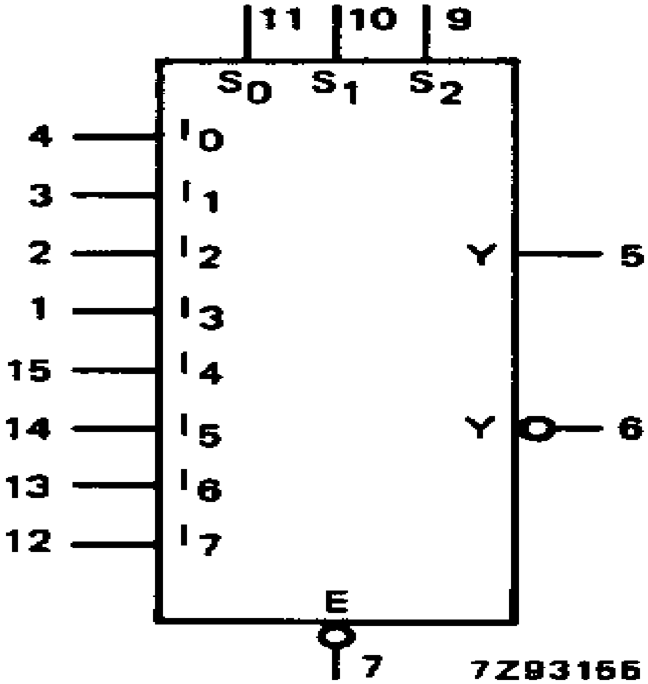
PROGRAMS:

<u>8X1 MULTIPLEXER:</u> (74151)

1) Data Flow Model:

```
library ieee;
use ieee.std_logic_1164.all;
entity mux8x1 is
        port(I0,I1,I2,I3,I4,I5,I6,I7:in std_logic_vector(3 downto 0);
        s:in std_logic_vector (2 downto 0);
        Y:out std_logic_vector(3 downto 0));
end mux8x1;
architecture mux8x1df of mux8x1 is
begin
        with s select
        Y<=I0 when "000",
        I1 when "001",
        I2 when "010",
        I3 when "011",
        I4 when "100",
        I5 when "101",
        I6 when "110",
        I7 when "111",
        "0000" when others;
end mux8x1df;
```

LOGIC SYMBOL:



FUNCTION TABLE:

INPUTS												OUTPUTS	
$\overline{E}$	S_2	S_1	S_0	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	$\overline{Y}$	Y
H	X	X	X	X	X	X	X	X	X	X	X	H	L
L	L	L	L	L	X	X	X	X	X	X	X	H	L
L	L	L	L	H	X	X	X	X	X	X	X	L	H
L	L	L	H	X	L	X	X	X	X	X	X	H	L
L	L	L	H	X	H	X	X	X	X	X	X	L	H
L	L	H	L	X	X	L	X	X	X	X	X	H	L
L	L	H	L	X	X	H	X	X	X	X	X	L	H
L	L	H	H	X	X	X	L	X	X	X	X	H	L
L	L	H	H	X	X	X	H	X	X	X	X	L	H
L	H	L	L	X	X	X	X	L	X	X	X	H	L
L	H	L	L	X	X	X	X	H	X	X	X	L	H
L	H	L	H	X	X	X	X	X	L	X	X	H	L
L	H	L	H	X	X	X	X	X	H	X	X	L	H
L	H	H	L	X	X	X	X	X	X	L	X	H	L
L	H	H	L	X	X	X	X	X	X	H	X	L	H
L	H	H	H	X	X	X	X	X	X	X	L	H	L
L	H	H	H	X	X	X	X	X	X	X	H	L	H

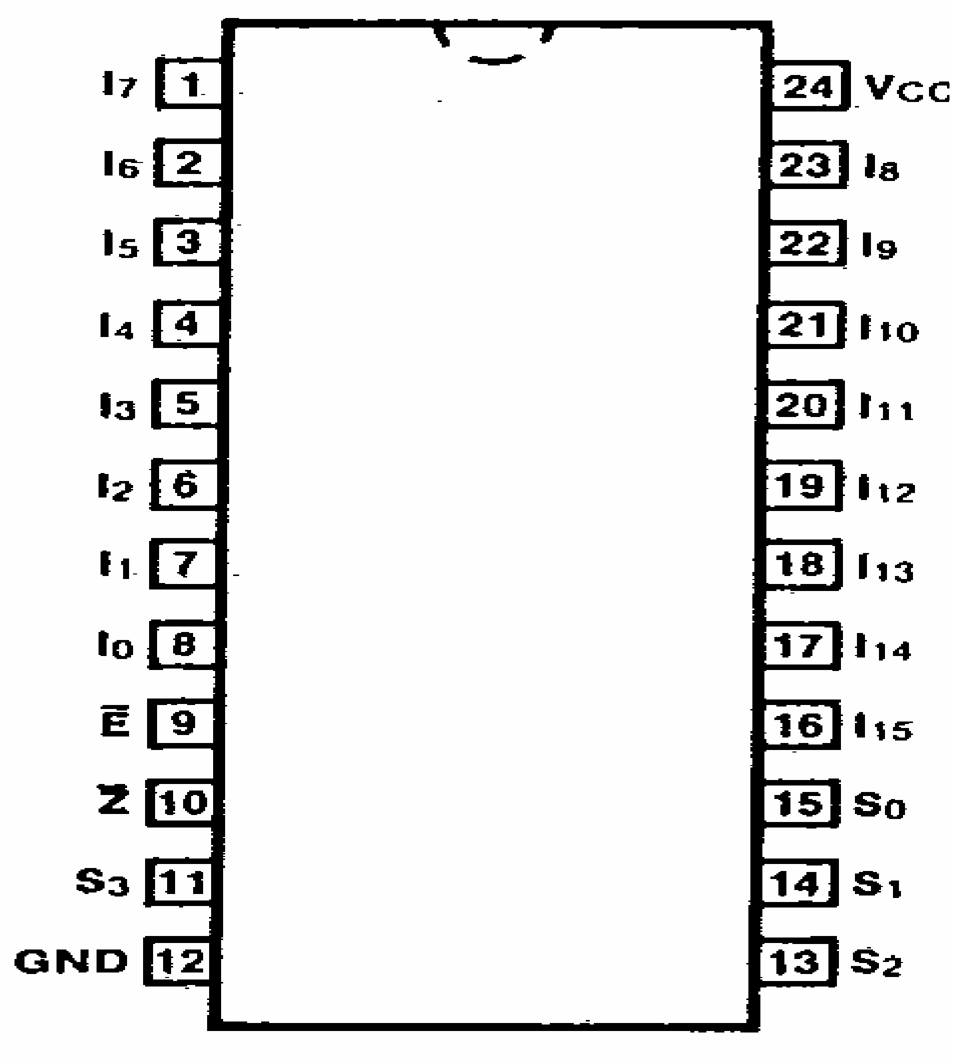
Notes

1. H = HIGH voltage level
   L = LOW voltage level
   X = don't care.

2) Behavioural Model:

```
  library ieee;
  use ieee.std_logic_1164.all;
  entity mux8x1 is
        port(I0,I1,I2,I3,I4,I5,I6,I7:in std_logic_vector(3 downto 0);
        s:in std_logic_vector (2 downto 0);
        Y:out std_logic_vector(3 downto 0));
end mux8x1;
  architecture mux8x1bh of mux8x1 is
  begin
          process(s,I0,I1,I2,I3,I4,I5,I6,I7)
          begin
                  case s is
                          when "000"=>y<=I0;
                          when "001"=>y<=I1;
                          when "010"=>y<=I2;
                          when "011"=>y<=I3;
                          when "100"=>y<=I4;
                          when "101"=>y<=I5;
                          when "110"=>y<=I6;
                          when "111"=>y<=I7;
                          when others=>y<=(others=>'U');
                  end case;
                  end process;
  end mux8x1bh;
```

PIN DIAGRAM – 74150(16X1 MULTIPLEXER):



PIN DESCRIPTION:

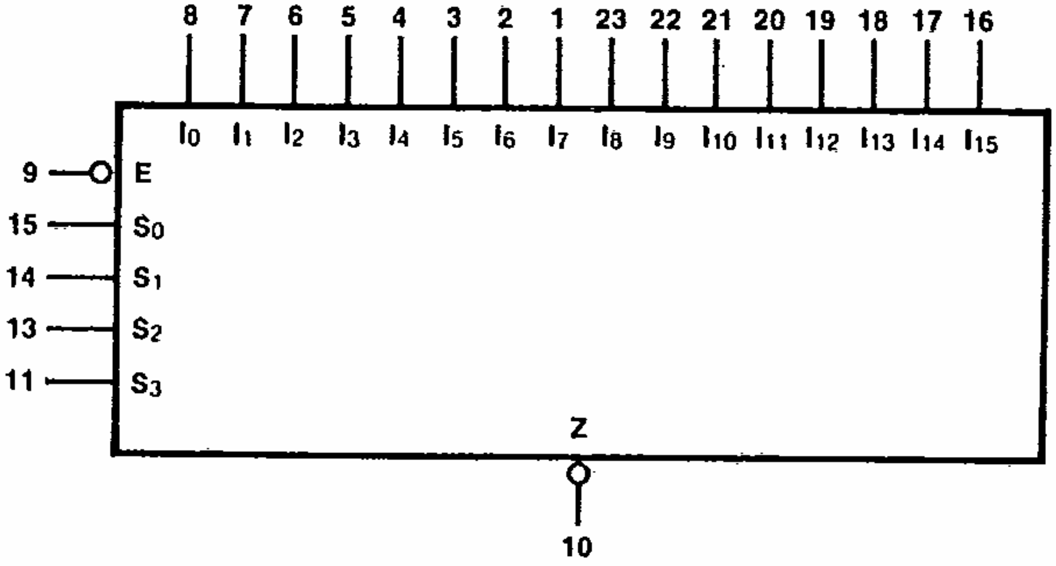
PIN NAMES	DESCRIPTION
$I_0 — I_{15}$	Data Inputs
$S_0 — S_3$	Select Inputs
$\bar{E}$	Enable Input (Active LOW)
$\bar{Z}$	Inverted Data Output

16X1 MULTIPLEXER: (74150)

1) Behavioural Model:

```
  library ieee;
  use ieee.std_logic_1164.all;
  entity mux16x1 is
port(I0,I1,I2,I3,I4,I5,I6,I7,I8,I9,I10,I11,I12,I13,I14,I15:in std_logic_vector(3
downto 0);
          s:in std_logic_vector (3 downto 0);
          Z:out std_logic_vector(3 downto 0));
  end mux16x1;
  architecture mux16x1bh of mux16x1 is
  begin
          process(s, I0,I1,I2,I3,I4,I5,I6,I7,I8,I9,I10,I11,I12,I13,I14,I15)
          begin
                  case s is
                          when "0000"=>y<=I0;
                          when "0001"=>y<=I1;
                          when "0010"=>y<=I2;
                          when "0011"=>y<=I3;
                          when "0100"=>y<=I4;
                          when "0101"=>y<=I5;
                          when "0110"=>y<=I6;
                          when "0111"=>y<=I7;
                          when "1000"=>y<=I8;
                          when "1001"=>y<=I9;
                          when "1010"=>y<=I10;
                          when "1011"=>y<=I11;
                          when "1100"=>y<=I12;
                          when "1101"=>y<=I13;
                          when "1110"=>y<=I14;
                          when "1111"=>y<=I15;
                          when others=>y<=(others=>'U');
                  end case;
                  end process;
  end mux16x1bh;
```

LOGIC SYMBOL:



Vcc = Pin 24
GND = Pin 12

FUNCTION TABLE:

INPUTS					OUTPUT
S_3	S_2	S_1	S_0	$\bar{E}$	Z
X	X	X	X	H	H
L	L	L	L	L	$\bar{I}_0$
L	L	L	H	L	$\bar{I}_1$
L	L	H	L	L	$\bar{I}_2$
•	•	•	•	•	•
H	H	L	L	L	$\bar{I}_{12}$
H	H	L	H	L	$\bar{I}_{13}$
H	H	H	L	L	$\bar{I}_{14}$
H	H	H	H	L	$\bar{I}_{15}$

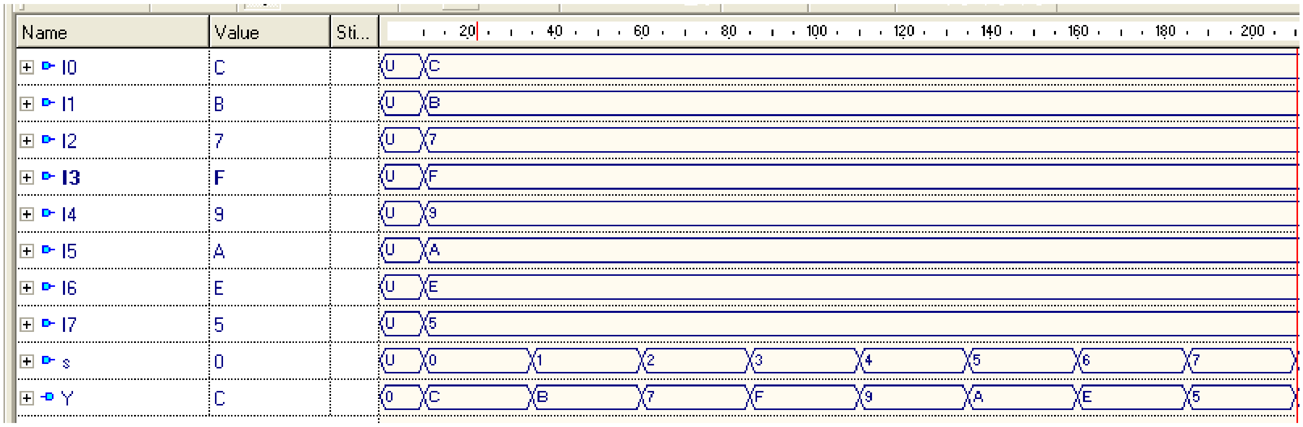
H = HIGH Voltage Level
L = LOW Voltage Level
X = Immaterial

2) Data Flow Model:

```
library ieee;
use ieee.std_logic_1164.all;
entity mux16x1 is
        port(I0,I1,I2,I3,I4,I5,I6,I7,I8,I9,I10,I11,I12,I13,I14,I15:in
std_logic_vector(3 downto 0);
        s:in std_logic_vector (3 downto 0);
        Z:out std_logic_vector(3 downto 0));
end mux16x1;
architecture mux16x1df of mux16x1 is
begin
        with s select
        Z<=I0 when"0000",
        I1 when"0001",
        I2 when"0010",
        I3 when"0011",
        I4 when"0100",
        I5 when"0101",
        I6 when"0110",
        I7 when"0111",
        I8 when"1000",
        I9 when"1001",
        I10 when"1010",
        I11 when"1011",
        I12 when"1100",
        I13 when"1101",
        I14 when"1110",
        I15 when"1111",
        "0000"when others;
end mux16x1df;
```

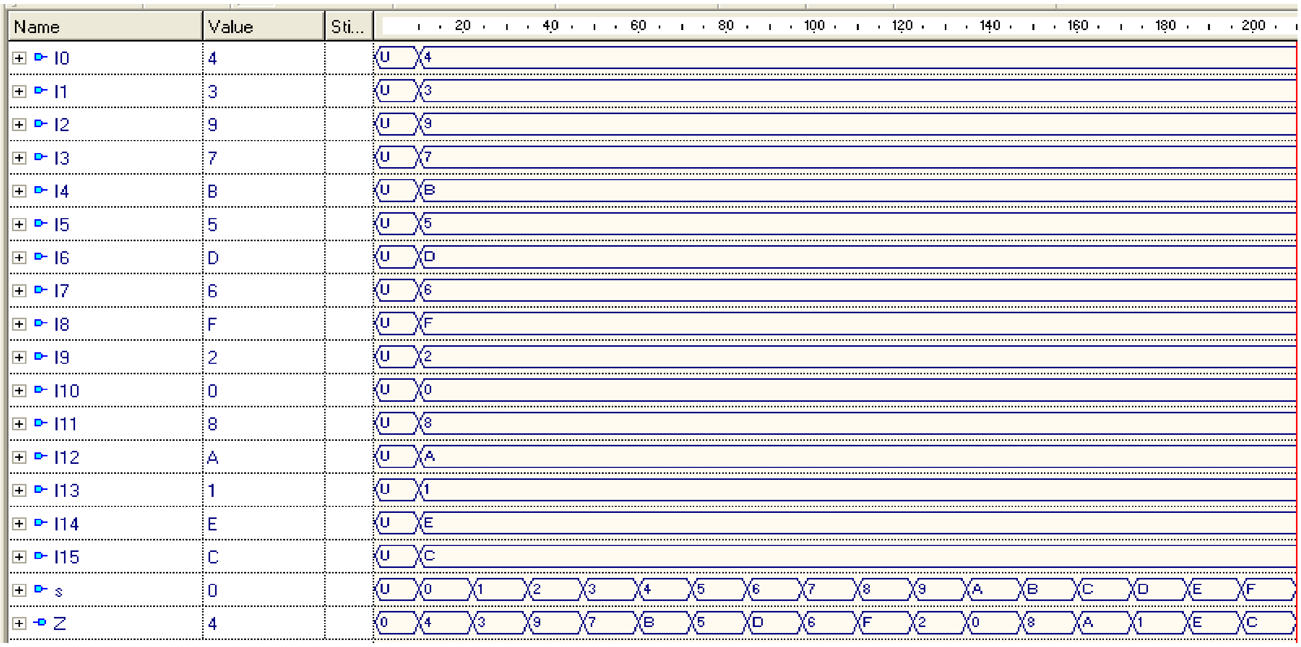
OUTPUT WAVEFORMS:

MULTIPLEXER 8X1:



Name	Value	Sti...
I0	C	
I1	B	
I2	7	
I3	**F**	
I4	9	
I5	A	
I6	E	
I7	5	
s	0	
Y	C	

MULTIPLEXER 16X1:



Name	Value	Sti...
I0	4	
I1	3	
I2	9	
I3	7	
I4	B	
I5	5	
I6	D	
I7	6	
I8	F	
I9	2	
I10	0	
I11	8	
I12	A	
I13	1	
I14	E	
I15	C	
s	0	
Z	4	

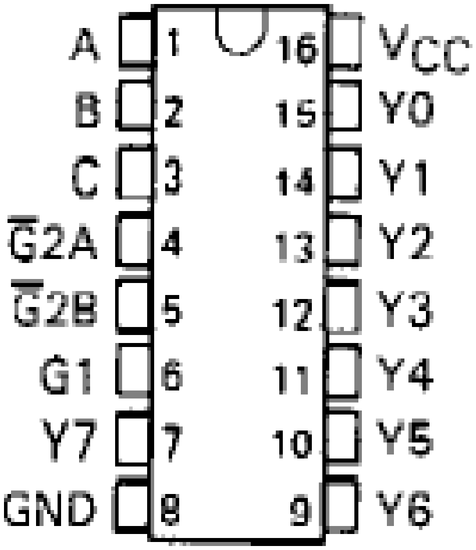
THEORY:

The multiplexers contains full on-chip decoding unit to select desired data source. The 74151 selects one-of-eight data sources. It has a enable input which must be at a LOW logic level to enable these devices. These perform parallel-to-serial conversion. The 74150 selects one-of sixteen data sources.

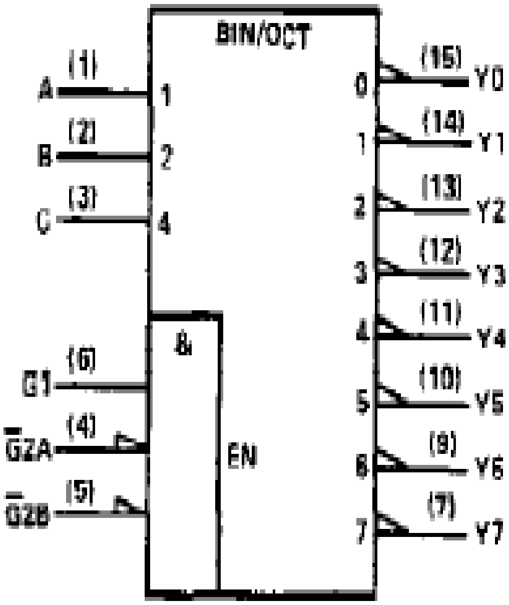
PROCEDURE:

RESULT:

PIN DIAGRAM – 74138 (3-TO-8 DECODER):



LOGIC SYMBOL:



EXPERIMENT NO.6

3-8 DECODER (74138)

AIM:

PROGRAM:

1) Data Flow Model:

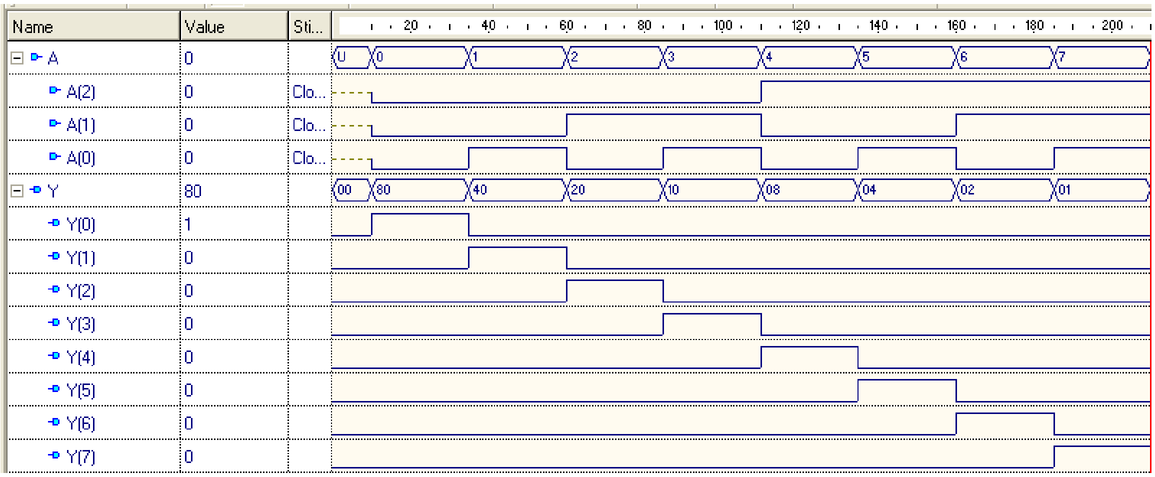
```
library ieee;
use ieee.std_logic_1164.all;
entity dec3to8 is
        port(A: in STD_LOGIC_VECTOR(2 downto 0);
                Y: out STD_LOGIC_VECTOR(0 to 7));
end dec3to8;
architecture dec3to8df of dec3to8 is
begin
        with A select
          Y <= "10000000" when "000",
                  "01000000" when "001",
                  "00100000" when "010",
                  "00010000" when "011",
                  "00001000" when "100",
                  "00000100" when "101",
                  "00000010" when "110",
                  "00000001" when "111",
                  "00000000" when others;
end dec3to8df;
```

FUNCTION TABLE:

INPUTS					OUTPUTS							
ENABLE		SELECT										
G1	$\overline{G2}$*	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

*$\overline{G2} = \overline{G2A} + \overline{G2B}$
H = high level, L = low level, X = irrelevant

OUTPUT WAVEFORM:



2) Behavioral Model:

```
library ieee;
use ieee.std_logic_1164.all;
entity dec3to8 is
        port(A: in STD_LOGIC_VECTOR(2 downto 0);
                Y: out STD_LOGIC_VECTOR(0 to 7));
end dec3to8;
architecture dec3to8bh of dec3to8 is
begin
        process(A)
  begin
    case A is
      when "000" => Y <= "10000000";
      when "001" => Y <= "01000000";
      when "010" => Y <= "00100000";
      when "011" => Y <= "00010000";
      when "100" => Y <= "00001000";
      when "101" => Y <= "00000100";
      when "110" => Y <= "00000010";
      when "111" => Y <= "00000001";
      when others => Y <= "00000000";
    end case;
        end process;
end dec3to8bh;
```

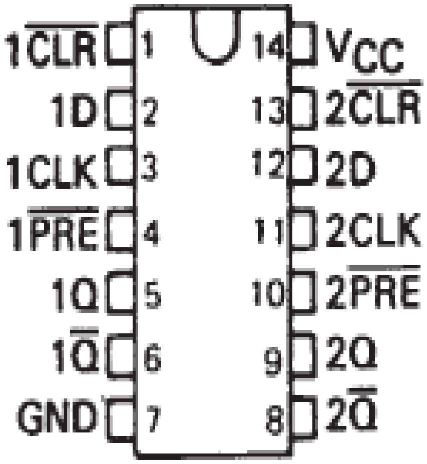
THEORY:

The 74138 decodes one-of-eight lines based upon the conditions at the three binary select inputs and the three enable inputs. Two active low and one active-high enable inputs reduce the need for external gates or inverters when expanding. A 24-line decoder can be implemented with no external inverters, and a 32-line decoder requires only one inverter. An enable input can be used as a data input for demultiplexing applications.

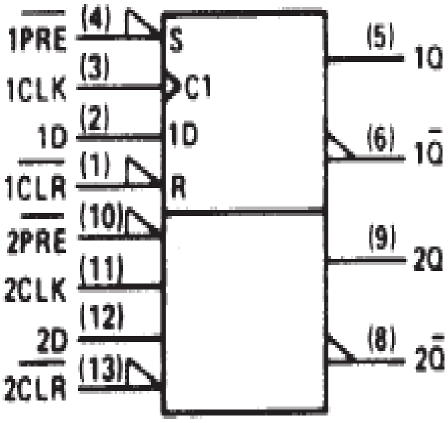
PROCEDURE:

RESULT:

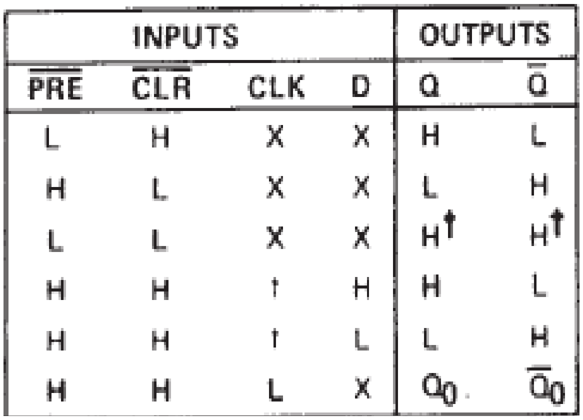
PIN DIAGRAM – 7474 (DUAL D FLIP FLOP):



LOGIC SYMBOL:



FUNCTION TABLE:



INPUTS				OUTPUTS	
$\overline{PRE}$	$\overline{CLR}$	CLK	D	Q	$\overline{Q}$
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	$H^{\uparrow}$	$H^{\uparrow}$
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q_0	$\overline{Q}_0$

EXPERIMENT NO.7

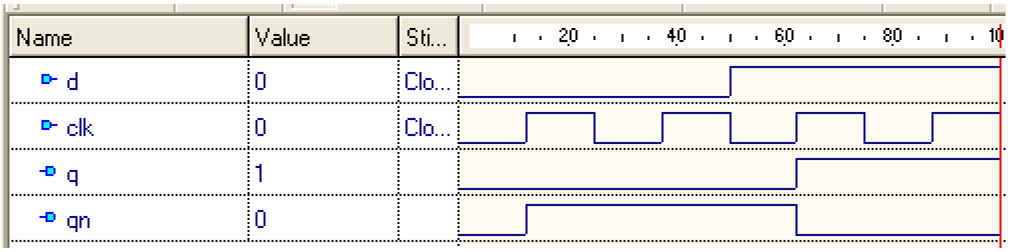
D FLIPFLOP (7474)

AIM:

PROGRAM:

```
library ieee;
use ieee.std_logic_1164.all;
entity dff is
        port(d,clk:in bit;
        q,qn:out bit);
end dff;
architecture dffbh of dff is
begin
        process(d,clk)
        begin
                if(clk'event and clk='1')then
                        q<=d;
                        qn<=not d;
                        end if;
                end process;
end dffbh;
```

OUTPUT WAVEFORM:



Name	Value	Sti...
d	0	Clo...
clk	0	Clo...
q	1	
qn	0	

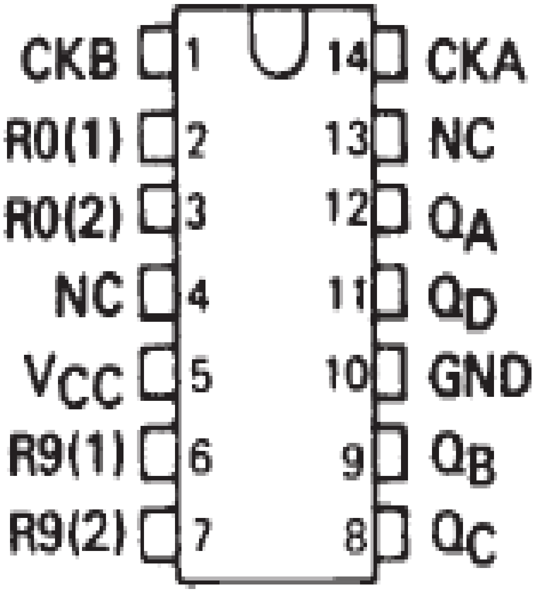
THEORY:

This device contains two independent positive edge-triggered D flip-flops with complementary outputs. The information on the D input is accepted by the flip-flops on the positive going edge of the clock pulse. The triggering occurs at a voltage level and is not directly related to the transition time of the rising edge of the clock. The data on the D may be changed while the clock is low or high without affecting the outputs as long as the data setup and hold times are not violated. A LOW logic level on the preset or clear inputs will set or reset the outputs regardless of the logic levels on the other inputs.

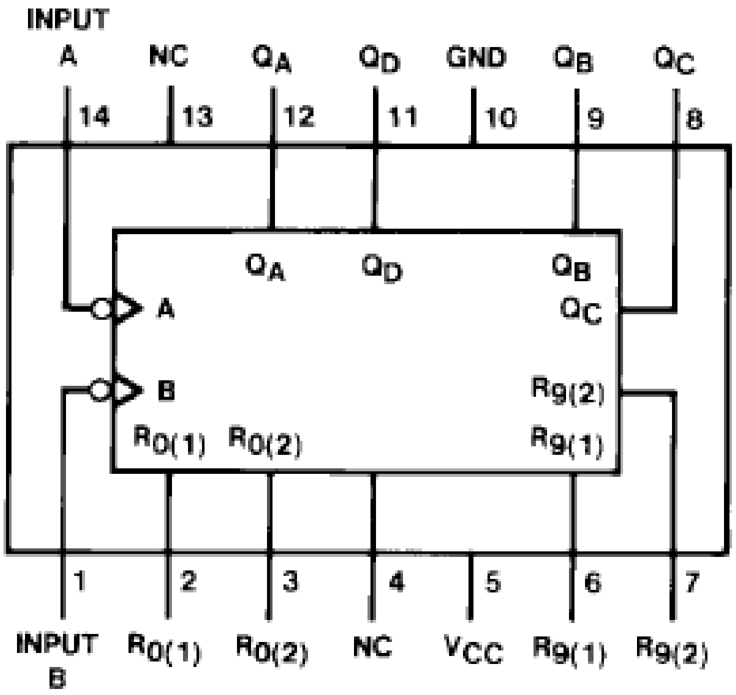
PROCEDURE:

RESULT:

PIN DIAGRAM – 7490 (DECADE COUNTER):



CONNECTION DIAGRAM:



EXPERIMENT NO.8

DECADE COUNTER (7490)

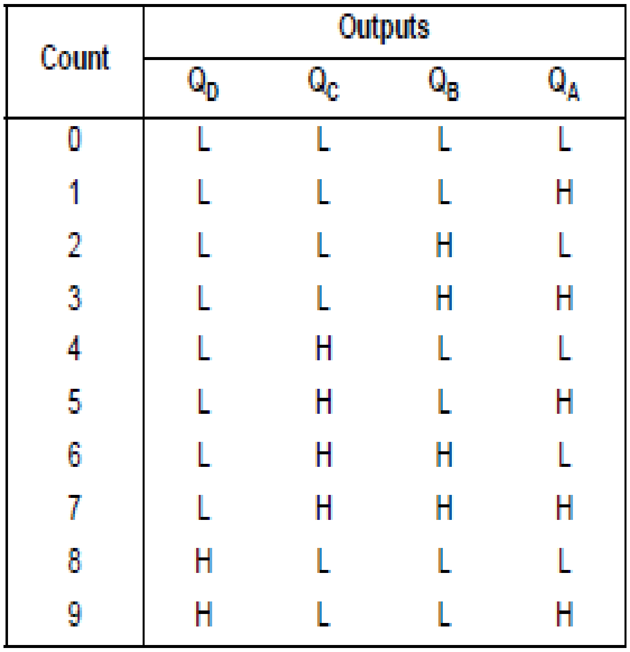
AIM:

PROGRAM:

```
 library IEEE;                                    --library definition
 use IEEE.std_logic_1164.all;
 use IEEE.std_logic_unsigned.all;

entity Counter is                                 --entity definition
  port (
   clk:in std_logic;
   reset: in std_logic;
   q: out std_logic_vector(3 downto 0) );
end Counter;
```

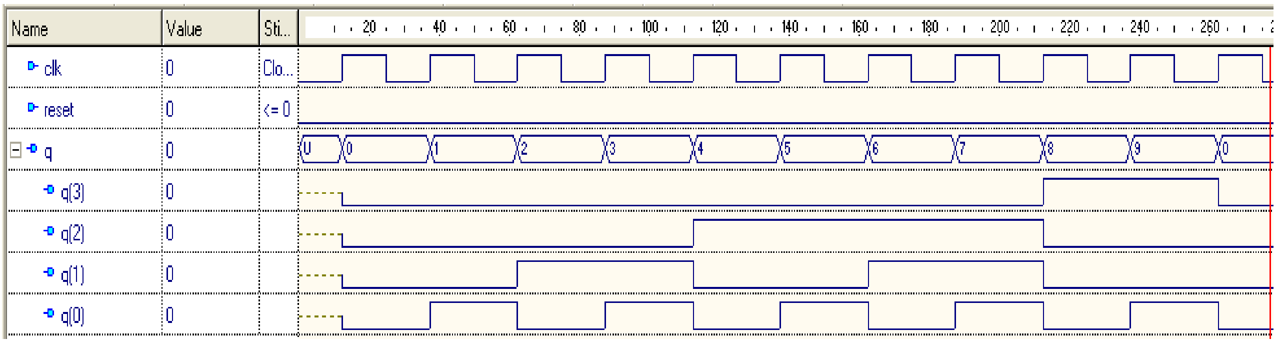
```
architecture Counter of Counter is                -- Architecture definition
begin
   process(clk,reset)                            -- Process definition

variable qtemp: std_logic_vector(3 downto 0);  -- temporary variable for
 begin                                             output q[3..0]
  if reset='1' then
   qtemp:="0000";                                -- Reset asychroniously
  else
   if clk'event and clk='1' then                  -- Counting state
    if qtemp<9 then
     qtemp:=qtemp+1;                             -- Counter increase
    else
     qtemp:="0000";                              -- Return the zero state
    end if;
   end if;
  q<=qtemp;                                     -- Output
   end if;
 end process;                                    -- End Process
end Counter;
```

FUNCTION TABLE:



Count	Outputs			
	Q_D	Q_C	Q_B	Q_A
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	L	H	H	L
7	L	H	H	H
8	H	L	L	L
9	H	L	L	H

OUTPUT WAVEFORM:



Name	Value	Sti...
clk	0	Clo...
reset	0	<= 0
q	0	
q(3)	0	
q(2)	0	
q(1)	0	
q(0)	0	

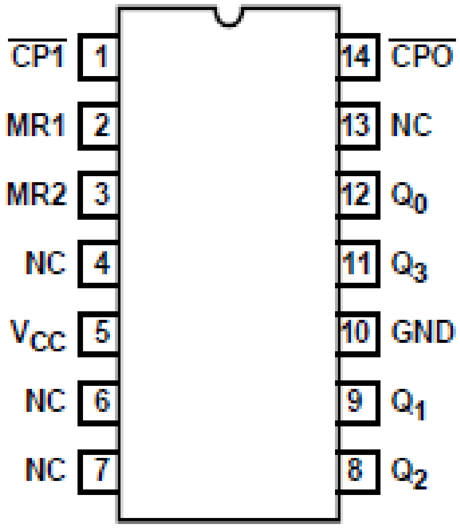
THEORY:

The 7490 decade counter contains four master slave flip-flops and additional gating to provide a divide-by-two counter and a three-stage binary counter for which the count cycle length is divide-by-five. The counter has a gated zero reset and also has a gated set-to-nine input for use in BCD nine's complement applications. A symmetrical divide-by-ten count can be obtained from the counters by connecting QD output to the A input and applying the input count to the B input which gives a divide-by-ten square wave at output QA.

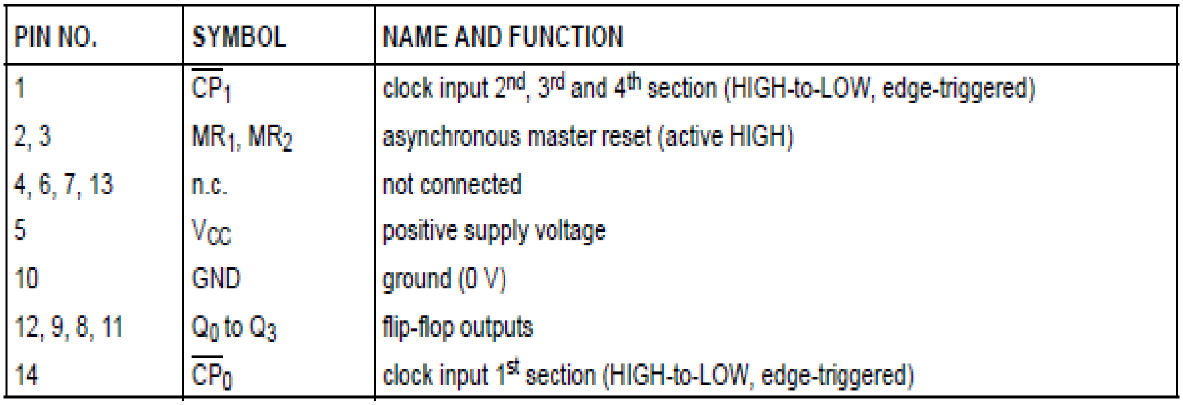
PROCEDURE:

RESULT:

PIN DIAGRAM – 7493 (4 BIT BINARY RIPPLE COUNTER):

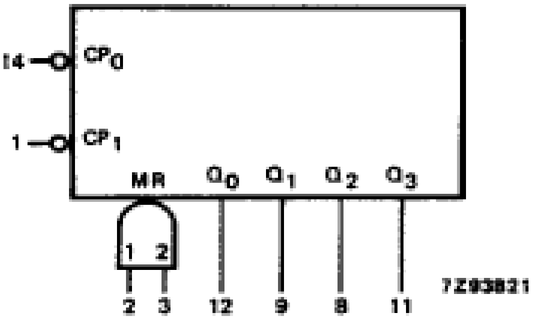


PIN DESCRIPTION:



PIN NO.	SYMBOL	NAME AND FUNCTION
1	$\overline{CP}_1$	clock input 2nd, 3rd and 4th section (HIGH-to-LOW, edge-triggered)
2, 3	MR_1, MR_2	asynchronous master reset (active HIGH)
4, 6, 7, 13	n.c.	not connected
5	V_{CC}	positive supply voltage
10	GND	ground (0 V)
12, 9, 8, 11	Q_0 to Q_3	flip-flop outputs
14	$\overline{CP}_0$	clock input 1st section (HIGH-to-LOW, edge-triggered)

LOGIC SYMBOL:



EXPERIMENT N0.9

4 BIT BINARY RIPPLE COUNTER (7493)

AIM:

PROGRAM:

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity AsynCounter4 is
        port( CLK, RESET, EN : in std_logic;
                                count: out std_logic_vector(3 downto 0) ) ;
end AsynCounter4;

architecture arch_AsynCounter4 of AsynCounter4 is
  signal count_t : std_logic_vector(3 downto 0) ;
  begin
```

```
          process(RESET,CLK)
          begin
                if (RESET = '1') then count_t <= "0000" ;
                elsif (CLK'event and (CLK='1')and(EN = '1'))then count_t <=
count_t +"0001" ;
              end if ;
       end process ;
   count <= count_t ;
end arch_AsynCounter4;
```

TRUTH TABLE

COUNT	OUTPUTS			
	Q_0	Q_1	Q_2	Q_3
0	L	L	L	L
1	H	L	L	L
2	L	H	L	L
3	H	H	L	L
4	L	L	H	L
5	H	L	H	L
6	L	H	H	L
7	H	H	H	L
8	L	L	L	H
9	H	L	L	H
10	L	H	L	H
11	H	H	L	H
12	L	L	H	H
13	H	L	H	H
14	L	H	H	H
15	H	H	H	H

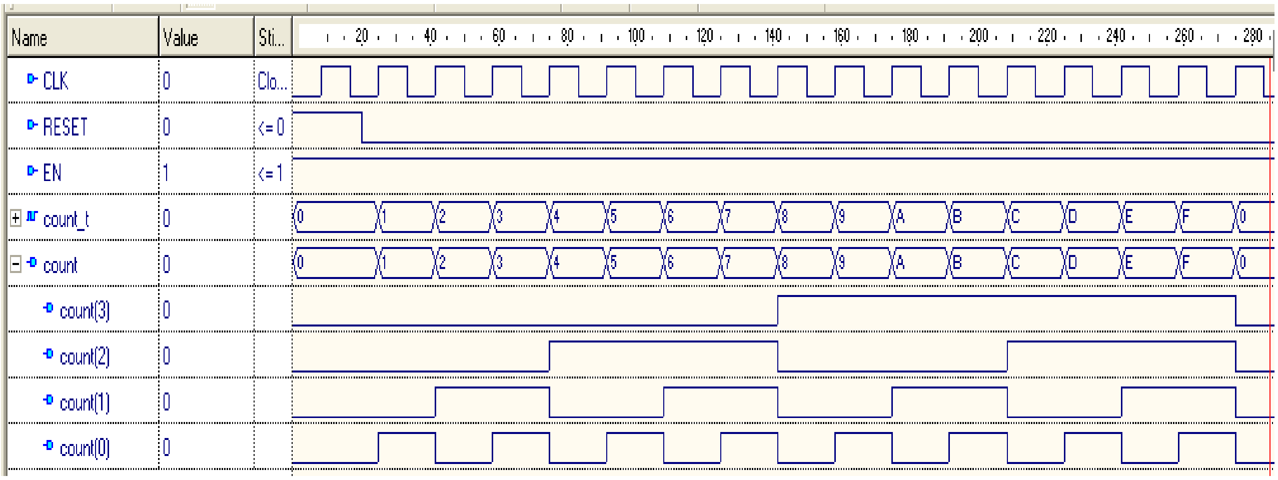
NOTE: H = High Voltage Level, L = Low Voltage Level

MODE SELECTION

RESET OUTPUTS		OUTPUTS			
MR1	MR2	Q_0	Q_1	Q_2	Q_3
H	H	L	L	L	L
L	H	Count	Count	Count	Count
H	L				
L	L				

NOTE: H = High Voltage Level, L = Low Voltage Level

OUTPUT WAVEFORM:



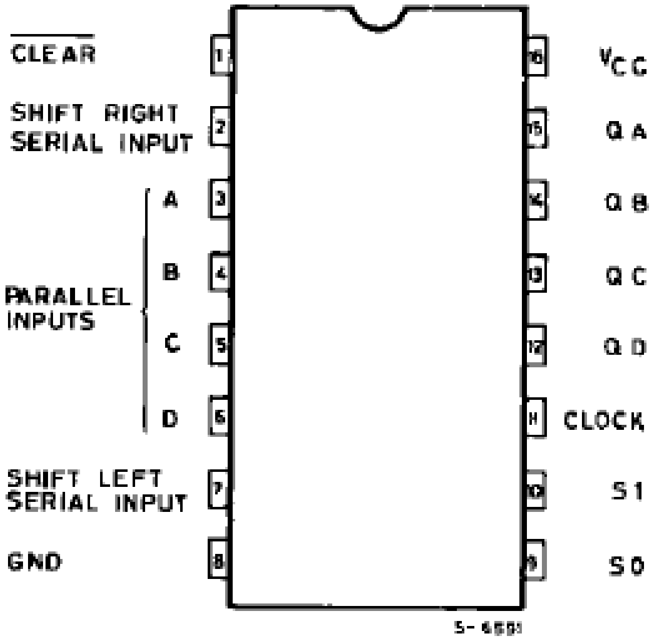
THEORY:

The 7493 is a 4 bit binary ripple counter consists of four master slave flip-flops internally connected to provide a divide-by-two section and a divide-by-eight section. Each section has a separate clock input (CP0 and CP1) to initiate state changes of the counter on the HIGH to LOW clock transition. State changes of the Qn outputs do not occur simultaneously because of internal ripple delays. A gated AND asynchronous master reset (MR1 and MR2) is provided which overrides both clocks and resets (clears) all flip-flops. Because the output from the divides by two section is not internally connected to the succeeding stages, the device may be operated in various counting modes.

PROCEDURE:

RESULT:

PIN DIAGRAM – 74194 (UNIVERSAL SHIFT REGISTER):



PIN DESCRIPTION:

PIN No	SYMBOL	NAME AND FUNCTION
1	$\overline{\text{CLEAR}}$	Asynchronous Reset Input (Active LOW)
2	SR	Serial Data Input (Shift Right)
3, 4, 5, 6	A to D	Parallel Data Input
7	SL	Serial Data Input (Shift Left)
9, 10	S0, S1	Mode Control Inputs
11	CLOCK	Clock Input (LOW to HIGH Edge-triggered)
15, 14, 13, 12	QA to QD	Paralle Outputs
8	GND	Ground (0V)
16	V_{CC}	Positive Supply Voltage

EXPERIMENT NO.10

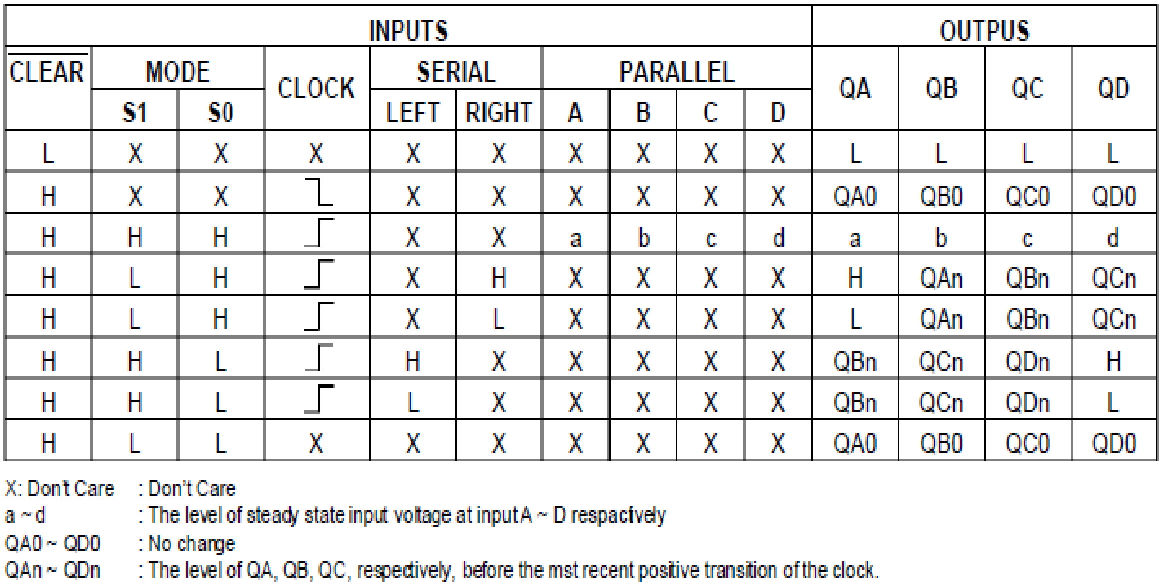
UNIVERSAL SHIFT REGISTER (74194/95)

AIM:

PROGRAM:

```
library ieee;
use ieee.std_logic_1164.all;
ENTITY ushift4 IS
PORT (clk, clrb, sl_in, sr_in : in bit;
mode : in bit_vector ( 1 downto 0 );
d : in bit_vector ( 3 downto 0 );
q : inout bit_vector (3 downto 0 ));
END ushift4;
ARCHITECTURE behav OF ushift4 IS
BEGIN
PROCESS (clk, clrb)
begin
-- Asynchronous, active-low Clear input:
if clrb = '0' then q <= "0000" ;
-- Rising edge-triggered D flip-flops:
elsif clk'event and clk = '1' then
case mode is
-- "Do Nothing" mode: retain current flip-flop outputs
when "00" => null;
-- Shift Right Serial Input mode:
when "01" =>
q <= (q srl 1) or (sr_in & "000") ;
-- Shift Left Serial Input mode:
when "10" =>
q <= (q sll 1) or ("000" & sl_in) ;
-- Parallel (Broadside) Load mode:
when "11" => q <= d ;
end case;
end if;
end process;
END behav;
```

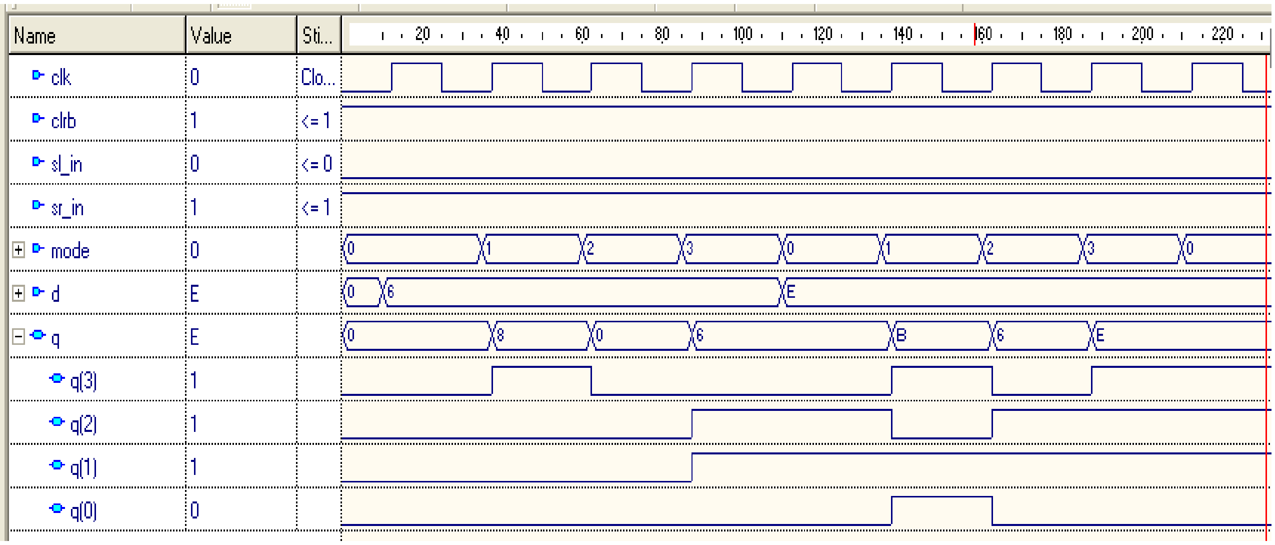
FUNCTION TABLE:



INPUTS										OUTPUS			
CLEAR	MODE		CLOCK	SERIAL		PARALLEL				QA	QB	QC	QD
	S1	S0		LEFT	RIGHT	A	B	C	D				
L	X	X	X	X	X	X	X	X	X	L	L	L	L
H	X	X	↓	X	X	X	X	X	X	QA0	QB0	QC0	QD0
H	H	H	↑	X	X	a	b	c	d	a	b	c	d
H	L	H	↑	X	H	X	X	X	X	H	QAn	QBn	QCn
H	L	H	↑	X	L	X	X	X	X	L	QAn	QBn	QCn
H	H	L	↑	H	X	X	X	X	X	QBn	QCn	QDn	H
H	H	L	↑	L	X	X	X	X	X	QBn	QCn	QDn	L
H	L	L	X	X	X	X	X	X	X	QA0	QB0	QC0	QD0

X: Dont Care : Don't Care
a ~ d : The level of steady state input voltage at input A ~ D respacively
QA0 ~ QD0 : No change
QAn ~ QDn : The level of QA, QB, QC, respectively, before the mst recent positive transition of the clock.

OUTPUT WAVEFORM:



Name	Value	Sti...
clk	0	Clo...
clrb	1	<= 1
sl_in	0	<= 0
sr_in	1	<= 1
mode	0	
d	E	
q	E	
q(3)	1	
q(2)	1	
q(1)	1	
q(0)	0	

THEORY:

The 74194 is a high speed 4 bit shift registers. This is called "universal" because is incorporates virtually all of the features a system designer may want in a shift register. The circuit provides parallel inputs, parallel outputs, right-shift and left-shift serial inputs, operating-mode-control inputs, and a direct overriding clear line. In the parallel load mode, the unit functions as a set of four D flip-flops. The two mode control bits SI and SO provide four modes of operation:

(SI, SO)=0 0: retain the present state (do nothing)
0 1: Shift Right (in the direction QA toward QD).
1 0: Shift Left (in the direction QD toward QA).
1 0: Parallel (Broadside)Load of A,B,C,D into QA,QB,QC,QD.

PROCEDURE:

RESULT: