LENDI INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Approved by A.I.C.T.E & Affiliated to JNTU, Kakinada)
Jonnada (Village), Denkada (Mandal), Vizianagaram Dist – 535005
Phone No. 08922-241111, 241112
E-Mail: lendi_2008@yahoo.com **Website:** www.lendi.org

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DIGITAL SYSTEM DESIGN & DICA LABORATORY OBSERVATION

Name:

Regd No: **Year&Sem:**

Academic Year:

LIST OF EXPERIMENTS

- **REALIZATION OF LOGIC GATES**
- **3 to 8 DECODER - 74138**
- **8 x 1 MULTIPLEXER-74151 AND 2 x 1 DE- MULTIPLEXER-74155**
- **4- Bit COMPARATOR- 7485**
- **D FLIP-FLOP -7474**
- **DECADE COUNTER -7490**
- **4-BIT BINARY COUNTER -7493**
- **SHIFT REGISTER -7495**
- **UNIVERSAL SHIFT REGISTER 74194/195**
- **RAM(16x4)-74189(READ AND WRITE OPERATIONS)**
- **ALU(74381)**

INTRODUCTION TO DSD&DICA LAB:

ECAD environments provide the tools for generating a physical representation of the integrated circuit from a high-level description. Traditionally, the designer starts with a schematic representation at a transistor or logical level, but due to the huge complexity of modern integrated circuits the trend is to use higher ones, such as in *Hardware Description Languages.*

The physical representation specifies how a particular part of the integrated circuit will be constructed. At the lowest level it is the photo-mask data necessary to perform the several processing steps. To simplify, a model based on material layers is used. These layers (forming the layout) are afterwards translated into photo-masks. The POLY1 layer, for example, specifies where the first layer of polysilicon will be deposited to form the gate of a transistor or a local interconnection. Sometimes these layers do not relate to a specific material but to material modifications as in implantation/diffusion steps.

The layout design rules are the link between the circuit designer and the technology constraints which the former must fulfill so that the fabricated integrated circuits may achieve a sufficiently high yield. ECAD tools always provide a *Design Rule Checker (DRC)* to inform the designer whether they are being correctly considered or not. Aside those rules the circuit designer is usually not concerned with technology related aspects, as in the ECAD environment these are encapsulated in simple parameters like the sheet resistance of one layer, or capacitance per unit area between two layers. This makes the actual thickness of the layers of little importance. For the active devices, circuit simulator models as the *SPICE* Level 3 model are employed. They must be calibrated for a given technology and afterwards need only relatively few layout-dependent parameters. For integrated circuits with millions of transistors, similarly to what happens in its description, a circuit-level simulation is not practical and ECAD tools provide logic-level and register level simulators as well. A very important aspect in future integrated circuit design is testability. ECAD frameworks include also tools to support the design for testability. In this lab we will Simulate the internal structure of the different Digital ICs (like D Flip-Flop 7474 ,Decade counter-7490 ,shift registers-7495 7 ,3-8 Decoder -74138 ,4 bit Comparator-7485 ,8 x 1 Multiplexer -74151 and 2x4 Demultiplexer-74155 RAM (16x4)-74189 (Read and Write operations) using VHDL / VERILOG and verify the operations of the Digital IC's (Hardware) in the Laboratory. The prototyping of the designed systems will be done by using FPGA Hardware boards like Spartan3, Spartan6 and Vertex5.

DESIGN PROCEDURE OF XILINX ISE 12.1

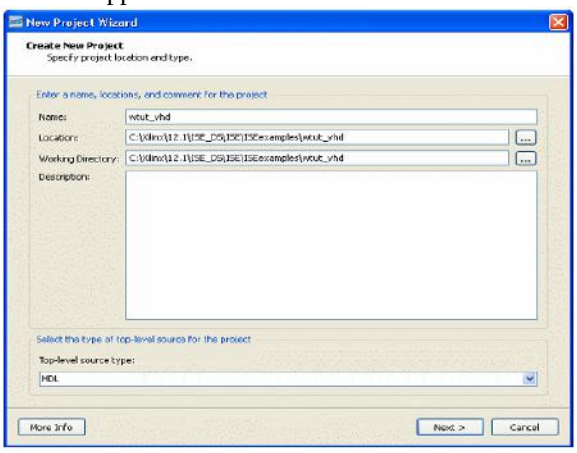
Starting the ISE Software To start the ISE software, double-click the ISE Project Navigator icon on your desktop, or select Start > All Programs > Xilinx ISE Design Suite 12.1 > ISE Design Tools > Project Navigator

: **Project Navigator Desktop Icon**

Creating a New Project

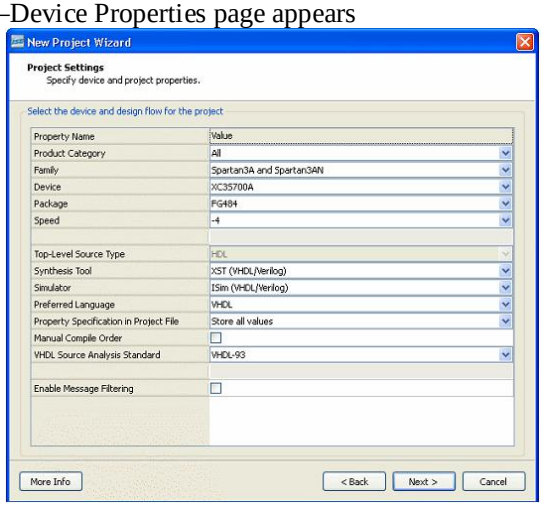
To create a new project using the New Project Wizard, do the following:

1. From Project Navigator, select **File > New Project**.
   The New Project Wizard appears.



2. In the Location field, browse to **c:\xilinx\12.1\ISE\ISEexamplesor** to the directory in which you installed the project.
3. In the Name field, enter **wtut_vhdor wtut_ver**.
4. Verify that HDL is selected as the Top-Level Source Type, and click Next.

The New Project Wizard—Device Properties page appears

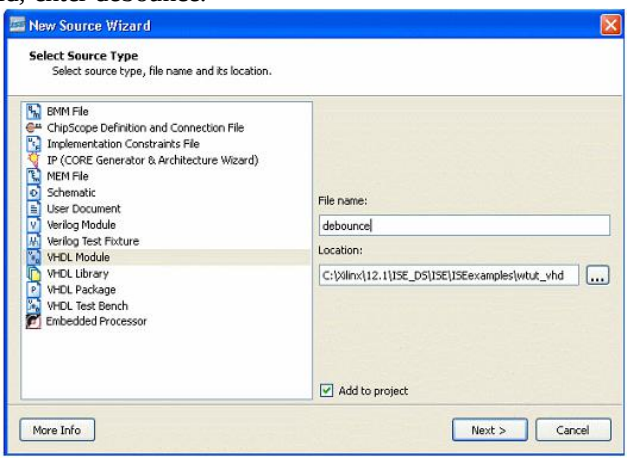


5. Select the following values in the **New Project Wizard—Device Properties page**:
   - ♦ Product Category: All
   - ♦ Family: Spartan3
   - ♦ Device: XC3S200

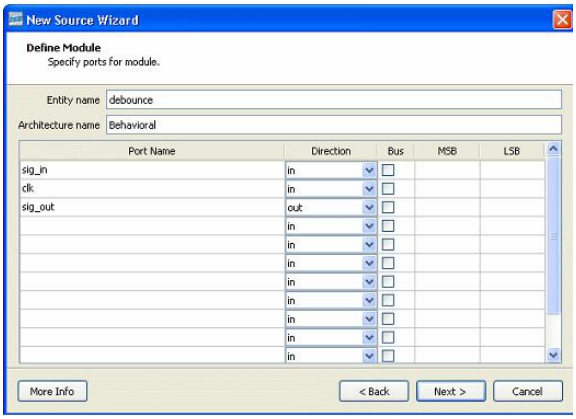
♦ Package: FT256
♦ Speed: -5
♦ Synthesis Tool: XST (VHDL/Verilog)
♦ Simulator: ISim (VHDL/Verilog)
♦ Preferred Language: VHDL or Verilog depending on preference. This will determine the default language for all processes that generate HDL files.

Other properties can be left at their default values.
6. Click next, and then Finish to complete the project creation.

To create the source file, do the following:
1. Select **Project > New Source**.
The New Source Wizard opens in which you specify the type of source you want to create.
2. In the Select Source Type page, select VHDL Module or Verilog Module.
3. In the File Name field, enter debounce.



4. Click Next.
5. In the Define Module page, enter two input ports named sig_in and clk and an output port named sig_out for the debounce component as follows:
a. In the first three Port Name fields, enter sig_in, clk and sig_out.
b. Set the Direction field to input for sig_in and clk and to output for sig_out.
c. Leave the Bus designation boxes unchecked.



6. Click Next to view a description of the module.
7. Click Finish to open the empty HDL file in the ISE Text Editor.
Following is an example VHDL file.

```
7   -- Module Name:    debounce - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25  ---- Uncomment the following library declaration if instantiating
26  ---- any Xilinx primitives in this code.
27  --library UNISIM;
28  --use UNISIM.VComponents.all;
29
30  entity debounce is
31      Port ( sig_in : in  STD_LOGIC;
32             clk : in  STD_LOGIC;
33             sig_out : out  STD_LOGIC);
34  end debounce;
35
36  architecture Behavioral of debounce is
37
38  begin
39
40
41  end Behavioral;
42
```

SIMULATION

The simulation processes in the ISE software enable you to run simulation on the design using ISim. To locate the ISim processes, do the following:

1. In the View pane of the Project Navigator Design panel, select Simulation, and select Behavioralfrom the drop-down list.
2. In the Hierarchy pane, select the test bench file (stopwatch_tb).
3. In the Processes pane, expand ISim Simulator to view the process hierarchy

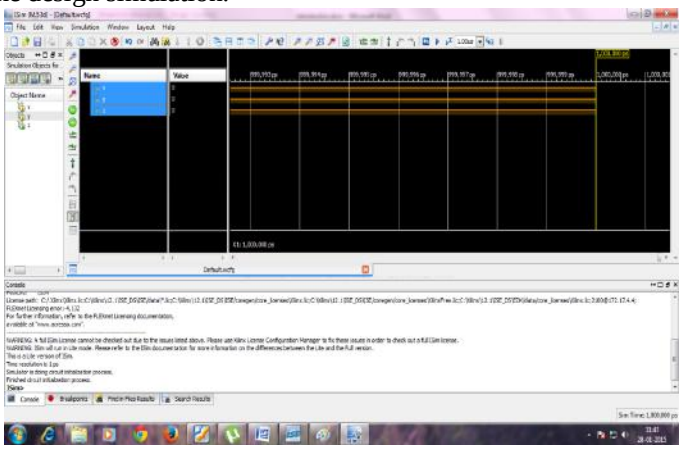
The following simulation processes are available:

♦ Check Syntax

This process checks for syntax errors in the test bench.

♦ Simulate Behavioral Model

This process starts the design simulation.



Force different values to signals by clicking on force option and run.

Design Implementation

Create **user constraint file** one as follows:

1. In the Hierarchy pane of the Project Navigator Design panel, select the top-level source file stopwatch.
2. Select Project > New Source.
3. Select Implementation Constraints File.
4. Enter stopwatch.ucf as the file name.

5. Click Next.
6. Click Finish.

To set the implementation properties for this tutorial, do the following:

1. In the View pane of the Project Navigator Design panel, select Implementation.
2. In the Hierarchy pane, select the stopwatchtop-level file.
3. In the Processes pane, right-click the Implement Designprocess, and select Process Properties.
4. Ensure that you have set the Property display level to Advanced.

This global setting enables you to see all available properties.

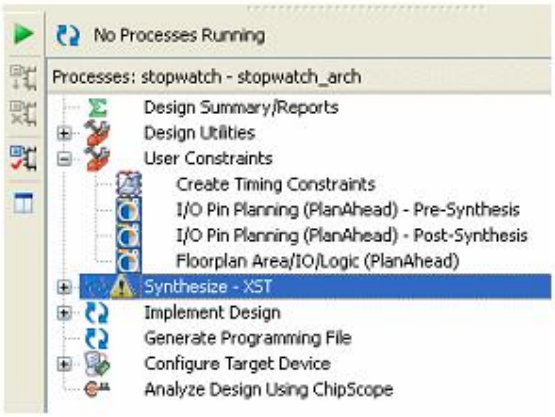
5. Click the Place & Route Propertiescategory.
6. Change the Place & Route Effort Level (Overall) to High.

Creating Timing Constraints

The User Constraints File (UCF) is a text file and can be edited directly with a text editor. To facilitate editing of this file, graphical tools are provided to create and edit constraints. The Constraints Editor and Plan Ahead software are graphical tools that enable you to enter timing and I/O and placement constraints.

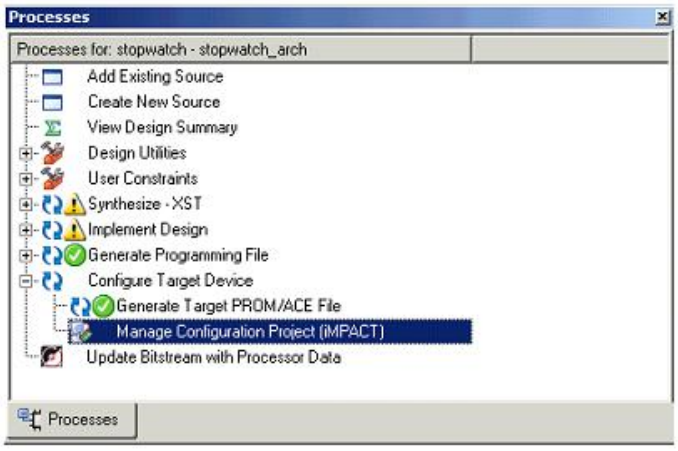
To launch the Constraints Editor, do the following:

1. In the Hierarchy pane of the Project Navigator Design panel, select the stopwatch module.
2. In the Processes pane, expand User Constraints, and double-click Create Timing Constraints.



Opening iMPACT from Project Navigator

To start iMPACT from Project Navigator, double-click Manage Configuration Project (iMPACT)in the Processes pane in the Design panel, as shown in the following figure.



Opening iMPACT Standalone

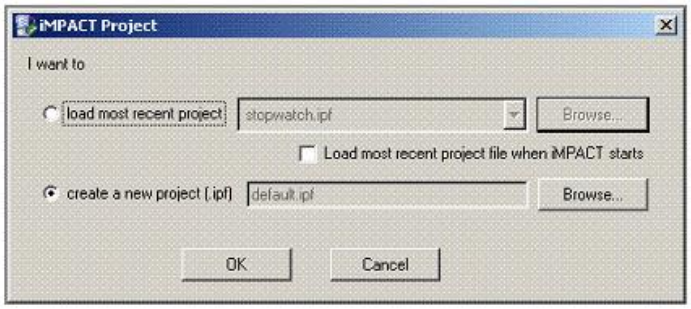
To open iMPACT without going through an ISE project, use one of the following methods:

• PC only: Click Start> All Programs> Xilinx ISE Design Suite 12.1 > ISE Design Tools > Tools> iMPACT.

• PC or Linux: Type impactat a command prompt.

Creating an iMPACT New Project File

When iMPACT is initially opened, the iMPACT Project dialog box opens. This dialog box enables you to load a recent project or to create a new project



To create a new project for this tutorial, do the following:

1. In the iMPACT Project dialog box, select create a new project (.ipf).
2. Click the Browsebutton.
3. Browse to the project directory, and then enter stopwatchin the File Name field.
4. Click Save.
5. Click OK

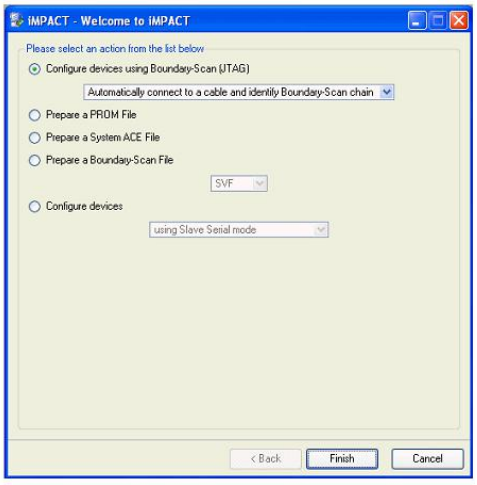
Using Boundary-Scan Configuration Mode

For this tutorial, you will be using the Boundary-Scan configuration mode. Boundary-Scan configuration mode enables you to perform Boundary-Scan operations on any chain comprising JTAG compliant devices. The chain can consist of both Xilinx and non-Xilinx devices; however, limited operations will be available for non-Xilinx devices. To perform operations, the cable must be connected and the JTAG pins, TDI, TCK, TMS, and TDO, must be connected from the cable to the board.

Specifying Boundary-Scan Configuration Mode

After opening iMPACT, you are prompted to specify the configuration mode and the device to program. To select Boundary-Scan Mode, do the following:

1. Select Configure Devices using Boundary-Scan (JTAG).
2. Ensure that Automatically connect to a cable and identify Boundary-Scan chain is selected.
3. Click Finish.



Assigning Configuration Files

After initializing a chain, the software prompts you for a configuration file

The configuration file is used to program the device. There are several types of configuration files:

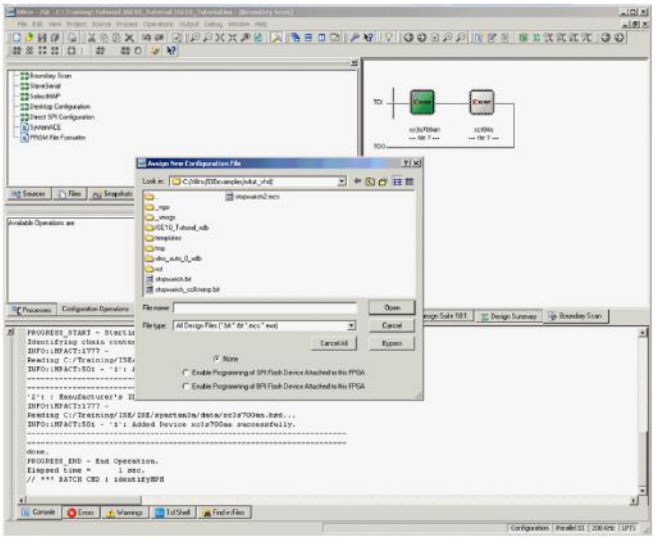
- Bitstream file (*.bit, *.rbt, *.isc) is used to configure an FPGA.
- JEDEC file (*.jed,*.isc) is used to configure a CPLD.
- PROM file (*.mcs, *.hex) is used to configure a PROM.

When the software prompts you to select a configuration file for the first device (XC3S700A), do the following:

1. Select the BIT file from your project working directory.
2. Click Open.

You should receive a warning stating thatthe startup clock has been changed to JtagClk.

3. Click OK.



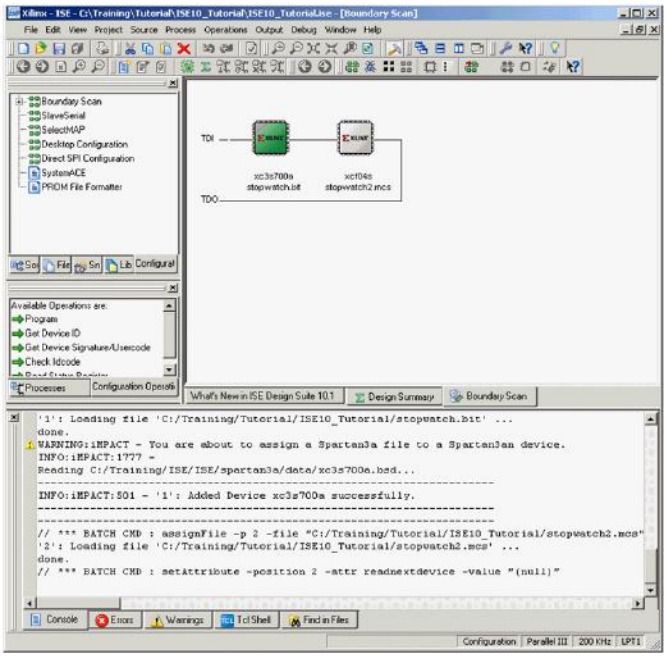
4. When the software prompts you to select a configuration file for the second device (XCF02S), select the MCS file from your project working directory.
5. Click Open.

Performing Boundary-Scan Operations

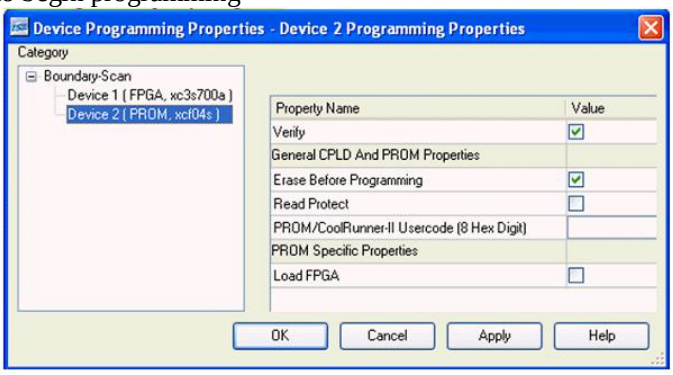
You can perform Boundary-Scan operations on one device at a time. The available Boundary-Scan operations vary based on the device and the configuration file that was applied to the device. To see a list of the available options, right-click on any device in the chain. This brings up a window with all of the available options. When you select a device and perform an operation on that device, all other devices in the chain are automatically placed in BYPASS or HIGHZ, depending on your iMPACT Preferences setting. For more information about Preferences, see "Editing Preferences."

To perform an operation, right-click on a device and select one of the options. In this section, you will retrieve the device ID and run the programming option to verify the first device as follows:

1. Right-click on the XC3S700A device, and select Get Device ID.

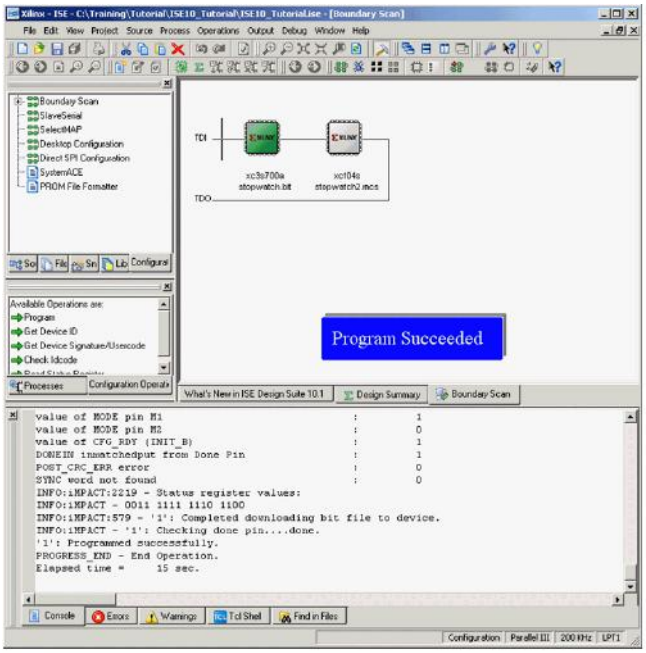


2. Right-click on the XC3S700A device, and select Set Programming Properties.
The Device Programming Properties dialog box opens.
3. Select the Verify option.
The Verify option enables the device to beread back and compared to the BIT file using the MSK file that was created earlier.
4. Click OKto begin programming



5. Right-click on the XC3S700Adevice again, and select Program.
The Programming operation begins and an operation status window appears. At the same time, the log window reports all of the operations being performed.

When the Program operation completes, a large blue message appears showing that programming was successful, as shown in the following figure. This message disappears after a few seconds.


Boundary Scan
SlaveSerial
Selectmap
Desktop Configuration
Direct SPI Configuration
SystemACE
PROM File Formatter
TDI
TDO
xc3s700a
stopwatch.bit
xc104a
stopwatch2.mcs
Program Succeeded
Available Operations are:
Program
Get Device ID
Get Device Signature/Usercode
Check Idcode
What's New in ISE Design Suite 10.1
Design Summary
Boundary Scan
value of MODE pin M1 : 1
value of MODE pin M2 : 0
value of CFG_RDY (INIT_B) : 1
DONEIN inmatchedput from Done Pin : 1
POST_CRC_ERR error : 0
SYNC word not fouad : 0
INFO:iMPACT:2219 - Status register values:
INFO:iMPACT - 0011 1111 1110 1100
INFO:iMPACT:579 - '1': Completed downloading bit file to device.
INFO:iMPACT - '1': Checking done pin....done.
'1': Programmed successfully.
PROGRESS_END - End Operation.
Elapsed time = 15 sec.
Console
Errors
Warnings
Tcl Shell
Find in Files
Configuration
Parallel III
200 kHz
LPT1

Exp.No: **Date:**

LOGIC GATES

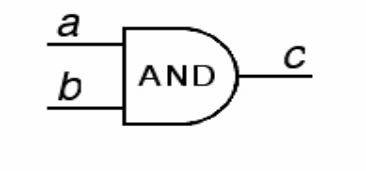
AIM: To simulate the Logic Gates.

APPARATUS:
1. Xilinx 12.1 tool
2. ISim simulator
3. XST synthesizer
4. FPGA Board-Spartan3

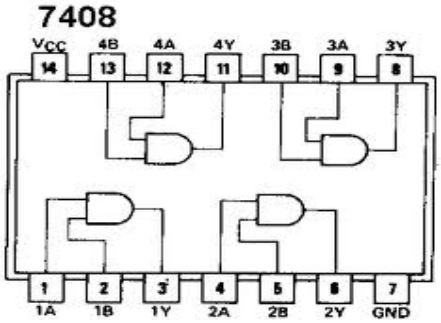
AND gate (IC 7408) Definition:

A Logic AND Gate **is** a type of digital logic gate that has an output which is normally at logic level "0" and only goes "HIGH" to a logic level "1" when ALL of its inputs are at logic level "1". The output of a Logic AND Gate only returns "LOW" again when ANY of its inputs are at a logic level "0". The logic or Boolean expression given for a logic AND gate is that for *Logical Multiplication* which is denoted by a single dot or full stop symbol, (.) giving us the Boolean expression of: A.B = Q.

Logic Diagram:



Pin Diagram:



Truth Table:

AND gate

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

Program:

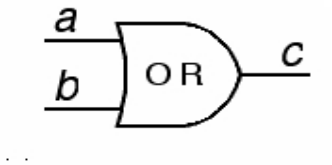
VHDL CODE:

```
        Library IEEE;
use IEEE.std_logic_1164.all;
entity AND2 is
        port(
        x : in STD_LOGIC;
        y : in STD_LOGIC;
        z : out STD_LOGIC
        );
end AND2;

--Dataflow model
        architecture behav1 of AND2 is
        begin
        Z<= x and y; --Signal Assignment Statement
        end behav1;

-- Behavioral model
        architecture behav2 of AND2 is
        begin
        process (x, y)
        begin
        if (x='1' and y='1') then -- Compare with truth table
        Z <= '1';
        else
        Z <= '0';
        end if;
        end process;
end behav2;
```

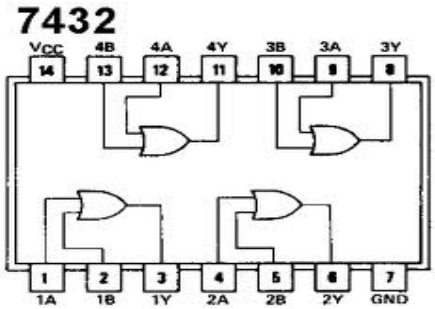
OR gate (IC 7432) Definition:

A Logic OR Gate or Inclusive-OR gate is a type of digital logic gate that has an output which is normally at logic level "0" and only goes "HIGH" to a logic level "1" when one or more of its inputs are at logic level "1". The output, Q of a Logic OR Gate only returns "LOW" again when ALL of its inputs are at a logic level "0". The logic or Boolean expression given for a logic OR gate is that for *Logical Addition* which is denoted by a plus sign, (+) giving us the Boolean expression of: A+B = Q.

Logic Diagram:



Pin Diagram:



Truth Table:

OR gate

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

Program:

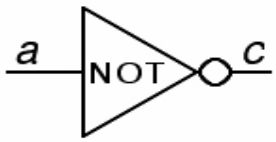
VHDL CODE:

```
Library IEEE;
use IEEE.std_logic_1164.all;
entity OR2 is
        port(
        x : in STD_LOGIC;
        y : in STD_LOGIC;
        z : out STD_LOGIC
        );
        end OR2;
--Dataflow model
        architecture behav1 of OR2 is
        begin
        Z <= x or y; --Signal Assignment Statement
end behav1;

-- Behavioral model
        architecture behav2 of OR2 is
        begin
        process (x, y)
        begin
        if (x='0' and y='0') then -- Compare with truth table
        Z <= '0';
        else
        Z<= '1';
        end if;
        end process;
end behav2;
```

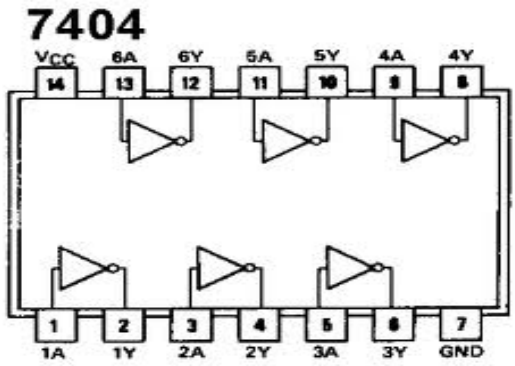
NOT gate (IC 7404) Definition:

The digital Logic NOT Gate is the most basic of all the logical gates and is sometimes referred to as an Inverting Buffer or simply a Digital Inverter. It is a single input device which has an output level that is normally at logic level "1" and goes "LOW" to a logic level "0" when its single input is at logic level "1", in other words it "inverts" (complements) its input signal. The output from a NOT gate only returns "HIGH" again when its input is at logic level "0" giving us the Boolean expression of: A = Q.

Logic Diagram:



Pin Diagram:



Truth Table:

INPUT (A)	OUTPUT (C)
0	1
1	0

Program:

VHDL CODE:

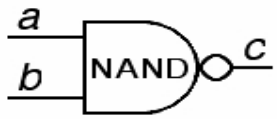
```
Library IEEE;
use IEEE.std_logic_1164.all;
entity not1 is
port(
X: in STD_LOGIC;
Z: out STD_LOGIC
);
end not1;
--Dataflow model
architecture behav1 of not1 is
begin
```

```
        Z<= not X; --Signal Assignment Statement
        end behav1;
-- Behavioral model
         architecture behav2 of not1 is
         begin
         process (X)
         begin
         if (x='0') then -- Compare with truth table
         Z <= '1';
         else
        Z<= '0';
          end if;
          end process;
end behav2;
```

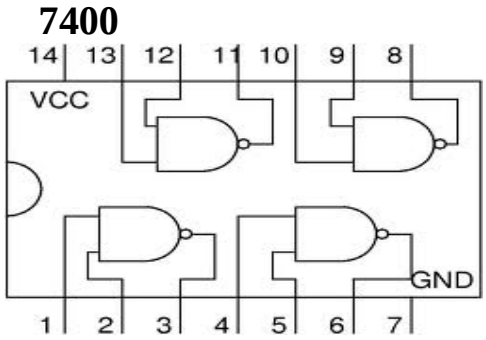
NAND gate (IC 7400) Definition:

The Logic NAND Gate is a combination of the digital logic AND gate with that of an inverter or NOT gate connected together in series. The NAND (Not - AND) gate has an output that is normally at logic level "1" and only goes "LOW" to logic level "0" when ALL of its inputs are at logic level "1". The Logic NAND Gate is the reverse or "*Complementary*" form of the AND gate we have seen previously.

Logic Diagram:



Pin Diagram:



Truth Table:

NAND gate

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

Program:

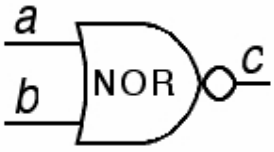
VHDL CODE:

```
        Library IEEE;
use IEEE.std_logic_1164.all;
entity nand2 is
        port(
        x : in STD_LOGIC;
        y : in STD_LOGIC;
        z : out STD_LOGIC
        );
        end nand2;
--Dataflow model
        architecture behav1 of nand2 is
        begin
        z<= x nand y; --Signal Assignment Statement
        end behav1;
-- Behavioral model
        architecture behav2 of nand2 is
        begin
Process (x, y)
Begin
        If (x='1' and y='1') then -- Compare with truth table
Z <= '0';
else
Z <= '1';
end if;
end process;
end behav2;
```

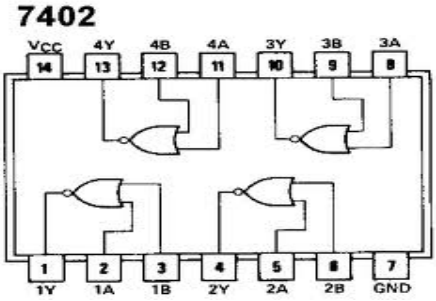
NOR gate (IC 7402) Definition:

The Logic NOR Gate or Inclusive-NOR gate is a combination of the digital logic OR gate with that of an inverter or NOT gate connected together in series. The NOR (Not - OR) gate has an output that is normally at logic level "1" and only goes "LOW" to logic level "0" when ANY of its inputs are at logic level "1". The Logic NOR Gate is the reverse or "*Complementary*" form of the OR gate we have seen previously.

Logic Diagram:



Pin Diagram:



Truth Table:

NOR gate

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

Program:

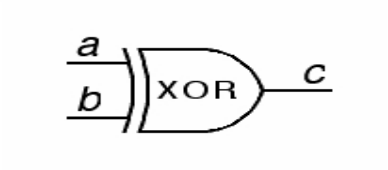
VHDL CODE:

```
Library IEEE;
use IEEE.std_logic_1164.all;
        entity nor2 is
        Port (
        X: in STD_LOGIC;
        Y: in STD_LOGIC;
        Z: out STD_LOGIC
        );
end nor2;

--Dataflow model
        architecture behav1 of nor2 is
        begin
        Z<= x nor y; --Signal Assignment Statement
        end behav1;

-- Behavioral model
architecture behav2 of nor2 is
begin
process (x, y)
begin
        If (x='0' and y='0') then -- Compare with truth table
Z <= '1';
else
Z <= '0';
end if;
end process;
end behav2;
```

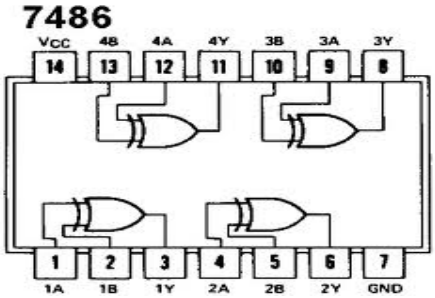
XOR gate (IC 7486) Definition:

Previously, we have seen that for a 2-input OR gate, if A = "1", OR B = "1", then the output from the digital gate must also be at a logic level "1" and because of this, this type of logic gate is known as an exclusive-OR function. If the two inputs are equal, then the output will be "0".

Logic Diagram:



Pin Diagram:



Truth Table:

EX-OR gate

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

Program:

VHDL CODE:

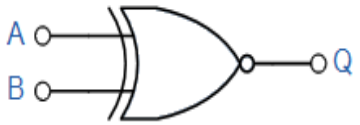
```
        Library IEEE;
use IEEE.std_logic_1164.all;
entity xor2 is
        Port (
        X: in STD_LOGIC;
        Y: in STD_LOGIC;
        Z: out STD_LOGIC
        );
        end xor2;
--Dataflow model
        architecture behav1 of xor2 is
        begin
        Z<= x xor y; --Signal Assignment Statement
end behav1;
```

```
-- Behavioral model
        architecture behav2 of xor2 is
        begin
process (x, y)
begin
        If (x/=y) then -- Compare with truth table
Z <= '1';
else
Z<= '0';
end if;
end process;
end behav2;
```

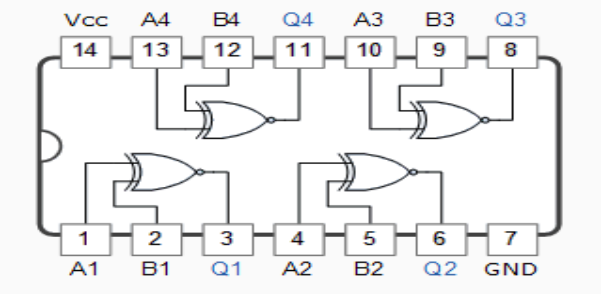
XNOR gate (IC 74266) Definition:

The Exclusive-NOR Gate function or Ex-NOR for short, is a digital logic gate that is the reverse or complementary form of the Exclusive-OR. The Exclusive-NOR gate is a combination of the Exclusive-OR gate and the NOT gate. It has an output that is normally at logic level "0" when ANY of its inputs are at logic level "1".

Logic Diagram:



Pin Diagram:



Truth Table:

EX-NOR gate

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

Program:

VHDL CODE:

```
        Library IEEE;
use IEEE.std_logic_1164.all;
entity xnor2 is
        Port (
        X: in STD_LOGIC;
        Y: in STD_LOGIC;
        Z: out STD_LOGIC
        );
        end xnor2;

--Dataflow model
        architecture behav1 of xnor2 is
        begin
        Z<= x xnor y; --Signal Assignment Statement
        end behav1;
-- Behavioral model
        architecture behav2 of xnor2 is
        begin
process (x, y)
begin
If (x=y) then -- Compare with truth table
Z <= '1';
else
Z<= '0';
end if;
end process;
end behav2;
```

Result & Analysis:

Synthesis Report:

Power Analysis:

Timing Analysis:

VIVA QUESTIONS:

1. Implement the following function using VHDL coding. (Try to minimize if you can).
   F(A,B,C,D)=(A′+B+C) . (A+B′+D′). (B+C′+D′) . (A+B+C+D)
2. What will be the no. of rows in the truth table of N variables?
3. What are the advantages of VHDL?
4. Design Ex-OR gate using behavioral model?
5. Implement the following function using VHDL code
   f=AB+CD.
6. What are the differences between half adder and full adder?
7. What are the advantages of minimizing the logical expressions?
8. What does a combinational circuit mean?
9. Implement the half adder using VHDL code?
10. Implement the full adder using two half adders and write VHDL program in structural model?

GRAPH SHEET

Exp.No: **Date:**

3 to 8 DECODER 74138

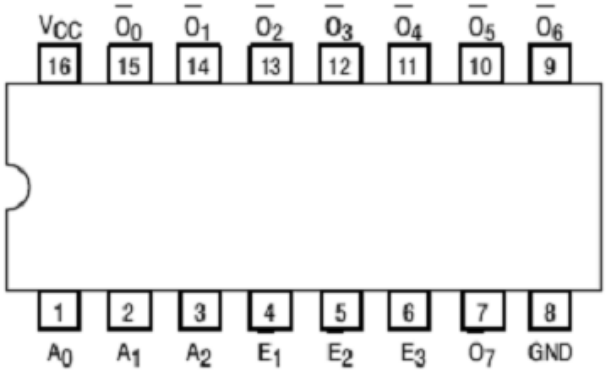
AIM: To simulate and synthesize 3 to 8 decoder (74138).

APPARATUS:
1. Xilinx 12.1 tool
2. ISim simulator
3. XST synthesizer
4. FPGA Board-Spartan3

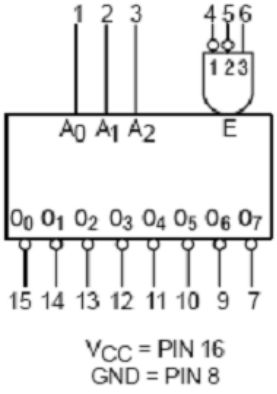
THEORY:

A decoder is a multiple-input, multiple output logic circuit that converts coded inputs into coded outputs, where the ipnput and output codes are different. The input code generally has fewer bits than the output code.

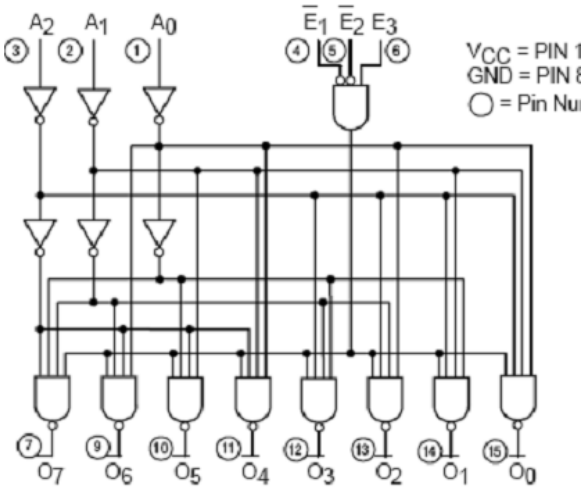
PIN DIAGRAM:



LOGIC SYMBOL:



LOGIC DIAGRAM:



TRUTH TABLE:

INPUTS						OUTPUTS							
$\overline{E_1}$	$\overline{E_2}$	E_3	A_0	A_1	A_2	$\overline{O_0}$	$\overline{O_1}$	$\overline{O_2}$	$\overline{O_3}$	$\overline{O_4}$	$\overline{O_5}$	$\overline{O_6}$	$\overline{O_7}$
H	X	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	L	X	X	X	H	H	H	H	H	H	H	H
L	L	H	L	L	L	L	H	H	H	H	H	H	H
L	L	H	H	L	L	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	L	H	H	H	H	H
L	L	H	H	H	L	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	L	H	H
L	L	H	L	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	L

PROGRAM:

***DATAFLOW MODEL*:**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity decoder_d is
   Port ( g1,g2a_l,g2b_l : in  STD_LOGIC;
          a : in  STD_LOGIC_VECTOR (2 downto 0);
          y_l : out  STD_LOGIC_VECTOR (0 to 7));
end decoder_d;
architecture dataflow of decoder_d is
signal y: std_logic_vector (0 to 7);
begin
with a select y<=
           "01111111" when "000",
           "10111111" when "001",
           "11011111" when "010",
           "11101111" when "011",
           "11110111" when "100",
           "11111011" when "101",
           "11111101" when "110",
           "11111110" when "111",
           "11111111" when others;
```

```
y_l <= y when (g1 and (not g2a_l) and (not g2b_l)) = '1' else "11111111";
end dataflow;
```

BEHAVIORAL MODEL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity decoder_b is
   Port ( g1,g2a_l,g2b_l : in  STD_LOGIC;
          a : in  STD_LOGIC_VECTOR (2 downto 0);
          y_l : out  STD_LOGIC_VECTOR (0 to 7));
end decoder_b;
architecture Behavioral of decoder_b is
signal y : std_logic_vector (0 to 7);
begin
process (g1,g2a_l,g2b_l,y,a)
begin
case a is
           when "000" => y <= "01111111";
           when "001" => y <= "10111111";
           when "010" => y <= "11011111";
           when "011" => y <= "11101111";
           when "100" => y <= "11110111";
           when "101" => y <= "11111011";
           when "110" => y <= "11111101";
           when "111" => y <= "11111110";
           when others => y <= "11111111";
end case;
if (g1 and (not g2a_l) and (not g2b_l)) = '1' then y_l <= y;
else y_l<= "11111111";
end if;
end process;
end Behavioral;
```

Result & Analysis:

Synthesis Report:

Power Analysis:

Timing Analysis:

VIVA QUESTIONS :

1. Write the behavioral code for the IC 74x138.
2. Write the VHDL code for the IC 74x138 using CASE statement.
3. Write the VHDL code for the IC 74x138 using WITH statement.
4. Write the VHDL code for the IC 74x138 using WHEN--ELSE statement.
5. Write the structural program for IC 74x138.
6. What does priority encoder mean?
7. How many decoders are needed to construct 4X16 decoder?
8. What is the difference between decoder and encoder?
9. Write the syntax for exit statement?
10. Explain briefly about next statement?
11. How to specify the delay in VHDL program?
12. Write the syntax for component declaration.

GRAPH SHEET

Exp.No: **Date:**

8 x 1 MULTIPLEXER 74151

AIM: To simulate and synthesize 8 x 1 multiplexer (74151).

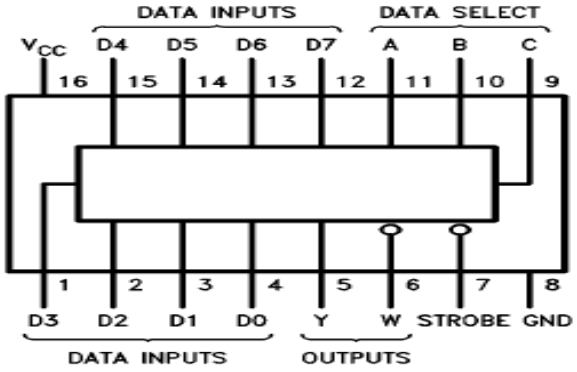
APPARATUS:

1. Xilinx 12.1 tool
2. ISim simulator
3. XST synthesizer
4. FPGA Board-Spartan3

THEORY:

The 151 selects one-of-eight data sources. The 151A have a strobe input which must be at a low logic level to enable these devices. A high level at the strobe forces the W output high and the Y output (as applicable) low.

PIN DIAGRAM:



TRUTH TABLE:

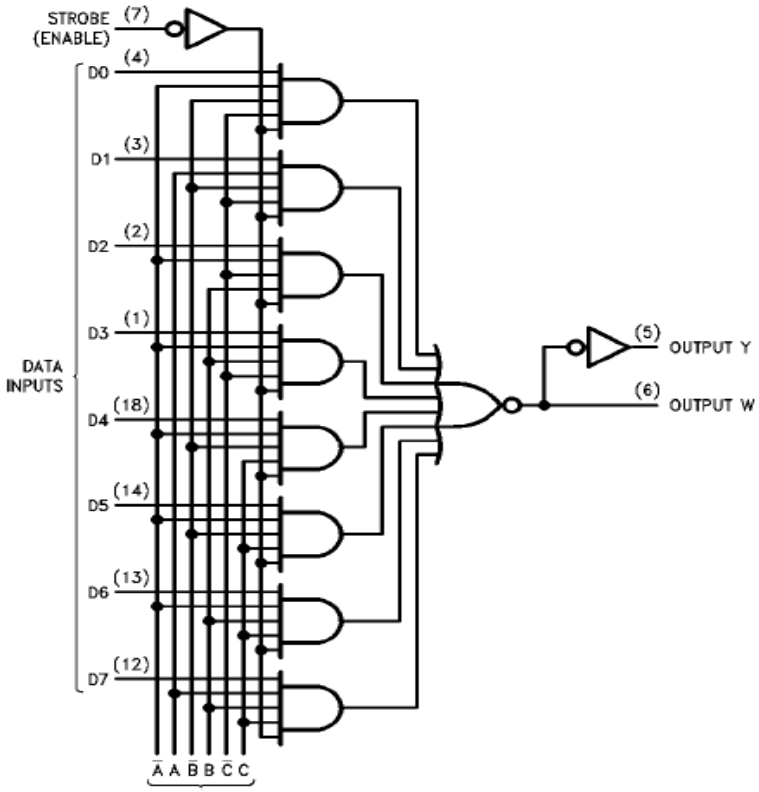
54151A/75151A

Inputs				Outputs	
Select			Strobe S	Y	W
C	B	A			
X	X	X	H	L	H
L	L	L	L	D0	$\overline{D0}$
L	L	H	L	D1	$\overline{D1}$
L	H	L	L	D2	$\overline{D2}$
L	H	H	L	D3	$\overline{D3}$
H	L	L	L	D4	$\overline{D4}$
H	L	H	L	D5	$\overline{D5}$
H	H	L	L	D6	$\overline{D6}$
H	H	H	L	D7	$\overline{D7}$

H = High Level, L = Low Level, X = Don't Care

D0, D1 . . . D7 = the level of the respective D input

LOGIC DIAGRAM:



PROGRAM:

DATAFLOW MODEL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
entity multiplexer_d is
   Port ( en_L : in  STD_LOGIC;
          s : in  STD_LOGIC_VECTOR (2 downto 0);
          d : in  STD_LOGIC_VECTOR (0 to 7);
          y : inout  STD_LOGIC;
          y_l : out  STD_LOGIC);
end multiplexer_d;
architecture dataflow of multiplexer_d is
signal x : std_logic;
begin
with s select x <=
           d(0) when "000",
           d(1) when "001",
           d(2) when "010",
           d(3) when "011",
           d(4) when "100",
           d(5) when "101",
           d(6) when "110",
           d(7) when "111",
           'U' when others;
```

```
y <= x when en_l = '0' else 'U';
y_l <= not y;
end dataflow;
```

BEHAVIORAL MODEL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity multiplexer_b is
   Port ( en_l : in  STD_LOGIC;
          s : in  STD_LOGIC_VECTOR (2 downto 0);
          d : in  STD_LOGIC_VECTOR (0 to 7);
          y: inout STD_LOGIC;
y_l : out  STD_LOGIC);
end multiplexer_b;
architecture Behavioral of multiplexer_b is
signal x : std_logic;
begin
process (en_l,s,d,x,y)
begin
case s is
           when "000" => x <= d(0);
           when "001" => x <= d(1);
           when "010" => x <= d(2);
           when "011" => x <= d(3);
           when "100" => x <= d(4);
           when "101" => x <= d(5);
           when "110" => x <= d(6);
           when "111" => x <= d(7);
           when others => x <= 'U';
end case;
if (en_l = '0') then y<= x; y_l<=not x;
else y<= 'U';y_l<='U';
end if;
end process;
end Behavioral;
```

STRUCTURAL MODELING:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity multiplexer2_1 is
    port(
       a : in STD_LOGIC;
       b : in STD_LOGIC;
       sel : in STD_LOGIC;
       dout : out STD_LOGIC
       );
end multiplexer2_1;

architecture multiplexer2_1_arc of multiplexer2_1 is
component and2 is
   port (a : in STD_LOGIC;
         b : in STD_LOGIC;
```

```
        dout : out STD_LOGIC);
end component and2;

component or2 is
    port (a : in STD_LOGIC;
        b : in STD_LOGIC;
        dout : out STD_LOGIC );
end component or2;

component not1 is
    port (a : in STD_LOGIC;
        dout : out STD_LOGIC );
end component not1;

signal m : std_logic;
signal n : std_logic;
signal o : std_logic;

begin

    u0 : and2 port map (a,m,n);
    u1 : and2 port map (sel,b,o);
    u2 : or2 port map (n,o,dout);
    u3 : not1 port map (sel,m);

end multiplexer2_1_arc;
```

AND gate design:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity and2 is
    port (a : in STD_LOGIC;
        b : in STD_LOGIC;
        dout : out STD_LOGIC
        );
end and2;

architecture and2_arc of and2 is
begin

    dout <= a and b;

end and2_arc;
```

OR gate design:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity or2 is
    port (a : in STD_LOGIC;
        b : in STD_LOGIC;
        dout : out STD_LOGIC
        );
```

```
end or2;

architecture or2_arc of or2 is
begin
      dout <= a or b;
   end or2_arc;
```

Not Gate Design:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity not1 is
   port (a : in STD_LOGIC;
        dout : out STD_LOGIC
        );
end not1;

architecture not1_arc of not1 is
begin

   dout <= not a ;
   end not1_arc;
```

Result & Analysis:

Synthesis Report:

Power Analysis:

Timing Analysis:

VIVA QUESTIONS:

1. Write the behavioral code for the IC 74x151.
2. Write the VHDL code for the IC 74x151 using IF statement.
3. Write the VHDL code for the IC 74x151 using WITH statement.
4. Write the VHDL code for the IC 74x151 using WHEN--ELSE statement.
5. Write the structural program for IC 74x151.
6. What is meant by multiplexer?
7. What does demultiplexer mean?
8. How many 8X1 multiplexers are needed to construct 16X1 multiplexer?
9. Compare decoder with demultiplexer?
10. Design a full adder using 8X1 multiplexer?
11. What are the two kinds of subprograms?
12. What are the difference between function and procedure?

GRAPH SHEET

Exp.No: **Date:**

2x1 DEMULTIPLEXER-74155

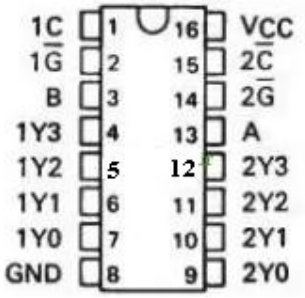
AIM: To simulate and synthesize 2:4 Demultiplexer-74155

APPARATUS:
1. Xilinx 12.1 tool
2. ISim simulator
3. XST synthesizer
4. FPGA Board-Spartan3

THEORY:

In digital electronics, a decoder can take the form of a multiple-input, multiple-output logic circuit that converts coded inputs into coded outputs, where the input and output codes are different e.g. n-to-2n , binary-coded decimal decoders. Decoding is necessary in applications such as data multiplexing, 7 segment display and memory address decoding. Let us suppose that a logic network has 2 inputs A and B. They will give rise to 4 states A, A', B, B' . The truth table for this decoder is shown below:

PIN DIAGRAM:



TRUTH TABLE:

S_1	S_0	E	O_0	O_1	O_2	O_3
×	×	0	0	0	0	0
0	0	1	1	0	0	0
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	1

Table 1: Truth Table of 2:4 decoder

LOGIC DIAGRAM:

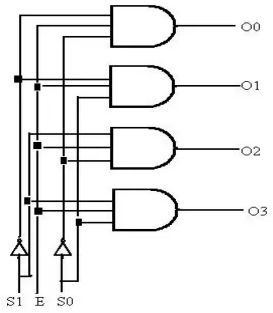


Fig 1: Logic Diagram of 2:4 decoder

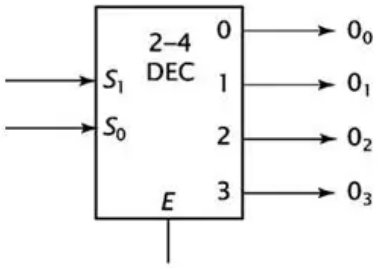


Fig 2: Representation of 2:4 decoder

VHDL PROGRAM FOR 2:4 DECODER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity demultiplexer2_4 is
   Port ( A : in  STD_LOGIC_VECTOR (1 downto 0);
          S : in  STD_LOGIC;
          F,G,H,J : out  STD_LOGIC);
end demultiplexer2_4;
architecture Behavioral of demultiplexer2_4 is
begin
PROCESS (A,S)
BEGIN
CASE A IS WHEN "00"=>F<=S;
                        WHEN "01"=>G<=S;
                        WHEN "10"=>H<=S;
                        WHEN "11"=>J<=S;
                        WHEN OTHERS=>NULL;
END CASE;
END PROCESS;
end Behavioral;
```

VHDL PROGRAM FOR 2x1 DEMULTIPLEXER:

(A)Data Flow Modelling

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY mux2to1 IS
        PORT ( w0, w1, s : IN STD LOGIC ;
        f : OUT STD LOGIC ) ;
END mux2to1 ;
ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
        WITH s SELECT
                f <= w0 WHEN '0',
                   w1 WHEN OTHERS ;
END Behavior ;
```

Behavioral Modelling:

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY mux2to1 IS
        PORT ( w0, w1, s : IN STD LOGIC ;
        f : OUT STD LOGIC ) ;
END mux2to1 ;
ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
PROCESS ( w0, w1, s )
BEGIN
IF s ='0' THEN
        f < =w0 ;
ELSE
        f <= w1 ;
END IF ;
END PROCESS ;
END Behavior ;
```

Result & Analysis:

Synthesis Report:

Power Analysis:

Timing Analysis:

VIVA QUESTIONS

Q.1 Explain about Demultiplexer?
Q.2 Draw a circuit diagram of 1: 4 Demultiplexer?
Q.3 Make a logic diagram of 1: 4 Demultiplexer?
Q.4 What is the application of Demultiplexer?
Q.5 What is the difference between Multiplexer and Demultiplexer?

GRAPH SHEET

Exp.No: **Date:**

4- Bit COMPARATOR 7485

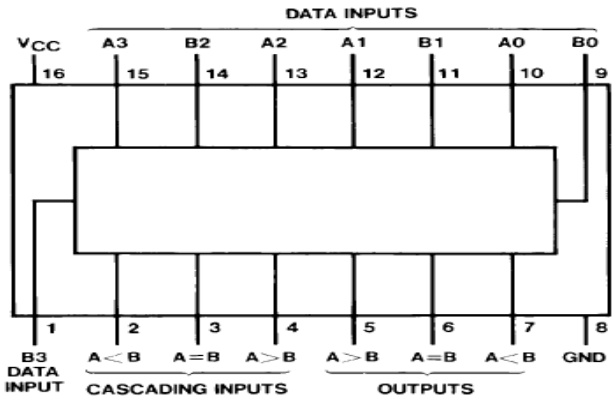
AIM: To simulate and synthesize 4-bit comparator (7485).

APPARATUS:
1. Xilinx 12.1 tool
2. ISim simulator
3. XST synthesizer
4. FPGA Board-Spartan3

THEORY:

These 4-bit magnitude comparators perform comparison of straight binary or BCD codes. Three fully-decoded decisions about two, 4-bit words (A, B) are made and are externally available at three outputs. These devices are fully expandable to any number of bits without external gates. Words of greater length may be compared by connecting comparators in cascade. The A < B, A > B, and A = B outputs of a stage handling less-significant bits are connected to the corresponding inputs of the next stage handling more-significant bits. The stage handling the least-significant bits must have a high-level voltage applied to the A = B input.

PIN DIAGRAM:

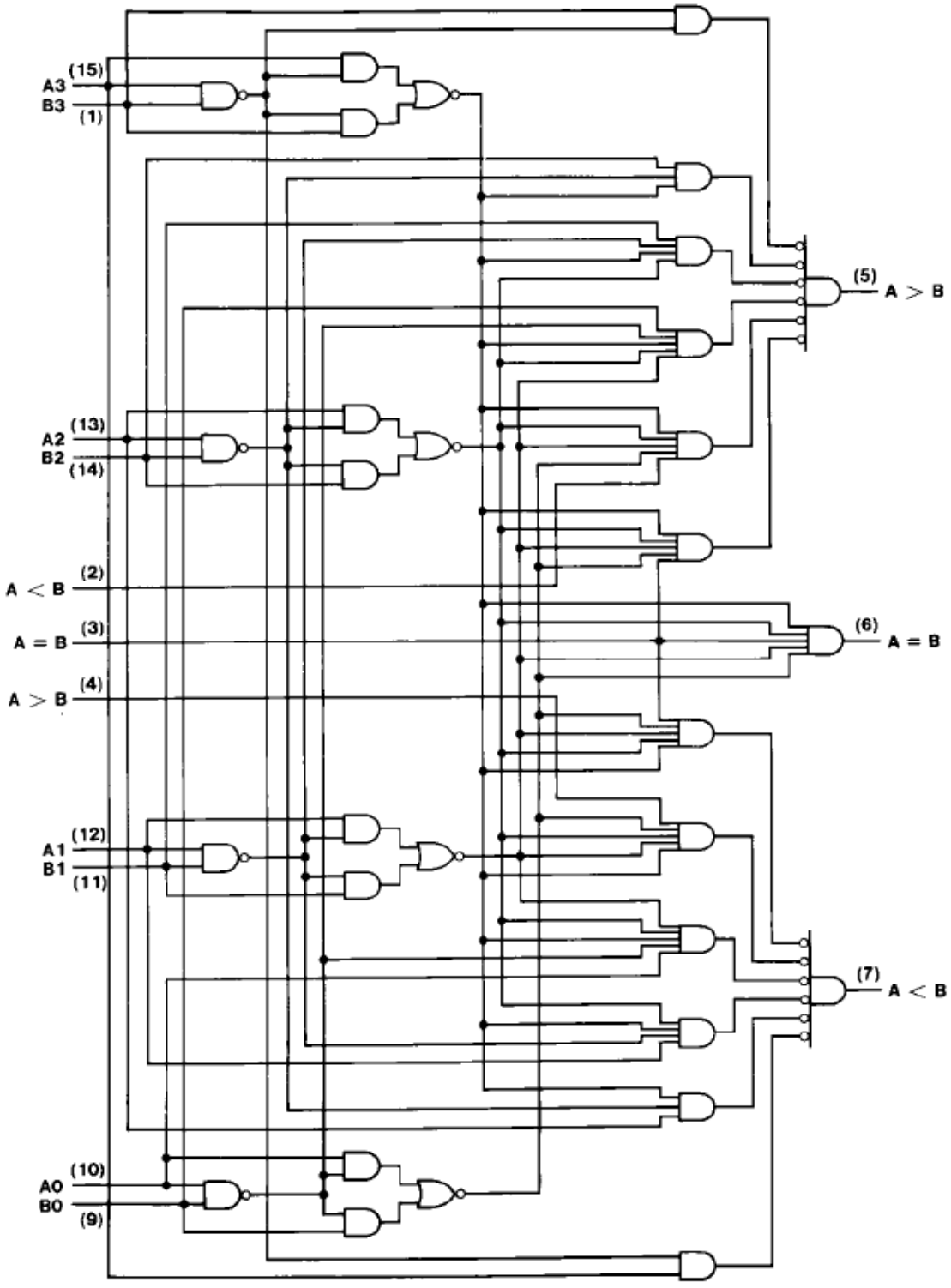


FUNCTION TABLE:

Comparing Inputs				Cascading Inputs			Outputs		
A3, B3	A2, B2	A1, B1	A0, B0	A > B	A < B	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 Level, L = Low Level, X = Don't Care

LOGIC DIAGRAM:



PROGRAM:

DATAFLOW MODEL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity comp74x85 is
   Port ( a,b : in  STD_LOGIC_VECTOR (3 downto 0);
          aeqbin,agtbin,altbin : in  STD_LOGIC;
          aeqbout,agtbout,altbout : out  STD_LOGIC);
end comp74x85;
architecture dataflow of comp74x85 is
begin
aeqbout <= '1' when (a = b and (aeqbin = '1' and agtbin = '0' and altbin = '0')) else '0';
agtbout <= '1' when (a > b or (a=b and agtbin='1')) else '0';
altbout <= '1' when (a < b or (a=b and altbin='1')) else '0';
end dataflow;
```

BEHAVIORAL MODEL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity comp74x85_b is
   Port ( a,b : in  STD_LOGIC_VECTOR (3 downto 0);
          aeqbin,agtbin,altbin : in  STD_LOGIC;
          aeqbout,agtbout,altbout : out  STD_LOGIC);
end comp74x85_b;

architecture Behavioral of comp74x85_b is
begin
process (a,b,aeqbin,agtbin,altbin)
begin
if (a=b and (aeqbin ='1')) then aeqbout <= '1'; agtbout <= '0'; altbout <= '0';
elsif (a >b or(a=b and (agtbin ='1'))) then aeqbout <= '0'; agtbout <= '1'; altbout <= '0';
else aeqbout <= '0'; agtbout <= '0'; altbout <= '1';
end if;
end process;
end Behavioral;
```

structural style model.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity \4_comp_stru\ is
          port(
                    a : in STD_LOGIC_VECTOR(0 to 3);
                    b : in STD_LOGIC_VECTOR(0 to 3);
                    aeqb : inout STD_LOGIC;
                    agtb : inout STD_LOGIC;
                    altb : out STD_LOGIC
               );
end \4_comp_stru\;
```

```
architecture \4_comp_str\ of \4_comp_stru\ is
        component xnor2
                port(l,m: in std_logic;n: out std_logic);
        end component;
        component and2
                port(x,y: in std_logic; z: out std_logic);
        end component;
        component inv
                port (u: in std_logic; v: out std_logic);
        end component;
        component and3
                port(l,m,o: in std_logic;n: out std_logic);
        end component;
        component or4
                port(m1,m2,m3,m4: in std_logic; mf: out std_logic);
        end component;
        component and4
                port(q1,q2,q3,q4: in std_logic; qf: out std_logic);
        end component;
        component and5
                port (e1,e2,e3,e4,e5: in std_logic; ef: out std_logic);
        end component;
        component nor2
                port(l1,l2: in std_logic; lf: out std_logic);
        end component;
        signal i0,i1,i2,i3,j0,j1,j2,j3,j4,j5,h0,h1,h2,h3: std_logic;
        begin
                m1: inv port map (b(3),i3);
                m2: inv port map (b(2),i2);
                m3: inv port map (b(1),i1);
                m4: inv port map (b(0),i0);
                m5: xnor2 port map (a(3),b(3),j3);
                m6: xnor2 port map (a(2),b(2),j2);
                m7: xnor2 port map (a(1),b(1),j1);
                m8: xnor2 port map (a(0),b(0),j0);
                m9: and2 port map (a(3),i3,h3);
                m10: and3 port map (a (2),i2,j3,h2);
                m11: and4 port map (a(1),i1,j2,j3,h1);
                m12: and5 port map (a (0), i0,j1,j2,j3,h0);
                m13: and4 port map (j0,j1,j2,j3,aeqb);
                m14: or4 port map (h0, h1,h2,h3,agtb);
                m15: nor2 port map (aeqb,agtb,altb);

end \4_comp_str\;
```

Result & Analysis:

Synthesis Report:

Power Analysis:

Timing Analysis:

VIVA QUESTIONS:

1. Write the dataflow model for the IC 74x85.
2. Write the VHDL code for the IC 74x85 using CASE statement.
3. Write the VHDL code for the IC 74x85 using WITH statement.
4. Write the VHDL code for the IC 74x85 using WHEN--ELSE statement.
5. Write the structural program for IC 74x85.
6. How many 4-bit comparators are needed to construct 12-bit comparator?
7. What does a digital comparator mean?
8. Design a 2-bit comparator using gates?
9. Explain the phases of a simulation?
10. Explain briefly about wait statement?

GRAPH SHEET

Exp.No: **Date:**

D FLIP-FLOP 7474

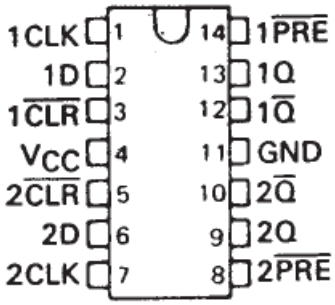
AIM: To simulate and synthesize D Flip-Flop (7474).

APPARATUS:
1. Xilinx 12.1 tool
2. ISim simulator
3. XST synthesizer
4. FPGA Board-Spartan3

THEORY:

These device contains two independent D-type positive edge triggered flip-flops. A low level at he preset or clear inputs set or resets the output regardless of the levels of the other inputs. When preset and clear are inactive, data at the D flip-flop inputare transferred to the output on the positive going edge of the clock pulse.

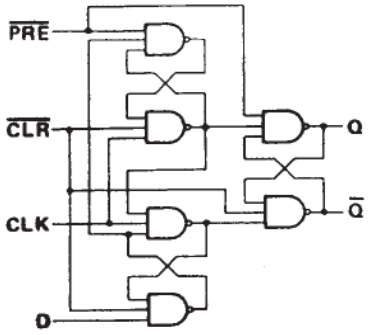
PIN DIAGRAM:



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†	H†
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q_0	$\overline{Q}_0$

LOGIC DIAGRAM:



PROGRAM:

BEHAVIORAL MODEL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity dff7474 is
  Port ( clk1, clk2, clr_l1, clr_l2, pr_l1, pr_l2, d1, d2 : in  STD_LOGIC;
         q1,qbar1, q2, qbar2 : out  STD_LOGIC);
end dff7474;
architecture Behavioral of dff7474 is
begin
p1: process (clk1, clr_l1, pr_l1, d1)
begin
if (pr_l1 ='0' and clr_l1 ='0')  then q1<= '1'; qbar1 <= '1';
elsif (pr_l1 = '0') then q1<= '1'; qbar1 <= '0';
elsif (clr_l1 = '0') then q1<= '0'; qbar1 <= '1';
elsif (pr_l1 ='1' and clr_l1 ='1' and (clk1'event and clk1 = '1')) then q1<= d1; qbar1 <= not d1;
end if;
end process;

p2: process (clk2, clr_l2, pr_l2, d2)
begin
if (pr_l2 ='0' and clr_l2 ='0')  then q2<= '1'; qbar2 <= '1';
elsif (pr_l2 = '0') then q2<= '1'; qbar2 <= '0';
elsif (clr_l2 = '0') then q2<= '0'; qbar2 <= '1';
elsif (pr_l2 ='1' and clr_l2 ='1' and (clk2'event and clk2 = '1')) then q2<= d2; qbar2 <= not d2;
end if;
end process;
end Behavioral;
```

STRUCTURAL MODEL:

```
library IEEE;
use ieee.std_logic_1164.all;
entity dff is
port ( pr_l: in STD_LOGIC; -- active low preset input
clr_l:in STD_LOGIC; -- active low clear input
clk :in STD_LOGIC; -- clock input
d :in STD_LOGIC; -- D input
q :inout STD_LOGIC; -- output of D flip flop
qn :inout STD_LOGIC -- inverted output );
end dff;
architecture dff of dff is
signal e,f,g,h:std_logic;
component nand3
port (
a,b,c: in STD_LOGIC;
d : out STD_LOGIC );
end component;
begin
g1:nand3 port map(pr_l,h,f,e); -- creates g1 gate
g2:nand3 port map(clr_l,e,clk,f); -- creates g2 gate
g3:nand3 port map(f,clk,h,g); -- creates g3 gate
g4:nand3 port map(g,clr_l,d,h); -- creates g4 gate
```

```
g5:nand3 port map(pr_l,f,qn,q); -- creates g5 gate
g6:nand3 port map(q,g,clr_l,qn); -- creates g6 gate
end dff;
--VHDL code for 3 i/p nand gate
library IEEE;
use IEEE.std_logic_1164.all;
entity nand3 is
port ( a,b,c: in STD_LOGIC;
d : out STD_LOGIC );
end nand3;
architecture \nand\ of nand3 is
begin
d<= not (a and b and c); -- creates a 3 i/p nand gate
end \nand\;
```

Result & Analysis:

Synthesis Report:

Power Analysis:

Timing Analysis:

VIVA QUESTIONS:

1. Write the behavioral code for the IC 74x74.
2. Write the dataflow code for the IC 74x74.
3. What is the difference between sequential and combinational circuit?
4. What is a flip-flop?
5. Explain the functions of preset and clear inputs in flip-flop?
6. What is meant by a clocked flip-flop?
7. What is meant by excitation table?
8. What is the difference between flip-flop and latch?
9. What are the various methods used for triggering flip-flops?
10. Explain level triggered flip-flop?
11. Write the behavioral code for IC 74X74.
12. Write the syntax of IF statement?

GRAPH SHEET

Exp.No: **Date:**

DECADE COUNTER 7490

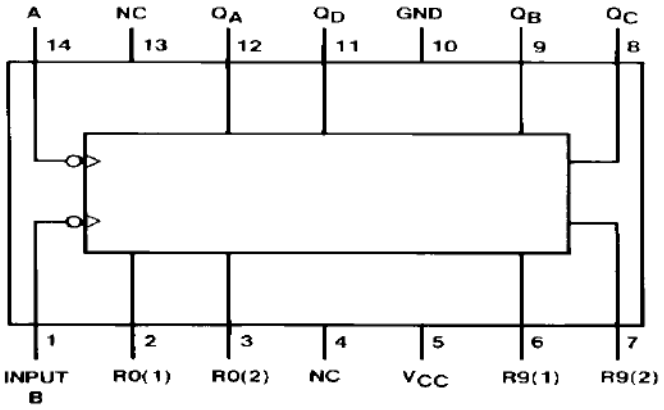
AIM: To simulate and synthesize decade counter (7490).

APPARATUS:
1. Xilinx 12.1 tool
2. ISim simulator
3. XST synthesizer
4. FPGA Board-Spartan3

THEORY:

Each of these monolithic counters 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 for the DM74LS90. All of these counters have a gated zero reset and the DM74LS90 also has gated set-to-nine inputs for use in BCD nine's complement applications. To use their maximum count length (decade or four bit binary), the B input is connected to the QA output. The input count pulses are applied to input A and the outputs are as described in the appropriate truth table. A symmetrical divide-by-ten count can be obtained from the DM74LS90 counters by connecting the 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.

PIN DIAGRAM:



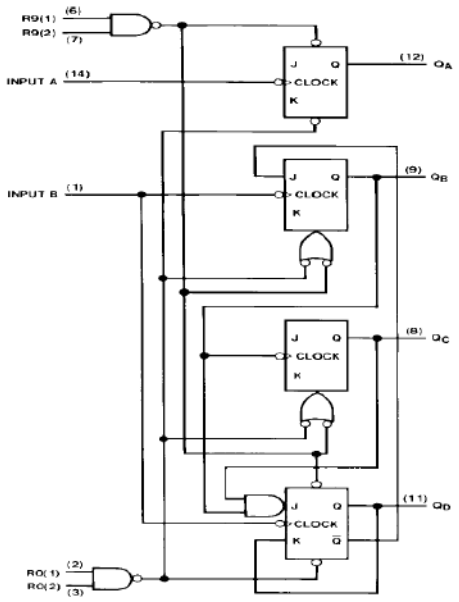
RESET/COUNT TRUTH TABLE:

Reset Inputs				Output			
R0(1)	R0(2)	R9(1)	R9(2)	Q_D	Q_C	Q_B	Q_A
H	H	L	X	L	L	L	L
H	H	X	L	L	L	L	L
X	X	H	H	H	L	L	H
X	L	X	L	COUNT			
L	X	L	X	COUNT			
L	X	X	L	COUNT			
X	L	L	X	COUNT			

TRUTH TABLE:

Count	Output			
	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

LOGIC DIAGRAM:



PROGRAM:

BEHAVIORAL MODEL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
entity dec_counter is
   Port ( clk : in  STD_LOGIC;
          r0_1, r0_2, r9_1, r9_2 : in  STD_LOGIC;
          q : out  STD_LOGIC_VECTOR (3 downto 0));
end dec_counter;
architecture Behavioral of dec_counter is
signal count: std_logic_vector (3 downto 0);
begin
process (clk, r0_1, r0_2, r9_1, r9_2, count)
begin
if (r0_1 and r0_2)= '1' then count <= "0000";
elsif (r9_1 and r9_2)= '1' then count <= "1001";
elsif (clk'event and clk = '0') then count <= count + 1;
if (count = 9) then count <= "0000";
```

```
end if;
end if;
q <= count;
end process;
end Behavioral;
```

STRUCTURAL MODEL:

```
library IEEE;
Use IEEE.std_logic_1164.all;
Entity count is
port (s0, s1, r0, r1: in STD_LOGIC; --set and reset i/ps for mod2 and
-- Mod5 counters
Clk0: in STD_LOGIC; --Clock signal for mod2 counter
Clk1: inout STD_LOGIC; --Clock signal for mod5 counter
q : inout STD_LOGIC_VECTOR(3 downto 0) --o/p of -- mod2 X mod5= mod10 );
end count;
architecture count of count is
component jk_ff -- jk flip flop instantiation
port ( jk : in STD_LOGIC_VECTOR(1 downto 0);
clk,pr_l,clr_l : in STD_LOGIC;
q,nq : inout STD_LOGIC );
end component;
signal preset,clear,S, q3bar:STD_LOGIC;
begin
preset <= s0 nand s1; -- common preset inputs for mod2 and mod5 counters
clear <=r0 nand r1; -- common reset inputs for mod2 and mod5 counters
S<=q(2) and q(1); -- to set the last flip flop
q3bar <= not q(3); -- complemented output of q(3)
clk1<=q(0); --to work as asynchronous mod10 counter
jk1:jk_ff port map("11",clk0,preset,clear,q(0),open);
jk2:jk_ff port map(jk(1)=> q3bar,
jk(0)=>'1',
clk=>clk1,
pr_l=>preset,
clr_l=>clear,
q=>q(1),
nq=>open); -- jk1.jk2,jk3,jk4 create four JK flip flops
jk3:jk_ff port map("11",q(1),preset,clear,q(2),open);
jk4:jk_ff port map(jk(0)=>q(3),
jk(1)=>s,
clk=>clk1,
pr_l=>preset,
clr_l=>clear,
q=>q(3),
nq=> q3bar);
end count;
```

--Program for JK flip-flop

```
library IEEE;
use IEEE.std_logic_1164.all;
entity jk_ff is
port (
jk : in STD_LOGIC_VECTOR(1 downto 0);
```

```
--jk(1)=J;jk(0)=K;
clk,pr_l,clr_l : in STD_LOGIC;
q,nq : inout STD_LOGIC );
end jk_ff;
architecture jk of jk_ff is
begin
process(clk,pr_l,clr_l,jk)
variable temp:std_logic:='0';
begin
q<='0';nq<='1';
if (pr_l='1' and clr_l='0') then
q<='0';nq<='1';
elsif (pr_l='0' and clr_l ='1') then
q<='1';nq<='0';
elsif (pr_l='1' and clr_l='1') then
if (clk 'event and clk='0') then --performs during the falling edge of clock
                    case jk is
when "00"=>temp:=temp;
when "01"=>temp:='0';
when "10"=>temp:='1';
when "11"=>temp:=not temp;
when others=>null;
end case;
end if;
q<=temp;
nq<= not temp;
end if;
end process;
end jk;
```

VHDL Code For D-FF Data Flow Model

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity dffdf is
Port ( d,clk : in  STD_LOGIC;
q,qb : inout STD_LOGIC);
end dffdf;
architecture dffdfar of dffdf is
signal d1,s1,r1:STD_LOGIC;
begin
s1 <= d nand clk;
d1 <= d nand d;
r1 <= d1 nand clk;
q <= s1 nand qb;
qb <= r1 nand q;
end dffdfar;
```

Result & Analysis:

Synthesis Report:

Power Analysis:

Timing Analysis:

VIVA QUESTIONS:

1.Write the behavioral code for IC 74x90.
1.What is a sequential circuit?
2.Differentiate between synchronous and asynchronous counter?
3.How many no.of flip-flops are required for decade counter?
4.What is meant by excitation table?
5.What are the meanings of different types of values in std_ulogic?
6.What are the objects in VHDL?
7.Write the syntax for a signal?
8.Write the difference between signal and variable?
9.Explain about enumeration types?
10.If the modulus of a counter is 12 how many flip-flops are required?

GRAPH SHEET

Exp.No: **Date:**

4-BIT BINARY COUNTER 7493

AIM: To simulate and synthesize 4-bit binary counter (7493).

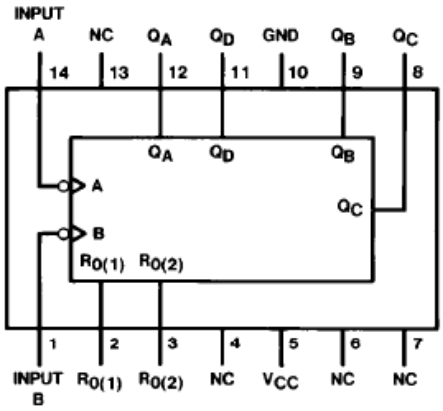
APPARATUS:
1. Xilinx 12.1 tool
2. ISim simulator
3. XST synthesizer
4. FPGA Board-Spartan3

THEORY:

The 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-eight for the 93A. All of these counters have a gated zero reset.

To use their maximum count length (decade or four-bit binary), the B input is connected to the QA output. The input count pulses are applied to input A and the outputs are as described in the appropriate truth table. A symmetrical divide-by-ten count can be obtained from the 90A counters by connecting the 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.

PIN DIAGRAM:



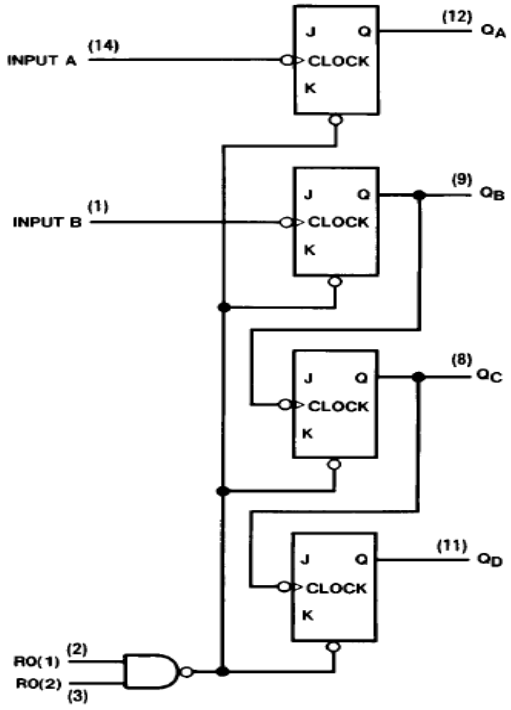
RESET/COUNT TRUTH TABLE:

Reset Inputs		Outputs			
R0(1)	R0(2)	Q_D	Q_C	Q_B	Q_A
H	H	L	L	L	L
L	X	COUNT			
X	L	COUNT			

TRUTH 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
10	H	L	H	L
11	H	L	H	H
12	H	H	L	L
13	H	H	L	H
14	H	H	H	L
15	H	H	H	H

LOGIC DIAGRAM:



PROGRAM:

BEHAVIORAL MODEL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
entity counter is
    Port ( r0_1, r0_2, clk : in  STD_LOGIC;
           q : out  STD_LOGIC_VECTOR (3 downto 0));
end counter;

architecture Behavioral of counter is
signal count: STD_LOGIC_VECTOR (3 downto 0);
begin
process (r0_1, r0_2, clk, count)
begin
if (r0_1 = '1' and r0_2 = '1') then count <= "0000";
elsif (clk'event and clk = '1') then
 if (r0_1 = '0') then
  count <= count + 1;
 elsif (r0_2 = '0') then
  count <= count + 1;
  end if;
  end if;
  q <= count;
  end process;
end Behavioral;
```

STRUCTURAL MODEL:

```
--Program for 4-bit counter
library IEEE;
use IEEE.std_logic_1164.all;
entity cnt is
port (
clk0: in STD_LOGIC;
mr0: in STD_LOGIC;
mr1: in STD_LOGIC;
clk1: inout STD_LOGIC;
Q:inout STD_LOGIC_VECTOR(3 downto 0));
end cnt;
architecture cnt of cnt is
Component tff -- T- flip flop instantiation
port (
t : in STD_LOGIC;
clk : in STD_LOGIC;
clr_l : in STD_LOGIC;
q,nq : out STD_LOGIC
);
end component;
signal clear : std_logic;
begin
```

```
clear<= mr0 nand mr1; -- common reset inputs for mod2 and mod8
--counters
CLK1<=q(0); --to work as asynchronous mod16 counter
t1:tff port map('1',clk0,clear,Q(0),open);--t1,t2,t3,t4 create four T-flip flops
t2:tff port map('1',clk1,clear,Q(1), open);
t3:tff port map('1',Q(1),clear,Q(2), open);
t4:tff port map('1',Q(2),clear,Q(3), open);
end cnt;

--Program for T flip-flop
library IEEE;
use IEEE.std_logic_1164.all;
entity tff is
port (
t : in STD_LOGIC;--input to the T-flip flop
clk : in STD_LOGIC;--Clock signal for T-flip flop
clr_l : in STD_LOGIC;--active low clear input
q,nq : out STD_LOGIC--actual and complemented outputs of T-flip flop
);
end tff;
architecture tff of tff is
begin
process(t,clk,clr_l)
variable temp:STD_LOGIC:='0';
begin
if (clr_l='0') then
temp:='0';

elsif ((clr_l='1') and (clk'event and clk='0')) then--perfoms during falling edge
if ( t='0') then
temp:=temp;
else temp:= not temp;
end if;
end if;
q<= temp;
nq<= not temp;
end process;
end tff;
```

Result & Analysis:

Synthesis Report:

Power Analysis:

Timing Analysis:

VIVA QUESTIONS:

1. Write the behavioral code for IC 74x93.
2. What is the difference between decade counter and 4 bit counter?
3. What is meant by a modulus of a counter?
4. Write the behavioral code for IC74X93?
5. Explain the operation of IC74X93?
6. Write the syntax for component instantiation?
7. What is netlist?
8. Briefly explain about generics?
9. Write the difference between sequential statement and concurrent statement?
10. Write the syntax for loop statements?
11. Write the syntax for generate statements?
12. Write the differences between loop and generate?

GRAPH SHEET

Exp.No: **Date:**

SHIFT REGISTER 7495

AIM: To simulate and synthesize shift register (7495).

APPARATUS:
1. Xilinx 12.1 tool
2. ISim simulator
3. XST synthesizer
4. FPGA Board-Spartan3

THEORY:

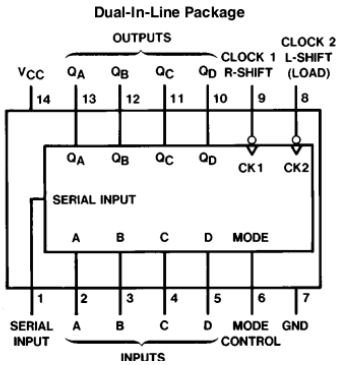
Shift registers area type of sequential logic circuit, mainly for storage of digital data. They are a group of flip -flops connected in a chain so that the output from one flip -flop becomes the input of the next flip -flop. Most of the registers possess no characteristic internal sequence of states. All flip -flop is driven by a common clock, and all are set or resetsimultaneously.

The basic types of shift registers are

- Serial In -Serial Out
- Serial In -Parallel Out
- Parallel In –Serial Out
- Parallel In -Parallel Out and bidirectional shift registers.

Serial In -Serial Out Shift Registers The serial in/serial out shift register accepts data serially –that is, one bit at a time on a single line. It produces the stored information on its output also in serial form.

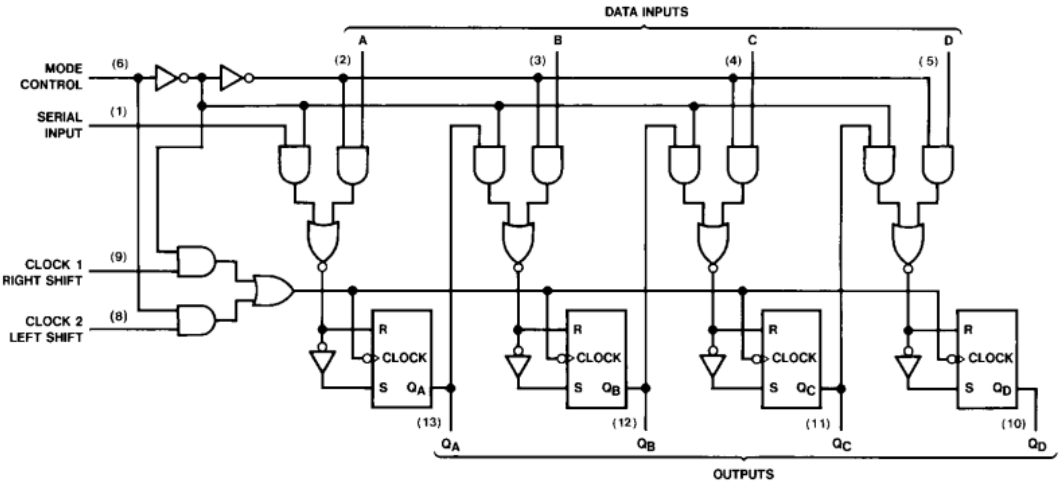
PIN DIAGRAM:



Function Table

Inputs								Outputs			
Mode Control	Clocks		Serial	Parallel				Q_A	Q_B	Q_C	Q_D
	2(L)	1(R)		A	B	C	D				
H	H	X	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}
H	↓	X	X	a	b	c	d	a	b	c	d
H	↓	X	X	$Q_{B\dagger}$	$Q_{C\dagger}$	$Q_{D\dagger}$	d	Q_{Bn}	Q_{Cn}	Q_{Dn}	d
L	L	H	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}
L	X	↓	H	X	X	X	X	H	Q_{An}	Q_{Bn}	Q_{Cn}
L	X	↓	L	X	X	X	X	L	Q_{An}	Q_{Bn}	Q_{Cn}
↑	L	L	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}
↓	L	L	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}
↓	L	H	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}
↑	H	L	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}
↑	H	H	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}

LOGIC DIAGRAM:



PROGRAM:

--VHDL code for Barrel Shifter:

```
library ieee;
use ieee.std_logic_1164.all;

entity barrel is
port(inp:in std_logic_vector(7 downto 0);
shift:in std_logic_vector(2 downto 0);
outp:out std_logic_vector(7 downto 0));
end barrel;

architecture behaviour of barrel is
begin
process(inp,shift)
variable temp1:std_logic_vector(7 downto 0);
variable temp2:std_logic_vector(7 downto 0);
begin
--1st shifter
if(shift(0)='0') then
temp1:=inp;
else
temp1(0):='0';
for i in 1 to 7 loop
temp1(i):=inp(i-1);
end loop;
end if;

--2nd shifter

if(shift(1)='0') then
temp2:=temp1;
else
```

```
for i in 0 to 1 loop
temp2(i):='0';
end loop;
for i in 2 to inp'high loop
temp2(i):=temp1(i-2);
end loop;
end if;

--3rd shifter
if(shift(2)='0') then
outp<=temp2;
else
for i in 0 to 3 loop
outp(i)<='0';
end loop;
for i in 4 to inp'high loop
outp(i)<=inp(i-4);
end loop;
end if;
end process;
end behaviour;
```

Result & Analysis:

Synthesis Report:

Power Analysis:

Timing Analysis:

VIVA QUESTIONS:

1.what is a shift register?

2.what are the operation of shift register?

3.explain the working of four opearations?

GRAPH SHEET

Exp.No: **Date:**

UNIVERSAL SHIFT REGISTER 74194

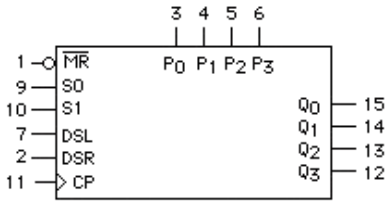
AIM: To simulate and synthesize universal shift register (74194).

APPARATUS:
1. Xilinx 12.1 tool
2. ISim simulator
3. XST synthesizer
4. FPGA Board-Spartan3

THEORY:

The 74X194 is a high speed CMOS 4-BIT PIPO SHIFT REGISTER. This SHIFT REGISTER is designed to incorporate virtually all of the features a system designer may want in a shift register. It features parallel inputs, parallel outputs, right shift and left shift serial inputs, clear line. The register has four distinct modes of operation: PARALLEL (broadside) LOAD; SHIFT RIGHT (in the direction QA QD); SHIFT LEFT; INHIBIT CLOCK (do nothing). Synchronous parallel loading is accomplished by applying the four data bits and taking both mode control inputs, S0 and S1 high. The data are loaded into their respective flip-flops and appear at the outputs after the positive transition of the CLOCK input. During loading, serial data flow is inhibited. Shift right is accomplished synchronously with the rising edge of the clock pulse when S0 is high and S1 is low. Serial data for this mode is entered at the SHIFT RIGHT data input. When S0 is low and S1 is high, data shifts left synchronously and new data is entered at the SHIFT LEFT serial input. Clocking of the flip-flops is inhibited when both mode control inputs are low. The mode control inputs should be changed only when the CLOCK input is high. All inputs are equipped with protection circuits against static discharge and transient excess voltage.

PIN DIAGRAM:

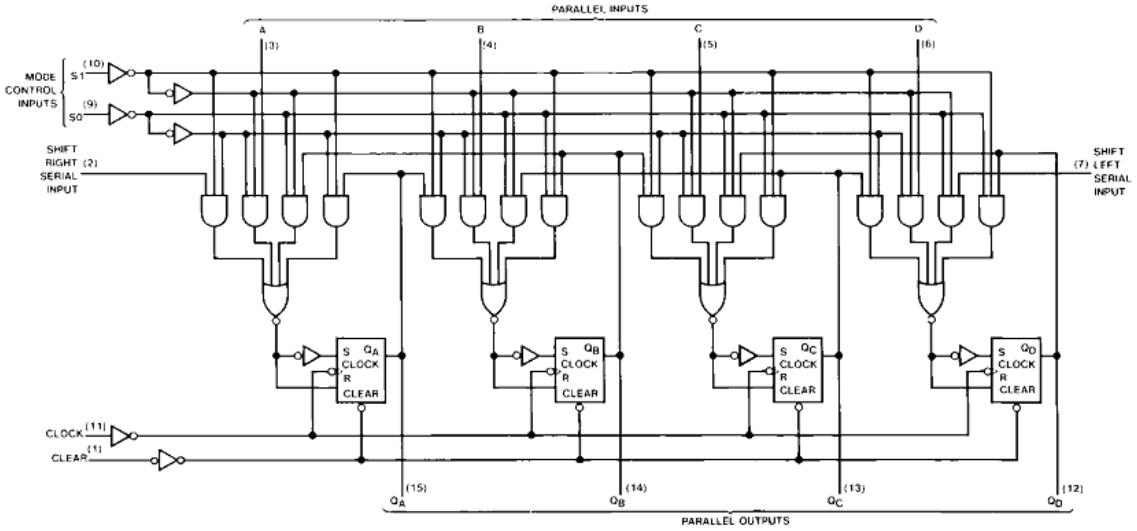


74194 4-bit BiDirectional Shift Register
S1 S0 are mode controls.
00 - Hold 01 - Shift Right
10 - Shift Left 11 - Parallel Load

TRUTH 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

LOGIC DIAGRAM:



PROGRAM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
entity usr74194 is
   Port ( clk, clr_l, lin, rin : in  STD_LOGIC;
          s : in  STD_LOGIC_VECTOR (1 downto 0);
          d : in  STD_LOGIC_VECTOR (3 downto 0);
          q : inout  STD_LOGIC_VECTOR (3 downto 0));
end usr74194;

architecture Behavioral of usr74194 is
begin
process (clk, clr_l, lin, rin, s, d, q )
begin
if clr_l = '0' then q<= "0000";
elsif ( clk'event and clk = '1') then
 case s is
            when "00" => q <= q;
            when "01" => q <= rin & q (3 downto 1);
            when "10" => q <= q (2 downto 0) & lin;
            when "11" => q <= d;
            when others => q <= "UUUU";
end case;
end if;
end process;
end Behavioral;
```

Following is the VHDL code for an 8-bit shift-left register with a positive-edge clock, serial in, and serial out.

```
library ieee;
use ieee.std_logic_1164.all;
entity shift is
 port(C, SI : in  std_logic;
```

```
      SO : out std_logic);
end shift;
architecture archi of shift is
 signal tmp: std_logic_vector(7 downto 0);
 begin
   process (C)
     begin
       if (C'event and C='1') then
         for i in 0 to 6 loop
           tmp(i+1) = tmp(i);
         end loop;
         tmp(0) = SI;
       end if;
   end process;
   SO = tmp(7);
end archi;
```

Result & Analysis:

Synthesis Report:

Power Analysis:

Timing Analysis:

VIVA QUESTIONS:

1. Why is the universal shift register named so?
2. is there any difference between shift register and universal shift register?
3. how can be the operations differentiated in universal shift register?

GRAPH SHEET

Exp.No: **Date:**

RAM (74189)

AIM: To simulate and synthesize RAM (74189).

APPARATUS:
1. Xilinx 12.1 tool
2. ISim simulator
3. XST synthesizer
4. FPGA Board-Spartan3

THEORY:

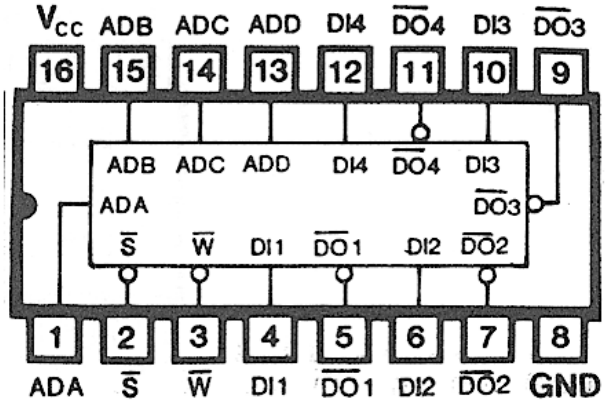
The behaviour of the 74189 circuit is controlled by just two active-low control lines, namely the chip select and read/write inputs:

- nCS=1: the data outputs are tri-stated and the clock signal for the latches in the memory matrix is disabled.
- nCS=0, Read/nWrite=1: the data outputs are enabled and driven with the contents of the currently addressed memory word. When the address input is changed, the contents of the newly selected memory word will appear on the data outputs, delayed by the memory access time.
- nCS=0, Read/nWrite=0: the clock signal of the currently addressed memory latches is enabled, so that the values on the data input bus is copied into the selected memory word (transparent latches). Also, the data outputs are enabled. Switch the Read/nWrite signal back to the high (1) state to store the data.

To get accustomed to the behaviour of the SRAM, it is a good exercise to try to write a few data words into the memory (e.g. the values shown in the screenshot above).

Due to the asynchronous interface of the RAM, great care must be taken to avoid hazard conditions on either of the Read/nWrite and address inputs. A good example is provided by the following procedure to clear all RAM contents: 1. clear the data inputs (value 0000), 2. enable the Read/nWrite signal (value 0), 3. step through all addresses. Needless to say that such tricks are not recommended for real system designs...

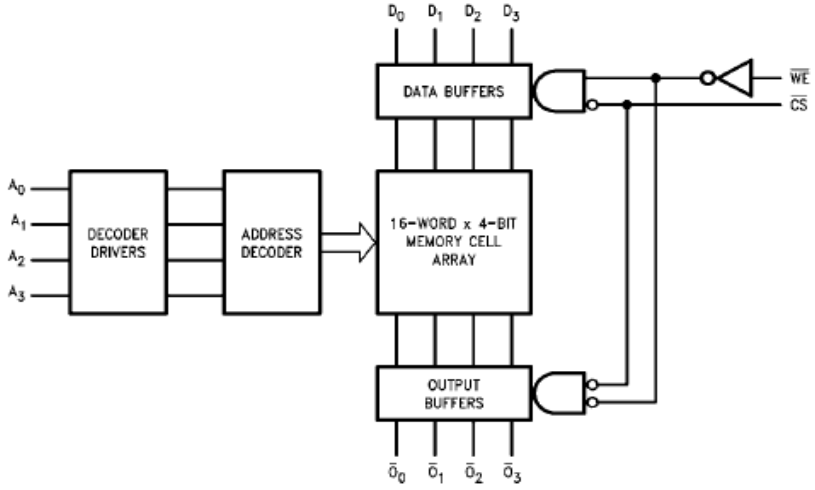
PIN DIAGRAM:



Function Table

Inputs		Operation	Condition of Outputs
$\overline{CS}$	$\overline{WE}$		
L	L	Write	High Impedance
L	H	Read	Complement of Stored Data
H	X	Inhibit	High Impedance

Block Diagram



PROGRAM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity Dm74189 is
   Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
          d : in  STD_LOGIC_VECTOR (3 downto 0);
          cs_bar : in  STD_LOGIC;
          we_bar : in  STD_LOGIC;
          q : out  STD_LOGIC_VECTOR (3 downto 0));
end Dm74189;
architecture Ram of Dm74189 is
type ram_type is array (0 to 15) of STD_LOGIC_VECTOR (3 downto 0);
signal temp:Ram_type;
begin
write: process(cs_bar,we_bar,a,d)
begin
if (cs_bar='0' and we_bar='0') then temp(conv_integer(a))<=d;
end if;
end process write;
read : process (cs_bar,we_bar,a,d)
variable v1:std_logic_vector (3 downto 0);
begin
if (cs_bar='0' and we_bar ='1') then v1:=temp(conv_integer(a));
```

```
else
v1:=(v1'range =>'Z');
end if;
q<= v1;
end process read;
end Ram;
```

Result & Analysis:

Synthesis Report:

Power Analysis:

Timing Analysis:

VIVA QUESTIONS:

1. Write the behavioral code for IC 74x189 without declaring the function.
2. Explain about different types of RAMs?
3. How to specify the memory size?
4. Explain read and write operations?
5. What are the differences between RAM and RAM?
6. Explain the steps of a compilation process of a VHDL program?
7. Explain the types of design units?
8. Why configurations are needed?
9. What is binding?
10. What is subprogram in vhdl ?

GRAPH SHEET

Exp.No: **Date:**

ALU(74381)

AIM: To simulate and synthesize ALU (74381).

APPARATUS:
1. Xilinx 12.1 tool
2. ISim simulator
3. XST synthesizer
4. FPGA Board-Spartan3

THEORY:

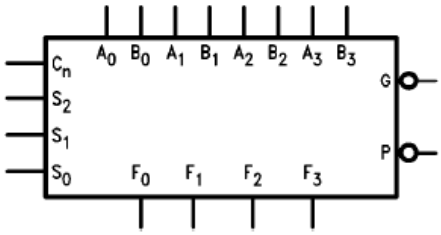
The 'F381 performs three arithmetic and three logic operations on two 4-bit words, A and B. Two additional select input codes force the function outputs LOW or HIGH. Carry propagate and generate outputs are provided for use with the 'F182 carry look ahead generator for high-speed expansion to longer word lengths.

Features

- Low input loading minimizes drive requirements
- Performs six arithmetic and logic functions
- Selectable LOW (clear) and HIGH (preset) functions
- Carry generate and propagate outputs for use with carry.

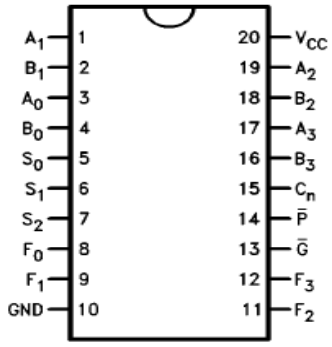
PIN DIAGRAM:

Logic Symbols



Connection Diagram

Pin Assignment for DIP and SOIC



FUNCTION TABLE:

Input			
S2	S1	S0	Function
0	0	0	F=0000
0	0	1	$F=B-A-1+C_{in}$
0	1	0	$F=A-B-1+C_{in}$
0	1	1	$F=A+B+C_{in}$
1	0	0	$F = A \oplus B$
1	0	1	F=A+B
1	1	0	F=A.B
1	1	1	F=1111

PROGRAM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity IC74381 is
   Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
          b : in  STD_LOGIC_VECTOR (3 downto 0);
          s : in  STD_LOGIC_VECTOR (2 downto 0);
          cin : in  STD_LOGIC;
          f : out  STD_LOGIC_VECTOR (3 downto 0));
end IC74381;
architecture Behavioral of IC74381 is
begin
process(a,b,cin,s)
begin
case s is
when "000" =>f<="0000";
when "001" =>f<= b-a-1+cin;
when "010" =>f<= a-b-1+cin;
when "011" =>f<= a+b+cin;
when "100" =>f<= a xor b;
when "101" =>f<= a or b;
when "110" =>f<= a and b;
when  “111” =>f<= "1111";
when  others =>f<= "UUUU";
end case;
end process;
end Behavioral;
```

Result & Analysis:

Synthesis Report:

Power Analysis:

Timing Analysis:

GRAPH SHEET

PROJECTS TO BE IMPLEMENTED

1) Design a 4-bit Carry Look-Ahead Adder and Develop the VHDL Code for it.

2) Design a 4x4 Array Multiplier and Develop the VHDL Code for it.

3) Design a Full Adder by using two Half Adders and Develop the VHDL Code for it.

4) Design a Priority Encoder and Develop the VHDL Code for it.

5) Design a BCD to 7-Segment Decoder and Develop the VHDL Code for it.

6) Write a VHDL Code for 4-bit Barrel Shifter.

7)Write VHDL Code for UART Transmitter and Receiver.