WEST GODAVARI INSTITUTE OF SCIENCE & ENGINEERING

PRAKASARAOPALEM, TADEPALLIGUDEM-534112

Department Of

ELECTRONICS & COMMUNICATION ENGINEERING

LABORATORY MANUAL

For

DIGITAL IC APPLICATIONS LAB

Prepared by
K.Lalitha
for

II-B.Tech: II-Sem

NAME	
BRANCH	**ELECTRONICS & COMMUNICATION ENGINEERING**
REGULATION	**R 20**
YEAR & SEM	**II YEAR – IInd SEM**
ACADEMIC YEAR	**2022-2023**
ROLL NUMBER	

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA

KAKINADA – 533 003, Andhra Pradesh, India

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

II Year – II Semester		L	T	P	C
		0	0	3	1.5
DIGITAL IC DESIGN LAB					

Note: The students are required to design and draw the internal logical structure of the followingDigital Integrated Circuits and to develop VHDL/Verilog HDL Source code, perform simulationusing relevant simulator and analyze the obtained simulation results using necessary synthesizer.All the experiments are required to verify and implement the logical operations on the latest FPGA Hardware in the Laboratory.

List of Experiments: (Minimum of Ten Experiments has to be performed)

1. Realization of Logic Gates
2. Design of Full Adderusing 3 modeling systems
3. 3 to 8 Decoder-74138
4. 8 to 3 Encoder (with and without parity)
5. 8x1Multiplexer-74151 and 2x4De-multiplexer-74155
6. 4-Bit comparator-7485
7. D Flip-Flop-7474
8. Decade counter -7490
9. Shift registers-7495
10. 8-bit serialin-parallel out and parallel in-serial out
11. Fast In & Fast Out (FIFO)
12. MAC (Multiplier &Accumulator)
13. ALU Design.

***Department Of* ELECTRONICS & COMMUNICATION ENGINEERING**

DIGITAL IC APPLICATIONS LABORATORY

List of Experiments

1. Realization of Logic Gates
2. Design of Full Adder using 3 modeling systems
3. 3 to 8 Decoder -74138
4. 8 to 3 Encoder (with and without parity)
5. 8 x 1 Multiplexer-74151 and 2x 4 De-multiplexer-74155
6. 4- Bit comparator-7485
7. D Flip-Flop-7474
8. Decade counter -7490
9. Shift registers-7495
10. 8-bit serial in-parallel out and parallel in-serial out
11. Fast In & Fast Out (FIFO)
12. MAC (Multiplier & Accumulator)
13. ALU Design.

ADDITIONAL EXPERIMENTS

1. UNIVERSAL SHIFT REGISTERS

2. 4-bit counter

Lab Internal-1	Present	Absent	**Lab Internal-2**	Present	Absent
Date		2023	**Date**		2023
Signature of Examiner			**Signature of Examiner**		

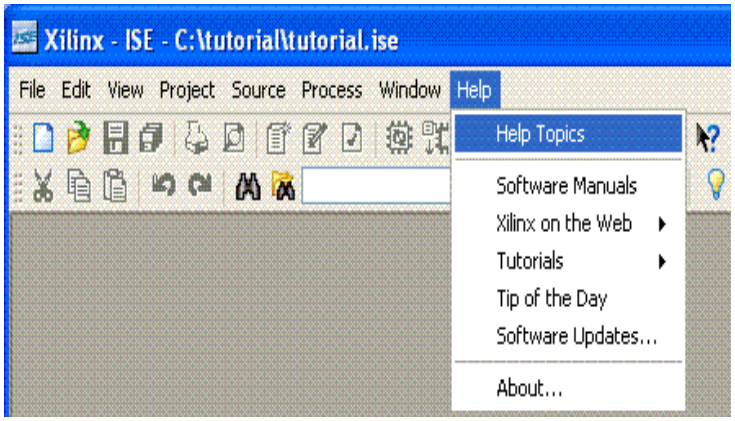
INTRODUCTION TO XILINX

Starting the ISE Software:

- To start ISE, double-click the desktop icon,or start ISE from the Start menu by selecting:
- **Start → All Programs → Xilinx ISE 8.2i → Project Navigator**
- ***Note:*** Your start-up path is set during the installation process and may differ from the one above.

Accessing Help

- At any time during the tutorial, you can access online help for additional information
- about the ISE software and related tools.
- To open Help, do either of the following:
- Press **F1** to view Help for the specific tool or function that you have selected or highlighted. Launch the **ISE Help Contents** from the Help menu. It contains information about creating and maintaining your complete design flow in ISE.



Create a New Project:

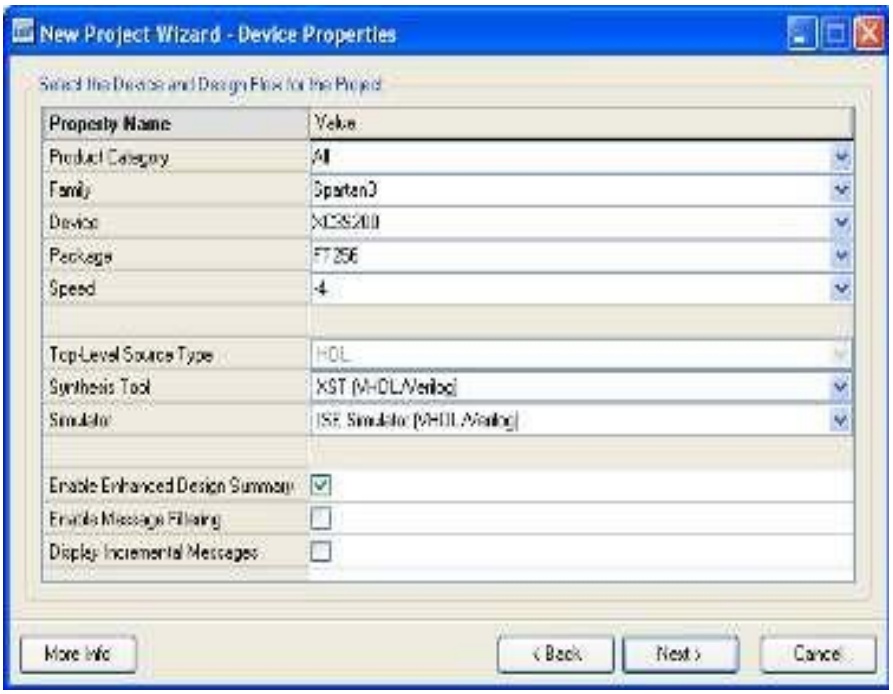
Create a new ISE project which will target the FPGA device on the Spartan-3 Startup Kit demo board.

To create a new project:

1. Select **File** > **New Project...** The New Project Wizard appears.
2. Type **tutorial** in the Project Name field.
3. Enter or browse to a location (directory path) for the new project. A tutorial subdirectory is created automatically.
4. Verify that **HDL** is selected from the Top-Level Source Type list.
5. Click **Next** to move to the device properties page.
6. Fill in the properties in the table as shown below:

♦ Product Category: **All**

♦ Family: **Spartan3**
♦ Device: **XC3S200**
♦ Package: **FT256**
♦ Speed Grade: **-4**
♦ Top-Level Module Type: **HDL**
♦ Synthesis Tool: **XST (VHDL/Verilog)**
♦ Simulator: **ISE Simulator (VHDL/Verilog)**
♦ Verify that **Enable Enhanced Design Summary** is selected.
7.Leave the default values in the remaining fields.
8. Click **Next** to proceed to the Create New Source window in the New Project Wizard. At the end of the next section, your new project will be complete.
9.When the table is complete, your project properties will look like the following:



Create an HDL Source:

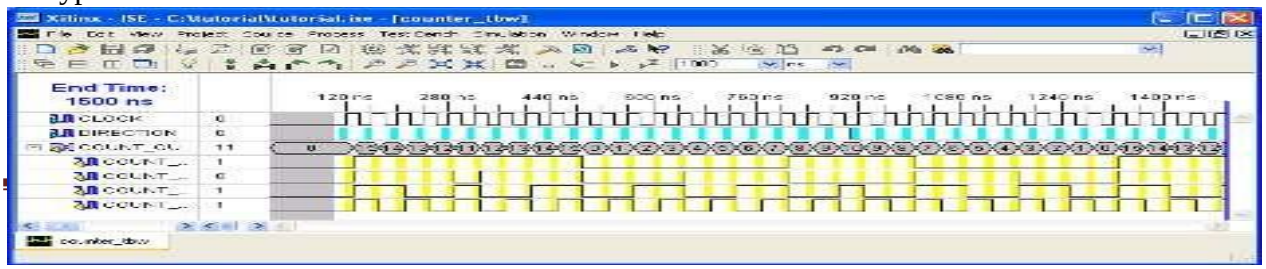
In this section, you will create the top-level HDL file for your design. Determine the language that you wish to use for the tutorial. Then, continue either to the "Creating a VHDL Source" section below, or skip to the "Creating a Verilog Source" section.

Create a VHDL source file for the project as follows:

1. Click the **New Source** button in the New Project Wizard.

2. Select **VHDL Module** as the source type.

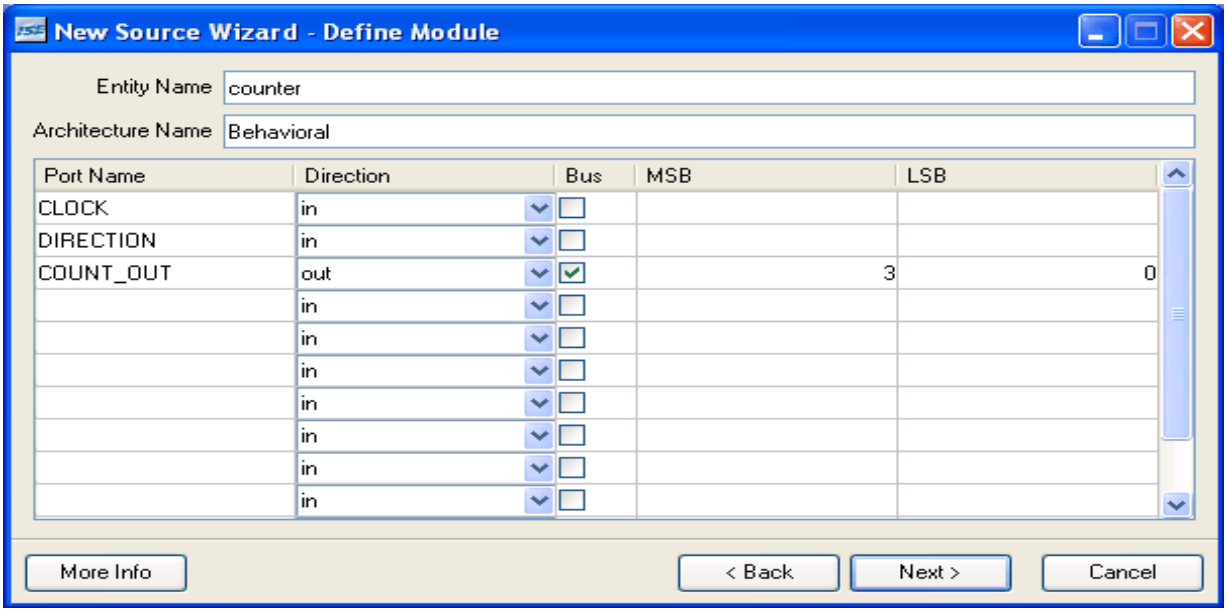
3. Type in the file name **counter**.



4. Verify that the **Add to project** checkbox is selecte

5. Click **Next**.

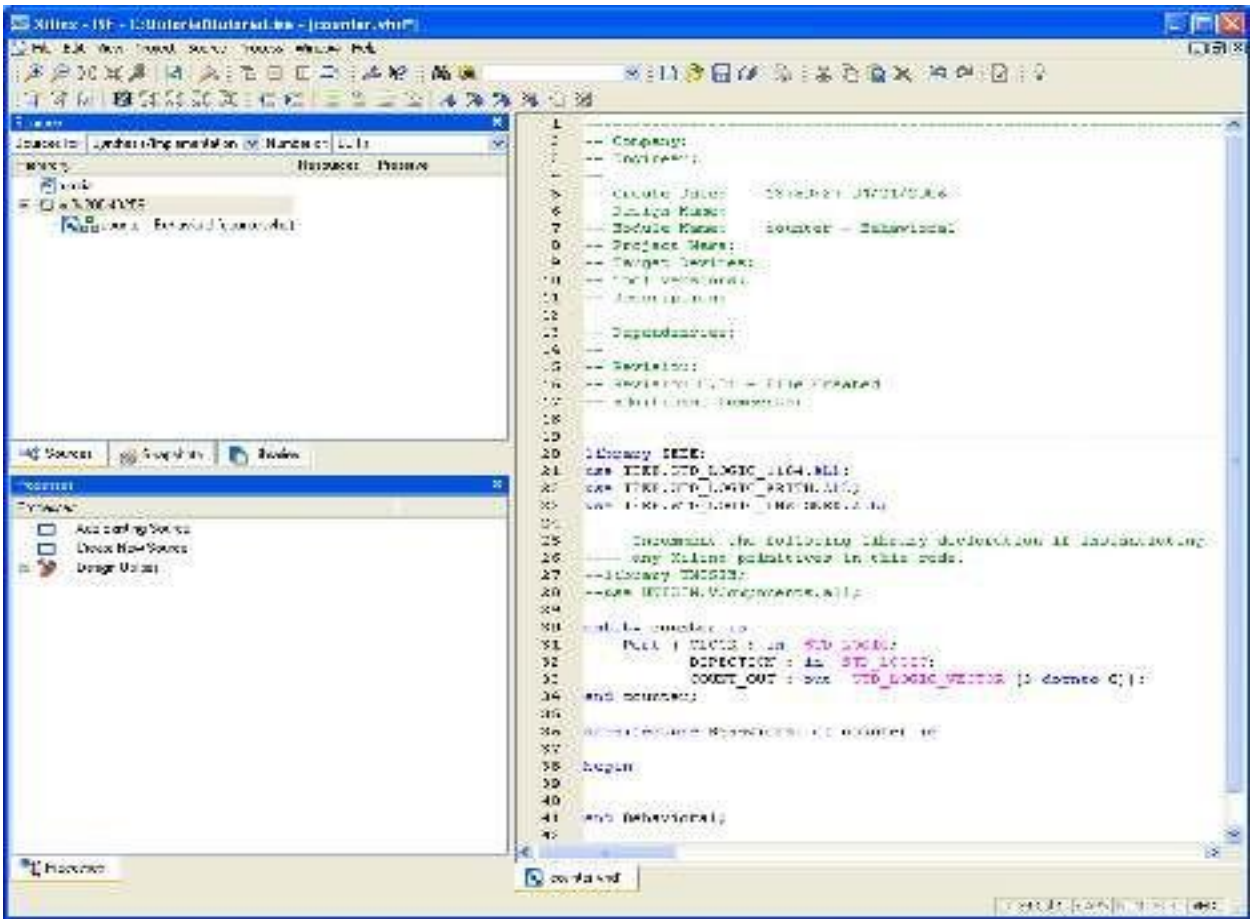
6. Declare the ports for the counter design by filling in the port information as shown below:



7. Click **Next**, then **Finish** in the New Source Information dialog box to complete the new source file template.

8. Click **Next**, then **Next**, then **Finish**.

The source file containing the entity/architecture pair displays in the Workspace, and the counter displays in the Source tab, as shown below:



Using Language Templates (VHDL):

The next step in creating the new source is to add the behavioral description for the counter. To do this you will use a simple counter code example from the ISE Language

Templates and customize it for the program design.

1. Place the cursor just below the begin statement within the counter architecture.

2. Open the Language Templates by selecting **Edit → Language Templates…**

Note: You can tile the Language Templates and the counter file by selecting **Window → Tile**

Vertically to make them both visible.

3. Using the "+" symbol, browse to the following code example:

VHDL → Synthesis Constructs → Coding Examples → Counters → Binary →

Up/Down Counters → Simple Counter

4. With Simple Counter selected, select **Edit → Use in File**, or select the **Use Template in File** toolbar button. This step copies the template into the counter source file.

5. Close the Language Templates.

Final Editing of the VHDL Source:

1. Add the following signal declaration to handle the feedback of the counter output below the architecture declaration and above the first begin statement:

signal count_int : std_logic_vector(3 downto 0) := "0000";

2. Customize the source file for the counter design by replacing the port and signal name placeholders with the actual ones as follows:

♦ replace all occurrences of <clock> with CLOCK

♦ replace all occurrences of <count_direction> with DIRECTION

♦ replace all occurrences of <count> with count_int

3. Add the following line below the end process; statement:

COUNT_OUT <= count_int;

4. Save the file by selecting **File → Save**.

When you are finished, the counter source file will look like the following:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;
```

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- Uncomment the following library declaration if instantiating
-- any Xilinx primitive in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
entity counter is
Port ( CLOCK : in STD_LOGIC;
DIRECTION : in STD_LOGIC;
COUNT_OUT : out STD_LOGIC_VECTOR (3 downto 0));
end counter;
architecture Behavioral of counter is
signal count_int : std_logic_vector(3 downto 0) := "0000";
begin
process (CLOCK)
begin
if CLOCK='1' and CLOCK'event then
if DIRECTION='1' then
if count_int < ”1111” then
count_int <= count_int + 1;
else
count_int<=”0000”;
end if;
else
count_int <= count_int - 1;
end if;
end if;
end process;
COUNT_OUT <= count_int;
end Behavioral;
```

Checking the Syntax of the New Counter Module:

When the source files are complete, check the syntax of the design to find errors and typos.

1. Verify that **Synthesis/Implementation** is selected from the drop-down list in the Sources window.
2. Select the **counter** design source in the Sources window to display the related processes in the Processes window.
3. Click the "**+**" next to the Synthesize-XST process to expand the process group. Double-click the **Check Syntax** process.
4. ***Note:*** You must correct any errors found in your source files. You can check for errors in the Console tab of the Transcript window. If you continue without valid syntax, you will not be able to simulate or synthesize your design.
5. Close the HDL file.

Design Simulation:

Verifying Functionality using Behavioral Simulation

Create a test bench waveform containing input stimulus you can use to verify the functionality of the counter module. The test bench waveform is a graphical view of a test bench.

Create the test bench waveform as follows:

1. Select the **counter** HDL file in the Sources window.
2. Create a new test bench source by selecting **Project → New Source**.
3. In the New Source Wizard, select **Test Bench WaveForm** as the source type, and type **counter_tbw** in the File Name field.
4. Click **Next**.
5. The Associated Source page shows that you are associating the test bench waveform with the source file counter. Click **Next**.
6. The Summary page shows that the source will be added to the project, and it displays the source directory, type and name. Click **Finish**.
7. You need to set the clock frequency, setup time and output delay times in the Initialize Timing dialog box before the test bench waveform editing window opens.

The requirements for this design are the following:

♦ The counter must operate correctly with an input clock frequency = 25 MHz.
♦ The DIRECTION input will be valid 10 ns before the rising edge of CLOCK.
♦ The output (COUNT_OUT) must be valid 10 ns after the rising edge of CLOCK.

The design requirements correspond with the values below.

Fill in the fields in the Initialize Timing dialog box with the following information:

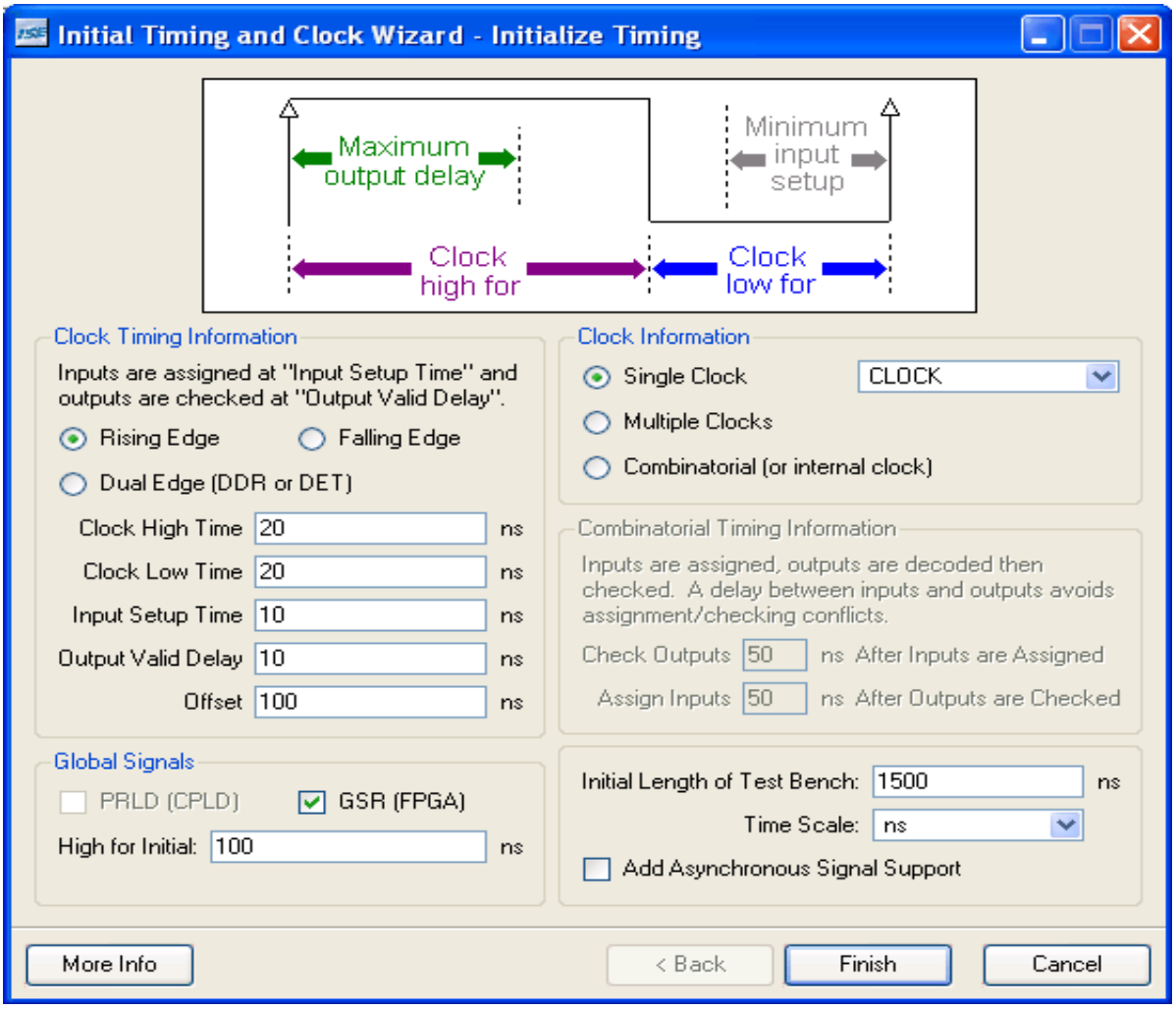
♦ Clock Time High: **20** ns.
♦ Clock Time Low: **20** ns.

♦ Input Setup Time: **10** ns.
♦ Output Valid Delay: **10** ns.
♦ Offset: **0** ns.
♦ Global Signals: **GSR (FPGA)**

Note: When GSR(FPGA) is enabled, 100 ns. is added to the Offset value automatically.

♦ Initial Length of Test Bench: **1500** ns.

Leave the default values in the remaining fields.



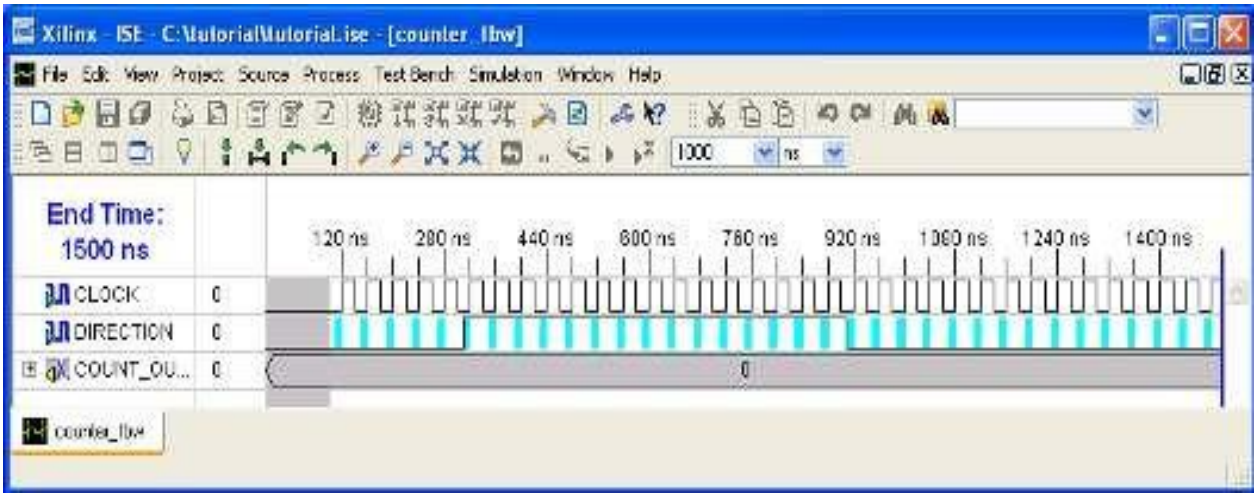
8. Click **Finish** to complete the timing initialization.

9. The blue shaded areas that precede the rising edge of the CLOCK correspond to the Input Setup Time in the Initialize Timing dialog box. Toggle the DIRECTION port to define the input stimulus for the counter design as follows:

♦ Click on the blue cell at approximately the **300** ns to assert DIRECTION high so that the counter will count up.

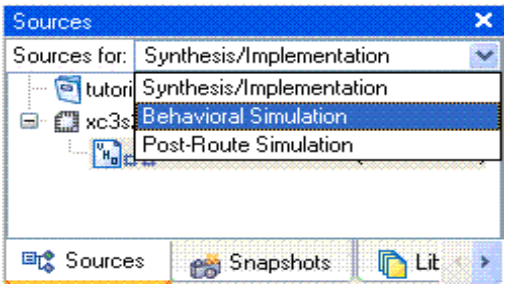
♦ Click on the blue cell at approximately the **900** ns to assert DIRECTION high so that the counter will count down.

Note: For more accurate alignment, you can use the **Zoom In** and **Zoom Out** toolbar buttons.



10. Save the waveform.

11. In the Sources window, select the **Behavioral Simulation** view to see that the test bench waveform file is automatically added to your project.



12. Close the test bench waveform.

Create a Self-Checking Test Bench Waveform:

Add the expected output values to finish creating the test bench waveform. This transforms the test bench waveform into a self-checking test bench waveform. The key benefit to a self-checking test bench waveform is that it compares the desired and actual output values and flags errors in your design as it goes through the various transformations, from behavioral HDL to the device specific representation.

To create a self-checking test bench, edit output values manually, or run the Generate Expected Results process to create them automatically. If you run the Generate Expected Results process, visually inspect the output values to see if they are the ones you expected for the given set of input values.

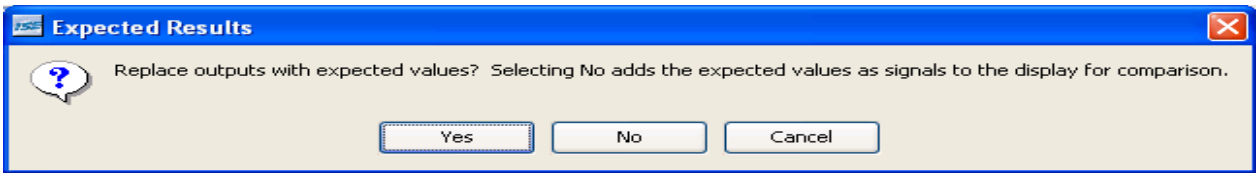
To create the self-checking test bench waveform automatically, do the following:

1. Verify that **Behavioral Simulation** is selected from the drop-down list in the Sources window.

2. Select the **counter_tbw** file in the Sources window.

3. In the Processes tab, click the "+" to expand the Xilinx ISE Simulator process and double-click the **Generate Expected Simulation Results** process. This process simulates the design in a background process.

4. The **Expected Results** dialog box opens. Select **Yes** to annotate the results to the test bench.



5. Click the "+" to expand the COUNT_OUT bus and view the transitions that correspond to the Output Delay value (yellow cells) specified in the Initialize Timing dialog box.

6. Save the test bench waveform and close it. You have now created a self-checking test bench waveform. Simulating Design Functionality

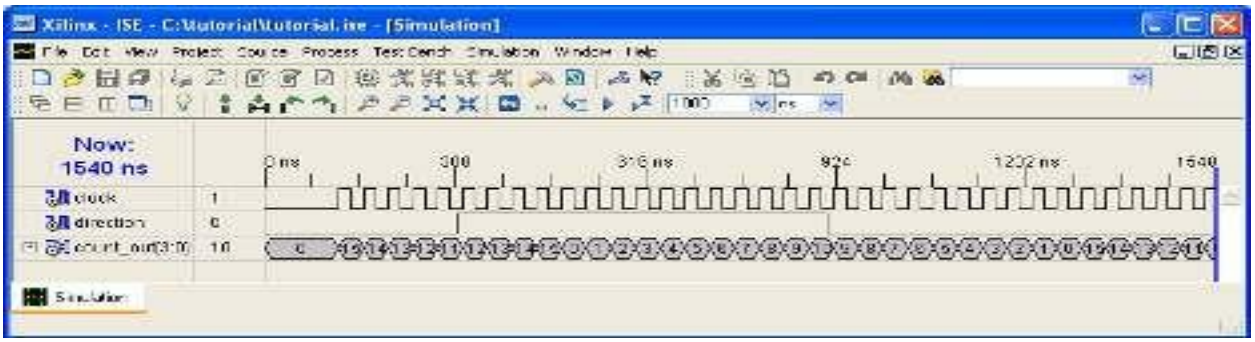
Verify that the counter design functions as you expect by performing behavior simulation as follows:

1. Verify that **Behavioral Simulation** and **counter_tbw** are selected in the Sources window.

2. In the **Processes** tab, click the "+" to expand the Xilinx ISE Simulator process and double-click the **Simulate Behavioral Model** process. The ISE Simulator opens and runs the simulation to the end of the test bench.

3. To view your simulation results, select the **Simulation** tab and zoom in on the transitions.

The simulation waveform results will look like the following:



Experiment No: 1 **REALIZATION OF LOGIC GATES**

AIM:

To write a VHDL program for Logic gates and simulate them by using XILINX 9.2i Soft ware.

SOFTWARE:

1. ILINX 9.2i
2. ISE Simulator

PROGRAM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity gates is
   Port ( a : in STD_LOGIC;
          b : in  STD_LOGIC;
          c,d,e,f,g,h,i : out STD_LOGIC);
end gates;

architecture Behavioral of gates is
begin
process(a,b)
begin
c<=a and b;
d<=a or b;
e<=not a;
f<=a nand b;
g<=a nor b;
h<=a xor b;
i<=a xnor b;
end process;
end Behavioral;
```

BLOCK DIAGRAM:

RTL SCHEMATIC DIAGRAM:

OUTPUT WAVEFORMS (AFTER SIMULATION):

TRUTH TABLE:

a	b	c=a.b	d=a+b	e=a'	f=(a.b)'	g=(a+b)'	h=a xor b	i=aΘb
0	0	0	1	1	1	0	0	1
0	1	0	1	1	1	0	1	0
1	0	0	1	0	1	0	1	0
1	1	1	0	0	0	1	0	1

PROCEDURE:

1. Start the Xilinx ISE software by Double clicking on the Icon or
   Start → All Programs → Xilinx ISE 9.2i → Project Navigator
2. Create a new project by Selecting File > New Project...
3. Create a VHDL source file, then, continue either to the "Creating a VHDL Source" section below.
4. The next step in creating the new source is to add the behavioral description for the program.
5. Place the cursor just below the begin statement within the program architecture.
6. Save the file by selecting File → Save.
7. When the source files are complete, check the syntax of the design to find errors and types.
8. Verify the Functionality using Behavioral Simulation.
9. Create a test bench waveform containing input stimulus you can use to verify the functionality of the program module.
10. Save the waveform.

RESULT:

Experiment No: 2 **Design of Full Adder using 3 modeling styles**

ABSTRACT: To study and simulate Full adder circuit in three different modeling styles using VHDL.

THEORY: A full adder is a combinational circuit that forms the arithmetic sum of input; it consists of three inputs and two outputs. A full adder is useful to add three bits at a time but a half adder cannot do so. In full adder sum output will be taken from X-OR Gate, carry output will be taken from OR Gate.

PROCEDURE:

- The Full adder Design is entered through VHDL.
- The design is simulated by applying test vectors-a,b, c and observe the output sum, carry.
- After simulation obtain the RTL, technology schematics and synthesis report.
- It is required to lock the pins and give timing constraints.
- Implement the design by passing the design by various stages by mapping, time analysis and bit stream. For locking the pins write UCF file before implementation and guide the same through option set control files. Output can be directly programmed into target device FPGA.

VHDL PROGRAM FOR FULLADDER:

BEHAVIORAL STYLE

```
entity  fulladderbeh is

        Port ( a,b,c : in std_logic;

        sum,carry : out std_logic);

        end fulladrbeh;

architecture Behavioral of fulladrbeh is

begin

        process( a,b,c)

begin

if(a='0' and b='0' and c='0') then sum<='0';

carry<='0';

elsif(a='0' and b='0' and c='1') then sum<='1';

carry<='0';
```

```
elsif(a='0' and b='1' and c='0') then sum<='1';
carry<='0';
elsif(a='0' and b='1' and c='1') then sum<='0';
```

TIMING DIAGRAM:

```
carry<='1';
elsif(a='1' and b='0' and c='0') then sum<='1';
carry<='0';
elsif(a='1' and b='0' and c='1') then sum<='0';
carry<='1';
elsif(a='1' and b='1' and c='0') then sum<='0';
carry<='1';
else
sum<='1'; carry<='1';
end if;
end process;
end Behavioral;
```

DATA FLOW MODELING

```
entity fulladder is
        Port ( a,b,c : in std_logic;
        s,cout : out std_logic);
        end fulladr;
architecture data of fulladr is
begin
 sum<=a xor b xor cin;
 cout<= ( a and b) or ( b and cin) or ( cin and a);
end data;
```

STRUCTURAL STYLE

```
entity fullstru is
Port ( a,b,cin : in std_logic;
     sum,carry : out std_logic);
end fullstru;
architecture structural of fullstru is
 signal c1,c2,c3:std_logic;
 component xor_3
 port(x,y,z:in std_logic;
u:out std_logic);
end component;
 component and_2
 port(l,m:in std_logic;
 n:out std_logic);
end component;
component or_3
```

```
 port(p,q,r:in std_logic;
 s:out std_logic);
end component;
begin
X1: xor_3 port map ( a, b, cin,sum);
A1: and_2 port map (a, b, c1);
A2: and_2 port map (b,cin,c2);
A3: and_2 port map (a,cin,c3);
O1: or_3 port map (c1,c2,c3,carry);
end structural;
```

LINKUP FOR STRUCTURAL MODELLING

```
//and gate//
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity and_2 is
 Port ( l,m : in std_logic;
 n : out std_logic);
end and2;
architecture dataf of and2 is
begin
n<=l and m;
end dataf;
//or gate//
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity or_3 is
 Port ( p,q,r : in std_logic;
```

```
 s : out std_logic);
end or3;
architecture dat of or_3 is
begin
 s<= p or q or r;
end dat;
//xor gate//
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity xor_3 is
 Port ( x,y,z : in std_logic;
 u : out std_logic);
end xor_3;
architecture dat of xor_3 is
begin

 u<=x xor y xor z;
end dat;
```

RESULT:

Experiment No:3

3 TO 8 DECODER -74138

AIM:

To write a VHDL program for 3 to 8 Decoder and simulate it by using XILINX9.2i Soft ware.

SOFTWARE:

1. ILINX 9.2i
2. ISE Simulator

PROGRAM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity dc1 is
   Port ( en : in  STD_LOGIC;
          x : in  STD_LOGIC_VECTOR (2 downto 0);
          y : out STD_LOGIC_VECTOR (7 downto 0));
end dc1;

architecture Behavioral of dc1 is
begin
process(en,x)
begin
if en='0' then y<="00000000";
else
case x is
when "000"=>y<="00000001";
when "001"=>y<="00000010";
when "010"=>y<="00000100";
when "011"=>y<="00001000";
when "100"=>y<="00010000";
when "101"=>y<="00100000";
when "110"=>y<="01000000";
when "111"=>y<="10000000";
when others=>null;
end case;
end if;
end process;
end Behavioral;
```

BLOCK DIAGRAM:

RTL SCHEMATIC DIAGRAM:

OUTPUT WAVEFORMS (AFTER SIMULATION):

TRUTH TABLE:

en	X2	X1	X0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

PROCEDURE:

1. Start the Xilinx ISE software by Double clicking on the Icon or

   Start → All Programs → Xilinx ISE 9.2i → Project Navigator

2. Create a new project by Selecting File > New Project...

3. Create a VHDL source file, then, continue either to the “Creating a VHDL Source” section below.

4. The next step in creating the new source is to add the behavioral description for the program.

5. Place the cursor just below the begin statement within the program architecture.

6. Save the file by selecting File → Save.

7. When the source files are complete, check the syntax of the design to find errors and types.

8. Verify the Functionality using Behavioral Simulation.

9. Create a test bench waveform containing input stimulus you can use to verify the functionality of the program module.

10. Save the waveform.

RESULT:

The VHDL program for 3 to 8 Decoder is written and simulated by using XLINX9.2i version and the Output is verified.

Experiment No: 4

8 to 3 Encoder (with and without priority)

ABSTRACT: To study and simulate design of 8 to 3 Encoder (with and without priority) using VHDL.

THEORY: 8 to 3 encoder has 8 inputs and only one output based on the select inputs (2^n) stress out one output n. Priority encoders are available in standard IC form and the TTL 74LS148 is an 8-to-3 bit priority encoder which has eight active LOW (logic "0") inputs and provides a 3-bit code of the highest ranked input at its output.

PROCEDURE:

- The 8 to 3 encoder Design is entered through VHDL.
- The design is simulated by applying test vectors- ENABLE_L, D_IN and observing output D-OUT.
- After simulation obtain the RTL, technology schematics and synthesis report.
- It is required to lock the pins and give timing constraints.
- Implement the design by passing the design by various stages by mapping, time analysis and bit stream. For locking the pins write UCF file before implementation and guide the same through option set control files. Output can be directly programmed into target device FPGA.

VHDL PROGRAM FOR 8 TO 3 ENCODER (with out priority):

```
LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

USE IEEE.STD_LOGIC_ARITH.ALL;

USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ENCODER8_3 IS

PORT ( ENABLE_L : IN STD_LOGIC;

D_IN: IN STD_LOGIC_VECTOR(7 DOWNTO 0);

D_OUT: OUT STD_LOGIC_VECTOR(2 DOWNTO 0) );

END ENCODER8_3;

ARCHITECTURE ENCODER_ARCH OF ENCODER8_3 IS

BEGIN

PROCESS(ENABLE_L,D_IN)

BEGIN
```

```
IF ( ENABLE_L = '1') THEN
D_OUT <= "000";

ELSE

CASE D_IN IS

WHEN "00000001" => D_OUT <= "000";

WHEN "00000010" => D_OUT <= "001";

WHEN "00000100" => D_OUT <= "010";

WHEN "00001000" => D_OUT <= "011";

WHEN "00010000" => D_OUT <= "100";

WHEN "00100000" => D_OUT <= "101";

WHEN "01000000" => D_OUT <= "110";

WHEN “10000000” => D_OUT <= "111";

WHEN OTHERS => NULL";

END CASE;

END IF;

END PROCESS;

END ENCODER_ARCH;
```

LOGIC DIAGRAM:

INTERNAL DIAGRAM:

VHDL PROGRAM FOR 8 TO 3 ENCODER (with out priority):

```
Library IEEE;
Use IEEE.std_logic_1164.all;
Entity V74x148 is
Port (  EI_L: in std_logic;
        I_L: in std_logic_vector(7 downto 0);
        A_L: out std_logic_vector(2 downto 0);
        EO_L,GS_L: out std_logic);
End V74x148;
Architecture behavioral of V74x148 id
Signal EI:  std_logic;
Signal I: std_logic_vector(7 downto 0);
```

```
Signal EO,GS: std_logic;
Signal A: std_logic_vector(2 downto 0);
Begin
process(EI_L,I_L,EI,EO,GS,I,A)
variable j;integer range 7 downto 0;
begin
EI<= not EI_L;
I<= not I_L;
EO<='1';
GS<='0';
A<="000";
If (EI)='0' then EO<='0';
Else for j in 7 downto 0 loop
If I(j)='1' then
GS<='1'; EO<='0'; A<=conv_std_logic_vector(j,3);
Exit;
```

TRUTH TABLE :

INPUTS									OUTPUTS				
EI_L	**I0_L**	**I1_L**	**I2_L**	**I3_L**	**I4_L**	**I5_L**	**I6_L**	**I7_L**	**A2_L**	**A1_L**	**A0_L**	**GS_L**	**EO_L**
1	X	X	X	X	X	X	X	X	1	1	1	1	1
0	X	X	X	X	X	X	X	0	0	0	0	0	1
0	X	X	X	X	X	X	0	1	0	0	1	0	1
0	X	X	X	X	X	0	1	1	0	1	0	0	1
0	X	X	X	X	0	1	1	1	0	1	1	0	1
0	X	X	X	0	1	1	1	1	1	0	0	0	1
0	X	X	0	1	1	1	1	1	1	0	1	0	1
0	X	0	1	1	1	1	1	1	1	1	0	0	1
0	0	1	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0

End if;

End loop;

End if;

EO_L<= not EO;

GS_L<= not GS;

A_L<= not A;

End process;

End behavioral;

RESULT:

Experiment No: 5

8 x 1 Multiplexer-74151 and 1 x 4 De-multiplexer-74155

AIM:

To write a VHDL program for 8X1 Multiplexer and simulate it by using XILINX9.2i Soft ware.

SOFTWARE:

1. ILINX 9.2i
2. ISE Simulator

PROGRAM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity mul1 is
   Port ( sel : in STD_LOGIC_VECTOR (2 downto 0);
          x : in  STD_LOGIC_VECTOR (7 downto 0);
          y : out STD_LOGIC);
end mul1;

architecture Behavioral of mul1 is
begin
process(sel,x)
begin
case sel is
when "000"=>y<=x(0);
when "001"=>y<=x(1);
when "010"=>y<=x(2);
when "011"=>y<=x(3);
when "100"=>y<=x(4);
when "101"=>y<=x(5);
when "110"=>y<=x(6);
when "111"=>y<=x(7);
when  others=>null;
end case;
```

```
end process;
end Behavioral;
```

BLOCK DIAGRAM:

RTL SCHEMATIC DIAGRAM:

OUTPUT WAVEFORMS (AFTER SIMULATION):

TRUTH TABLE:

sel2	sel1	sel0	X7	X6	X5	X4	X3	X2	X1	X0	Y
0	0	0	0	0	0	0	0	0	0	1	X0
0	0	1	0	0	0	0	0	0	1	0	X1
0	1	0	0	0	0	0	0	1	0	0	X2
0	1	1	0	0	0	0	1	0	0	0	X3
1	0	0	0	0	0	1	0	0	0	0	X4
1	0	1	0	0	1	0	0	0	0	0	X5
1	1	0	0	1	0	0	0	0	0	0	X6
1	1	1	1	0	0	0	0	0	0	0	X7

PROCEDURE:

1. Start the Xilinx ISE software by Double clicking on the Icon or

   Start → All Programs → Xilinx ISE 9.2i → Project Navigator

2. Create a new project by Selecting File > New Project...

3. Create a VHDL source file, then, continue either to the "Creating a VHDL Source" section below.

4. The next step in creating the new source is to add the behavioral description for the program.

5. Place the cursor just below the begin statement within the program architecture.

6. Save the file by selecting File → Save.

7. When the source files are complete, check the syntax of the design to find errors and types.

8. Verify the Functionality using Behavioral Simulation.

9. Create a test bench waveform containing input stimulus you can use to verify the functionality of the program module.

10. Save the waveform.

RESULT:

1 X 4 DE-MULTIPLEXER

AIM:
To write a VHDL program for 2X4 De-Multiplexer and simulate it by using XILINX9.2i Soft ware.

SOFTWARE:
1. ILINX 9.2i
2. ISE Simulator

PROGRAM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity dem1 is
   Port ( sel : in STD_LOGIC_VECTOR (1 downto 0);
        x : in  STD_LOGIC;
        y : out STD_LOGIC_VECTOR (3 downto 0));
end dem1;

architecture Behavioral of dem1 is
begin
process(x,sel)
begin
y<="0000";
case sel is
when "00"=>y(0)<=x;
when "01"=>y(1)<=x;
when "10"=>y(2)<=x;
when "11"=>y(3)<=x;
when others=>null;
end case;
end process;
end Behavioral;
```

BLOCK DIAGRAM:

RTL SCHEMATIC DIAGRAM:

OUTPUT WAVEFORMS (AFTER SIMULATION):

TRUTH TABLE:

sel1	sel0	X	Y3	Y2	Y1	Y0
0	0	A	0	0	0	A
0	1	A	0	0	A	0
1	0	A	0	A	0	0
1	1	A	A	0	0	0

PROCEDURE:

1. Start the Xilinx ISE software by Double clicking on the Icon or

   Start → All Programs → Xilinx ISE 9.2i → Project Navigator

2. Create a new project by Selecting File > New Project...

3. Create a VHDL source file, then, continue either to the "Creating a VHDL Source" section below.

4. The next step in creating the new source is to add the behavioral description for the program.

5. Place the cursor just below the begin statement within the program architecture.

6. Save the file by selecting File → Save.

7. When the source files are complete, check the syntax of the design to find errors and types.

8. Verify the Functionality using Behavioral Simulation.

9. Create a test bench waveform containing input stimulus you can use to verify the functionality of the program module.

10. Save the waveform.

RESULT:

Experiment No: 6

4-BIT COMPARATOR-7485

AIM:

To write a VHDL program for 4-Bit Comparator and simulate it by using XILINX9.2i Software.

SOFTWARE:

1. ILINX 9.2i
2. ISE Simulator

PROGRAM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity compare is
   Port ( A,B : in STD_LOGIC_VECTOR (3 downto 0);
          EQ,NE,GT,GE,LT,LE : out  STD_LOGIC);
end compare;

architecture Behavioral of compare is
begin
process(A,B)
begin
EQ<='0';NE<='0';GT<='0';GE<='0';LT<='0';LE<='0';
if A = B then EQ<='1';end if;
if A /= B then NE<='1';end if;
if A > B then GT<='1';end if;
if A >= B then GE<='1';end if;
if A < B then LT<='1';end if;
if A <= B then LE<='1';end if;
end process;
end Behavioral;
```

BLOCK DIAGRAM:

RTL SCHEMATIC DIAGRAM:

OUTPUT WAVEFORMS (AFTER SIMULATION):

TRUTH TABLE:

COMPARING INPUTS				OUTPUTS		
A3,B3	**A2,B2**	**A1,B1**	**A0,B0**	**A>B**	**A<B**	**A=B**
A3>B3	**X**	**X**	**X**	**1**	**0**	**0**
A3<B3	**X**	**X**	**X**	**0**	**1**	**0**
A3=B3	**A2>B2**	**X**	**X**	**1**	**0**	**0**
A3=B3	**A2<B2**	**X**	**X**	**0**	**1**	**0**
A3=B3	**A2=B2**	**A1>B1**	**X**	**1**	**0**	**0**
A3=B3	**A2=B2**	**A1<B1**	**X**	**0**	**1**	**0**
A3=B3	**A2=B2**	**A1=B1**	**A0>B0**	**1**	**0**	**0**
A3=B3	**A2=B2**	**A1=B1**	**A0<B0**	**0**	**1**	**0**
A3=B3	**A2=B2**	**A1=B1**	**A0=B0**	**0**	**0**	**1**

PROCEDURE:

1. Start the Xilinx ISE software by Double clicking on the Icon or

   Start → All Programs → Xilinx ISE 9.2i → Project Navigator

2. Create a new project by Selecting File > New Project...

3. Create a VHDL source file, then, continue either to the "Creating a VHDL Source" section below.

4. The next step in creating the new source is to add the behavioral description for the program.

5. Place the cursor just below the begin statement within the program architecture.

6. Save the file by selecting File → Save.

7. When the source files are complete, check the syntax of the design to find errors and types.

8. Verify the Functionality using Behavioral Simulation.

9. Create a test bench waveform containing input stimulus you can use to verify the functionality of the program module.

10. Save the waveform.

RESULT:

Experiment No: 7 **D-FLIPFLOP-7474**

AIM:
To write a VHDL program for D-FLIP FLOP and simulate it by using XILINX9.2i Software.

SOFTWARE:
1. ILINX 9.2i
2. ISE Simulator

PROGRAM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity dff1 is
   Port ( clk : in STD_LOGIC;
          d : in  STD_LOGIC;
          q1 : out STD_LOGIC;
          q2 : out  STD_LOGIC);
end dff1;

architecture Behavioral of dff1 is

begin
process(d,clk)
begin
if(d='0'and clk='1') then
q1<='0';
q2<='1';
else if(d='1' and clk='1') then
q1<='1';
q2<='0';
end if;
end if;
end process;
end Behavioral;
```

BLOCK DIAGRAM:

RTL SCHEMATIC DIAGRAM:

OUTPUT WAVEFORMS (AFTER SIMULATION):

TRUTH TABLE:

clk	d	Q1	Q2
0	X	U	U
1	0	0	1
1	1	1	0
0	X	Q_0	Q_0'

PROCEDURE:

1. Start the Xilinx ISE software by Double clicking on the Icon or

   Start → All Programs → Xilinx ISE 9.2i → Project Navigator

2. Create a new project by Selecting File > New Project...

3. Create a VHDL source file, then, continue either to the "Creating a VHDL Source" section below.

4. The next step in creating the new source is to add the behavioral description for the program.

5. Place the cursor just below the begin statement within the program architecture.

6. Save the file by selecting File → Save.

7. When the source files are complete, check the syntax of the design to find errors and types.

8. Verify the Functionality using Behavioral Simulation.

9. Create a test bench waveform containing input stimulus you can use to verify the functionality of the program module.

10. Save the waveform.

RESULT:

Experiment No: 8 **DECADE COUNTER-7490**

AIM:

To write a VHDL program for Decade Counter and simulate it by using XILINX9.2i Soft ware.

SOFTWARE:

1. ILINX 9.2i
2. ISE Simulator

PROGRAM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Counter is
  port (clk: in STD_LOGIC;
        reset: in STD_LOGIC;
         q: out STD_LOGIC _VECTOR(3 downto 0) );
end Counter;
architecture Behavioural of Counter is
begin
process(clk,reset)
variable qtemp: std_logic_vector(3 downto 0);
 begin
 if reset='1' then
qtemp:="0000";
else
 if   clk='1'  then
 if qtemp<9 then
 qtemp:=qtemp+1;
 else
 qtemp:="0000";
 end if;
 end if;
q<=qtemp;
end if;
end process;
end Behavioural;
```

BLOCK DIAGRAM:

RTL SCHEMATIC DIAGRAM:

OUTPUT WAVEFORMS (AFTER SIMULATION):

TRUTH TABLE:

COUNT	OUTPUTS			
	Q_D	Q_C	Q_B	Q_A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

PROCEDURE:

1. Start the Xilinx ISE software by Double clicking on the Icon or
   Start → All Programs → Xilinx ISE 9.2i → Project Navigator
2. Create a new project by Selecting File > New Project...
3. Create a VHDL source file, then, continue either to the "Creating a VHDL Source" section below.
4. The next step in creating the new source is to add the behavioral description for the program.
5. Place the cursor just below the begin statement within the program architecture.
6. Save the file by selecting File → Save.
7. When the source files are complete, check the syntax of the design to find errors and types.
8. Verify the Functionality using Behavioral Simulation.
9. Create a test bench waveform containing input stimulus you can use to verify the functionality of the program module.
10. Save the waveform.

RESULT:

Experiment No: 9

SHIFT REGISTERS-7495

AIM:
To write a VHDL program for Shift Register and simulate it by using XILINX 9.2i Soft ware.

SOFTWARE:
1. ILINX 9.2i
2. ISE Simulator

PROGRAM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity shift is
    Port ( C,SI : in STD_LOGIC;
           SO : out  STD_LOGIC);
end shift;

architecture Behavioral 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 Behavioral;
```

BLOCK DIAGRAM:

RTL SCHEMATIC DIAGRAM:

OUTPUT WAVEFORMS (AFTER SIMULATION):

PROCEDURE:

1. Start the Xilinx ISE software by Double clicking on the Icon or

   Start → All Programs → Xilinx ISE 9.2i → Project Navigator

2. Create a new project by Selecting File > New Project...

3. Create a VHDL source file, then, continue either to the "Creating a VHDL Source" section below.

4. The next step in creating the new source is to add the behavioral description for the program.

5. Place the cursor just below the begin statement within the program architecture.

6. Save the file by selecting File → Save.

7. When the source files are complete, check the syntax of the design to find errors and types.

8. Verify the Functionality using Behavioral Simulation.

9. Create a test bench waveform containing input stimulus you can use to verify the functionality of the program module.

10. Save the waveform.

RESULT:

Experiment No: 10

8-BIT SERIAL IN-PARALLEL OUT AND PARALLEL IN-SERIAL OUT

ABSTRACT: To study and simulate SIPO and PISO using VHDL.

THEORY: In **Serial In Parallel Out (SIPO) shift registers**, the data is stored into the register serially while it is retrieved from it in parallel-fashion. In **Parallel In serial Out (PISO) shift registers**, the data is stored into the register parallel while it is retrieved from it in Serial-fashion.

PROCEDURE:

- The SIPO AND PISO Design is entered through VHDL.
- The design is simulated by applying test vectors- i, clk and observing output Dataout.
- After simulation obtain the RTL, technology schematics and synthesis report.
- It is required to lock the pins and give timing constraints.
- Implement the design by passing the design by various stages by mapping, time analysis and bit stream. For locking the pins write UCF file before implementation and guide the same through option set control files. Output can be directly programmed into target device FPGA.

VHDL PROGRAM FOR SIPO:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity SIPO is
Generic(N:integer :=8);
port(sin,clk :in STD_LOGIC;
sout : out STD_LOGIC );

end SIPO;

architecture SHIFT of SIPO is

component d_flip_flop is
port(D,clk :in STD_LOGIC;
Q,nQ : out STD_LOGIC);

end component d_flip_flop;

signal Z: std_logic_vector (N downto 0);
begin
z(0)<=sin;
Q1:for I in 0 to N-1 generate
```

```
d_flip_flopx:d_flip_flop port map(clk,z(i),z(i+1),open);
end generate;

sout<=z(8);

end SHIFT;
```

LINKUP FOR STRUCTURAL MODELLING

```
//D FLIPFLOP//
entity dff is
Port ( data,clk,reset : in STD_LOGIC;output : out STD_LOGIC);
end dff;
 architecture Behavioral of dff is
begin
process(clk)
begin
if(reset='1') then output<= '0';
elsif( clk'event and clk='1') then output<= data;
end if;
end process;
end Behavioral;
```

VHDL PROGRAM FOR PISO:

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

entity parallel_in_serial_out is
    port(
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        load : in STD_LOGIC;
        din : in STD_LOGIC_VECTOR(7 downto 0);
        dout : out STD_LOGIC );
end parallel_in_serial_out;

architecture piso_arc of parallel_in_serial_out is
begin
   piso : process (clk,reset,load,din) is
   variable temp : std_logic_vector (din'range);
   begin
      if (reset='1') then
         temp := (others=>'0');
```

```
elsif (load='1') then
   temp := din ;
      elsif (rising_edge (clk)) then
          dout <= temp(7);
          temp := temp(6 downto 0) & '0';
       end if;
    end process piso;
end piso_arc;
```

PIN DIAGRAM:

INTERNAL ARCHITECTURE OF FIFO:

RESULT

Experiment No: 11

FIFO(FIRST IN FIRST OUT)

ABSTRACT: To study and simulate FIFO using VHDL.

THEORY: FIFOs (First In, First Out) are essentially memory buffers used to temporarily store data until another process is ready to read it. As their name suggests the first byte written into a FIFO will be the first one to appear on the output. Typically FIFOs are used when you have two processes that operate and a different rate. A common example is a high speed communications channel that writes a burst of data into a FIFO and then a slower communications channel that read the data as need to send it at a slower rate. The FIFO module has two settings that can be configured to adjust the width and depth of the FIFO. The *DATA_WIDTH* variable adjusts the size of the *DataIn* and *DataOut* buses so that you can write different sizes of bytes if needed and the *FIFO_DEPTH* variable adjusts how big the internal memory of the FIFO is.

PROCEDURE:

- The FIFO Design is entered through VHDL.
- The design is simulated by applying test vectors- i, clk and observing output Dataout.
- After simulation obtain the RTL, technology schematics and synthesis report.
- It is required to lock the pins and give timing constraints.
- Implement the design by passing the design by various stages by mapping, time analysis and bit stream. For locking the pins write UCF file before implementation and guide the same through option set control files. Output can be directly programmed into target device FPGA.

VHDL PROGRAM FOR FIFO:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;

entity fifo is

generic (depth : integer := 8); --depth of fifo

port (    clk : in std_logic;

        reset : in std_logic;

        enr : in std_logic; --enable read,should be '0' when not in use.

        enw : in std_logic;    --enable write,should be '0' when not in use.

        data_in : in std_logic_vector (7 downto 0);     --input data

        data_out : out std_logic_vector(7 downto 0);    --output data
```

```
 fifo_empty : out std_logic;      --set as '1' when the queue is empty
        fifo_full : out std_logic      --set as '1' when the queue is full      );
```

Timing diagram for WRITE operation:

Timing diagram for READ operation:

```
end fifo;
architecture Behavioral of fifo is
type memory_type is array (0 to depth-1) of std_logic_vector(7 downto 0);
signal memory : memory_type :=(others => (others => '0')); --memory for queue.
signal readptr,writeptr : integer := 0;  --read and write pointers.
signal empty,full : std_logic := '0';
begin
```

```
fifo_empty <= empty;
fifo_full <= full;
process(Clk,reset)
--this is the number of elements stored in fifo at a time.
--this variable is used to decide whether the fifo is empty or full.
variable num_elem : integer := 0;
begin
if(clk'event and clk='1') then
if(reset = '1') then
    data_out <= (others => '0');
    empty <= '0';
    full <= '0';
    readptr <= 0;
    writeptr <= 0;
    num_elem := 0;
elsif(enr = '1' and empty = '0') then --read
        data_out <= memory(readptr);
        readptr <= readptr + 1;
        num_elem := num_elem-1;
    end if;
    if(enw ='1' and full = '0') then    --write
        memory(writeptr) <= data_in;
        writeptr <= writeptr + 1;
        num_elem := num_elem+1;
```

Timing diagram for reset operation :

```
end if;
   --rolling over of the indices.
   if(readptr = depth-1) then      --resetting read pointer.
      readptr <= 0;
   end if;
   if(writeptr = depth-1) then       --resetting write pointer.
      writeptr <= 0;
   end if;
   --setting empty and full flags.
   if(num_elem = 0) then
      empty <= '1';
   else
      empty <= '0';
   end if;
   if(num_elem = depth) then
      full <= '1';
   else
      full <= '0';
```

```
 end if;
end if;
end process;
end Behavioral;
```

RESULT :

Experiment No: 12

MAC (Multiplier & Accumulator)

ABSTRACT: To study and simulate MAC using VHDL.

THEORY: the multiply–accumulate operation is a common step that computes the product of two numbers and adds that product to an accumulator. The hardware unit that performs the operation is known as a multiplier–accumulator (MAC, or MAC unit)

PROCEDURE:

- The MAC Design is entered through VHDL.
- The design is simulated by applying test vectors- l, m and observing output n.
- After simulation obtain the RTL, technology schematics and synthesis report.
- It is required to lock the pins and give timing constraints.
- Implement the design by passing the design by various stages by mapping, time analysis and bit stream. For locking the pins write UCF file before implementation and guide the same through option set control files. Output can be directly programmed into target device FPGA.

VHDL PROGRAM FOR MAC:

```
entity multiplier is
Port ( l,m : in STD_LOGIC_VECTOR (2 downto 0);
       n : out STD_LOGIC_VECTOR (5 downto 0));
end multiplier;.
architecture Behavioral of multiplier is
component and1
Port (  a,b : in STD_LOGIC;
         c : out STD_LOGIC);
end component;
component hadder
port( i,j : in STD_LOGIC;
      su,ca : out STD_LOGIC);
end component;
component fulladd
Port ( d,e,f : in STD_LOGIC;
         sum,carry : out STD_LOGIC);
end component;
component or1
Port ( a1,b1 : in STD_LOGIC;
         c1 : out STD_LOGIC);
end component;
signal p,q: std_logic;
signal r : std_logic_vector( 8 downto 0);
signal s : std_logic_vector ( 3 downto 0);
signal si: std_logic_vector(3 downto 1);

signal sic: std_logic_vector(3 downto 1);
signal sumor: std_logic;
begin
```

```
        l0: and1 port map ( l(0),m(0),r(0));
         l1: and1 port map ( l(1),m(0),r(1));
                                        l2: and1 port map ( l(0),m(1),r(2));
                                        l3: and1 port map ( l(2),m(0),r(3));
        l4: and1 port map ( l(1),m(1),r(4));
        l5:and1 port map ( l(0),m(2),r(5));
        l6: and1 port map ( l(2),m(1),r(6));
        l7: and1 port map ( l(1),m(2),r(7));
        l8: and1 port map ( l(2),m(2),r(8));
n(0)<= r(0);
m1: hadder port map (r(1),r(2),n(1),s(1));
m2: hadder port map (s(1),r(3),si(1),sic(1));
m3: hadder port map (r(4),r(5),si(2),sic(2));
m4: hadder port map (si(1),si(2),n(2),s(2));
m5: hadder port map (sic(1),sic(2),si(3),sic(3));m8: or1 port map (s(2),si(3),sumor);
m6: fulladd port map (sumor,r(6),r(7),n(3),s(3));
m7: fulladd port map (s(3),sic(3),r(8),n(4),n(5));
end Behavioral;
```

LINKUP FOR STRUCTURAL MODELLING

```
//AND GATE//

entity and1 is
Port ( a,b : in STD_LOGIC;c : out STD_LOGIC);
end and1;
architecture Behavioral of and1 is
begin
c<= a and b;
end Behavioral;

//OR GATE//
entity or1 is
Port ( a1,b1 : in STD_LOGIC;c1 : out STD_LOGIC);
```

```
end or1;
architecture Behavioral of or1 is
begin
c1<= a1 or b1;
end Behavioral;

//HALF ADDER//
entity hadder is
Port ( i,j : in STD_LOGIC;
       su,ca : out STD_LOGIC);
end hadder;
architecture Behavioral of hadder is
begin
su<= i xor j;ca<= i and j;
end Behavioral;

//FULL ADDER//
entity fulladd is
Port ( d,e,f : in STD_LOGIC;sum,carry : out STD_LOGIC);

end fulladd;
architecture Behavioral of fulladd is
component and1Port ( a,b : in STD_LOGIC;c : out STD_LOGIC);
end component;
begin
sum <= d xor e xor f;carry <= ((d and e) or ( e and f ) or ( f and d));
end Behavioral;
```

LOGIC DIAGRAM:

TRUTH TABLE

sel	Operation	Function	Unit
0000	y <= a	Transfer a	Arithmetic
0001	y <= a+1	Increment a	
0010	y <= a-1	Decrement a	
0011	y <= b	Transfer b	
0100	y <= b+1	Increment b	
0101	y <= b-1	Decrement b	
0110	y <= a+b	Add a and b	
0111	y <= a+b+cin	Add a and b with carry	
1000	y <= NOT a	Complement a	Logic
1001	y <= NOT b	Complement b	
1010	y <= a AND b	AND	
1011	y <= a OR b	OR	
1100	y <= a NAND b	NAND	
1101	y <= a NOR b	NOR	
1110	y <= a XOR b	XOR	
1111	y <= a XNOR b	XNOR	

RESULT:

Experiment No: 13

ALU DESIGN

AIM:

To write a VHDL program for Arithmetic and Logic Unit and simulate it by using XILINX 9.2i Soft ware.

SOFTWARE:

1. ILINX 9.2i

2. ISE Simulator

PROGRAM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
 entity alu is
   port
   (
      Input1, Input2 : in std_logic_vector(3 downto 0);
      Operation : in std_logic_vector(2 downto 0);
      Flag : out std_logic;
      Result : out std_logic_vector(3 downto 0)
   );
end entity alu;

architecture Behavioral of alu is
signal Temp: std_logic_vector(4 downto 0);
begin
process(Input1, Input2, Operation, temp) is
begin
Flag <= '0';
case Operation is
when "000" => -- res = in1 + in2, flag = carry = overflow
Temp <= (unsigned("0" & Input1) + unsigned(Input2));
Result <= temp(3 downto 0);
Flag  <= temp(4);
when "001" => -- res = |in1 - in2|, flag = 1 if in2 > in1
if (Input1 >= Input2) then
Result <= (unsigned(Input1) - unsigned(Input2));
Flag  <= '0';
else
Result <= (unsigned(Input2) - unsigned(Input1));
Flag  <= '1';
end if;
```

```
when "010" =>
Result <= Input1 and Input2;
when "011" =>
Result <= Input1 or Input2;
when "100" =>
Result <= Input1 xor Input2;
when "101" =>
Result <= not Input1;
when "110" =>
Result <= not Input2;
when others => -- res = in1 + in2 + 1, flag = 0
Temp <= (unsigned("0" & Input1)) + (unsigned(not Input2) + 1);
Result <= temp(3 downto 0);
Flag <= temp(4);
end case;
end process;
end architecture Behavioral;
```

BLOCK DIAGRAM:

RTL SCHEMATIC DIAGRAM:

OUTPUT WAVEFORMS (AFTER SIMULATION):

PROCEDURE:

1. Start the Xilinx ISE software by Double clicking on the Icon or

   Start → All Programs → Xilinx ISE 9.2i → Project Navigator

2. Create a new project by Selecting File > New Project...

3. Create a VHDL source file, then, continue either to the "Creating a VHDL Source" section below.

4. The next step in creating the new source is to add the behavioral description for the program.

5. Place the cursor just below the begin statement within the program architecture.

6. Save the file by selecting File → Save.

7. When the source files are complete, check the syntax of the design to find errors and types.

8. Verify the Functionality using Behavioral Simulation.

9. Create a test bench waveform containing input stimulus you can use to verify the functionality of the program module.

10. Save the waveform.

RESULT:

ADDITIONAL EXPXPERIMENTS

Experiment No: 1

UNIVERSAL SHIFT REGISTERS

AIM:

To write a VHDL program for Universal Shift Register and simulate it by using XILINX 9.2i Soft ware.

SOFTWARE:

1. ILINX 9.2i
2. ISE Simulator

PROGRAM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity univ1 is
   Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
          CLK,RST : in  STD_LOGIC;
          SIR,SIL : in  STD_LOGIC;
          S : in  STD_LOGIC_VECTOR (1 downto 0);
          Q : out STD_LOGIC_VECTOR (3 downto 0));
end univ1;
architecture Behavioral of univ1 is
begin
process(CLK, RST) is
variable REG : std_logic_vector(3 downto 0);
begin
if (RST = '0') then
REG := (others => '0');
elsif rising_edge(clk) then
case S is
when "11" =>
REG := D;
when "01" =>
REG := SIR & REG(3 downto 1);
when "10" =>
REG := REG(2 downto 0) & SIL;
when others =>null;
end case;
end if;
```

```
Q <= REG;
end process;
end Behavioral;
```

BLOCK DIAGRAM:

RTL SCHEMATIC DIAGRAM:

OUTPUT WAVEFORMS (AFTER SIMULATION):

TRUTH TABLE:

RESET	S1	S0	CLOCK	SIL	SIR	data				Q_A	Q_B	Q_C	Q_D
						A	B	C	D				
0	X	X	X	X	X	X	X	X	X	0	0	0	0
1	X	X	0	X	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}
1	1	1	1	X	X	a	b	c	d	a	b	c	d
1	0	1	1	X	a	X	X	X	X	a	Q_{AN}	Q_{BN}	Q_{CN}
1	1	0	1	a	X	X	X	X	X	Q_{BN}	Q_{CN}	Q_{DN}	a
1	0	0	X	X	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}

PROCEDURE:

1. Start the Xilinx ISE software by Double clicking on the Icon or

   Start → All Programs → Xilinx ISE 9.2i → Project Navigator

2. Create a new project by Selecting File > New Project...

3. Create a VHDL source file, then, continue either to the “Creating a VHDL Source” section below.

4. The next step in creating the new source is to add the behavioral description for the program.

5. Place the cursor just below the begin statement within the program architecture.

6. Save the file by selecting File → Save.

7. When the source files are complete, check the syntax of the design to find errors and types.

8. Verify the Functionality using Behavioral Simulation.

9. Create a test bench waveform containing input stimulus you can use to verify the functionality of the program module.

10. Save the waveform.

RESULT:

Experiment No: 2

4-BIT COUNTER

AIM:

To write a VHDL program for 4-Bit Counter and simulate it by using XILINX 9.2i Soft ware.

SOFTWARE:

1. ILINX 9.2i

2. ISE Simulator

PROGRAM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity count1 is
   Port ( CLOCK : in STD_LOGIC;
          DIRECTION : in  STD_LOGIC;
          COUNT : out  STD_LOGIC_VECTOR (3 downto 0));
end count1;

architecture Behavioral of count1 is
signal count_int : std_logic_vector(3 downto 0) := "0000";
begin
process (CLOCK)
begin
if CLOCK='1' and CLOCK'event then
if  count_int < "1111" then
count_int <= count_int + 1;
else
count_int<="0000";
end if;
end if;
end process;
COUNT <= count_int;
end Behavioral;
```

BLOCK DIAGRAM:

RTL SCHEMATIC DIAGRAM:

OUTPUT WAVEFORMS (AFTER SIMULATION):

TRUTH TABLE:

COUNT	OUTPUTS			
	Q_D	Q_C	Q_B	Q_A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
0	0	0	0	0

PROCEDURE:

1. Start the Xilinx ISE software by Double clicking on the Icon or

   Start → All Programs → Xilinx ISE 9.2i → Project Navigator

2. Create a new project by Selecting File > New Project...

3. Create a VHDL source file, then, continue either to the "Creating a VHDL Source" section below.

4. The next step in creating the new source is to add the behavioral description for the program.

5. Place the cursor just below the begin statement within the program architecture.

6. Save the file by selecting File → Save.

7. When the source files are complete, check the syntax of the design to find errors and types.

8. Verify the Functionality using Behavioral Simulation.

9. Create a test bench waveform containing input stimulus you can use to verify the functionality of the program module.

10. Save the waveform.

RESULT: