**MAJOR PROJECT REPORT ON**

**“Implementation of FSM using XILINX/VHDL”**

**Submitted in Partial Fulfillment of the Requirements**

**For the Award**

**Of**

**Degree of B. Tech**

**To**



**Guru Gobind Singh Indraprastha University, Delhi**

**Under the Guidance of**

**Submitted By:**

**NAME-1 NAME-2**

**NAME-3 NAME-4**



**Electronics and Communication Department**

**Guru Tegh Bahadur Institute of Technology**

**G-8 Area, Rajouri Garden, New Delhi, Delhi-64**

****

**CERTIFICATE**

We hereby certify that the work which is being presented in the project report entitled

**“Design of State Machine Using Xilinx”** in the partial fulfillment of the requirements for the awardof **Bachelor of Tech.** in **Electronics and Communication Engineering** and submitted tothe **Electronics and Communication Engineering Department** of **Guru Tegh Bahadur Institute of Technology**, New Delhi is an authentic record of my own work carried out during a period from **January 2017 to June 2017**, under the guidance of, Electronics and Communication Engineering Department. The matter presented in this project has not been submitted by us for the award of other degree elsewhere.

This is to certify that the above statement made by the candidate is correct to the best of our knowledge.

**(Project Coordinator) (Project Mentor)**

**(Project Coordinator) (H.O.D. ECE) Department)**

**(Project Coordinator)**

**ACKNOWLEDGEMENT**

We would like to acknowledge the contributions of the following people, without whose help and guidance this project would not have been completed.

We acknowledge the counsel and support of our project mentor **MR. , Assistant Professor**, Electronics and Communication Engineering Department, with respect and gratitude whose expertise, guidance, support, encouragement, and enthusiasm has made this project possible. His feedback vastly improved the quality of this report and provided an enthralling experience. I am indeed proud and fortunate to be supervised by her.

We are also thankful to **Dr., H.O.D.** Electronics and Communication Engineering Department, **Guru Tegh Bahadur Institute of Technology,** New Delhi for his constant encouragement, valuable suggestions and moral support and blessings.

We shall ever remain indebted to the faculty members of **GTBIT, New Delhi.**

Finally, yet importantly, we would like to express our heartfelt thanks to our friends/classmates for their help and wishes for successful completion of this project. This acknowledgement will remain incomplete if we fail to express our deep sense of obligation to our parents and god for their consistent blessings and encouragement.

**NAME1 NAME2**

**ROLL NO ROLL NO**

**NAME3 NAME4**

**ROLL NO ROLL NO**

**CONTENTS**

certificateiII

acknowledgementiV

introduction5

software requirements6

software used6

Xilinx 9.26

Finite STATE MACHINE13

Mealy state machine13

MOORE STATE MACHIINE15

appendix a16

code for Mealy state machine 16

code for MOORE STATE MACHIINE 20

code for DEMUX USING STATE MACHINE 24

references29

**INTRODUCTION TO VHDL:**

VHDL (Very High Speed IC Hardware description Language) is one of the standard hardware description language used to design digital systems. VHDL can be used to design the lowest level (gate level) of a digital system to the highest level (VLSI module). VHDL though being a rigid language with a standard set of rules allows the designer to use different methods of design giving different perspectives to the digital system.

Other than VHDL there are many hardware description languages available in the market for the digital designers such as Verilog, ABEL, PALASM, CUPL, and etc but VHDL and Verilog are the most widely used HDLs. The major difference between hardware description programming languages and others is the integration of time. Timing specifications are used to incorporate propagation delays present in the system.

**TYPES OF REPRESENTATION:**

VHDL representation can be seen as text file describing a digital system. The digital system can be represented in different forms such as a behavioral model or a structural model. Most commonly known as levels of abstraction, these levels help the designer to develop complex systems efficiently.

**DATAFLOW MODEL:**

Dataflow Model is convenient for illustrating asynchronous and concurrent events, where the delays represent actual hardware component delays.

**BEHAVIORAL MODEL:**

Behavioral level describes the system the way it behaves instead of a lower abstraction of its connections. Behavioral model describes the relationship between the input and output signals. The description can be a Register Transfer Level (RTL) or simple Boolean equations.

**STRUCTURAL MODEL:**

Structural level describes the systems as gates or component block interconnected to perform the desired operations. Structural level is primarily the graphical representation of the digital system and so it is closer to the actual physical representation of the system.

**VHDL PROGRAMMING**

One can design hardware in a VHDL IDE (for FPGA implementation such as Xilinx ISE, Altera Quartus, Synopsys Synplify or Mentor Graphics HDL Designer) to produce the [RTL](http://en.wikipedia.org/wiki/Register-transfer_level" \o "Register-transfer level)schematic of the desired circuit. After that, the generated schematic can be verified using simulation software which shows the waveforms of inputs and outputs of the circuit after generating the appropriate testbench.

**INTODUCTION TO XILINX**

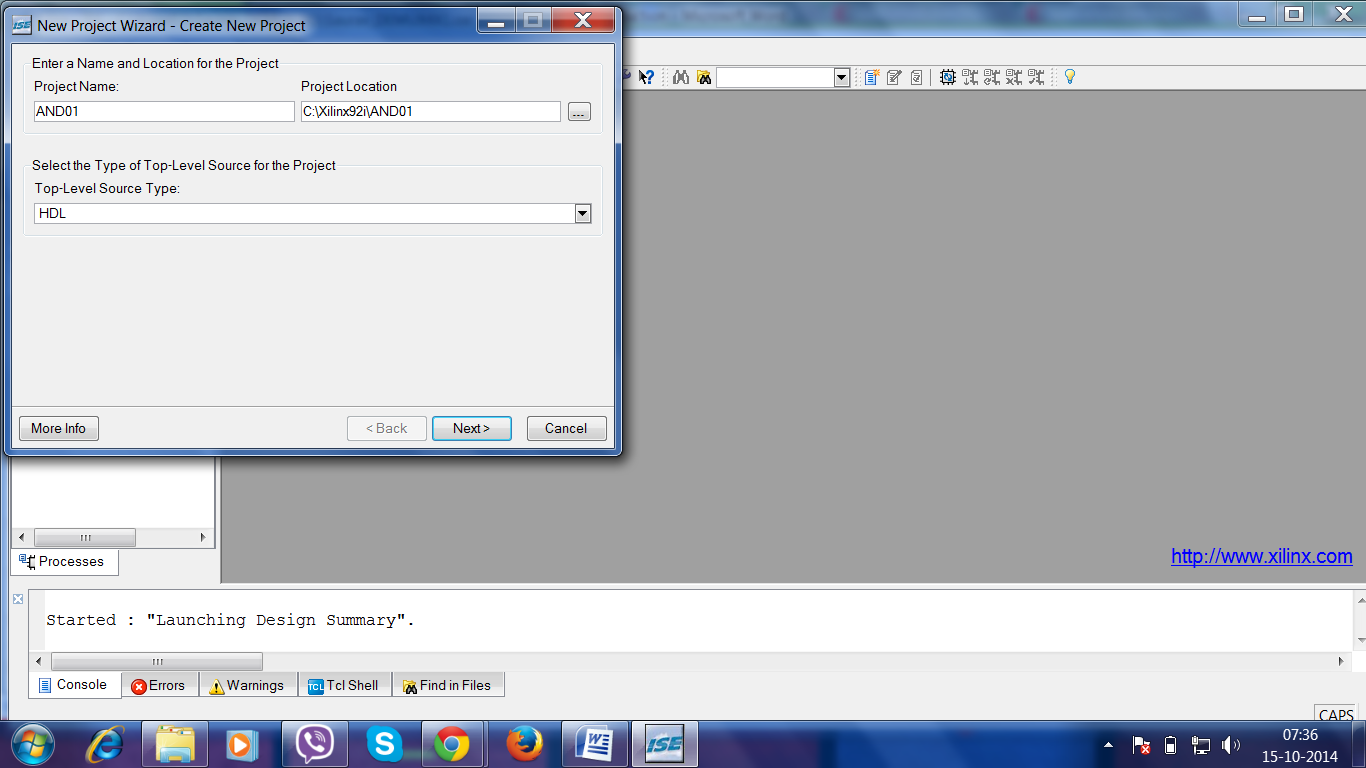
Xilinx Integrated Software Environment (ISE) is a design software suite that allows you to take your design from design entry through Xilinx device programming. The ISE Project Navigator manages and processes your design through several steps in the ISE design flow. These steps are Design Entry, Synthesis, Implementation, Simulation/Verification, and Device Configuration.

Let us see, a program to implement AND GATE in the Xilinx ISE and how to simulate it with ISE Simulator.

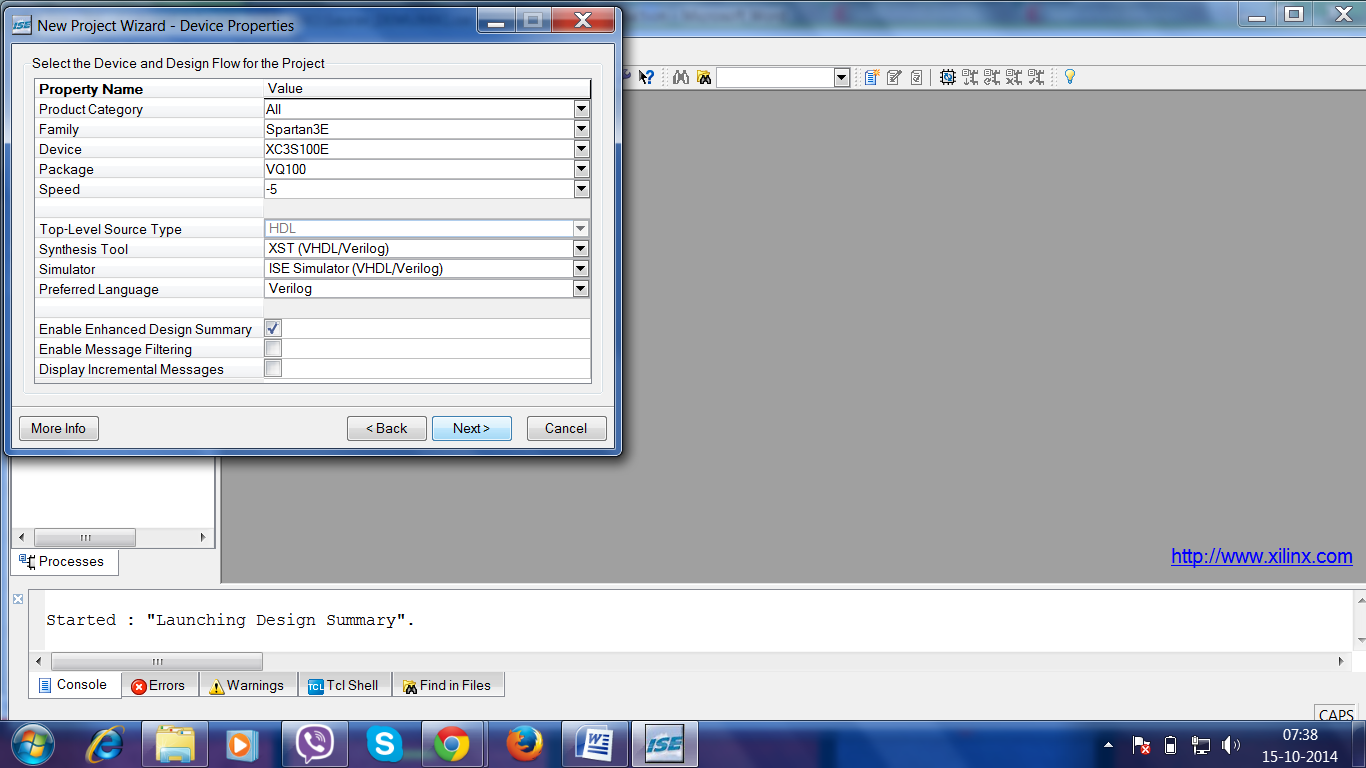
## Step-1- Create a project

In this section, you will create a new ISE project. A project is a collection of all files necessary to create and to download a design to a selected FPGA or CPLD device.

1. Select **File > New Project**. The New Project Wizard appears.
2. First, enter a location (directory path) for the new project, and then give a name for the project. For example, we name it **AND01**.



1. Select **HDL** from the Top-Level Module Type list, indicating that the top-level file in your project will be HDL, rather than Schematic or other stuffs.
2. Click on **Next** to move to the project properties page.
3. Fill in the properties in the table as shown below:

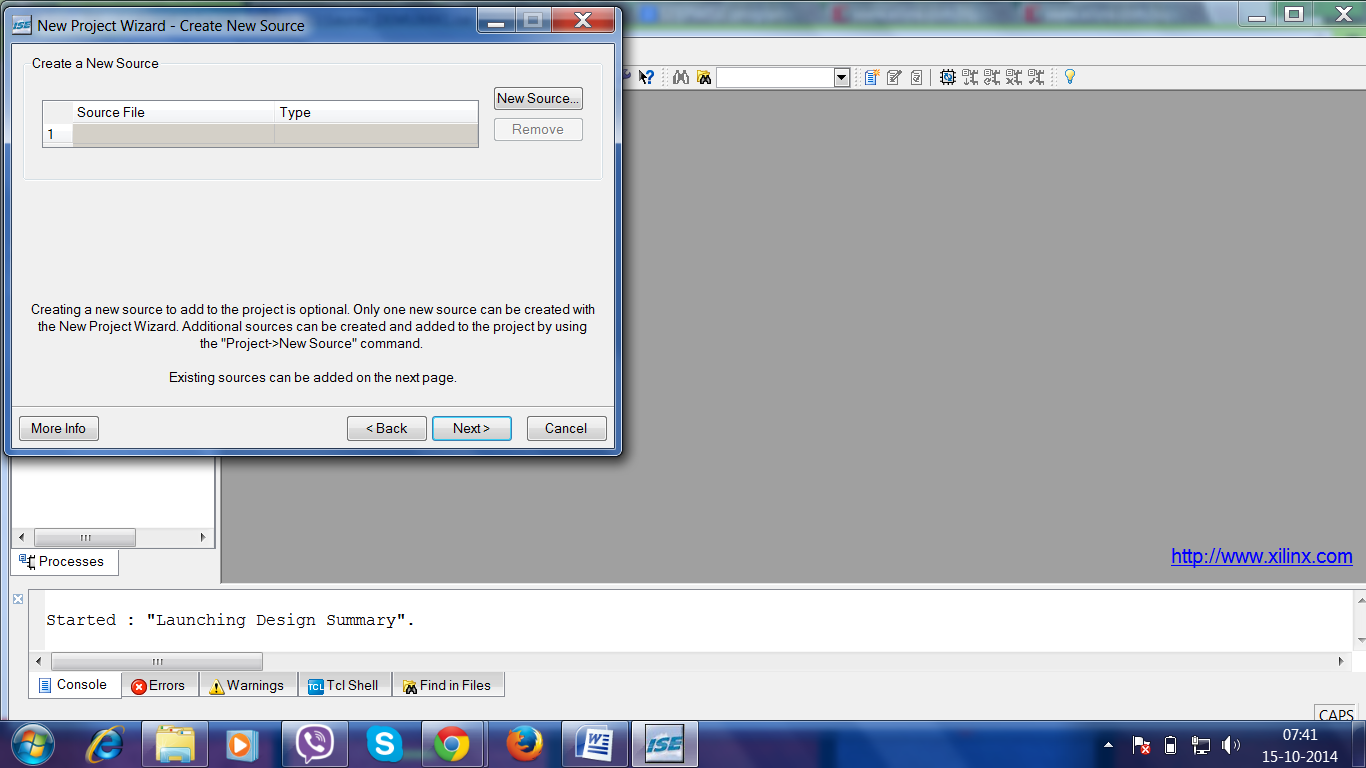
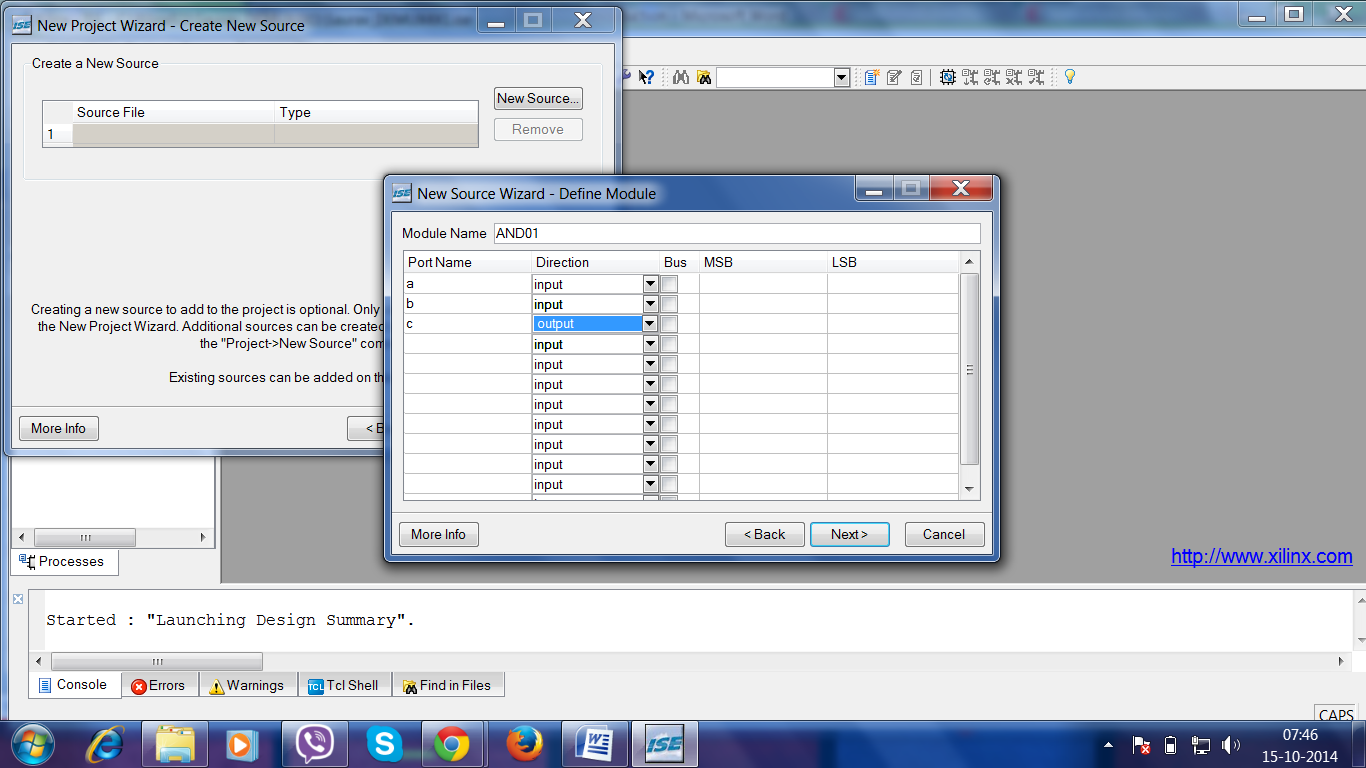


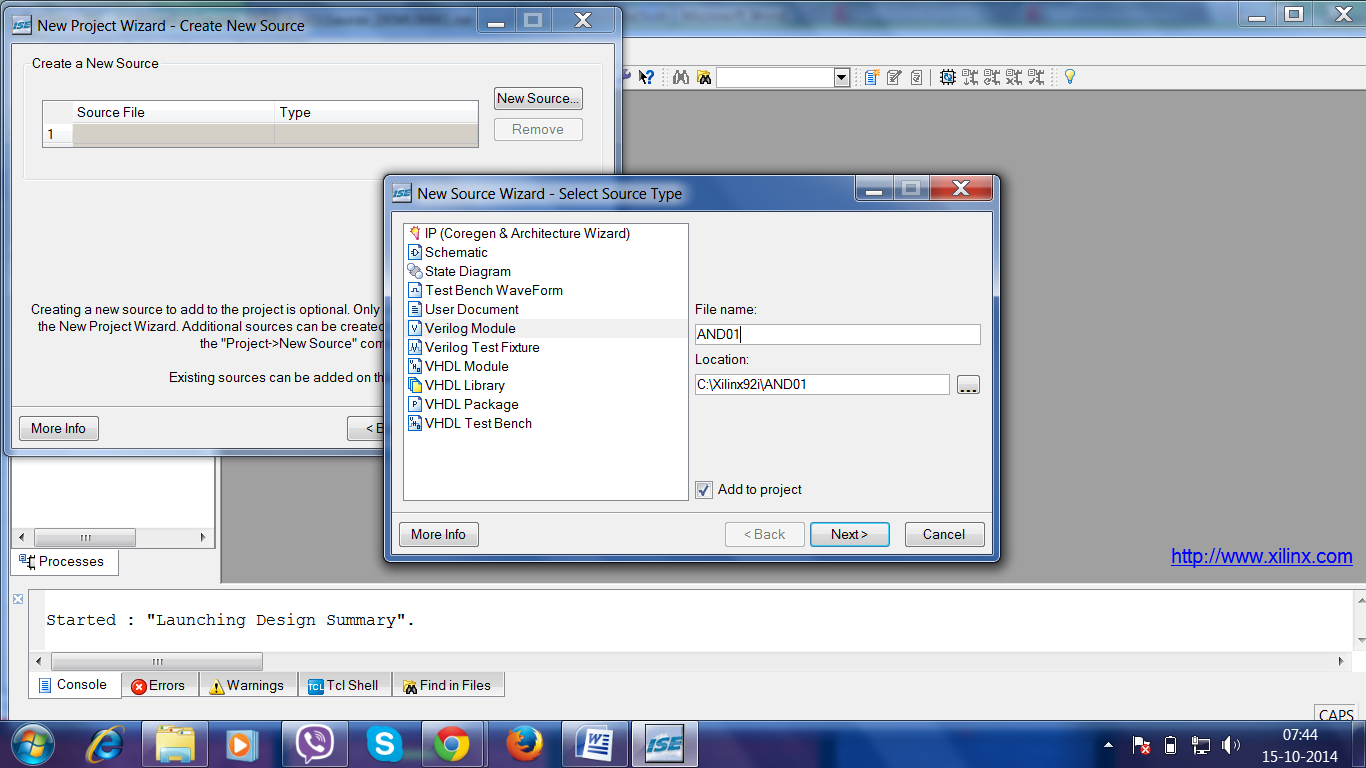
1. Click **Next** to proceed to the Create New Source window in the New Project Wizard. At the end of the next section, your new project will be created.

**Step-2-Create VHDL Source**

In this section, you will create a top-level HDL file for your design. You are going to design an up-down counter which is the same as what you did in the previous lab.

1. Click **New Source** in the New Project Wizard to add to one new source to your project.
2. Type in the file name **AND01**.
3. Select **VHDL Module** as the source type in the New Source Dialog box.
4. Verify that the **Add to Project** checkbox is selected.
5. Click **Next**.
6. Define the ports for your VHDL source.



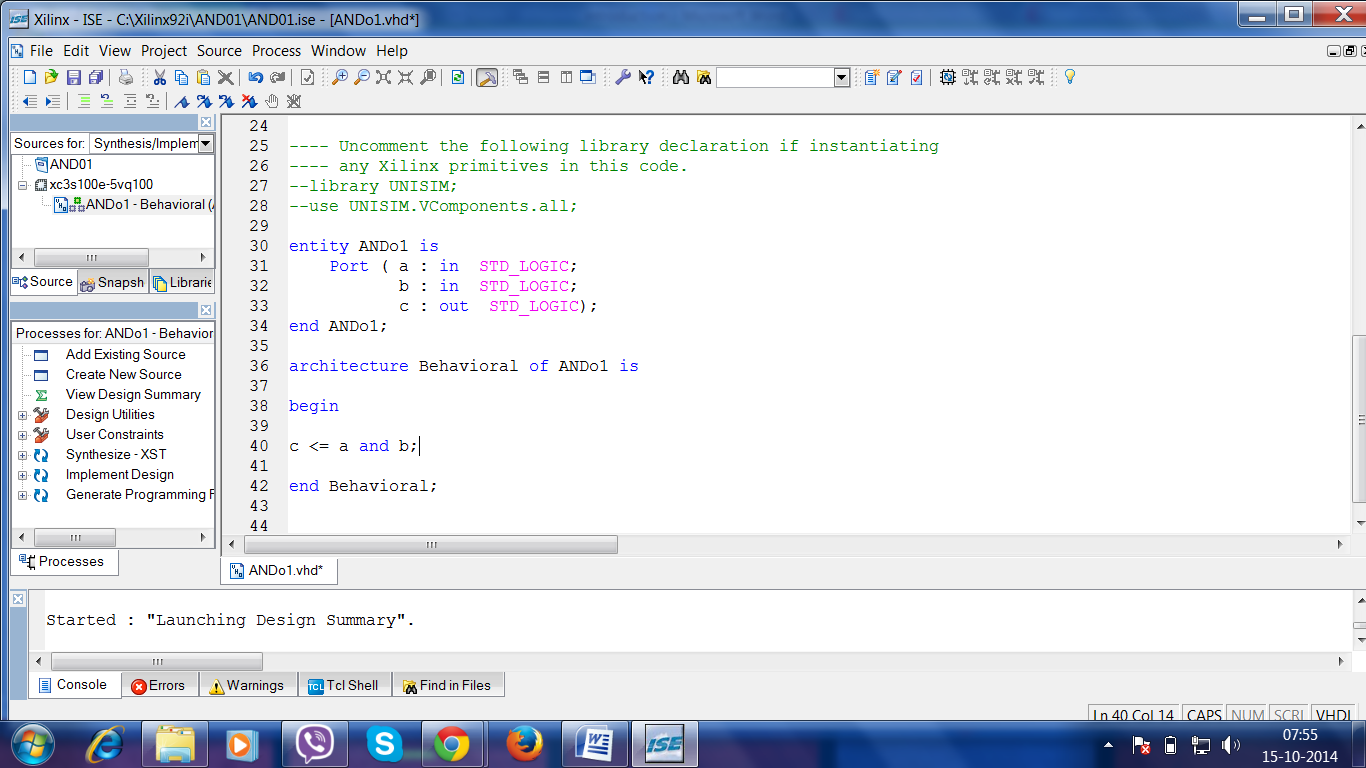


1. Click **Next** in the Define Module dialog box.
2. Click **Finish** in the New Project Wizard - Summary dialog box to complete the new source file template.
3. If the following window appears, click on "yes".
4. Click **Next** in the New Project Wizard.
5. Click **Next** again.
6. Click **Finish** in the New Project Wizard - Project Summary dialog box.

ISE creates and displays the new project in the Source in Project window and adds the **AND01.vhd** file to the project.

**Step-3-Enter and Edit VHDL Code**

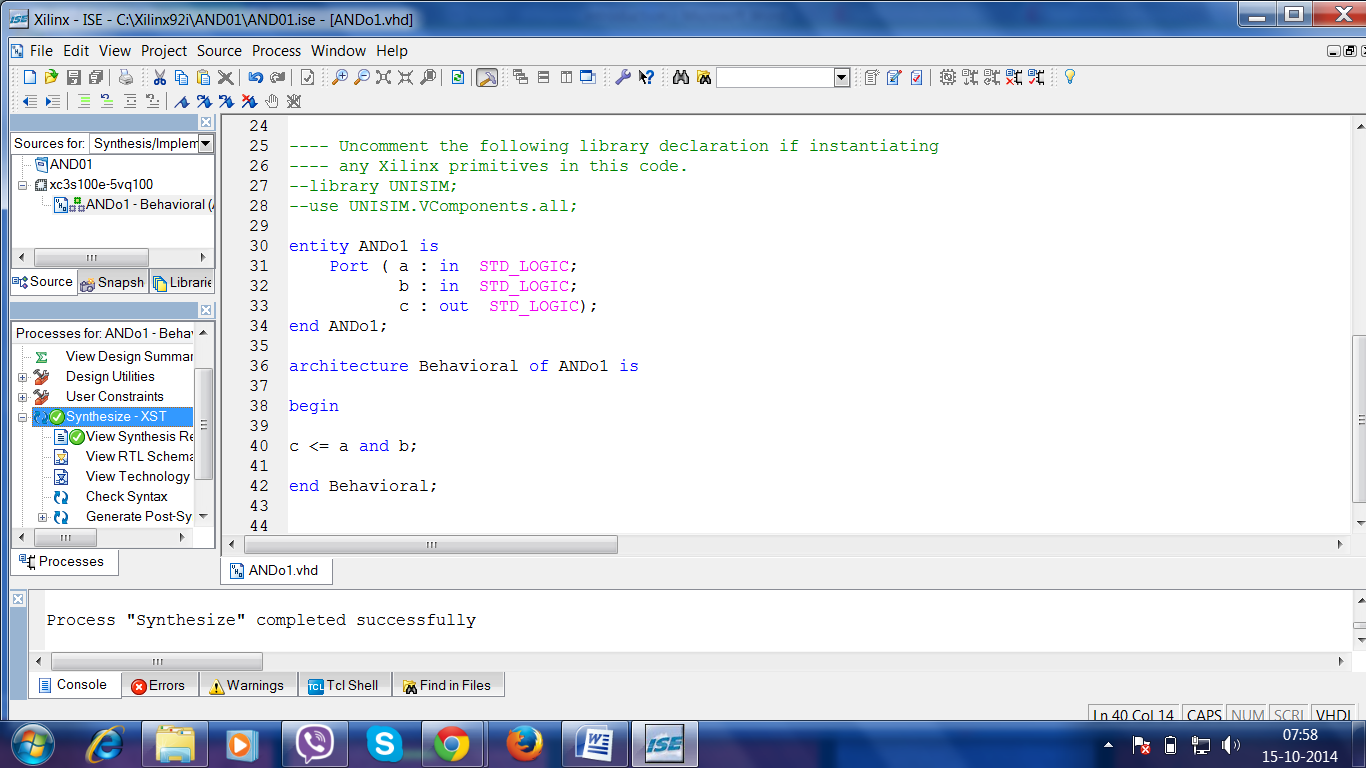
1. Save the file by selecting **File -> Save**. When you are finished, the code for the AND01 should look like the following.



**Step-4-Check the Syntax of your VHDL source - Synthesize Your Code**

When source files are complete, the next step is to check the syntax of the design. Syntax errors and typos can be found using this step.

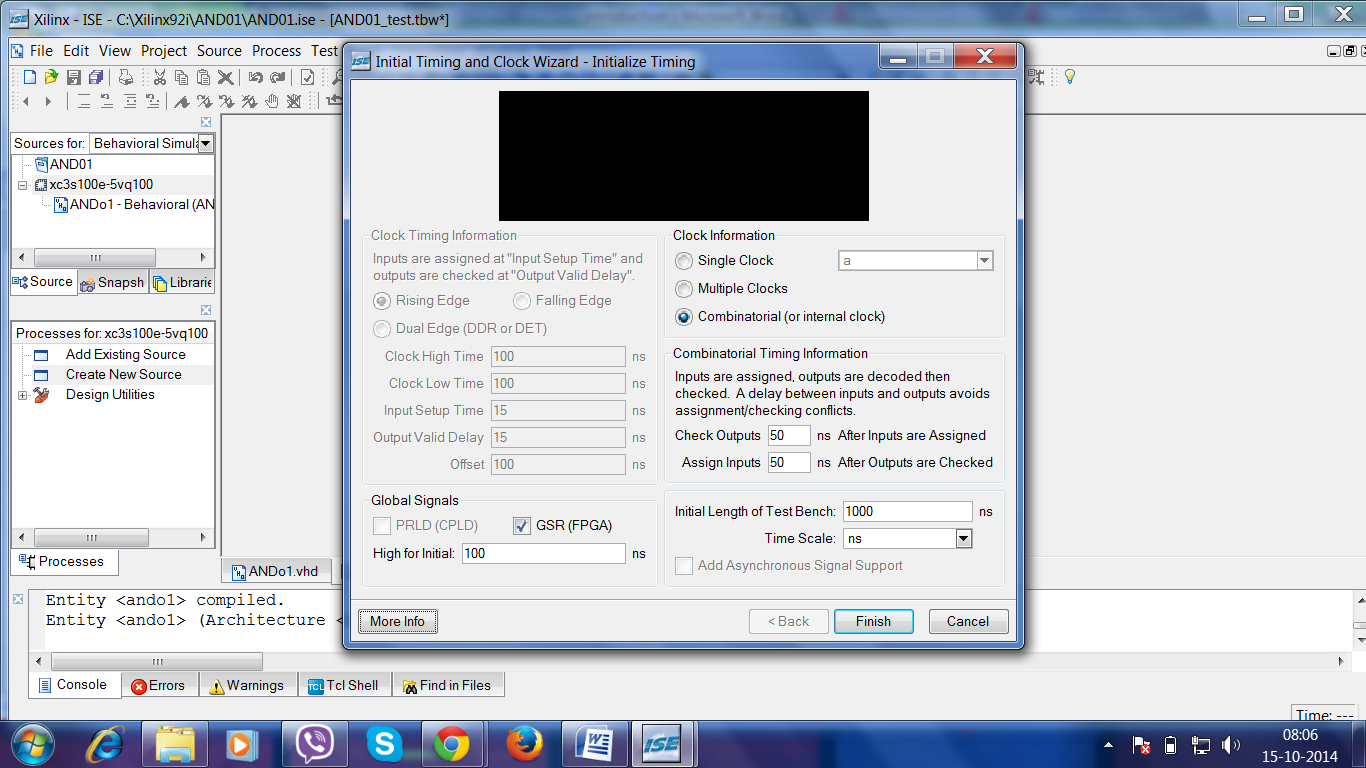
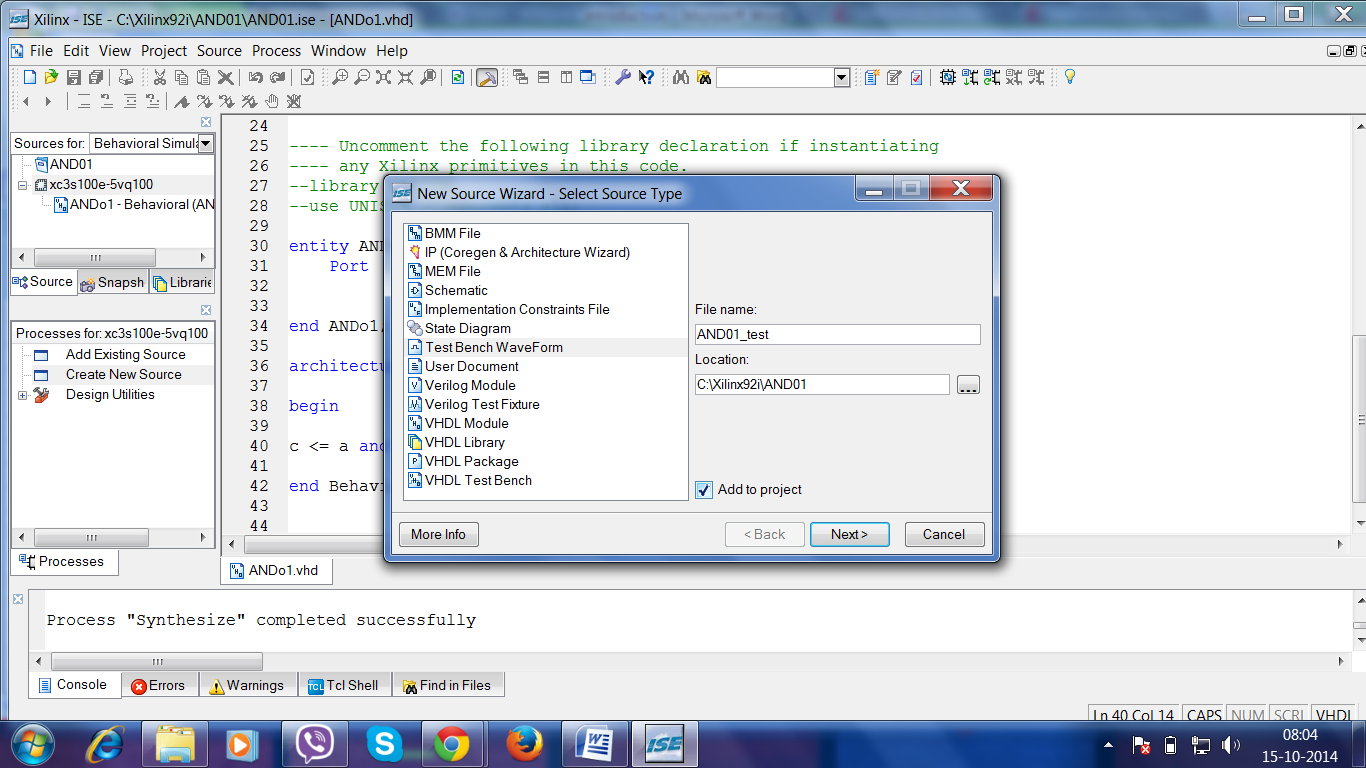
1. Click **+** next to the Synthesize-XST process to expand the hierarchy.
2. Double-click on the **Synthesize -XST** process.



**Step-5-Create a Test Bench for Simulation**

In this section, you will create a test bench waveform containing input stimulus you can use to simulate the counter module. This test bench waveform is a graphical view of a test bench. It is used with simulator to verify that the counter design meets both behavioral and timing design requirements. You will use the waveform editor to create a test bench waveform (TBW) file.

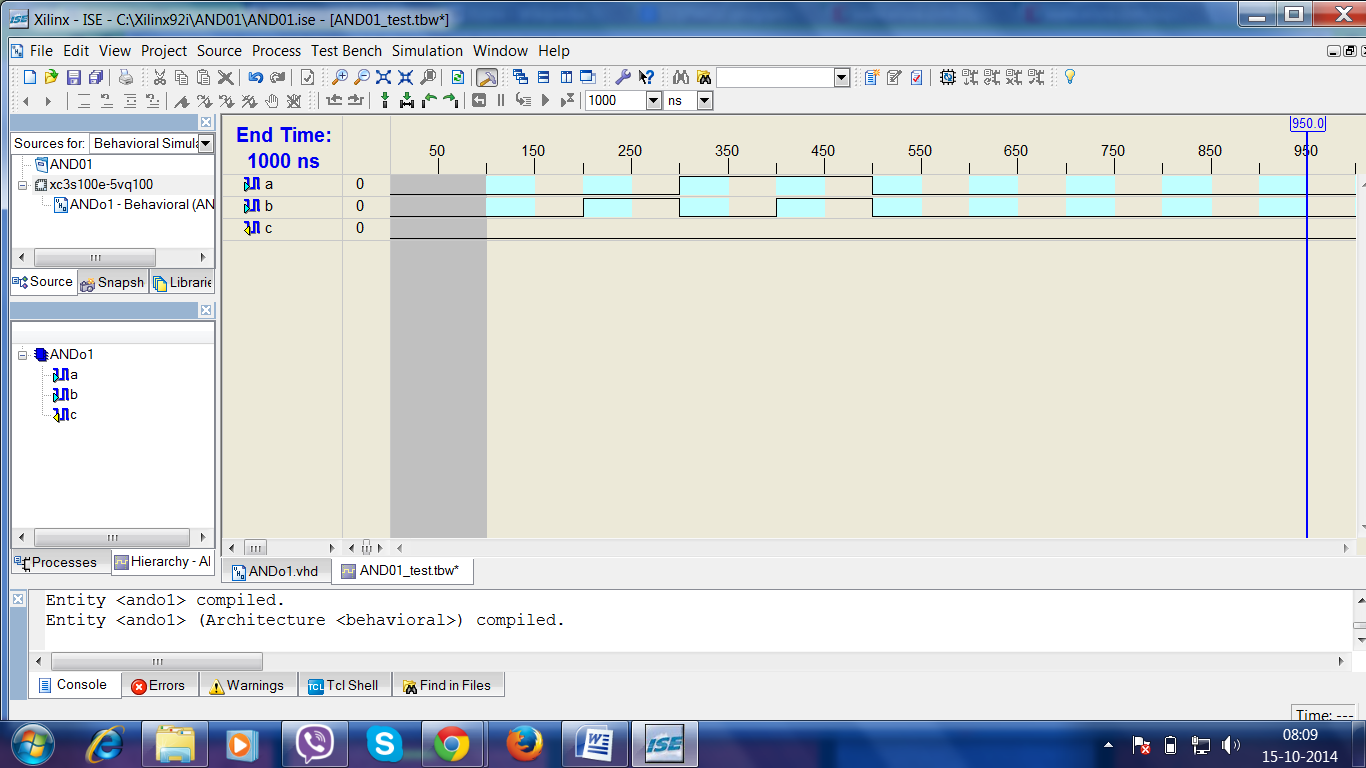
1. Select the **counter** HDL file in the Sources in Project window.
2. Create a new source by selecting **project -> New Source**.
3. In the New Source window, select **Test Bench Waveform** as the source type, and type **AND01\_test** in the File Name field.
4. Make sure the box for Add to Project is checked.
5. Click **Next**.
6. The source File dialog box shows that you are associating the test bench with the source file: counter. Click **Next**.
7. Click **Finish**.   
   You need to set the initial values for test bench waveform in the Initialize Timing dialog box before the test bench waveform editing window opens.

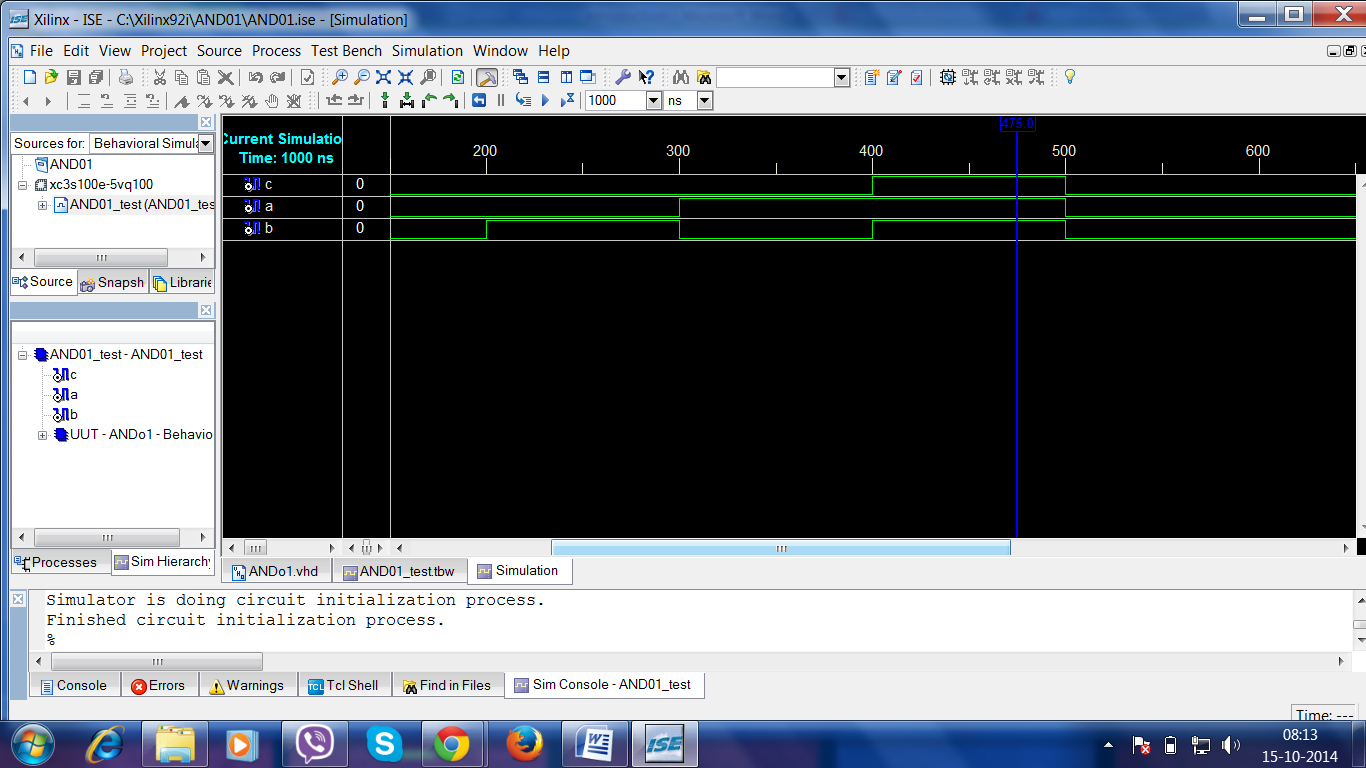


1. Select **File -> Save** to save the waveform.
2. Select the **Behavioral Simulation**in the Source window.
3. On the Sources in Project Window, the TBW file AND01\_test**.tbw** is automatically added to your project.

### Step-6-Simulating Behavioral Model (ISE Simulator)

### To run the integrated simulation process in ISE:

1. Select the **AND01\_test** waveform in the Sources in Project window. You can see Xilinx ISE Simulator processes in the Processes for Source window.
2. Double-click on the **Simulate Behavioral Model**process in the Project window. The ISE Simulator opens and run the simulation to the end of the test bench.



1. **FINITE STATE MACHINE**

**Finite State Machines**

**Introduction**

Finite State Machines (FSM) are sequential circuit used in many digital systems to control the behavior of systems and dataflow paths. Examples of FSM include control units and sequencers. This lab introduces the concept of two types of FSMs, Mealy and Moore, and the modelling styles to develop such machines.

**Objectives**

After completing this lab, you will be able to:

* Model Mealy FSMs
* Model Moore FSMs

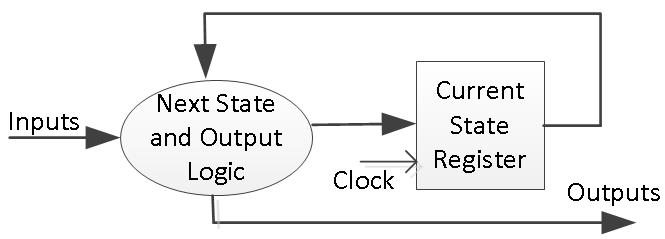
**Mealy FSM** **Part 1**

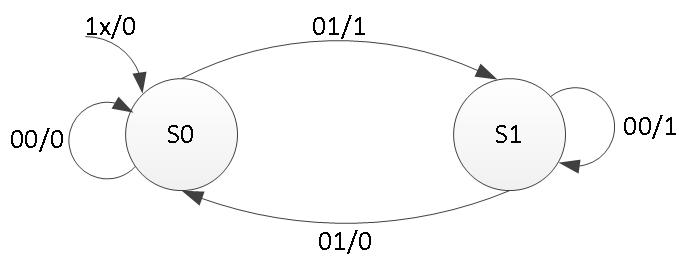
A finite-state machine (FSM) or simply a state machine is used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of user-defined states. The machine is in only one state at a time; the state it is in at any given time is called the *current state*. It can change from one state to another when initiated by a triggering event or condition; this is called a *transition*. A particular FSM is defined by a list of its states, and the triggering condition for each transition.

The behavior of state machines can be observed in many devices in modern society performing a predetermined sequence of actions depending on a sequence of events with which they are presented. Simple examples are vending machines which dispense products when the proper combination of coins are deposited, elevators which drop riders off at upper floors before going down, traffic lights which change sequence when cars are waiting, and combination locks which require the input of combination numbers in the proper order.

The state machines are modelled using two basic types of sequential networks- Mealy and Moore. In a Mealy machine, the output depends on both the present (current) state and the present (current) inputs. In Moore machine, the output depends only on the present state.

A general model of a Mealy sequential machine consists of a combinatorial network, which generates the outputs and the next state, and a state register which holds the present state as shown below. The state register is normally modelled as D flip-flops. The state register must be sensitive to a clock edge. The other block(s) can be modelled either using the always procedural block or a mixture of the always procedural block and dataflow modelling statements; the always procedural block will have to be sensitive to all inputs being read into the block and must have all output defined for every branch in order to model it as a combinatorial block. The two blocks Mealy machine can be viewed as

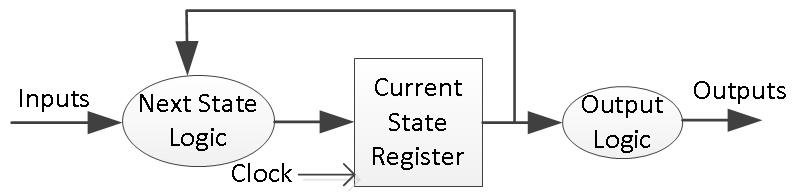


Here are the state diagram of a z checker Mealy machine and the associated model.

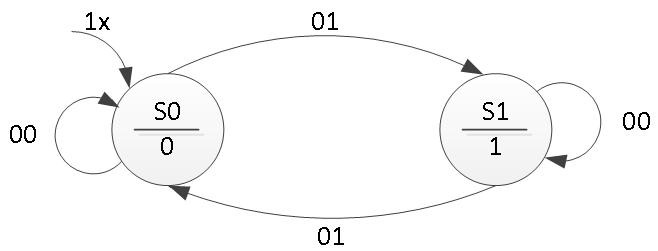
The state assignments can be of one-hot, binary, gray-code, and other types. Usually, the synthesis tool will determine the type of the state assignment, but user can also force a particular type by changing the synthesis property as shown below. The state assignment type will have an impact on the number of bits used in the state register; one-hot encoding using maximum number of bits but decodes very fast to compact (binary) encoding using smallest number of bits but taking longer to decode.

**Moore FSM**

A general model of a Moore sequential machine is shown below. Its output is generated from the state register block. The next state is determined using the present (current) input and the present (current) state. Here the state register is also modelled using D flip-flops. Normally Moore machines are described using three blocks, one of which must be a sequential and the other two can be modelled using always blocks or a combination of always and dataflow modelling constructs.



Here is the state graph of the same z checker to be modelled as a Moore machine. The associate model is shown below.



**DEMUX**

A demultiplexer (or demux) is a device that takes a single input line and routes it to one of several digital output lines. A demultiplexer of 2n outputs has n select lines, which are used to select which output line to send the input. A demultiplexer is also called a data distributor. Demultiplexers can be used to implement general purpose logic. By setting the input to true, the demux behaves as a decoder.

The reverse of the digital demultiplexer is the digital multiplexer

**1 to 4 - Demultiplexer**

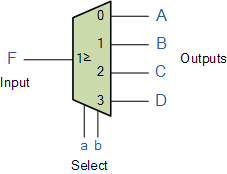
A 1 to 4 multiplexer uses 2 select lines (S0, S1) to determine which one of the 4 outputs (Y0 - Y3) is routed from the input (D). Its characteristics can be described in the following simplified truth table.

Truth Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S1 | S0 | Y3 | Y2 | Y1 | Y0 |
| 0 | 0 | 0 | 0 | 0 | D |
| 0 | 1 | 0 | 0 | D | 0 |
| 1 | 0 | 0 | D | 0 | 0 |
| 1 | 1 | D | 0 | 0 | 0 |

DEMUX is a combinational circuit that routes one input to various channels depending on the states. The states are defined by user in terms of binary input that is regulated by the type of DEMUX, for example the 2 X 4 DEMUX depends on 2 states and routes the output in 4 different outputs. In the above diagram, the states have values in the form of INPUT/OUTPUT, for example – input of S0 state is 00 and output is 0 and it goes back to its own state as shown in the diagram.

Analyzing the mealy circuit, we can say that it depends on both the output and next state. The output will change multiple times even for the same state depending upon the input. In a DEMUX the output is fixed and it cannot be changed with respect to the states, changing states can regulate output on different ports or channel but the value cannot be changed. So, a mealy circuit cannot be used to implement a DEMUX.



**APPENDIX A**

**A.1 Code for MEALY STATE MACHINE**

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

-- Create Date: 00:38:27 04/01/2017

-- Design Name:

-- Module Name: fsm\_mealy\_mod - Behavioral

-- Project Name:

-- Target Devices:

-- Tool versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

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 primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity fsm\_mealy\_mod is

Port ( reset : in STD\_LOGIC;

clk : in STD\_LOGIC;

x : in STD\_LOGIC;

z : out STD\_LOGIC);

end fsm\_mealy\_mod;

architecture Behavioral of fsm\_mealy\_mod is

type state\_type is (S0, S1);

signal state, next\_state : state\_type;

begin

SYNC\_PROC : process (clk)

begin

if rising\_edge(clk) then

if (reset = '1') then

state <= S0;

else

state <= next\_state;

end if;

end if;

end process;

NEXT\_STATE\_DECODE : process (state, x)

begin

z <= '0';

case (state) is

when S0 =>

if (x = '1') then

z <= '1';

next\_state <= S1;

else

next\_state <= S0;

end if;

when S1 =>

if (x = '1') then

next\_state <= S0;

else

z <= '1';

next\_state <= S1;

end if;

when others =>

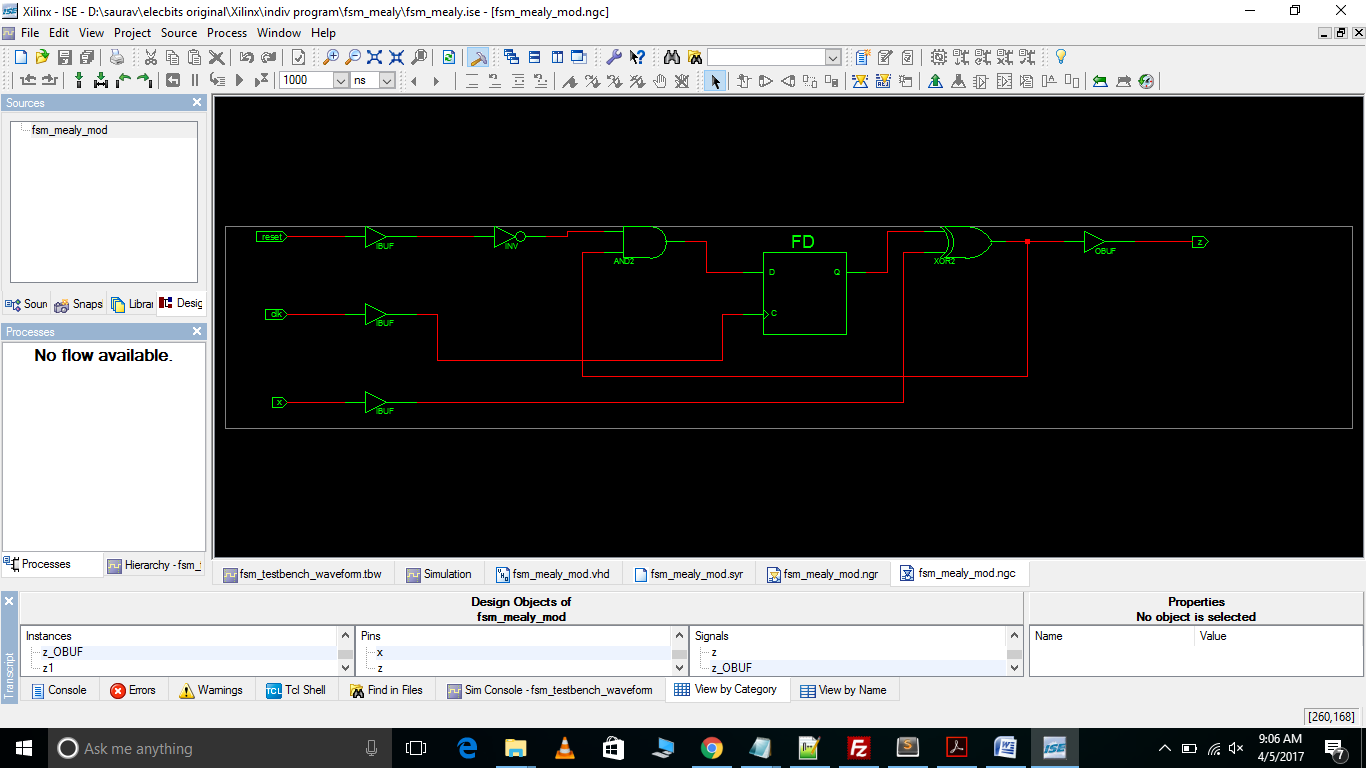
next\_state <= S0;

end case;

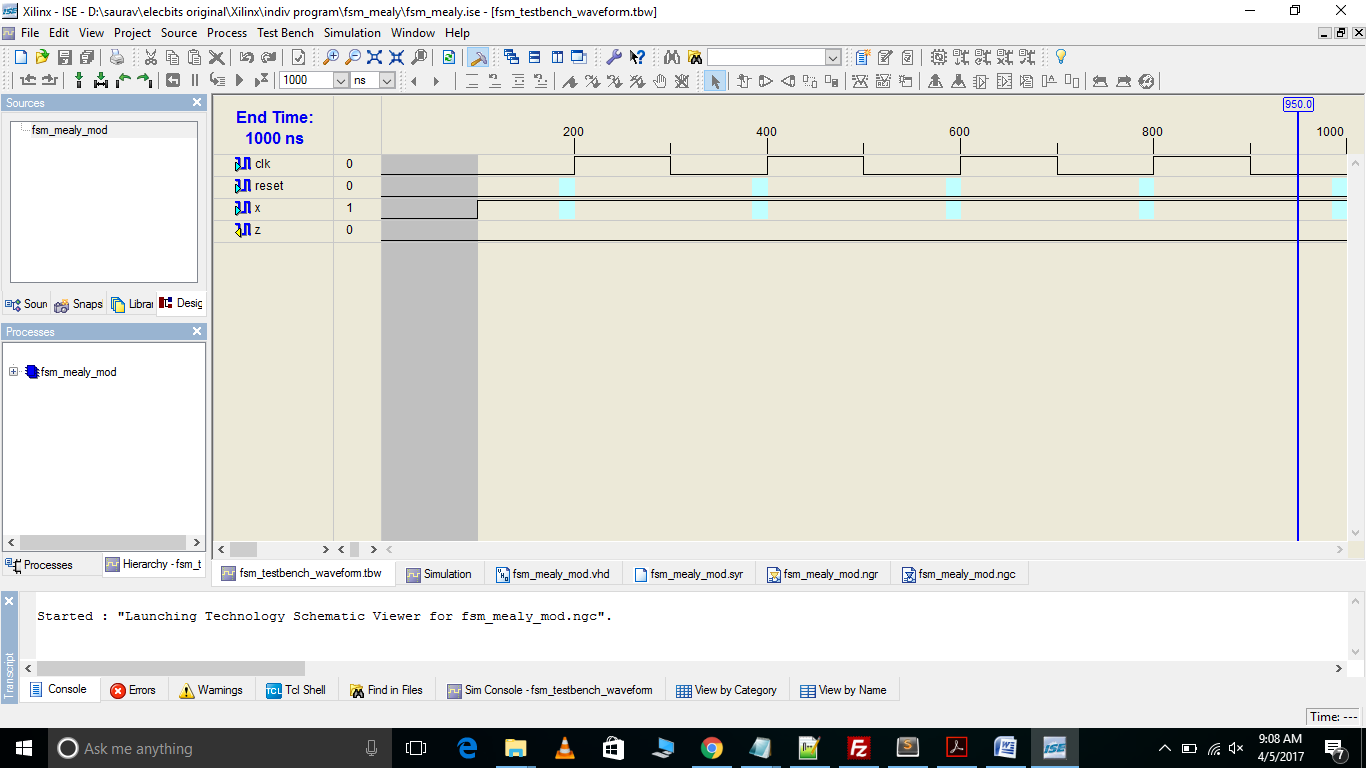
end process;

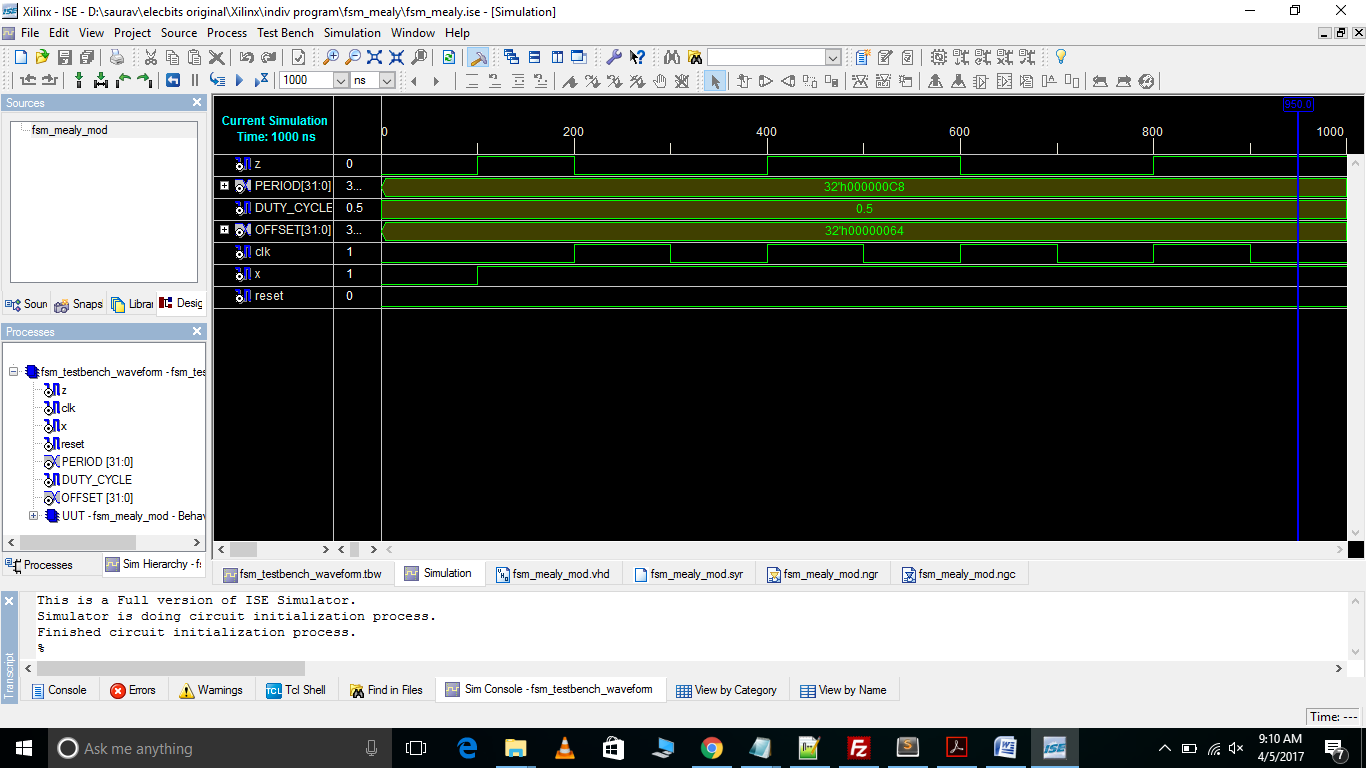
end Behavioral;

View RTL Schematic

View Technology Schematic  
  


Test Bench Waveform



Stimulation  
  


**A.2 Code for MOORE STATE MACHINE**

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 03:19:02 04/01/2017

-- Design Name:

-- Module Name: fsm\_moore\_mod - Behavioral

-- Project Name:

-- Target Devices:

-- Tool versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

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 primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity fsm\_moore\_mod is

Port ( reset : in STD\_LOGIC;

clk : in STD\_LOGIC;

x : in STD\_LOGIC;

z : out STD\_LOGIC);

end fsm\_moore\_mod;

architecture Behavioral of fsm\_moore\_mod is

type state\_type is (S0, S1);

signal state, next\_state : state\_type;

begin

SYNC\_PROC : process (clk)

begin

if rising\_edge(clk) then

if (reset = '1') then

state <= S0;

else

state <= next\_state;

end if;

end if;

end process ;

OUTPUT\_DECODE : process (state)

begin

case (state) is

when S0 =>

z <= '0';

when S1 =>

z <= '1';

when others =>

z <= '0';

end case;

end process;

NEXT\_STATE\_DECODE : process (state, x)

begin

next\_state <= S0;

case (state) is

when S0 =>

if (x = '1') then

next\_state <= S1;

end if;

when S1 =>

if (x = '0') then

next\_state <= S1;

end if;

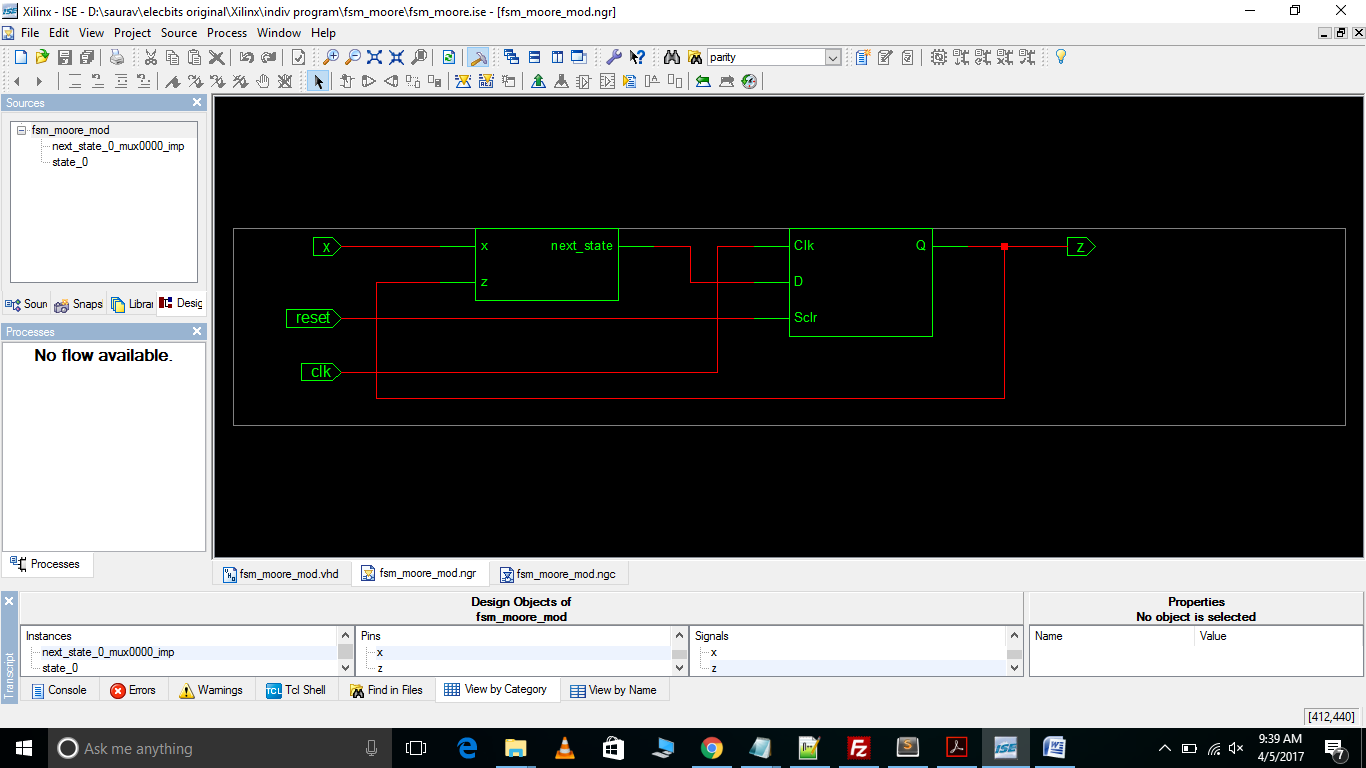
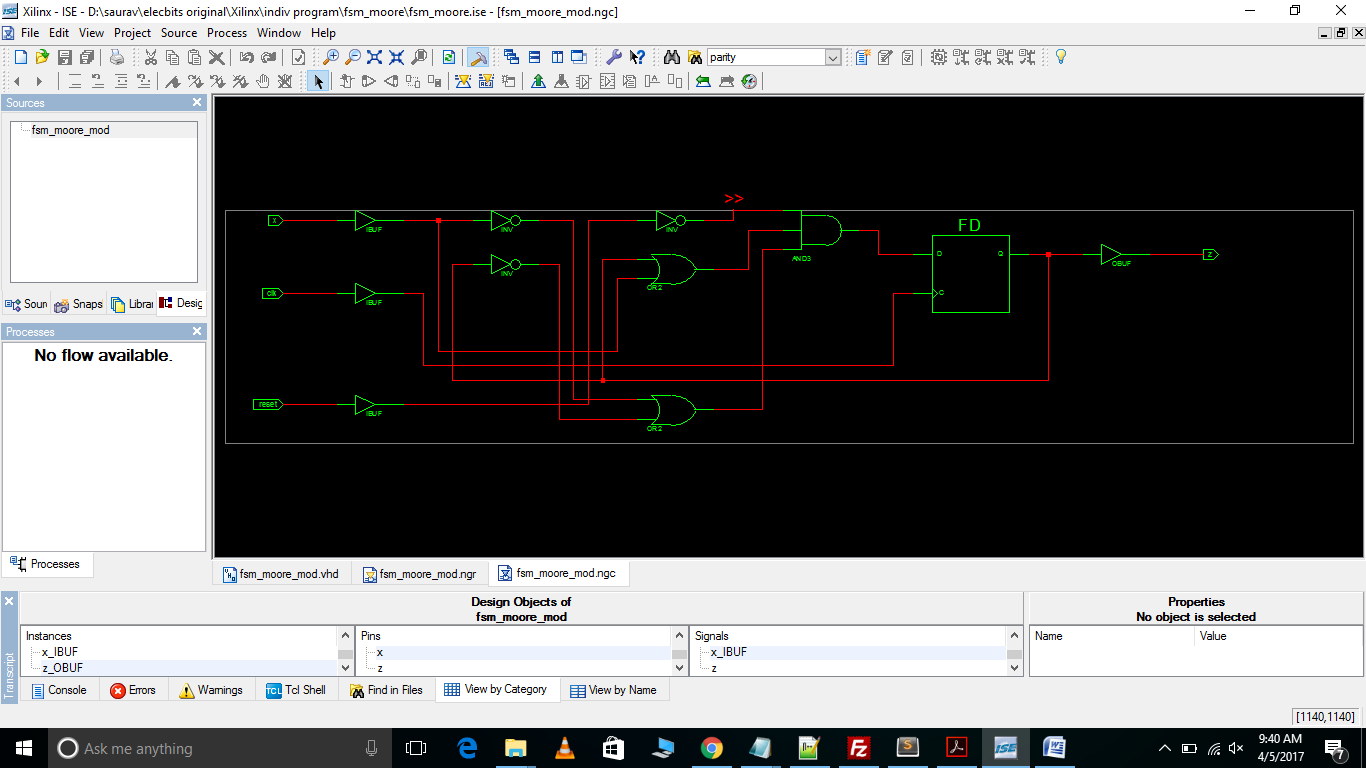
when others =>

next\_state <= S0;

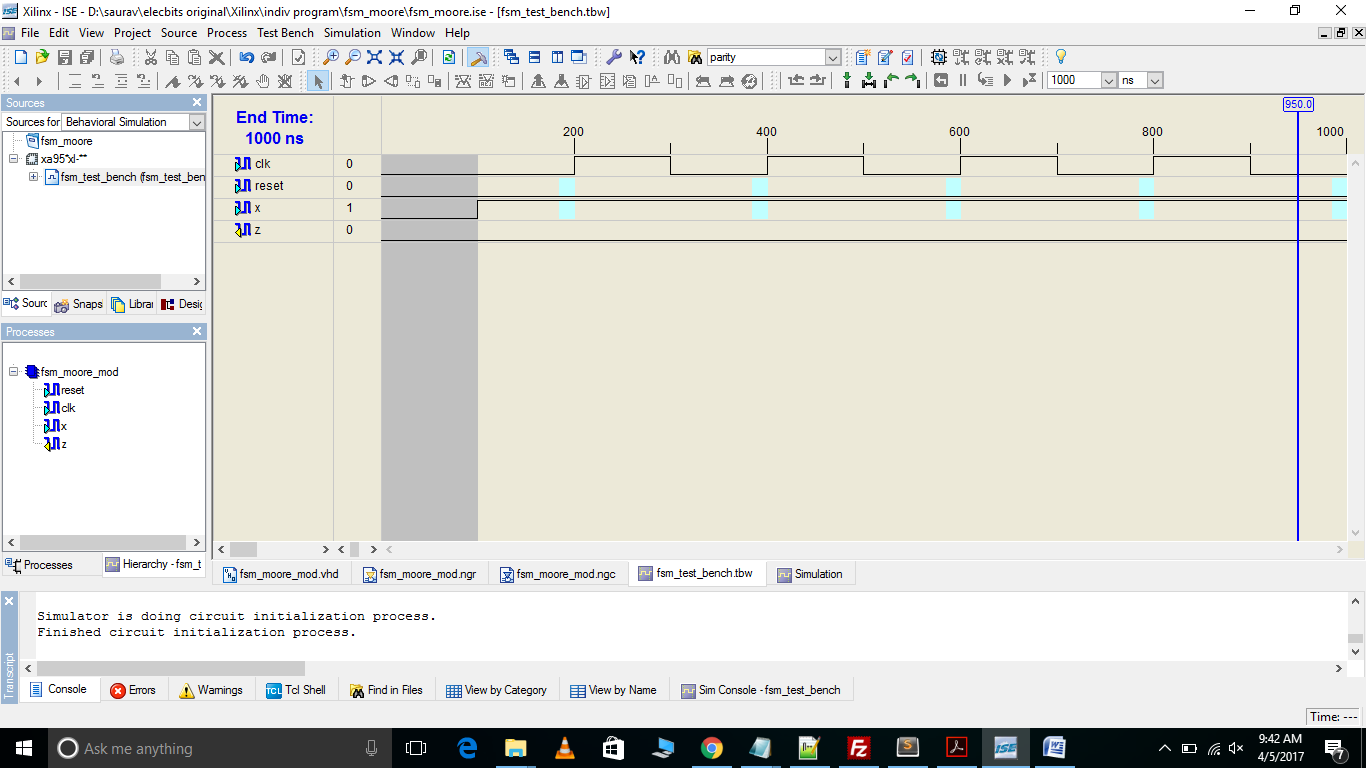
end case;

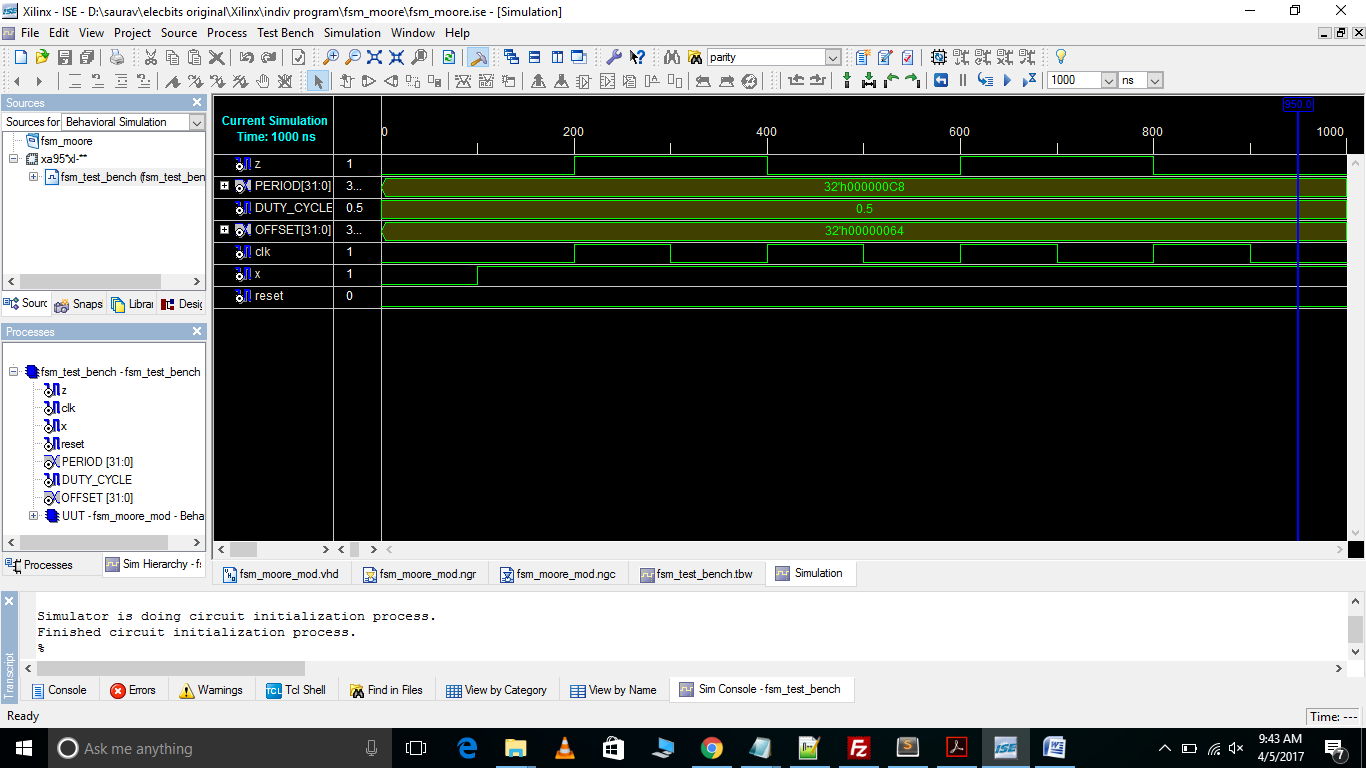
end process; end behavioral;

View RTL Schematic

  
View Technology Schematic  
  


Test Bench Waveform



Stimulation  
  


**A.3 Code for DEMUX USING STATE MACHINE**

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 17:43:02 04/11/2017

-- Design Name:

-- Module Name: fsm\_demux\_impl - Behavioral

-- Project Name:

-- Target Devices:

-- Tool versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

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 primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity fsm\_demux\_impl is

Port ( reset : in STD\_LOGIC;

x : in STD\_LOGIC\_VECTOR(1 downto 0);

clk : in STD\_LOGIC;

z : out STD\_LOGIC);

end fsm\_demux\_impl;

architecture Behavioral of fsm\_demux\_impl is

type state\_type is (S0, S1, S2, S3);

signal state, next\_state : state\_type;

begin

SYNC\_PROC : process (clk)

begin

if rising\_edge(clk) then

if (reset = '1') then

state <= S0;

else

state <= next\_state;

end if;

end if;

end process;

NEXT\_STATE\_DECODE : process (state, x)

begin

z <= '0';

case (state) is

when S0 =>

if (x = "00") then

z <= '1';

next\_state <= S0;

elsif (x = "01") then

z <= '0';

next\_state <= S1;

elsif (x = "10") then

z <= '0';

next\_state <= S2;

else

z <= '0';

next\_state <= S3;

end if;

when S1 =>

if (x = "00") then

z <= '0';

next\_state <= S0;

elsif (x = "01") then

z <= '1';

next\_state <= S1;

elsif (x = "10") then

z <= '0';

next\_state <= S2;

else

z <= '0';

next\_state <= S3;

end if;

when S2 =>

if (x = "00") then

z <= '0';

next\_state <= S0;

elsif (x = "01") then

z <= '0';

next\_state <= S1;

elsif (x = "10") then

z <= '1';

next\_state <= S2;

else

z <= '0';

next\_state <= S3;

end if;

when S3 =>

if (x = "00") then

z <= '0';

next\_state <= S0;

elsif (x = "01") then

z <= '0';

next\_state <= S1;

elsif (x = "10") then

z <= '0';

next\_state <= S2;

else

z <= '1';

next\_state <= S3;

end if;

when others =>

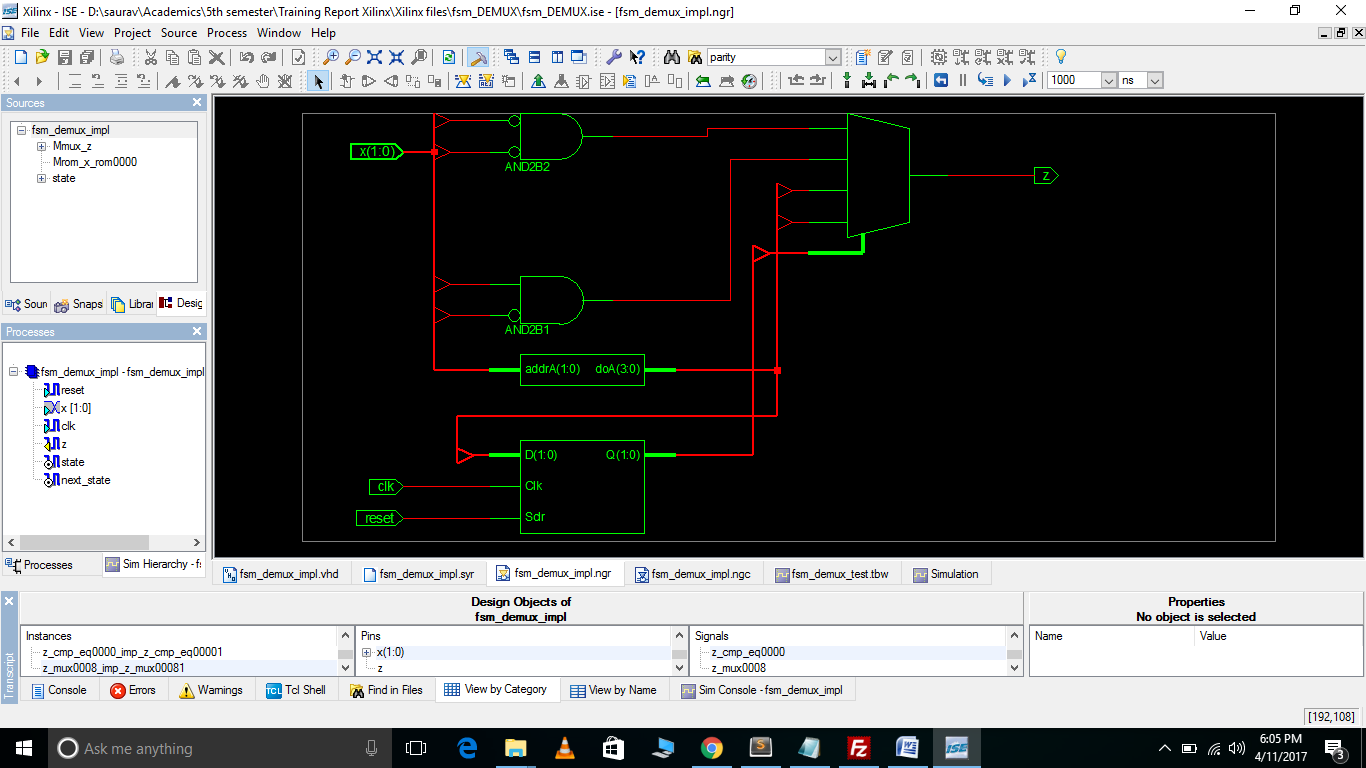
next\_state <= S0;

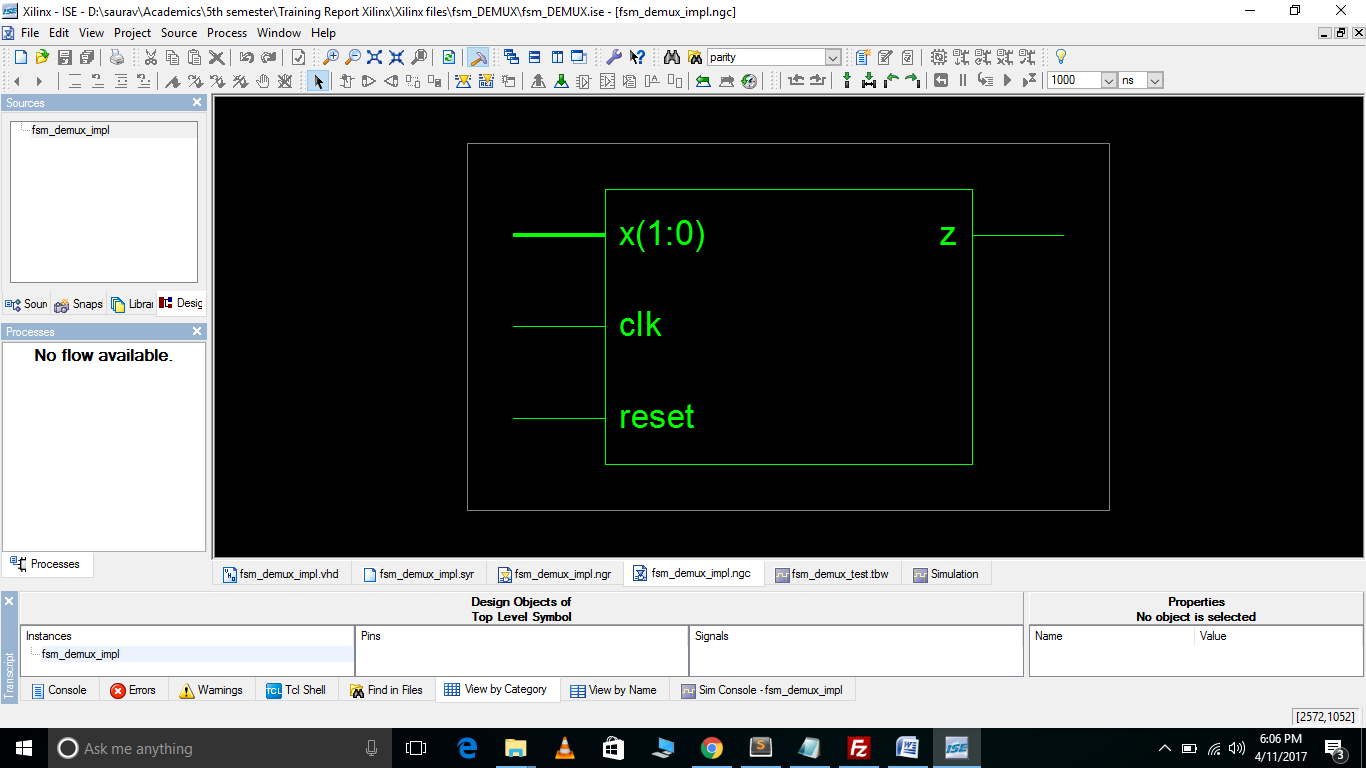
end case;

end process;

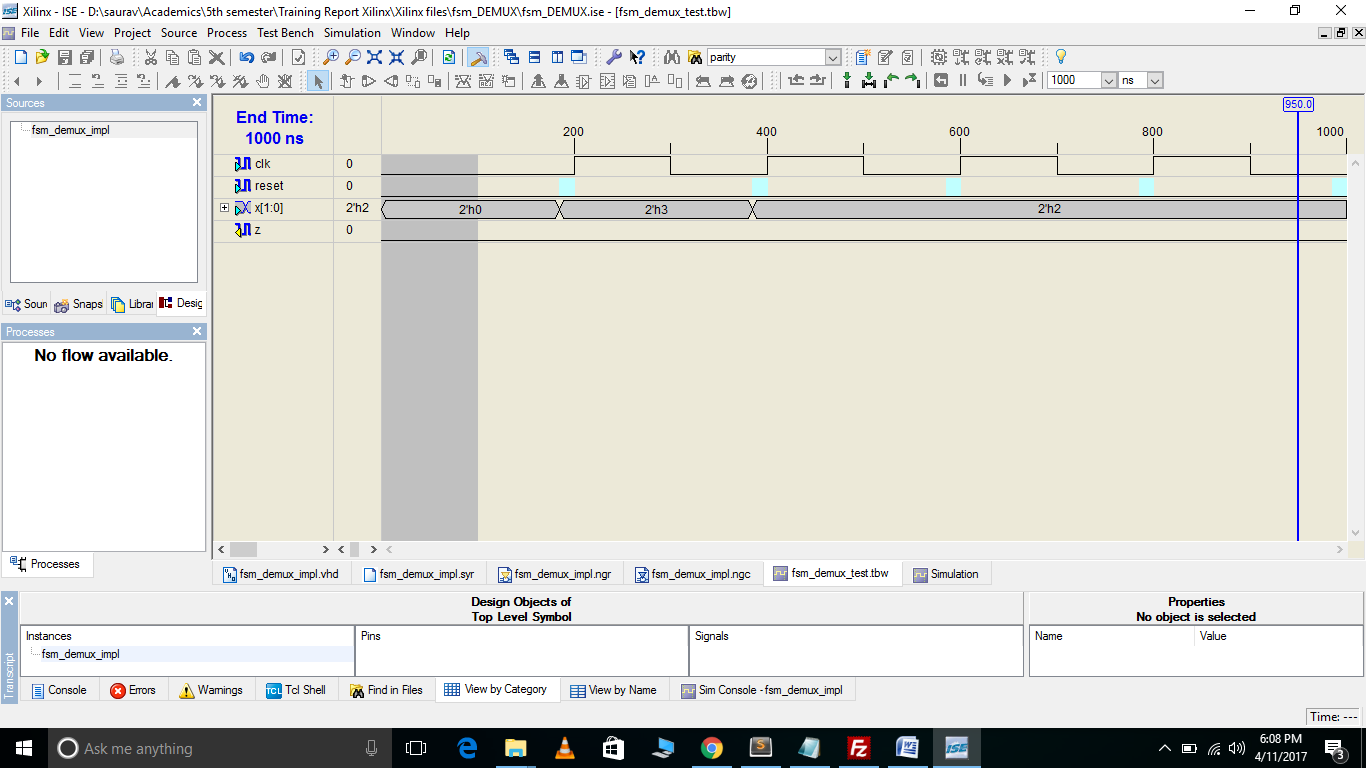
end Behavioral;

View RTL Schematic



View Technology Schematic  
  


Test Bench Waveform



Stimulation  
  


**A.4 Code for AND DATAFLOW (Sample)**

----------------------------------------------------------------------------------

-- Company:

-- Engineer: Suraj

--

-- Create Date: 10:37:19 03/20/2017

-- Design Name:

-- Module Name: AND\_Dataflow - Behavioral

-- Project Name:

-- Target Devices:

-- Tool versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

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 primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity AND\_Dataflow is

Port ( a : in STD\_LOGIC;

b : in STD\_LOGIC;

c : out STD\_LOGIC);

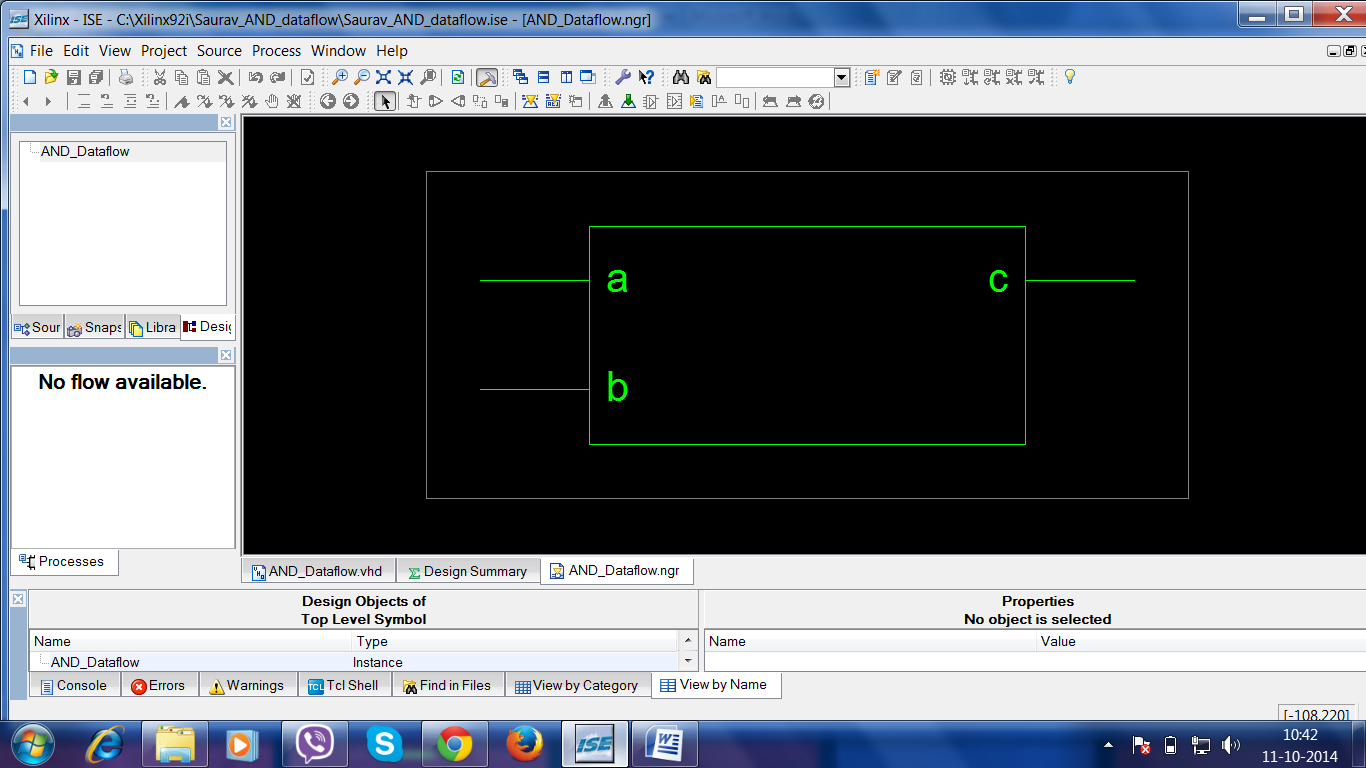
end AND\_Dataflow;

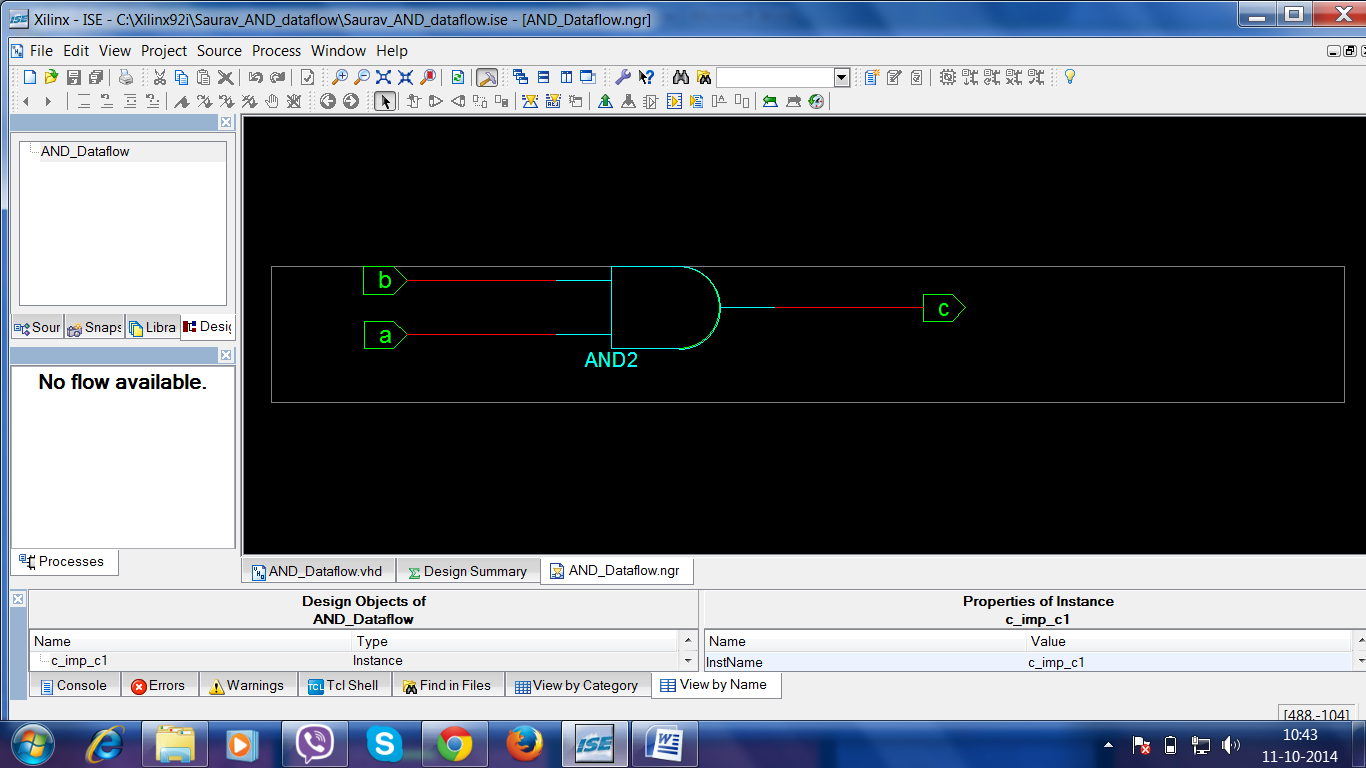
architecture dataflow of AND\_Dataflow is

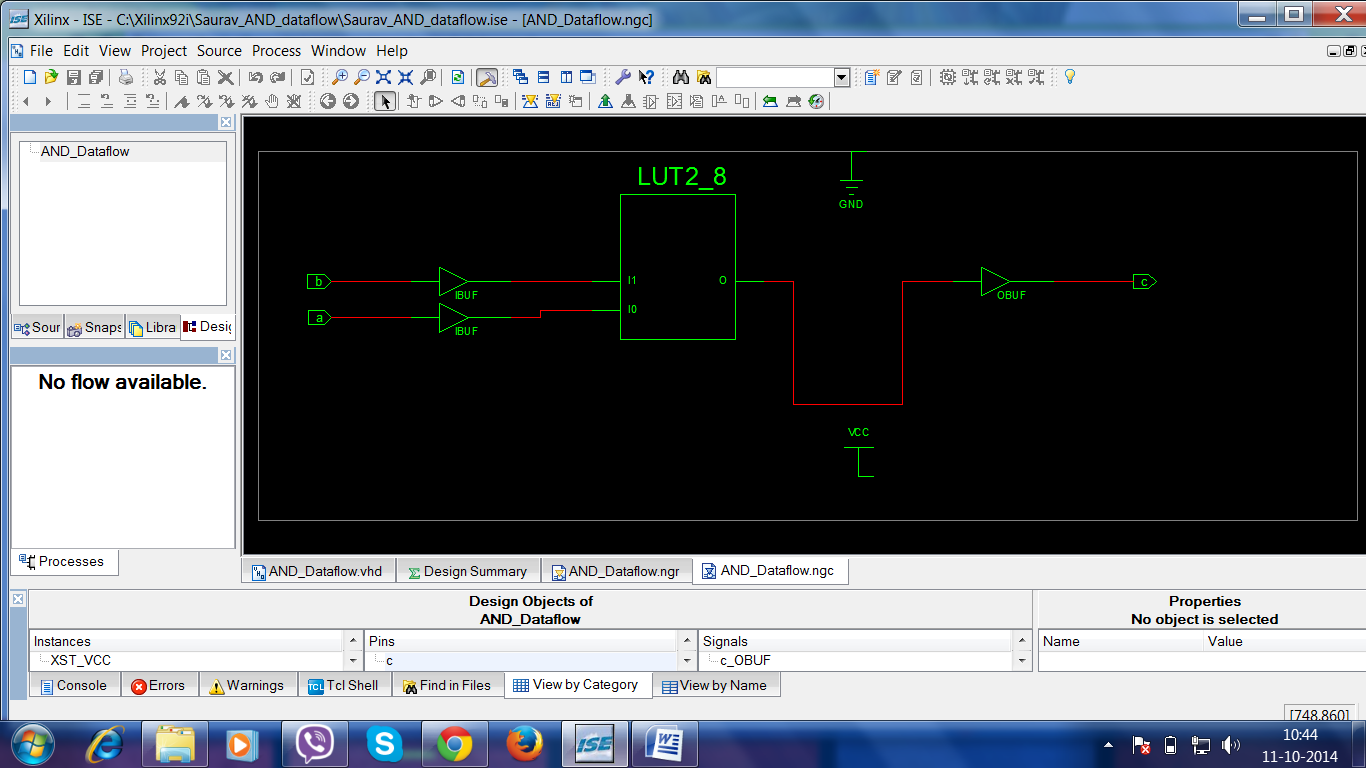
begin

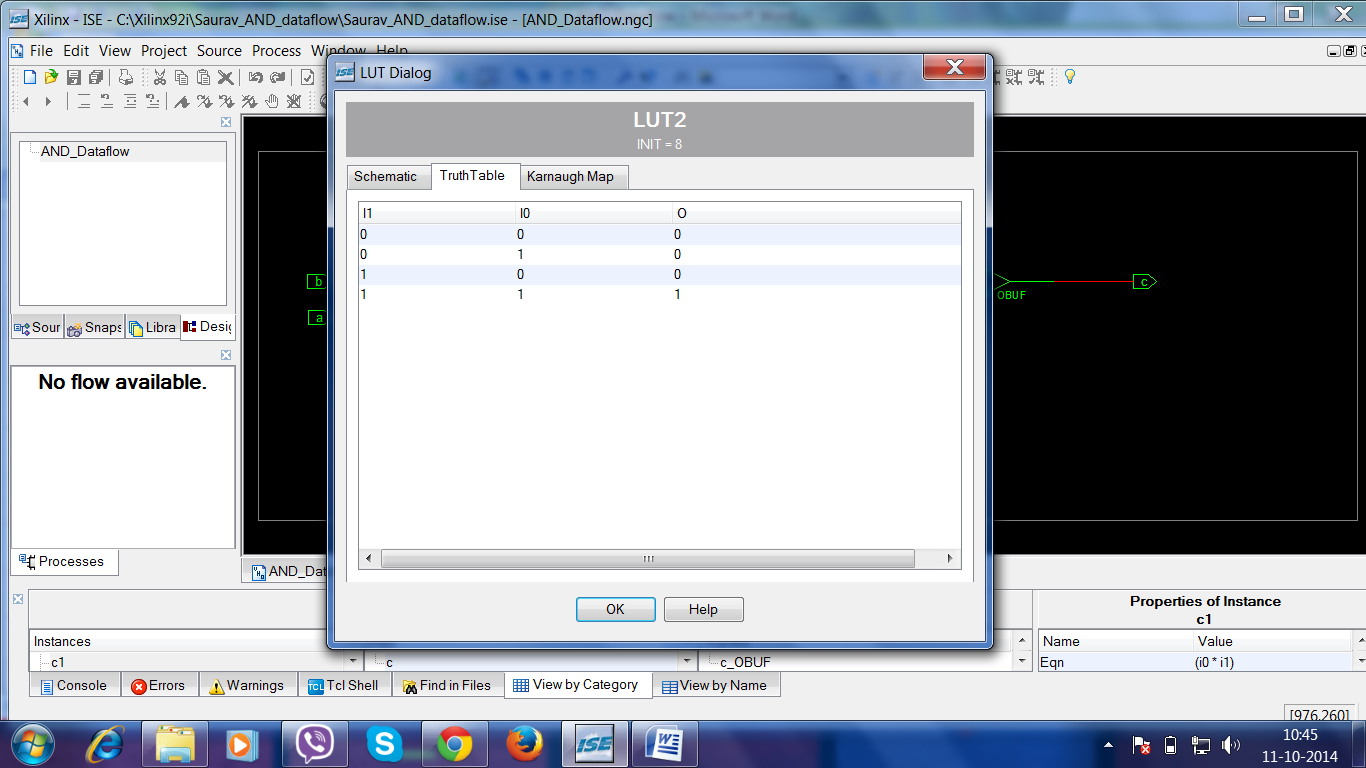
c <= a and b;

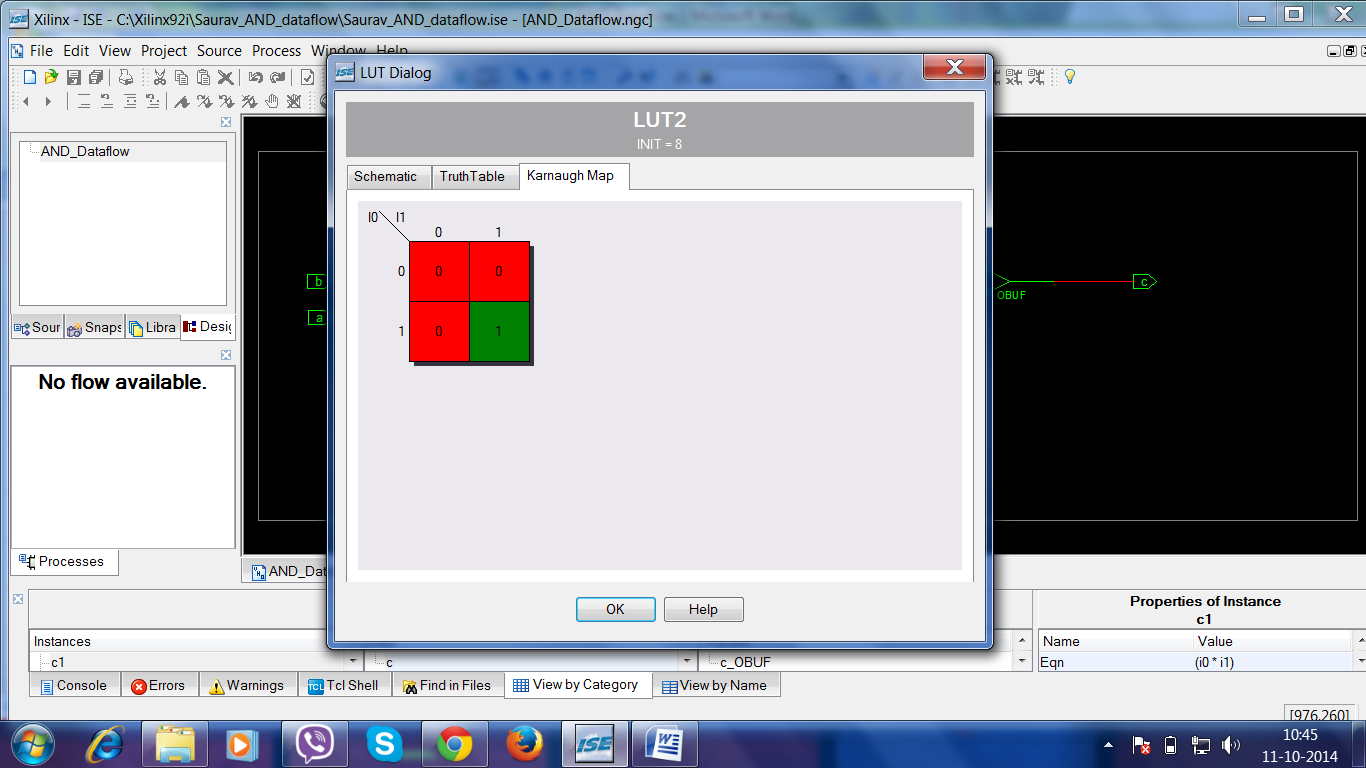
end dataflow;

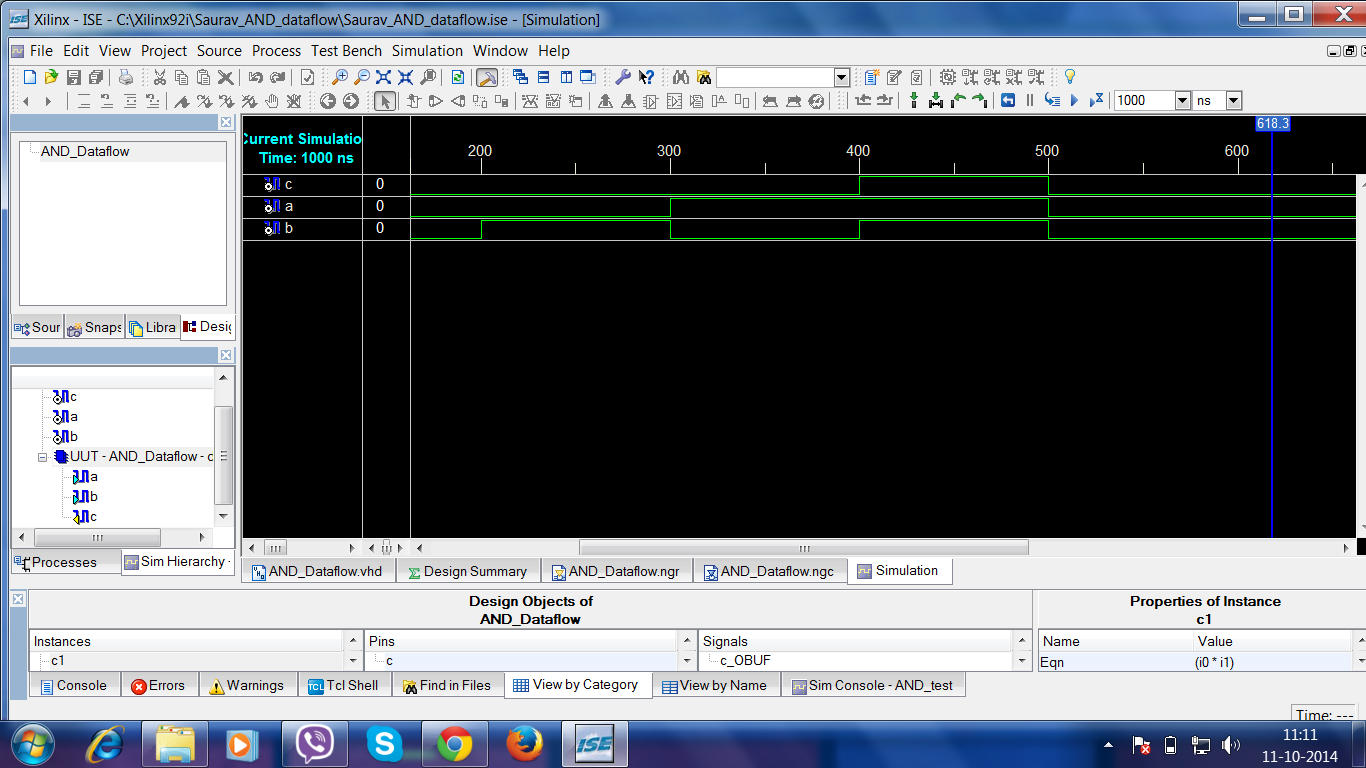












**REFERENCES**

**Books:**

* **“Verilog HDL" by Samir Palntikar**

**Web URLs:**

* <http://ieeexplore.ieee.org/document/6726639/?reload=true&arnumber=6726639>
* http://ieeexplore.ieee.org/document/404581/?tp=&arnumber=404581&queryText%3Dxilinx%20State%20CAD%20FSM