Digital Circuit Design with Xilinx FPGA

Lab Manual

Workshop No.1

Lab1:

Getting Started with Xilinx ISE Design Suite

Summer 2015

Outline

This manual covers the following topics:

- Introduction about ISE Project Navigator
- Design creation using ISE Project Navigator
- Synthesis the design using XST
- View RTL and Technology Schematics

Introduction

ISE Design Suite consists of different tools to cover all aspects of the digital design flow. In the Project Navigator, the designer can access all of these tools and also the files and documents related to the project. Figure. 1 shows the details of the Project Navigator interface, which includes three main parts:

1. **Left panels**
   These panels include Start, Design, Files, and Libraries panels (additional panels can be added by selecting from the View menu).
   - **Start Panel**
     Provides quick access to open projects as well as frequently access reference material, documentation and tutorials.
   - **Design Panel**
     Provides access to the View, Hierarchy, and Processes Panes, which are described below:
     - **View Pane**
       The View Pane radio buttons enable you to view the source modules associated with the Implementation or Simulation Design View in the Hierarchy Pane. If you select Simulation, you must select a simulation phase from the drop-down list.
     - **Hierarchy Pane**
       Displays the project name, the target device, user documents, and design source files associated with the selected Design View. The View Pane at the top of the Design Panel allows you to view only those source files associated with the selected Design View, such as Implementation or Simulation. Each

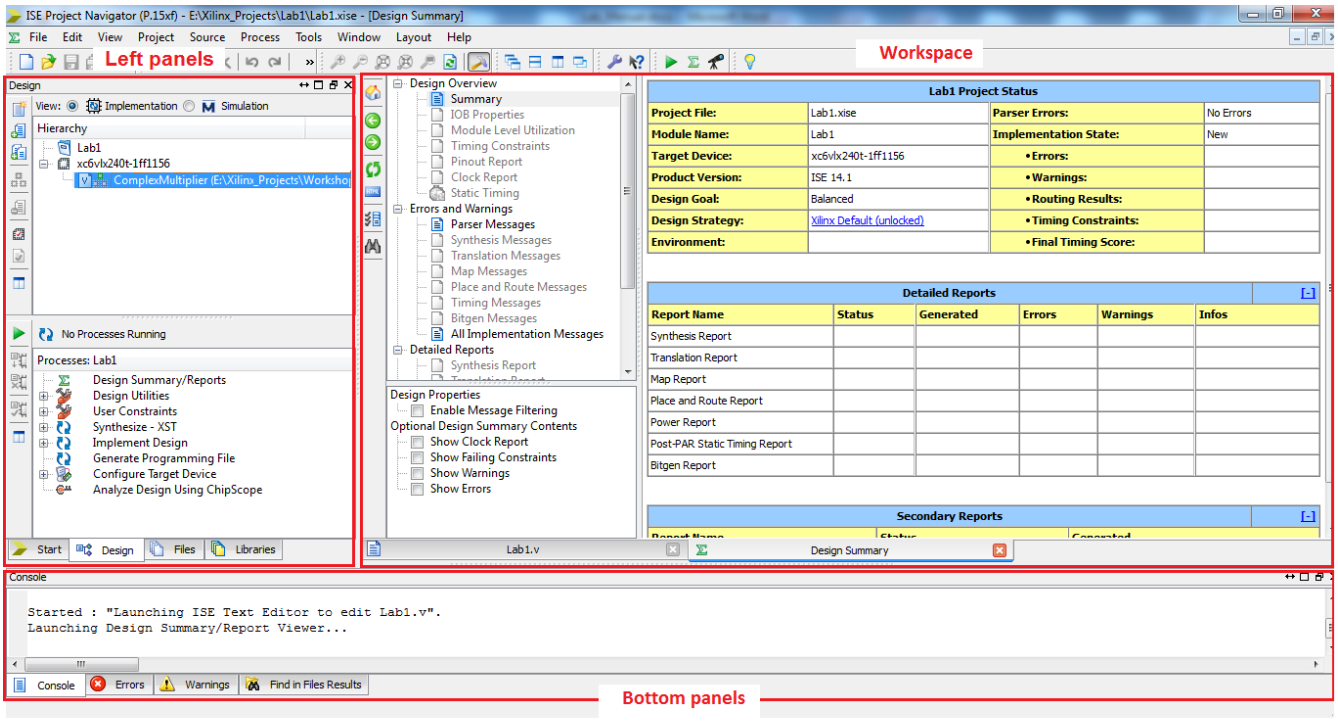

Figure. 1. ISE Project Navigator interface.

file in the Hierarchy Pane has an associated icon, which indicates the file type. If a file contains lower levels of hierarchy, the icon has a plus symbol to the left of the name.

- **Processes Pane**

  The Processes Pane changes based on the selected source type in the Sources Pane and the top-level source in your project. This pane provides access to some functions such as **Design Summary/Reports, Design Utilities, User Constraints, Synthesis, Implement Design, Generate Programming File, Configure Target Device.**

➢ **Files Panel**

Provides a sortable list (based on any of the columns in the view) of all the source files in the project. Properties for each file can be viewed and modified by right-clicking on the file and selecting **Source Properties**.

➢ **Libraries Panel**

Enables you to manage HDL libraries and their associated HDL source files. You can create, view, and edit libraries and their associated sources.

2. **Bottom panels**
   - **Console Panel**
     Displays all standard output from processes run from Project Navigator, such as errors, warnings, and information messages.
   - **Errors Panel**
     Displays only error messages and the other console messages are filtered out.
   - **Warnings Panel**
     Displays only warning messages and the other console messages are filtered out.

3. **Workspace**
   The Workspace is where design editors, viewers, and analysis tools open. These include ISE Text Editor, Schematic Editor, Constraint Editor, Design Summary/Report Viewer, RTL and Technology Viewers, and Timing Analyzer. Other tools such as the PlanAhead, ISim, third-party text editors, XPower Analyzer, and iMPACT open in separate windows outside the main Project Navigator environment when invoked.

Required Software

To perform this manual, you must have **Xilinx ISE Design Suite 14.1** (or above) installed. Note that in this manual the pictures are captured in **Xilinx ISE Design Suite 14.1**, but all the procedures can be done in other versions of this software.

Objectives

After completing this lab, you will be able to:

- *Create a complete design in ISE Project Navigator.*
- *Synthesize a design using Xilinx XST with different synthesis options and settings.*
- *Work with RTL/Technology schematic viewer.*

General Flow of this Lab

This lab comprises of three steps as follows:

Figure 2. General flow of Lab1.

Design Description

The target design of Lab1 is a complex multiplier, which is shown in Figure 3. Note that there are many different architectures for a complex multiplier. But the goal of this lab is not to propose an efficient complex multiplier. This core has two complex-domain input operands (i.e. InputMultiplier1 and InputMultiplier2), clock (i.e. Clk), and reset (i.e. Reset), which are shown in this figure. The output of this block is the complex multiplication of two input operands (i.e. MultiplicationResult).

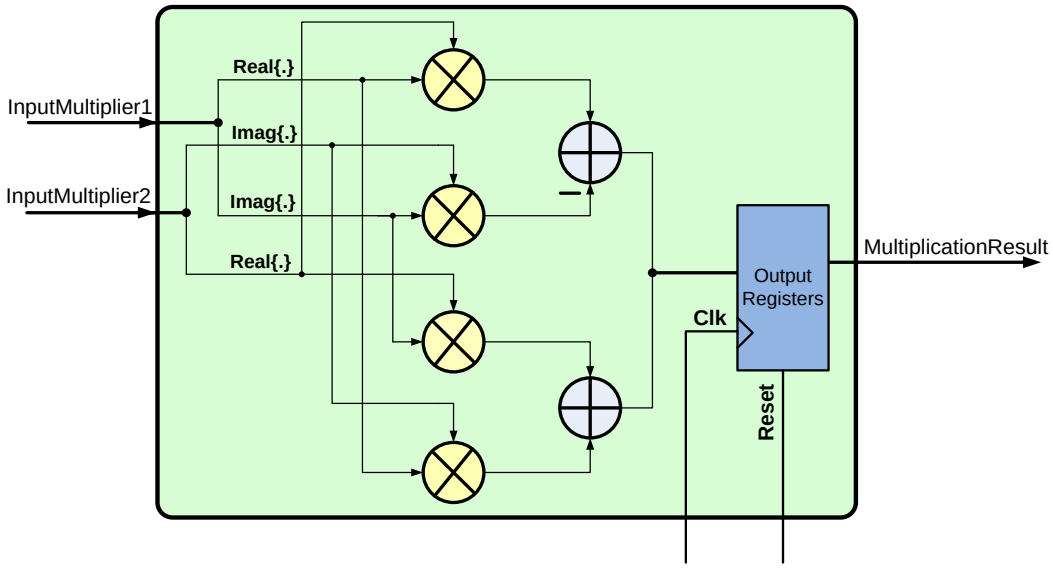

Figure 3. The detailed block diagram of the target design of Lab1.

Note that the format of the inputs and outputs of this block is as follow:

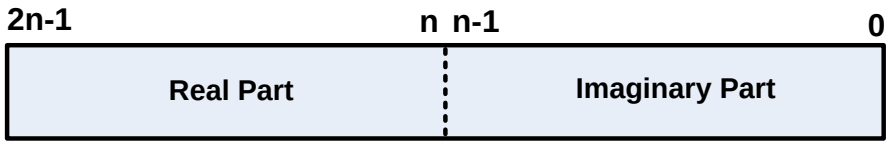

Figure 4. A sample input/output of the complex multiplier block with the length of **2n** bits.

Step1: Design Creation

1. Start the Project Navigator

Double click the Project Navigator icon on your desktop, or select **Start > All Programs > Xilinx Design Tools > ISE Design Suite 14.1 > ISE Design Tools > Project Navigator.**

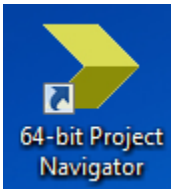

Figure 5. Project Navigator desktop icon.

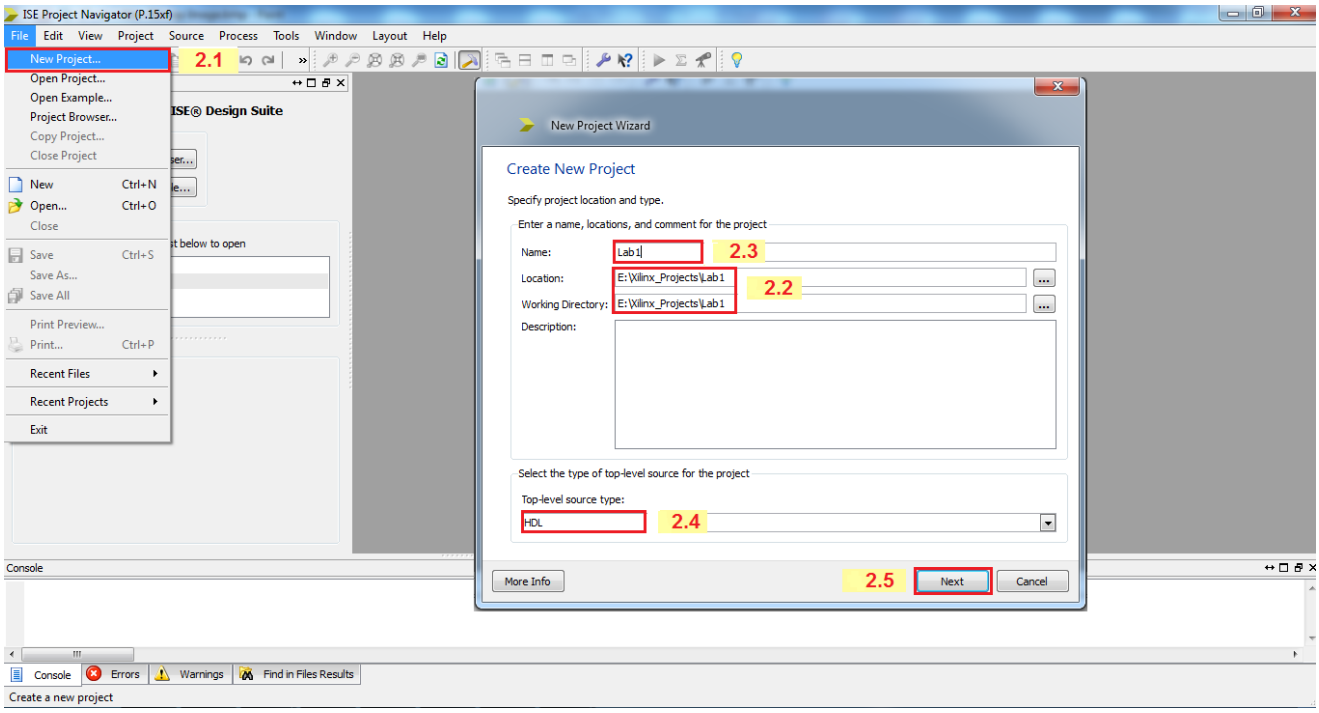

Figure 6. New Project Wizard.

2. Creating a New Project (Figure 6-7)

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

2.2. In the Location field, browse to the directory in which you want to save the project.

2.3. In the Name field, enter **Lab1**.

2.4. Verify that **HDL** is selected as the Top-Level Source Type

2.5. Click **Next**. The New Project Settings page appears.

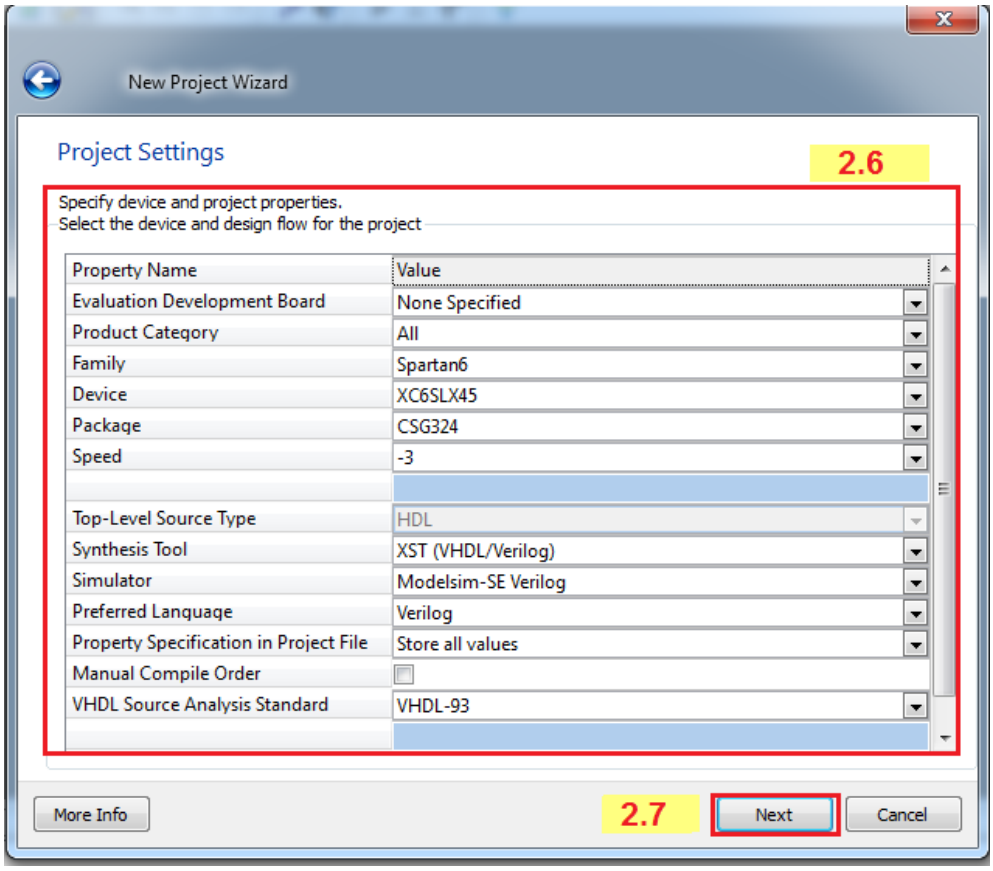

Figure 7. Project Setting page.

2.6. Select the following values in the Project Settings page: (see Figure 7)

- Product Category: **All**
- Family: **Spartan6**
- Device: **XC6SLX45**
- Package: **CSG324**
- Speed: **-3**
- Synthesis Tool: **XST (VHDL/Verilog)**
- Simulator: **ModelSim-SE Verilog**

Note: If you have not ModelSim installed on your system, select **ISim(VHDL/Verilog)** as your simulator.

- 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.

2.7. Click **Next**, then **Finish** to complete the project creation.

3. Creating New Source File (Figure 8-9)

In this section, you will create a Verilog source file using the New Source wizard. The resulting HDL file is then modified in the ISE Text Editor. In order to create the new source file, do the following:

3.1. Select **Project > New Source**, or right-click on the Hierarchy Pane and then select **New Source**. The New Source Wizard opens in which you specify the type of source you want to create.

3.2. In the Select Source Type page, select **Verilog Module**.

3.3. In the File Name field, enter **ComplexMultiplier**.

3.4. Verify that the Add to project box is checked.

3.5. Click **Next**.

3.6. In the Define Module page, enter ports named **Clk**, **Reset**, **InputMultiplier1**, **InputMultiplier2**, and **MultiplicationResult** in the Port Name fields.

3.7. Set the Direction field to **input** for Clk, Reset, InputMultiplier1, and InputMultiplier2 and to **output** for MultiplicationResult**.**

3.8. Leave the Bus designation boxes unchecked for the first two ports and check them for the other ports. Then, set the port width for these ports to 16, and 34, respectively.

Note:

It is better to specify the size of ports and other variables in a parametric format. In this way, your Verilog code has more flexibility for future usages. You can see this note in the final Verilog code of this lab.

3.9. Click **Next** to view a description of the module.

3.10. Click **Finish** to open the empty HDL file in the ISE Text Editor.

3.11. Complete the Verilog code according to the above architecture to realize the complex multiplier.

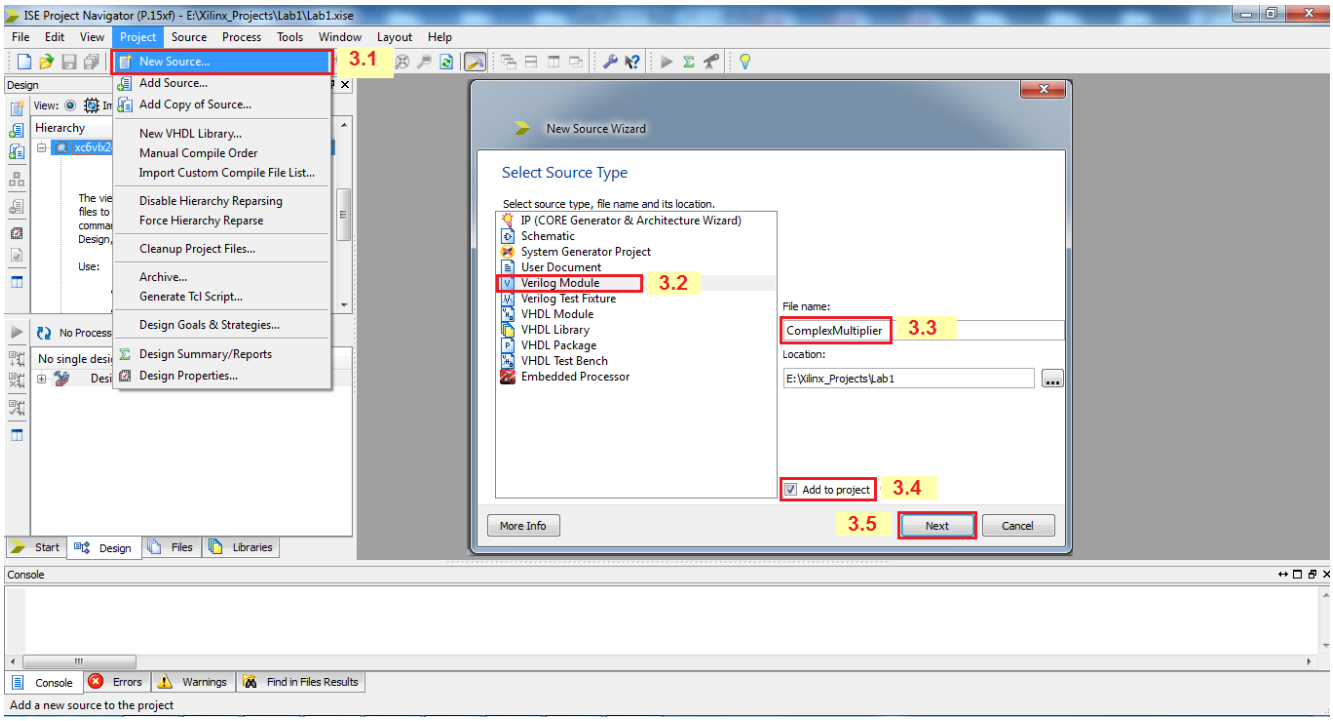

Figure 8. New Source Wizard.

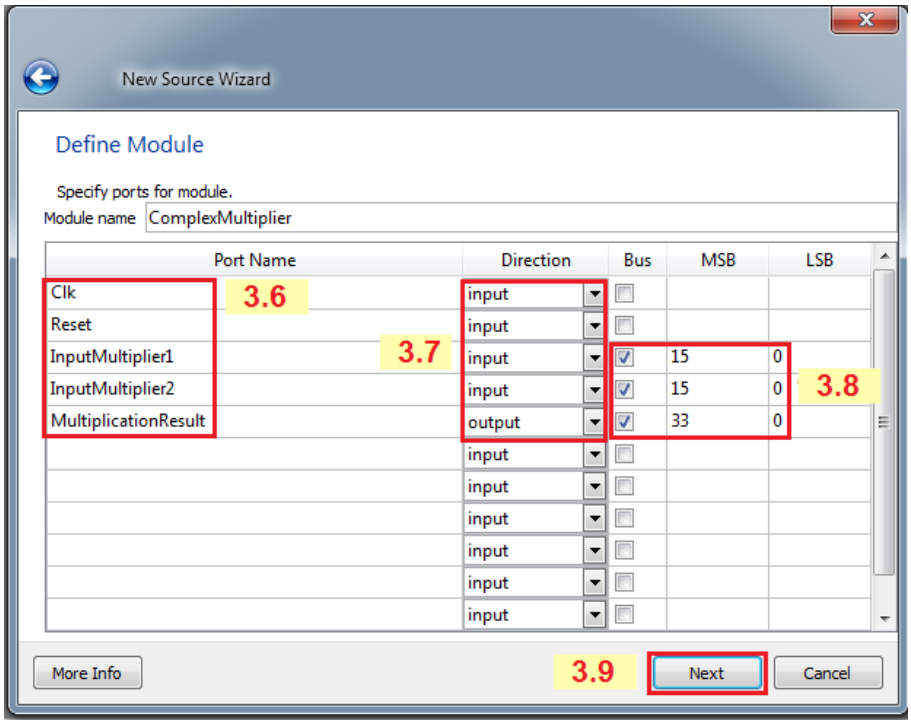

Figure 9. New Source Wizard.

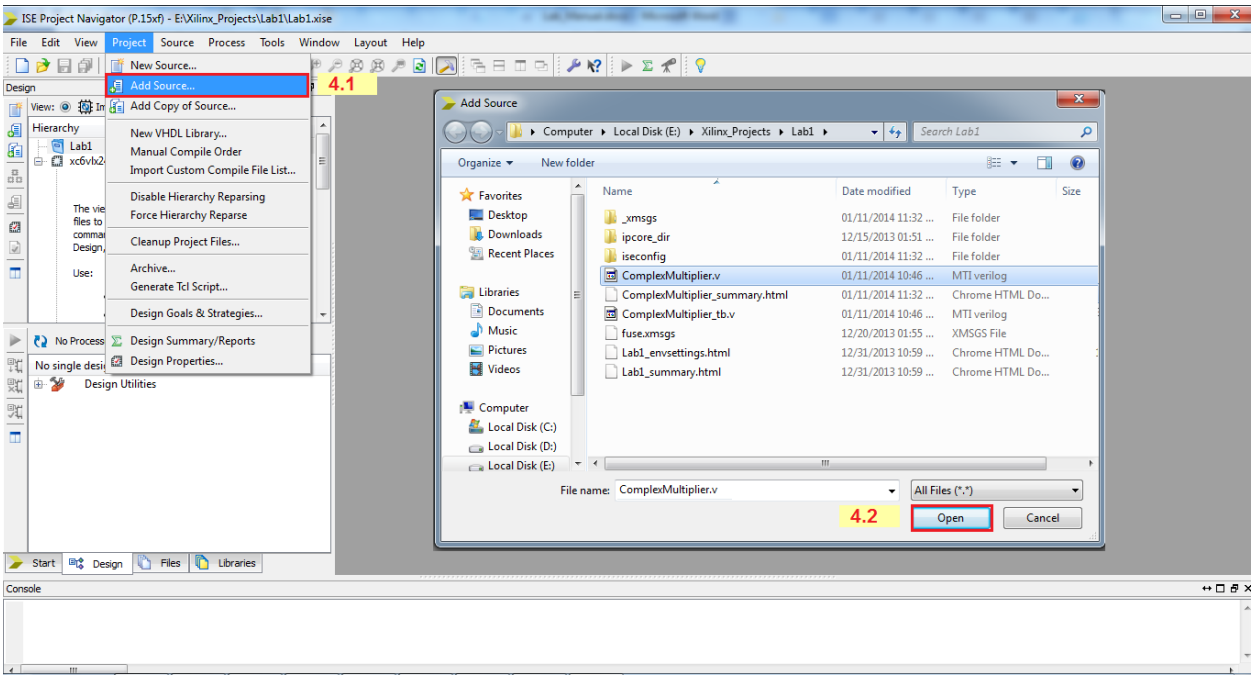

Figure 10. Adding a source to the project.

4. Adding Source Files (Figure 10)

It is possible to write the Verilog code outside the Project Navigator and then add it to the project. Moreover, you can add any existing source file to your project from the previous projects. It is better to copy the source file in the current project directory and then perform the following steps to add it to your project (another method for this purpose will be described in Lab3):

4.1. Select **Project > Add Source**.

4.2. Select the files from the desired directory, and click **Open**.

4.3. Instantiated components with no entity or module declaration are displayed with a question mark in the Hierarchy pane.

Step2: Synthesizing the Design Using XST

During synthesis, the HDL files are translated into gates and optimized for the target architecture.

1. Specify/Change the Top Module of the Design (Figure 11)

In order to specify the top module of your design do following:

1.1. In the Hierarchy pane of the Project Navigator Design panel, select the file, which you need to choose it as the top module of your design.

1.2. Select **Set as Top Module**.

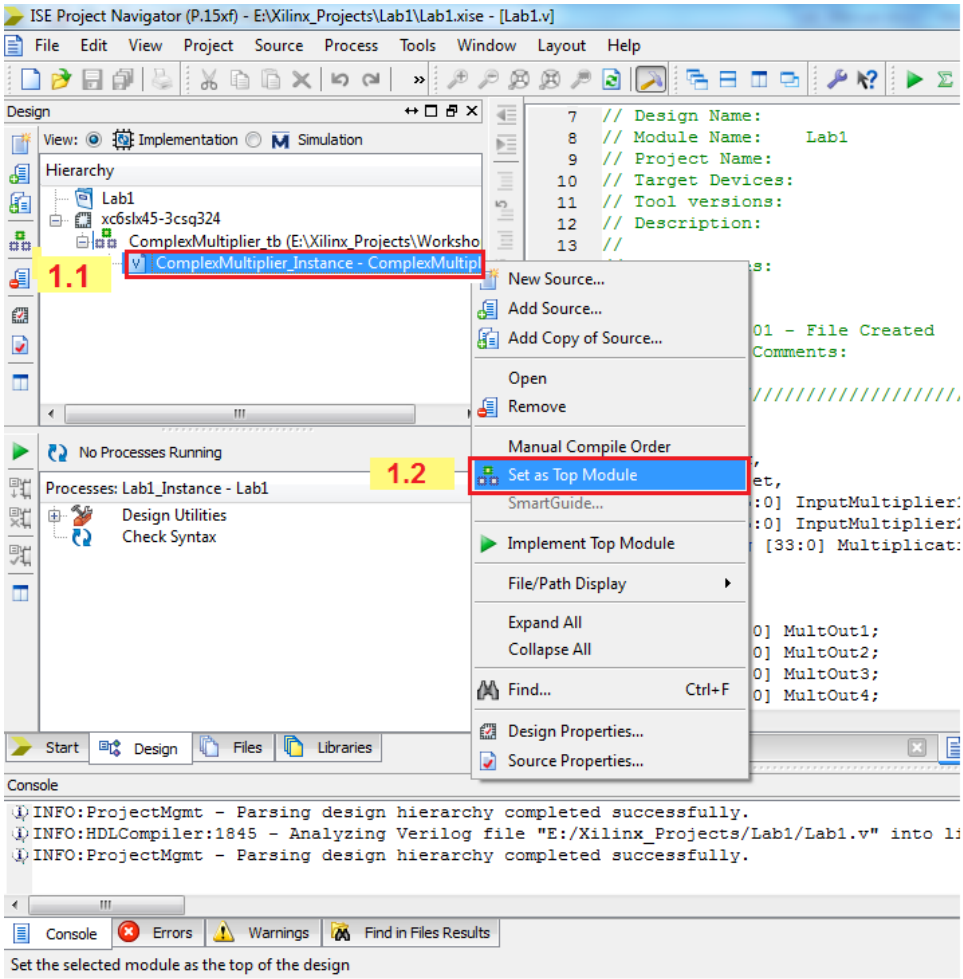

Figure 11.

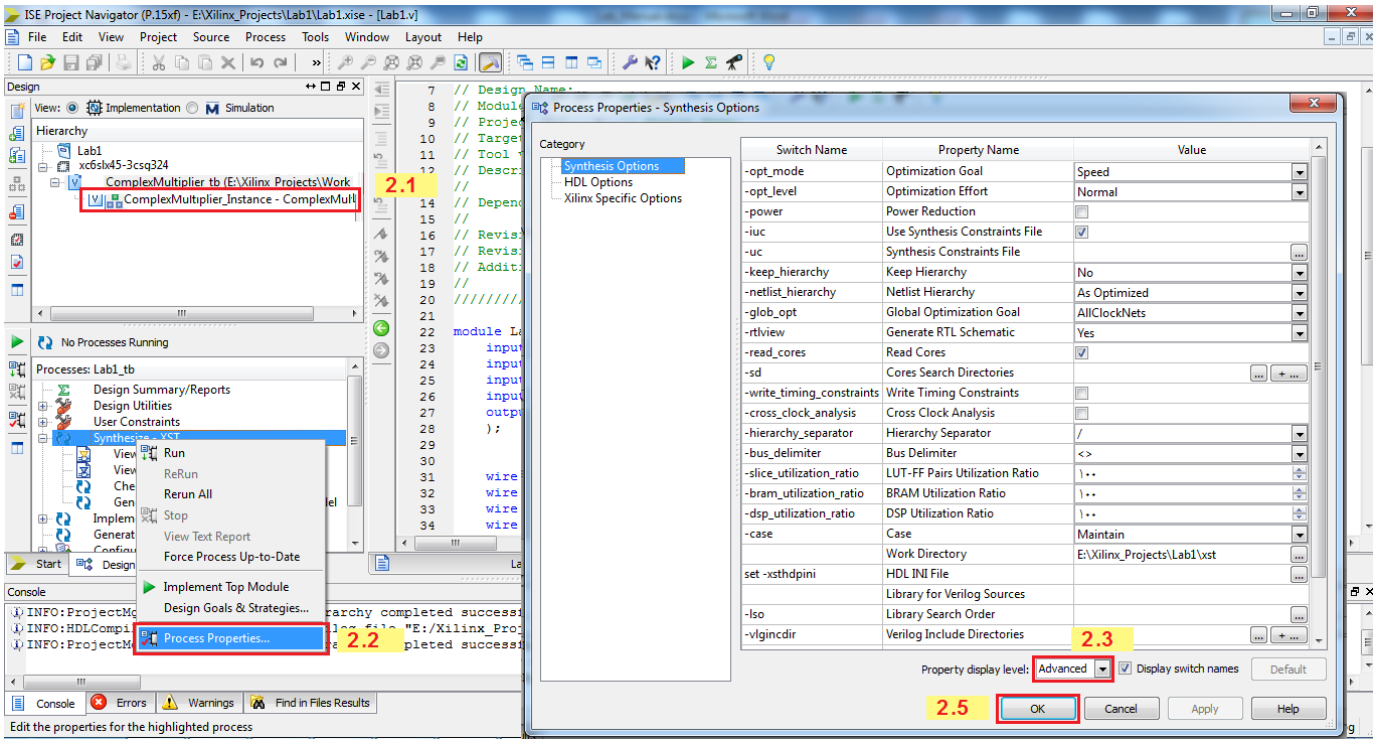

Figure 12.

2. Entering Synthesis Options (Figure 12)

Synthesis options enable you to modify the behavior of the synthesis tool to make optimizations according to the needs of the design including optimizations based on area or speed. For further information please refer to the Lab 10.

In order to enter synthesis options, perform the following steps:

2.1. In the Hierarchy pane of the Project Navigator Design panel, select the top module (i.e. **ComplexMultiplier.v**).

2.2. In the Processes pane, right-click the **Synthesize** process, and select **Process Properties**.

2.3. Set the Property display level to **Advanced**.

2.4. Modify the synthesis options.

2.5. Click **OK**.

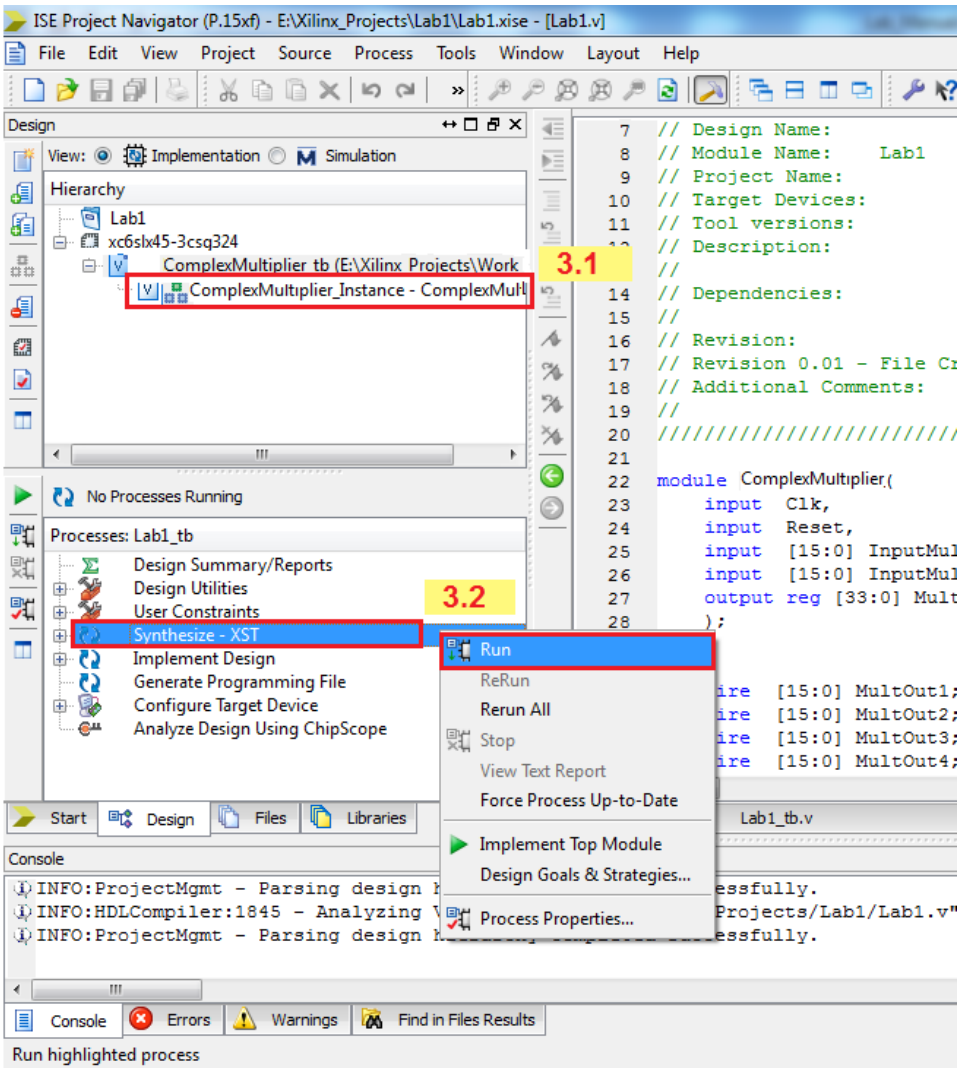

Figure 13.

3. Perform the Synthesis process for the Design (Figure 13)

Synthesizing the design can be done as follow:

3.1. In the Hierarchy pane, select the top module (i.e. **ComplexMultiplier.v**).

3.2. In the Processes pane, double-click the **Synthesize** process or right-click on the **Synthesize** process and then select **Run/ReRun**.

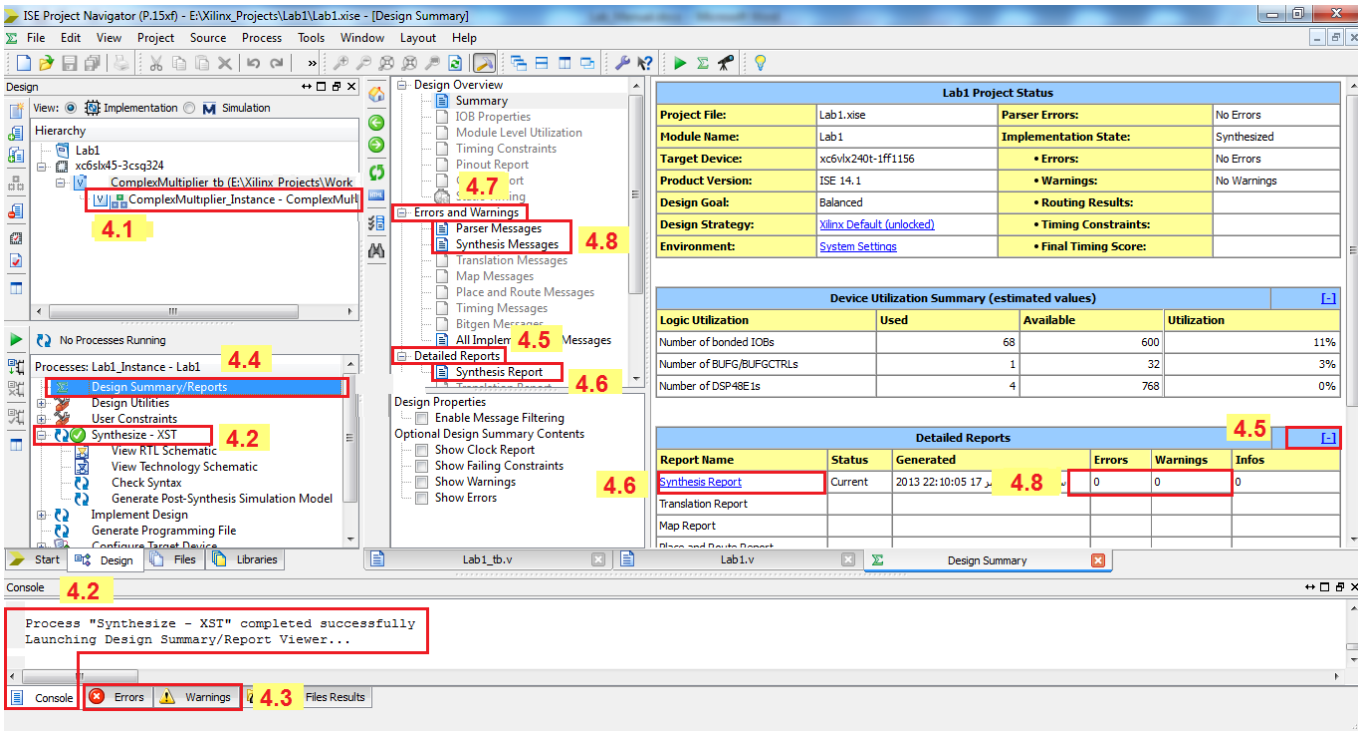

Figure 14.

4. View the Design Summary/Reports (Figure 14)

4.1. In the Hierarchy pane, select the top module (i.e. **ComplexMultiplier.v**).

4.2. In the Processes pane, double-click the **Design Summary/Reports**.

4.3. If the Synthesis process is not successful, read the content of Errors/Warnings tabs in the Transcript Window to check the Errors/Warnings of Synthesis process.

4.4. In the Processes pane, double-click **Design Summary/Reports**.

4.5. In the Design Summary page, expand **Detailed Report**.

4.6. Select **Synthesis Report**.

4.7. In order to read the Errors/Warnings of Synthesis process expand **Errors and Warnings**.

4.8. Select **Synthesis Messages**.

5. Change the Synthesis Tool (Figure 15)

The synthesis tool can be changed at any time during the design flow.

To change the synthesis tool, do the following:

5.1. In the Hierarchy pane of the Project Navigator Design panel, select the targeted **FPGA part number**.

5.2. Right-click and select **Design Properties**.

5.3. In the Design Properties dialog box, click the Synthesis Tool value and use the pull-down arrow to select the desired synthesis tool from the list.

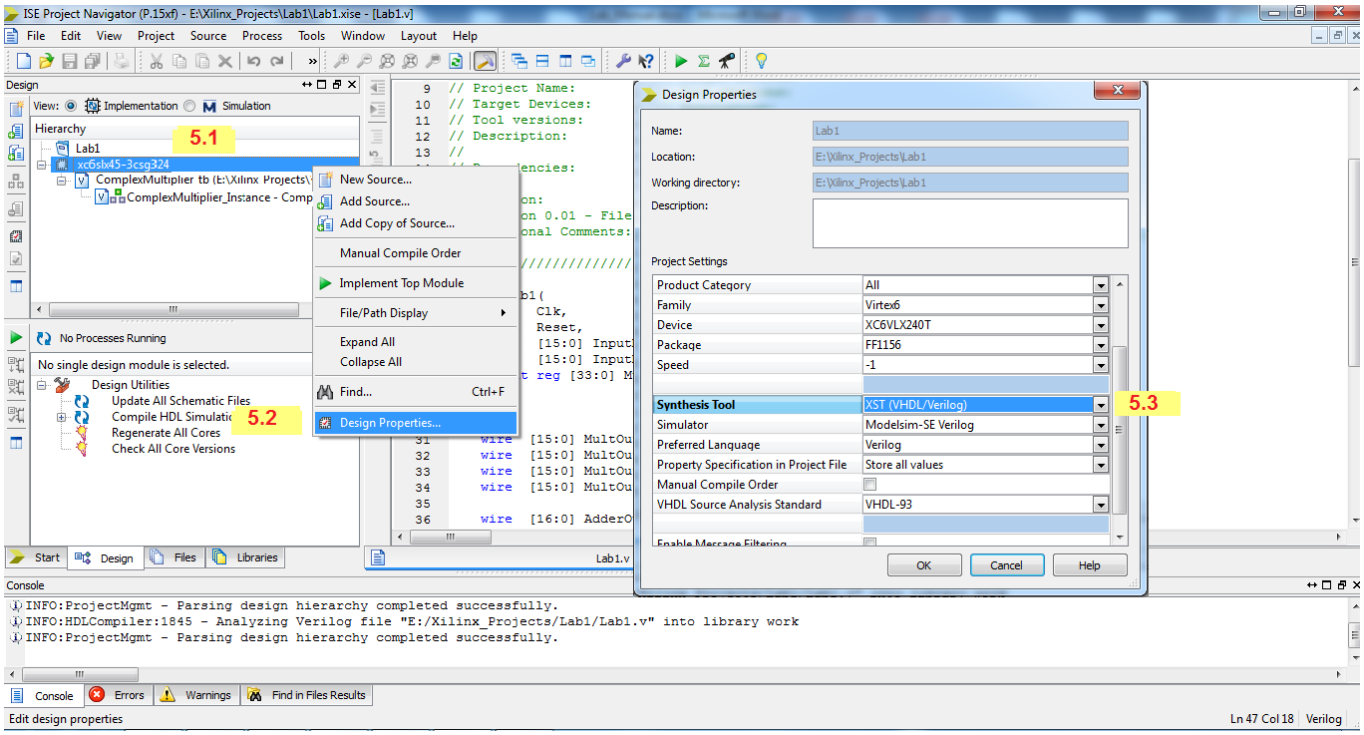

Figure 15.

Step3: Using the RTL/Technology Viewer

XST can generate a schematic representation of the HDL code that you have entered. You can generate a schematic view of your RTL netlist using **RTL Viewer** and also you can generate a schematic view of your technology netlist using **Technology Viewer**.

1. View the RTL/Technology Schematics (Figure 16)

To view a schematic representation of the HDL code, do the following:

1.1. In the Hierarchy pane, select the top module (i.e. **ComplexMultiplier.v**).

1.2. In the Processes pane, expand **Synthesize**, and double-click **View RTL Schematic** or **View Technology Schematic**.

1.3. In the Create Schematic start page, select **Start with a schematic of the top-level block**.

Note: You can select **Start with the Explorer Wizard** and then select the desired components to create the target schematic.

1.4. The schematic viewer allows you to select the portions of the design to display as schematics. When the schematic is displayed, double-click on the symbol to push into the schematic and view the various design elements and connectivity. Right-click the schematic to view the various operations that can be performed in the schematic viewer.

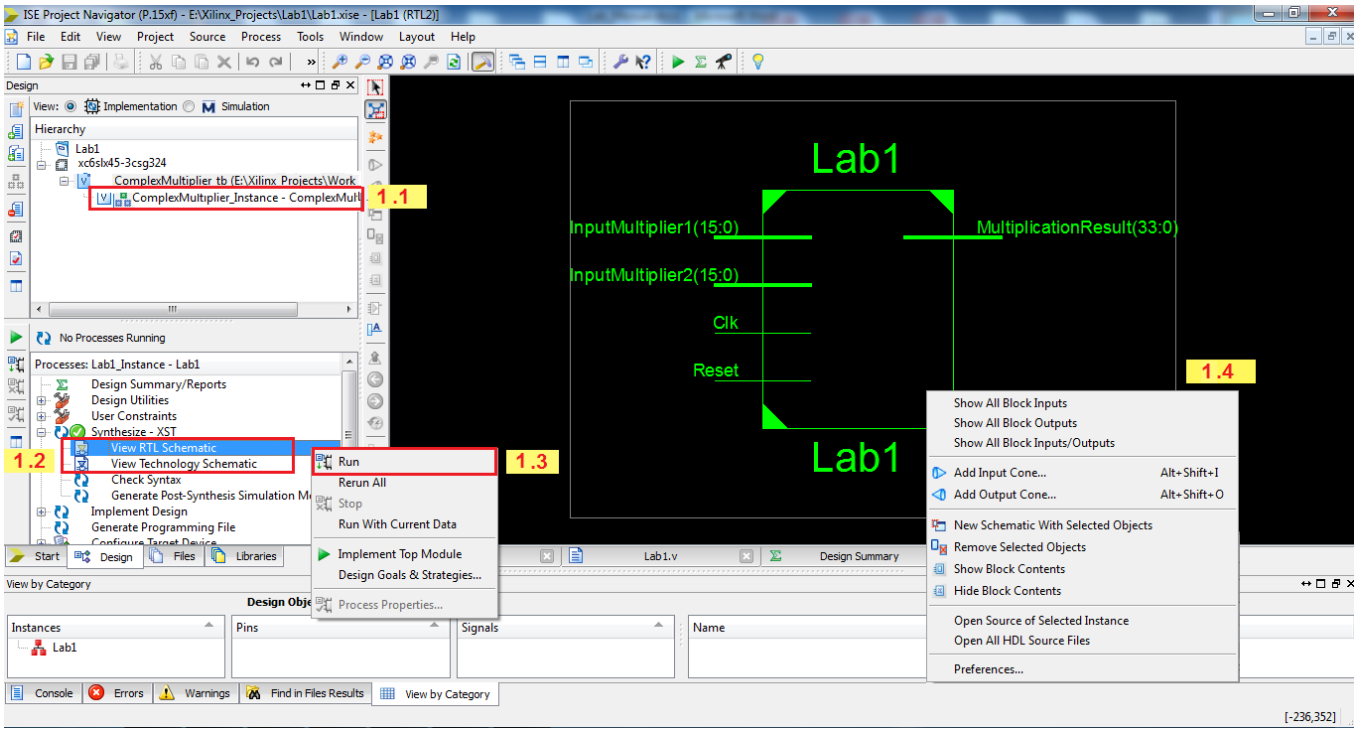
ISE Project Navigator (P.15xf) - E:\Xilinx_Projects\Lab1\Lab1.xise - [Lab1 (RTL2)]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
Lab1
xc6slx45-3csg324
ComplexMultiplier_tb (E:\Xilinx_Projects\Work
ComplexMultiplier_Instance - ComplexMulti
1.1
No Processes Running
Processes: Lab1_Instance - Lab1
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis Simulation M
Implement Design
Generate Programming File
1.2
Run
Rerun All
Stop
Run With Current Data
Implement Top Module
Design Goals & Strategies...
Process Properties...
1.3
Start Design Files Libraries
Lab1.v
Design Summary
Lab1
InputMultiplier1(15:0)
InputMultiplier2(15:0)
Clk
Reset
MultiplicationResult(33:0)
Lab1
1.4
Show All Block Inputs
Show All Block Outputs
Show All Block Inputs/Outputs
Add Input Cone... Alt+Shift+I
Add Output Cone... Alt+Shift+O
New Schematic With Selected Objects
Remove Selected Objects
Show Block Contents
Hide Block Contents
Open Source of Selected Instance
Open All HDL Source Files
Preferences...
View by Category
Instances
Lab1
Pins
Signals
Name
Console Errors Warnings Find in Files Results View by Category
[-236,352]

Figure 16.

Digital Circuit Design with Xilinx FPGA

Lab Manual

Workshop No.1

Lab2:

Getting Started with ModelSim/ISim Simulators

Outline

This manual covers the following topics:

- Preparing the design for simulation
- Introduction about the integrated simulation flow in the Project Navigator
- Simulating the design using ModelSim
- Simulating the design using ISim

Introduction

In order to verify the functionality of the design, it is necessary to perform a behavioral simulation for the design. You can run "Mentor ModelSim" simulator or the "Xilinx ISim simulator" to perform a behavioral simulation from the Xilinx Project Navigator using integrated simulation flow of Project Navigator. Project Navigator automatically links to the specified simulator and prepares the design for the simulation.

Required Software

To perform this manual, you must have **Xilinx ISE Design Suite 14.1** and **ModelSim 10.1** (or above) installed. Note that in this manual the pictures are captured in **Xilinx ISE Design Suite 14.1**, but all the procedures can be done in other versions of this software.

Objectives

After completing this lab, you will be able to:

- *Prepare a test bench for your design*
- *Use integrated simulation flow in the Project Navigator*
- *Perform behavioral simulation using ModelSim*

- *Perform behavioral simulation using ISim*
- *Analyze the simulation results*

General Flow of this Lab

This lab comprises of three steps as follows:

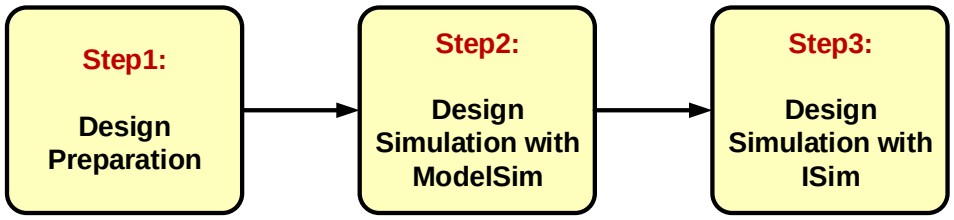

Figure 1. General flow of Lab2.

Design Description

In this lab, you should use the final design of Lab1. However, the simulation process, which is described in this manual is useful for simulating the other designs. In order to start this lab, perform the following steps.

Design Preparation for Behavioral Simulation

In order to perform simulation for your design the following files are required:

- Design files
  - These are the source files such as Verilog/VHDL files, which are generated in the previous steps.
- Test Bench

  - To simulate a design, a test bench file is required to provide stimulus to the design. In this manual we provide a simple test bench, but you can write a new one. To create your own test bench file in Project Navigator, select **Project > New Source**, and select either **VHDL Test Bench** or **Verilog Text Fixture** in the New Source Wizard. An empty stimulus file is added to your project. Then, you can complete the content of that file in the Project Navigator workspace.
- Simulation libraries
  - If a Xilinx primitive or IP core is instantiated in the design, Xilinx simulation libraries are required.

1. Start the Project Navigator

Double click the Project Navigator icon on your desktop, or select **Start > All Programs > Xilinx Design Tools > ISE Design Suite 14.1 > ISE Design Tools > Project Navigator.**

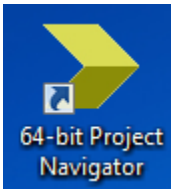

Figure 2. Project Navigator desktop icon.

2. Open the project (Figure 3)

2.1. From Project Navigator, select **File > Open Project**.

2.2. Go to the project directory of Lab1.

2.3. Select **Lab1.xise**.

2.4. Click **Open**.

3. Copy the project (Figure 4)

3.1. From Project Navigator, select **File > Copy Project**.

3.2. Specify the location, in which you want to save the result of this lab.

3.3. Enter **Lab2** as the name of the copied project.

3.4. Select **Copy sources to the new location**.

3.5. Select the **Open the copied project** to open the Lab2 project automatically.

3.6. Click **OK**.

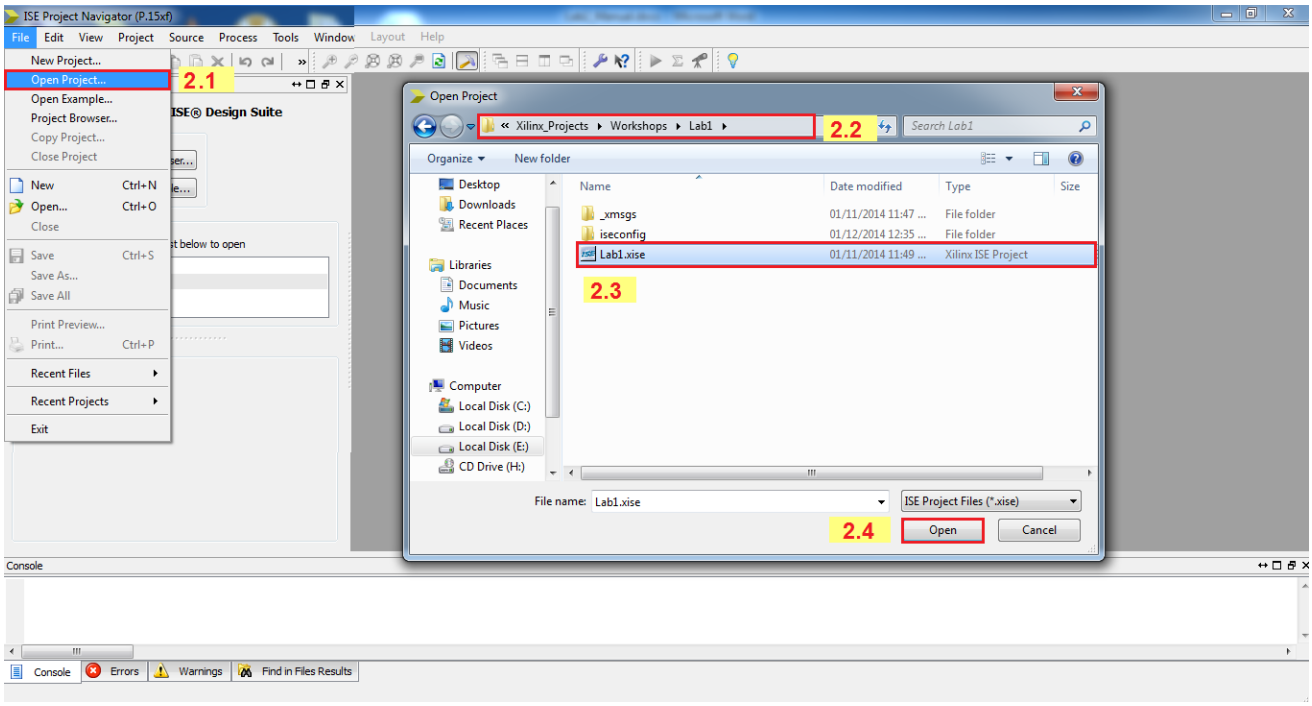

Figure 3.

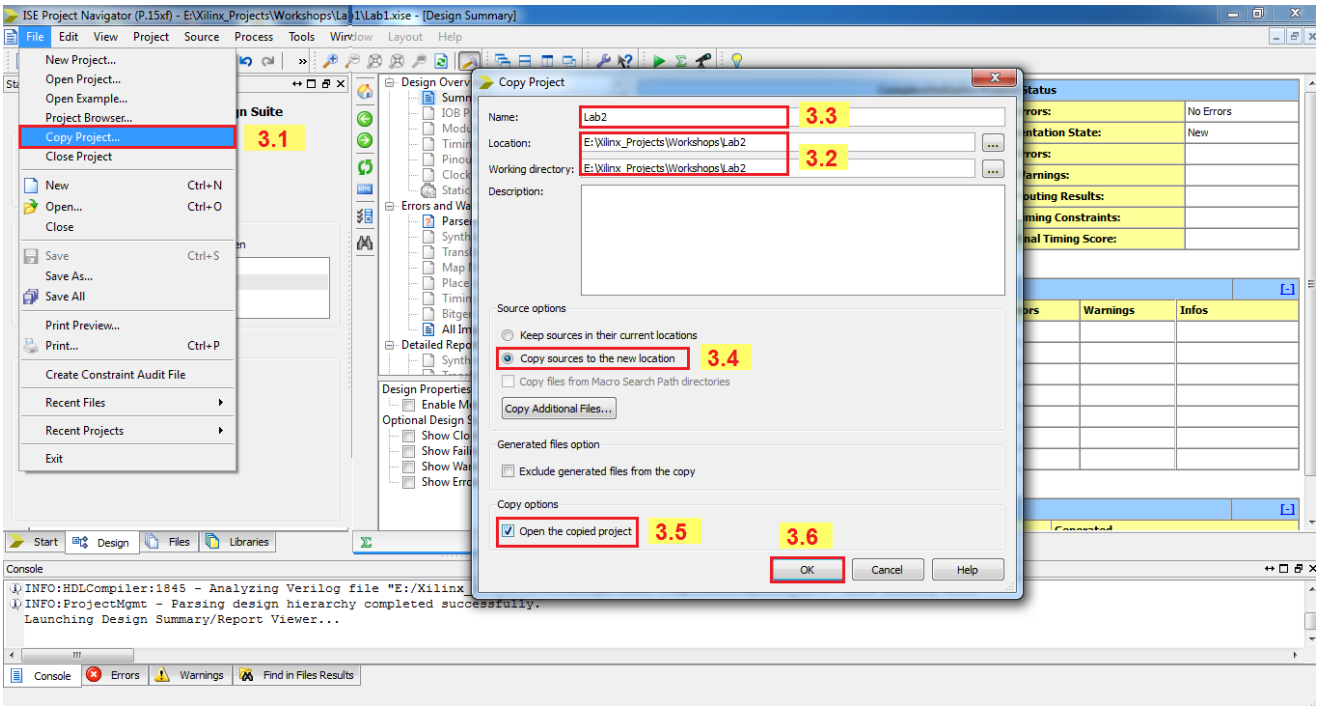

Figure 4.

Behavioral Simulation using ModelSim

1. Add a Test Bench to the Design (Figure 5-6)

In order to add your existing test bench to the design, do the following steps:

1.1. In the Hierarchy pane, select the top module (i.e. **ComplexMultiplier.v**).

1.2. In Project Navigator, select **Project > Add Source**.

1.3. Select the test bench file **ComplexMultiplier_tb.v**.

1.4. Click **Open**.

1.5. Ensure that Simulation is selected for the file association type.

1.6. Click **OK**.

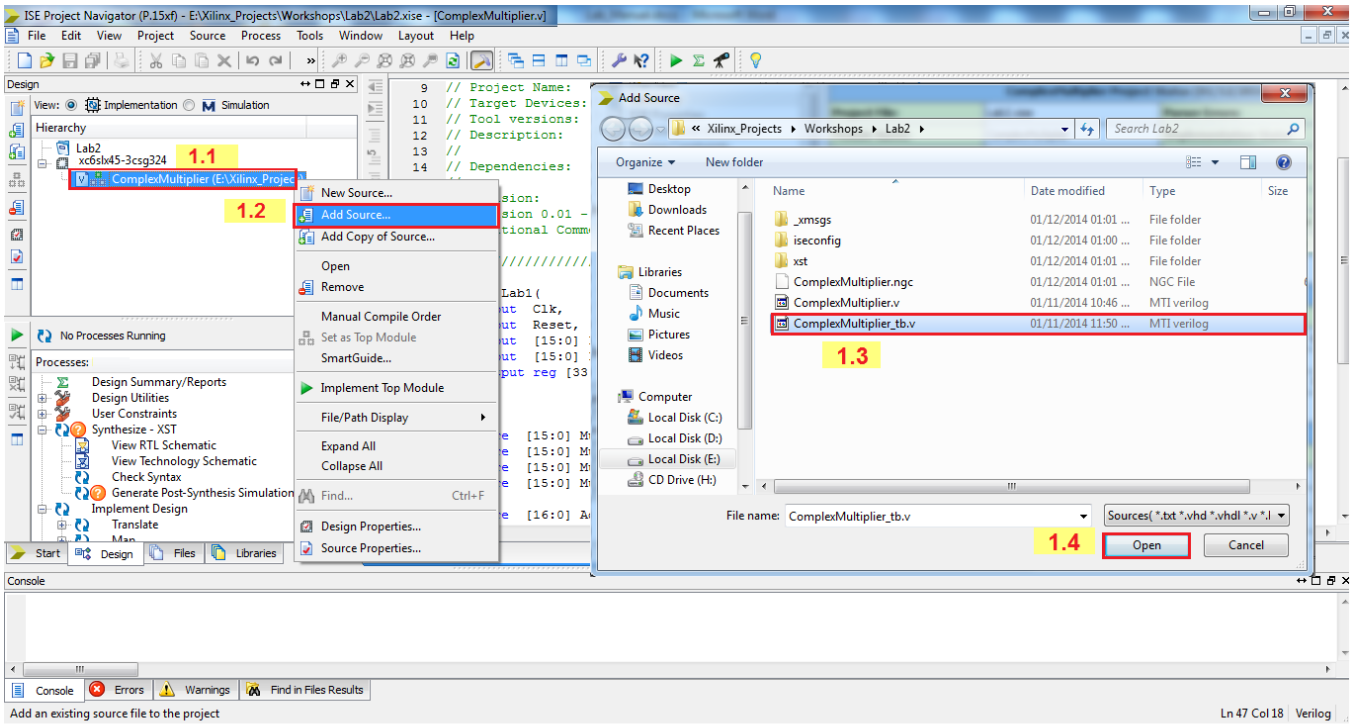

Figure 5.

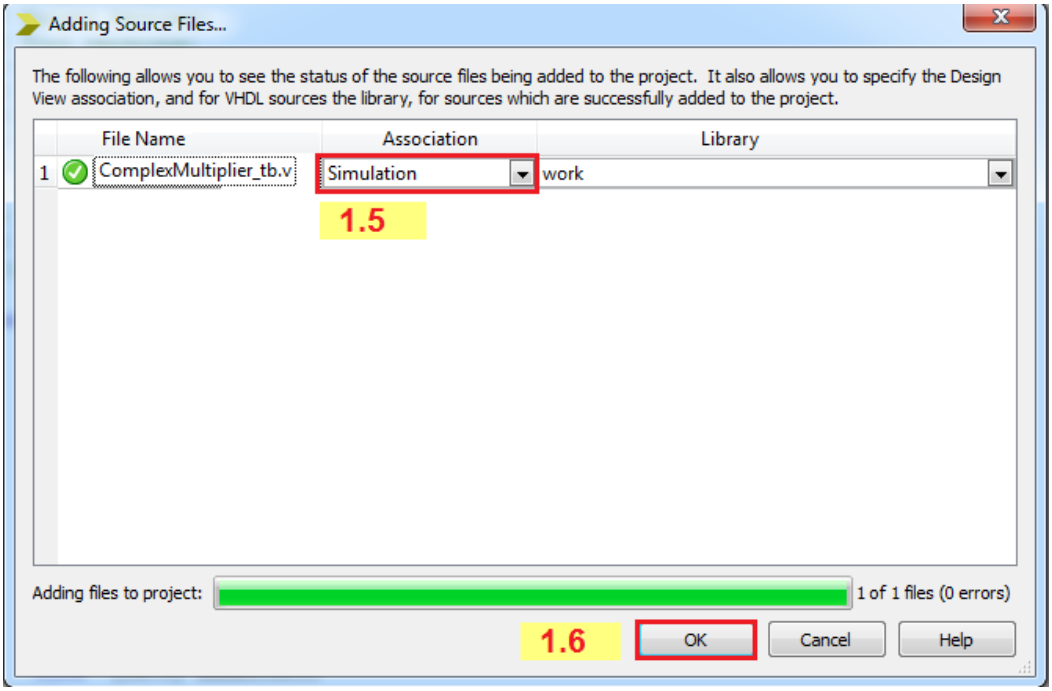

Figure 6.

2. Select ModelSim as the Simulator (Figure 7)

2.1. Select the **FPGA part number** in the Hierarchy pane of the Project Navigator Design panel.

2.2. Right-click on the target **FPGA part number** and select **Design Properties**.

2.3. In the Design Properties dialog box, set the Simulator field to **ModelSim** (with the appropriate type and language).

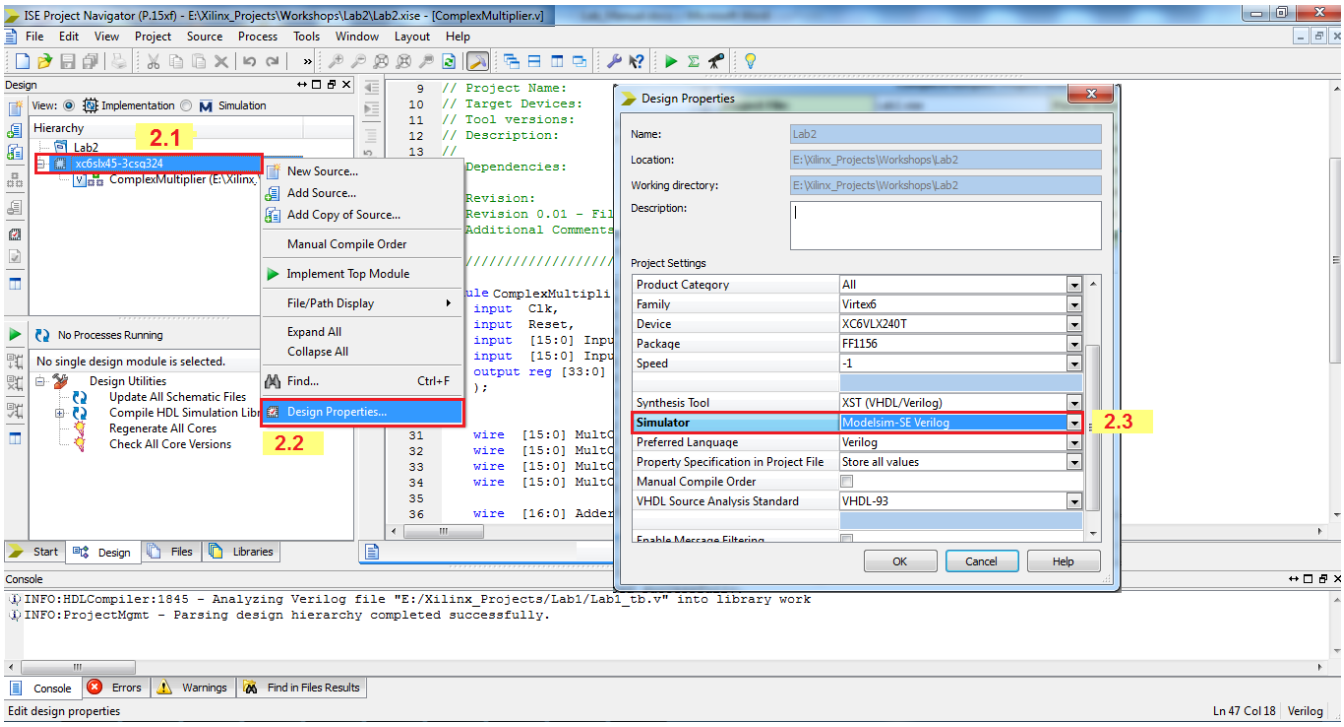

Figure 7.

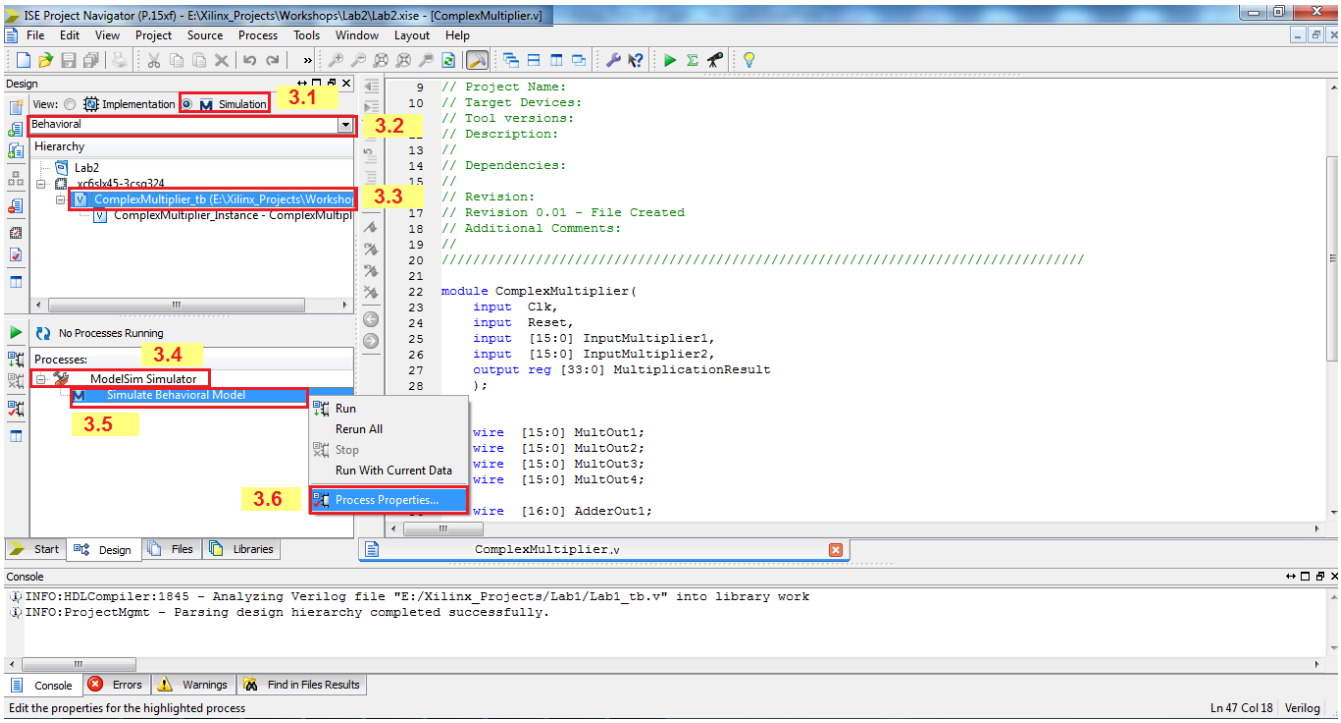

Figure 8.

3. Specifying Simulation Properties (Figure 8-9)

To locate the ModelSim simulator processes and set the simulation properties, do the following:

3.1. In the View pane of the Project Navigator Design panel, select **Simulation**.

3.2. Select **Behavioral** from the drop-down list.

3.3. In the Hierarchy pane, select the **test bench file** (**ComplexMultiplier_tb.v**).

3.4. In the Processes pane, expand **ModelSim Simulator** to view the process hierarchy.

3.5. Right-click **Simulate Behavioral Model.**

3.6. Select **Process Properties**.

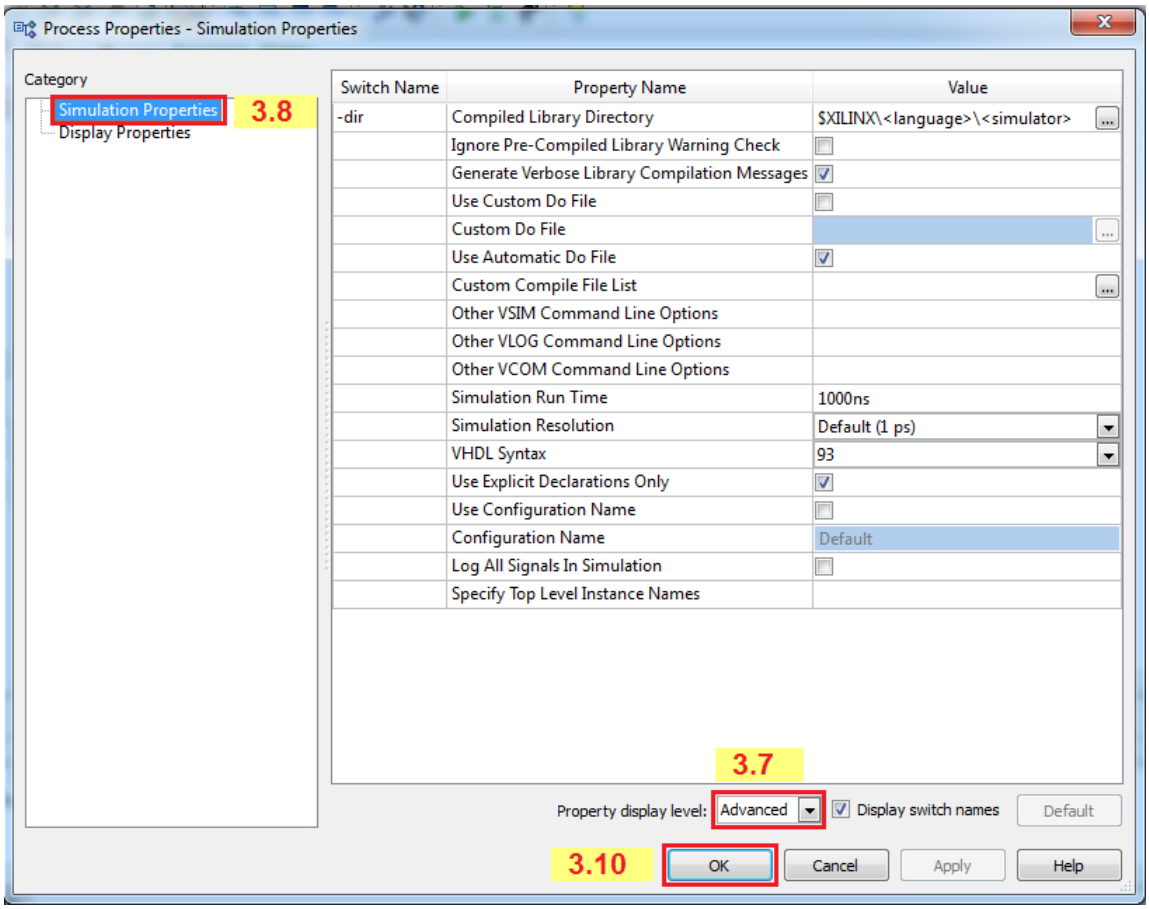

Figure 9.

3.7. In the Process Properties dialog box, set the Property display level to **Advanced**.

3.8. Select **Simulation Properties** in the left side of the window.

3.9. Change the simulation properties.

3.10. Click **OK**.

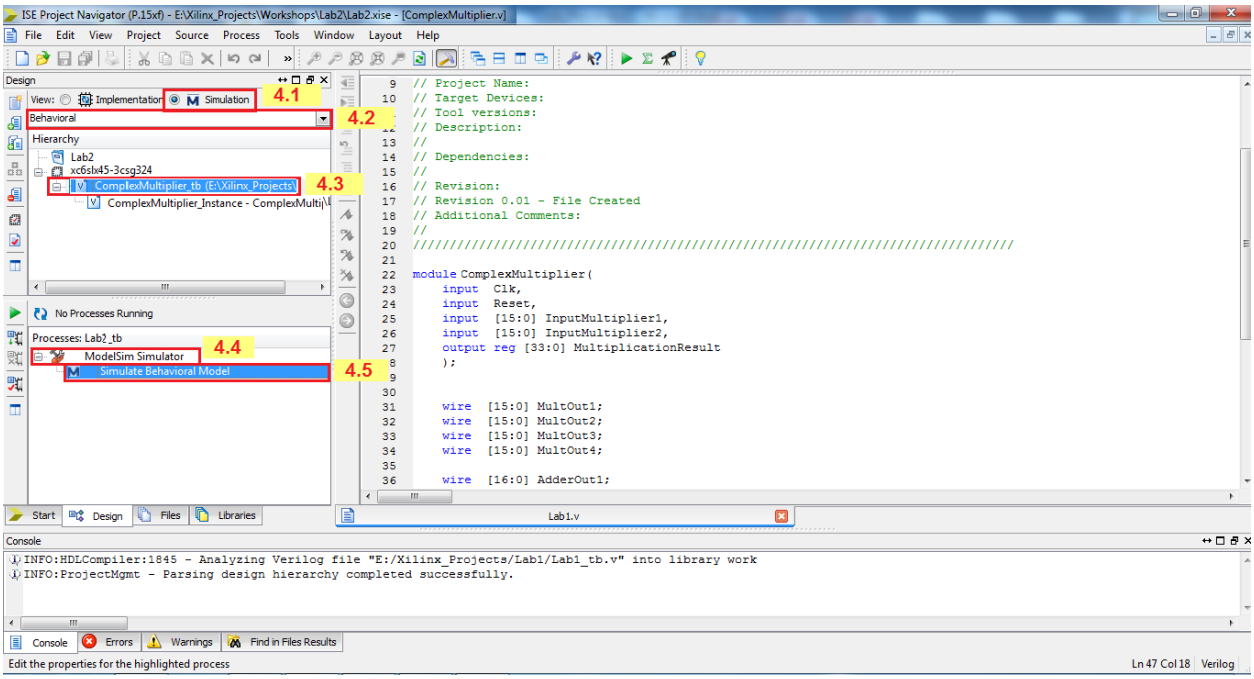

Figure 10.

4. Start the Behavioral Simulation (Figure 10)

4.1. In the View pane of the Project Navigator Design panel, select **Simulation**.

4.2. Select **Behavioral** from the drop-down list.

4.3. In the Hierarchy pane, select the **test bench file** (**ComplexMultiplier_tb.v**).

4.4. In the Processes pane, expand **ModelSim Simulator** to view the process hierarchy.

4.5. Double- click **Simulate Behavioral Model**.

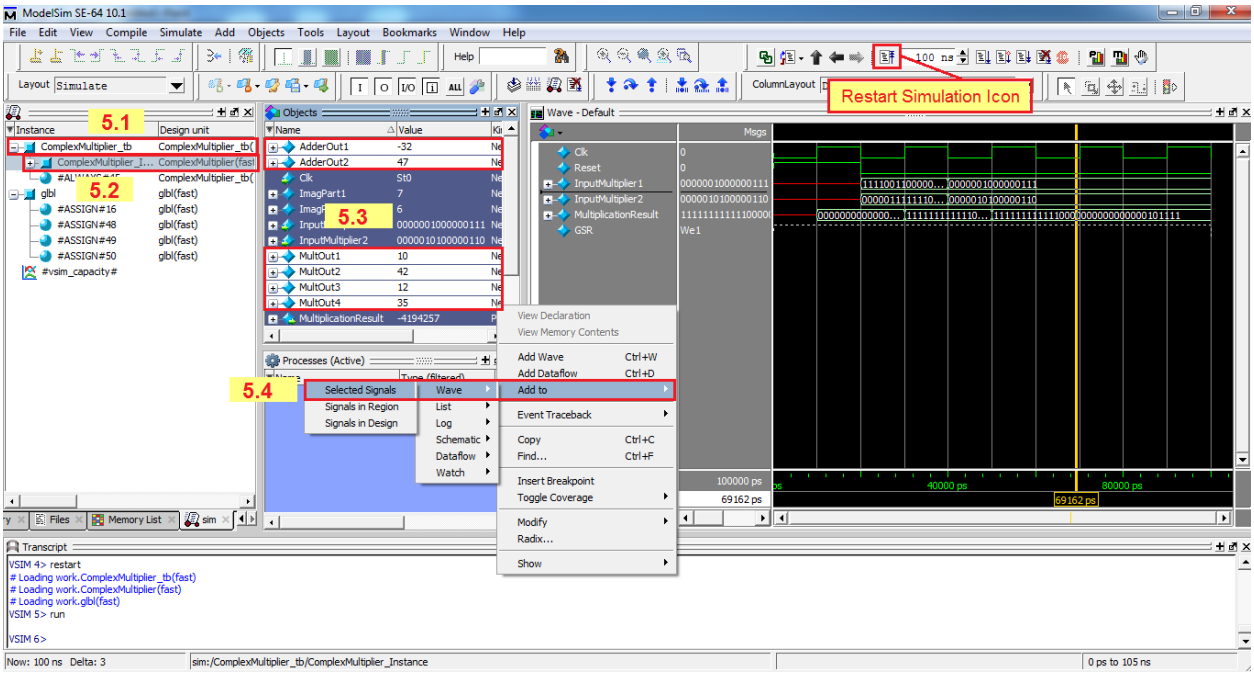

Figure 11.

5. Adding Signals to Wave Window (Figure 11)

The ISE Design Suite automatically adds all the top-level ports to the Wave window. But if you need to view and analyze the intermediate signals, perform the following steps:

5.1. In the Instance window (i.e. in the left side of the main window in the ModelSim), expand the **ComplexMultiplier_tb**.

5.2. Select and expand the **ComplexMultiplier _Instance**. All the components within this design instance will be shown. Also, all of the corresponding signals will be shown in the Object window.

5.3. Select the intermediate signals named **MultOut1, MultOut2, MultOut3, MultOut4, AdderOut1,** and **AdderOut2.**

5.4. Right-click on the selected signals and select **Add to > Wave > Selected Signals**.

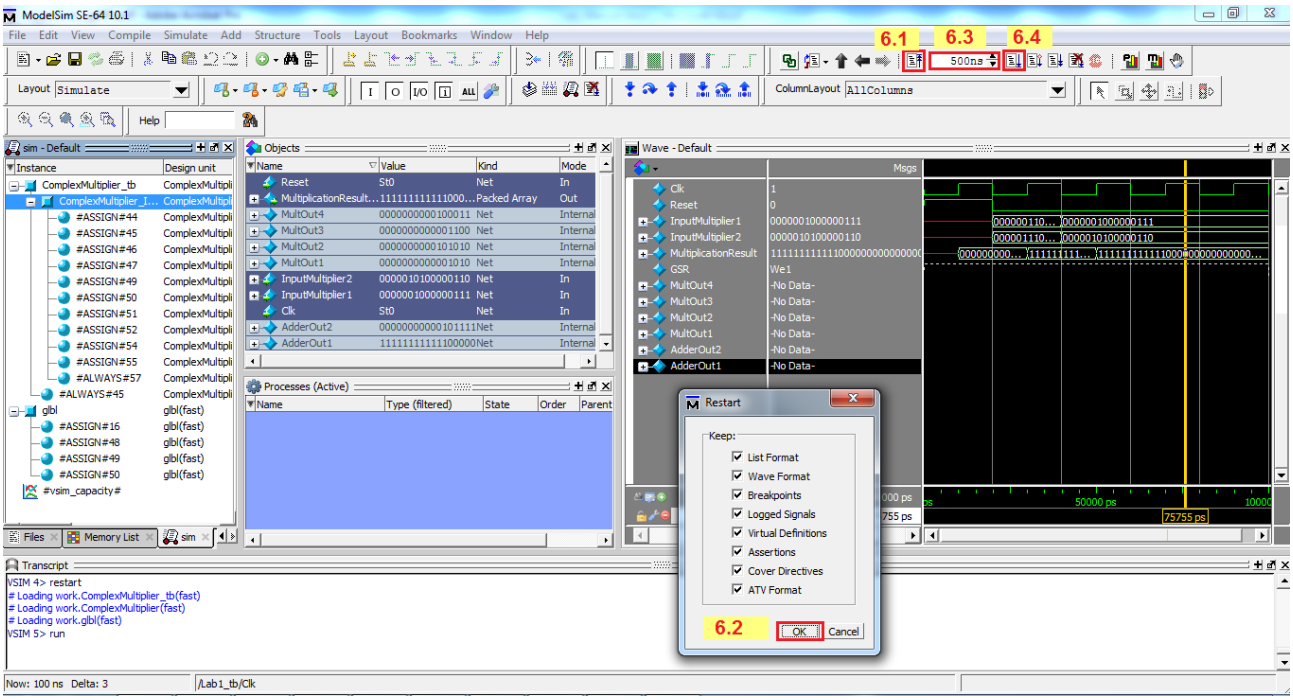

Figure 12.

6. Performing the Simulation (Figure 12)

In order to run the simulation in the ModelSim, do the following steps:

6.1. On the top of the main window, click on the **Restart Simulation Icon** (See Figure 12).

6.2. In the Restart dialog box, click **OK**.

6.3. In the Run Length box, set the simulation time to **500 ns**.

6.4. Click **Run** to start the simulation.

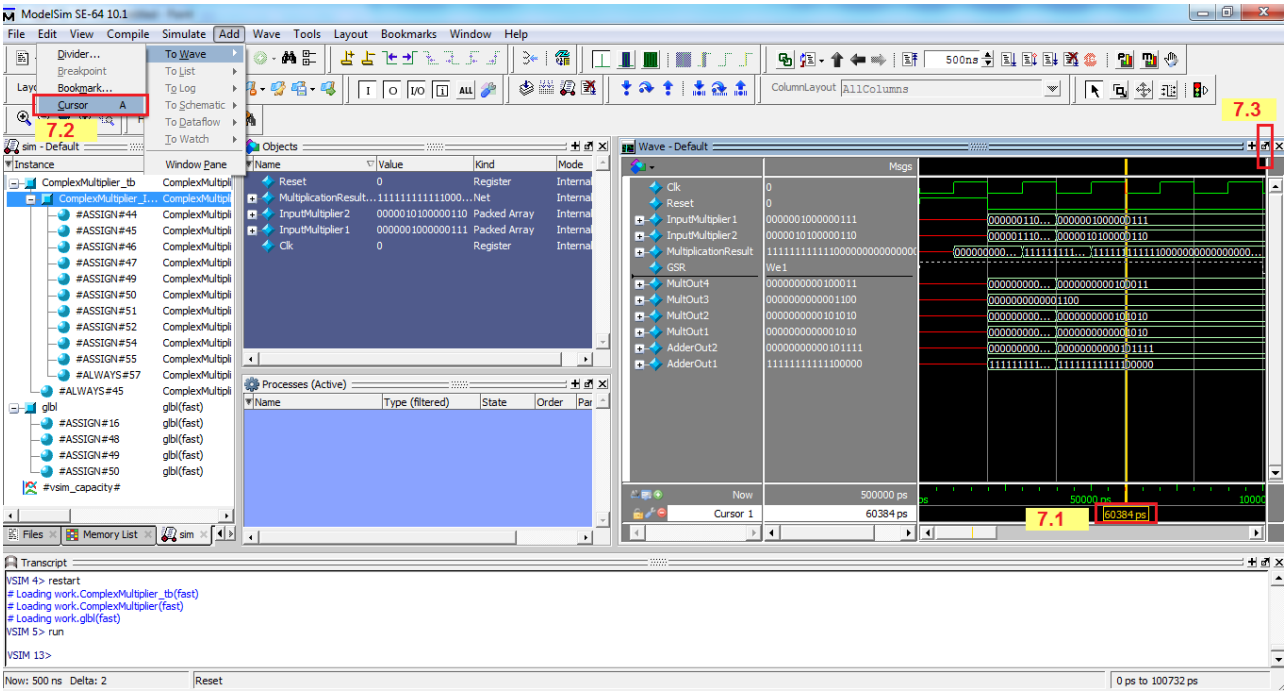

Figure 13.

7. Simulation Result Analysis (Figure 13)

7.1. Use **Cursor** to track the signals or find/compare different point in the simulation results.

7.2. Select **Add > To Wave > Cursor** to add another cursor for further comparison or additional actions on the signals.

7.3. You can use **Dock/Undock** icon to view the Wave window in a separate window.

7.4. Walk through the signals using the cursors and check the results. You can verify the simulation results in this step using MATLAB. It is very easy to perform the complex multiplier in MATLAB and then compare the results with ModelSim outputs based on the same inputs.

NOTE:

You should perform this verification for each design to assure that it works well and its functionality is true. Almost MATLAB is useful in this step.

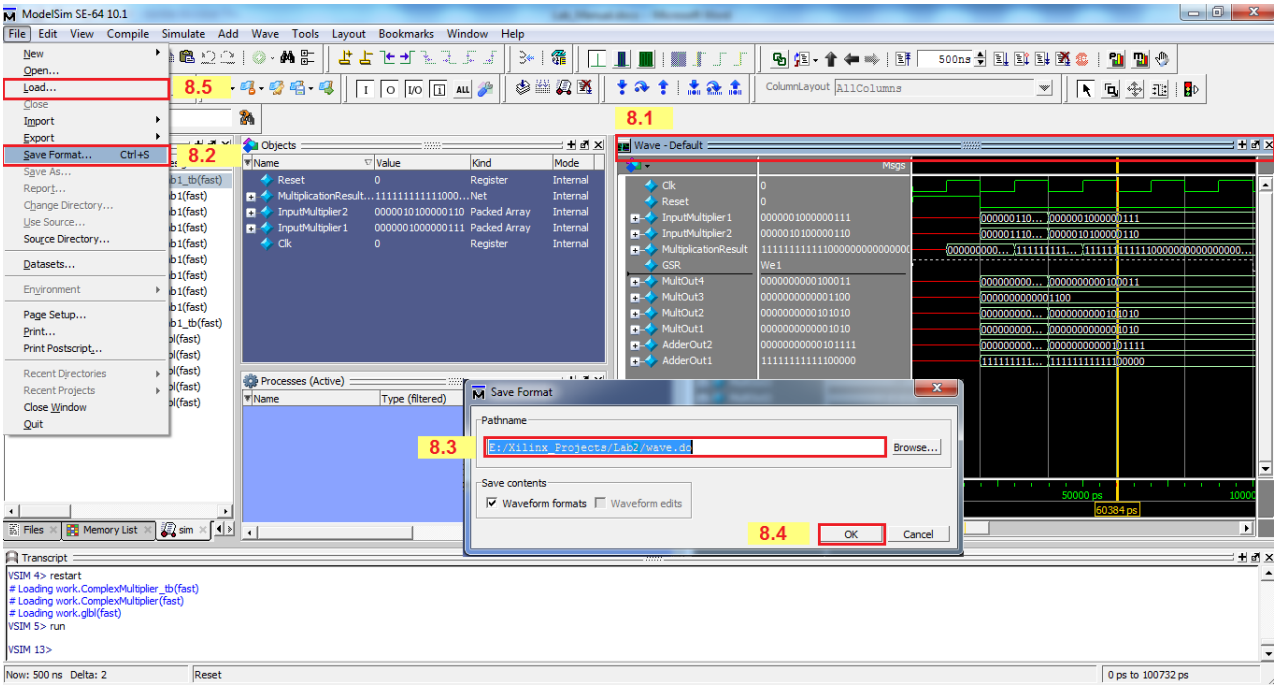

Figure 14.

8. Saving the Simulation (Figure 14)

You can save the signal list of the Wave window. The saved signals list can easily be opened each time the simulation is started.

To save the signals list, do the following:

8.1. Click on the **Wave window**.

8.2. Refer to the menus and select **File > Save Format**.

8.3. In the Save Format dialog box, enter the desire **name and location**.

8.4. Click **OK**.

8.5. After restarting the simulation, select **File > Load** in the Wave window to load this file.

Behavioral Simulation using ISim

1. Add a Test Bench to the Design (Figure 15-16)

In order to add your existing test bench to the design, perform the following steps:

1.1. In the Hierarchy pane, select the top module (i.e. **ComplexMultiplier.v**).

1.2. In Project Navigator, select **Project > Add Source**.

1.3. Select the test bench file **ComplexMultiplier _tb.v**.

1.4. Click **Open**.

1.5. Ensure that Simulation is selected for the file association type.

1.6. Click **OK**.

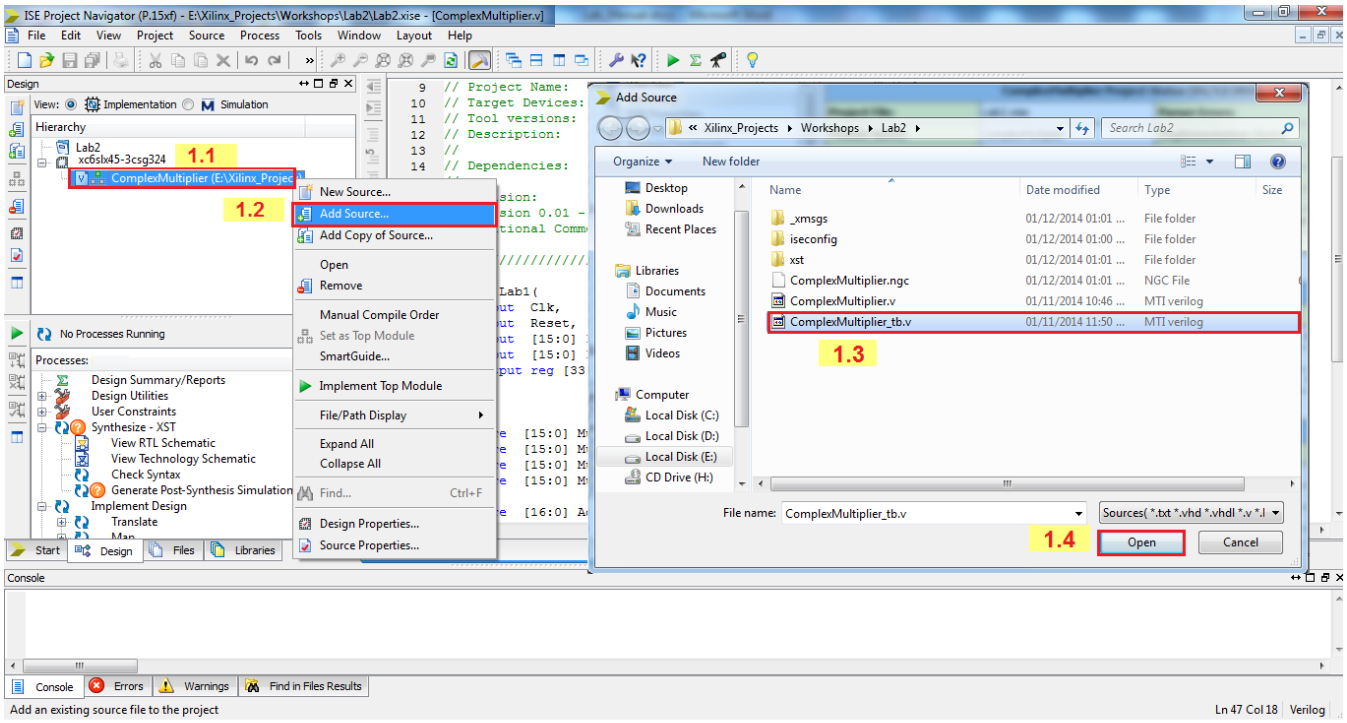

Figure 15.

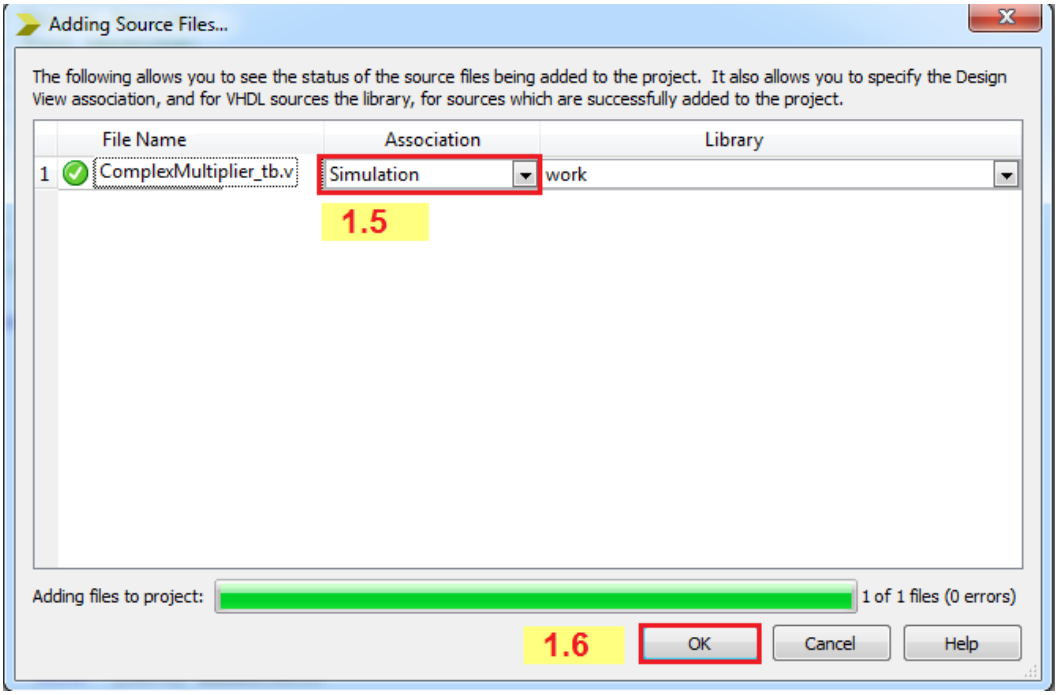

Figure 16.

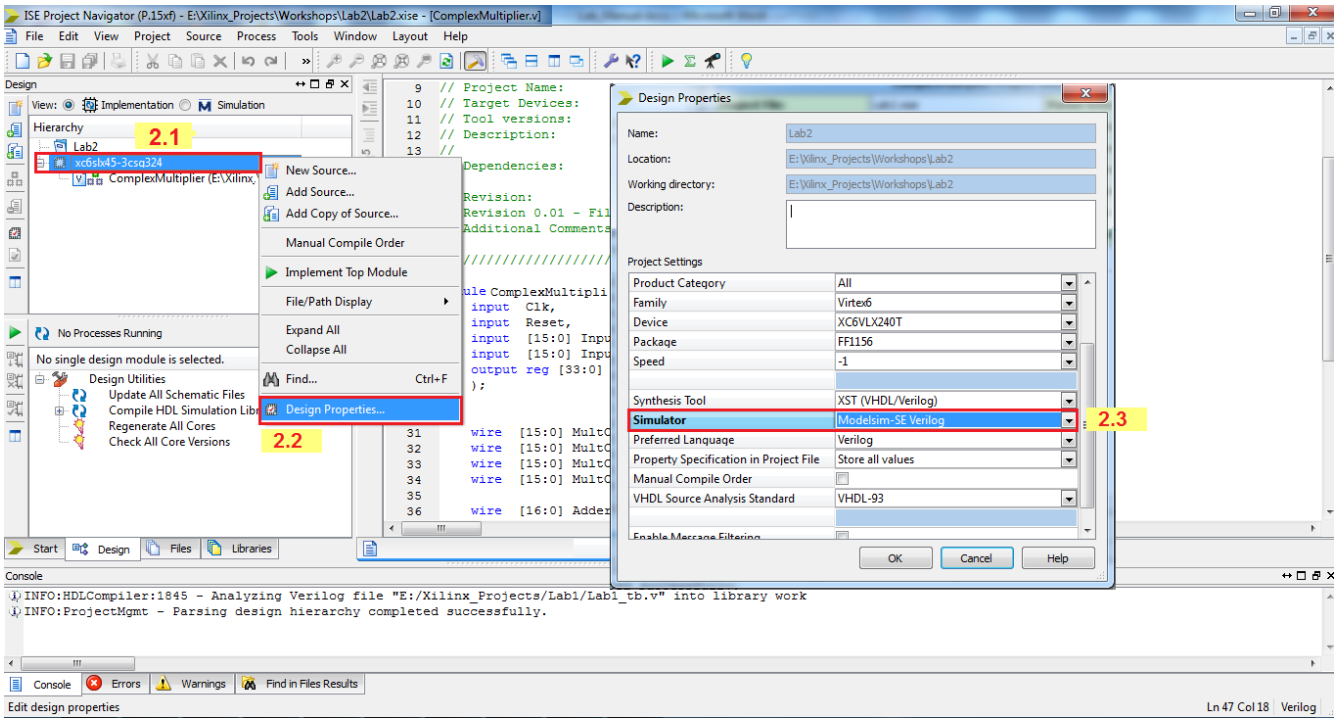

Figure 17.

2. Select ISim as the Simulator (Figure 17)

2.1. Select the **FPGA part number** in the Hierarchy pane of the Project Navigator Design panel.

2.2. Right-click on the target **FPGA part number** and select **Design Properties**.

2.3. In the Design Properties dialog box, set the Simulator field to **ISim(VHDL/Verilog)** (with the appropriate type and language).

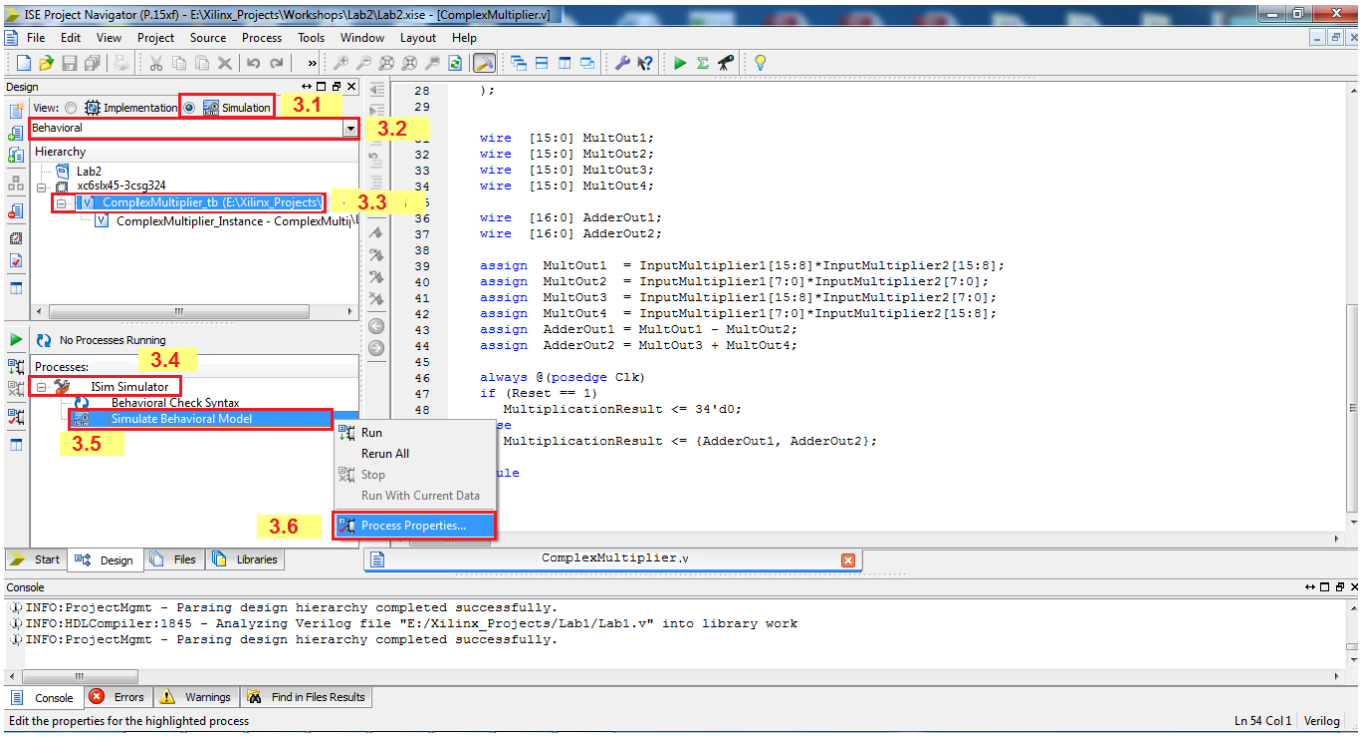

Figure 18.

3. Specifying Simulation Properties (Figure 18-19)

To locate the ISim simulator processes and set the simulation properties, do the following:

3.1. In the View pane of the Project Navigator Design panel, select **Simulation**.

3.2. Select **Behavioral** from the drop-down list.

3.3. In the Hierarchy pane, select the **test bench file** (**ComplexMultiplier _tb.v**).

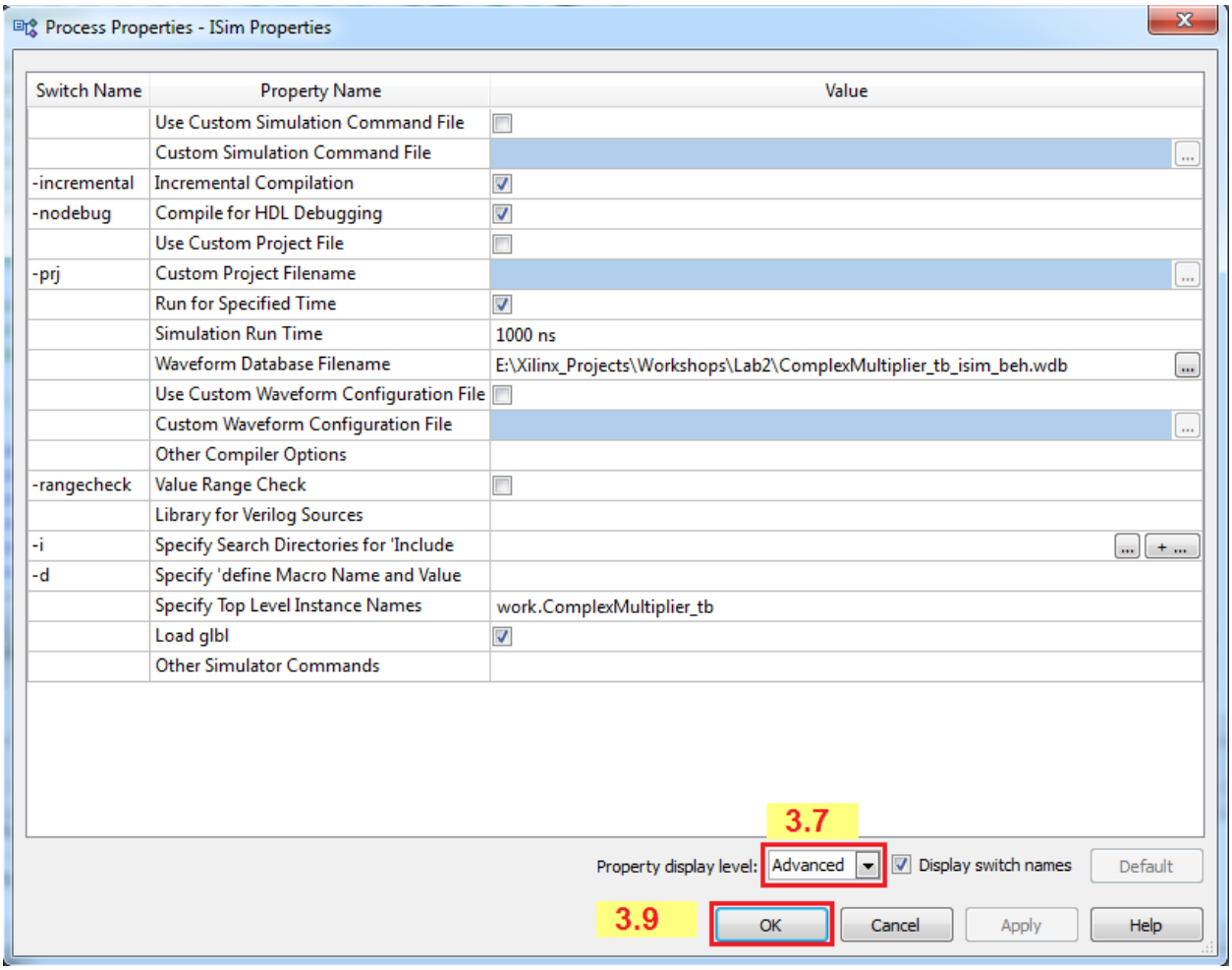

Figure 19.

3.4. In the Processes pane, expand **ISim Simulator** to view the process hierarchy.

3.5. Right-click **Simulate Behavioral Model.**

3.6. Select **Process Properties**.

3.7. In the Process Properties dialog box, set the Property display level to **Advanced**.

3.8. Change the simulation properties.

3.9. Click **OK**.

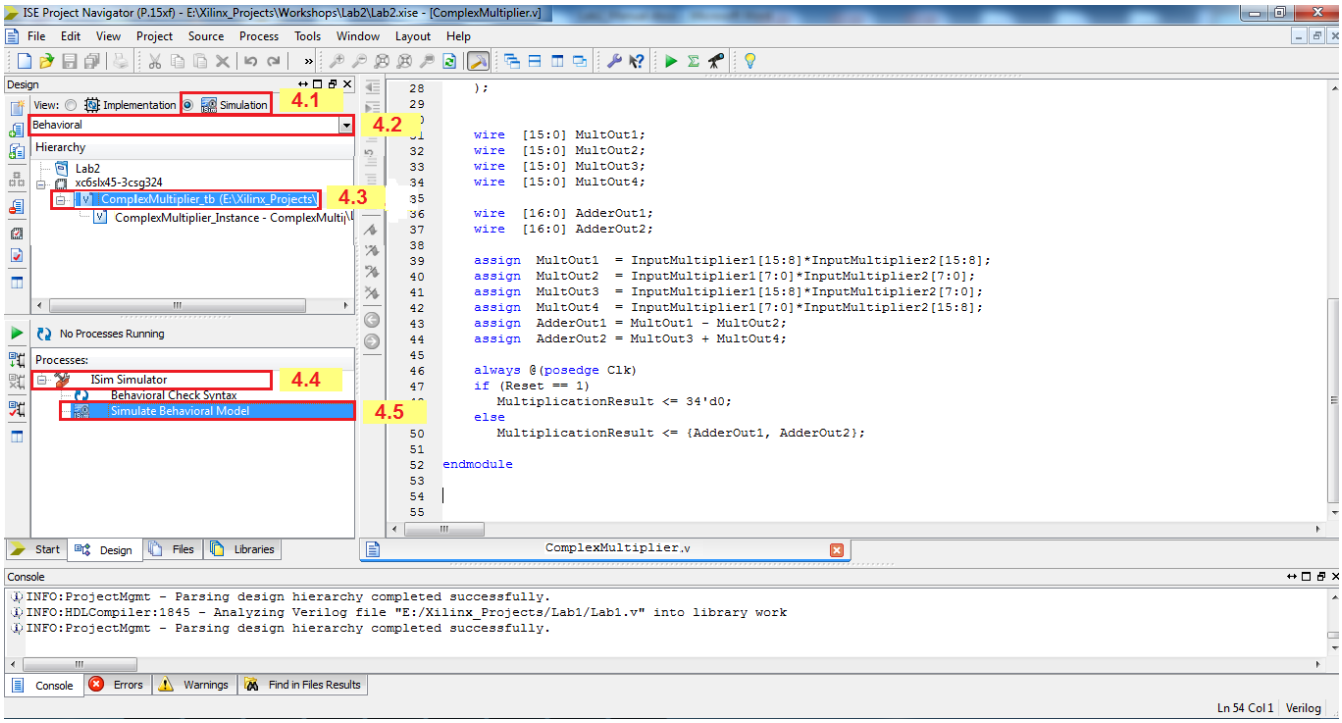

Figure 20.

4. Starting the Behavioral Simulation (Figure 20)

4.1. In the View pane of the Project Navigator Design panel, select **Simulation**.

4.2. Select **Behavioral** from the drop-down list.

4.3. In the Hierarchy pane, select the **test bench file** (**ComplexMultiplier _tb.v**).

4.4. In the Processes pane, expand **ISim Simulator** to view the process hierarchy.

4.5. Double- click **Simulate Behavioral Model**.

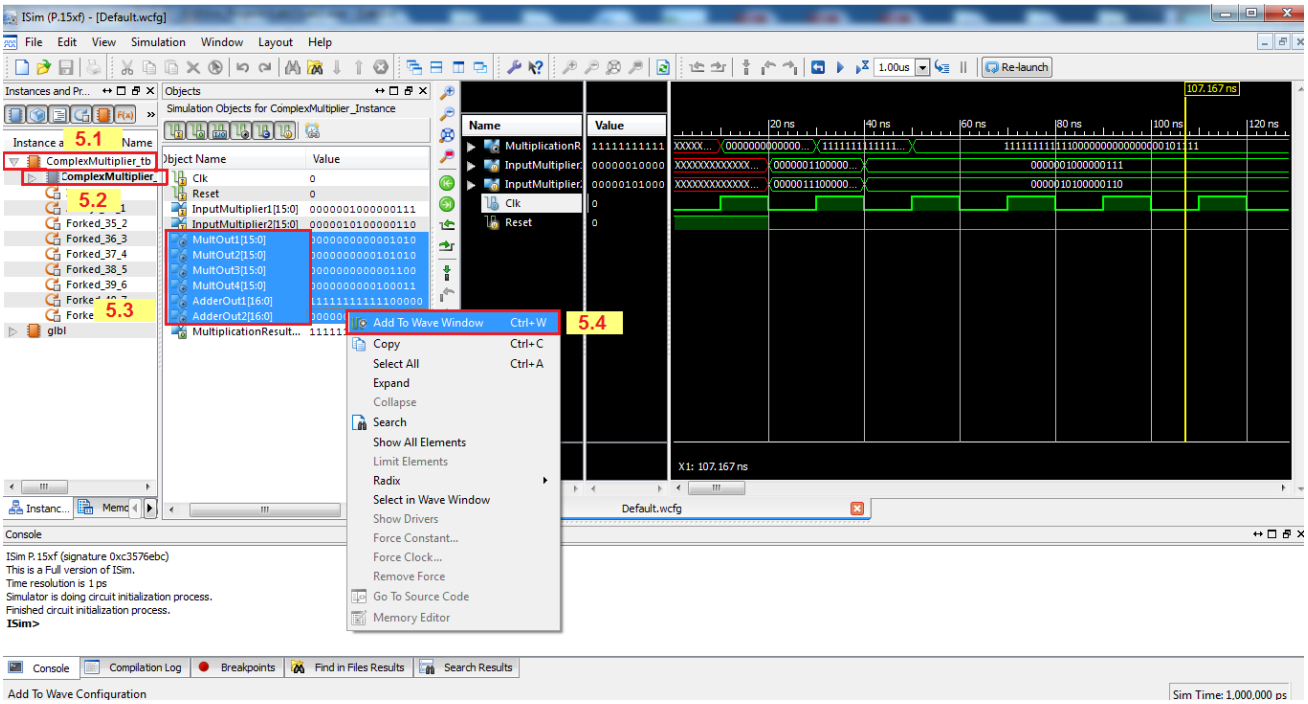

Figure 21.

5. Adding Signals to Wave Window (Figure 21)

The ISE Design Suite automatically adds all the top-level ports to the Wave window. But if you need to view and analyze the intermediate signals, perform the following steps:

5.1. In the **Instance and Process Name** window (i.e. in the left side of the main window in the ISim), expand the **ComplexMultiplier _tb**.

5.2. Select and expand the **ComplexMultiplier _Instance**. All the components within this design instance will be shown. Also, all of the corresponding signals will be shown in the **Objects** window.

5.3. Select the intermediate signals named **MultOut1, MultOut2, MultOut3, MultOut4, AdderOut1,** and **AdderOut2.**

5.4. Right-click on the selected signals and select **Add To Wave Window**.

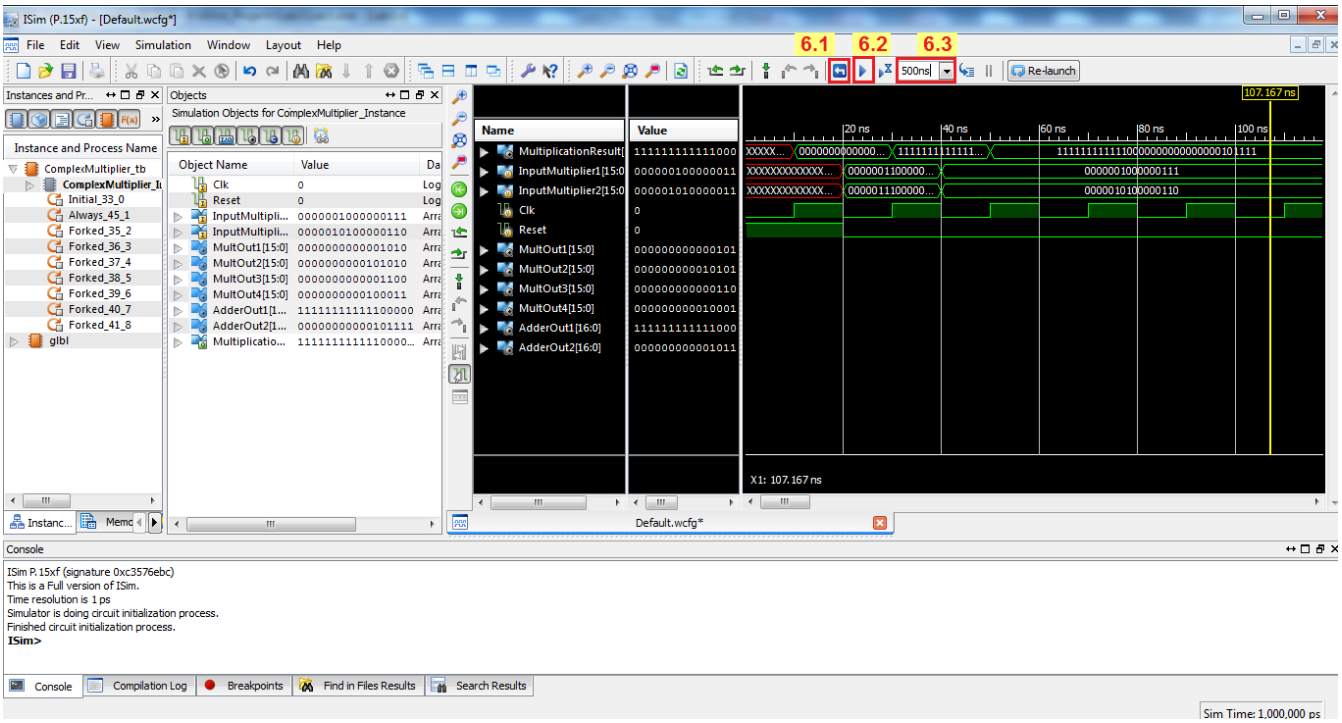

Figure 22.

6. Performing the Simulation (Figure 22)

In order to run the simulation in the ISim, do the following steps:

6.1. On the top of the main window, click on the **Restart Simulation Icon**.

6.2. In the Run Length box, set the simulation time to **500 ns**.

6.3. Click **Run** to start the simulation.

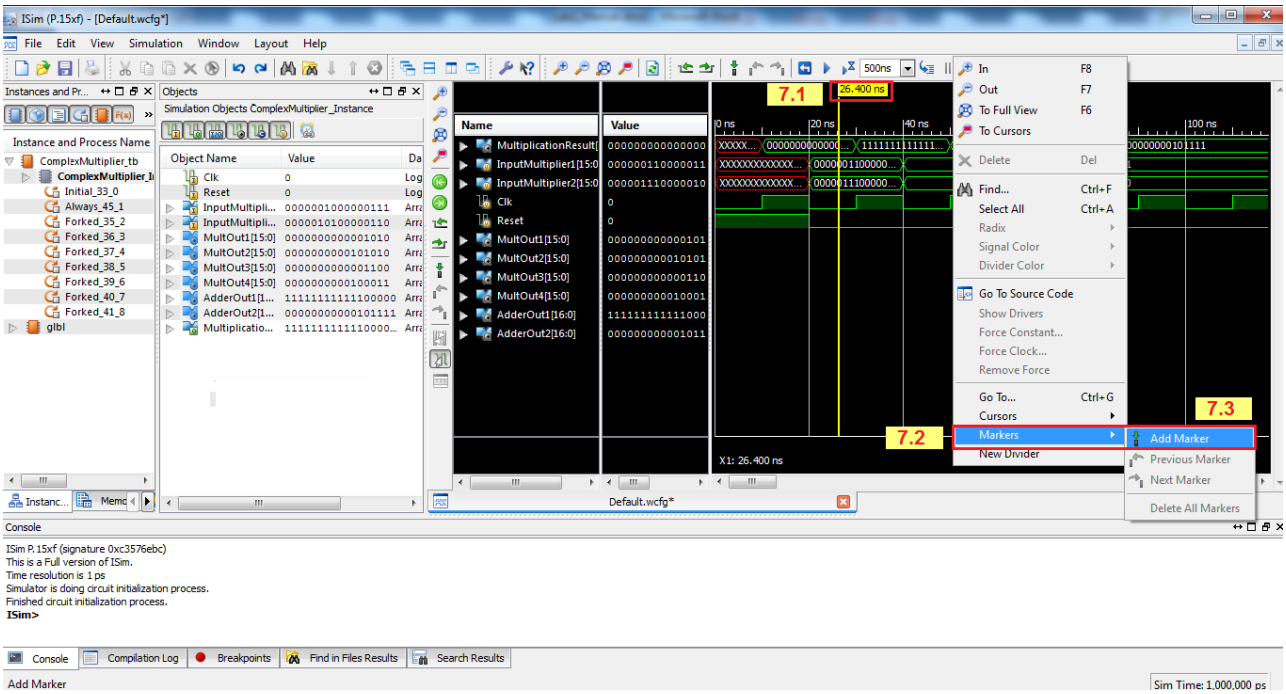

Figure 23.

7. Simulation Result Analysis (Figure 23)

7.1. Use **Cursor** to track the signals or find/compare different point in the simulation results.

7.2. Right-click on the waveform window.

7.3. Select **Markers > Add Marker** for further comparison or additional actions on the signals.

7.4. Walk through the signals using the cursors and check the results. You can verify the simulation results in this step using MATLAB. It is very easy to perform the complex multiplier in MATLAB and then compare the results with ISim outputs based on the same inputs.

NOTE:

You should perform this verification for each design to assure that it works well and its functionality is true. Almost MATLAB is useful in this step.

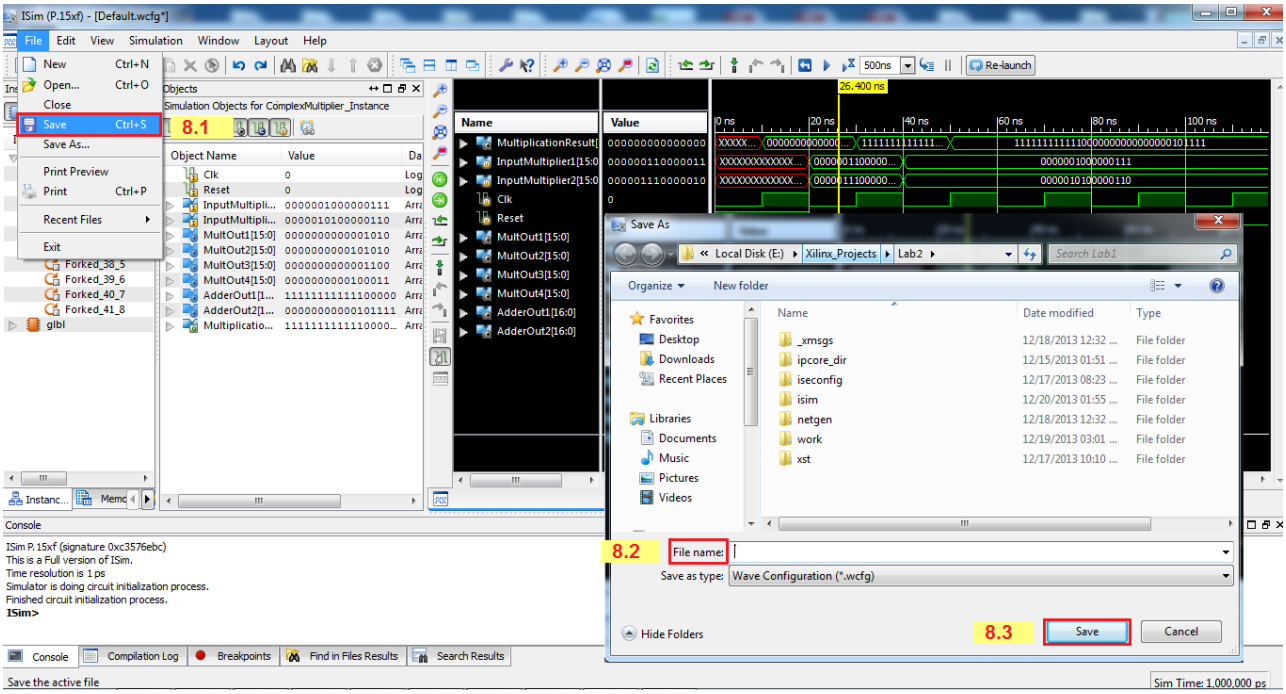

Figure 24.

8. Saving the Simulation (Figure 24)

You can save the signal list of the Wave window. The saved signals list can easily be opened each time the simulation is started.

To save the signals list, do the following:

8.1. Select **File > Save**.

8.2. In the Save As dialog box, enter the desire **name and location**.

8.3. Click **OK**.

8.4. After restarting the simulation, select **File > Open** to load this file.

Digital Circuit Design with Xilinx FPGA

Lab Manual

Workshop No.1

Lab3:

Getting Started with Xilinx CORE Generator

Outline

This manual covers the following topics:

- Design creation using ISE Project Navigator
- Instantiating a pre-designed module
- Working with Xilinx CORE Generator to add/customize IP COREs to the design
- Instantiating CORDIC and Multiply Adder cores to the design
- Synthesizing the design using XST, which includes IP COREs
- View the RTL/Technology Schematic
- Simulating the design using ModelSim

Introduction

This lab describes the method of using Xilinx CORE Generator Tool, which is used for parameterizing cores, optimized for Xilinx FPGAs. The Xilinx CORE Generator System is an easy to use design tool that delivers parameterizable cores developed for the Xilinx FPGAs. The Xilinx CORE Generator System provides you with a catalog of ready-made functions ranging in complexity from simple arithmetic operators such as adders, accumulators and multipliers, to system level building blocks including filters, transforms and memories and also complicated blocks such as Viterbi Decoder and Turbo Decoder.

Cores are organized by functional type into folders that expand or contract on demand. Detailed information on each core is included in a specification or data sheet (PDF format), which is easily accessed by clicking on the Datasheet button in the core customization window or by clicking on the Datasheet icon in the main CORE Generator application toolbar. This launches the Adobe Acrobat Reader and calls up the datasheet for the selected core.

The CORE Generator System can customize a generic functional building block such as a FIR filter or a multiplier to meet the needs of your application and simultaneously deliver high levels of performance and area efficiency.

Required Software

To perform this manual, you must have **Xilinx ISE Design Suite 14.1** and **ModelSim 10.1** (or above) installed. Note that in this manual the pictures are captured in **Xilinx ISE Design Suite 14.1**, but all the procedures can be done in other versions of this software.

Objectives

After completing this lab, you will be able to:

- Instantiate and add a pre-designed module to your project
- Working with Xilinx CORE Generator
- Add/customize the desired IP core to your design

General Flow of this Lab

This lab comprises three steps as follows:

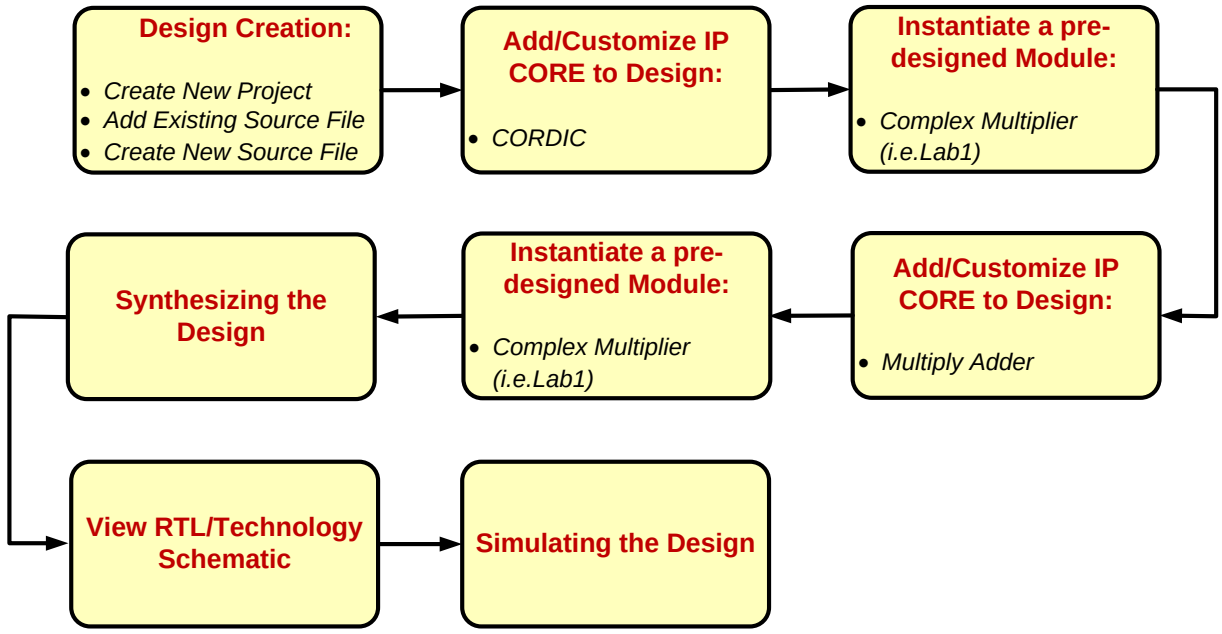

Figure 1. General flow of Lab3.

Design Description

In this lab, you'll design a complicated module, which is shown in Figure 2. You'll use a few instances of the complex multiplier, which was designed in Lab1. Also, you'll write Verilog code to realize some simple logics (i.e. control logics). Finally, you'll learn the method of adding some IP cores to the design using the Xilinx CORE Generator, which is the main goal of this lab. The target design of this lab is a computational module, which includes a state machine with four states. The functionality of each state is described in the following section.

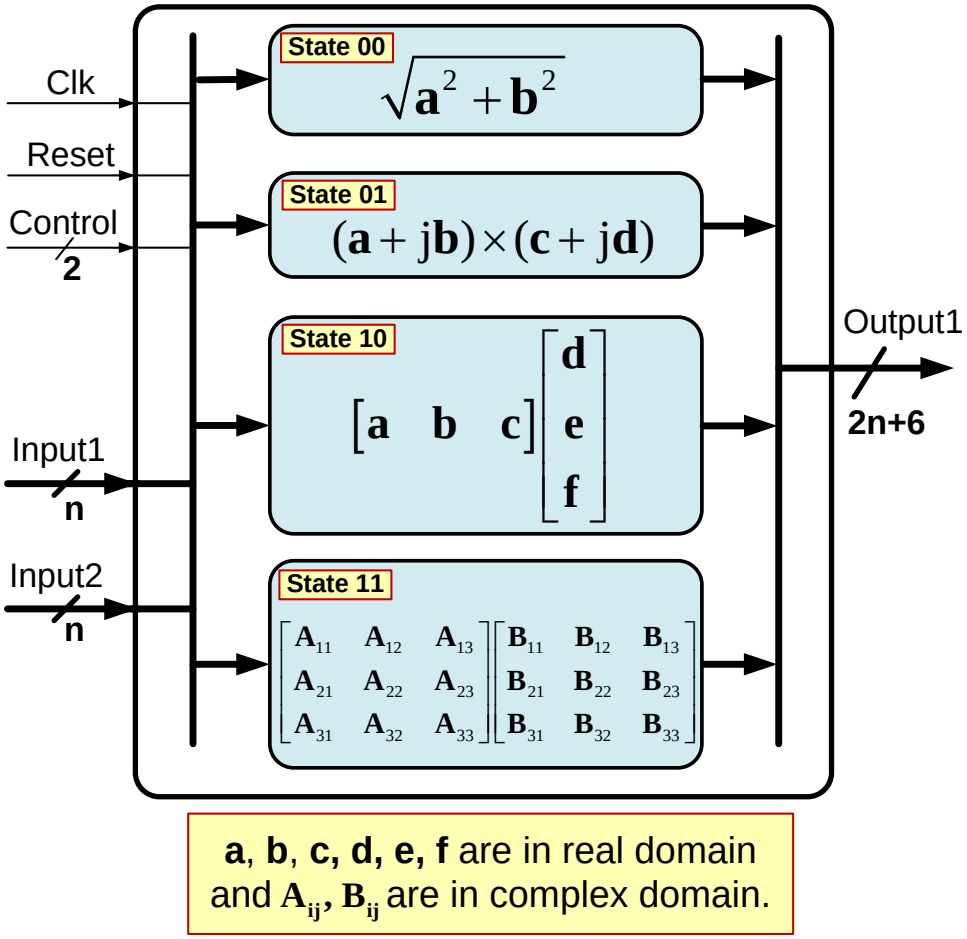

Figure2. The block diagram of the target design in Lab3.

NOTE:

- *The input values of state0 have signed-integer real values and its output has unsigned-integer real.*
- *The input/output values of state1 have signed-integer complex values.*
- *The input/output values of state2 have signed-integer real values.*
- *The input/output values of state3 have signed-integer complex values.*
- The format of the complex values of inputs/outputs is as follow:

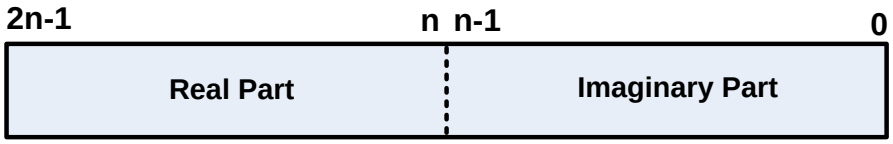

Figure 3. A sample complex input/output with the length of **2n** bits.

State 0:

In this step you should design a sub module to realize the following equation:

$$\sqrt{\mathbf{a}^2 + \mathbf{b}^2}$$

The detailed architecture of this module is shown in Figure4. First, the power two of both inputs should be calculated using simple multipliers. Then, the square root of the summation of these values will be calculated using a Xilinx IP core, CORDIC. A brief description about the CORDIC core is mentioned below:

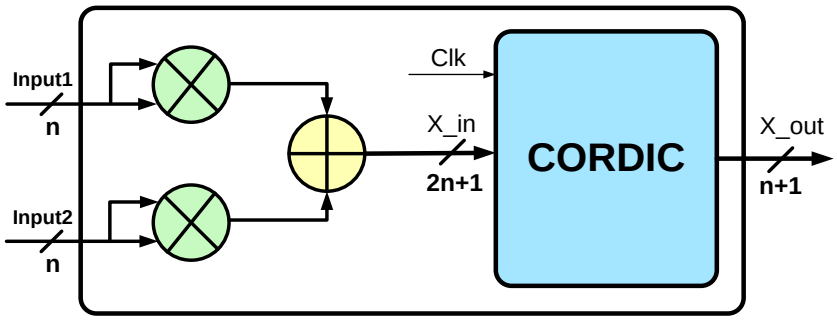

Figure4. The detailed block diagram of State 0.

LogiCORE IP CORDIC

The Xilinx LogiCORE IP CORDIC implements a generalized coordinate rotational digital computer (CORDIC) algorithm, initially developed by Volder [1] to iteratively solve trigonometric equations, and later generalized by Walther [2] to solve a broader range of equations, including the hyperbolic and square root equations. CORDIC uses simple shift-add operations for the following computing tasks:

- Vector rotation (polar to rectangular)
- Vector translation (rectangular to polar)
- Sin and Cos
- Sinh and Cosh
- Atan and Atanh
- Square root

As a consequence, CORDIC has been utilized for applications in diverse areas such as signal and image processing, communication systems, robotics and 3-D graphics apart from general scientific and technical computation.

When the square root functional configuration is selected a simplified CORDIC algorithm is used to calculate the positive square root of the input. The input, X_in, and the output, X_out, are always positive and are both expressed as either unsigned fractions or unsigned integers:

- When data format is set to Unsigned Fraction, X_IN is limited to the range: 0 <= X_IN < +2.
- When data format is set to Unsigned Integer, X_IN is limited to the range: 0 <= X_IN < 2**Input Width, and the output width is determined automatically based on the input width.

In this design, **a** and **b** have signed-integer real values. Therefore, the Unsigned Integer configuration is chosen for the CORDIC. Consequently, the input/output values of the CORDIC core are unsigned integers.

Note:

Because of the pipeline stages and internal latency of the sub blocks the **Control** *signal should be unchanged for 5 clock cycle after the last input data arrival. Please see the simulation waveform in the last section of this lab to understand this fact.*

State 1:

In this state you should use the target design of Lab1, the complex multiplier, which is shown in Figure 5. The goal of this state is to realize the following equation:

$$(a + jb) \times (c + jd)$$

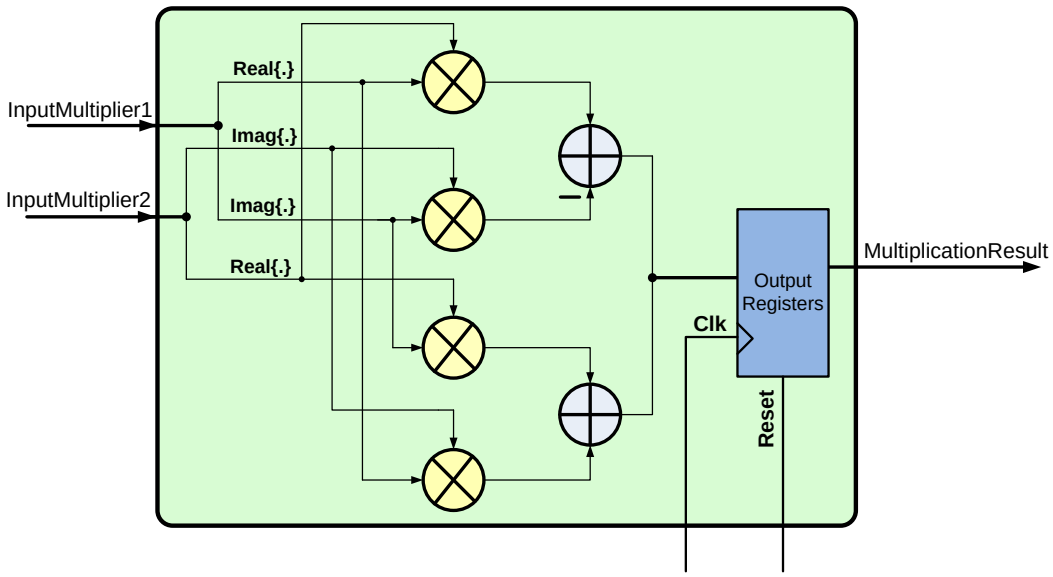

Figure 5. The detailed architecture of State 1.

Note:

Because of the pipeline stages and internal latency of the sub blocks the **Control** *signal should be unchanged for 3 clock cycle after the last input data arrival. Please see the simulation waveform in the last section of this lab to understand this fact.*

State 2:

In this state you should design a sub module to perform the vector multiplication in the real domain:

$$\begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} d \\ e \\ f \end{bmatrix}$$

All of the entries of these vectors have signed-integer real values. In order to implement this operation, you'll use three instances of **Multiply Adder** IP core. Moreover, you should design a simple control logic to manage timing of input data for these cores. Because the data should arrive at the inputs of the three cores by considering the internal latency of each stage. We mange it using a simple state machine, which is mentioned in the Verilog source file (i.e. DUT.v). The detailed block diagram of State 2 is shown in Figure 6. A brief description about the **Multiply Adder** IP core is described below.

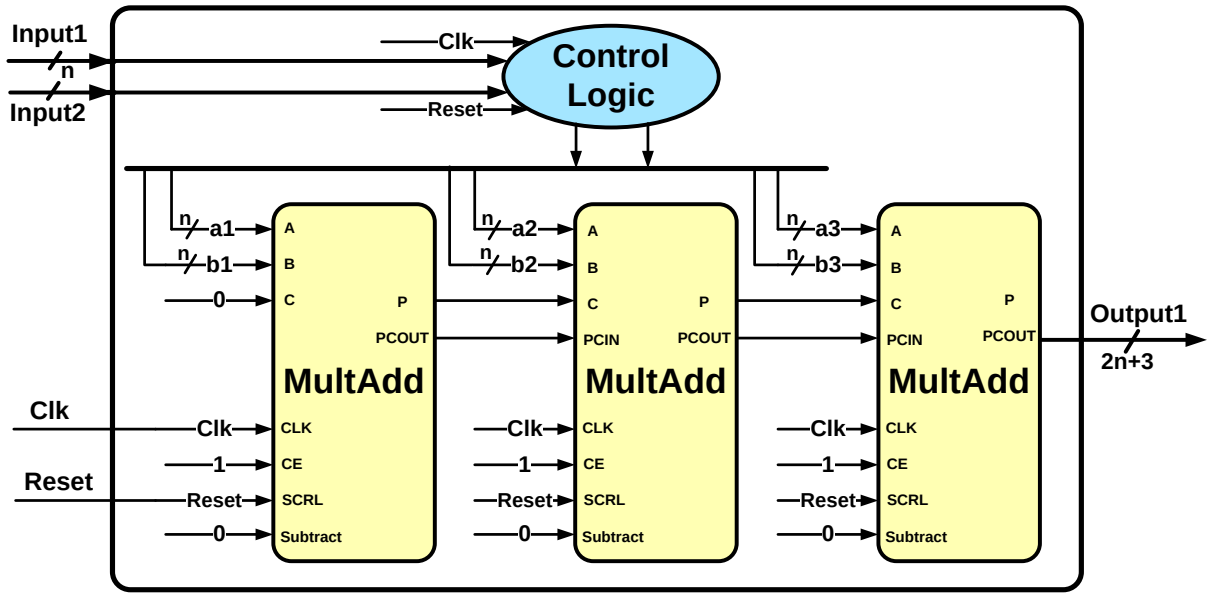

Figure 6. The detailed architecture of State 2.

Multiply Adder

The Xilinx LogiCORE IP Multiply Adder core provides implementations of multiply-add using XtremeDSP slices. It performs a multiplication of two operands and adds (or subtracts) the full-precision product to a third operand. The Multiply Adder module operates on signed or unsigned data. The Multiply Adder module can be pipelined. It generates mult-add and mult-subtract functions and supports twos complement-signed and unsigned operations.

Note:

Because of the pipeline stages and internal latency of the sub blocks the **Control** *signal should be unchanged for 5 clock cycle after the last input data arrival. Please see the simulation waveform in the last section of this lab to understand this fact.*

State 3:

In this state you'll design a sub module to perform the matrix multiplication with complex-signed-integer values as follows:

$$\begin{bmatrix} \mathbf{a}_{11}+\mathrm{j}\mathbf{b}_{11} & \mathbf{a}_{12}+\mathrm{j}\mathbf{b}_{12} & \mathbf{a}_{13}+\mathrm{j}\mathbf{b}_{13} \\ \mathbf{a}_{21}+\mathrm{j}\mathbf{b}_{21} & \mathbf{a}_{22}+\mathrm{j}\mathbf{b}_{22} & \mathbf{a}_{23}+\mathrm{j}\mathbf{b}_{23} \\ \mathbf{a}_{31}+\mathrm{j}\mathbf{b}_{31} & \mathbf{a}_{32}+\mathrm{j}\mathbf{b}_{32} & \mathbf{a}_{33}+\mathrm{j}\mathbf{b}_{33} \end{bmatrix} \begin{bmatrix} \mathbf{c}_{11}+\mathrm{j}\mathbf{d}_{11} & \mathbf{c}_{12}+\mathrm{j}\mathbf{d}_{12} & \mathbf{c}_{13}+\mathrm{j}\mathbf{d}_{13} \\ \mathbf{c}_{21}+\mathrm{j}\mathbf{d}_{21} & \mathbf{c}_{22}+\mathrm{j}\mathbf{d}_{22} & \mathbf{c}_{23}+\mathrm{j}\mathbf{d}_{23} \\ \mathbf{c}_{31}+\mathrm{j}\mathbf{d}_{31} & \mathbf{c}_{32}+\mathrm{j}\mathbf{d}_{32} & \mathbf{c}_{33}+\mathrm{j}\mathbf{d}_{33} \end{bmatrix}$$

In order to implement this operation, you need to use three instances of the complex multiplier, which was designed in Lab1. Each entry of both matrixes enters to this module via Input1 and Input2 ports per clock and all of them will be saved in the corresponding memory (i.e. Mem1_temp and Mem2_temp). After nine clock cycles all of the matrix entries are stored in the memories and the multiplication process will start. After the initial latency, each entry of the multiplication result will appear in the Output port ring nine clock cycles. The detailed architecture of State 3 is shown in Figure 7.

Note:

Because of the pipeline stages and internal latency of the sub blocks the **Control** *signal should be unchanged for 11 clock cycle after the last input data arrival. Please see the simulation waveform in the last section of this lab to understand this fact.*

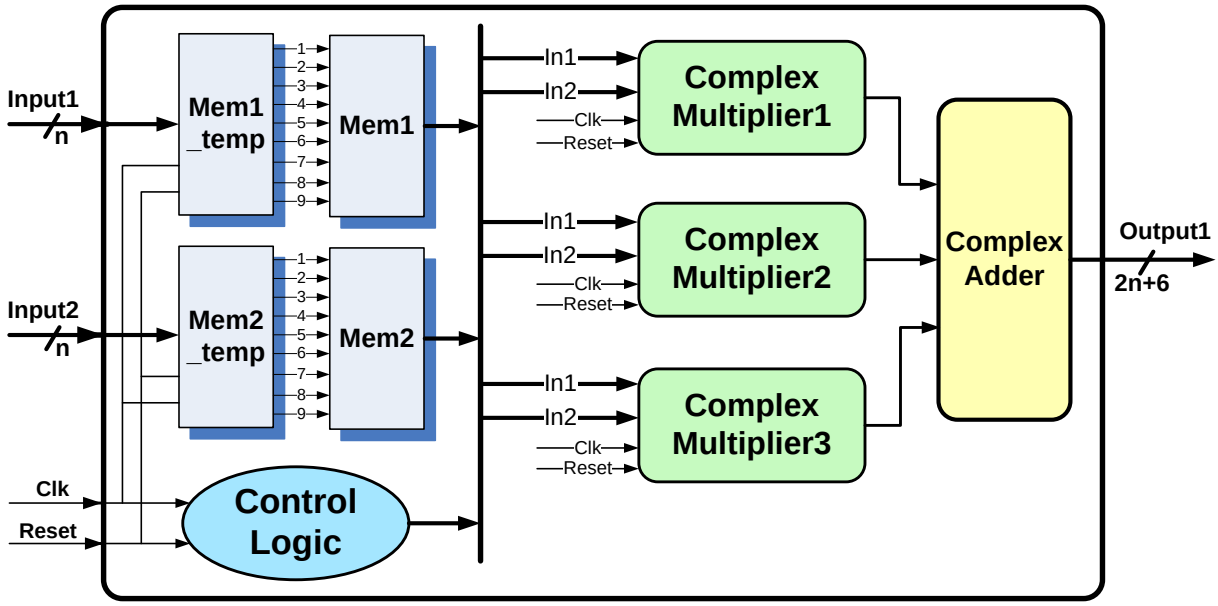

Figure 7. The detailed block diagram of State 3.

Design Creation, Instantiation of Lab1, and Adding IP Cores

1. Start the Project Navigator

Double click the Project Navigator icon on your desktop, or select **Start > All Programs > Xilinx Design Tools > ISE Design Suite 14.1 > ISE Design Tools > Project Navigator.**

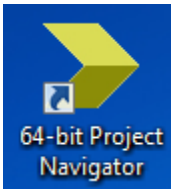

Figure 8. Project Navigator desktop icon.

2. Creating a New Project (Figure 9-10)

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

2.2. In the Location field, browse to the directory in which you want to save the project.

2.3. In the Name field, enter **Lab3**.

2.4. Verify that **HDL** is selected as the Top-Level Source Type

2.5. Click **Next**. The New Project Settings page appears.

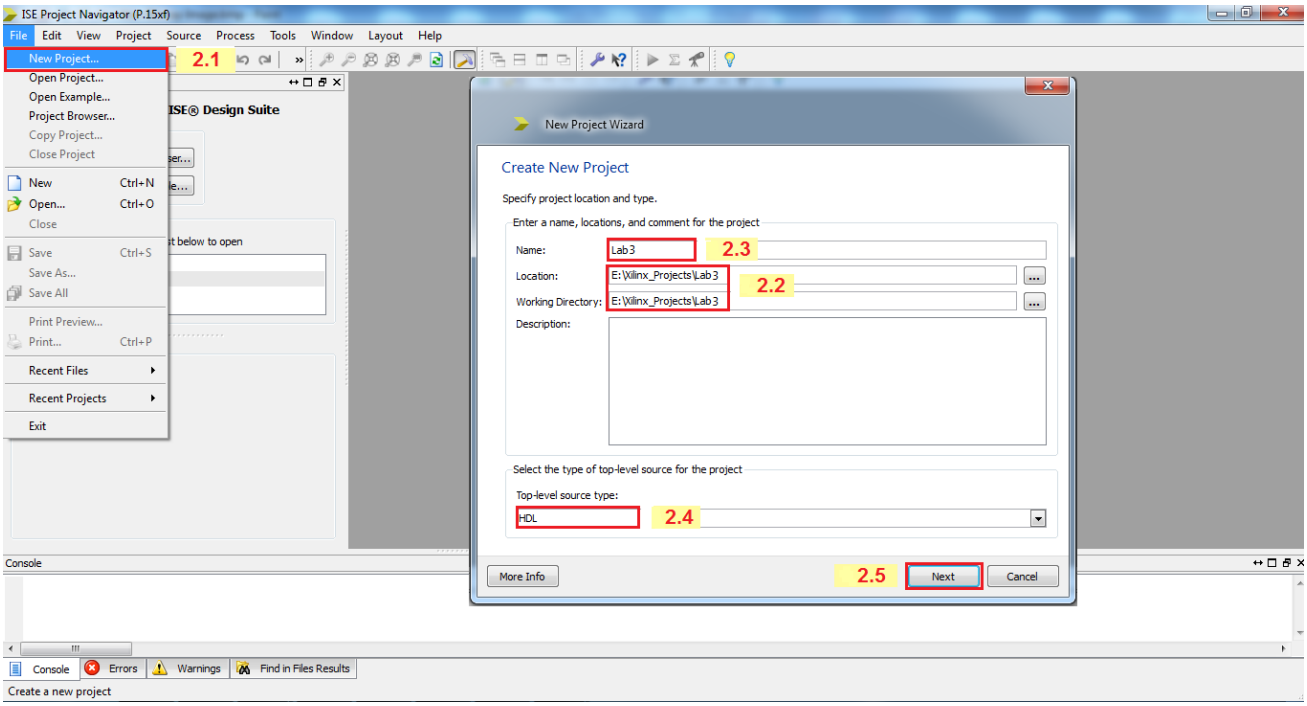

Figure 9.

2.6. Select the following values in the Project Settings page:

- Product Category: **All**
- Family: **Spartan6**
- Device: **XC6SLX45**
- Package: **CSG324**
- Speed: **-3**
- Synthesis Tool: **XST (VHDL/Verilog)**
- Simulator: **ModelSim-SE Verilog**

Note: If you have not ModelSim installed on your system, select ISim(VHDL/Verilog) as your simulator.

- 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.

2.7. Click **Next**, then **Finish** to complete the project creation.

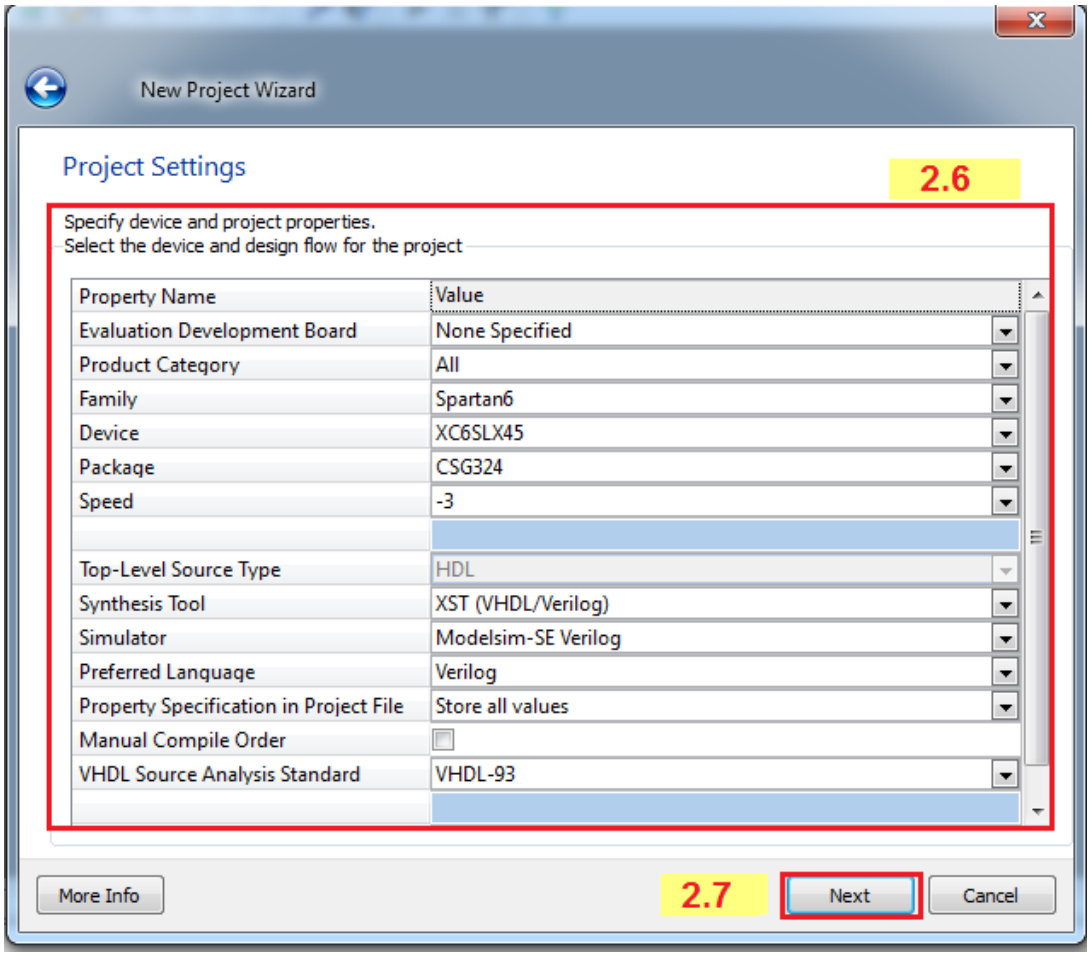

Figure 10.

3. Adding Existing Source Files (Figure 11)

You can add any existing source file to your project from the previous projects. Another method was described in Lab1. In the following method a copy of the desired source will be added to the current project directory automatically and then it will be added to the current project. You can do this via the following steps:

3.1. Select **Project > Add Copy of Source**.

3.2. Go to the project directory of **Lab1**.

3.3. Select **ComplexMultiplier.v** from the project directory.

3.4. Click **Open**.

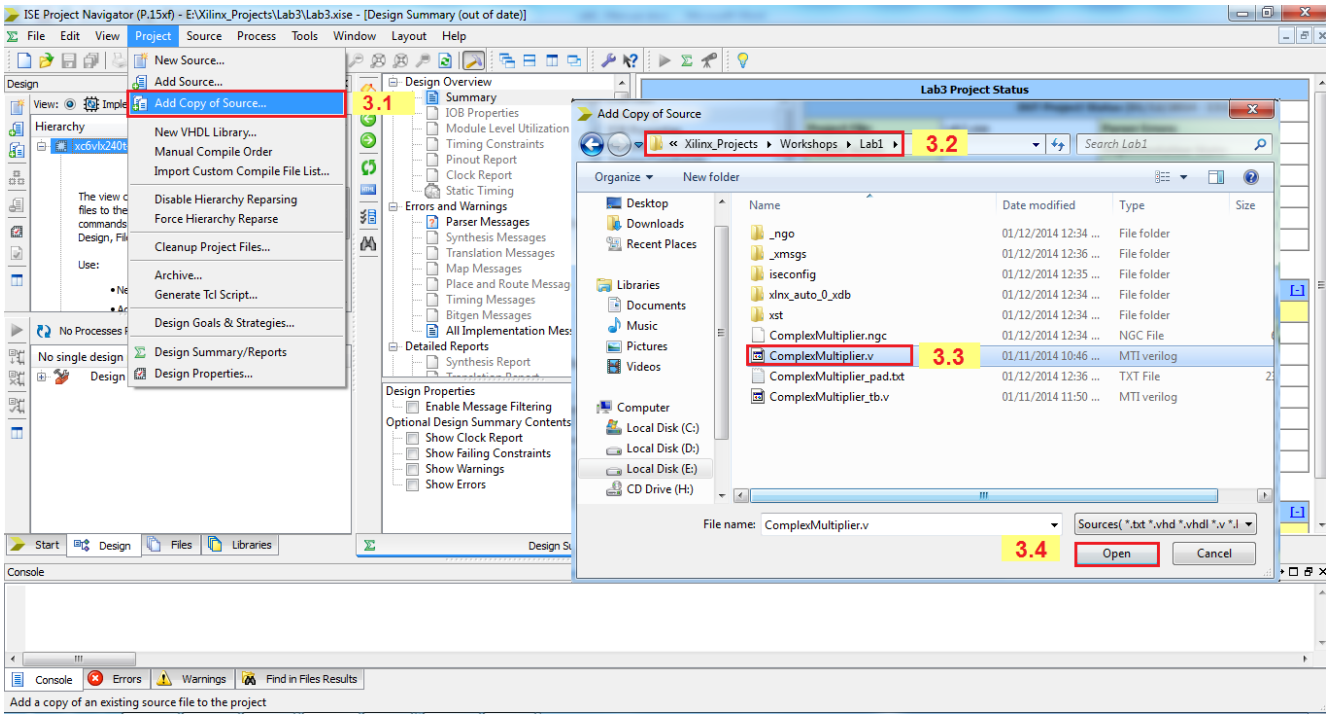

Figure 11.

4. Creating New Source File (Figure 12-13)

In this section, you create a Verilog source file using the New Source wizard. The resulting HDL file is then modified in the ISE Text Editor. In order to create the new source file, do the following:

4.1. Select **Project > New Source**, or right-click on the Hierarchy Pane and then select **New Source**. The New Source Wizard opens in which you specify the type of source you want to create.

4.2. In the Select Source Type page, select **Verilog Module**.

4.3. Verify the location field is set to the current project directory.

4.4. In the File Name field, enter **DUT**.

4.5. Verify that the Add to project box is checked.

4.6. Click **Next**.

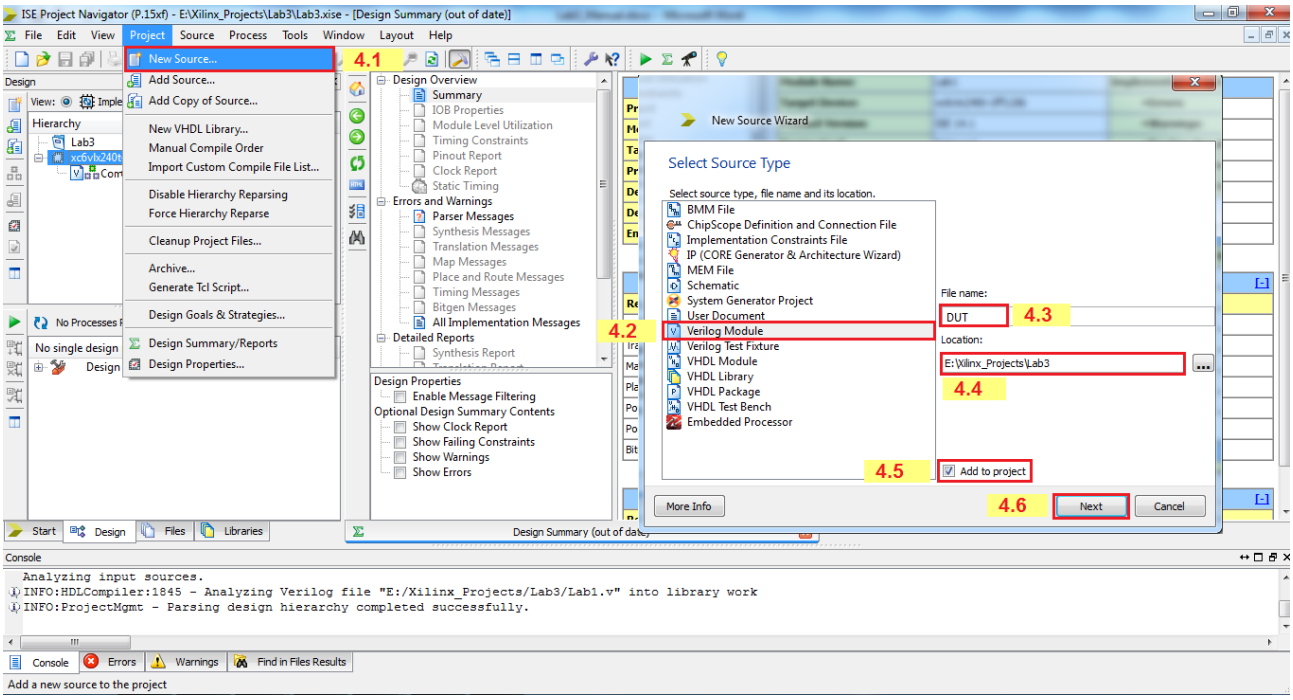

Figure 12.

4.7. In the Define Module page, enter five input ports named **Clk**, **Reset, Control, Input1,** and **Input2** in the Port Name fields.

4.8. Set the Direction field to **output** for the last port and set it to **input** for the other ports**.**

4.9. Leave the Bus designation boxes unchecked for the first two ports and check it for the other ports.

4.10. Enter the values shown in Figure 13 for bus width of last four ports.

4.11. Click **Next** to view a description of the module.

4.12. Click **Finish** to open the empty HDL file in the ISE Text Editor.

4.13. Complete the Verilog code according to the above architecture to realize the complex multiplier.

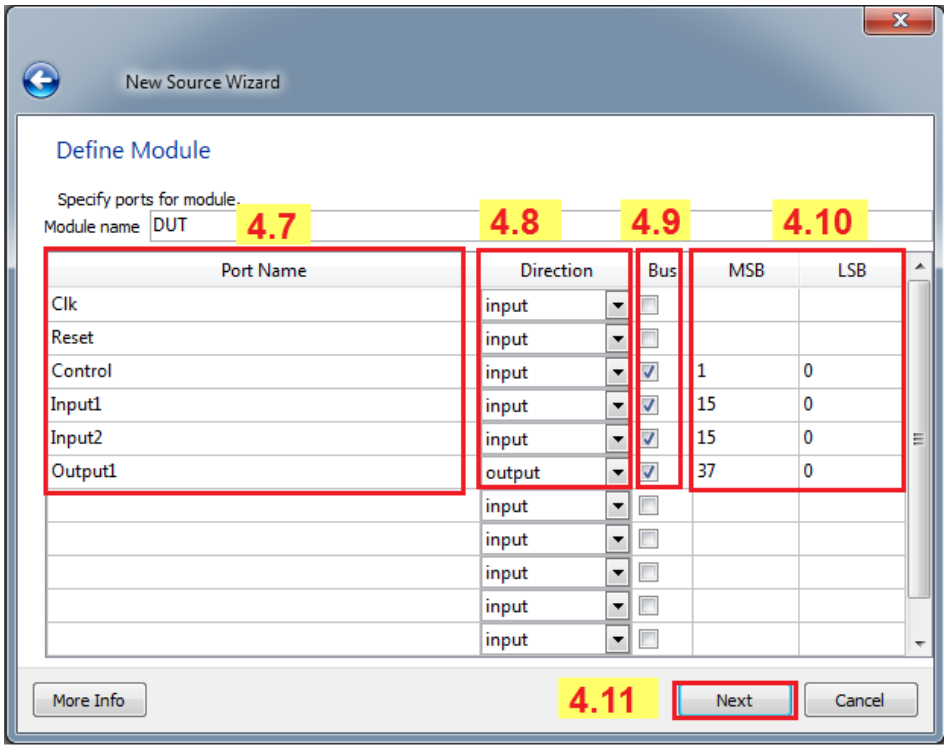

Figure 13.

5. Specify/Change the Top Module of the Design(Figure 14)

In this lab the top module is **DUT.v**, which uses **ComplexMultiplier.v** as a sub module. Project Navigator usually sets the first added source file to the top module and may be that is not the real top module. Thus, you should always verify the top module of your project when you apply any modification on your source files such as adding/removing a source file. Set **DUT.v** to the top module of your project to continue this lab.

In order to specify the top module of your design do following:

5.1. In the Hierarchy pane of the Project Navigator Design panel, select **DUT.v**.

5.2. Select **Set as Top Module**.

5.3. A new window will appear and ask you about this change. Click **OK**.

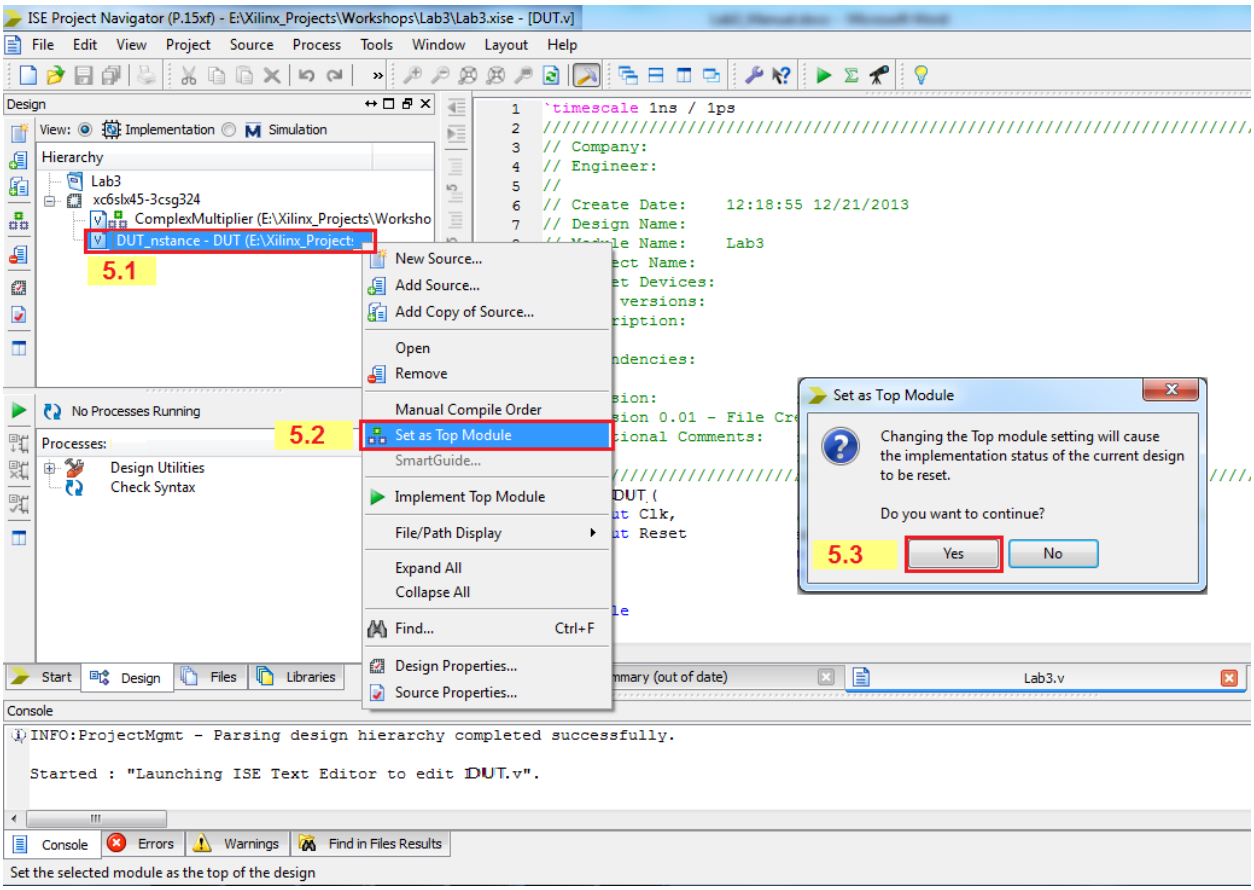

Figure 14.

6. Adding CORDIC IP to the Design-Implementation of State0 (Figure 15-20)

In this section you will learn the method of adding an IP core, CORDIC, to your design. This process is shown step by step as follow:

6.1. Right-click on the top module name and select **New Source**.

6.2. In the source types, select **IP (CORE Generator & Architecture Wizard)**.

6.3. Set the location to the project directory of Lab3 and enter the name (i.e. **CORDIC**) in the file name field.

6.4. Verify that Add to project box is checked.

6.5. Click **Next**.

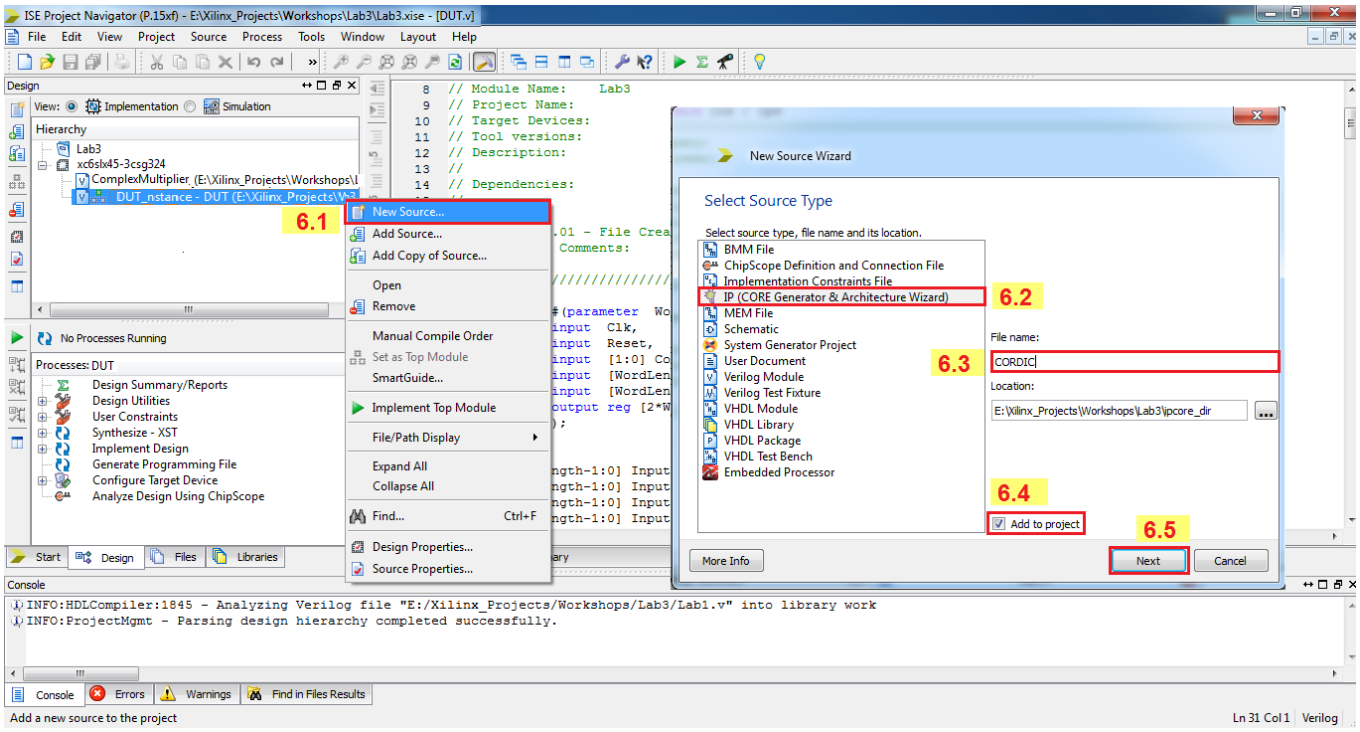

Figure 15.

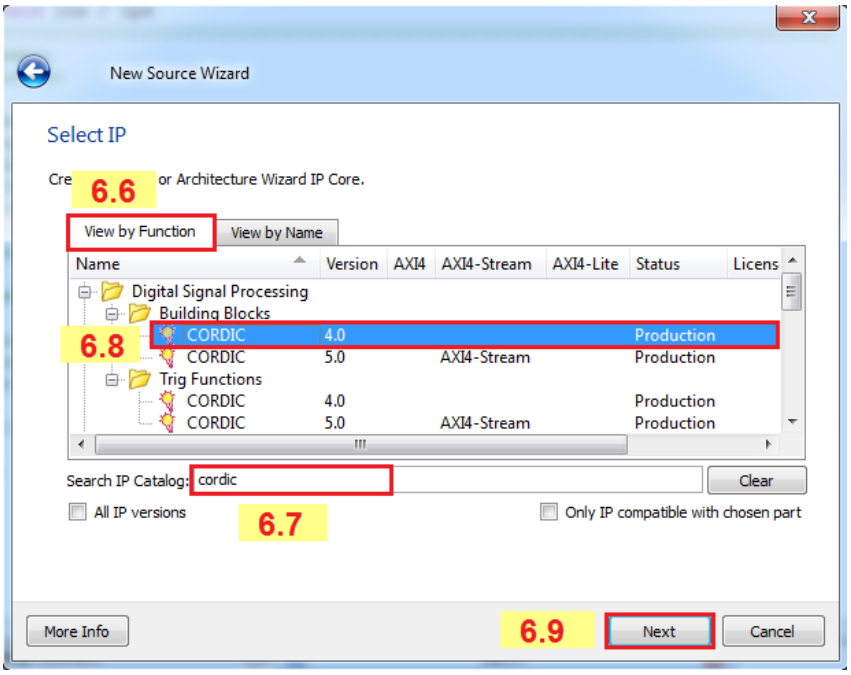

Figure 16.

6.6. In the New Source Wizard, select **View by Function** tab.

6.7. Type the key word, **CORDIC**, in the Search IP Catalog field to filter the IPs and only view the corresponding IPs.

6.8. Select the **CORDIC Version 4.0**.

6.9. Click **Next**.

6.10. In the next page, click **Finish**.

6.11. In the Functional Selection, select **Square Root** function.

6.12. In the pipelining Mode, select **No Pipelining**.

6.13. Click **Next**.

Figure 17.

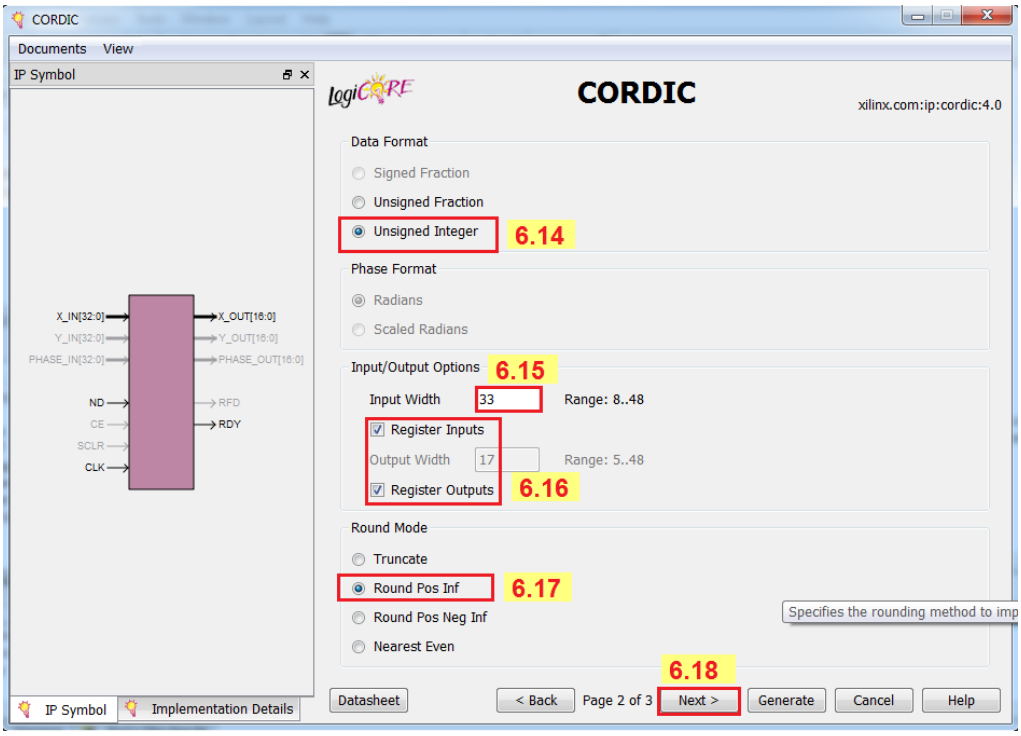

Figure 18.

Figure 19.

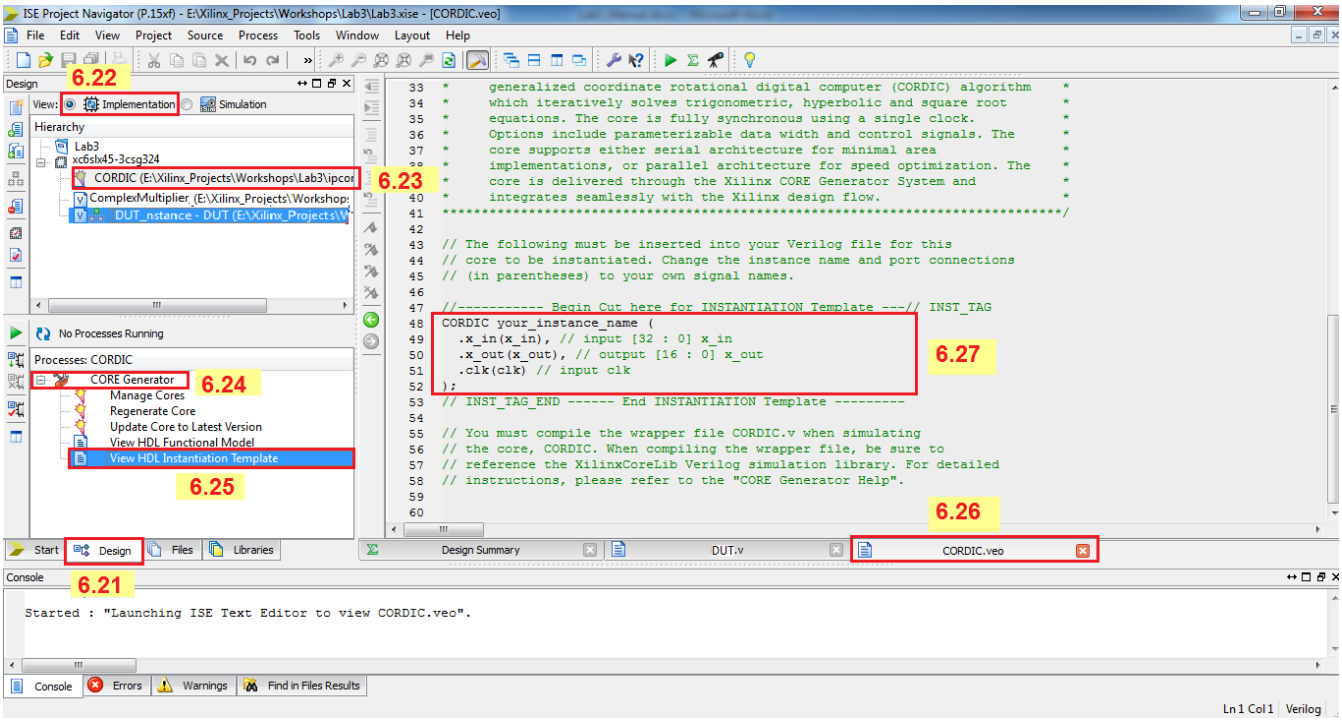

Figure 20.

6.14. In the Data Format part, select **Unsigned Integer**.

6.15. Set input Width to **33**.

6.16. Leave the Register Inputs/Outputs boxes checked.

6.17. Select **Round Pos Inf** as the Rounding Mode.

6.18. Click **Next**.

6.19. Uncheck the ND and RDY boxes.

6.20. Click **Generate** to generate your customized IP core.

6.21. Select **Design Panel**.

6.22. Set the Design View to the **Implementation**.

6.23. In the Hierarchy pane, select the **CORDIC IP source**.

6.24. In the Processes Pane, expand **GORE Generator**.

6.25. Double-click on **View HDL Instantiation Template**.

6.26. CORDIC.veo file, which includes the instantiation template of the CORDIC IP will be opened.

6.27. Copy and paste the uncommented part of instantiation template file to the proper place in the top module file (i.e. **DUT.v**).

6.28. Complete the Verilog code of **State 0** in the top module (i.e. **DUT.v**) based on the proposed architecture shown in Figure 4.

6.29. Choose the desired name for the CORDIC instance.

6.30. In the top module (i.e. **DUT.v**), define the required signals and connect the proper signals to the input/output ports of CORDIC instance.

State0 of your state machine is complete now!

Go ahead and realize the other parts of project.

7. Adding an Instance of Lab1 -Implementation of State1 (Figure 21)

In this part of Lab3, you should add an Instance of Lab1 project (i.e. the complex multiplier) to realize the State1 of the final project of Lab3. Do the following steps:

7.1. Select **Design Panel**.

7.2. Set the Design View to the **Implementation**.

7.3. In the Hierarchy pane, select the **ComplexMultiplier.v** source.

7.4. In the Processes Pane, expand **Design Utilities**.

7.5. Double-click on **View HDL Instantiation Template**.

7.6. ComplexMultiplier.tfi file, which includes the instantiation template of the Lab1 project will be opened.

7.7. Copy and paste the instantiation template file to the proper place in the top module file (i.e. **DUT.v**).

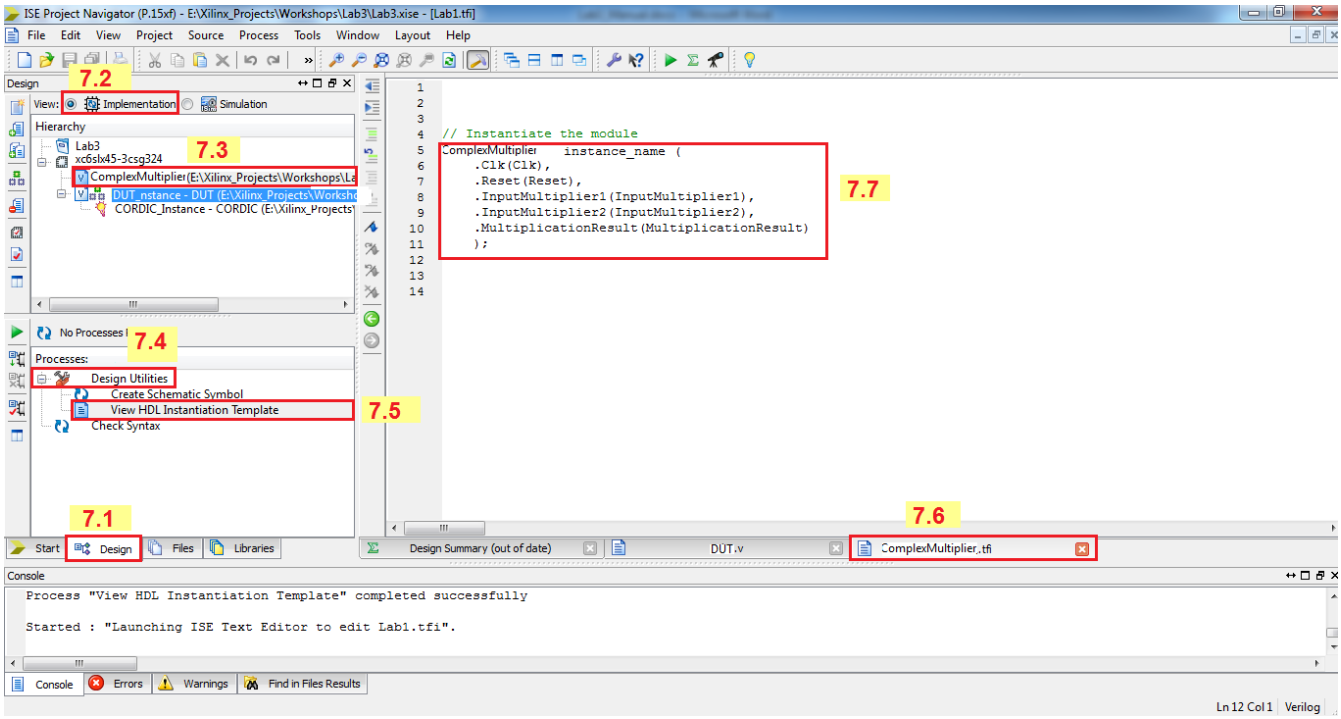

Figure 21.

7.8. Choose the desired name for the Lab1 instance.

7.9. Complete the Verilog code of **State 1** in the top module (i.e. **DUT.v**) and define the required signals and connect the proper signals to the input/output ports of the complex multiplier instance.

State1 of your state machine is complete now! Go ahead and realize the other parts of project.

8. Implementation of Real-valued vector multiplication-State2 (Figure 22-25)

In this part you will design a sub module to realize **State2** of the final project of Lab3, which performs the vector multiplication (with the size of 3x1 and 1x3) in the real domain. The detailed architecture of this sub module is shown in Figure 6. In order to implement this part of the project three samples of MultAdd IP core will be added to your project as. Do the following steps to add the MultAdd IP cores and complete the State1.

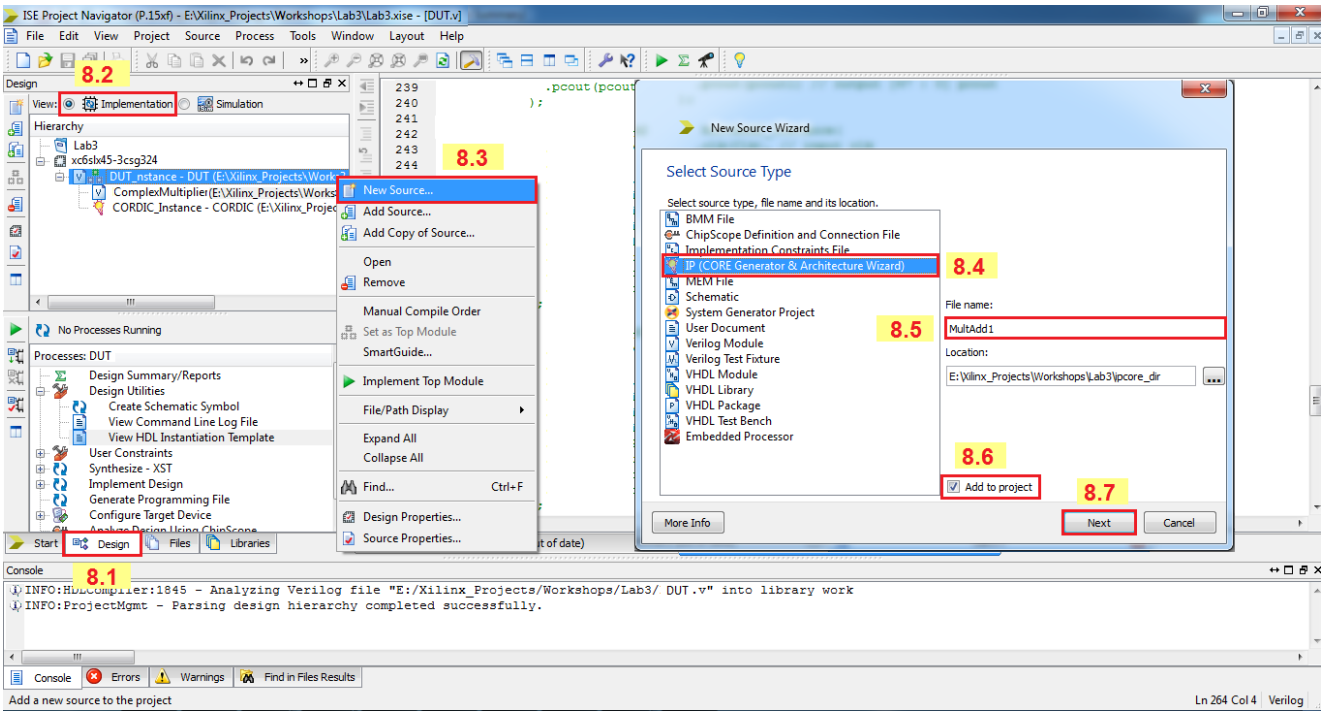

Figure 22.

8.1. Select **Design Panel**.

8.2. Set the Design View to the **Implementation**.

8.3. In the Hierarchy pane, Right-click on the top module name and select **New Source**.

8.4. In the source types, select **IP (CORE Generator & Architecture Wizard)**.

8.5. Set the location to the project directory of Lab3 and enter the name (i.e. **MultAdd1**) in the file name field.

8.6. Verify that Add to project box is checked.

8.7. Click **Next**

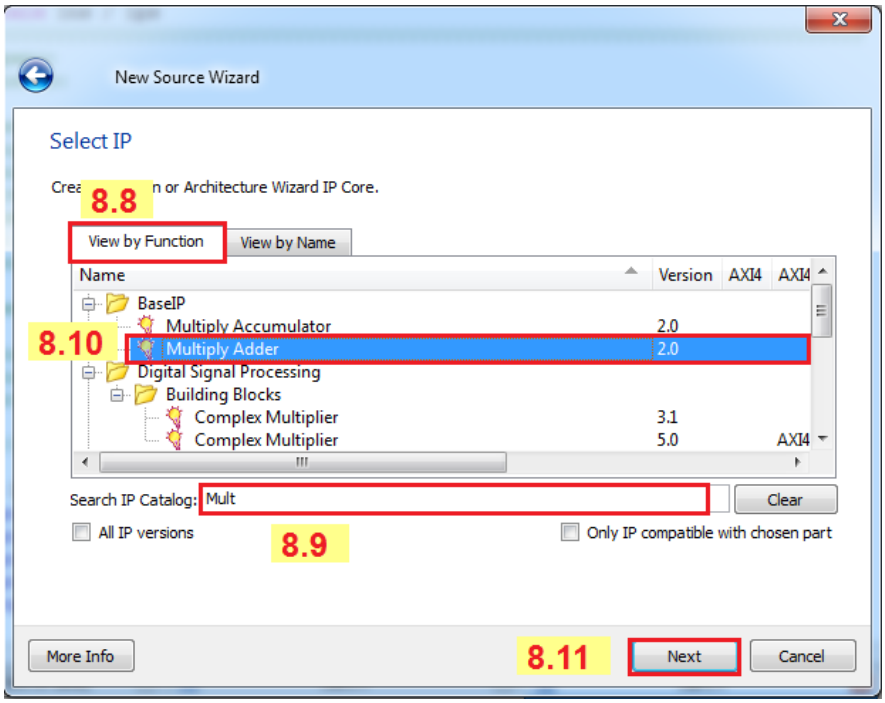

Figure 23.

8.8. In the New Source Wizard, select **View by Function** tab.

8.9. Type the key word, **Mult**, in the Search IP Catalog field to filter the IPs and only view the corresponding IPs.

8.10. Select the **Multiply Adder Version 2.0**.

8.11. Click **Next**.

8.12. In the next page, click **Finish**.

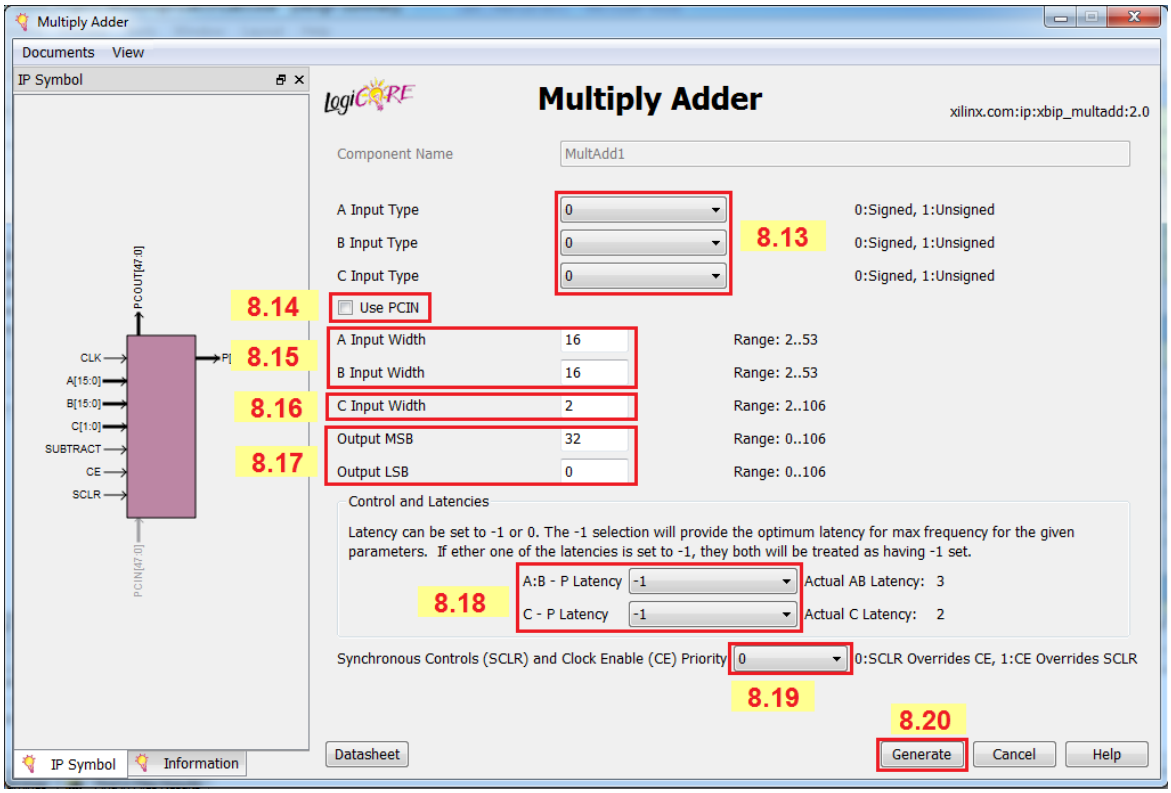

Figure 24.

8.13. Set the input type to **Signed** for all of the three inputs (i.e. A, B, and C).

8.14. Leave Use PCIN box unchecked.

8.15. Set A Input Width and B Input Width to **16**.

8.16. Set C Input Width to **2**.

8.17. Set Output MSB and Output LSB to **32** and **0**, respectively.

8.18. Set A:B-P Latency and C-P Latency to **-1** to provide optimum latency for maximum frequency for the current configuration.

8.19. Set SCR and CE priority to **0**.

8.20. Click **Generate** to generate your customized IP core.

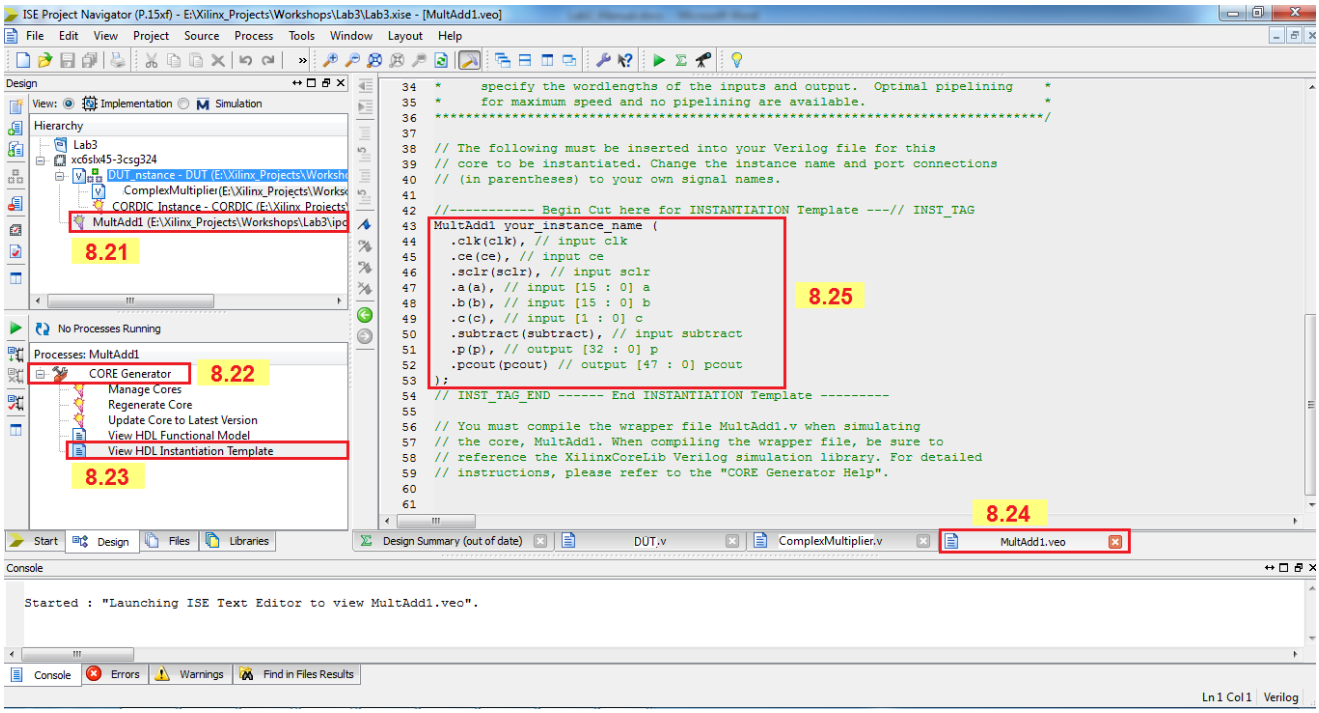

Figure 25.

8.21. In the Hierarchy pane, select the **MultAdd1 IP source**.

8.22. In the Processes Pane, expand **GORE Generator**.

8.23. Double-click on **View HDL Instantiation Template**.

8.24. MultAdd1.veo file, which includes the instantiation template of the MultAdd1 IP will be opened.

8.25. Copy and paste the uncommented part of instantiation template file to the proper place in the top module file (i.e. **DUT.v**).

8.26. Choose the desired name for the AddMult1 IP core instance.

8.27. Repeat step8.1 to step8.26 again by considering the following modifications:

- In step 8.14, check the Use PCIN box. As a result, C Input Width in step 8.16 and C-P Latency in step 8.18 would be inactive.
- Instep 8.17, set Output MSB and Output LSB to **33** and **0**, respectively.

8.28. Repeat step8.1 to step8.26 again by considering the following modifications:

- In step 8.14, check the Use PCIN box. As a result, C Input Width in step 8.16 and C-P Latency in step 8.18 would be inactive.
- Instep 8.17, set Output MSB and Output LSB to **34** and **0**, respectively.

8.29. Write a small state machine in Verilog to manage input data as described in Design Description section.

8.30. In the top module (i.e. **DUT.v**), define the required signals and connect the proper signals to the input/output ports of MultAdd instances based on the proposed architecture shown in Figure 6 and complete the **State 2**.

State2 of the state machine is complete now!

Go ahead and realize the other parts of project.

9. Implementation of Complex-Domain Matrix Multiplication (State3)

In this section you will realize **State3**, based on the proposed architecture in Figure 7. In order to implement this sub module, you need three instances of the complex multiplier. In section 7, an instance of complex multiplier was added to the project, which is useful for this section too. Thus, perform the following steps to complete this part of project:

9.1. Repeat steps 7.1 to 7.6.

9.2. Perform step 7.7 twice to add two instances of the complex multiplier.

9.3. Choose the desired names for these two instances.

9.4. In the top module (i.e. **DUT.v**), define the required signals and connect the proper signals to the input/output ports of these three instances based on the proposed architecture shown in Figure 7.

9.5. Implement the other parts of the proposed architecture using Verilog coding to complete **State3** of the state machine.

Finally, **State3** of the state machine and the final project of Lab3 are complete now!

Synthesizing the Design Using XST (Figure 26)

After completing the design entry and implementation of the proposed architecture, shown in figure 2, you should synthesize the design. In order to do that, please refer to Lab1 manual and perform Step2 of Lab1. The summary of the synthesis report is shown in Figure 26.

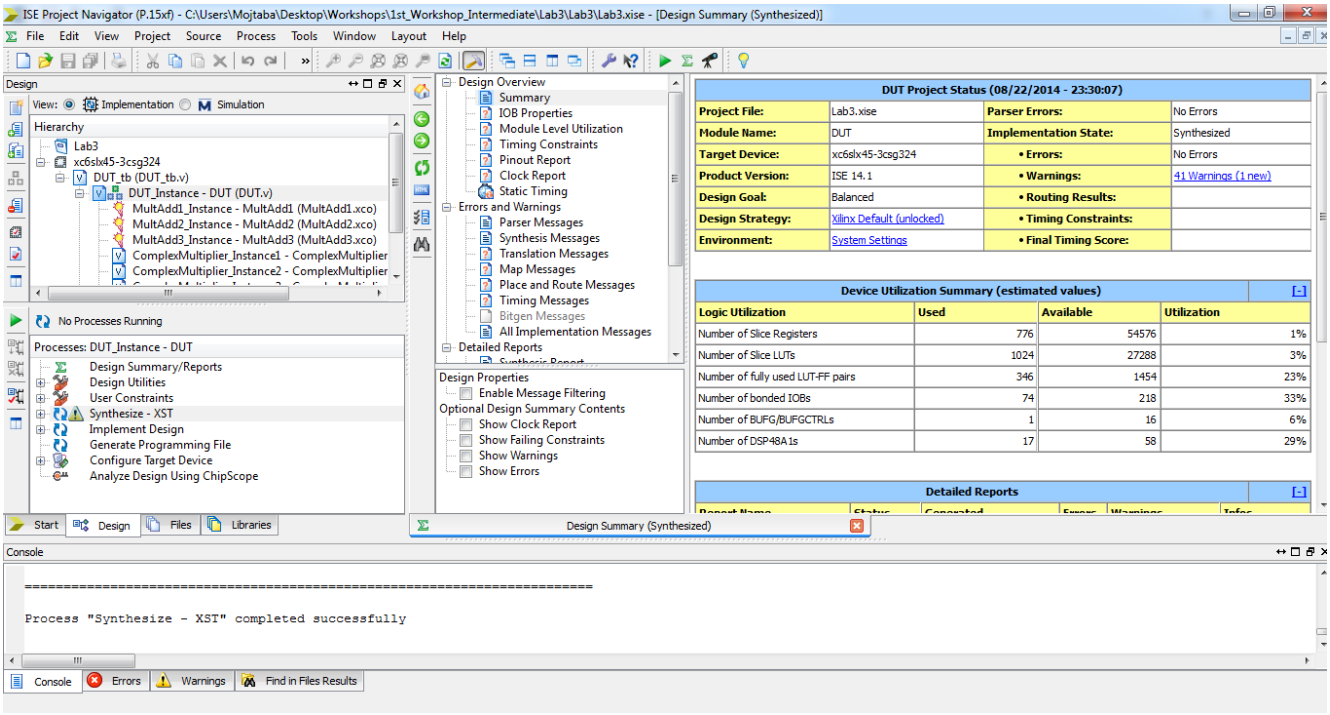

Figure 26.

View the RTL/Technology Schematic (Figure 27)

In order to generate the RTL/Technology schematic, please refer to Step3 of Lab1. Figure 27 shows the top module of Lab3 in RTL/Technology viewer.

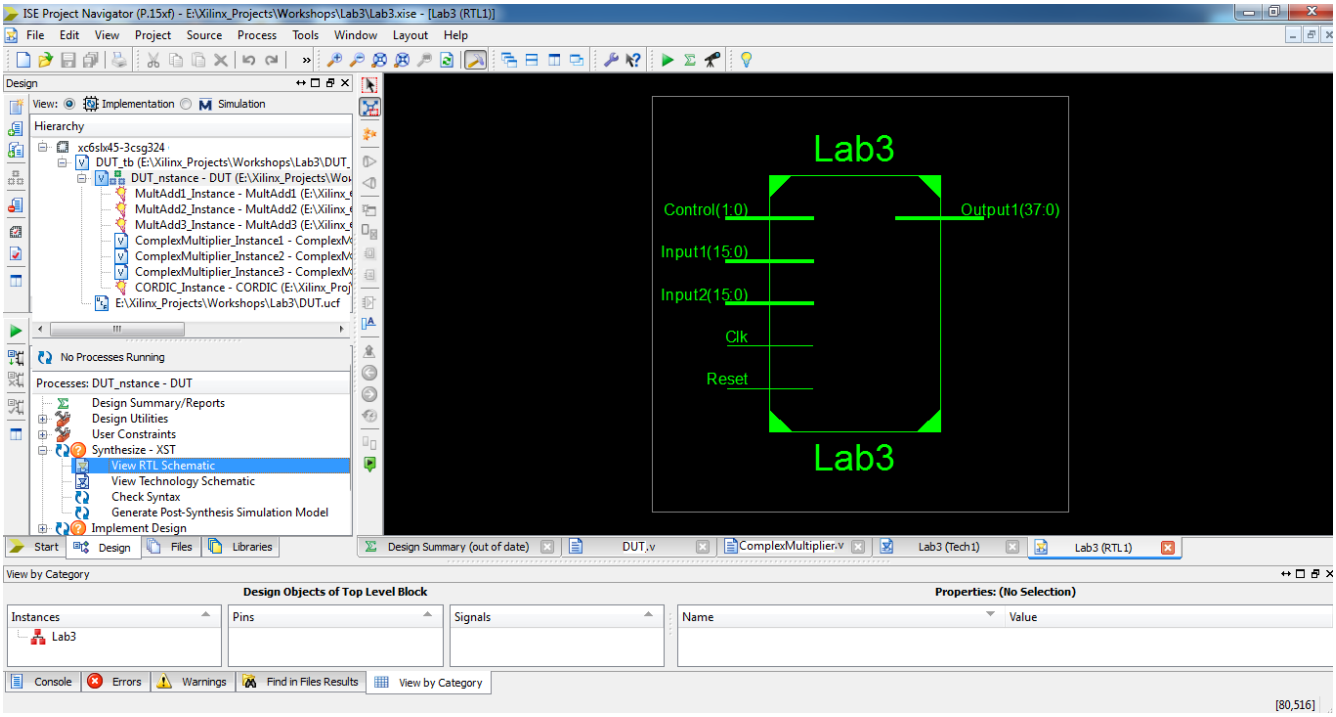

Figure 27.

Behavioral Simulation using ModelSim(Figure 28-29)

Now, you can perform behavioral simulation on your design to verify its functionality. In this lab, we prepare a test bench, in which different inputs are injected to the design to trigger the different states of the state machine. The accuracy of the output values can be verified in MATLAB. In order to simulate the final design, please refer to the Lab2 manual. A snap shot of the simulation result is shown in Figure 28 and Figure 29.

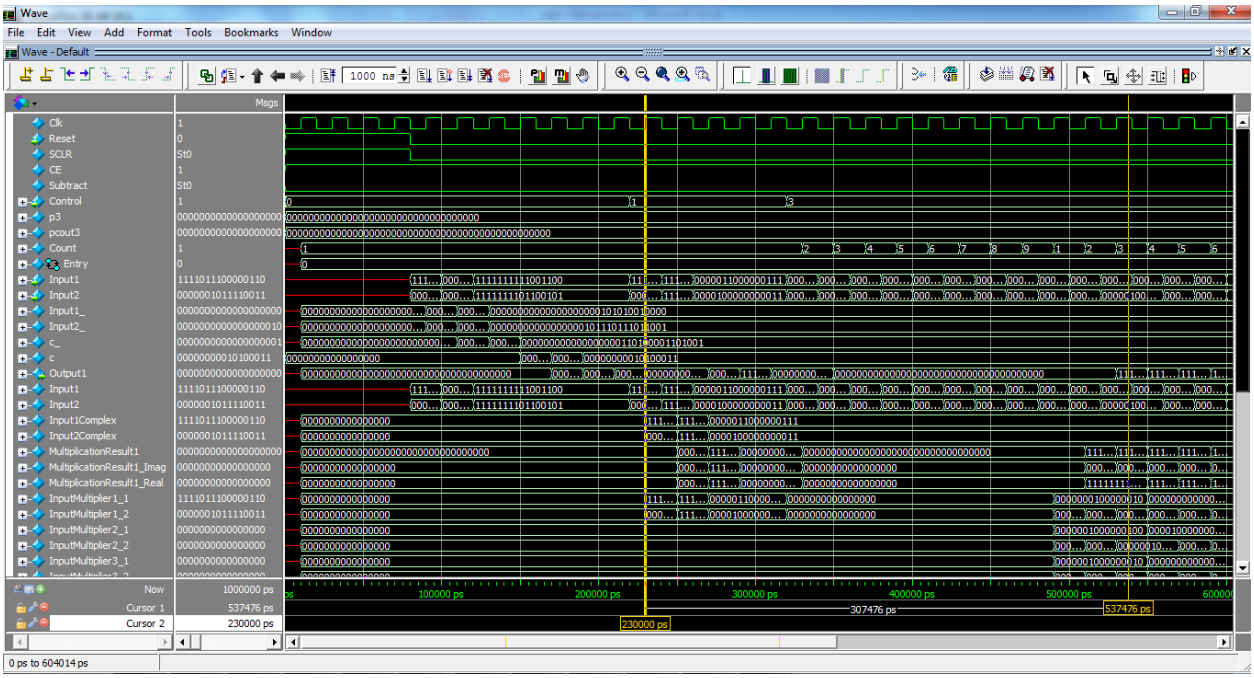

Figure 28.

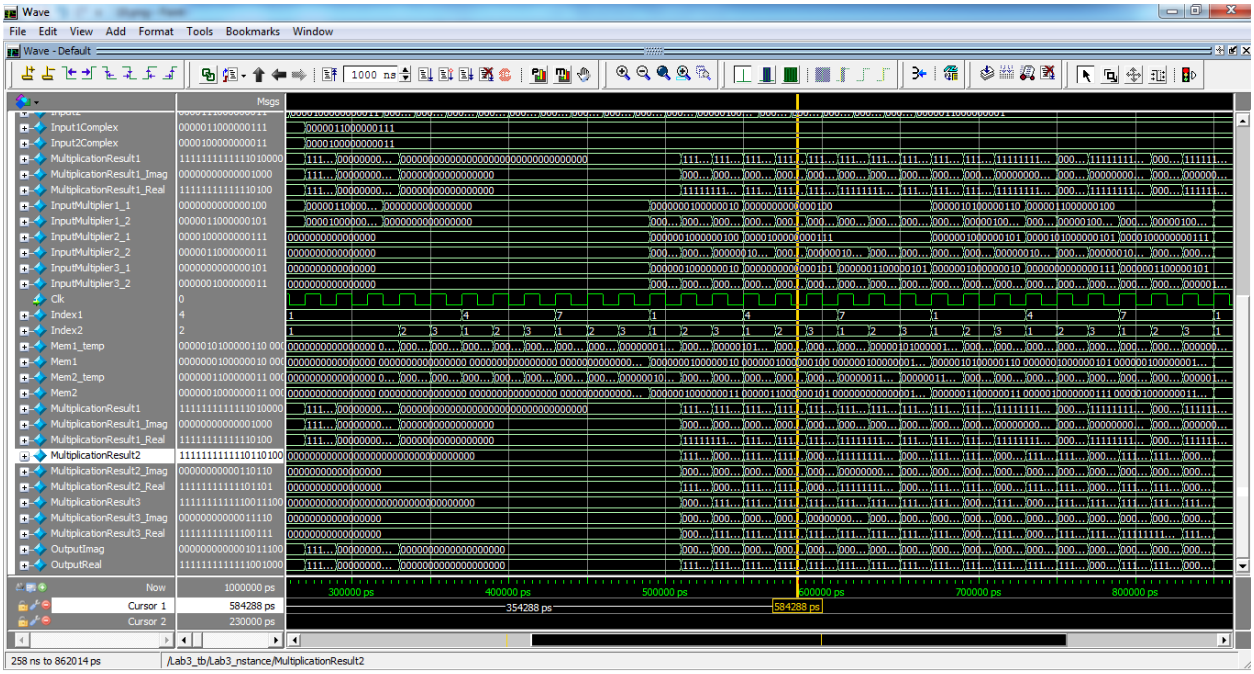

Figure 29.

References

1. Volder, J., "*The CORDIC Trigonometric Computing Technique*" IRE Trans. Electronic Computing, Vol. EC-8, Sept. 1959, pp330-334.
2. Walther, J.S., "*A Unified Algorithm for Elementary Functions*," Spring Joint computer conf., 1971, proc., pp379-385.

Digital Circuit Design with Xilinx FPGA

Lab Manual

Workshop No.1

Lab4:

Entering Constraint

Outline

This manual covers the following topics:

- Entering **Timing** constraints in **UCF** using **Constraints Editor**
- Entering **Placement** constraints in **UCF** using **ISE Text Editor**
- Entering **Synthesis, map,** and **placement** constraints in **HDL** source file using **ISE Text Editor**
- Entering **Timing** constraints in **HDL** source file using **ISE Text Editor**
- Entering constraints in **XST Constraint File (XCF)** using **ISE Text Editor**

Introduction

Constraints are essential to help you meet your design goals or obtain the best implementation of your circuit. Constraints are available in XST to control various aspects of the synthesis process itself, as well as placement and routing. Synthesis algorithms and heuristics have been tuned to automatically provide optimal results in most situations. In some cases, however, synthesis may fail to initially achieve optimal results; some of the available constraints allow you to explore different synthesis alternatives to meet your specific needs.

An implementation constraint is an instruction given to the FPGA implementation tools to direct the mapping, placement, timing or other guidelines to follow while processing an FPGA design.

The most common types of constraints are as follows:

- Timing Constraints
- Placement Constraints
- Synthesis Constraints

The most commonly used methods of entering constraints are as follows:

- User Constraints File (UCF), using one of the following tools:
  - Constraints Editor
  - ISE Text Editor
  - PlanAhead tool (for FPGAs)
  - Pinout and Area Constraints Editor (PACE) (for CPLDs)
  - Commercially available text editors
- HDL source file using a text editor
- XST Constraint File (XCF) using a text editor

The main goal of this lab is to give you some examples in the above topics. Thus, you'll learn about different methods of entering constraints to your design and also you'll learn about different types of constraints.

Required Software

To perform this manual, you must have **Xilinx ISE Design Suite 14.1** and **ModelSim 10.1** (or above) installed. Note that in this manual the pictures are captured in **Xilinx ISE Design Suite 14.1**, but all the procedures can be done in other versions of this software.

Objectives

After completing this lab, you will be able to:

- Enter various types of constraints such as timing, synthesis, map, and placement constraints using different methods:
  - **UCF**
    - Using Constraints Editor
    - Using ISE/commercial Text Editor
  - **HDL**
    - Using ISE/commercial Text Editor
  - **XCF**
    - Using ISE/commercial Text Editor

General flow of this lab

This lab comprises three steps as follows:

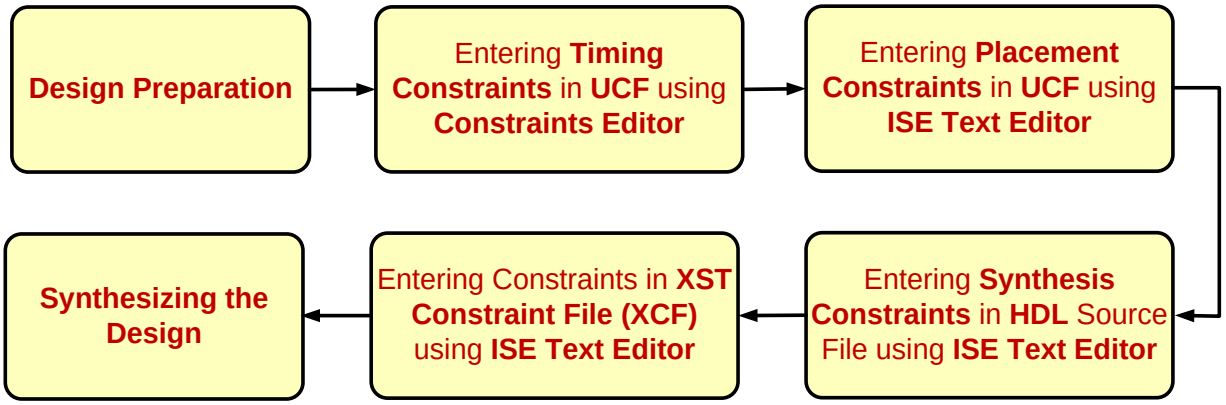

Figure 1. General flow of Lab4.

Design Description

In this lab, you'll use the completed design of Lab3. So, you will constrain this design with different types of constraints. This process will be done via different methods of entering constraints. In order to get more information about the design, please refer to Lab3 manual.

Design Preparation

1. Start the Project Navigator

Double click the Project Navigator icon on your desktop, or select **Start > All Programs > Xilinx Design Tools > ISE Design Suite 14.1 > ISE Design Tools > Project Navigator.**

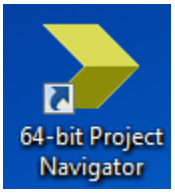

Figure 2. Project Navigator desktop icon.

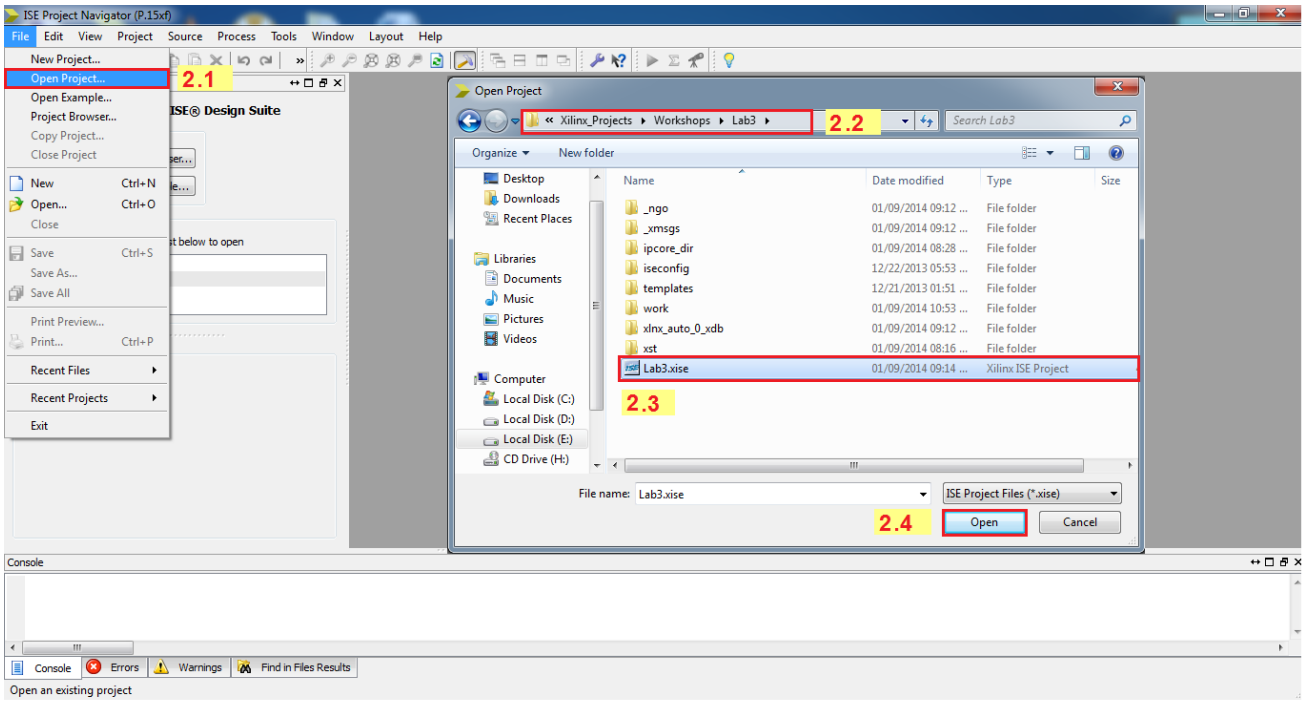

Figure 3.

2. Open the project (Figure 3)

2.1. From Project Navigator, select **File > Open Project**.

2.2. Go to the project directory of Lab3.

2.3. Select **Lab3.xise**.

2.4. Click **Open**.

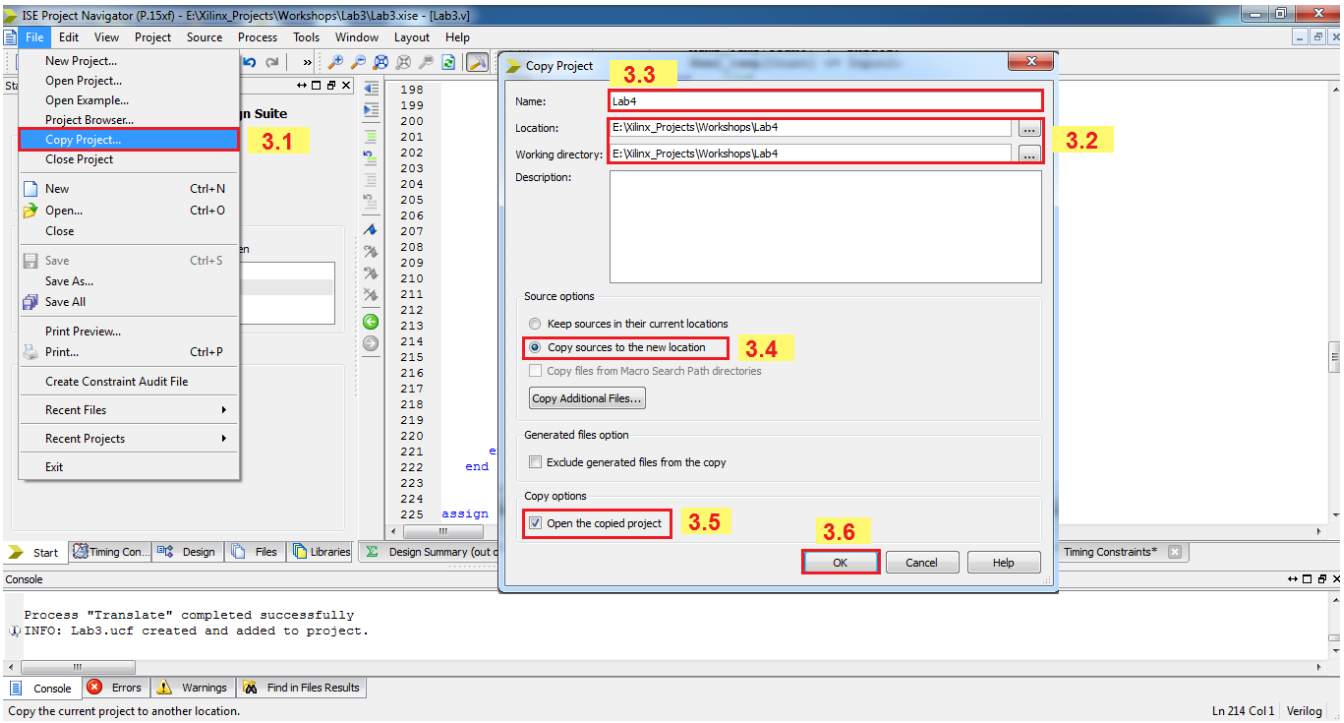

Figure 4.

3. Copy the project (Figure 4)

3.1. From Project Navigator, select **File > Copy Project**.

3.2. Specify the location, in which you want to save the result of this lab.

3.3. Enter **Lab4** as the name of the copied project.

3.4. Select **Copy sources to the new location**.

3.5. Select the **Open the copied project** to open the Lab4 project automatically.

3.6. Click **OK**.

Entering Timing Constraints in UCF using Constraints Editor

The method of applying constraints given in this section is to provide a User Constraints File (UCF) and add it to the current project. The User Constraints File (UCF) is a text file and can be edited directly with a text editor such as:

- ISE Text Editor
- Commercially available text editors

To facilitate editing of this file, graphical tools are provided to create and edit constraints:

- Constraints Editor
- PlanAhead tool (for FPGAs)

The Constraints Editor and PlanAhead tool are graphical tools that enable you to enter timing and I/O and placement constraints.
However, the easiest way to enter design constraints is to use Constraints Editor, which:

- Provides a unified location in which to manage all timing constraints associated with a design.
- Provides assistance in creating timing constraints from the design requirements.

In this section, we'll create a UCF for the current design to enter various constraints. We perform this process via three possible methods (i.e. ISE text editor, Commercial text editors, and Constraints Editor).

1. Creating UCF using Constraint Editor to Enter Constraints for Clock Period (Figure 5-7)

The Constraints Editor enables you to do the following:

- Edit constraints previously defined in a UCF file.
- Add new constraints to your design.

In order to enter timing constraints using Constraint Editor, do the following:

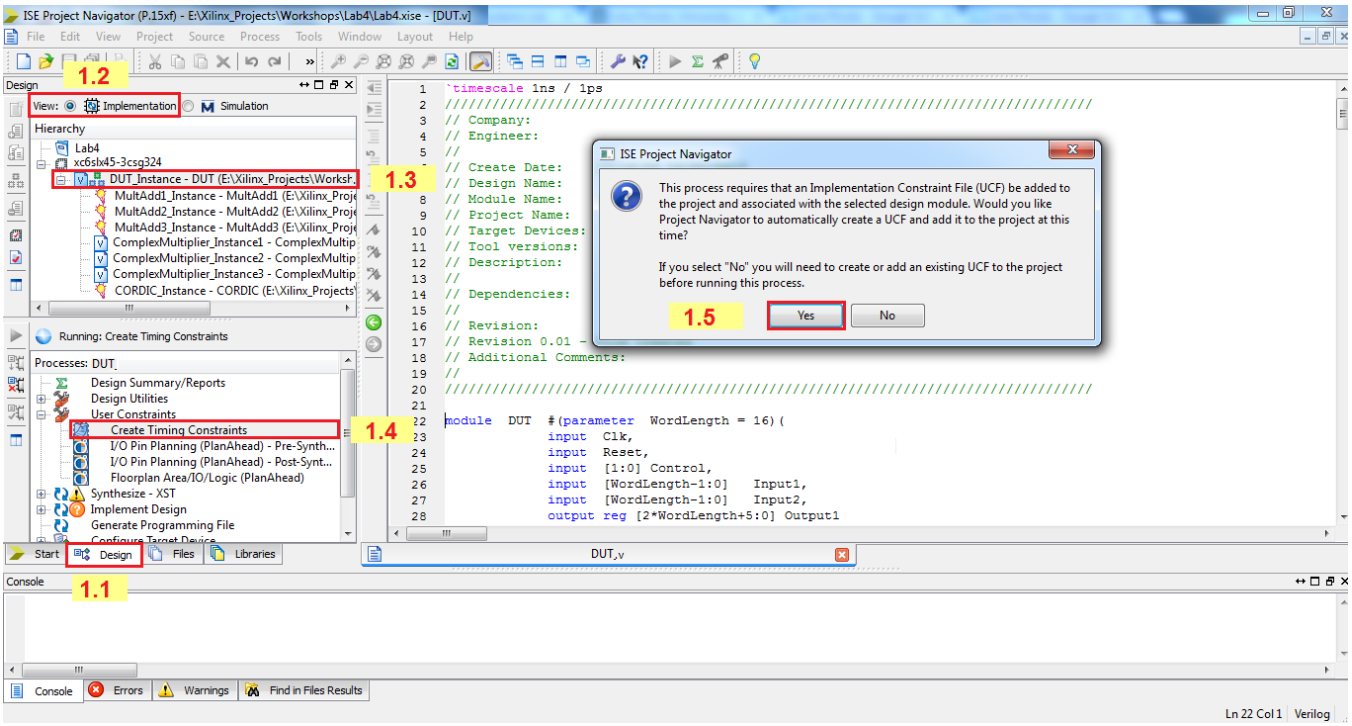

Figure 5.

1.1. Select **Design Panel**.

1.2. Set the Design View to the **Implementation**.

1.3. In the Hierarchy pane, select the **top module** (i.e. **DUT.v**)

1.4. In the Processes pane, expand **User Constraints**, and double-click **Create Timing Constraints**.

When you run the Create Timing Constraints process, Translate (described in Lab6) is automatically run and the ISE Design Suite launches the Constraints Editor.

1.5. Click **Yes** to create a UCF and add it to the current project.

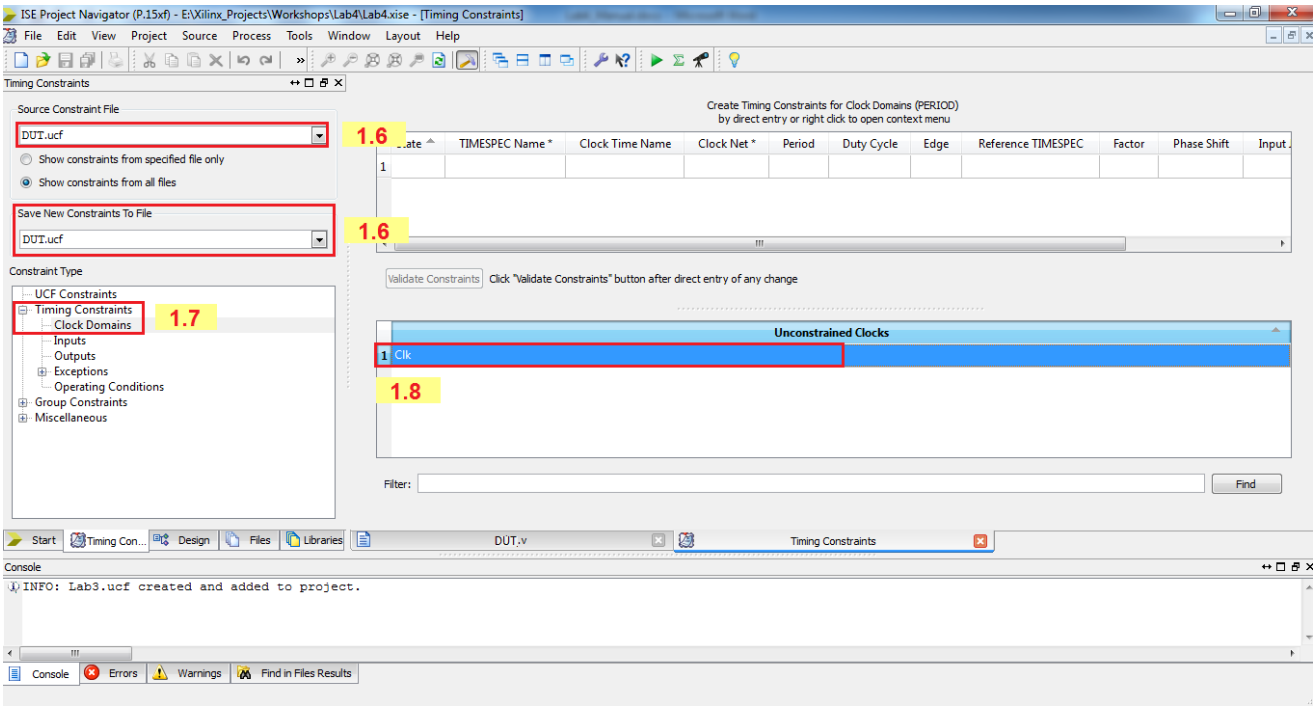

Figure 6.

1.6. If your design has multiple UCF, choose the proper file.

1.7. Expand **Timing Constraints** and select **Clock Domains**.

1.8. Double-click the row containing the **Clk** signal in the Unconstrained Clocks table.

1.9. In the designs with multiple clock signals, the proper signal should be set in this part.

1.10. In the Clock Signal Definition, select **Specify Time** is selected, which enables you to define an explicit period for the clock.

1.11. Enter a value of **30** in the Time field.

1.12. Verify that **ns** is selected from the Units drop-down list.

1.13. Select **Rising (HIGH)** as the Initial clock edge.

1.14. Enter a value of **50** in the Rising duty cycle.

1.15. Verify that **%** is selected from the Units drop-down list.

1.16. For the Input Jitter section, enter a value of **60** in the Time field.

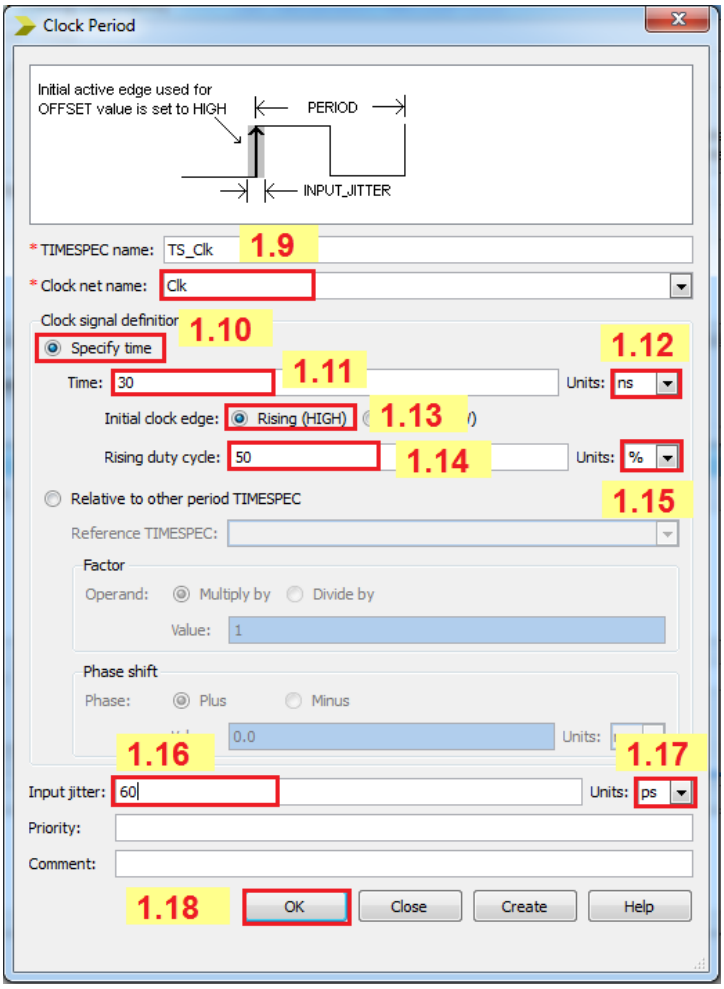

Figure 7.

1.17. Verify that **ps** is selected from the Units drop-down list.

1.18. Click **OK**.

The period constraint is displayed in the constraint table at the top of the window. The period cell is updated with the global clock period constraint that you just defined (with a default 50% duty cycle).

2. Entering Constraints for Inputs (OFFSET IN) in UCF using Constraint Editor (Figure 8-10)

2.1. In the Constraint Type tree view, expand **Timing Constraints**.

2.2. Select the **Inputs**.

2.3. In the Global OFFSET IN Constraint table, double-click the **Control<0>** to bring up the Create Setup Time (OFFSET IN) wizard.

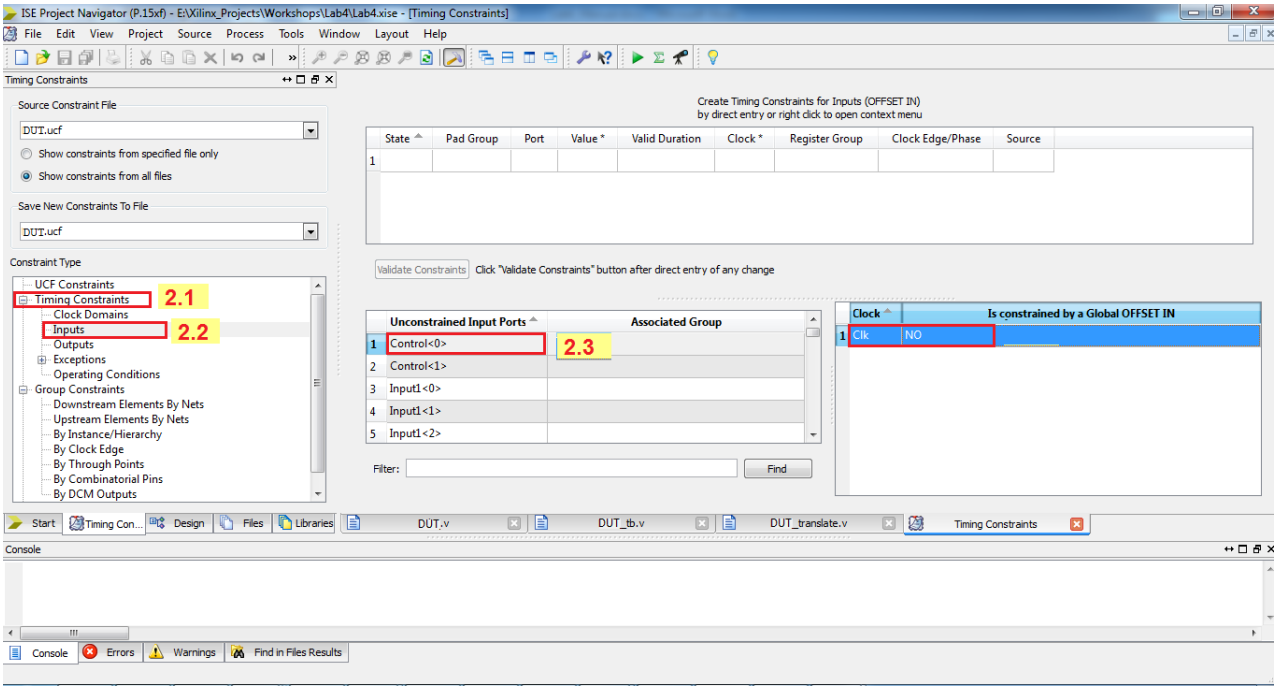

Figure 8.

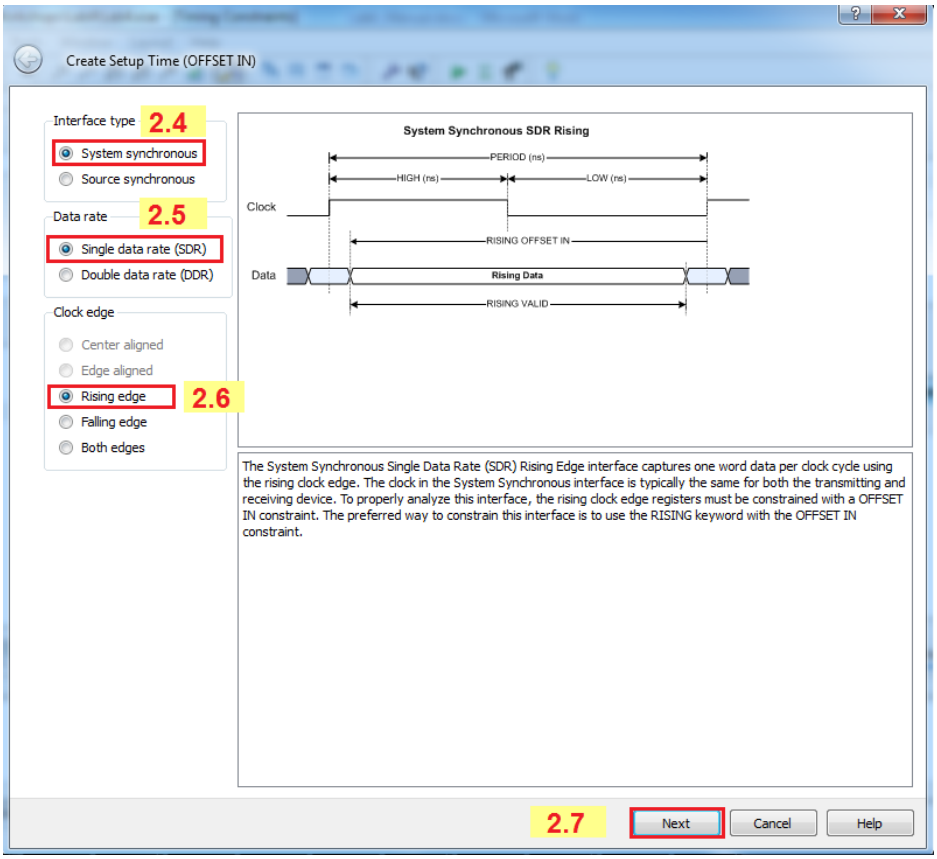

Figure 9.

2.4. Select **System synchronous** as the Interface type.

2.5. Select **Single data rate (SDR)** as the type of the Data rate.

2.6. Select **Rising edge** for the type of Clock edge.

2.7. Click **Next**.

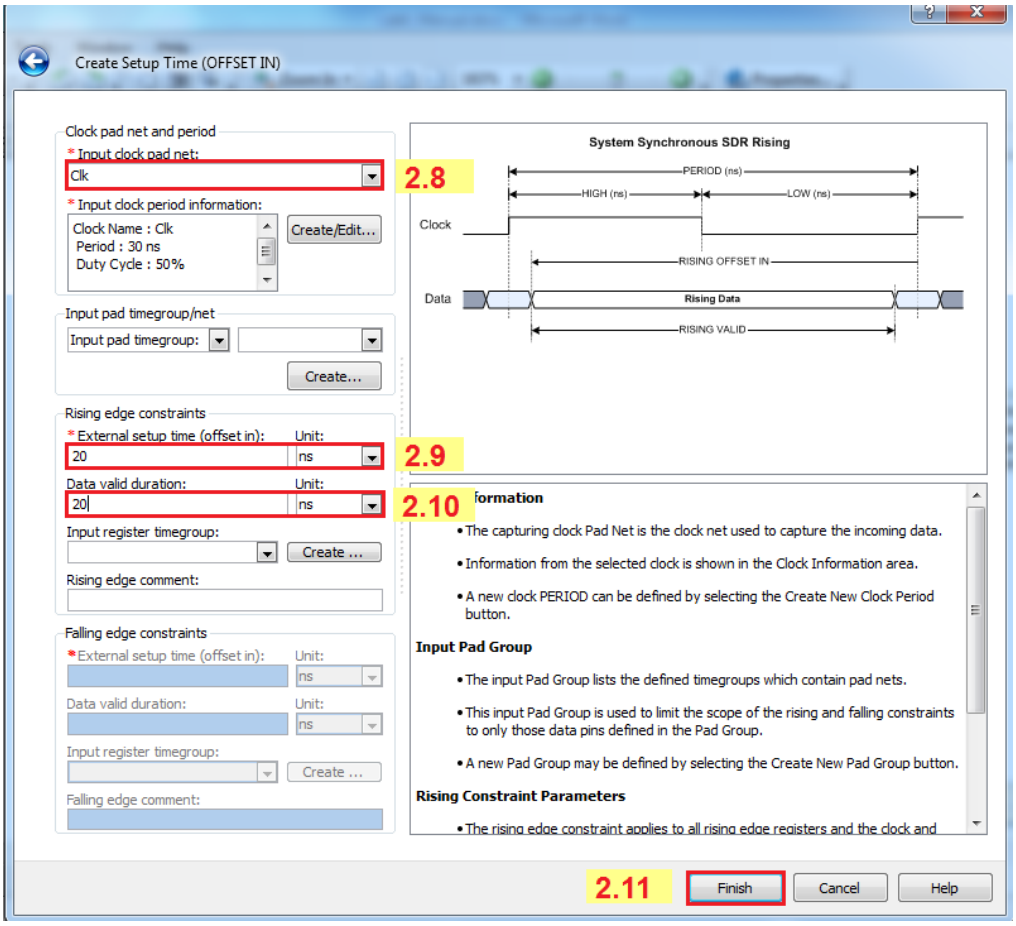

Figure 10.

2.8. Select **Clk** as the Input clock pad net.

2.9. In the External setup time (offset in) field, enter **20 ns**.

2.10. In the Data valid duration field, enter **20 ns**.

2.11. Click **Finish**.

This creates a Global OFFSET IN constraint for the **Control<0>** signal.

3. Creating Time Group in Constraint Editor (Figure 11-13)

3.1. If your design has multiple UCF, choose the proper file.

3.2. In the Constraint Type tree view, expand **Timing Constraints**.

3.3. Select the **Outputs**.

3.4. In the Unconstrained Output Ports, select **Output1<0> to Output1<37>**.

3.5. Right-click on the selected signals and select **Create Time Group**.

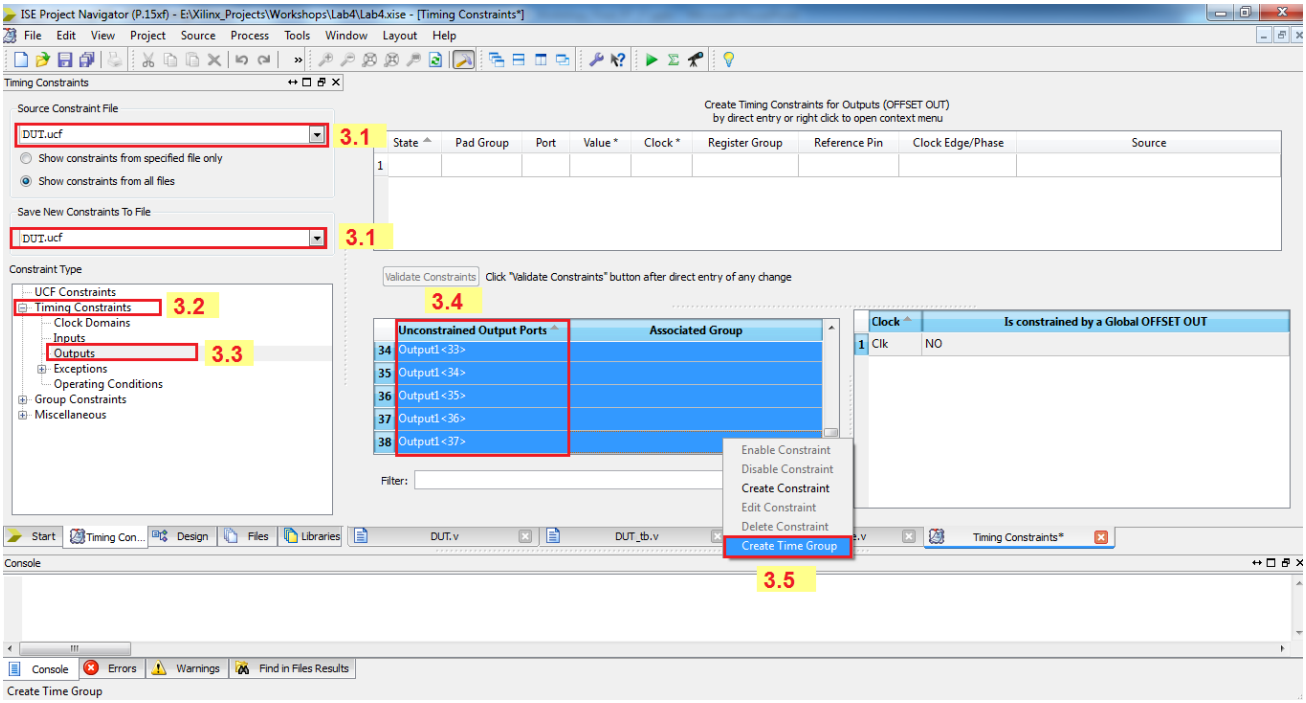

Figure 11.

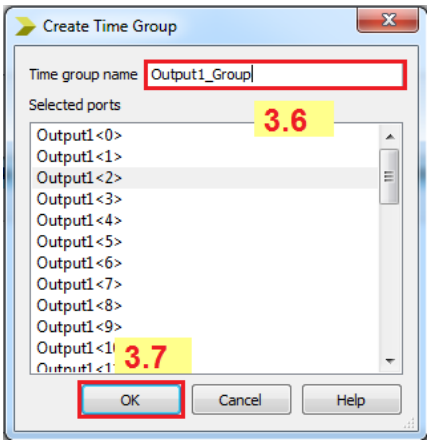

Figure 12.

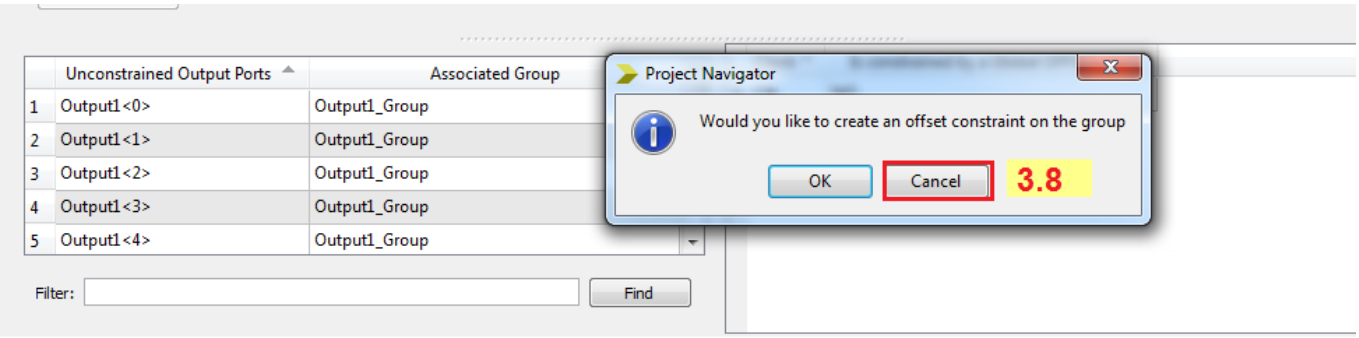

Figure 13.

3.6. Enter a name for the group of selected signals.

3.7. Click **OK**.

3.8. In the next window click **Cancel**. You'll enter constraints for these output signals in the next section of this lab.

4. Entering Constraints for Outputs (OFFSET OUT) in UCF using Constraint Editor (Figure 14-15)

4.1. If your design has multiple UCF, choose the proper file.

4.2. In the Constraint Type tree view, expand **Timing Constraints**.

4.3. Select the **Outputs**.

4.4. In the Associated Group column, select one of the output signals, which is associated with **Output1_Group.**

4.5. Right-click on the selected signal and select **Create Constraints**.

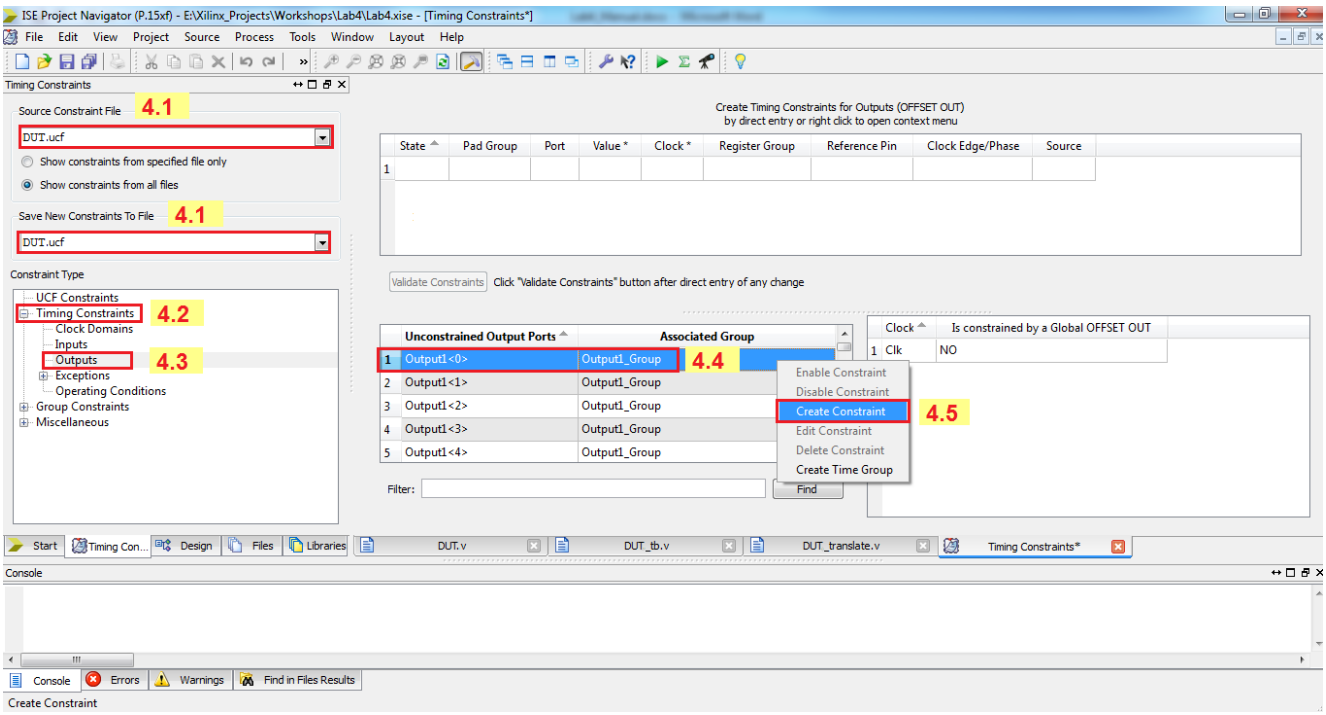

Figure 14.

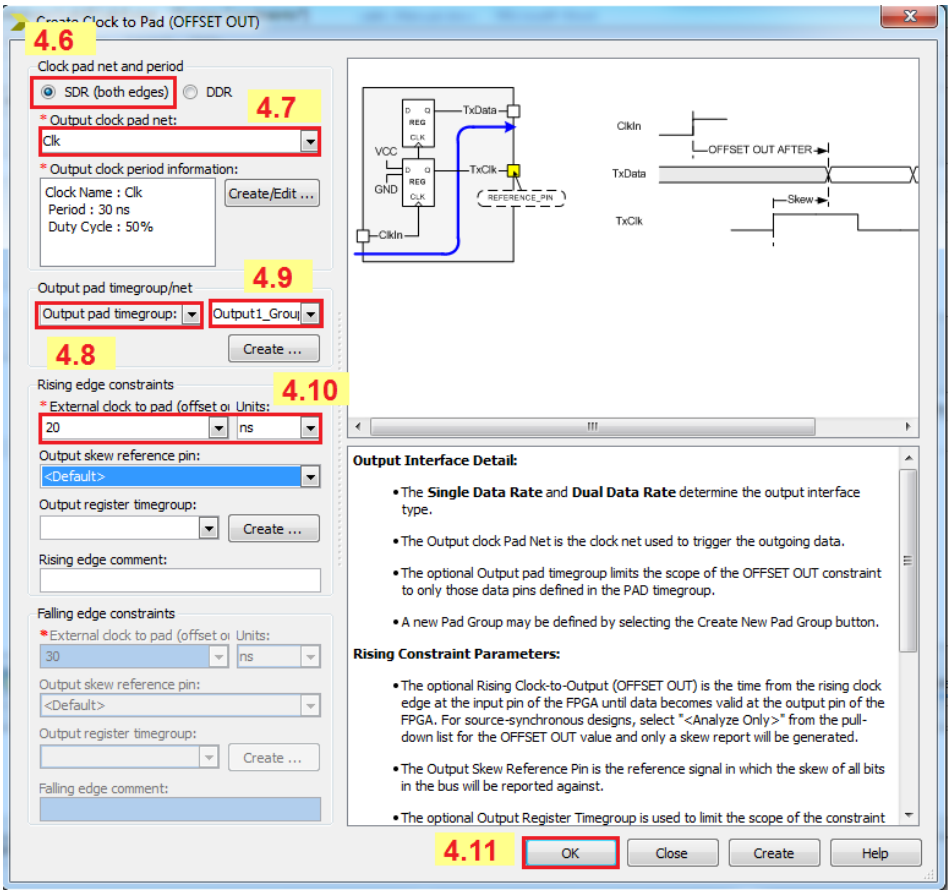

Figure 15.

4.6. Select the **SDR** as the Clock pas net and period.

4.7. Select **Clk** (in the designs with multiple clock signals).

4.8. Select **Output pad timegroup** in the Output pad timegroup/net part.

4.9. Select **Output1_Group**, which you created it in the previous section.

4.10. Enter **20 ns** for the External clock to pad.

4.11. Click **OK**.

4.12. In the Constraints Editor, select **File > Save**.

The changes are now saved in the **DUT.ucf** file in your current working directory.

4.13. Select **File > Close**, to close the Constraints Editor.

Entering Placement Constraints in UCF using ISE Text Editor

In this section you'll enter a sample of placement constraints in the UCF for your project. You'll enter the Location (LOC) constraint, which is a basic placement constraint. This constraint helps the designer to map the input/output ports of the design (i.e. the ports of the top module) to the desired pins of FPGA. Do the following to enter this constraint for Clk and Reset ports in the UCF via the ISE Text Editor.

1. Entering Location Constraints in UCF using ISE Text Editor (Figure 16)

1.1. Select **Design Panel**.

1.2. Set the Design View to the **Implementation**.

1.3. In the Hierarchy pane, expand the **top module** (i.e. **DUT.v**)

1.4. In the Hierarchy pane, select **DUT.ucf**.

1.5. In the Processes pane, expand **User Constraints**, and double-click **Create Timing Constraints**.

1.6. Double-click **Edit Constraints (Text)**.

1.7. **DUT.ucf**, which you created it in the previous sections of this lab, would be opened. Enter the Location constraints using the following syntax:

NET "Signal Name" LOC=Pin Number in FPGA Package;

This process is done for **Clk** and **Reset** ports as follows:

NET "Clk" LOC = L15;

NET "Reset" LOC = A10;

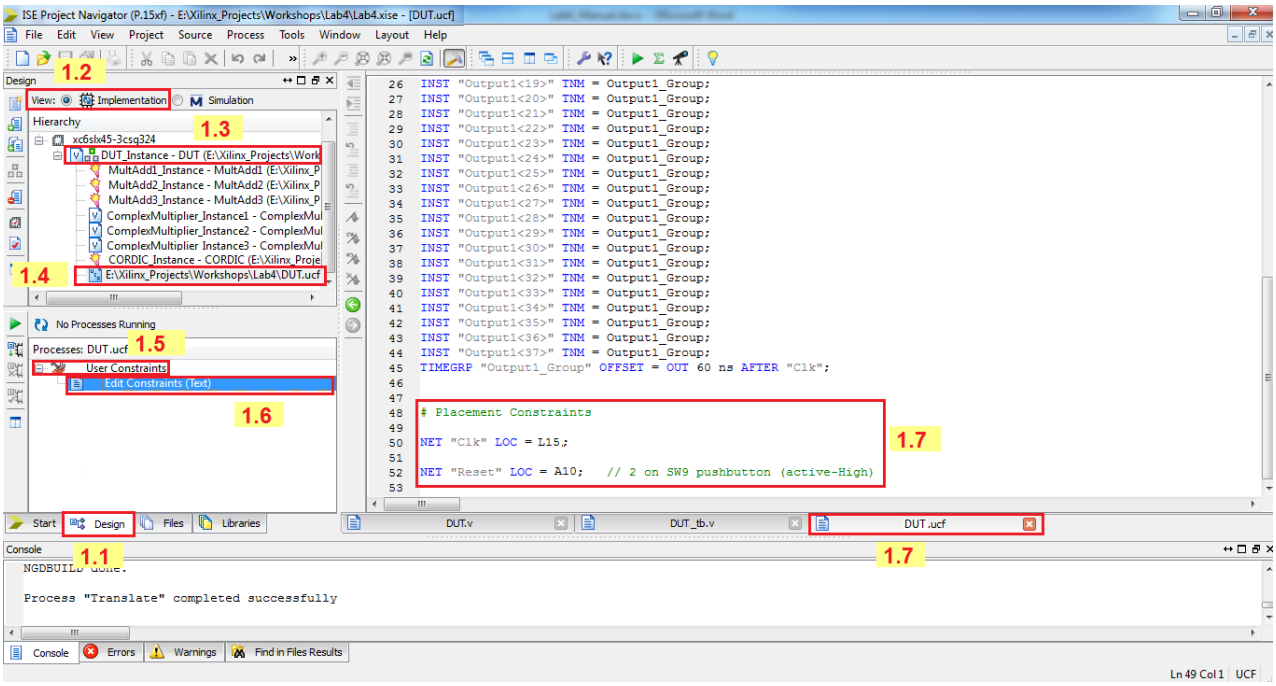

Figure 16.

NOTE:

- *You can add these Location constraints and also the other placement constraints in the UCF using the ISE text editor (i.e. step1.1 to 1.6 of this section).*
- *It is necessary to add the Location constraint for the other ports of the project, if you want to test the design on the FPGA board.*
- *Entering constraints in UCF can be done outside the ISE tool by using a commercial text editor.*
- *This action can be done via the above method or it can be done using PlanAhead Tool, which is described in a separate lab.*
- *The other placement, synthesis, and timing constraints can be applied to the design via the above method.*

Entering Synthesis/Implementation Constraints in HDL Source File using ISE Text Editor (Figure 17)

You can apply the constraints to your design by entering the in the HDL source file directly. There are some rules and a unique syntax. In this section, we'll show you this method by a few examples as follows:

1. Select **Design Panel**.

2. Set the Design View to the **Implementation**.

3. In the Hierarchy pane, double-click the **top module** (i.e. **DUT.v**) to open it in the ISE text editor.

4. Enter the following **synthesis/implementation constraints** in the proper place of the top module as follow:

Keep Hierarchy:
The **Keep Hierarchy** (KEEP_HIERARCHY) constraint is a synthesis and implementation constraint, which is used to maintain the hierarchy during synthesis. The implementation software uses Keep Hierarchy to:

- Preserve the hierarchy throughout the implementation process.
- Allow a simulation netlist to be created with the desired hierarchy.

Place the Verilog constraint immediately before the module or instantiation. Specify the Verilog constraint as follows:

(* KEEP_HIERARCHY = "{TRUE|FALSE}" *)

5. Enter the following **synthesis/map constraints** in the proper place of the top module as follow:

Keep:
The Keep (KEEP) constraint is an advanced mapping constraint and also a synthesis constraint. It is used to prevent a net from being absorbed into a logic block. When a design is mapped, some nets may be absorbed into logic blocks. When a net is

absorbed into a logic block, it can no longer be seen in the physical design database. This may happen, for example, when the components connected to each side of a net are mapped into the same logic block. The net may then be absorbed into the logic block containing the components. The Keep constraint prevents the net from being absorbed.

Place the Verilog constraint immediately before the module or instantiation as follows:

(* KEEP = "{TRUE|FALSE}" *)

6. Enter the following placement constraints in the proper place of the top module as follow:

LOC:

This is a basic placement constraint, which is used to map the design ports to the FPGA pins. Also, the Location (LOC) constraint specifies the absolute placement of a design element on the FPGA die. It can be a single location, a range of locations, or a list of locations. You can specify LOC from the design file and also direct placement with statements in a constraints file.

Place the Verilog constraint immediately before the module or instantiation as follows:

(* LOC = " location" *)

NOTE:

- *Entering constraints in HDL source file can be done outside the ISE tool by using a commercial text editor.*

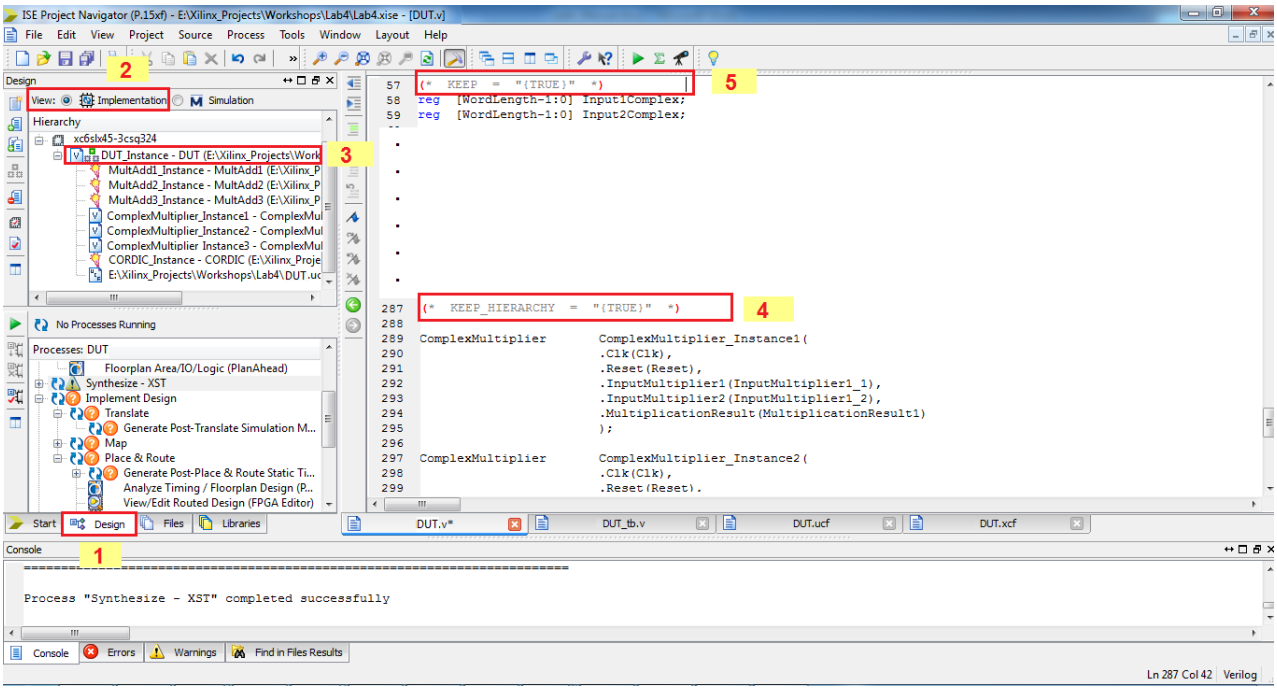

Figure 17.

Entering Constraints in XST Constraint File (XCF) using ISE Text Editor (Figure 18-21)

In this section you will learn the method of entering constraints to the design using an XST Constraint File (XCF). Do the following steps to constrain the design using an XCF:

1. Select **File>New** from the menu.
2. Select **Text File**.
3. Click **OK**.

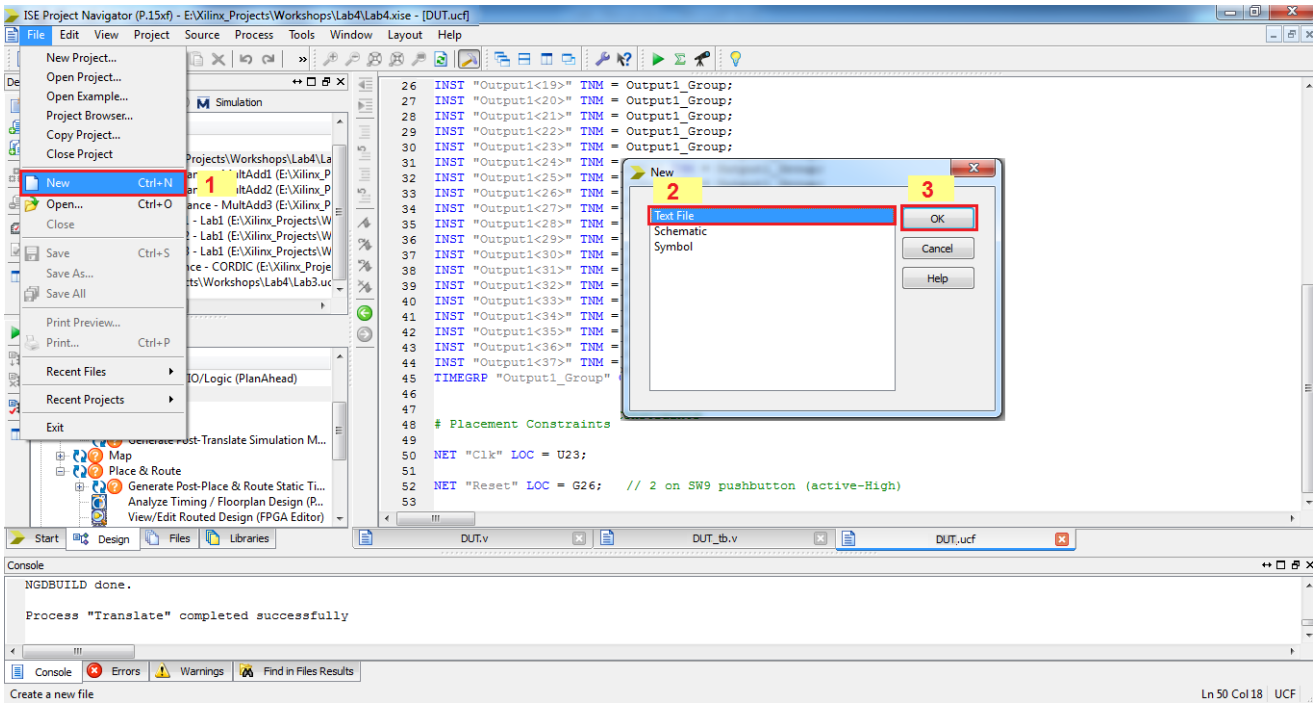

Figure 18.

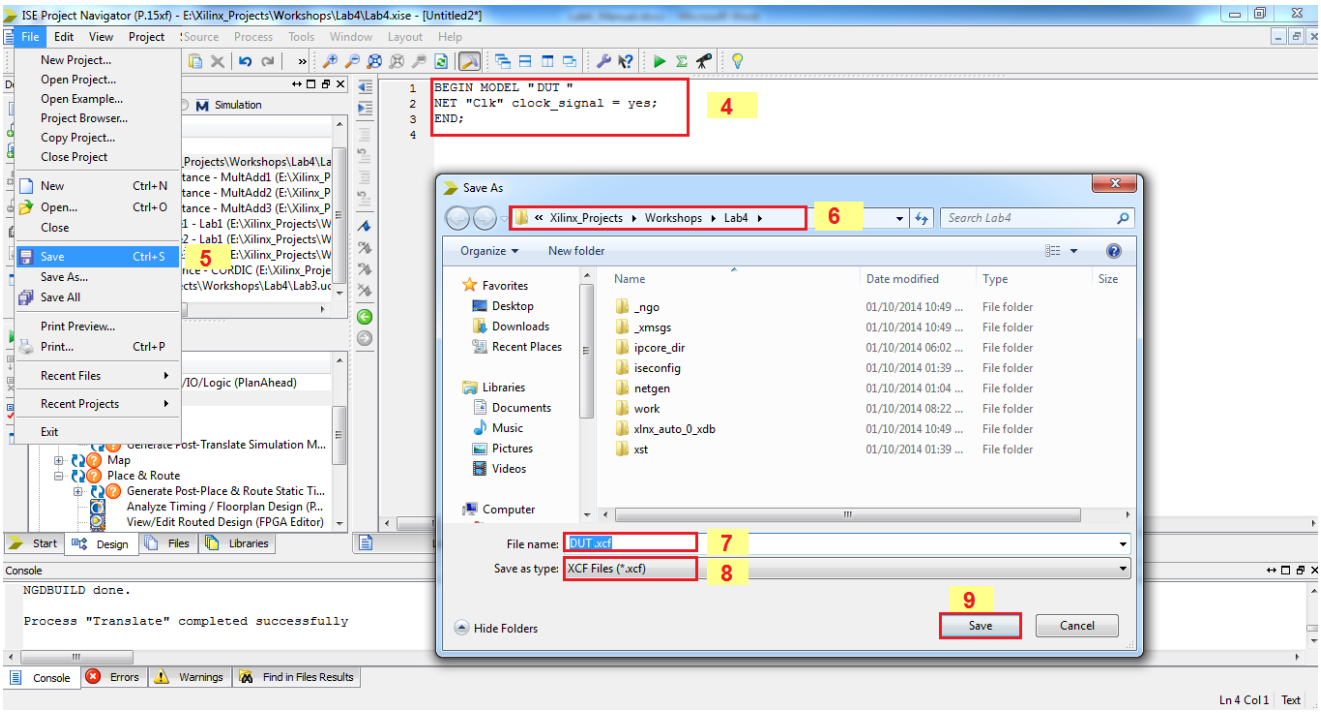

Figure 19.

4. Using the standard syntax of XCF, you can enter the required constraints in this file. For example, we enter the "Clock constraint" to specify the clock signal as follow:

BEGIN MODEL "DUT"
NET "Clk" clock_signal = yes;
END;

Also, you can enter the other **synthesis/implementation constraints** in the proper place of the top module based on the XCF syntax. For example, you can enter the **Keep Hierarchy** constraint as follow:

MODEL "DUT" keep_hierarchy = {yes|no};

5. Select **File > Save**.

6. Choose the project directory path to save the XCF file

7. In the Save as File options, select **XCF Files (*.xcf)**.

8. Enter a name for the XCF file.

9. Click **Save**.

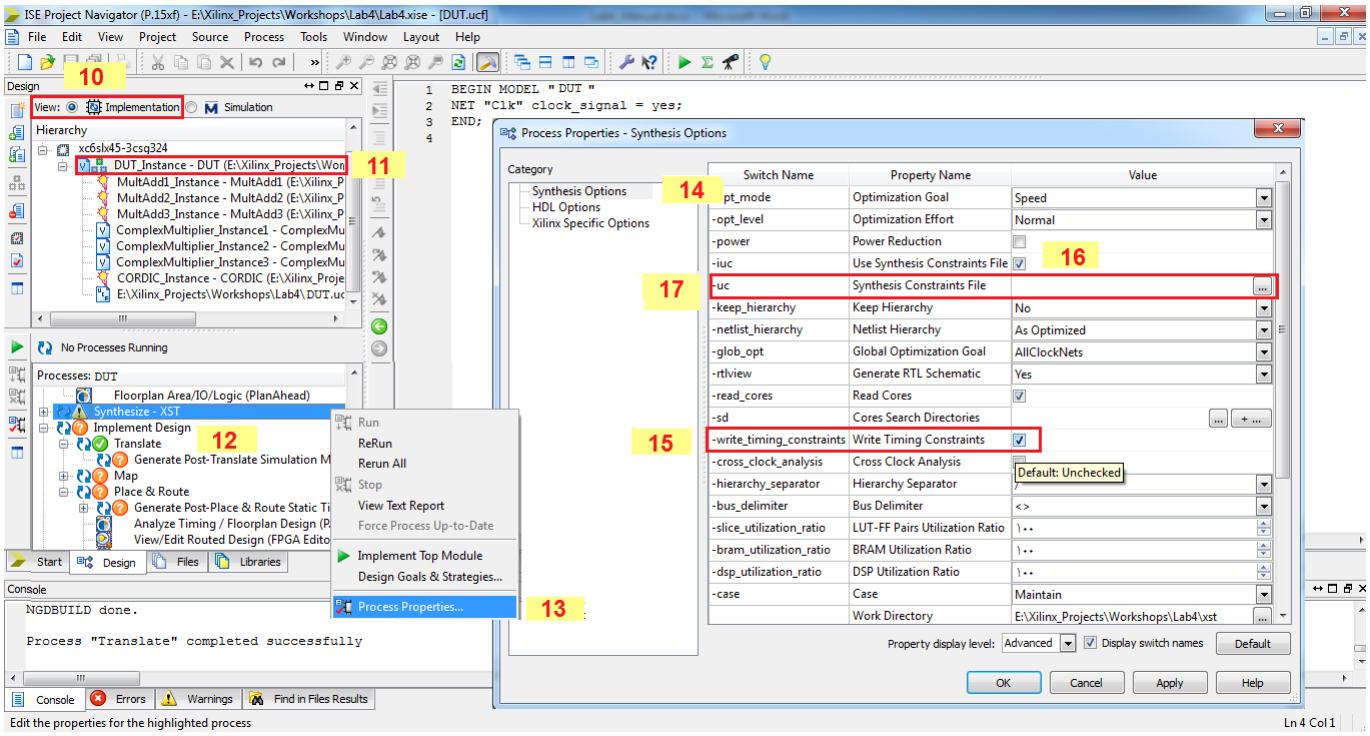

Figure 20.

10. In the View Pane of the Design Panel, select **Implementation**.

11. Select the top module (i.e. **DUT.v**).

12. In the Processes Pane, select **Synthesize-XST**.

13. Right-click and select **Process Properties.**

14. Select **Synthesis Options** from the Category.

15. Select the **Write Timing Constraints**.

16. Select the **Use Synthesis Constraint** File box.

17. Click on the **brows button**, in the Synthesis Constraint File row.

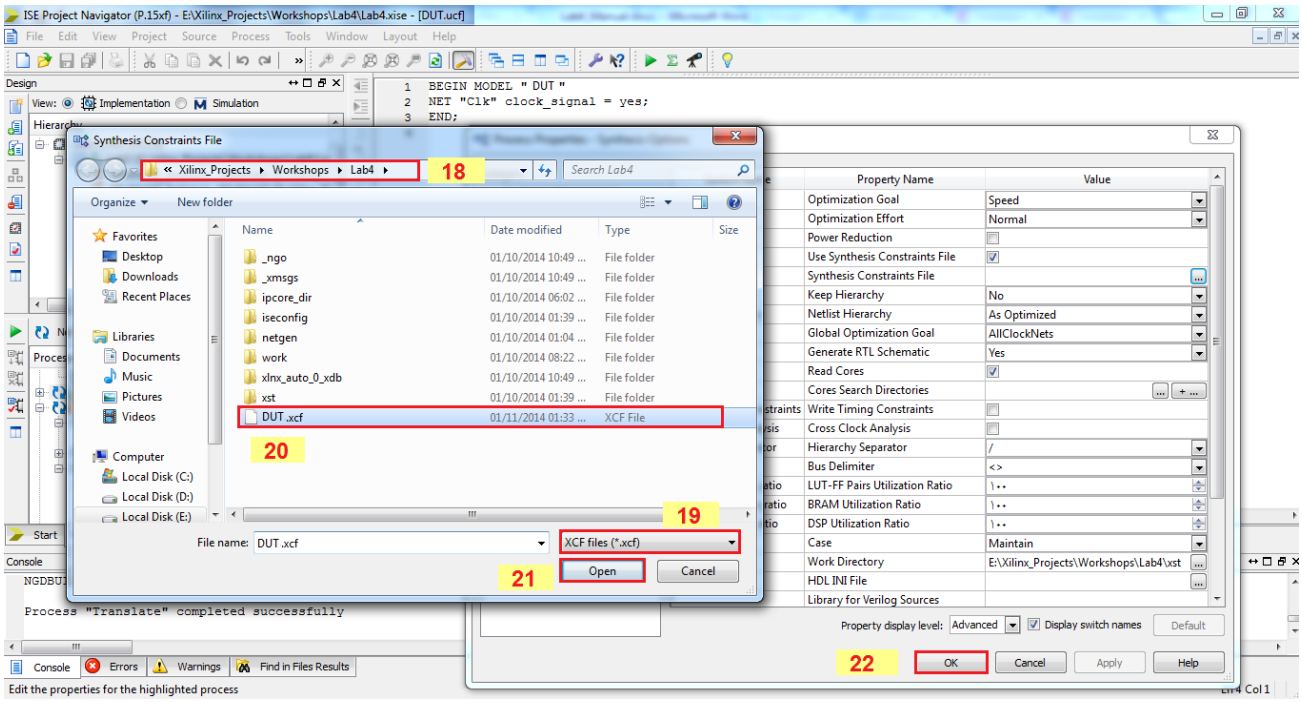

Figure 21.

18. Choose the project directory path.

19. In the file types, select **XCF Files (*.xcf)**.

20. Select **DUT.xcf**.

21. Click **Open**.

22. Click **OK**.

NOTE:

- *Placements constraints, synthesis constraints, and the other timing constraints can be applied to the design via the above method.*
- *In order to enter the other constraints only step 4 of this section will be changed.*
- *You can use any text editor instead of ISE text editor to create the XCF file. But the most important thing is saving the file with .xcf extension. XST uses this extension to determine if the syntax is related to the new or old XCF style. Please note that if the extension is not .xcf, XST will interpret it as the old constraint style.*

Synthesizing the Design (Figure 22)

After entering various constraints to the project, you should synthesize your design to check the syntax and other rules and also to correct possible errors. Do the following to complete this lab:

1. In the toolbars click **Save All** to save any changes in all of the sources of the project.
2. Select **Design Panel**.
3. Set the Design View to the **Implementation**.
4. In the Hierarchy pane, select the **top module** (i.e. **DUT.v**).
5. In the processes pane, double-click on the **Synthesize-XST.**
6. In the Transcript Window, check the **Console** tab to view the result of this process.
7. Read the content of Errors/Warnings tabs in the Transcript Window to check the Errors/Warnings after the synthesis process.

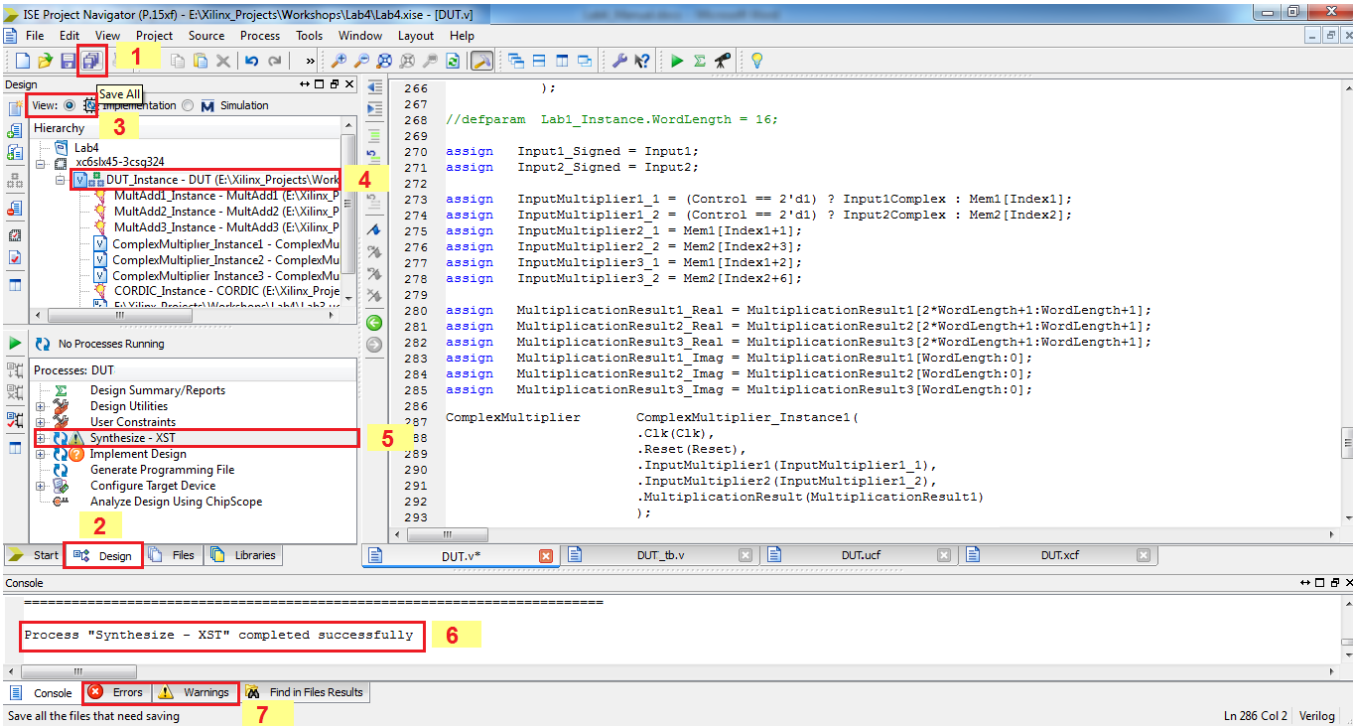

Figure 22.

Digital Circuit Design with Xilinx FPGA

Lab Manual

Workshop No.1

Lab5:

Getting Started with ATLYS Board

Outline

This manual covers the following topics:

- Introduction about the Atlys Board
- Connect to the Atlys Board
- Getting Started with Adept System
- FPGA Configuration using Adept Software
- Perform a Test Interface for Peripherals and I/O
- Displayed the Real-time Current and Power

Introduction

The Atlys board is a complete, ready-to-use digital circuit development platform based on a Xilinx Spartan-6 LX45 FPGA, speed grade -3. Atlys board is an ideal host for a wide range of digital systems. Atlys is compatible with all Xilinx CAD tools. The main components and features of this board, which are shown in Figure 1, are as follow:

- Xilinx Spartan-6 LX45 FPGA, 324-pin BGA package
- 128Mbyte DDR2 with 16-bit wide data
- 10/100/1000 Ethernet PHY
- on-board USB2 ports for programming and data transfer
- USB-UART and USB-HID port (for mouse/keyboard)
- Two HDMI video input ports and two HDMI output ports
- AC-97 Codec with line-in, line-out, mic, and headphone
- real-time power monitors on all power rails
- 16Mbyte x4 SPI Flash for configuration and data storage
- 100MHz CMOS oscillator
- 48 I/O's routed to expansion connectors
- GPIO includes eight LEDs, six buttons, and eight slide switches

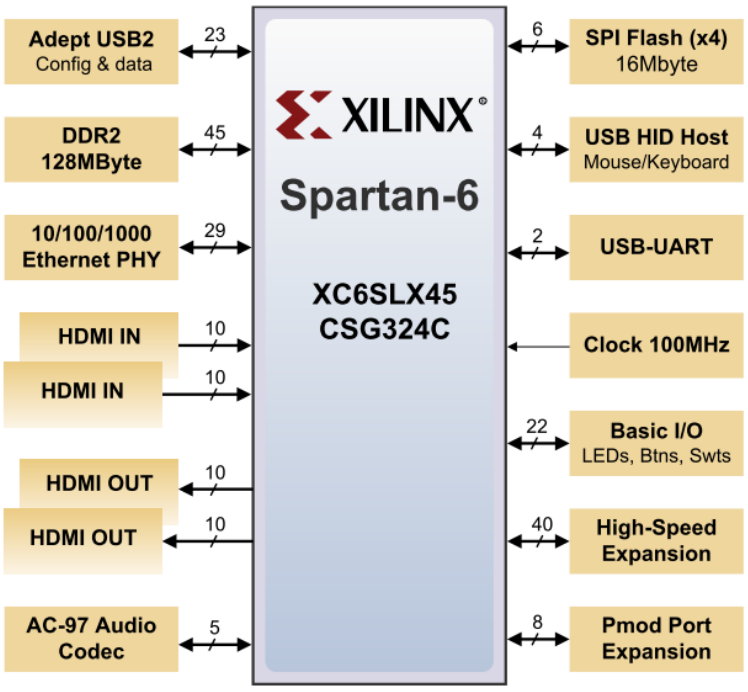

Figure 1.

FPGA Configuration

After power-on, the FPGA on the Atlys board must be configured before it can perform any functions. The FPGA can be configured in three ways:

1) A USB-connected PC can configure the board using the JTAG port any time power is on.
2) A configuration file stored in the SPI Flash ROM can be automatically transferred to the FPGA at power-on.
3) A programming file can be transferred from a USB memory stick attached to the USB HID port.

Digilent's Adept software and Xilinx's iMPACT software can be used to program the FPGA or ROM using the Adept USB port. In this lab you will perform the FPGA configuration using Adept software. In the next sections of this workshop, you will learn about the iMPACT. We'll introduce a comprehensive description and also a separate lab for iMPACT tool in the last day of this workshop.

Adept Software

The Atlys board includes Digilent's newest Adept USB2 system, which offers device programming, real-time power supply monitoring, automated board tests, virtual I/O, and simplified user-data transfer facilities. In this lab you will work with this software to connect to the Atlys board and also perform some practical tests on the board.

Adept has a simplified programming interface and many additional features as described in the following sections. The Adept port is compatible with Xilinx's iMPACT programming software. The plug-in automatically translates iMPACT-generated JTAG commands into formats compatible with the Digilent USB port, providing a seamless programming experience without leaving the Xilinx tool environment.

Required Software

To perform this manual, you must have **Xilinx ISE Design Suite 14.1** (or above) and **Adept Software** installed.

Objectives

After completing this lab, you will be able to:

- Connect and work with the Atlys board
- Work with the Adept software
- FPGA Configuration using Adept Software
- Test the peripherals and I/O
- Displayed the Real-time Current and Power

General flow of this lab

This lab comprises three steps as follows:

Figure 1. General flow of Lab5.

1. Start the Adept Software

Double click the **Adept** icon on your desktop, or select **Start > All Programs > Digilent > Adept > Adept.exe.**

Figure 2. Adept desktop icon.

2. Programming Interface (Figure 3-4)

To program the Atlys board using Adept, first set up the board and initialize the software:

2.1. Plug in and attach the power supply.

2.2. Plug in the USB cable to the PC and to the USB port on the board.

2.3. **Turn ON** Atlys' power switch.

2.4. Wait for the FPGA to be recognized.

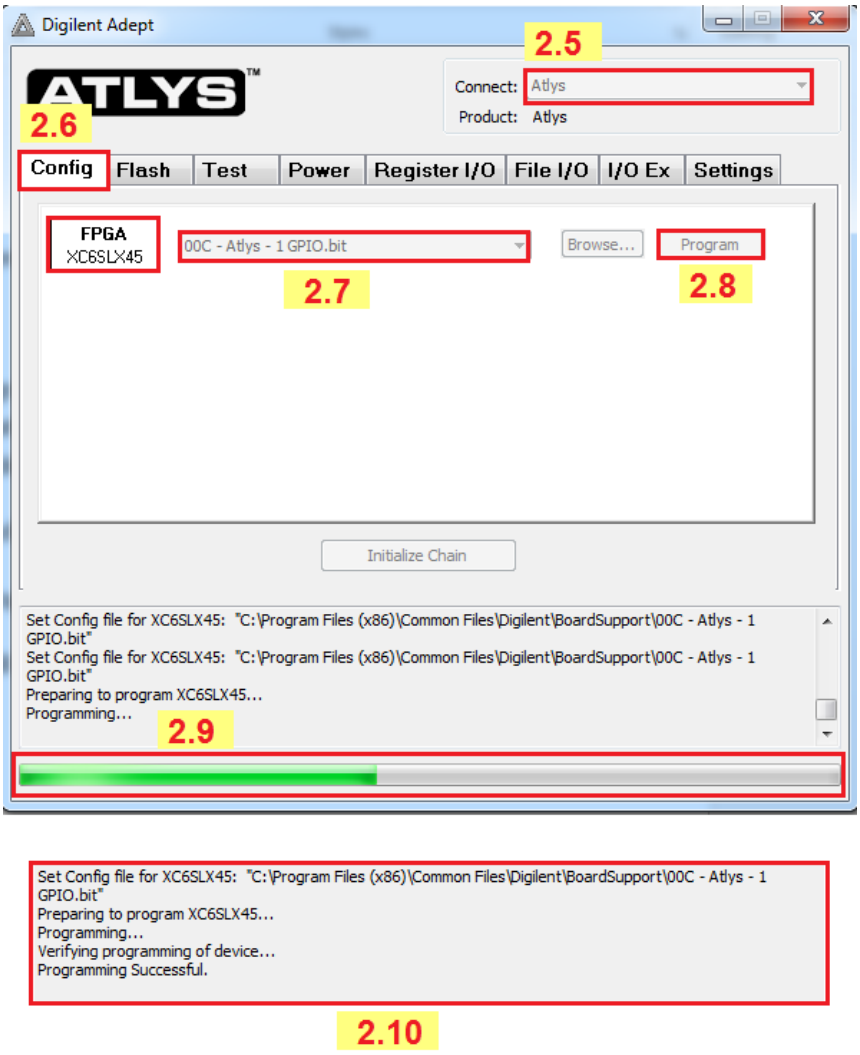

Figure 3.

2.5. Choose **Atlys** from the drop down menu.

2.6. Choose **Config** tab in the Adept software.

2.7. Choose **00C - Atlys - 1 GPIO.bit.**

NOTE: This is a general platform to program the FPGA on the Atlys board. Thus, always you can use the **Browse** button to associate the desired **.bit** file with the FPGA. But, in this part of the lab, we decide to use the pre-designed test module for Atlys.

2.8. Click on the **Program** button. The configuration file will be sent to the FPGA

2.9. The status bar and the messages will indicate whether programming was successful.

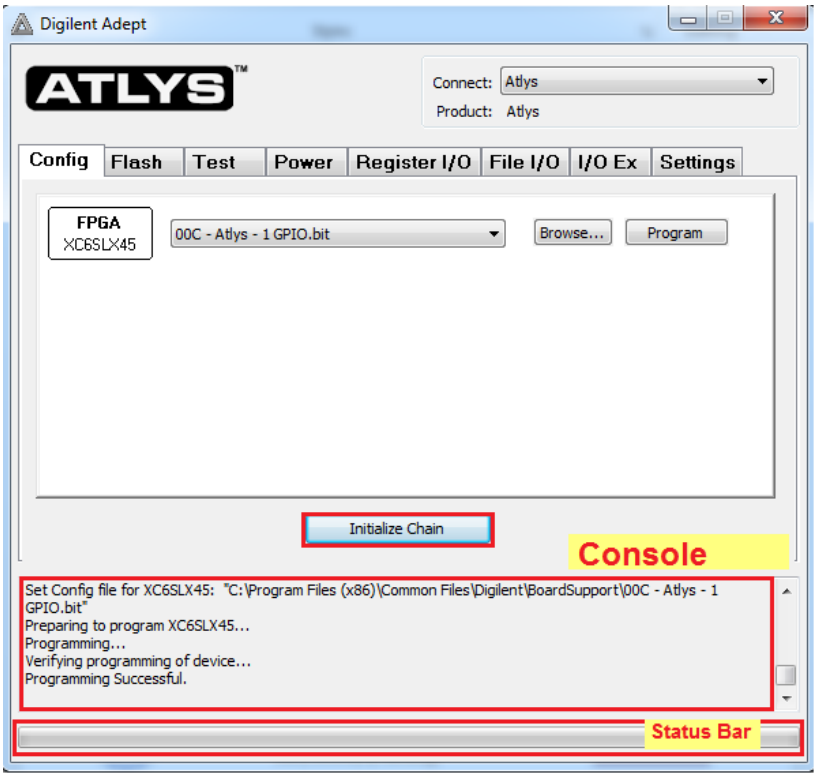

Figure 4.

2.10. The configuration **"done"** LED will light after the FPGA has been successfully configured.

Before starting the programming sequence, Adept ensures that any selected configuration file contains the correct FPGA ID code – this prevents incorrect .bit files from being sent to the FPGA.

In addition to the navigation bar and browse and program buttons, the Config interface provides an **Initialize Chain button**, **console window**, and **status bar**.

- The **Initialize Chain button** is useful if USB communications with the board have been interrupted.
- The **console window** displays current status.
- The **status bar** shows real-time progress when downloading a configuration file.

3. Test Interface (Figure 5-6)

The test interface provides an easy way to verify many of the board's hardware circuits and interfaces such as peripherals. In this case, the FPGA is configured with test and PC-communication circuits, overwriting any FPGA configuration that may have been present. Do the following steps to perform the test interface:

3.1. Choose **Test** tab in the Adept software.

3.2. Click the **Start Peripherals Test** button to initialize GPIO and user I/O testing.

3.3. Once the indicator near the Start Peripherals Test button turns green, all peripheral tests can be run. The configuration **"done"** LED will light after the FPGA has been successfully configured.

3.4. The switches and buttons graphics show the current states of those devices on the Atlys board.

3.5. Connect a speaker or a headphone to the LINE-OUT or HP-OUT audio connectors, respectively.

3.6. Press a **push button** to drive a tone out of the connected speaker/headphone.

3.7. Stop the test, by clicking **Stop Peripheral Test**.

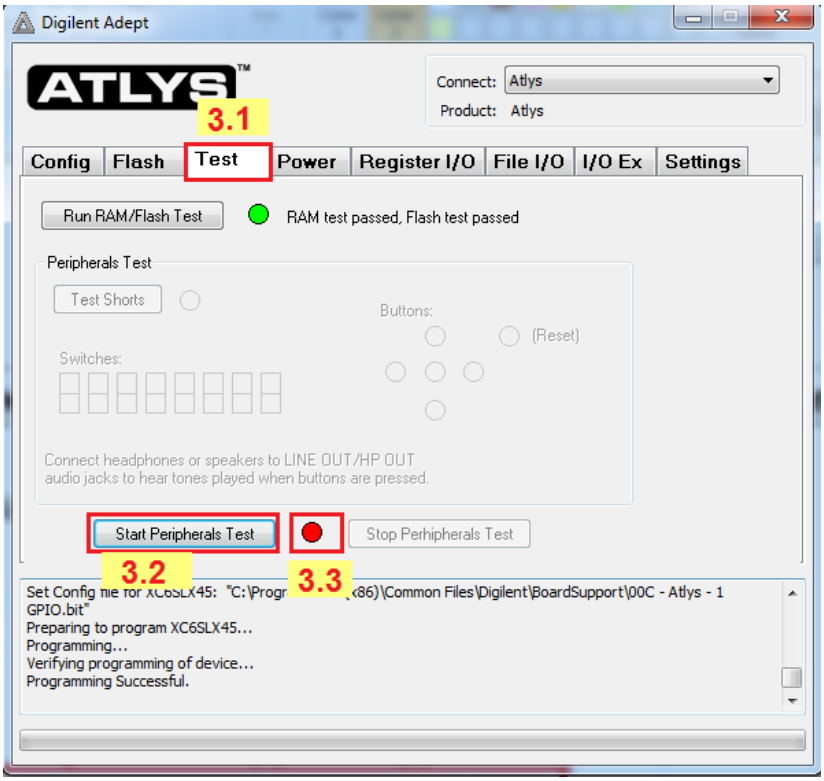

Figure 5.

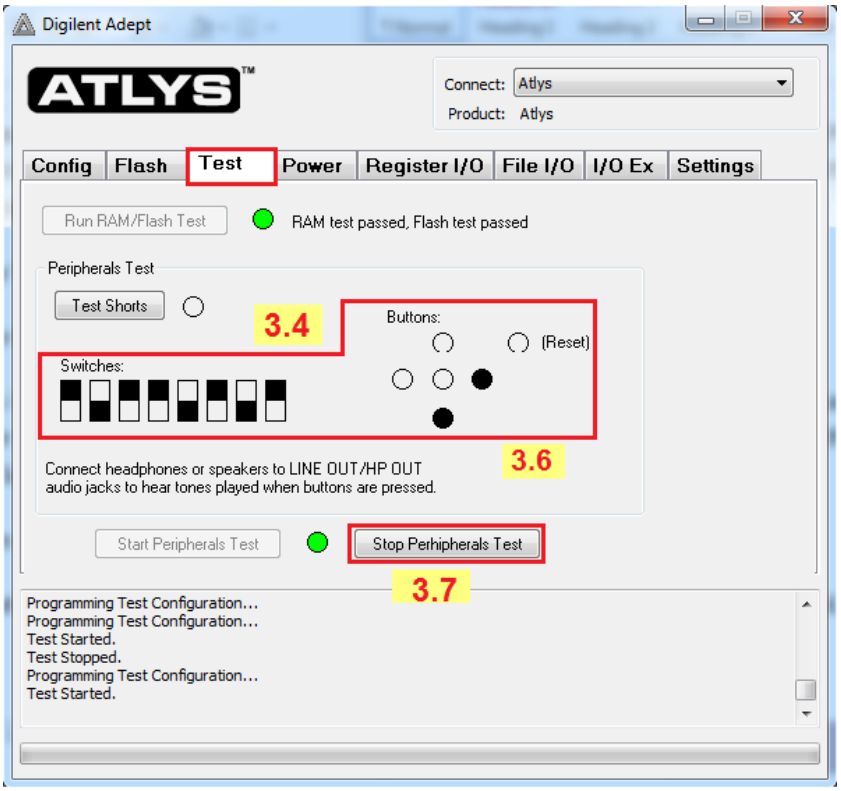

Figure 6.

4. Power (Figure 7-8)

The power application provides highly-accurate real-time current and power readings from four on-board power-supply monitors.

4.1. Choose **Test** tab in the Adept software.

4.2. Click **Start** button.

4.3. Real-time current and power data is displayed in tabular form and updated continuously when the power meter is active (or started).

4.4. Historical data is available using the Show Graph feature, which shows a graph with current data for all four power supplies for up to ten minutes.

4.5. Recorded values are also stored in a buffer that can be saved to a file for later analysis. Click **Save Buffer** to save the historical data in the buffer.

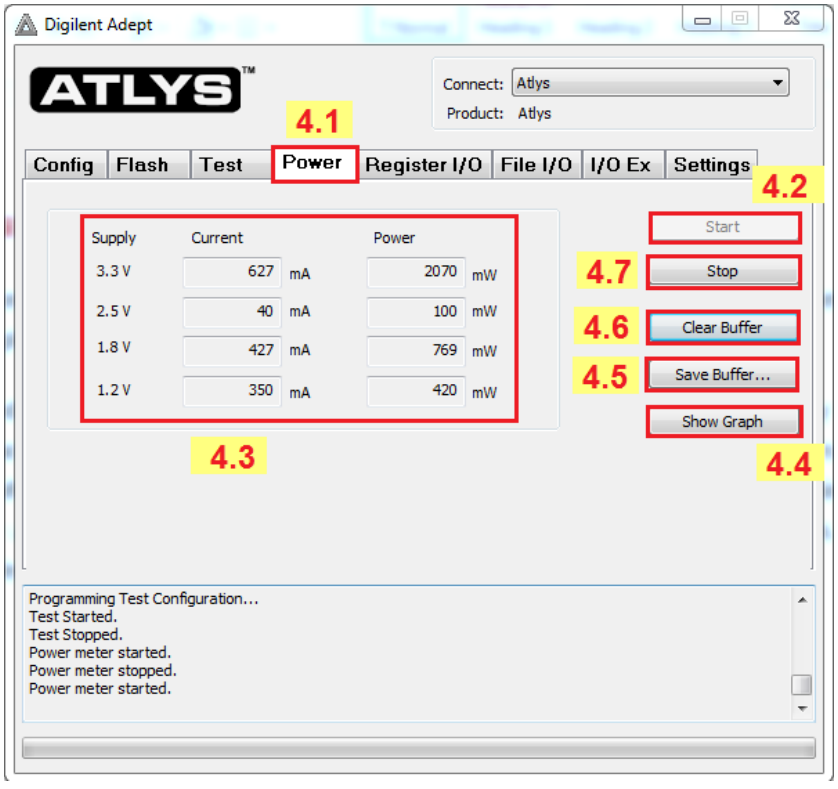

Figure 7.

Figure 8.

4.6. Click **Clear Buffer** to clear the historical data in the buffer.

4.7. Click **Stop** to finish the process.

Digital Circuit Design with Xilinx FPGA

Lab Manual

Workshop No.1

Lab6:

Design Implementation

Outline

This manual covers the following topics:

- Implementation of design via the following processes:
  - Translating the design
  - Mapping the design
  - Place and Route the design
- Review the corresponding reports for each process

Introduction

After the synthesizing the design you should enter the required constraints such as timing, synthesis, and placement constraints to your design. Entering these constraints can be done through a User Constraints File (UCF), the HDL source file, and a XST Constraint File (XCF). Moreover, it is assumed that you performed behavioral simulation on your design and the functionality of the design is confirmed. In order to synthesize, simulate, and enter the constraint to your design, please refer to the Lab1, Lab2, and Lab4, respectively. After the completion of these steps your design is ready for implementation. Design implementation is the process of translating, mapping, placing, routing, and generating a bitstream file for your design. The process of generating a bitstream is described Lab8. The design implementation tools are embedded in the Xilinx ISE Design Suite for easy access and project management. In the design implementation process, you will pass a synthesized netlist (EDN, NGC) from the front-end tool to the back-end design implementation tools. After the design implementation you can load the generated bitstream file to the target FPGA and test your design in the hardware, which are described in Lab8.

Required Software

To perform this manual, you must have **Xilinx ISE Design Suite 14.1** and **ModelSim 10.1** (or above) installed. Note that in this manual the pictures are captured in **Xilinx ISE Design Suite 14.1**, but all the procedures can be done in other versions of this software.

Objectives

After completing this lab, you will be able to

- Implement a synthesized design
- Translate, Map, and place & route the design separately and review the corresponding reports

General flow of this lab

This lab comprises three steps as follows:

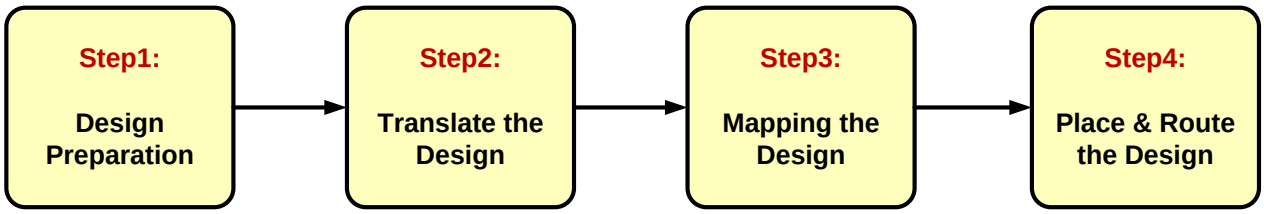

Figure 1. General flow of Lab6.

Design Description

In this lab, you'll use the completed design of Lab4, which is a constrained design with the various timing, synthesis, and placement constraints. You'll complete this design and make it ready for testing in the hardware.

1. Design Preparation

1. Start the Project Navigator

Double click the Project Navigator icon on your desktop, or select **Start > All Programs > Xilinx Design Tools > ISE Design Suite 14.1 > ISE Design Tools > Project Navigator.**

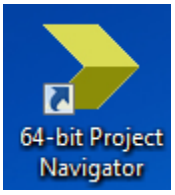

Figure 2. Project Navigator desktop icon.

2. Open the project (Figure 3)

2.1. From Project Navigator, select **File > Open Project**.

2.2. Go to the project directory of Lab4.

2.3. Select **Lab4.xise**.

2.4. Click **Open**.

3. Copy the project (Figure 4)

3.1. From Project Navigator, select **File > Copy Project**.

3.2. Specify the location, in which you want to save the result of this lab.

3.3. Enter **Lab6** as the name of the copied project.

3.4. Select **Copy sources to the new location**.

3.5. Select the **Open the copied project** to open the Lab6 project automatically.

3.6. Click **OK**.

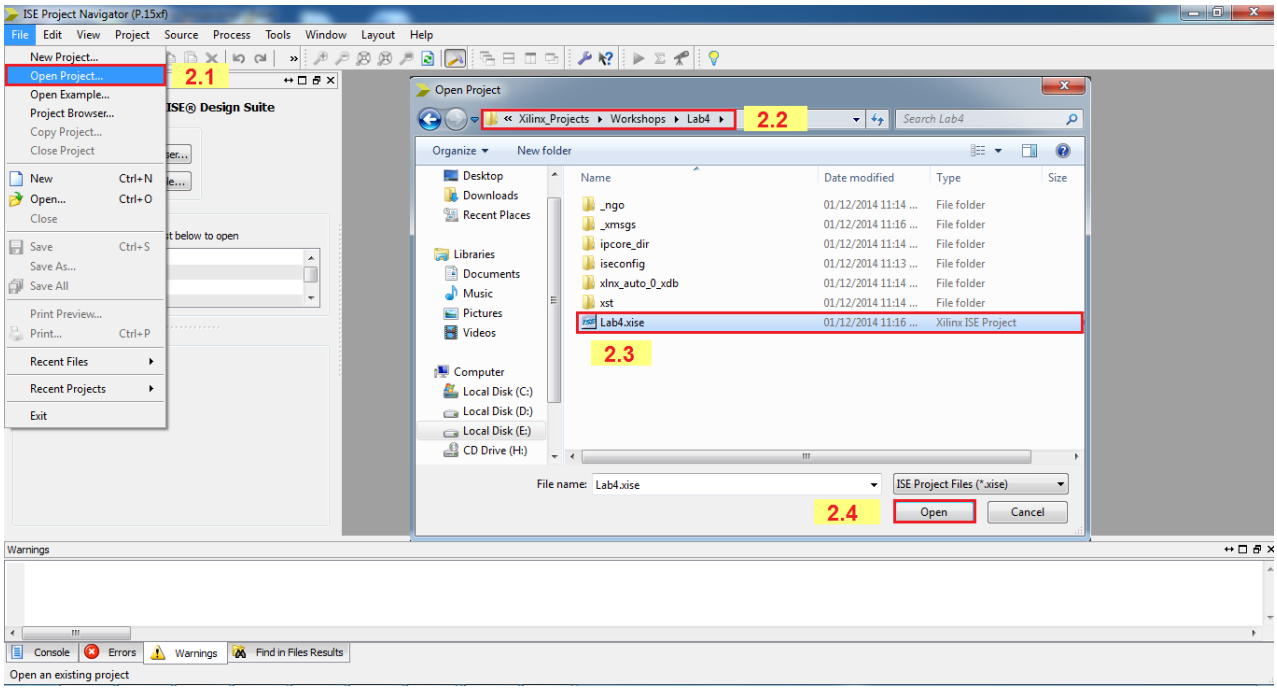

Figure 3.

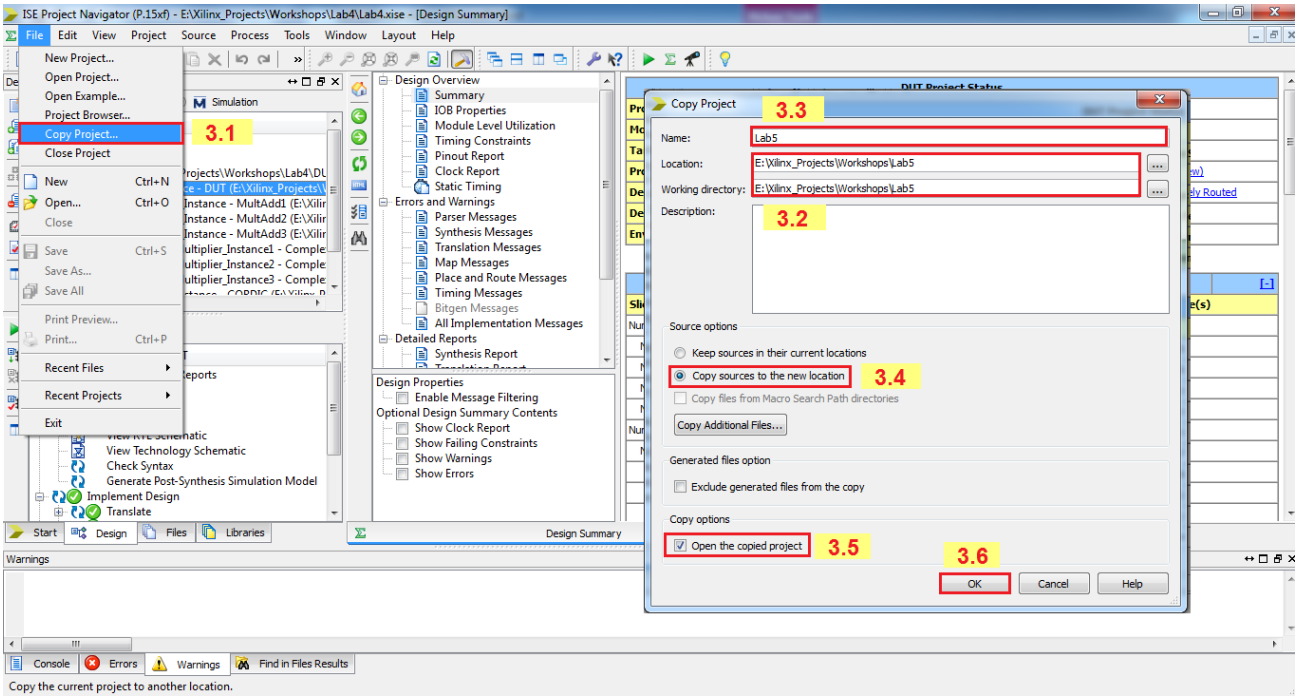

Figure 4.

2. Translate (Figure 5-8)

The first step of design implementation is translating the design. During translation, the NGDBuild program performs the following functions:

- Converts input design netlists and writes results to a single merged NGD netlist. The merged netlist describes the logic in the design as well as any location and timing constraints.
- Performs timing specification and logical design rule checks.
- Adds constraints from the User Constraints File (UCF) to the merged netlist.

Perform the following steps to Translate the design:

2.1. Select **Design Panel**.

2.2. Set the Design View to the **Implementation**.

2.3. In the Hierarchy pane, select the **top module** (i.e. **DUT.v**).

2.4. In the Processes pane, expand **Implementation**.

2.5. Right-click **Translate** and select **Process Properties**.

2.6. In the Property display level, select **Advanced**.

2.7. Specify the **Translate Properties**.

2.8. Click **OK**.

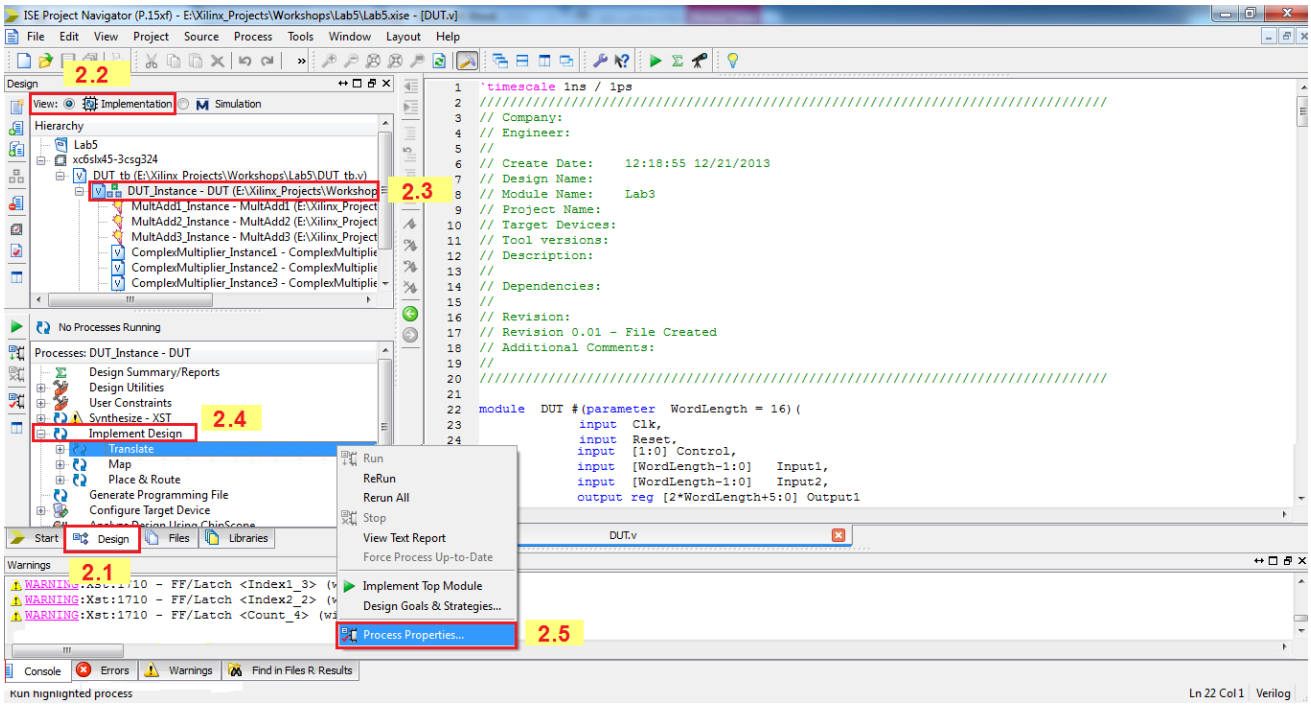

Figure 5.

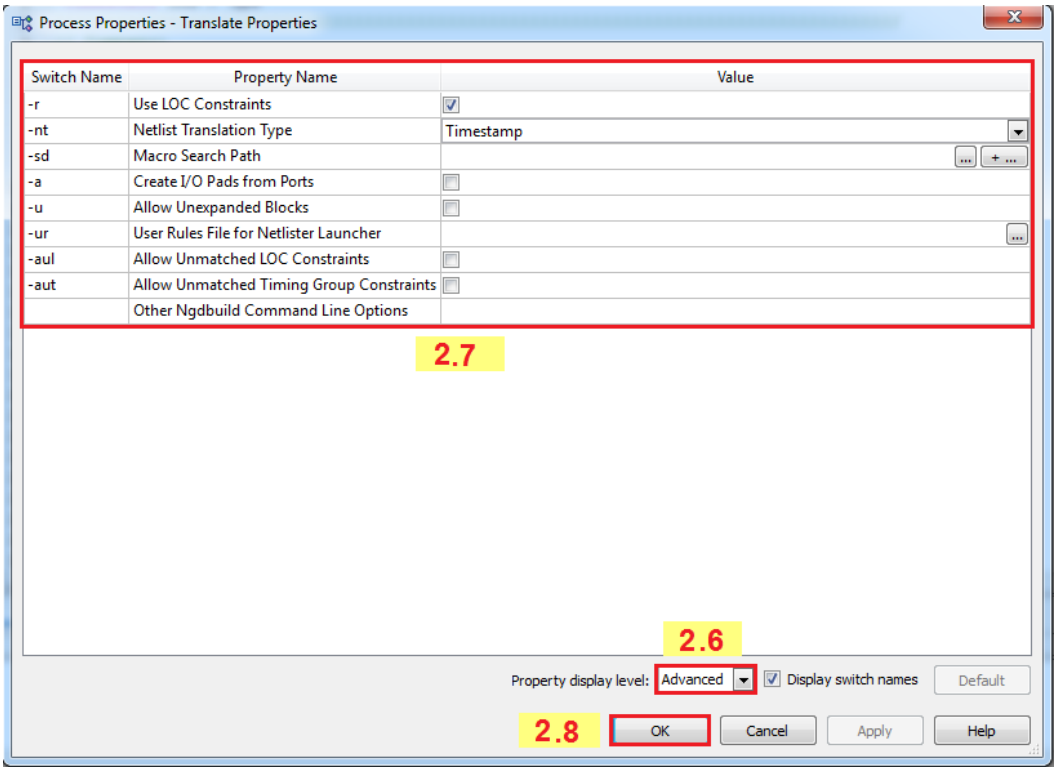

Figure 6.

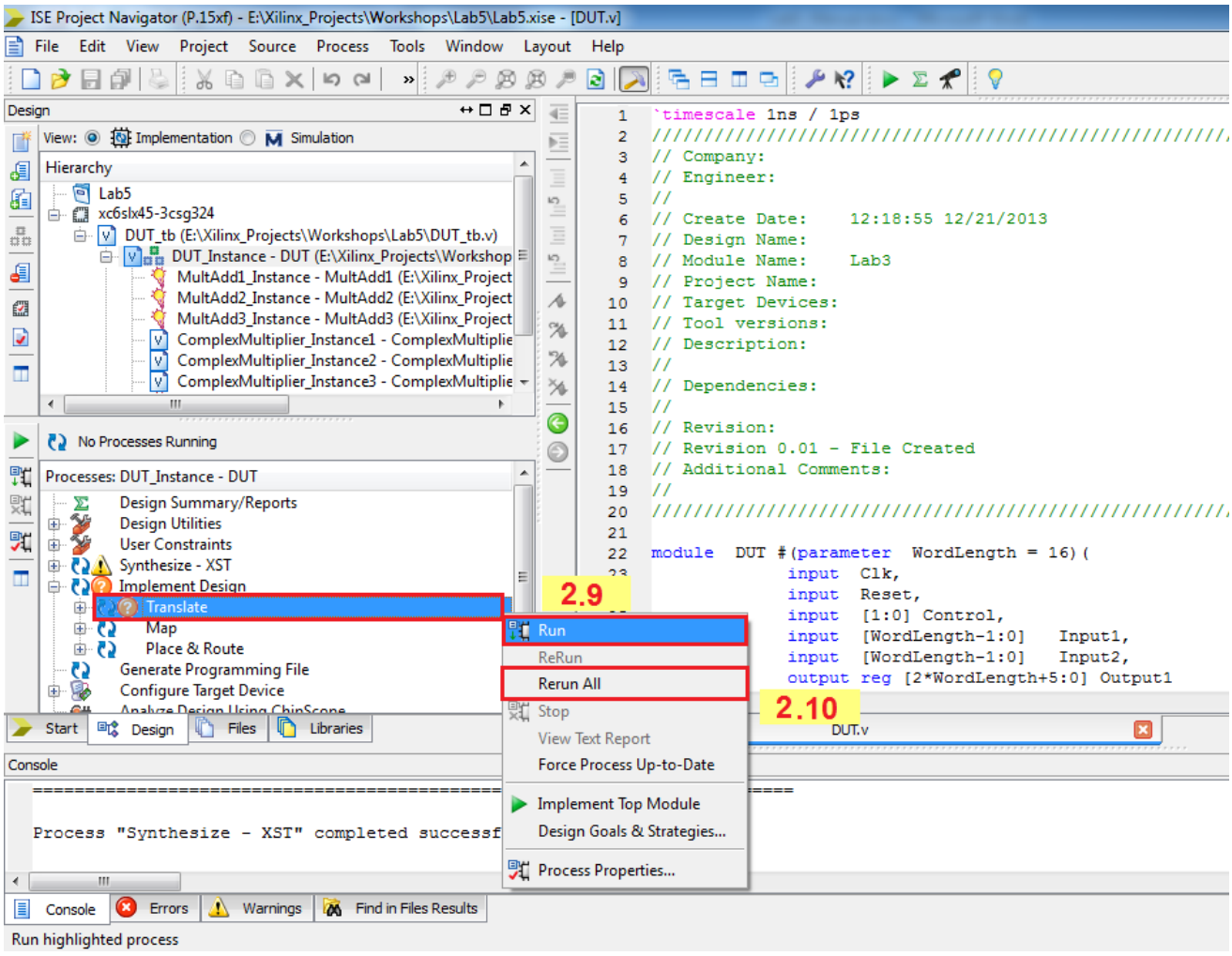

Figure 7.

2.9. Right-click on **Translate** and select **Run**.

2.10. If you modify any of the source files, you should perform synthesis before the design translation. So in this case, select **Rerun All**.

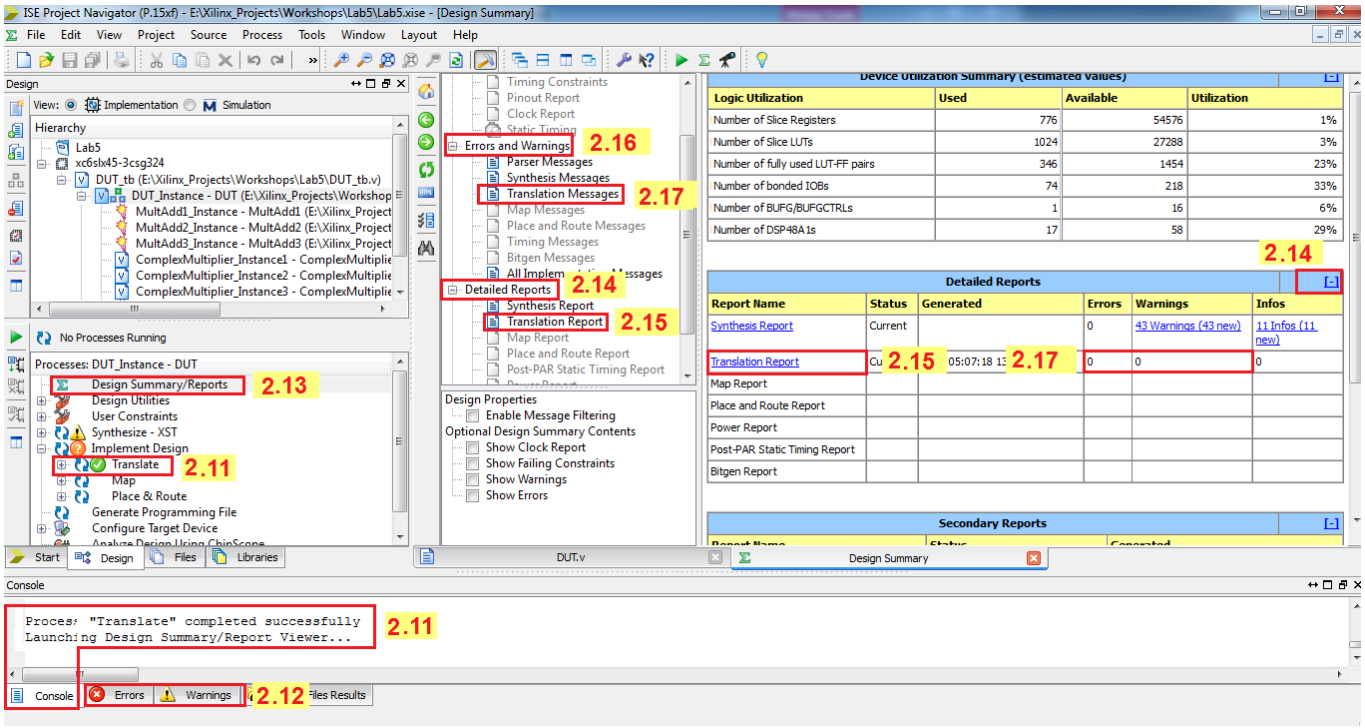

Figure 8.

2.11. In the Transcript Window, check the **Console** tab to view the result of this process. Also, you can check the status of process result in the Processes pane.

2.12. If the Translate process is not successful, read the content of Errors/Warnings tabs in the Transcript Window to check the Errors/Warnings of Translate process.

2.13. In the Processes pane, double-click **Design Summary/Reports**.

2.14. In the Design Summary page, expand **Detailed Report**.

2.15. Select **Translation Report**.

2.16. In order to read the Errors/Warnings of Translate process expand **Errors and Warnings**.

2.17. Select **Translation Messages**.

3. Map (Figure 9-12)

The second step of design implementation is mapping the design. In the Map process, the design is mapped into CLBs and IOBs. Map process performs the following functions:

- Allocates CLB and IOB resources for all basic logic elements in the design.
- Processes all location and timing constraints, performs target device optimizations, and runs a design rule check on the resulting mapped netlist.

In order to perform the Map process, do the following:

3.1. Select **Design Panel**.

3.2. Set the Design View to the **Implementation**.

3.3. In the Hierarchy pane, select the **top module** (i.e. **DUT.v**).

3.4. In the Processes pane, expand **Implementation**.

3.5. Right-click **Map** and select **Process Properties**.

3.6. In the Property display level, select **Advanced**.

3.7. Specify the **Map Properties**.

3.8. Click **OK**.

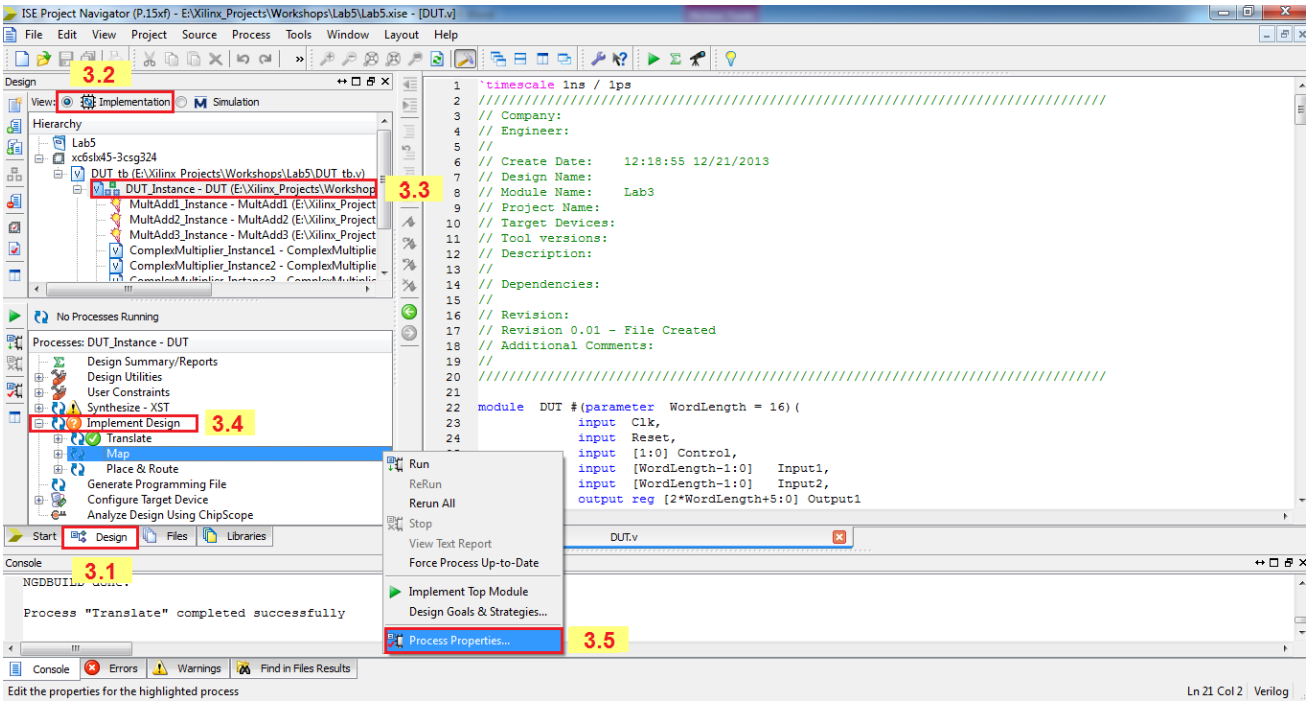

Figure 9.

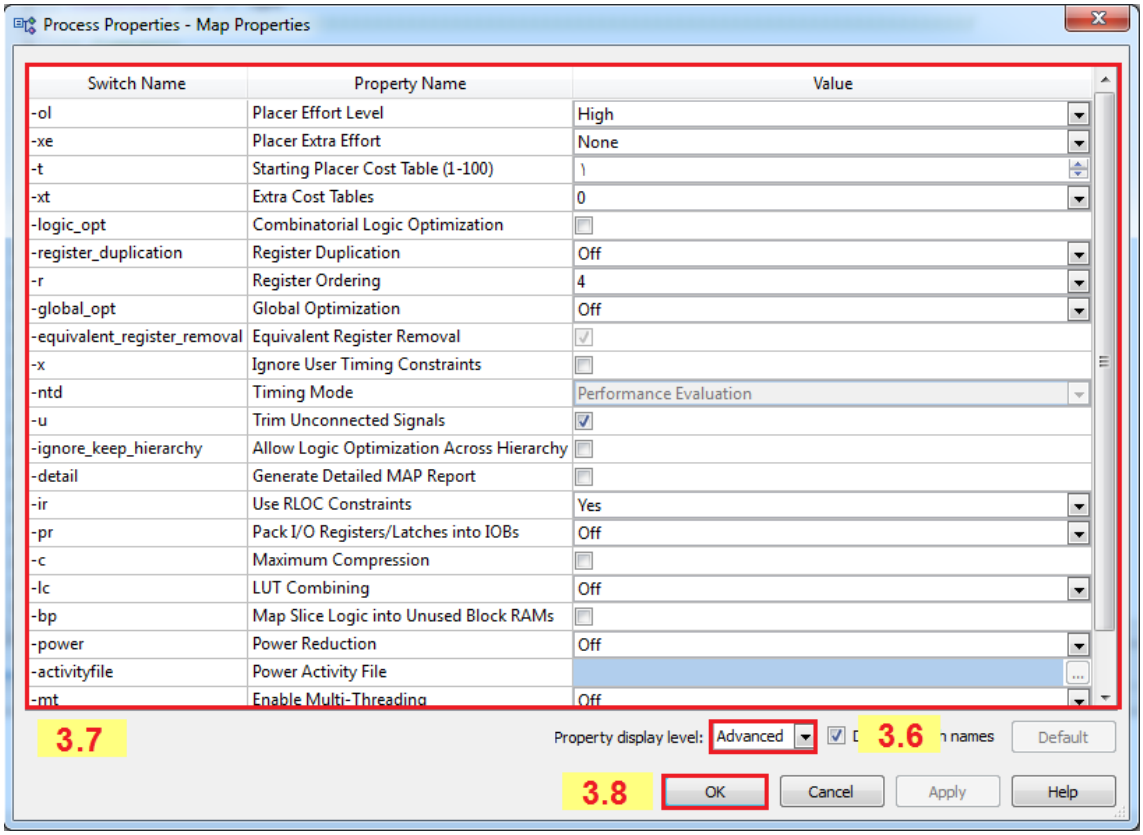

Figure 10.

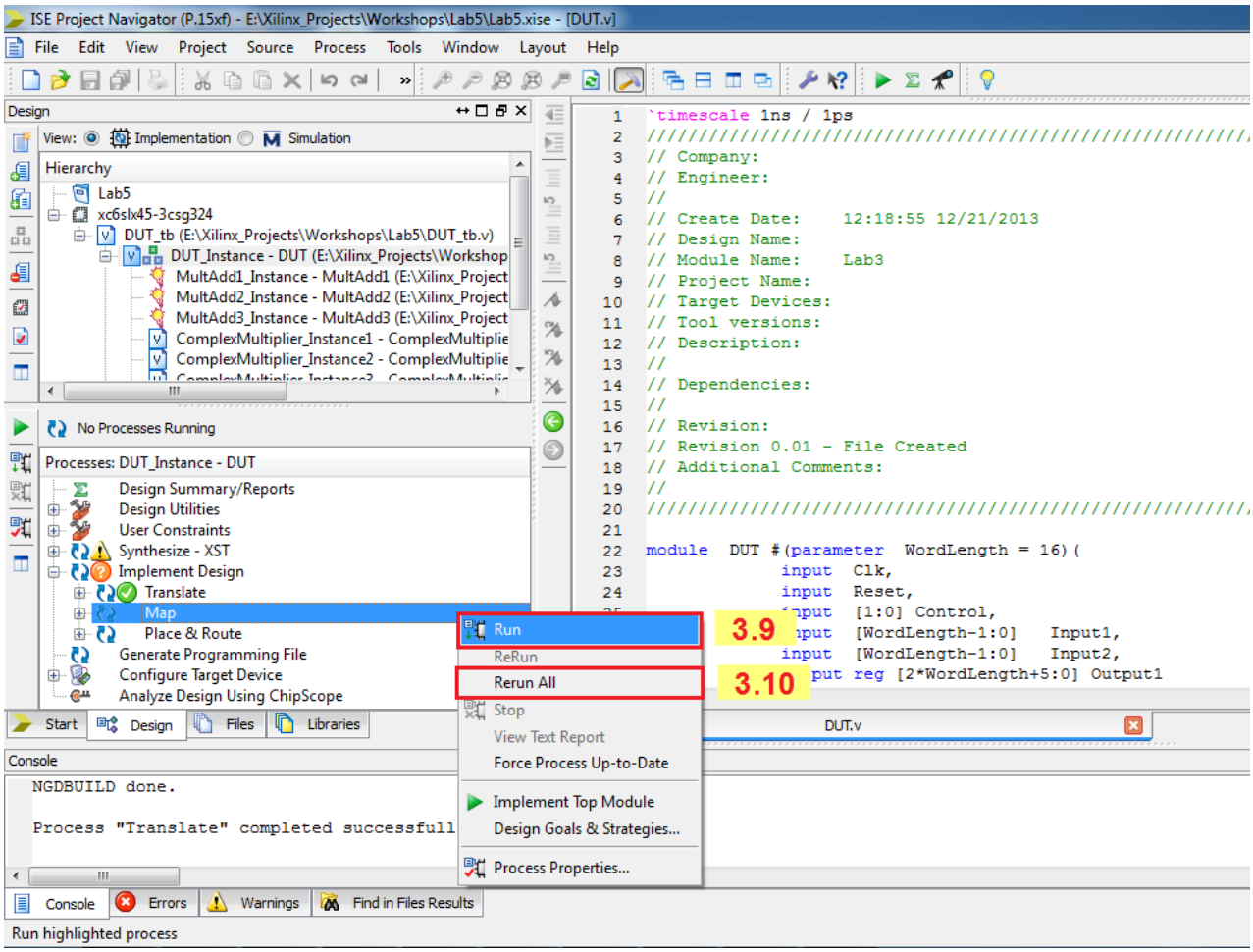

Figure 11.

3.9. Right-click on **Map** and select **Run**.

3.10. If you modify any of the source files, you should perform synthesis and translation before mapping the design. So in this case, select **Rerun All**.

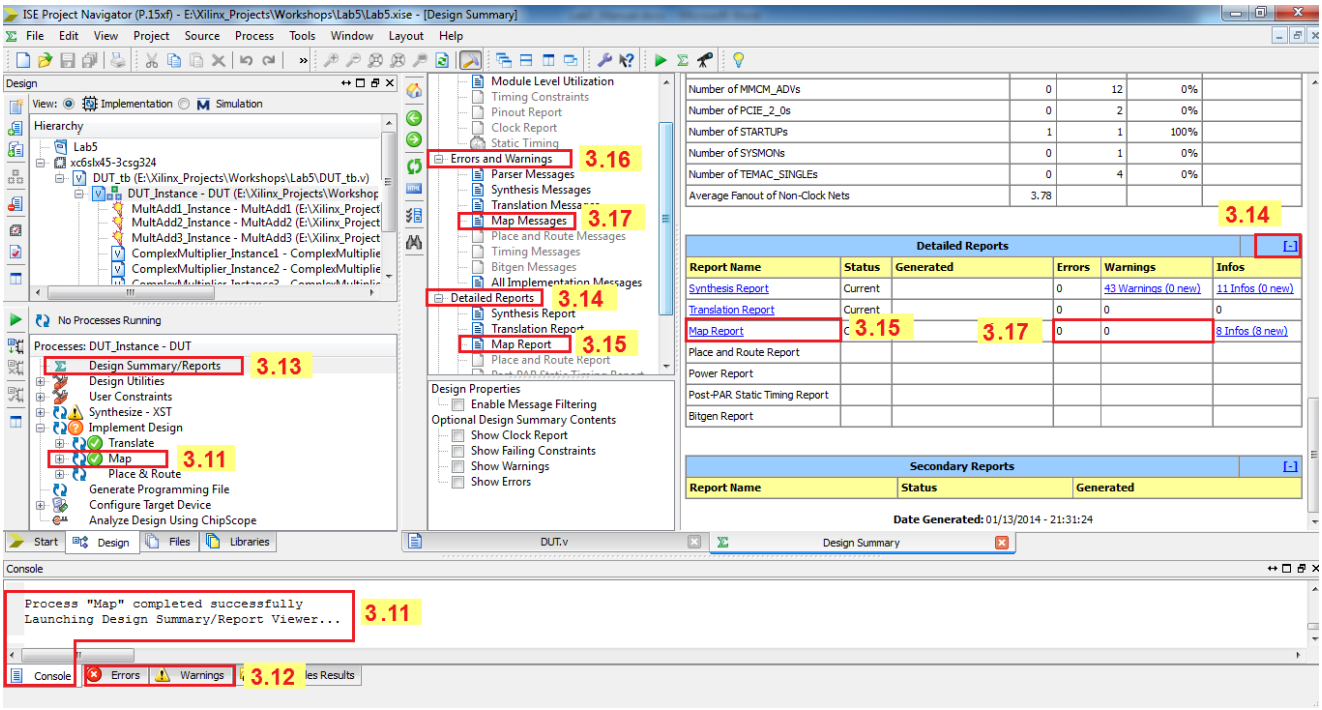

Figure 12.

3.11. In the Transcript Window, check the **Console** tab to view the result of this process. Also, you can check the status of process result in the Processes pane.

3.12. If the Map process is not successful, read the content of Errors/Warnings tabs in the Transcript Window to check the Errors/Warnings of Map process.

3.13. In the Processes pane, double-click **Design Summary/Reports**.

3.14. In the Design Summary page, expand **Detailed Report**.

3.15. Select **Map Report.**

3.16. In order to read the Errors/Warnings of Map process expand **Errors and Warnings**.

3.17. Select **Map Messages**.

4. Place & Route (Figure 13-16)

The third step of design implementation is Place and Route. After mapping the design, the design can be placed and routed. In the Place and Route process one of the two place-and-route algorithms is performed during the Place and Route (PAR) process:

- **Timing-Driven PAR**
  - PAR is run with the timing constraints specified in the input netlist, the constraints file, or both.
- **Non-Timing-Driven PAR**
  - PAR is run, ignoring all timing constraints.

According to the fact that you defined timing constraints earlier in Lab4 for this design, the Place and Route (PAR) process performs timing-driven placement and routing for this design.

Perform the following steps to Place and Route the design:

4.1. Select **Design Panel**.

4.2. Set the Design View to the **Implementation**.

4.3. In the Hierarchy pane, select the **top module** (i.e. **DUT.v**).

4.4. In the Processes pane, expand **Implementation**.

4.5. Right-click **Place & Route** and select **Process Properties**.

4.6. In the Property display level, select **Advanced**.

4.7. Specify the **Place & Route Properties**.

4.8. Click **OK**.

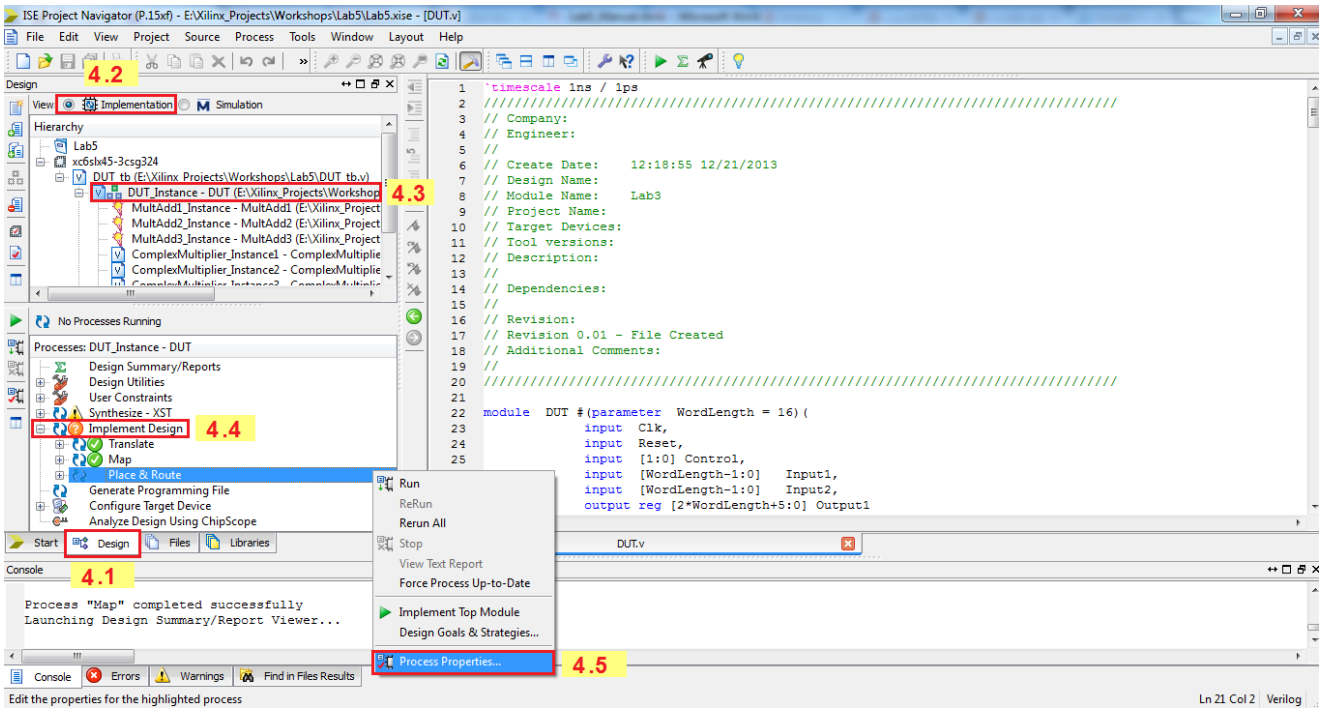

Figure 13.

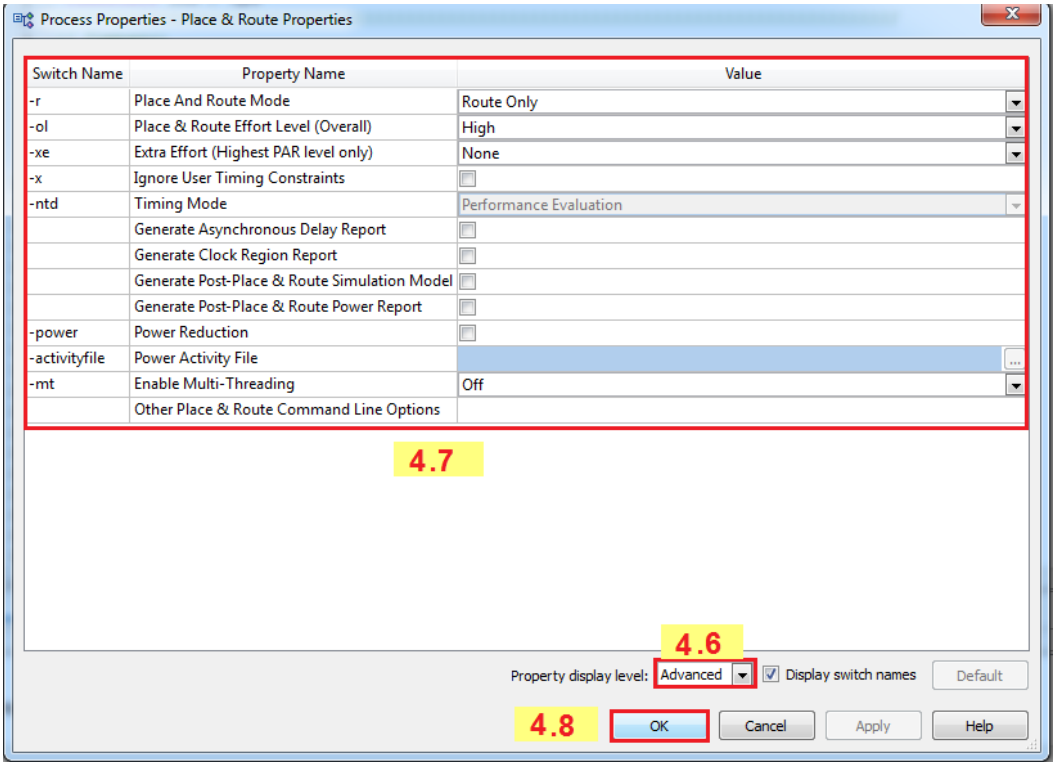

Figure 14.

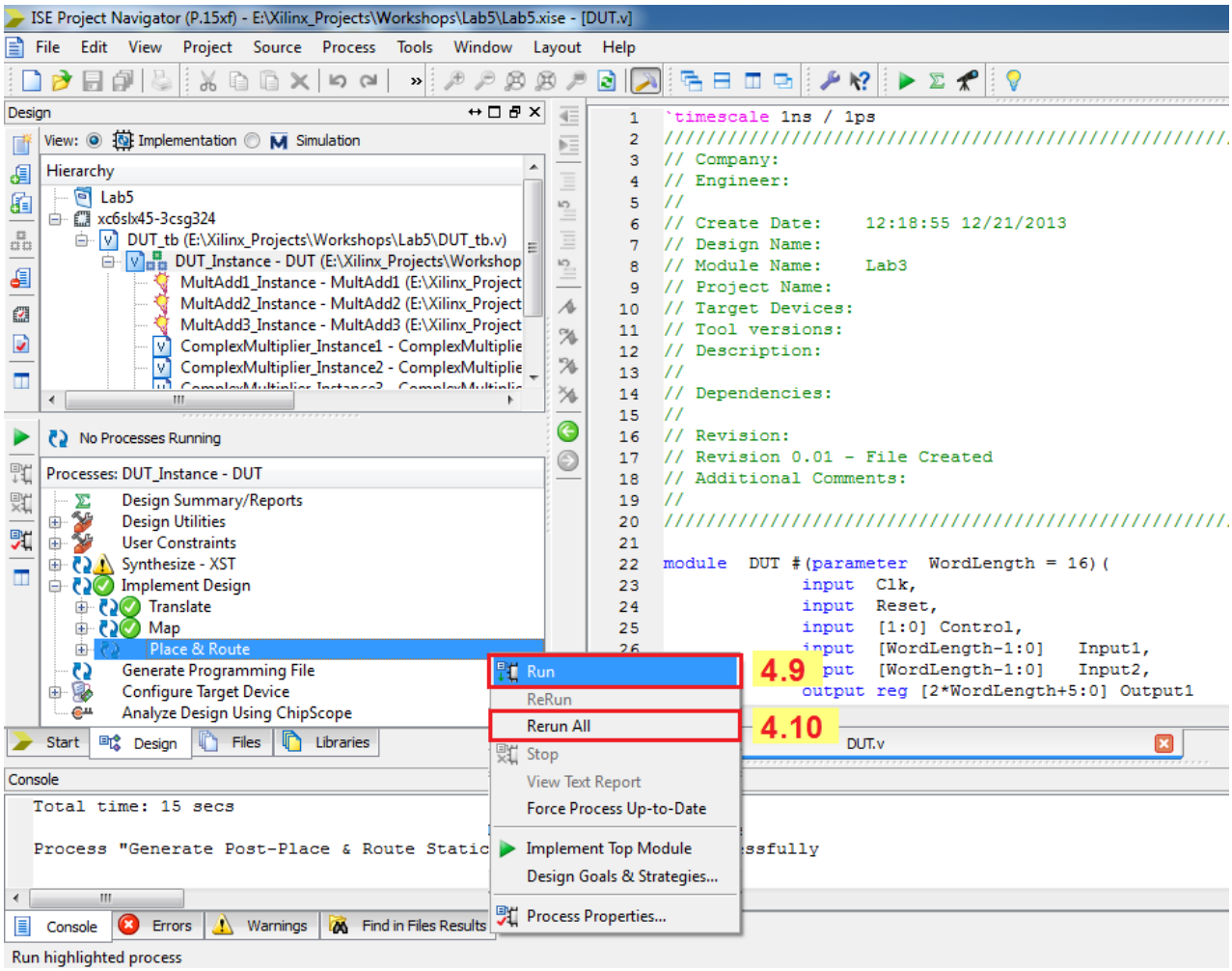

Figure 15.

4.9. Right-click on **Place & Route** and select **Run**.

4.10. If you modify any of the source files, you should perform synthesis, translation, and mapping before place & route the design. So in this case, select **Rerun All**.

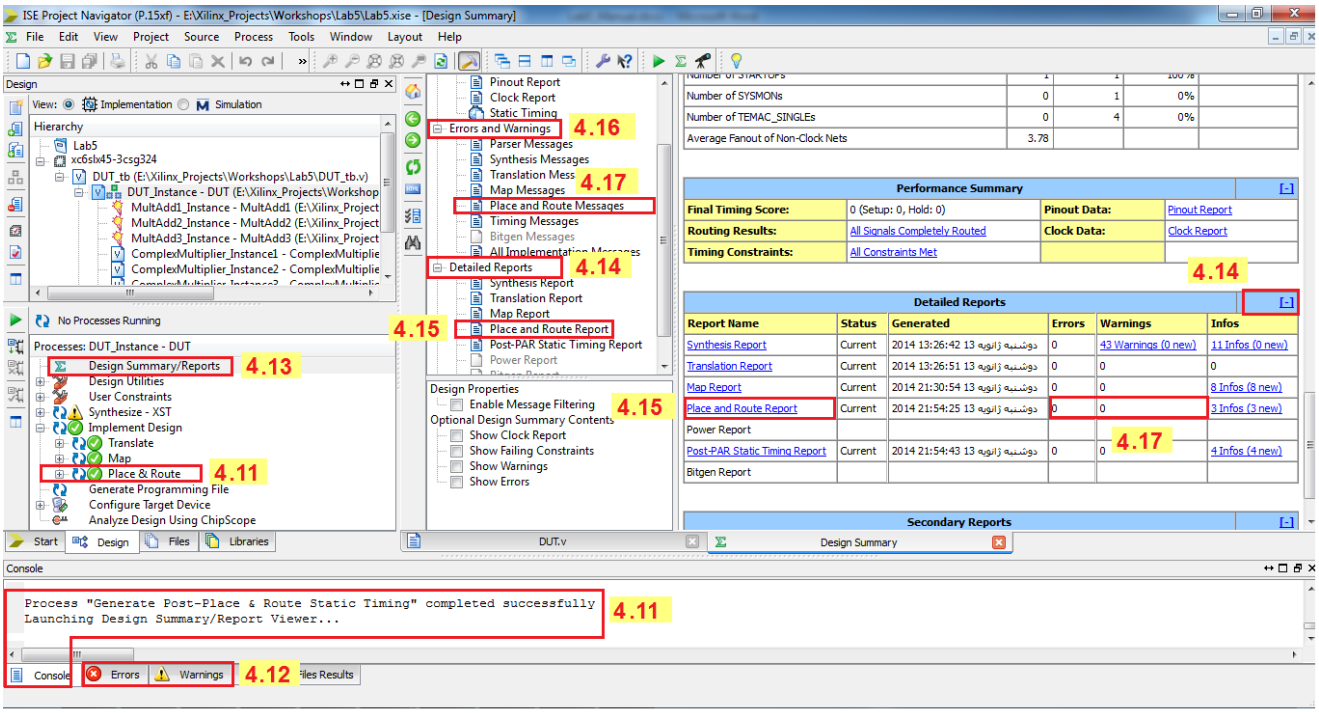

Figure 16.

4.11. In the Transcript Window, check the **Console** tab to view the result of this process. Also, you can check the status of process result in the Processes pane.

4.12. If the place & route process is not successful, read the content of Errors/Warnings tabs in the Transcript Window to check the Errors/Warnings of place & route process.

4.13. In the Processes pane, double-click **Design Summary/Reports**.

4.14. In the Design Summary page, expand **Detailed Report**.

4.15. Select **Place & Route Report.**

4.16. In order to read the Errors/Warnings of place & route process expand **Errors and Warnings**.

4.17. Select **Place & Route Messages**.

Digital Circuit Design with Xilinx FPGA

Lab Manual

Workshop No.1

Lab7:

Post-implementation Simulations

Outline

This manual covers the following topics:

- Generating post-Translate simulation model
- Performing post-Translate simulation
- Generating post-Map simulation model
- Performing post-Map simulation
- Generating post-Place & Rout simulation model
- Performing post- Place & Rout simulation

Introduction

After design creation and synthesizing the design (Lab1), a behavioral simulation should be done on the design to verify its functionality (Lab2). Then, various constraints such as timing, placement, and synthesis constraints can be entered to the design (Lab4). After these steps, the implementation process, which consists of Translate, Map, and Place and Route processes should be performed on the design (Lab6). After each step of the implementation process, a simulation model can be generated, which is used in the corresponding simulation. For example, after the Translate process a Post-Translate Simulation Model can be generated to perform Post-Translate Simulation for the design. In this lab, you'll learn the method of generating these simulation models and also perform the corresponding simulation for the design.

Required Software

To perform this manual, you must have **Xilinx ISE Design Suite** and **ModelSim 10.1** (or above) installed. Note that in this manual the pictures are captured in **Xilinx ISE Design Suite 14.1**, but all the procedures can be done in other versions of this software.

Objectives

After completing this lab, you will be able to

- Perform post-Translate simulation
- Perform post-Map simulation
- Perform post-Rout simulation

General flow of this lab

This lab comprises three steps as follows:

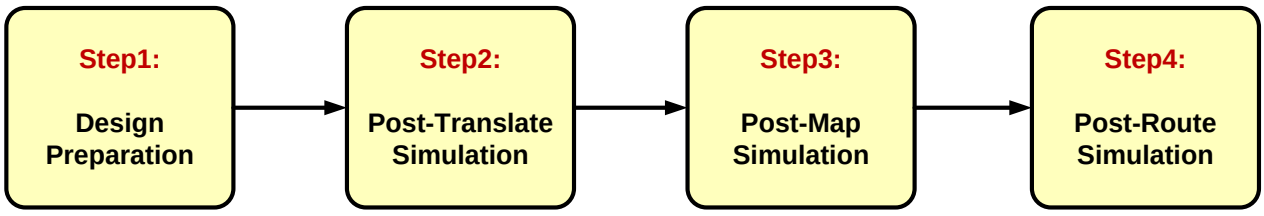

Figure 1. General flow of Lab7.

Design Description

In this lab, you'll use the completed design of Lab6, which is implemented completely. In this lab, you'll perform various simulations on this design.

1. Design Preparation

1.1. Start the Project Navigator

Double click the Project Navigator icon on your desktop, or select **Start > All Programs > Xilinx Design Tools > ISE Design Suite 14.1 > ISE Design Tools > Project Navigator.**

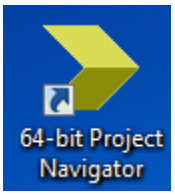

Figure 2. Project Navigator desktop icon.

1.2. Open the project (Figure 3)

1.2.1. From Project Navigator, select **File > Open Project**.

1.2.2. Go to the project directory of Lab6.

1.2.3. Select **Lab6.xise**.

1.2.4. Click **Open**.

1.3. Copy the project (Figure 4)

1.3.1. From Project Navigator, select **File > Copy Project**.

1.3.2. Specify the location, in which you want to save the result of this lab.

1.3.3. Enter **Lab7** as the name of the copied project.

1.3.4. Select **Copy sources to the new location**.

1.3.5. Select the **Open the copied project** to open the Lab7 project automatically.

1.3.6. Click **OK**.

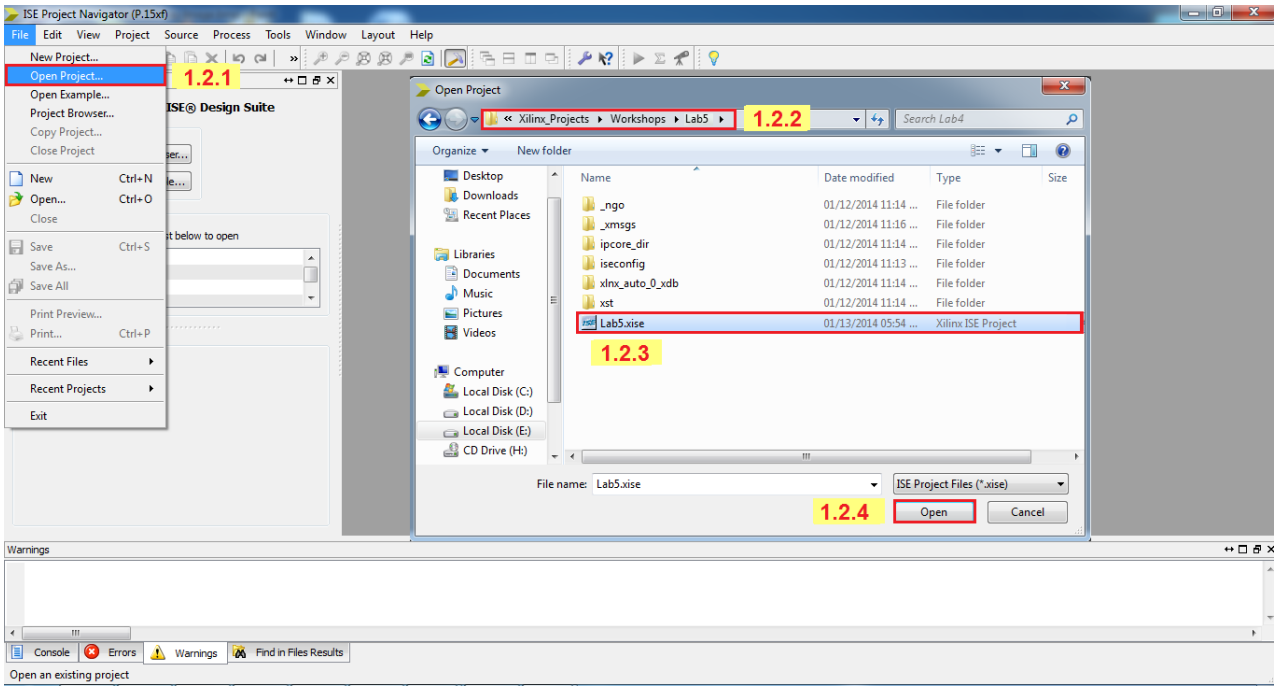

Figure 3.

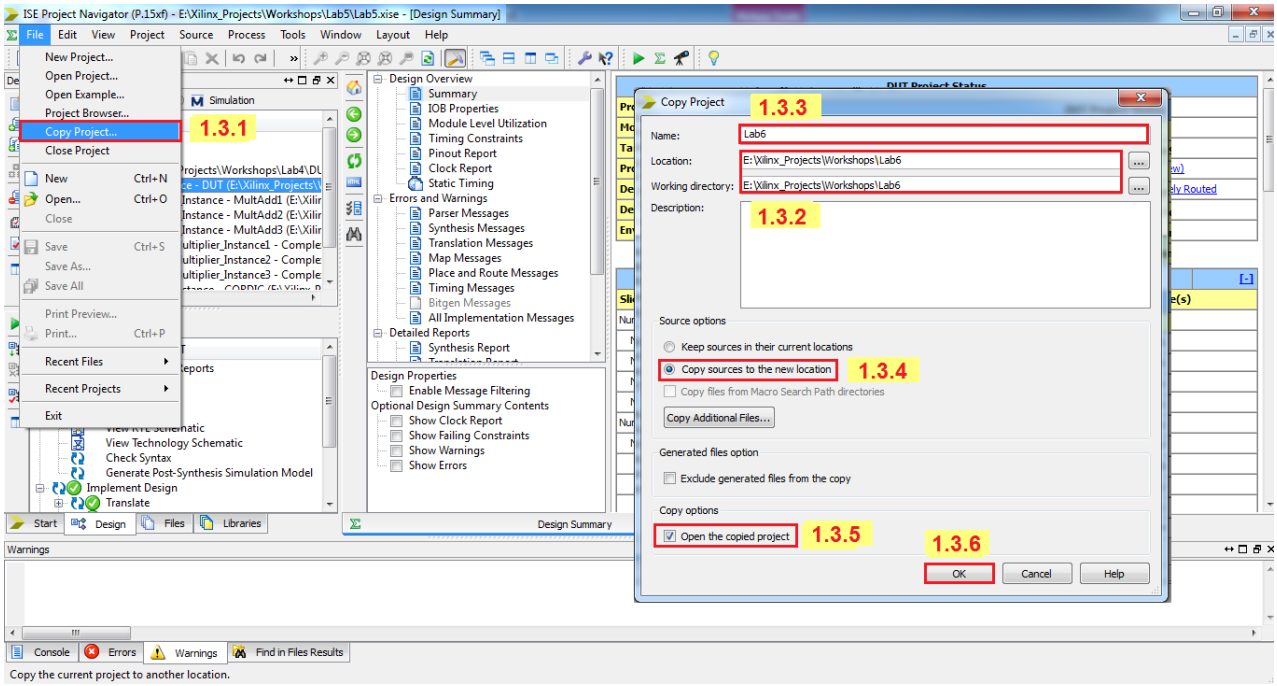

Figure 4.

2. Post-Translate Simulation

Post-Translate simulation can be done after the design translation. In order to perform post-Translate simulation for the project, you should generate a post-translate simulation model and then walk through the simulation. In this section all of these three steps are described.

2.1. Translate

The first step for the post-Translate simulation is design translation, which is described in detail in step 2 of Lab6. Thus, please refer to Lab6 for more details. Perform Translate process completely and then you can continue step2 of Lab7.

2.2. Generate Post-Translate Simulation Model (Figure 5-7)

2.2.1. Select **Design Panel**.

2.2.2. Set the Design View to the **Implementation**.

2.2.3. In the Hierarchy pane, select the **top module** (i.e. **DUT.v**).

2.2.4. In the Processes pane, expand **Implementation**.

2.2.5. In the Processes pane, expand **Translate**.

2.2.6. In the Processes pane, right-click **Generate Post-Translate Simulation Model** and select **Process Properties**.

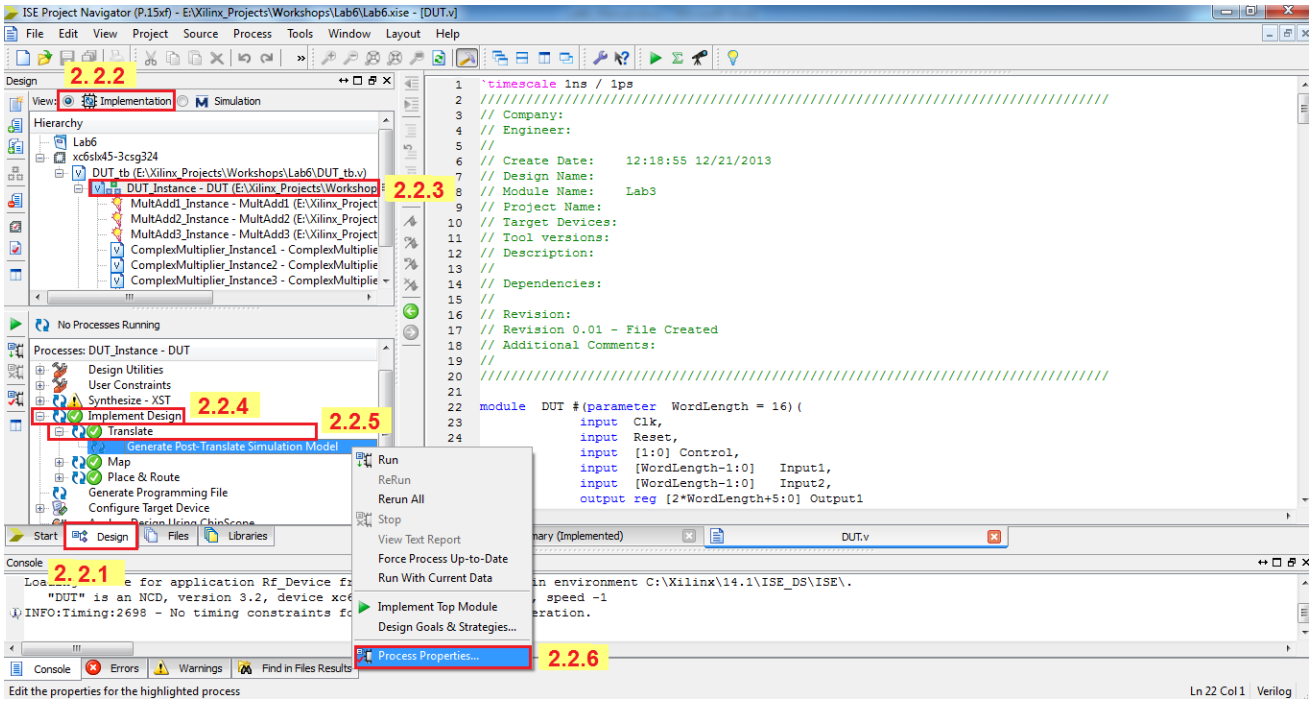

Figure 5.

2.2.7. In the Property display level, select **Advanced**.

2.2.8. Specify the **Simulation Model Properties**.

2.2.9. Click **OK**.

2.2.10. In the Processes pane, right-click **Generate Post-Translate Simulation Model** and select **Run.**

2.2.11. In the Transcript Window, check the **Console** tab to view the result of this process. Also, you can check the status of process result in the Processes pane.

2.2.12. If this process is not successful, read the content of Errors/Warnings tabs in the Transcript Window to check the Errors/Warnings of Translate process.

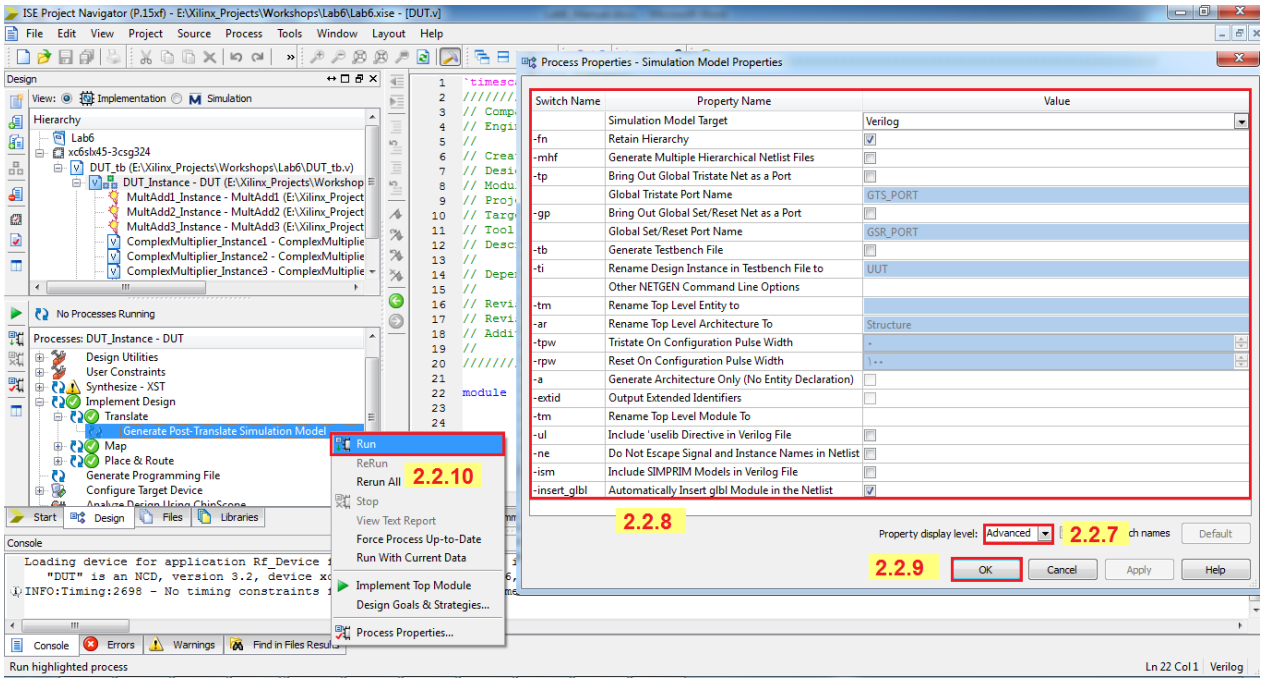

Figure 6.

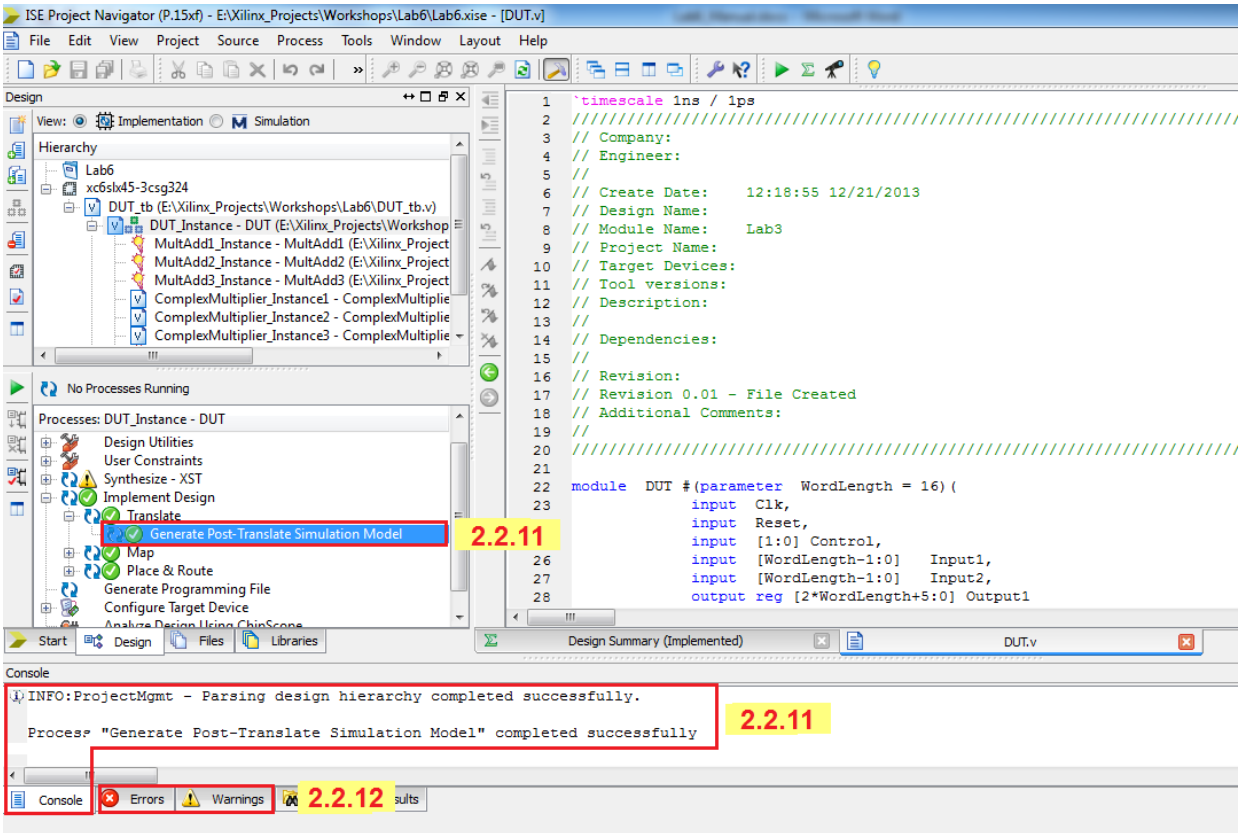

Figure 7.

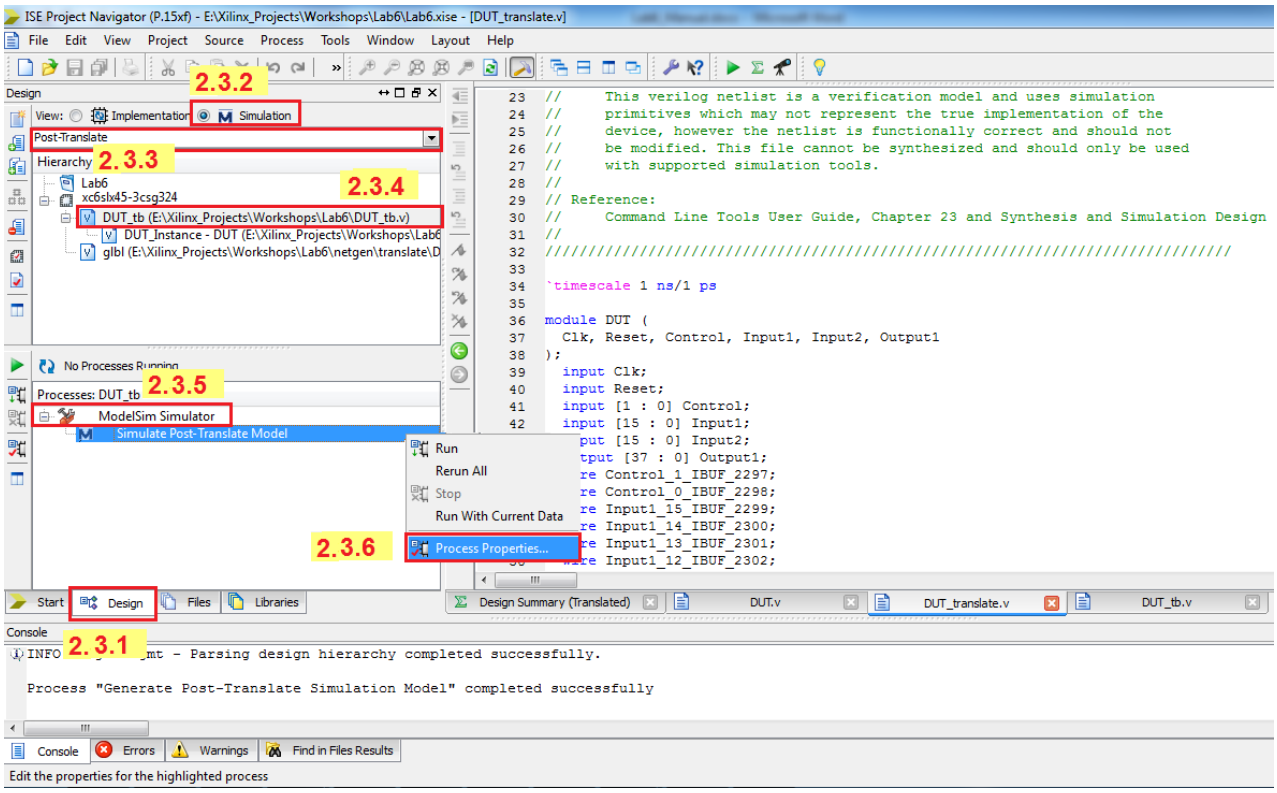

Figure 8.

2.3. Perform the Post-Translate Simulation (Figure 8-10)

2.3.1. Select **Design Panel**.

2.3.2. Set the Design View to the **Simulation**.

2.3.3. In the drop down menu of Design panel, select **Post-Translate**.

2.3.4. In the Hierarchy pane, select the **test bench** (i.e. **DUT_tb.v**).

2.3.5. In the Processes pane, expand **ModelSim Simulator**.

2.3.6. In the Processes pane, right-click **Simulate Post-Translate Model** and select **Process Properties**.

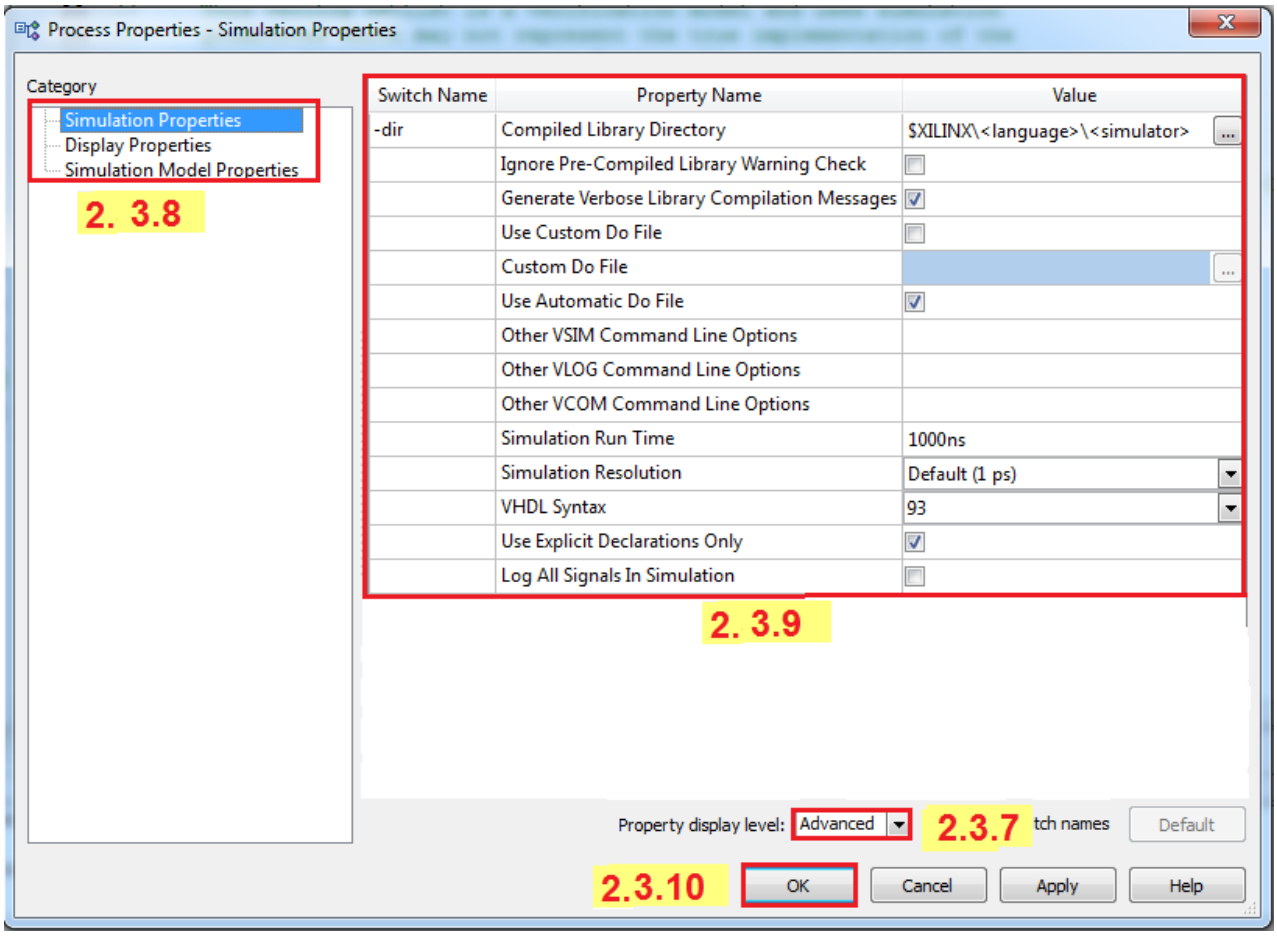

Figure 9.

2.3.7. In the Property display level, select **Advanced**.

2.3.8. Select the desired **category** among the following options:

- Simulation Properties
- Display Properties
- Simulation Model Properties

2.3.9. Specify the **Properties**.

2.3.10. Click **OK**.

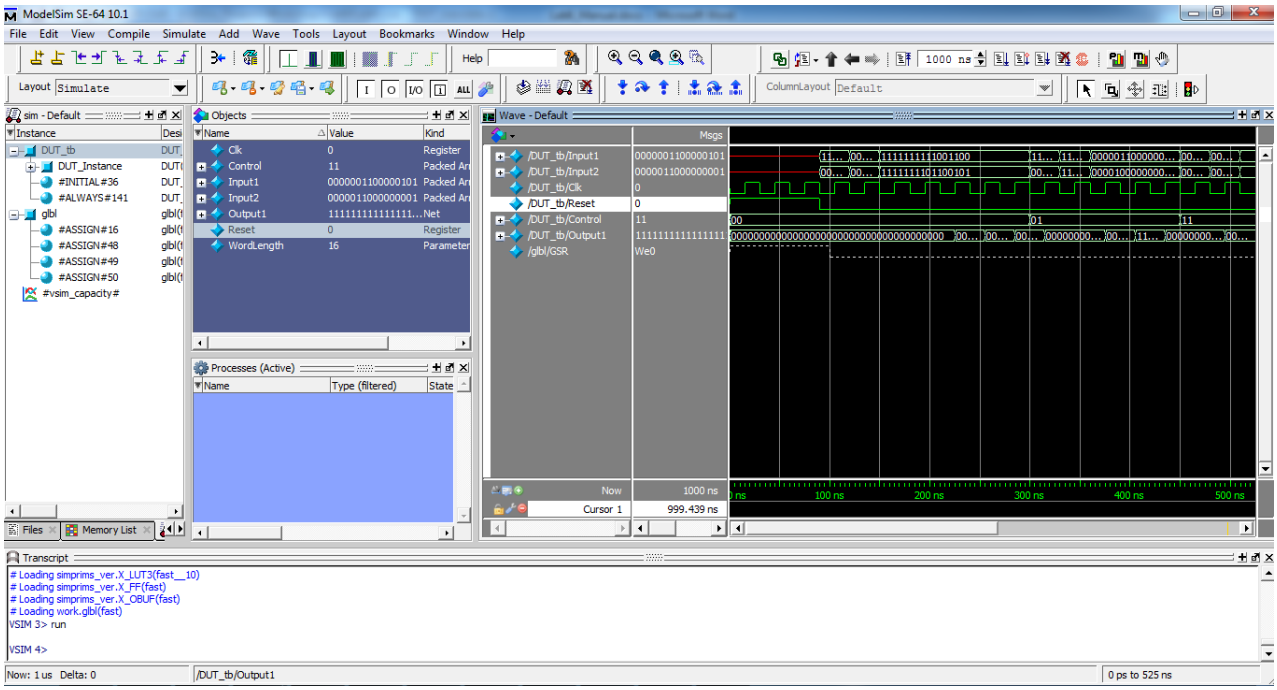

Figure 10.

2.3.11. In the Processes pane, double-click **Simulate Post-Translate Model** to launch to the simulator.

2.3.12. The simulator compiles the input files and then it is ready to start the post-Translate simulation. To continue this process, please refer to Lab2, which helps you to add signals to waveform and configure/set the other simulator options.

3. Post-Map Simulation

Post-Map simulation can be done after mapping the design. In order to perform post-Map simulation for the project, you should generate a post-Map simulation model and then walk through the simulation. In this section all of these three steps are described.

3.1. Map

The first step for the post-Map simulation is design mapping, which is described in detail in step 3 of Lab6. Thus, please refer to Lab6 for more details. Perform Map process completely and then you can continue step3 of Lab7.

3.2. Generate Post-Map Simulation Model (Figure 11-13)

3.2.1. Select **Design Panel**.

3.2.2. Set the Design View to the **Implementation**.

3.2.3. In the Hierarchy pane, select the **top module** (i.e. **DUT.v**).

3.2.4. In the Processes pane, expand **Implementation**.

3.2.5. In the Processes pane, expand **Map**.

3.2.6. In the Processes pane, right-click **Generate Post-Map Simulation Model** and select **Process Properties**.

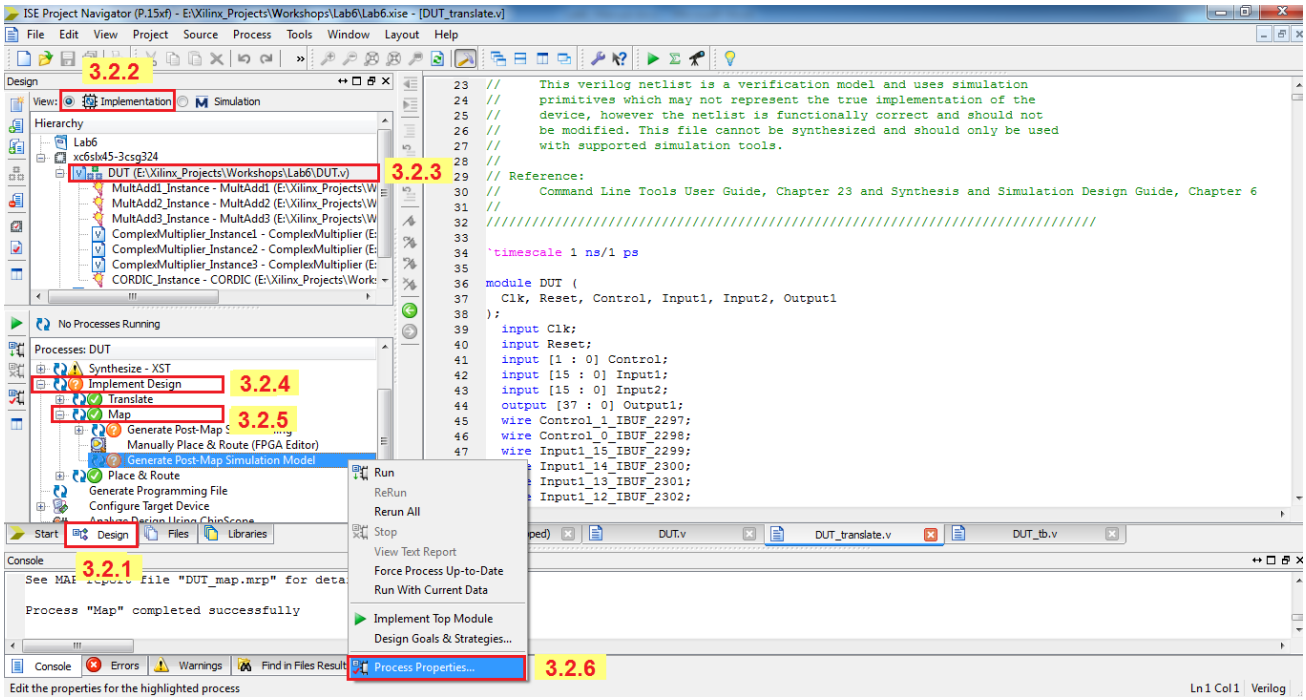

Figure 11.

3.2.7. In the Property display level, select **Advanced**.

3.2.8. Specify the **Simulation Model Properties**.

3.2.9. Click **OK**.

3.2.10. In the Processes pane, right-click **Generate Post-Map Simulation Model** and select **Run.**

3.2.11. In the Transcript Window, check the **Console** tab to view the result of this process. Also, you can check the status of process result in the Processes pane.

3.2.12. If this process is not successful, read the content of Errors/Warnings tabs in the Transcript Window to check the Errors/Warnings of Map process.

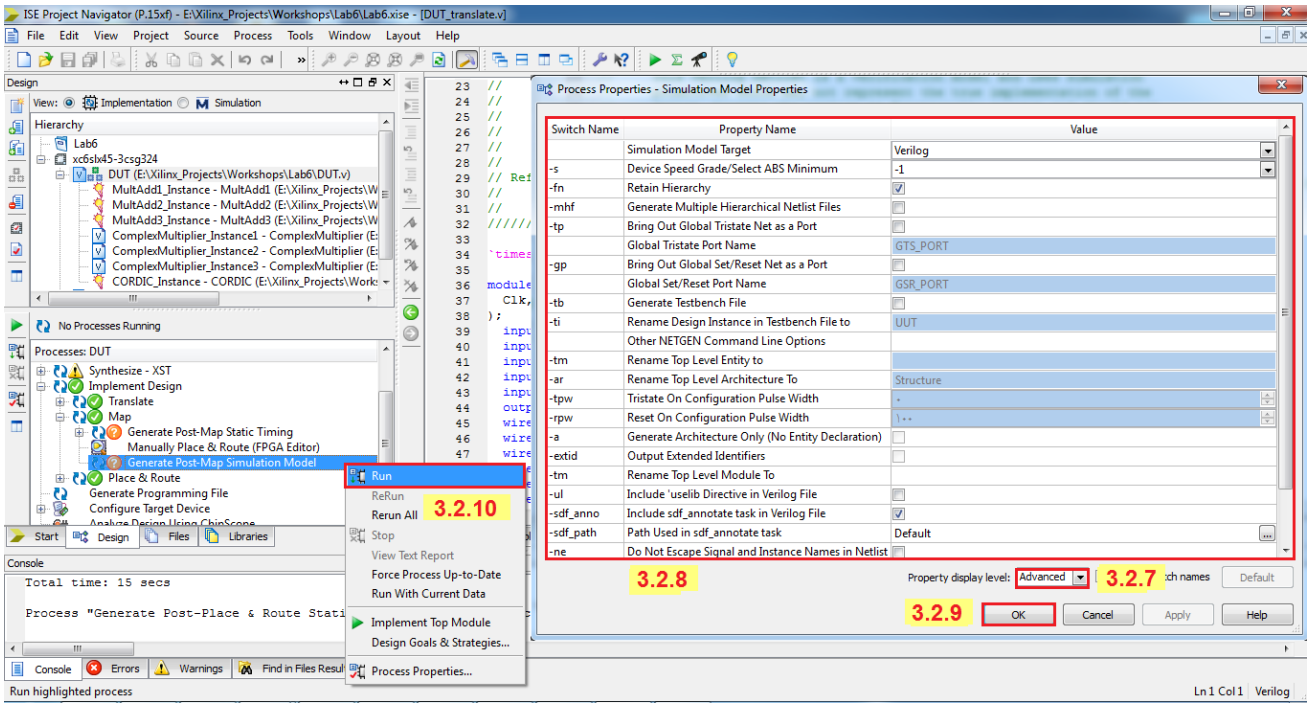

Figure 12.

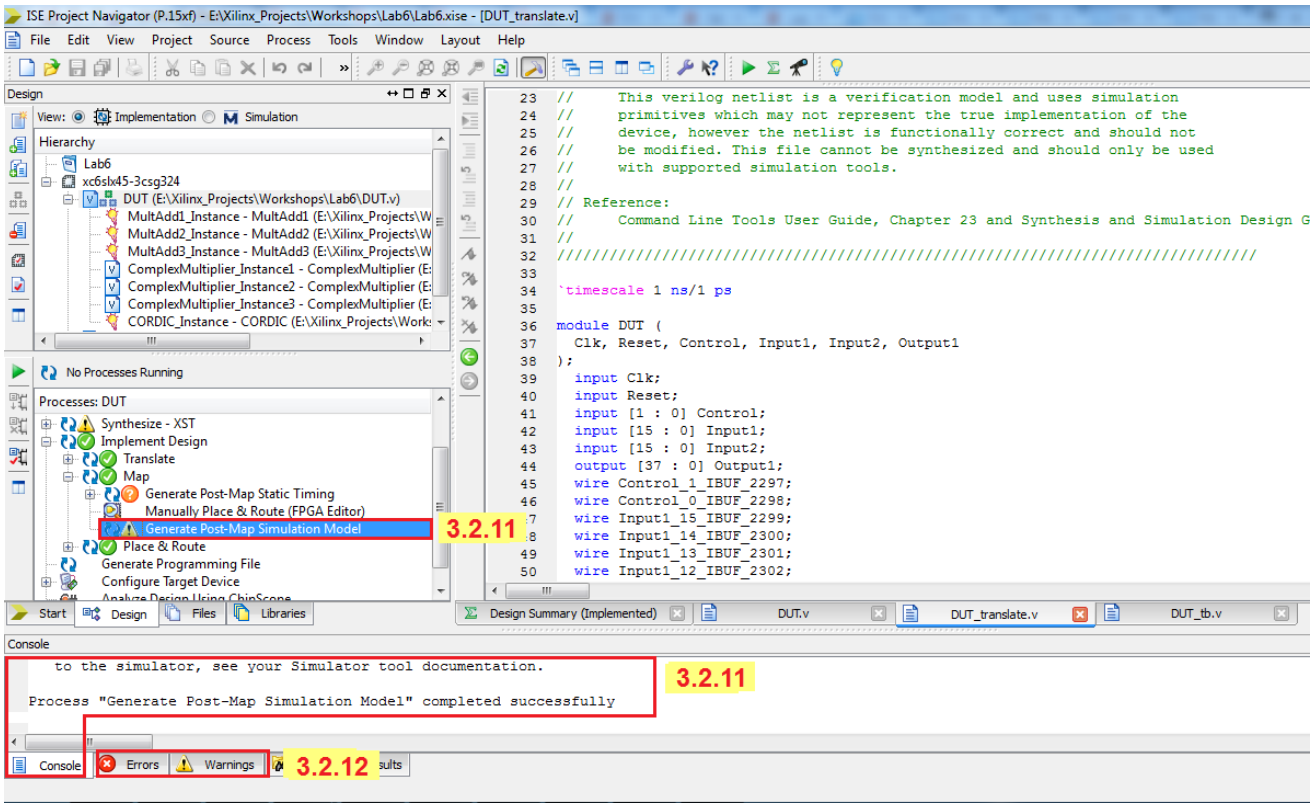

Figure 13.

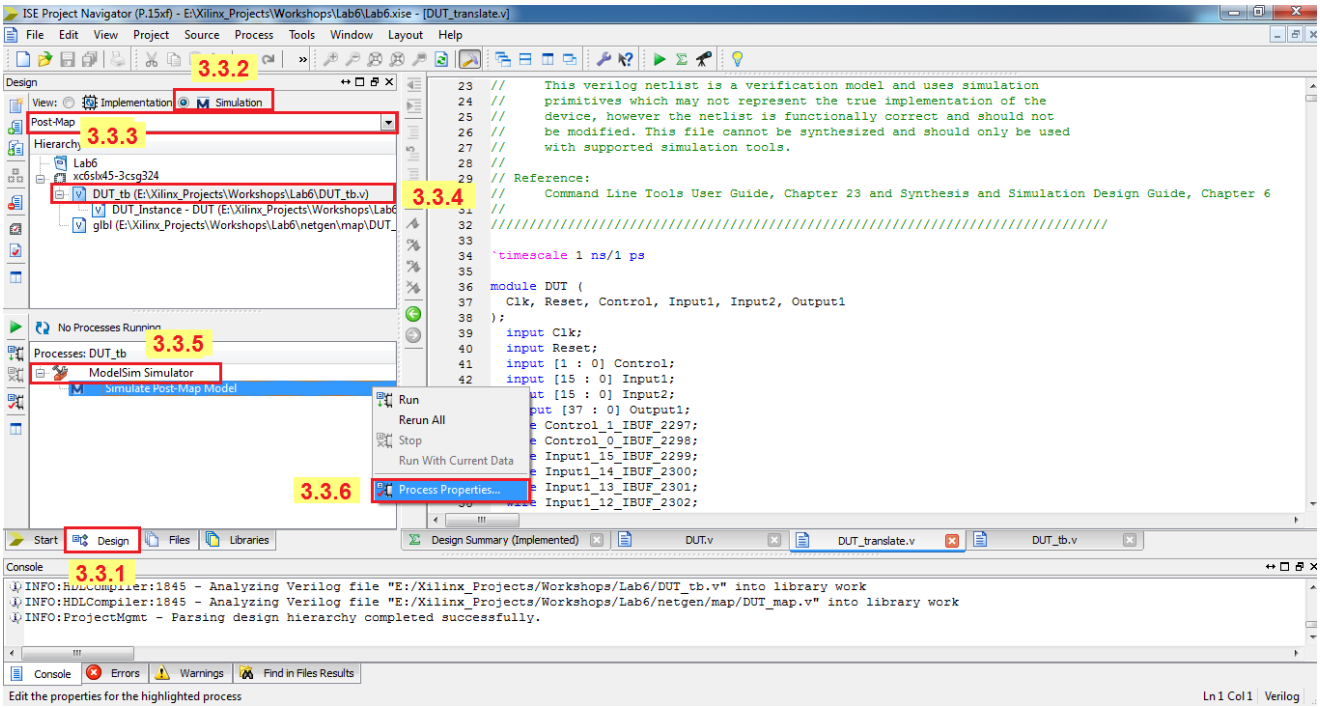

Figure 14.

3.3. Perform the Post-Map Simulation (Figure 14-16)

3.3.1. Select **Design Panel**.

3.3.2. Set the Design View to the **Simulation**.

3.3.3. In the drop down menu of Design panel, select **Post-Map**.

3.3.4. In the Hierarchy pane, select the **test bench** (i.e. **DUT_tb.v**).

3.3.5. In the Processes pane, expand **ModelSim Simulator**.

3.3.6. In the Processes pane, right-click **Simulate Post-Map Model** and select **Process Properties**.

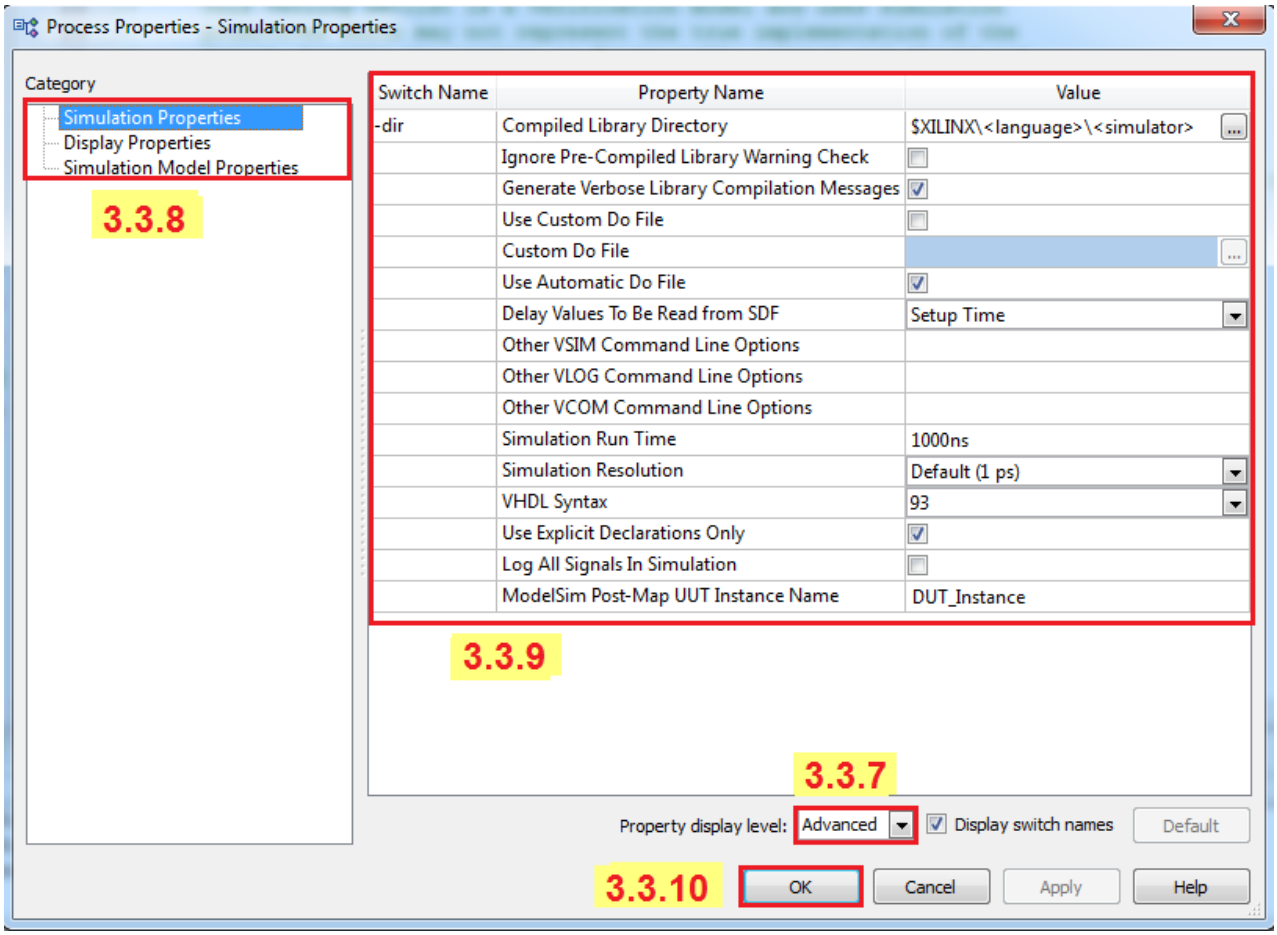

Figure 15.

3.3.7. In the Property display level, select **Advanced**.

3.3.8. Select the desired **category** among the following options:

- Simulation Properties
- Display Properties
- Simulation Model Properties

3.3.9. Specify the **Properties**.

3.3.10. Click **OK**.

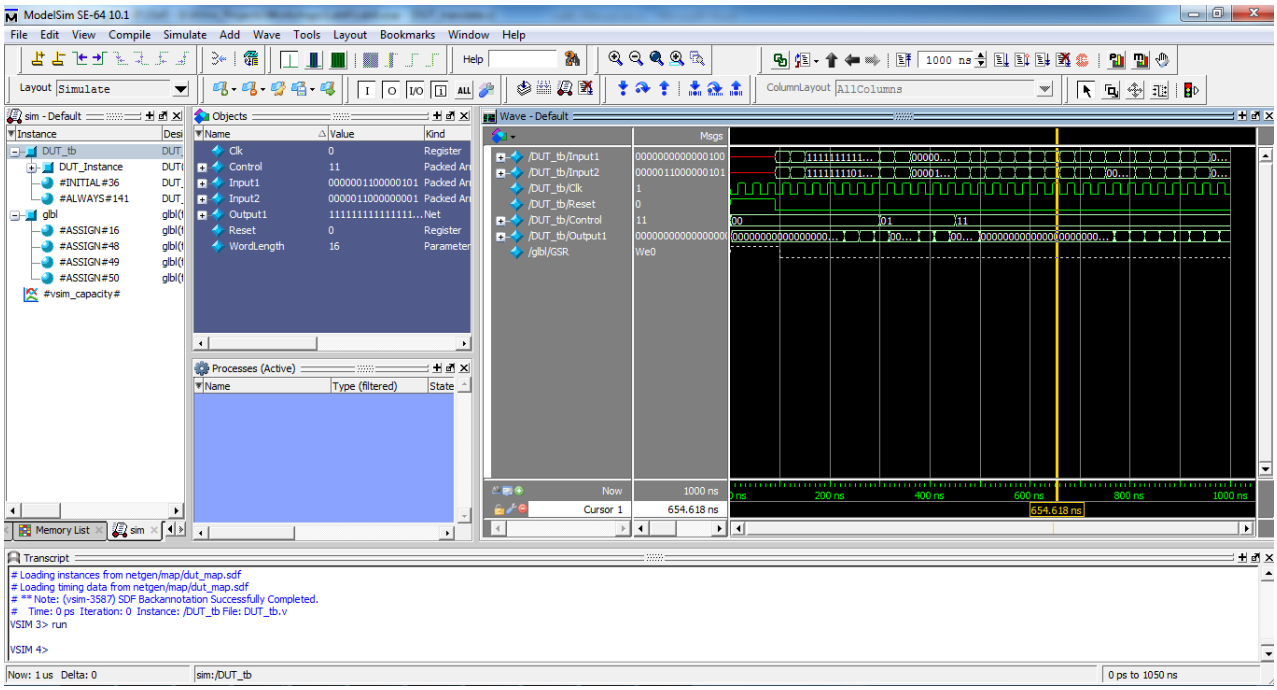

Figure 16.

3.3.11. In the Processes pane, double-click **Simulate Post-Map Model** to launch to the simulator.

3.3.12. The simulator compiles the input files and then it is ready to start the post-Map simulation. To continue this process, please refer to Lab2, which helps you to add signals to waveform and configure/set the other simulator options.

4. Post-Route Simulation

Post-route simulation, which is usually referred as the timing simulation can be done after the Place and Route process. Timing simulation uses the block and routing delay information from a routed design to give a more accurate assessment of the behavior of the circuit under worst-case conditions. For this reason, timing simulation is performed after the design has been placed and routed. Timing simulation (i.e. post-Place and Route) is a highly recommended part of the HDL design flow for Xilinx devices. Timing simulation uses the detailed timing and design layout information that is available after Place and Route. This enables simulation of the design, which closely matches the actual device operation. In order to perform post-route simulation for the project, you should generate a post-route simulation model and then walk through the simulation. In this section all of these three steps are described.

4.1. Place and Route

The first step for the post-Route simulation is the Place and Route process, which is described in detail in step 4 of Lab6. Thus, please refer to Lab6 for more details. Perform the Place and Route process completely and then you can continue Lab7.

4.2. Generate Post- Place and Route Simulation Model (Figure 17-19)

4.2.1. Select **Design Panel**.

4.2.2. Set the Design View to the **Implementation**.

4.2.3. In the Hierarchy pane, select the **top module** (i.e. **DUT.v**).

4.2.4. In the Processes pane, expand **Implementation**.

4.2.5. In the Processes pane, expand **Place & Route**.

4.2.6. In the Processes pane, right-click **Generate Post- Place & Route Simulation Model** and select **Process Properties**.

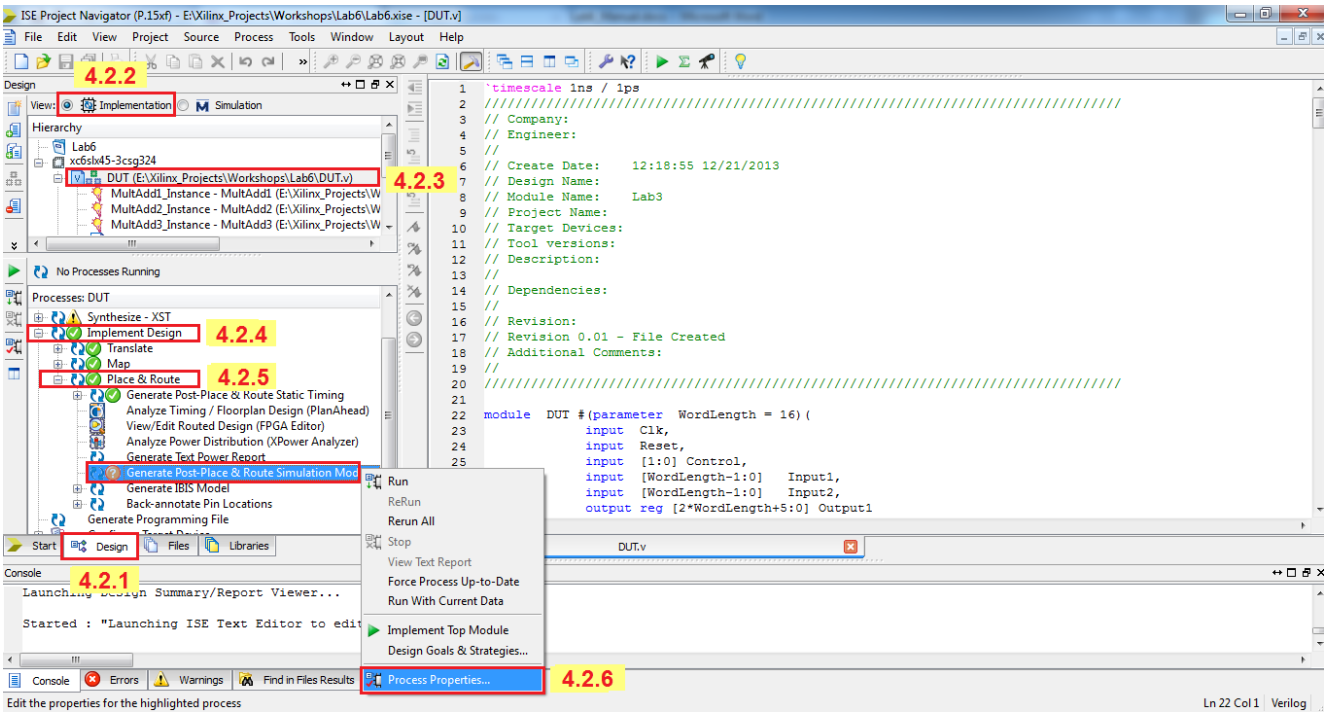

Figure 17.

4.2.7. In the Property display level, select **Advanced**.

4.2.8. Specify the **Simulation Model Properties**.

4.2.9. Click **OK**.

4.2.10. In the Processes pane, right-click **Generate Post- Place & Route Simulation Model** and select **Run.**

4.2.11. In the Transcript Window, check the **Console** tab to view the result of this process. Also, you can check the status of process result in the Processes pane.

4.2.12. If this process is not successful, read the content of Errors/Warnings tabs in the Transcript Window to check the Errors/Warnings of Place and Route process.

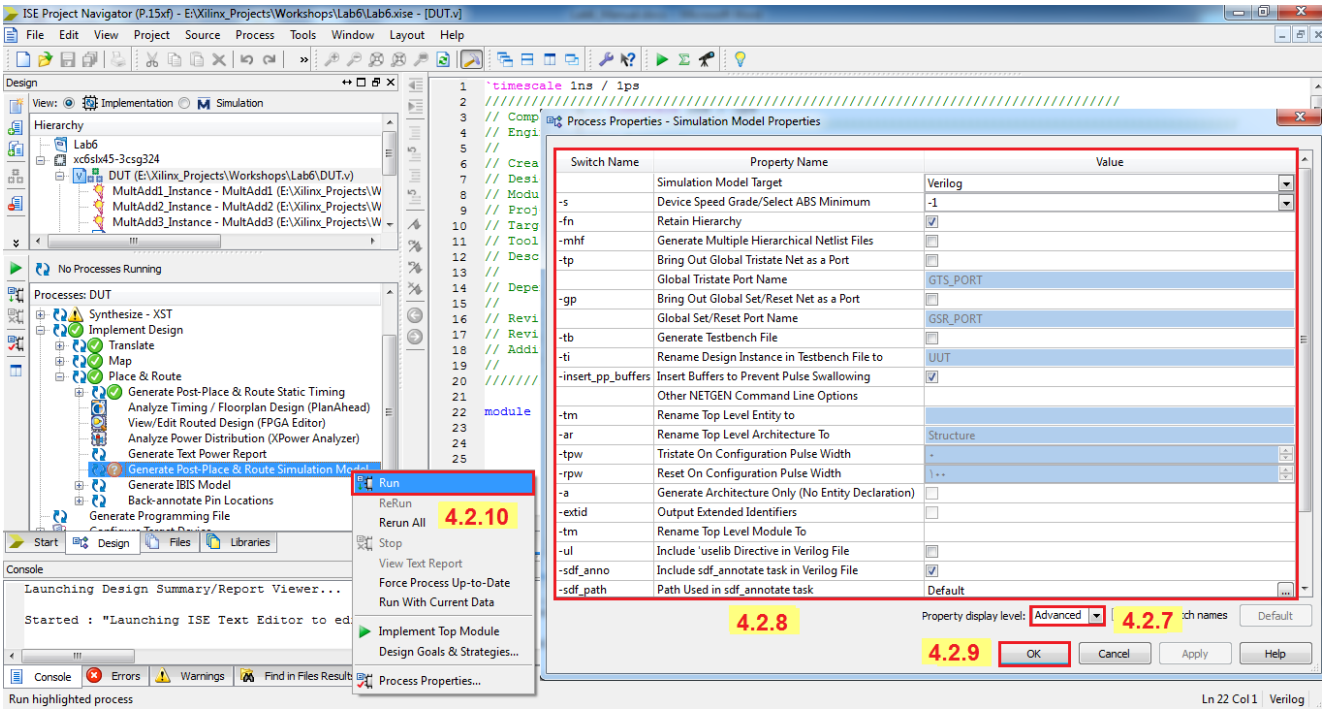

Figure 18.

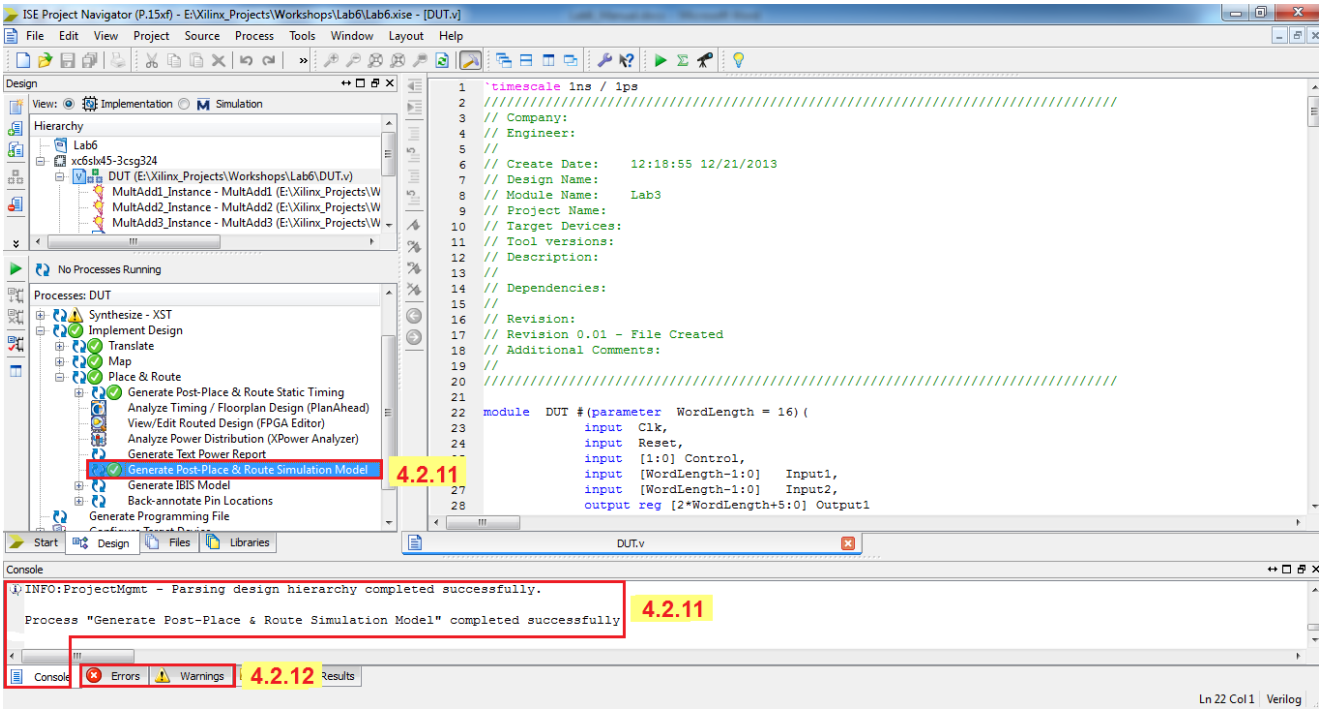

Figure 19.

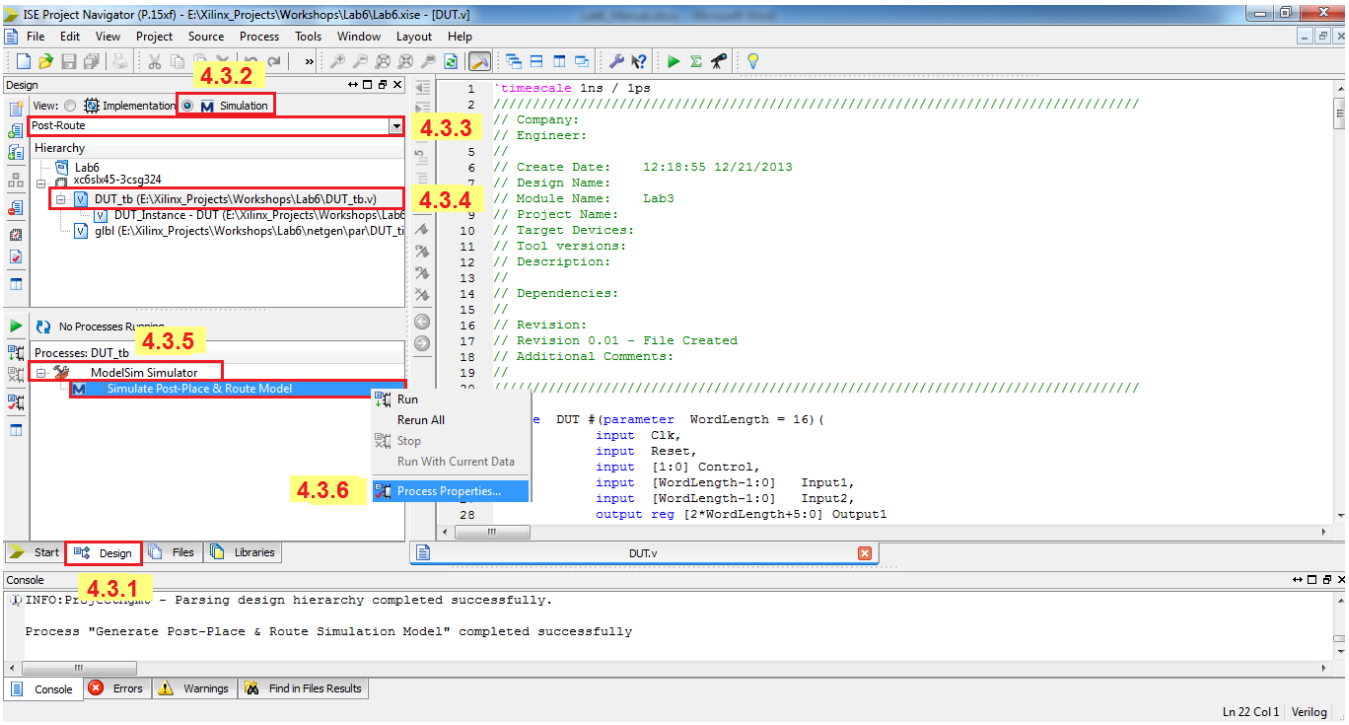

Figure 20.

4.3. Perform the Post- Place & Route Simulation (Figure 20-22)

4.3.1. Select **Design Panel**.

4.3.2. Set the Design View to the **Simulation**.

4.3.3. In the drop down menu of Design panel, select **Post-Route**.

4.3.4. In the Hierarchy pane, select the **test bench** (i.e. **DUT_tb.v**).

4.3.5. In the Processes pane, expand **ModelSim Simulator**.

4.3.6. In the Processes pane, right-click **Simulate Post- Place & Route Model** and select **Process Properties**.

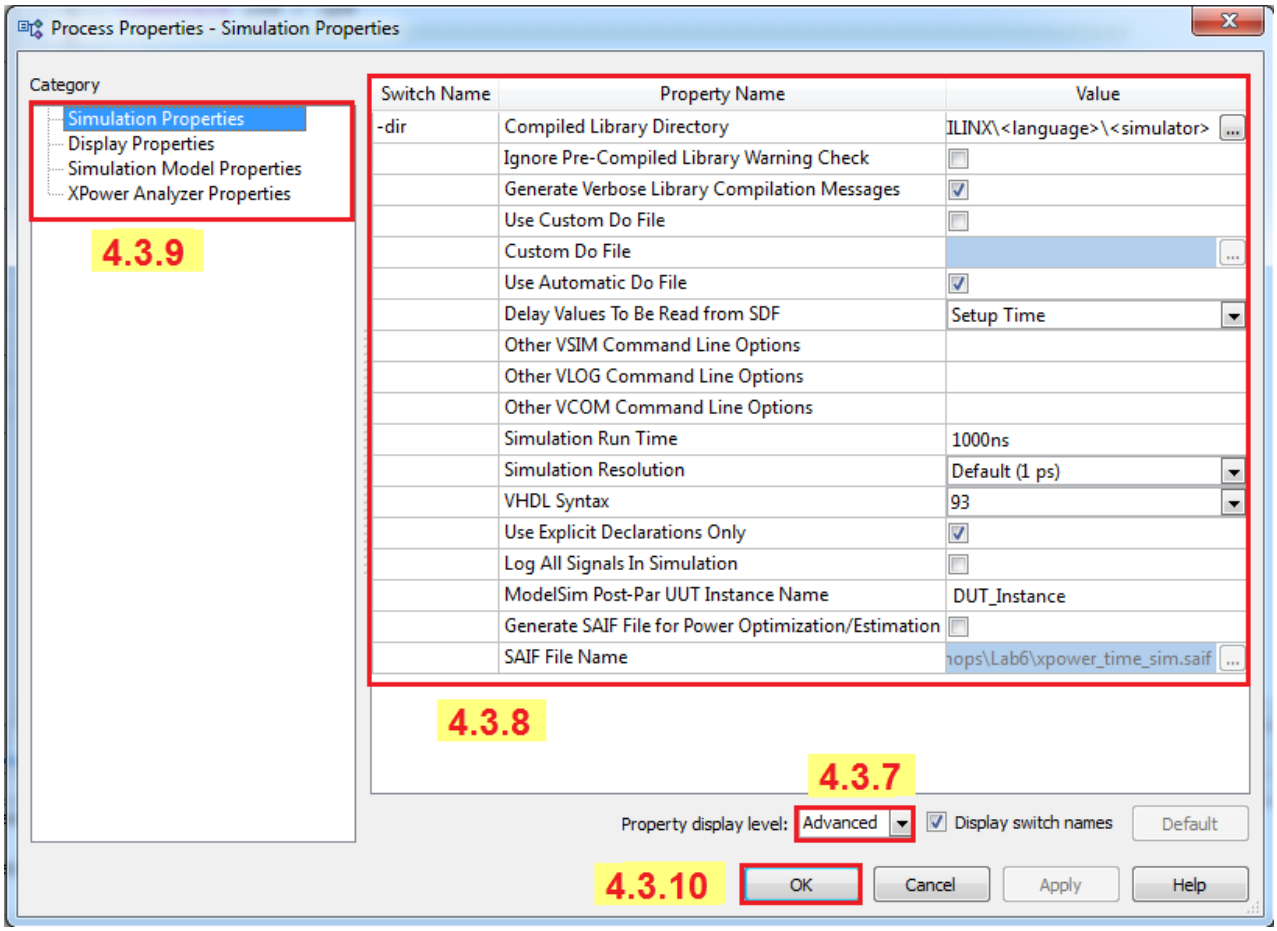

Figure 21.

4.3.7. In the Property display level, select **Advanced**.

4.3.8. Select the desired **category** among the following options:

- Simulation Properties
- Display Properties
- Simulation Model Properties

4.3.9. Specify the **Properties**.

4.3.10. Click **OK**.

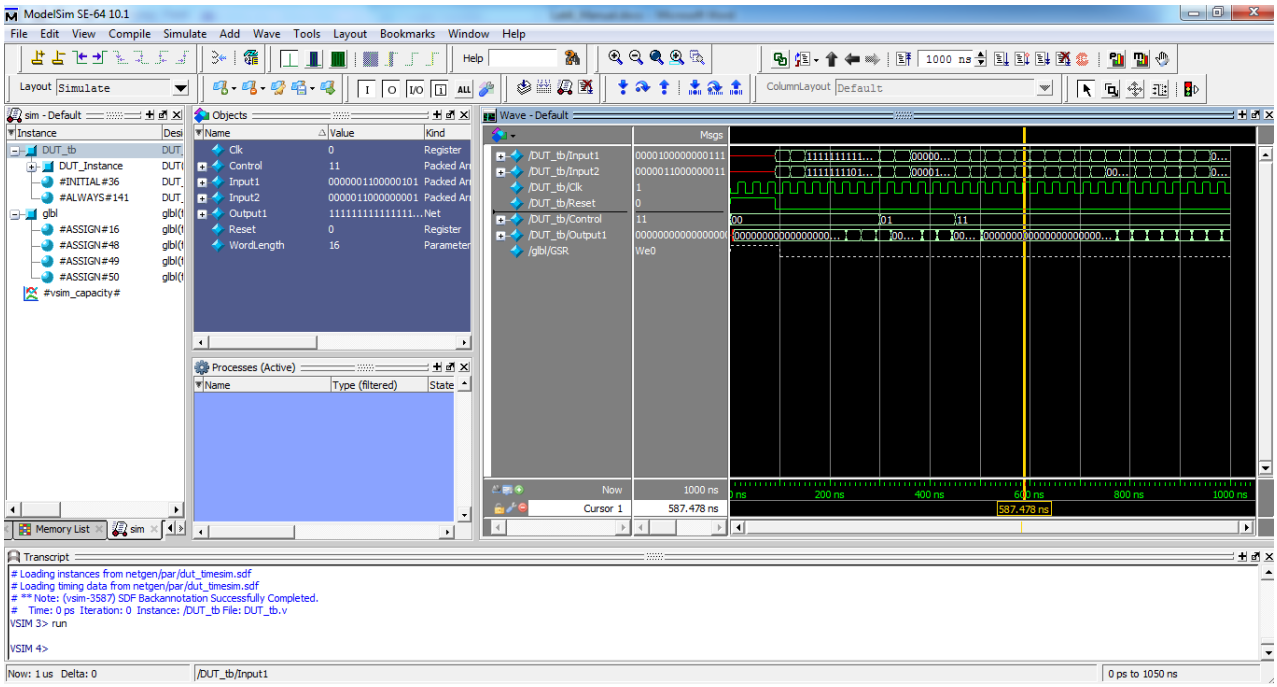

Figure 22.

4.3.11. In the Processes pane, double-click **Simulate Post- Place & Route Model** to launch to the simulator.

4.3.12. The simulator compiles the input files and then it is ready to start the post-Place & Route simulation. To continue this process, please refer to Lab2, which helps you to add signals to waveform and configure/set the other simulator options.

Digital Circuit Design with Xilinx FPGA

Lab Manual

Workshop No.1

Lab8:

FPGA Configuration Using iMPACT

Outline

This manual covers the following topics:

- Generating programming file for FPGA configuration
- Working with iMPACT for file generation
- Generating PROM files using iMPACT
- FPGA configuration using iMPACT

Introduction

After the design creation and synthesizing (Lab1), behavioral simulation (Lab2), entering constraints to the design (Lab4), and timing simulation (Lab7), the design is ready to implement into the target FPGA. After the design is completely routed, you must configure the device so it can execute the desired function. During this process, ISE produces a bitstream for a Xilinx device, which should be downloaded from a host computer to the target device. iMPACT tool, which is a file generation and device programming tool allows you to perform Device Configuration and File Generation. iMPACT can create bitstream files, System ACE solution files, PROM files, and SVF/XSVF files. Moreover, iMPACT enables you to do the following:

- Device Configuration
- File Generation
- Readback and verify design configuration data
- Debug configuration problems
- Execute SVF and XSVF files

In this lab, you will learn how to generate different types of the above files and configure/program the target FPGA.

Required Software

To perform this manual, you must have **Xilinx ISE Design Suite 14.1** and **ModelSim 10.1** (or above) installed. Note that in this manual the pictures are captured in **Xilinx ISE Design Suite 14.1**, but all the procedures can be done in other versions of this software.

Objectives

After completing this lab, you will be able to

- Generate a bitstream for FPGA configuration
- Create and manage a project in iMPACT
- Create a PROM file using iMPACT
- Configure the FPGA using iMPACT

General flow of this lab

This lab comprises three steps as follows:

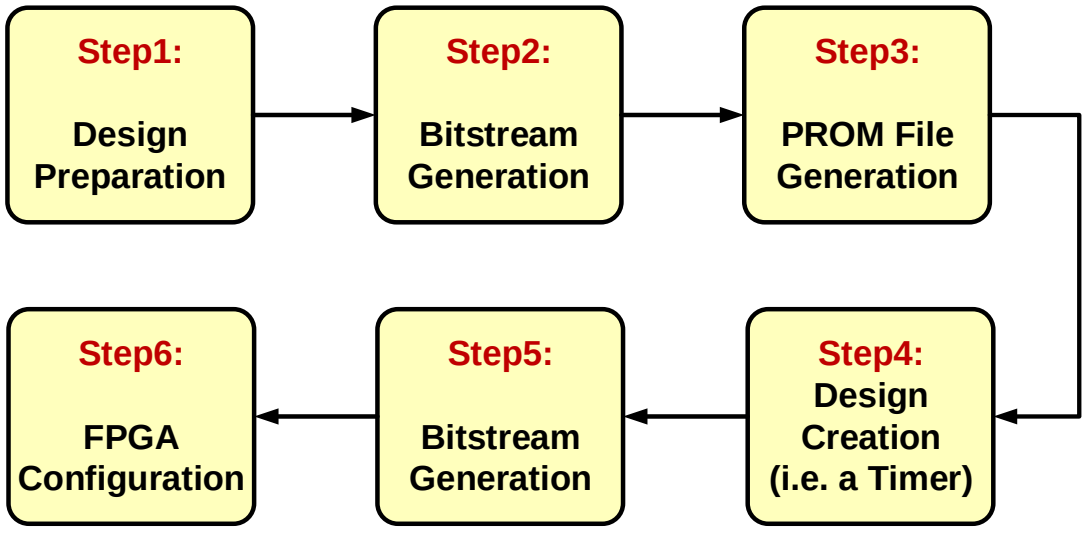

Figure 1. General flow of Lab 8.

Design Description

In this lab, you'll work with **two projects**:

1- First, you'll use the completed design of Lab7, which is implemented completely and it passed the timing simulation. In this lab, you'll generate a bitstream for the design. Thus, you'll perform Step1, Step2, and Step3 of this lab for the completed design of Lab7.
2- In Step 4 of this lab, you should create a new ISE project to implement a **Timer**. Then, you'll generate the corresponding bitstream. Finally, you'll load this design into the FPGA by programming the FPGA using iMPACT tool. Thus, you'll perform Step4, Step5, and Step6 of this lab for the Timer. The specification of the Timer is described in section 4 of this manual.

1. Design Preparation

1.1. Start the Project Navigator (Figure 2)

Double click the Project Navigator icon on your desktop, or select **Start > All Programs > Xilinx Design Tools > ISE Design Suite 14.1 > ISE Design Tools > Project Navigator.**

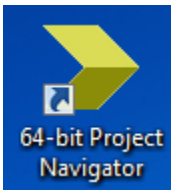

Figure 2. Project Navigator desktop icon.

1.2. Open the project (Figure 3)

1.2.1. From Project Navigator, select **File > Open Project**.

1.2.2. Go to the project directory of Lab7 or Lab6.

1.2.3. Select **Lab7.xise** or **Lab6.xise**.

1.2.4. Click **Open**.

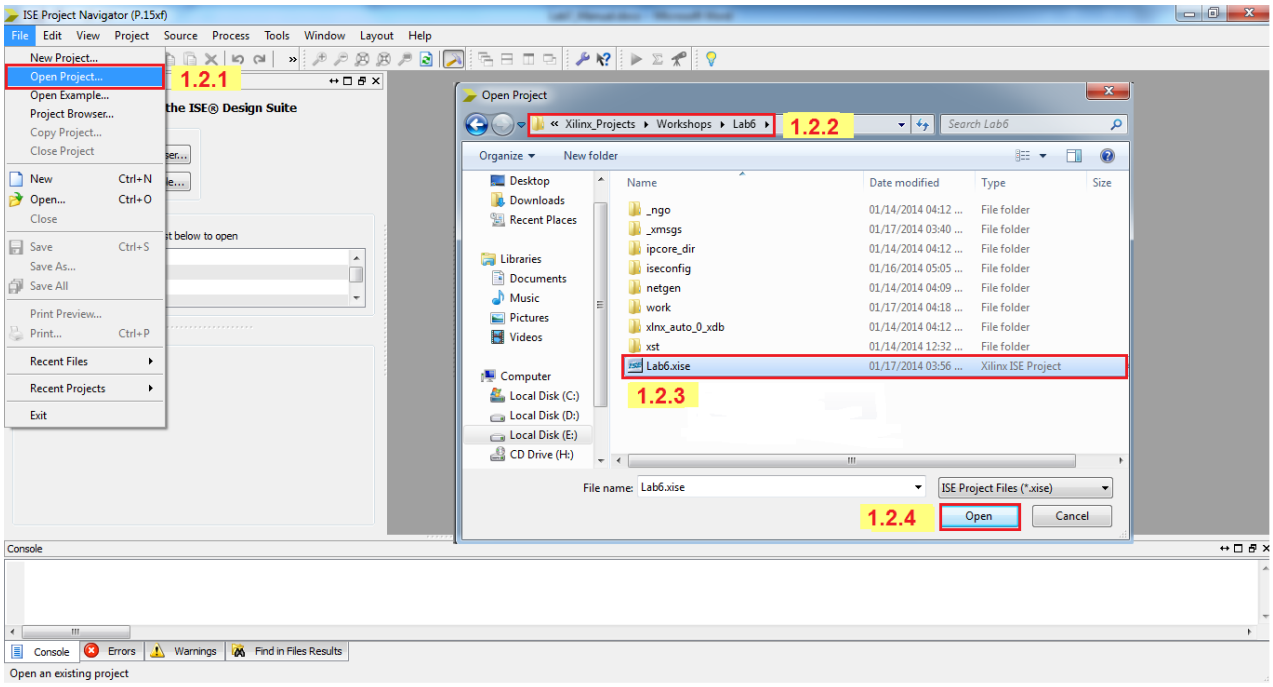

Figure 3.

1.3. Copy the project (Figure 4)

1.3.1. From Project Navigator, select **File > Copy Project**.

1.3.2. Specify the location, in which you want to save the result of this lab.

1.3.3. Enter **Lab8** as the name of the copied project.

1.3.4. Select **Copy sources to the new location**.

1.3.5. Select the **Open the copied project** to open the Lab8 project automatically.

1.3.6. Click **OK**.

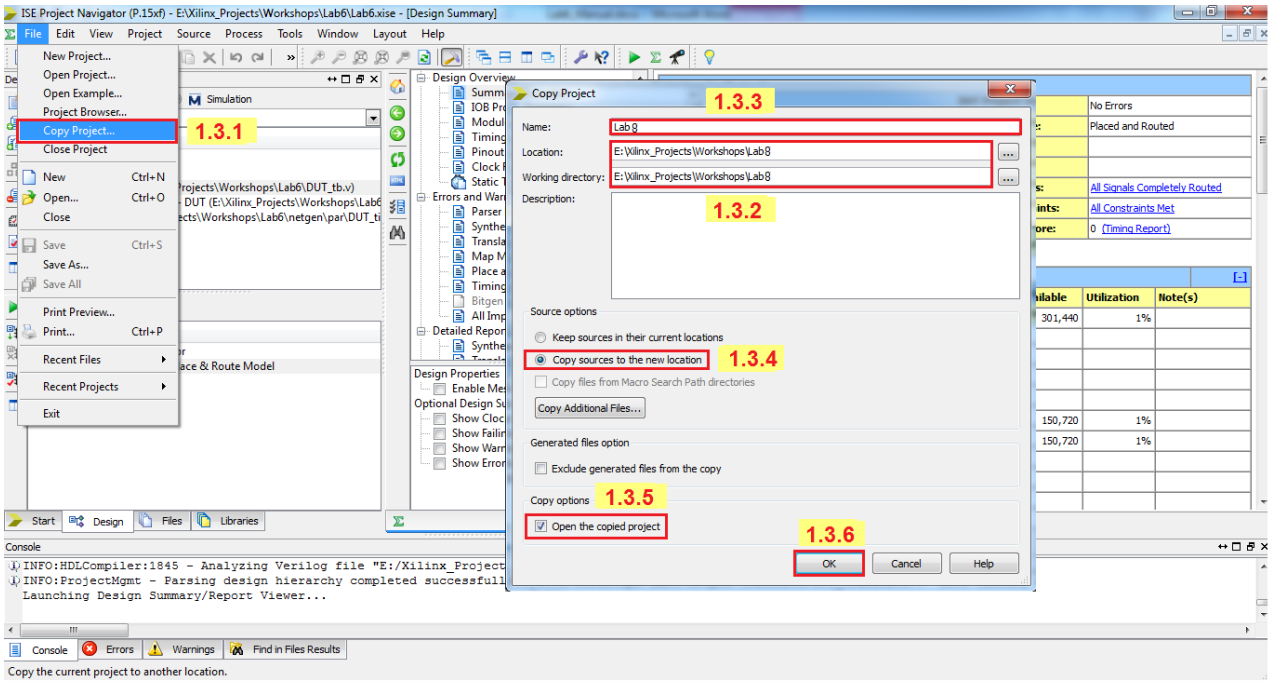

Figure 4.

2. Generating Programming File (Figure 5-6)

A programming file or configuration bitstream is created for downloading to a target device or for formatting into a PROM programming file. In this step, you'll generate a bitstream for the target FPGA as follow:

2.1. Select **Design Panel**.

2.2. Set the Design View to the **Implementation**.

2.3. In the Hierarchy pane, select the **top module** (i.e. **DUT.v**).

2.4. In the Processes pane, expand **Implementation**.

2.5. In the Processes pane, right-click **Generate Programming File** and select **Process Properties**.

2.6. Select **Advanced** in the Property display level.

2.7. Select one of the desired categories.

2.8. Specify the process properties.

2.9. Click **OK**.

2.10. In the Processes pane, right-click **Generate Programming File** and select **Run**.

2.11. View the result of Generate Programming File process.

2.12. In the Processes pane, double-click **Design Summary/Reports**.

2.13. Expand **Errors and Warnings**.

2.14. To view the corresponding messages, select **Bitgen Messages** or select **Errors/Warnings** tab in the Transcript Window.

2.15. Expand **Detailed Reports**.

2.16. Select **Bitgen Report** to review the Programming File Generation Report. You can verify that the specified options were used when creating the configuration data.

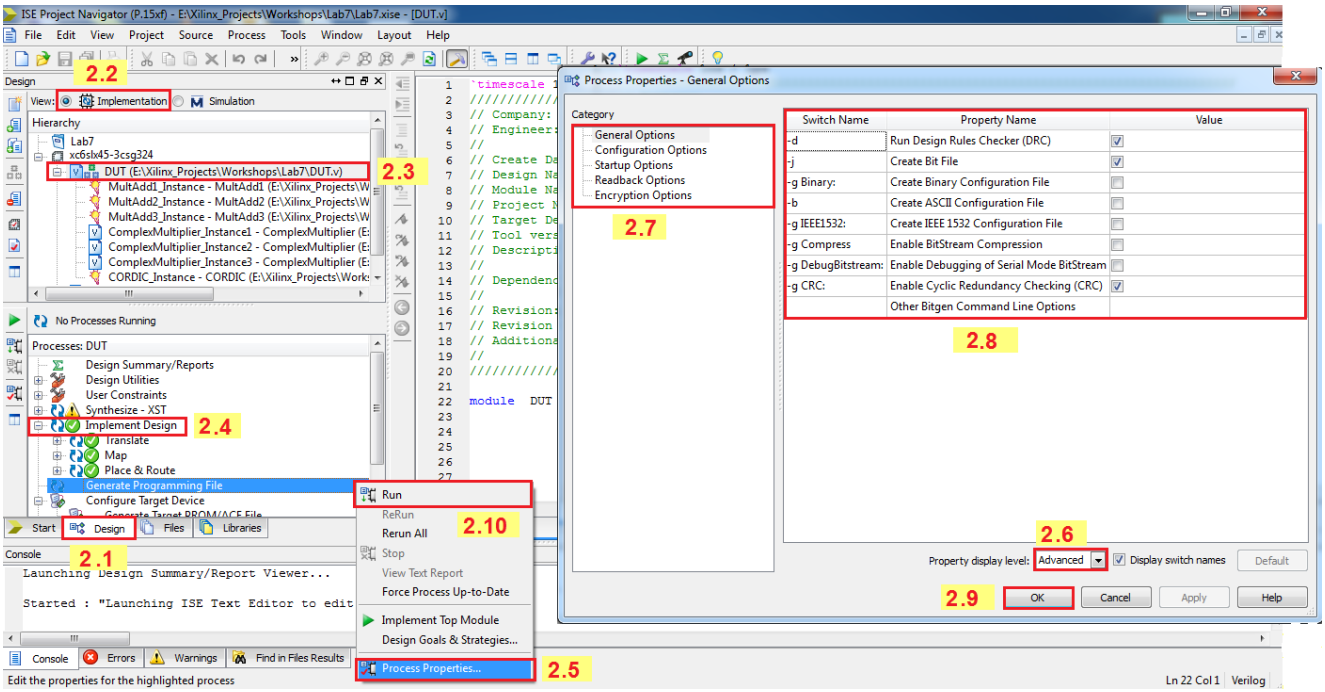

Figure 5.

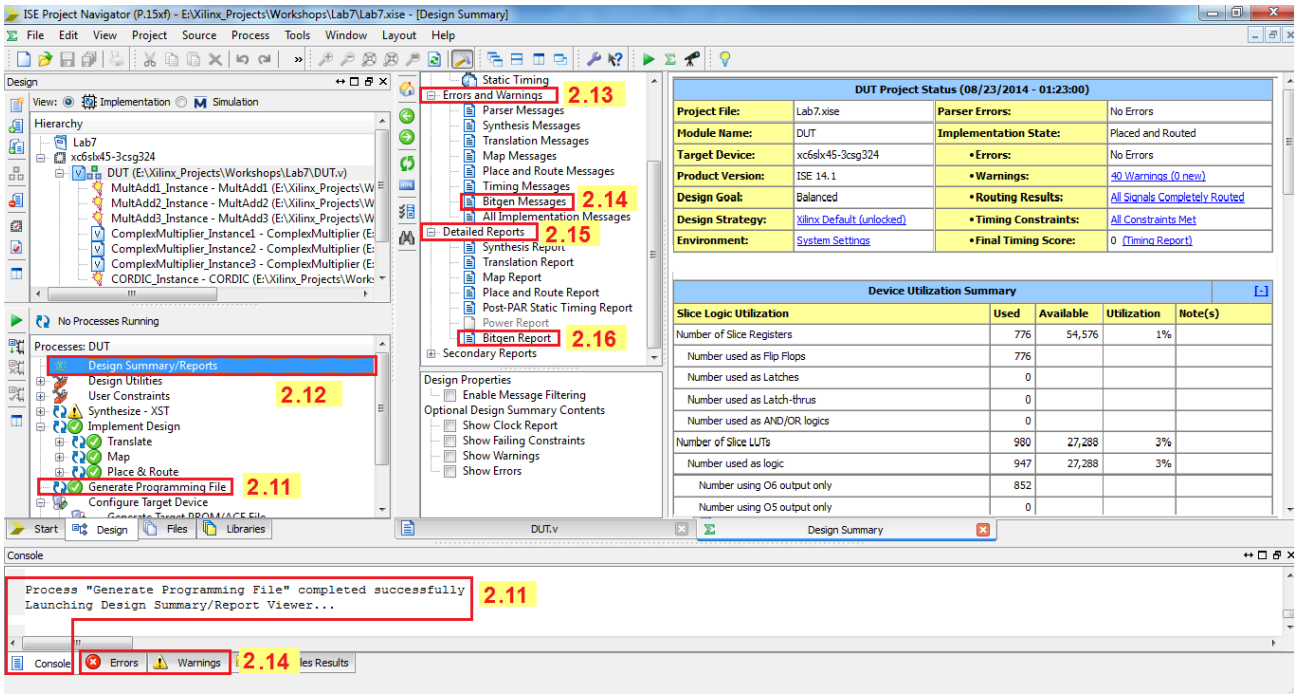

Figure 6.

3. Generating a PROM File using iMPACT (Figure 7-15)

To program a single device using iMPACT, all you need is a bitstream file. To program several devices in a daisy chain configuration or to program your devices using a PROM, you must use iMPACT to create a PROM file. iMPACT accepts any number of bitstreams and creates one or more PROM files containing one or more daisy chain configurations. You can create a PROM file, add additional bitstreams to the daisy chain, and create additional daisy chains in iMPACT. Perform the following steps to create a PROM file in iMPACT:

3.1. Select **Design Panel**.

3.2. Set the Design View to the **Implementation**.

3.3. In the Hierarchy pane, select the **top module** (i.e. **DUT.v**).

3.4. In the Processes pane, expand **Configure Target Device**

3.5. In the Processes pane, double-click **Generate Target PROM/ACE File**.

3.6. Click **OK** in the warning message.

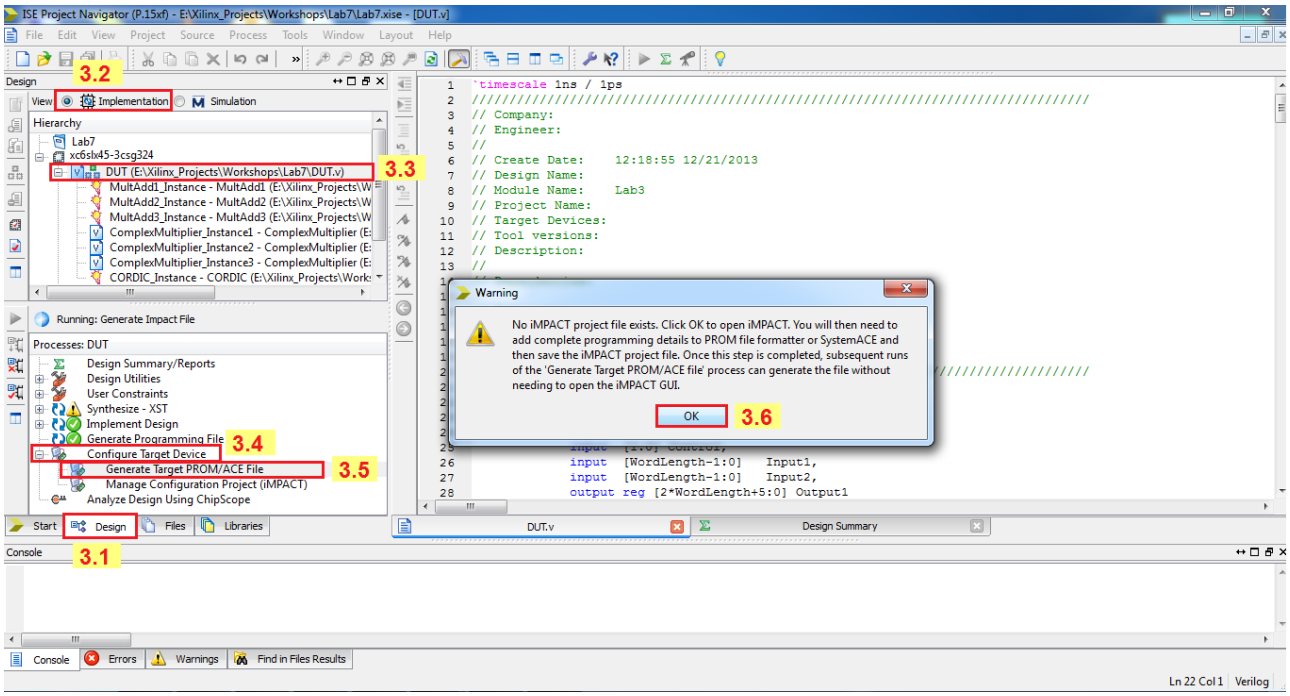

Figure 7.

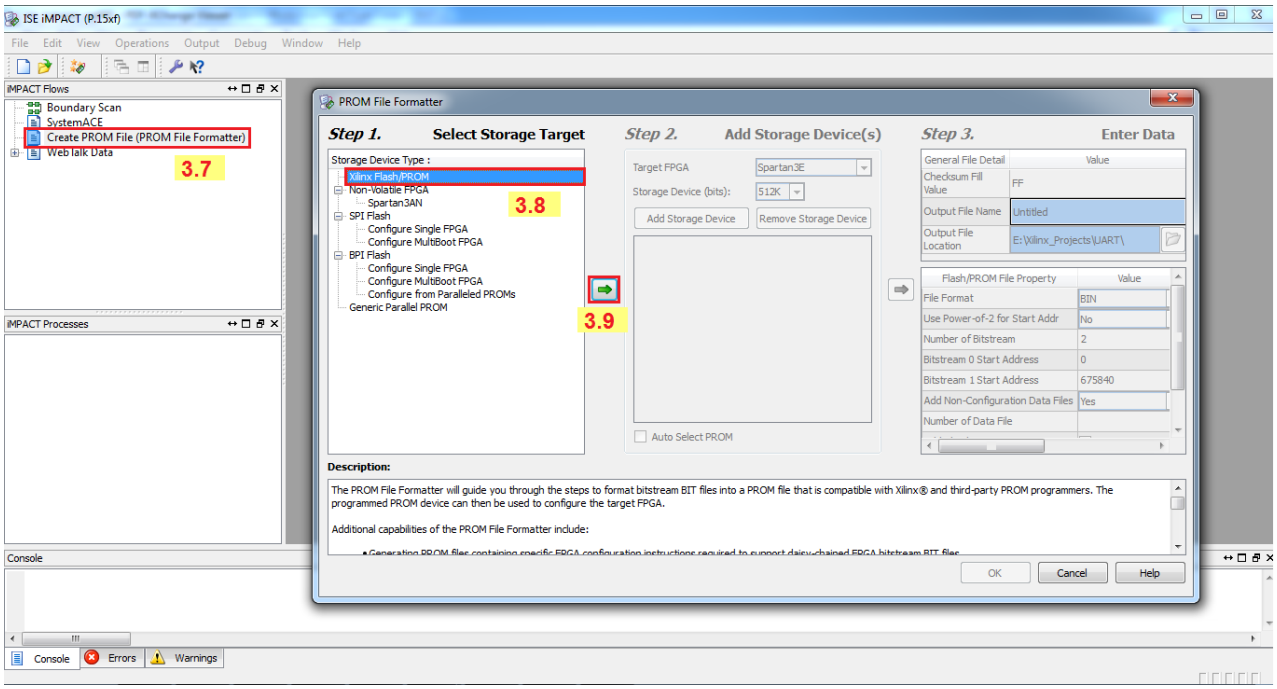

Figure 8.

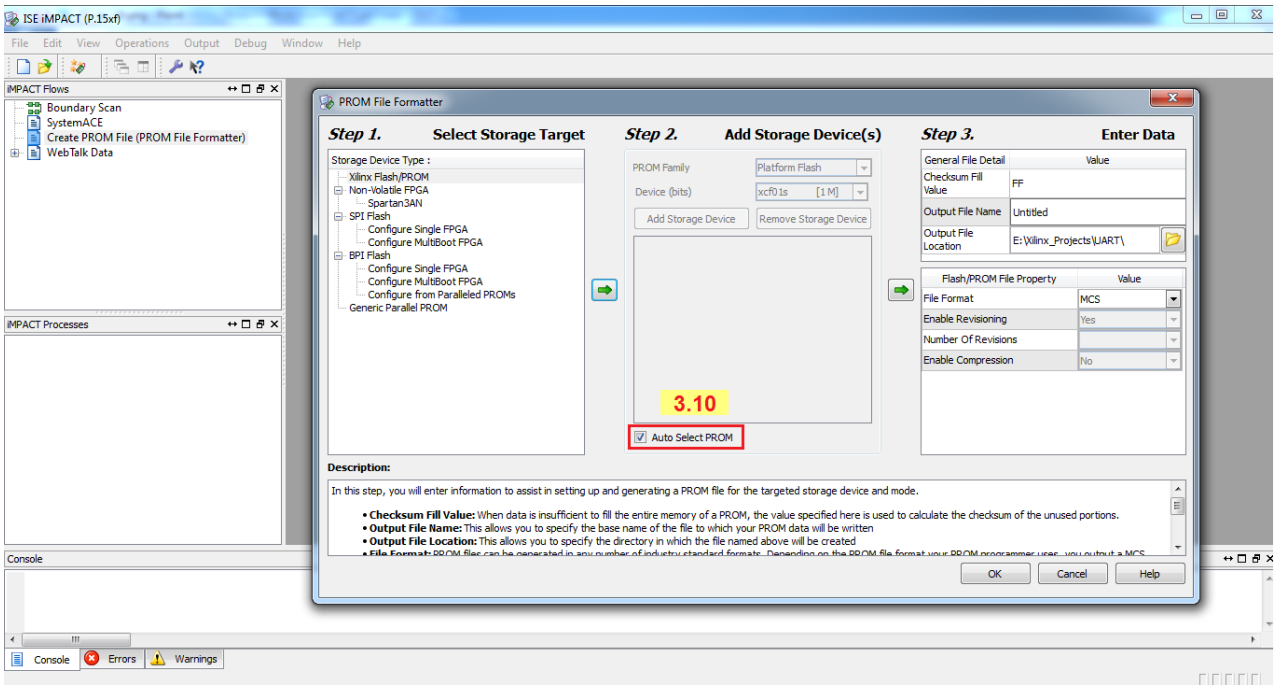

Figure 9.

3.7. In iMPACT, double-click on **Create PROM File (PROM File Formatter)** in the iMPACT Flows window.

3.8. In the PROM File Formatter window, select **Xilinx Flash/PROM** in the Select Storage Target section **(Step 1)**.

3.9. Click the **green arrow** to activate the next section.

3.10. In the Add Storage Device(s) section **(Step 2)**, click the **Auto Select PROM** checkbox.

3.11. In the Enter Data section **(Step 3)**, specify the **Output File Location**.

3.12. In the Enter Data section **(Step 3)**, enter an **Output File Name**.

3.13. Verify that the Checksum Fill Value is set to **FF** and the File Format is **MCS**.

3.14. Click **OK** to close the PROM File Formatter.

3.15. In the Add Device dialog box, click **OK**

3.16. Select the project directory in the location field.

3.17. Select the **DUT.bit** file.

3.18. Click **Open**.

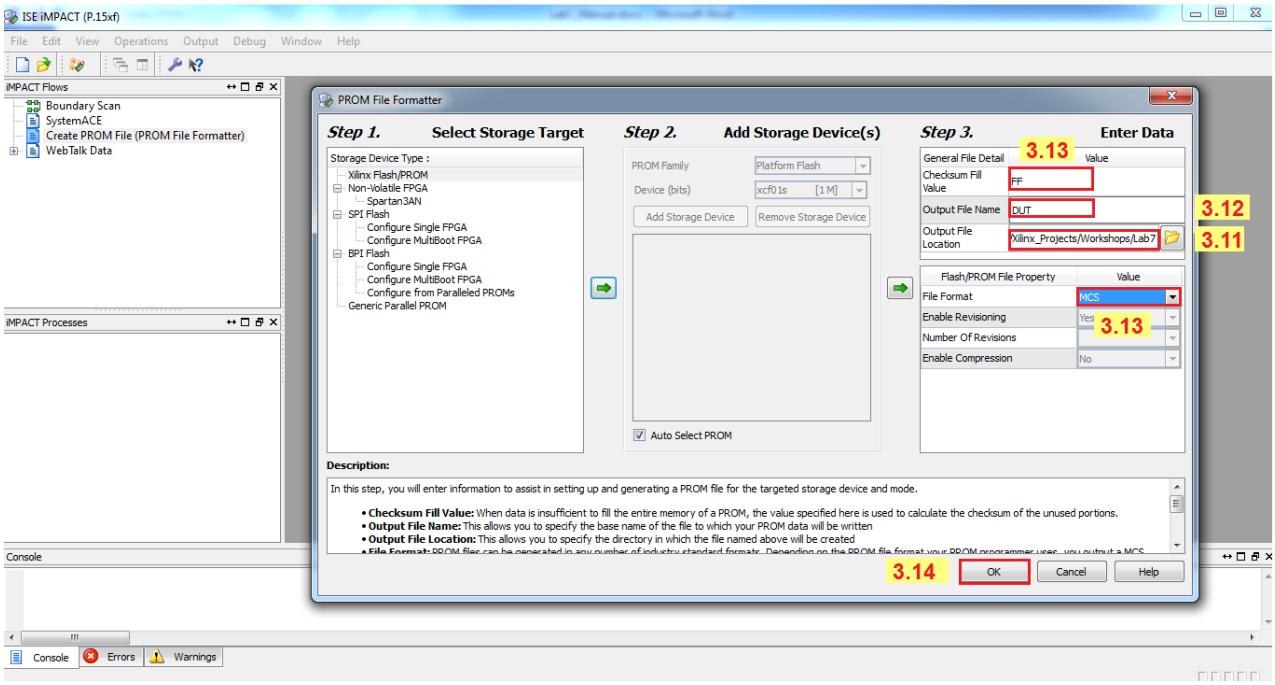

Figure 10.

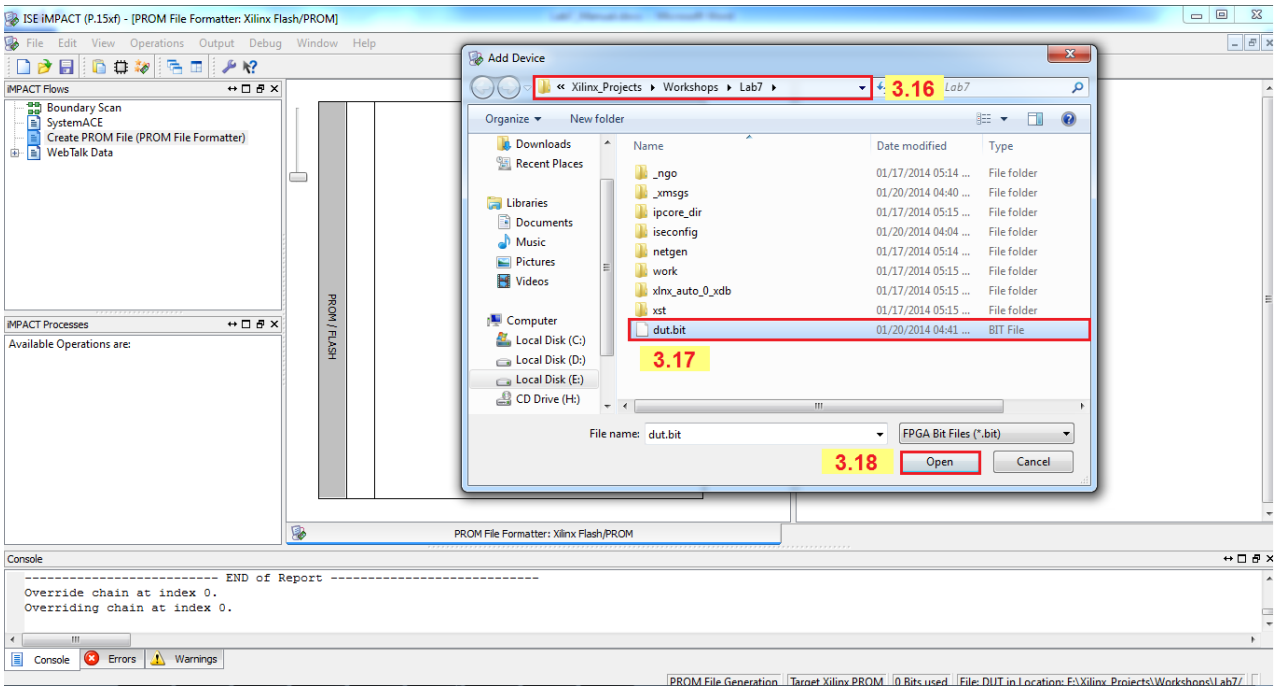

Figure 11.

3.19. Click **No** when you are asked if you would like to add another design file to the datastream. If you want to add another bitstream click **Yes**.

3.20. Click **OK** to complete the process.

3.21. Select the **device graphic** in the workspace area.

3.22. In the iMPACT Processes view, double-click **Generate File**.

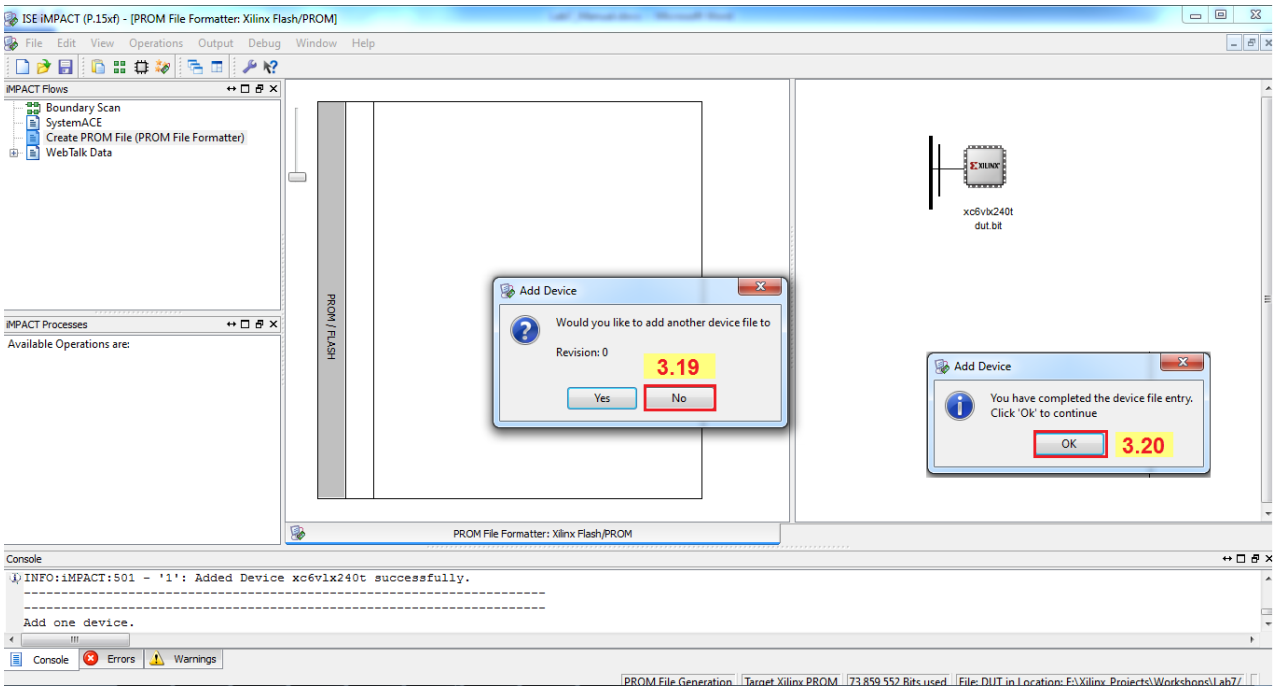

Figure 12.

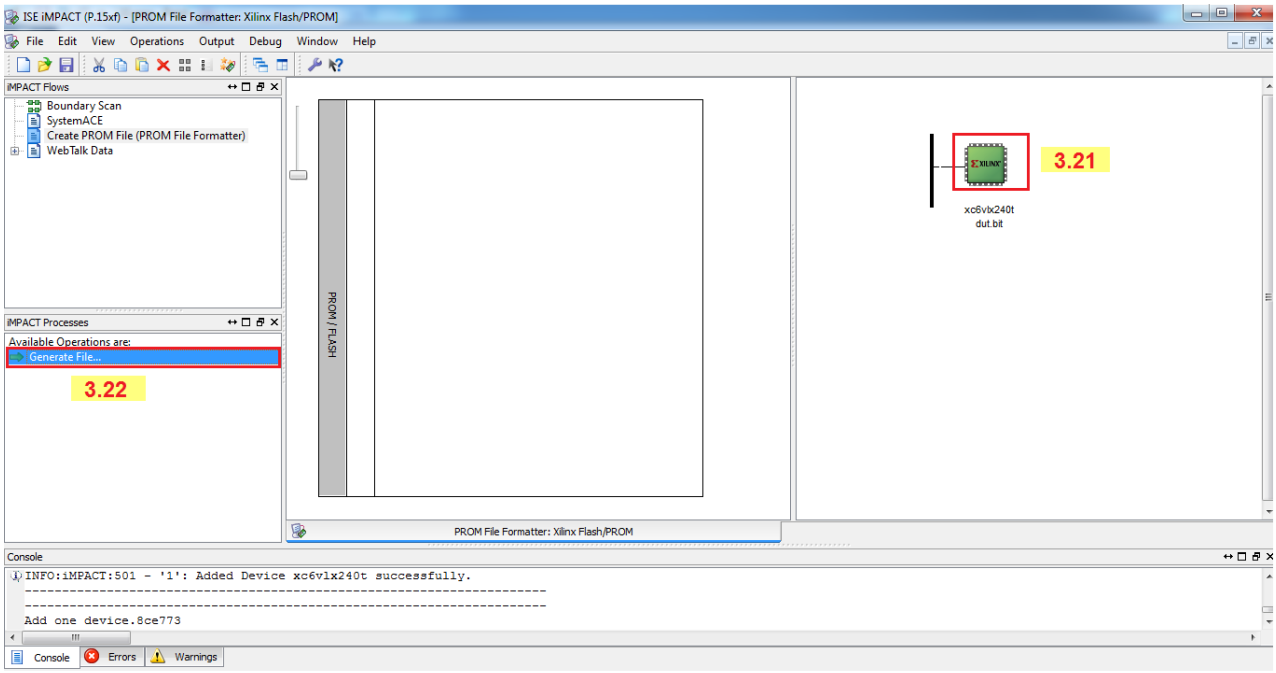

Figure 13.

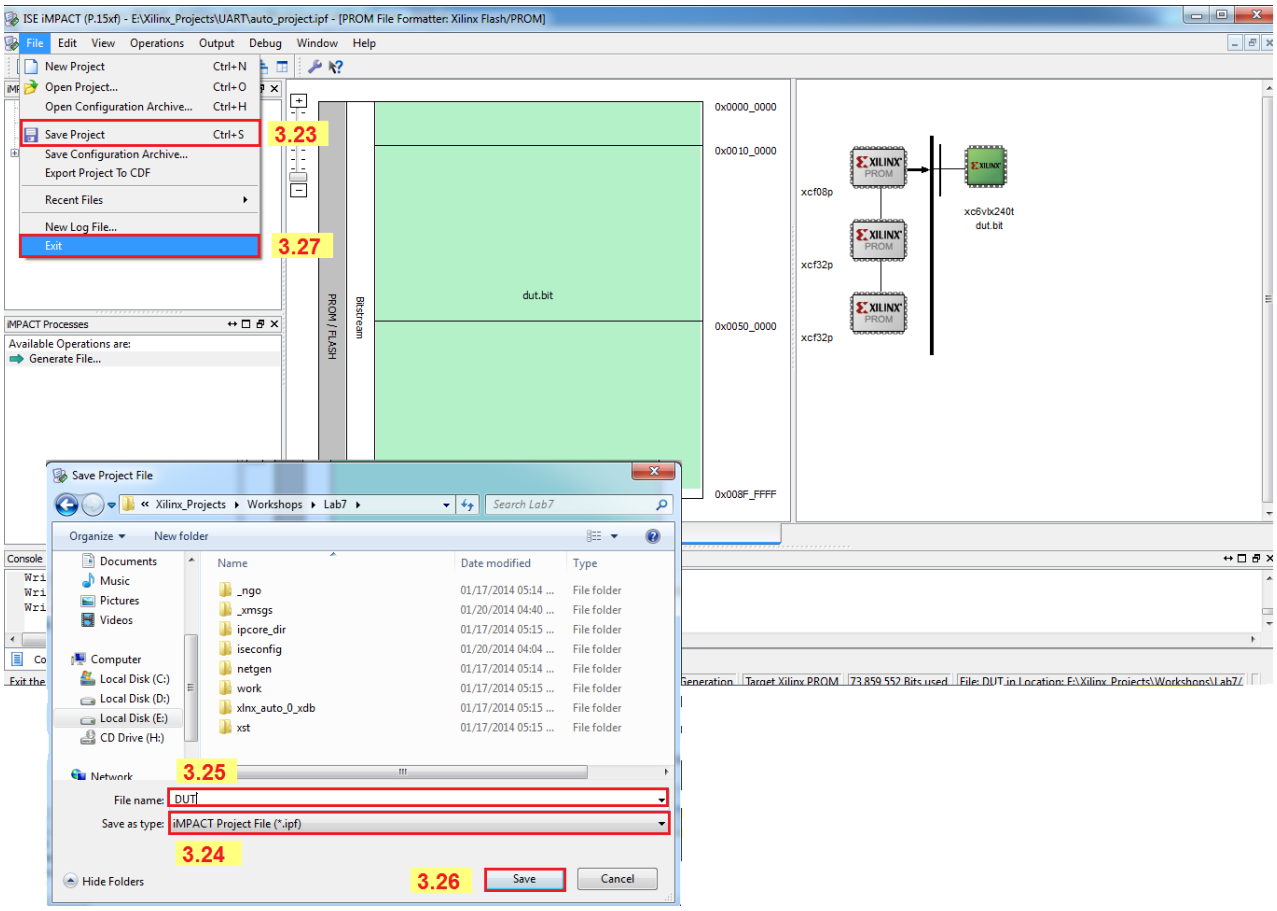

Figure 14.

3.23. In the iMPACT menus, select **File > Save**.

3.24. In the Save as type drop-down menu, select **iMPACT Project File (*.ipf)**.

3.25. Enter a name in the File name field.

3.26. Click **Save**.

3.27. Select **File > Exit**, to close iMPACT.

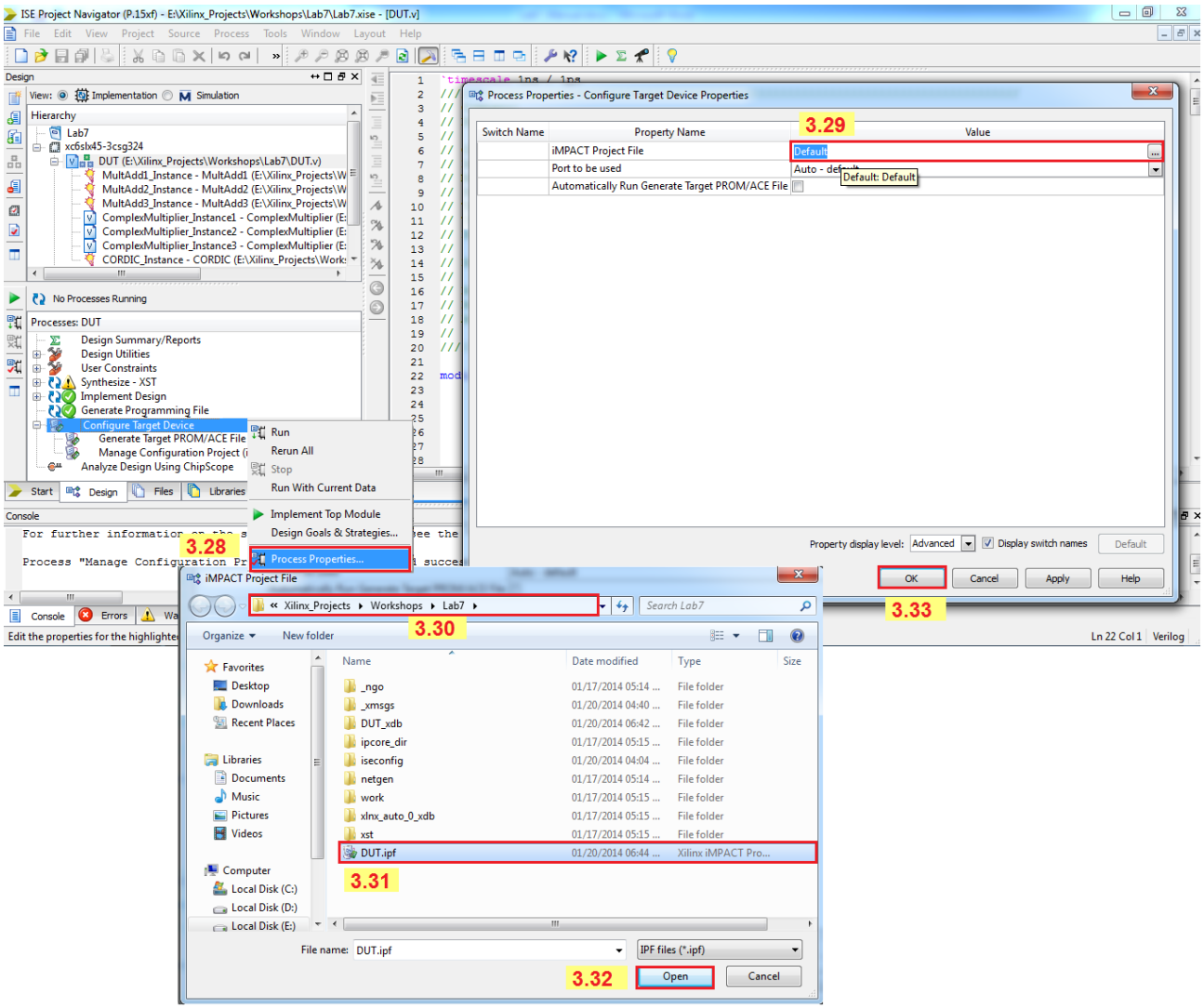

Figure 15.

3.28. In the Processes pane, right-click **Configure Target Device** and select **Process Properties.**

3.29. Click **Brows** button.

3.30. Go to the project directory.

3.31. Select **DUT.ipf**.

3.32. Click **Open**.

3.33. Click **OK**.

NOTE 1:

In this lab, you only generate the PROM File and perform the process of file generation completely, but you do not program the FPGA via the PROM. You'll program the FPGA using iMPACT tool in JTAG mode, which is described in section 6 of this manual.

NOTE 2:

In this lab, you'll program the FPGA for the second design (i.e. **Timer**). This is due to the test limitations and available features of the test platform (i.e. ATLYS board).

4. New Project Creation (Figure 16)

4.1. Design Specification

In this step of Lab 8, you should create a new project to implement a **Timer**. The block diagram of the targeted design is shown in Figure 16. The **Timer** block is a simple counter. The output value of the timer (i.e. **Timer Out**) increases per second (like a real clock). The **Timer** block has the following input/output ports:

- **Clk:** This is the main clock signal of the design, which is generated by the on-board oscillator.
- **Reset:** This is an active-high synchronous signal, which resets all of the internal registers. Thus, if **Reset =1**, all of the 8-LEDs should be turned **OFF.**
    - *Note that the **Reset** signal has a higher priority that the **Active** signal.*
- **Active:** This is an active-high synchronous signal, which controls the **Timer** operation. If **Active = 0**, the **Timer** block works normally. This means that the LEDs show the value of **Timer Out**, which increases per second. If **Active = 1,** the timer outputs should be set to 8'b 11111111. In other words, any time the user assert the **Active** signal (i.e. **Active = 1**), all of the 8-LEDs should be turned **ON**.
- **Timer Out**: This is the only output port of the **Timer** block with 8-bits width. The output value of the **Timer** increases per second (like a real clock).

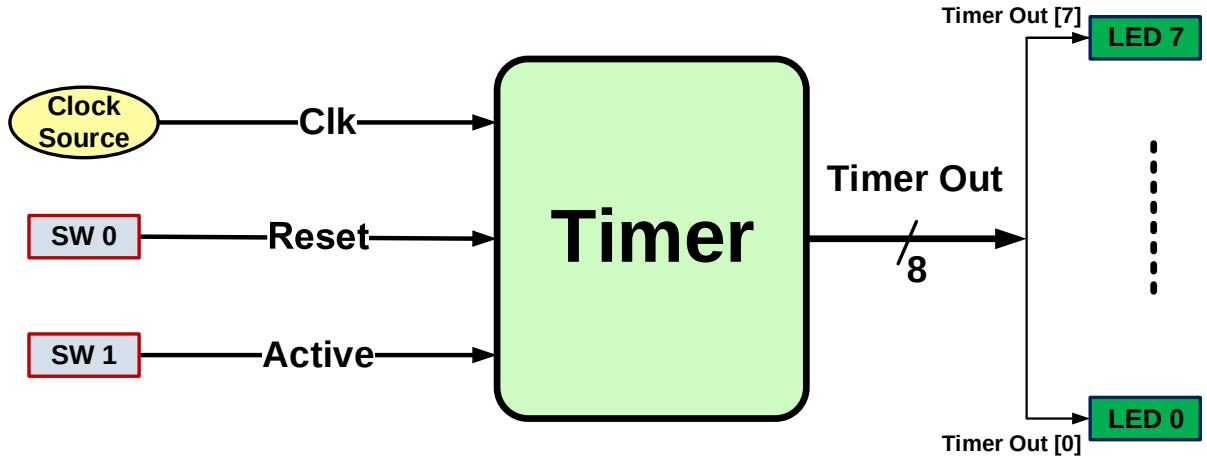

Figure 16. Block diagram of the second design.

The **Reset** and **Active** signals should be connected to SW0 and SW1, respectively. The **Timer Out** signal should be connected to the 8-LEDs on the ATLYS board.

Now, you should write a Verilog code to implement the above design completely.

4.2. Synthesizing and Implementing the Design

After writing the Verilog code for the proposed design, you should perform the following steps to complete the design flow:

4.2.1. Synthesize the design based on the instruction given in **Lab1**.

4.2.2. Perform the behavioral simulation (Refer to the **Lab2**). This is an **optional** step.

4.2.3. Enter the proper constraints to your design, based on the instruction given in **Lab4**. In this project, entering the timing constraint is optional, but you must enter the placement constraint. Thus, please refer to the ATLYS documentation and find the corresponding information about the pin locations. You must enter the proper placement constraint for ClK, SW0, SW1, and LEDs.

4.2.4. Implement the design completely. You can get more information from **Lab6**.

5. Generating Programming File

In this step, you should generate a programming file or configuration bitstream to download it into a target FPGA. Thus, please refer to section 2 of the current lab manual and follow the instructions. After completing this step, you should have an error-free bitstream file for the design.

Now, you can go to the final step and configure the FPGA using the generated bitstream to realize your design. Finally, you can test the **Timer** on the ATLYS board in different conditions. You can check its functionality for different values of **Reset** and **Active** signals.

6. FPGA Configuration using iMPACT in Boundary-Scan Mode (Figure 17-21)

Boundary-Scan configuration mode enables you to perform Boundary-Scan operations on any chain comprising JTAG compliant devices. In this section, you will use the Boundary-Scan configuration mode to configure the FPGA as follow:

6.1. Connect the Cable:

- Prior to launching iMPACT, connect the USB cable to one of your computer's USB ports, and connect the other end to the USB port of the FPGA board. Be sure that the board is powered. According to your FPGA board, you can use the other download cables for FPGA configuration.

6.2. Open iMPACT in one of the following methods:

- *Start iMPACT from Project Navigator:*
  - 6.2.1. Select **Design Panel**.
  - 6.2.2. Set the Design View to the **Implementation**.
  - 6.2.3. In the Hierarchy pane, select the **top module** (i.e. **DUT.v**).
  - 6.2.4. In the Process pane, expand **Configure Target Device**.
  - 6.2.5. In the Processes pane of the Design panel, double-click **Manage Configuration Project (iMPACT)**
- *Start iMPACT Standalone without going through an ISE project:*

  Click **Start > All Programs > Xilinx Design Tools > ISE Design Suite 14.1> ISE Design Tools > Tools > 64bit-Tools> iMPACT.**

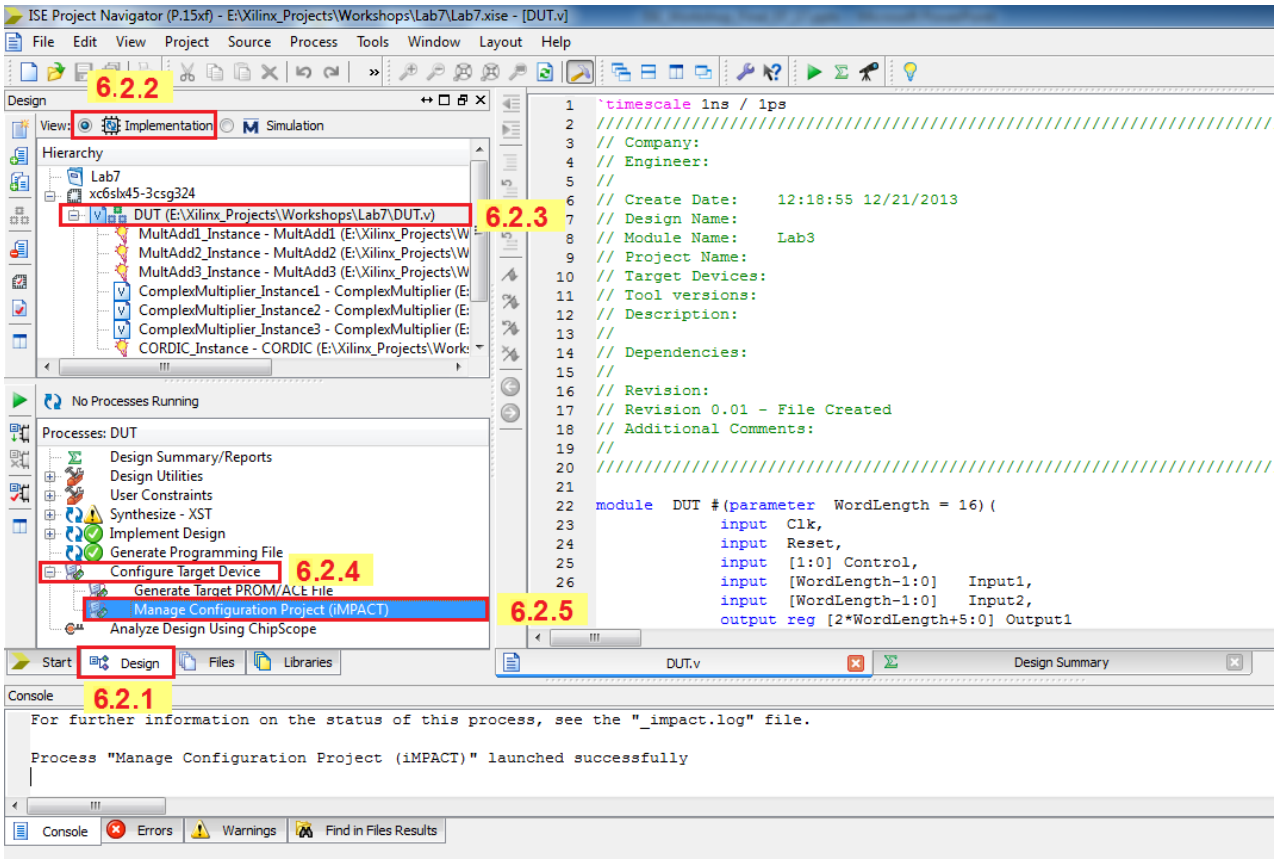

Figure 17.

6.3. Select **File > New Project**.

6.4. In the Automatically create and save a project dialog box, select **Yes**.

6.5. In the Welcome to iMPACT dialog box, select **Configure Devices using Boundary-Scan (JTAG)**.

6.6. Select the **Automatically connect to a cable and identify Boundary-Scan chain** from the drop-down menu.

6.7. Click **OK**.

6.8. After initializing a chain, the software prompts you for a configuration file, which is used to program the device. So, go to the project working directory.

6.9. Select the **BIT file**.

6.10. Click **Open**.

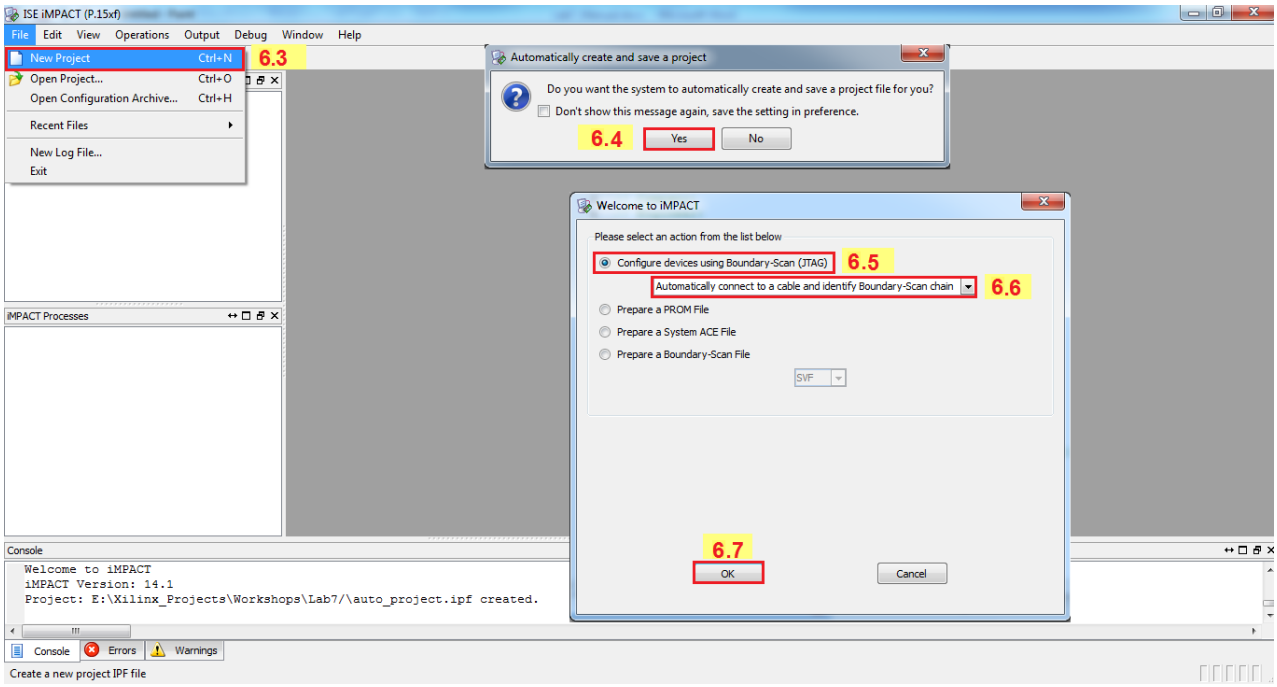

Figure 18.

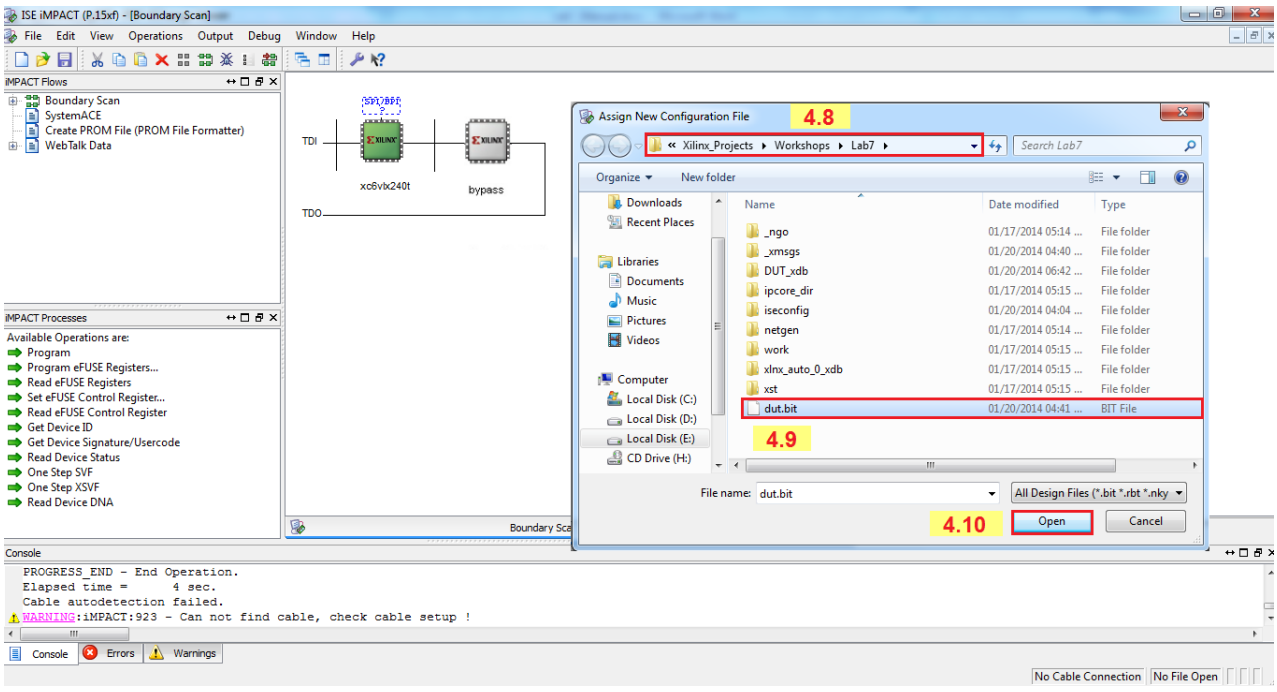

Figure 19.

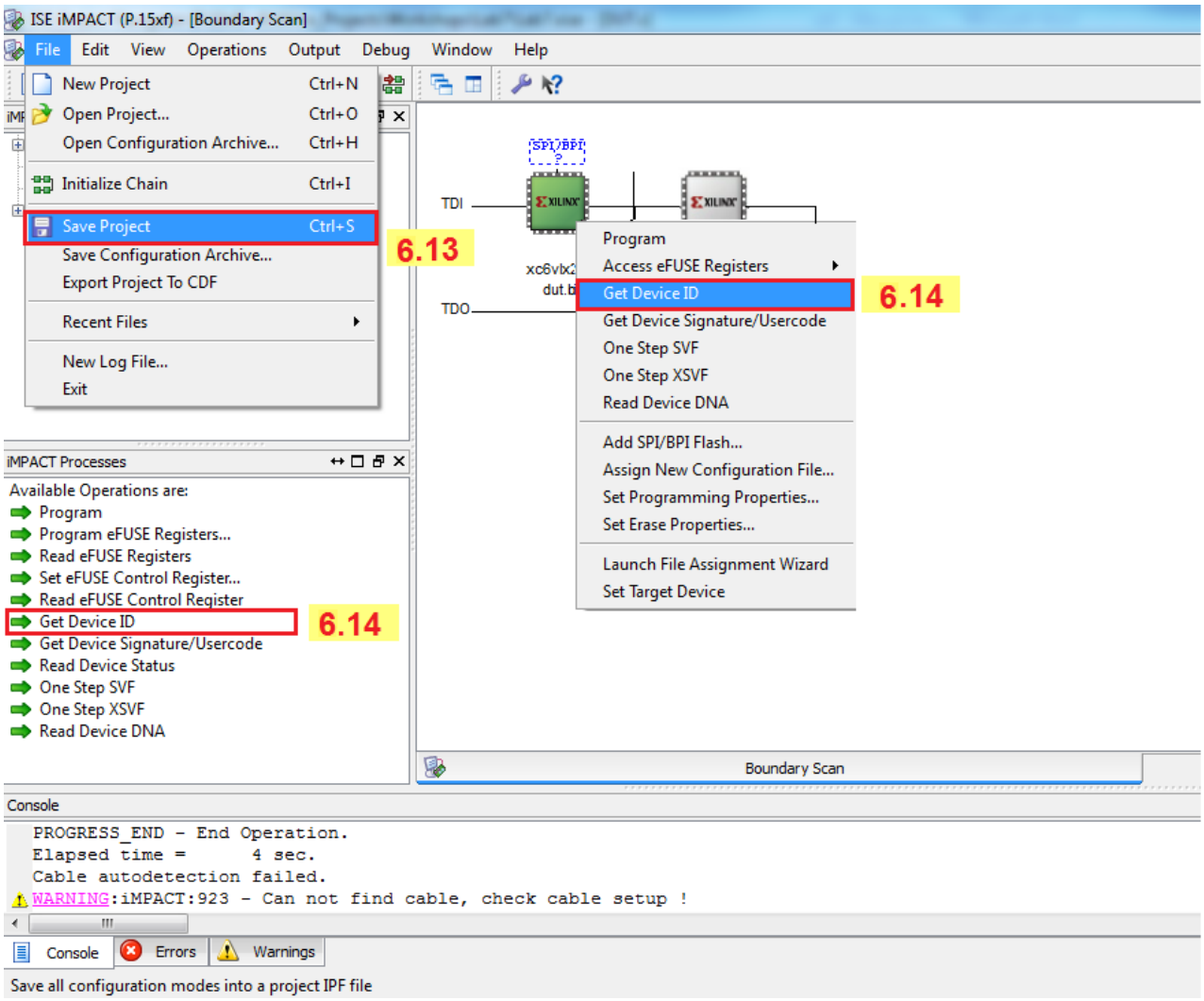

Figure 20.

6.11. Perform this step if you have a PROM in your configuration chain. When the software prompts you to select a configuration file for the second device, select the **MCS file** from your project working directory.

6.12. Click **Open**.

6.13. Select **File > Save Project,** after the chain has been fully described and configuration files are assigned. This iMPACT Project File (IPF) will be used later.

6.14. Right-click on the **FPGA device**, and select **Get Device ID**. The software accesses the IDCODE for this FPGA device. Also, you can select **Get Device ID** from iMPACT Processes window. The result is displayed in the log window.

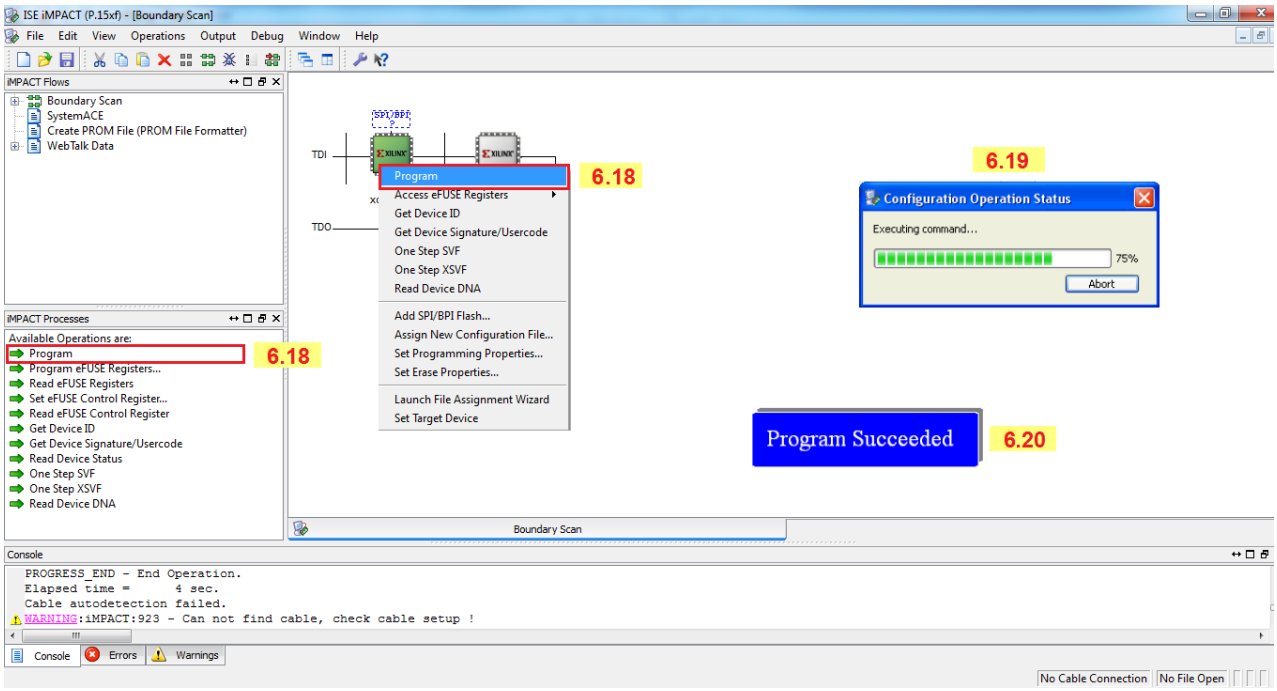

Figure 21.

6.15. Right-click on the **FPGA device**, and select **Set Programming Properties**.

6.16. Select the **Verify** option.

The Verify option enables the device to be read back and compared to the BIT file using the MSK file that was created earlier.

6.17. Click **OK** to begin programming.

6.18. Right-click on the **FPGA device**, and select **Program**. Also, you can select **Program** from iMPACT Processes window.

6.19. The Programming operation begins and an operation status window appears. At the same time, the log window reports all of the operations being performed.

6.20. When the Program operation completes, a large blue message appears showing that programming was successful. This message disappears after a few seconds.

7. File Generation

The method of generating the **Programming file** (i.e. ***.bit file**) was described in Section 2 of this lab. Also, The method of generating the **PROM file** (i.e. ***.MCS file**) was described in Section 3. Moreover, iMPACT enables you to generate **Boundary Scan or JTAG Vector Files** (i.e. **SVF, XSVF, or STAPL**) and **System ACE** files for your design. These processes are not covered in this manual, which are similar to the PROM file generation.