San Jose State University
Department of Electrical Engineering
EE178, Spring 2006, Crabill

Laboratory Assignment #1

Objectives

This lab is an introduction to logic design using Verilog-HDL with the Xilinx ISE 6.3i tools. No new logic design concepts are presented in this lab. The goals of this lab are for you to become familiar with the tools you will be using for the rest of the semester:

- Xilinx's ISE Project Navigator tool for Verilog-HDL.
- Xilinx's Spartan-3 Starter Kit.
- Model Technology's Modelsim simulator for Verilog-HDL.

Consider this lab a "no-brainer" warm up for the next labs. In previous semesters, I have heard students make wistful remarks such as, "I wish I had paid more attention to lab one…" Please read carefully, pay attention, and take your time. This lab is not a race to see who gets done first.

In order to receive credit for this lab, you must demonstrate to the instructor that your final design works correctly in hardware. The details of the required demonstration are at the end of the lab handout. For this lab, no report is due.

Bibliography

This lab draws heavily from documents on the Xilinx website http://www.xilinx.com. The lab leverages content from the *ISE 6 In-Depth Tutorial*, the *ISE 6 Quick Start Tutorial*, and the *Spartan-3 Starter Kit User Guide*. I would like to thank Xilinx for making this material available. This lab is effectively a customized tutorial using Verilog-HDL and the Spartan-3 Starter Kit.

Project Navigator Overview

The Project Navigator is divided into four main sub-windows, as seen in Figure 1. On the top left is the Sources in Project window which hierarchically displays the elements included in the project. Beneath the Sources in Project window is the Processes for Current Source window which displays available processes for the currently selected source. The third window at the bottom of the Project Navigator is the Console window which displays status messages, errors, and warnings, and which is updated during all project actions. The fourth window to the right is for viewing and editing text files. Each window may be resized, undocked from Project Navigator or moved to a new location within the main Project Navigator window. The default layout can always be restored by selecting View→Restore Default Layout.

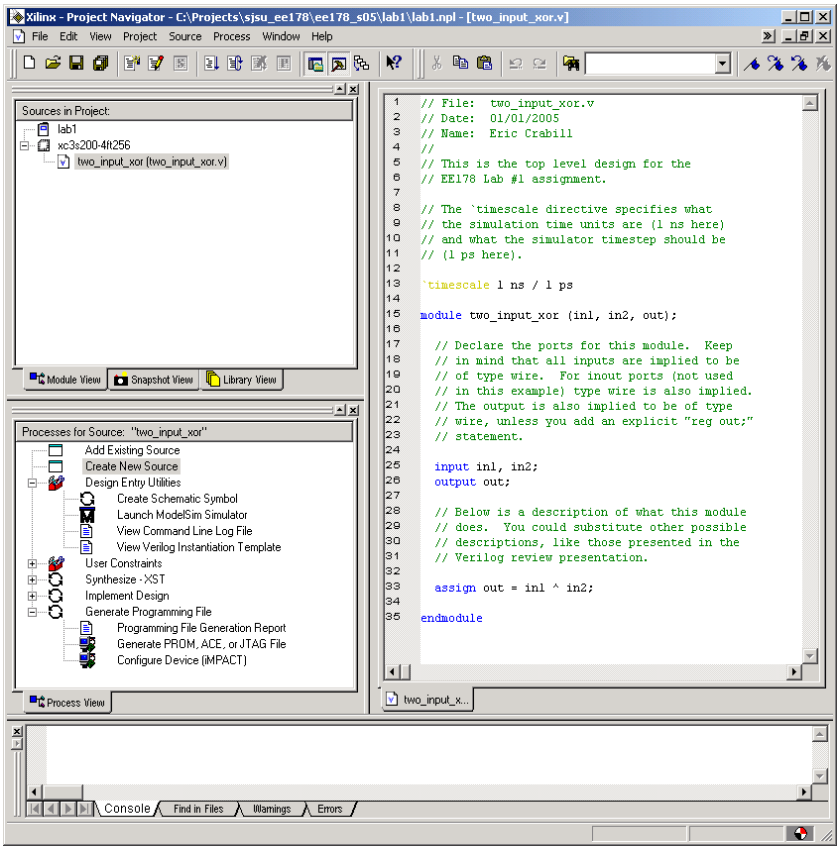

Figure 1: Typical Project Navigator Window

The Sources in Project window consists of three tabs which provide information for the user. Each tab is discussed in further detail below:

- The Module View tab displays the project name, any user documents, the specified part type and design flow/synthesis tool, and design source files. Each file in the Module View has an associated icon. The icon indicates the file type (Verilog-HDL file or text file, for example). For a complete list of possible source types and their associated icons, see the Project Navigator online help. Select Help→ISE Help Contents, select the Index tab and click Source / File types. If a file contains lower levels of hierarchy, the icon has a + to the left of the name. Verilog-HDL files have this + to show the modules within the file. You can expand the hierarchy by clicking the +. You can open a file for editing by double-clicking on the filename.
- The Snapshot View tab displays all snapshots associated with the project currently open in Project Navigator. A snapshot is a copy of the project including all files in the working directory, and synthesis and simulation subdirectories. A snapshot is stored with the project for which it was taken, and can be viewed in the Snapshot View. You can view the reports, user documents, and source files for all snapshots. All information displayed in the Snapshot View is read-only. Using snapshots provides an excellent version control system.
- The Library View tab displays all libraries associated with the project open in Project Navigator.

The Processes for Current Source window contains the Process View tab. The Process View tab is context sensitive and changes based upon the source type selected in the Sources for Project window. From the Process View tab, you can run the functions necessary to define, run and view your design. The Process View tab provides access to the following functions:

- Design Entry Utilities. Provides access to symbol generation, instantiation templates, HDL Converter, Command Line Log Files, Launch MTI, and simulation library compilation.

- User Constraints. Provides access to editing location and timing constraints.
- Synthesis. Provides access to Check Syntax, Synthesize, View RTL Schematic, and synthesis reports. This varies depending on the synthesis tools you use.
- Implement Design. Provides access to implementation tools and design flow reports.
- Generate Programming File. Provides access to the configuration tools and bitstream generation.

The Processes for Current Source window incorporates automake technology. This enables the user to select any process in the flow and the software automatically runs the processes necessary to get to the desired step. For example, when you run the Implementation process, Project Navigator also runs the synthesis process, if necessary, because implementation is dependent on up-to-date synthesis results.

The Console window displays errors, warnings, and informational messages. Errors are signified by a red box next to the message, while warnings have a yellow box. Warning and Error messages may also be viewed separately from other console text messages by selecting either the Warnings or Errors tab at the bottom of the console window.

You can navigate from a synthesis error or warning message in the Console window to the location of the error in a source Verilog-HDL file. To do so, select the error or warning message, right-click the mouse, and from the menu select Goto Source. The Verilog-HDL source file opens and the cursor moves to the line with the error.

You can also navigate from an error or warning message in the Console window to the relevant solution records on the Xilinx support website. These types of errors or warnings can be identified by the web icon to the left of the error. To navigate to the solution record, select the error or warning message, right-click the mouse, and from the menu select Goto Solution Record. The default web browser opens and displays all solution records applicable to this message.

In the fourth window, you can access the ISE Text Editor, the ISE Language Templates, and HDL Bencher Text Editor. The ISE Text Editor enables you to edit source files and to access the ISE Language Templates, which is a catalog of Verilog-HDL and User Constraint File templates. You can use and modify these templates for your own design.

Design Entry

The design used in this tutorial is a simple two-input XOR. The design will be described in Verilog-HDL. Double-click the Project Navigator icon on your desktop or select Start→Programs→Xilinx ISE→Project Navigator. From Project Navigator, select File→New Project. The first of the New Project dialog boxes will appear, as shown in Figure 2.

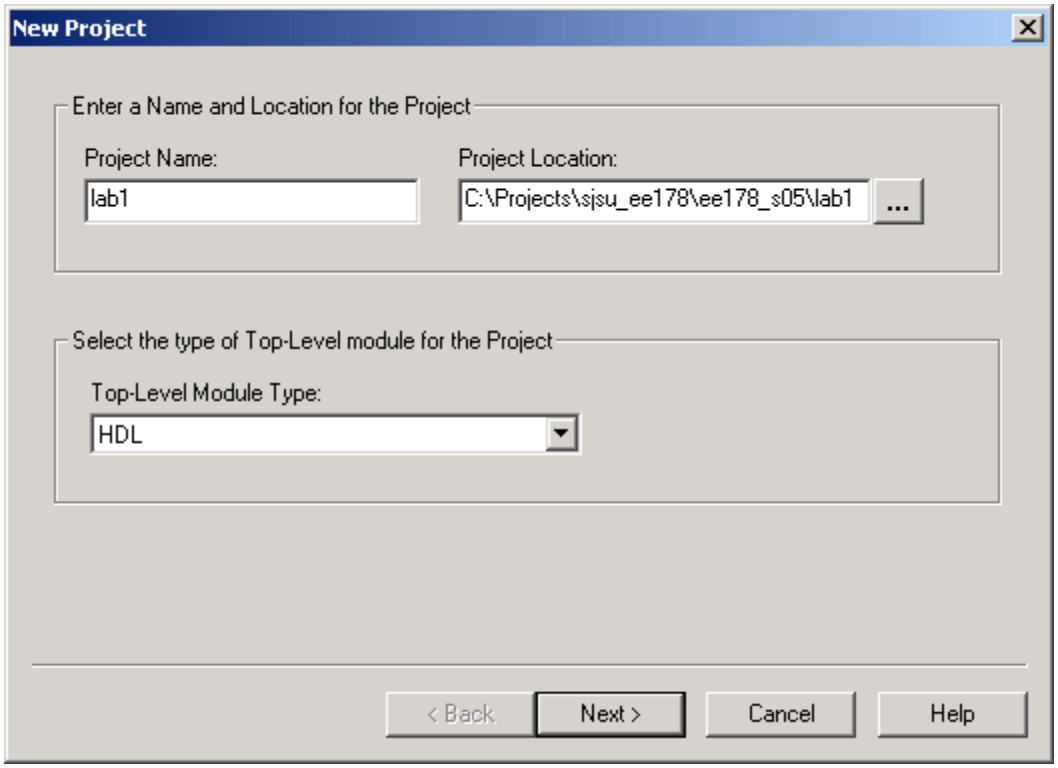

Figure 2: New Project Dialog 1 of 5

You are prompted to enter a project name, a project location, and a top level module type, as shown in Figure 2. You may change the project location to another folder if you wish. Do not use file or folder names that contain spaces. I advise all students to purchase a USB memory stick and store their work on removable media. Even if you are doing most of your work from home, you must have some means to transport your project to the lab if you need help debugging it. Never store your projects on the lab machines. When you are satisfied with the project name and location, click "Next".

The next dialog allows you to set additional project options. The first group of settings shown in Figure 3 represents the hardware target that is available to you on the Spartan-3 Starter Kit board. The second group of settings represents the design entry language, synthesis tool, and simulator preferences. Set the options as shown in Figure 3 and click "Next".

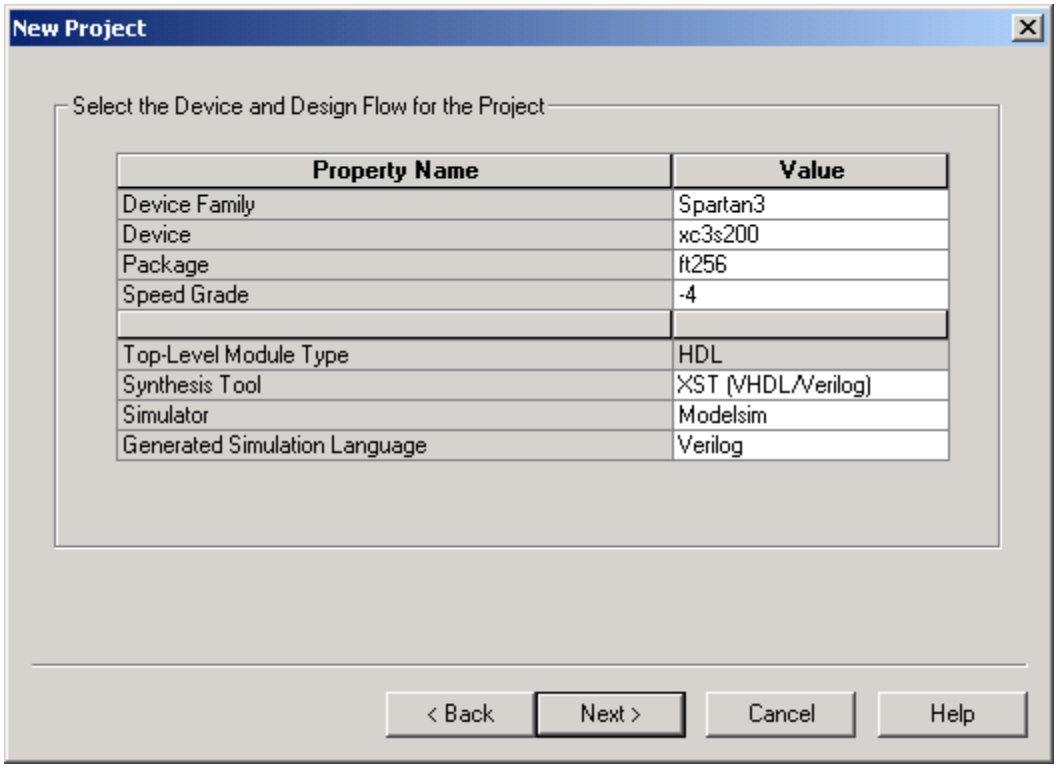

Figure 3: New Project Dialog 2 of 5

The following dialog box of Figure 4 gives you the opportunity to create new source files as part of the new project process. Do not create new source files at this time, simply click "Next" to proceed.

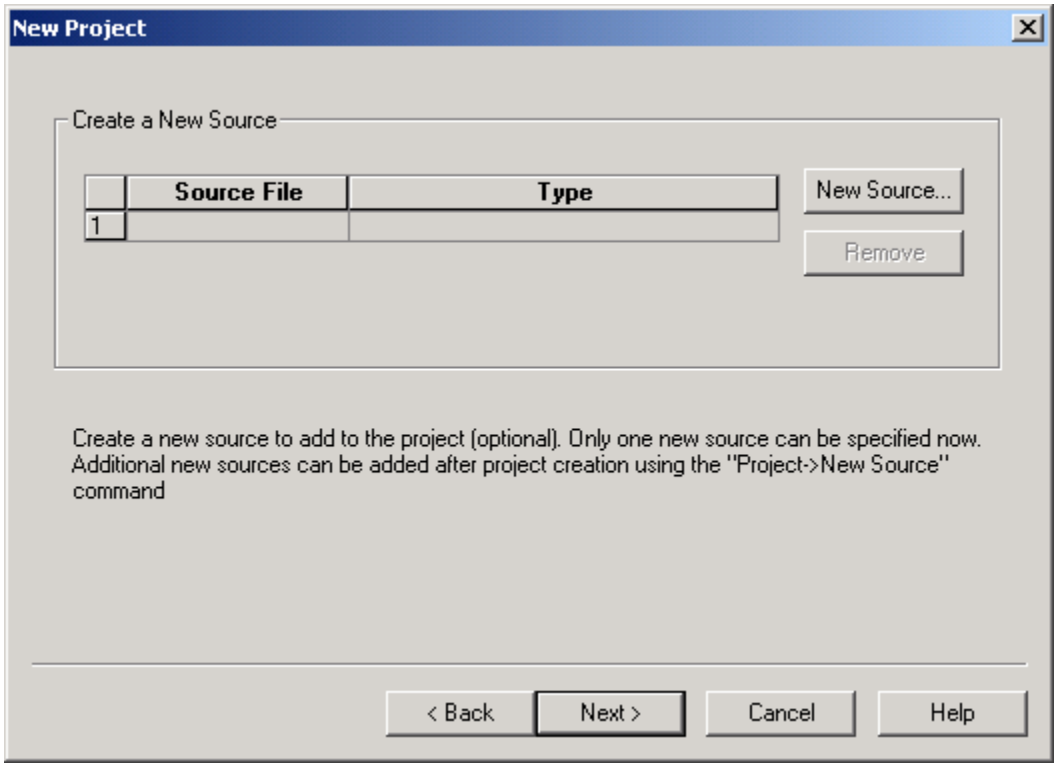

Figure 4: New Project Dialog 3 of 5

The following dialog box of Figure 5 gives you the opportunity to add existing source files as part of the new project process. Do not add existing source files at this time, simply click “Next” to proceed.

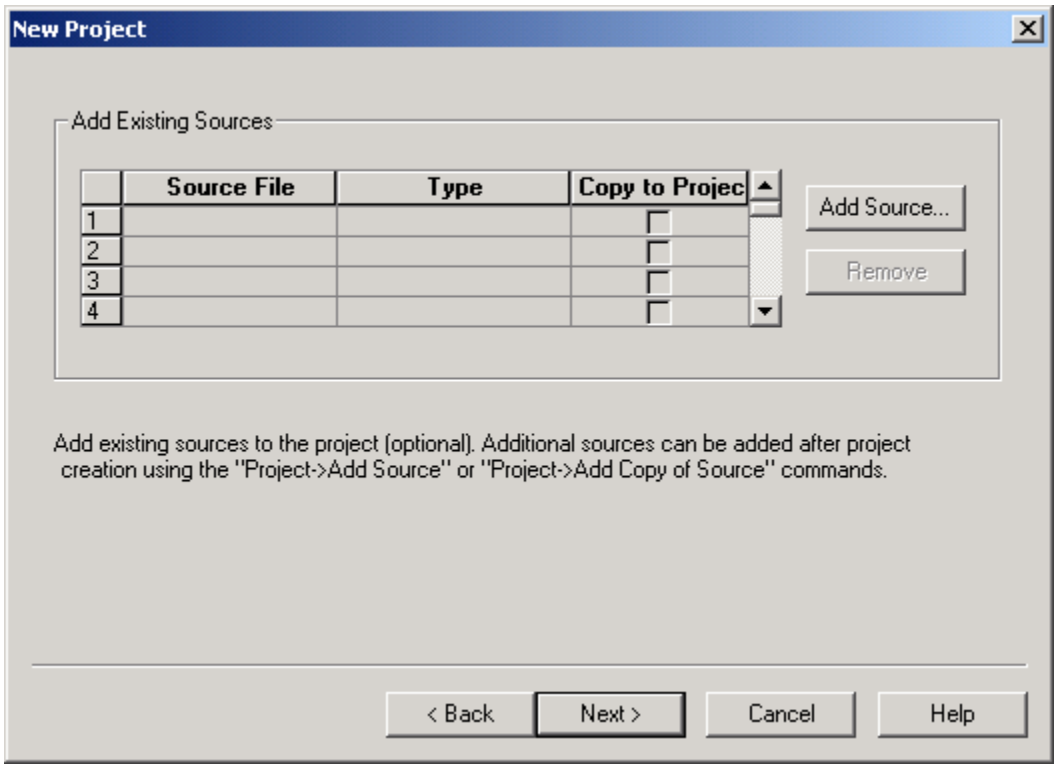

Figure 5: New Project Dialog 4 of 5

The final dialog box in the new project process, shown in Figure 6, provides a summary of the project that Project Navigator will create based on your settings. Review the summary to make sure it matches what is shown in Figure 6. If it does not, go “Back” and correct any errors. Otherwise, click “Finish” to complete this process.

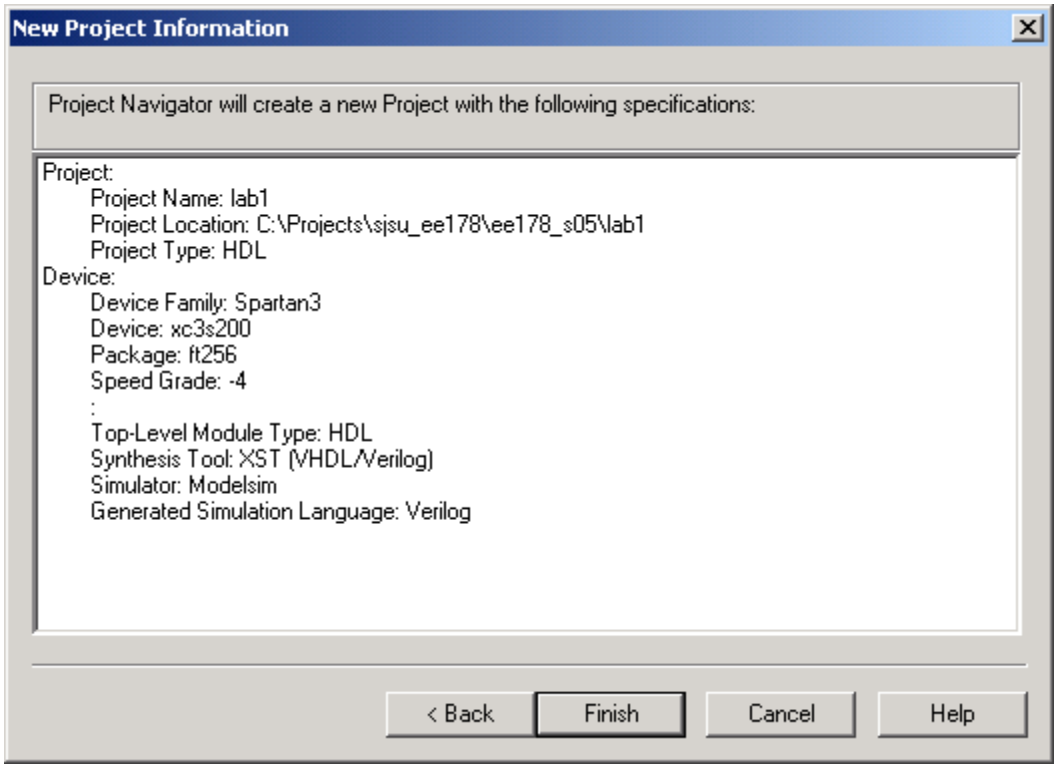

Figure 6: New Project Dialog 5 of 5

At this point, the project has been created but it does not contain any source files. Create a new source file for the two-input XOR. Either select Project→New Source from the main menu or use the equivalent process in the Processes for Current Source window. The first of the New Source dialog boxes will appear, as shown in Figure 7.

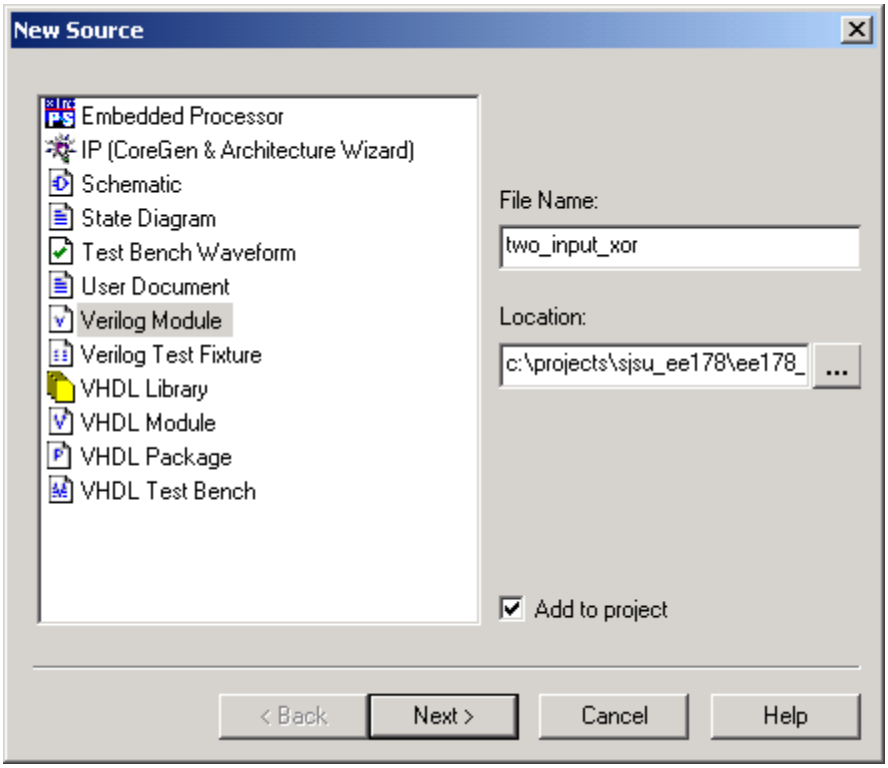

Figure 7: New Source Dialog 1 of 3

Select Verilog Module to indicate you are creating a Verilog-HDL design module. Then, provide a file name as shown in Figure 7. You should not need to change the specified location, which should be inside the project directory you created earlier. Click “Next”.

The next dialog optionally allows you to specify the ports of the module. This may also be done in the text editor, when creating the module, so skip it at this stage. Simply confirm that the settings match those shown in Figure 8 and click “Next”.

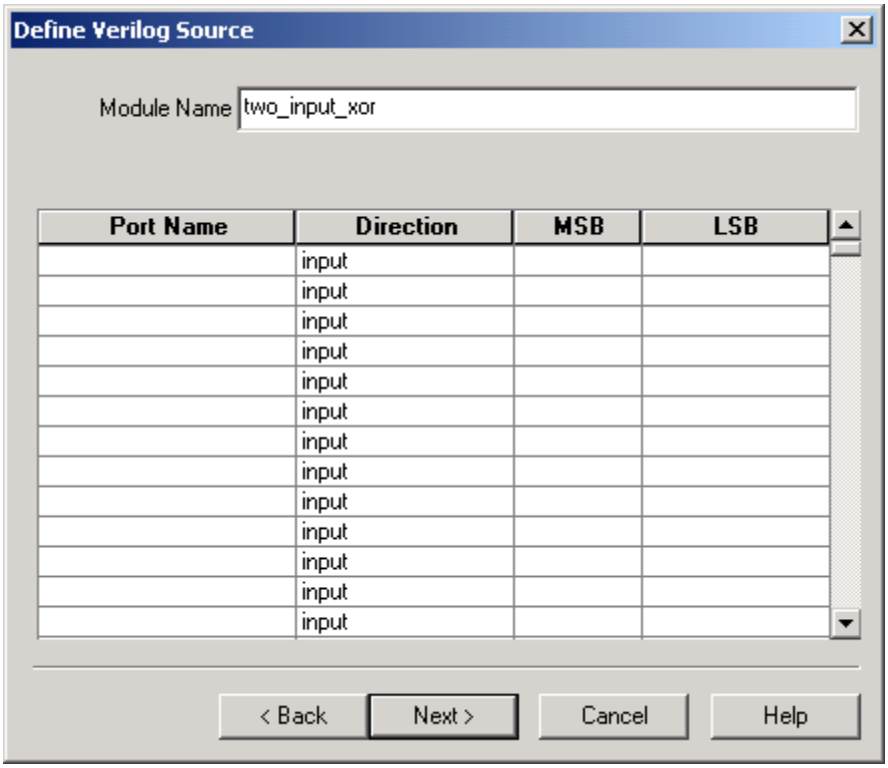

Figure 8: New Source Dialog 2 of 3

The final dialog box of Figure 9 provides a summary of the source that Project Navigator will create based on your settings. Review the summary to make sure it matches what is shown in Figure 9. If it does not, go "Back" and correct any errors. Otherwise, click "Finish" to complete this process. The new source file will be automatically opened in the text editor.

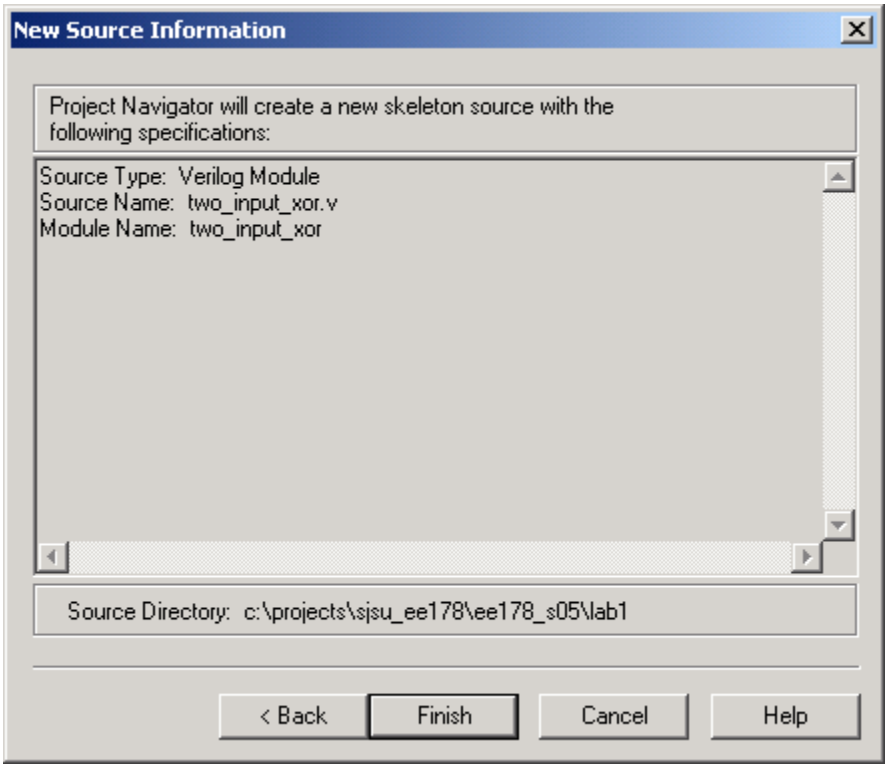

Figure 9: New Source Dialog 3 of 3

In the text editor, some of the basic file structure is already in place. Keywords are displayed in blue, data types in red, comments in green, and values in black. This color-coding enhances readability and recognition of typographical errors. Now, enter the two-input XOR design. You may be able to simply copy and paste this from the lab handout, but if that doesn't work, transcribe the contents:

```
// File:  two_input_xor.v
// Date:  01/01/2005
// Name:  Eric Crabill
//
// This is the top level design for the
// EE178 Lab #1 assignment.

// The `timescale directive specifies what
// the simulation time units are (1 ns here)
// and what the simulator timestep should be
// (1 ps here).

`timescale 1 ns / 1 ps

module two_input_xor (in1, in2, out);

  // Declare the ports for this module.  Keep
  // in mind that all inputs are implied to be
  // of type wire.  For inout ports (not used
  // in this example) type wire is also implied.
  // The output is also implied to be of type
  // wire, unless you add an explicit "reg out;"
  // statement.
```

```
  input in1, in2;
  output out;

  // Below is a description of what this module
  // does.  You could substitute other possible
  // descriptions, like those presented in the
  // Verilog review presentation.

  assign out = in1 ^ in2;

endmodule
```

At this point, you should end up with a window that looks somewhat like that shown in Figure 10. Once you are satisfied, save the file and close the window. It is a good idea to get in the habit of saving your project. There are options on the main menu to save individual files or the complete project.

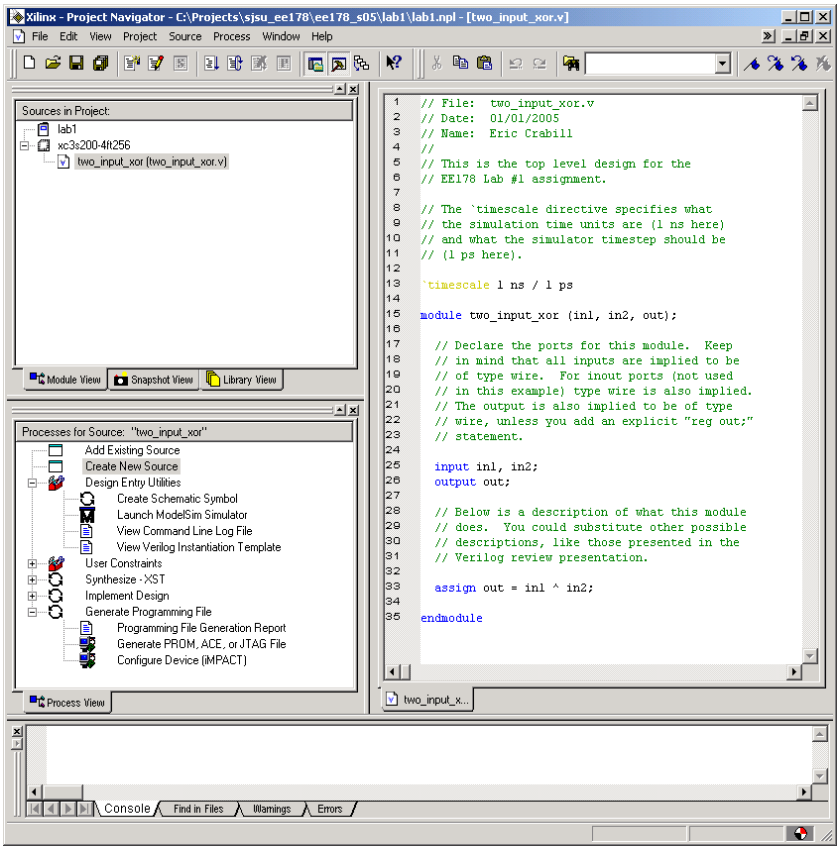

Figure 10: Completed Design

Functional Simulation

Functional simulation is done before the design is synthesized to verify that the logic you have created is correct. This allows a designer to find and fix any bugs in the design before spending time with subsequent steps. Project Navigator provides an integrated flow with the Modelsim simulator that allows simulations to be run from the Project Navigator. In order to simulate the design, a test bench is required to stimulate the design. Create a new source file for the test bench. Either select Project→New Source from the main menu or use the equivalent process in the Processes for Current Source window. The first of the New Source dialog boxes will appear, as shown in Figure 11.

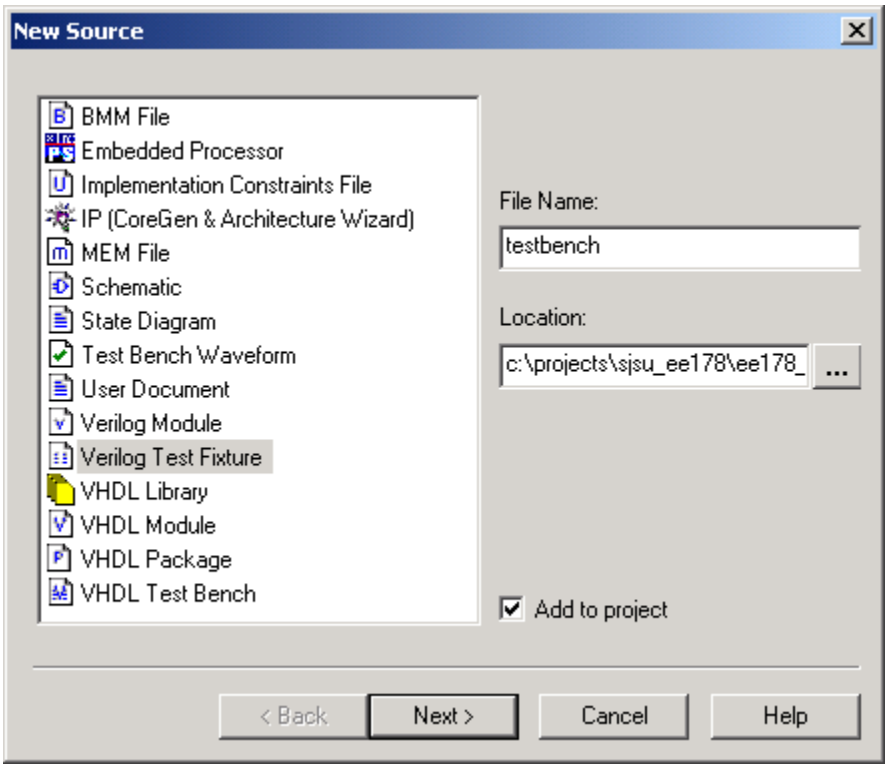

Figure 11: New Source Dialog 1 of 3

Select Verilog Test Fixture to indicate you are creating a Verilog-HDL testbench module. Then, provide a file name as shown in Figure 11. You should not need to change the specified location, which should be inside the project directory you created earlier. Click “Next”.

The second dialog, shown in Figure 12, asks you to identify a design module with which the test bench should be associated. Select the two-input XOR design as shown and click “Next”.

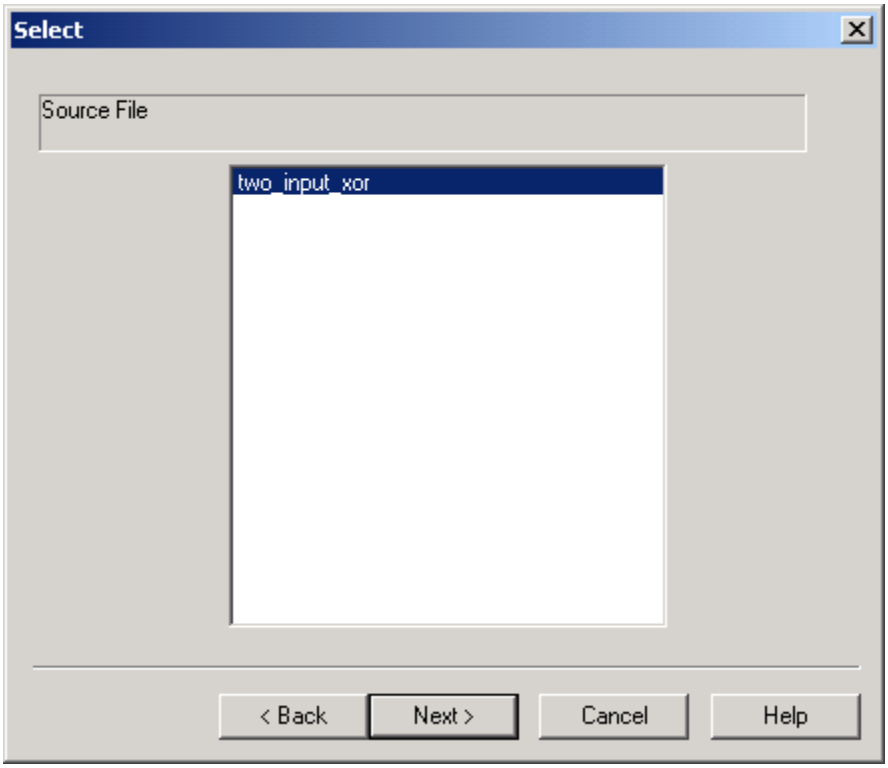

Figure 12: New Source Dialog 2 of 3

The final dialog box of Figure 13 provides a summary of the source that Project Navigator will create based on your settings. Review the summary to make sure it matches what is shown in Figure 13. If it does not, go "Back" and correct any errors. Otherwise, click "Finish" to complete this process. The new source file will be automatically opened in the text editor.

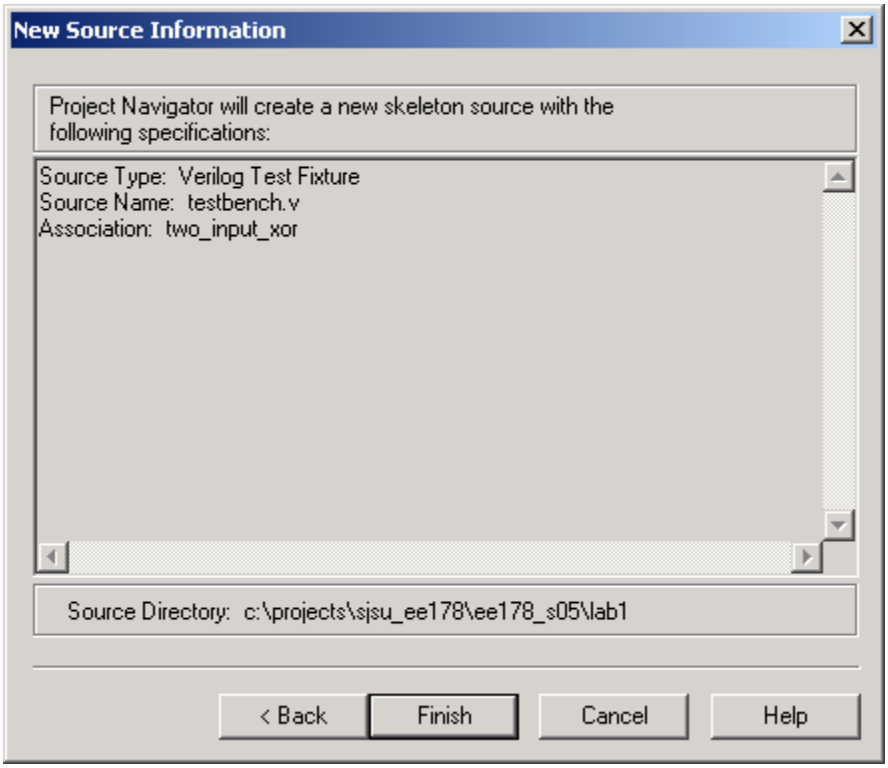

Figure 13: New Source Dialog 3 of 3

In the text editor, some of the basic file structure is already in place. Keywords are displayed in blue, data types in red, comments in green, and values in black. This color-coding enhances readability and recognition of typographical errors. Now, enter the test bench for the two-input XOR design. You may be able to simply copy and paste this from the lab handout, but if that doesn't work, transcribe the contents:

```
// File:  testbench.v
// Date:  01/01/2005
// Name:  Eric Crabill
//
// This is a top level testbench for the
// two_input_xor design, which is part of
// the EE178 Lab #1 assignment.

// The `timescale directive specifies what
// the simulation time units are (1 ns here)
// and what the simulator timestep should be
// (1 ps here).

`timescale 1 ns / 1 ps

module two_input_xor_testbench_v_tf;

  // Declare a wire to be driven by the output
  // of the two_input_xor design.  Also declare
  // two regs to drive the input of the design.
  // These two regs may be assigned values by
  // a behavioral stimulus.

  wire sig3;
```

```
reg sig1, sig2;

// Instantiate the two_input_xor design module.

two_input_xor my_xor (.in1(sig1),.in2(sig2),.out(sig3));

// Assign values to the input signals and
// check the output results.  This example
// is meant to illustrate the concept of a
// self-checking testbench, not to suggest
// that you should feel the need to verify
// the correct behavior of logical operators.

reg test_passed;

initial
begin
  // Let's start off assuming we are going
  // to pass the tests until we find a case
  // that contradicts!
  test_passed = 1'b1;

  // Test Case #0
  sig1 = 1'b0;
  sig2 = 1'b0;
  #5;
  $display("At time %t, sig1 = %b, sig2 = %b, output = %b.",
           $time, sig1, sig2, sig3);
  if (sig3 != 1'b0) test_passed = 1'b0;

  // Test Case #1
  sig1 = 1'b0;
  sig2 = 1'b1;
  #5;
  $display("At time %t, sig1 = %b, sig2 = %b, output = %b.",
           $time, sig1, sig2, sig3);
  if (sig3 != 1'b1) test_passed = 1'b0;

  // Test Case #2
  sig1 = 1'b1;
  sig2 = 1'b0;
  #5;
  $display("At time %t, sig1 = %b, sig2 = %b, output = %b.",
           $time, sig1, sig2, sig3);
  if (sig3 != 1'b1) test_passed = 1'b0;

  // Test Case #3
  sig1 = 1'b1;
  sig2 = 1'b1;
  #5;
  $display("At time %t, sig1 = %b, sig2 = %b, output = %b.",
           $time, sig1, sig2, sig3);
  if (sig3 != 1'b0) test_passed = 1'b0;

  // Now, print out a message with the test
  // results and then finish the simulation.
  if (test_passed) $display("Result: PASS");
```

```
    else $display("Result: FAIL");
    $stop;
  end

endmodule
```

At this point, you should end up with a window that looks somewhat like that shown in Figure 14. Once you are satisfied, save the file and close the window. It is a good idea to get in the habit of saving your project. There are options on the main menu to save individual files or the complete project.

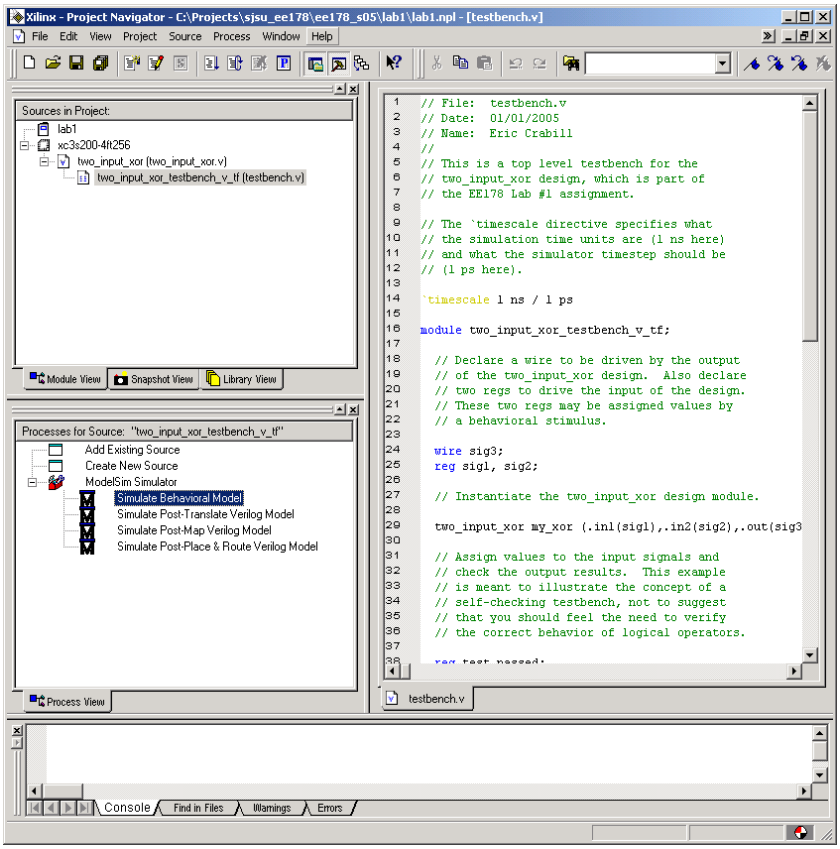

Figure 14: Completed Test Bench

Now that you have a test bench in your project, you can perform functional simulation on the design. The simulation processes enable you to run simulation on the design using Modelsim. To locate the Modelsim simulator processes, select the test bench in the Sources in Project window. Then, click the + next to the Modelsim Simulator entry in the Processes for Current Source window to expand the item, this is also shown in Figure 14. The following simulation processes are available:

- Simulate Behavioral Model. This process will start the design simulation.
- Simulate Post-Translate Verilog Model. Simulates the netlist after the NGDBuild stage.
- Simulate Post-Map Verilog Model. Simulates the netlist after the Map stage.
- Simulate Post-Place & Route Verilog Model. Simulates the netlist after Place & Route.

At this point, you will perform a functional simulation using Simulate Behavioral Model but you must specify the simulation process properties first. Right click on Simulate Behavioral Model, and select Simulation Properties. The Process Properties dialog box appears, as shown in Figure 15.

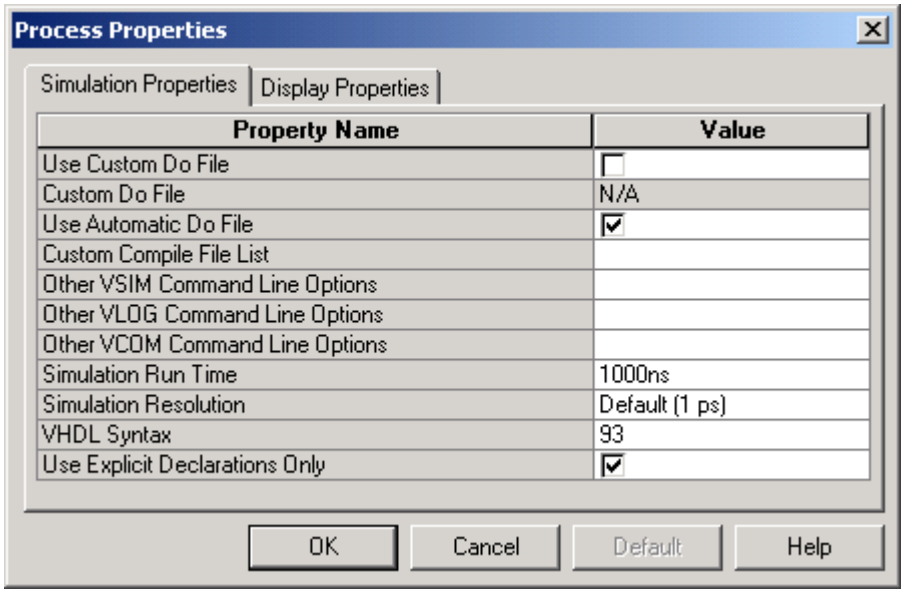

Figure 15: Simulation Process Properties

If you do not have all the properties shown in Figure 15, you can make them visible by canceling the dialog box, then selecting Edit → Preferences from the main menu. Select the Processes tab, set the Property Display Level to advanced, and then return to Simulation Properties. Make sure the properties are set as shown in Figure 15. The most interesting of these parameters is probably the simulation run time – 1000 ns is more than sufficient for the test bench in the project. For test benches that require more simulation time, this property should be adjusted as needed. Click "Ok".

To start the simulation, double-click Simulate Behavioral Model. Modelsim creates a work directory, compiles the source files, loads the design, and performs simulation for the time specified. Four Modelsim windows will appear. The first, and most important, is the main Modelsim console, shown in Figure 16. This window displays messages from the simulator. These messages include notes, warnings, and errors, plus any output created by the design being simulated. You should see text output from the test bench.

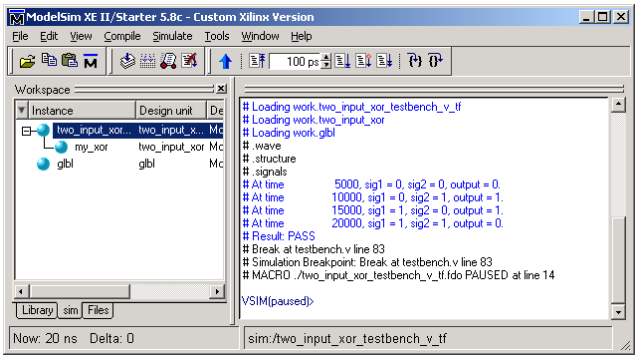

Figure 16: Modelsim Console Window

The second window is the structure window, shown in Figure 17. This window allows you to browse the hierarchy of the test bench and the design under test. In large hierarchical designs, it is very handy.

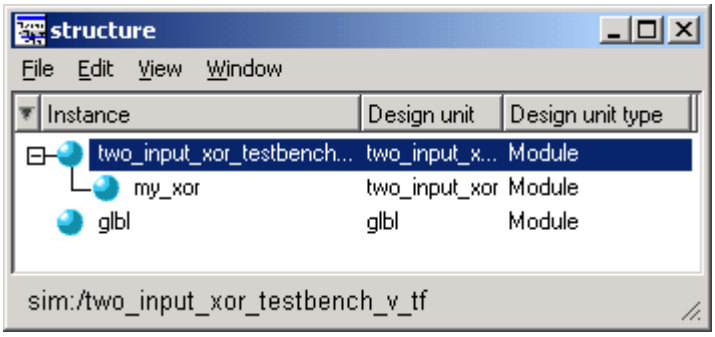

Figure 17: Modelsim Structure Window

The third window is the signals window, shown in Figure 18. This window shows the signals that are present in the portion of the design selected in the structure window.

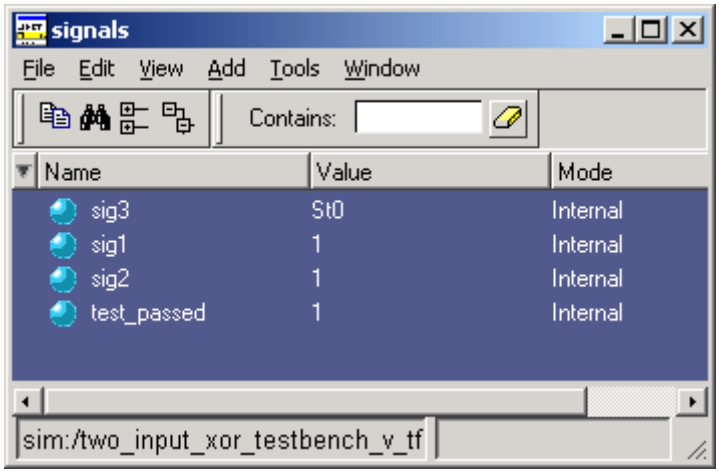

Figure 18: Modelsim Signals Window

The fourth and final window is the wave window, which is used to display simulated waveforms. Project Navigator automatically adds all top-level signals to the wave window, as shown in Figure 19. Additional signals are displayed in the signal window based upon the selected structure in the structure window.

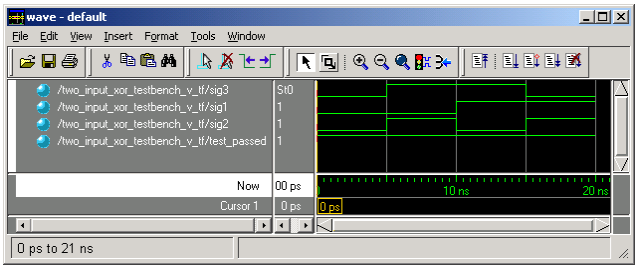

Figure 19: Modelsim Wave Window

There are two basic methods for adding signals to the wave window. You can drag and drop them from the signals window, or highlight them in the signals window and then select Add→Wave→Selected Signals. If you use this second technique, you will see that there are additional options available.

When you add new signals to the wave window, you will notice that waveforms do not automatically appear. This is because Modelsim did not record the simulation data for these signals. By default, Modelsim will only record data for the signals that have been added to the waveform window before or during the simulation. Therefore, when new signals are added to the waveform window, the simulation

needs to be restarted and re-run for the desired amount of time. To restart and re-run the simulation, click the "Restart Simulation" button at the top of the console window. This button is shown in Figure 20.

Figure 20: Restart Simulation Button

The Restart dialog box appears, as shown in Figure 21. Simply click "Restart". At the Modelsim prompt, you will need to manually enter the run command. Enter "run 1000 ns" and hit enter. The simulation will run again, just like it did the first time.

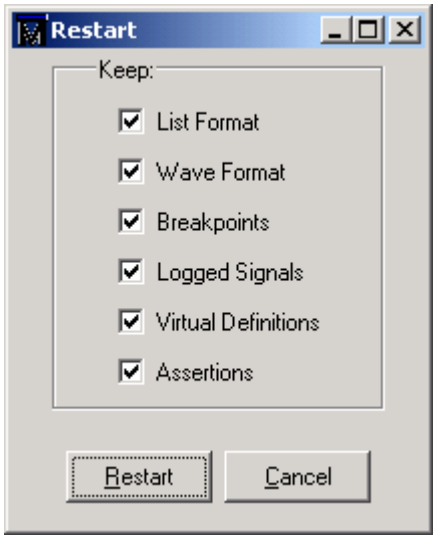

Figure 21: Restart Dialog

The Modelsim simulator provides the capability of saving the signals list in the wave window. This can be important when additional signals or stimuli are added, and the simulation is restarted. In the wave window, select File→Save Format. After restarting a simulation, you can select File→Load Format in the wave window to restore.

Design Synthesis

With a functionally correct design description in Verilog-HDL, the next step is to use a synthesis tool to transform your description into a netlist. A netlist is a machine-readable schematic. In this class, we will be using a tool called XST, which is integrated with Project Navigator and can only target Xilinx devices.

Select two_input_xor in the Sources in Project window. Then, double click on the Synthesize—XST process in the Processes for Current Source window. Project Navigator will synthesize the design and print information to the Console window in the process. As an informational note, it is possible to change the synthesis options before you synthesize by right clicking on Synthesize—XST and then selecting Properties. For this tutorial, however, leave the options at their default settings.

You should not see any errors in the Console window. However, you should always review the log file, which is available for viewing if you expand the Synthesize—XST process item by clicking on the + next to it. Select View Synthesis Report. If you don't understand a particular message, you should not simply ignore it. Instead, search the Xilinx support web site or ask the instructor. For comparison purposes, here is a sample log file.

```
Release 6.3.03i - xst G.38
```

TABLE OF CONTENTS


```
=========================================================================
*                      Synthesis Options Summary
=========================================================================

---- Source Parameters
Input File Name                    : two_input_xor.prj
Input Format                       : mixed
Ignore Synthesis Constraint File   : no
Verilog Include Directory          :

---- Target Parameters
Output File Name                   : two_input_xor
Output Format                      : ngc
Target Device                      : xc3s200-4-ft256

---- Source Options
Top Module Name                    : two_input_xor
Automatic FSM Extraction           : yes
FSM Encoding Algorithm             : auto
FSM Style                          : lut
RAM Extraction                     : yes
RAM Style                          : auto
ROM Extraction                     : yes
ROM Style                          : auto
Mux Extraction                     : yes
Mux Style                          : auto
Decoder Extraction                 : yes
Priority Encoder Extraction        : yes
Shift Register Extraction          : yes
Logical Shifter Extraction         : yes
XOR Collapsing                     : yes
Resource Sharing                   : yes
Multiplier Style                   : auto
Automatic Register Balancing       : no

---- Target Options
Add IO Buffers                     : yes
Global Maximum Fanout              : 500
Add Generic Clock Buffer (BUFG)    : 8
Register Duplication               : yes
Equivalent register Removal        : yes
Slice Packing                      : yes
Pack IO Registers into IOBs        : auto
```

```
---- General Options
Optimization Goal                  : speed
Optimization Effort                : 1
Keep Hierarchy                     : no
Global Optimization                : allclocknets
RTL Output                         : yes
Write Timing Constraints           : no
Hierarchy Separator                : _
Bus Delimiter                      : <>
Case Specifier                     : maintain
Slice Utilization Ratio            : 100
Slice Utilization Ratio Delta      : 5

---- Other Options
Lso                                : two_input_xor.lso
Read Cores                         : yes
Cross_clock_analysis               : no
Verilog2001                        : yes
Optimize Instantiated Primitives   : no

=========================================================================
*                           HDL Compilation
=========================================================================

Compiling source file "two_input_xor.v"
Module <two_input_xor> compiled
No errors in compilation
Analysis of file <two_input_xor.prj> succeeded.

=========================================================================
*                            HDL Analysis
=========================================================================

Analyzing top module <two_input_xor>.
Module <two_input_xor> is correct for synthesis.

=========================================================================
*                            HDL Synthesis
=========================================================================

Synthesizing Unit <two_input_xor>.
    Related source file is two_input_xor.v.
    Found 1-bit xor2 for signal <out>.
Unit <two_input_xor> synthesized.

=========================================================================
*                       Advanced HDL Synthesis
=========================================================================

Advanced RAM inference ...
Advanced Multiplier inference ...
Advanced Registered AddSub inference ...
Dynamic shift register inference ...

HDL Synthesis Report
Macro Statistics
```

```
# Xors                              : 1
 1-bit xor2                        : 1

=========================================================================
*                        Low Level Synthesis
=========================================================================

Optimizing unit <two_input_xor> ...
Loading device for application Xst from file '3s200.nph'
in environment C:/Xilinx.

Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block two_input_xor,
actual ratio is 0.

=========================================================================
*                            Final Report
=========================================================================

Final Results
RTL Top Level Output File Name     : two_input_xor.ngr
Top Level Output File Name         : two_input_xor
Output Format                      : ngc
Optimization Goal                  : speed
Keep Hierarchy                     : no

Design Statistics
# IOs                              : 3

Cell Usage :
# BELS                             : 1
#      LUT2                        : 1
# IO Buffers                       : 3
#      IBUF                        : 2
#      OBUF                        : 1

=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s200ft256-4
 Number of Slices:                        1  out of   1920     0%
 Number of 4 input LUTs:                  1  out of   3840     0%
 Number of bonded IOBs:                   3  out of    173     1%

=========================================================================

TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE
      REPORT GENERATED AFTER PLACE-AND-ROUTE.

Clock Information:
------------------
```

```
No clock signals found in this design

Timing Summary:
---------------
Speed Grade: -4

   Minimum period: No path found
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: No path found
   Maximum combinational path delay: 7.836ns

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

--------------------------------------------------------------------------
Timing constraint: Default path analysis
Delay:               7.836ns (Levels of Logic = 3)
  Source:            in2 (PAD)
  Destination:       out (PAD)

  Data Path: in2 to out
                              Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O             1   1.930   0.240  in2_IBUF (in2_IBUF)
     LUT2:I0->O            1   0.551   0.240  Mxor_Result1 (out_OBUF)
     OBUF:I->O                 4.875          out_OBUF (out)
    ---------------------------------------
    Total                      7.836ns (7.356ns logic, 0.480ns route)
                                       (93.9% logic, 6.1% route)

==========================================================================
CPU: 22.00 / 26.81 s | Elapsed: 22.00 / 27.00 s
Total memory usage is 63464 kilobytes
```

Reading the report is a good way to find out what types of (and how many) resources the synthesis tool used. You can also catch other problems this way. For example, if you found that this design description resulted in flip flops, in addition to a look-up table and I/O buffers, you had better go back and figure out what went wrong. This is why you must have an understanding of the hardware you are attempting to create when you write your design description. At this point, you should have a green checkmark next to the Synthesize—XST process.

Design Implementation

Design implementation is the sequence of events that translates your synthesized design netlist into a programming file for the FPGA device. Your design description, which you have now synthesized, has a number of ports at the top level. The implementation tools need to know how to assign the ports in your top level to physical pins on the FPGA, which are connected to various resources on the Spartan-3 Starter Kit board. If you do not make explicit assignments, the tools will randomly assign pins for you. However, this is generally a bad idea because random assignments will be wrong.

The top-level design has two input ports, and a single output port. We will want to have two switches, SW0 and SW1, connected to the inputs. Additionally, we will want the output connected to an LED so that we can observe it – indicator LD0 is appropriate for this purpose.

If you inspect the top of the Spartan-3 Starter Kit board, you will notice that almost every resource has been thoughtfully annotated with text indicating which FPGA pins are connected to it. This information is also available in the *Spartan-3 Starter Kit User Guide* in tabular and schematic form. Try to identify which FPGA pins are used for SW0, SW1, and LD0, and then check your results with what is shown below. You will need to be able to do this on your own in future lab assignments:

- SW0 → FPGA Pin F12
- SW1 → FPGA Pin G12
- LD0 → FPGA Pin K12

You now have enough information to create what is called a user constraint file, or UCF. This file contains design constraints that you did not specify in the Verilog-HDL description, such as pin location and design performance constraints. It is convenient to provide them in a UCF rather than in the Verilog-HDL description. For instance, if you make a mistake in the pin assignments, you do not need to go back and re-synthesize your design.

You can add a UCF to the project using the same process you used for adding the design and its test bench. Create a new source file; select Project→New Source from the main menu or use the equivalent process in the Processes for Current Source window. The first of the New Source dialog boxes will appear, as shown in Figure 22.

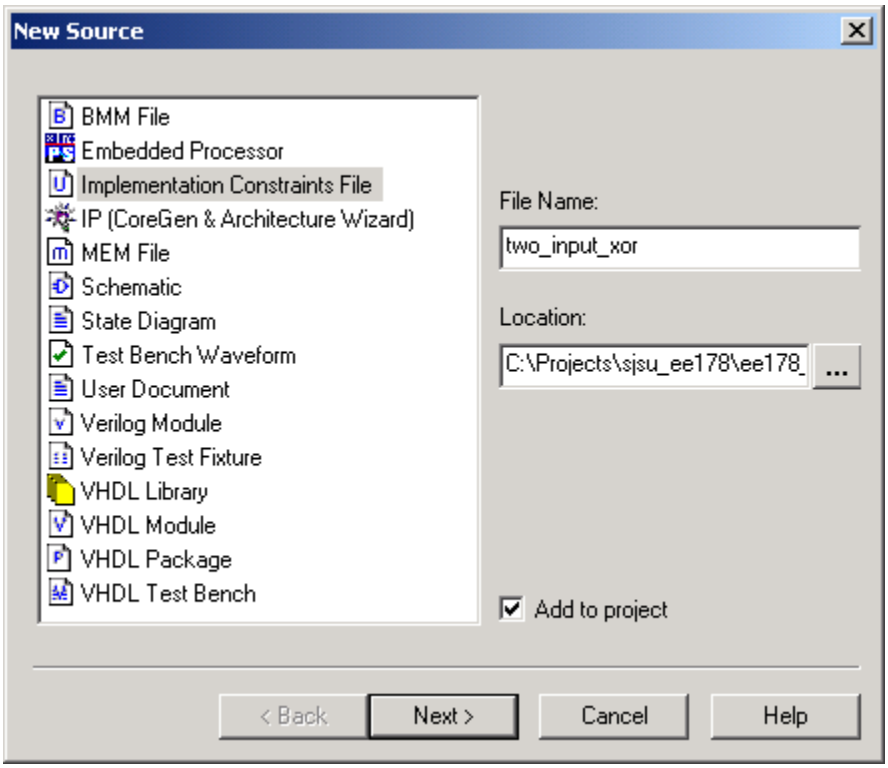

Figure 22: New Source Dialog 1 of 3

Select Implementation Constraints File to indicate you are creating a constraints file. Then, provide a file name as shown in Figure 22. You should not need to change the specified location, which should be inside the project directory you created earlier. Click "Next".

The second dialog, shown in Figure 23, asks you to identify a design module with which the constraints file should be associated. Select the two-input XOR design as shown and click "Next".

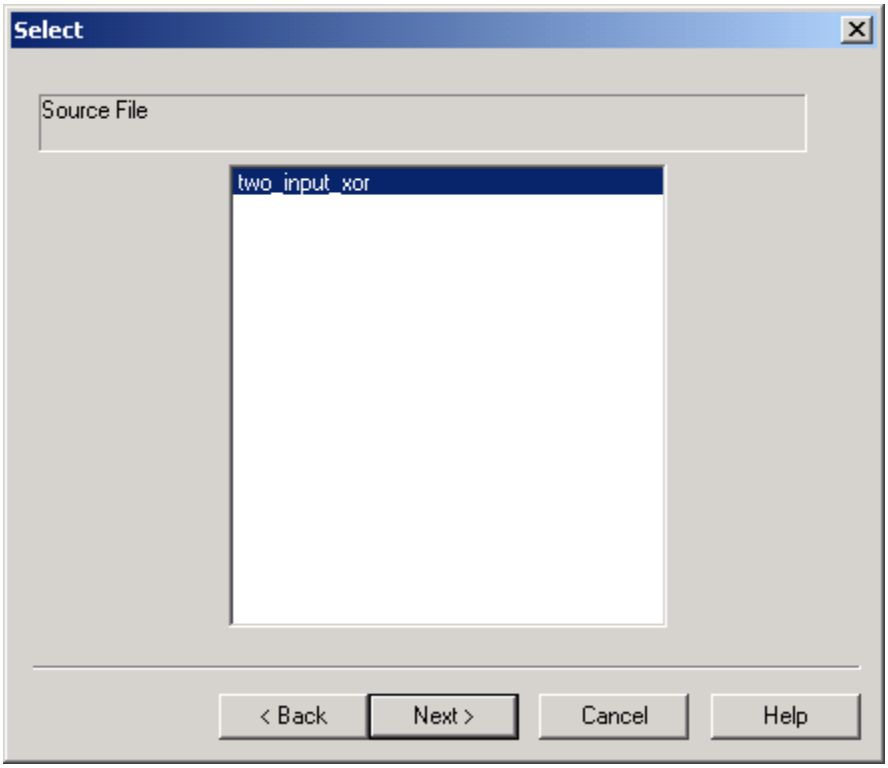

Figure 23: New Source Dialog 2 of 3

The final dialog box of Figure 24 provides a summary of the source that Project Navigator will create based on your settings. Review the summary to make sure it matches what is shown in Figure 24. If it does not, go "Back" and correct any errors. Otherwise, click "Finish" to complete this process. This time, however, you will notice that the new source file is not automatically opened in the text editor.

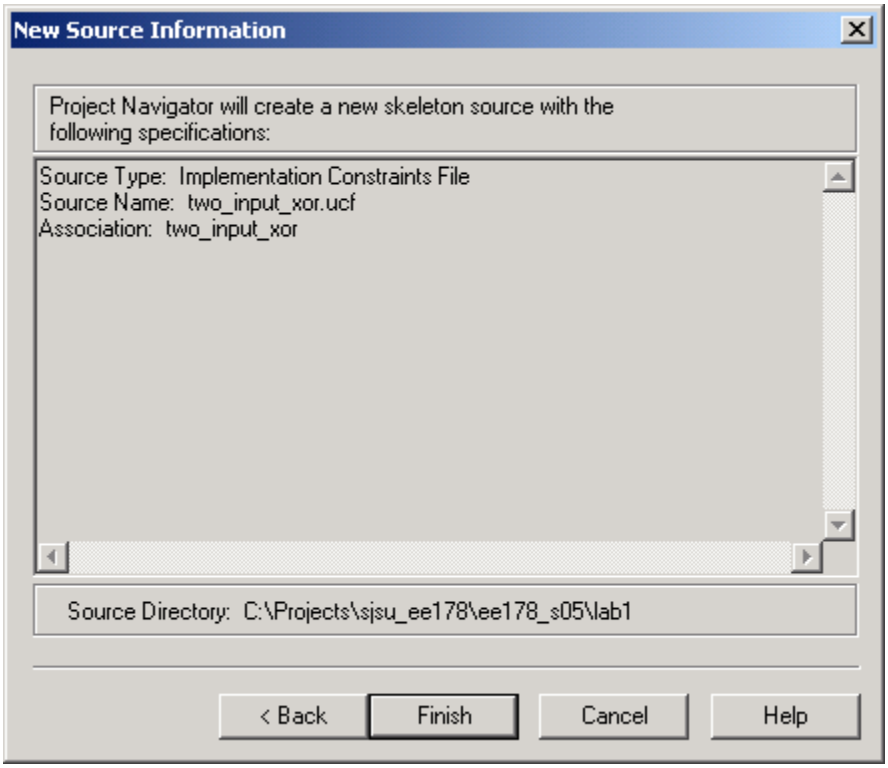

Figure 24: New Source Dialog 3 of 3

If you select the constraint file in the Sources in Project Window, and then expand the Processes for Current Source window item for User Constraints by clicking on the + next to it, you will see that there are a number of ways to edit a user constraint file, including a text editor. The default user constraint editor is called PACE. Simply double click the constraint file in the Sources in Project window, and PACE will open, see Figure 25. PACE is a fairly powerful constraint editor but we will only be using a small portion of its capabilities in this tutorial.

The PACE sub-windows shown in Figure 25 have been moved from their default positions in order to yield an improved screen capture. First click on I/O Pins in the Design Browser window. The Design Object List window will then show the names of the three top level ports and their signal directions. In this window, fill in the LOC fields based on the previously determined FPGA pin assignments. For the three IO STD fields select LVCMOS33, which is a common 3.3 volt signaling standard.

After entering each pin assignment, you will notice that the corresponding package pin shown in the Package Pins window will be grayed out, indicating it is in use. This diagram represents the physical pins on the package that holds the FPGA die. You will also notice the highlighting of regions shown in the Device Architecture window. This diagram represents resources in use on the FPGA die related to your constraints – in this case, the input and output buffers and I/O pads. When you are done, save your work and exit the PACE program.

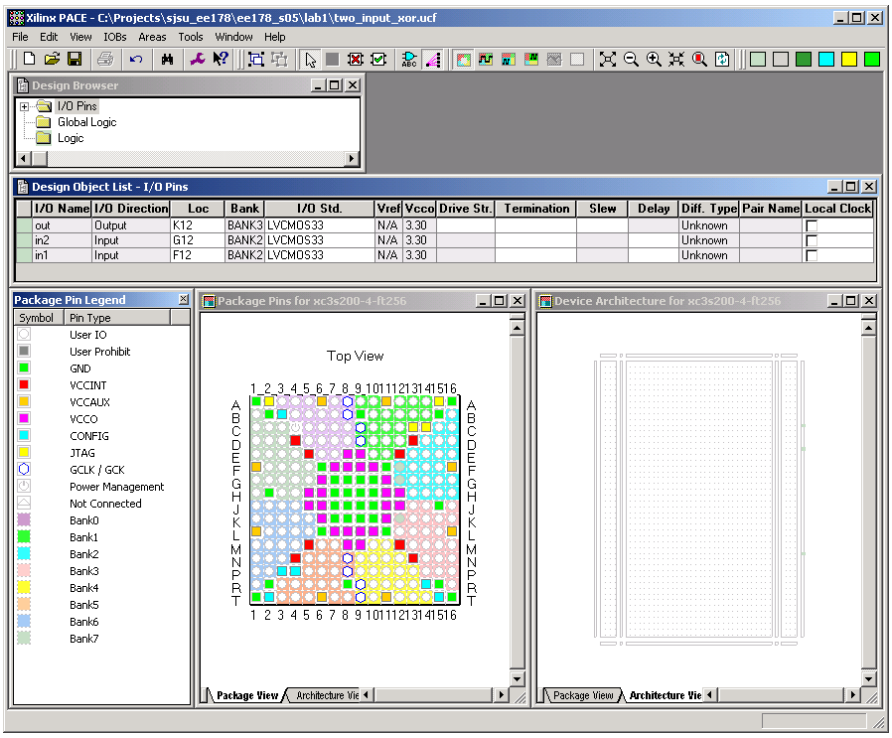

Figure 25: Entering Pin Location Constraints using PACE

Now that you have a constraint file in your project, you can implement the design. Select two_input_xor in the Sources in Project window. Then, double click on the Implement Design process in the Processes for Current Source window. Project Navigator will implement the design and print information to the Console window in the process. As an informational note, it is possible to change the implementation options before you implement by right clicking on Implement Design and then selecting Properties. For this tutorial, however, leave the options at their default settings.

You should not see any errors in the Console window. However, you should always review the three log files, which are available for viewing if you expand the Implement Design process item by clicking on the + next to it. There are log files located under Translate, Map, and Place and Route. If you don't understand a particular message, you should not simply ignore it. Instead, search the Xilinx support web site or ask the instructor. At this point, you should have a green checkmark next to the Implement Design process.

Timing Simulation

After completing the implementation steps, you can simulate your design again – this time, using a structural representation of your synthesized, placed, and routed design with worst-case delay information. The idea is to simulate your design, as physically implemented in the FPGA device.

The simulation processes enable you to run simulation on the design using Modelsim. To locate the Modelsim simulator processes, select the test bench in the Sources in Project window. Then, click the + next to the Modelsim Simulator entry in the Processes for Source window to expand the item. You will perform a timing simulation using Simulate Post-Place & Route Verilog Model but you must specify the simulation process properties first, just like you did for functional simulation. Right click on Simulate Post-Place & Route Verilog Model, and select Simulation Properties. The Process Properties dialog box appears, as shown in Figure 26.

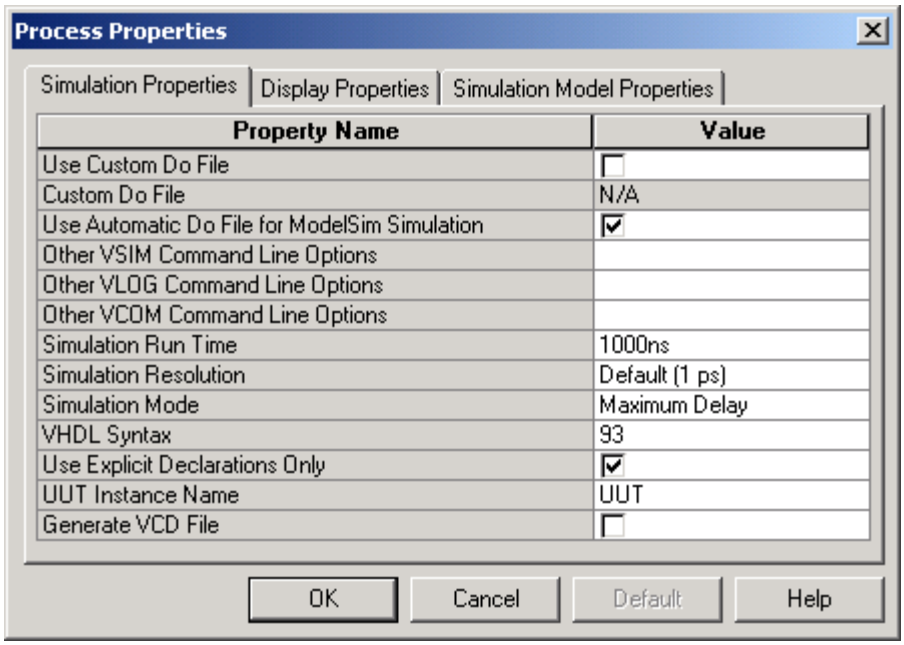

Figure 26: Simulation Process Properties

Make sure the properties are set as shown in Figure 26. The most interesting of these parameters is probably the simulation run time – again, 1000 ns is more than sufficient for the test bench in the project. For test benches that require more simulation time, this property should be adjusted as needed. Click "Ok".

To start the simulation, double-click Simulate Post-Place & Route Verilog Model. Modelsim creates a work directory, compiles the source files, loads the design, and performs simulation for the time specified. The simulator will run, and you'll see results as before. Unless you are lucky, your simulation will fail…

The testbench should report that your design has failed because it is checking the output of your design five nanoseconds after it has changed the inputs. If you look at the wave window, it appears that the FPGA implementation requires more than five nanoseconds for signals to propagate through the design – more like ten to fifteen nanoseconds. The exact number depends on the device you are using, the placement, the routing, and your design. Close Modelsim, go back to your testbench, and change the delays to fifty nanoseconds (to be safe…) Then perform the simulation again. This time, it should pass.

At this point, you are ready to program the FPGA with your design. The Spartan-3 Starter Kit board may be programmed by two different methods. One is to program the FPGA by download cable. The other is to program the PROM by download cable, and then have the PROM program the FPGA. Both are covered in the following sections.

Programming the FPGA by Download Cable

Programming the FPGA directly by the download cable is a convenient way to try out a design. This method is useful when you want to quickly test something, or are not certain your design is final. For example, at this point you are fairly confident your design is correct. However, you should realize by this point in your education that complex designs rarely ever work "on the first try". One of the great advantages FPGAs have over ASICs is that the penalty for being wrong on the first try is minimal.

The first order of business is to create a programming file for the FPGA. Select two_input_xor in the Sources in Project window. In the Processes for Current Source window, right click on Generate Programming File and then select Properties. The Process Properties dialog box appears. Select the Configuration Options tab, as shown in Figure 27.

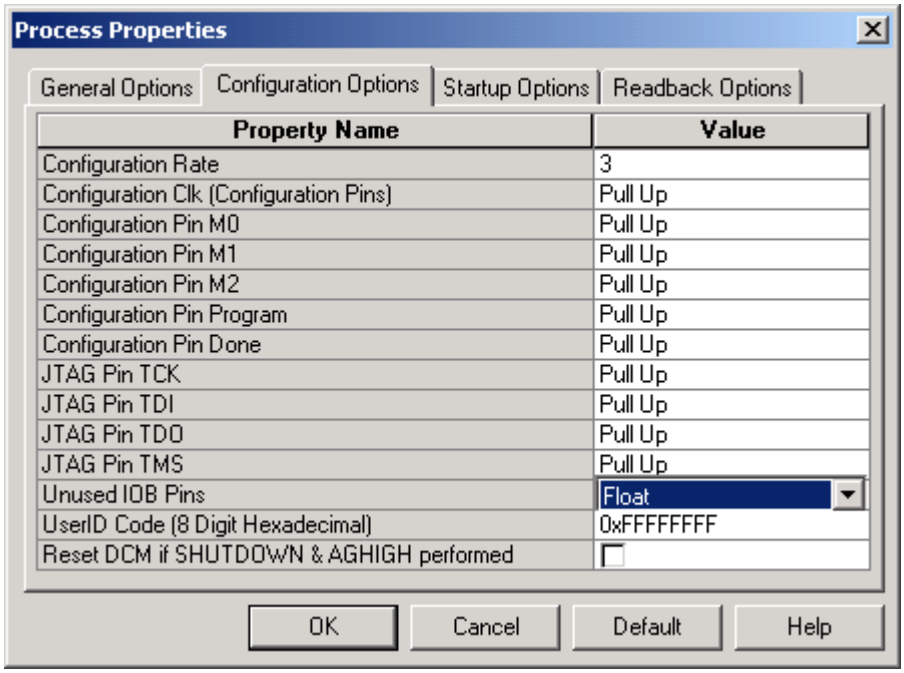

Figure 27: Generate Programming File Process Properties

Change the Unused IOB Pins option to Float. The other settings should already be correct, but make sure they match what is shown in Figure 27. Next, select the Startup Options tab, as shown in Figure 28.

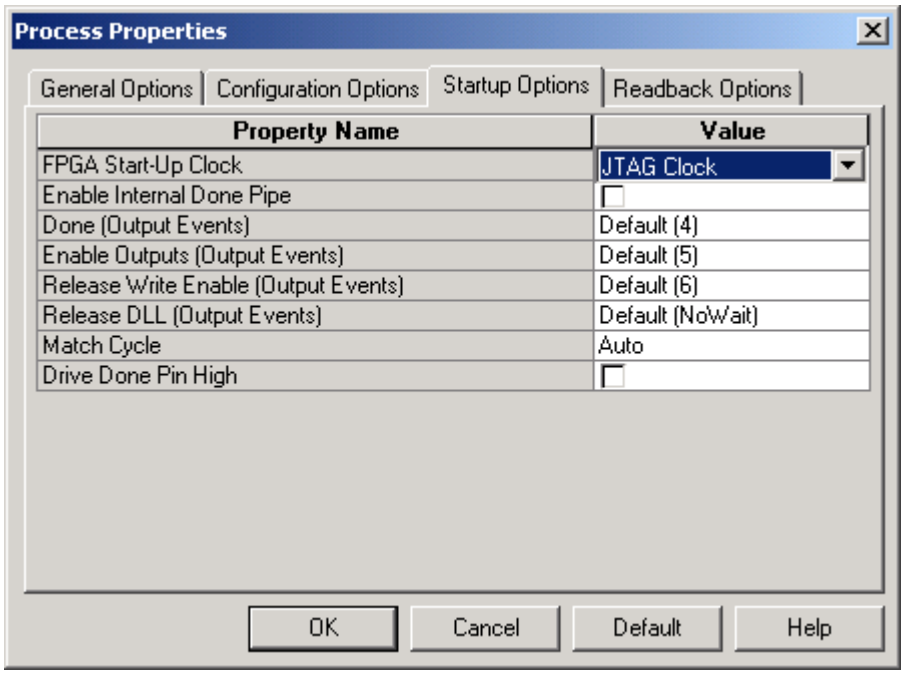

Figure 28: Generate Programming File Process Properties

Change the FPGA Start-Up Clock option to JTAG Clock. The other settings should already be correct, but make sure they match what is shown in Figure 28. Click “Ok” to save the settings.

Confirm that two_input_xor is selected in the Sources in Project window. Then, double click on the Generate Programming File process in the Processes for Current Source window. Project Navigator will generate a programming file and print information to the Console window in the process.

Before you continue, you must have the Spartan-3 Starter Kit board, power supply, and download cable available. Connect the download cable to the parallel port of the machine you are using. Plug the power supply into the wall. Look at the Spartan-3 Starter Kit board and identify the following. If you need help, ask the instructor or refer to the *Spartan-3 Starter Kit User Guide*:

- PROM Jumper, JP1, at Top Right
- Mode Jumper, J8, near Top Center
- DC Power Jack, J4, at Left Center
- Download Cable Connector, J7, at Top Center

Make sure that the PROM Jumper is set to Default and that the Mode Jumper has all three jumpers installed. You should have received the board in this state, but it is better to confirm. Then, insert the power plug into the DC Power Jack. Be aware that if a programming file was previously stored in the PROM, it will automatically load, and may result in board activity, like flashing LEDs, etc… This can be safely ignored. Finally, connect the download cable to its connector, as shown in Figure 29.

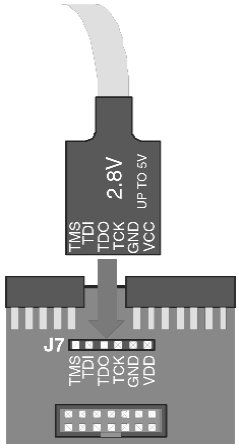

Figure 29: Download Cable Connection

To download your bitstream to the FPGA device, expand the Generate Programming File process by clicking on the + next to it, and then double click on the Configure Device (iMPACT) process. This will launch the iMPACT program in another window. You will be immediately presented with several dialog boxes, the first of which is shown in Figure 30.

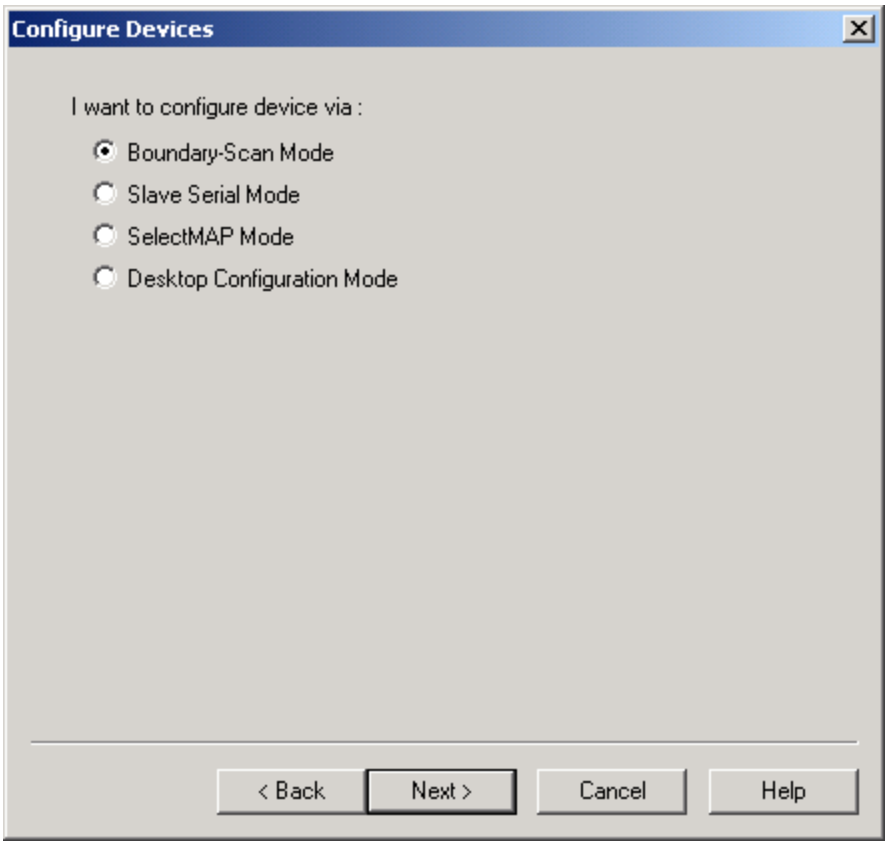

Figure 30: Configuration Mode Selection

There are actually a bewildering number of ways to configure an FPGA device. The board has an integrated JTAG programming function, which is also called Boundary-Scan mode. Select this option and proceed to the next dialog box, shown in Figure 31.

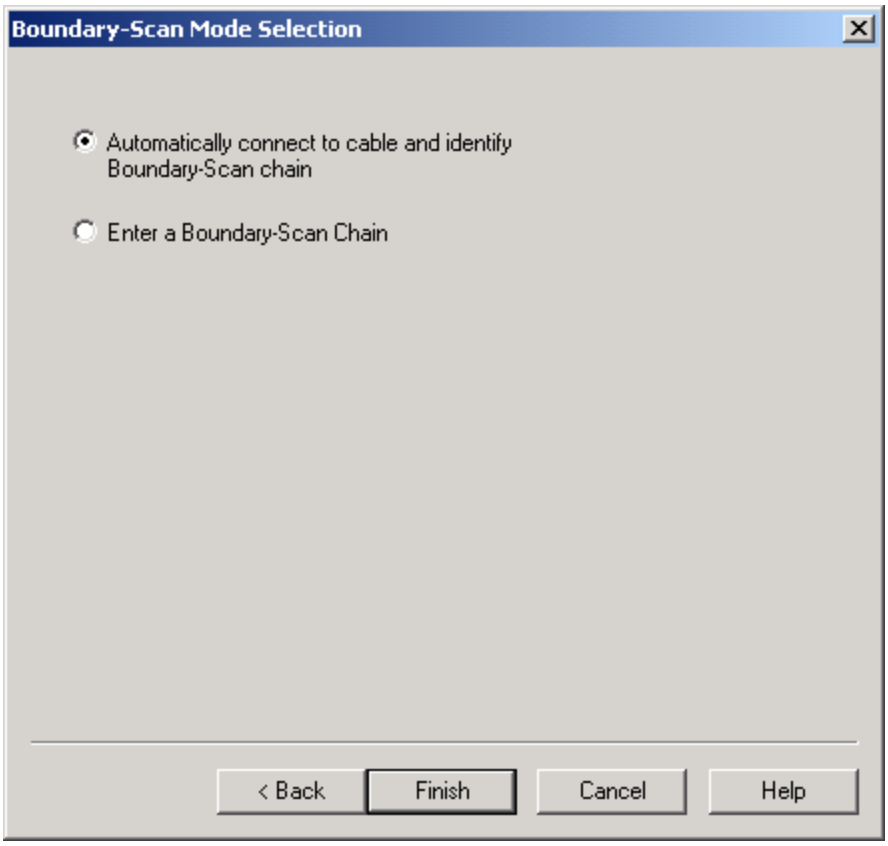

Figure 31: Boundary-Scan Mode Selection

Allow the program to automatically connect to the cable and identify the devices on the board. After you finish this sequence, the program will automatically detect the FPGA and PROM devices and prompt you to specify a programming file for each device. You should see a message like that shown in Figure 32. Click "Ok".

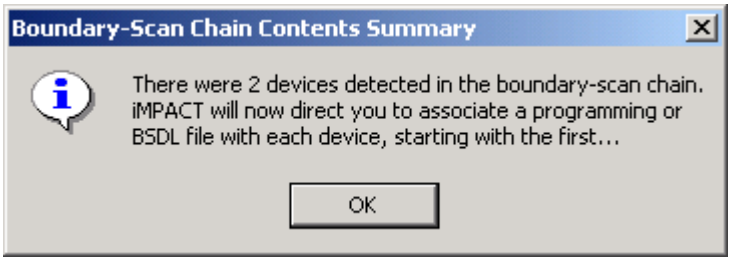

Figure 32: Notification

Next, you should get a sequence of two file requestors. I have observed, on occasion, that this does not occur; I think it may be a bug in iMPACT. If you do not get file requestors at this point, skip forward to Figure 35. Otherwise, keep reading…

In the first file requestor, shown in Figure 33, select the two_input_xor.bit file you created with the implementation process. This is the FPGA programming file.

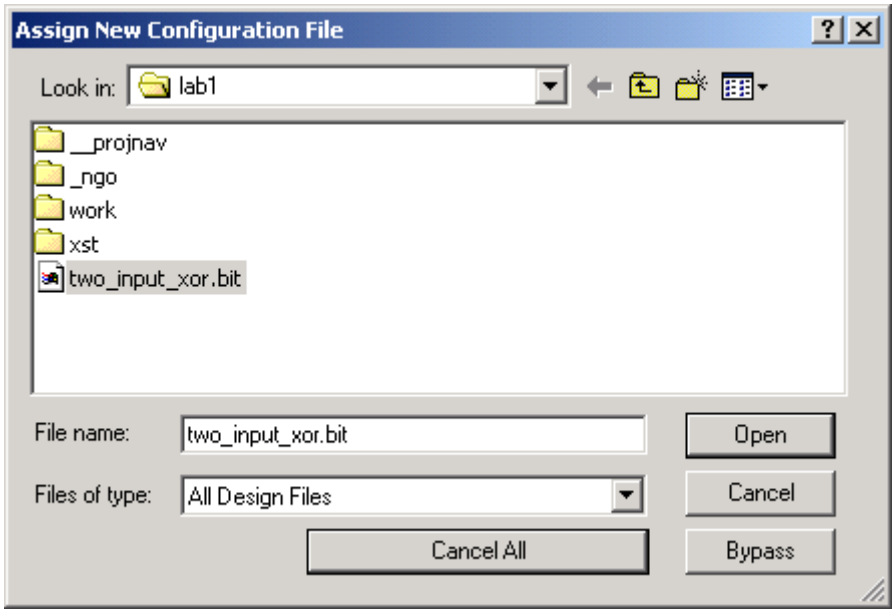

Figure 33: Selecting the FPGA Programming File

The next file requestor, shown in Figure 34, asks for a PROM programming file. We are not programming the PROM at this time, therefore select Bypass.

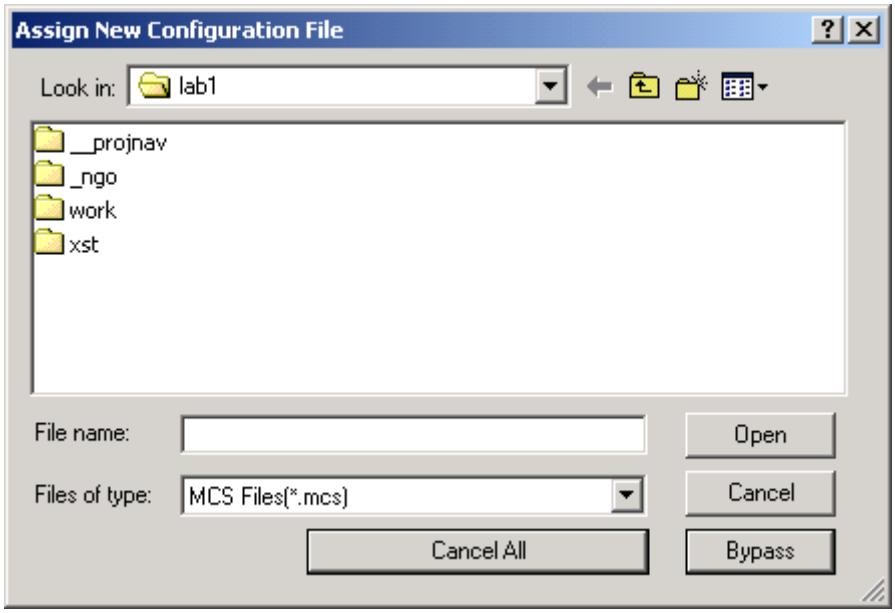

Figure 34: Placing the PROM in Bypass Mode

At this point, you should be ready to program the FPGA. If you have made a mistake, you can correct your assignments by using the technique illustrated in Figure 35.

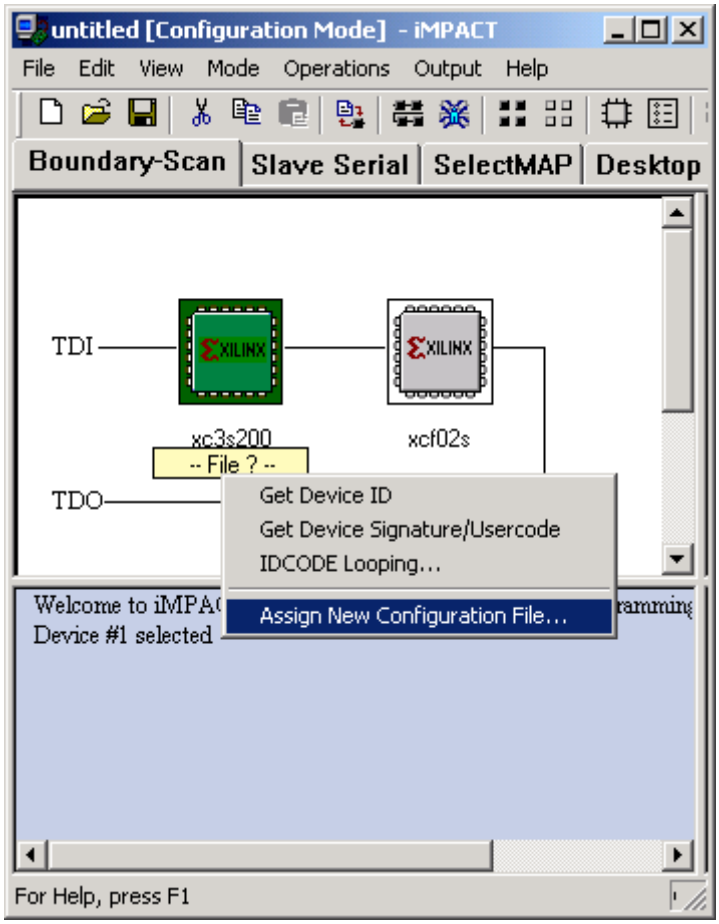

Figure 35: Alternate File Assignment Method

To correct a file assignment, or to make initial assignments if iMPACT does not automatically prompt you for programming files, select the FPGA icon in the iMPACT window. Right click and select Assign New Configuration File. You will get a file requestor like that shown in Figure 33. You can repeat this process with the PROM icon, and you will get a file requestor like that shown in Figure 34.

Finally, you will reach the point shown in Figure 36. iMPACT is ready to program the FPGA. Select the FPGA icon in the window and then use the right mouse button to activate the menu as shown and select the Program option.

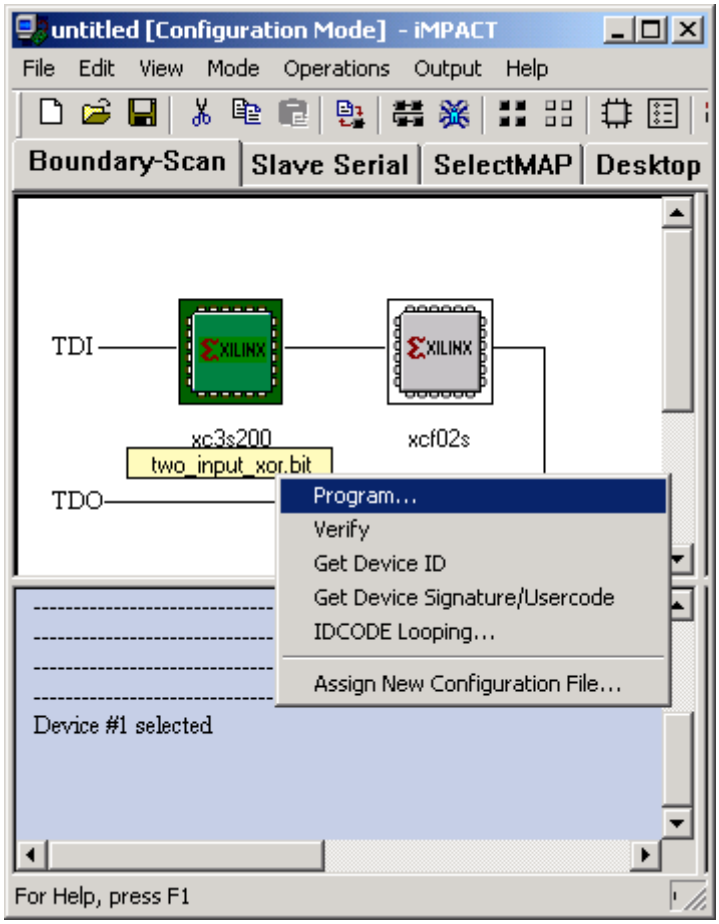

Figure 36: Select Program Device

You will be presented with a dialog box listing programming options. Most of these options are ghosted out for FPGA programming and are of no concern, see Figure 37. Disable the Verify option, if selected, and then click "Ok" to start the programming sequence.

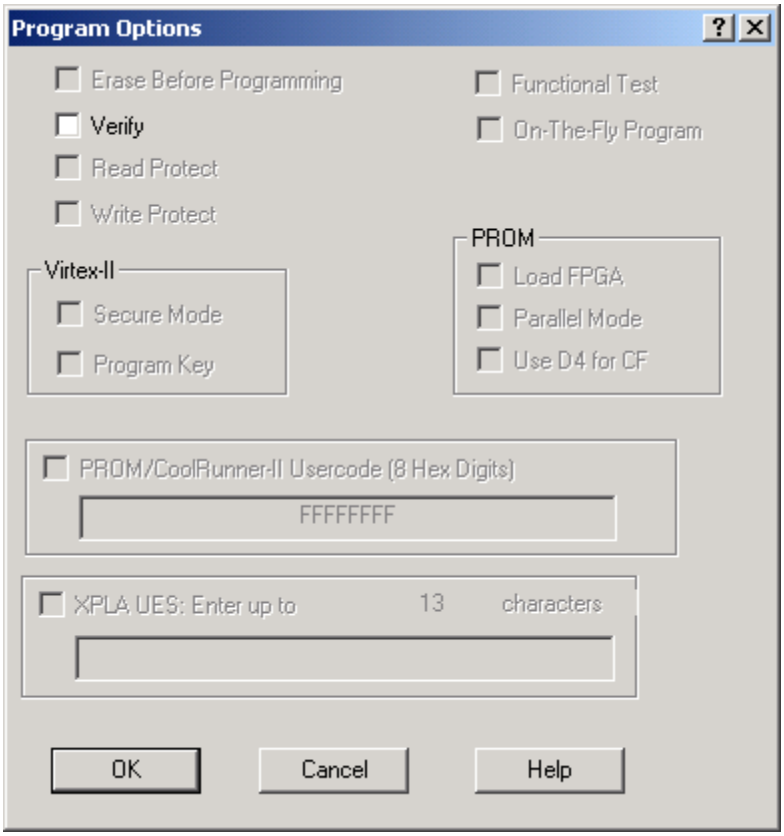

Figure 37: Programming Options

A progress indicator will appear. Once the programming is complete, the program will be sure to let you know if it was successful or if it failed. If the programming has failed, re-check your cable connections, the power connections, and the jumpers – and then try again. If it still fails, ask the instructor for assistance.

Now, you can test your design in hardware. Locate SW0 and SW1 on the board, and exercise your design by trying the four possible combinations of switch settings while observing LD0. Does the circuit behave as you expect? If it does not, seek assistance. If it does work properly, you are ready to try the other programming method. Exit iMPACT (you do not need to save). Keep the board connected to power and the download cable.

Programming the PROM by Download Cable

The other method is to program the PROM by download cable, and then have the PROM program the FPGA. Typically you would program the PROM when you believe your design is completely done. After the PROM is programmed, each time the power is cycled, the FPGA will automatically load the programming file from the PROM. After the PROM is programmed, the need for the download cable is eliminated.

Expand the Generate Programming File process by clicking on the + next to it, and then double click on the Generate PROM, ACE, or JTAG File process. This will launch the iMPACT program again.

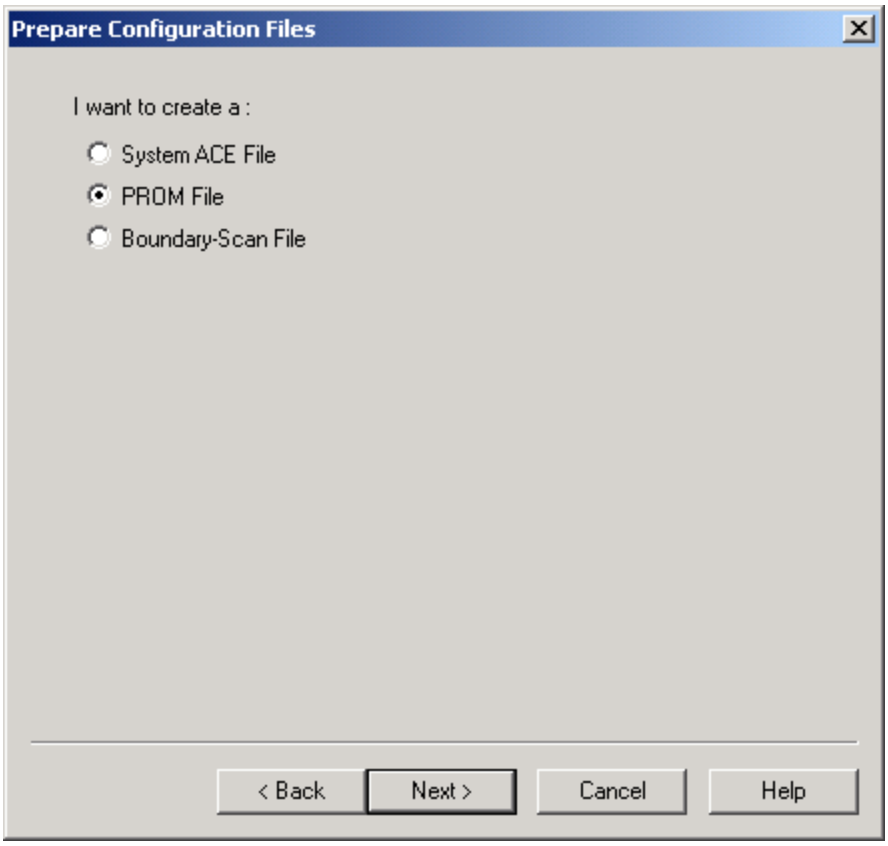

Figure 38: File Type Selection

You will be immediately presented with several dialog boxes, the first of which is shown in Figure 38. Select the PROM File option and proceed to the next dialog box, shown in Figure 39.

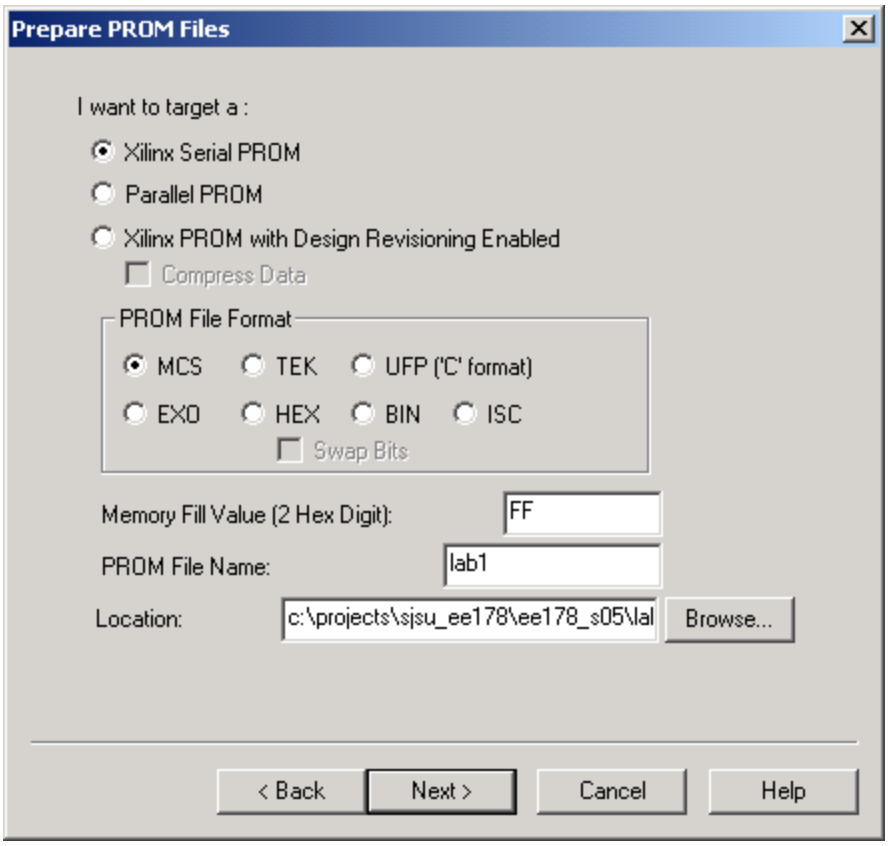

Figure 39: PROM Property Selection

In the dialog box of Figure 39, change the settings to match those shown. Do not forget to change the PROM File Name. Then proceed to the next dialog box.

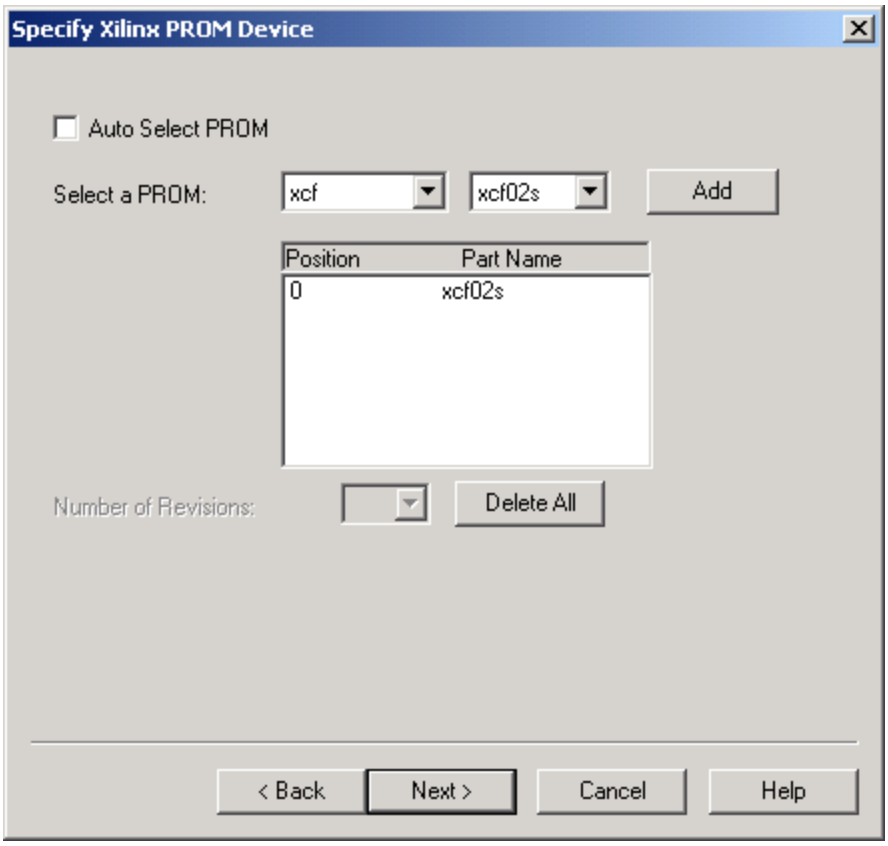

Figure 40: PROM Selection

In the dialog box of Figure 40, select the XCF—XCF02S PROM type, and then click “Add”. You should see the PROM listed in the sub-window, at position zero. Then click “Next”.

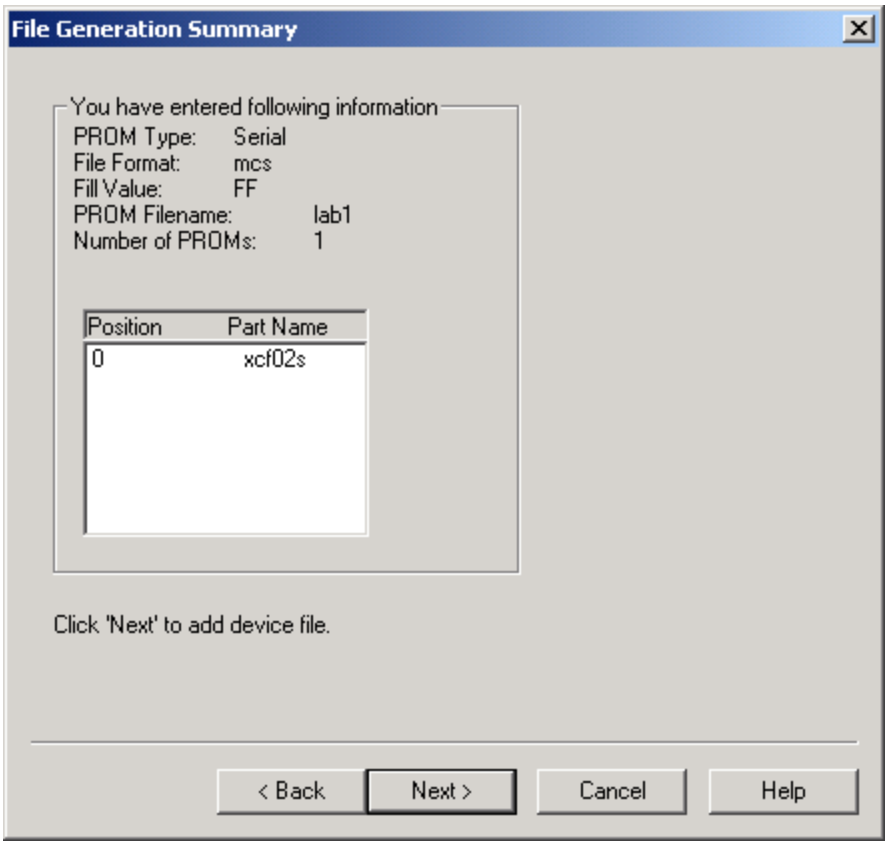

Figure 41: Summary Window

Figure 41 shows a summary of what you have selected. If your results do not match that shown in Figure 41, go “Back” and correct your error. Otherwise, click “Next” to proceed.

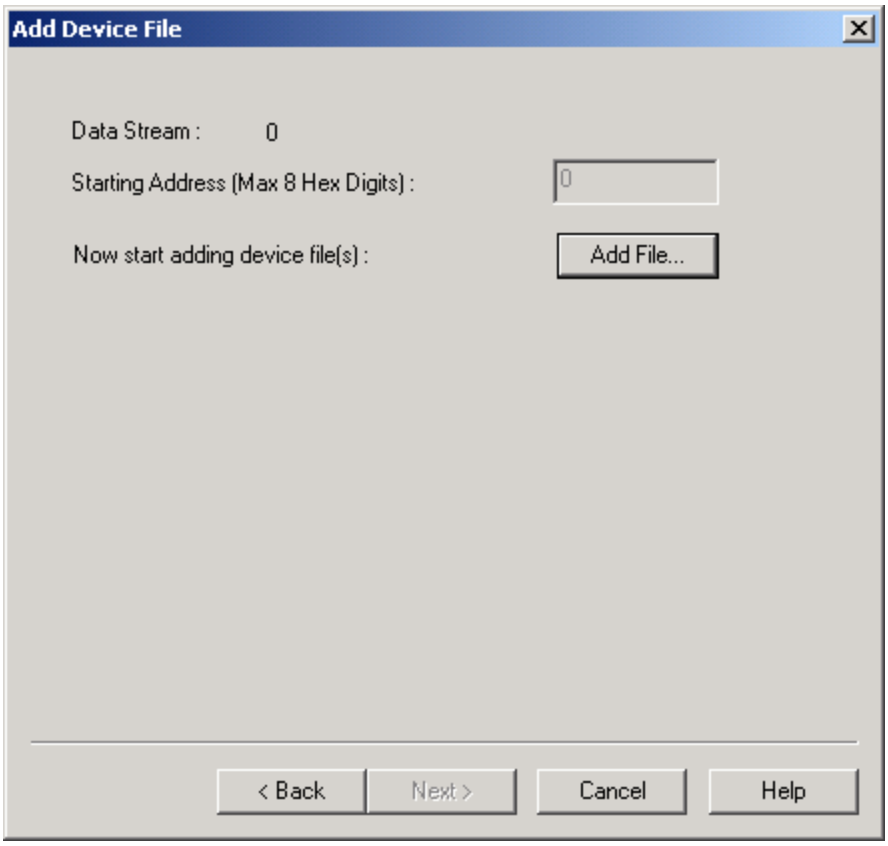

Figure 42: Add FPGA Programming Files

In the dialog box of Figure 42, click "Add File…" When the file requestor dialog box appears, select the two_input_xor.bit file, which is the same one you used before. You will receive a warning that iMPACT needed to change the startup clock; dismiss the warning. You may recall, from a previous step, that we set the Startup Clock option to JTAG Clock when creating the programming file. This setting is required when programming the FPGA directly by the cable, but for programming the PROM the CCLK setting should be used. iMPACT makes this change for you without requiring that you revisit the Generate Programming File process.

After you add the two_input_xor.bit file, iMPACT will ask you if you want to add another design file to the PROM data stream. Click "No". You will see another dialog box that looks almost identical to Figure 42, which instructs you to click "Finish" to start generating the PROM file. Click "Finish". iMPACT will ask you if you want to create the file now. Click "Yes".

You have now created the PROM programming file. You need to program the PROM. From the iMPACT main menu, select Mode→Configuration Mode. Then, select File→Initialize Chain. At this point, you will be prompted for programming files for the two devices in the chain, just like you were in the previous section. However, this time around, put the FPGA in Bypass mode and assign the lab1.mcs file to the PROM. Then, select the PROM icon, right click, and select Program.

You will be presented with a dialog box listing programming options. Most of these options are ghosted out for PROM programming and are of no concern, see Figure 43. Verify the options are set as shown in Figure 43 and click "Ok" to start the programming sequence.

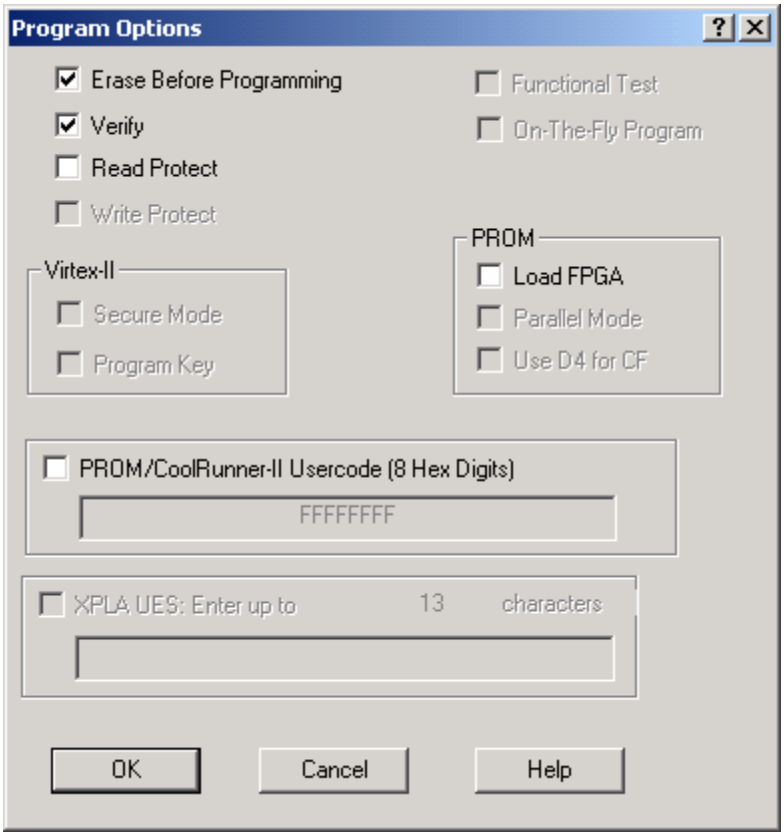

Figure 43: Programming Options

A progress indicator will appear. Once the programming is complete, the program will be sure to let you know if it was successful or if it failed. If the programming has failed, re-check your cable connections, the power connections, and the jumpers – and then try again. If it still fails, ask the instructor for assistance.

Now, you can test your design again. Exit iMPACT (you do not need to save). Unplug the download cable from the board. Unplug the power supply, wait three seconds, and then reapply power. The FPGA should load your design automatically from the PROM. To verify it worked properly, locate SW0 and SW1 on the board, and exercise your design by trying the four possible combinations of switch settings while observing LD0. Does the circuit behave as you expect? If it does not, seek assistance. If it does work properly, you are done with the lab. In order to receive credit, demonstrate your final result to the instructor.