San Jose State University
Department of Electrical Engineering
EE178, Fall 2017, Crabill

Laboratory Assignment #1

Objectives

This lab is an introduction to logic design using Verilog-HDL with Xilinx Vivado 2016.2. No new logic design concepts are presented in this lab. The goal of this lab is for you to become familiar with the tools you will be using for the rest of the semester.

Consider this lab a "no-brainer" warm up. In previous semesters, I have heard students lament, "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.

Bibliography

This lab draws heavily from documents on the Xilinx website http://www.xilinx.com. I would like to thank Xilinx for making this material available. This lab is effectively a customized introduction to Xilinx Vivado using Verilog-HDL and the Digilent Basys3.

Download , Installation, and Licensing

The Xilinx guide to download, installation, and licensing is linked from the class website. You can read the guide, but the process is easy. This section of the lab assignment captures the essentials of the process. If you do not have an account on the Xilinx website, the first step is to create one:

https://secure.xilinx.com/webreg/createUser.do

Once you have an account on the Xilinx website, and log in, you can download the Vivado HLx 2016.2 Windows Web Installer from:

http://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html

The Xilinx website may ask you to confirm your registration information. If you encounter an "export" issue, please consult the instructor to resolve it. Save the installer to disk.

After the download has completed, run the installer. Once the installer has started, it will guide you through the process. While you are stepping through the installation and licensing steps, Windows may ask to make changes to your computer, and your firewall may ask to grant access – accept these requests.

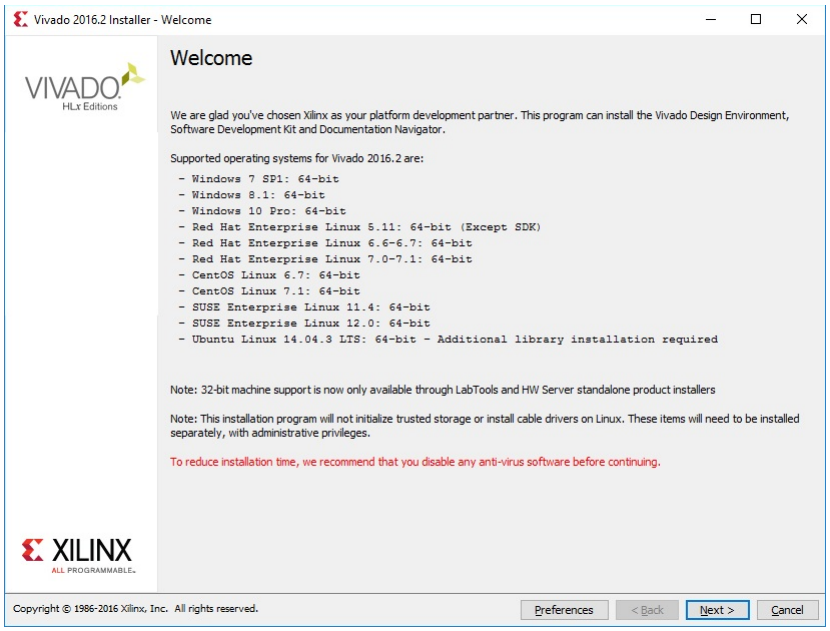

Figure 1: Vivado Installer

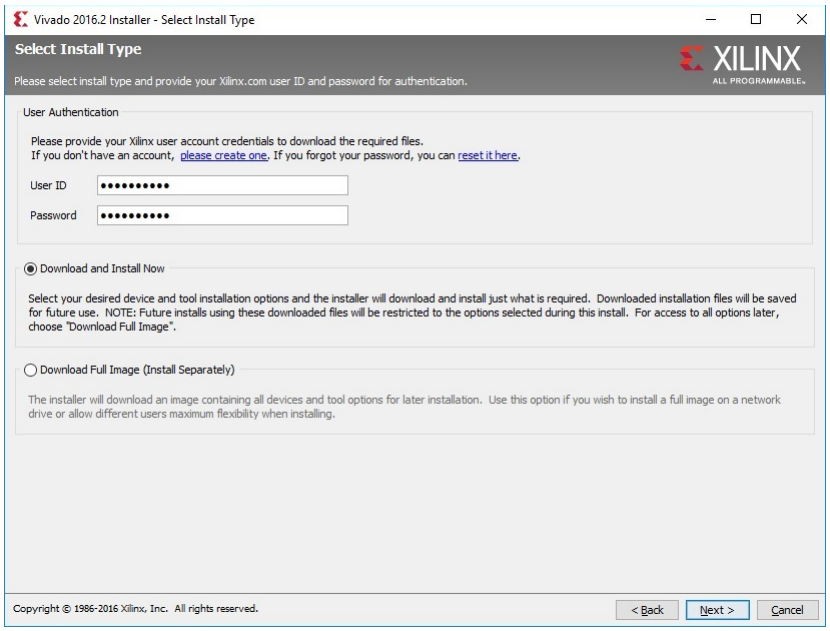

Figure 2: Vivado Installer

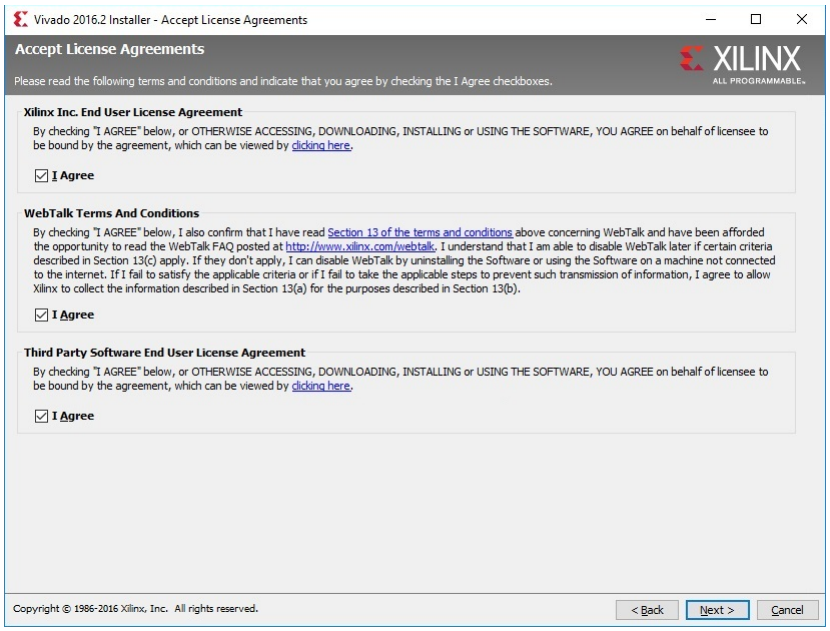

Figure 3: Vivado Installer

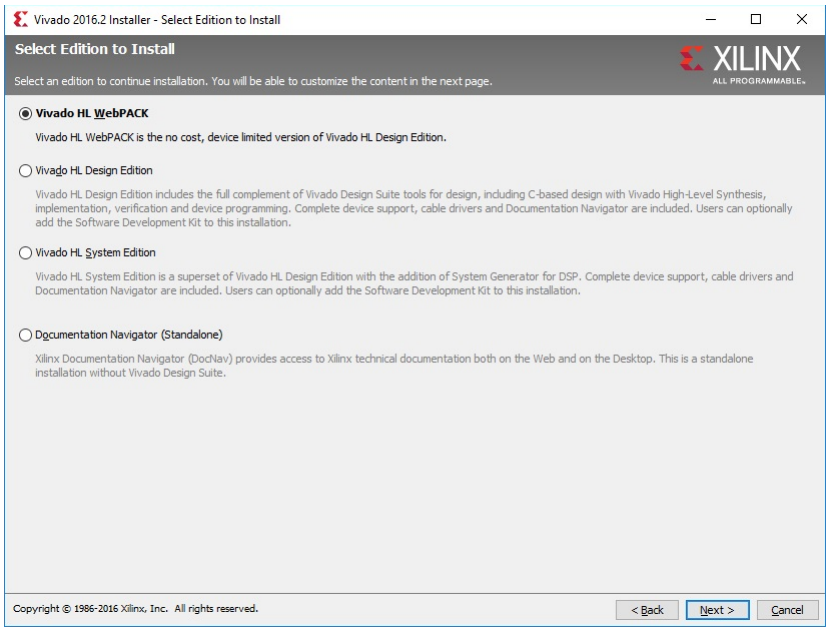

Figure 4: Vivado Installer

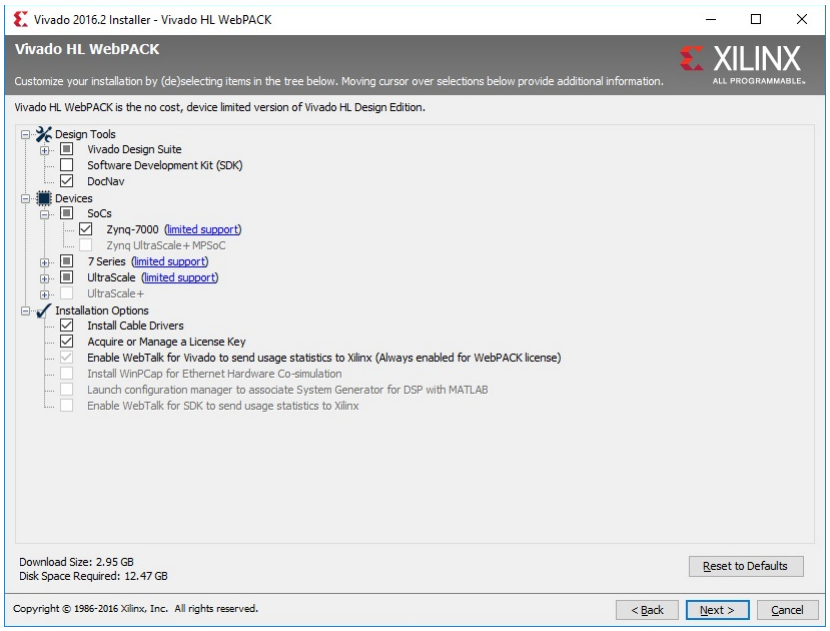

Figure 5: Vivado Installer

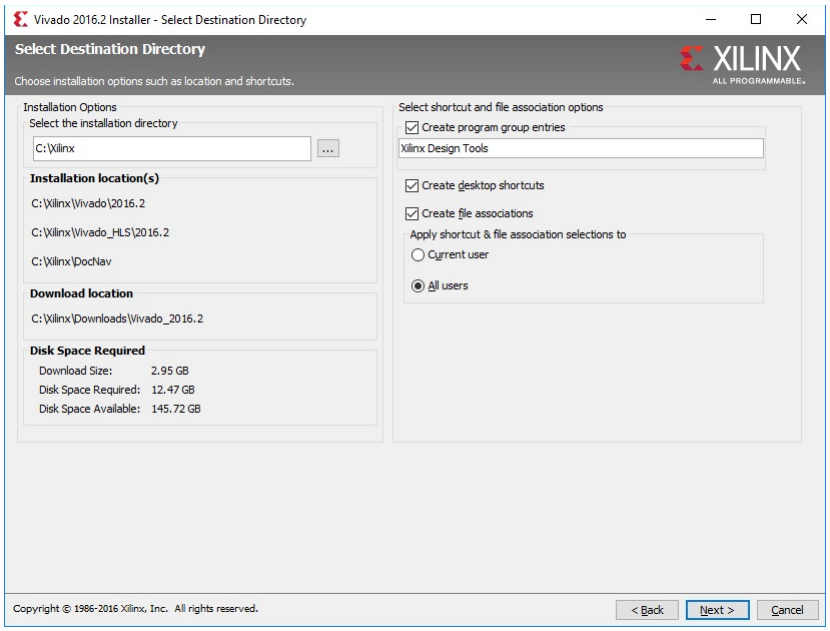

Figure 6: Vivado Installer

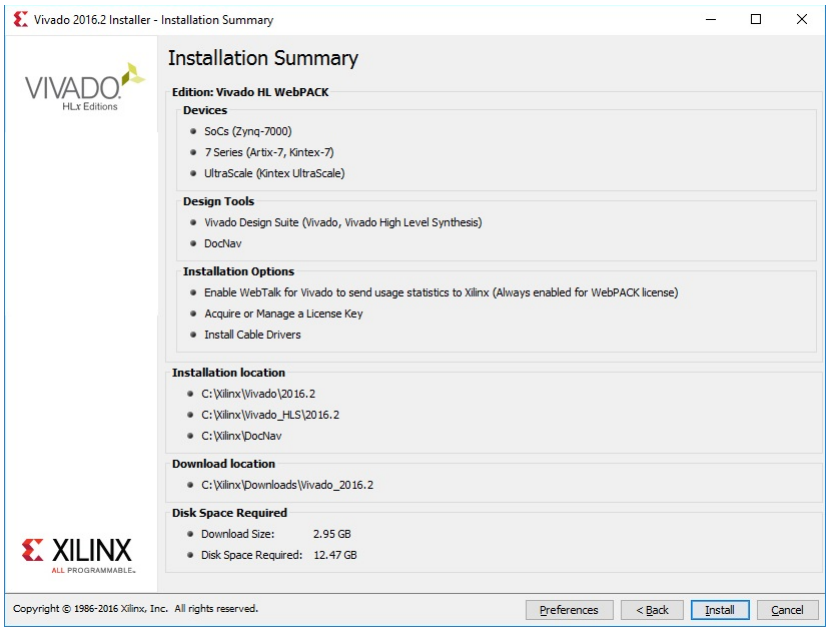

Figure 7: Vivado Installer

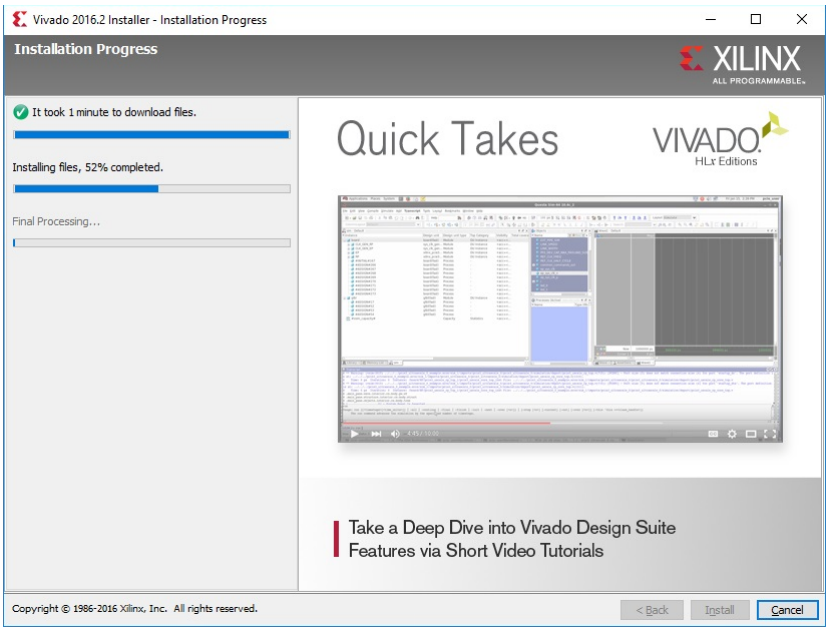

Figure 8: Vivado Installer

At this point, the installer will launch two sub-installation tasks for device drivers. Before device driver installation, make sure you do not have the Digilent Basys3 connected to your computer.

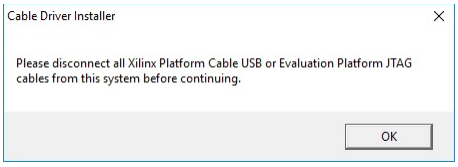

Figure 9: Vivado Installer

Figure 10: Vivado Device Driver Sub-Installation

Figure 11: Vivado Device Driver Sub-Installation

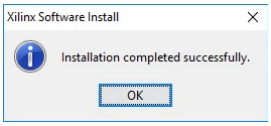

Figure 12: Vivado Installer

At this point, the Vivado installation is complete, and the installer launches the Vivado License Manager. Simply close the Vivado License manager. The free version of Vivado we are using, called Vivado WebPACK, does not require licensing.

Design Entry

The design used in this tutorial is a simple two-input XOR. The design will be described in Verilog-HDL. Launch Vivado using the desktop or start menu shortcut. From the Vivado start screen, select “Create New Project”. The first of the New Project dialog boxes will appear, as shown in Figure 13.

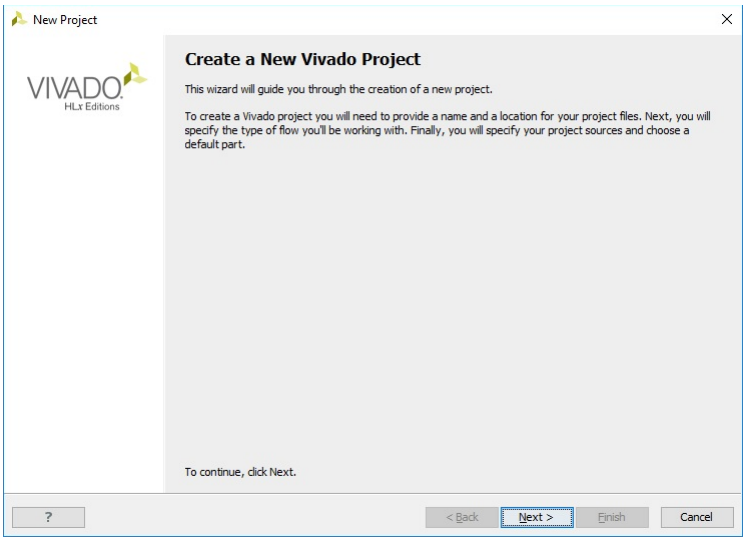

Figure 13: Create New Project

You are prompted to enter a project name, a project location, and if you want to create a project subdirectory. Do not use file or folder names that contain spaces or are excessively long (i.e. avoid creating projects on the desktop). When you are satisfied with the project name and location, click "Next" as shown in Figure 14.

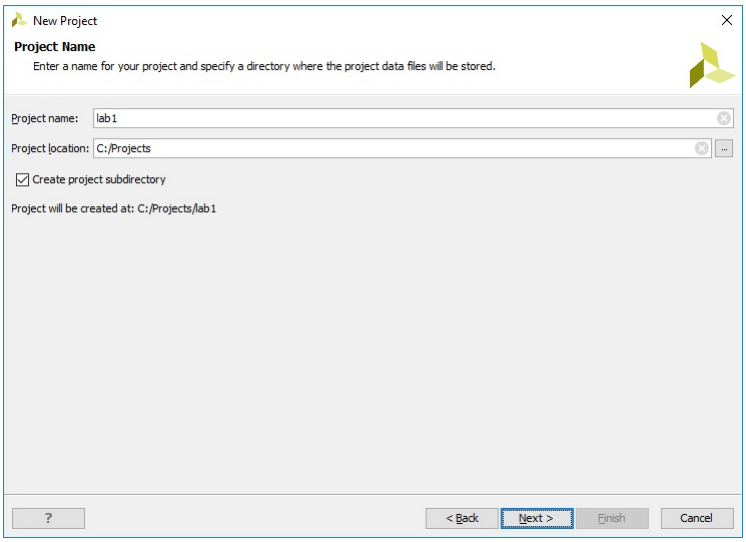

Figure 14: Create New Project

The next dialog allows you to select the project type. Select an "RTL Project" as shown in Figure 15. Also select "Do not specify sources at this time" because you don't have source files to add to your project yet.

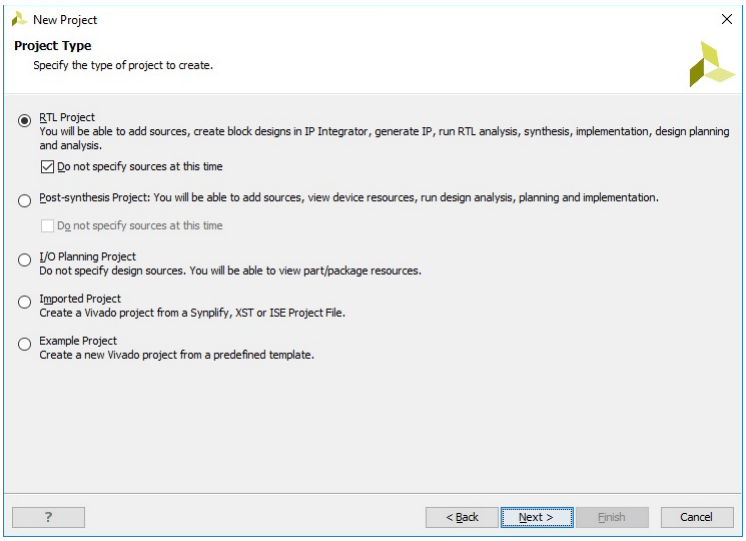

Figure 15: Create New Project

The next dialog allows you to select the FPGA device you are using. Select specification by part, and then use the filters to limit the selection as shown in Figure 16. Select the "xc7a35tcpg236-1" device. This is the device on the Digilent Basys3 board. Please pay close attention to select the correct device.

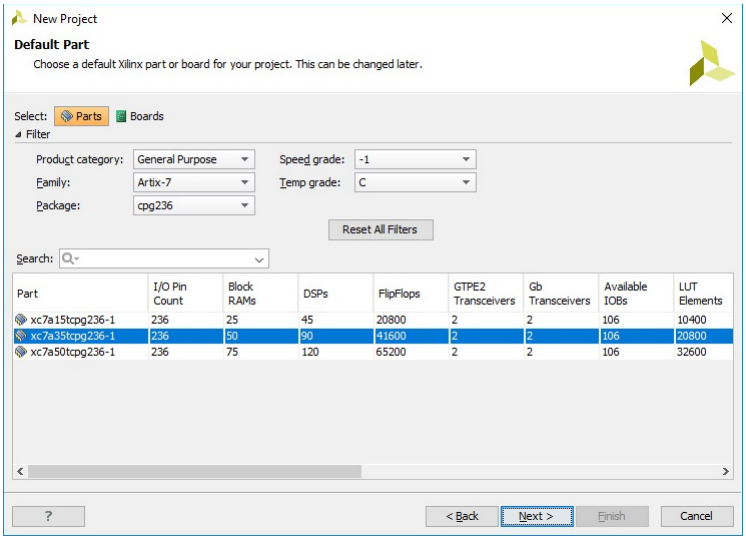

Figure 16: Create New Project

Before completing the creation of a new project, Vivado provides a summary of your selections as shown in Figure 17. Finish the creation of your new project.

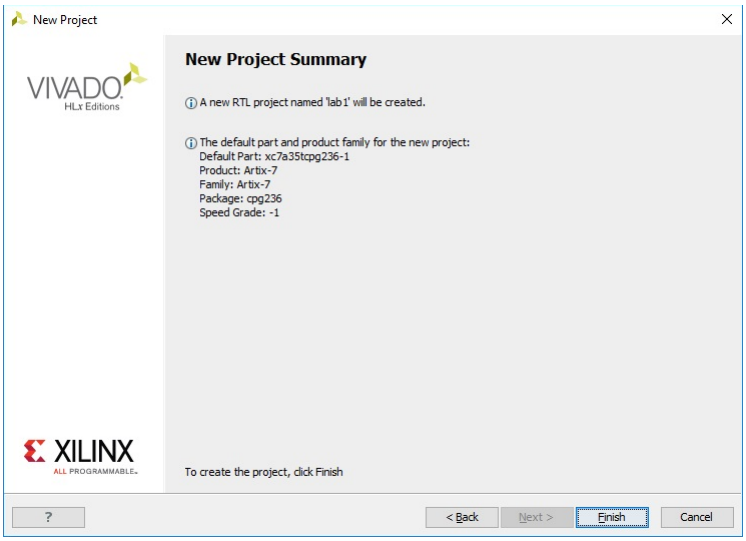

Figure 17: Create New Project

With the new project created and automatically opened, Vivado transitions to the Project Manager interface. In the Flow Navigator pane of the Project Manager, click on "Add Sources". This will step you through the addition of existing files to the project, or the creation of new files for the project. Indicate that you want to add or create design sources as shown in Figure 18.

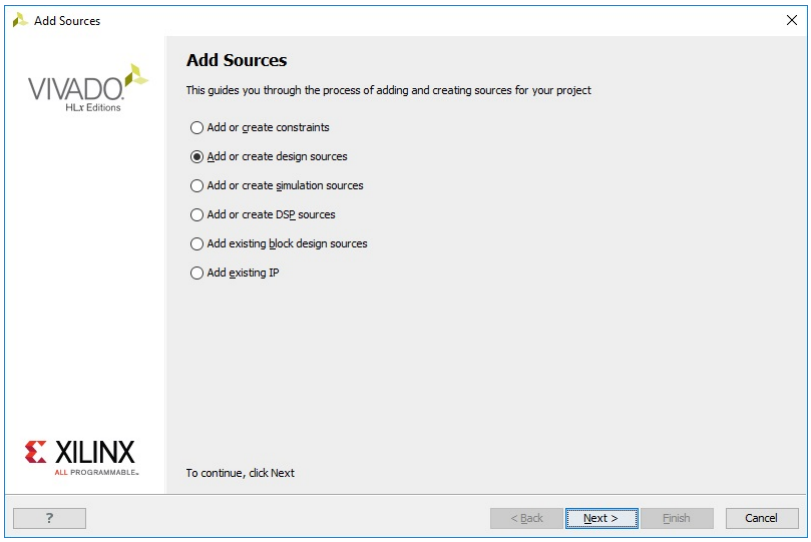

Figure 18: Add or Create Sources

Click the "+" button to access the "Create File" option, to create a new file as shown in Figure 19. You will enter the source file contents after the file is created.

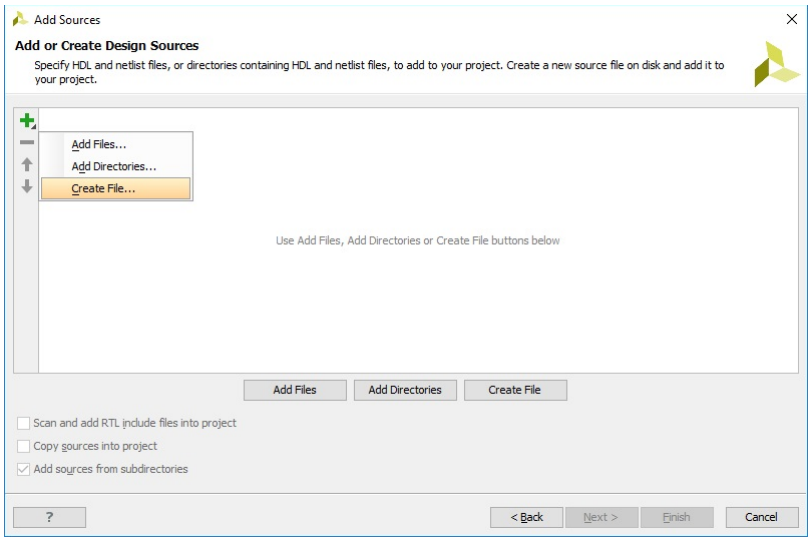

Figure 19: Add or Create Sources

During the source file creation process, you must also indicate the file type, the file name, and the file location. Refer to Figure 20 to create the two_input_xor.v source file for this project.

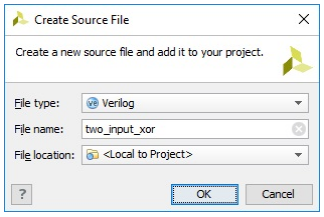

Figure 20: Create Source

After the two_input_xor.v source file has been created, it will appear in the "Add Sources" dialog as shown in Figure 21. Here, simply click "Finish".

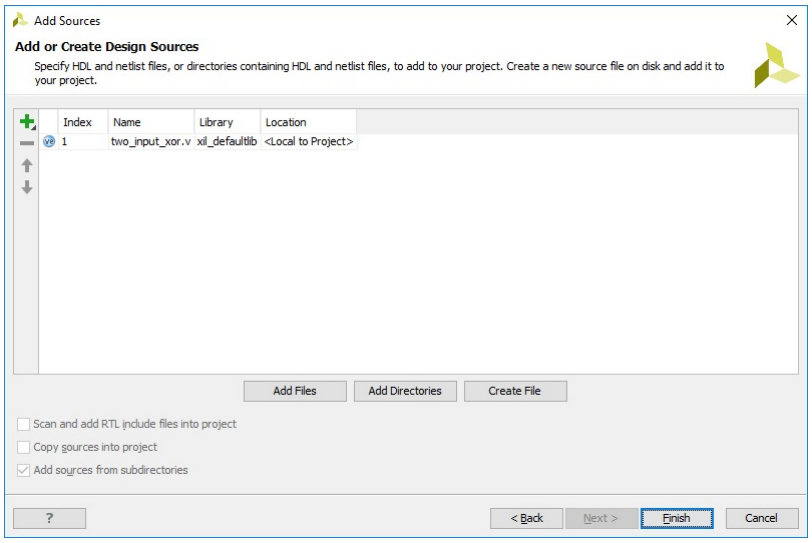

Figure 21: Add or Create Sources

As a convenience, Vivado will offer to assist you in creating the contents of the source file. However, we will enter the file contents from scratch, so do not enter anything into the "Define Module" dialog, just click through it as shown in Figure 22.

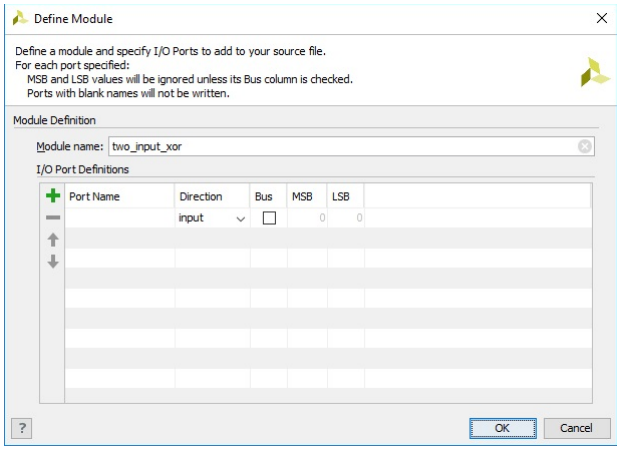

Figure 22: Define Module

Since you didn't change anything in the "Define Module" dialog, Vivado will prompt you to confirm that is what you had intended. After you confirm your intent, Vivado will return to the Project Manager interface, and in the sources pane, you can expand the design sources tree to locate the newly created two_input_xor.v source. Double click on the source file to open it for editing as shown in Figure 23.

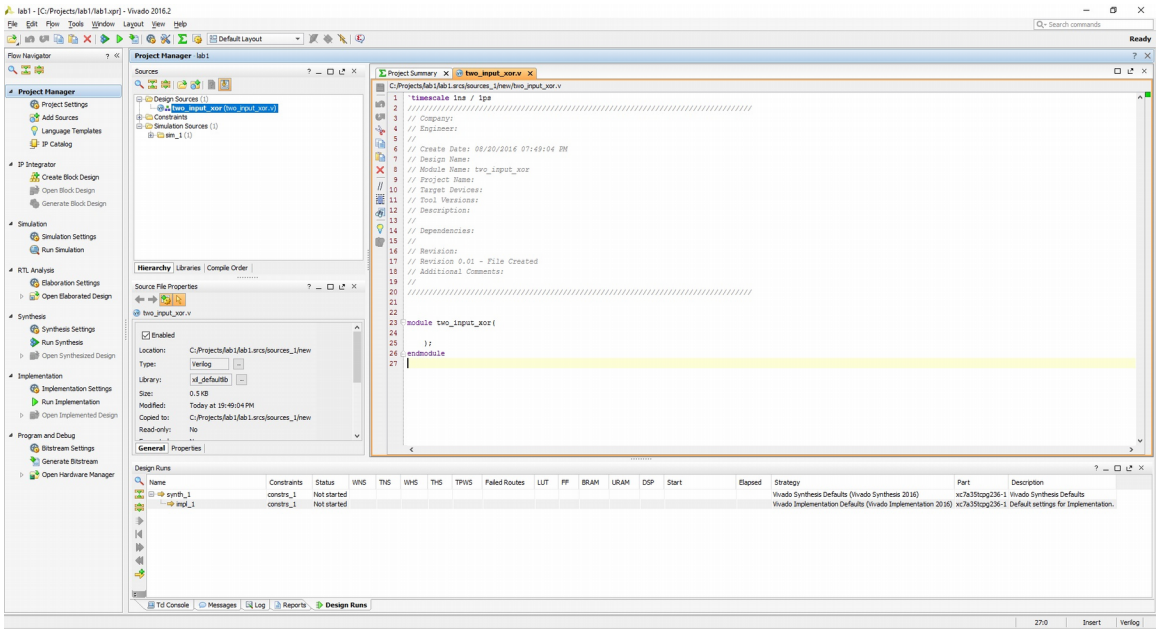

Figure 23: New Source (Auto Generated)

In the text editor, some of the basic file structure is already in place although we are going to replace everything that was automatically generated. Enter the design:

```
// File:  two_input_xor.v
// This is the top level design for EE178 Lab #1.

// The `timescale directive specifies what the
// simulation time units are (1 ns here) and what
```

```
// the simulator time step should be (1 ps here).

`timescale 1 ns / 1 ps

// Declare the module and its ports.  This is
// using Verilog-2001 syntax.

module two_input_xor (
  input  wire       in1,
  input  wire       in2,
  output wire       out
  );

  // You could substitute other possible descriptions.

  assign out = in1 ^ in2;

endmodule
```

In many of the lab assignments, code is provided in the handout and you may copy and paste it into your project. Please be aware that if you are selecting text that spans multiple pages, you may also select the page numbers at each page boundary. If you then paste it into the text editor, you'll end up with occasional page numbers in the middle of your code which is most definitely a syntax error.

So yes, please copy and paste the text, but be aware of the page boundaries and copy and paste in small pieces around the page numbers or review what you have pasted afterward, to remove the page numbers. Be aware that if you get a syntax error during compilation of code from copy and paste, it's not expected. The code has been tested and is known to work – so the first things you should check for are page numbers that escaped your detection.

Once you are satisfied, save the file using the “Save File” icon in the vertical toolbar along the left edge of the text editor. At this point, you should end up with a window that looks like that shown in Figure 24.

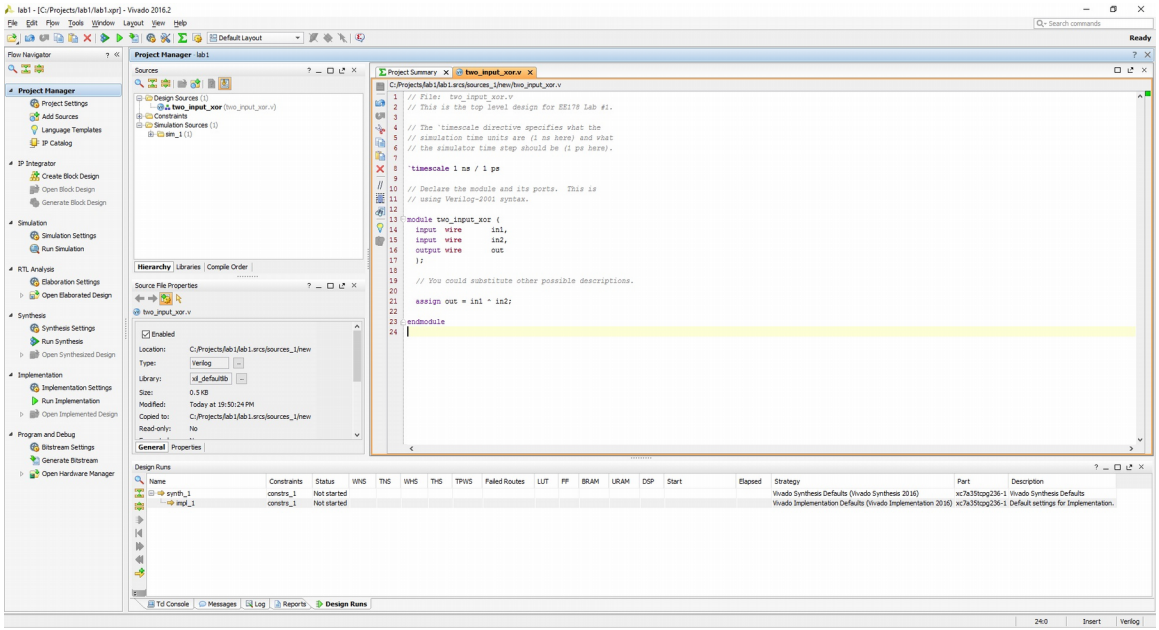

Figure 24: New Source (Edited)

Functional Simulation

Functional simulation is done before the design is implemented to verify the logic you have described is correct. This allows a designer to find and fix any bugs in the design before spending time with subsequent steps. Vivado contains an integrated logic simulator.

You will need to add a simulation testbench to your project. Adding the testbench uses virtually the same "Add Sources" process you completed for the design. The one exception is that you must select "Add or Create Simulation Sources" instead of "Add or Create Design Sources". Create the testbench.v file in the project, and replace the automatically generated file contents with the following:

```
// File:  testbench.v
// This is a top level testbench for EE178 Lab #1.

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

`timescale 1 ns / 1 ps

module testbench;

  // 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.  Then wait 100 ns for
    // the Xilinx global reset/tristate to be
    // deasserted before doing anything.
    test_passed = 1'b1;
    #100;

    // Test Case #0
    sig1 = 1'b0;
```

```
    sig2 = 1'b0;
    #50;
    $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;
    #50;
    $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;
    #50;
    $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;
    #50;
    $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
```

Once you are satisfied, save the file. At this point, you should end up with a window that looks like that shown in Figure 25.

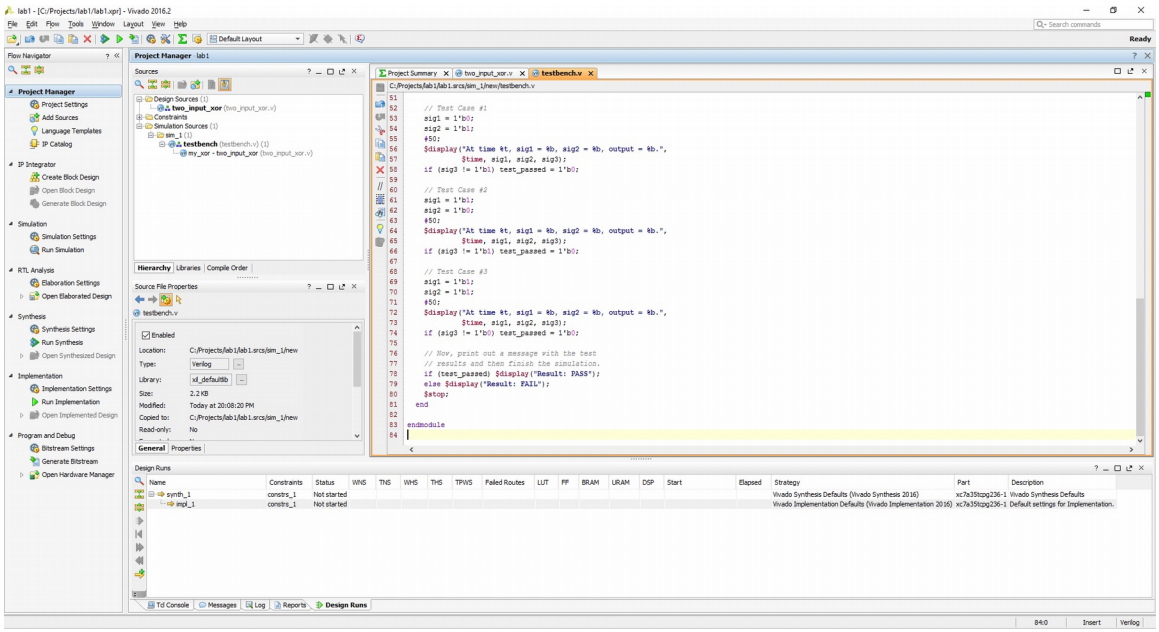

Figure 25: New Source (Edited)

Now that you have a testbench in your project, you can perform behavioral simulation of the design. While there is a "Run Simulation" button in the Flow Navigator pane, instead to go the main menu and select Flow→Run Simulation→Run Behavioral Simulation. The simulator will compile the testbench and the design, and run the simulation. A successful result is shown in Figure 26, note the "Result: PASS" printed by the testbench to the console window.

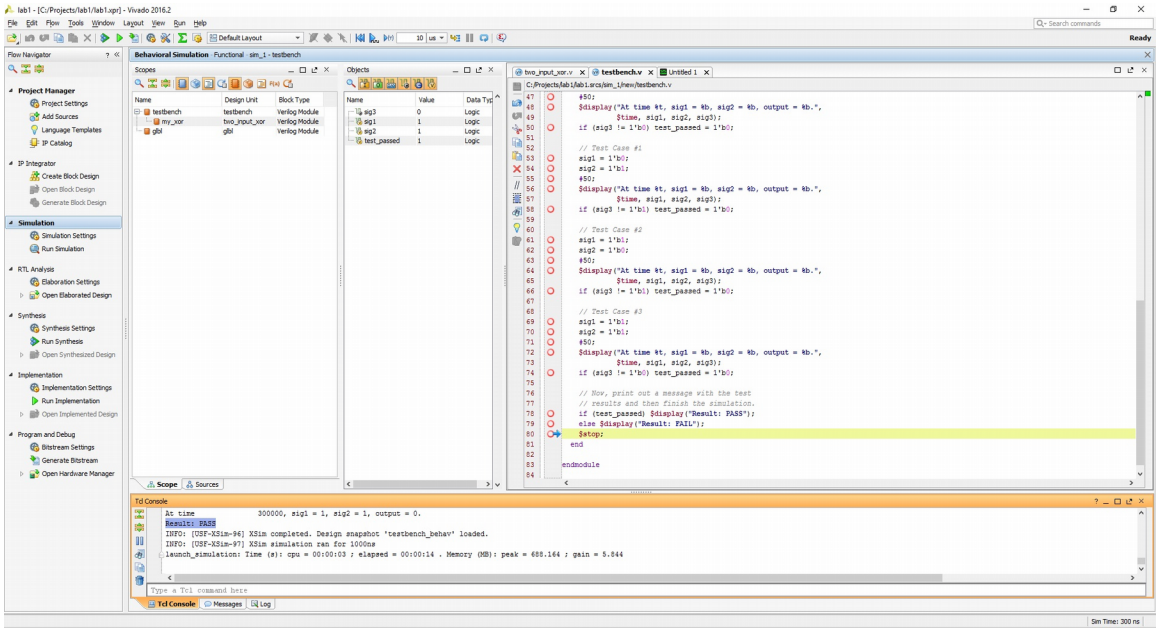

Figure 26: Behavioral Simulation

This simulation is short, so it will run to completion without your intervention. In cases where a testbench is running a longer simulation, you will find that the simulator pauses after a fixed amount of simulation time. If you need to continue to simulate longer, you can adjust the "Simulation Settings" run time through the button in the Flow Navigator prior to running the simulation, or use the options in the main menu listed

under “Run” to continue the simulation once you are already running the simulation. You’ll also find a simulation restart option available under the “Run” menu.

The simulator generates a waveform display of the testbench signals, accessible through the “Untitled 1” tab above the text editor. Select the waveform display and play around with the waveform window (including right clicking in it for a context sensitive menu) to figure out how to zoom in, zoom out, zoom fit, and scroll the waveform. If you zoom fit, you should see a waveform like that in Figure 27.

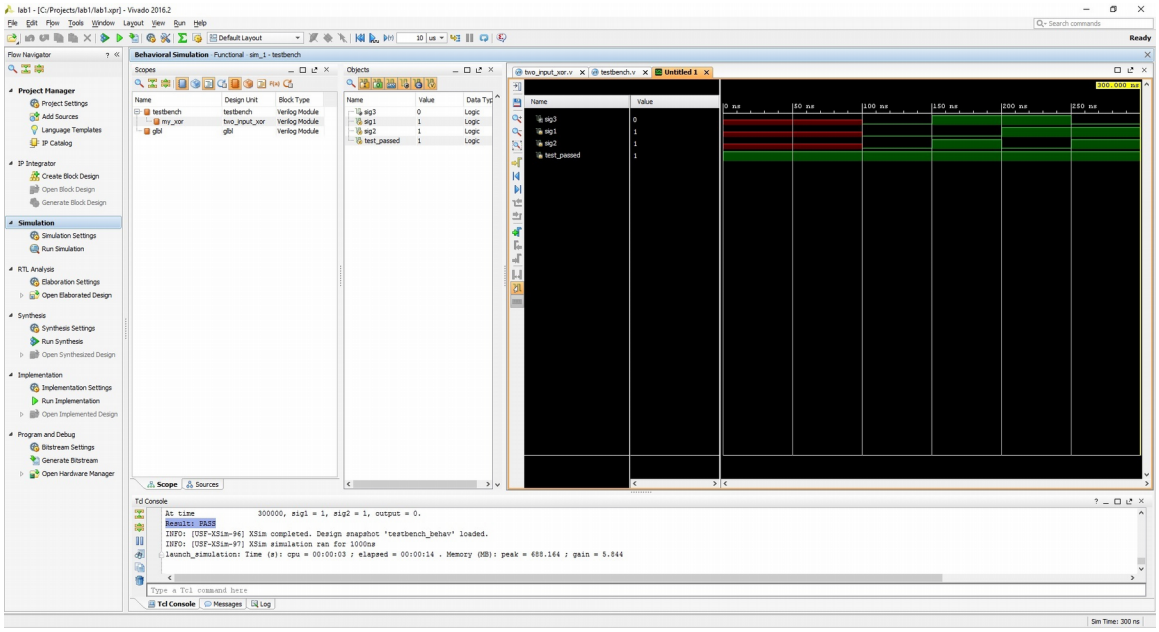

Figure 27: Behavioral Simulation Waveform

It is possible to explore the testbench as well as the design under test. Using the Scopes pane and the Objects pane, you can browse through the design and locate signals to add to the waveform window. Adding signals to the waveform display is accomplished by right clicking on the signal name in the Object pane, or simply dragging and dropping the signal into the waveform window.

By default, when the simulation runs, data is stored only for the signals present in the waveform window. When new signals are added, it is necessary to restart the simulation and re-run it.

Now that you are done with behavioral simulation, close the simulator by clicking on the “X” in the light blue Behavioral Simulation title bar.

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. Vivado contains an integrated logic synthesizer. Simply click on “Run Synthesis” in the Flow Navigator pane. When synthesis has completed, you are offered several options. Select “View Reports” which returns you to the Project Manager.

Since you have been cutting and pasting source file contents, you should not expect any errors. However, you should always review the report files, some of which are now available for viewing after the completion of synthesis. Browse through the available reports by selecting the “Reports” tab at the bottom of the window.

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 Artix-7 FPGA, which are connected to various resources on the Digilent Basys3 board.

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 Digilent Basys3 board, you will notice that many resources have been thoughtfully annotated with text indicating which FPGA pins are associated with them. This information is also available the board schematic and also conveniently provided in a sample constraints file provided by Digilent. The schematic and the sample constraints file are linked from the class website. The following constraints were created from the sample constraints file, extracting only those constraints related to SW0, SW1, and LD0:

```
# Constraints for SW0
set_property PACKAGE_PIN V17 [get_ports {in1}]
set_property IOSTANDARD LVCMOS33 [get_ports {in1}]

# Constraints for SW1
set_property PACKAGE_PIN V16 [get_ports {in2}]
set_property IOSTANDARD LVCMOS33 [get_ports {in2}]

# Constraints for LD0
set_property PACKAGE_PIN U16 [get_ports {out}]
set_property IOSTANDARD LVCMOS33 [get_ports {out}]

# Constraints for CFGBVS
set_property CFGBVS VCCO [current_design]
set_property CONFIG_VOLTAGE 3.3 [current_design]
```

You now have the information to create what is called a Xilinx design constraints file, or XDC file. This file contains design constraints that you did not specify in the Verilog-HDL description, such as pin locations, signaling voltages, and design performance constraints. It is convenient to provide constraints in a separate file rather than in the Verilog-HDL description.

You will need to add a constraints file to your project. Adding the constraints file uses virtually the same "Add Sources" process you completed for the design. The one exception is that you must select "Add or Create Constraints" instead of "Add or Create Design Sources". Create the two_input_xor.xdc file in the project, find it in the Sources pane, edit it, and enter the constraints given above. Save the file.

Simply click on "Run Implementation" in the Flow Navigator pane. Vivado may request to re-run synthesis if file timestamps or contents have changed – allow it to do so. When implementation has completed, you are offered several options. Select "View Reports" which returns you to the Project Manager.

Since you have been cutting and pasting source file contents, you should not expect any errors. However, you should always review the report files, some of which are now available for viewing after the completion of implementation. Browse through the available reports by selecting the "Reports" tab at the bottom of the window.

Programming the FPGA Directly

Programming the FPGA directly is a convenient way to try out a design. This method is useful for prototyping when you are not certain your design is final. At this point you are fairly confident your tutorial design is correct. However, 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, called a bitstream, for the FPGA. Click on the "Generate Bitstream" button in the Flow Navigator. When bitstream generation has completed, you are offered several options. Select "View Reports" which returns you to the Project Manager.

Before you continue, you must connect the Digilent Basys3 board to your computer using the supplied USB cable. Turn the power switch on. Please note that the first time you connect the board to a new USB port, you should expect a visit from the operating system's "Device Setup" wizard. Allow the driver installation to compete before you proceed (it may take several minutes).

Xilinx FPGAs are called SRAM FPGAs because they store their bitstream in internal SRAM. When power is removed, the SRAM contents are lost – so each time power is applied, the FPGA is blank. However, be aware that if a bitstream has been programmed into the Basys3 on-board non-volatile memory (a separate device from the FPGA) then the FPGA will load itself with that bitstream when power is applied. This may result in board activity like flashing LEDs, etc. and can safely be ignored.

From the Flow Navigator pane, expand the "Open Hardware Manager" and select "Open Target". Use the option to "Auto Connect". During this process, your firewall may issue warnings, in response to which you should allow Vivado access to the network ports it requests.

At the completion of this process, the hardware target is ready for programming and Vivado takes you to the Hardware Manager. You may ignore any warnings or informational notes regarding "debug cores" as we have not included any in this project. There are several ways to initiate device programming, the easiest being to click on "Program Device" shown in the green status bar towards the top of the window.

When you click "Program Device" the Hardware Manager will provide a list of devices available for programming. You should select the only device listed, "xc7a35t_0", after which a window will appear to confirm the filename for the bitstream to be used. Select the bitstream file you created in your project, as shown in Figure 28. Accept the default file.

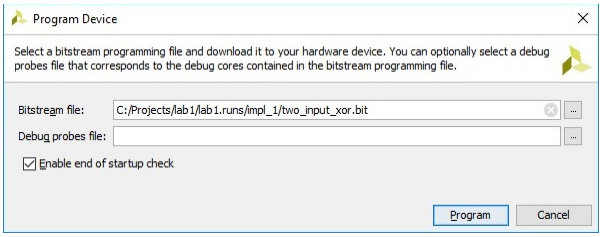

Figure 28: Bitstream File Confirmation

A programming progress indicator will appear. Once the programming is complete, the program will be sure to let you know if it failed. If the programming has failed, re-check your cable connections and the power switch. 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. Once you are confident it works properly, demonstrate your final result to the instructor.

As a convenience feature for giving demonstrations, there is another method to program the device on the Digilent Basys3 board. Using a USB flash drive freshly formatted with the FAT32 filesystem, copy a single file – the bitstream (see Figure 28) onto the drive. Plug the drive into the board, and the board will program the device from the bitstream on the drive, provided the JP1 jumper on the board has been set to enable device programming from a USB flash drive. Details on this feature are in the Digilent Basys3 reference manual posted on the class website.

Laboratory Hand-In Requirements

Once you have completed a working design, prepare for the submission process. You are required to demonstrate a working design. Within four hours of your demonstration, you are required to submit your entire project directory in the form of a compressed ZIP archive. Use WinZIP to archive the entire project directory, and name the archive lab1_yourlastname_yourfirstname.zip. For example, if I were to make a submission, it would be lab1_crabill_eric.zip. Then email the archive to the instructor. Only WinZIP archives will be accepted.

Demonstrations must be made on or before the due date. If your circuit is not completely functional by the due date, you should turn in what you have to receive partial credit.