San Jose State University
Department of Electrical Engineering
EE178, Fall 2021

Laboratory Assignment #1

Objectives

This lab is an introduction to logic design using Verilog-HDL with Xilinx Vivado. 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. Please read carefully, pay attention, and take your time. This lab is not a race to see who gets done first.

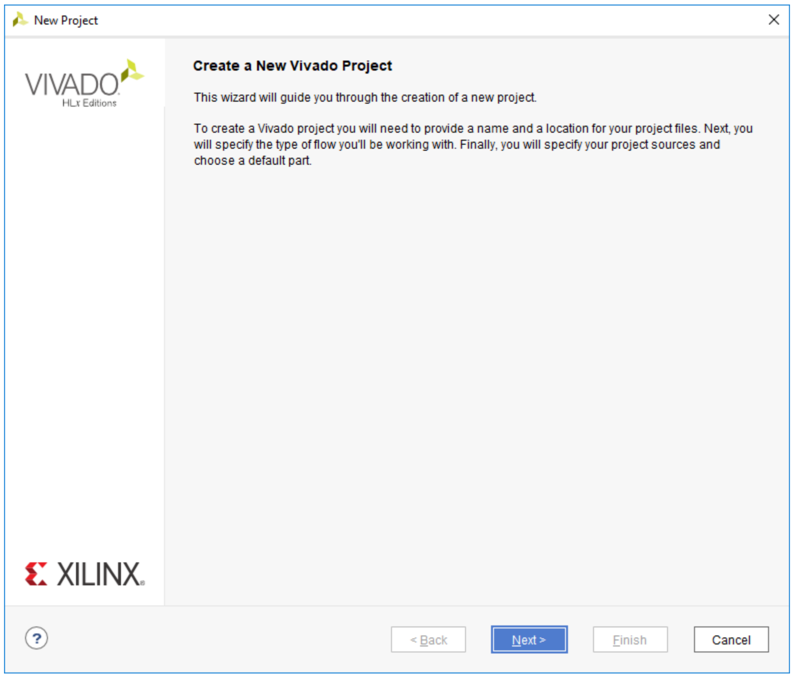
In order to receive credit for this lab, you must demonstrate to the instructor that your final design works correctly on hardware. For online demonstration, a narrated video recording submission along with the archival zip file will be required. The instruction to create the archival zip file is at the end of this document.

Bibliography

This lab draws heavily from documents on the Xilinx website https://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 Real Digital Blackboard.

Design Entry

The design used in this tutorial is a 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, under "Quick Start" select "Create Project". The first of the New Project dialog boxes will appear, as shown in Figure 1.



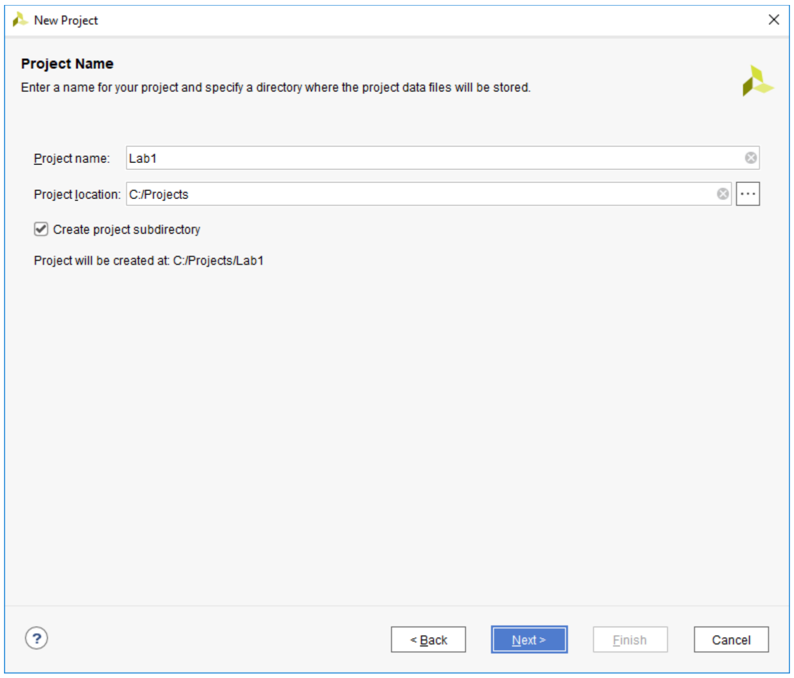
Create a New Vivado Project

This wizard will guide you through the creation of a new project.

To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.

Figure 1: New Project

Clicking “Next” brings you to the next dialog. 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 2.



Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: Lab1

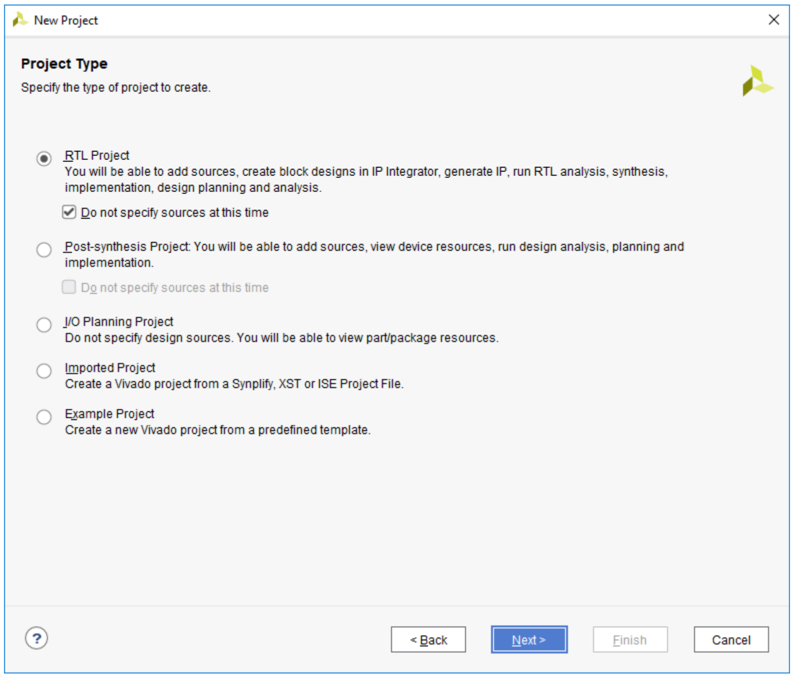
Project location: C:/Projects

☑ Create project subdirectory

Project will be created at: C:/Projects/Lab1

Figure 2: New Project

The next dialog allows you to select the project type. Select an “RTL Project” as shown in Figure 3. Also select “Do not specify sources at this time” because you don’t have source files to add to your project yet.



New Project

Project Type

Specify the type of project to create.

[✓ RTL Project]
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.

[✓ Do not specify sources at this time]

Post-synthesis Project: You will be able to add sources, view device resources, run design analysis, planning and implementation.

Do not specify sources at this time

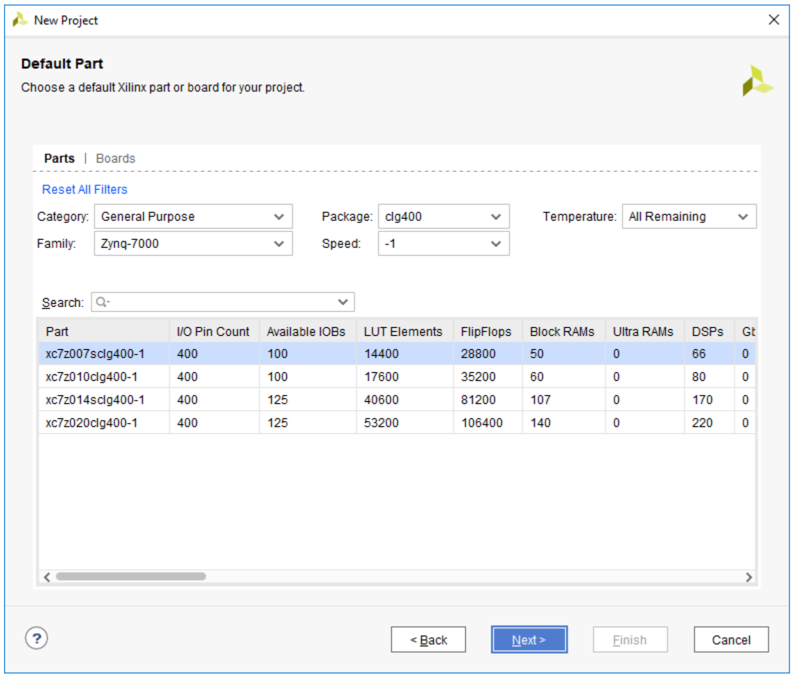
I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

Imported Project
Create a Vivado project from a Synplify, XST or ISE Project File.

Example Project
Create a new Vivado project from a predefined template.

Figure 3: New Project

The next dialog allows you to select the Xilinx device you are using. Select by "Parts", and then use the filters to limit the selection as shown in Figure 4. Select the "xc7z007sclg400-1" device. This is the device on the Real Digital Blackboard. Please pay close attention to select the correct device.



New Project

Default Part

Choose a default Xilinx part or board for your project.

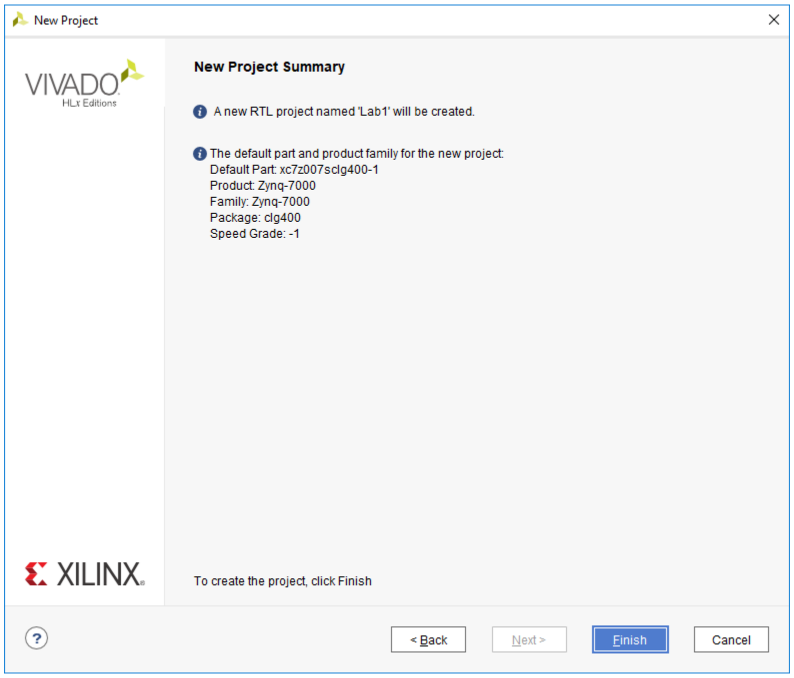
Category: General Purpose
Package: clg400
Temperature: All Remaining
Family: Zynq-7000
Speed: -1

Search:

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gt
xc7z007sclg400-1	400	100	14400	28800	50	0	66	0
xc7z010clg400-1	400	100	17600	35200	60	0	80	0
xc7z014sclg400-1	400	125	40600	81200	107	0	170	0
xc7z020clg400-1	400	125	53200	106400	140	0	220	0

Figure 4: New Project

Before completing the creation of a new project, Vivado provides a summary of your selections as shown in Figure 5. Finish the creation of your new project.


New Project

New Project Summary

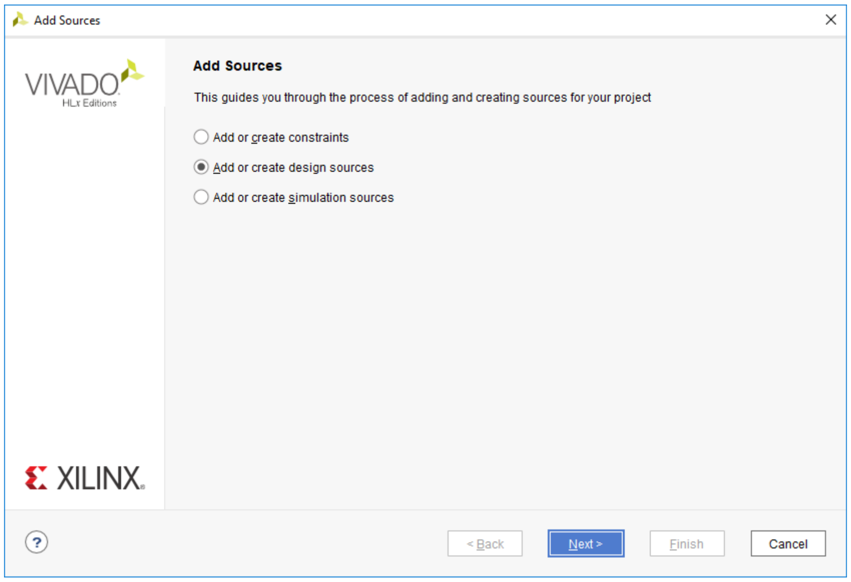
A new RTL project named 'Lab1' will be created.

The default part and product family for the new project:
Default Part: xc7z007sclg400-1
Product: Zynq-7000
Family: Zynq-7000
Package: clg400
Speed Grade: -1

To create the project, click Finish

Figure 5: 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 6.


Add Sources

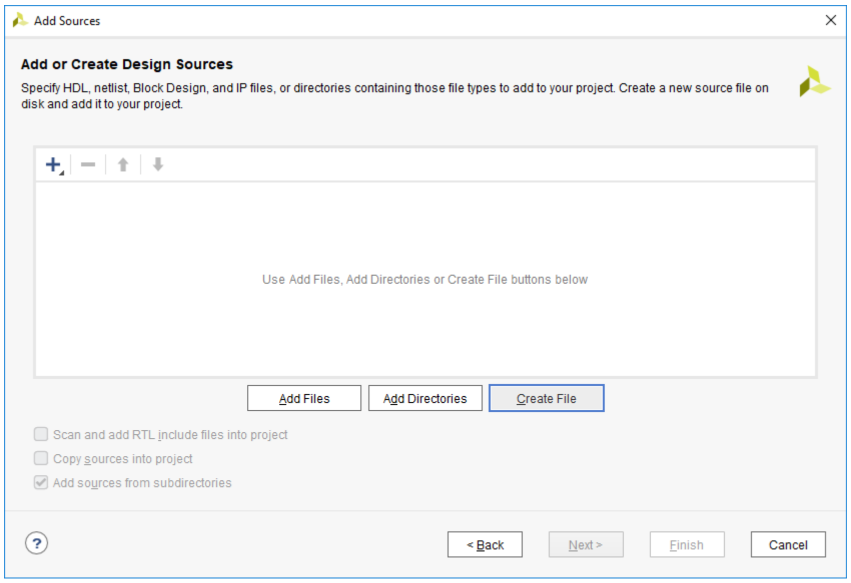
Add Sources

This guides you through the process of adding and creating sources for your project

- ○ Add or create constraints
- ● Add or create design sources
- ○ Add or create simulation sources

Figure 6: Add Sources

Click the "Create File" button, to create a new file as shown in Figure 7. You will enter the source file contents after the file is created.



Add Sources

Add or Create Design Sources

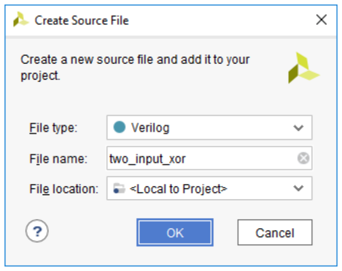
Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.

Use Add Files, Add Directories or Create File buttons below

Add Files
Add Directories
Create File
Scan and add RTL include files into project
Copy sources into project
Add sources from subdirectories
< Back
Next >
Finish
Cancel

Figure 7: Add Sources

During the source file creation process, you must also indicate the file type, the file name, and the file location. Refer to Figure 8 to create the two_input_xor.v source file for this project.



Create Source File

Create a new source file and add it to your project.

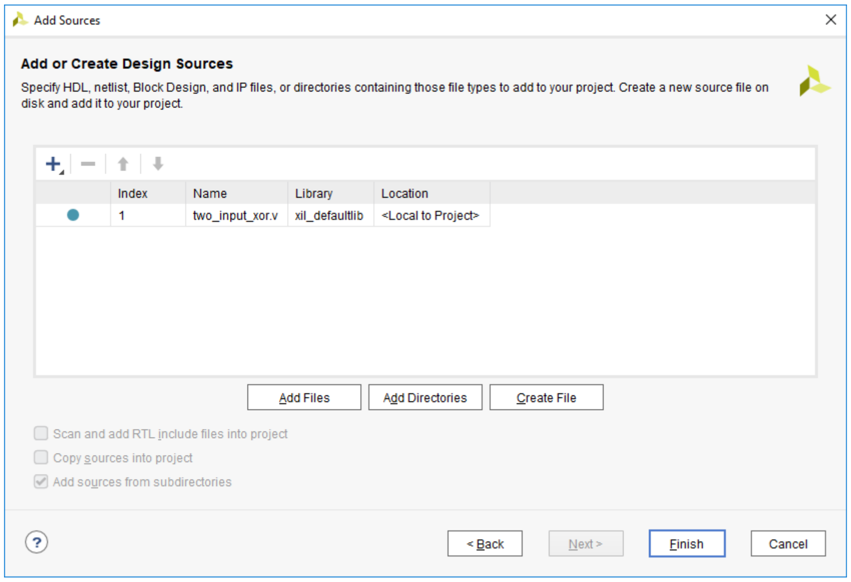
File type: Verilog

File name: two_input_xor

File location: <Local to Project>

Figure 8: Create Source File

After the two_input_xor.v source file has been created, it will appear in the dialog as shown in Figure 9. Here, simply click “Finish”.



Add Sources

Add or Create Design Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.

	Index	Name	Library	Location
●	1	two_input_xor.v	xil_defaultlib	<Local to Project>

Add Files | Add Directories | Create File

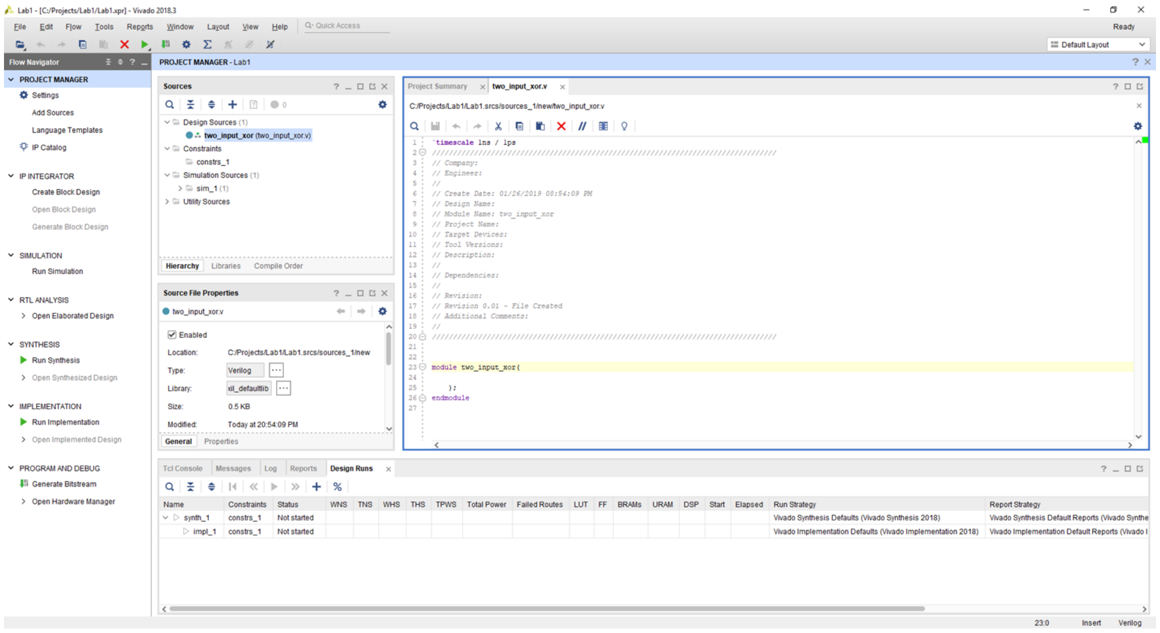
☐ Scan and add RTL include files into project
☐ Copy sources into project
☑ Add sources from subdirectories

< Back | Next > | Finish | Cancel

Figure 9: Add Sources

At this point, Vivado will offer to assist you in creating the contents of the source file by presenting a Define Module dialog. However, you will enter the file contents from scratch, so do not change anything in the Define Module dialog, just click "Okay". When you do this, since you haven't changed anything, Vivado will ask if you are sure, to which you should respond "Yes".

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 10.



Lab1 - [C:/Projects/Lab1/Lab1.xpr] - Vivado 2018.3

C:/Projects/Lab1/Lab1.srcs/sources_1/new/two_input_xor.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01/26/2019 08:54:09 PM
// Design Name:
// Module Name: two_input_xor
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module two_input_xor(

    );
endmodule
```

Source File Properties: two_input_xor.v

☑ Enabled
Location: C:/Projects/Lab1/Lab1.srcs/sources_1/new
Type: Verilog
Library: xil_defaultlib
Size: 0.5 KB
Modified: Today at 20:54:09 PM

Name	Constraints	Status	Run Strategy	Report Strategy
synth_1	constrs_1	Not started	Vivado Synthesis Defaults (Vivado Synthesis 2018)	Vivado Synthesis Default Reports (Vivado Synthe
impl_1	constrs_1	Not started	Vivado Implementation Defaults (Vivado Implementation 2018)	Vivado Implementation Default Reports (Vivado I

Figure 10: New Source (Auto Generated)

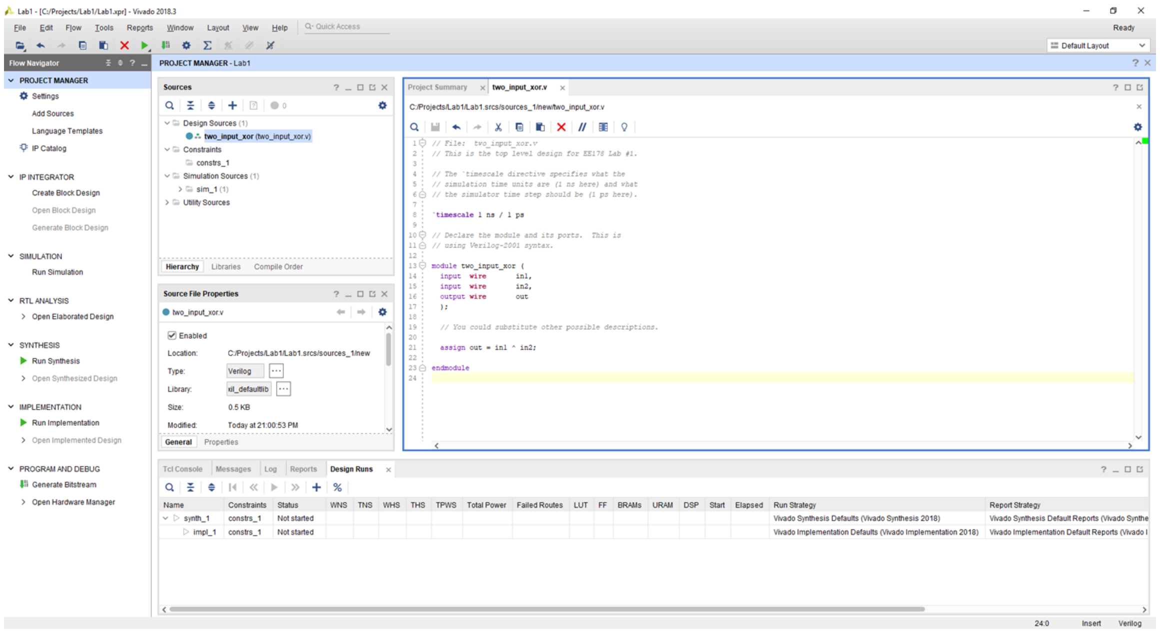
In the text editor, some of the basic file structure is already in place although you are going to replace everything that was automatically generated. Enter the design:

```
// 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 PDF 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.

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" button in the horizontal toolbar along the top edge of the text editing window. At this point, you should end up with a window that looks like that shown in Figure 11.



C:/Projects/Lab1/Lab1.srcs/sources_1/new/two_input_xor.v

```
// 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
```

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy	Report Strategy
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (Vivado Synthesis 2018)	Vivado Synthesis Default Reports (Vivado Synthe
impl_1	constrs_1	Not started															Vivado Implementation Defaults (Vivado Implementation 2018)	Vivado Implementation Default Reports (Vivado I

Figure 11: New Source (Edited and Saved)

Functional Simulation

Functional simulation is done before the design is implemented to verify the logic as 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.

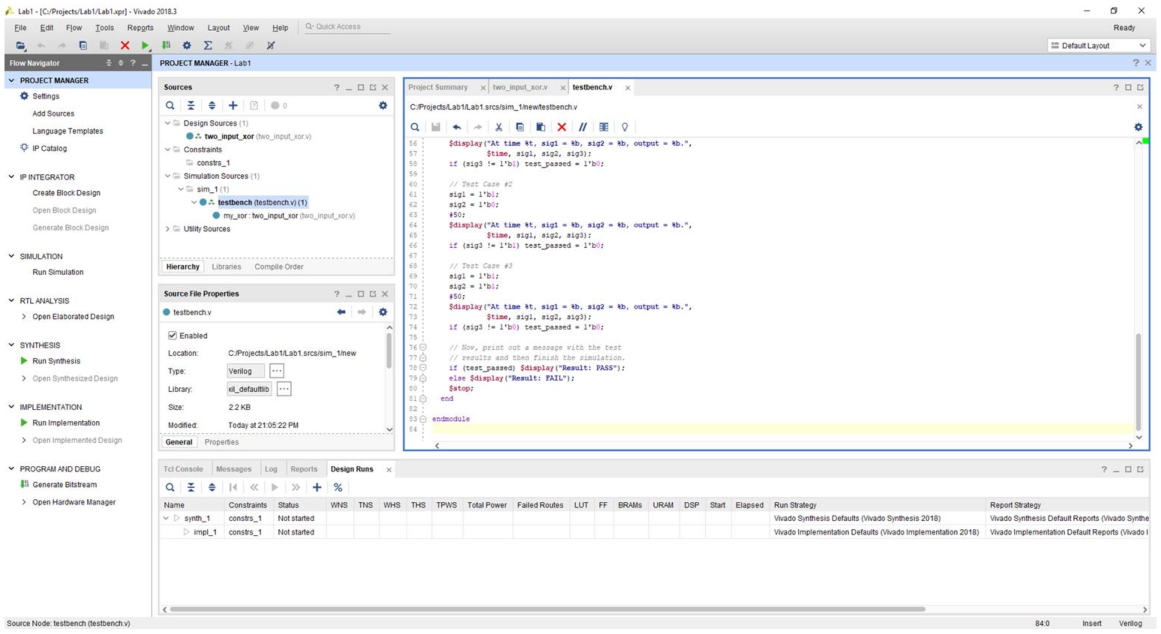
You will need to add a simulation testbench to your project. Adding the testbench uses virtually the same "Add Sources" process you completed for the logic design source. The one exception is that you must select "Add or create simulation sources" instead of "Add or create design sources". This step is very important as it ensures Vivado knows the testbench is not intended as part of the logic design. 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 12.

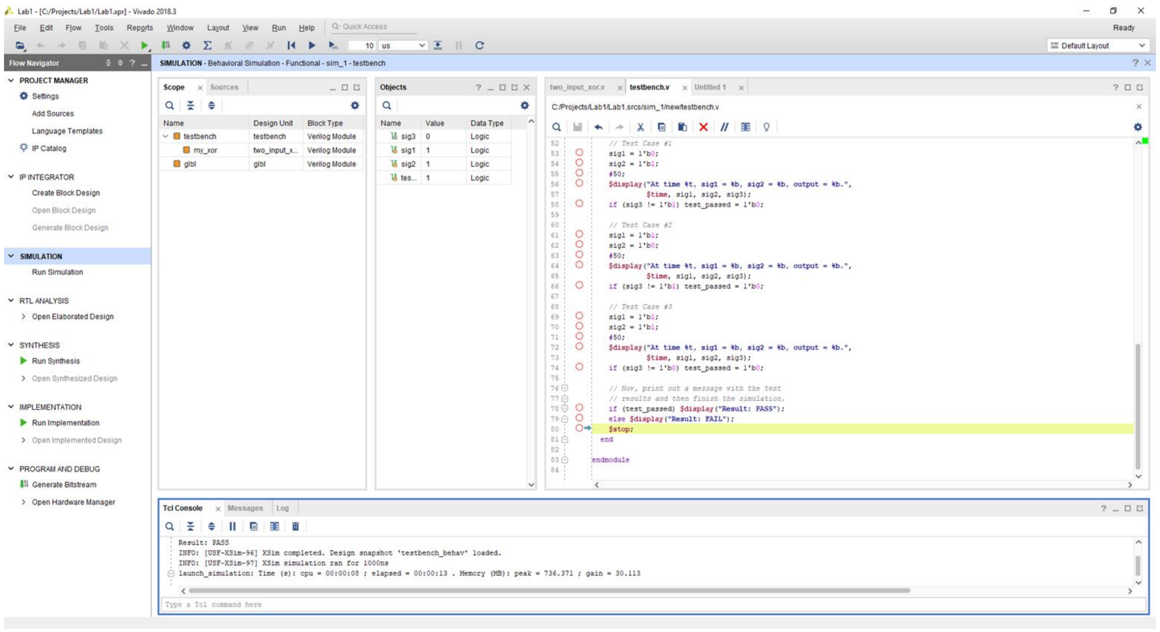


```
C:/Projects/Lab1/Lab1.srcs/sim_1/new/testbench.v

    $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
```

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy	Report Strategy
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (Vivado Synthesis 2018)	Vivado Synthesis Default Reports (Vivado Synthe
impl_1	constrs_1	Not started															Vivado Implementation Defaults (Vivado Implementation 2018)	Vivado Implementation Default Reports (Vivado I

Figure 12: New Source (Edited and Saved)

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 13, note the “Result: PASS” printed by the testbench to the console window. Depending on the size of the window, you may need to scroll up a few lines to see it.



C:/Projects/Lab1/Lab1.srcs/sim_1/new/testbench.v

```verilog
    // 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
```

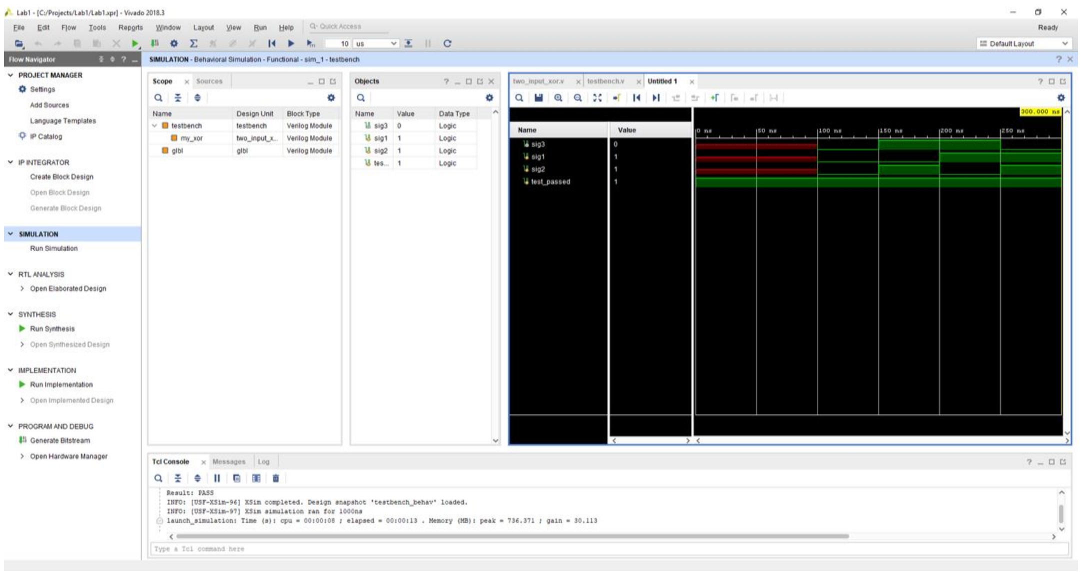
Name	Value	Data Type
sig3	0	Logic
sig1	1	Logic
sig2	1	Logic
tes...	1	Logic

```
Result: PASS
INFO: [USF-XSim-96] XSim completed. Design snapshot 'testbench_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:08 ; elapsed = 00:00:13 . Memory (MB): peak = 736.371 ; gain = 30.113
```

Figure 13: Behavioral Simulation

This simulation is short, so it will run to completion quickly, and without your intervention. In cases where a testbench is running a longer simulation, you will find that the simulator pauses after an interval of simulation time. If you need to continue to simulate longer, you can use the options in the main menu listed under "Run" to continue the simulation once you have started it. You'll also find a simulation restart option available under the "Run" menu.

The simulator generates a waveform display, accessible through the "Untitled 1" tab above the text editor. By default, the only signals shown are those in the testbench, as it is the top level of the simulation. 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 14.



Name	Value
sig3	0
sig1	1
sig2	1
test_passed	1

```
Result: PASS
INFO: [USF-XSim-96] XSim completed. Design snapshot 'testbench_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:08 ; elapsed = 00:00:13 . Memory (MB): peak = 736.371 ; gain = 30.113
```

Figure 14: Behavioral Simulation Waveform

It is possible to probe signals in 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. When you do this, you will notice that the signal name shows up in the waveform window, but there is no waveform shown for it.

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. You can do this using the menu, use "Run" → "Restart..." and then "Run" → "Run All".

Now that you are done with behavioral simulation, close the simulator by clicking on the "X" in the light blue 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 and then accept the default settings to launch the run. 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 Xilinx 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 device, which are connected to various resources on the Real Digital Blackboard.

The top-level design has two input ports, and a single output port. Switches SW0 and SW1 are going to be connected to the inputs. Additionally, the output is going to be connected to an LED – indicator LD0 is appropriate.

If you inspect the top of the Real Digital Blackboard, you will notice that many resources have been thoughtfully annotated with text indicating which physical pins on the device are associated with them. This information is also available in the board schematic and also provided in reference materials by Real Digital. The following constraints were created in part by using the markings on the Blackboard:

```
# Constraints for SW0
set_property PACKAGE_PIN R17 [get_ports {in1}]
set_property IOSTANDARD LVCMOS33 [get_ports {in1}]

# Constraints for SW1
set_property PACKAGE_PIN U20 [get_ports {in2}]
set_property IOSTANDARD LVCMOS33 [get_ports {in2}]

# Constraints for LD0
set_property PACKAGE_PIN N20 [get_ports {out}]
set_property IOSTANDARD LVCMOS33 [get_ports {out}]
```

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 – allow it to do so. Accept the default settings to launch the run. 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 Device Directly

Programming the device 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 programmable devices have over ASICs is that the penalty for being wrong is minimal.

The first order of business is to create a programming file, called a bitstream. Click on the "Generate Bitstream" button in the Flow Navigator and then accept the default settings to launch a run. When bitstream generation has completed, you are offered several options. Select "View Reports" which returns you to the Project Manager.

Before you continue, you will need your Real Digital Blackboard and an appropriate USB data cable. Please also verify the jumper settings are set so that the Blackboard is running from USB power (JP3 set to USB) and in JTAG mode (JP2 set to JTAG). There should be no other shorting blocks installed on any other jumpers, and the power switch should be in the OFF position.

Then, connect the USB data cable between a port on your computer and connector J10 on the Blackboard, which is labeled PROG. Turn the power switch on. Please note that the first time you connect the board to a new USB port, you might get a visit from the operating system's new hardware wizard. If this occurs, allow the driver installation to compete before you proceed (it may take several minutes).

The programmable logic in Xilinx FPGAs and Xilinx SoCs is configured by the bitstream. The bitstream is stored in internal SRAM, and the values that are loaded set the behavior of logic and routing resources. When power is removed, the SRAM contents are lost – so each time power is applied, the programmable logic starts off blank.

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.

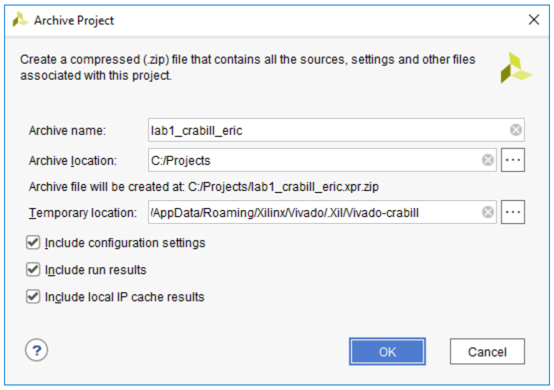
At the completion of this process, the hardware target is ready for programming and Vivado takes you to the Hardware Manager. There are several ways to initiate device programming, the easiest being to click on "Program Device" shown in the green status bar towards the top of the window. After you click on this, a window will appear to confirm the filename for the bitstream. By default it is the bitstream file you created in your project.

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. Now that you are done, close the Hardware Manager by clicking on the "X" in the light blue Hardware Manager title bar.

Laboratory Hand-In Requirements

Once you have completed a working design, prepare for the submission process. You are required to demonstrate a working design. You are also required to submit an archive of your project in the form of a ZIP file. Use the Vivado "File" → "Project" → "Archive..." option to create the ZIP file. Name the archive lab1_yourlastname_yourfirstname.zip. See Figure 15 for an example.


Archive Project

Create a compressed (.zip) file that contains all the sources, settings and other files associated with this project.

Archive name: lab1_crabill_eric

Archive location: C:/Projects

Archive file will be created at: C:/Projects/lab1_crabill_eric.xpr.zip

Temporary location: /AppData/Roaming/Xilinx/Vivado/.Xil/Vivado-crabill

☑ Include configuration settings

☑ Include run results

☑ Include local IP cache results

Figure 15: Archive Project

You will submit your archive to the instructor through Canvas by the due date and time along with a narrated video of your demonstration. If your circuit is not completely functional by the due date, you should demonstrate and turn in what you have accomplished to receive partial credit. See the syllabus for the late penalty guideline.