



THE IMAGINATION UNIVERSITY PROGRAMME

RVfpga Lab 5

Creating a Vivado Project

1. INTRODUCTION

In order to work with and modify the RVfpga System, you will need to build a project that includes all of the Verilog, SystemVerilog, header, configuration, and text files that define the system. In this lab, we show how to create a Vivado project that targets the SoC used in this course to Digilent's Nexys A7 FPGA board, -100T version (i.e. RVfpgaNexys). (Remember that if you have a Nexys4 DDR board already, you can also use that as well.) By following these same steps, you will be able to modify RVfpgaNexys and resynthesize it.

IMPORTANT: If you haven't already done so, install Xilinx's Vivado 2019.2 WebPACK as explained in the Getting Started Guide.

2. Creating a Vivado Project for RVfpgaNexys

You will use Xilinx's Vivado Design Suite¹ to build the RVfpgaNexys system using the RTL, the Verilog files that define the system. Follow these steps, detailed below, to build the RVfpgaNexys system and target it to a Nexys A7 FPGA board.

- Step 1. Open Vivado**
- Step 2. Create a new RTL project**
- Step 3. Add the RTL source files and the constraint files**
- Step 4. Select Nexys A7 as target board**
- Step 5. Set rvfpganexys as Top Module and common_defines.vh as global**
- Step 6. Generate Bitstream**

Step 1. Open Vivado

If you did not install Vivado on your machine as described in the RVfpga Getting Started Guide, do so now. Be sure to install the board files as well.

Now, run Vivado (in **Linux**, open a terminal and type: vivado; in **Windows**, open Vivado from the Start menu). The Vivado welcome screen will open. Click on Create Project (see Figure 1).

¹ In these materials we use Vivado 2019.2. Although most things should also work in more recent Vivado distributions, we highly recommend the use of this version.

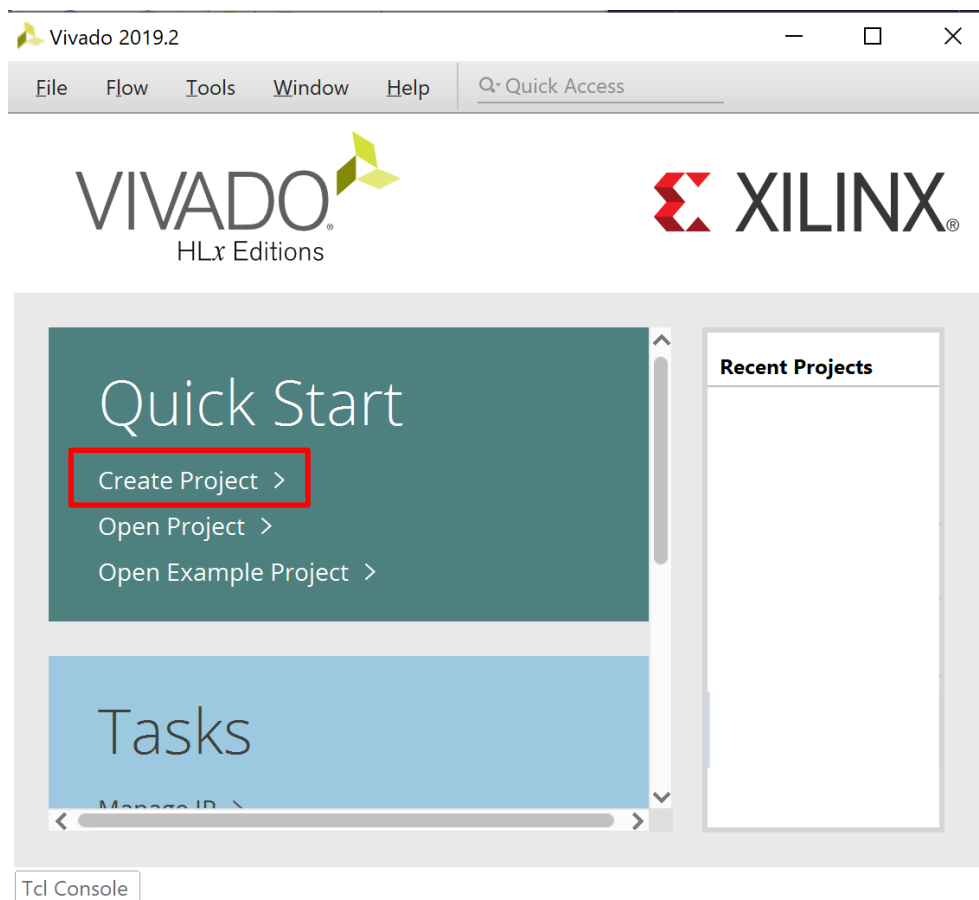


Figure 1. Vivado welcome screen: Create Project

Step 2. Create a new RTL project

The Create a New Vivado Project Wizard will now open (see Figure 2). Click Next.

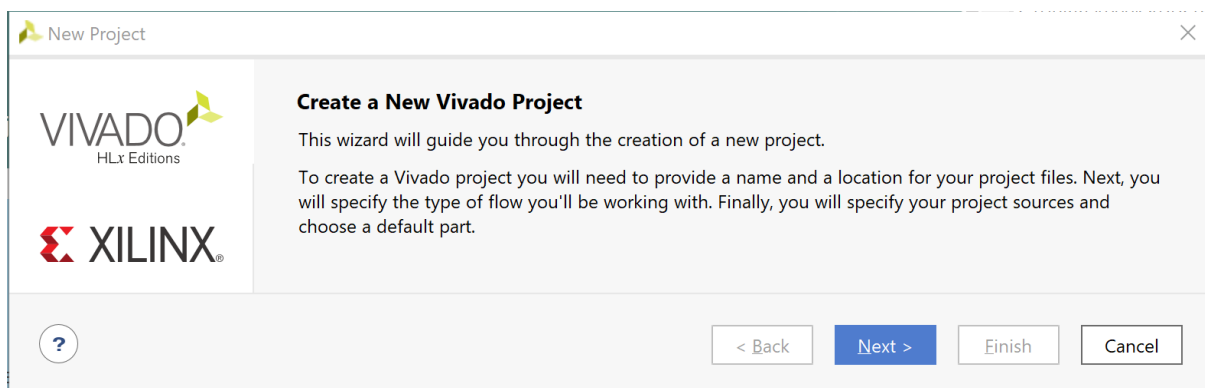
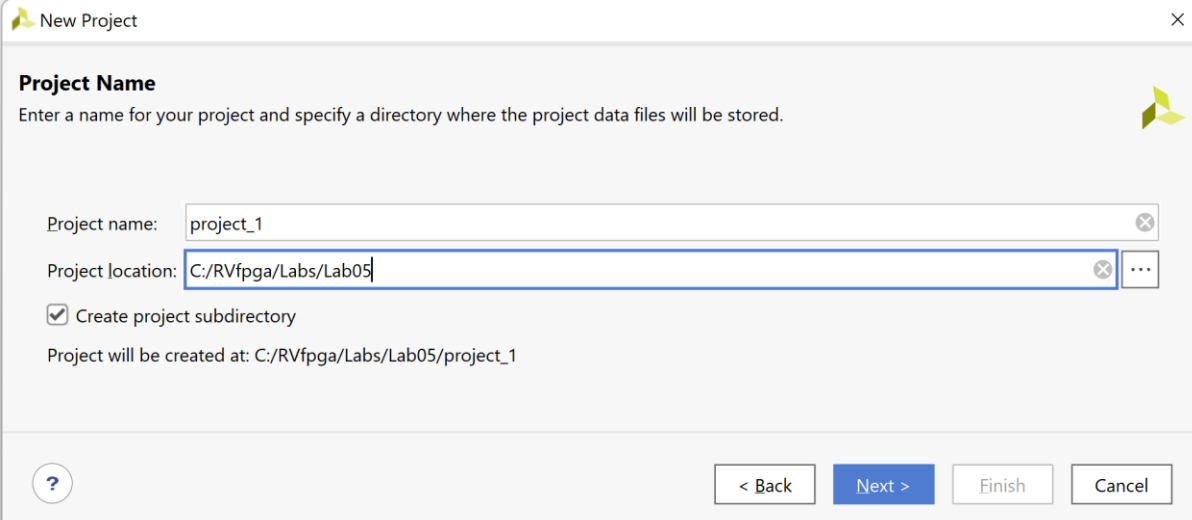


Figure 2. Create a New Vivado Project Wizard

Call the Project name Project1 and place it in the `[RVfpgaPath]/RVfpga/Labs/Lab05` folder. Select option *Create project subdirectory*. Then click Next (see Figure 3).



New Project

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

Project name:

Project location: ...

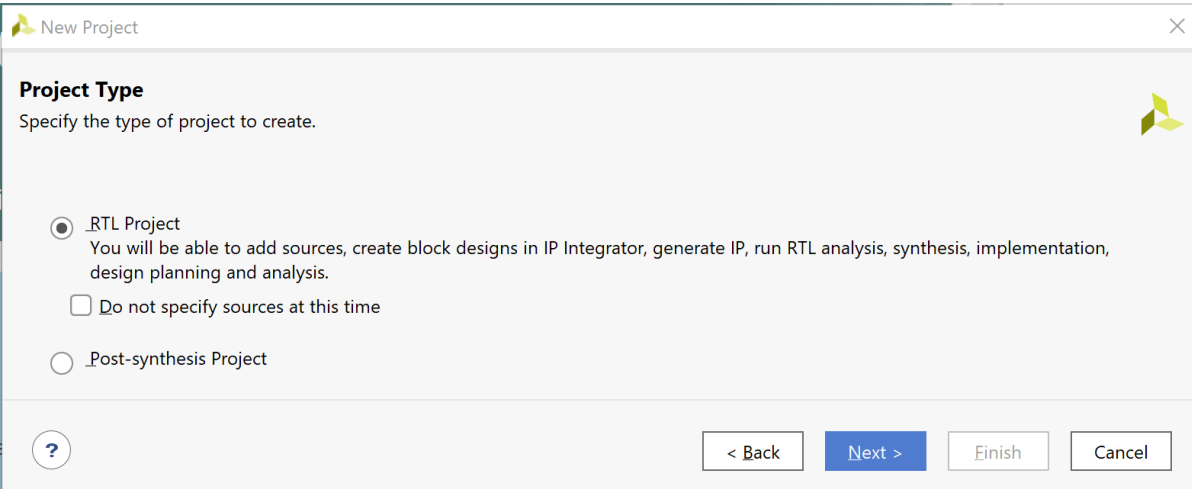
☒ Create project subdirectory

Project will be created at: C:/RVfpga/Labs/Lab05/project_1

? < Back Next > Finish Cancel

Figure 3. Project Name

Select the project type as RTL Project, and click Next (see Figure 4).



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

? < Back Next > Finish Cancel

Figure 4. RTL Project

Step 3. Add the RTL source files and the constraint files

In the Add Sources window, click on Add Directories, and select *[RVfpgaPath]/RVfpga/src* (see Figure 5). Make sure both of the following options are selected (as shown in Figure 5):

- Scan and add RTL include files into project
- Add sources from subdirectories

Then click Next.

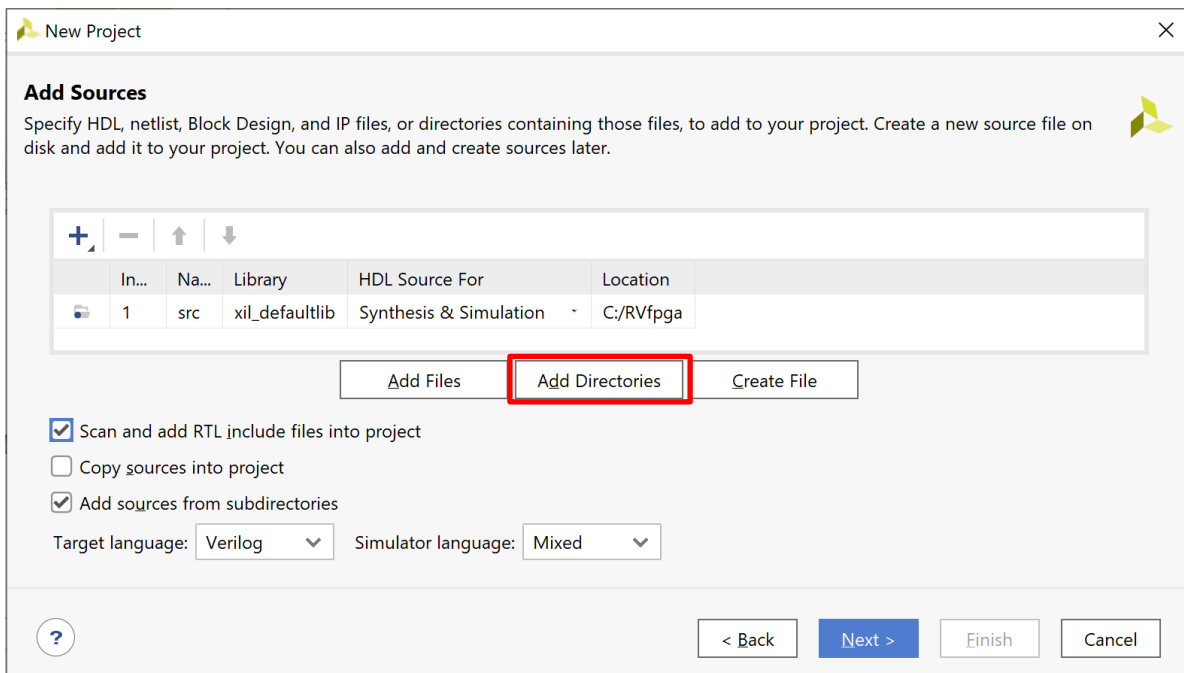


Figure 5. Add Sources

You will now add the constraints for the system. These files map the signal names to the pins on the board. For example, the LEDs on the Nexys A7 FPGA board are connected to FPGA pins on the board through traces in the PCB. Vivado must know this so that it maps the correct signal name in the RTL to the correct FPGA pin. For example, the following line in the `[RVfpgaPath]/RVfpga/src/rvfpganexys.xdc` file, a Xilinx design constraints file, indicates that FPGA pin H17 maps to the least significant LED (`o_led[0]`) and that it uses LVCMOS 3.3V signalling:

```
set_property -dict { PACKAGE_PIN H17    IOSTANDARD LVCMOS33 } [get_ports { o_led[0] }]
```

Note that the signal name `o_led` is the name used in the Verilog code to drive the Nexys A7 board's LEDs.

In the Add Constraints window, click on Add Files and select the following two files (see Figure 6):

```
[RVfpgaPath]/RVfpga/src/rvfpganexys.xdc
[RVfpgaPath]/RVfpga/src/LiteDRAM/liteDRAM.xdc
```

Then click Next.

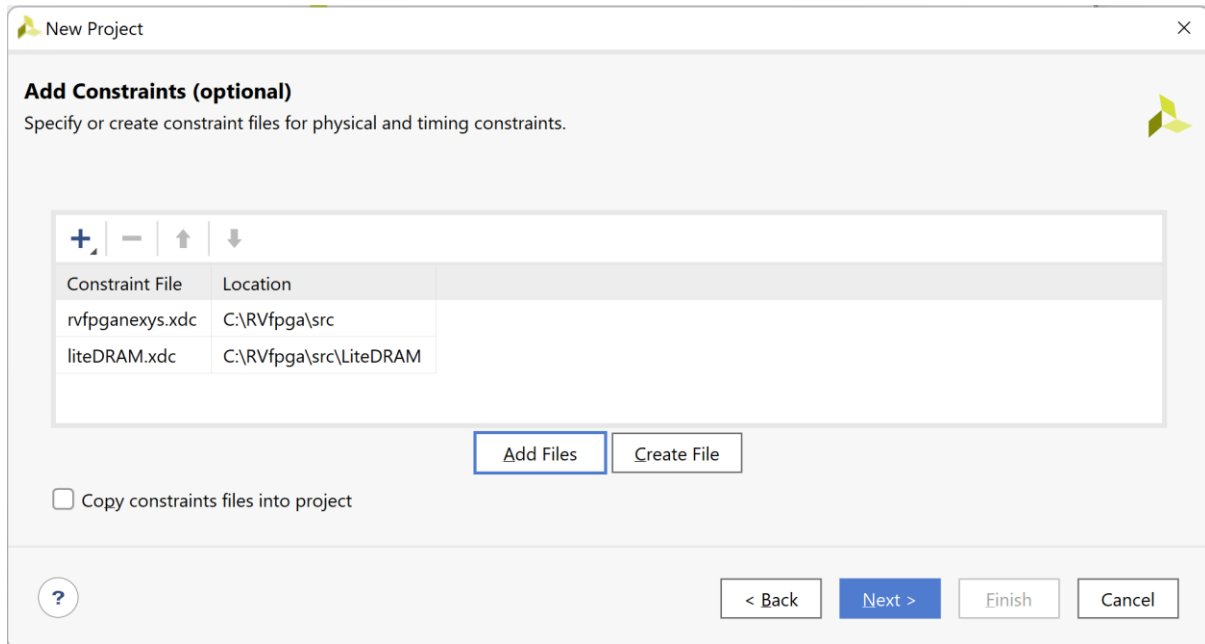


Figure 6. Add Constraints

Step 4. Select Nexys A7 as target board

In the Default Part window, click on Boards and then select Nexys A7-100T (see Figure 7). You may use the Search box to narrow down the results. You will also notice that the name of the actual target FPGA is listed in the Part column: xc7a100tcsg324-1. This indicates that it is a Xilinx Artix-7 FPGA with 100k equivalent gates with a CSG (chip-scale grid) package and 324 pins.

Click Next.

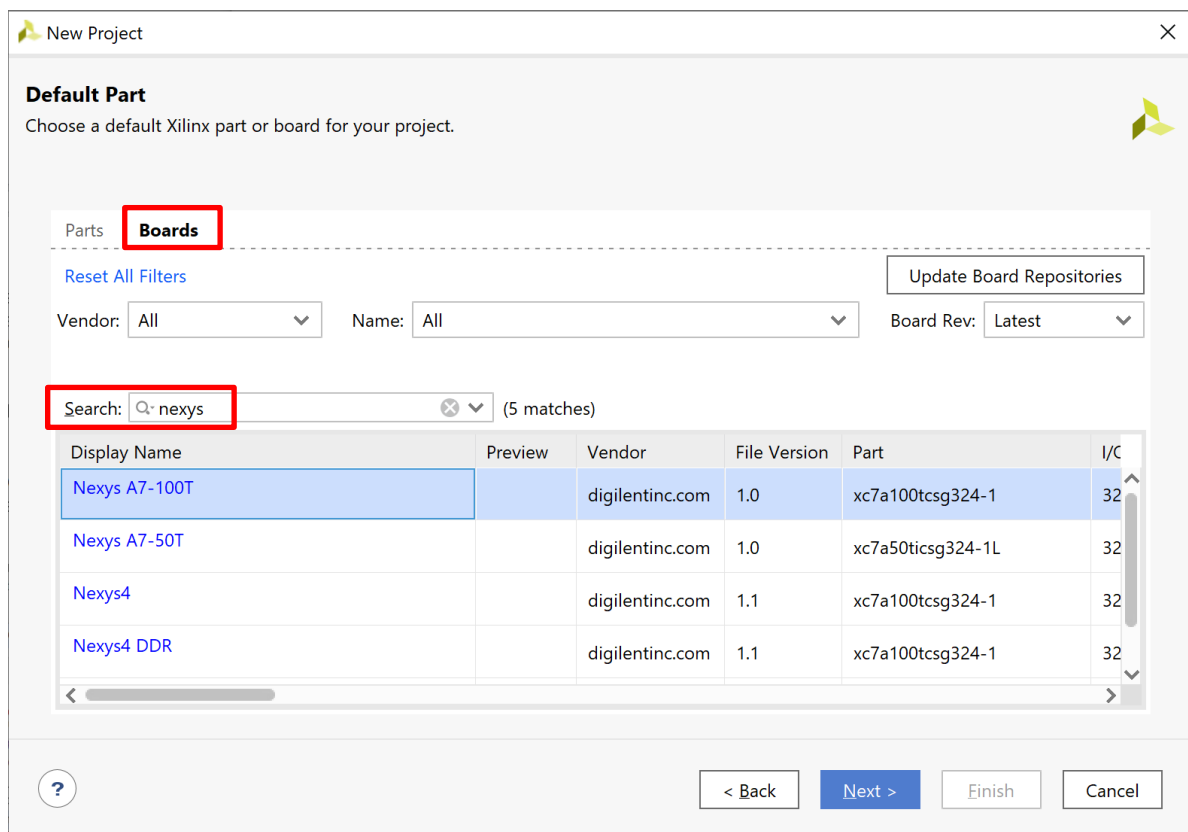


Figure 7. Select target board: Nexys A7-100T

In the New Project Summary window, click Finish (see Figure 8).

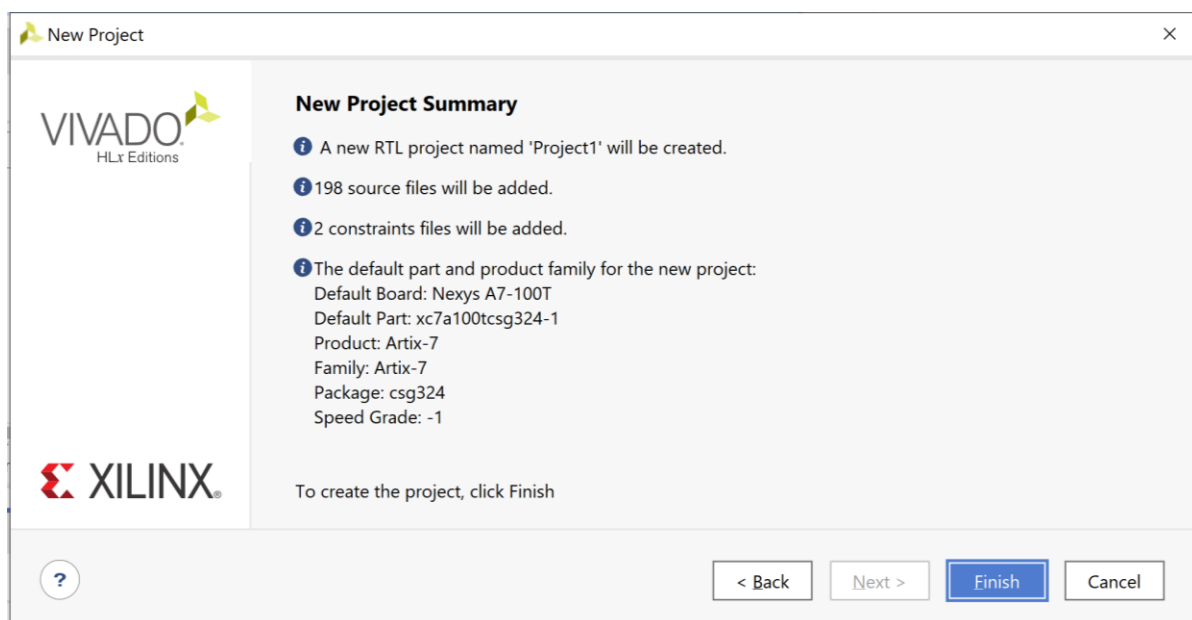


Figure 8. New Project Summary Window

Note that once the project completes being set up, it will indicate that files exist with Syntax Errors – this will be fixed in the next step.

Step 5. Project Configuration: Set rvfpganexys as Top Module, set file *common_defines.vh* as global, add *boot_main.mem* to the project and include Pulp Platform folders

Set rvfpganexys as Top Module: You will now set the rvfpganexys module as the top module. In the Sources pane, scroll down under Design Sources, right-click on the rvfpganexys module, and select Set as Top (see Figure 9). You can also find the rvfpganexys module by typing this name in the search box, as shown. This sets rvfpganexys as the highest-level module in the hierarchy and the target to be synthesized and implemented onto the FPGA. After setting rvfpganexys as the top module, the hierarchy will update.

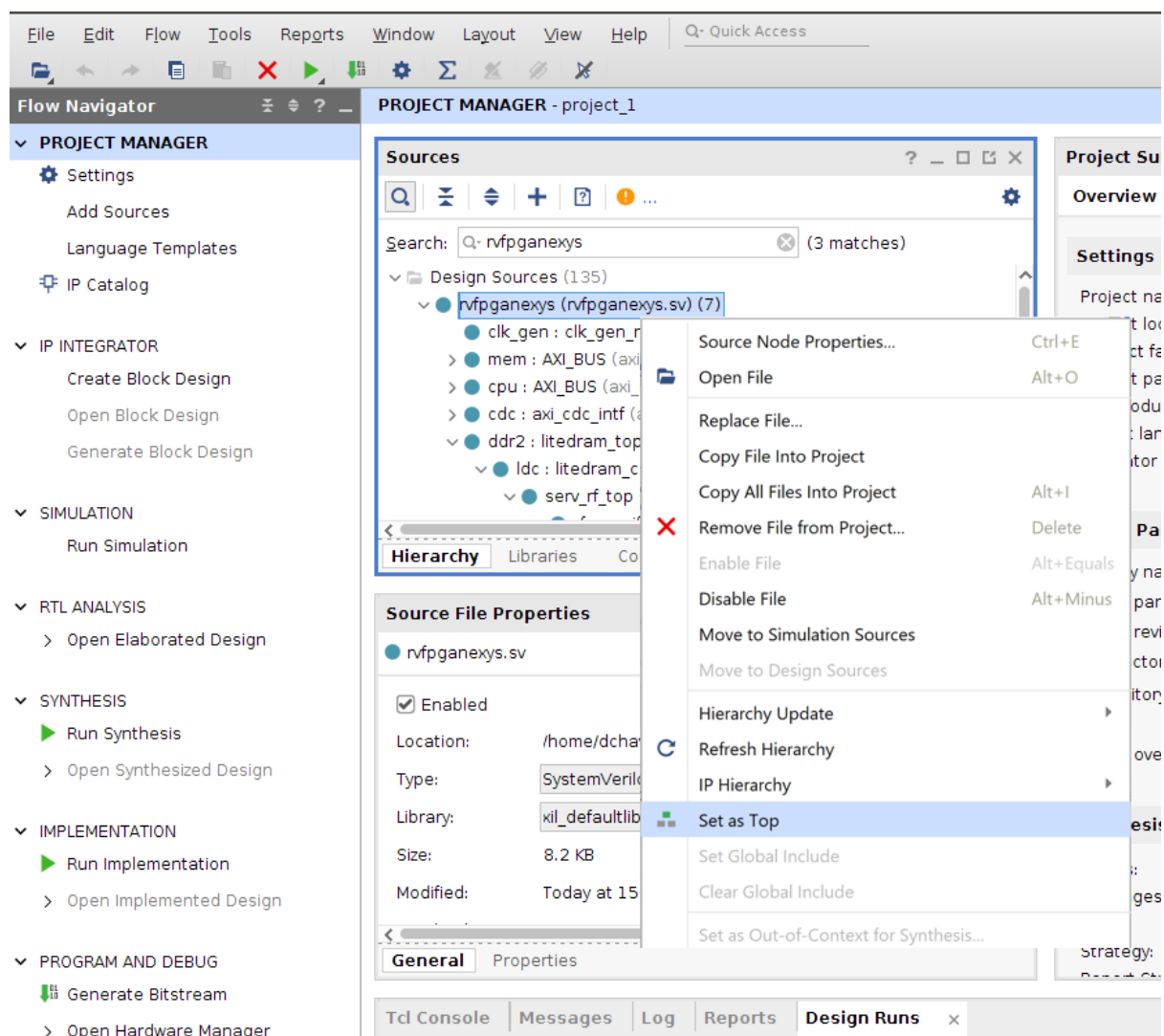


Figure 9. Set rvfpganexys as top module

Set file *common_defines.vh* as Global Include: Now, still in the Sources pane under Design Sources, expand the Non-modules file group and click on *common_defines.vh*. The properties of the file will then open in the Source File Properties pane, just below the Sources pane. Click on Global Include to tick that box (see Figure 10). The hierarchy will now update and include that file in Design Sources/Global Include. Note that the Syntax Error Files will disappear in a future step.

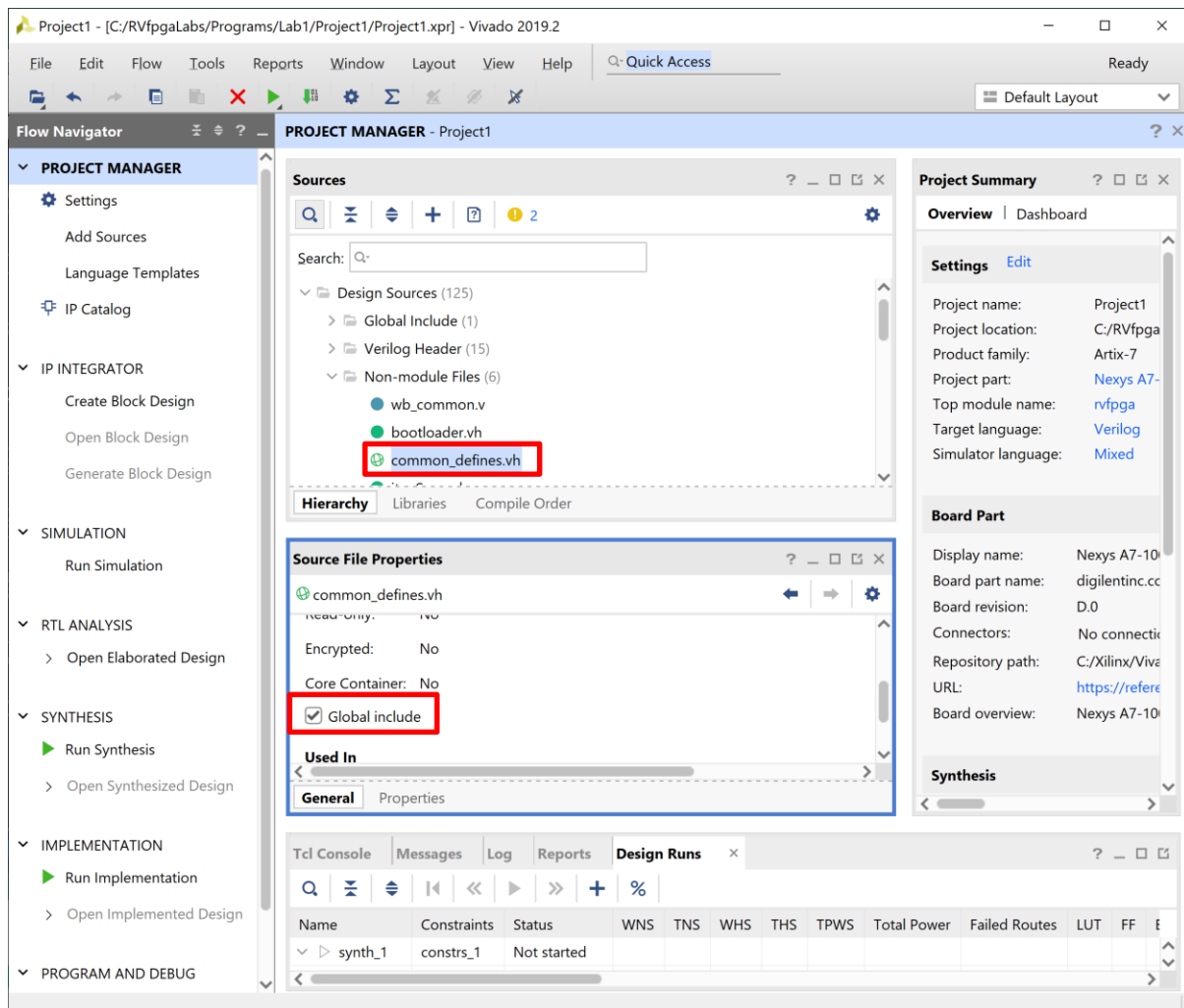


Figure 10. Set common_defines.vh as a Global include file

Add *boot_main.mem* to the project: In the Flow Navigator pane, click on Add Sources, leave the default option (“Add or create design sources”), and click on Add Files (see Figure 11). Navigate to *[RVfpgaPath]/RVfpga/src/SweRVolfSoC/BootROM/sw* and select *boot_main.mem* (as shown in Figure 11). The hierarchy will update and include that file in Design Sources/Memory File.

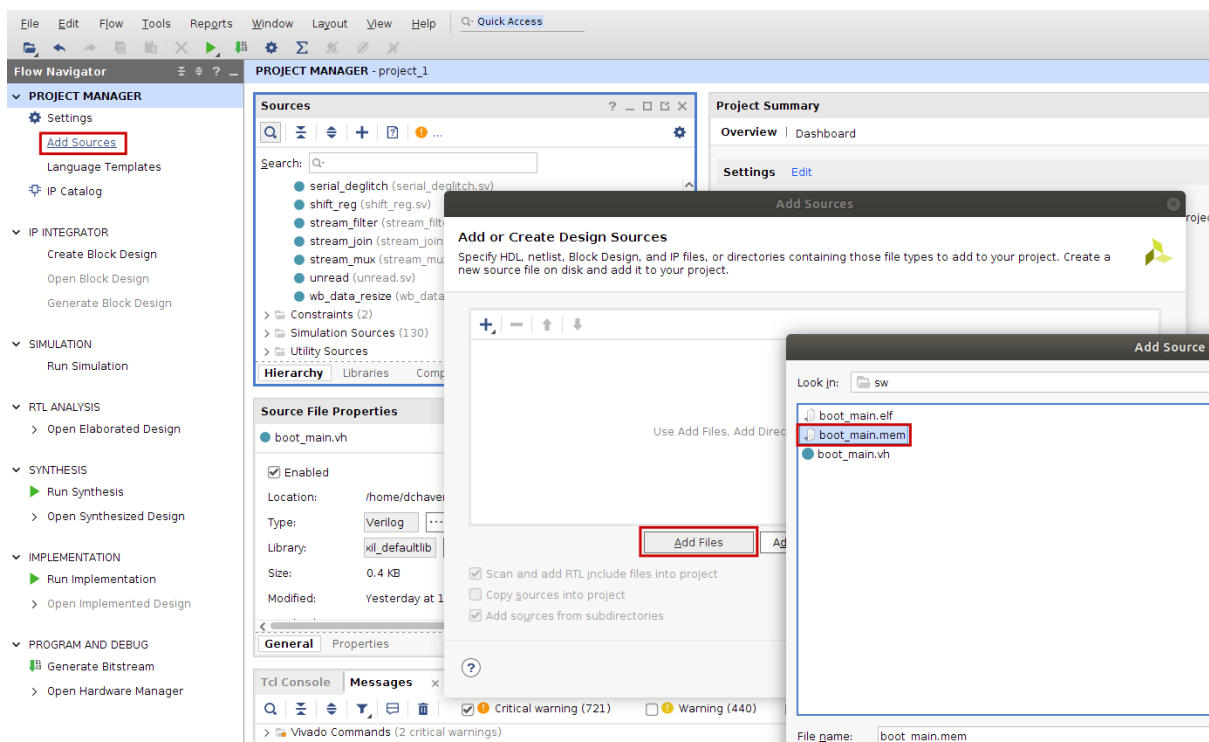


Figure 11. Add Memory File boot_main.mem

This file (*boot_main.mem*) is used for initializing the Boot ROM of our SoC by invoking it as a parameter in file `[RVfpgaPath]/RVfpga/src/rvfpganexys.rv`.

```
25 module rvfpga
26     #(parameter bootrom_file = "boot_main.mem")
```

Section 6.A in the Getting Started Guide contains more information about this file.

Include folders: Finally, include two folders for the Pulp Platform (see Figure 12). In the Flow Navigator pane click on *Settings*, and on the window that opens click on *General* and

then on *Verilog options* (⋮). In the new window, add the two following include directories by clicking on **+** and browsing to the directories:

```
[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/AxiInterconnect/pulp-platform.org__axi_0.25.0/include
[RVfpgaPath]/RVfpga/src/OtherSources/pulp-platform.org__common_cells_1.20.0/include
```

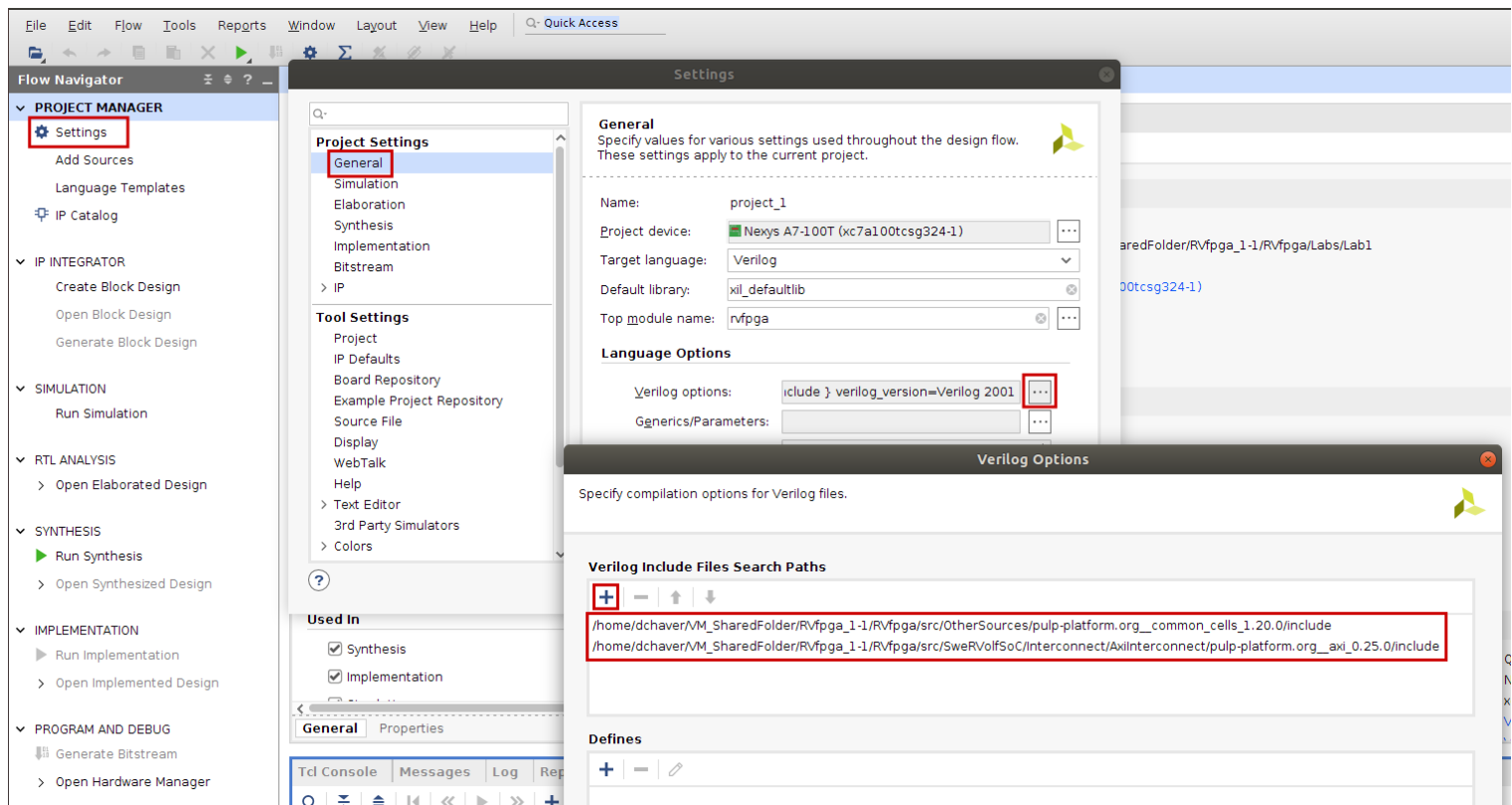


Figure 12. Include folders for the Pulp Platform

Step 6. Generate Bitstream

Now Click on Flow → Generate Bitstream as shown in Figure 13. A window might pop up that says there are no implementation results available and ask to launch synthesis and implementation (see Figure 14). Click Yes. Then click OK in the Launch Runs window (see

Figure 15). This step synthesizes RVfpgaNexys (as defined by the Verilog and SystemVerilog files in the project), maps it onto the FPGA, and creates the bitstream. This process typically takes 20-50 minutes, depending on the speed of your computer.

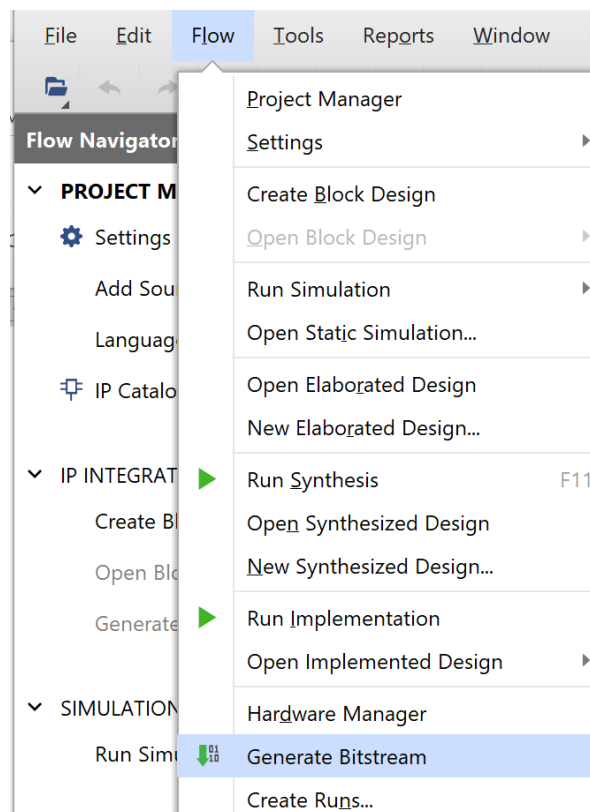


Figure 13. Generate Bitstream

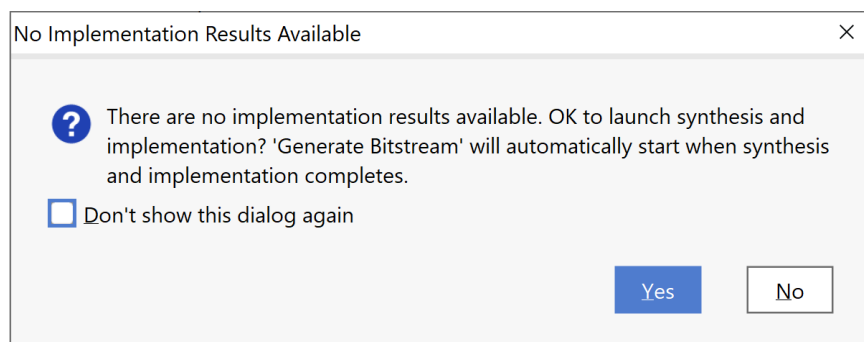


Figure 14. Launch synthesis and implementation window

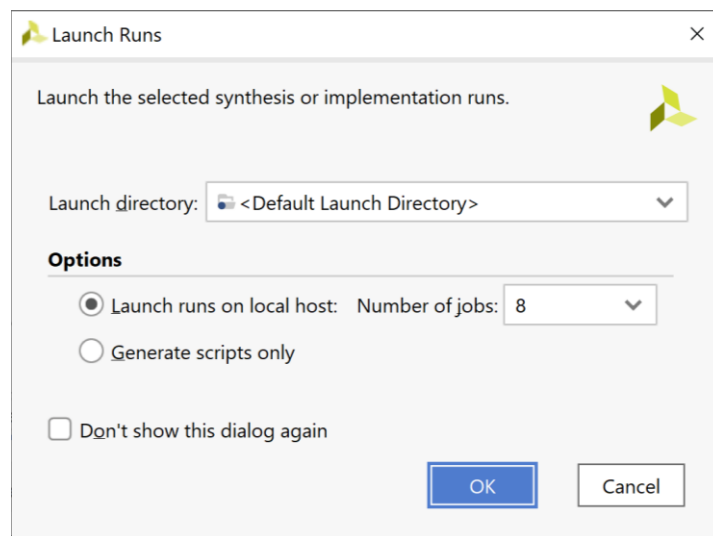


Figure 15. Launch Runs

After the bitstream has been generated, a window will pop up as shown in Figure 16. Click on the **X** button in the top-right corner to close the window.

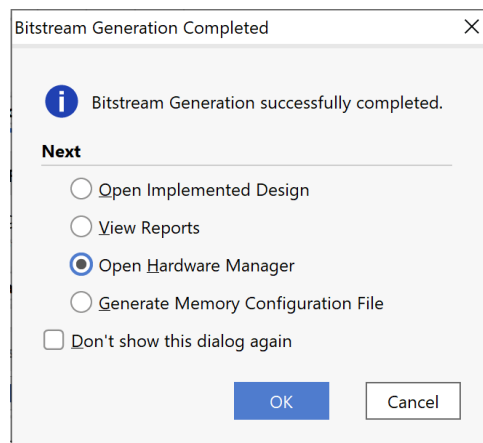


Figure 16. Bitstream Generation Completed

Now that you have built the RVfpgaNexys system yourself, you will be able to rebuild RVfpgaNexys after you make modifications to it in Labs 6-10. You can now also use RVfpgaNexys system you just built to download and run programs on it using PlatformIO – instead of the one used in the past four labs and in the Getting Started Guide (at [RVfpgaPath]/RVfpga/src/rvfpganexys.bit).

IMPORTANT: You can find the bitstream just generated by Vivado in folder:

[RVfpgaPath]/RVfpga/Labs/Lab05/project_1/project_1.runs/impl_1

It is recommended that you use PlatformIO (instead of Vivado) to download RVfpgaNexys onto the Nexys A7 board, as described in Labs 1-4 and in the RVfpga Getting Started Guide

(GSG) in detail. As also described in the GSG and Labs 1-4, after downloading the RVfpgaNexys system onto the FPGA on the Nexys A7 board, you will use PlatformIO to download and run/debug programs on RVfpgaNexys.

You can also use Verilator, an HDL simulator, to simulate programs running on RVfpgaSim, as described in Section 7 of the RVfpga Getting Started Guide. These RTL-level simulations allow you to view low-level hardware signals as the software program runs. We will rely on Verilator heavily in Labs 6-10 as you extend the RVfpga System and test and debug your changes.